**Collider Software Interview Assignment**

**Continuous SLA/DLP Printer**

# Introduction

In this assignment, you will be designing, and coding up a basic SLA/DLP 3d printer!

A few things to know, that might bring you some peace of mind.*[[1]](#footnote-0)*

1. **It is not expected that you are be able to complete the entire assignment in the allotted time--we know as an interesting candidate you have a busy schedule. What is most important is NOT full completion, but rather a well-designed system which has design that you can explain when questioned. If you do not have time to complete something, please take a few minutes to think about how you would fit it into your system design, given the time.**
2. **Parts of this assignment are intentionally vague, and parts of it are intentionally verbose. We want to know how our potential peers can extract useful information, and deal with ambiguity in instructions.**
3. **You may use any resources available to you, however if you extensively use an external library, or you take code from somewhere else, you must cite your references, and provide a link to the author/source.**

**If you have questions about this assignment, please speak with your technical recruiter, and they’ll make every effort to clarify.**

With that out of the way, let’s get to the fun stuff :)

# System Description

### As mentioned above, you’ll be coding a simple SLA/DLP 3D Printer. If you don’t know what that is, or need a refresher, you should do few quick searches on the topic and become familiar. There are many youtube videos of homemade DLP 3dP machines. The one you’re making will be a little different than many other DLP machines, in that it will not have any peel movements, but that is the primary difference.

### At its core, the system you’ll be designing software for looks like the below diagram. The only extra element you should be concerned with are the limit switches, which in the software model, are contained within the Liftmotor (Z height control) system.

### 

*Source:* [*http://tjsnyman.com/tjsnymanblogspot/wp-content/uploads/2011/04/diagram.gif*](http://tjsnyman.com/tjsnymanblogspot/wp-content/uploads/2011/04/diagram.gif)

### LiftMotor / Limit Switches

This is the linear actuator / limit switch combo that provides the motion of the buildplate for the printing process. Because you probably don’t have a liftmotor, you won’t be implementing real versions of these functions, but making a mock Liftmotor object that behaves in a manner consistent with the descriptions above. You won’t be sending commands, as the description says, but rather just returning values and performing operations on this mock object.

There are varying ways of achieving this goal. The liftmotor class (if you choose to make it a class) can have as many variables and extra functions as you deem appropriate for the simulation of this object. Two useful variables might be the current states of each of the sensors.

### Projector

This is another piece of hardware that you will be making a mock object of. It’s simpler than the liftmotor, it only has three main functions; to send an image to the projector, to turn on the LED, and to turn off the LED. Again, this is a mock object--so it doesn’t have to talk to any real hardware, but it would be cool if it produced some sort of screen output to signify the projection of slices, and the turning on and off of the LED. More details in **Requirements**.

### PrintData.tar.gz

This is the source of the parameters for the print.

It’s a tarfile that has the following contents:

* printersettings
* layersettings.csv
* Slice\_1.png
* Slice\_2.png
* …
* Slice\_N.png

You should be provided with this file.

# Requirements

(These are written as a series of statements, which your code should implement.)

### Operating System & Language Specifications

* The Machine’s code operates on a linux distribution such as debian, ubuntu, openembedded, etc. If you don’t have a computer with these OSes, it’s okay. I’d recommend using a virtual machine, such as VirtualBox, which you can download for free and comes with ubuntu.
* The Machine’s code is programmed in whatever languages the programmer determines fit for the application. It is okay for the codebase to be composed of different programming languages. **Be able to explain your choice of programming language(s).** *For reference, at Collider we primarily use C/C++ for firmware, Python for Mesh Preparation / Computational Geometry, and Javascript for web programming.*

### Version Control Specifications

* The Machine’s source code will be version controlled, preferably using GIT, but optionally SVN, CVS, or any similar protocol--provided you can explain why you chose it.

### Documentation & Code Style

* The Machine’s code should have a README file that explains the code’s modules and files at whatever level of detail the programmer deems appropriate for the understanding of the code.
* The Machine’s codebase should have variable names and function names that are clear, both to the programmer and to the average programmer.
* The Machine’s codebase should be commented enough such that it is clear what is going on in the code---this does not mean that every line should be commented--there is such a thing as too many comments.
* The programmer should use classes when necessary, but not more than necessary. The creation of a class should be explainable by the system designer/programmer.
* The programmer should avoid Global variables except where these are unavoidable.

### Testing

* The Machine’s codebase must have a suite of unit/integration tests that can be run by a developer who has the source code. The detail and number of the Machine’s unit/integration tests are at the programmer’s discretion, but they should be sufficient to determine that the codebase is mostly performing as expected.
* The Machine’s README file should contain instructions for running the tests.

### Error Handling & Logging

* The machine should gracefully handle any errors that may arise, and store the relevant text of these errors so they are accessible to the user when they get the status of the machine via the “/status” endpoint discussed later.
* The machine should produce meaningful console output as it performs certain tasks. Alternatively, and for some bonus points, make the machine create a logfile where it stores machine data that seems fit to keep a record of.

### Actuators / Sensors

* Motor
  + A programmer is able to send commands to the mock motor that allow it to travel up and down, respectively, by a specified number of microns (i.e. moveup(100) moves motor 100 microns up, etc)
  + A programmer is able to send commands to the mock motor that allow it to travel until it reaches the top, and bottom limit switches, respectively, and this function returns immediately, even though the motion of the mock motor does not complete immediately. (i.e. moveToTopLimit() sends motor to the top limit switch, and returns immediately, even though the mock motor is still “moving”...think multithreading here.)
  + A programmer is able to retrieve the status of the limit switches that are connected to the mock motor.
  + The mock motor moves at a given speed (e.g. moving it from the bottom to the top takes time, as it would in the real world. Again, think multithreading here.)
* Projector
  + A programmer is able to send a black and white .png image to the projector using a function. This function immediately returns, but this takes some amount of time to actually occur. Bonus points if this function actually writes to the screen somewhere. Bonus points x2 if you can make the image cover the whole screen.
  + A programmer is able to turn the LED of the projector on and off. (When the LED is on, the projector will be curing the resin for a single layer of the part)

### Print Process

* When the machine receives the start command, the machine will use its liftmotor to move the build plate to the top sensor. When that motion is complete, it will move the liftmotor to meet the vat, or the bottom sensor, going to its “Start Position”. The liftmotor will cease this motion when it meets the bottom limit switch. At this point, the code will wait for this motion to complete. While it is waiting, the code will write the first slice to the framebuffer.
* When the motor has reached the bottom limit switch, the computer will wait the number of seconds specified in the ModelApproachWaitMS variable in the layersettings.csv file for that particular layer. This value is in Milliseconds.
* After waiting the specified number of seconds, the computer will turn on the projector LED to begin exposing the photopolymer to light--the computer will keep the projector on for the number of seconds specified under the ModelExposureSec variable in “printsettings” for that particular layer. When the exposure is complete, the LED should turn off. **This is a very time sensitive operation, so the system needs to make sure that the time of exposure (the time the LED is on) is as close to exact as is possible.**
* The computer should then move the liftmotor upwards by the amount specified in the printersettings file under “layerThicknessMicrons”
* The computer should perform the same Delay, exposure, movement pattern until there are no layers left, and then it should go to its Home Position, the top of the linear actuator’s travel, where it will meet with the Top Limit Switch.

### Webserver

* The Machine has a (probably quite basic) webserver at the address **[HOSTNAME]:8000** with at minimum, the following routes:
  + **[HOSTNAME]:8000/printData**
  + **[HOSTNAME]:8000/command**
  + **[HOSTNAME]:8000/status**
* The Machine listens on a webserver at the route **[HOSTNAME]:8000/printData** for a printData.tar.gz file to be posted (almost certainly as a multi-part request).
* When The Machine receives the printData.tar.gz file, it stores it at some appropriate-seeming location (again, assume a normal linux directory structure and make your best effort to choose a good location for these files), and then does one of two things:
  + If it is not currently printing, it makes the received print data the currentPrintData (the printdata file that will print when the printer starts)
  + Otherwise, the received print data will become the currentPrintData AFTER the print has completed.
* The Machine listens on a webserver at the address **[HOSTNAME]:8000/printerCommand** for several commands, including “start”, “pause”, and “cancel”
  + For a “start” command. When this command is received, the machine begins to print with the parameters and slices stored in the currently active print file.
  + For a “pause” command, the machine turns off the LED, and waits until another “start” command is sent.
  + For a “cancel” command, the machine essentially acts as if there are no layers left, turns off its LED and goes home.
* When the webserver receives a GET to the **/status** endpoint, several things happen, depending on the situation:
  + If the machine is printing, it will send a response with {‘status’: ‘printing’} somewhere within the response.
  + If the machine has print data, it will send a response with {‘print\_data’: True} somewhere in the response. If it doesn’t, this value is false.
  + If the machine is in a state of error, the response will contain {‘status’: ‘error’, ‘error’: **[Basic String Description of Error]** }
* The webserver will have any other Post, Get, Update endpoints that the programmer sees fit for the functionality of the machine, provided their functionality is fully described somewhere in the documentation.

## More Bonus Points

* Choose or design a Z-Axis liftmotor / Motion system, and explain your choice. If you want, write your code such that it can interface with the liftmotor you have chosen.
* Choose or design a Projector, and explain your choice / design. If you want, write your code such that it can interface with the projector you have chosen.

# Concepts, and Hints

We’d like to see how you’d go about designing this system yourself, however there are a few concepts which, if you’re not familiar with, might help you in your endeavors.

* Finite State Machines
* Multithreading / Multiprocess conventions
  + Mutexes, for shared resources
* Object Oriented Programming
* Basic HTTP
* Real-Time Programming Conventions

1. *"Peace of mind is the most important prerequisite for creative work." - Richard Feynman* [↑](#footnote-ref-0)