Autonomous Stocktaking Solution

Vision Solution Component

4.1.2022

Nishant Wadhwani
Machine Learning Engineer

Vision Solution-Box Counting Algorithm

Introduction

Stocktaking is a routine activity which is carried out at Warehouses and storage facilities to keep track of inward and outward inventory. Wipro is developing an automated Stocktaking solution which could run on multiple form factors like drones, robots, AGVs etc(here known as Agents). This solution aims to reduce the shutdown time at warehouse facilities and decrease the effort required to carry out these tasks significantly. This document specifically explains about the computer vision aspects related to the overall solution development for the warehouse stock-taking.

Background

The solution will be deployed in a warehouse environment where the end user will raise a task request via the WMS system or a custom UI application. The system will create a task request, schedule it, trigger the task on time and assign it to the identified agent. Agent (for the current requirement is a quadcopter/drone) is provided with the available map / layout information of the warehouse and uses this information navigate to the aisle of interest in order to carry out the operation. Once it reaches the aisle, the agent will proceed to the racks to be scanned. Agent will perform the required localization procedures and carry out the data collection, which includes image capture and barcode scanning of the required pallets with SKUs. The collected data is shared with the vision server where the solution algorithm runs, and the end result is generated - which is the total count of the SKU boxes on the particular rack, and then passed to the user in a pre-defined format.

Counting number of boxes stacked on a pallet at warehouse is time consuming and labor-intensive processes. Periodically, these boxes need to be counted to keep check the inflow/outflow of them at warehouses. Variation in box dimensions increase the complexity of the counting process. For each sku, stacking pattern is predefined and accordingly boxes are stacked. Stacking pattern can vary from one level to another level for the same sku. Hence, it is needed to automate the box counting process to save time and also to get the accurate number of boxes. Idea here discussed is given a orthogonal image/images of a pallet with boxes, count the number of boxes, applying computer vision techniques.

Scope and Objectives

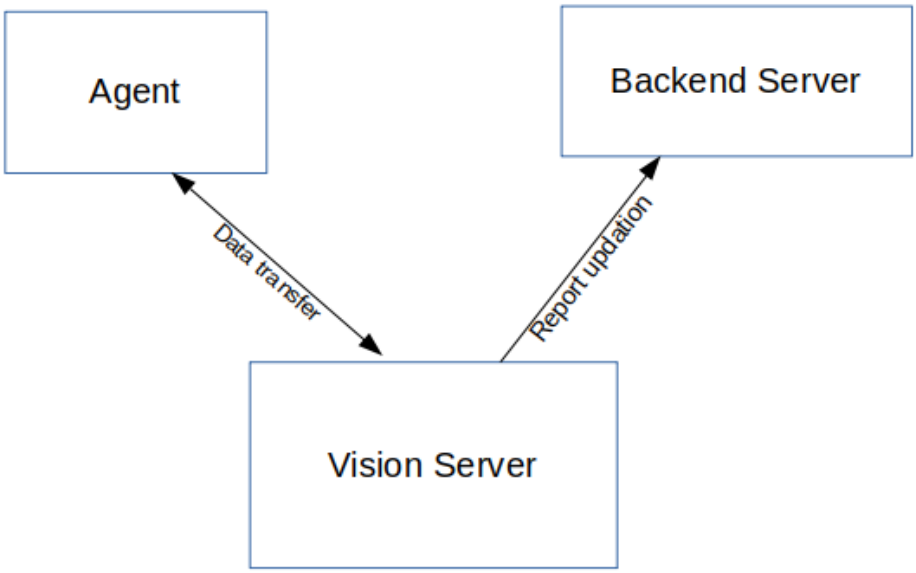
i. The current scope of this design document is to develop and integrate the solution algorithms which will account for counting the number of boxes stacked on a pallet inside a warehouse.
ii. The main challenge in counting is getting an estimate of hidden boxes, after a few boxes have been removed.Based on the positioning of visible boxes and stacking pattern, it's count is decided. Hence, it is needed to automate the box counting process to save the time and also to get the accurate number of boxes.
iii. Another constraint for the count back process required is that an aruco marker should be there on the pallet for getting a reference point for pose estimation of all the SKUs with respect to camera which will help in level wise segregation of the pallet both height wise as well as depth wise.
iv. After getting the box count, the next objective is to get the SKU Identification done in order to get the detailed information regarding the product using either barcode or optical character recognition of the text written on the SKU carton box.

High Level Design Overview

The current overall design has been developed with the long-term product view of the solution – Box Counting Algorithm and SKU Identification.
This document describes the Design for the Software System that runs in each of the many Agents that can be used to perform the warehouse operations and the Interfaces to the Central Server.

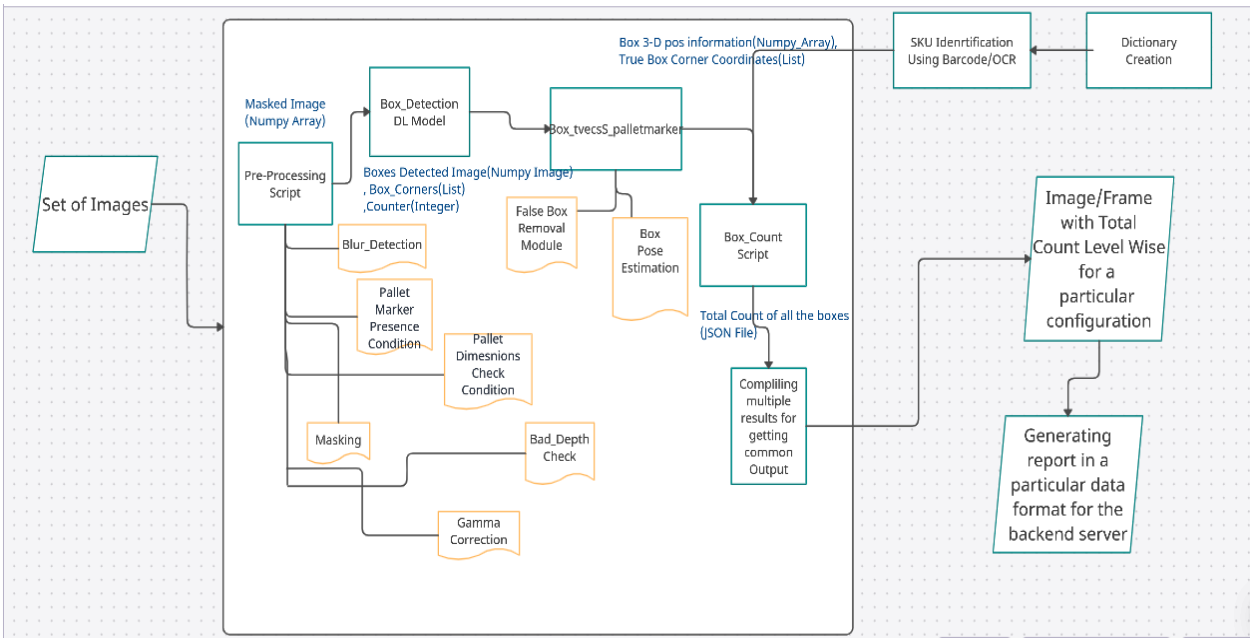
System Overview and Design Consideration

The system objectives from the designer's point of view should be stated here.
The objectives should flow from the user requirements;
some of them might relate to design issues such as performance, memory constraints, portability, reliability, etc.,
Alternate designs, their pros & cons and criteria for selecting the design should be mentioned
The diagram shown below is the overall high-level system design for the solution. The key components are categorized based on their functionalities and the flow shows how they will be interacting with each other through using different interfaces.



Vision server Software Architecture

This below architecture describes detailed overview of the modules involved in the box-counting algorithm part of the solution algorithm:



This architecture describes detailed overview of the computer-vision solution algorithm which consists of different modules: -

1. The vision service will receive the input -which will be a set of ‘n’ images/frames captured by realsense camera module and the barcode data from the barcode scanner module because we need RGB data, depth frame together with the SKU ID and other data extracted from the frames.

Realsense Camera Node

The Realsense camera is an important component for the whole functioning. The RGBD Camera data (RGB, Depth and Point Cloud) will be required by multiple components for successful execution. Therefore, a dedicated module will continuously read the different required data and publish them to various ROS topics for consumption as required.

Barcode Scanner Node

Barcodes are used in warehouses for identifying individual SKUs, identifying an entire batch which is shrink-wrapped or even for pallet location detection. The current custom drone is configured to have a barcode scanner along with the RGBD camera (both are front facing). We will define a custom barcode scanner ROS node which publishes the detected barcode information on the corresponding topic.

2. After acknowledging the receipt of proper input data, the box-count scripts will run all the individual modules sequentially starting from the pre-processing script that will be involved in cleaning, and verifying the integrity of the data, along with some image enhancements part to eventually increase the accuracy of the solution and will be making use of only the good images to the upcoming modules. So, it includes: -

a) Blur Detection which is quite important to check for further modules, which will check if an image contains high variance, then there is a wide spread of responses, both edge-like and non-edge-like, representative of a normal in-focus image. But if there is very low variance, then there is a tiny spread of responses, indicating there are very little edges in the image. And as we know, the more an image is blurred, the less edges there are. In python it will get edge sharpness by returning the Laplacian Operator which will compute the Laplacian of the image and then returns the focus measure, which is simply the variance of the Laplacian. It will take an image in the form of a numpy array and return a boolean flag as 1 if the image is non-blur.

Non-Blurry Image

Blurry Image

b) Pallet Marker presence Condition: This will check for pallet aruco marker presence as it is used as a reference point for calculations to get translation vectors/3-D position information of each and every SKU. It will calculate the length of corners and thereby check the status of pallet marker present condition. It will take a frame in form of numpy array and returns the boolean flag to 1 if there is pallet marker present.

No Pallet Marker Present

Pallet Marker Present

c) *Pallet Dimension Check Condition*- it will check for whether the full FOV of the pallet as expected is captured or not, this is important since we don't want to lose any information/detail related to a particular pallet, it can be the side boxes or the upper-level boxes. It will take two arguments which is a list containing corner points of a particular box and marker Length of the pallet marker and therefore returns the conversion factor of pixels to cm.

Full Pallet View Visible

No Full Pallet View Visible

d) Bad Depth Check, it is for filtering out the set of frames with a bad depth frame and this needs to be checked, because we will be getting depth information of all the visible boxes present on the pallet and this will be quite helpful in the box-counting algorithm too. It will check if there is any significant noise associated within the region of a SKU, which might affect getting the depth values.

Good Depth Image

Bad Depth Image

e) Masking: - This is technically to remove all the unnecessary information from the image, and getting our region of Interest- which is all the SKUs only in the frame. Given an input image, it will mask the unwanted region, just to keep the pallet portion along-with boxes that are stacked on top of it. At the end, we will get the masked image/frames in numpy format and that will be fed further.

Unmasked Image

Masked Image

f) Gamma Correction: - It is to increase the brightness of image enhancement; this is done by gamma correction and this particular module will help in increasing our overall accuracy. It will build a lookup table mapping the pixel values [0, 255] to their adjusted gamma values. It will take an image in form of numpy array and gamma which is the parameter that can be tuned according to the environment.

Masked Image without Gamma Correction

Masked Image with Gamma Correction

At the end, we will get the masked image/frames in numpy format and that will be fed further.

Box Detection DL Model

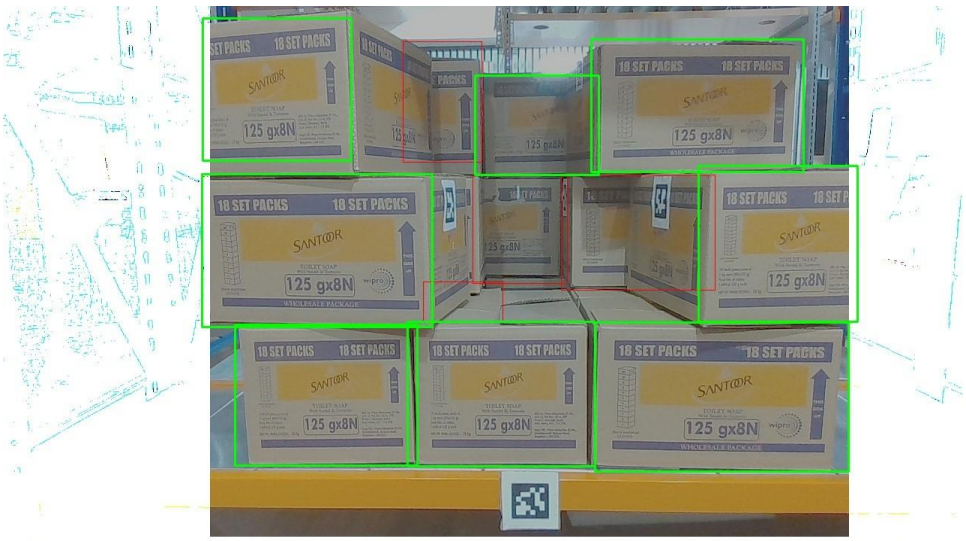
So, after the preprocessing stage, there will be the box detection deep learning module which will use Faster RCNN object detection framework to detect carton boxes or the SKU's. In this particular module, we will be getting boxes detected Image which contain the bounding boxes(Numpy Image), Box Corners and the counter for the visible boxes that are stacked on the pallet.
We have gathered 536 images for training Faster RCNN object detection model and have trained for 240000 iterations which is 450 epochs approximately and got these results:-

iter: 240000 ; Total_loss: 0.0931 (0.2337) ; loss_classifier: 0.0182 (0.0392);
loss_box_reg: 0.0055 (0.0134); loss_quad_box_reg: 0.0084 (0.0193); loss_objectness: 0.0008 (0.0113);
loss_rpn_box_reg: 0.0345 (0.1196); Atten_loss: 0.0215 (0.0308)

mAP: 0.9089 for 0.9 iou

Sample output from the box detection Deep Learning model Faster RCNN

SKU boxes pose info:-After the Box detection deep learning module, this output will be fed into box_tvecsS_palletmarker that will further remove all the false boxes i.e., side faces detected as bounding boxes plus it will also give us the box pose estimation which will be the 3-d position information of the SKU's with respect to the camera. The final output will be the Box 3-d pos info in numpy format and the box corner coordinates in list format of only the true boxes detected.



Output from box_tvecs_palletmarker

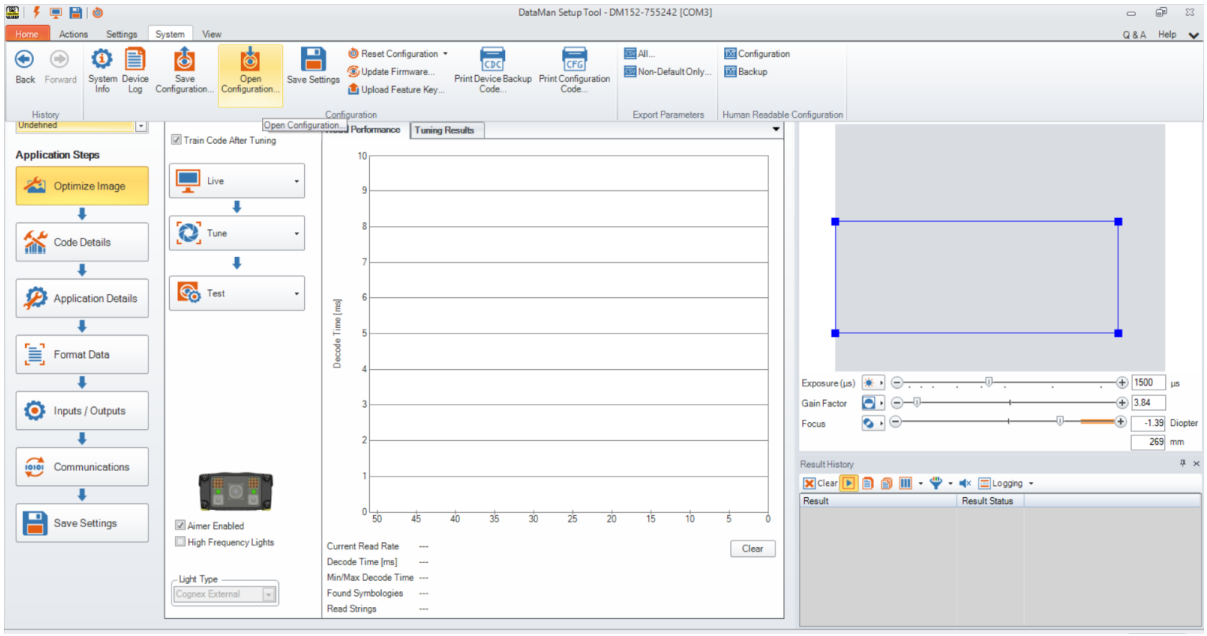
Box-counting algorithm-

This will be fed into the box_count script which will also take input from the one-time creation of the dictionary associated with the stacking pattern of the associated SKU id and will get the image/frame with total count level wise.
It requires SKU id also for keeping track of the stacking pattern. SKU id will be given by either dynamsoft software/barcode scanner/optical character recognition technique.

```
Barcode Format : CODE_128
Barcode Format : CODE_128
Barcode Text : ABC-abc-1234
Barcode Localization Points : [(707, 513), (924, 509), (925, 584), (710, 588)]
Barcode Format : CODE_128
Barcode Format : CODE_128
Barcode Text : ABC-abc-1234
Barcode Localization Points : [(709, 365), (980, 359), (983, 436), (713, 441)]
Barcode Format : CODE_128
Barcode Format : CODE_128
Barcode Text : ABC-abc-1234
Barcode Localization Points : [(1171, 351), (1424, 352), (1417, 450), (1167, 445)]
Barcode Format : CODE_128
Barcode Format : CODE_128
Barcode Text : ABC-abc-1234
Barcode Localization Points : [(1199, 518), (1405, 520), (1399, 615), (1196, 610)]
Barcode Format : CODE_128
Barcode Format : CODE_128
Barcode Text : ABCBa`cE,AB
Barcode Localization Points : [(1206, 671), (1378, 675), (1371, 767), (1206, 761)]
```

Dynamsoft Output

Keras-OCR Output



Barcode Scanner Output

SKU Information Estimation: -

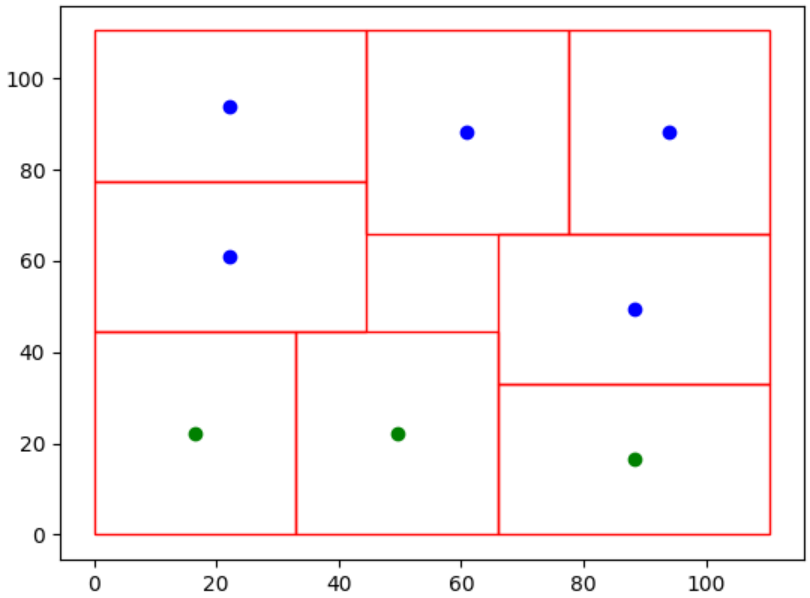
Based on the shape/size of the detected boxes and marker/identifier pasted on them, get the "SKU_ID", fetch all available information from the dictionary for this particular SKU.

Get the 3D pose information of boxes:-

Using shape/size of the detected boxes and markers/identifiers, estimate the position information (height, depth and distance) of each and every visible box with respect to the pallet location.

Level Estimation: -

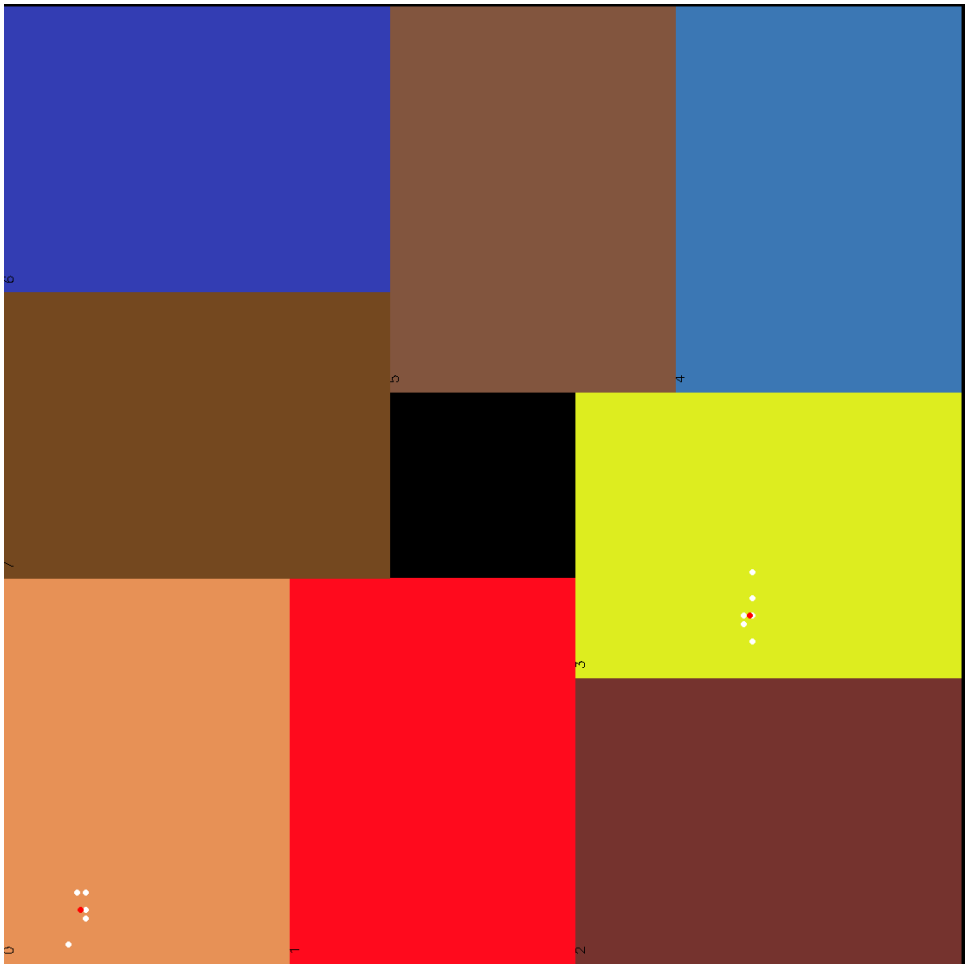
Based on the height of the detected boxes, group boxes level wise. For example, in above image boxes (1, 2, 6) belong to level-1, boxes (4, 3, 5) belong to level-2, boxes (7, 8) belong to level-3. After that stacking pattern estimation for each level will be plotted for user convenience based on the shape/size of the boxes, stacking pattern for them is decided.



Now, the task is to get the stacking pattern for each level. For the boxes shown in above image, There are two stacking patterns possible. Error estimation-based approach is used to decide the “Stacking-pattern". Visible box's locations are plotted on all of the possible floor plans and error is calculated for each and every “stacking-pattern”, then arranging pattern/plan for which error is least is selected. Deep learning-based model can also be trained to estimate the “stacking pattern” of boxes for each level. Input to the model would be shape/sizes of the visible boxes and their 3-D position information and accordingly stacking/arranging pattern can be selected.

Getting List of Hidden and Absent boxes: -

Once “stacking pattern” is estimated, next task is for each level, finding the location of hidden as well as absent boxes. Here, main challenge is based on the positioning of the visible boxes, have to estimate the location of hidden and absent boxes. Below images shows the stacking pattern for level-3: -



All the boxes marked with white colored dots are visible. Provided this information,have to find the list of boxes that are hidden by the visible boxes. To get this, shadowed image needs to be generated.

Predicting Total Count: -

Total count for level = count of visible boxes + count of hidden boxes,
Similarly, total count = Sum of total count for all the levels

Final Output Image showing count of all the boxes stacked on the pallet.

Pallet identification using barcode

The barcode corresponding to the pallet location on the rack will have an ArUco marker right next to it. Drone will detect the ArUco marker, perform error correction & localization and then detect the barcode, Barcode detection may be done either by using the on-board barcode scanner or from the image captured using the camera.

SKU identification using barcode

After localizing & error correction, the agent will capture an image of the pallet that is of interest. This image is processed to identify the location of the barcode. A coordinate transformation is carried out to estimate the location of the barcode in the real world with respect to the camera. This information is used to compute the distance and direction the agent (drone) needs to travel to capture the barcode of interest correctly. This process is applicable, IF the barcode is placed at any place on the front facing part of the rack. If we know the location of the barcode prior to the operation, we can skip the barcode location detection logic and provide required information to the drone to carry out the barcode detection.

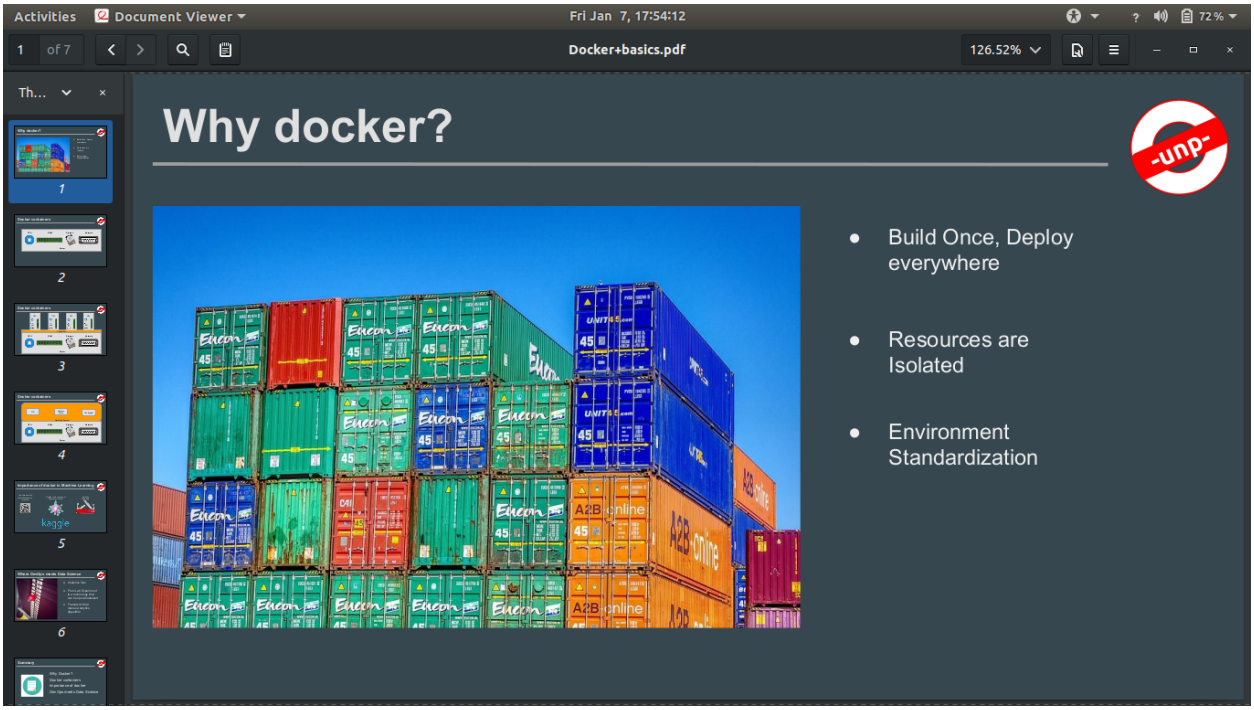
Global Data Structures and Shared Data Functions

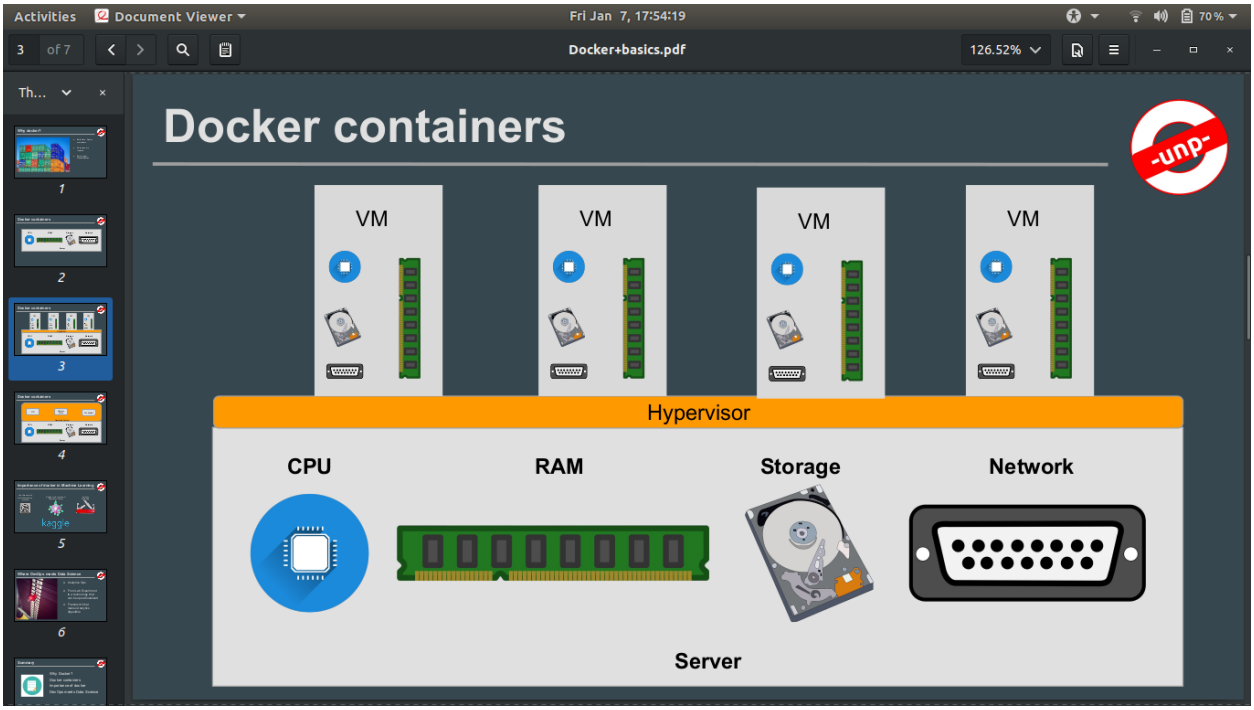
The details of Global Data Structures used by all modules should be specified.
All macros / prototypes / functions common to a set of programs should be described.

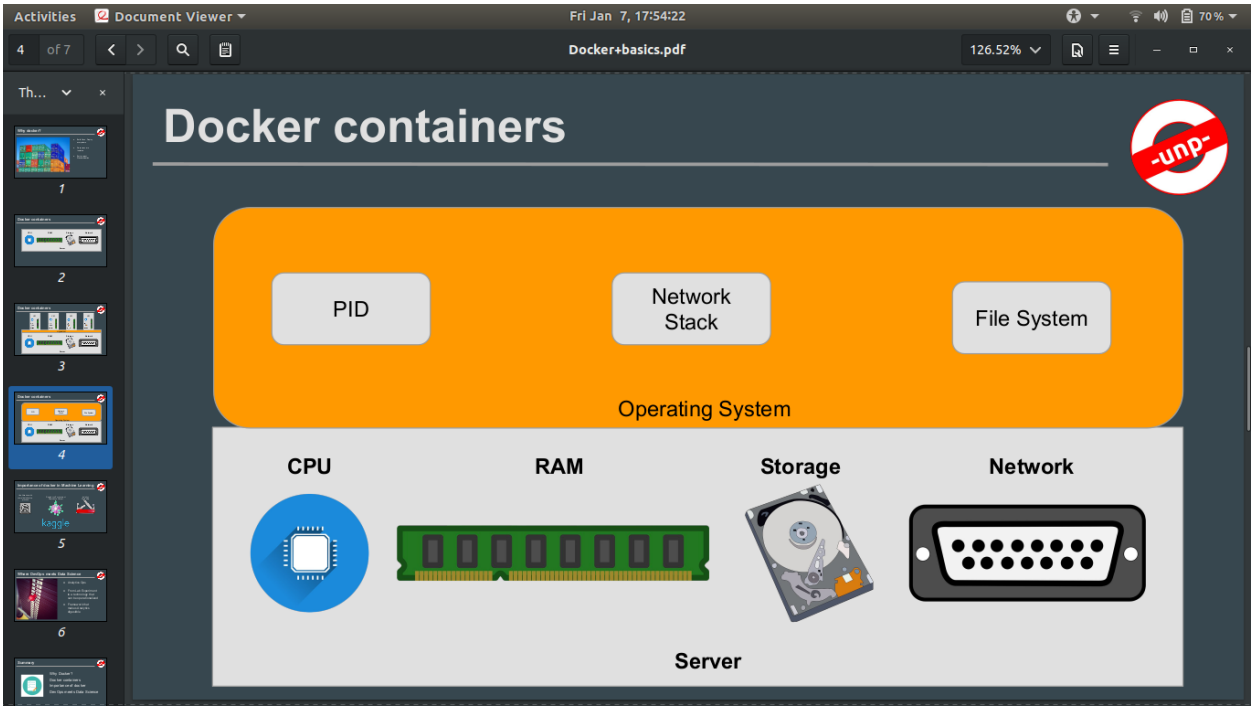
Topic Name / Service Name	Publisher / Server	Consumer	Message / Service Type	Purpose

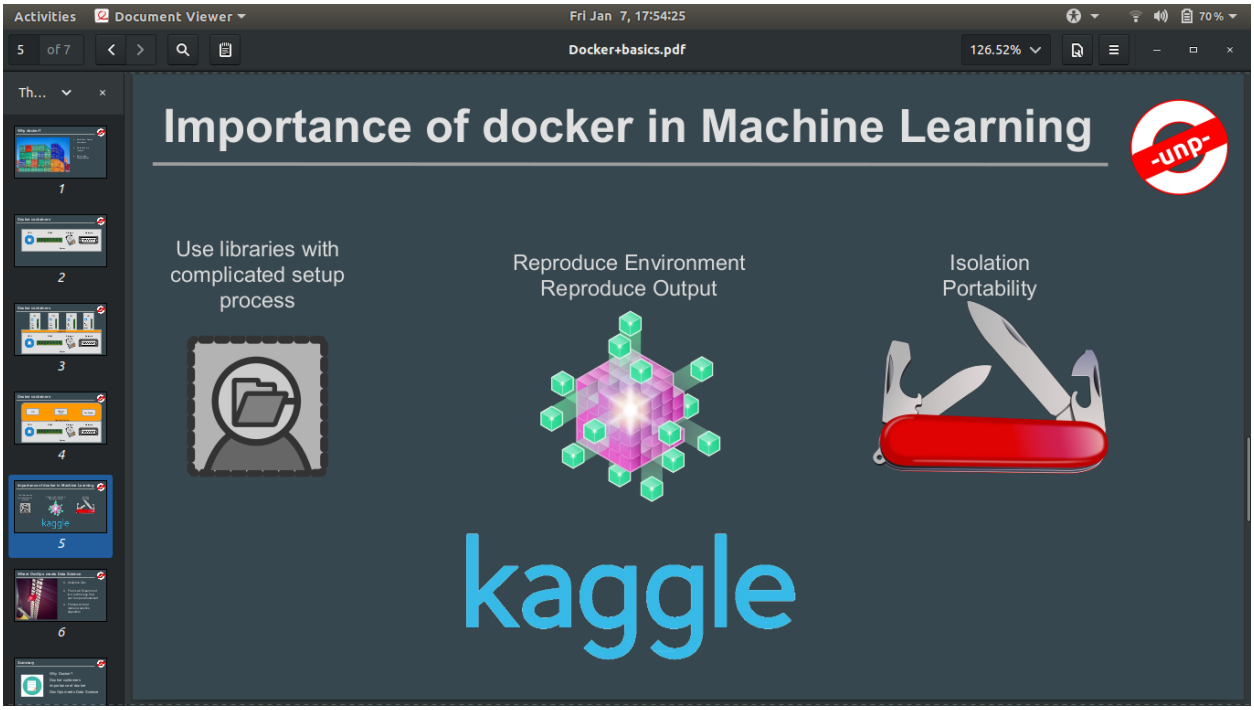
/camera/image/raw	Realsense_camera	Visual Navigation, Data Collection Module	Image	Sends RGB Images from the camera on a ROS topic. Will be used for Navigation, Marker Detection and Data Collection
/camera/depth/image_rect_raw	Realsense_camera	Visual Naivgation	Distance in mm	Depth information with conversion scale of 1mm
/barcode/type	Barcode_scanner	Barcode detection	Type of barcode	Type of barcode used
/barcode/id	Barcode_scanner	Barcode detection	Id of the detected barcode	Barcode id detected by the scanner

Dockerizing the whole module:




Activities
Document Viewer
Fri Jan 7, 17:54:19
3 of 7
Docker+basics.pdf
126.52%
Docker containers
VM
VM
VM
VM
Hypervisor
CPU
RAM
Storage
Network
Server


Activities
Document Viewer
Fri Jan 7, 17:54:22
4 of 7
Docker+basics.pdf
126.52%
Docker containers
PID
Network Stack
File System
Operating System
CPU
RAM
Storage
Network
Server


Activities
Document Viewer
Fri Jan 7, 17:54:25
5 of 7
Docker+basics.pdf
126.52%
Importance of docker in Machine Learning
-unp-
Use libraries with complicated setup process
Reproduce Environment
Reproduce Output
Isolation
Portability
kaggle

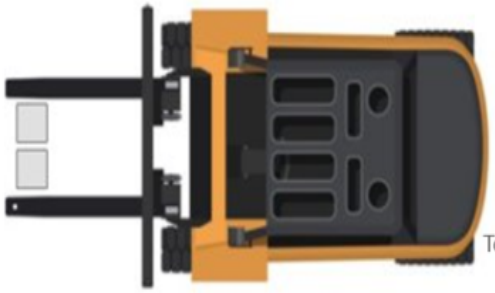

Activities
Document Viewer
Fri Jan 7, 17:54:27
6 of 7
Docker+basics.pdf
126.52%
Where DevOps meets Data Science
-unp-
Analytics Ops
From Lab Experiment to a technology that can be operationalized
Framework that makes Analytics digestible

Box Count-Back Process(Nestle Destination Pallet Approach)

Objective

- In Warehouse the Picker goes to the SKU location
- Picks N boxes from the source pallet and places it into the destination pallet:
  - Take N# boxes from source pallet
    - Trigger START
    - Take one SKUs at a time from source pallet
    - Put these in the destination pallet
    - Trigger STOP
  - Move to the next location
- Our objective is, during this task, we need to give them the count-back of the total number of SKUs being kept in the destination pallet –to be moved elsewhere.
- The WMS will check for any mismatch in the source pallet and give feedback

Side view



Top view

Objective

- Constraints
  - the height/level of boxes in the destination pallet has to be capped
  - The view of the smaller boxes away from the camera will be obstructed
  - Shadow of person on the box might cause detection issues
  - No rotation of boxes while in FOV
- Assumptions:
  - Proper lighting conditions
  - Only rectangular/square boxes considered for now
  - No obstruction of the view of boxes
- Mechanical Setup
  - Camera exact position discussion
  - Checking feasibility and Designing the mechanical setup according to forklift specifications
  - Building custom setup(mount and stand) for the camera on the forklift
- Requirements
  - Depth camera/IP camera
  - Laptop/ Display device

Workflow

Dataset Preparation/Collection:

Dataset will consist of:

- RGB camera captured images from the top view(or top-angled view) which will be like the forklift mounted camera scenario
- Captured the same set of boxes with various possible configurations and different heights
- Used Google open dataset images for increasing the level of granularity and variation of data.





Data Annotation:

- Collected Sample videos for reference purposes and to understand the process
- Annotate the whole data using LabelImg Tool
- For google open dataset, we have used OIDv4 converter which will give annotation in yolo required fomat

Data Training:

- The training was done for almost 1000 annotated images from the dataset mix
- We have trained the model for 3000 iterations(in each iteration model will take 64 images), for IoU threshold = 50 %

Object Detection Algorithms:

Approach1(YOLOv4 + Deep-Sort):

→ **Yolov4** are one stage detectors which achieves state of the art results at a real time speed on various datasets. We Converted our yolo weights to tensorflow framework and ran the box count-back algo on that.

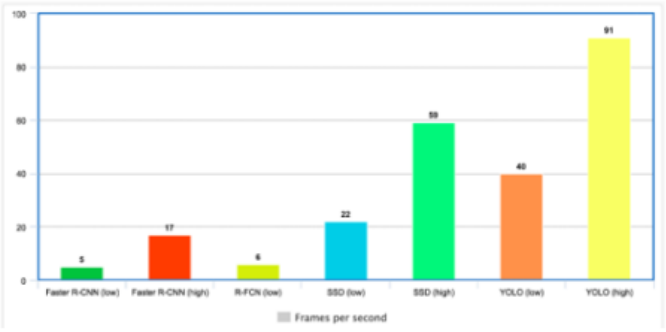
→ **Deepsort** is the fastest object tracking algorithm which will track the object of interest between sequential images of the video.

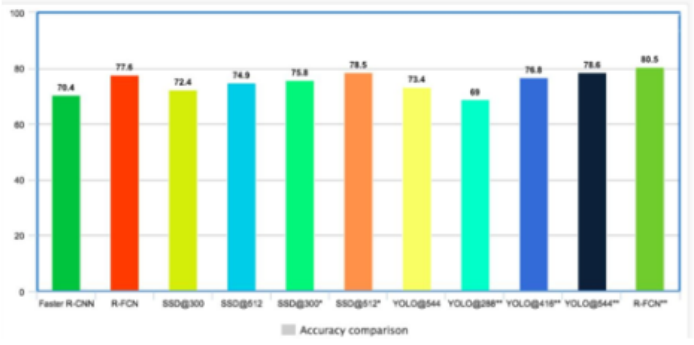
Object Detection Algorithms:

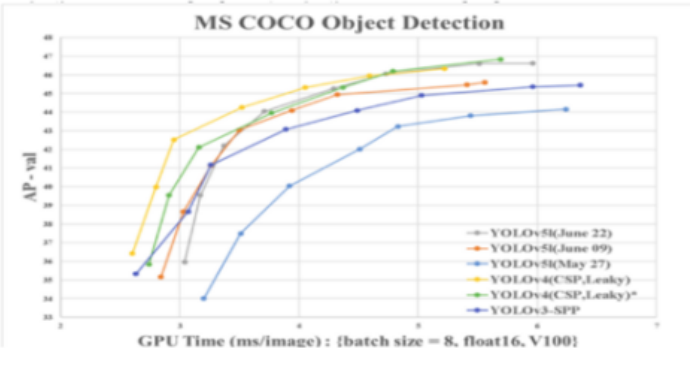
Approach-2(**YOLOv5):**

→ Here the goal is to produce an object detector model that is very performant relative to it's inference time. It derives most of its performance from Pytorch training procedures, while the model architecture remains close to yolov4.

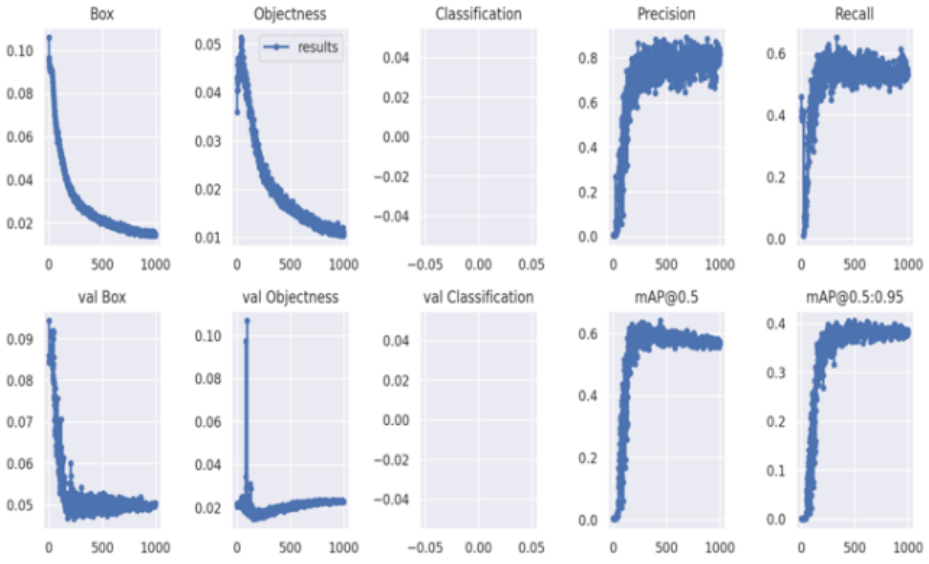
→ There is one additional improvement from yolov4 which is Auto-Learning Bounding Box Anchors when you input your custom data.



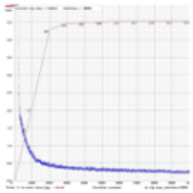




Training Sample Outputs:



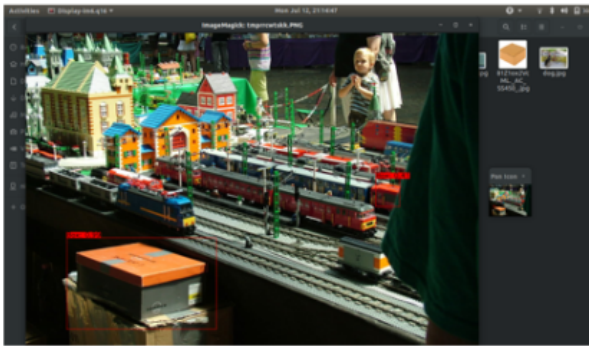
Yolo v4 + Deep-Sort:



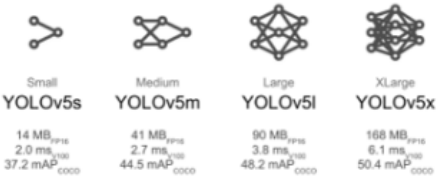
Results and Discussion:

- Pre-processed the individual frames from the video stream using implicit image enhancement techniques which will increase the accuracy of the model by focusing on important features.
- Build Deep Learning model(*yolov4*) using around 1000 images suitable for video frames using *darknet* architecture and trained it in GPU based environment. After getting the output, we have converted darknet weights into tensorflow framework to make it compatible.
- We have trained the model for 3000 iterations(in each iteration model will take 64 images), for IoU threshold = 50 %, used Area-Under-Curve for each unique Recall, we got mean average precision (mAP@0.50) = 0.654714, or **65.47 %** which is far better than many of the two stage detectors like Faster RCNN.
- Got the output as the box detection results (in top view)
- Implemented object tracking module i.e., Deep-Sort on the output of the Deep-Learning-model
- developed record/counter keeping algo for the incoming SKUs in the Pallet area
- The final output will be how many boxes have come inside the destination pallet within time t1 to t2
- The complete object detection yolov4+ object tracking deep-sort model for our testing sample videos in CPU based system give real time detections at around **2fps with 65.47% mAP.**

Output:



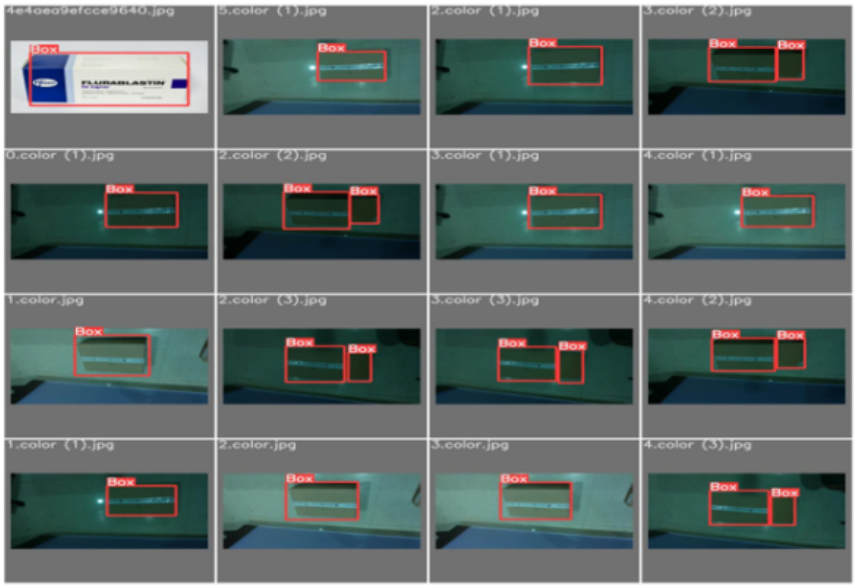
Yolo v5:



Results and Discussion:

- Pre-processed the individual frames from the video stream using implicit image enhancement techniques which will increase the accuracy of the model by focusing on important features.
- Build Deep Learning model(*yolov5*) using around 1000 images suitable for video frames, have implicitly translated the *darknet weights* to *Pytorch* framework which makes it faster to port with custom dataset.
- We have selected a *medium pretrained model* for training
- Got the output as the box detection results (in top view)
- Developed record/counter keeping algo for the incoming SKUs in the Pallet area
- The final output will be how many boxes have come inside the destination pallet within time t1 to t2
- We have trained the model for 1000 epochs(with 80 images in a batch), for IoU threshold = 50 %, used Area-Under-Curve for each unique Recall, we got mean average precision (mAP@0.50) = 0.573, or **57.3 %** which is far better than many of the two stage detectors like Faster RCNN.
- The complete object detection yolov5 model for our testing sample videos in CPU based system give real time detections at around **7fps with 57.3% mAP.**

Output:



Yolov5 output

Jetson Nano board with Laptop

The Jetson Nano Developer Kit doesn't include a WiFi module, so we have two options. We can either connect our Jetson Nano directly to our laptop using an ethernet cable and then set up a static IP and share our network, or we can add a USB WiFi adapter and connect the Nano to the same WiFi network that our laptop is using.

What is SSH?

SSH or Secure Shell is a network communication protocol that enables two computers to communicate (c.f http or hypertext transfer protocol, which is the protocol used to transfer hypertext such as web pages) and share data. An inherent feature of ssh is that the communication between the two computers is encrypted meaning that it is suitable for use on insecure networks.

SSH is often used to "login" and perform operations on remote computers, but it may also be used for transferring data.

How do I use SSH?

You use a program on your computer (ssh client), to connect to our service (server) and transfer the data to/from our storage using either a graphical user interface or command line. There are many

programs available that enable you to perform this transfer and some operating systems such as Mac OS X and Linux have this capability built in.

SSH clients will typically support SCP (Secure Copy) and/or SFTP (SSH File Transfer Protocol) for transferring data; we tend to recommend using SFTP instead of SCP but both will work with our service.

What is the difference between SCP and SSH?

SSH and SCP are two network protocols that can be used to exchange data through a secure channel between two remote devices in a network. SSH stands for Secure Shell, while SCP stands for Secure Copy Protocol. SSH is a protocol for establishing a secure connection between two remote computers, and this secure connection offers encryption, authentication and compression mechanisms. SCP is a protocol for transferring files between the computers in a network, or over the internet using SSH connection. SCP preserves the authenticity and the confidentiality of the data exchanging.

The main difference between SSH and SCP is that SSH is used for logging into remote systems and for controlling those systems while SCP is used for transferring files among remote computers in a network.

Control it through SSH

Instead of working directly on the Jetson we could control it through SSH from our own computer, which is what is recommended. During the initial phase of setting everything up and installing libraries we might have to run commands following instructions such as the Chromium web browser is installed by default but even a few tabs can be tough on the 4 GB RAM that the Nano comes with when installing some packages. Plus, working from a desktop or laptop probably feels more comfortable anyway.

We can use SSH in Ubuntu, macOS, and Windows 10. To control the Jetson Nano through SSH we need to have it connected to the same local network through Ethernet or Wi-Fi.

Find its network address by running ifconfig on the Jetson. Run ssh mircea@192.168.1.70 on desktop or laptop, where we replace mircea with your username on the Nano, and 192.168.1.70 with the IP address that we found through ifconfig.

It will warn that the authenticity of the host can’t be established. Type yes to continue, enter the Jetson Nano password, and we're in!

What to do if SSH stops working

If we set up SSH and afterwards we reflash the SD card and restart everything from scratch, we’ll get an error message.

If we’re using Ubuntu or macOS, simply run the suggested command

ssh-keygen -f "/home/mircea/.ssh/known_hosts" -R "192.168.1.70"

where we replace mircea with our own username and 192.168.1.70 with the IP address of our Jetson Nano.

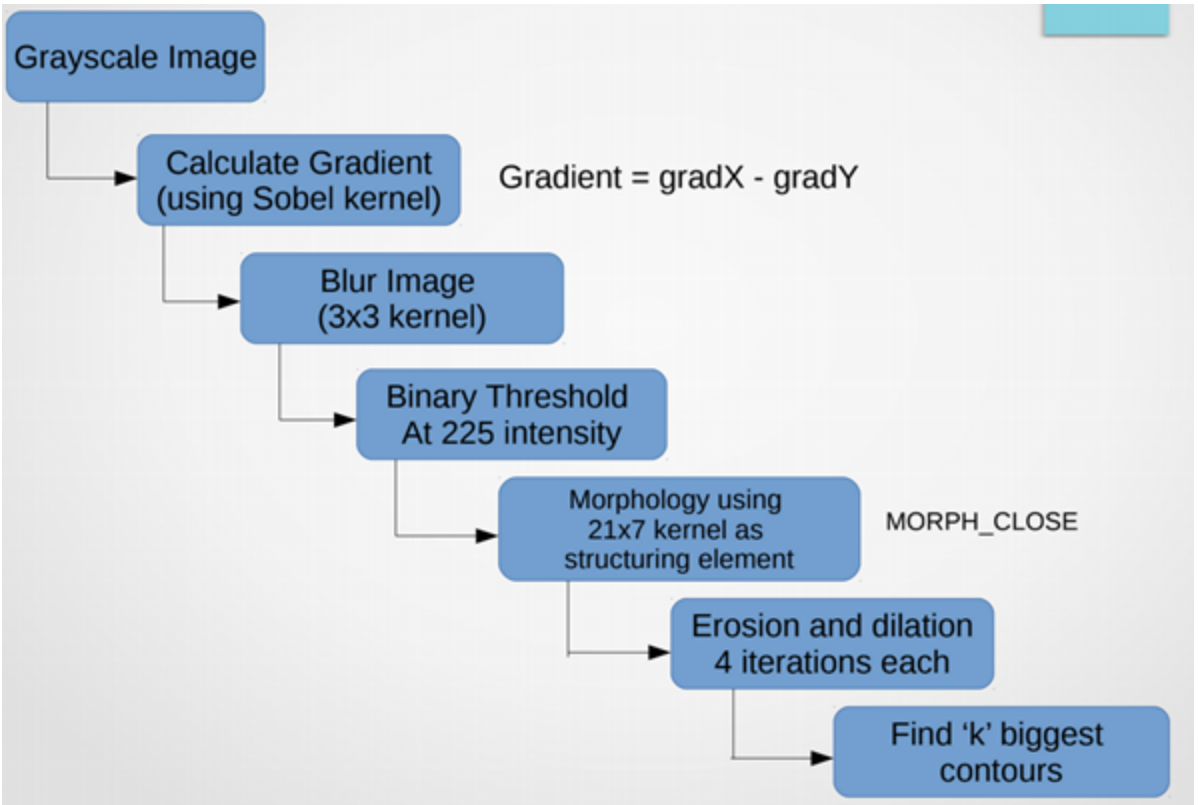
If we're using Windows, take note of the number shown at the end of the line starting with Offending ECDSA key in.

notepad C:\\Users\\Mircea/.ssh/known_hosts

where we replace C:\\Users\\Mircea/.ssh/known_hosts with the path you see in the error message. We can simply copy paste the path. Now remove the offending line shown in the error message and save the file. The line should contain the IP address of your Jetson.

Barcode localization:

a. Github reference: https://github.com/pyxploiter/Barcode-Detection-and-Decoding
b. Algorithm:



c. File "detect_barcode_opencv_exp.py" function name "box_detection()"

2. **Barcode decoding**

a. Pyzbar

Installation: pip install pyzbar

file : “dynamosoft.py” function name “read_barcodes(img)”

```
def read_barcodes(img):
    filter = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
    img=cv2.filter2D(img,-1,filter)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, thresh1 = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)
    thresh1 = convert_3_channel(thresh1)
    barcodes = pyzbar.decode(thresh1)
```

b. Dynamsoft

Installation: pip install DBR

file : “dynamosoft.py” function name “dynamosoft()”

```
def dynamosoft():
    reader = BarcodeReader()
    reader.init_license("t0069fQAAAGmLNgeYFjz3mbu95Ea52IhY4ISkZHT7DYS23O06xAtUa5oQE/PRvfBM2fTPr4bUucQJMLuZtRDXvh7gonXy857T")
    settings = reader.get_runtime_settings()
    settings.barcode_format_ids = EnumBarcodeFormat.BF_ALL
    settings.barcode_format_ids_2 = EnumBarcodeFormat_2.BF2_POSTALCODE | EnumBarcodeFormat_2.BF2_DOTCODE
    settings.excepted_barcodes_count = 32
    reader.update_runtime_settings(settings)
    try:
        image = r"./temp/temp.png"
        text_results = reader.decode_file(image)
    except BarcodeReaderError as bre:
        print(bre)
    return text_results
```

c. Scandit

Installation:

* Camera samples: Video4Linux2 for camera access:
    `$ sudo apt-get install libv4l-dev`
* CommandLineBarcodeScannerImageProcessingSample: SDL2 for loading images.
    `$ sudo apt-get install libsdl2-dev libsdl2-image-dev`
* CommandLineBarcodeScannerImageProcessingSample.py: SDL2 for python3 also for image loading
    `$ sudo apt-get install python3-sdl2`
* CommandLineBarcodeGeneratorSample: libpng for generating the output image.

`$ sudo apt-get install libpng-dev`

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/$USER/scanditsdk/lib

File: “scandit.py” function name “ scandit()”

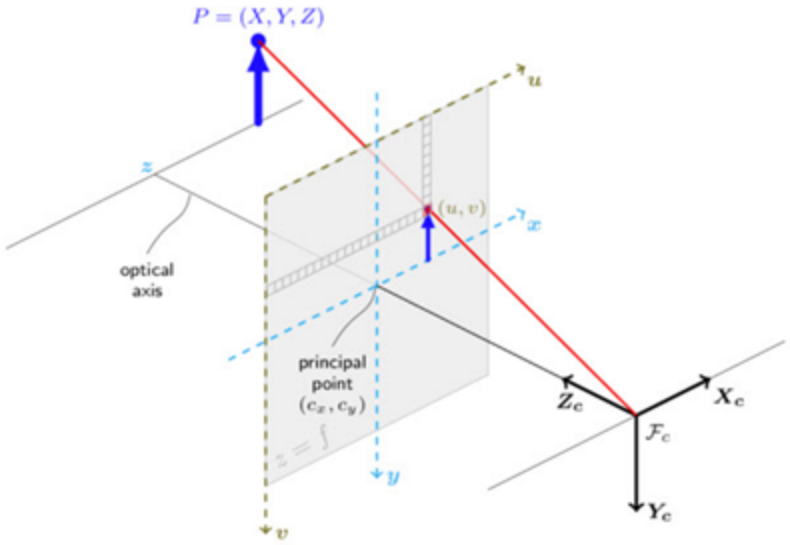
```
def scandit(opencv_image_data):
    # recognition_context = createRecognitionContext()
    recognition_context = createRecognitionContext()
    # Create a scanner settings.
    scanner_settings = createScannerSettings()

    # Create a scanner and start the scanning.
    scanner = sc.BarcodeScanner(recognition_context, scanner_settings)
    scanner.wait_for_setup_completed()

    frame_seq = recognition_context.start_new_frame_sequence()
    # print('&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&7')
    scanning_status = processOpenCvImage(frame_seq, opencv_image_data)
    # print('&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&7')

    frame_seq.end()
    # print('%%%%%%%%%%%%%%%%%%',scanning_status.status, sc.RECOGNITION_CONTEXT_STATUS_SUCCESS)
    if scanning_status.status != sc.RECOGNITION_CONTEXT_STATUS_SUCCESS:
        print(
            "Processing frame failed with code {}: {}".format(
                scanning_status.status, scanning_status.get_status_flag_message()
            )
        )
        exit(2)

    codes = scanner.session.newly_recognized_codes

    return codes
```

3. **Finding camera parameter for Real Sense camera**

   We can use Pyrealsense library in python to get camera parameter both intrinsic and extrinsic

```
import pyrealsense2 as rs
pipeline = rs.pipeline()
cfg = pipeline.start() # Start pipeline and get the configuration it found
profile_color = cfg.get_stream(rs.stream.color)
profile_depth = cfg.get_stream(rs.stream.depth)
# Downcast to video_stream_profile and fetch intrinsics
intr_color = profile_color.as_video_stream_profile().get_intrinsics()
# Downcast to video_stream_profile and fetch intrinsics
intr_depth = profile_depth.as_video_stream_profile().get_intrinsics()
print("color intrinsic parameter---->",intr_color)
print("depth intrinsic parameter---->",intr_depth)
```

4. **Pixel to word coordinate transformation**



World to pixel coordinate projection

Equation for calculating world coordinate from pixel

$$\left(s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} A^{-1} - t \right) R^{-1} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

solving for Real World X Y Z coordinates

Where A in intrinsic matrix
R is rotation matrix
t is translation matrix
In our case extrinsic matrix not required (R|t) because we used 3-d camera along with depth value

Input: pixel coordinate and depth value

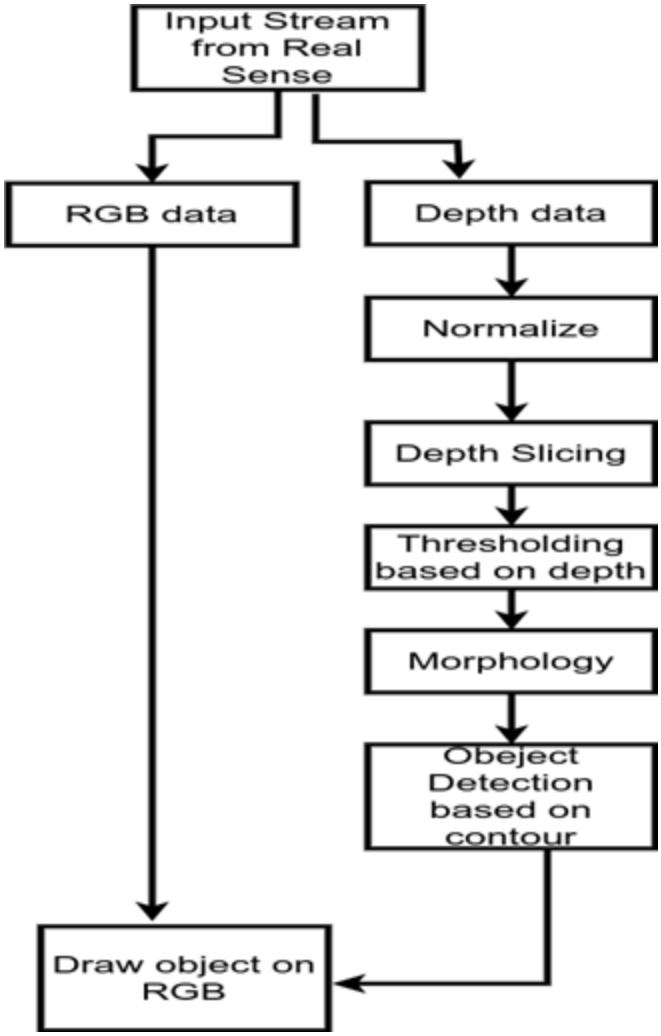
```
def calculate_XYZ(u,v,d):
        cameraMatrix_2 = [[958.069, 0, 947.880],[0, 958.069, 548.147],[0, 0, 1]]
        cameraMatrix = np.array(cameraMatrix_2)
        uv_1=np.array([[u,v,1]], dtype=np.float32)
        uv_1=uv_1.T
        suv_1=d*10*uv_1
        xyz_c=np.linalg.inv(cameraMatrix).dot(suv_1)
        return xyz_c
```

5. **Object detection collision avoidance**

a) In this algorithm we detect obstacle based on depth

Algorithm

- Extract depth data from input stream
- Normalize depth data and change range between 0-255
- Using thresholding segment object on the basis of depth information
- Use morphological operation to remove noise
- Find contour of detected object and discard object on basis of smaller area
- Assign depth value to detected obstacle
- Draw object on RGB stream to show obstacle detection and show danger alarm on the basis of obstacle proximity.



File: “object_detection.py”

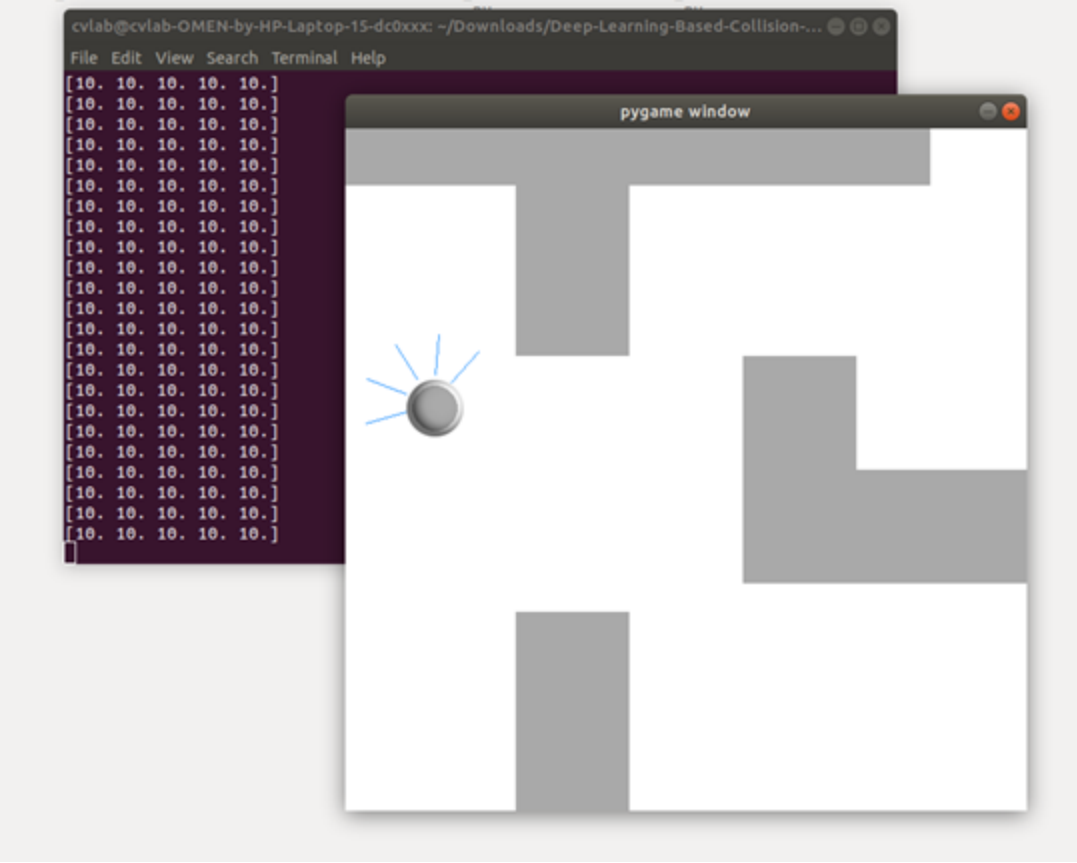
Installation:

i) pip install realsense2
ii)pip install opencv-python

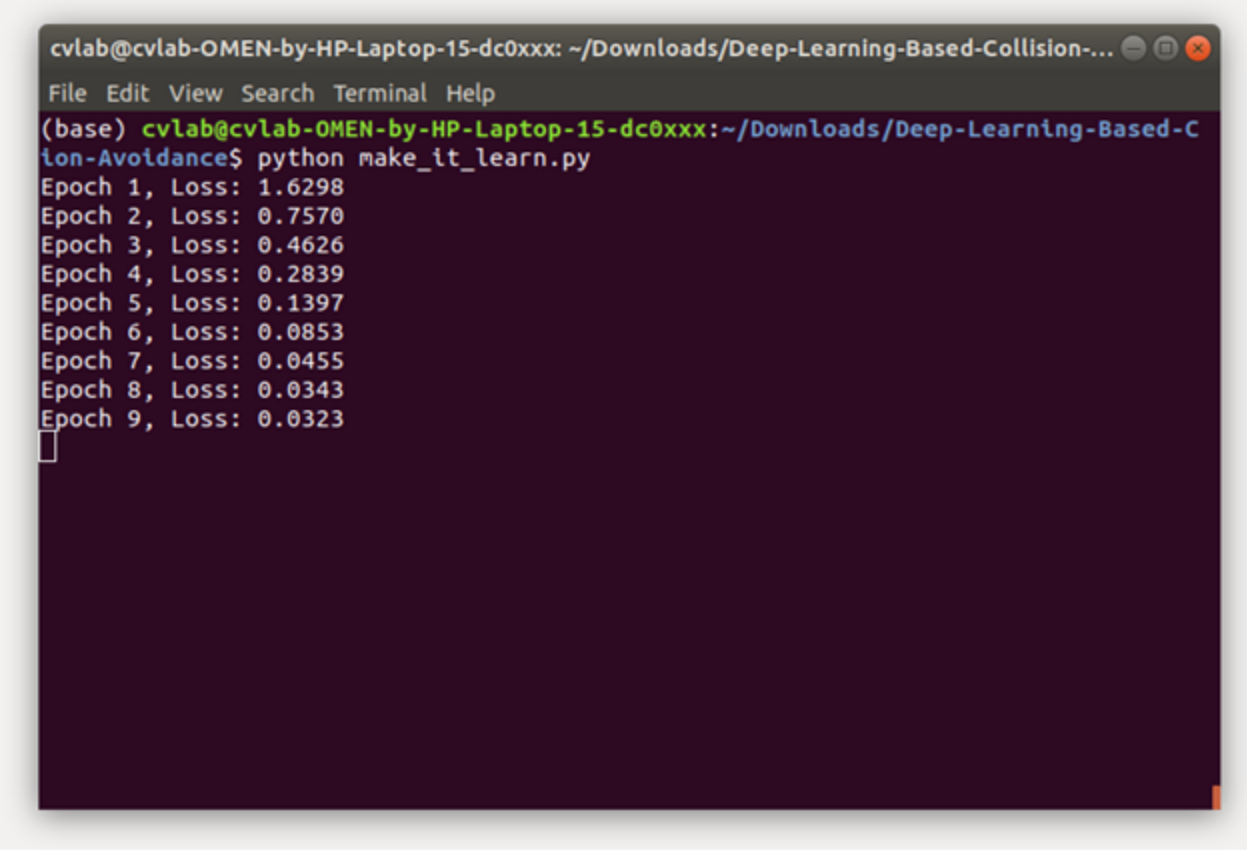
b) Collision detection and avoidance based on the Pygame simulation environment.

i) File **“training_the_model.py”** create the environment and run the agent and collect the dataset.

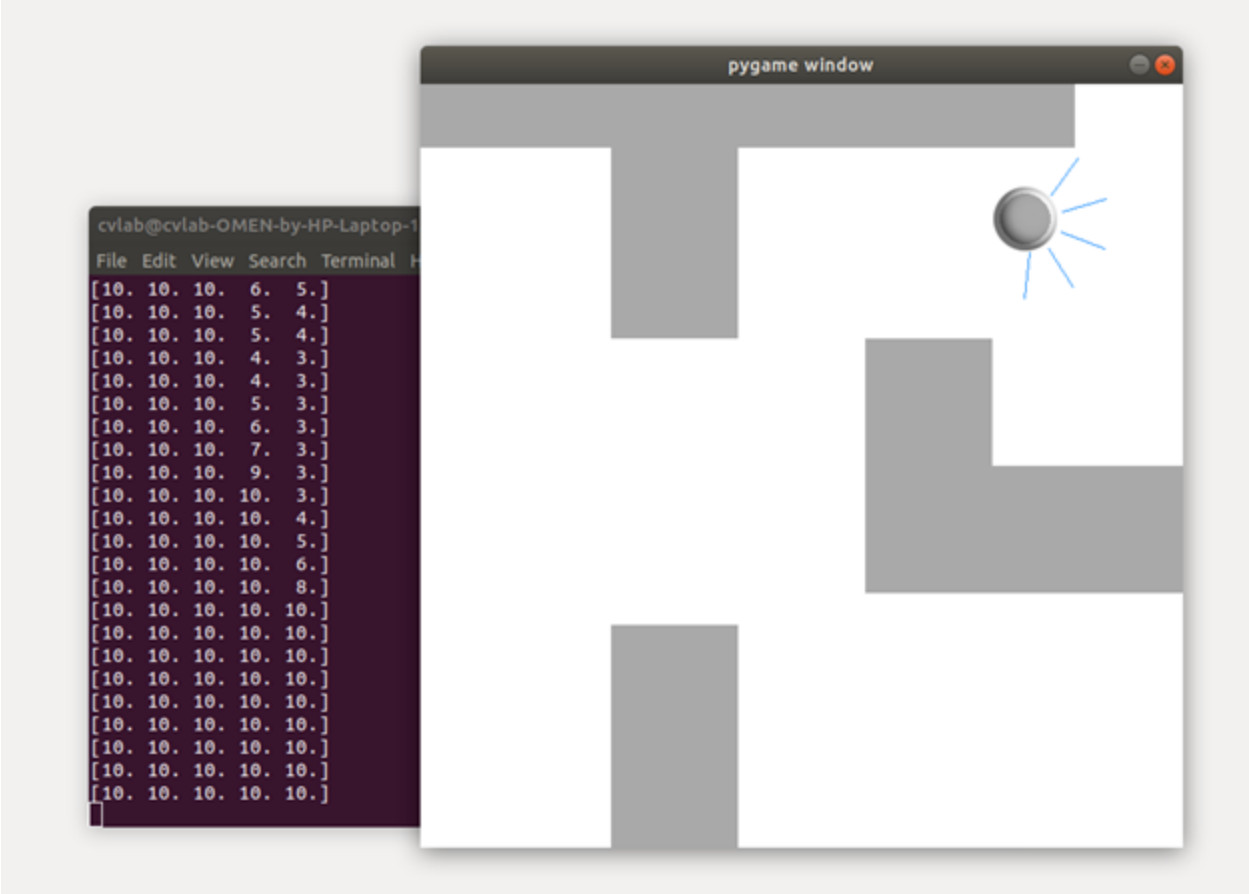
ii) Agent has five proximity sensor values, based on these values the dataset consists of the ground truth.



iii) File **“make_it_learn.py”** train the neural network with two fully connected layers from collected data from the previous step.



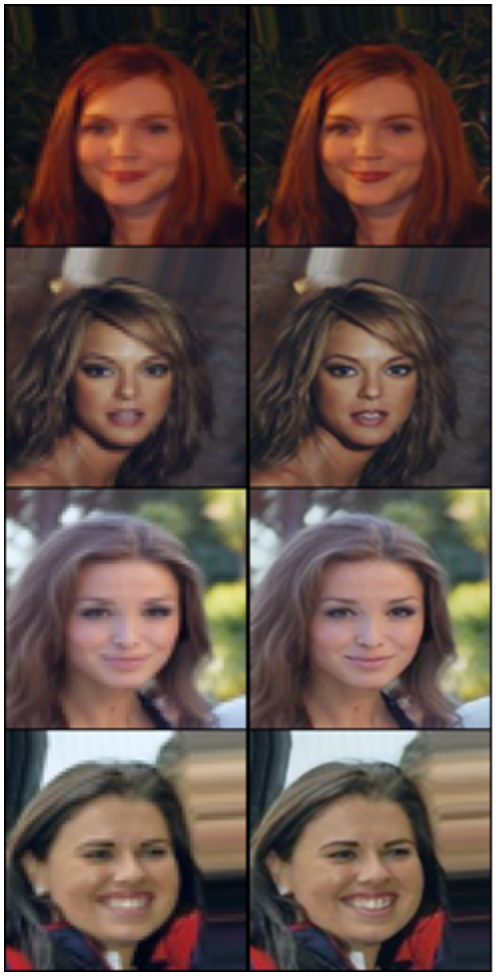
iv) File “playing_the_model.py” run the agent in environment where trained neural network detecting the collision and avoiding based proximity data received by the sensor.



6. **Enhanced Super-Resolution GAN**
   Paper: https://arxiv.org/pdf/1809.00219.pdf

   Code:https://github.com/eriklindernoren/PyTorch-GAN#enhanced-super-resolution-gan

   The Super-Resolution Generative Adversarial Network (SRGAN) is a seminal work that is capable of generating realistic textures during single image super-resolution. However, the hallucinated details are often accompanied with unpleasant artifacts. To further enhance the visual quality, we thoroughly study three key components of SRGAN - network architecture, adversarial loss and perceptual loss, and improve each of them to derive an Enhanced SRGAN (ESRGAN). In particular, the Residual-in-Residual Dense Block (RRDB) without batch normalization as the basic network building unit. Moreover, the idea from relativistic GAN to let the discriminator predict relative realness instead of the absolute value.



Faster RCNN Approach Model Building:

https://github.com/Xiangyu-CAS/R2CNN.pytorch

https://github.com/facebookresearch/maskrcnn-benchmark

Autonomous Navigation of Drones:

1. **Autonomous navigation using Intel® RealSense™ Technology: -**
   A prototype robot that uses the Intel® RealSense™ D435 and T265 to plan a path from A to B, but also to react to objects thrown in its intended path. This prototype has these features: -

- Object detection

A depth camera like the Intel RealSense depth camera D455, D435i or D415 can be used to detect many different objects easily, but there are some challenging materials that can be hard to recognize automatically. Detecting objects and recognizing objects are two different capabilities. Intel RealSense depth cameras can detect objects and measure objects within their field of view trivially, since measuring size and distance is an inherent capability of the cameras. **Recognizing what an object is requires additional effort however, utilizing a machine learning framework**.

- Collision avoidance

A key factor in safety, collision avoidance is the ability of a robot to be able to navigate around or avoid any obstacles. This includes both static obstacles like walls or uneven surfaces, but also temporary or moving obstacles like people or pets. We can use an Intel RealSense D415 depth camera in order to measure the depth difference between drivable surfaces and road anomalies that have a **height larger than 5cm** from that surface. Using the depth camera allows them to use a Self-Supervised Label Generator (SSLG) to automatically label road anomalies and drivable areas. Areas beyond the range of the camera are automatically labelled as ‘unknown’ where everything else in the scene is segmented into drivable and non-drivable or road anomaly areas.

- Path Planning

Once drivable areas are identified, the next stage of autonomous navigation is to plan a path of motion that avoids obstacles. This prototype uses an algorithm developed to take data from a D435 depth camera to plan paths through any environments.

- Simultaneous Localization and Mapping

For awareness of a space and location within it, **Simultaneous Localization and Mapping or SLAM is necessary. SLAM algorithms like those embedded in the Intel RealSense Tracking Camera T265 are designed to help an object know not only where it is but where it has been within a space**. It does this by taking the visual feed from a pair or more of cameras, combined with other sensors like wheel odometers or inertial measurement units (IMU) and compare how visual features and the data for those sensors move over time. This happens extremely rapidly, and with this changing data, the algorithms create a ‘map’ of the area a robot or drone has travelled. **The tracking is based primarily on information gathered from two onboard fish-eye cameras, each with approximately a 163-degree range of view (±5 degrees) and performing image capture at 30 frames per second.** The wide field of view from each camera sensor helps keep points of reference visible to the system for a relatively long time, even if the platform is moving quickly through space. **Using both a RealSense D435i sensor and a RealSense T265 sensor can provide both the**

maps and the better quality visual odometry for developing a full SLAM system. The D435i used for the mapping, and the T265 for the tracking.

2. **Easing the development of visual SLAM applications: -**

Simultaneous Localization and Mapping (SLAM) describes the process by which a device, such as a robot, uses sensor data to build a picture of its surrounding environment and simultaneously determine its position within that environment. **There are different ways of implementing SLAM, both in terms of the software algorithms deployed and the sensors used, which may include cameras, Sonar, Radar, LiDAR and basic positional data, using an inertial measurement unit (IMU).**

Visual SLAM systems also offer advantages where GPS is not available, for example in indoor areas or in big cities, where obstruction by buildings reduces GPS accuracy. There are many different approaches to the implementation of visual SLAM, but all use the same overall method, tracking set points through consecutive camera frames to triangulate their 3D position whilst simultaneously using this information to approximate camera pose. **In parallel, SLAM systems are continuously using complex algorithms to minimize the difference between projected and actual points – the reprojection error. Visual SLAM systems can be classified as direct or feature-based, according to the way in which they use the information from a received image.** Direct SLAM systems compare entire images to each other, providing rich information about the environment, enabling the creation of a more detailed map but at the expense of processing effort and speed. While feature-based SLAM methods, which search the image for defined features, such as corners and “blobs” and base the estimation of location and surroundings only on these features. Although feature-based SLAM methods discard significant amounts of valuable information from the image, the trade-off is a simplified process that is computationally easier to implement.

Visual SLAM processing is **extremely computationally intensive, placing high loads on traditional, CPU-based implementations, leading to excessive power consumption and low frame rates, with consequent impacts on accuracy and battery life.** The developers are increasingly using dedicated vision processing units (VPUs) in their designs. A VPU is a type of microprocessor with an architecture designed specifically for the acceleration of machine vision tasks, such as SLAM, and that can be used to offload the vision processing from the main application CPU.

Even with VPUs, however, the visual SLAM developer must still overcome several challenges since creating efficient code for the different SLAM modules is a non-trivial undertaking and it can also be difficult to interface the VPU to the main processor.

VSLAM Development Tools

With speed-to-market critical in today’s environment it is not always practical for a developer to take the time to acquire the skills and knowledge required to implement efficient vision processing code. Fortunately a number of tools exist to facilitate the acceleration of cost-effective SLAM applications; **application development kits are available which provide a combination of vision-specific software libraries, optimized hardware and integration tools to enable the developer to easily offload the vision specific tasks from the CPU to the VPU.** The CEVA SLAM SDK, is a leading example of such an application development toolset. Based on the CEVA XM6 DSP and CEVA NeuPro AI processor hardware, the CEVA SLAM SDK enables the efficient integration of SLAM implementations into low-power embedded systems. **The SDK features several building blocks including image processing libraries providing efficient code for feature detection and matching as well as bundle adjustment.** The CEVA XM6 hardware is optimized for image processing with innovative features such as the parallel load instruction, which addresses the non-consecutive memory access problem, and a

unique, dedicated instruction for executing the Hamming Distance calculation. The SDK also includes a detailed CPU interface, enabling the developer to easily integrate the vision processing functionality with the main application CPU. As an illustration of the performance of the SDK as a development tool, a reference implementation of a full SLAM tracking module running at **60 frames per second was measured to have a power consumption of only 86mW.**

3. **Obstacle Avoidance Drone by Deep Reinforcement Learning**

Drones with obstacle avoidance capabilities have attracted much attention from researchers recently. They typically adopt either supervised learning or reinforcement learning (RL) for training their networks. The drawback of supervised learning is that labelling of the massive dataset is laborious and time-consuming, whereas RL aims to overcome such a problem by letting an agent learn with the data from its environment.

PEDRA is a programmable engine for Drone Reinforcement Learning (RL) applications. The engine is developed in Python and is module-wise programmable. **PEDRA is targeted mainly at goal oriented RL problems for drones, but can also be extended to other problems such as SLAM, etc.** The engine interfaces with the Unreal gaming engine using AirSim to create the complete platform.

Unreal Engine is used to create 3D realistic environments for the drones to be trained in. Different levels of details can be added to make the environment look as realistic or as required as possible. PEDRA comes equipped with a list of 3D realistic environments that can be selected by the user. Once the environment is selected, it is interfaced with PEDRA using AirSim. AirSim is an open-source plugin developed by Microsoft that interfaces Unreal Engine with Python. It provides basic python functionalities controlling the sensory inputs and control signals of the drone. PEDRA is built onto the low-level python modules provided by AirSim creating higher-level python modules for the purpose of drone RL applications. **The current version of PEDRA supports Windows and requires python3.**