4-28-2022

Hydra Research 3D Printing Error Detection

Braeden Lane

Senior Honors Project Final Report

University of Portland

Shiley School of Engineering

Class: EGR 483/484

Hydra Research – Multi-Disciplinary Capstone Project II

Hydra Research 3D Printing Error Detection

Braeden Lane

Group Members: Jordan Jhaveri, Maxwell McAtee, Kavita Thomas

Advisor: Dr. Surj Patel

Industry Representative: John Kray

April 28, 2022

Introduction

The Hydra Research Final Report will provide documentation of our team's prototype for our project. Our goal was to design an error detection prototype that can be implemented in Hydra Research LLC's Nautilus 3D Printer. Errors are common in 3D printing, so our goal was to design a device that detects errors in a print job, and using image processing, communicates an error message to the user. The user will then be able to decide if they would like to stop the print job or continue. The team successfully provided proof of concept showing this is a viable design. This document will provide details of our design, both hardware and software, as well as the results of our design. We will be documenting our milestones and to what extent we accomplished our goals. In addition to this, we will be discussing alternative designs that could be possible given more time, what went wrong, what went right, and future work.

Overview

The prototype we developed can capture images of a 3D print job at regular intervals and is controlled by a Raspberry Pi. This allows users to keep an eye on their print job throughout the process to make sure that there are no major errors. This is a useful tool for people looking for an economical option when printing large jobs but not wanting to "babysit" the job as it is printed. This can save the user time and material by having a way to check in and make sure that progress is being made appropriately while helping to avoid potential mechanical damage caused by errors in the print.

Figure one shows a block diagram of the basic structure of the prototype's design. The Raspberry Pi automatically captures images with a 4K camera that is installed in the back of the printer. These images are sent back to the Raspberry Pi and used in an object detection algorithm where the live image is compared to a 3D model of the print job. Any large discrepancies between the images are reported to the 3D printer firmware, which sends the status of the printer to the Raspberry Pi.

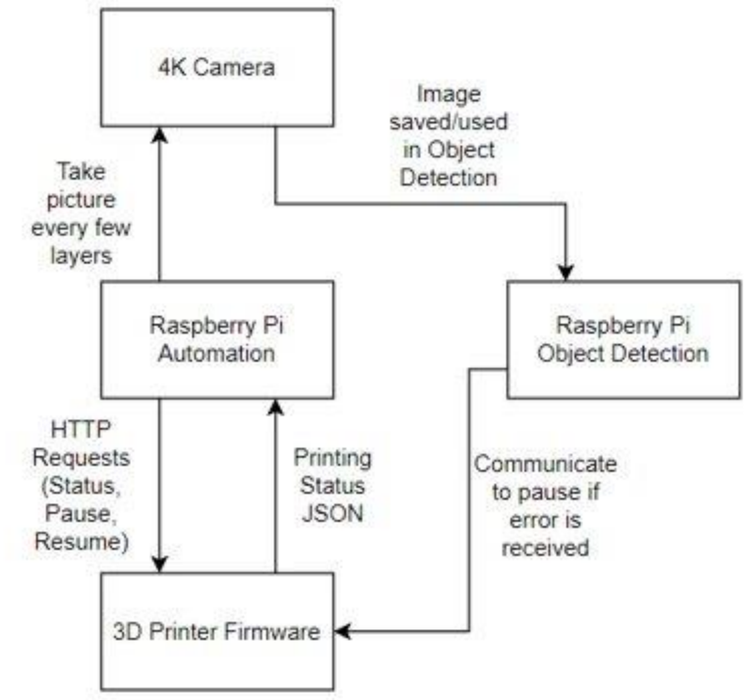

4K Camera
Image saved/used in Object Detection
Take picture every few layers
Raspberry Pi Automation
Raspberry Pi Object Detection
HTTP Requests (Status, Pause, Resume)
Printing Status JSON
Communicate to pause if error is received
3D Printer Firmware

Figure 1: A block diagram of the structure of communication in the prototype.

Context & Social Impact

Our main motivation for this project is to save the customer money and time. A device that can detect errors mid-print can stop a failing print before it uses more filament than necessary and save time spent waiting for a job that will not turn out as a user intended. Saving filament is great for customers economically, but it also saves them from adding more waste to the environment, as prints that include errors tend to be thrown out by users. Even if it happens to only be a small amount of material saved if an error is caught early in a print, this waste can quickly add up if errors occur frequently in 3D printers. As 3D printers are becoming more common in the everyday home, the ability to cut down on the waste from failed prints can make a dramatic difference in the amount of filament being wasted and thrown away.

Finally, our device aims to help address safety concerns and potential damage to the printer, as well as preventing any machine downtime if repairs are needed due to an error. While it is more uncommon, sometimes print errors can result in a backup of filament and either damage the extruder or lead to a fire hazard. Too much filament being spit out can be a hassle to clean up for the user, but it can also damage the printer by getting hot material on elements that are not meant to withstand such heat. Ideally, a successful device will lower or eliminate these errors from occurring.

Hardware Components

The hardware components in this design are the camera, light, and the Raspberry Pi, which is a small microprocessor that can store and run code. We used a webcam with 4k resolution mounted in the back corner of the printer to capture images of the print job. When the print job pauses, the extruder moves to the back corner. When the job resumes, the extruder moves to the front corner to deposit excess filament and the camera takes a picture of the print job in progress.

To get a stable image, we needed a way to mount the camera in the printer. We considered many options, such as raising the ceiling of the printer to get a true top-down view, removing the back to put the camera inside, putting a tripod inside the printer, or screwing the camera into the front of the printer. After completely disassembling the printer, we decided it would be easiest to put the camera on a gooseneck, welding the screw and the mount together to keep it stable. A gooseneck is a long, sturdy, and flexible metal rod that can be a reliable mount for objects that need to be placed in openings that are bigger than the object itself. To test the gooseneck design, we removed the back of the printer and put the camera inside, wrapping it in a plastic bag to try to keep the heat inside. We liked the results we got and decided the next step was to cut a hole in the back of the printer to have a more professional and consistent angle for the camera.

This plan required approval from the school to make major and permanent changes to the printer. In lieu of that, our industry sponsor, John Kray, sent us a back panel we could alter for the project’s needs. With the help of one of UP’s engineering technicians, Jacob Amos, we cut a square opening in the left corner where we inserted the camera and can get a good angle of the print job in progress.

Opting for a more consistent and immovable angle, we did away with the gooseneck and took a piece of welded metal screwed into the camera and clamped it to the back of the printer. This way, the camera could remain in place between each job and when moving the printer from one location to another. To make the angle and setup permanent, we plan to bolt the metal to the back of the printer.

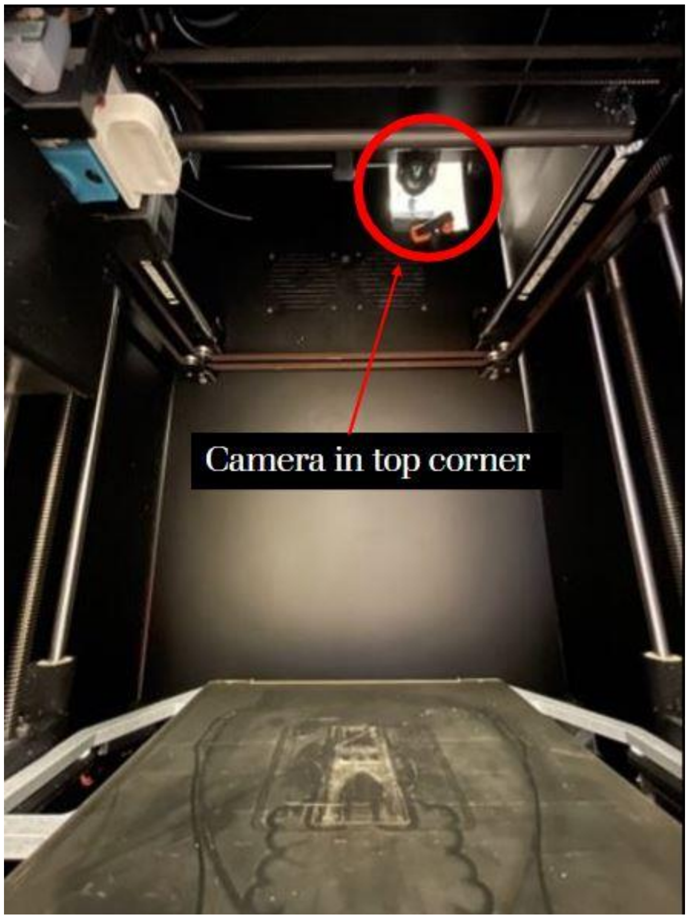


Figure 2: Picture taken from the outside of the printer looking in (from the door of the printer).

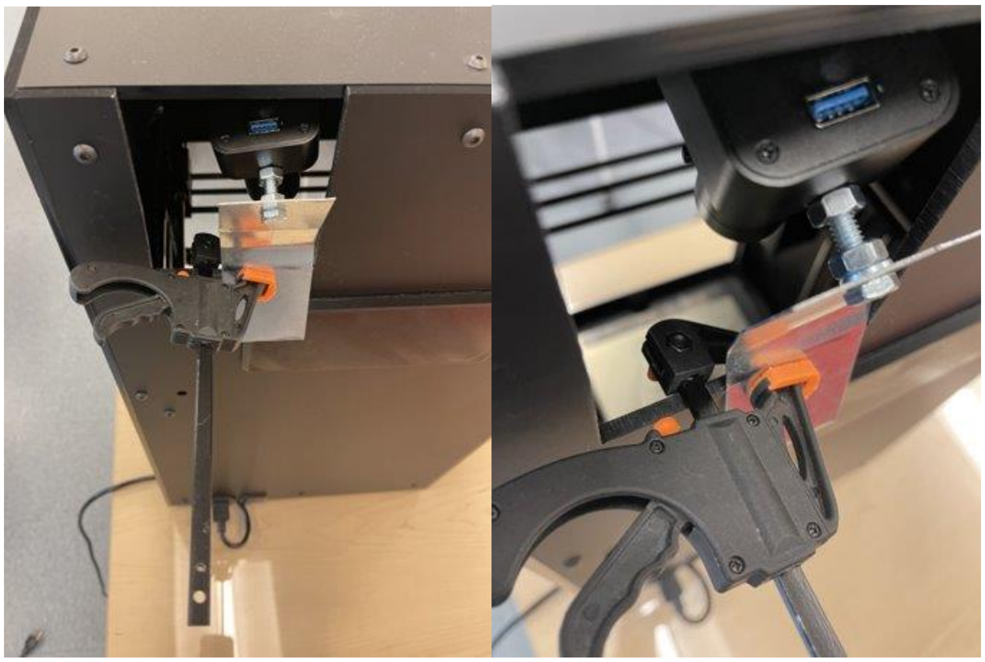
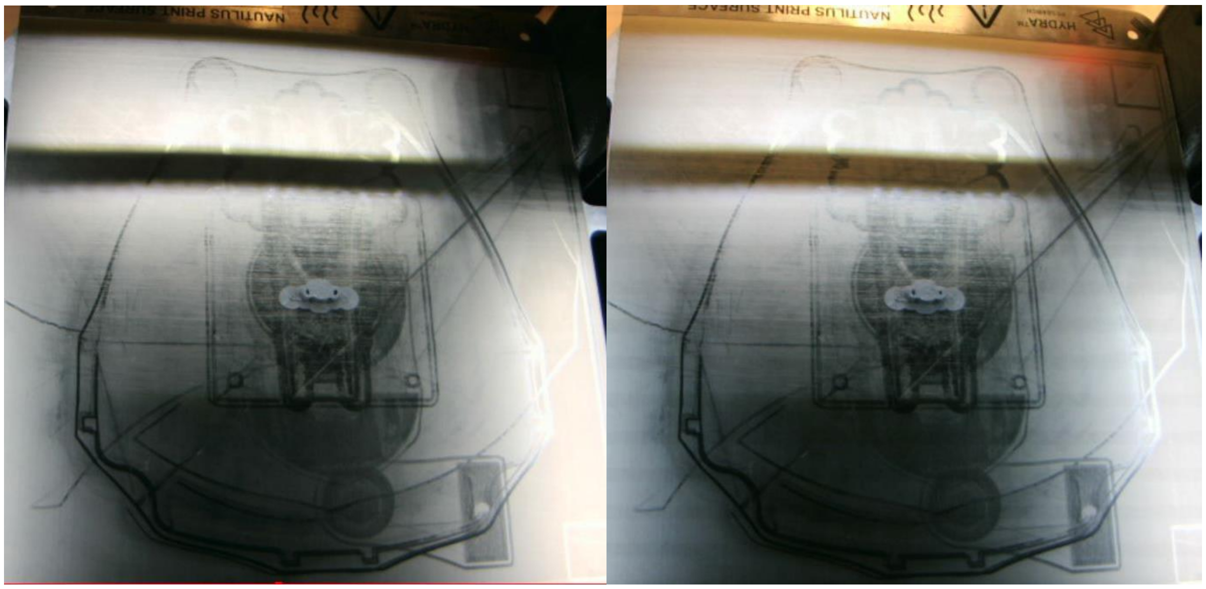


Figure 3: Camera mounted in the back panel sent from Hydra Research, LLC. We moved to a small piece of sheet metal that holds the camera at a more consistent angle than the previous gooseneck design. As can be seen in this image, there is still plenty of room for the USB connection to the camera, and the metal can be bent backwards or forwards for slight adjustments.

The camera is connected directly to the Raspberry Pi, which both powers the camera and processes the images. The board we used is a Raspberry Pi 3, and it is located outside the printer and connects via USB to the camera.

The printer has lights inside, but the camera is situated such that these lights cause a lot of shadows and create images that are difficult to analyze. Because of this, we turned off the lights inside the printer and installed our own light. This way we can control the brightness and the angle of the light, so the camera gets the best view possible. This was a challenge throughout the project because even with our own lighting, we were getting images that were either overexposed or underexposed. This made the image processing exceedingly difficult.

The lighting issue was one of the biggest problems we ran into throughout the project. The camera itself was advertised to have 4k imaging and was compatible with USB connections to a Raspberry Pi, which are the main reasons we chose this model. However, the camera does not zoom in enough or focus well enough to pick up the minuscule details that were needed for error detection. The lighting posed challenges because even using different settings for the built-in lights, the camera would automatically adjust the brightness, so they all looked the same. We put our own lighting inside the printer, but it did not improve the quality of the image very much. The object would be too bright on the side with the lights on it, and too dark on the side opposite the lights.



a. *b.*

Figure 4: Image (a) shows the print job with built in lights set to highest brightness. The image is too dark to notice the minuscule details needed for error detection. Image (b) shows the same print job photographed using the nightlight setting for the built-in lights. Comparing figures 4a and 4b shows that despite the stark difference in the light settings, the images come out identical.

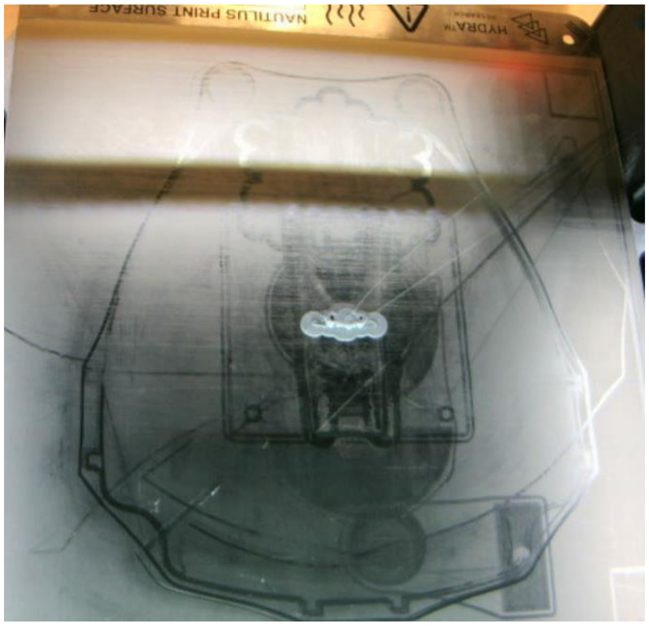


Figure 5: Using the nightlight setting for the built-in lights and using our own light strip mounted in the back of the printer we got the brightest and clearest image yet.

Software

The software for the prototype is written in the programming language Python so that it is compatible with a Raspberry Pi board.

I. automation.py

The main script we wrote is called automation.py. The functional part of the software uses the Selenium and OpenCV Python libraries. The core of the software uses the Selenium library to produce three WebDrivers, for getting the status, pausing the print, and resuming the print, respectively. WebDrivers are just a way for the Raspberry Pi to communicate to the printer over the network, as they share the same network during the process. After it calls these status requests, it can use the object returned by the printer to determine its current state. States include *Printing*, *Idle*, *Paused*, etc. It is set up so that it takes a photograph every *n* layers, where *n* is a layer interval that can be manipulated by the user by changing it in the Raspberry Pi's automation.py code. Currently, it is set to pause every 10 layers, which we found in testing to be a reliable setting so that the base of the print was not interrupted but was still checked early enough to catch errors that happened in the beginning.

The script then saves the image to the local directory on the Raspberry Pi so that it can be used by the templatematching.py script.

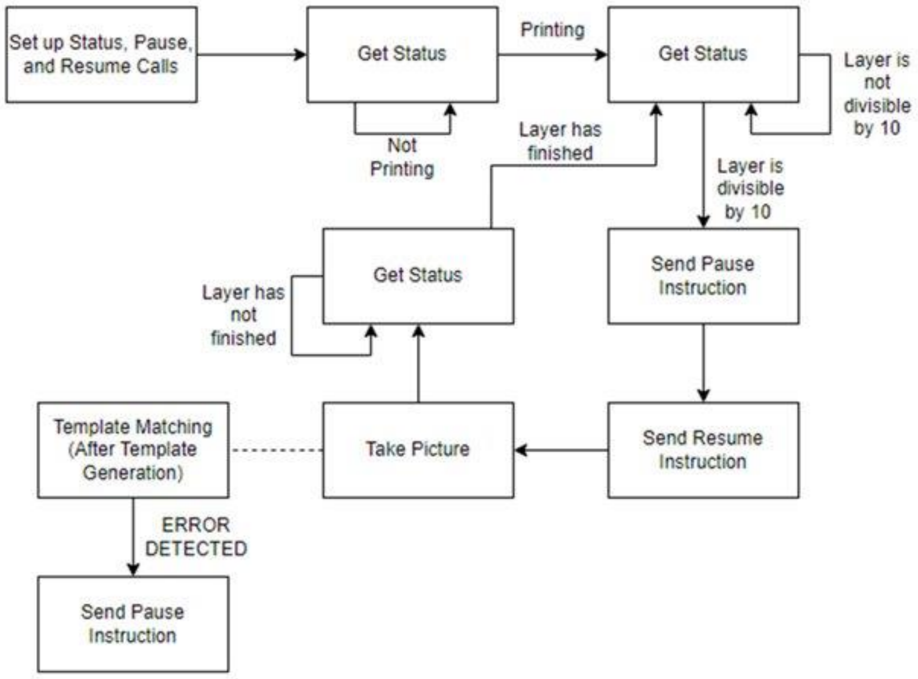


Figure 6: The block diagram above illustrates the process that the software uses to capture images at regular intervals. The diagram starts in the top left block and moves along as conditions are met. The block diagram uses a fixed point of capturing images every 10 layers.

II. <u>templatematching.py</u>

The templatematching.py script utilizes the openCV Python library for object detection, specifically the matchTemplate function that openCV provides. It reads in an image to overlay a template with, and checks to see how much the photos match, based on the pixel coloring.

Since we are working with a template-generated image from the .stl file of the 3D-model being printed and a captured image from a camera, there will be slight differences with the template and the actual image, based on coloration differences and point-of-view differences. To overcome these, there is a threshold worked into the function, such that the script identifies any matches in the image and the template that overcome this threshold. If a match occurs, then the print knows it can continue, since the object was detected. If no match is detected, it can be assumed that an error has occurred, and a pause command is issued to the printer to stop the print until the user resumes it.

In its current state, since render.py is not fully functional, templatematching.py relies on a specified input and template to compare it to, since no template can be generated by render.py.

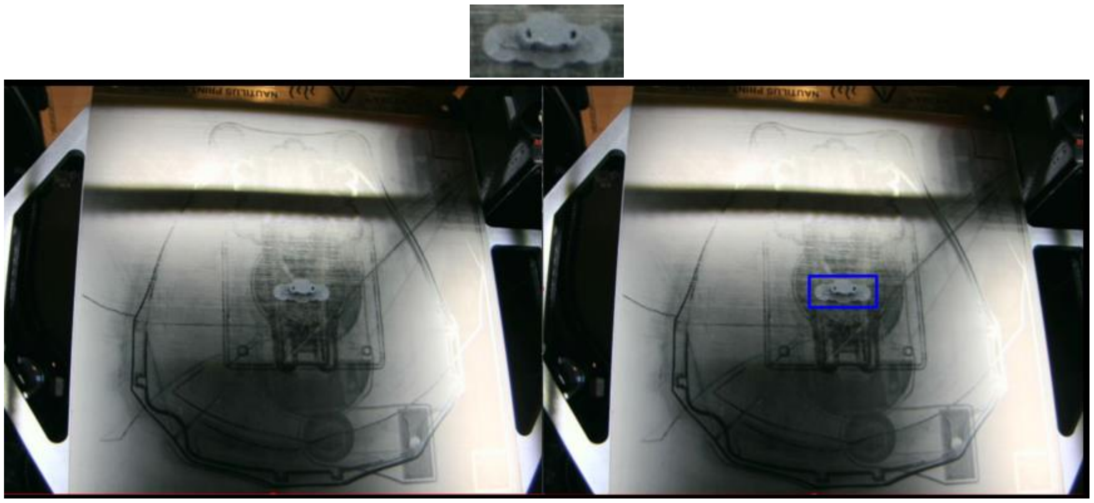


Figure 7: Template Matching algorithm adjusted to produce a box highlighting the spot that matched the template provided using a manually made template. The image in the bottom right of the figure was produced by templatematching.py.

III. <u>render.py</u>

The render.py script is utilized to take an object and render a template. It uses a path specified in the script to an .stl file, which utilizes the vtkplotlib Python library to produce a mesh, or 3D model, of the .stl file, and then save a 2D .png file of the .stl file from which the templatematching.py script can use as a template to overlay on the captured image.

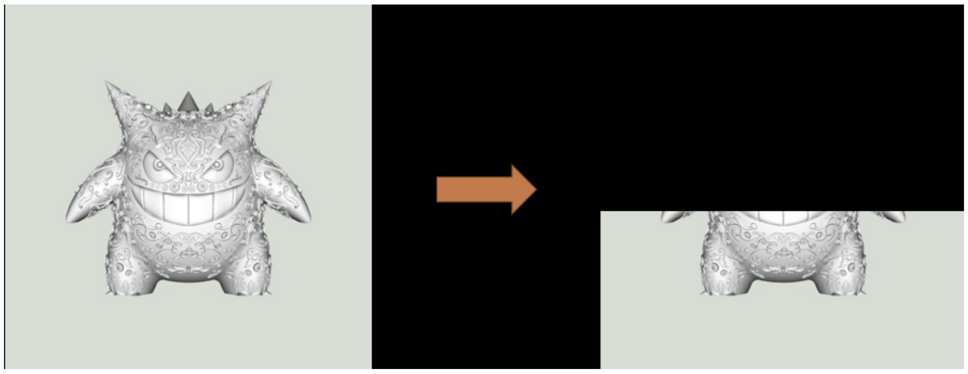


Figure 8: A visualization of the process needed for the project to be completed. The computer-generated image on the left would need to be sliced to the current layer the print is at. An example is shown on the right, manually sliced using a cover.

However, it should be noted that currently, the render.py script is non-functioning, as it should slice the .stl file to the specified layer that the print is currently at. If it could slice this .stl file into a mesh that represents the model at the layer the print is at, we could use this template to accurately compare what the print looks like to what is expected.

This is an integral part of the project, as without the slicing of the 3D model, we cannot produce a reliable template to use for our object detection script.

User Interface

The user interface is limited. The final product would not need a user interface as it would passively run whenever the printer is printing. However, for the prototype, there is not a user interface nor is the process to run the prototype automatic. Many of the files and variables need to be set manually in the code. If a customer were to use our prototype, they would need to start the algorithm by first starting up the Pi and going into the command line to run the script before starting the print job on the Nautilus printer. Ideally, with future work, we would automate the process of starting the prototype so the customer would not need to interact with the device at all, unless an error was detected.

Test Plan Results

Throughout this project, we tested the design at many phases. We tested the camera and the mount for the camera by taking the back of the printer off and checking the angle of the camera. We tested different placements of the camera to ensure the bed and the extruder would not hit the camera at any point during the print job. Once we realized the best placement would be at the top of printer in the corners, we decided to go in from the back of the printer so the camera could have a good angle when the print was resuming because the extruder would be out of the way.

After we verified the back corner was a good spot, we decided to find a way to place the camera more permanently. We got a small piece of metal and welded a screw to it that attached to the camera. We were then able to clamp this mount to the printer, giving us a smaller, more stable setup to take photos.

We also tested that the Pi can successfully take photos automatically every ten layers of the print job. We automated the process of image capturing and pausing the print job so that there are consistent images being taken that can be used to check the progress of the print job. On top of testing the automation process, we tested the template matching algorithm with manually created data so that we could verify that the algorithm would be able to detect a template within a live image.

Alternative Designs

When creating the design for our error-checking prototype, we spent a while brainstorming other ways of detecting errors. We wanted to make sure our final design would be easy to implement while maintaining a high standard of accuracy.

Our first idea was to use multiple cameras. This was part of the initial project proposal that came from Hydra Research. Using cameras seemed to be the simplest idea because we could use a Raspberry Pi, a camera, and a Python library to do some image processing and compare the photo to the slicer image. This is the idea we decided to go with, however, instead of using multiple cameras, we only used one. This is because of space constraints within the printer, challenges with mounting multiple cameras, and the trouble of combining multiple images into one cohesive image.

One idea we rejected was the use of infrared sensors to detect the changes in the layers. The thought was if the layer put down differed by too much from the previous layer, warping or layer shifting had occurred. The IR sensors would detect deltas, and we would use other data sensors (like cameras) for redundancy and guaranteed accuracy. However, due to the limited time we have on the project and the lack of knowledge surrounding IR and its implementation, we decided to stick with only the cameras. Also, it was pointed out that it could be difficult to get usable data until the printer got a significant way into the job, along with issues surrounding the size of the object being printed and it potentially not being in the path of the IR beams, due to the object's design.

We also considered using LiDAR to detect errors. This would potentially be more accurate than the cameras and could be used in conjunction with them as a way of error checking. This would be difficult to implement due to time constraints and lack of knowledge surrounding LiDAR and its implementation. Since our idea was to use the cameras and the LiDAR together, we decided to pursue the camera plan, get it working, and if time allowed, attempt to implement LiDAR. Both the LiDAR and infrared sensors would be detecting changes from one layer to the next (detecting deltas). This would not work to detect every kind of error we were interested in finding. For example, if the 3D object began peeling up from the bed, it would occur slowly, and the layers potentially would not differ significantly enough to detect errors.

We also had to spend vast amounts of time deciding how to mount the camera. The biggest problem was finding a non-invasive way to mount the camera without making any permanent changes to the printer. Once we decided to go with a single camera, we had to choose where we would put the camera in the printer. It had to be inside the printer to get clear enough images but needed to stay completely out of the way of the extruder so the camera would neither be bumped out of place, nor would the extruder be thrown off course. We needed to get a top-down view to capture enough of the surface area of the print job to identify any errors. This meant either mounting the camera to the ceiling or in the corner.

To avoid making permanent changes to the printer, we worked with the school's engineering technicians, Jacob Amos and Jared Rees, to brainstorm ways of mounting the camera. This was challenging because with the Maker's Space manager Allen Hansen leaving the university, no one knew the printer well enough to say what would be easiest. We considered removing the top of the printer and setting it on a wooden frame, then putting the camera on that wooden frame to get a true top-down view. This would have been extremely challenging because the top of the printer is connected to the front side, which is filled with complicated connections to the screen and other areas of the printer.

Another idea we considered was removing the back of the printer and putting the camera there, using some sort of insulation to trap in the heat. This was rejected because too much heat was escaping, and the camera angle would not be consistent enough between jobs – it would have to be adjusted in between each job to get a good angle. This would be challenging since the Raspberry Pi does not have a screen to show the user what the camera is seeing. It would require unplugging the camera from the Pi and connecting it to a computer with the camera's viewer software (OBS Studio) installed just to check whether the image was appropriate for the user's needs.

Real-World Applications

3D printers are becoming much more common in our world, as companies, like Hydra Research LLC, are aiming to make them more financially available to the everyday household. Because these printers are becoming more common place, it is essential to ensure the safety of the device to maintain the lifetime that the machine has and can be used, as well as to ensure the safety of the user and the environment. Our prototype comes into play to help prevent small errors from becoming large-scale errors that damage the machine. Because our prototype only adds a camera to the overall design of the Nautilus 3D printer that Hydra Research LLC makes, since our software can be put onto the Raspberry Pi that already runs the printer, this does not raise the overall price of the machine and creates a reliable way to ensure that users are able to leave their worries behind when using their printers.

Our prototype can also be expanded to help prevent errors in 3D printers that support industrial use. Like how 3D printers are becoming more common in households, they are also becoming more common in several industries, as they can be a very reliable way to prototype and test out potential products that companies are looking to produce. Being able to save the time and money for these companies could be beneficial in large scales, depending on how much the companies use the printers. If companies are also using printers for parts that they are mass producing, being able to catch errors saves massive amounts of plastic filament from being wasted if errors are being produced, and companies are unable to utilize the parts the need for their products.

Not only can it be beneficial to companies and the everyday user, but there has also been plenty of research being put into how the medical field can 3D print organs that are in short supply to people who need them. Because the organs being printed would need to be incredibly precise and expensive to make, ensuring that the organs come out exactly as they are meant to before being transplanted into someone who needs that organ is essential. While our prototype might not be able to detect the miniscule errors that could cause complications in organ printing, it is a step towards a goal where 3D prints, either organs or product prototypes, can be printed without worry that the other complications will arise.

Milestones

Due Date	Task	Description	Completed?
11/2/2021	Print something	Download the Hydra software, practice uploading a 3D print job, and learn how the printer works from user perspective.	YES
11/19/2021	Tell printer to pause	Use Duet Web Control to send a command to pause a print in process	YES
12/1/2021	Tell board to take photo	Using the Raspberry Pi, connected to a camera, to take a photo with the board and camera.	YES
1/15/2022	Research Handshake	Research a way for the Pi to handshake with DWC- report results, make plan	YES
2/2/2022	Access G-Code one layer at a time	Find a way to view the G-Code one layer at a time so that when an object is being printed the layer that is being put down can be compared to the same layer in the code.	YES
3/16/2022	Mount Camera	To get a stable image, we needed to mount the camera. We tried different methods and eventually settled on screwing the camera into the back panel.	YES
3/18/2022	Automate capturing images at regular intervals	Rather than manually pause the print job to take a photo every few layers, we made a script to automate the process	YES
3/25/2022	Fix Lighting	The built-in lighting in the printer was bright, but not good for the photos we needed to take. We installed our own lights to fix this issue.	YES

4/10/2022	Slicing STL file/Mesh object	To process and compare the ideal object to the image, we need a layer-by-layer image of the printed object	NO
Future Work	Overlay Sliced image with captured image	Putting the sliced image over the photo taken by the camera to check if there are any differences between the two within a certain margin of error	NO

Lessons Learned/ Future Work

Throughout this project we ran into many issues that became lessons learned and outlined work to be completed in the future.

The first major lesson was to be sure to choose a camera that would be more compatible with the needs of the design. In the future, it may be better to get a camera that can automatically focus on objects in the frame. While the camera we used was high quality and had an excellent lens for focusing on small objects and picking up details, it was challenging to test as every time the camera moved slightly, or the print job was larger or smaller than usual, the camera lens had to be manually adjusted to focus. This is a big issue, especially since the board is connected to a microcontroller, as there is no screen to check the image that is output.

In addition to having a camera that autofocuses, given more time we would test our prototype using multiple cameras to capture images from multiple angles. This was our original plan, but we had multiple recommendations from multiple sources, such as our client and Allen Hansen, about the number of cameras to use. We eventually chose to go with what our client wanted, which was a top-down view, using one camera. For future work, we would test both options and settle on the more efficient method

In terms of future work that could be done to the software, in a finished product we envisioned that the Raspberry Pi would start on startup of the printer and automatically start running the script that gets the status of the printer. The software in automation.py already checks for when a print starts, the program itself would be the part that needs to be automatically started upon powering up the Raspberry Pi.

One of the biggest lessons we learned from this project was that the image processing code needed to be one of the first portions of the project that we completed. Because we focused primarily on the automation of the image capturing, as well as the camera mounting and placement, we put aside work towards the later part of the project involving object detection and template rendering. Because we failed to see how integral this part of the project was, the project ended up not coming to fruition as we had envisioned, since the solution we envisioned relied on the ability to render templates with the approach that we took. After that approach failed, we ran out of time to try other solutions for template rendering, which led to a failed prototype.

Conclusion

Our goal was to create a prototype that would detect errors in 3D printing, in partnership with Hydra Research LLC. While our plan to capture images every few layers during a print was successful, the overarching goal of detecting errors in a print while it is being built has yet to be achieved. The prototype as it stands serves as a good model for a proof-of-concept, as the only piece missing from achieving the overarching goal is the ability to generate the templates using software to slice a 3D model to a specified layer. The resulting code can pause the printer and take a picture of the print on a repeating interval, as well as being able to detect a template given to it in a provided image. While we were unable to achieve the goal we originally set out on, the project is in a place where another team could complete it with further research into the slicing of the 3D models.

Bibliography

"IEEE Code of Ethics." *Ieee.Org*, June 2020, www.ieee.org/about/corporate/governance/p7-8.html.

IEEE Computer Society. "Code of Ethics." *IEEE Computer Society*, 1999, www.computer.org/education/code-of-ethics.

Raspberry Pi. "Raspberry Pi 4 Model B Specifications." *Raspberry Pi*, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

Appendices

Appendix A – RepRap Firmware Reference Guide

The RepRapFirmware is open-source software that is designed for controlling 3D printers, but it also has applications in laser engraving and cutting, as well as computer numerical control (CNC). The software runs exclusively on 32-bit microprocessors that are found in Duet electronics, which are boards that are engineered to support RepRap and help printers execute instructions accurately and quietly. RepRap runs every operation off G-code, which is a programming language that is most used in CNC. While G-code is a language that is relatively primitive in the fact that it does not contain loops and conditional operators, the use of macros is necessary when it comes to any sort of automation. A combination of macros and G-code can be sent through a USB to the board to execute the print job, but there is also an option for users to upload G-code through a web interface where the files containing the instructions for the print can be stored in the SD card for the board to interpret using the RepRap firmware.

More information about the RepRap Firmware can be found through Braeden's reference guide: https://docs.google.com/document/d/18VJqcqPXn8kginbI3mvbvc9jUI2oYt3nrEgrxISqO14/edit?usp=sharing

Appendix B – Duet Web Control Reference Guide

Duet Web Control is a user interface for RepRapFirmware and 3D printer management. It runs in modern browsers that support HTML5. Its primary functionality is to control the printer and manage the printing process. It can control and monitor prints, manage configurations, utilize macro files, process errors, and create a 3D display height map of the bed. A user can utilize Duet either through the Graphical User Interface or by entering G-code directly into a console.

More information about Duet Web Control can be found through Max's reference guide: https://docs.google.com/document/d/1tZZTLIAa0m7zuEXtusQrlCqX_WhRmJZdxbsFN6VNY5o/edit?usp=sharing