Optical in-situ documentation and surveillance for FDM-Printing with integrated electronics

Dennis Struhs

Arbeitsbereich Technische Aspekte Multimodaler Systeme

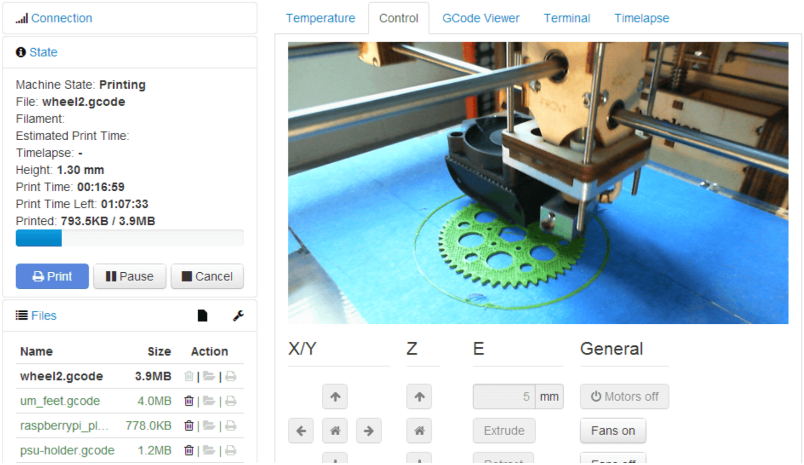
16.01.2018

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG
MIN-Fakultät
Fachbereich Informatik
TAMS

Topics Overview

Introduction

Common way of print documentation



Machine State: **Printing**
File: **wheel2.gcode**
Filament:
Estimated Print Time:
Timelapse: **-**
Height: **1.30 mm**
Print Time: **00:16:59**
Print Time Left: **01:07:33**
Printed: **793.5KB / 3.9MB**

Name	Size	Action
wheel2.gcode	**3.9MB**	
um_feet.gcode	4.0MB	
raspberrypi_pl...	778.0KB	
psu-holder.gcode	1.2MB	

Figure: Webcam based timelapse documentation of the print.

Good but not good enough

The previous frame showed the most common way of documenting a print job by using a webcam for a live time lapse. However there's several reasons why we want a better solution:

1. The image has a low resolution.
2. The image perspective is not orthographic.
3. Due to points 1 and 2 it's not well suited for computer vision based error detection.

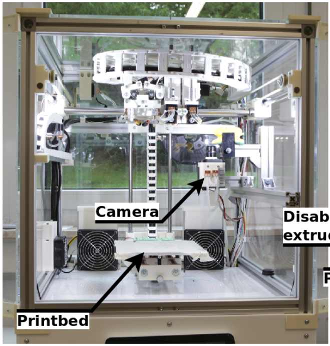
Solution idea

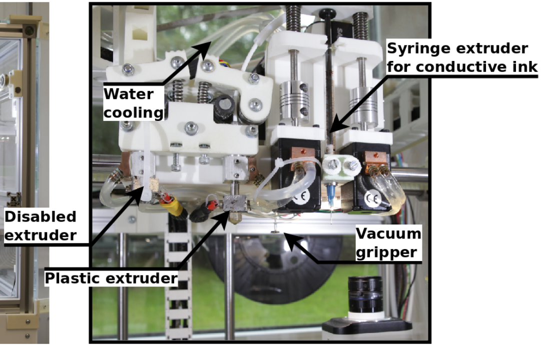
Instead of using a fixed webcam, we'll use a print-head mounted camera taking many small images instead of one big one. This grants us the following advantages:

1. A high resolution image without using expensive high-definition cameras that scales to the size of the printable object while preserving the resolution.
2. An unobstructed top-down view on each printed layer.
3. A picture that can be used for computer vision to detect errors on the currently printed object.

Current printer setup

The FDM 3D-printer has been modified to produce conductive traces with a second extruder during the printing process. SMD-components are then placed into the uncured conductive ink by a camera-guided pick and place system to complete the circuit.





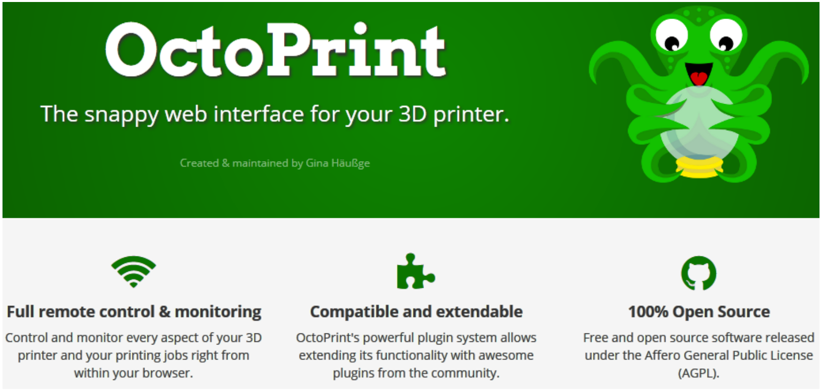
Thesis paper goals

Taking the previous points into consideration, the thesis paper should achieve the following goals:

1. Automatic consecutive image capture on finished layers.
2. Composition of the captured images into a big picture.
3. Using the composed pictures to detect faulty conduits.
4. Automatic correction of faulty conduits.

We'll achieve this by creating a new plug-in for **OctoPrint** which we'll call **OctoCamDox**.

What is Octoprint?



OctoPrint

The snappy web interface for your 3D printer.

Created & maintained by Gina Häußge

Full remote control & monitoring

Control and monitor every aspect of your 3D printer and your printing jobs right from within your browser.

Compatible and extendable

OctoPrint's powerful plugin system allows extending its functionality with awesome plugins from the community.

100% Open Source

Free and open source software released under the Affero General Public License (AGPL).

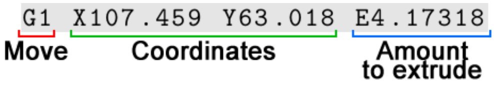
A web interface to control an FDM-Printer. It can load and print objects, specified inside **GCode**-Files.

What is GCode?

GCode files contain the information of the printed object as well as printer exclusive control commands.The movement commands, which we're particularly interested in, represent single points on the tray which will be all covered by the printer while placing the extrusion material.

Listing 1: GCode example

```
G1 X53.982 Y39.541 E2.54361
G1 X106.018 Y39.541 E3.65775
```



```
G1 X107.459 Y63.018 E4.17318
```

How to create GCode?

Slicing programs such as **Slic3er** can turn 3D Objects into FDM-Printer readable GCode by "slicing" it into layers.

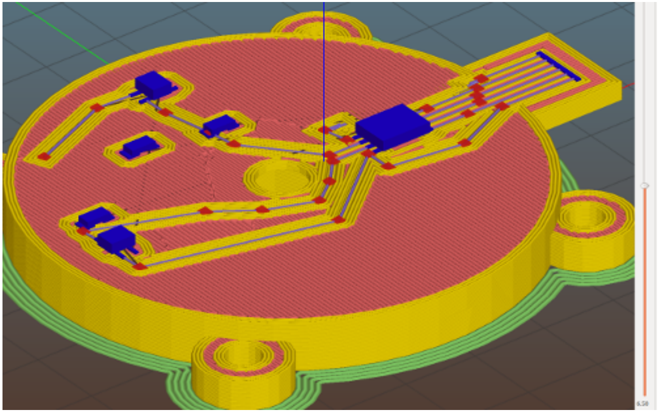


Figure: 3D Slicing

Picture generation

Simple GCode visualization

By simply drawing lines between GCode move coordinates we can create a simple GCode visualization, which will be used for the camera sector grid generation.

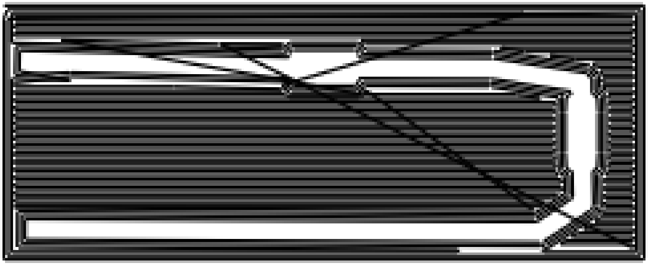
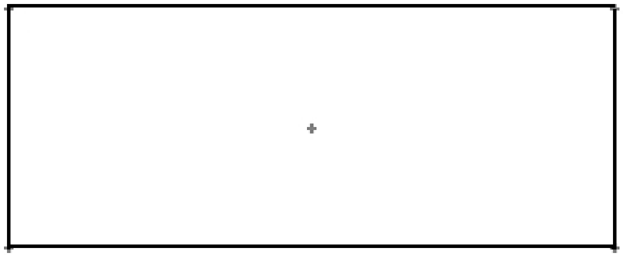


Figure: Simple GCode Visualization.

Setting up the bounding box

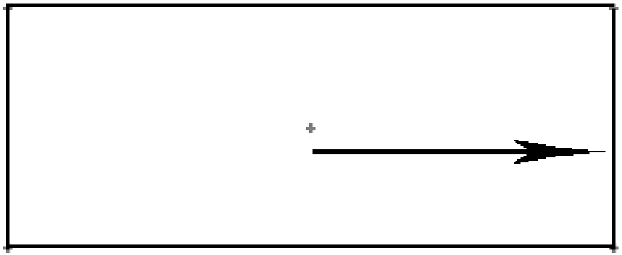
By using only the information provided by a single GCode file, the current implementation will generate the camera sector grid by performing the following steps:

1. Create a bounding box around the GCode object.



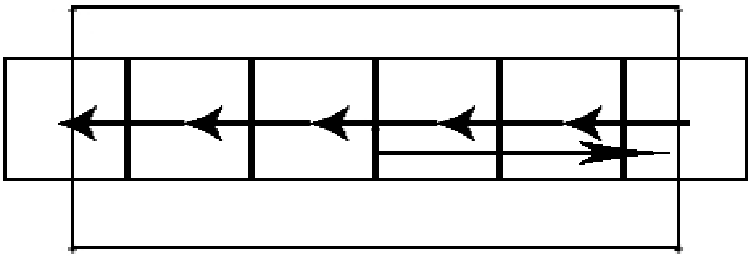
Finding the edge

2 Start going n$\times$camera-picture-width-steps right from the center of the bounding box on the X-Axis, until it leaves the bounding box the first time.



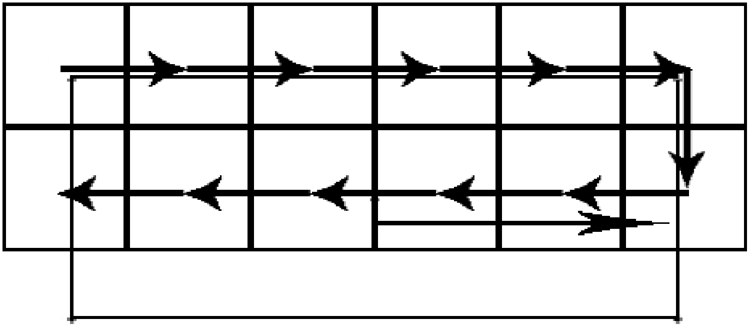
The first row

3. Add the center coordinates of the found grid tiles to our CameraGrid Array from right to left until the left limit of the bounding box was reached.



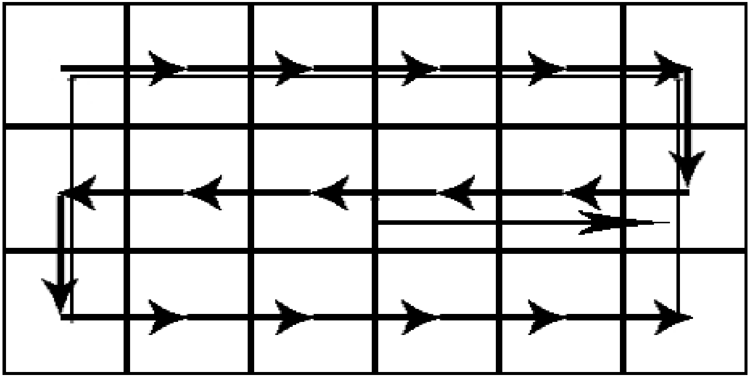
Expanding the grid

4. Move one camera-picture-width-step up on the Y-Axis and repeat the process from point 2 but this time from right to left. Do so until you reach the upper bound of the bounding box, reversing the direction each time.



Finishing the grid

5. Make a point symmetrical copy of the upper half of the grid.



Applied example

After the algorithm finished, we'll get a nicely lined up grid over the original GCode. Unfortunately this grid is not optimal yet.

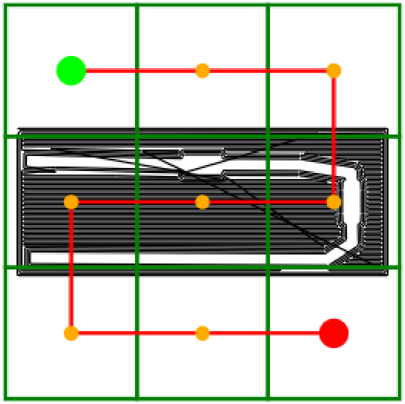


Figure: Intermediate of the camera grid algorithm.

Improving the result

To improve the grid we delete one entire row and column and see if that brought an improvement while still covering the entire GCode.

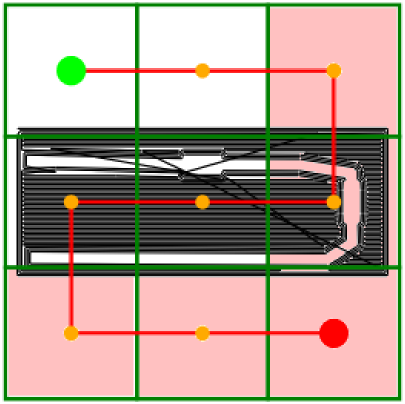


Figure: Red areas will be only deleted if they improve the grid.

The optimal grid

After applying our optimization we get the final grid layout that is used to control the camera later on.

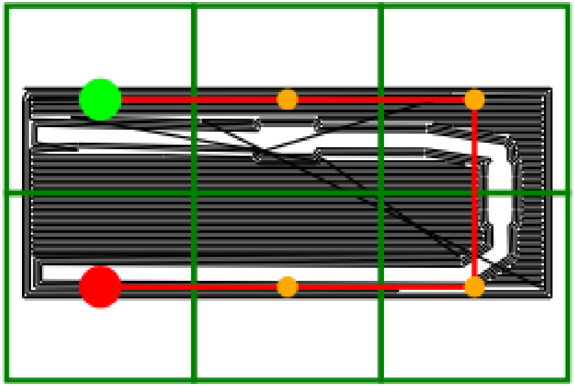
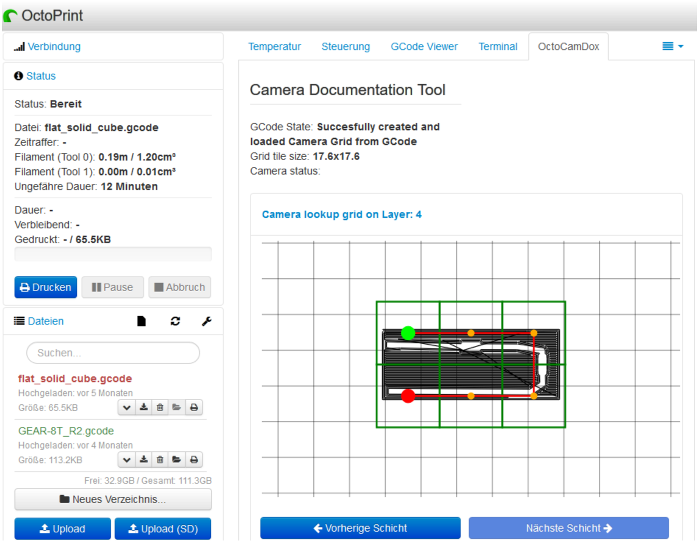


Figure: Final result and optimal grid.

Result implementation: OctoCamDox


OctoPrint

Verbindung

Status

Status: **Bereit**

Datei: **flat_solid_cube.gcode**
Zeitraffer: **-**
Filament (Tool 0): **0.19m / 1.20cm³**
Filament (Tool 1): **0.00m / 0.01cm³**
Ungefähre Dauer: **12 Minuten**

Dauer: **-**
Verbleibend: **-**
Gedruckt: **- / 65.5KB**

Dateien

flat_solid_cube.gcode
Hochgeladen: vor 5 Monaten
Größe: 65.5KB

GEAR-8T_R2.gcode
Hochgeladen: vor 4 Monaten
Größe: 113.2KB

Frei: 32.9GB / Gesamt: 111.3GB

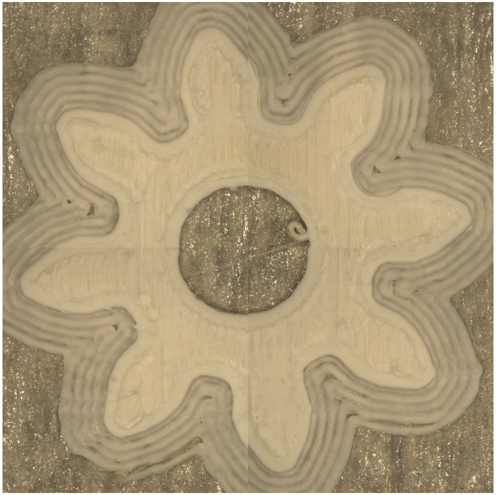
Camera Documentation Tool

GCode State: **Succesfully created and loaded Camera Grid from GCode**
Grid tile size: **17.6x17.6**
Camera status:

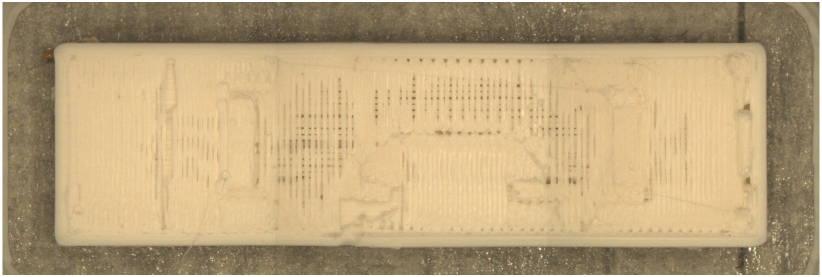
Camera lookup grid on Layer: 4

Picture examples

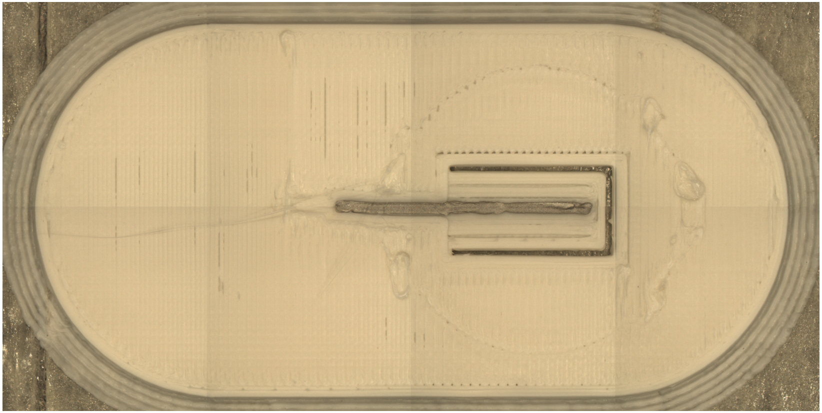
OctoCamDox in action I



OctoCamDox in action II



OctoCamDox in action III



Picture analysis

Solution Idea

We'll use the result image from the previous algorithm and apply the following steps on it:

1. Extract the circuit coordinates from the GCode.

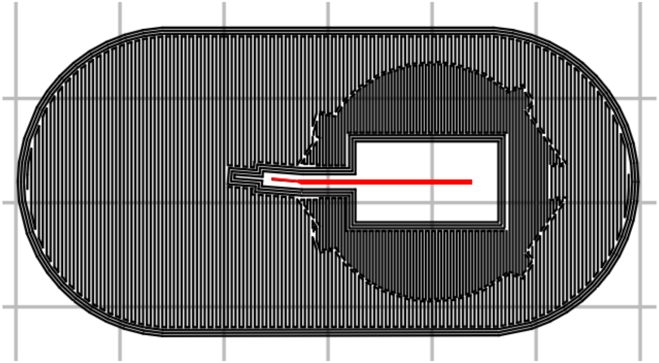


Figure: Circuit information from GCode.

Binarization

2 Binarize the captured layer Image (Otsu's method).

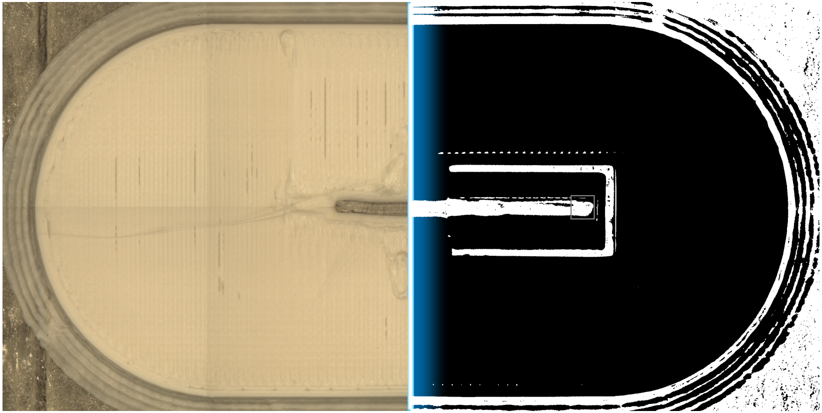


Figure: Binarized result Image.

Create the mask

3. Create a white circle of radius *r* with *r* being the circuit width, extracted from the GCode file as well.

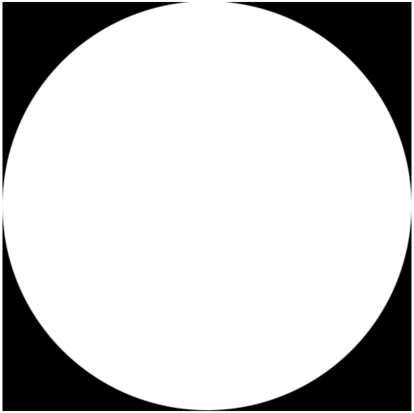


Figure: The mask for our region of interest (ROI).

Cropping the image

4 Crop the image around the area we're interested in.

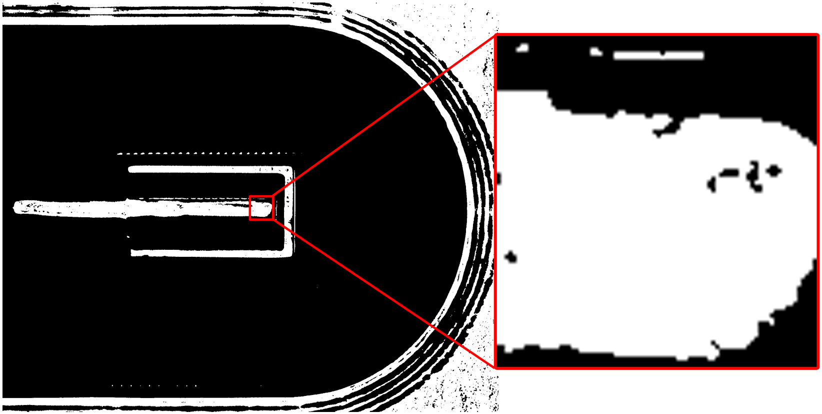


Figure: Cropped image area.

Masking the image

5 Crop the image and mask the unwanted pixels via bit-wise AND to only look at the pixels that matter.

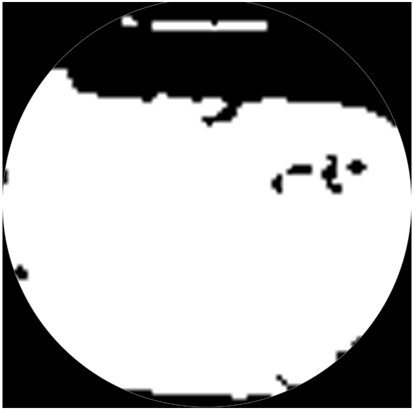


Figure: Applied mask on cropped area.

Pixel counting

6 Inside the detector, compute the number of white pixels versus the total amount of white pixels from the original white circle.

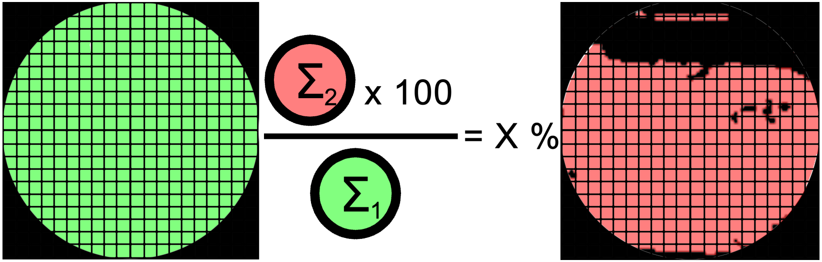


Figure: Found white pixels versus total amount of white pixels.

Quality checking

7. If the amount of found white pixels was below a certain threshold, save the coordinate as a start point. The first coordinate beeing above the threshold again, marks the endpoint of the area that needs to be repaired.

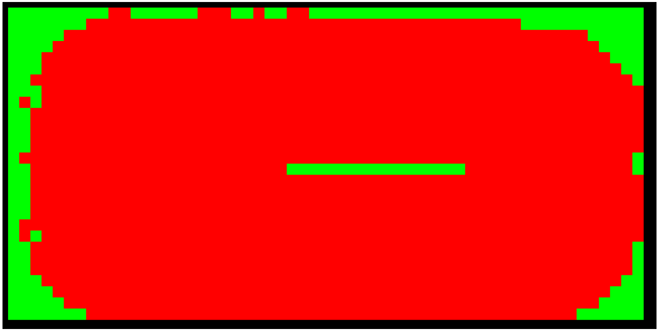


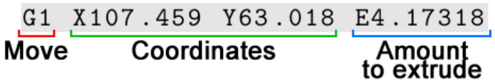
Figure: Sample application of our detector over the entire image.

Repair broken circuits

8 After we've found any possiby broken areas, we can attempt to repair them by simply sending additional GCode commands to the printer, before continuing the print job again.

Listing 2: Sample GCode commands

```
G1 X(Start) Y(Start) # Move without extruding
G1 X(Target) Y(Target) E0.19897 # Plant material
```



```
G1 X107.459 Y63.018 E4.17318
```

Conclusion

State of affairs

Let’s have a look again at our goal list from the start of the presentation, to see what has been implemented so far:

1. Automatic consecutive image capture on finished layers. ✓
2. Composition of the captured images into a big picture. ✓
3. Using the composed pictures to detect faulty conduits. ✓ (Partially implemented)
4. Automatic correction of faulty conduits. ✗

Thank you for your patience!

Literature

Literature

Bernd Jähne.
Digitale Bildverarbeitung.
2017.

Reinhard Klette.
Concise Computer Vision.
2017.

Milan Sonka, Vaclav Hlavac, and Roger Boyle.
Image Processing, Analysis and Machine Vision.
2017.