Smart 3D Printer for Detecting Printing Defects by using Image Processing Deep Learning and Robot Manipulator

DEPARTMENT OF MECHATRONICS ENGINEERING

Cansu TUNÇ
Mustafa OVALI
Onur SEVER

Supervisor: **Dr. Fatih Cemal Can**

İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ 2010

Abstract

In this study, our purpose is, real-time image processing by using an artificial neural network algorithms which are wroten by us. With the camera we attach to the 5Dof robot manipulator we have designed, we aim to monitor our 3-D printer during the printing and to detect errors with the algorithm that we have wrote by performing real-time image processing.

Introduction

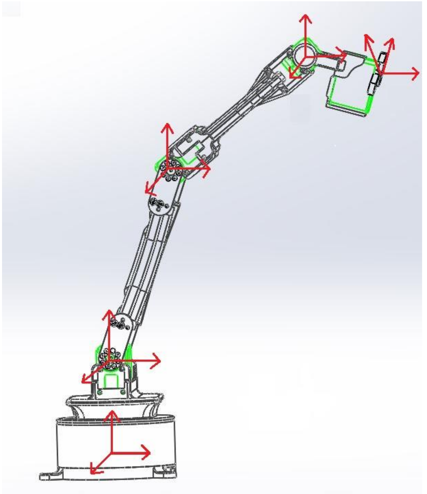
Today, 3D printing technology is becoming a technology that continues to be widely used, it is used in many areas in the health sector, aviation and industry. It is possible to get both cheap and practical prints. However, 3-D printers can also produce faulty parts from time to time, and there is no common system to track these parts. Our aim in this project is to perform real-time image processing with the camera we attach to the 5Dof robot manipulator we designed, and to monitor our 3D printer during printing and detect defects with the artificial intelligence algorithms we wrote. In this study, we bring together today's most critical fields such as robotics, software (artificial intelligence-deep learning), 3-D printer.

At the beginning of this project, we first made kinematic analyzes of the Robot manipulator we designed using the Denavit Hartenberg table method and calculated the trejacty for smooth movement and observed our results on solidwork. After finishing the robot manipulator calculations and montage, we started to set up the Algorithm.

First observation was defect detection on printed objects by using Image Processing, OpenCV and Python. We detected defects with high accuracy by using intersection over union, contour differences, and area differences. For the second approach we used Convolutional Neural Networks, Yolov5 for real time defect detection during printing objects from 3d printer. Observed results and graphical represenations by using Tensorflow.

Then we were able to compare results of the image processing and results of the deep learnign algorithms.

Robot Manipulator Kinematics Analysis



Denavit Hartenberg table

i	a	α	S_i	θ	
1	0	0	0	θ_1	${}^{0}_{1}T$
2	0	$\pi/2$	S_2	θ_2	${}^{1}_{2}T$
3	a_3	0	0	θ_3	${}^{2}_{3}T$
4	a_4	0	0	θ_4	${}^{3}_{4}T$
5	a_5	$-\pi/2$	0	θ_5	${}^{4}_{5}T$

$$ {}^{0}_{1}T = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{1}_{2}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_2 & -s_2 & 0 & 0 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{2}_{3}T = \begin{bmatrix} 1 & 0 & 0 & a_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_3 & -s_3 & 0 & 0 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{3}_{4}T = \begin{bmatrix} 1 & 0 & 0 & a_4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{4}_{5}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{0}_{1}T \; x \; {}^{1}_{2}T \; x \; {}^{2}_{3}T \; x \; {}^{3}_{4}T \; x \; {}^{4}_{5}T = {}^{0}_{5}T $$

Design Of Robot Manipulator

$$\theta_1 = a_1 t^3 + b_1 t^2 + c_1 t + d_1$$
$$\theta_2 = a_2 t^3 + b_2 t^2 + c_2 t + d_2$$
$$\theta_3 = a_3 t^3 + b_3 t^2 + c_3 t + d_3$$

X

I

X $t = t_{m1}$

II

$\theta_1^{I}(t_{m1}) = \theta_1^{II}(t_{m1})$

$\theta_1^{I}(t_{m1}) = \theta_{1m1}(t_{m1})$

$\dot{\theta}_1^{I}(t_{m1}) = \dot{\theta}_1^{II}(t_{m1})$

$\ddot{\theta}_1^{I}(t_{m1}) = \ddot{\theta}_1^{II}(t_{m1})$

X $t = t_{m2}$

III

X

$\theta_1^{II}(t_{m2}) = \theta_1^{III}(t_{m2})$

$\theta_1^{II}(t_{m2}) = \theta_{1m2}(t_{m2})$

$\dot{\theta}_1^{II}(t_{m2}) = \dot{\theta}_1^{III}(t_{m2})$

$\ddot{\theta}_1^{II}(t_{m2}) = \ddot{\theta}_1^{III}(t_{m2})$

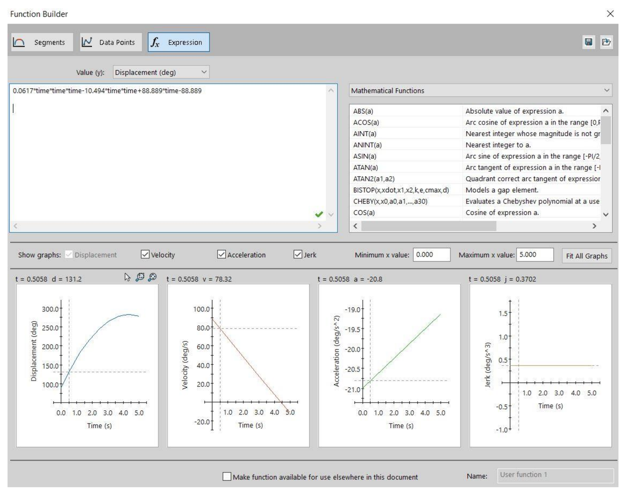
$$d_1 = \theta_{1i}$$
$$c_1 = 0$$
$$a_1 t_{m1}^3 + b_1 t_{m1}^2 + c_1 t_{m1} + d_1 = a_2 t_{m1}^3 + b_2 t_{m1}^2 + c_2 t_{m1} + d_2$$
$$a_1 t_{m1}^3 + b_1 t_{m1}^2 + c_1 t_{m1} + d_1 = \theta_{1m1}$$
$$3a_1 t_{m1}^2 + 2b_1 t_{m1} + c_1 = 3a_2 t_{m1}^2 + 2b_2 t_{m1} + c_2$$
$$6a_1 t_{m1} + 2b_1 = 6a_2 t_{m1} + 2b_2$$
$$a_2 t_{m2}^3 + b_2 t_{m2}^2 + c_2 t_{m2} + d_2 = a_3 t_{m2}^3 + b_3 t_{m2}^2 + c_3 t_{m2} + d_3$$
$$a_2 t_{m2}^3 + b_2 t_{m2}^2 + c_2 t_{m2} + d_2 = \theta_{1m2}$$
$$3a_2 t_{m2}^2 + 2b_2 t_{m2} + c_2 = 3a_3 t_{m2}^2 + 2b_3 t_{m2} + c_3$$
$$6a_2 t_{m2} + 2b_2 = 6a_3 t_{m2} + 2b_3$$
$$a_3 t_f^3 + b_3 t_f^2 + c_3 t_f + d_3 = \theta_{1f}$$
$$d_3$$

Purpose of the Study

Our aim in this project is to perform real-time image processing with the camera we attach to the 5 Dof robot manipulator we designed, and to monitor our 3D printing and detect defects with the image processing and deep learning algorithms that we have developed.

Experimental Setup

Robot Manipulator



Deep Learning

TRAINING DATA
%85 of the photos for training, %8 of the photos for validation, %7 of the photos for testing
PREPROCESSING
Auto-Orient: Applied
Resize: Stretch to 416x416
AUGMENTATIONS
Outputs per training example: 3
Flip: Horizontal
90° Rotate: Clockwise, Counter-Clockwise
Rotation: Between -45° and +45°
Grayscale: Apply to 10% of images
Annotation Group: edgewarping-stringing-spaghetti

To train our detector we take the following steps:

- Install YOLOv5 dependencies
- Download Custom YOLOv5 Defect Detection Data
- Define YOLOv5 Model Configuration and Architecture
- Train a custom YOLOv5 Detector
- Evaluate YOLOv5 performance
- Visualize YOLOv5 training data
- Run YOLOv5 Inference on test images
- Export Saved YOLOv5 Weights for Future Inference

We collected data photos from printed objects.To start off with YOLO v5 we first clone the YOLO v5 repository and install dependencies. This set up our programming environment to be ready to running defect detection training and inference commands. The GPU will allow us to accelerate training time. Colab comes preinstalled with torch and cuda. We downloaded custom defect detection data in YOLOv5 format from Roboflow. Once we have labeled data, to moved our data into Roboflow. We choose different preprocessing and augmentation steps. The export creates a YOLO v5 .yaml file called data.yaml specifying the location of a YOLO v5 images folder, a YOLO v5 labels folder, and information on our custom classes. Next, we write a model configuration file for our custom detector. With our data.yaml and custom_yolov5s.yaml files ready, we started with training. During training, the YOLOv5 training pipeline creates batches of training data with augmentations. We can visualize the training data ground truth as well as the augmented training data. YOLO v5 is lightweight to use because it trains quickly, inferences fast, and performs well.
--img 416 --batch 16 --epochs 150
150 epochs completed in 0.135 hours.
Model summary: Model summary: 270 layers, 7027720 parameters, 7027720 gradients, 15.9 GFLOPs
Class Images Labels P R mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 6.24it/s]
!python detect.py --weights runs/train/exp/weights/best.pt --img 416 --conf 0.1 --source {dataset.location}/test/images
hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5,

Training results:
33.7% mAP, 73.3% precision, 33.0% recall

Results and Discussion

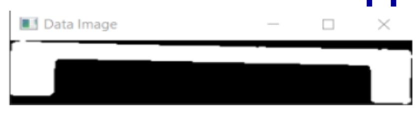
Results of Image Processing and OpenCV

Image Processing Algorithms

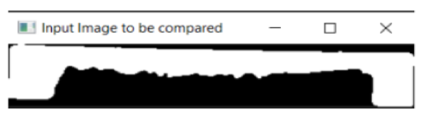
• Preprocessing • Contour detection • Contour matching
• Defect detection • Feature extraction

OUTPUTS OF PROGRAM

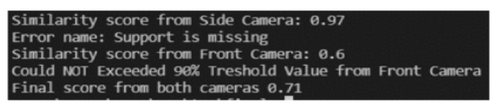
Support is Missing



Correct images output to unsupported failure from front camera

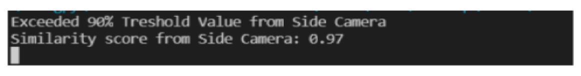


Failure images output to unsupported failure from front camera



Output result front camera

Error detect to unsupported failure

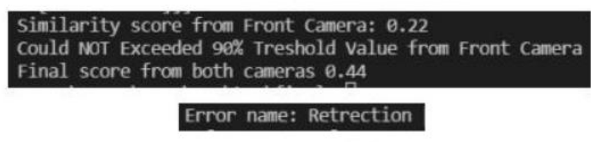


Output result side camera

Rectraction

Correct images output to retraction failure from front camera

Output result side camera

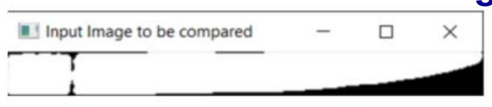


Output result front camera

Failure image output to retraction failure from front camera

Error detect to retraction failure

Edge Warping



Failure images output to retraction failure from front camera

Correct images output to retraction failure from front camera

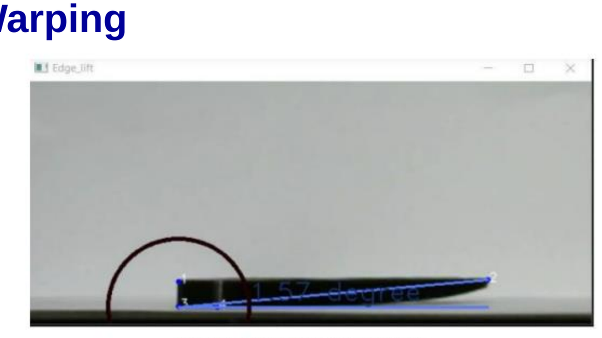


Figure IV -41) Error detect to edge lift failure

Output result front camera

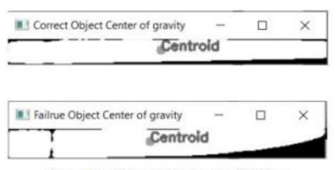
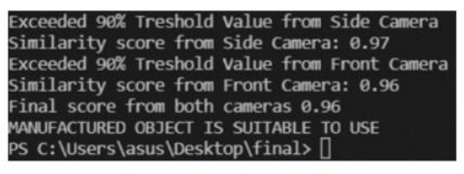


Figure IV -42) Error detect to edge lift failure



Output result side camera

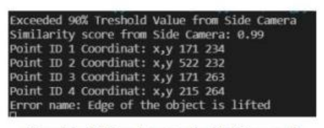
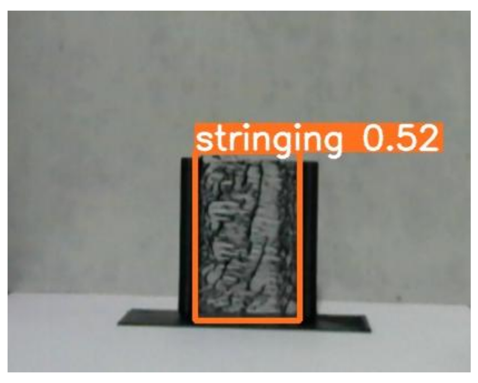
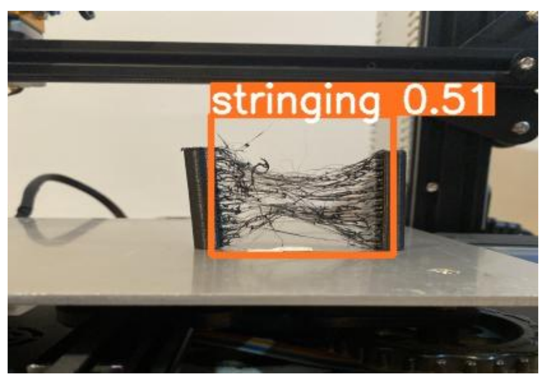
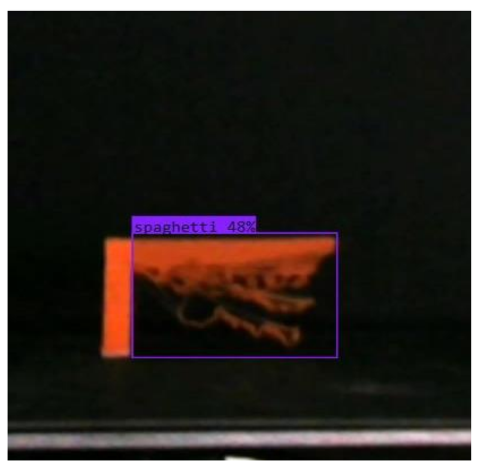


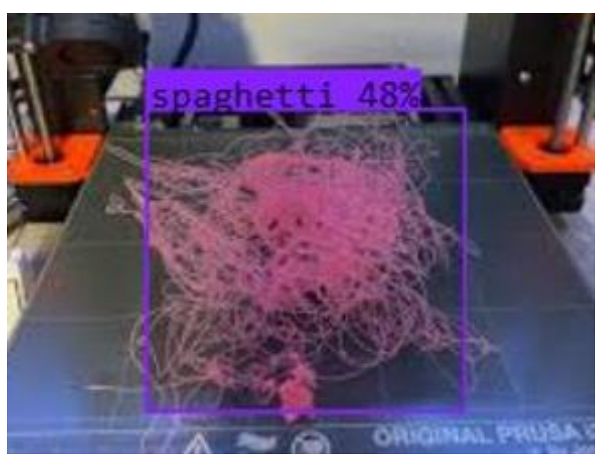
Figure IV -43) Error detect to edge lift failure result

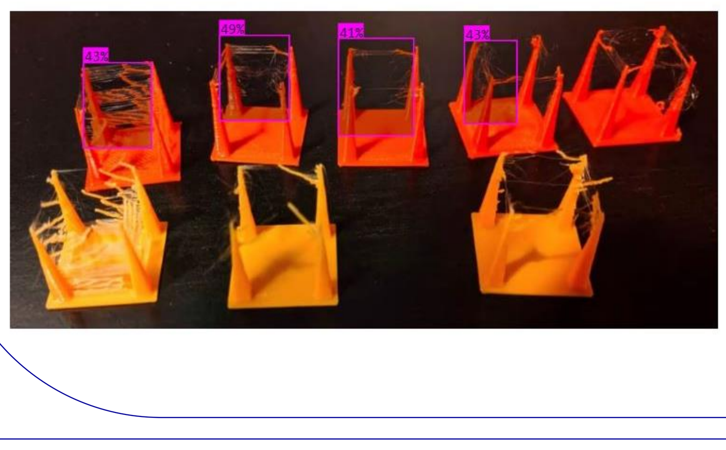
Results of Convolutional Neural Networks and YOLOv5

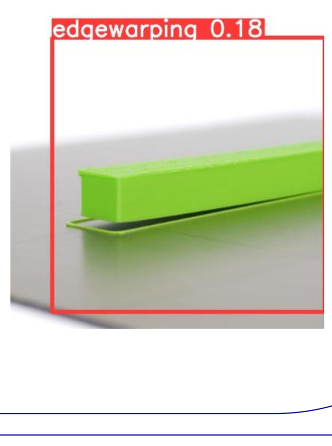












References

https://drive.google.com/drive/u/0/folders/1znV034cRwYnKdD1CSx-tw5pNIVaZnvIV
https://drive.google.com/drive/folders/1ocfsyLbK9hZSZ_zu5OQy1_6l-vVmwg9D