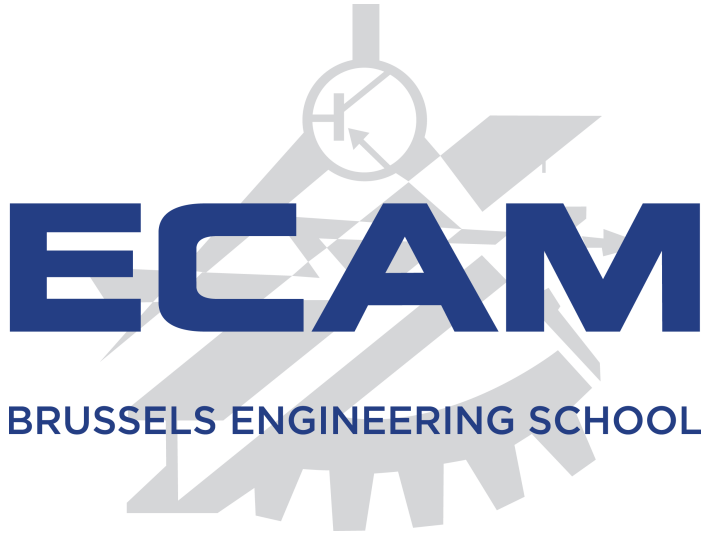

ECAM Brussels Engineering School

2021-2022

Robotic Project:

Stereo Imaging Device

Raphaël	**JADIN**	21295
Gabriel	**LOHEST**	16205
Nicolas	**SAMELSON**	17288

May 21, 2022

Contents

List of Figures

1 Introduction

The first 3D laser scanning device was developed in the 1960s, and consisted of lights, cameras and projectors. Due to the limitations of the equipment at that period, the scans took a lot of time and effort to be roughly accurate. It's only in the end of the 20th century, with the democratisation of computers that the technology took a big leap forward. The methodology carried on improving, by making a scan more accurate, faster, truly three dimensional, and able to distinguish surfaces by color.
Later, with the democratization of 3D printing devices in 2005, the 3D laser scanning market grew even further.

During the Robotic Project course, we focused on creating a 3D scanner device, with the technologies of a laser and 2 cameras. Our objective was to make it open-source, affordable and modular, allowing a large range of compatibility with components.

The scanner is able to scan objects of a maximum size of 14cm in diameter and 15cm in height. The object is placed on a rotating plate, which is controlled by one of the two Raspberry Pi 4. After a scan, the device will transform the images taken in a 3D point cloud through stereovision technology. Finally, the user will be able to export a 3D file (in STL format) which could be imported into a 3D CAD software.

2 Materials and technologies

2.1 Hardware

Below is a list of the components used in this project. As mentioned earlier, the project is open-source and the design is modular. This means that the components could easily replaced by other ones.

List of components :

- 1x 11-inch LCD touchscreen
- 2x Raspberry Pi 4
- 1x 5 mW stripe laser
- 2x Pi Camera v2
- 1x NEMA 17HS15 stepper motor
- 1x A4988 Driver
- 1x AC/DC 5V and 12V power supply
- 1x Custom designed PCB
- 1x Custom 3D-printed assembly
- 2x USB-A to USB-C cables
- Multiple breadboard jumper wires

2.1.1 PCB & wiring

A PCB has been designed to handle all connections between the 2 Raspberry Pis, the stepper motor, the laser and the power supply. It was sketched with the help of the software EasyEDA.

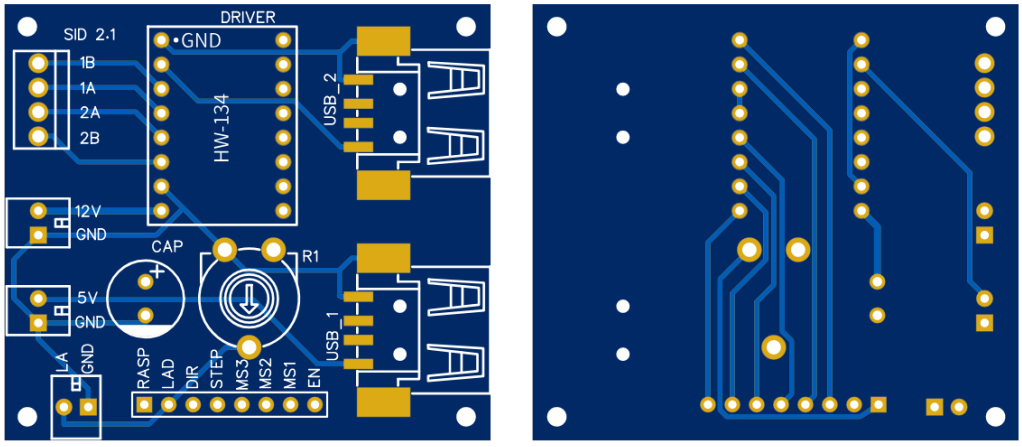

Figure 1: PCB design

On the PCB design at Figure 1, we can find footprints for the A4988 driver, 2 female USB-A ports, a potentiometer, a capacitor, and 5 blocks of connectors :

- 2x 2-pin connectors, used to connect the power supply outputs (5V and 12V)
- 1x 4-pin connector, to plug in the stepper motor
- 1x 2-pin connector, to plug in the laser
- 1x 8-pin connector, to link the Raspberry Pi and control the components.

First, the power supply brings 12V and 5V on the PCB. The 12V is used to power the stepper motor, which accepts a voltage between 9V and 35V. The 12V tracks, which are 0.69mm thick, pass through the driver. A 100 μF capacitor is also connected in parallel to the 12V source (see Figure 2).
The 5V tracks, which are 0.49mm thick, power the driver itself as well as the rest of the components.

The 2 USB ports are used to power each Raspberry Pi through a USB-A to USB-C cable. On the schematic (Figure 2), the USB connectors we used have the GND and VCC pins inverted (the component in red) between the schematic and the actual footprint, and therefore precaution is needed to avoid a short-circuit.

The 8-pin connector is used to control all components with the Raspberry Pi.
The 2 first pins (RASP and LAD) are used to power the laser. LAD connects it directly to the laser connector while RASP is connected in series with a potentiometer. This was originally intended to help the user control the intensity of the laser (with a knob on the potentiometer) in case the control through software wouldn't be reliable (the Raspberry Pi needs to use its Pulse Width Modulation).

The 6 other pins are used to control the stepper motor. The DIR and STEP pins are used to control the direction of the rotation and the instruction to make a step, while MS1, MS2 and MS3 determine what the step size should be (the Nema 17HS15 is able to go down to a 16th of a step thanks to its gearbox [ref. 24].
Finally, the EN pin enables or disables the stepper.

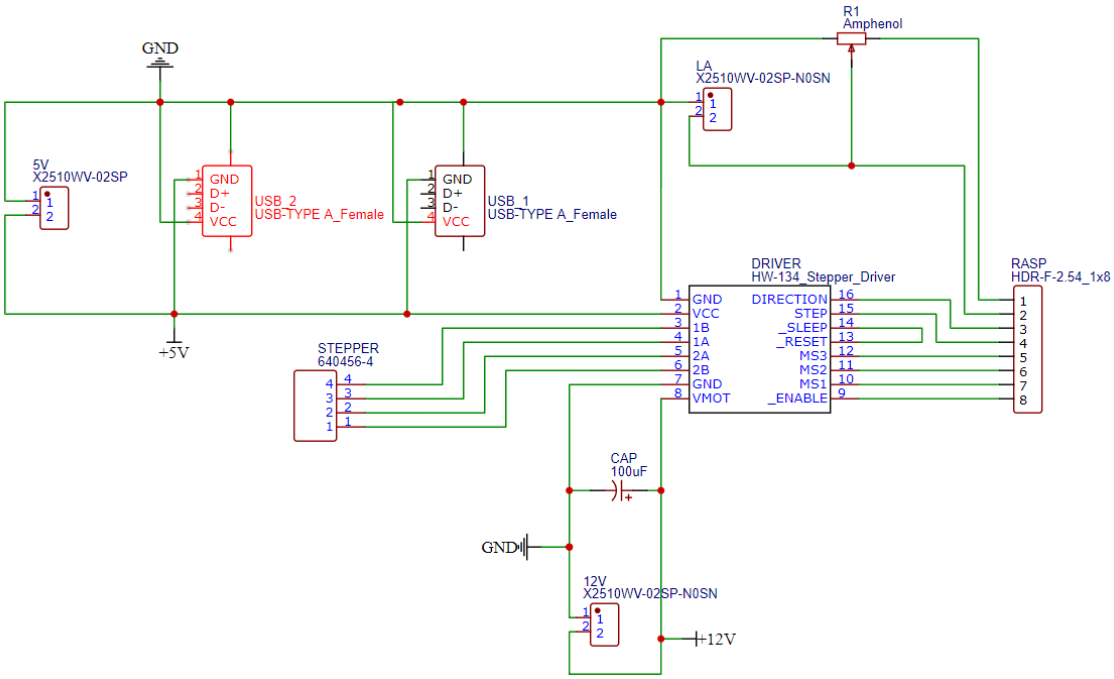

Figure 2: PCB schematic

2.1.2 3D Design & printing

The main body of the project is made of several 3D-printed parts, that have all been designed with Dassault Systemes's SOLIDWORKS software. The initial goal was to be able to integrate all components in a single piece of hardware, so that the project could hold itself together as a whole. This has been achieved, thanks to a modular approach, resulting in a number of individual parts.

A slide rail mounting system has been conceived and separately tested before being integrated to most parts. This solution was chosen in order to make it easier to iterate design changes, to avoid having to print out the whole thing again at every new version. It also happened to be convenient because the 3D printer had a limited footprint. The slide rail mount also eliminated the need for fasteners, like screws and bolts, or even glue, thus making the final assembly an easier process.

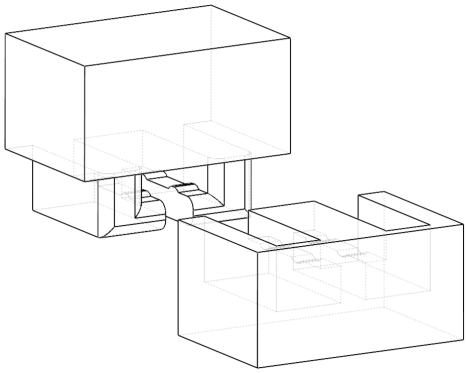

Figure 3: Slide rail mounting system separate design & testing

The assembly is essentially composed of 3 main parts, which are themselves made of subparts. The camera and laser housings, with the supports holding the raspberries in place. This is a nearly symmetrical part, except for a few details.

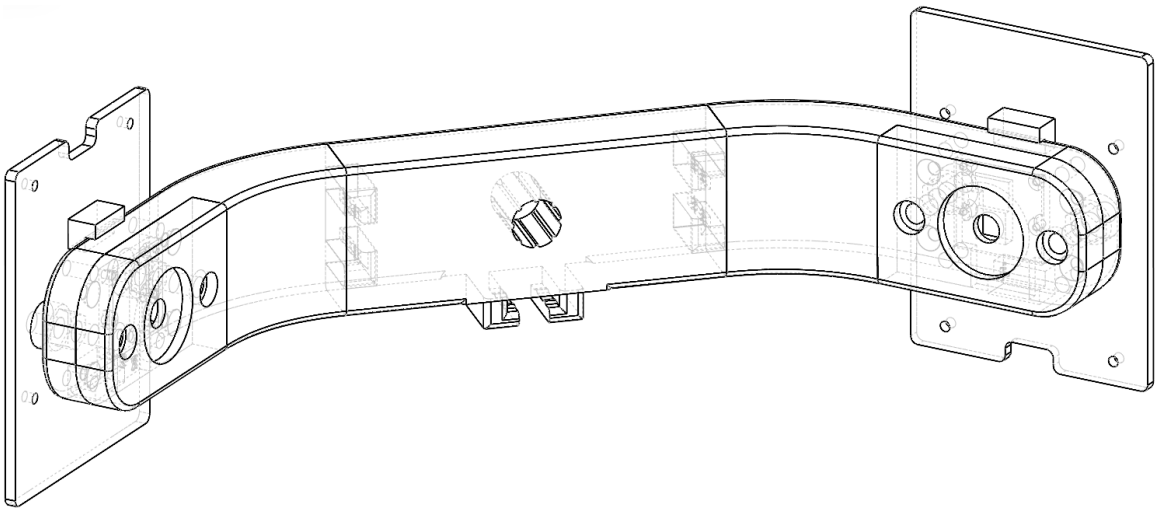

Figure 4: Camera & laser housing, with Raspberry Pi supports

Then, we have a motor mount for the Nema stepper motor, and a rotating table, lying on an appropriate thrust roller bearing, serving as mechanical coupling between the two.

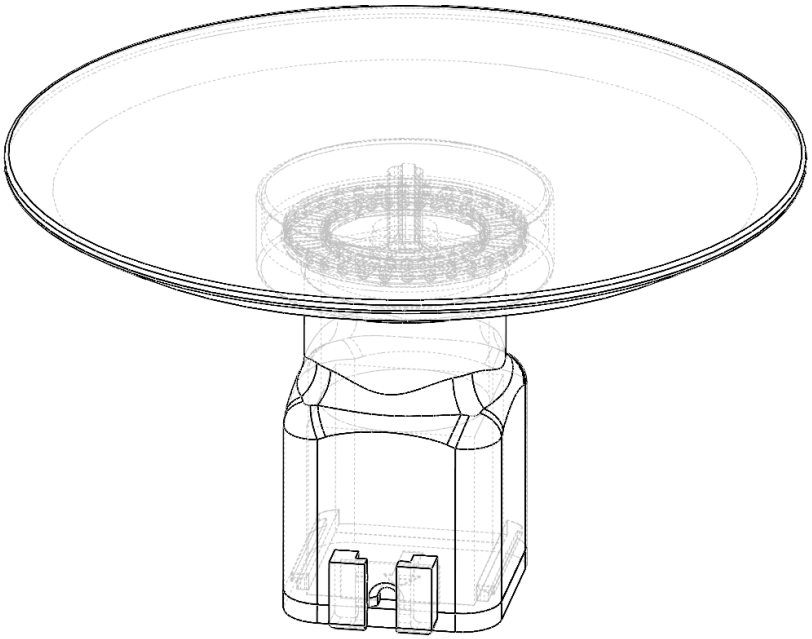

Figure 5: Motor mount & rotating table

Finally, the power supply case holds all the wires, the PCB, as well as the power supply unit. The vertical link part that connects to the camera and laser housing has a groove serving as a guide for the wires.

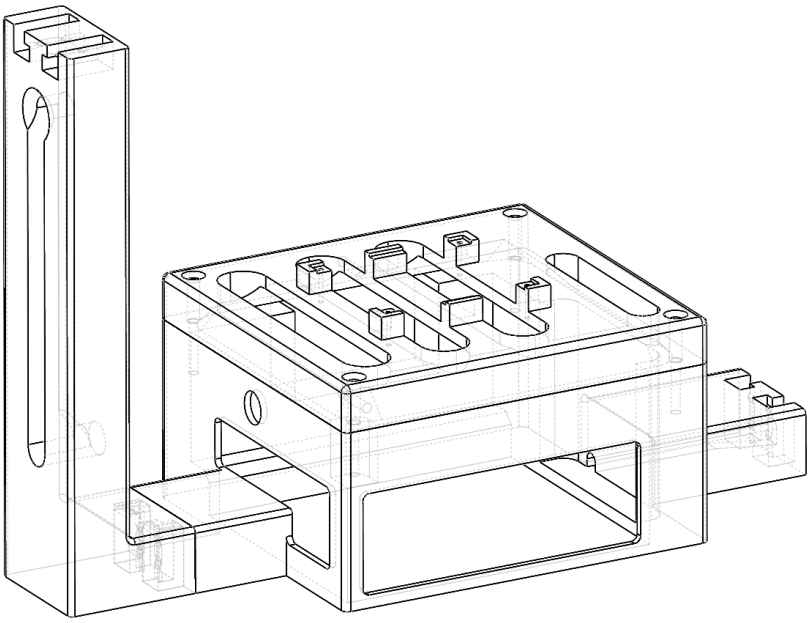

Figure 6: Power supply case & vertical linking element

The final assembly looks like this:

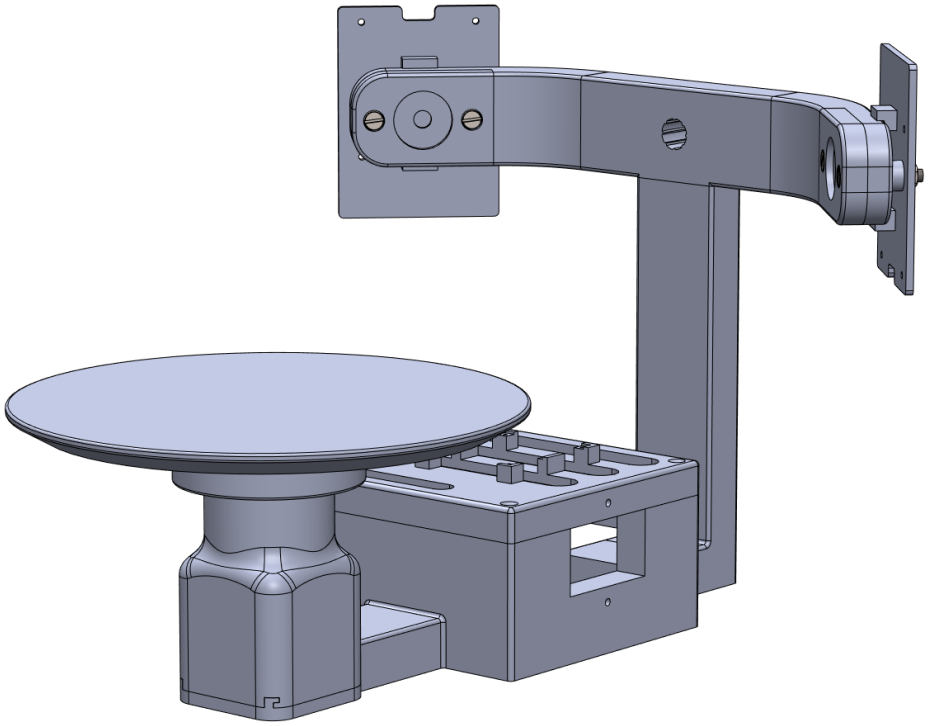

Figure 7: Final assembly

The choice was made not to integrate the touchscreen into the assembly. This is because of its bulky dimensions, but mainly because there were doubts during the development of the project as to whether it was needed or not. However, it ended up being part of the prototype, and can easily be connected and put in position right next to the project assembly, thanks to its integrated folding leg.
All files concerning the 3D CAD are available on GitHub [ref. 22].

2.2 Software

2.2.1 User Interface

For the user to be able to interact with the scanner, we have connected a 11-inch touchscreen to the Raspberry Pi master. In order to create the UI, we used the Kivy framework (based on python) as it offers cross-platform ability, an easy interaction and works well with touchscreens.

The UI is used to start the connection between the 2 Raspberry Pi's together, calibrate the cameras, launch a scan, and convert the scan into a usable 3D file (STL format). A complete description of each page and their functionalities can be found on GitHub [ref. 21].

2.2.2 Communication protocols

Because we are using a second Raspberry pi (to connect the second camera), we used multiple communication protocols in order to take pictures on the slave and also to copy files from the slave to the master.

NFS

NFS, or Network File System, is a protocol that allows a client computer to mount a remote computer's disk to its own file system, in order to access the remote files over the network. In our case, we implemented it to share the home folder of the slave Raspberry Pi over the master Pi. The network connection between both computers is secured by an Ethernet cable.
By accessing the files of the slave Pi via NFS, we can retrieve the images taken during the scan, and considerably faster than over another protocol, like for example, SSH.

SSH

The SSH protocol is widely used to remotely access the terminal of another machine. In our specific case, we used it to send commands to the slave Pi, for it to capture an image when needed.
The goal here is to run the script `prise_image.py`, which is stored on the slave Pi. This script, when run, triggers the capture of a picture with the slave Pi's camera, and then stores that picture on the slave Pi's own file system. One tricky part was to use SSH from a script on the master Pi, instead of directly from the terminal. In order to achieve this, we used the `pexpect` [ref. 18] and `pxssh` [ref. 20] libraries. Another difficulty arose, concerning the robustness; we need to make sure the slave Pi's image has been captured, before the stepper makes the next step. To solve this problem, we used the NFS to check whether the image was successfully taken. If so, the stepper was then instructed to do the next step; if not, the image-taking script was re-run on the slave.

RTSP

RTSP, or Real-Time Streaming Protocol, is a network protocol designed to transport information streams, such as a video live-stream, over a transport protocol (i.e. UDP [ref. 31] or TCP [ref. 29]).
In our case, it is used for the cameras' calibration. The calibration script launches the stream on the slave, through SSH. The master is then able to access it and take video captures.

2.2.3 Image pre-processing, stereovision and point cloud post-processing

This part is composed of three main steps. First, we need to get the image coordinates (i.e. the x and y position relative to the upper left corner on the image) of the pixels which represent the laser projection. Then, based on those points, the stereovision technology is used to get the world coordinates of the points (i.e. the coordinates in a referential given by the OpenCV calibration). Finally, we need to process all the points in such a way that allows to model the scanned object.

Image pre-processing

This is the part where the images taken by the camera are processed in order to isolate the geometry of the laser projection on the object. A binary image[1] is then generated: the pixels are black everywhere, except where the laser projection was detected, where the pixels are white.
To this purpose, the `OpenCV` library was used. This is an open source library developed by Intel, dedicated to image processing. It can be used in C++ or in Python, the latter of which is our case.

Figure 8: Image taken from the slave (Left) and image taken from the master (Right)

On the images at Figure 8, one can observe that the center of the laser line appears to be white. This is due to the intensity of the laser, that saturates the sensors of the cameras. In order to isolate the desired pixels, we took advantage of this. The threshold function of OpenCV was used on all three channels of the image (blue, green and red). This allows to generate these binary images, containing the laser line.

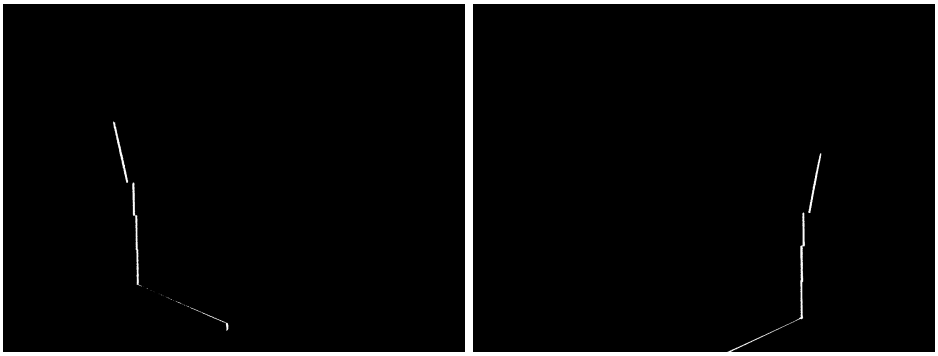

Figure 9: Generated binary pictures

Thanks to these cleaned up images, the actual stereovision processing can now start!

[1] A binary image is an image where the pixels have two possibles values: 0 for black and 255 for white

Stereovision

The following explanation is about one single left and right image pair at a time, but the process is obviously looped in such a way that all image pairs of a scan are processed in this way.
The first step is to narrow the laser line's width down to a single pixel, on the binary image from the left camera, by taking the average coordinates of that width. These coordinates are then stored in the following format: (x = average_x, y = line_number, z = 1).

The next step is to find the corresponding pixels on the binary image from the right camera. This is done thanks to the epipolar geometry [ref. 7].

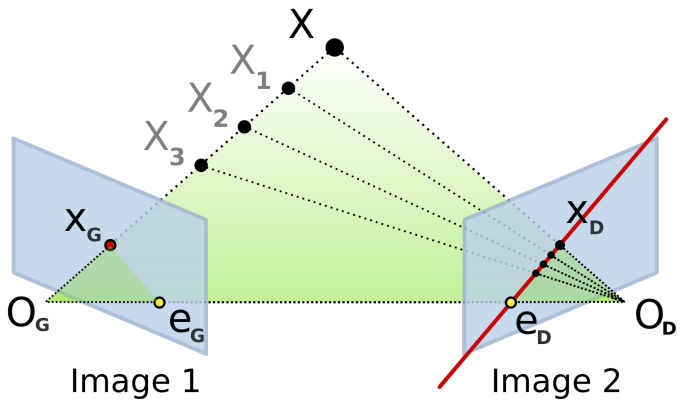

Figure 10: Epipolar geometry

Then, matching the pixels from the left and right images (i.e. X_G on the left image and X_D on the right image, in Figure 10) makes it possible to find the intersection of the $[O_g, X_G]$ and $[O_D, X_D]$ lines, on Figure 10. The coordinates of that intersection are in fact the previously mentioned "world coordinates" of the laser line.
As every pair of left and right images corresponds to a step of the motor, all the world coordinates from a pair of images are stored in a JSON file, and referenced as follows :

$$\{step_n : [[Y_1, Y_1, Z_1, 1], [X_2, Y_2, Z_2, 1], ..., [X_p, Y_p, Z_p, 1]], step_{n+1} : ...\}$$

each point having coordinates in the homogeneous [x, y, z, 1] form.

At this point, the current coordinates do not yet permit the reconstitution of the scanned part. To make this possible, we need to integrate the concept of rotation in order to stitch together all the data from every individual image pair, to be able to create the point cloud model. This is done in four steps :

1. Apply a transformation to move the origin of the coordinate system given by the OpenCV calibration ("world coordinates"), to the center of the table of the scanner.

2. For each step of the motor, and thus each image pair, a rotation has to be applied. The rotation angle for each image pair, or step n, can be computed as follows:

   $$\alpha = step\_angle * step_n$$

   where *step_angle* is the angle covered by a step.

3. Find the ratio between a distance unit in the virtual referential and a millimeter in the real world.

4. Convert the homogeneous coordinates to "real-world coordinates" in a referential with the same origin but where a unit is one millimeter.

Once all these steps are done, every "real-world point" has to be written in a JSON file in the same way than previously explained. This JSON file then contains what we call the point cloud model of the object. The coordinates are organized in such a way that every column is the set of points from a step, i.e. a pair of pictures. This formatting is well suited for the point cloud post-processing.

2.2.4 Point cloud post-processing

The script `create_stl.py` takes care of the transformation of the point cloud into an exploitable STL file. When run, the script first retrieves the point cloud JSON file, and transforms it in a 3-dimensional array of points. The first dimension is the column (each column representing a step in our scan), the second is the row (or the points) and the last dimension contains the x, y and z coordinates of each point.

```
[
    [[1.0, 0.0, 0], [1.0, 0.0, 1], [1.0, 0.0, 2]],
    [[-1.0, 0, 0],  [-1.0, 0, 1],  [-1.0, 0, 2]]
]
```

In the example above, we have a 3D array of 2 columns and 3 points for each column. The 3D array is constructed with the help of the `numpy` library, allowing fast manipulations of arrays.

Before turning the 3D array into a STL file, each column of the point cloud needs to have the exact same amount of rows (or points). This is because a 3D mesh is constructed of triangles (see Figure 11). In our case, 2 triangles are formed from 2 points of 2 consecutive columns. For example on Figure 11, the triangle a and b are based on four points (2 are in common).

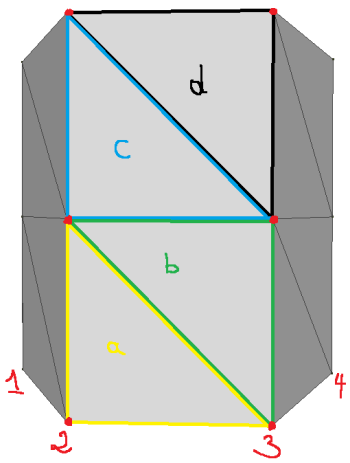

Figure 11: Example of mesh

In order for all columns to have the same amount of points, the array has to be processed by an algorithm that adds missing points where it is needed. We set the reference column as the one with the biggest number of points in it, in order to keep maximum resolution.

This algorithm first checks the endpoints of each column and compares it with its neighboring columns (e.g. we study the column number 2 on Figure 11, its neighbors are the column 1 and 3). As the first item in the column is the point with the lowest z coordinate and the last the highest z coordinate, we can compare those heights with with the endpoints of the neighbors columns. If the difference in height between the studied endpoints (bottom and top of column 2) and its neighbor endpoints (column 1 and 3) are greater than 10%, it fills in a new point at the mean height of its 2 neighbors.

The second part of the algorithm adds missing points between the 2 ends. This is done by calculating the average step the studied column should have, with the following formula :

$$step = lowest + \frac{highest - lowest + 1}{max\_rows}$$

where

- *lowest* is the first item of the column, which has the lowest z coordinate,
- *highest* is the last item of the column, having the highest z coordinate,
- *max_rows* corresponds to the number of points the reference column has.

For a given step, we can iterate through its column and add missing points if the difference between the step and the next point's z coordinate is greater than 10%. After repeating this for each column of the array, they should now all have the same number of points.

We can now transform the 3D array into a 2D array of vertices and faces. The vertices correspond to our points in the 3D array that is flattened into a 2D array [ref. 15].
The faces are an array of the triangles we construct. A single triangle needs 3 points, and for the mesh to show the face in the correct direction, we need to order the 3 points as to draw it clock-wise (if we were to join the 3 points together).

Now all the points (vertices) are linked to triangles. But the top and bottom of the object are still not connected. To solve this problem, we can simply add a point in the middle at the top of the piece (see Figure 12) and draw triangles between the center and 2 points at the top of their respective column. The same was done for the bottom.
Finally, with the help of the `numpy-STL` library, we can create a 3D mesh and save it in STL format.

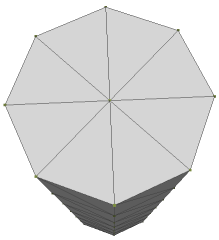

Figure 12: Top view of a cylinder

3 Project timeline

3.1 Sprint 1

- design a PCB for the driver;
- first version of 3D design for laser and camera support;
- establish the connection between the slave Pi and the master Pi;
- validate and command the pcb;
- taking images from both rapsberry pi and transfert the images from slave Pi to master Pi;

3.2 Sprint 2

- first verion of the U.I.;
- design the box for the stepper motor and the scanner's table;
- print the box for the stepper motor;
- calibrate the cameras;
- import the U.I. on the master PI;
- test and weld the components on the pcb;
- finalize the U.I.;

3.3 Sprint 3

- make the stepper operational;
- insert the code of the stepper into the U.I.;
- 3D design : attached the slide rail;
- insert the scripts for the scan in the U.I.;
- finalize the pcb and the wiring;
- new supports for the camera and the stepper motor.

3.4 Sprint 4

- achieve a scan of an object and get pictures from the slave;
- extract the pixels of the laser and get their world coordinates;
- make a json with the coordinates of the extracted points from the scan and format it properly;
- all the 3D pieces are printed and fits together;

- image processing code refactored to make it more extensible;
- create an stl file from a point cloud
- cable management is done properly;
- camera calibration is runable from the interface.

4 Demonstration

A piece was placed on the center of the table (see Figure 13). We launched a scan which took a 100 photos (for each camera). Here is the result of the 3D mesh in a STL format at Figure 14.

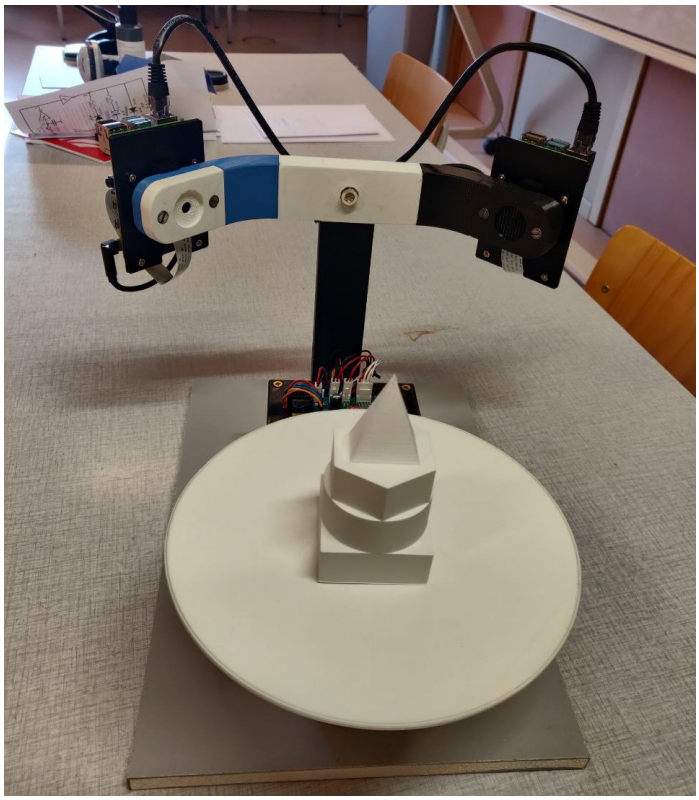

Figure 13: SID photo

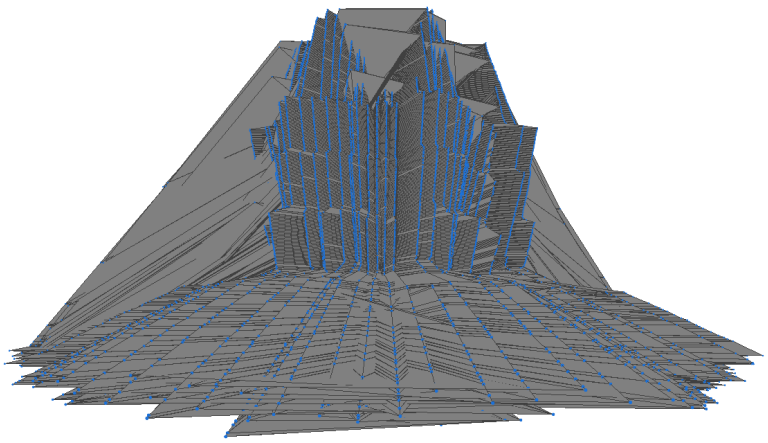

Figure 14: 3D mesh of the scanned piece

One can see that the table has not been removed from the model.

5 Critics & alternatives

5.1 Initial specifications retrospective

The goal of this project was to produce a device able to precisely scan an object of small size (14cm in diameter, and 15cm in height) thanks to its cameras, and output a usable 3D file. The objective has been partially met. The device is indeed able to scan an object and output a 3D file, (Figure 14) but due to poor calibration, the accuracy and precision of the scan is not in par with the expectations, even if every part is working properly.

5.2 Difficulties encountered

During this project, we faced many difficulties that we tried to resolve, some of which couldn't be overcome, so we could not fulfill some of the specifications we presented during Q1 of this year.

Kivy framework

First, on the software side, we encountered many difficulties with the Kivy framework.
One of them was that the framework is a single threaded application. So, when we run a scan, the whole app is blocked, waiting for the script to end. Due to the limited time, we have not been able to explore and implement multi-threading on Python.
Also, for an unknown reason, Kivy was handling 2 video feeds and the implementation of OpenCV's video capture very badly.

Communication between the Raspberry Pis for image capture

Other protocols than the ones used (SSH) were studied, to try to improve the latency, quality and time between each scan, but the latency remained roughly the same as the RTSP (over 8 seconds and very inconsistent). So, we decided to run SSH commands in a loop, which gave more control of the moment of the image is captured. We concluded that the main limitation was the Kivy framework as outside of the app it could run more smoothly. And as we couldn't get a confirmation that the 2 video feeds had the same latency, we kept using the SSH protocol.

Calibration

In order to use the epipolar geometry, we need several parameters about the system, like the cameras' optical center coordinates, their projections matrices and the fundamental matrix of the system. This is done thanks to the calibration function of OpenCV. The problem is that this function takes quite a lot of time to master. Hence, since this wasn't the only time consuming task, it turns out we did not have enough time to do it properly so our calibration isn't totally correct. This results in some error in our model of the scanned object. This error can easily be removed with a better calibration. One part of the problem here was that in order to get a good calibration, we need the (very) precise physical location (relative to the rotating table) of the bottom left corner of the chessboard that is put in place when proceeding to make the calibration. This is further discussed in the subsection 5.3.

Optimisation of the computations

Since the Raspberry Pi has limited RAM, we needed to optimize our computations in order to avoid memory saturation. For the image processing and the stereovision, we kept this in mind at all times when developing, due to the amount of data to be processed. Special care was also put into the robustness and the extensibility of the code.

Point cloud post-processing

The algorithm responsible for turning the point cloud into a 3D mesh was coded from scratch, and as we were limited in time, it is quite basic compared to other existing algorithms. Furthermore, it is handling the points in a somewhat naive way, and does not remove any aberrant point (with absurd coordinates). This is one of the reasons why the final result is not very conclusive.

5.3 Improvements

With the many issues we encountered during development, we thought about multiple improvements we could bring if we had more time to put into the project.

Communication protocols

First, we would dedicate more time into finding a reliable solution concerning communication protocols. Indeed, outside of the Kivy environment, our streaming tests over the TCP protocol were very concluding (less than a second of latency and very consistent). We would have liked to implement this, after finding a way to make sure the pictures were taken at the same time. This could potentially improve the speeds of the scanning process by factor of about 10, which would make a scan with the highest resolution (4000 steps) reasonably fast ($\sim$ 2h, vs $\sim$ 20h currently).

Calibration

The calibration could be enhanced in different ways. A good way to improve it could be to dedicate a whole project to it, since the calibration is a quite demanding task by itself, both in terms of time and expertise. The goal of that project would be to automate the calibration with a homemade script, eliminating the need to manually measure the offset between the center of the calibration's reference frame and the center of the rotating plate.

Image processing

One of the biggest improvements we could achieve concerning the image processing would be to remove the scanner's table from the detected points. Indeed, if the scanned object is smaller than the table, part of the table is also going to be included in the model, which is unwanted.

Another reason to remove the table points while proceeding to the pre-processing, is that these points being useless, it makes sense not to process them, to avoid consuming resources unnecessarily.
A good way to achieve this would be to take an image with each camera of an empty table and the laser turned on. Then, by comparison between the empty table and the images of the scan, it should be possible to remove the undesired data from the scan.

Point cloud post-processing

As mentioned in the subsection 5.2, the algorithm is rather primitive, and cannot handle aberrant points. We would suggest to take more time in tweaking the algorithm, improving its robustness, smoothing surfaces, etc.
An alternative to this would be to use yet another python library (e.g. Open3D library [ref. 10]) , which handles all the algorithms necessary to build a 3D mesh.

Turntable
Another good idea would be to cover up the turntable with rubber, in an effort to increase the friction between the scanned object and the turntable. This can be useful to make sure the object doesn't skid around every time the stepper motor makes a rotation impulse. This could also be helpful for the image processing responsible for removing the turntable from the scanned points.

5.4 Alternatives

As mentioned earlier, we had a lot of issues related to the communication protocols, as consequence of the design constraint of using 2 Raspberry Pis 4 (to be able to use a second camera, since a Raspberry Pi only has one single camera port).
In the specifications we presented on Q1 this year, we first wanted to use a single Raspberry Pi 4, along with a multi-camera adapter [ref. 11]) , making it possible for a single Pi to hande both cameras. There were also other alternatives, like using a Compute Module 4, webcams over USB, or even the Arducam synchronized stereo camera HAT. However, due to the chip shortage during 2021, most of those components were out of stock until later in 2022.

If we had to start the project again from the beginning, we would definitely use one of the simpler single-Raspberry Pi alternatives mentioned above.

There are different ways to achieve a 3D scan with cameras. Another very promising method is the one with a single camera and a laser. This would of course require to precisely quantify the geometry (position, orientation) of the plane defined by the laser beam.

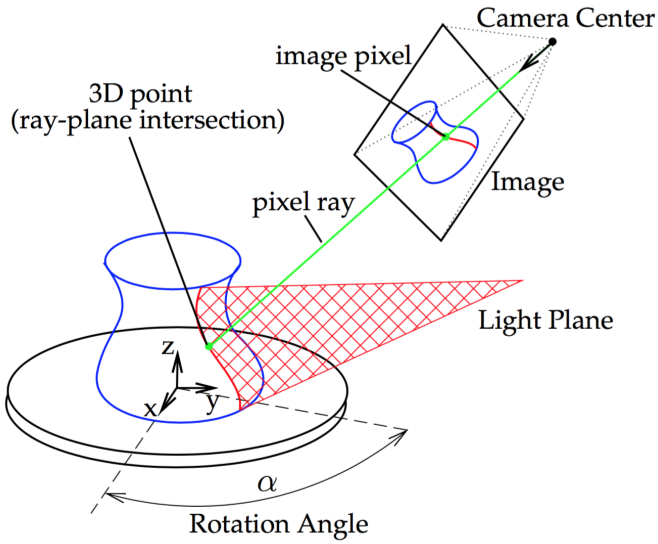

Figure 15: Schema of the application with one laser and one camera

Let's define the line going through both the camera center and the image pixel, as the pixel ray, as displayed on Figure 15. By finding the intersection of that pixel ray with the laser light plane, one can easily construct the world coordinates of the point on the object ("3D point" on Figure 15). What follows is the same as our implementation, concerning what happens next with the world coordinates.

This method would drastically reduce the costs of our project, without negatively impacting its precision. Moreover, as there is a single camera, the overall complexity of the project would be far lesser, and way cheaper.

6 Conclusion

This project has been quite an enriching experience for us, as it allowed us to expand our knowledge and engineering skills. We have learned a lot in many domains.
First, the management aspect of a project taught us many things about the importance of planning and prioritizing tasks. Looking back on the overall organization, we regret not having followed the planning work accomplished on Q1, which was completely decoupled from the manner we actually proceeded. It is however safe to say that the division of labor between the partaking members was done quite right, considering each and every one's strengths, even though some of us would have liked to get involved somewhat more in other tasks.

We also noticed that we took the biggest decisions when our knowledge of the subject was at its lowest. This has helped us understand the value of studying the subject before launching the project.

We have acquired technical skills that we would not have been able to develop in such depth during our school, career without participating in this kind of project. This is likely to serve us as a great asset in the future.

In closing, even though the end result is far from the original expectations, we managed to check all the specified tasks and release a functional prototype.

References

[1] *(PDF) 3D LASER SCANNERS: HISTORY, APPLICATIONS, AND FUTURE*. URL: https://www.researchgate.net/publication/267037683_3D_LASER_SCANNERS_HISTORY_APPLICATIONS_AND_FUTURE (visited on 05/21/2022).

[2] *3D LASER SCANNERS: HISTORY AND APPLICATIONS*. URL: https://www.actasimulatio.eu/issues/2018/IV_2018_01_Edl_Mizerak_Trojan.pdf (visited on 05/21/2022).

[3] Aish Dubey. "Stereo vision—Facing the challengesand seeing the opportunities for ADAS applications". In: (2020). URL: https://www.ti.com/lit/wp/spry300a/spry300a.pdf (visited on 12/28/2021).

[4] Clayton Darwin. *Distance (Angles+Triangulation) - OpenCV and Python3 Tutorial - Targeting Part 5*. 2018. URL: https://www.youtube.com/watch?v=sW4CVI51jDY (visited on 12/26/2021).

[5] dtrewren. *Ciclop 3D Scanner (BQ & Horus)*. Instructables. URL: https://www.instructables.com/Ciclop-3D-Scanner-BQ-Horus/ (visited on 12/29/2021).

[6] *Epipolar geometrie*. URL: https://upload.wikimedia.org/wikipedia/commons/9/9d/Epipolargeometrie-fr.svg (visited on 05/21/2022).

[7] *Epipolar Geometry*. URL: https://www.cs.toronto.edu/~jepson/csc420/notes/epiPolarGeom.pdf (visited on 05/21/2022).

[8] FU-Fighters. "Stereo vision with a single camera". In: (2004). URL: http://www.inf.fu-berlin.de/lehre/SS06/Robotik/stereo_vision_newest.pdf (visited on 12/20/2021).

[9] Enric Meinhardt Gabriele Facciolo Carlo de Franchis. "MGM: A Significantly More Global Matching for Stereovision". In: (2015). URL: http://dev.ipol.im/~facciolo/mgm/mgm.pdf (visited on 12/29/2021).

[10] *Generate 3D meshes from point clouds with Python — Towards Data Science*. URL: https://towardsdatascience.com/5-step-guide-to-generate-3d-meshes-from-point-clouds-with-python-36bad397d8ba (visited on 05/21/2022).

[11] *How to attach multiple cameras to Raspberry Pi - Arducam*. URL: https://www.arducam.com/docs/cameras-for-raspberry-pi/multi-camera-adapter-board/introduction/ (visited on 05/21/2022).

[12] *Kivy: Cross-platform Python Framework for NUI Development*. URL: https://kivy.org/#home (visited on 05/21/2022).

[13] *Machine-Vision-Applications*. GeT Cameras, industrial vision cameras and lenses. URL: https://www.get-cameras.com/Machine-Vision-Application (visited on 05/21/2022).

[14] *Network File System - Wikipedia*. URL: https://en.wikipedia.org/wiki/Network_File_System (visited on 05/21/2022).

[15] *numpy.ndarray.flatten — NumPy v1.22 Manual*. URL: https://numpy.org/doc/stable/reference/generated/numpy.ndarray.flatten.html (visited on 05/21/2022).

[16] *OpenCV: Depth Map from Stereo Images*. URL: https://docs.opencv.org/4.x/dd/d53/tutorial_py_depthmap.html (visited on 12/29/2021).

[17] M. Pesce et al. “A Low-cost Multi Camera 3D Scanning System for Quality Measurement of Non-static Subjects”. In: *Procedia CIRP*. 3rd CIRP Global Web Conference - Production Engineering Research 28 (Jan. 1, 2015), pp. 88–93. ISSN: 2212-8271. DOI: 10.1016/j.procir.2015.04.015. URL: https://www.sciencedirect.com/science/article/pii/S2212827115002772 (visited on 12/23/2021).

[18] *Pexpect version 4.8 — Pexpect 4.8 documentation*. URL: https://pexpect.readthedocs.io/en/stable/ (visited on 05/21/2022).

[19] *Pololu - A4988 Stepper Motor Driver Carrier*. URL: https://www.pololu.com/product/1182 (visited on 12/26/2021).

[20] *pxssh - control an SSH session — Pexpect 4.8 documentation*. URL: https://pexpect.readthedocs.io/en/stable/api/pxssh.html (visited on 05/21/2022).

[21] Nicolas Samelson, Raphael Jadin, and Gabirel Lohest. *16205/Sid_project: Projet Robotique - Scanneur SID*. GitHub. URL: https://github.com/16205/Sid_project/tree/main (visited on 05/21/2022).

[22] Nicolas Samelson, Raphael Jadin, and Gabirel Lohest. *16205/Sid_project: Projet Robotique - Scanneur SID - 3D CAD Branch*. GitHub. URL: https://github.com/16205/Sid_project/tree/3D-printed-hardware-CAD (visited on 05/21/2022).

[23] Stephen Se and Piotr Jasiobedzki. “Stereo-vision based 3D modeling and localization for unmanned vehicles”. In: *International Journal of Intelligent Control and Systems* 13 (Mar. 18, 2008). URL: https://www.researchgate.net/publication/228618177_Stereo-vision_based_3D_modeling_and_localization_for_unmanned_vehicles.

[24] *Stepper motor 17HS15*. URL: https://www.omc-stepperonline.com/download/17HS15-1684S-HG10.pdf (visited on 05/21/2022).

[25] Super Make Something. *DIY 3D Scanner (Arduino, 3D Printing, PCB Design, Stepper Motors, IR Sensing) - Super Make Something*. Mar. 14, 2016. URL: https://www.youtube.com/watch?v=-qeD2__yK4c (visited on 12/28/2021).

[26] The Coding Lib. *Depth Estimation Using Stereo Vision - Computer Vision Project with OpenCV C++ Code*. 2021. URL: https://www.youtube.com/watch?v=snJVyfl9ZMg (visited on 12/20/2021).

[27] *The FabScan Project - Media Computing Group - RWTH Aachen University*. URL: https://hci.rwth-aachen.de/fabscan (visited on 12/29/2021).

[28] *TI4L-L1 - Traitement d'images 1 (2021-2022) - General - All Documents*. URL: https://bit.ly/3NqGWqF (visited on 05/21/2022).

[29] *Transmission Control Protocol*. In: *Wikipedia*. Page Version ID: 1088448616. May 18, 2022. URL: https://en.wikipedia.org/w/index.php?title=Transmission_Control_Protocol&oldid=1088448616 (visited on 05/21/2022).

[30] *What is a stereo vision camera? — Camera blog*. URL: https://www.e-consystems.com/blog/camera/camera-board/what-is-a-stereo-vision-camera/amp/ (visited on 12/20/2021).

[31] *What is User Datagram Protocol (UDP)? Definition from SearchNetworking*. URL: https://www.techtarget.com/searchnetworking/definition/UDP-User-Datagram-Protocol (visited on 05/21/2022).

[32] Zhiyi Zhang Zhihua Lv. “Build 3D Scanner System based on Binocular Stereo Vision”. In: (2012). URL: https://cie.nwsuaf.edu.cn/docs/20170614173931092961.pdf (visited on 12/20/2021).