VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

3D scanner automation

Automatizace 3D skeneru

Bc. Pavel Mandrla

Semestral project

Supervisor: Mgr. Ing. Michal Krumnikl, Ph.D.

Ostrava, 2021

Contents

Chapter 1

Introduction

The 3D printing industry has experienced a giant boom in the last fifteen years. Intensive research and innovation in this area have greatly expanded its possibilities, and 3D printing is therefore more and more often used by big industrial companies to manufacture high-tech products.

However the roots of this industry come from open source projects, driven by a passionate community of hobbyists and tinkerers, and that is greatly reflected in today's 3D printer market. Many companies do not focus their products on large industrial subjects, but on public. That means that the price of 3D printers has gone down, while their accessibility and capability highly increased, and it is not uncommon for a household to own a 3D printer.

With these devices becoming more widespread comes a great demand for high quality 3D models. These can be created either by using CAD software, which creates high precision 3D model, but has a very steep learning curve, or by creating a 3D scan of an already existing object.

As the price of an industrial 3D scanner can be very high, the 3D printing community had to turn to more cost friendly alternatives. Devices, like the Kinect from Microsoft can be used, when scanning larger object, but when it comes to smaller ones, the quality of the resulting 3D model is not sufficient. Several open source alternatives exist for scanning small objects. They utilize easily sourced off the shelf parts and are therefore much more accessible to public. Whilst these solutions are not utilizing as complex technologies, as their industrial counterparts, the quality of the resulting 3D prints can still be highly satisfactory.

Chapter 2

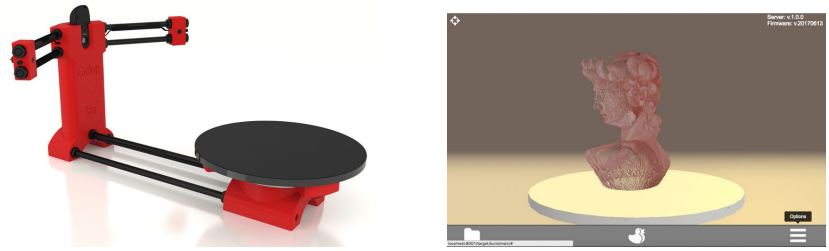
Open source solutions

3D scanners can be divided into two categories, based on the technology, that is used to scan the object and create the 3D mesh. These categories are contact based and non-contact based. Contact based 3D scanners require physical contact with the scanned object to measure the location of each point on the objects surface, while non-contact based estimate the position of each point by using only passive scanners, such as cameras. [1] The latter are much easier to design and construct and therefore are much more prevalent within the open source community. They can be further divided into two subcategories, which are passive and active.

2.1 Active scanners

Active scanners actively emit radiation such as light towards the surface of the object to scan it. It can be used similarly to contact based scanners, but instead of having to use physical probe to measure the position of the surface, the distance can be measured by a time-of-flight sensor, which measures the time it takes the emitted light to fly to the object and bounce back.

A different approach is the use of structured light. In this approach, light patterns are projected on the surface, which is then captured by the camera. These pattern enhance the contours of the object and help with the triangulation of the surfaces points.



(a) (b)

Figure 2.1: (a) BQ Ciclop (taken from [2]) (b) 3D scan created using FabScan (taken from [3])

Examples of popular projects, that use active scanning to create the 3D mesh, are FabScan, BQ Ciclop and FreeLSS. [2, 3, 4] All of these projects are based on the same technique, where the scanned item is placed on a turntable, which rotates it along the Z axis and allows a camera to capture the object from different angles. Meanwhile a planar laser is shining towards the turntable, intersecting the subject. The line created by the laser on the objects surface highlights its outline. Because the outline, illuminated by the laser, is brighter, than the ambient ligt, which illuminates the rest of the object, it can be easily extracted from the image. The position of each point along the outline can then be triangulated based on the position of the camera and the rotation of the turntable. The points from all the scanned pictures are then stitched together to create the final mesh.

2.2 Passive scanners

Passive scanners use visual information acquired by cameras to create the resulting 3D mesh. The technique used by passive scanners is called photogrammetry or Structure from Motion (SfM). It calculates positions of the objects points in space based on photographs of the object taken from multiple angles. [5] Photogrammetric reconstruction software compares individual images and tries to find tie points, which represent the same point in space between two images. Based on these tie points, the camera position, from which each photo was taken, is calculated and after that the position of each point of the object can be determined.

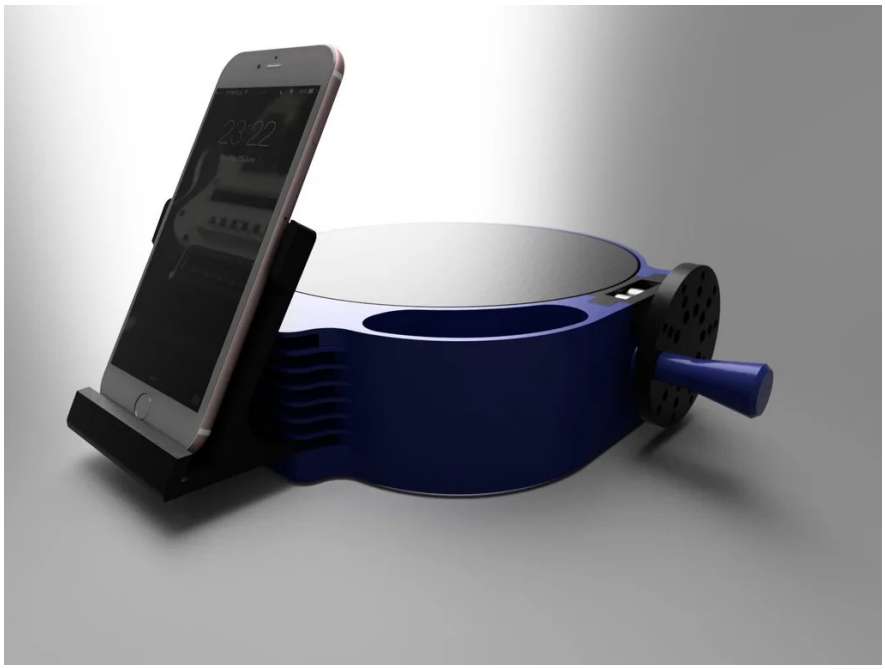


Figure 2.2: The $30 3D scanner (Taken from [6])

Because even cameras available in today's mobile phones are sufficient for high quality photogrammetric 3D reconstruction, this technique is very accessible and cost efficient. Since capturing the images by hand can be a quite tedious task, as the reconstruction requires many images taken from various angles, the available scanners focus on automation of this process.

2.2.1 OpenScan

OpenScan is a project created by Thomas Megel in 2018. It is a device, that automatically takes photos of the scanned object from multiple angles, by rotating it along two axis. These photos can be fed into a photogrammetry reconstruction software, to create the resulting 3D mesh.

It supports many different types of cameras, so it is very inviting to all types of users. As a cheaper option, it allows for the use of phones or webcams, to scan the object, but if the user desires higher quality scans, the scanner is also compatible with DSLR cameras. It is powered either by a Raspberry Pi or an Arduino, and multiple variations of this scanner exist. The development of this project is still very active and it receives regular updates.

Figure 2.3: OpenScan (Taken from [7])

Chapter 3

My contribution

I have chosen to work with the OpenScan project because of multiple reasons. First I was very interested in photogrammetry and its use for 3D reconstruction. It also seemed to me, that the project has a lot of potential to become successful even with the more casual part of the 3D printing community, as it is easy to setup and has a simple user interface. It also uses the more simplistic approach of using photogrammetry, which requires less calibration than the use of structured light, so it is very user friendly. Unlike some of the previously mentioned projects, it is still actively supported by the projects author, who is still trying to improve the the project by bringing new features and designs to the platform. I also liked that the scanner does not just rotate the object along the Z axis, but allows the camera to capture the object from the top and bottom, which means, that the scanner can capture more of the object and create a higher quality result.

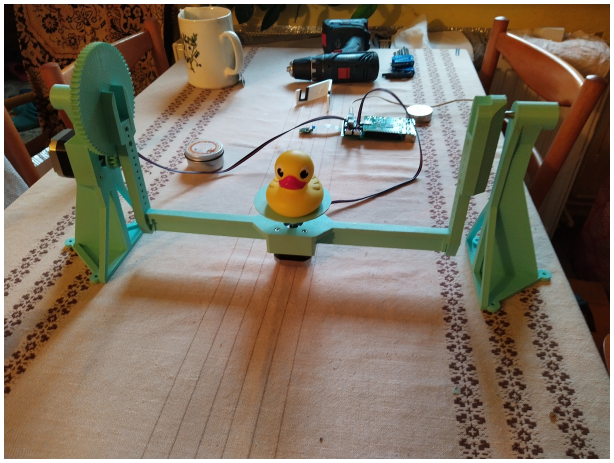


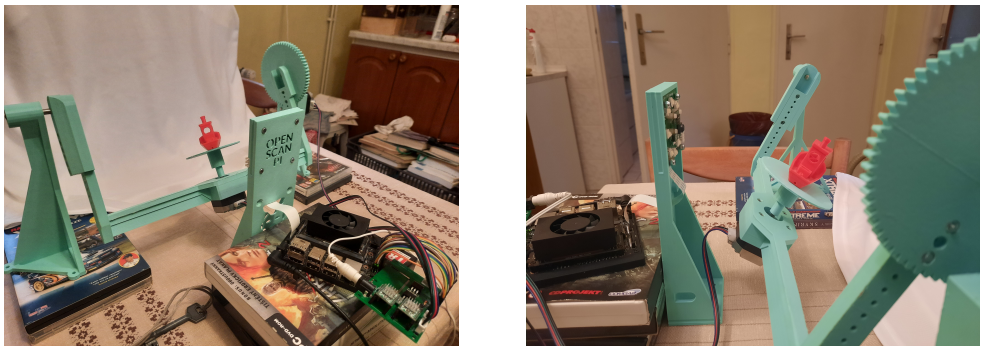
Figure 3.1: Assembled scanner

Where I saw room for improvement, was that the output of the scanner is not a finished 3D representation of the scanned object, such as an .stl file, but just a set of images that the user has to process himself with an additional tool, such as Meshroom. The author of the project has since

started working on his own solution to this problem and in February launched a beta of OpenScan Cloud, which is a cloud service, that automatically processes the scanned images and produces the resulting 3D mesh. [8]

I have decided to use a slightly different approach, and instead of sending the scanned pictures to a server, where the processing and assembly would be done, I would do the required calculation locally on the scanner itself. The only problem with this approach, is that the Raspberry Pi, which is used by the OpenScan project, is not powerful enough, to do the required processing. Also the photogrammetry reconstruction pipeline AliceVision [9, 10], which I have chosen for the mesh reconstruction is dependent on CUDA, so a device with a CUDA capable GPU would be needed to run the reconstruction pipeline. Because of these reasons, I have decided to replace the Raspberry Pi with a more powerful alternative, which is the Jetson Xavier NX by Nvidia. Its more powerful 6 core processor, larger memory and CUDA capable graphics card would allow me to process the scanned data locally, without the need to send it to a server.

The assembly of the scanner was very easy and straight forward. I have printed all the required parts on my 3D printer, which are designed to be easy to print without the need for any support material and assembled them together. I ordered the electronics from the projects author. The main control board came already soldered together, but other parts, such as the front ring light needed some soldering. Because of the Covid-19 crisis, the Jetson Xavier NX took a few months to arrive, so I temporarily used my Raspberry Pi 3 to test the scanner. With the arrival of the Jetson, I could finally begin to port the scanners software to the new device.



(a) (b)

Figure 3.2: Assembled scanner

3.1 Evaluation of the existing solution

With the change of the platform, some problems with the current solution became apparent. Even though the software for the scanner was created using platform independent technologies, the code itself was written without the prospect of running on a different platform. The main cause of this was in my opinion the use of the programming tool Node-RED. It is a graphical programming language built on top of Node.js, which allows the user, to connect together different nodes that represent parts of the code. [11] Some nodes can for example contain JavaScript code, that will be executed, when the node gets activated.

The author of the OpenScan project decided to use an extension, which allowed him to use Python instead of JavaScript. This solution did not allow for easy reuse of the python code, and so the code for each functionality had to be rewritten inside each node, that was using that functionality.

I also belive, this was the reason, why the author decided to store each parameter of the scanners configuration in a separate file. Whenever the software needs to know the value of a parameter, such as the number of photos, the scaner should take, or the number of the GPIO pin used by a stepper motor, it loads a file, which contains the parameters value. Because the paths to these configuration files are hard coded in the codebase, making changes to the configuration code can be very difficult.

For these reasons, the code structure of the scanners software became quite bloated and chaotic, which must make maintenance of this project a hard task. It also means, that moving the codebase to a different platform would be quite complicated.

3.2 Used technologies

After trying to port the existing codebase to the Jetson platform, I have decided to create my own software, which would control the scanner, as it would be much easier then to struggle with the problems stated in the previous section. As my language of choice, I used Python 3, because the Jetson provides an easy to use interface to use the GPIO pins for this language. It also allowed me to reuse some parts of the original OpenScan software, such as the code that controls the stepper motors, which I adjusted to suite my needs. I opted for GStreamer in combination with OpenCV, to interface with the camera and capture images. To circumvent problems with OpenCVs buffering of the captured images, I had to run the image capturing code in a separate thread to always get the latest frame. I also replaced the many configuration files, with a single JSON document, which contains the values of each parameter and is loaded into memory at the start of the scanning procedure.

My implementation was mainly concerned with the scanning process itself, as I decided not to reimplement some features of the OpenScan Project, such as the web-based user interface, or the use

of a SAMBA protocol to easily share the scanned files, as they were not crucial for my project. The program starts by parsing the configuration JSON file. It then executes the scanning procedure, positioning the model according to the settings specified by the configuration, and taking a photo of the model each time. The captured images are saved into a new directory, whose path is also specified in the configuration. After the scanning procedure is finished, the resulting images are fed into the Structure from motion pipeline, which creates the final 3D mesh.

The move from Node-RED to Python3 made the codebase much more compact and easily maintainable. I believe, that this would be the right move for the project in the future, because it would make addition of new features and maintenance much easier.

Figure 3.3: Example of captured images

3.3 Building the Structure from Motion Pipeline

For the reconstruction of the 3D mesh, I have chosen the AliceVision photogrammetry framework [9, 10]. I chose this framework, because I have been testing the Meshroom photogrammetry reconstruction software, which is based on the AliceVision framework, and provides an easy to use user interface for it, and I have had very good results with it.

The framework provides the user with several binaries, each representing a step in the reconstruction pipeline. The user only has to execute these binaries with the correct parameters in the correct order to create the 3D mesh from the captured photos of the object. Unfortunately, the creators of this framework were only testing it on the x86_64 architecture, so no available binaries existed for the Jetson Xavier NX which is built on the AArch64 ARM architecture, which meant, that I would have to build the binaries myself. This turned out to be a quite challenging task, as the project depends on a large number of libraries, most of which had to be compiled manually as well and had dependencies of their own. This phenomenon is also known as dependency hell.

Figure 3.4: 3D reconstruction created with MicMac (Taken from [12])

Before trying to compile all the necessary dependencies, I installed and teseted a different photogrammetry software Micmac, which was much easier to install. Micmac is free open-source photogrammetry reconstruction software, which was developed by the French National Geographic Institute and the French national school for geographic sciences. [13] It claims to be highly versatile and allow the creation of both 3D models and ortho-imagery.

I have experimented with MicMac for a week and followed several tutorials, but I was not able to create a 3D model from my scanned images, which would normally result in a 3D mesh in Meshroom. The error messages, which MicMac showed me after failed reconstructions were quite vague and non informative and sometimes in french, so I am unsure, where the problem was, but I suspect, that even though MicMac claims to be versatile, it is not suited for reconstruction of small

objects. This suspicion is based on the fact, that most of the materials available on the MicMac wiki are focused on orthophotography and reconstruction of large objects, such as buildings.

After this I looked into other available photogrammetry 3D reconstruction solutions, such as OpenSfM [14] and COLMAP [15, 16], but similarly to AliceVision, they required many dependencies, so I returned to building AliceVision and its dependencies, as I tested it before and knew I could successfully use it to reconstruct a 3D mesh.

It took quite a long time to build it, because some of the libraries did not expect to be compiled on the AArch64 platform and I would get stuck on trying to figure out problems with their compilation. Most of the times the libraries wanted to use features such as SSE2 instructions, which were not supported by the Jetson and the build system was not setup correctly to exclude these parts of the code. Unfortunately I made a mistake, while fixing such error, which has overwritten the C build flags, which lead to the build of the AliceVision framework failing after an hour of compilation, which was frustrating to debug and took me two weeks to find where the problem was.

Thankfully I have successfully managed to compile and install the AliceVision framework on the Jetson Xavier NX and run Meshroom on it. But when I tried to run the photogrammetry pipeline and create a 3D model, Meshroom refused to load the images. After some debugging, I found out, that the problem was in Boost library, on which AliceVision is dependent. When AliceVision tries to load the input images, boost uses a system call, which is not implemented on the Jetson platform, which unfortunately meant, that I could not use the AliceVision photogrammetry pipeline to reconstruct the 3D model on the Jetson Xavier NX. I managed to run the pipeline at least on a desktop computer, where it ran without any problems. I believe, that if these system calls were implemented, or boost wasn't used for the task, the pipeline would run and the Jetson Xavier NX would be a suitable platform for such task.

3.4 Used SfM pipeline

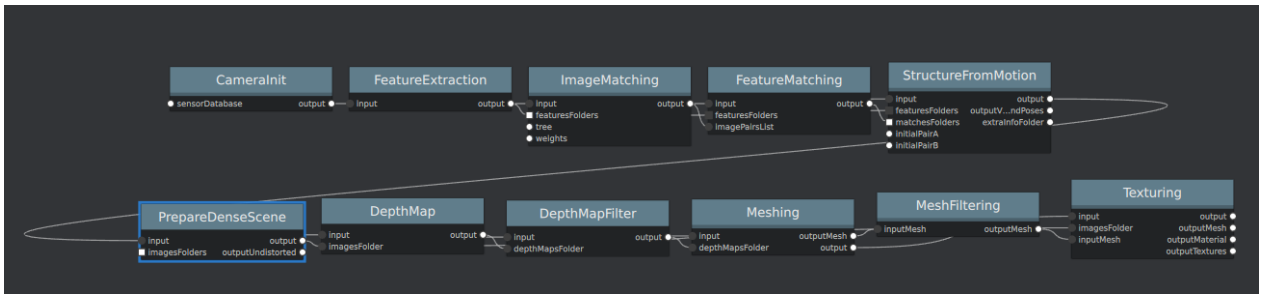


Figure 3.5: visualization of the used pipeline

For the actual reconstruction of the 3D object, I used the default pipeline used by Meshroom pictured in figure 3.5, which I had good results with, while testing. As can be seen in the picture,

the pipeline consists of several nodes each representing a different binary from the AliceVision framework. The pipeline works in the following steps. [17]

1. Visual data and EXIF information are loaded.
2. Features are extracted from each image using the SIFT descriptor.
3. Using the extracted features, the images are compared between each other to find images that are looking at the same area of the scene.
4. The extracted features are matched between neighboring pictures.
5. The StructureFromMotion node takes the camera information and information about the correspondence of features and tries to place them and the cameras into space.
6. The depth value of each pixel in each picture is calculated.
7. Finally the geometric representation of the scene can be calculated and then textured.

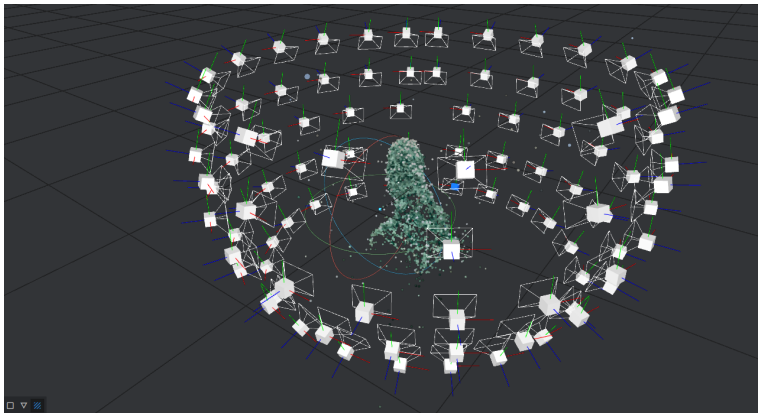


Figure 3.6: visualization of cameras and feature points placed in space after StructureFromMotion step

Chapter 4

Testing and results

In my initial testing, I found multiple factors, that greatly affected, whether the object could be reconstructed and its final quality. The factors are:

Background uniformity

When the reconstruction pipeline to position the cameras in space, it is matching images by looking at similar features in them and trying to find the closest match. Since it is the object and not the camera, that is moving, the background between each image stays static, which introduces a large number of features that are identical between all images, which makes the image matching a very hard task.

Harshness and uniformity of the lighting

The fact that it is the object, which changes position and rotation and not the camera also means, that if harsh directional lighting is used to illuminate the object, it will cast sharp shadows that will change between each shot, which will also complicate image matching.

Texture of the scanned object

Since only passive visual information is used to reconstruct the final mesh, as the used scanner falls into the passive scanner category, the texture of the object is a crucial factor, that not only affects the quality of the resulting mesh, but also wether the object can be reconstructed at all. The more textured the object is, the more features can be extracted from each image. If the texture is too uniform, not enough features will be extracted from the images and the reconstruction will fail, as there is not enough information to reconstruct it.

The importance of the objects texture can be seen in Figure 4.1. The reconstruction in this example failed mainly in the areas, where there was no text or pictures, but blank paper of the newspaper, that was glued onto the cube to give it more texture. It is important to say, that this mesh was created with draft meshing, which is much quicker, than the standard pipeline, that I was using, but also creates a mesh of a much lower quality.

(a) (b)

Figure 4.1: 3D reconstruction of a cube with newspapers on it to give it more texture (20 pictures,draft meshing)
(a) Picture taken from the cube dataset (b) Screenshot of the resulting geometry

4.1 Adjusting scanning setup for better results

(a) (b)

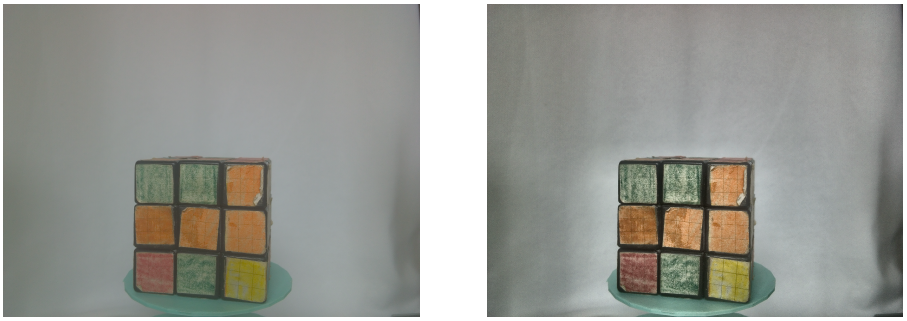
Figure 4.2: Examples of images after adjustments to scanning setup
(a) shot with Nikon D7100 DSLR (b) shot with Raspberry Pi cammera V2

With the factors stated above in mind, I tried to adjust my scanning setup to improve the quality of the resulting 3D models. I tried to light the object as uniformly as possible with soft light in order to reduce hard shadows as much as possible. I also solved the problem with background uniformity by using a large white fabric as the backdrop.

This assured that all of the features would be taken from the scanned object and not the background. This worked quite well, when I was testing the setup with my DSLR camera, with which I could isolate the object in the image very well and I could easily setup the camera settings, so the background is blown out and blured, while the object is correctly exposed and in focus. Moving from the DSLR to the Raspberry Pi camera V2 meant that I would lose some of the control, that the DSLR offered. Since the Raspberry Pi camera has a fixed view-angle, which is quite wide, i

would lose some of the isolation of the object and occasionally the borders of my backdrop would be visible in the scanned pictures. Thankfully the quality of the scanned pictures was still good enough for photogrammetric reconstruction.

With some objects such as the rubber ducky in Figure 4.2 the reconstruction pipeline failed, because the objects texture is too uniform and hard to reconstruct. During my testing I have tried to enhance the features within the scanned images by equalizing the histograms of these images by using Contrast Limited Adaptive Histogram Equalization implemented in OpenCV.



(a) (b)

Figure 4.3: (a) object before histogram equalization (b) object after histogram equalization

It seemed to help with some objects, but with the Raspberry Pi camera it also enhanced the noise, which was introduced into the images by the small sensor, and details in the background, which would be out of focus with the use of the DSLR, but not with the Raspberry Pi camera V2, as the bokeh effect cannot be utilized on the Raspberry Pi camera module to blur the background due to its small aperture. For this reason I have stopped using the histogram equalization within my implementation, but I believe, that with a better camera and a more uniform background it could help the reconstruction process to achieve better results.

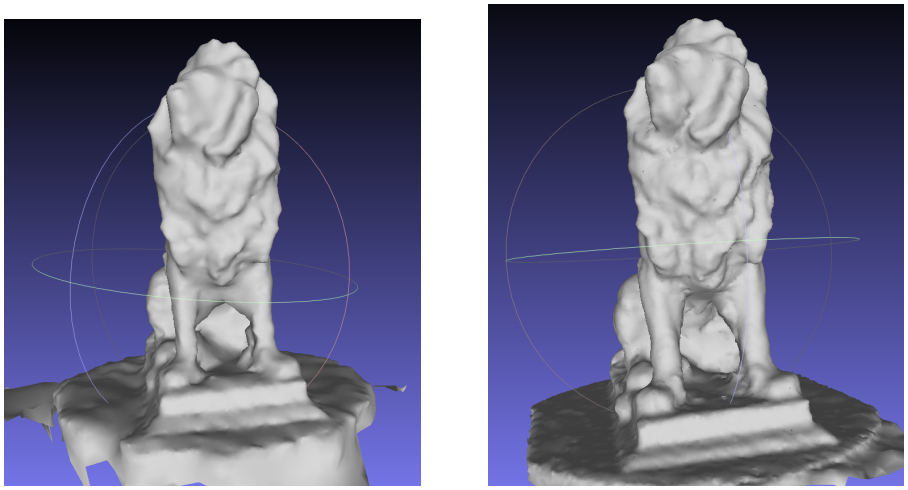
Even with these adjustments to my scanning setup, there were still some objects, that could not be reconstructed. This was mainly because they were not textured enough. Some examples of such objects are the rubber ducky pictured in Figure 4.2 or 3D print of the 3DBenchy model in Figure 4.4. For these objects using a different scanning technique, such as the use of structured light to enhance the geometry of the object, would be recommended.

4.2 Testing effects of objects texture

To further investigate the effects of the texture on the quality of the resulting 3D mesh, I used a 3D printed statue of a lion, which was printed in a single color filament, so its texture was very uniform. First I scanned it as it came from the 3D printer using 100 pictures for the reconstruction. After that I used acrylic paints to give the lion some texture and repeated the scanning process and reconstructed both scans using the same reconstruction pipeline. The resulting 3D models can

Figure 4.4: One of the photos taken from the failed 3DBenchy scan

be seen in Figure 4.5. As can be seen in the pictures, the model created using the textured model retained the features of the original much better than the one scanned without any additional texturing. This can be seen best in the area between the lions front legs, where the right lion scanned with added texture is much more detailed. The texturing also helped with other areas, such as the lions back, where bumps, that were not present on the original model appeared on the scan created without added texture. Unfortunately even with the added texture, the lion lost some detail. Comparison with the original can be seen in figure 4.6.



(a) (b)

Figure 4.5: (a) Scan created without added texture (b) Scan created with added texture

(a) (b)

Figure 4.6: Comparison of scanned model with the original

4.3 Performance of the reconstruction pipeline

When reconstructing the object, the majority of time within the pipeline is spent calculating depth maps for each image. It can be skipped in order to speed up the whole process, but it greatly impacts the quality of the resulting 3D geometry, which would be much less detailed. This is known within the Meshroom community as draft meshing. It can be used to quickly calculate a rough representation of the scanned object. It is also the only part, which is dependent on CUDA, which might be another reason to skip it, if the user doesn't have a CUDA capable GPU.

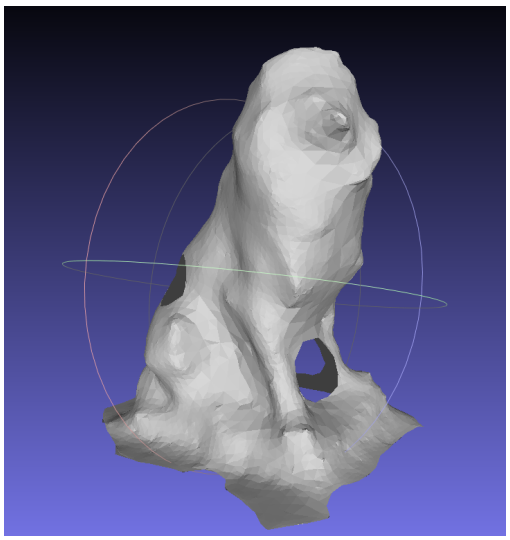


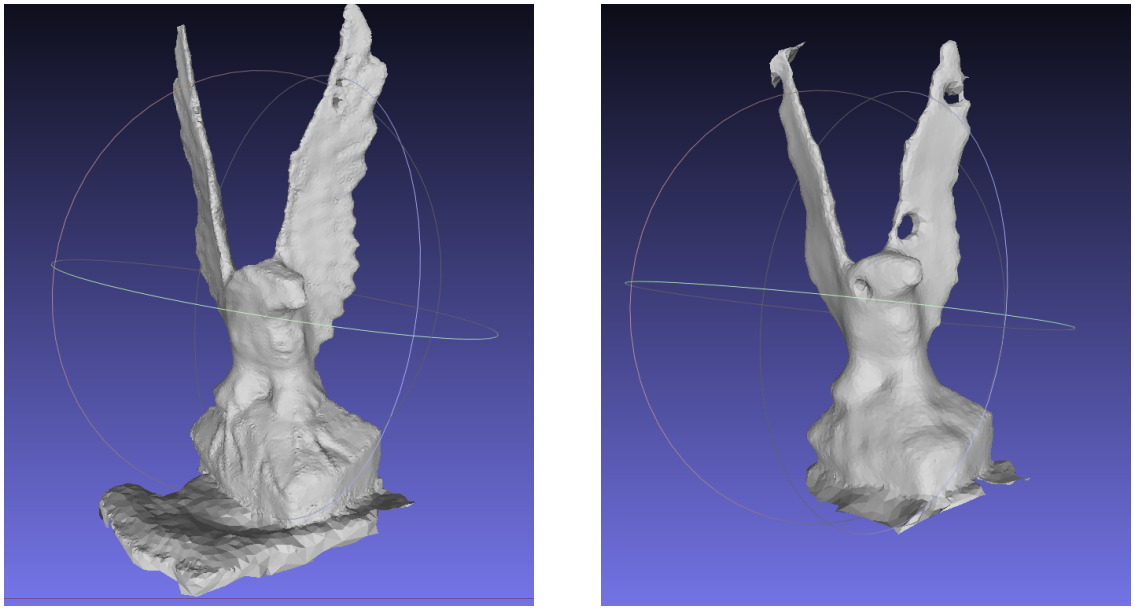
Figure 4.7: Draft meshing of the lion dataset

Result of draft meshing can be seen in figure 4.7, where it has been used on the lion dataset used by figures 4.6 and 4.5. Compared to the models in figure 4.6, it is ease to see, that the model

created with draft meshing isn't as detailed and has holes in places, where the pipeline struggled to detect and match SIFT features.

While testing this pipeline on my computer with a Quadro K610M graphics card with compute capability 3.5 and 1 GB of memory, the part of depth mapping took about two hours to complete with the images downscaled by a factor of two. Without it, the whole pipeline completed in mere minutes.

Speedup can also be achieved by downscaling the depth maps even more, but it affects the quality of the resulting 3D model as well. The effect can be seen in figure 4.8. The first eagle used 2x downscaling of the depth map and took about two hours to reconstruct. The second eagle used 4x downscaling and took about 30 minutes to reconstruct. As can be seen in the pictures, the eagle that was reconstructed with larger downscaling lost more of its details and has large holes in his wings. The second eagle is much more detailed, but also has some minor holes in his left wing.

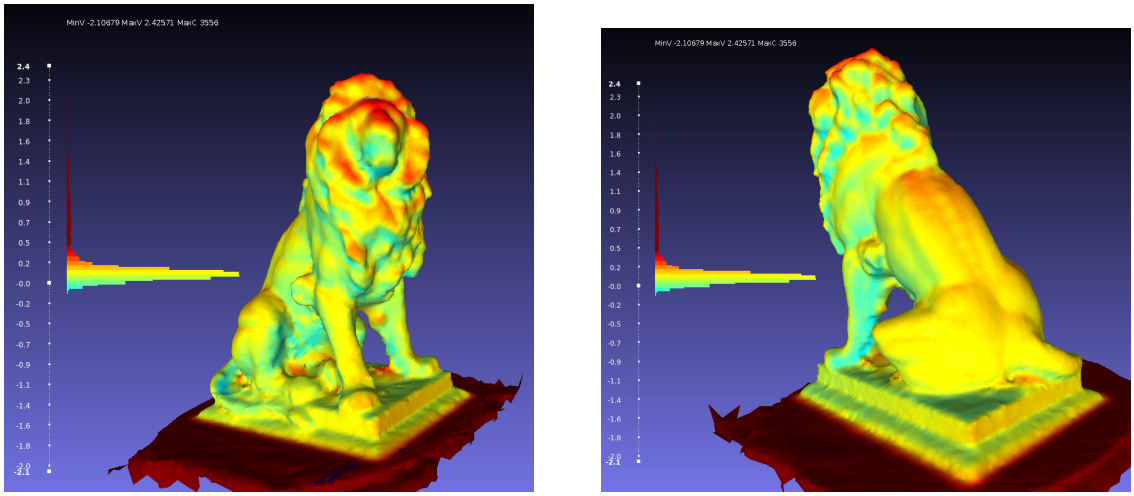


(a) (b)

Figure 4.8: Impact of downscaling the depth map [(a) -> 2x (b) -> 4x]

4.4 Scan accuracy testing

When researching how to test the accuracy of the scanner, I found a scientific paper from Ognjan Lužanin and Irma Puškarević, that focuses on measuring the accuracy of close-range photogrammetry using 3D printed models. [18] The idea of the method is, that the geometry of the 3D model is exactly known, so after 3D printing it and scanning it, the scanned model can be compared with the original CAD geometry to measure the accuracy of the 3d scanner. For the comparison of the



(a) (b)

Figure 4.9: Visualization of quality of the 3D scan on scanned model (histogram units are in centimeters)

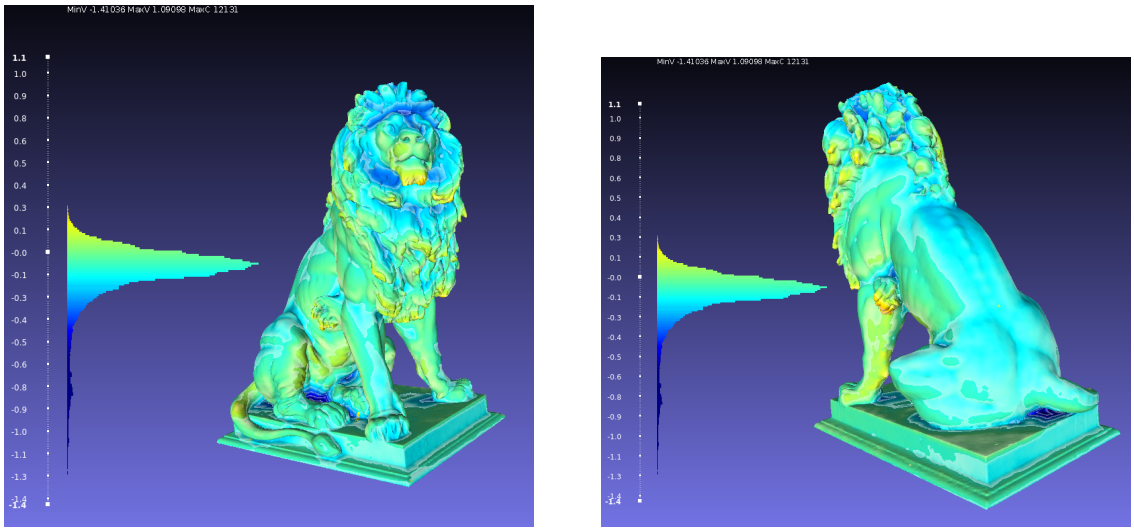
models the authors used Computer Aided Inspection software GOM Inspect, which is normaly used by manufacturing companies to verify the accuracy of manufactured parts and compare them with the original design.

I have decided to use this method to get an idea of the accuracy of the used scanner, but instead of GOM Inspect, which is proprietary and only runs on the Windows platform, I used Meshlab, which offers a similar feature, which calculates a distance from reference mesh for each vertex in the tested mesh.

For testing I used the 3D printed lion to which I manually added texture pictured in figure 4.6, because I could compare the scanned mesh with the original .stl file. It is important to note, that the lion was scaled down before printing and some of the details were lost due to the limitations of the slicing and printing process. After creating the scanned mesh, both the original and the scanned mesh were imported into Meshlab, where they were aligned using the alignment tool and the positions of corners of the pedestal, on which the lion is sitting. After that, the distances of the original mesh and the scanned one could be calculated.

The results can be seen in figures 4.9 and 4.10. The colors represent the distance of each vertex to the closest vertex in the compared mesh. Red color represents, that the vertex in the compared mesh is in front the vertex and blue color shows, that the vertex in the compared mesh is behind the vertex. If the color was green, than the vertices are in the same location.

As the color of the scanned model is mainly yellow and getting more and more red, the higher the vertex is, we can assume, that the scanned model is slightly shorter, than the original. This could be caused by slight misalignment, which caused a difference in scaling of the compared models. The visualization also shows some more problematic areas, where the scanner struggled. These are



(a) (b)

Figure 4.10: Visualization of quality of the 3D scan on original model (histogram units are in centimeters)

located mainly in smaller nooks and crannies, for example under the tail and between the lions legs, that were not visible to the camera for many frames. The loss of quality can also be easily seen by the naked eye.

Chapter 5

Conclusion

In this term thesis I have looked into the available open source 3D scanners, compared them and chose to work with the OpenScan project and improve it by adding the ability to reconstruct the object on the scanner itself. I then build the scanner using my 3D printer, and started on migrating it to the Jetson Xavier NX platform. After experimenting with the available software used by the scanner, I implemented my own solution to control it.

After that I evaluated the available photogrammetry 3D reconstruction solutions, and decide to use the AliceVision framework to create 3D models from the scanned images. I have compiled the framework on the Jetson Xavier NX, but unfortunately one of the dependencies of the framework is using system calls, which are not implemented by the kernel of the Jetsons operating system, which means it does not work on the platform. I installed the AliceVision framework on my desktop computer, where everything works as intended and proceeded with the testing of the scanner there.

I experimented with changing my scanning setup to get a better consistency and quality of the resulting 3D models and speeding up the reconstruction process. I also looked into testing the accuracy of the 3D scanner, which is not very precise. I do not think, that the precision of the used scanner is high enough for the use in industrial tasks, archaeology or any other professional tasks, as the quality of the resulting scans is in my opinion highly restricted by the use of the Raspberry Pi Camera. Using a higher quality camera would remove this bottleneck, but also greatly increase the scanners price. With the scanner being as is, I think it is perfect for use by hobbyist, or in education as it is very simple and cheap to build and the price to performance ration of the project is really good.

Chapter 6

Appendix

6.1 Examples of scans

Figure 6.1: Scan of eagle statue, Reconstructed using 100 pictures

Figure 6.2: Scan of a painted 3D printed chameleon, Reconstructed using 100 pictures

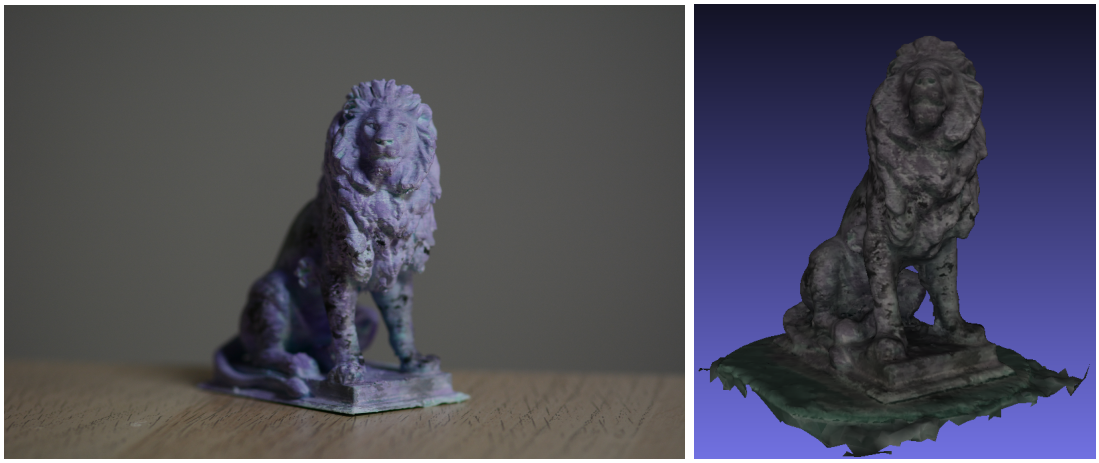


Figure 6.3: Scan of a 3D printed lion with added texture, Reconstructed using 100 pictures

6.2 Running the project

The code for controlling the scanner can be found on my GitHub page. [19] To run it, the user needs to install Python3 (tested with Python 3.6) and OpenCV compiled with support for GStreamer. If the user wants to use the reconstruction feature, he also needs to download and install Meshroom. After cloning the repository from GitHub, the user can run the scanning procedure either by running the python sript main.py in the /src directory, which takes as an argument the path to a scanner configuration JSON file, or by running the run.sh script, which calls the command explained above and uses the configuration file downloaded with the rest of the project.

```
#cloning project
git clone https://github.com/PavelMandrla/SemestralniPrace.git
cd SemestralniPrace

#running the scanning procedure
# using python3
   python3 ./src/main.py ./path_to_config.json
# using bash command
   ./run.sh
```

Listing 6.1: Running the scanning procedure

As stated above, the project needs a configuration file to run. To the user the most important parameters in the configuration file are:

arm_positions which states, in how many arm positions, will the turntable routine be conducted

turn_table_positions which states, how many photos will be taken during each turntable routine

save_dir which specifies directory, in which the scans and scanned images will be saved

reconstruct states, wether the reconstruction pipeline should be executed after the images are taken

path_to_meshroom_batch path to the meshroom_batch executable, which is used to run the reconstruction pipeline

The configuration file looks like this:

```
1  {
2     "arm_angle" :60,
3     "arm_positions" :5,
4     "turn_table_positions" :20,
5
6     "save_dir" :"/home/user/",
7     "path_to_meshroom_batch": "~/Meshroom/meshroom_batch",
8     "reconstruct": true,
9
10    "turn_table" :{
11          "dir_pin" :9,
12          "step_pin" :11,
13          "spr" :3200,
```

```
        "delay" :0.0005,
        "ramp" :200,
        "acc" :200,
        "gear_ratio" :1
    },
    "arm" :{
        "dir_pin" :5,
        "step_pin" :6,
        "spr" :17067,
        "delay" :0.0001,
        "ramp" :200,
        "acc" :200,
        "gear_ratio" :0.5
    },
    "camera": {
        "capture_width" :3280,
        "capture_height" :2464,
        "display_width" :1600,
        "display_height" :1200,
        "framerate" :21,
        "flip_method" :2
    }
}
```

Listing 6.2: Configuration file example

If the user want to reconstruct the model on a different machine, he can do that by installing Meshroom running it and using the default photogrammetry pipeline. [20]

Sources

1. ENGELMANN, Francis. FabScan Affordable3D Laser Scanning of Physical Objects. [N.d.]. Available also from: https://hci.rwth-aachen.de/publications/engelmann2011a.pdf.
2. *Ciclop*. 2017-07. Available also from: https://reprap.org/wiki/Ciclop.
3. *FabScan Open-Source Raspberry Pi based 3D-Scanner*. 2020-09. Available also from: https://fabscan.org/.
4. *The Free 3D Printable Laser Scanning System For the Raspberry Pi*. [N.d.]. Available also from: http://www.freelss.org/.
5. ZUZA, Mikolas. *Fotogrammetrie - 3D skenování s použitím fotoaparátu či mobilu*. 2018-03. Available also from: https://blog.prusaprinters.org/cs/fotogrammetrie-3d-skenovani-s-pouzitim-fotoaparatu-ci-mobilu_7811/.
6. THINGIVERSE.COM. *The $30 3D scanner V7 updates*. 2016-09. Available also from: https://www.thingiverse.com/thing:1762299.
7. THINGIVERSE.COM. *OpenScan - 3D Scanner v2 by OpenScan*. 2018-08. Available also from: https://www.thingiverse.com/thing:3050437.
8. *OpenScan Cloud*. [N.d.]. Available also from: https://en.openscan.eu/openscan-cloud.
9. MOULON, Pierre; MONASSE, Pascal; MARLET, Renaud. Adaptive Structure from Motion with a Contrario Model Estimation. In: *Proceedings of the Asian Computer Vision Conference (ACCV 2012)*. Springer Berlin Heidelberg, 2012, pp. 257–270. Available from DOI: 10.1007/978-3-642-37447-0_20.
10. JANCOSEK, Michal; PAJDLA, Tomas. Multi-view reconstruction preserving weakly-supported surfaces. In: *CVPR 2011*. IEEE, 2011-06. Available from DOI: 10.1109/cvpr.2011.5995693.
11. *Node-RED*. [N.d.]. Available also from: https://nodered.org/.
12. *Fontaine tutorial*. 2017-08. Available also from: https://micmac.ensg.eu/index.php/Fontaine_tutorial.
13. *Presentation*. 2017-06. Available also from: https://micmac.ensg.eu/index.php/Presentation.
14. *OpenSfM¶*. [N.d.]. Available also from: https://www.opensfm.org/docs/index.html.

15. SCHÖNBERGER, Johannes Lutz; FRAHM, Jan-Michael. Structure-from-Motion Revisited. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

16. SCHÖNBERGER, Johannes Lutz; ZHENG, Enliang; POLLEFEYS, Marc; FRAHM, Jan-Michael. Pixelwise View Selection for Unstructured Multi-View Stereo. In: *European Conference on Computer Vision (ECCV)*. 2016.

17. *AliceVision - photogrammetry*. [N.d.]. Available also from: `https://alicevision.org/#photogrammetry`.

18. LUŽANIN, Ognjan; PUŠKAREVIĆ, Irma. Investigation of the accuracy of close-range photogrammetry – a 3D printing case study. *Journal of Graphic Engineering and Design*. 2015, vol. 6, no. 2, pp. 13–18. Available also from: `https://www.grid.uns.ac.rs/jged/download/v6n2/jged_v6_n2_p2.pdf`.

19. MANDRLA, Pavel. *SemestralniPrace - GitHub page*. [N.d.]. Available also from: `https://github.com/PavelMandrla/SemestralniPrace`.

20. ALICEVISION. *Meshroom*. [N.d.]. Available also from: `https://alicevision.org/#meshroom`.