3D Printing Monitoring System

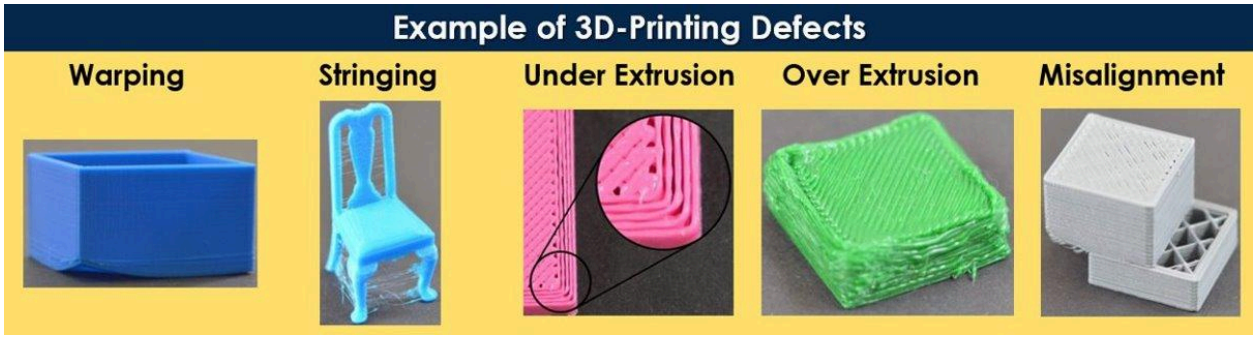
Richard Brown, Micheal Morgan, Anthony Tarchichi, Zachary Tucker, and Harrison VanDewater
Department of Mechanical Engineering
Rowan University
Fall 2023

Abstract

To avoid frequent errors in additive manufacturing, or 3d printing, a monitoring system can be devised to watch over ongoing prints and determine defects that occur. The 3D Printing Monitoring System at Rowan has five students working together to achieve this goal. The printer is connected to the internet via a Raspberry Pi that uses Octoprint to monitor the printer. By creating a g-code of a data basis object using parameters of a default print, defects can be added to the code and run on a printer with the monitoring system set up on it. This allows the group to obtain data sets for different defects that can train an artificial intelligence (AI) to recognize the error in other prints. The AI is trained on Roboflow and then called back to in Python code that runs on Octoprint to stop defective prints. This project is easily scalable to more defects, models, printers, and camera angles.

Introduction

3D printing represents a significant leap forward in additive manufacturing technology. Despite its promising capabilities, the printing process often encounters various challenges, leading to frequent failures. These issues, such as under extrusion, stringing, and spaghettification, are diverse and recurrent, significantly impeding the seamless execution of print jobs. Consequently, these common errors pose substantial hurdles to the reliability and efficiency of 3D printing technology. These deficiencies can be rectified by having a closed-loop monitoring system, that can see defects and adjust printing properties to remove this defect from the rest of the print. This is the scope of the 3D Printing Monitoring System clinic.



(Figure 1. Examples of Defects)

Similar Studies

This form of the monitoring system has been used in similar studies by Carnegie Mellon University, COEP Technical University, and the Dr. Daulatrao Aher College of Engineering. Each used Octoprint to create their own real-time online monitoring system. Octoprint is a publicially available program that primarily runs on a Raspberry Pi and can control a 3D printer remotely. At COEP Technical University and the Dr. Daulatrao Aher College of Engineering in India, they both use Octoprint and refer to the Raspberry Pi camera module to monitor the print. However, they used hardware encoders to

determine real-time defects (2, 3). Although functional, these encoders are not cost-effective, nor are they accessible without a length set up time (2, 3).

At the Carnegie Mellon University they used Octoprint to look at four different defects (1). These defects were the lack of build plate adhesion, extrusion stopping mid-print, layer shifting, and ”Hair-balling” (1). After multiple runs using custom computer vision algorithms, they had a %20 false positive rate, a %10 percent false negative rate, and performed with an overall %85 accuracy rate (1). Although custom computer vision programs offer complete control, open-source AI programs have more complicated algorithms already implemented.

Clinic Progression

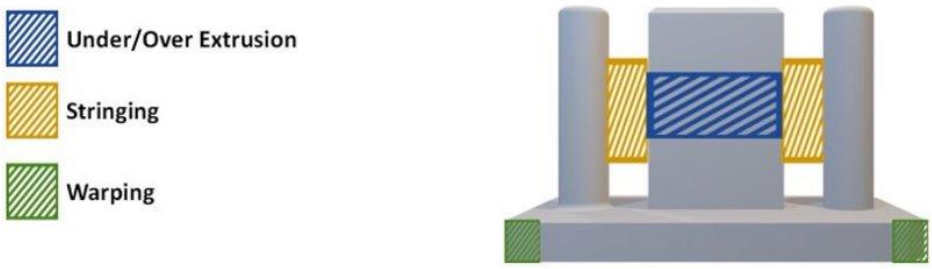
The 3D Printing Monitoring System clinic was tasked to develop a system using multiple programs that will watch and react to 3D printing defects. As a team, the clinic constructed a proof of concept for a closed-loop machine learning-based 3D printing system. This system creates a framework that can be further developed for private, commercial, or industrial use.

This clinic project objective was not only to create a monitoring system but to collaborate and work together as a group to efficiently deliver our tasks and make progress towards our goals. This was a new project created by Dr. Antonios Kontsos. The goal of this project was to create a machine learning-based system that can accurately and efficiently detect 3D printing errors using AI detection. The ultimate goal is to correct and modify errors in real time for various 3D print jobs. To accomplish this monitoring system, a webcam, a Raspberry Pi, Octoprint, and Roboflow were used.

Design and Approach

Base Object and G-code

The model used to test these errors was a set of two cylindrical pillars with a rectangular tower in the middle. As demonstrated in the figure below:



(Figure 2. Base Test Object)

The space between the pillars and tower was the testing area for the stringing error. When the printer’s extruder is moved from one position to another, the printer is told to retract the filament a specific amount to prevent stringing. If the filament is not retracted, the filament can slowly leak out of the nozzle, and when that leaking occurs over a space, it presents itself as small strings between those two points. The gaps between the pillars and tower were necessary to create these nucleation points for stringing. The object was sliced in Cura using the default settings to represent a basic printing type for many printers. These parameters are listed in the figure below.

Property	Value
Layer Height	0.2mm
Wall Count	2
Top & Bottom Layers	4
Infill	10%
Infill Pattern	Cubic
Nozzle Temperature	200.0 °C
Build Plate Temperature	60.0 °C
Flow	100%
Speed	50 mm/s
Build Plate Adhesion Type	Skirt
Skirt Line Count	3

(Figure 3. Cura Slicing Settings)

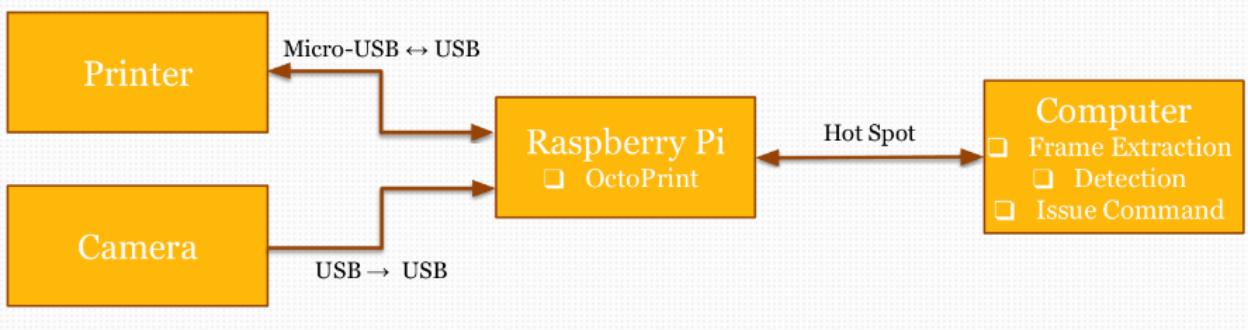
The g-codes developed for the clinic were made by slicing a base detection object in Cura. To train the AI model for errors, purposeful errors were inserted into the g-code exported from Cura. The AI model was trained on four different types of errors: under extrusion, stringing, and spaghettification. To create the under-extrusion, the g-code was modified to decrease the extrusion rate of the printer. Stringing errors were achieved similarly but also included a higher printing temperature and removing retraction in Cura. The spaghettification errors were created by telling the printer to create layers midair, meaning the extruded plastic would have no support underneath, and would result in long curls of plastic all over the print bed.

Camera and Connections

To have the ability to control the 3D printer, some connection is necessary between the printer and the host computer. The printer is connected to a Raspberry Pi which hosts the open source software named Octoprint. The computer hosting the trained AI and the separate webcam used are also connected to the Raspberry Pi. The Octoprint program is used to monitor the printer's status, and it also can use the webcam's input to create a live video feed of the printer. The computer takes a snapshot of the video feed every few seconds, and those images are fed into the AI for error detection. The figure below shows the connections in this system.

Our initial intention with connecting the Raspberry Pi to the host PC was through the campus' WiFi, however, we encountered issues while attempting to do so. The campus provides a wireless network that can be connected to by students and faculty. This network uses a captive portal security system which requires users to log in through a webpage. Unfortunately, raspberry pies are unable to connect to networks using a captive portal. Technically, there is one more way of connecting the Raspberry Pi to the network, but the method would use a port forwarding method of connection, which is very insecure for the network. This unfortunately left us with two options: connect the computer to the Raspberry Pi through a

wired connection, or use one of the team's cellular hotspot connections to connect the two. In the end, we chose to use a hotspot connection method to demonstrate the capability of the system working over a wireless connection.



(Figure 4. Connection Network)

The issues encountered in the process of connecting the system to a network introduced some difficulty when trying to extract images from the Octoprint program, as the provided stream link from Octoprint was unusable, and we had to extract the direct URL to the images. These images were extracted from the Octoprint program and were saved to a folder to create a data set. The Octoprint program uses the webcam connected to the Raspberry Pi to provide a video feed of the printer. Below is an image of the webcam mounted on a bipod stand viewing the print bed.

(Figure 5. Camera and Mount)

The webcam used was a Logitech C270 webcam with 720p resolution. This model was chosen because it required little effort to position it in a place where the images could be collected. The camera had a decent-quality image, but unfortunately, its fixed focal distance prevented us from positioning it too close to the printing part itself. Because of this, we had to limit our error detection to extremely obvious cases, as small errors would simply be unnoticeable in the image. In the future, it may be beneficial to mount a camera on the extruder frame and point it toward the nozzle. Being able to see the extruded filament leave the nozzle would greatly increase error detection time. More cameras with different viewing angles would also provide much more training data, as limiting our project to one camera with a single angle of print errors would likely result in a poorly trained AI model if the view had been changed. Limiting the training data to as few variables as possible is a good solution for training a model quickly, however, the model has great difficulty adapting to new situations, such as different view angles, different lighting, and different 3D printed parts. Adding more variation to the data could improve the adaptability of the end product, but would certainly increase the time needed to create such an AI model.

Object Detection Software Development

The goal of this part of the project is to make a dataset that the OctoPrint will use. This dataset takes the pictures from the camera and scans for the defects we want to find during the print. To make the dataset, we used Roboflow to compile images from the tests, and then we used bounding boxes to train the dataset model to have the learning algorithm be able to find the defects. We chose Roboflow due to the simplicity and many of the mentioned tools used are all encompassed in this program and shareable.

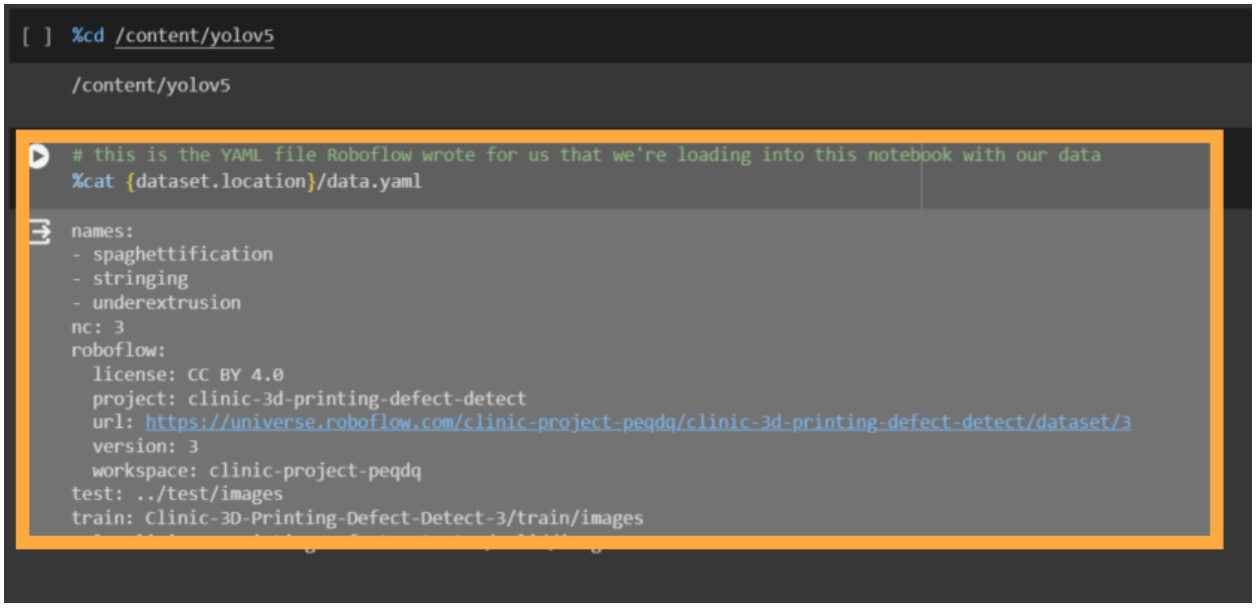
To make the dataset, bounding boxes were added to every image we used by hand. This entails making rectangles around the defects in each image. This can be seen below in Figures 5-7. When we finished making the bounding boxes, we classified each image by which defect each image was depicting, either under extrusion, stringing, or spaghettification. After this, the images were split between train, test, and valid. These three types of images are used to train the data, and make sure the trained data is on track during the training with the valid, or test the trained data.

(Figure 6-8. Samples of Bounding Boxes from Algorithm)

After annotating the images we added alterations to modify and multiply the images to broaden the amount of data for the algorithm. This includes blurring the images up to 2.5px and changing the brightness and saturation up to 25% in both the negative and positive directions. After doing this alteration, we made the images ready for training, so we imported the data into our code to train the data. This code is used in Google Colab, a Python-based coding platform to host our code. We first imported Yolov5, the object detection system we used. After this, we then export our Roboflow dataset, and instead of downloading the files themselves, we export a download code and then import it into the code by copying said code. This can be seen in the red box below.

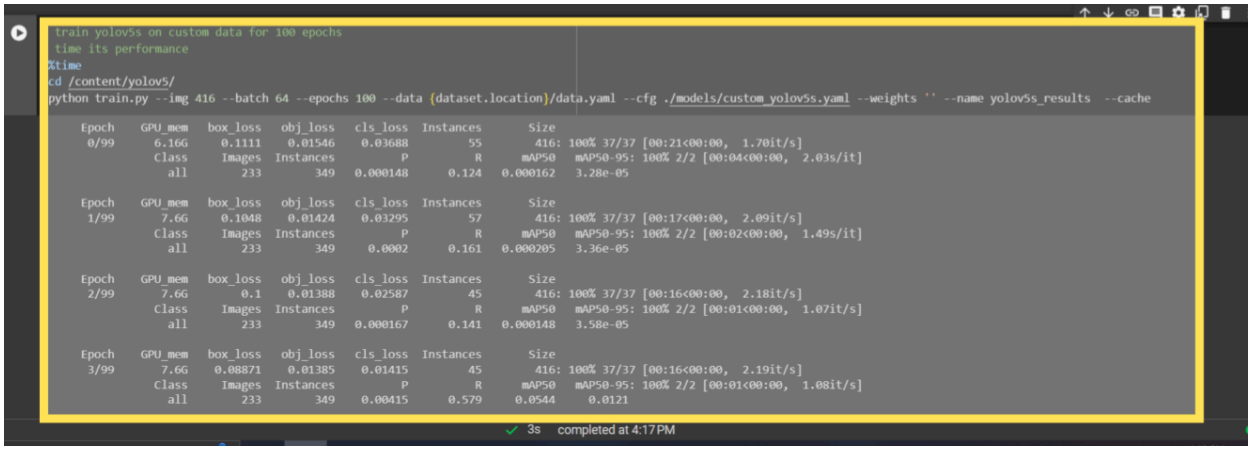
(Figure 8. Coding Block: Dataset Import)

The next code of note is shown below in the orange box. After downloading the data, this code recognizes the classes used in the data labeling and shows what kinds of boxes the data will label.



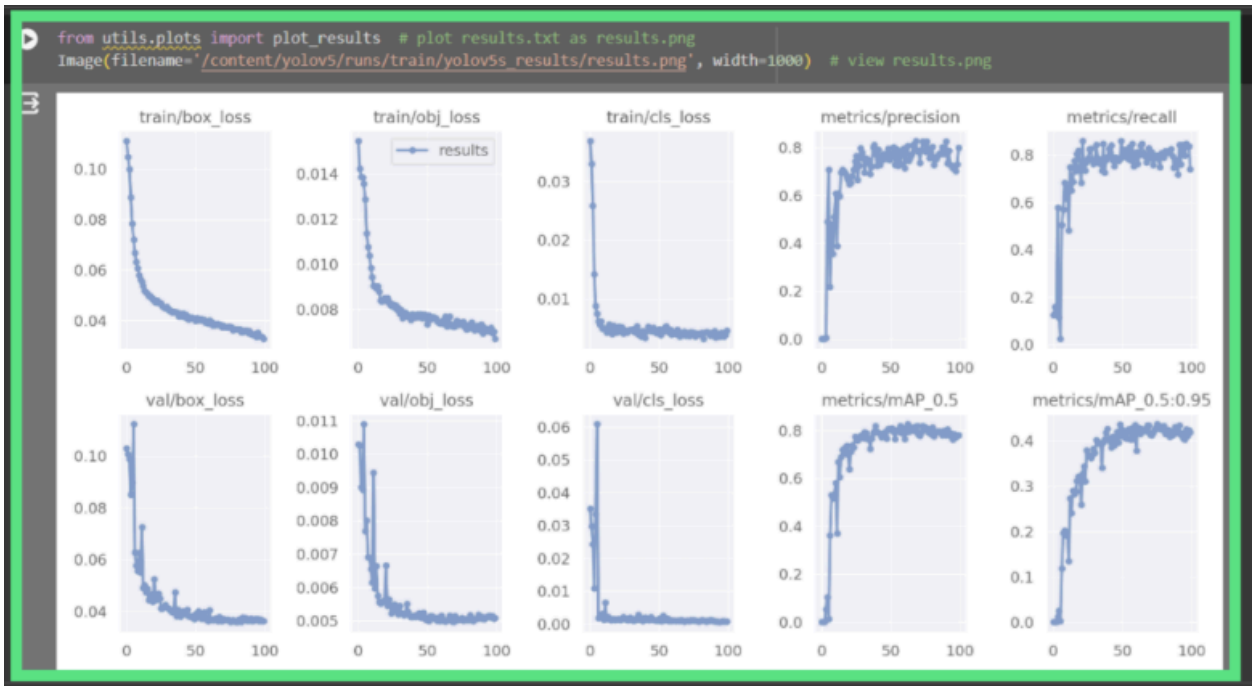
(Figure 10. Coding Block: Class Definition)

The yellow box is the main code for training the dataset. This code takes the images and labels and runs through them to develop an algorithm and create patterns to understand and apply the algorithm to later images. The main variables being changed are the batches and epochs. The batches are how many images are processed at a time when going through all of the images. Every time it goes through all of the data, that is what is referred to as an epoch. The epochs variable tells the code how many times to go through the images.



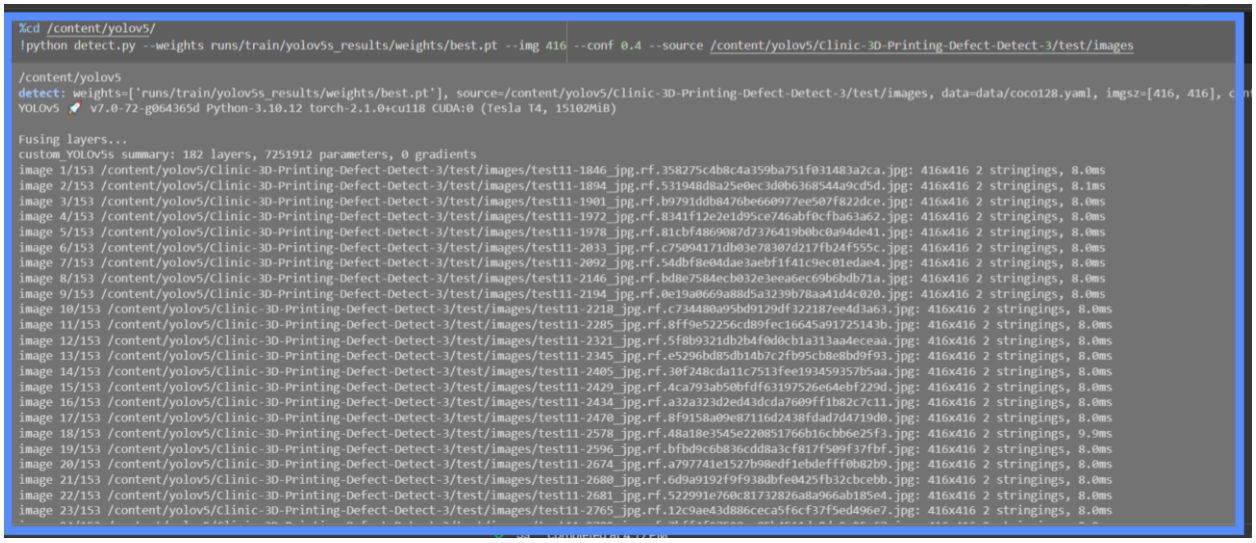
(Figure 11. Coding Block: Training AI)

After training the data, the results of the training are reflected below in green. These charts tell many of the specifics on how the training went, but the main information of note for our use is the metrics/mAP_0.5. This graph essentially shows the confidence level of the training, and how accurately the model can box the object it is trying to detect. As seen above in Figures ?-?+2, the intervals vary between each image. The higher the percentage this model can get, the better it will be at detecting defects. We were initially shooting for above 65%, but the peak our final model did was a peak of 80%, so this exceeded our expectations.



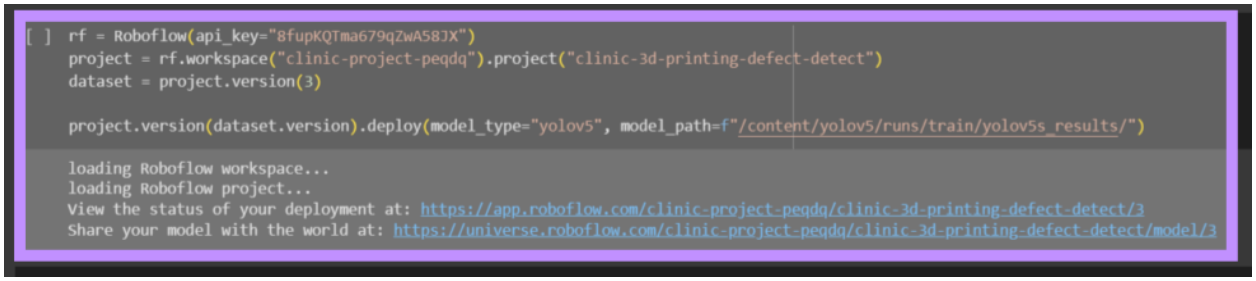
(Figure 12. Coding Block: Graph Statistics Output)

The code in blue takes the weights from the training data, which compiles what bounding boxes were the best and what was the worst. This allows for a more accurate run through the data without necessarily running through the training again.



(Figure 13. Coding Block: Training with Weights)

Finally, the purple code shows how the algorithm is exported. This is later used in the Python code in real-time to detect the images, instead of being manually supplied images.



(Figure 14. Coding Block: Exporting Trained AI)

Closed Loop Software Development

With each component of our system created it was important to connect them into a closed loop using Python. Development for this was undertaken in the PyCharm IDE because of its easy organization of multiple files and a team members familiarity with it. The first step completed when the code is run is to create an instance of OctoPrint which can be manipulated using the OctoRest API for Python. This is accomplished in the following lines of code.

```
def make_client(url, apikey):
    """Creates and returns an instance of the OctoRest client.

    Args:
        url - the url to the OctoPrint server
        apikey - the apikey from the OctoPrint server found in settings
    """
    try:
        client = OctoRest(url=url, apikey=apikey)
        return client
    except ConnectionError as ex:
        # Handle exception as you wish
        print(ex)
```

(Figure 15. Coding Block: Creating OctoRest Instance)

Once connected to OctoPrint we could connect to our AI model being hosted on RoboFlow. This also required an API key and additionally required the project name and version number as shown in the following code.

```
rf = Roboflow(api_key="8fupKQTma679qZwA58JX")  # Roboflow API key
project = rf.workspace().project("clinic-3d-printing-defect-detect")  # Roboflow project name
model = project.version(3).model  # Roboflow version number
```

(Figure 16. Coding Block: Creating YOLO Instance)

With both ends of our closed loop system connected in the same script we could then begin downloading and analyzing the picture feed from our camera. Rather than connecting to the camera directly through the Raspberry Pi we decided to use the snapshot URL generated by OctoPrint. This allowed us to call the latest image from the camera as a URL which made downloading the image substantially less involved than trying to interact with the camera directly.

```
def image_download(url, name):  # Download an image to a given location from a given URL

    imgURL = url
    fileLocation = (folder, name, '.jpg')
    fileLocation = ''.join(fileLocation)
    urllib.request.urlretrieve(imgURL, fileLocation)
```

(Figure 17. Coding Block: Downloading Camera Image)

After downloading the image was run though our AI model and it was determined whether a defect was present or not. This presented an issue since the output of our AI model was a JSON file. After some file manipulation it could be treated as a string which was searched for instances of defect names. If a defect was detected in an image the code would add 1 to a buffer variable, otherwise it would reset the buffer variable to 0. If the buffer variable reached 5 (i.e. 5 consecutive images had detected defects) then a command was issued to the printer through the OctoRest API to pause the print job. The buffer was

added to eliminate false positives which would cause the print job to pause without reason although we noticed very few false positives in our testing. This loop can be seen in the following code block which omits some of the logic used for looping only if the printer is currently in the state of “Printing”.

```
imageDownloader.image_download( url: 'http://192.168.112.115/webcam/?action=snapshot',
                               Name + str(n))  # Replace with the url of the snapshot
path = (r'C:\Users\htv8h\Desktop\New Test Folder 1\\', Name + str(n), '.jpg')  # Define the path to save the image
path = ''.join(path)  # Convert from raw string to string
inference = yoloV5.infer(path)  # Run an inference on the image

if 'x' in inference:  # Check if the inference contains defects
    print('defect:')
    """
    Check what type of defects are detected
    """
    if 'spaghettification' in inference:
        print('spaghettification')
    if 'underextrusion' in inference:
        print('underextrusion')
    if 'overextrusion' in inference:
        print('overextrusion')
    if 'stringing' in inference:
        print('stringing')
    buffer += 1  # Add 1 to buffer when defect is detected
else:
    buffer = 0  # Reset buffer to 0 if defect is not detected

if buffer > 4:
   octoRest.pause(Client)  # Pause the print job if 5 defects detected consecutively
```

(Figure 18. Coding Block: Main Loop)

When running the code prints the name of the defect detected each time a defect is detected. When a defect is detected consecutively 5 times the printer pauses.

Technological Impact Statement

By advancing the progress of additive manufacturing technology, the system built by this group is beneficial to society as a whole. By progressively enhancing the quality of 3D prints, the end user is guaranteed items that satisfy safety and performance standards. As a result of this, the market and audience widen and gain confidence as the quality in the 3D printing space increases. Development of this could easily scale into bigger things and become a benefit to multiple consumers. These customers could include bigger corporations like Quality Assurance professionals, Government agencies, and or smaller companies like 3D printing manufacturers and R&D teams. All things considered, the 3D Printing Monitoring System serves a wide range of consumers and provides advantages including greater productivity, cost savings, better product quality, and better trust in the dependability of 3D-printed goods. The benefits of this technology should be seen by many different additive manufacturing-related companies and spaces as it becomes more widely used in the future. Not only is this project an advancement from a technological standpoint but it also has an educational impact. This idea could be broadened and contribute to global manufacturing processes to reshape industry standards.

Ethical and Professional Responsibilities Statement

As we were going through this project, making sure the ethicality was sound as well as making sure the work we were doing was up to code was taken into consideration. Since we were dealing with AI, making sure the program is developed with a non-biased system would be important to make it so it works not just for our purposes, but those after us. We ensured this to be the case by leaving ample documentation throughout the code to help the next group of students in this clinic understand what the code entails, and how to train the detection system for the next groups of images. We also made it very transparent what the AI and camera are detecting, and they only work when in use for detecting and recording prints using OctoPrint. This allows us to make sure our work is professional and compliant ethically.

In a broader context the possible applications of our work are as varied as the uses of additive manufacturing. Despite this it is certain that there are a few ethical benefits from the development of the system. By reducing the number of failed prints we will be reducing the amount of waste generated as a result of additive manufacturing. This is both environmentally conscious as well as economically advantageous. Additionally we will be increasing confidence in additive manufacturing. This is environmentally advantageous because additive manufacturing inherently has less waste than subtractive manufacturing.

Conclusion

This semester has proven the capabilities of this clinic and the opportunity it poses. The printer is together and fully functional with a direct drive extruder. The camera was successfully connected to the Raspberry Pi and Octoprint was able to see through the camera and control the printer. Roboflow created a successful data set that the OctoPrint Python script could refer to. The clinic was successful in combining AI training and Octoprint to identify real-time defects and stop an ongoing print.

In the future, we plan to add more sensors for invisible defect detection, find a permanent solution to the camera mount situation, incorporate different prints into the AI data processing system, and correct defects in real time. Similar to the camera, acoustic sensors relay real-time information with prints. The sensors would detect vibrations in both the nozzle and bed of the printer. These vibrations would be used to sense over-extrusion, stringing, warping, nozzle jams, and other errors. The sensor data would be trained in the AI alongside the image data to improve detection time and accuracy. A permanent mount for the camera will be made to account for complete consistency across future prints. The current bi-pod design for the camera mount was used to determine the optimum distance from the print, as the focal distance of the camera was fixed. In the future, a different camera will be mounted on the z-axis, or the extruder would be used for consistent data. To train the AI in the future, parameters that will change are the model, filament color, background for the camera view, the camera angle, and a larger variety of defects. To fix defects in real time we would need to implement logic to isolate the defects on the top layer that could be solved, we would have to do testing to determine how each defect could be solved by changing print settings, and we would have to create functions using the OctoRest API to send commands to the printer.

Acknowledgments

A major acknowledgment for this clinic project is Ahmed Ben Sidhom. He was a major help for all facets of our project and was there every day we met to be there to assist and guide us through what we needed to get accomplished. He was a major help in particular with the Object Detection portion of this project, as he gave us the baseline code to start with and work our model around.

References

1. Bas, Joshua, et al. “3D Printing Error Detection System.” *Carnegie Mellon University*, March 2020.

2. Kakade, Shubham, et al. “IOT-based real-time online monitoring system for open ware FDM Printers.” *MaterialsToday: Proceedings* , vol. 67, 28 July 2022, pp. 363–367, https://doi.org/10.1016/j.matpr.2022.07.210.

3. Prof.Gujar. M. P, et al. “IoT based 3D Printer” *International Research Journal of Engineering and Technology* , vol. 67, no. 3, March 2019.