Automated Counting and Inspection System

Yasmein Asfour

Professor Nikolaos Papakostas

University College Dublin
Ireland's Global University

The George Washington University, Washington DC

INTRODUCTION

This project centers on automated counting and inspection tasks in manufacturing using the Intel RealSense D455 camera, a robotic arm, and MATLAB. Provided with a pallet of drills from the ATA Drills group, the objective is twofold: firstly, to develop a method for accurately counting the drills, and secondly, to identify any defects present.

OBJECTIVES

- Develop a method for accurately counting the drills on the pallet using the Intel RealSense D455 camera attached to a robotic arm, with the images being processed on MATLAB.
- Implement image analysis techniques to identify defects or irregularities in the drills.
- Assess the effectiveness and reliability of the developed counting method and defect detection process.
- Provide comprehensive documentation of the project methodology and findings to serve as a guide for future research in automated inspection systems.

BACKGROUND

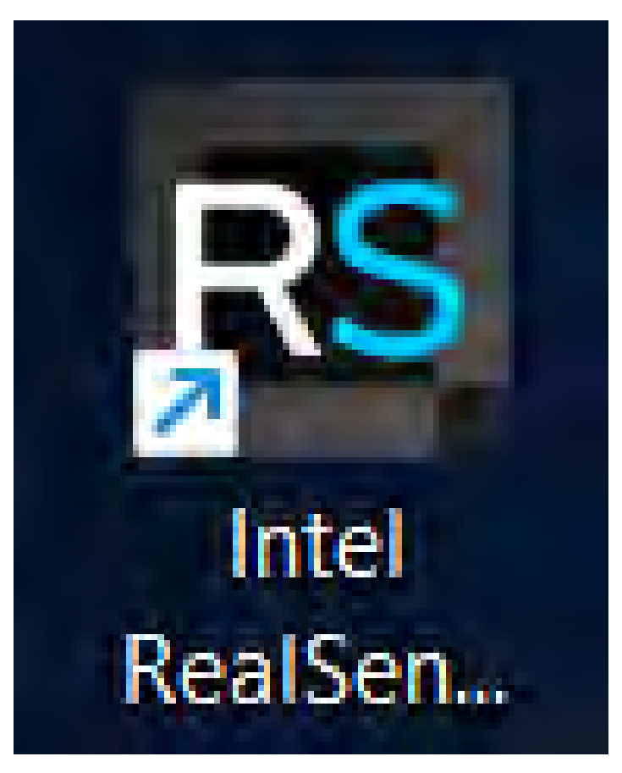
In preparation for this project, I self-studied essential coursework to equip myself with the necessary knowledge and skills. I completed the MATLAB Image Processing course, which provided a solid foundation in image analysis techniques crucial for the endeavor. Additionally, I looked into two books to augment my understanding, including Richard Szeliski's "Computer Vision: Algorithms and Applications" and Peter Corke's "Robotics, Vision, and Control: Fundamental Algorithms in MATLAB®." These resources offered valuable insights into the practical applications of computer vision and robotics, thereby enhancing my proficiency in the subject matter. Armed with this background knowledge, I attained a solid grasp of the principles and methodologies essential for the success of this project.

METHADOLOGY

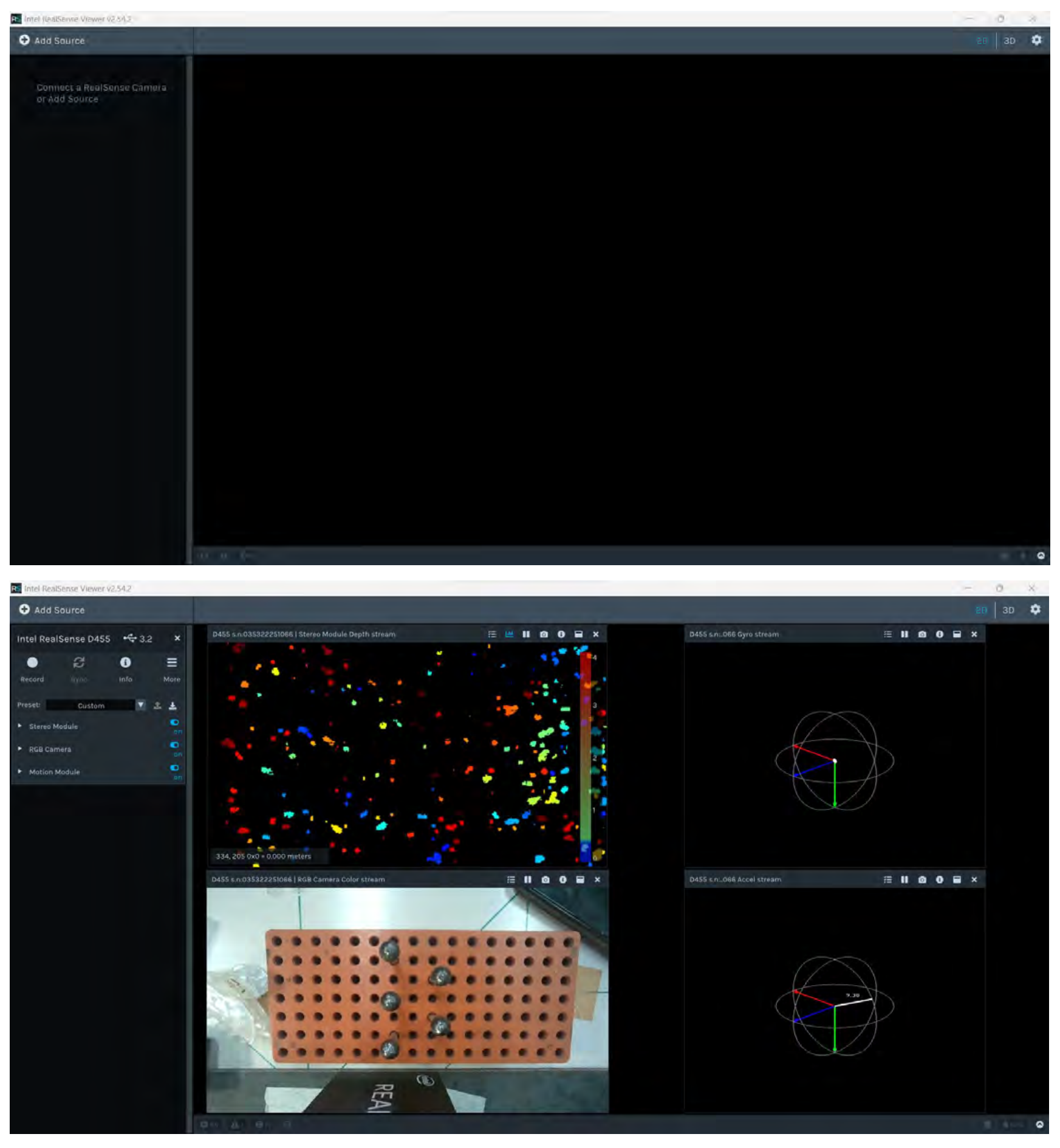
First, to select the appropriate camera, a comparative analysis was conducted between the Azure Kinect and Intel RealSense cameras. Factors considered include:

- Depth Sensing Technology: Azure Kinect uses time-of-flight (ToF) technology, while Intel RealSense employs active stereo depth technology.
- Field of View: Azure Kinect typically offers a wider field of view, but not essential for this project.
- Integration with MATLAB: Both cameras are supported by MATLAB, offering functions for accessing color, depth, and infrared streams.
- Extra Features: Azure Kinect includes an integrated microphone array and seamless integration with Microsoft Azure services, while Intel RealSense is known for its compact design and open-source SDK.
- Cost and Availability: Both cameras were accessible.

After analysis, the Intel RealSense camera was chosen for compatibility with MATLAB and its compact design.

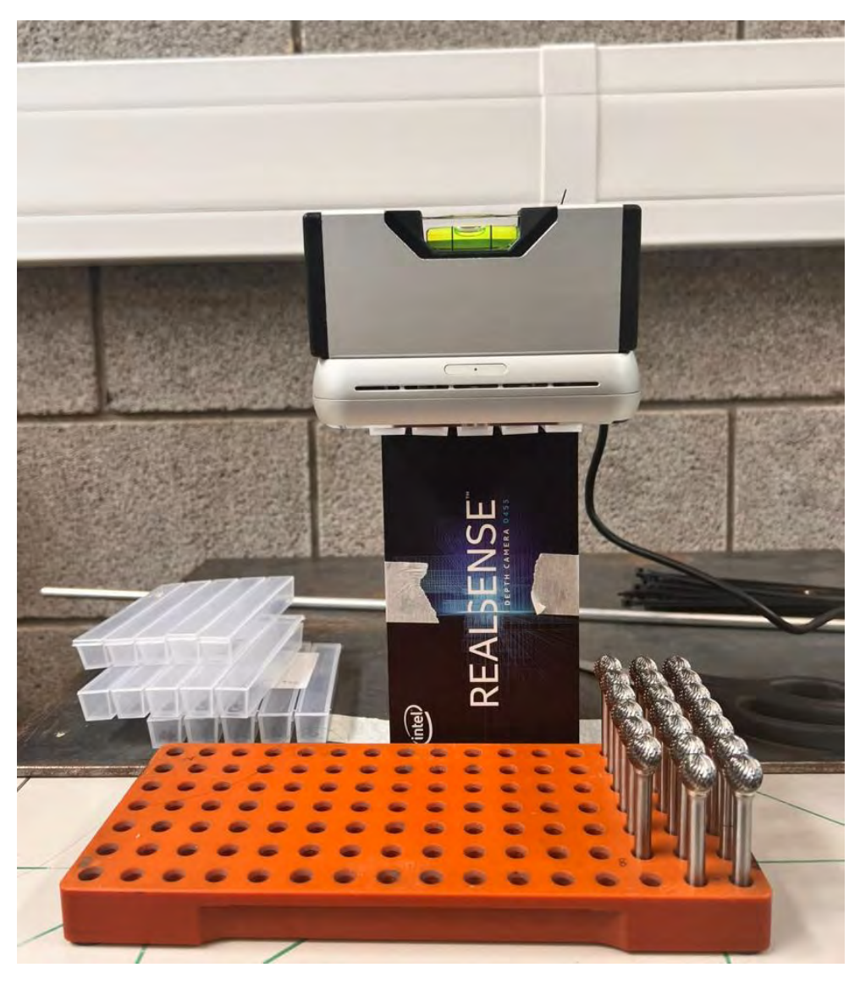


To connect the camera to the laptop, I downloaded the Intel RealSense Viewer by visiting the Intel RealSense website at https://www.intelrealsense.com/sdk-2/. Upon arriving at the site, I clicked on the "Download" button located near the upper left corner and scrolled down to find the assets section where I could download the SDK. It appeared, as seen to the right, on my Desktop.

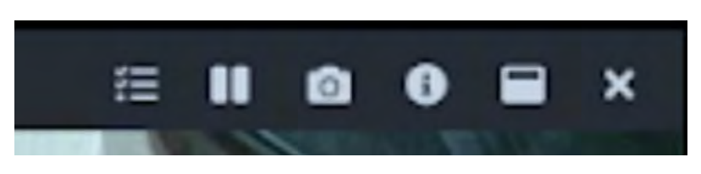


The image to the right is what the viewer looks like when you first start it. Upon unboxing the camera, I connected the provided USB cable to the computer and then to the camera. The viewer immediately detected the camera, presenting three options to activate: the Stereo Module, RGB Camera, and Motion Module.

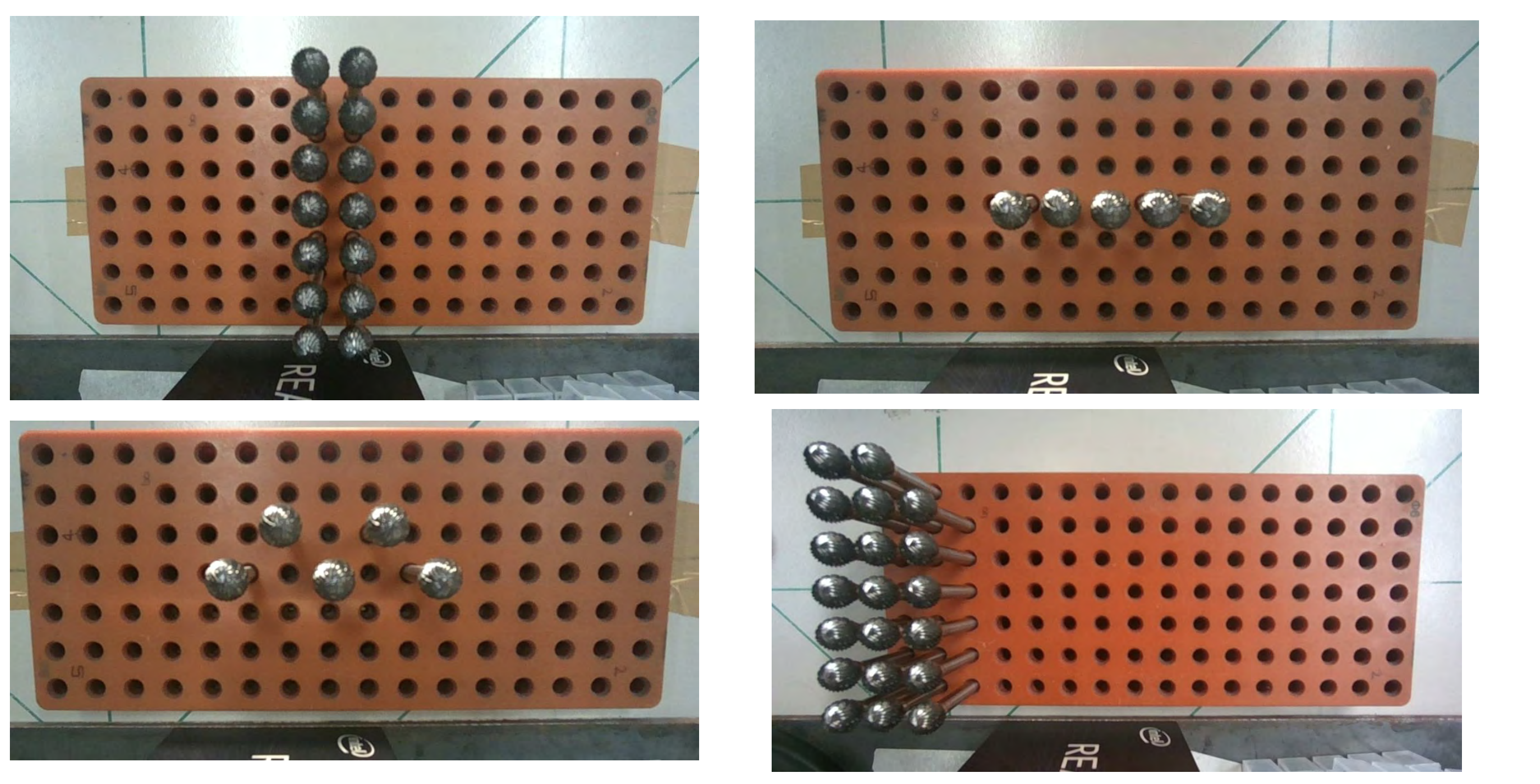
The next task was to mount the camera, as the robotic arm was to be implemented in a later stage. Since the lab lacked equipment for an overhead view, I improvised by using the camera box, its stand, tape, and zip ties to create a makeshift stand, as shown in the photo. To ensure the camera was properly aligned, I used a level.



To prevent the pallet from moving, I first aligned it with the table and then securely taped it in place to ensure stability.

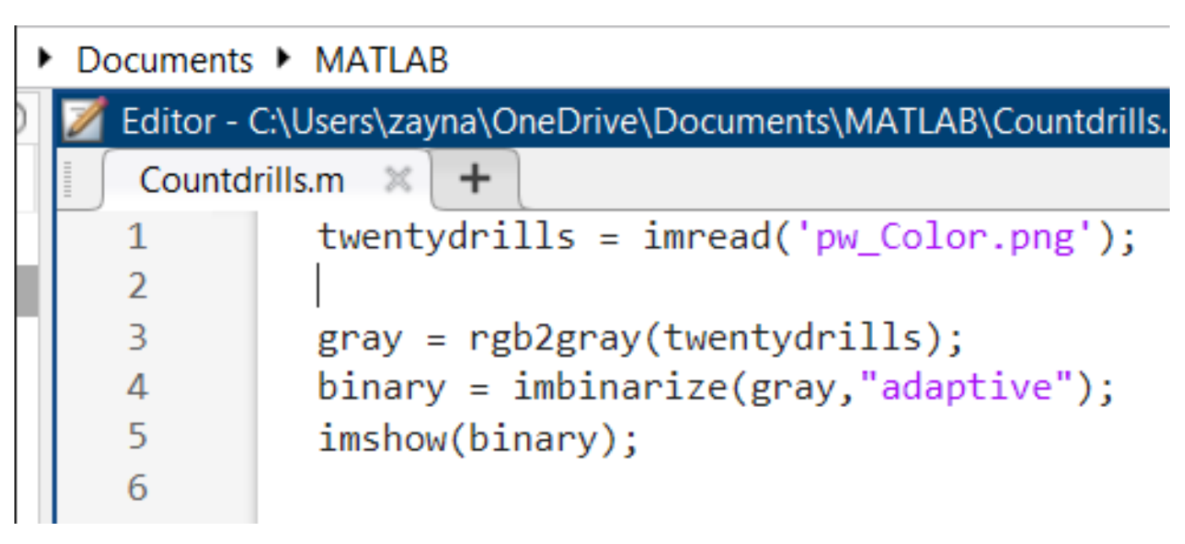


For an initial trial I used the D61212 drills and placed them in different areas on the pallet and captured photos using the camera button on the upper right corner as seen to the left:



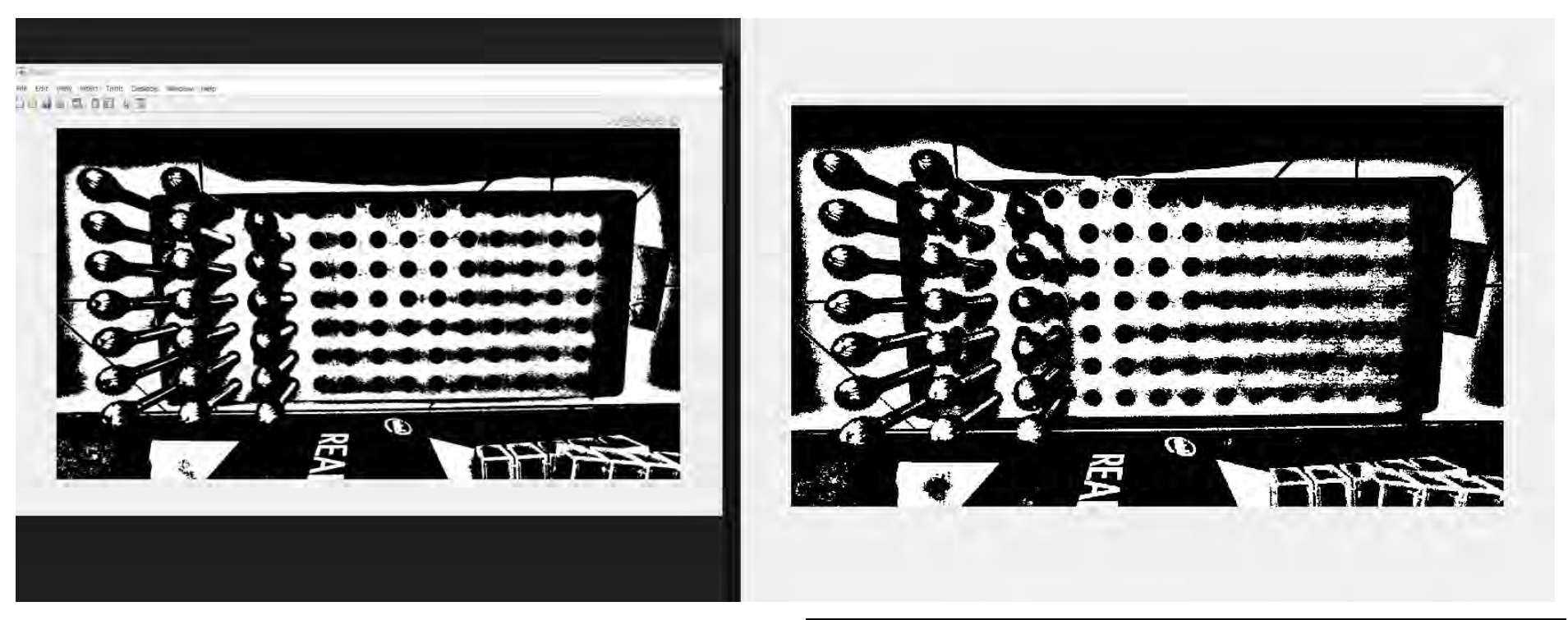
To start the image analysis, I downloaded the MATLAB program from their official website. I used the image in the lower right corner for analysis. I opened a new script and began the process. Initially, when I binarized the image, it appeared too dark. To address this, I brought my dormitory lamp and set it up accordingly.

Above is the picture before adding the light. Below is a comparison of the different lighting options. After selecting an image, I binarized it and applied the adaptive command to enhance clarity. The code can be seen to the left.

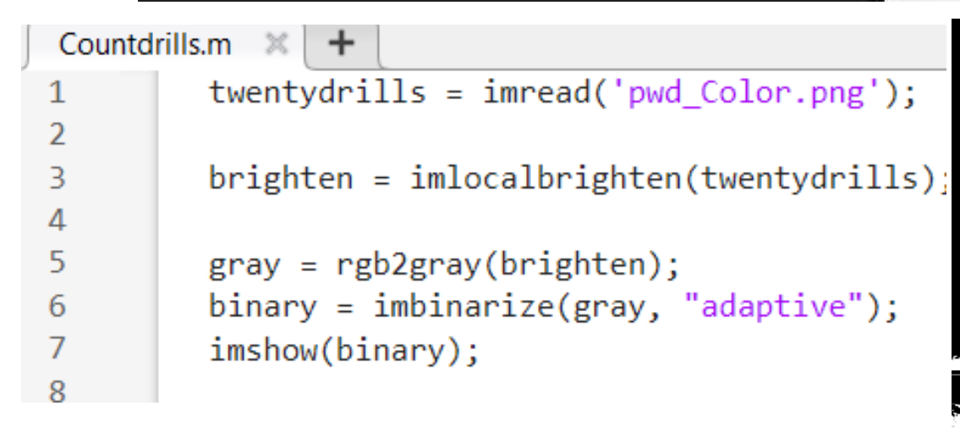


Documents ▸ MATLAB
Editor - C:\Users\zayna\OneDrive\Documents\MATLAB\Countdrills.
Countdrills.m

```matlab
twentydrills = imread('pw_Color.png');

gray = rgb2gray(twentydrills);
binary = imbinarize(gray,"adaptive");
imshow(binary);
```

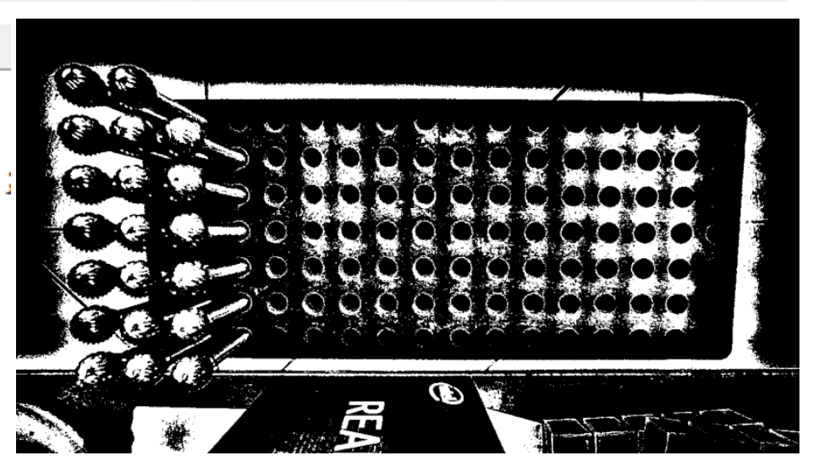


Seeing as the neither the empty circles nor the drills are very distinct, I tried brightening the image with this code:

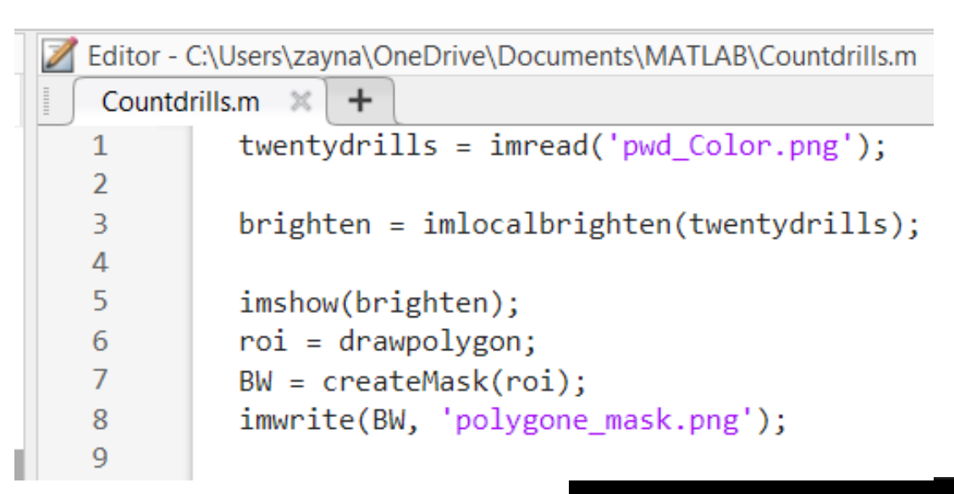


Countdrills.m

```matlab
twentydrills = imread('pwd_Color.png');

brighten = imlocalbrighten(twentydrills);

gray = rgb2gray(brighten);
binary = imbinarize(gray, "adaptive");
imshow(binary);
```

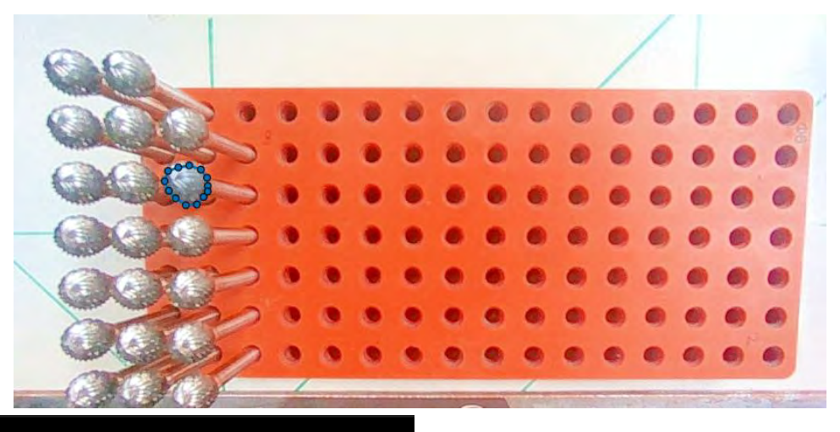


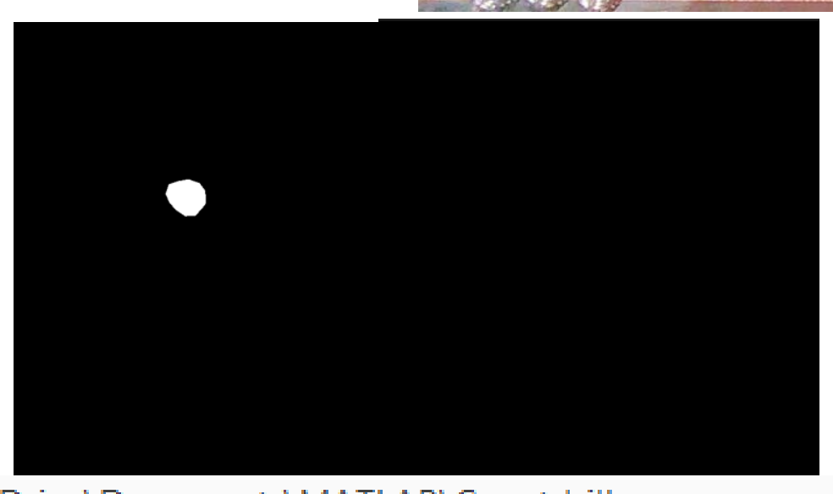
The drills were somewhat more distinctive after the lighting adjustments; however, counting them still posed a challenge. Therefore, I opted to use a mask and drew a polygon around the drill tip to create a mask.



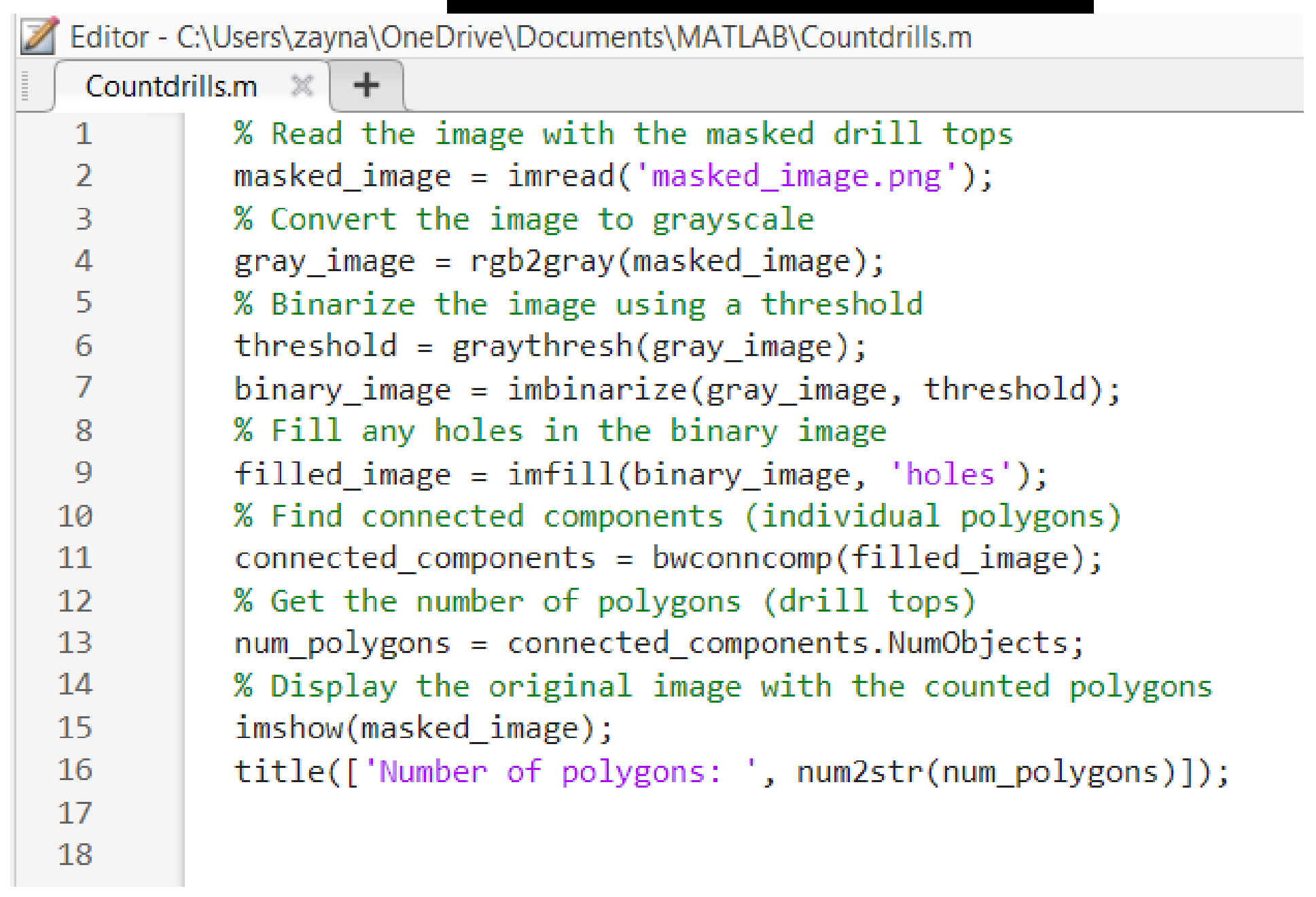
Editor - C:\Users\zayna\OneDrive\Documents\MATLAB\Countdrills.m
Countdrills.m

```matlab
twentydrills = imread('pwd_Color.png');

brighten = imlocalbrighten(twentydrills);

imshow(brighten);
roi = drawpolygon;
BW = createMask(roi);
imwrite(BW, 'polygone_mask.png');
```





The next step would be to burn the mask such that all the drill tops would appear as circles, then write a program to count those circles. To the right is a sample code of how I would approach the problem



Editor - C:\Users\zayna\OneDrive\Documents\MATLAB\Countdrills.m
Countdrills.m

```matlab
% Read the image with the masked drill tops
masked_image = imread('masked_image.png');
% Convert the image to grayscale
gray_image = rgb2gray(masked_image);
% Binarize the image using a threshold
threshold = graythresh(gray_image);
binary_image = imbinarize(gray_image, threshold);
% Fill any holes in the binary image
filled_image = imfill(binary_image, 'holes');
% Find connected components (individual polygons)
connected_components = bwconncomp(filled_image);
% Get the number of polygons (drill tops)
num_polygons = connected_components.NumObjects;
% Display the original image with the counted polygons
imshow(masked_image);
title(['Number of polygons: ', num2str(num_polygons)]);
```

REFERENCES

matlabacademy.mathworks.com. (n.d.). Image Processing with MATLAB | Self-Paced Online Courses - MATLAB & Simulink. [online] Available at: https://matlabacademy.mathworks.com/details/image-processing-with-matlab/mlip.

Szeliski, R. (2022). Computer vision : algorithms and applications. Cham: Springer.

Corke, P.I. (2013). Robotics, vision and control : fundamental algorithms in MATLAB. Berlin: Springer.

CONTACT INFORMATION: yasfour04@gwu.edu, yasmein.asfour@ucdconnect.ie