3D Occluded Object Detection Vision System

Evan Kolin, Hannah Hillhouse, Samiha Elahi, Tony Jeong

FINAL PROJECT REPORT

REVISION – Final
3 December 2021

Final Project Report
for
3D Occluded Vision Detection System

Team 5

Approved by:

_____
Evan Kolin 12/05/2021

_____
Kevin Nowka 12/05/2021

_____
Souryendu Das 12/05/2021

Table of Contents

INTERFACE CONTROL DOCUMENT

SUBSYSTEM REPORTS

Table of Figures

Table of Tables

3D Occluded Object Detection Vision System

Evan Kolin, Hannah Hillhouse, Samiha Elahi, Tony Jeong

CONCEPT OF OPERATIONS

REVISION – Final
3 December 2021

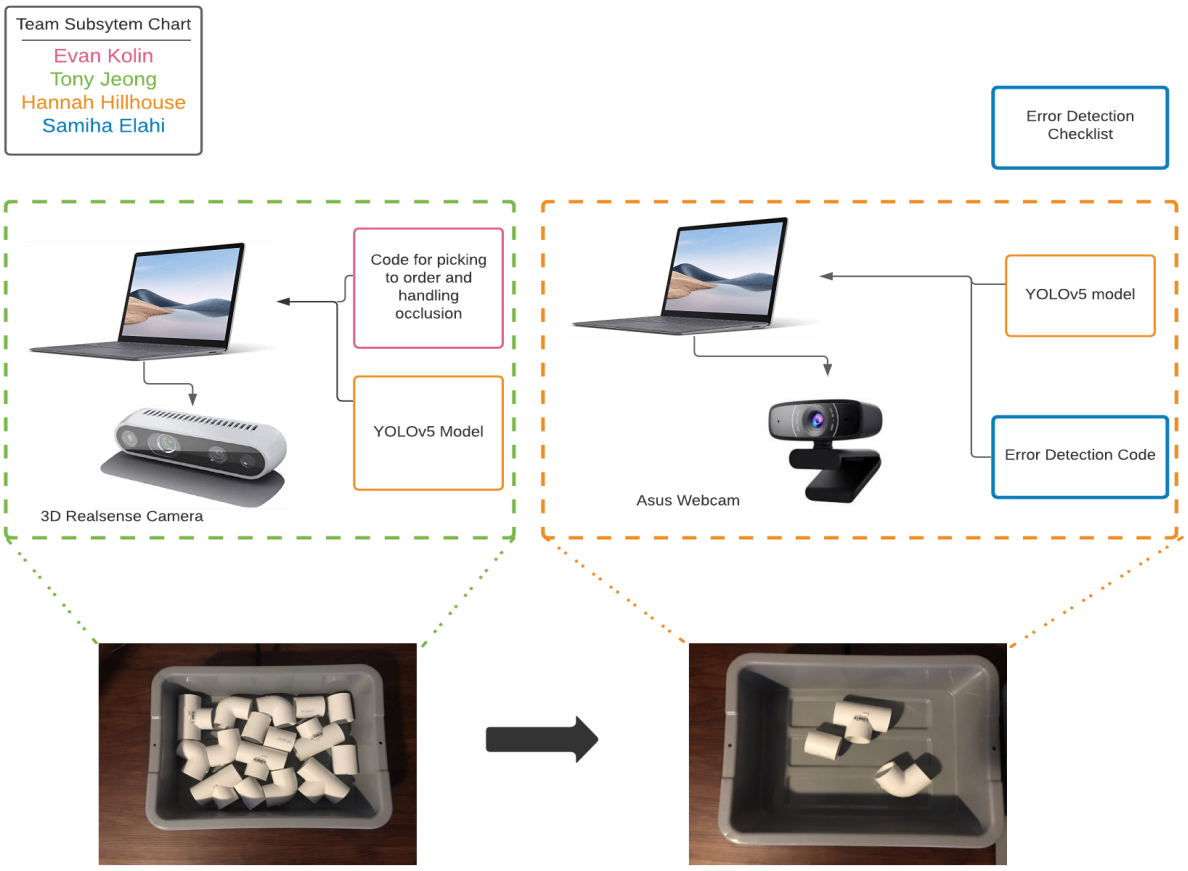
Change Record

Rev.	Date	Originator	Approvals	Description
0	2/7/2021	Evan Kolin		First Pass
1	4/28/2021	Evan Kolin		Final Submission ECEN 403
2	12/3/2021	Evan Kolin		Final Submission ECEN 404

1. Executive Summary

Our project is to create a 3D vision system that is capable of classifying and localizing multiple overlapping items. This system could be used in factories to increase overall productivity and bring new jobs in the tech field to maintain the vision system. The system will utilize machine learning to classify objects used in manufacturing such as different shaped pvc pipes. There will be two cameras that relay information to the AI, allowing for it to work in real time and adapt to the location of required parts. Additionally, both cameras will localize the objects in the bin, providing the coordinates of the piece to the robotic arm. Note that our system does not use the robot arm, it simply provides location and classification to the arm. Finally, once the pieces have been selected, there is a final check that ensures that the pieces selected are in fact the right ones.

Fig 1. System Diagram of 3D Object Vision Detection System



2. Introduction

The purpose of this document is to present the 3D object detection system, this is a system which will help commercial industries regulate and organize machine parts. This project will have the potential of improving commercial performance and help enable companies to pick and place objects in a reliable and timely manner.

2.1 Background

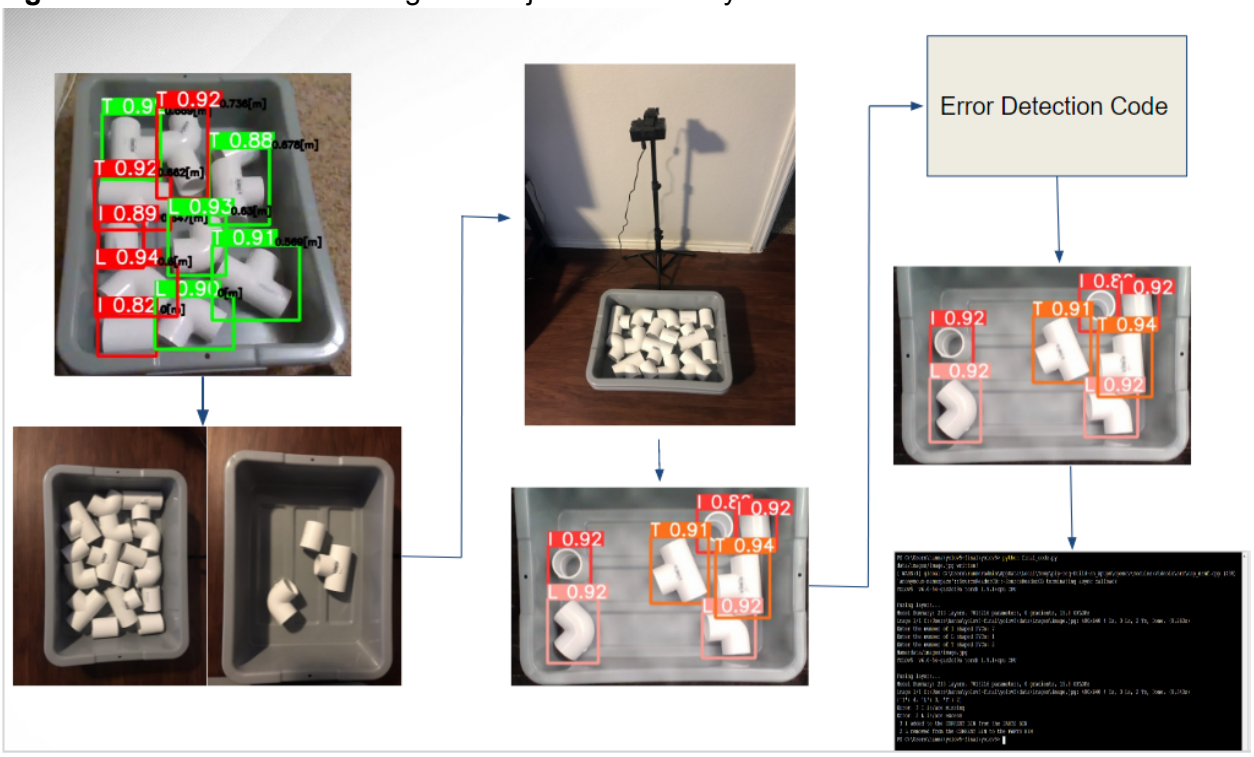
Warehouse facilities and management are important for efficient logistics and effective response to customers' demand. To keep up with such a high demand for products warehouses have to store products in an orderly manner such that the object could be located and shipped effectively. The bin picker visual system has the benefit of being able to capture an image of the objects in the bin, locate and label the parts in the bin so that the robotic arm knows where the part is, what it is, and what bin it needs to go in. Generally order picking, the process of retrieving products from the warehouse in response to a consumer demand is one of the most important operational activities in a warehouse. Companies are always looking for ways to cut down costs and increase efficiency throughout their business. Labor intensive operations with a manual system are costly especially when companies are trying to do more with less [1].

Commercial infrastructures have had people hand sort out parts and place them in their corresponding bins for many years now. However, there is a more efficient way of sorting which is an upcoming technology. This technology allows for humans to be able to have comfort in sorting without the drag of bending over and placing the parts in the bins themselves [2]. Manufacturers have many bins for parts which need to be sorted of many varying shapes and sizes. The robotic bin picker will require less people to operate in sorting of parts and allow for a faster pace in the workplace. People are effectively able to pick parts out of a bin and place them in a corresponding bin however there is no way to maintain complete accuracy at a time efficient manner. The robotic bin picker will effectively sort parts and place them in an orderly fashion at a faster pace than any human could keep up with. It has been shown that picking efficiencies have increased by 15% when using a robotic bin picker rather than hand picking[2].

2.2 Overview

The robotic vision system will consist of 2 cameras which will have a microcontroller attached to store data. The cameras will help with location and labeling of objects within the bin and this information will be passed on to the trained AI machine which will allow for the robotic arm to effectively pick the object out of the bin and place it into the correct bin. This will allow for warehouses to effectively and efficiently store parts all while being cost effective and fast paced. There is a constant demand for products so being able to keep up/ahead of the demand will allow for further success in the company. The flow of data will be as shown in Figure 2 below.

Fig 2. Flow of information through the object detection system



2.3 Referenced Documents and Standards

[1] H. Diefenbach and C. Glock, "Ergonomic and economic optimization of layout and item assignment of aU-shaped order picking zone", *Computers & Industrial Engineering*, vol. 138, pp. 106094, 2019.

[2] Nookea, Wanwipa, and Assadej Vanichchinchai. “An Ergonomics-Based Storage Bin Allocation for Picking Efficiency Improvement.” *IEEE Xplore,* 2020, ieeexplore.ieee.org/document/9101951

[3]Scimeca, D. (Ed.). (2020, October 14). Advances in 3D object detection in vision-guidedrobotics,https://www.vision-systems.com/home/webinar/14183801/advances-in-3d-object-detection-in-visionguided-robotics

[4] Fritz AI, https://www.fritz.ai/object-detection/#top

3. Operating Concept

3.1 Scope

For the scope of the project, a 3D occluded object detection vision system is being designed that will make the method of bin-picking simpler. Bin-picking is the task of sorting through overlapping and occluded parts to fill an order. Our goal is to have a robot with a a camera attached to it pick-up known objects out of a box and sorts them out. The deliverables for the scope of this project are as follows:

- Machine Learning
- Classification/Localization
- 3D Image Processing
- Alarm/Control

Documentation for the design, construction, and programming of the units will be provided for all parts of the project.

3.2 Operational Description and Constraints

The 3D occluded object detection vision system is intended for use in factories or manufacturing industries, where they must sort thousands of parts daily to go to the next production step. The object detection system will save a lot of time as factory workers no longer need to hand-pick certain objects to send it to the next step. The system is also intended to operate in a production facility, where lighting is particularly good for the cameras to capture the objects properly. If the lighting is not good then it will negatively impact the picture quality, thus there is a high chance that the system might not be able to identify the object. As a result, production will come to a halt and the company might suffer a huge financial loss. Furthermore, the system is not designed to operate under water. There is a limit to the amount of objects that any given bin can hold, putting an upper limit on what the AI is capable of detecting. In our testing we never imaged more than 20 objects at a time, due to us not having any more parts to image.

3.3 System Description

The 3D occluded object detection vision system would be created by training the AI through machine learning to gather insights from data and automate tasks. Then image detection would be used to identify the location of an object in an image and classification would be used to categorize the object based on the previously defined classes and draw a box around it. The 3D image processing would show the different confidence values for different classes, basically with the help of a 3D camera the system would be able to detect the height and depth of the different types of objects. Finally the Error Detection sub-system provides a check for the user to determine if the order and all the parts within the order were received and picked out by the robot, as well as what to do if the order was not completed correctly.

3.4 Modes of Operations

The 3D object detection system will be a mode of operation that collects images from the cameras and transmits it to an AI machine. The AI machine will be trained and will direct the robotic arm to pick up the identified object and place it into the corresponding bin.
The system will have an error detection mechanism in place to alert the user in the event of any noticeable problems during operation of the system as well as making sure that the parts are in the correct bin and the right amount of that part is in the correct space.

3.5 Users

The 3D object detection system is intended to be used by industries and other commercial companies to essentially replace people who manually sort out parts into bins. This system will be designed in such a way that workers are able to interpret the data given from the system and make sure that the correct parts are in the correct bins. This system will essentially pick and place objects in a way that the average human could do except it will do it constantly without break, making it a more efficient rate. The level of training required for installation and maintenance is that the user must have a basic understanding of the components in the system, microcontrollers, AI training, cameras, and programming.

3.6 Support

Support for 3D object detection programs will be provided via user manual. The user manual will include its usage, maintenance, hardware installation instructions, and full explanation of the software. Our project mainly focuses on the software which will be deeply explained in the manual such as its functions and structures.

4. Scenario(s)

4.1 Car Detection in traffic

This program could be source data for car detection in traffic. It will identify each car with its shape and color. This system can be held in the traffic police system. If any crime occurs, this system will help to find the car the suspect was in. It also can be used in car statistics. Identifying each car's shape and color means its model. Car companies are looking to see what models and how many of them are on the street. The last is autonomous driving. This system is crucial on cars with self-driving systems, such as *Tesla*. Autonomous driving requires prediction, planning, and motion control. 3D space with identifying cars around the ego car helps this requirement.

4.2 Factory Robotic Vision Advancement

Robotic vision advancements have had quite the impact on today's manufacturing and distribution environments. There are numerous challenges remaining in the workspace such as random bins in factories and distribution centers. Our system is basically the software for robotic arms used for bin picking based on 3D object detection. It can be directly used for the source program and system for the factory robotic vision advancement.

4.3 Crowd Counting

The ability to localize and track people, how many of them, whether they are on foot or car, helps businesses optimize anything from logistics pipelines and inventory management. For example, it will help on store hours, traffic around that area, and scheduling for any events. Our system can be the source date for Crowd Counting.

5. Analysis

5.1 Summary of Proposed Improvements

Introducing the 3D occluded vision system to a process will allow for a more consistent source of manufacturing power. A problem with human workers is that they get tired, and can only work for a set amount of hours, while a robot will work around the clock. Also, the vision system will be easier to maintain. After the initial purchase and installation, there will be minimum maintenance effort. While humans need food, pay, insurance, and other work benefits, the robot will only need to be repaired if there is a manufacturing incident resulting in it being damaged.

5.2 Disadvantages and Limitations

While a robot is able to work around the clock, it will be more prone to making mistakes in object identification. A human can look at a piece, think it's the right piece, pick it up, and then realize it's the wrong piece. The robot will not go through this same process. Instead, once the robot thinks that it has found the right piece, it will pick up and place the piece whether it's right or not. To make up for this possible error there is a second check on the items retrieved. While this increases the chance of the correct piece being chosen, it results in more time being used to select each set of parts.

5.3 Alternatives

Instead of introducing the vision system, sort the items so that there is a spot for each type of item. By doing so, there is no need to recognize objects, instead the system just retrieves one item from the correct bucket.

In the visual provided below we see that on the left there is no visual required. If item A is needed, then there is an exact spot and path available.On the right there is only one path, but multiple items, so a vision system is required to identify the items. This approach favors space saving and most importantly, universality.

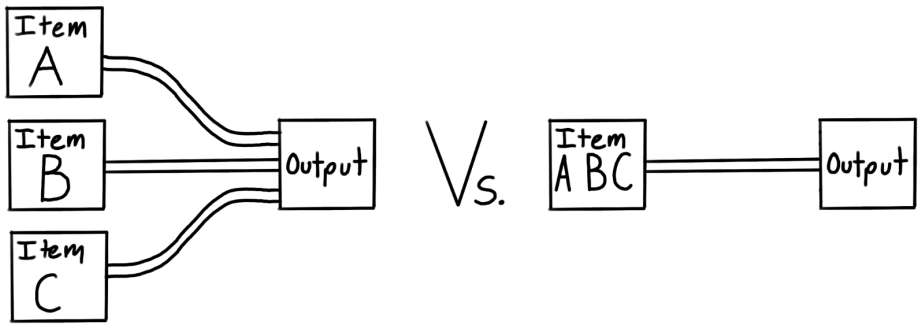


Fig 3. Item Storage Example

While it may initially seem that the left approach is better, the right can be used on multiple systems. If what is being made changes, the right system doesn't need any changes to continue functioning, while the left system needs to be reconfigured to contain the new parts.
Humans are an alternative to using technology, and what is most frequently used. As stated before, humans can only work so much. While it's possible that humans are more accurate, they simply can't keep up with a machine. Let's say that the vision system is only accurate 50% of the time. Well it is fairly well agreed that a human works for eight hours a day. That means that within sixteen hours our system has done just as much work. This still leaves eight hours of the day left, and 50% of that means four hours of meaningful work. Adding this up results in twelve hours of meaningful work from the robot but only eight from a human. And let it be known that this is only an example; the outcome of our systems accuracy comes out to about 90%, resulting in many times more work being done.

5.4 Impact

Factory workers providing manual labor will lose their jobs, but there will be new jobs to handle the maintenance and development of the vision system. The factories will have increased output and a decrease in cost. The public health, safety, and welfare will not be impacted as this product does not produce major safety concerns and if there are any safety concerns it will be addressed in the manual. An example of a safety concern would be do not stick your hand in the socket while plugging in your computer. Environmentally, we have created this project in such a way that it will not produce more heat than any average computer does.

3D Occluded Vision Detection System

Evan Kolin, Samiha Elahi, Tony Jeong, Hannah Hillhouse

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – Final
3 December 2021

Change Record

Rev.	Date	Originator	Approvals	Description
0	3/1/21	Evan Kolin		Draft Release
1	4/28/21	Evan Kolin		403 Final Submission
2	12/3/21	Evan Kolin		404 Final Submission

1. Introduction

1.1. Purpose and Scope

3D object detection has recently received major attention and became an active research topic in 3D technology. While 2D detection is widely used already, 3D object detection has more potential and requires more skills and knowledge in the field of study. For example in traffic, objects such as pedestrians, cyclists, motorcycle riders, or traffic cones are usually represented by sparse points with 2D cameras which makes the detection quite complex using only the upper view of a 2D camera. In this project, we implement 3D object detection with multi-view object recognition. We will use the 2D camera to localize the objects and 3D camera to identify each object. The goal is to detect and identify each of the tools in a box using both 2D and 3D cameras. We shall utilize the 3D camera to classify even occluded objects. This system shall be used in factories for item classification or traffic controls with further research and development.

1.2. Responsibility and Change Authority

Evan Kolin is the team leader. He has managed the purchasing of parts and signing off of team documents. The entire team is accountable for their section and any updates that they make, alongside Evan. Both parties will sign off on future changes.

2. Applicable and Reference Documents

2.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Table 1:

Document Number	Revision/Release Date	Document Title
https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf	2016	Raspberry Pi 3 model B+Data Sheet
https://www.mouser.com/pdfdocs/Multiple_Camera_WhitePaper_rev11.pdf	N/A	Using the Intel® RealSenseTM Depth cameras D4xx in Multi-Camera Configurations

2.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Table 2: Reference Documents

Document Number	Revision/Release Date	Document Title
https://stackabuse.com/image-recognition-in-python-with-tensorflow-and-keras/	2020	Image Recognition in Python with TensorFlow and Keras
https://colab.research.google.com/github/ultralytics/yolov5/blob/master/tutorial.ipynb	2020	YOLOv5-Tutorial

2.3 Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable document are considered to be for guidance and information only, with the exception of ICDs that have their applicable documents considered to be incorporated as cited.

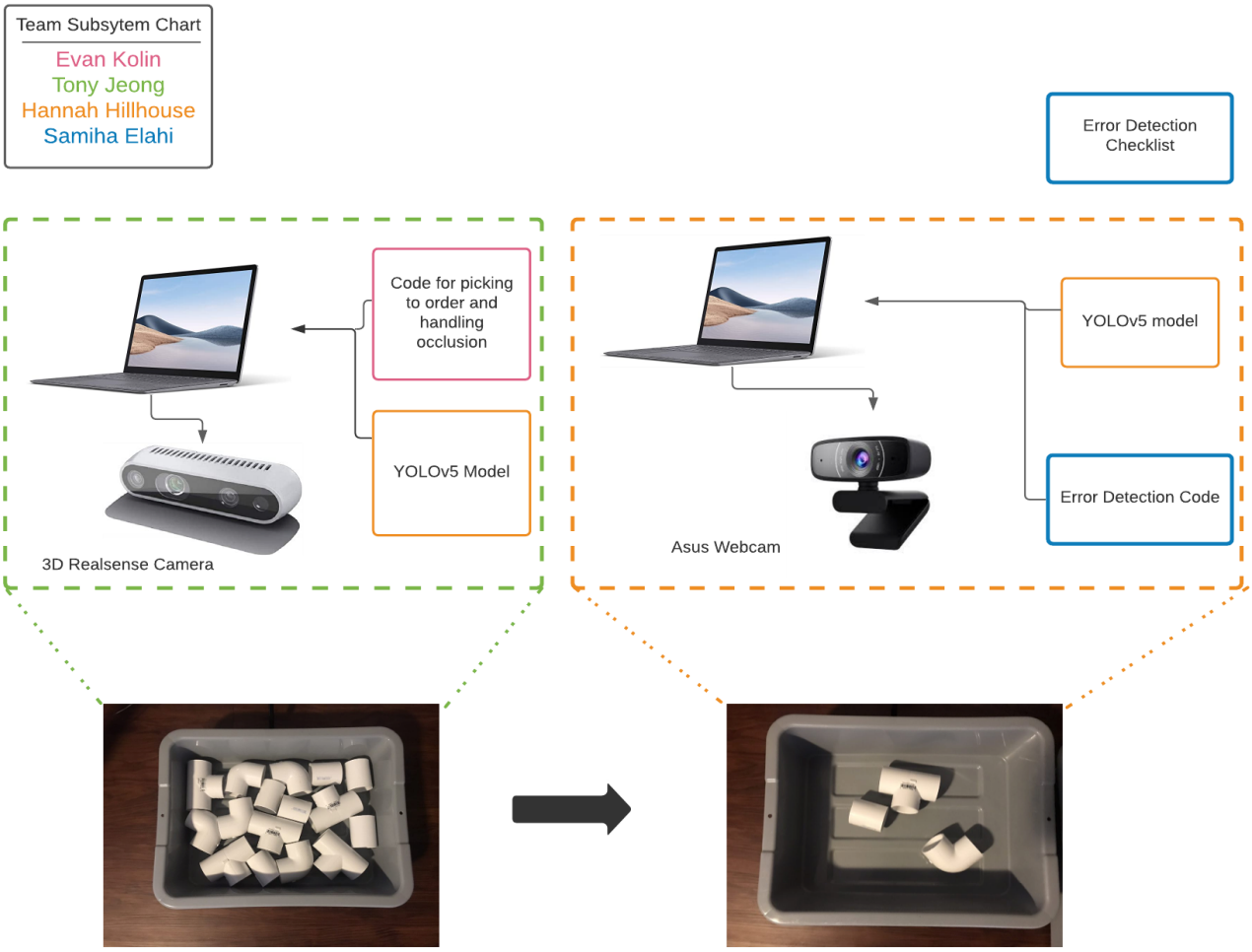
3. Requirements

This section defines the minimum requirements that the development item(s) must meet. The requirements and constraints that apply to performance, design, interoperability, reliability, etc., of the system, are covered.

3.1 *System Definition*

The 3D occluded vision system is broken into four subsystems: 2D camera, 3D camera, AI/Machine Learning, and Error Detection. The cameras take pictures of items in a bucket and then the AI names the objects. The final result is to output the name and location of all objects in a bucket so that a robotic arm could move the items. Our team however, is not in charge of moving or using the arm.

Fig 4. Block Diagram of System



The RealSense D435 and Asus Webcam will classify and localize images of the items in Bucket 1, which will be sent, via 3D Realsense camera, to the AI which classifies and localizes the objects. Once the correct pieces are found and moved into Bucket 2, the Asus Web Camera will again image the items in Bucket 2 to ensure that the correct parts were chosen.

3.1.1 2D Camera Subsystem

This is Hannah Hillhouse's subsystem. The 2D camera subsystem is responsible for taking a picture of the part box and using the Machine Learning model to classify and localize the contents within the image.

3.1.2 3D Camera Subsystem

This is Tony Jeong's subsystem. The 3D camera subsystem is responsible for capturing the object's size and shape that will identify each object. Same as the 2D camera, it's using the Machine Learning model to classify and localize the contents.

3.1.3 Machine Learning Classification and Localization Training Subsystem

This is Evan Kolin's subsystem. This subsystem focuses on designing and training a machine learning (ML) algorithm to classify and localize objects.

3.1.4 Error Detection Subsystem

This is Samiha Elahi's subsystem. The error handling and checking subsystem is responsible for providing a check for the user to determine if their order and all the parts within the order were received and picked up by the robot. The error handling and checking subsystem works closely with the bin area image processing subsystem to obtain data and ensure that all the parts were picked up correctly, if not then it will send an error message to the operator and state how to handle that error.

3.2 Characteristics

3.2.1 Functional / Performance Requirements

The entire 3D Occluded vision system shall localize, classify, and move all requested pieces to the order bin in two minutes.

3.2.1.1. Frequency of Measurements

The error handling and checking subsystem will take data from the bin area image processor continuously to check for errors and inconsistencies. The system will only stop operating if a major error has been detected and it needs operator's attention immediately. Detection of a full bin can be done every two seconds.

3.2.1.2. Lifespan and Maintenance

The 3D Occluded Detection System is mostly made up of software resulting in a very low need for maintenance. The lifespan will be dependent on either the willingness of the operator to continue using the system, or a failure in some of the hardware.

Listed below is the hardware included in the system, the expected lifetime, and the replacement cost. In the expected lifetime it will be assumed that the part is handled correctly, and the replacement cost will be the price paid for the part as of February 28, 2021. Information

regarding lifetime is found in manufacturer specifications or speculative data from forums. For example, since the Pi Camera V2 was released in 2016, the camera's have not stopped working yet and so there is no concrete number for it's lifetime and it can only be speculated by comparing it to similar cameras.

Table 3. Lifespan and Maintenance

Item Name	Expected Lifetime (years)	Replacement Cost (USD)
Raspberry Pi 3	10+	34.99
Pi Camera V2	10	24.99
Realsense D435	5-7	190.99
LT2 Multi-Axis Arms	10+	19,700.00

As can be seen, the hardware involved in this project is expected to last for a very long time. The items indexed at 10+ years have accounts of quiet long lifespans, but were left at 10 to ensure a worst case representation.

3.2.1.3 Object Localization

The localization algorithm shall create a bounding box around objects with 90% accuracy. It shall return the image to the image labeler to allow for the AI to carry out the task.

3.2.1.4 Object Classification

The machine learning algorithm shall return the name of an object with at least 90% accuracy. It shall return the name of all requested objects, including occluded objects, in under one minute.

3.2.1.5 Mounting

Both the Asus webcamera and the Realsense D435 will be mounted. The D435 is our 3D camera, and will be mounted above bucket 1 by a built tripod. The Asus would be connected to the robotic arm in practice, but our team does not have access to the arm and so it will instead be attached to a tripod stand that will overlook the bucket.

3.2.2 Electrical Characteristics

3.2.2.1 Power Consumption

The maximum peak power of the system shall not exceed 6 watts.

> *Rationale: The only high wattage piece is the Realsense D435 at 4.5 watts, other pieces all require less than 1 watt.*

3.2.2.2 External Commands

The system will have a command for moderators to tell the system what to do after an error has occurred.

3.2.2.3 Data Output

The Vision System shall output the name and location of all visible objects within a bucket of items. It also provides a bucket to the user containing the requested parts.

> *Rationale: The information gathered is sent to the robotic arm which would then move the pieces.*

3.2.2.4 Connectors

The Vision System shall contain few connectors, used only to connect the two cameras to the laptops via usb.

.

3.2.3 Environmental Requirements

The Vision System shall be designed to operate indoors in a factory setting and away from any harsh weather.

> *Rationale: The intended use of our system is in an assembly line, as such it is expected to be used indoors.*

3.2.4 Failure Propagation

3.2.4.1 Diagnostic Errors

The 3D Occluded Detection system will monitor the inputs or the photos captured by the camera and look for any errors or abnormalities. The object will detect errors by extracting the test images and then comparing them with the trained images. If an error is detected, the error will be displayed on a screen in order to seek the operator's attention immediately.

3.2.4.2 Fault Detection

The 3D Occluded Vision Detection System system shall be able to detect an incorrect part 90 percent of the time.

3.2.4.3 False Alarms

The vision system shall have a false alarm rate of less than 2 percent.

4. Support Requirements

The 3D occluded vision system will need minimal support. The system comes with all of the materials needed to function except for software updates and the parts being chosen. As the system is used in new environments it may need to learn how to detect new objects. Issues in the field will be managed by an on site operator. If there is an error detected, the nature of the error and a description will be sent to the operator who will then fix the error and notify the vision system that it can resume working.

Appendix A: Acronyms and Abbreviations

ML	Machine Learning
AI	Artificial Intelligence
2D	2 Dimensional
3D	3 Dimensional
CNN	Convolutional Neural Network
ReLu	Rectified Linear Unit
SeLu	Scaled Exponential Unit

3D Occluded Vision Detection System

Evan Kolin, Hannah Hillhouse, Tony Jeong, Samiha Elahi

INTERFACE CONTROL DOCUMENT

REVISION – 2
3 December 2021

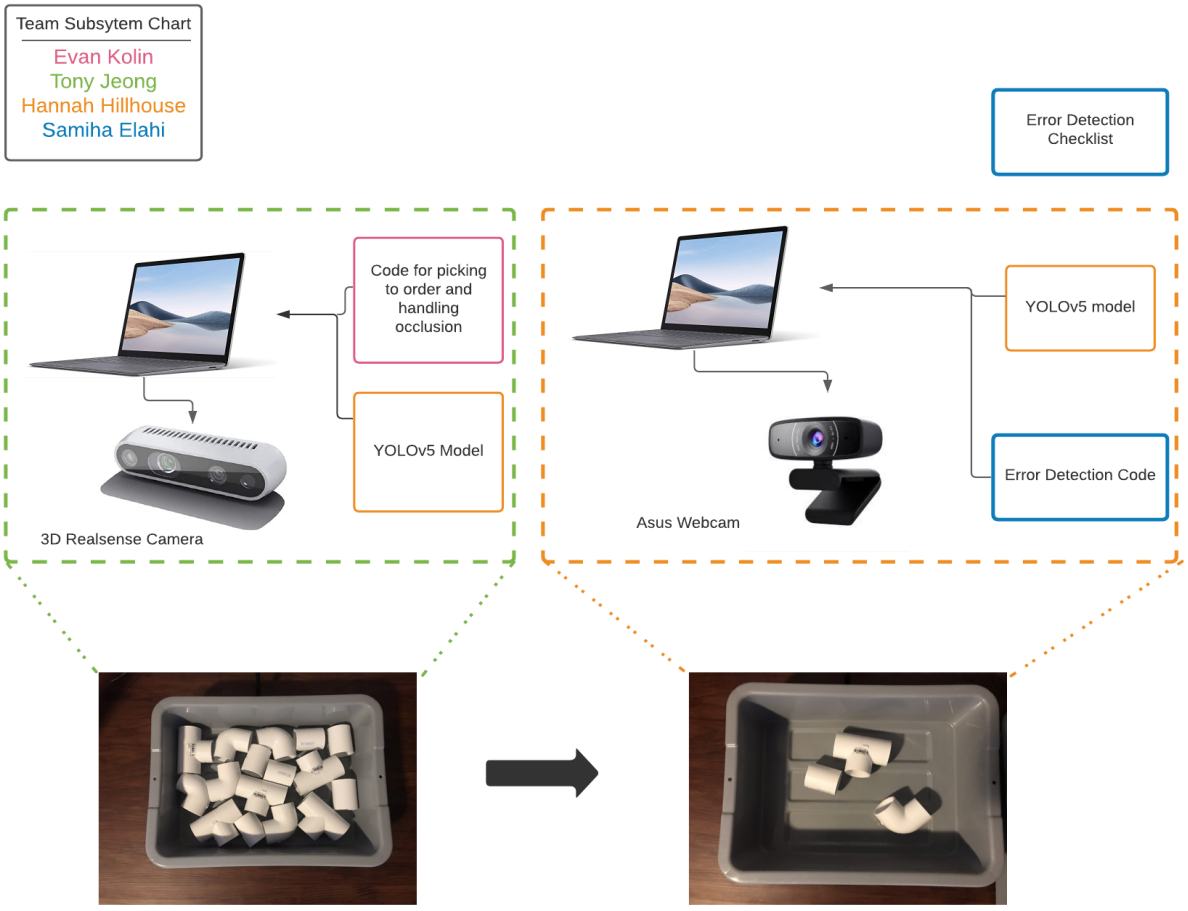
Change Record

Rev.	Date	Originator	Approvals	Description
0	3/1/21	Evan Kolin		Draft Release
1	4/28/21	Evan Kolin		403 Final Submission
2	12/03/21	Evan Kolin		404 Final Submission

1. Overview

The ICD will detail the entire system of 3D occluded object detection and how it will accomplish our requirements specified in the FSR. It will include detailed descriptions of our entire system's physical interface, location, angle of the camera, and size. The following will be the electrical interface that will describe the user control interface. The last section is communication and Device interface section, it will include how each device will communicate with other devices and what is the host device that will control all.

Figure 5: System Overview Diagram



2. References and Definitions

2.1 References

IEEE Xplore, 2020
An Ergonomics-Based Storage Bin Allocation for Picking Efficiency Improvement
2020 revision

Engineering, vol. 138, pp. 106094, 2019.
Optimization of layout and item assignment of a U-shaped order picking zone
2019 revision

Vision Systems Design
Advances in 3D object detection in vision-guided robotics
2020 revision

Rev 3.4
Quad-A7 control
8/18/2014

2.2 Definitions

mA	Milliamp
2D	Two Dimensional
3D	Three Dimensional

3. Physical Interface

3.1 Weight

The 3D Occluded Vision System will weigh less than twenty pounds. Thus, it allows an average person to lift and move the entire station into place and the station will be more secure with added weights near its base.

Table 4: 3D Vision System Weight Specifications

Component	Weight
Intel RealSense Depth Camera D435	Est. 0.5 lb
Asus Web Camera	0.2 lb
PVC Pipes (20)	4 lb
Tripod (2)	10 lbs
Total	14.7 lbs

Weight of the 2D and 3D Camera Subsystem

Both the 2D and 3D camera subsystems will weigh less than 20 lbs. Both the cameras use the tripod to hold the cameras because it will allow an individual to easily move the station into places and allows the station to be more stable with added weight near its base.

3.2 Mounting Locations

3.2.1 Placement of Station

The station will need to be placed on a solid flat square meter section of land. The station will also need to be fixed into the ground with stakes if windy conditions will be present.

3.2.2 Mounting

The Pi camera and the Realsense D435 will be mounted above a bucket by a built tripod. The initial plan was to connect the cameras by a robotic arm in practice , unfortunately our team doesn't have access to it. Therefore it will instead be attached to a tripod stand that subsystem 2 will make overlooking the bucket. In practice, the tripod is located on one side of the bucket, so the far end of the bucket is warped to appear slightly smaller. We calculated the expected change by taking the known width of the bucket and comparing how it appears at the closest point and at the farthest point, concluding it warps to be smaller by about 3%.

4. Electrical Interface

4.1. *User Control Interface*

There will not be much control needed by the user, except for inputting the total number of parts needed to meet the order requirement. We will be using the command prompt screen to show our result and any error occurred. The user will be able to check the results and errors so he/she can make changes or fix the materials that the cameras will scan for 3D occluded object detection.

4.2. Thermal Interface

The cameras will not need a cooling system. However, the laptops hosting the AI will need cooling, but today every laptop has its own fans built in, which should be more than enough.

5. Communications / Device Interface Protocols

5.1. User Laptop

The host device shall be any laptop loaded with windows 10 or higher.

5.2. Video Interface

We will use a USB 2.0 to feed video to the Laptop from both the 2D and 3D camera. In Preliminary testing, the Information will be fed directly from the cameras to our computers.

5.3. Internal Communications

All subsystems will be using python to ensure that code and libraries are easily made compatible. When a camera localizes a picture and creates a bounding box it will transform the contents of the bounding box into an image and send it to the AI to classify. This is done using YOLOv5, to break one image with multiple parts into multiple images with only one part. Then the error detection code will take that classified image and compare the total parts detected in that image with the user's input to check if we have a missing or an extra part.

5.4. Device Peripheral Interface

The only communication interfaces we need to use are USB’s to connect our camera’s to their respective laptops so that image data can be transferred. Besides that, all software is stored inside of the same environment on the computers, so communication is done between files in the same folder.

5.5. Internal Communications

Most of the work is done inside of the 2D and 3D camera subsystem. Both systems take in external image data, then the rest of the work is done internally. Both systems use the YOLOv5 model to make predictions based on the image provided, then after storing that data into a prediction variable it can find individual classification and localization data for each piece detected.

Figure 6: Generating prediction using YOLOv5 model

```
# Inference
t1 = time_sync()
pred = model(img, augment=augment,
             visualize=increment_path(save_dir / 'features', mkdir=True) if visualize else False)[0]

# Apply NMS
pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)
t2 = time_sync()

# Apply Classifier
if classify:
    pred = apply_classifier(pred, modelc, img, img0)
```

Figure 7: Using prediction data

Classification stored in: names[c]

Localization stored in: bBox

```
if (count_I2 != 0) and (names[c] == 'I'):
    print("coint_I2 is: ", count_I2)
    count_I2-= 1
    boundingbox_color = (0, 255, 0)
    bBox = annotator.box_label(xyxy, label, color=boundingbox_color)
    order_list.append("I")
    #LowerLeft X and Y
    #UpperRight X and Y
    LLBBX = bBox[0][0]
    LLBBY = bBox[0][1]
    URBBX = bBox[1][0]
    URBBY = bBox[1][1]
    #Translating pixel values into real distance
    bB1X = round(LLBBX/640*29,2)
    bB1Y = round(LLBBY/480*22,2)
    bB2X = round(URBBX/640*29,2)
    bB2Y = round(URBBY/480*22,2)
    #putting values into a readable format
    bboxCords = [(bB1X,bB1Y),(bB2X,bB2Y),bBox[2]]
    #append to order list
    order_list.append(bboxCords)
```

3D Occluded Object Detection System

Hannah Hillhouse, Tony Jeong, Evan Kolin, Samiha Elahi

Subsystem Reports

REVISION – 2
3 December 2021

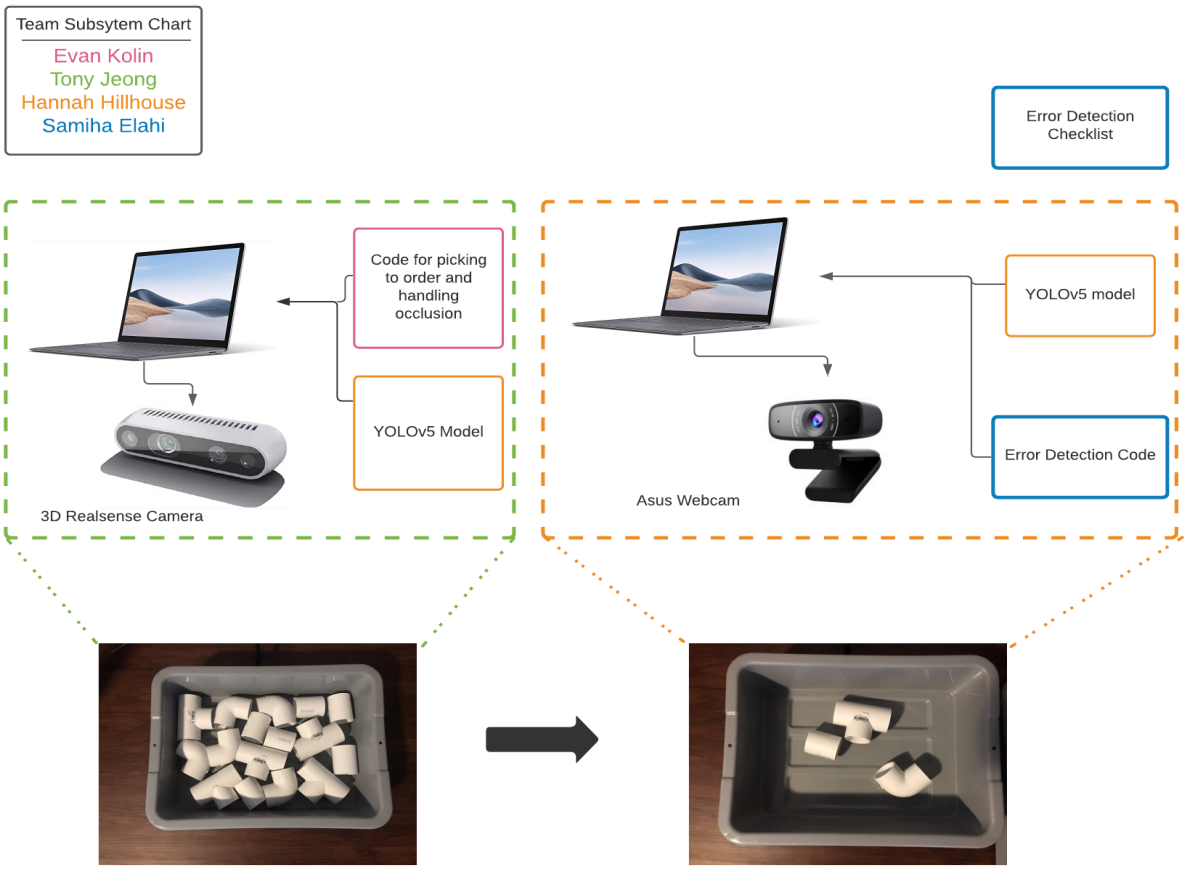
Change Record

Rev.	Date	Originator	Approvals	Description
0	4/27/2021	Evan Kolin		Subsystem Report Initial
1	12/4/2021	Evan Kolin		Subsystem Report Final 404

1. Introduction

The 3D occluded object detection system is made to classify and localize objects located in a bin full of parts. It will pass the name and location of an object to a robotic arm, which will then move the item to a second bin containing only the parts requested. Note that we are only building the system providing information to the arm, not the arm itself.
Our project is broken into four systems: 2D camera, 3D camera, Classification AI, Error Detection

Fig 8. Subsystem Break Up



2. 3D Camera Subsystem Report

2.1. Subsystem Introduction

The 3D camera subsystem will be used on a box with multiple occluded items for classification and localization of each item. This will be then sent to 2D camera subsystems and error detection subsystems.

2.2. Subsystem Details

The 3D camera used for this subsystem is Intel realsense D435 which provides quality depth with depth sensor that can be used for a variety of applications. For this project, this skill will be used to generate the z scale of the image which will differentiate it from a 2D camera. In modified resolution, 640x480, for best raw depth performance out of the Intel realsense camera, the program will record the distance from the camera to the objects by low left and upper right corner of each detected object within the resolution and record it. At the same time, the program will generate bounding boxes of each object in the box with skills of classification and localization. For bounding boxes, machine training with Yolov5 and Unit for depth de-noising and hole filling which provides better performance of 3D depth scaling and better accuracy of making bounding boxes of targeted objects.

2.3. Subsystem Validation

2.3.1 Bounding box

2.3.1.1 Lighting and Confidence Threshold

Although the system has gone through training on yolov5, it seems to have different accuracy levels depending on lighting environment when actually tested with intel realsense camera. This was because the PVC pipes were white and glossy, reflecting light easily. The lighting issues were solved by adding a confidence threshold. The confidence threshold is set to 0.7 which means it will only draw bounding boxes for the objects that the AI is at least 70% sure on classification. It may cause the program to miss some pieces to draw the bounding box, but it also works as notifying us that the lighting should be adjusted.

2.3.1.2 3D mapping noise

Also, after researching some articles with intel realsense features, there is an issue of noises occurring within 3D scaling. Figure 2 shows an example image of 3D scaling with a lot of noises. Any space with black colors is not recognized by a 3D camera, thus not classifying the objects. This was solved by training the intel realsense camera program (Figure 3).

Fig 9. Noise problem example

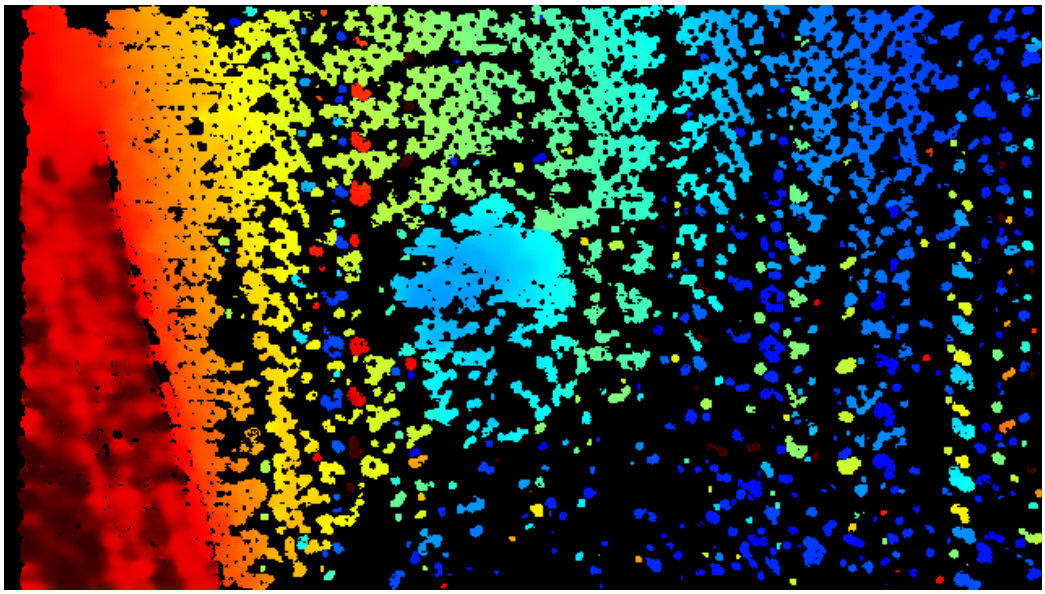
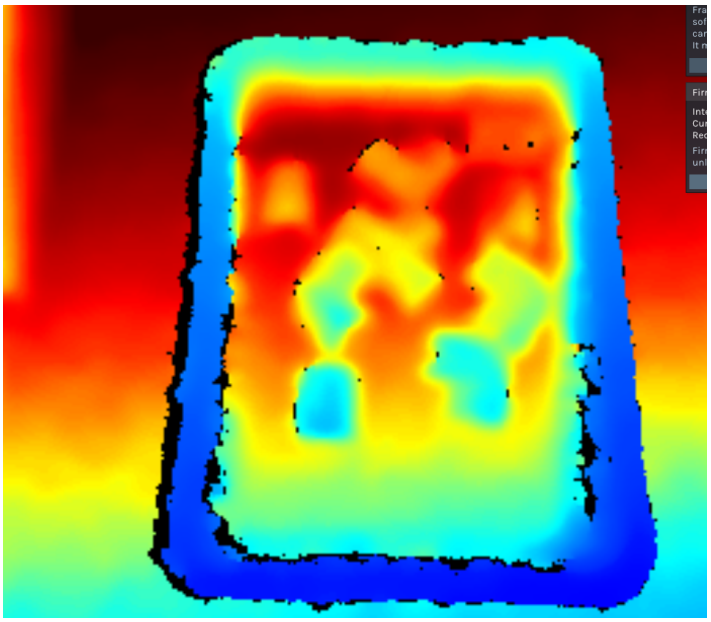


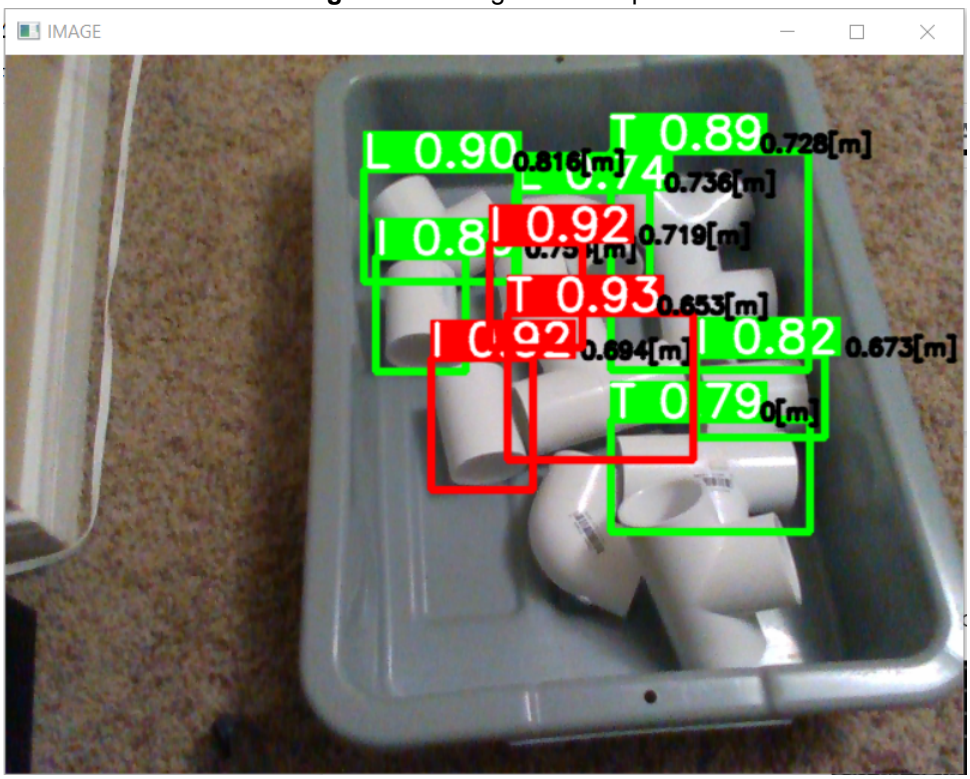
Fig 10. Noise problem solved



2.3.1.3 Sizing issues

The last problem encountered was that it was hard to recognize the objects when they were too small. It showed higher accuracy levels when the objects were bigger. This was solved by actually changing the objects from bolts and nuts to PVC pipes.

Fig 11. Bounding Box example



2.3.2 Distance by Pixel

In the first place, it was planned to build a program that generates z-scale of 3D camera view with color codes. This then shows the depth scales with colors on output, blue as closest and red as farthest. However, I was able to find a discussion where it has the same problem and found a solution. There was a direct code line that extracted z-scales with distances number of millimeters(mm). Two code lines mainly used were

frames.get_depth_frame()

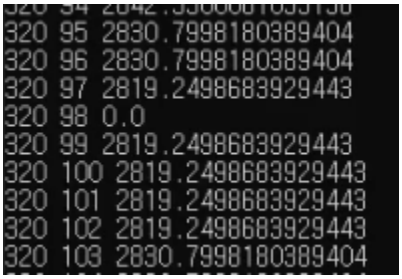
which generates a frame with depth data and

depth.get_distance()

which generates distances on selected spots with usage of depth data. I have written lines of

code that will generate distance from a 3D camera to the objects in each pixel of modified resolution, 640x480, which results in about 300,000 values.

Fig 12. Distance by pixel example



However, there was a problem encountered where it outputs a zero distance which could never happen. As it is shown in Figure 12, it is showing zero distance on x scale of 320 and y scale of 98. This is happening because of noises in the 3d depth sensor which was improved by training, but not fully denoised. For a better interface for users and the noise problem, I have decided to only print out a few distance outputs of the lower left corner and upper right corner of each detected object. In the command console window, it will print out those two values and on the result screen of camera view in figure 12, it will print out the distance between the camera to the center of detected boxes in meters.

2.3.3 User Interface

Fig 13. Outputs in user interface.

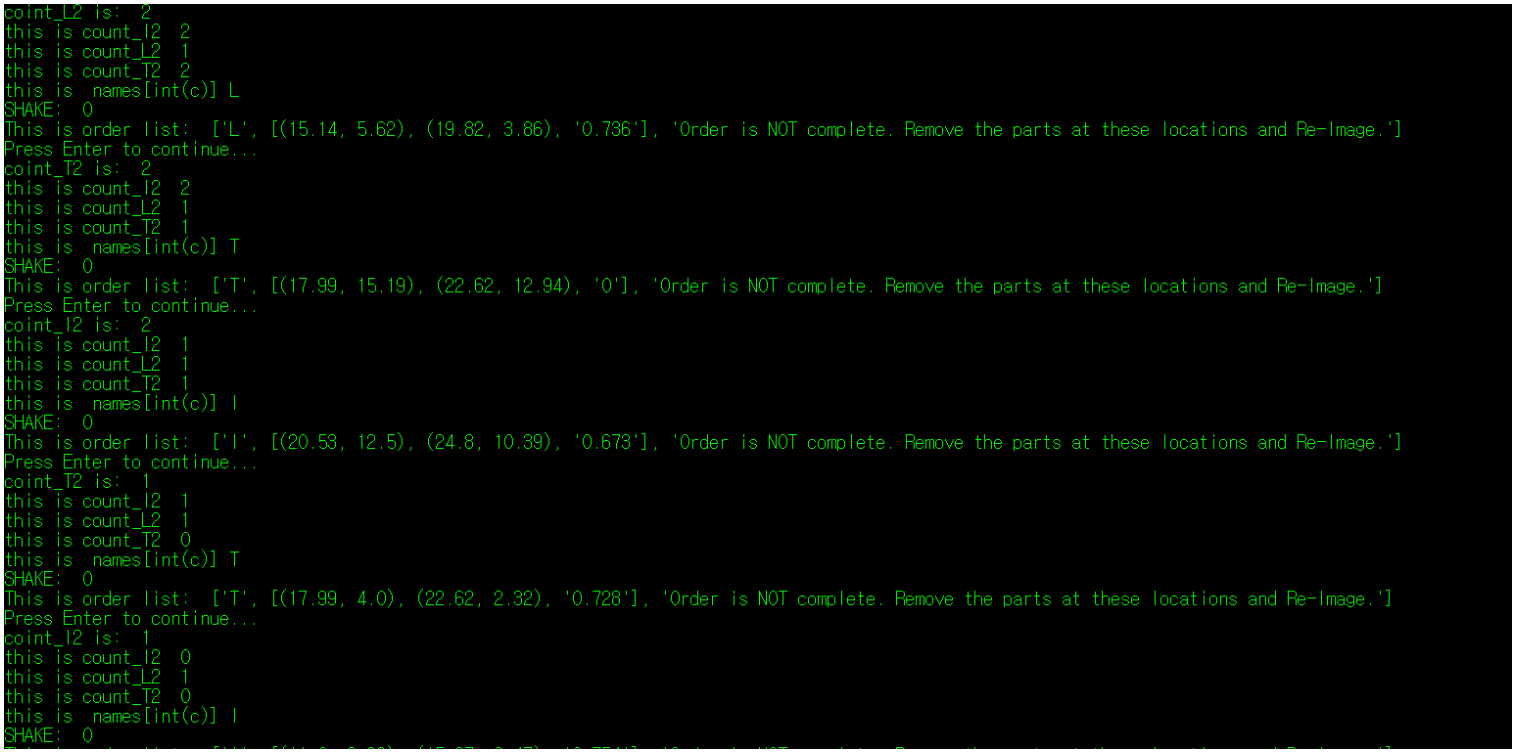
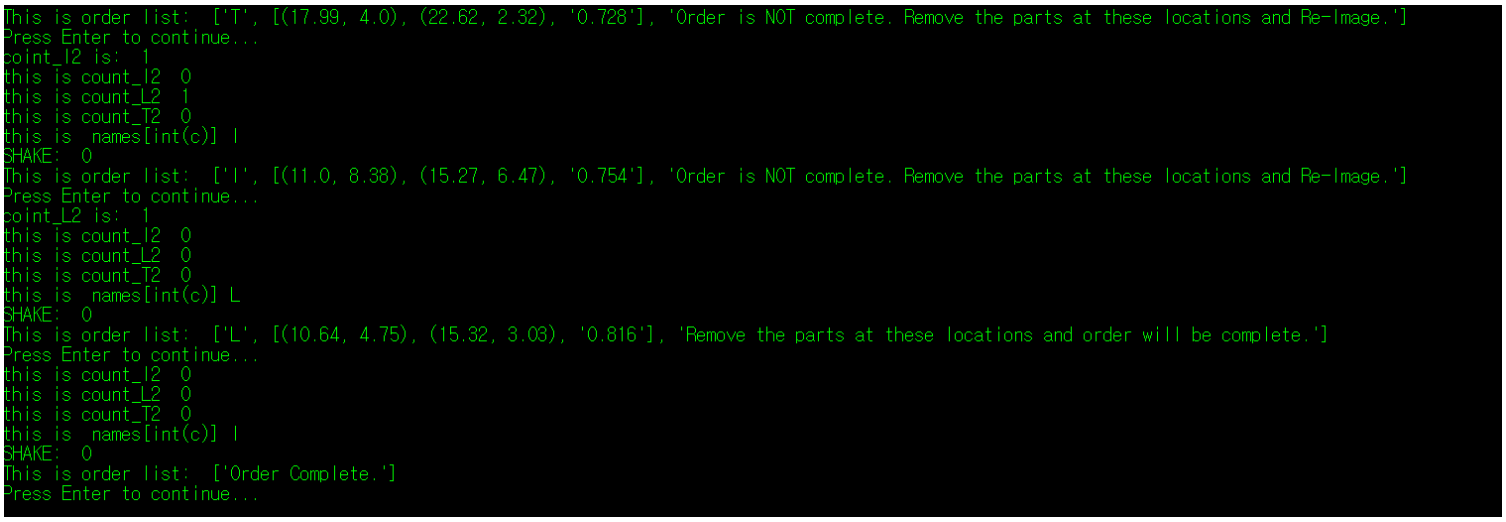


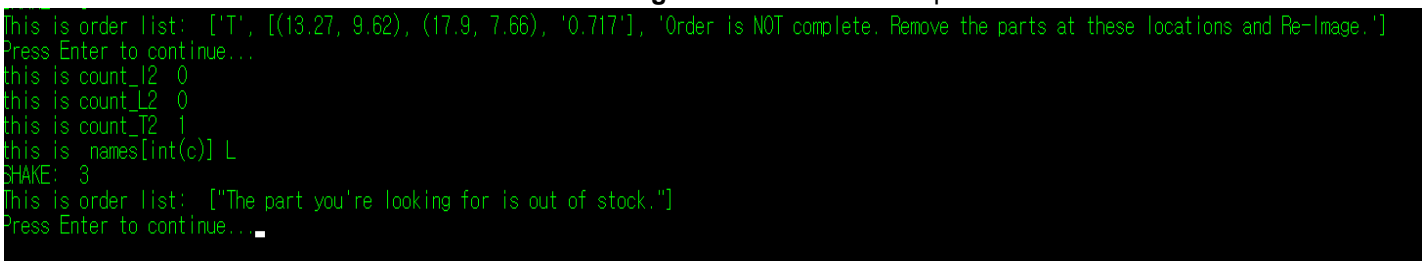
Fig 14. Outputs in user interface. cont.



The program will start by asking the user how many pieces he/she wants for each I,L, and T shape of PVC pipes. Then, it will print out the location of the user's wanted object in x, y, and z scale one object by one object. For example in figure 13, the user asked for 2 pieces of each I,

L, and T shaped PVC pipes. First it found a L shaped piece and printed out its x and y scale value in inches for lower left corner and upper right corner. Also, it printed out the distance from the camera to the center of the bounding box in meters. It will then count off from count_L2 which is the number of L pieces the user asked for. After finding a PVC pipe piece, the user hits enter on the keyboard which will then process the next desired piece. At the end, it will print out “order complete” as all counts for wanted pieces reach 0. Also everytime it has found a piece, it will show the output screen as in figure 14. It will be drawing bounding boxes for every detected piece and only the pieces the user wants will be colored in green and the other will be colored in red. On the top left corner of each bounding box, it will print out what type of piece it is, accuracy level, and the distance from camera in meters.

Fig 15. Out of stock example.



There will be some unexpected cases where the user asks for pieces more than we have or the pieces were occluded so much that it could not find the piece that was hiding. We added a ‘shake’ counter if it could not find the desired piece at first time which will ask the user to shake the bucket and re-image. If the user shakes the bucket for 3 times and could not find the desired piece, it will recognize it as the user asked for pieces more than what the bucket actually had. Then, it will print out “The part you’re looking for is out of stock”. For example in Figure 7, The user asked for 2 of T pieces and there was actually only one piece on the bucket. After detecting the T piece and the robot arm takes it out of the bucket, the counter for the user wanted T piece will remain 1 since there isn’t more piece to find. After the shake counter reaches 3, it will be recognized as the part is out of stock.

3. Classification and Localization AI Subsystem Report

3.1. Subsystem Introduction

The Classification and Localization AI subsystem is responsible for handling the detection of the object in an image given by the 2D and 3D subsystem. The only output of my system is two arrays containing the classification weights and the coordinates of the localized bounding box.

Fig 16. Classification and Localization

```
classification:  [ -6.888317 -14.068576  10.037672]
localization:  [0.39973783 0.17988908 0.9534514  0.6332836 ]
```

Figure 6 above shows the classification weights as [I_Shape, L_Shape, T_Shape], so the T shape weight is heavy, while the others are negatives, resulting in a classification of T_Shape. Then the localization data is in the format of the lower left corner coordinates then upper right hand coordinates [Xsmall,Ysmall,Xbig,Ybig].

An issue I originally had was a misunderstanding in what the (0,0) coordinate was. I had wrongly assumed that (0,0) was the bottom left, but it's actually the top left, with the positive Y axis running down the image.

3.2. Subsystem Details

3.2.1 Structure

Fig 17. Classification AI Block Model

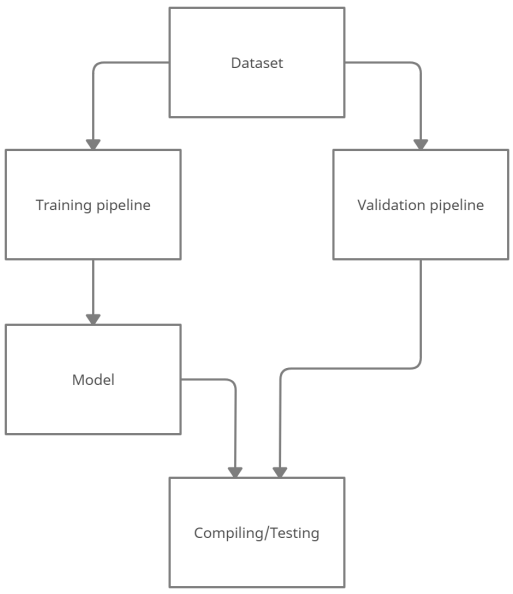


Figure 17 is an older example of the structure of the model, but it's still an accurate outline displaying that the concept of how data moves is fairly simple. The increased complexity comes from an expansion in the Training pipeline and the Model.

Fig 18. Training Pipeline

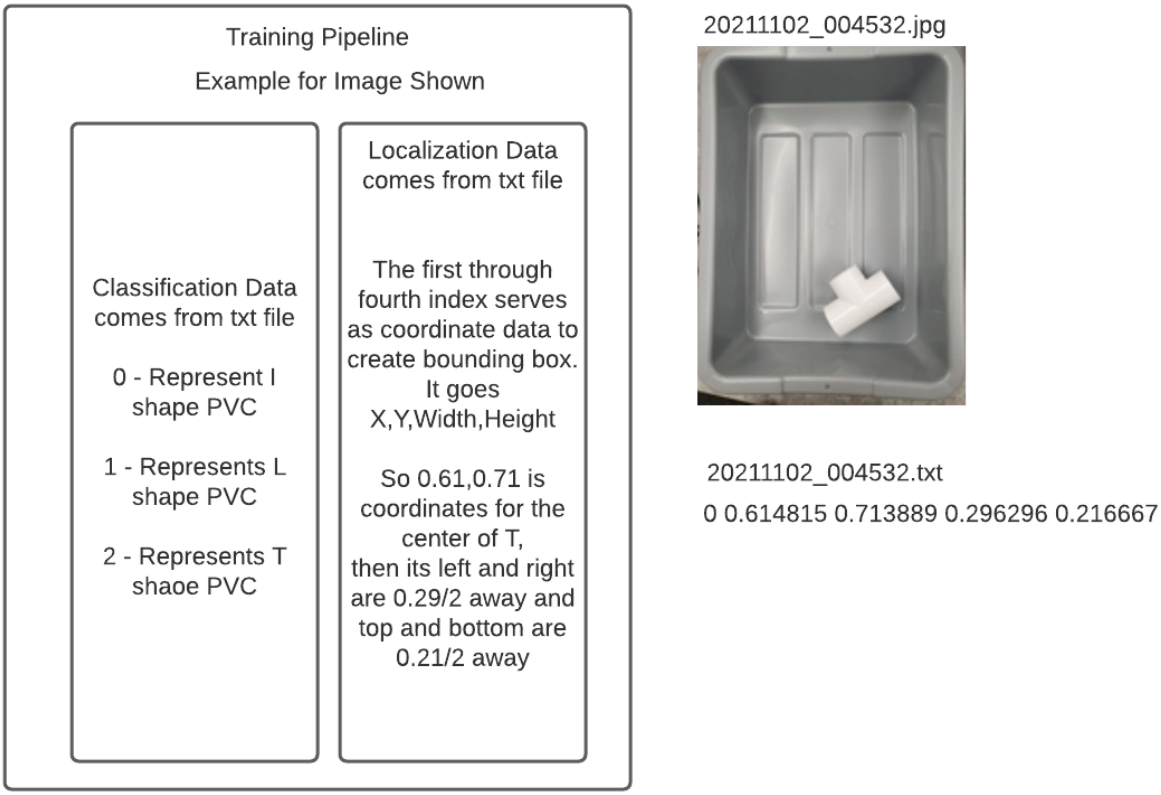


Fig 19. Model

```
#This model trains slower but has more trainable parameters
input_layer = layers.Input((height,width,3), name='input_layer')
#input_layer = layers.Input((None, None, 3), name='input_layer')
base_layers = layers.experimental.preprocessing.Rescaling(1./255, name='base_1')(input_layer)
base_layers = layers.Conv2D(16, 3, padding='same', activation='relu', name='base_extra')(base_layers)
#base_layers = layers.Input((28,28,1))(input_layer)
base_layers = layers.Dense(256, activation="selu", name='base_2')(base_layers)
base_layers = layers.Dense(128, activation="selu", name='base_3')(base_layers)
base_layers = layers.Conv2D(128,3, padding='same', name='base_4')(base_layers)
base_layers = layers.MaxPooling2D(name='base_5')(base_layers)
base_layers = layers.Dropout(rate=0.2, name='base_6')(base_layers)
base_layers = layers.Dense(64, activation="selu", name='base_7')(base_layers)
base_layers = layers.Dense(32, activation="selu", name='base_8')(base_layers)
base_layers = layers.Conv2D(32,3, padding='same', name='base_9')(base_layers)
base_layers = layers.MaxPooling2D(name='base_10')(base_layers)

classifier_branch = layers.Flatten(name='class_1')(base_layers)
classifier_branch = layers.Dense(3, name='class_head')(classifier_branch)

locator_branch = layers.Dense(128, activation='relu', name='locator_1')(base_layers)
locator_branch = layers.Dense(64, activation='relu', name='locator_2')(locator_branch)
locator_branch = layers.Dense(32, activation='relu', name='locator_3')(locator_branch)
                              #using sigmoid because not looking at image but numbers
locator_branch = layers.Flatten(name='loc_3')(locator_branch)
locator_branch = layers.Dense(4, activation='sigmoid', name='locator_head')(locator_branch)
```

Fig 20. How Data splits inside of model

```
#Define dictionary for train and validation
trainTargets = {
    "class_head": train_labels,
    "locator_head": train_targets
}

validationTargets = {
    "class_head": validation_labels,
    "locator_head": validation_targets
}


#fit model
history = model.fit(train_images, trainTargets,
             validation_data=(validation_images, validationTargets),
             batch_size=1,
             epochs=3,
             shuffle=True,
             verbose=1)
```

As seen in figure 19 there are a set of base layers that all information will pass through, but there are two final branches, classifier_branch and locator_branch. These branches' final layer is named “class_head” and “locator_head” respectively, which allows me to create a dictionary that defines what the model looks at for training. In figure 20 you can see the trainTargets and validationTargets taking advantage of the layer names, so that when using model.fit the model can split the data.

3.2.2 Training/Validation Pipeline

The Training/Validation Pipeline code has completely changed since last semester. I was previously using a prebuilt Tensorflow function, but it wouldn’t work anymore since I need two types of data for classification and localization.

Now what happens is the training pipeline takes the list of training files and reads them in one by one, transforming the data into arrays, and then tensors, that the model can use. This approach was taken and modified from Chatura Wijetunga, an author from medium.com.

Fig 21. Training Pipeline

```
TRAINING_CSV_FILE = 'C:/Users/evanp/AI/Capstone/bounded_dataset/Training3_data.csv'
TRAINING_IMAGE_DIR = 'C:/Users/evanp/AI/Capstone/bounded_dataset/Training3'

training_image_records = pd.read_csv(TRAINING_CSV_FILE)

train_image_path = os.path.join(os.getcwd(), TRAINING_IMAGE_DIR)
print("train_image_path: ", train_image_path)

train_images = [] #actual image
train_targets = [] #location of image
train_labels = [] #classification on image

for index, row in training_image_records.iterrows():

    #(filename, width, height, class_name, xmin, ymin, xmax, ymax) = row
    (class_name, xCenter, yCenter, width, height, filename) = row
    print('index: ',index)
    #Represented as a percentage of the entire image, so I multiply
    #by the true image width/height to get the actual pixel value.
    print('xCenter | yCenter | width | height')
    print(xCenter, '|', yCenter, '|', width, '|', height)

    #height = round(float(height) * 240)
    #width = round(float(width) * 180)
    #xCenter = xCenter * 180
    #yCenter = yCenter * 240

    xmin = round(xCenter - (width/2), 3)
    xmax = round(xCenter + (width/2), 3)
    ymin = round(yCenter - (height/2), 3)
    ymax = round(yCenter + (height/2), 3)

    print('xmin | xmax | ymin | ymax')
    print(xmin, '|', xmax, '|', ymin, '|', ymax)

    #filename = "20210909_125548 - Copy.jpg"
    train_image_fullpath = os.path.join(train_image_path, filename)
    print(train_image_fullpath, '\n')
    #train_img = keras.preprocessing.image.load_img(train_image_fullpath, target_size=(height, width))
    train_img = keras.preprocessing.image.load_img(train_image_fullpath, target_size=(240, 180))
    train_img_arr = keras.preprocessing.image.img_to_array(train_img)

    train_images.append(train_img_arr)
    train_targets.append((xmin, ymin, xmax, ymax))
    #train_targets.append((0, 0, 180, 240))
    train_labels.append(class_name)
```

```
train_images = tf.ragged.constant(train_images)
train_targets = tf.ragged.constant(train_targets)
train_labels = tf.ragged.constant(train_labels)
validation_images = tf.ragged.constant(validation_images)
validation_targets = tf.ragged.constant(validation_targets)
validation_labels = tf.ragged.constant(validation_labels)



print(type(train_images))
#print(train_images)
#print(len(train_images))
```

```
<class 'list'>
141
<class 'tensorflow.python.ops.ragged.ragged_tensor.RaggedTensor'>
```

```
train_images = train_images.to_tensor()
train_targets = train_targets.to_tensor()
#train_labels = train_labels.to_tensor()
```

3.2.3 Model

The model is a convolutional neural network made with the tensorflow keras sequential model framework. It has nineteen layers, at one point having up to twenty eight layers during testing. As of now, the model is not 100% accurate, due to requirements changing last minute. The issues we ran into were changes to the original design, libraries not loading, learning how to feed more than one type of data through one model, and localizing objects.

3.2.3.1 Changes to original design

Originally my model was only supposed to supply classification data, which it does with great accuracy, but in the last five weeks of the project the requirements changed so that I would also have to do localization. This was because originally YOLO was planned to be functional at the end of 403, but it wouldn't become functional until three weeks before our demo. Additionally, the 3D subsystem had no way of localizing, so it became my job to do that. I was able to recreate my model to handle localization, but even then, Tensorflow doesn't support multi-object classification. This meant that we were yet again waiting for YOLO to work, so that we could run my model through more training. Due to issues with libraries required for YOLO to run, it took too long to get YOLO working, so there wasn't enough time to train my model. Instead we used a pretrained model that Hannah Hillhouse, the 2D subsystem owner, then trained on our data through YOLO.

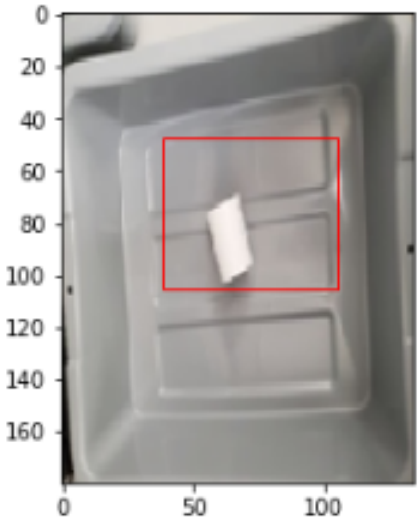
3.2.3.2 Localization errors

Due to the issues in 3.2.3.1, I never fully finished localization on my model. It will localize I and T shape pieces with roughly 50% accuracy, the box is always close and overlaps the piece at least little, frequently capturing the entire piece, but the L shape piece is very off. It will frequently draw a box more than a parts length away. These issues would have been fixed with more image data and during YOLO training, which again we ran out of time on.

3.2.3.3 Model Results

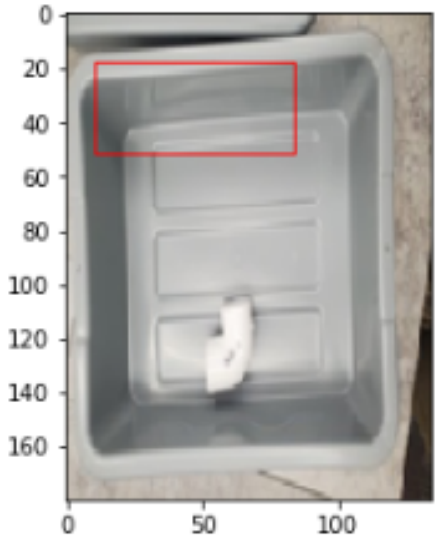
The original goal was to achieve 70% accuracy, which we quickly realized was too low, so then shot for 90% accuracy. My model ended up with about a 95% accuracy in classification, and only about 40% accuracy with localization.

Fig 22. Detection Example I



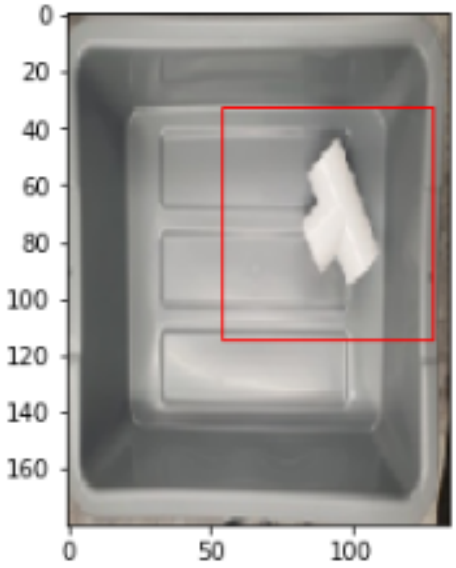

```
classification:  [ 12.256303   -29.048964     0.09246722]
localization:  [0.28325823 0.2619067  0.7787104  0.58345985]
```

Fig 23. Detection Example L




```
classification:  [-23.938196   17.707718   -2.3412948]
localization:  [0.07544905 0.09499556 0.61931837 0.28371584]
```

Fig 24. Detection Example T



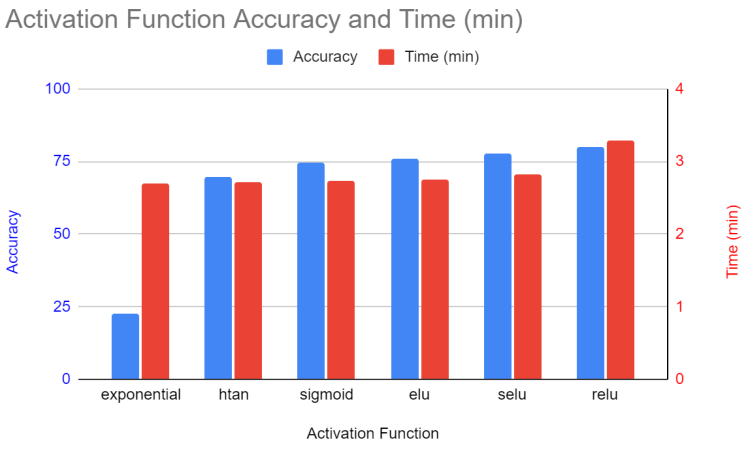

```
classification:  [ -6.888317 -14.068576  10.037672]
localization:  [0.39973783 0.17988908 0.9534514  0.6332836 ]
```

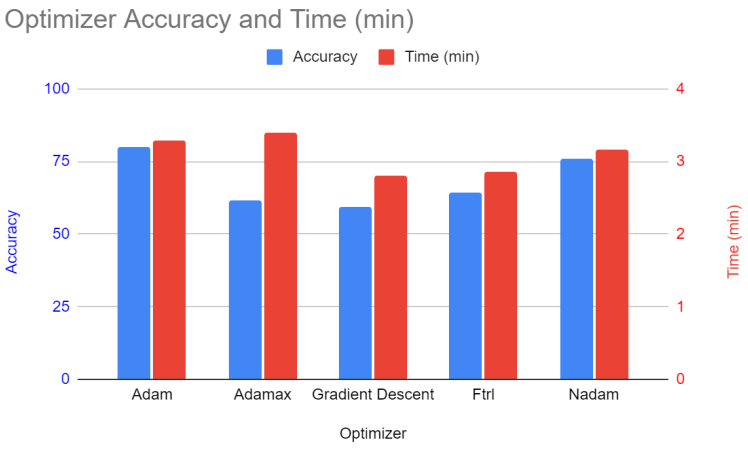
3.3. Subsystem Validation

3.3.1 Hyperparameters

The hyperparameters were tuned to get the best functioning model. I specifically looked at the activation functions, optimizers, learning rates, and loss functions. My initial testing was to exhaustively go through the list of activation functions provided by tensorflow and mark down their accuracy and training times. After taking the top five performing activation functions I applied them to different optimizers. Next, I took the best performing optimizer/activation function combinations and started messing with learning rates.

Fig 25. Activation Function and Optimizer Graphs





The Adam optimizer performed the best across the board with every single activation function, even after increasing/decreasing the CNN depth and changing dataset size. I then had to choose between the two best activation functions ReLu (Rectified Linear Unit) and SeLu (Scaled exponential Unit). While ReLu had a slightly better performance of 80.15% accuracy and SeLu had a 77.94% accuracy, I decided to go with SeLu for two reasons. It was less prone to the dying ReLu problem and it was built to work better with stacked depth layers. These thoughts made me go with SeLu, seeing the potential future increase as much higher. I ultimately ended up with an accuracy of 95%.

3.3.2 Accuracy

During validation, the classification had a few runs with perfect guesses, but accuracy balances out to about 95%, and this training resulted in an 84% accuracy. However, the localization data was roughly 40%. The model itself believed that the localization training was highly accurate, at 88%.

Fig 26. Accuracy Graph

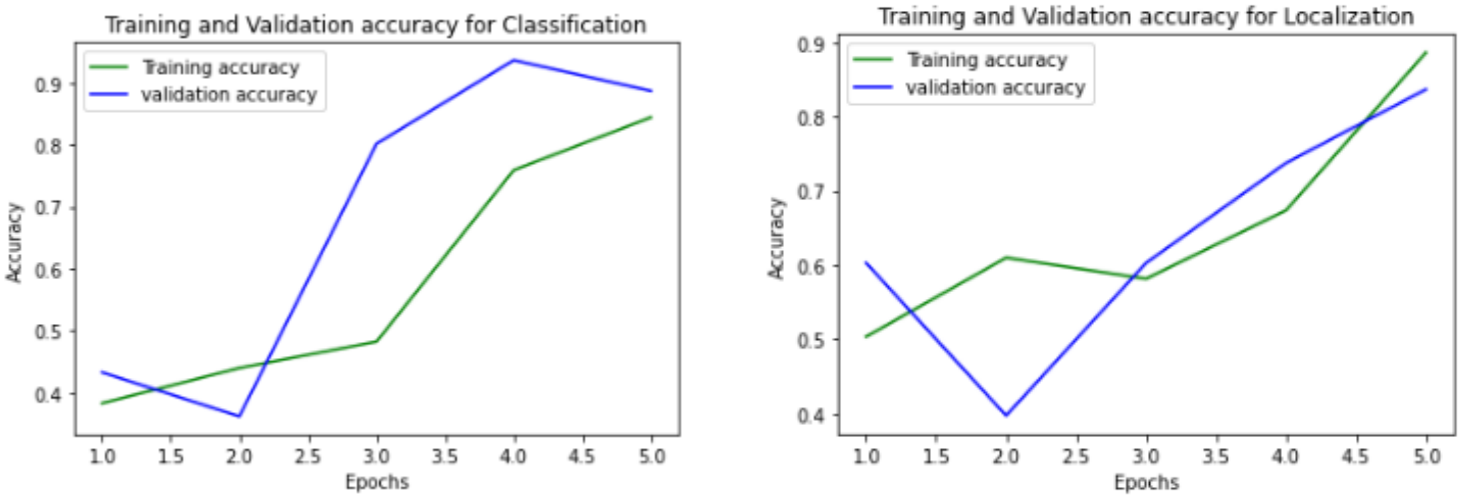


Fig 27. Confusion Matrix

```
Confusion Matrix
[[37  0 10]
 [ 0 37  6]
 [ 0  0 51]]
```

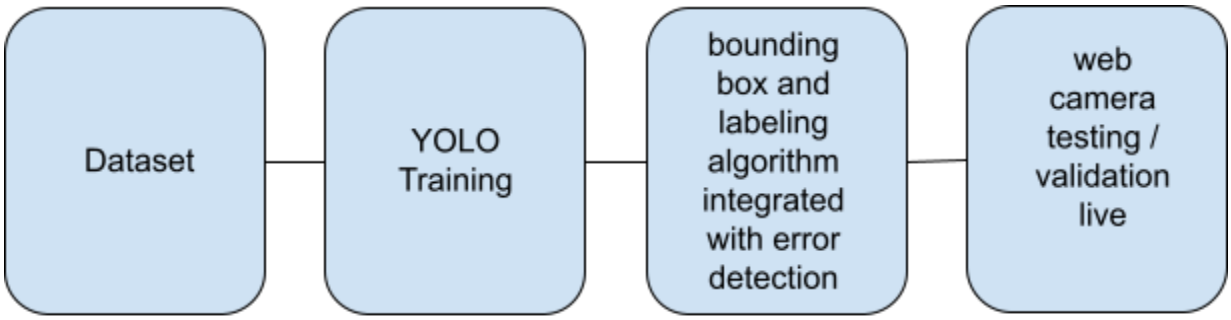
4. 2D Camera Subsystem Report

4.1. Subsystem Introduction

The 2D camera subsystem is responsible for taking a picture of the organized part box for localizing and labeling the parts in the image. This cheap camera subsystem is integrated with the AI and error detection subsystems to detect PVC parts in the organized part bucket.

4.2. Subsystem Details

Figure 28: 2D camera block model



I am using a webcam attached to my computer to localize and label the objects within the image. The camera will be included within an algorithm for which the bounding boxes and labels will be added to the image itself. A key feature of the 2D camera subsystem is its ability to take a video and capture an image, localize and label the objects within the image and send that to the other subsystems. I incorporated a webcam into my subsystem to feed images to my algorithm and create bounding boxes and label them as such. The webcam communicates with the AI subsystem as well as the error detection subsystem. The webcam requires a USB input to Windows 10, Mac OS 10.7 and later, or Linux Kernel 11 SP2. I have selected to use yolo for training purposes because it creates the bounding boxes within milliseconds, it is the most accurate providing a 80-90% accuracy, and is very compatible with most web-cameras.

4.3. Subsystem Validation

The 2D camera subsystem was validated to meet 90% accuracy on PVC parts dataset using the created model. I have also reused the data set that the group has created and manually labeled 400 images that were used for training. I used Labeling in python to label each of the 400 images (i.e. Figure 29). Labeling takes an image and imports it into the program where you draw a rectangle around the object/objects you need to label in the image and you label it. After you have labeled the images with the rectangles drawn on it it creates that into a .txt file after the original image as shown in Figure 30. The .txt file contains the object labeled number as well as the area of the bounding box for that object as shown in Figure 31. The contents of the folder with the images and the corresponding .txt file are then imported into yolo for training. I successfully trained the dataset with Yolov5 and had the accuracy graph as shown in Figure 33.

In Appendix 4.5.1, you can see that it takes in the training of the data set and the weights / configuration paths call on the object labels within a class file and bases the prediction of the webcam object outputs based on the yolo training images. Also, in Appendix 4.5, it shows a way to see the output bounding boxes on the webcam image as well as the label and accuracy as shown in Figure 32. This algorithm was successfully executed with 90% accuracy and in less than 0.5 seconds validating the subsystem. As seen in Figure 33, the accuracy curve over 140 epochs shows that it reaches approximately 90% in an efficient manner perfect for validating this system. I have validated using the model accuracy and localization graphs. The accuracy graph for classification is created by using the images and calculating the number of hits vs. misses. The localization is a comparison of the box that I drew on the image to the box the algorithm drew on the image and projects the number based on the overlap of the two boxes (i.e. if the box the training drew was the same as the box I drew on the image the accuracy would be 100%, but if it were 1% bigger/smaller than it would be 99% accurate). I set the threshold for detection to be 50% which eliminated guessing incorrectly on unknown objects and allowed for better success on testing and validation. I also tested many edge cases to make sure that the system is fully functioning and it did act as expected in each case. More images of validating edge cases can be found in the error detection validation as they were done during integration.

Figure 29: Example Image from Dataset

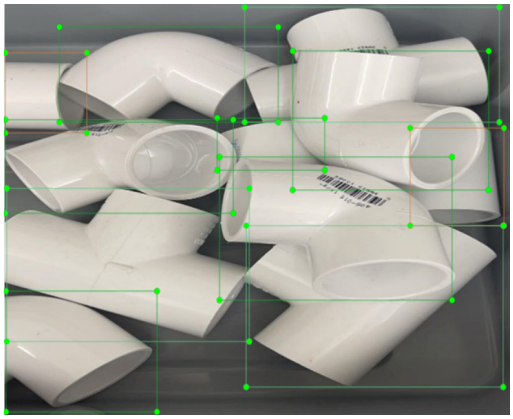


Figure 30: Folder containing labeled images

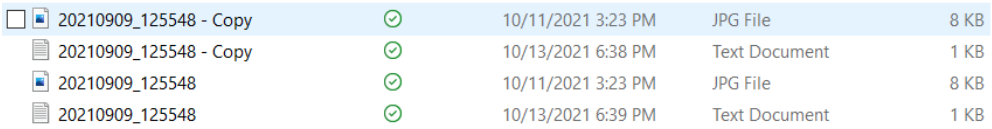
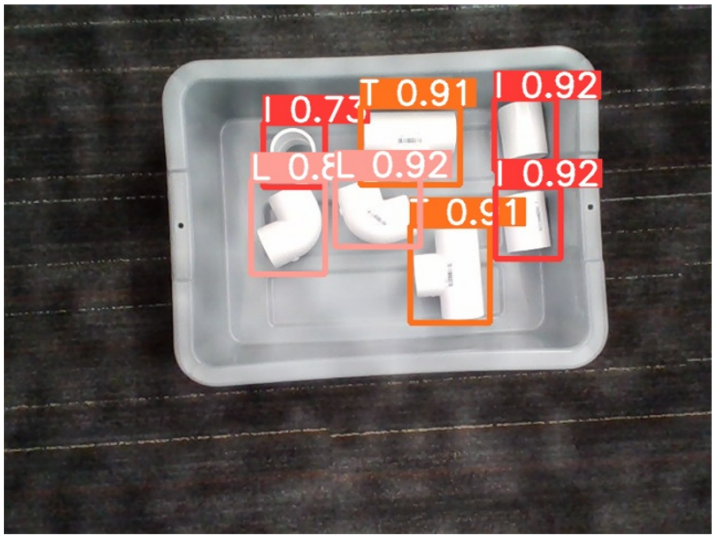


Figure 31: File_000.txt Contents

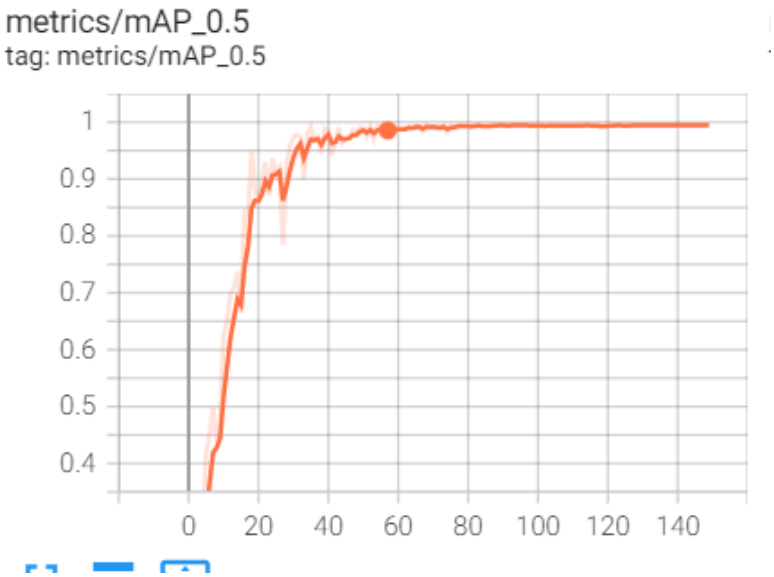
```
0 0.340608 0.476314 0.140873 0.087550
1 0.576389 0.560516 0.184524 0.156746
2 0.697586 0.398686 0.384590 0.190724
```

Figure 32: Validation of Working Algorithm




```
PS C:\Users\hanna\yolov5-final\yolov5> python final_code.py
data/images/image.jpg written!
[ WARN:0] global C:\Users\runneradmin\AppData\Local\Temp\pip-req-build-sn_xpupm\opencv\modules\videoio\src\cap_msmf.cpp (438)
`anonymous-namespace'::SourceReaderCB::~SourceReaderCB terminating async callback
YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 3 Is, 2 Ls, 2 Ts, Done. (0.242s)
```

Figure 33: Model Accuracy Graph



4.4. Subsystem Conclusion

The subsystem accurately draws the bounding boxes and label outputs for the PVC dataset. The system is integrated with an error detection system. The 2D camera subsystem is essential since it localizes and labels the final parts bin that will be sent for further use in the company. I have validated using the model accuracy and localization graphs.

4.4.1. Problems encountered

I have run into several problems throughout the semester such as raspberry pi not being able to download opencv. I ultimately had to cut out the raspberry pi in the system due to this issue and use a webcam instead. This did not affect the system as a whole since the code was written in a way to use many different camera options. I chose yolov5 since the compatibility of it with a raspberry pi was great, however due to opencv crashing the pi on download I was unable to successfully run my algorithm on the pi. Due to the system not functioning properly in the first semester, I did restart the whole system over again in the final semester allowing for validation to be done simultaneously with integration of error detection.

4.5 Appendix

4.5.1 Bounding box and labeling code

```
import argparse
import os
import platform
import sys
from pathlib import Path
import cv2
import numpy as np
import torch
import torch.backends.cudnn as cudnn
def run(weights=ROOT / 'best.pt',  # model.pt path(s)
        source=ROOT / 'data/images',  # file/dir/URL/glob, 0 for webcam
        imgsz=640,  # inference size (pixels)
        conf_thres=0.50,  # confidence threshold
        iou_thres=0.45,  # NMS IOU threshold
        max_det=1000,  # maximum detections per image
        device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=False,  # show results
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=None,  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=3,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,  # use OpenCV DNN for ONNX inference
        return_result=True  # return the prediction result
        ):
    source = str(source)
    save_img = not nosave and not source.endswith('.txt')  # save inference images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
    webcam = source.isnumeric() or source.endswith('.txt') or (is_url and not is_file)
    if is_url and is_file:
        source = check_file(source)  # download
    # Load model
    w = str(weights[0] if isinstance(weights, list) else weights)
    classify, suffix, suffixes = False, Path(w).suffix.lower(), ['.pt', '.onnx', '.tflite', '.pb', '']
    check_suffix(w, suffixes)  # check weights have acceptable suffix
    pt, onnx, tflite, pb, saved_model = (suffix == x for x in suffixes)  # backend booleans
    stride, names = 64, [f'class{i}' for i in range(1000)]  # assign defaults
    if pt:
```

```
    model = torch.jit.load(w) if 'torchscript' in w else attempt_load(weights, map_location=device)
    stride = int(model.stride.max())  # model stride
    names = model.module.names if hasattr(model, 'module') else model.names  # get class names
 # Loading data
if webcam:
    view_img = check_imshow()
    cudnn.benchmark = True  # set True to speed up constant image size inference
    dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt)
    bs = len(dataset)  # batch_size
else:
    dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt)
    bs = 1  # batch_size
vid_path, vid_writer = [None] * bs, [None] * bs
# Run inference
if pt and device.type != 'cpu':
    model(torch.zeros(1, 3, *imgsz).to(device).type_as(next(model.parameters())))  # run once
dt, seen = [0.0, 0.0, 0.0], 0
return_outputs = {"I": 0, "L": 0, "T": 0}
for path, img, im0s, vid_cap, s in dataset:
    t1 = time_sync()
    if onnx:
        img = img.astype('float32')
    else:
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float()  # uint8 to fp16/32
    img /= 255  # 0 - 255 to 0.0 - 1.0
    if len(img.shape) == 3:
        img = img[None]  # expand for batch dim
    t2 = time_sync()
    dt[0] += t2 - t1
        # Stream results
        im0 = annotator.result()
        if view_img:
            cv2.imshow(str(p), im0)
            cv2.waitKey(1)  # 1 millisecond
        # Save results (image with detections)
        if save_img:
            if dataset.mode == 'image':
                cv2.imwrite(save_path, im0)
            else:  # 'video' or 'stream'
                if vid_path[i] != save_path:  # new video
                    vid_path[i] = save_path
                    if isinstance(vid_writer[i], cv2.VideoWriter):
                        vid_writer[i].release()  # release previous video writer
                    if vid_cap:  # video
                        fps = vid_cap.get(cv2.CAP_PROP_FPS)
                        w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                        h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                    else:  # stream
                        fps, w, h = 30, im0.shape[1], im0.shape[0]
                        save_path += '.mp4'
                    vid_writer[i] = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                vid_writer[i].write(im0)
if return_result:
```

```
        return return_outputs
    # Print results
    t = tuple(x / seen * 1E3 for x in dt)  # speeds per image
    LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per image at shape {(1, 3, *imgsz)}'
% t)
    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ''
        LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
    if update:
        strip_optimizer(weights)  # update model (to fix SourceChangeWarning)
def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'best.pt', help='model path(s)')
    parser.add_argument('--source', type=str, default=ROOT / 'data/images', help='file/dir/URL/glob, 0 for webcam')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.25, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    parser.add_argument('--return_result', action='store_true', help='return results')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(FILE.stem, opt)
    return opt
def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))
if __name__ == "__main__":
    opt = parse_opt()
    main(opt)
#capture image and save to data/image path
#if path has images in it delete and save new image in there
#Done - call detect on new image
#clears file
for filename in os.listdir("data/images"):
    os.remove(os.path.join("data/images", filename))
```

```
img_name = "data/images/image.jpg"
cam = cv2.VideoCapture(0)
cv2.namedWindow("test")
img_counter = 0
ret, frame = cam.read()
if not ret:
    print("failed to grab frame")
else:
    cv2.imwrite(img_name, frame)
    print("{} written!".format(img_name))
cam.release()
cv2.destroyAllWindows()
```

5. Error Detection & Handling Subsystem Report

5.1 Subsystem Introduction

The error detection and handling subsystem is designed to alert the user in the event of any noticeable problems during operation of the system. It will also ensure that the order requirement has been met by making sure that there are the correct number of parts in each bin. If there are any missing or excess parts in the bucket then the system will immediately detect that error and either handle the error by adding or removing the missing or excessive part from the “CURRENT BIN” to the “PARTS BIN”. A separate checklist has also been created to handle edge cases. The error detection and handling subsystem was tested to confirm that it can detect and handle errors effectively.

Figure 34: Error detection & handling block model

5.2 Subsystem Details/Design

The main purpose of this subsystem was to provide a check for the user to determine if their order and all the parts within the order were received and picked up correctly because if not then it can decrease efficiency and halt production. In order to ensure that the order requirement has been met, the system compares the total number of parts detected in an image from the 2D camera subsystem with the user’s input to check if we have a missing or extra part in the bucket. It calculates it by subtracting the user’s input from the count provided by the model and if the difference is greater than zero then it means that the system is missing an object, and if not then it has excess objects in the bin. It is also assumed that there are two buckets called the “PARTS BIN” and the “CURRENT BIN”. The “PARTS BIN” contains infinitely many parts in it and the “CURRENT BIN” only contains the parts needed to meet a specific order requirement. Therefore after comparing the user’s input with the model’s output if the system detects that we have a missing or excess part then it at first states the number of parts missing or in excess. Then it will handle the missing parts by adding the number of missing parts to the “CURRENT BIN” from the “PARTS BIN" and handle the excess parts by removing it from the “CURRENT BIN” to the “PARTS BIN”. When the user’s input is the same as the model’s output then it

means that the order requirement has been met. In that case a statement will be printed out stating “Order requirement has been met, move to the staging area. However, image processing will be performed twice to confirm that the order requirement has actually been met thus it can safely be moved to the staging area. A part of the code that performs this operation has been provided below (Figure 35) with comments for better understanding.

Figure 35: Part of the error detection code

```
def error_detection(image_path, true_count, dataf, a):

    thresh = 0.50

    # run model detection
    count = run(weights="runs/train/exp/weights/best.pt",
                imgsz=[640, 640],
                source=image_path,
                return_result=True,
                conf_thres=thresh,
                save_txt=True)

    count1=count.copy()
    dataf.append(count1)
    print(count)

    image = PImage.open(image_path)
    (im_width, im_height) = image.size
    image_np = np.array(image.getdata()).reshape((im_height, im_width, 3)).astype(np.uint8)
```

```
    c=0
    cc=a
    dd=0
    for obj in count.keys():
        if count[obj] != true_count[obj]:
            diff = true_count[obj]-count[obj]
            if diff>=0:
                print("Error! {} {} is/are missing".format(diff,obj))
                dataf.append("Error! {} {} is/are missing".format(diff,obj))
                # count[obj] = count[obj]+diff

            else:
                print("Error! {} {} is/are excess".format(diff*-1,obj))
                dataf.append("Error! {} {} is/are excess".format(diff*-1,obj))
                # count[obj] = count[obj] - (diff*-1)

        else:
            # count=true_count for all the categories
            c=c+1
```

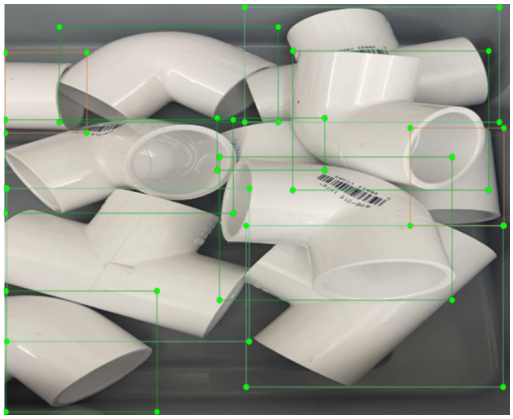
```
    for obj in count.keys():

        if count[obj] != true_count[obj]:
            diff = true_count[obj]-count[obj]
            if diff>0:
                #print("Error! {} {} are missing".format(diff,obj))
                #count[obj] = count[obj]+diff
                print(" {} {} added to the CURRENT BIN from the PARTS BIN ".format(diff,obj))
                dataf.append(" {} {} added to the CURRENT BIN from the PARTS BIN ".format(diff,obj))
            else:
                #print("Error! {} {} are excess".format(diff*-1,obj))
                #count[obj] = count[obj] - (diff*-1)
                print(" {} {} removed from the CURRENT BIN to the PARTS BIN".format(diff*-1,obj))
                dataf.append(" {} {} removed from the CURRENT BIN to the PARTS BIN".format(diff*-1,obj))
```

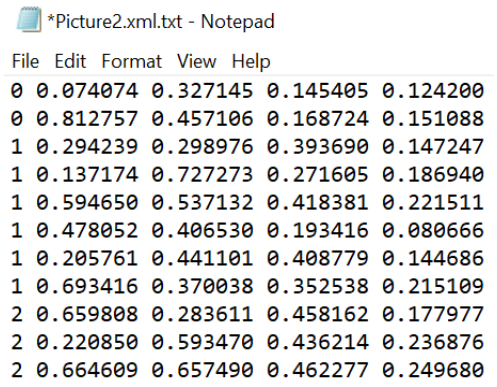
```
if c==len(count):
    # Here basically we are checking if the length of the count is 4 and if it is 4 then it means that the
    # true count and count is same for all the 4 parts. In that case we can move to the staging area
    print("Entire order is here, move to the staging area")
    dataf.append("Entire order is here, move to the staging area")
    # cc=a=0, so now cc=0+1=1. Then it will perform image processing again in the next cell
    cc=cc+1

return (count, image_np, cc, dd) # ignore dd but basically we are returning these values from the error detection function
```

```
            if c1==1:
                print("\n")
                print("Performing image processing again")
                print("\n")
                del dataf[-1] #deleting wrong values ie values before double check so that we only get the latest values on the txt file
                del dataf[-1]
                del dataf[-1]
                del dataf[-1]
                del dataf[-1]
                del dataf[-1]
                dataf.append(("Performed image processing again"))
                dataf.append("I needed: {}".format(t_count["I"]))
                dataf.append("L needed: {}".format(t_count["L"]))
                dataf.append("T needed: {}".format(t_count["T"]))

                count, image_np, c3, d1 = error_detection(im_path, t_count, dataf, a) # rechecking, basically performing image processing again
```

A separate error detection checklist has also been created to check if certain errors can be fixed by shaking the bin, rearranging the parts or changing the camera angle. In figure 36, I included a manually labeled image of heavily occluded parts that might throw an error. I also included the .txt file for reference. A more detailed description of such errors can be found in the subsystem validation section.

Figure 36: Labeled heavily occluded parts in a bucket & its txt file is included:

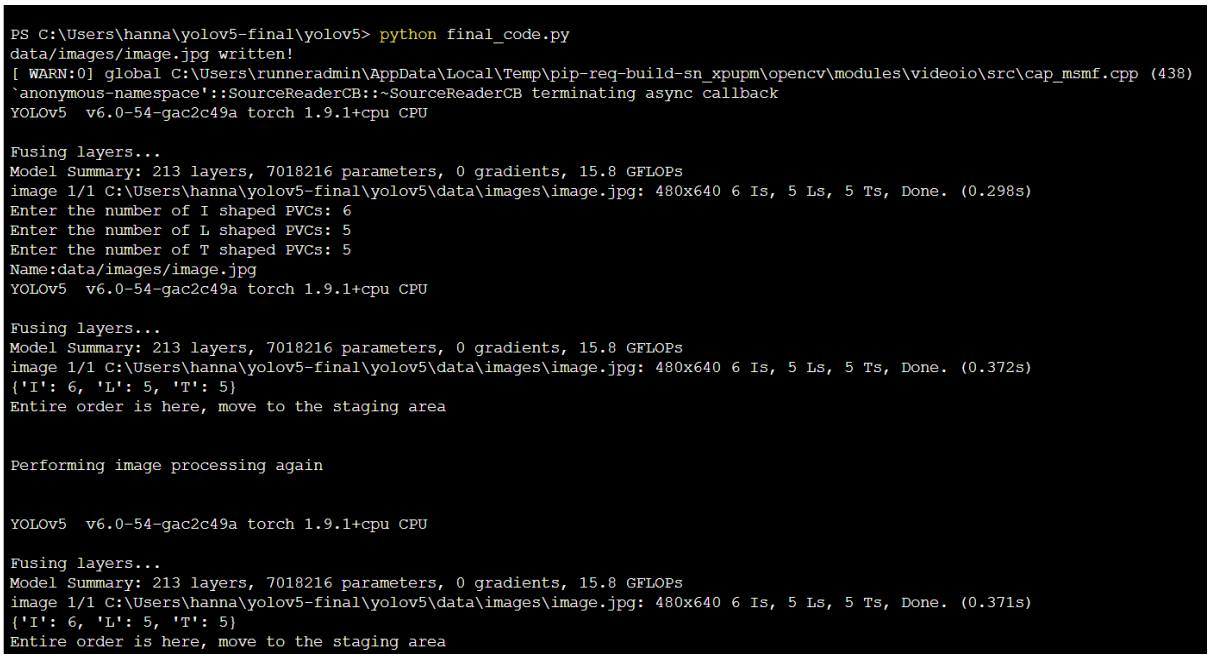




5.3 Subsystem Validation

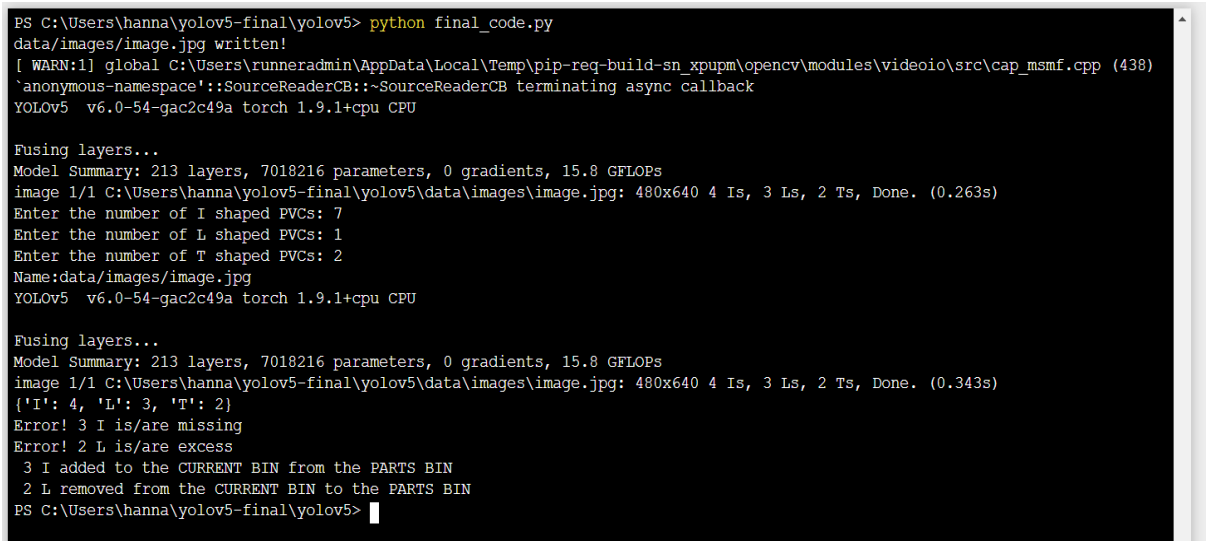
The error detection and handling subsystem has been thoroughly validated using different types of order requirements arranged in a bucket. If the order requirement has been met then the system performs image processing twice and then moves to the staging area. If the system did encounter an error then it logged that error in the console to make sure the operator is aware of the error. However, an error did not stop the program from running, instead it handled those errors by adding or removing the missing or excessive part from the “CURRENT BIN” to the “PARTS BIN”. The algorithm used to detect and handle errors were successfully executed with more than 95% accuracy validating the subsystem. I have also manually labeled around 100 images with heavily occluded and hidden parts using labelImg to see what kind of error it throws and if that can be fixed by shaking the bin, rearranging the parts or changing the camera angle. An example of such an image has been provided earlier in figure 36. The result/validation of the error detection and handling algorithm are explained below in detail with screenshots.

Figure 37: Order requirement met console:



The image shows that the user asked for 6 I-shaped, 5 L-shaped and 5 T-shaped PVCs and the model also detected 6 I-shaped, 5 L-shaped and 5 T-shaped PVCs. Thus, the order requirement has been met.

Figure 38: Missing or extra part error:



The image shows that the user asked for 7 I-shaped, 1 L-shaped and 2 T-shaped PVCs and the model also detected 4 I-shaped, 3 L-shaped and 2 T-shaped PVCs. Thus, we have 3 I-shaped PVC missing and 2 L-shaped PVC is excess. Therefore, 3 I-shaped will be added to the “CURRENT BIN” from the “PARTS BIN” and 2 L-shaped will be removed from the “CURRENT BIN” to the “PARTS BIN”.

Figure 39: Edge case and how it can be handled by rearranging the parts:

Before rearranging the parts:


```
Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 5 Is, 5 Ls, 5 Ts, Done. (0.244s)
Enter the number of I shaped PVCs: 6
Enter the number of L shaped PVCs: 5
Enter the number of T shaped PVCs: 5
Name:data/images/image.jpg
YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 5 Is, 5 Ls, 5 Ts, Done. (0.231s)
{'I': 5, 'L': 5, 'T': 5}
Error! 1 I is/are missing
 1 I added to the CURRENT BIN from the PARTS BIN
PS C:\Users\hanna\yolov5-final\yolov5>
```

On the top left corner, an I-shaped PVC is hidden under an L-shaped PVC. Therefore, the model is detecting that an I-shaped PVC is missing

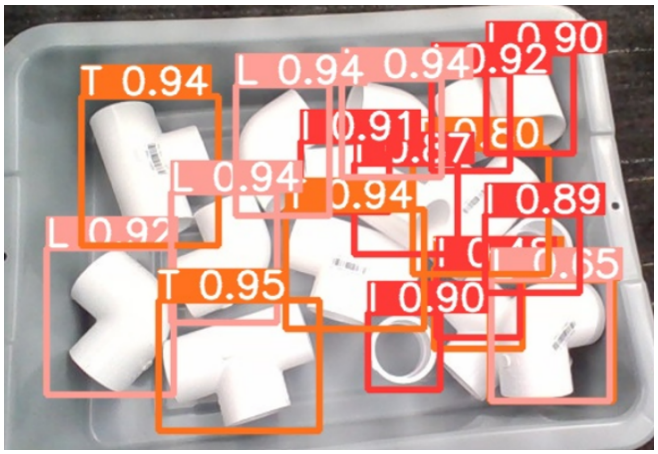
After rearranging the parts:

```
Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 6 Is, 5 Ls, 5 Ts, Done. (0.291s)
Enter the number of I shaped PVCs: 6
Enter the number of L shaped PVCs: 5
Enter the number of T shaped PVCs: 5
Name:data/images/image.jpg
YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 6 Is, 5 Ls, 5 Ts, Done. (0.270s)
{'I': 6, 'L': 5, 'T': 5}
Entire order is here, move to the staging area


Performing image processing again


YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 6 Is, 5 Ls, 5 Ts, Done. (0.246s)
{'I': 6, 'L': 5, 'T': 5}
Entire order is here, move to the staging area
PS C:\Users\hanna\yolov5-final\yolov5>
```

After rearranging all the parts are now visible. Therefore, the system was able to detect all the parts and then move to the staging area.

Figure 40: Edge case and how it can be handled by shaking the bucket:

Before shaking the bucket:



```
PS C:\Users\hanna\yolov5-final\yolov5> python final_code.py
data/images/image.jpg written!
[ WARN:0] global C:\Users\runneradmin\AppData\Local\Temp\pip-req-build-sn_xpupm\opencv\modules\videoio\src\cap_msmf.cpp (438)
`anonymous-namespace'::SourceReaderCB::~SourceReaderCB terminating async callback
YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 7 Is, 5 Ls, 6 Ts, Done. (0.226s)
Enter the number of I shaped PVCs: 6
Enter the number of L shaped PVCs: 5
Enter the number of T shaped PVCs: 5
Name:data/images/image.jpg
YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 7 Is, 5 Ls, 6 Ts, Done. (0.344s)
{'I': 7, 'L': 5, 'T': 6}
Error! 1 I is/are excess
Error! 1 T is/are excess
 1 I removed from the CURRENT BIN to the PARTS BIN
 1 T removed from the CURRENT BIN to the PARTS BIN
PS C:\Users\hanna\yolov5-final\yolov5>
```

The image is heavily occluded and therefore, the model is detecting an excess I and T-shaped PVC.

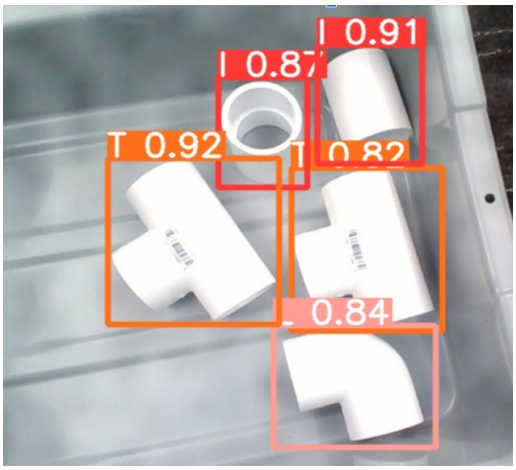
After shaking the bucket:


```
PS C:\Users\hanna\yolov5-final\yolov5> python final_code.py
data/images/image.jpg written!
[ WARN:0] global C:\Users\runneradmin\AppData\Local\Temp\pip-req-build-sn_xpupm\opencv\modules\videoio\src\cap_msmf.cpp (438)
`anonymous-namespace'::SourceReaderCB::~SourceReaderCB terminating async callback
YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 6 Is, 5 Ls, 5 Ts, Done. (0.239s)
Enter the number of I shaped PVCs: 6
Enter the number of L shaped PVCs: 5
Enter the number of T shaped PVCs: 5
Name:data/images/image.jpg
YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 6 Is, 5 Ls, 5 Ts, Done. (0.339s)
{'I': 6, 'L': 5, 'T': 5}
Entire order is here, move to the staging area


Performing image processing again


YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 6 Is, 5 Ls, 5 Ts, Done. (0.351s)
{'I': 6, 'L': 5, 'T': 5}
Entire order is here, move to the staging area
PS C:\Users\hanna\yolov5-final\yolov5>
```

After shaking the bucket, the system was successfully able to detect all the parts.

Figure 41: Edge cases handled by changing the camera angle:

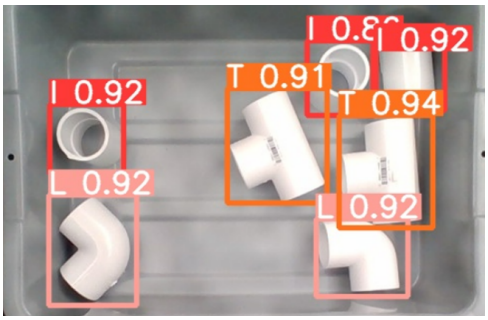
Before changing the camera angle:




```
PS C:\Users\hanna\yolov5-final\yolov5> python final_code.py
data/images/image.jpg written!
[ WARN:0] global C:\Users\runneradmin\AppData\Local\Temp\pip-req-build-sn_xpupm\opencv\modules\videoio\src\cap_msmf.cpp (438)
`anonymous-namespace'::SourceReaderCB::~SourceReaderCB terminating async callback
YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 2 Is, 1 L, 2 Ts, Done. (0.311s)
Enter the number of I shaped PVCs: 3
Enter the number of L shaped PVCs: 2
Enter the number of T shaped PVCs: 2
Name:data/images/image.jpg
YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 2 Is, 1 L, 2 Ts, Done. (0.340s)
{'I': 2, 'L': 1, 'T': 2}
Error! 1 I is/are missing
Error! 1 L is/are missing
 1 I added to the CURRENT BIN from the PARTS BIN
 1 L added to the CURRENT BIN from the PARTS BIN
PS C:\Users\hanna\yolov5-final\yolov5>
```

The system is detecting a missing I-shaped and L-shaped PVC because the entire bucket is not under the camera frame.

After changing the camera angle:




```
PS C:\Users\hanna\yolov5-final\yolov5> python final_code.py
data/images/image.jpg written!
[ WARN:1] global C:\Users\runneradmin\AppData\Local\Temp\pip-req-build-sn_xpupm\opencv\modules\videoio\src\cap_msmf.cpp (438)
`anonymous-namespace'::SourceReaderCB::~SourceReaderCB terminating async callback
YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 3 Is, 2 Ls, 2 Ts, Done. (0.228s)
Enter the number of I shaped PVCs: 3
Enter the number of L shaped PVCs: 2
Enter the number of T shaped PVCs: 2
Name:data/images/image.jpg
YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 3 Is, 2 Ls, 2 Ts, Done. (0.358s)
{'I': 3, 'L': 2, 'T': 2}
Entire order is here, move to the staging area


Performing image processing again


YOLOv5  v6.0-54-gac2c49a torch 1.9.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
image 1/1 C:\Users\hanna\yolov5-final\yolov5\data\images\image.jpg: 480x640 3 Is, 2 Ls, 2 Ts, Done. (0.345s)
{'I': 3, 'L': 2, 'T': 2}
Entire order is here, move to the staging area
PS C:\Users\hanna\yolov5-final\yolov5>
```

After changing the camera angle, the system identified all the parts in the bucket and moved to the staging area.

5.4. Subsystem Conclusion

Each part of the subsystem was shown to work correctly. The system works with the 2D camera subsystem to identify the parts in an image, detect the errors and handle them using the error detection algorithm. I have also tested and validated using various edge cases and everything seems to work as expected.

3D Occluded Object Detection System

Hannah Hillhouse, Tony Jeong, Evan Kolin, Samiha Elahi

Integrated System Results

REVISION – 2
3 December 2021

Change Record

Rev.	Date	Originator	Approvals	Description
0	12/3/2021	Evan Kolin		Final Submission 404

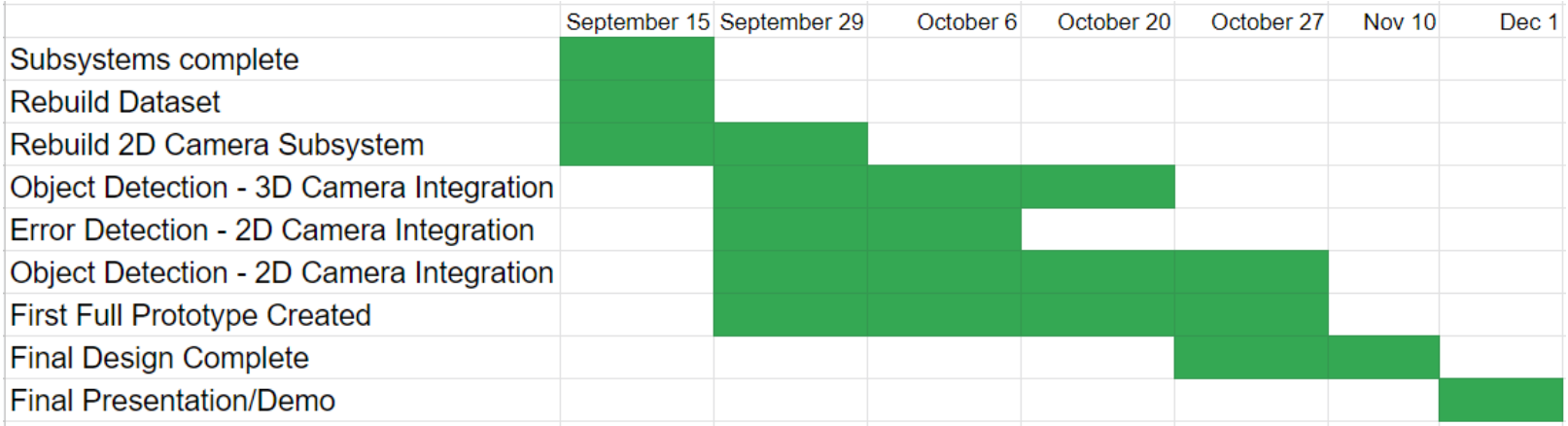
1. Overview

The system is broken into two main parts, the 3D and 2D camera systems. Both systems use object detection AI, with the difference being the 3D system usually has more parts to look at and also cares about the depth of each object in the bin.
The reason our integrated system remains as two separate parts is because we were supposed to create the system as if we had access to a robotic arm that would move the selected parts from the 3D bin into the 2D bin. From the beginning we never had access to the robotic arm though, so we instead pick and place parts by hand, but have designed the outputs so that *if* we had the arm it would connect the two systems together.

2. Execution

Table 1 shows our execution plan. This plan was updated and changed over the course of the semester to reflect our progress. Some things finished ahead of schedule so we moved them up, and other things required more time so we delayed the completion date. In 403, we focused mainly on the subsystems whereas 404 we focused on the integration of all systems. Due to setbacks on the 2D subsystem in 403, validation was done simultaneously with integration of 2D camera subsystem and error detection.

Table 5. Execution Plan



3. Validation

3.1 3D System Validation

The validation for the 3D system is all done within the validation for the 3D camera subsystem. This is because the AI is stored inside of the subsystem, and thus it's outputs are representative of the integrated system. In section 4. Performance, the validation results are also listed.

3.2 2D System Validation

The 2D validation was performed on a quantitative pass/fail scale based on whether or not the test was successful completely or not. The 2D system did meet all of the requirements and specifications during validation.

4. Performance

The accuracy for the two integrated systems are shown in the table below. The integrated system results for the 2D camera and error detection got well within the requirements set and performed very efficiently. The time required to detect the objects and run error detection was below what we expected time.

Table 6. Validation Results

System	Validation	Target	Result
3D System	Classification Accuracy	90%	92%
	Localization Accuracy	90%	90%
	Order Completion	Fulfills order 100% of time, assuming the pieces requested are in stock	Tested over 50 iterations, always completed or accurately alerts the user that the piece is out of stock.
2D System	Classification Accuracy	90%	92%
	Localization Accuracy	90%	90%
	Time per Image Detection	Less than 5 seconds	0.3 seconds

5. Conclusion

5.1 Limitations

5.1.1 Dataset Limitations

There is theoretically an upper limit on the number of things that the AI can learn, depending on how much available storage the user has. In semester 1, we were heavily limited by the storage on Evan's computer to hold a dataset. In Semester 2, we used the google drive to hold our data and were able to create a much larger dataset.

5.1.2 Environmental Conditions

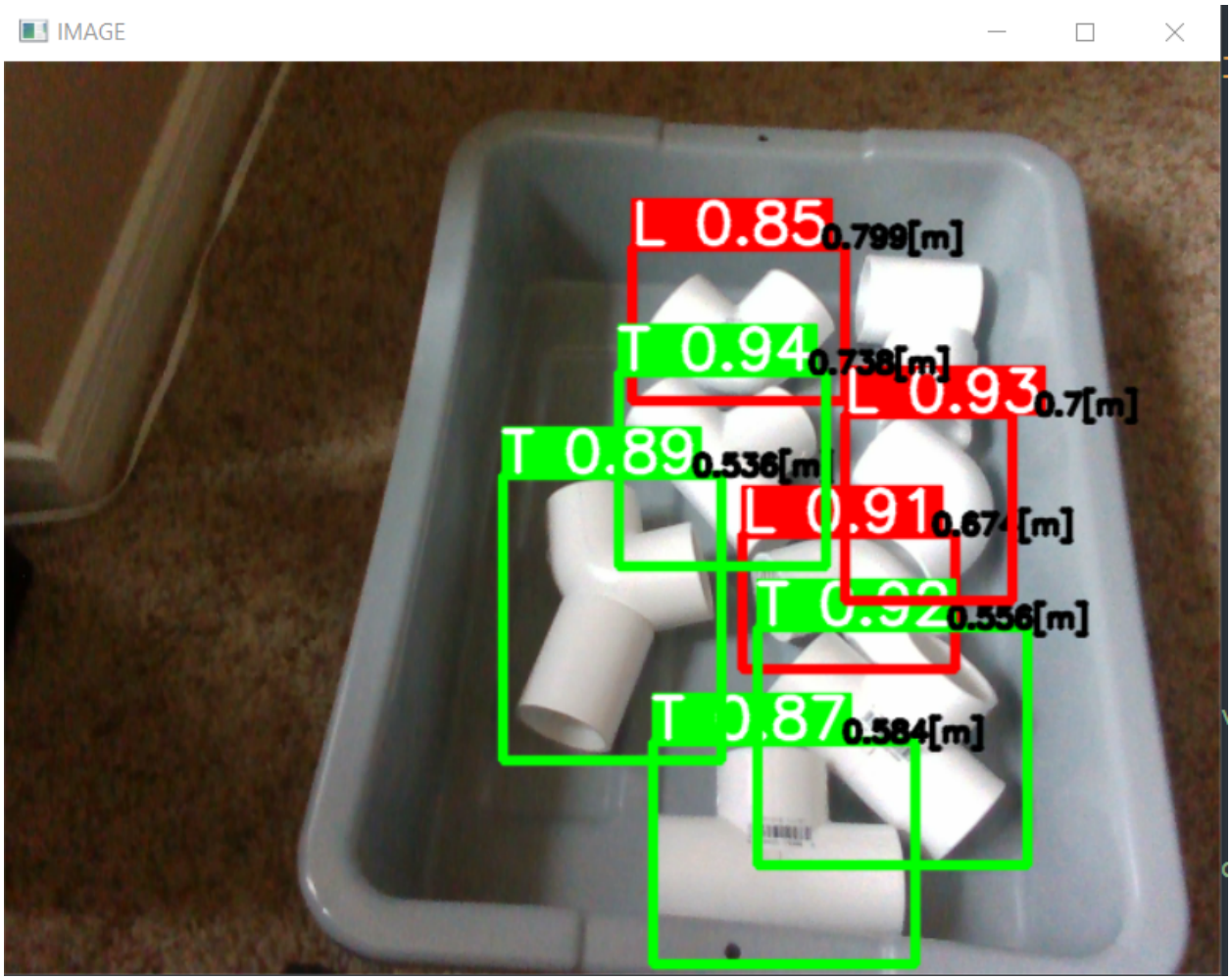
Due to the shiny nature of PVC, it would frequently reflect light strongly if detection was being done in a very bright environment like outside. This would result in a large drop in detection capabilities. The biggest problem was that when lit strongly, I shape piece detection confidence would drop below our desired threshold of about 70% for the 3D system and 50% for the 2D system, resulting in them not being considered for localization. When used inside under bright

lights the system works fine, as the dataset was created in these conditions to mimic a factory setting, the desired environment for this system to run in.

5.1.3 Convoluted Configurations

There were some cases where the program detected wrong pieces. For example in figure 1, the user asked for 4 pieces of T shaped PVC pipes while there are only 3 pieces of them in the bucket. However, it actually showed the result where it found all 4 pieces. This happened because the I piece and L piece are combined in shape which really looks like a T piece in real life as well.

Figure 42. Convoluted configuration example



5.1.4 Model limitations

When designing the model, we used Tensorflow. While Tensorflow has amazing tools for building up classification and localization, it does not inherently support any multi-object detection. This requires use of an external algorithm like YOLO (You only look once) or SSD (single shot detection). Due to this, we decided that we would run the model through YOLO training. Unfortunately, due to time constraints, we were not able to complete the YOLO training

with our model and had to use a pretrained model, that was then trained more specifically on our data.

5.2 Impact

The main goal of this project was to increase efficiency while decreasing cost to the company and we reached our goal. Some impacts of this project include:

1) Time - The amount of time spent by manual labor to pick and place the parts in their corresponding bins is more than that of our system (i.e. increasing customer satisfaction)
2) Money - You will decrease the amount of money spent on manual labor to pick and place the parts therefore allowing money to be spent elsewhere in the company.
3) Economy - Factory workers will lose their jobs picking and placing, but new jobs maintaining the systems will appear, which will likely be higher paying jobs due to the required technical knowledge.