Design and Development of a Complete End-to-End Software Solution for Lightcam 3D Printers

Abel Elbaile Serra

Facultat d'Informàtica de Barcelona
Università “La Sapienza” di Roma

March 2015
Thank you, ...

Aleix Tejada
...for his never ending support and help at all times, and for his amazing work with the Lightcam SLA printer.

Professor Andrea Vitaletti
...for accepting my thesis and giving me the chance to work at an amazing lab with great people.

Ugo Colesanti and the rest of the lab team
...for being so nice with me since the beginning, and for their support and help in tough times when the printer kept failing to work properly.

My parents, Agustin and Elisabeth
...for all the love and support, and for providing me the means to reach this point.

Friends
Barcelona, Rome, Zagreb… Everywhere I’ve met amazing people who have kept me going, have made me push harder and harder, even if they weren’t really sure of what I was actually building. I would have never reached this point without them.

FIB Vicedean and Mobility Staff
...for their infinite patience with me, my Erasmus, and my project. For their cooperation and their understanding when I didn’t meet deadlines. They really are part of this project, as I pretty much failed to deliver it according to what was initially expected.
Executive Summary

3D printing technology has been experiencing a significant growth during the past few years. New technologies are being introduced every day and as interest rises, demand of these new technologies also increases, thus lowering prices and making them affordable for all kinds of new, private users which may range from small engineering companies all the way to hobbyists and home tinkerers. This opens a whole new world full of possibilities, both for end users and for companies who want to address this issue in a wise and intelligent manner.

Having successfully completed the design and construction of a fully functional prototype of the first Lightcam SLA 3D printer, this project aims to entirely describe the development of all software required to allow end users to materialize their digital parts in an easy and efficient way. On this paper, we will discuss both the printer firmware, which runs on an ARM microcontroller architecture, and the client software, which is being developed in C++ using Qt libraries for cross-platform development and deployment on PC’s and Mac’s. Nevertheless, a brief introduction about the story of the actual building of the printer will be presented.
Table of Contents

Executive Summary .................................................................................................................... 3

Table of Contents .................................................................................................................... 4

1. Introduction .......................................................................................................................... 7
   1.1 What is 3D printing? ........................................................................................................ 7
   1.2 What role does software play in 3D printing? .............................................................. 8
   1.3 What is Lightcam? ......................................................................................................... 9
      1.3.1 Why build yet another 3D printer? .................................................................... 9
   1.4 Lightcam development overview .............................................................................. 10

2. End-User Software: Project definition .................................................................................. 18
   2.1 Scope ............................................................................................................................ 18
   2.2 Objectives .................................................................................................................... 18
   2.3 Requirements ............................................................................................................... 18
      2.3.1 Functional requirements .................................................................................... 18
      2.3.2 Non-functional requirements .............................................................................. 19
   2.4 Risks ............................................................................................................................ 19
   2.5 Use cases ....................................................................................................................... 20
      2.5.1 Open print job from disk ..................................................................................... 21
      2.5.2 Save print job to disk ......................................................................................... 21
      2.5.3 Change print job settings .................................................................................... 21
      2.5.4 Send job to printer ............................................................................................... 22
      2.5.5 Import part ........................................................................................................... 22
      2.5.6 Select part ............................................................................................................. 22
      2.5.7 Duplicate part ...................................................................................................... 23
      2.5.8 Move part .............................................................................................................. 23
      2.5.9 Rotate part ............................................................................................................ 23
      2.5.10 Scale part .......................................................................................................... 24
      2.5.11 Align part face to surface .................................................................................. 24
2.5.12 Generate support material ..................................................25
2.5.13 Remove support material ....................................................25
2.5.14 Change support material settings .......................................25
2.5.15 Rotate camera ......................................................................26
2.5.16 Change zoom ......................................................................26
2.5.17 Change view type .................................................................26

3. End-User Software: Architecture ..............................................27

3.1 Conceptual schema ..................................................................27

3.2 Class description ......................................................................28
3.2.1 MainWindow ........................................................................28
3.2.2 GraphicsView .......................................................................29
4.2.3 Scene ..................................................................................29
3.2.4 SideMenu ............................................................................30
3.2.5 PartListView ..........................................................................30
3.2.6 Platform ...............................................................................31
3.2.7 PlatformData .........................................................................31
3.2.8 Part ......................................................................................31
3.2.9 stl_file ................................................................................32
3.2.10 stl_stats ............................................................................32
3.2.11 Supports .............................................................................33
3.2.12 Build ..................................................................................33
3.2.13 Job .....................................................................................33
3.2.14 ResinType ..........................................................................34
3.2.15 Printer ................................................................................34
3.2.16 PrinterManager .................................................................34
3.2.17 USBWatcher ......................................................................35

3.3 Third-party software used .......................................................35
3.3.1 Qt Framework ......................................................................35
3.3.1.1 Signals and slots......................................................................................36

3.3.2 libusb........................................................................................................36

3.3.3 adMesh....................................................................................................36

3.3.4 Slic3r........................................................................................................37

4. Printer Firmware: Project definition.................................................................38

4.1 Scope.............................................................................................................38

4.2 Objectives.....................................................................................................38

4.3 Requirements ..............................................................................................38

  4.3.1 Functional requirements ........................................................................38

  4.3.2 Non-functional requirements ..................................................................39

5. Printer Firmware: Architecture .......................................................................40

5.1 Conceptual schema.......................................................................................40

5.2 Class description..........................................................................................41

  5.2.1 Main..........................................................................................................41

  5.2.2 config.h....................................................................................................41

  5.2.3 utils.h......................................................................................................42

  5.2.4 ZAxis..........................................................................................................42

  5.2.5 StatLog......................................................................................................42

5.3 Third-party software used...............................................................................42

  5.3.1 Arduino SPI implementation......................................................................42

  5.3.2 sdfatlib....................................................................................................43

  5.3.3 AccelStepper............................................................................................43

6. Conclusion.......................................................................................................43
1. Introduction

1.1 What is 3D printing?

3D printing, also known as additive manufacturing, is a process by which a three-dimensional digital model is made into a physical object, usually by laying down successive layers of material until the object is completed. Machines used to perform this task are usually controlled by computers, which run a program that tells the machine what movements and/or actions to take.

There are a few methods and technologies that are being used today. For the purpose of this document, it is worth introducing three of them: fused deposition modelling, selective laser sintering, and stereolithography. These are different approaches that try to solve the same problem in similar ways, but use different techniques and materials.

In fuel deposition modelling, “FDM” from now on, a plastic -usually- filament is pushed through a hot nozzle that melts it and extrudes the melted plastic while moving, laying it accordingly on top of the previous layer. The extruder containing the nozzle has to be able to move on an XYZ cartesian coordinate system, so it needs three axes of movement. On most designs it’s usually placed on a carriage that can travel to any position in an X-Y cartesian plane, either in X-Y or more commonly X-Z directions. The remaining axis, either Y or Z, is left to the bed, where the fused material is deposited.

FDM is probably one of the most versatile technologies in terms of materials for printing with. The most common are ABS and PLA plastics, but there are now other options such as PVA, nylon, wood*, or even edible materials, such as melted chocolate, which are extruded with a syringe. Nevertheless, part quality and precision are probably the weakest point, as we’ll explore later.

Selective laser sintering, from now on referred to as “SLS”, works quite differently. In this case, polyamide powder is transferred into a container for it to be hardened by a laser that draws cross sections of the part on each successive layer. On each layer change, a re-coating mechanism adds more powder for the following layer. One of the main benefits of this approach is that the powder that is not hardened not only acts as support material for further layers, but is also reusable for future prints.

SLS offers much higher quality results as it is usually less prone to mechanical error. Instead of having a carriage moving an extruder, here the laser is fixed and a laser scanner system is in charge of reflecting the beam in X and Y directions. Galvanometers are used in laser scanners to control closed-loop feedback servos. These are hard to drive robustly, from an electronic point of view, and must be calibrated correctly before using.

Stereolithography, or “SLA”, is a similar process to SLS but, instead of hardening powder, liquid resin is used as polymer. There is a build platform that is first aligned to the resin level for the first layer and then sinks into the resin tank while successive layers of the part are executed. Machines that work the other way around also have been proven to work. The same laser and scanner system present in SLS machines is also present in SLA machines. Regarding the laser, the wavelength must be carefully chosen, together with the desired resin, for them to be
compatible with each other. Common wavelengths are usually near the UV-A / visible light frontier of the light spectrum. The resin must be chosen accordingly, paying special attention to the wavelength it needs to catalyse and harden.

Part quality in SLA is usually even better than SLS. In terms of dimensions and tolerances, very similar results of those from SLS can be observed. But, visually, parts are much more appealing. Faces come out perfectly flat and smooth, whereas in SLS surfaces are also flat but rough. SLA does, however, require the generation of support material together with the actual part that’s being printed.

1.2 What role does software play in 3D printing?

There are multiple pieces of software that are used on a day to day basis in the 3D printing industry. This is because designers and engineers have to go through multiple steps in order to design the part, output it to a digital file, prepare it for printing, and actually print it. These few stages will be covered in the following paragraphs. Available commercial and free software examples will be discussed in section 2.

In order to design and draw parts, computer aided design (CAD) software is widely used. CAD software allows designers to efficiently translate or develop a design in a digital context. These programs usually offer a set of tools and aids to draw on planes, in either two or three dimensions depending on the requirements of the design. Some even allow to set parametric relationships between parts of the sketch, and even join different existing parts together in an assembly. It is very common for these programs to allow exporting the designed parts into a wide range of file formats, with which they can be shared, printed, machined, etc.

Once the part has been designed, drawn, and exported to a suitable file type, the next step is to prepare the part for manufacturing. On some cases, the designers already take the necessary considerations beforehand, but this might not always be the case in situations where for example a business orders fabrication of one or multiple in-house made parts for prototyping purposes. Depending on the technology that will be used to actually make the part, be it machining, laser-cutting, 3d printing, or whatever other CAM approach is chosen; the appropriate considerations will have to be taken in order to prepare the part for optimal fabrication, obeying machinery limitations. For basic parts that could be 3D printed, one would have to ensure the part is completely defined, align it accordingly to get best results depending on the printer used, and finally go through a process which is commonly referred to as slicing.

Part slicing, in general terms, means generating correlative cross sections of a chosen part. These can be then outputted to files as images, movement vectors, or what is more common on CAM based systems, G-Code files. G-Code files are plain text files that contain instructions based on the G-code instruction set. This is the file that will finally be fed into the printer / laser cutter / CNC mill / etc. for actual printing. The printer will execute the file line by line. A comprehensive set of G-code instructions can be found on Appendix A at the end of this document. Before slicing, the user must specify several options such as print speed, layer height, and, depending on the actual machine used and the part’s characteristics, many other adjustments might be needed for optimal results.
One would think now is the time when the user presses the magic button and the printer starts working. And it does - sometimes. But there is still software in the printer’s microcontroller or CPU that’s taking care of reading the file, executing each instruction, interfacing with other IC’s, etc. This is usually referred to as the firmware of any machine/device. The firmware is not only in charge of reading and executing instructions from files, but will also have to handle all other input/output controls available on the machine itself for manual operation. It must interface with motor drivers, linear encoders, LED display drivers, button panels, and many other components that might be present. It must also keep the machine safe, with whatever sensors or mechanisms are available, in order to prevent accidents.

1.3 What is Lightcam?

The name Lightcam refers to a 3D stereolithographic printer built by Abel Elbaile and Aleix Tejada between April 2013 and October 2014. Mechanics, electronics, software and firmware were all designed, coded and built in-house. Two prototypes were successfully developed.

1.3.1 Why build yet another 3D printer?

After my interest in rapid prototyping, computer aided manufacturing, and especially 3D printing rose substantially ending the year 2011, I decided to build my own 3D printer, which I successfully completed around the end of April 2012. It was based on the RepRap Mendel 3D printer, with some slight changes for easier calibration.

After playing around with it for a few months, it became clear that it eventually had some good things, but also some bad ones. For the purpose of this text, I’ll focus on the bad ones.

It was somewhat hard to reach an acceptable print quality, and detail was not enough for certain parts, especially small and fine detailed ones. Parts that composed an assembly had to be designed with a 0.2 mm tolerance in joints, which is not acceptable for certain applications. Bridges and other flying structures had to be taken care of (if possible) in the design phase, choosing best base for printing and trying to minimize the need of support material, which produces non-optimal results in single-material printers. Layer resolution could reach up to 0.1 mm somewhat reliably, but still flat surfaces on vertical faces weren’t accomplishable.

In terms of the machine itself, it had pretty obvious and serious structural weaknesses which generated vibration when printing thus affecting part quality. It was also difficult and annoying to calibrate and adjust it properly. And it had to be checked on a regular basis, every few months. When printing, situations like waiting for the heated bed and nozzle to heat up while paying attention to leaking material, and extruding some material before printing to ensure the nozzle was full were time-consuming and non-optimal.

For the mentioned reasons and many others, I started to investigate on photopolymerization techniques. I considered two main approaches: Light projection using a DLP projector, and laser scanning system to drive a laser. In both approaches a source of UV light is used to catalyze a photosensitive resin which hardens when exposed to that wavelength. I tried the DLP technique
and it did work, but who wants to open a 3D printer and find a cheap BenQ projector inside it? I didn’t. So I went on and started investigating further on the SLA approach.

1.4 Lightcam development overview

Disclaimer:
The following pages try to narrate the story of the development of this project. They describe some of the most relevant moments and aspects of the development of the Lightcam 3D printer, covering the interval: April 2013 to October 2014. They do not cover absolutely every thought process and probably some decisions which were important at the time have been omitted because I forgot to mention them. They also don’t expand on future development nor try to describe the finest detail any of the aspects covered.

—

We started by buying a cheap galvo set from eBay which came with no datasheet or user manual of any type. I had in my hands two galvanometers, which I didn’t even really know what they were by the time, with two analog driver circuits to drive them and had absolutely no idea how we would control them. We tested sending binary digital signals. We then understood we could drive the galvos to right-most and center positions by sending them logic 1’s and 0’s respectively on a 0-5V scale, which made us realize we were missing the lower half. I sometime after realized I had to send an analog continuous value between -5V and +5V, but at that point that was just a hypothesis that had to be tested.

This led us to investigate further on the topic, and after many reading we came up with the idea of using a DAC to convert a digital value obtained from a hypothetical 2D point on a cartesian plane into a continuous value between the two output limits of the DAC. We found a high-speed rail-to-rail DAC that I could operate through SPI and output values between -5V and 5V, so we ordered some samples.

At the time, wiring this chip wasn’t trivial at all. I knew close to nothing about analog electronics, I had probably never worked with any device that required negative voltage supply and had no clue on how to actually get a negative rail on my circuit. I tried several tricks I found but as soon as I put a load in, the system became unstable and I even burnt a few DACs during the process. I learned about voltage regulators, power supplies, and had to review all the basic electronic concepts I had learned on previous years, but this time for real.

Finally getting the DAC working and being able to see +5, 0, and -5V output loops with 100ms delays on the oscilloscope was one of the most exciting moments in the whole build. Getting two of them was a bit more tricky. Somehow, supposedly during the reflow process, some came out not working at all. Anyways, we finally got two working breakout boards, each with a DAC that would control the X and Y galvo respectively.

It was time for a bit of software. I chose the Arduino platform to prototype it for several reasons, but mainly because its super fast to get something up and running with a serial port output. At that moment, I coded a simple interpreter that would read G code instructions sent through the serial port and command the DACs to the corresponding equivalent values based on the
coordinates received. As soon as I got it working, just with integers, I hooked up a cheap laser module bought off from eBay and started manually commanding movements and watching astonished how the beam on the wall moved instantly from point to point. Of course it didn’t work perfectly the first time, but it eventually worked after a few adjustments. At this point I faced a problem of noise on the galvos, which was implicitly causing noise in the laser movement. I solved it by setting a common ground for both the Arduino and the galvo drivers. Again, these are things that may sound trivial, but at the time, these were all unanswered questions that were not particularly a search and find Google job. They required some further thinking and understanding of what was going on.

I soon added real number support and the time dimension to point-to-point movements, thus starting to move over time traveling through the segment obtained by simple linear interpolation between points. Once this was ready, it became clear I had to be able to execute a whole set of instructions, not just one at a time. This led me to the next part, which was interfacing with an SD card.

Why and SD card and not feeding a set of instructions through the serial port? Basically to give the printer autonomy in terms of being able to operate without hooking a PC during the whole print, as prints would eventually take some time. An SD card is fast and easy to work with in terms of loading files directly from a computer. Another option was integrating a flash memory of some kind, but this would require coding the file uploader through serial port (both in the PC and uC side) and general memory management controller (in the uC), which wasn’t a priority at the moment.

In this case I did use an existing library called SdFat, but had to adapt it for the ARM SAM 3X SoC family (instead of the traditional ATmega 328 regular Arduinos use). Once that was ready, I first tried an approach of pre-loading a set of instructions from the file and then executing them while reading more to always keep a buffer with a minimum number of instructions. Some time after I realized that optimizing the X-Y movement part of the code allowed me to read instructions on the fly, and execute them one after another, which resulted in a much simpler code. Finally reading from the SD card and executing these movements one after another resulted in the first time this experiment was resembling what it would be in the future.

My colleague Aleix built a simple test bed and mounted the galvos on it for further testing. We used this to calibrate the galvo drivers and correct bugs on the interpreter. We also printed some parts with the RepRap to build a simple linear stepper movement test bed to start programming the Z axis. From an electronic point of view, I started by using a Pololu stepper motor driver breakout board, with which I had experience with and is pretty easy to work with. The stepper motor was a regular NEMA17 we had laying around. At this point, the machine looked something like this:
In the picture, in the Z-axis motor module, there are several things to note. The first is the small mechanical endstop hooked on the right rail. This is of course for homing procedures and to stop the machine if anything went wrong and it was hit during a print. The second and most important thing to note is the linear encoder module hooked to the test bed between the left rail and the leadscrew. This encoder was bought from a Chinese distributor but wasn’t optimal. It did work perfectly and had the resolution we wanted, but the housing was very limiting in terms of size and shape. We would later on find another chip that was much easier to integrate in our final design and would be the definitive one, but we used this one for testing purposes at the beginning.

At this point, the following subsystems were present on the firmware:

- SD card management
- File opening, reading and closing
- Instruction interpretation and execution
- X-Y galvo control through DAC output
- Laser TTL control
- Z-Axis motor control + Linear encoder feedback

While my colleague Aleix worked on the physical design of the actual machine, I started optimizing code. I got rid of all the calls to the Arduino API and used direct port access whenever possible. I reduced overhead caused by SD library calls and further explored the hardware quadrature decoder logic included in the SAM 3X chip in order to manage encoder interrupts directly at a hardware level. Here I was still building on the first version, which would later on be completely rewritten for even further efficiency and code cleanness.
The machine would soon be ready, and the electronics seemed to work as far as I had tested. It was time for translating what was on the breadboard to a well done schematic, choosing the appropriate components for board routing and production. I had done things with eagle before, but this was the first time I was working on a “serious” (at the moment) two-sided board that had to obey several dimensional requirements in order to fit the Arduino and the multiple removable components that had to be stacked on top. I messed up quite a bit on the first revision with top and bottom connections. After adding some vias to fix the mess, it could be properly populated and worked fine. The board was 100% manufactured at our lab. This is the result of the first 1.0 final revision.

As mentioned before, I then completely rewrote the firmware. Kept the kernel with the optimizations but structured it in a much more convenient way considering the platform architecture. All subsystems were better integrated one with another. It worked much much better than the first version. The execution sped up considerably. The machine could finally be put to test.
Testing brought out new bug fixes and several additions for better printing. But, the fact was that as we kept testing, we kept bumping on to the same few problems:

- On layer change, the part was stuck to the resin tank floor. This caused several problems from a structural point of view, as the machine was flexing when peeling off the part, and also in terms of part quality. Once the glass even broke spilling all the resin.

We switched to plastic tanks and started investigating on materials or treatments we could apply to the tank floor in order to reduce stickiness, but also be resistant and optically transparent. It was difficult. My colleague Aleix finally found a very interesting plastic, but it’s very uncommon and thus very hard to find (unless you buy large quantities). We could finally get ourselves with some and tested it. It was perfect in terms of not sticking to the solid resin, but nevertheless, it was not comfortable to stick it on to the tank floor and it also wore out easily after several prints. It wasn't the solution we were looking for.

- It was hard to peel the part, once finished, from the polyamide build platform. Parts have to be built with supports, and at this time I didn’t have them implemented in our software, but still, it stuck quite a lot and was way too hard to get out completely. It wasn’t a good experience for a hypothetical final user of the product, even if it is just support material that doesn’t harm the part if it gets broken.

- Laser diodes kept burning or suddenly not working fine. At this point, we were using an eBay laser module, with included laser driver. The driver was covered with epoxy so I’m not really sure what was in it. Anyways, the laser is one of, if not the, key component in this machine. We needed more control over what was going on there.

- Regarding optics, parts came out with a goo-like half-catalyzed resin around them. This was because of two reasons. First of all, the lens used to focus the laser beam was build out of plastic, which produced a bad quality beam resulting in bad focusing and lots of light dispersion (thus loss of light output reaching the build area). Second, when we moved on to plastic tanks and added the layer of repellant plastic, some light was being both diffracted and diffused in that layer. This, of course, was affecting part quality.
One of the goals of this project was to build a machine that you didn’t have to worry about. A machine that just worked, giving little, if any, headaches to the user. After lots of testing and fixes, it became clear to us that this configuration was definitely not the one. So we rethought the machine from an upside down point of view, and little by little came up with a new design. It was our desire to make use of as many parts as possible from the existing machine, so the outcome may be non-optimal in terms of dimensions (could be smaller).

On this new design, the laser beam is projected directly into the resin tank from the top, thus hitting the resin directly with no glass or plastic interference. Optics have been upgraded to high quality aspheric glass lenses, which allow for better focus and light transmission. As the parts are now built on a platform that sinks inside the tank, the problem of layers sticking to the tank floor is also gone, thus ensuring prints came out as expected and reducing chances of parts unsticking or falling from the build platform. The laser driver and mount are now being handled in-house. The laser and lens mount is custom built and the laser driver has now been designed and built at our lab, which leads me to the following topic.

The laser driver has probably been one of the least trivial parts to solve after figuring out how to move the galvos. These 405nm diodes have proven to be very very delicate. They have to be handled with extreme care as any electromagnetic field from any charged object around (even ourselves) can permanently damage it. The theory of operation is pretty straightforward. Summarizing, the diode wants a constant stable current supply. Considering once again our low knowledge of electronics, as I worked on the new board Aleix had to focus on the topic and learn about how current sources worked and how to regulate the current going through the diode. The final design actually came out pretty simple, with a mosfet and an operational
amplifier to actually regulate the current, along with some voltage regulators for power control. But again, these are things we had never heard of and had to learn, try and test until they worked. Now, the new driver supports both continuous and pulsed operation, and of course, has TTL control support. No diode has died yet.

Unfortunately (or maybe not), the mainboard had to be redone to include all the new components. This was just a matter of time really. The hard part was done, here it was just a matter of hours getting the new board routed and concentrating to avoid messing up, as no one wants to finish such a board and have to redo it. This new board supported two stepper motors, one for the build platform and another for the VAT, each with its own encoder, two laser interface ports (one optocoupled and one direct), fixed linear voltage regulators for each subsystem, auxiliary power for motors switchable through a jumper switch, and other ports for sensors we’re working on at the moment, one of them being resin level sensor consisting of a laser beam pointing at the resin and bouncing off to hit a linear photodiode array. If both the laser source and the photodiode array are in fixed known positions, we are able to determine through simple trigonometry the resin level and thus correct accordingly when necessary.

The firmware was adapted for the new machine. Most of the work from the previous machine could be adapted and used perfectly with this new version. Some small things had to be taken care of, and some additions had to be made. For example, SAM 3X’s quadrature decoder logic could only be used for one of the encoders. So I decided to use the hardware decoding on the build platform one, which was the one that would be used more often, and used a software implementation with interrupts for the VAT one. Again, to speed up and avoid the Arduino API as much as possible, these interrupts were handled directly at register-level with the Atmel
specifications. VAT resin-level correction has also been implemented and in general terms the machine is working nicely at the moment.

In terms of future development, at a software level I’m right now working on finishing the support generation and mesh slicing. From an electronic point of view, we’d like to upgrade the movement system from discrete linear interpolation to a continuous output electronic integrator circuit, as well as getting rid of the Arduino and wiring up a similar chip from the same family which I plan on using for the final design. Finishing the power supply subsystem soon and work on laser state feedback using the integrated photodiode are also things we consider. And a bit more long-term, designing our own galvo drivers would be nice.
2. End-User Software: Project definition

2.1 Scope

The first part of this work aims to describe the design of a cross-platform native application for Windows, Linux and Mac OS; which will serve as an interface for end users to be able to work with the Lightcam 3D printer.

2.2 Objectives

The main objective of this project is

To design and develop a simple yet complete, cross-platform, user friendly software solution that provides end users with a tool with which they can easily interface with the Lightcam 3D printer.

2.3 Requirements

2.3.1 Functional requirements

The user must be able to:

- Open STL and OBJ files.
- See a live 3D preview of the build platform.
- Move the camera freely by panning and zooming.
- Select one or multiple parts.
- Transform parts individually: Move, rotate resize and align.
- Automatically generate support material for needed surfaces.
- Customize job settings, mainly resin type and part quality related settings.
- Save jobs.
- Load saved jobs.
- Manage connected printers.
- Monitor connected printers’ status.
- Send a job to a printer.
### 2.3.2 Non-functional requirements

- The application must run on computers with Windows, Linux, or Mac OS*
- The application’s user interface must be coherent with all platforms.
- System dialogs should be used to preserve coherence among platforms.
- The 3D engine used must support hardware acceleration.
- The application must be usable.
- The application should load quickly.
- The application should show progress bars to inform the user in heavy tasks such as slicing or support generation.
- Network connection must not be required to print a job.
- OS native support for serial port must be available to connect to the printer.

### 2.4 Risks

As many other software projects, due to their nature, this project may face some of the following risks, if any:

- Failing to recruit a competitive team of developers.
- Failing to plan the development in a coherent and chronological way.
- Schedule flaws or severe delays when failing to complete tasks on time.
- Increase or addition of important requirements that weren’t previously considered.
- Architectural or design changes that alter other people’s work.
2.5 Use cases

- **Print jobs**
  - Open print job file from disk
  - Save print job to disk
  - Change print job settings
  - Send job to printer (Print)

- **Parts**
  - Import part
  - Select part
  - Duplicate part
  - Translate part
  - Rotate part
  - Scale part
  - Align part face to surface

- **Support material**
  - Generate support material
  - Remove support material
  - Change support material settings

- **View**
  - Rotate camera
  - Change zoom
  - Change view type
2.5.1 Open print job from disk

Precondition: -
Actor: User
Trigger: User wants to open a previously saved print job.
Description:
1. User selects “Open” on the “File” menu.
2. A file chooser system dialog appears.
3. The user selects a file and clicks “Open”.
4. The part is added to the job, placed in a valid position inside the build platform.

Extensions:
3.a No part is chosen. “Cancel” is clicked
   3.a.1 No part is added to current job.
   3.a.2 The use case ends.
4.a The new part doesn’t fit on the build platform
   4.a.1 The part is placed outside the build platform and highlighted.
   4.a.2 The user is warned about the issue.

2.5.2 Save print job to disk

Precondition: -
Actor: User
Trigger: User wants to save the current job.
Description:
1. User selects “Save” on the “File” menu.
2. A file chooser system dialog appears.
3. The user navigates to the desired directory and enters a valid filename.
4. The job is saved into the desired file.

Extensions:
4.a Disk space full
   4.a.1 The user is notified and the dialog closes.
   4.a.2 The use case ends.

2.5.3 Change print job settings

Precondition: -
Actor: User
Trigger: User wants to change current job settings, such as layer height or resin type.
Description:
1. User displays “Job Settings” tab on side menu.
2. User selects desired resin from the Resin type combo box and/or sets the layer height through the Layer height slider control.
3. The user clicks on the Apply Job Parameters button.
4. The system updates the job settings for current job.

Extensions:
3.a The user wants to revert changes to previous settings
3.a.1 The user clicks on the Apply Job Parameters button.
3.a.2 The system resets values for both controls to reflect current settings.

2.5.4 Send job to printer

Precondition: There is a printer connected and all (at least one) part(s) have generated supports and are within boundaries.
Actor: User
Trigger: User wants to print his current job.
Description:
1. User clicks on the Send To Printer button on the side menu.
2. The system sends the job to the selected printer.
3. The use case ends. The user must continue in the actual printer.
Extensions:
2.a There is an error sending the job to the printer.
   2.a.1 The user is notified about the error.
   2.a.2 The use case ends.

2.5.5 Import part

Precondition:
Actor: User
Trigger: User wants to import an existing part file into his current job.
Description:
1. User selects “Import” on the “File” menu.
2. A file chooser system dialog appears.
3. The user selects an STL file and clicks “Open”.
4. The part is added to the job, placed in a valid position inside the build platform.
Extensions:
3.a No part is chosen. “Cancel” is clicked
   3.a.1 No part is added to current job.
   3.a.2 The use case ends.
4.a The new part doesn’t fit on the build platform
   4.a.1 The part is placed outside the build platform and highlighted.
   4.a.2 The user is warned about the issue.

2.5.6 Select part

Precondition: There is at least one part present in the build platform.
Actor: User
Trigger: User wants to select a part in the current build platform
Description:
1. User clicks on the part.
2. The part is now selected.
2.5.7 Duplicate part

Precondition: A part is selected.
Actor: User
Trigger: User wants to duplicate an existing part.
Description:
1. User selects “Duplicate” on the “Edit” menu.
2. The system creates another copy of the selected part, together with its settings, and places it in a valid position in the build platform.

Extensions:
2.a The duplicated part doesn’t fit on the build platform
   4.a.1 The part is placed outside the build platform and highlighted.
   4.a.2 The user is warned about the issue.

2.5.8 Move part

Precondition: A part is selected.
Actor: User
Trigger: User wants to manually change a part’s position for printing.
Description:
1. User displays “Transform” tab on side menu.
2. User manually changes X, Y, or Z component of current part’s position.
3. The system places the part accordingly.

Extensions:
3.a The moved part is touching another part.
   3.a.1 The part moved is shaded in red color to inform the user.
   3.a.2 The use case ends.

2.5.9 Rotate part

Precondition: A part is selected.
Actor: User
Trigger: User wants to manually change a part’s orientation for printing.
Description:
1. User displays “Transform” tab on side menu.
2. User manually changes X, Y, or Z component of current part’s rotation.
3. The system rotates the part accordingly. If any supports had been generated for this part they are automatically removed.

Extensions:
3.a The rotated part has vertices under in negative space coordinates.
   3.a.1 The system automatically adjusts the Z position component to place the part correctly on top of the build platform.
   3.a.2 The use case ends.
3.b The rotated part is touching another part.
   3.b.1 The part moved is shaded in red color to inform the user.
   3.b.2 The use case ends.
### 2.5.10 Scale part

**Precondition:** A part is selected.

**Actor:** User

**Trigger:** User wants to manually change a part’s size for printing.

**Description:**

1. User displays “Transform” tab on side menu.
2. User manually enters a percentage value for scaling X, Y, or Z component of currently selected part.
3. The system scales the part accordingly. If any supports had been generated for this part they are automatically removed.

**Extensions:**

3.a The scaled part has vertices under in negative space coordinates.
   3.a.1 The system automatically adjusts the Z position component to place the part correctly on top of the build platform.
   3.a.2 The use case ends.
3.b The scaled part is touching another part.
   3.b.1 The part moved is shaded in red color to inform the user.
   3.b.2 The use case ends.

### 2.5.11 Align part face to surface

**Precondition:** A part is selected.

**Actor:** User

**Trigger:** User wants to align a specific face of a part to the built platform surface.

**Description:**

1. User displays “Transform” tab on side menu.
2. User clicks the Select face button.
3. All parts are shaded except the selected part. The user is asked to click on desired facet to align.
4. The user clicks on the desired facet to align.
5. The system aligns the chosen facet with the surface of the build platform.

**Extensions:**

3.a The aligned part has vertices under in negative space coordinates.
   3.a.1 The system automatically adjusts the Z position component to place the part correctly on top of the build platform.
   3.a.2 The use case ends.
3.b The aligned part is touching another part.
   3.b.1 The aligned part is shaded in red color to inform the user.
   3.b.2 The use case ends.
2.5.12 Generate support material

Precondition:
Actor: User
Trigger: User wants generate support material for one or several parts.
Description:
1. User displays “Supports” tab on side menu.
2. User selects the part(s) for which to generate supports on the part listbox.
3. User adjusts support density, base height, and end size with the provided slider controls.
4. The user clicks on the Generate button.
5. The system computes supports and adds them their respective parts.

Extensions:
3.a The user wants to choose default settings.
   3.a.1 The user clicks on the Default button, right under the sliders.
   3.a.2 The system updates the sliders’ value with default settings.
   3.a.2 The use case ends.
5.a The user wants to cancel the process.
   5.b.1 The user clicks on cancel in the support generation progress dialog.
   5.b.2 The support generation sub-process is stopped and no supports are added.
   5.b.3 The use case ends.

2.5.13 Remove support material

Precondition:
Actor: User
Trigger: User wants remove the existing support material for one or several parts.
Description:
1. User displays “Supports” tab on side menu.
2. User selects the part(s) for which to remove supports on the part listbox.
3. User right clicks on the selected items and clicks on “Remove support material” from the displayed context menu.
4. The system removes supports from the selected part(s).

2.5.14 Change support material settings

Precondition:
Actor: User
Trigger: User wants change support material settings for one or several parts.
Description:
1. User displays “Supports” tab on side menu.
2. User selects the part(s) for which to change supports on the part listbox.
3. User adjusts support density, base height, and end size with the provided slider controls.
4. The user clicks on the Generate button.
5. The system computes the new supports and replaces the existing with the new ones on the selected parts.

Extensions:

3.a The user wants to choose default settings.
   3.a.1 The user clicks on the Default button, right under the sliders.
   3.a.2 The system updates the sliders’ value with default settings.
   3.a.3 The use case ends.

5.a The user wants to cancel the process.
   5.b.1 The user clicks on cancel in the support generation progress dialog.
   5.b.2 The support generation sub-process is stopped and no supports are added.
   5.b.3 The use case ends.

2.5.15 Rotate camera

Precondition:
Actor: User
Trigger: User wants rotate the camera.
Description:
1. User holds down “Alt” key.
2. User drags and drops the cursor in natural rotation direction.
3. The system updates the view accordingly.

2.5.16 Change zoom

Precondition:
Actor: User
Trigger: User wants to zoom the camera in or out.
Description:
1. User scrolls up or down to zoom in or out respectively.
2. The system updates the view accordingly.

2.5.17 Change view type

Precondition:
Actor: User
Trigger: User wants to change the view type.
Description:
1. User displays “View” menu.
2. User chooses desired view type from the options present.
3. The system updates the view to the new view type.
3. End-User Software: Architecture

3.1 Conceptual schema
3.2 Class description

3.2.1 MainWindow

MainWindow inherits from QMainWindow, which provides a main application window in which the user interface will be built on. It holds all system menus together with their corresponding actions. It also holds the main graphics view where the whole 3D scene will be rendered.

MainWindow is the first object created when the software is launched. On creation, it will create and initialise all UI components at runtime - fonts, icons, assets, and UI controls -, place them accordingly, and display itself once ready. A new empty build is always created by default when opening the software.

Furthermore, MainWindow is also responsible of notifying the user when a printer has been connected or disconnected. The PrinterManager class will emit a signal for each case, to which MainWindow has slots connected. A custom alert will be displayed in each case.
3.2.2 GraphicsView

GraphicsView inherits from QGraphicsView, which provides a widget for displaying the contents of a QGraphicsScene. This will be used as the main widget in MainWindow. Everything will be rendered inside the GraphicsView, except system menus. All mouse and keyboard events will be automatically propagated and handled by the QGraphicsScene set as viewport.

4.2.3 Scene

Scene inherits from QGraphicsScene, which provides a surface for managing a large number of 2D graphical items. QGraphicsScene has no appearance of its own, but as we’re rendering 3D content, we use the drawBackground method of Scene to render all OpenGL content.

Scene holds references to the current build platform, with all the parts present together with their applied transformations; as well as the side menu which allows the user to perform several relevant actions.

Scene is also in charge of taking action when a mouse or keyboard event is produced. All camera transformations are handled by Scene, which determines the action desired by the user and updates the render accordingly. In case the user uses a click to select a part, the click event is routed to the Platform object, where, if the click has hit a part, the part is eventually selected.
3.2.4 SideMenu

SideMenu inherits from QDialog and is in charge of creating and holding the layout and functionality for the side menu available to the user. Only one instance is created on each program execution. When the SideMenu is first instantiated, all the UI controls are created and placed in their corresponding layouts. Functionality for each control is also initialised and proper connections are made with local slots that will later emit corresponding signals for view updates on Scene.

The side menu is built in an accordion-type menu, composed of three tabs: Transform, Supports, and Job Settings.

The transform tab allows users to apply translation, rotation, scaling or aligning operations on selected parts. It will also display current part's position and rotation in space when selected.

The supports tab is used to generate supports for selected parts. It includes controls that allow the user to specify essential settings for support generation — density, base height, and end size —, as well as the actual button for generating supports. There is also a custom list control where all parts are listed together with their support status, which indicates if support material has been generated or not on each part.

The job settings tab allows the user to choose specific job settings for further printing or saving. In terms of print setup, the user can choose the resin type that will be used and set the desired layer height. The user can also choose to which machine(s) the job will be sent to, as well as configuring notifications.

3.2.5 PartListView

PartListView is a custom widget that acts as a regular list view item but allows for two column layout. This widget is only used on the supports tab of the side menu, and will display part names on the left, together with their support status on the right. Both columns can be rendered with specific font colours and sizes. Click events have also been implemented in order to handle right-click event for support deletion.
3.2.6 Platform

Platform represents the actual build platform where all parts are placed and further printed. All the part management is done in this class. Platform holds all parts in the current build, and is in charge of managing the addition or deletion of parts, aligning them correctly thus preventing them from touching each other, etc. It keeps track of what is the selected part, if any. It routes any transformations or support generation operations to the selected parts, offers volume estimation calculations, and finally renders the actual build platform and the parts on it, according to the view mode that has been set by the user (or by default).

3.2.7 PlatformData

PlatformData is a helper class which holds a few basic definitions, settings, enumerations, and helper functions that are used by several classes along the program. It offers basic "global" like access to these values and settings which allow for coherence among several classes that operate with them and are not directly linked to Platform.

3.2.8 Part

Part class holds all the information regarding a part. A part is the result of importing a digital 3D part file (usually STL or OBJ) and placed in the build platform. Part class takes care of all the stages, from reading and loading the actual file, repairing the part if it happens to have errors; all the way to rendering the part in the build platform, holding all the transformations applied to it. Each Part also holds its own position data. Part volume can be calculated, as well as other necessary operations such as computing bounding boxes, radius, part size, etc.

The STL file reading is done with the help of ADMesh, a small open source utility program written in plain C used for processing triangulated solid meshes. ADMesh is also used for part
analysis and repairing in Lightcam software. When loading a file, the actual resulting mesh is kept in a stl_file and is used through TriangleMesh. The latter is a helper class that has been borrowed from Slic3r, another open source software that will be discussed in further detail in section 4.3.4, which uses the stl_file data structure from ADMesh, and offers several interfaces for the multiple operations available.

### 3.2.9 stl_file

An stl_file keeps all the vertices, edges and facets of the part, along with some information calculated upon these values which is hold in another data structure called stl_stats. A file descriptor is also kept to the file from which the part was loaded.

### 3.2.10 stl_stats

The stl_stats structure keeps data of an stl_file that has been loaded into memory. Some relevant pieces of information include: number of facets, maximum and minimum points in the mesh, mesh size in all three axes, volume, collisions detected within a part, memory space allocated, and a few attributes needed for mesh repairing.
3.2.11 Supports

The Supports class stores generated (or not) supports for a Part. The actual support structure is generated and stored in a TriangleMesh, just as the part itself. Facets needing support are calculated by checking all facet normals of the original part, and stored in a vector structure. Additional information, such as basic support settings — density, base height and end size — is also stored.

3.2.12 Build

The Build class represents, in a sense, a Lightcam project. It contains the current Platform, with all the parts and their corresponding transformations; and a filename that refers to the saved disk file, in case the build was saved. This object, together with the platform (that contains all parts and their corresponding additional information and supports) is serialized in order to save and load projects from a file.

The Build class contains methods to save and load project files. When saving, parts can be embedded into the build file or copied into the build file directory and linked in the project file.

3.2.13 Job

A Job object represents a project that has been sent, or is meant to be sent, to a printer for it to be printed. It contains a Build object with all the subsequent data that can be extracted from it (see previous section). Furthermore, it contains generic job settings such as layer height, resin type, or timing statistics for further reference. Of course, it also contains references to the printer or printers that have been assigned with this job.
3.2.14 ResinType

ResinType aims to represent, as its name clearly states, a resin type. According to the resin chosen for a Job, some specific laser settings such as X-Y speed and/or beam intensity will be used when printing. It is important to choose the resin type that fits best with the resin that is being used to achieve best quality results. New resin types can be added manually or imported from a file. They can also be exported to a file for sharing.

3.2.15 Printer

The Printer class represents a Lightcam printer in the system. The user can have several printers saved for future use. If there is more than one Lightcam printer connected to the computer, the user will be asked to choose to which printer (or printers) the Job should be sent. These don’t have to be necessarily connected to the computer at all times, but they obviously must when sending a job to be printed. Printers can have a name assigned to them. Furthermore, the Printer class will keep track of each printer’s status.

Finally, all the interfacing with the communication layer is done in this class. Sending a job, retrieving printer status, or sending manual commands are some of the things this class will allow the user to do.

3.2.16 PrinterManager

PrinterManager acts as a unifying point for all supported connection interfaces. For now, only USB is supported, but in the future other interfaces might be available, such as network or Bluetooth. PrinterManager will initialise all “daemons” and will be notified by them whenever a new Lightcam printer is connected or disconnected. In the first case, PrinterManager is in charge of creating the corresponding Printer object and storing it in the available printer list, as well as notifying MainWindow which will then display a notification to the user. If a connected printer is disconnected, PrinterManager will update printer status to disconnected and notify MainWindow.
3.2.17 USBWatcher

USBWatcher is an interface that handles all USB printer connections through libusb. When initialised, the hotplug feature of libusb is started and a callback function is registered to handle events sent by libusb. Any event received is handled internally and, if necessary, a signal is emitted accordingly.

3.3 Third-party software used

In terms of how to develop the software and with what tools, a few considerations were studied based on some of the non-functional requirements defined earlier. Some of them may be:

- The software must be multi-platform (Windows, Linux, and Mac OS).
- The 3D engine used must support hardware acceleration.
- Parts of the software such as mesh slicing and generation of supports may be computationally intensive.
- Network connection is not required to run the software and print a job.
- OS functions such as serial port must be accessible.

Web-based technologies were the first to be discarded, because although they're fast for UI prototyping and Chrome's V8 engine is a beast, the expected behaviour and performance couldn't be guaranteed. Plus it would eventually have to be somehow enclosed in a native app for every supported OS in order to have access to OS functions such as access to the serial port. So I evaluated my possibilities in terms of native app development and multi-platform frameworks. The most complete development framework I found was Qt, for C++. Qt offered me the possibility of developing a multi-platform app by using its API, which at the time of this writing is in its 5th version and covers nearly all the basic and important aspects I need to complete this application. It just lacks, in my opinion, a good implementation of the serial port interface, at least a complete one, so I've chosen a well known C library called libusb, which is giving me great results.

3.3.1 Qt Framework

Qt is a cross-platform, open-source, C++ application development framework. Qt has been chosen, among other reasons, because of its:

- Native C++ nature.
- Great GUI controls and rendering engine.
- Broad API across all required platforms.
- Acceptable OpenGL support.
• *Signals and slots* event handling mechanism.

### 3.3.1 Signals and slots

Signals and slots is the way in which Qt solves communication between objects. And it is quite a powerful mechanism indeed, which kind of mimics the behaviour of the Observer pattern in software development. Slots are essentially functions defined in the “observer” objects. These are connected to signals which are declared and are meant to be emitted by the “observed” objects. These signals can contain event information, which can be received and used in it’s slot or slots, as a single signal can be connected to many different slots.

Currently, in the Lightcam software signals and slots are mainly being used to communicate the GUI events with the backend. But they are also being used on, for example, USB hotplug detection and notification, which is an event that is not directly caused by a user action on the GUI.

### 3.3.2 libusb

libusb is a well-known and widely-used C library that gives applications easy access to USB devices on many different operating systems. In this case, as the application is being developed in native C++, there is no need for using any language binding but the library itself. It offers compatibility across all required platforms, so it suits all functional and non functional requirements discussed in earlier sections.

Furthermore, since version 1.0.16, libusb has added support for hotplug event on some platforms. This is a very interesting feature as it allows the user, in this case the application developer, to request notifications for the connection and disconnection of specific USB devices. In our case, *USBWatcher* class is in charge of handling hotplug event initialization and callbacks for connection and disconnection of USB devices that specifically match Lightcam vendor id and product id.

### 3.3.3 adMesh

ADMesh is a small open source utility program written in plain C used for processing triangulated solid meshes. ADMesh is used for STL reading, writing, analysis, and repairing in Lightcam software. When loading a file, it’s read by the ADMesh interface and the actual resulting mesh is kept in a *stl_file*, which is an ADMesh provided data structure. When performing operations on meshes, the ADMesh implementation for basic operations —such as simple rotations or translations— is being used to maintain coherency.

ADMesh was chosen because of it’s minimalistic nature. It is intended to be a simple command-line program with no windowing capabilities. In that sense, the code is clean, simple, and straight to the point, which makes it extremely easy to port it or include it in any custom software design accurately. In terms of multi-platform support, it can be compiled on any system with a C
compiler and the standard C libraries, so it shouldn't be a problem to port it to Windows systems.

### 3.3.4 Slic3r

Slic3r is a G-code generator for 3D printers. As I've been using it since early versions and have seen it grow to what it is now (currently on version 1.1.7-stable), and due to its open source nature, I decided to include some of it's classes in case I could use them. The slicing module would be a very interesting module to hook to the Lightcam software, as really, the slicing and path generation feature in Slic3r is very matured and produces great results if properly calibrated and configured.

For now, only the `TriangleMesh` class is being used, which acts as an interface for ADMesh's `stl_file`. I would like to use more in the future, but it will all depend whether Slic3r can be trimmed down enough for our own purposes. Slic3r features complex movement algorithms which calculate carriage speeds and motor acceleration and deceleration based on, among other factors, inertial calculations to prevent the motor from losing steps when accelerating or decelerating. These calculations don't apply to our hardware setup, as it is radically different in every aspect. This is probably the most obvious feature that would have to be trimmed down, but there are many others.

The objective would be to keep Slic3r's slicing and path generation engines, but eliminate all speed calculations, as they suppose a considerable overhead in computation time and our speed calculation algorithms are much simpler, as they don't have to deal with regular FDM printer considerations.
4. Printer Firmware: Project definition

4.1 Scope

The second part of this project is focused on designing and developing a firmware for the Lightcam 3D printer. The software installed in the printer will run on an ARM AVR SAM3X SoC. The study of why this SoC was chosen is out of the scope of this document. The firmware will run on the microcontroller and must be able to manage all hardware components and interface with the application running on the PC side.

4.2 Objectives

The main objective of the second part of the project is

To design and develop a fast and lightweight firmware that runs on ARM AVR SAM3X SoC and can execute printer commands that have been either manually inputted or sent as part of a file; as well as handle all the interfacing and control of peripherals.

4.3 Requirements

4.3.1 Functional requirements

The firmware must allow for:

- Two-way USB serial communication with a host device.
- G-Code instruction parsing.
- G-Code instruction execution.
- Stepper motor control.
- Quadrature wave decoding for encoder position reading.
- Dynamic memory management.
- SD card interfacing.
- File reading from SD card.
- File writing to SD card.
- Two-way SPI communication for at least 3 peripherals.
- Laser on/off control
4.3.2 Non-functional requirements

- The firmware must run an ARM 3X SoC.
- Maximum CPU clock is 84MHz.
- 96 KBytes of SDRAM are available.
- 512 KBytes of flash memory can be used for code.
5. Printer Firmware: Architecture

5.1 Conceptual schema
5.2 Class description

5.2.1 Main

The main file will be the director of the whole program flow.

On startup, initialization code is executed to initialize several subsystems such as: Serial communication, ZAxis configuration with encoders and motors, SPI bus and attached peripherals, laser driver, statistics logger module, and SD card interface.

After initialization, if no errors were registered, the printer enters idle mode until a command is received. Commands, very briefly classified, can be either G-Code instructions or proprietary printer instructions.

G-Code instructions may vary, but are usually for motor movement, laser on/off control, or several other status checking commands.

Proprietary printer instructions are meant to set hardware configuration options, job options, upload firmware, check current version, etc.

Finally, when the software (or the user, manually) sends a file to the printer, the printer loads the file. The user confirms the job as reported by the printer and the job starts.

During the print stage, the file is read line by line as valid G-Code instructions present are executed sequentially. When the print ends, build platform is risen to leave the final part in an easily accessible location for its further extraction and finishing.

5.2.2 config.h

The configuration header file contains the whole firmware configuration. Around 70 constant definitions hold the current printer configuration. These include all subsystems of the printer, both at a software and hardware level, and are critical for the correct and secure operation of the printer itself.

Small helper classes like Position and Instruction are declared in this file, as well as all global variables.
5.2.3 utils.h

The utilities file contains a dozen helper functions that are used by all parts of the program. These range from input/output and interrupt handling all the way to string parsing and user feedback. Some are declared as inline for optimization purposes.

| utils.h |
|-----------------|-----------------|
| #include <stdio.h> |
| #include <string.h> |
| #include <math.h> |
| #define _GNU_SOURCE |
| static const int min = 0; |
| static const int max = 100; |
| #define MAX(x, y) (x > y ? x : y) |
| #define MIN(x, y) (x < y ? x : y) |
| #define clamp(x, a, b) (x < a ? a : (x > b ? b : x)) |

5.2.4 ZAxis

The ZAxis class handles all Z axis movement and feedback subsystems.

On initialization, encoder and motor interfaces for both stepper motors are initialized. Sensors, such as end-stops or resin level sensor, are also read and local variables that hold their value are updated accordingly.

Homing procedures, travels, and layer changes are all handled by ZAxis methods, in a non-blocking way whenever necessary.

5.2.5 StatLog

The StatLog module keeps track of printer data for statistical purposes. It handles all the saving and loading to and from the SD card. Some of the parameters that are stored for further reference include:

- Laser on time
- Machine on time
- Jobs started and finished
- Any errors that have occurred

5.3 Third-party software used

5.3.1 Arduino SPI implementation

For the purpose of prototyping as fast as possible, the Arduino implementation of SPI was used. It seemed like a good implementation and offers up to 3 hardware select lines, so the requirements were met. It has behaved flawlessly with two Texas Instruments high-speed DAC’s, operating with clock rates of about 10MHz; as well as with the SD card, operating at even higher speeds.
5.3.2 sdfatlib

The sdfat library contains multiple initialization and helper functions that allow direct interfacing with SD cards working at 3.3V. It provides read/write access to FAT16/FAT32 file systems on SD/SDHC flash cards. No need to reinvent the wheel here, so, with a few modifications it served adequately for our purposes.

5.3.3 AccelStepper

Most stepper motor movements, except homing and a few travels, are essentially layer changes. Layer heights oscillate between 0.015 and 0.03 millimeters in all our tests, so it’s not really necessary to calculate accelerations and decelerations, nor move motors in a non-blocking way. Anyways, the AccelStepper library provides an object-oriented interface for the control of stepper motors in a non-blocking way, supporting accelerations, decelerations, multiple simultaneous stepper motors, and other interesting features which is why it was chosen for use in this prototype.

6. Conclusion

This document has covered some of the many technical aspects that have been part of the creation of a stereolithographic 3D printer from the ground up, covering hardware, electronics, firmware, and software.

Unfortunately, I was forced to focus the project from a software point of view, and so I did. But a brief summary of the hardware and electronics was covered on section 1.4, which is meant to give a feel of what that part of development was like. On later sections, a much more in-detail view of the software engineering part for both the end-user application and the printer firmware was presented.

The development of both the hardware and software was done in about a year. Everything has been successfully completed and tested except the end-user software, which has been paused for now due to lack of time.

The Lightcam 3D printer was officially presented in Rome, the 28th of October of 2014, to professor Andrea Vitaletti, researcher at “La Sapienza” University, and some of the DIS lab team. This was his final impression:

“If one of the purposes of a technical university is the formation of good engineers capable of developing a product based on state-of-the-art technologies and compliant with credible and demanding user requirements, I think Abel is an excellent example of a success story. I evaluate his work among the top 5% of the technical thesis made by students in this university”