Contents

Introduction

Overview

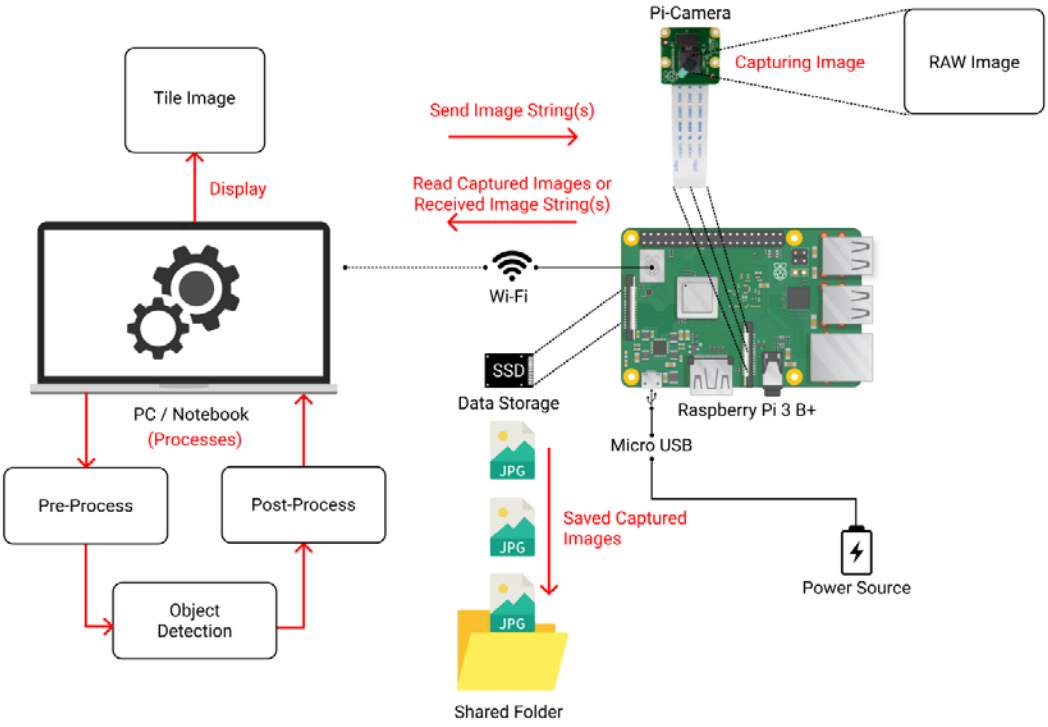


Figure 1: Image Recognition Process Architecture

The figure depicts the overview of our IR process from getting RAW image data from RPI's 'PiCamera' to detected image string and tile images.

Objective

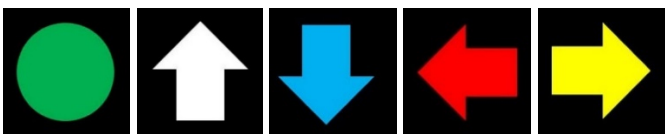
1. Ability to detect 15 classify object from images taken from the arena with camera mounted on the robot.
2. The object detected must be plotted and label correctly onto corresponding images.
3. All object detect images must be tile and display at the PC/Notebook or N7(Android)
4. Keep the solutions simple!

Model

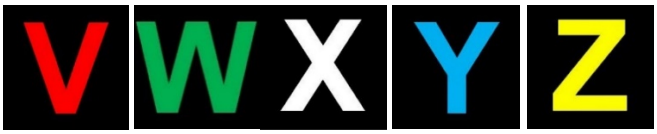
Introduction

The model is responsible for detecting the target object in an image. This model will help us identify the location of objects in an image, the images are captured by our RPI. In our context, our objects would be referring to 15 classes of symbols, arrows, alphabets or digits as shown below

Symbol Arrows



Alphabets



Digits

We need to build and train a model such that it can fit into our context and detect our object accurately. Apart from that we need to execute our trained model to test our object detection algorithm.

Design Choice

Consideration

- Deep Learning Frameworks provider
    - Selected choices
        - Tensorflow
        - Pytorch
    - We choose TensorFlow as our team members are more familiar with and were working on TensorFlow API as some of them are concurrently taking neural network modules. Therefore, it will be more intuitive to choose TensorFlow as our workspace.
- Model Selection
    - From Scratch
        - We choose not to build from scratch, although it helps us in understand how the model works but given the time constrain and effectiveness of the model. It is more advice able to choose an established model offer by the TensorFlow.
    - Pre-Built Models
        - We largely focus on COCO (Common Objects in Context), it is a database for object detection, the model below is pre-trained on COCO's dataset
    - TensorFlow1
        - https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md
    - TensorFlow 2
        - https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md
    - Model Trade-off
        - The higher decision speed (ms) and lower the accuracy (mAP) it can be. Such phenomenon is also known as speed-accuracy trade-off (SAT)
    - Model Architecture
        - mobilenet
            1. It is a type of CNN (Convolution Neural Network) that consist of 53 layers deep.

2. It introduced inverted residual structure; a residual block connects the beginning and end of a convolutional block with a skip connect.
3. Which is suitable for low computational power such as RPI.
4. It supports real-time object detection

- YOLO (You Only Look Once)
  1. It is fast computation
  2. Utilize entire image during training and test time
  3. Learn generalization representation of objects
  4. It supports real-time object detection
- Remarks: We must implement to test and see for ourselves.

o Platform use

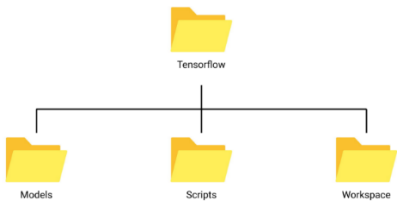
- 'COLAB'
  1. It is executable python code via online browser
  2. Utilize google's GPU
- Local
  1. Utilize PC/Notebook CPU and GPUs
- Remarks: We choose locally as it may take more time to learn how to operate using COLAB and given we have a relative decent GPUs to use.

Implementation

Pre-Requisite

1. cnDNN 7.6.5
2. CUDA 10.1
3. GPU with capability rating of 3.5 and above (GTX 650 and above)
4. Anaconda with Jypyter Notebook 6.1.4
5. OS Windows or Linux
6. Python version 3.8 series
7. TensorFlow 2.2.0
8. Pip version 20.2.4 and above
9. Protobuf 3.12.3 and above
10. Cython 3.0 and above
11. COCO API
12. Object Detection API
13. Visual C++ 2015 Build Tools
14. Downloaded respective models

Directory structure



- Models folder
  - All object detection API are placed here.
- Scripts folder
  - Consist of python scripts that generate records such that can be used for our model inputs. The records are binary file help minimize disk space and take less time to copy and better-read efficiency, it is easier to manipulate with TensorFlow libraries' functions
    - Train.record

        - Test.record
    - Note: Pre-requisite Dataset Process in order to successfully produce the records
- Workspace folder
    - Train models happen here.
    - Consist
        - Downloaded models
        - Configuration is needed such that
            - Label Text File
                - Label_map.pbtxt should consist 15 of our correct classify images
            - Pipeline Configuration
                - Pipeline.config found in downloaded models, it should be modified the number of classes and the right directory paths for the inputs such as records, data images and label XML file, default hyperparameter, outputs such as model file (.pb) and check point (.ckpt).
        - Export Models
            1. It consist of inference graph in .pb format such that it is possible to convert into a detection function using 'tensorflow.saved_model.load' method.
            2. These files will then be transfer into our image recognition repository for object detection

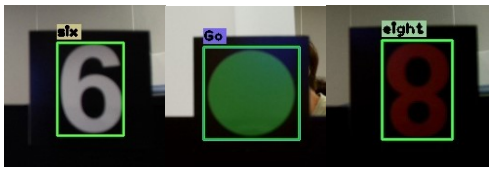
Dataset

Consideration

Pi-Camera Settings

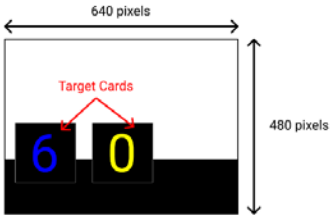
1. ISO
    a. It controls the camera ability to capture light. We choose ISO == 100 as the arena is daylight brightness, having high ISO would introduce noise that we do not want.
2. Exposure
    a. We set to 'auto' to ensure consistent lighting of the image.
3. Aspect Ratio
    a. We use maximum vision the PiCamera can offer which is 4:3 ratio as the tile image from the rule suggest maximum vision is needed.
4. Target
    a. Classes
        1. Multi Classifier

            1. May introduce more unintended noise and coordinate will be complicated to implement. Thus, not considered at the end.
        2. Single classifier



1. Experiments shows that it is more stable compare to Multi-classifier

b. Resolution



1. We use this resolution as it aligns well with the convolution kernel use for our model.
2. We tested with high resolution, it does not improve the detection instead increase object detection process speed and capture process time.

c. Distance

1. We experiment with different distance and notice the detection is sensitive to the target's size proportion. For example, if our dataset only consists of 25cm the detection is approximately +1cm. It shows how sensitive
2. We use distance 25 cm with +-3cm tolerance the reason as follows
    1. this is to compensate the offset, the result of inconsistent Arduino's wheel 'clipping'.
    2. To ensure right coordinate can be used correctly.
    3. As our experiment shows that at around 25cm we can capture 3 blocks. Thus, easier to identify the coordinate with left middle right logic, this is to reduce the complexity require for algorithm team.

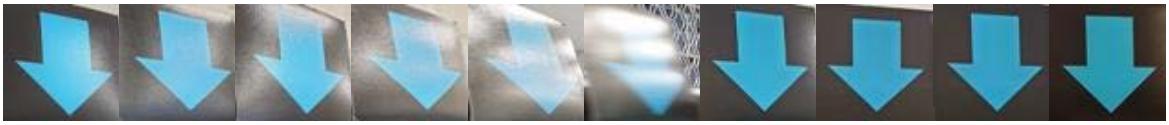
d. Orientation

1. Base, on our mobilenet model experiment it does not influence by orientation of the arrows

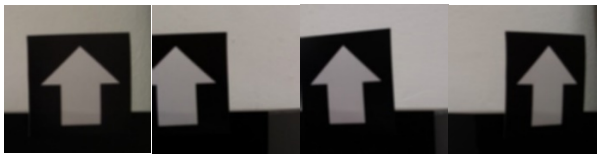
2. The tolerance of orientation is about 10~20 offset from origin angle. This should be fine as the target image should not be tilted at an angle.

e. Angle

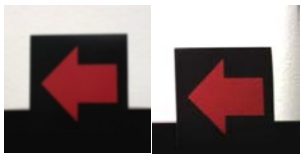
1. Initial we consider angle from front view to extreme, but we realize it may not be feasible for coordinate. We keep it simple up to 4 variations of angle.
2. Initial Angle Used



3. Final Angle Used



4. Motion
    1. The robot may take picture while it is in moving motion, therefore we mimic the motion by blurring the images with gaussian filtering



Blur vs Non-Blur

5. Aspect Ratio
    1. Based on our experiment, the aspect ratio is super sensitive to object detection, since our target card is 1:1 aspect ratio, we use a 1:1 ratio as well

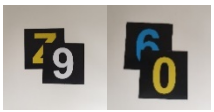
environment vs non environment

- We choose with environment because we may not capture the exact location at the robot may offset too much to the front or too less to the back.

6. Channels
    1. RGB

    2. Gray-Level

7. Overlap



    1. We ceased consider this issue since we fixed the distance of our detectable distance and they should not be any occlusion from our PiCamera line of sight

8. Number of Images per class
    1. We tested different variation of datasets
    2. In order to train effectively each class must have at least 100 images. Total of minimum 1500 images. This is to prevent underfitting

9. Reflection

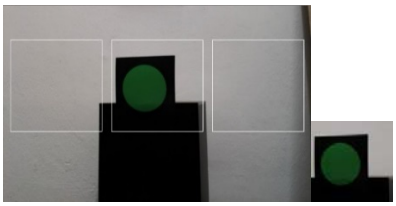
    1. This only happen when view at extreme angle disregard less of the position of the light source
    2. This was not our concern as we do not view the image at an extreme angle

Preparing Dataset

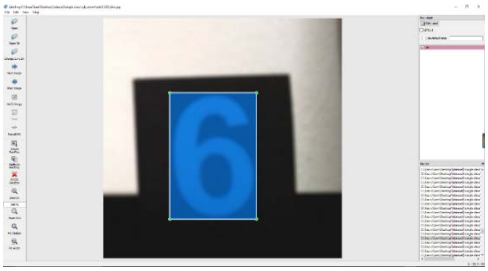
To find out what factors give the best results. We start off small by testing 1 class of 100 images. Base on result, we note down the effect and changes. This cycle repeats until we are satisfy with the result.

Steps

- Capture and gather images with PiCamera
- All images will go through cropping operation such that it was the right size
    - Each crop size was 190 pixels by 190 pixels. The crop size was derived by segment the image into 3 equal widths and further trimed.



- We use a software called 'labelImg' to label the images to generate XML file require for model.
    a. LabelImg



    o XML file content

```
<annotation>
    <folder>test</folder>
    <filename>6 (29)_blur.jpg</filename>
    <path>C:\tensorflow1\models\research\object_detection\images\test\6 (29)_blur.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>190</width>
        <height>190</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>six</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>53</xmin>
            <ymin>59</ymin>
            <xmax>122</xmax>
            <ymax>159</ymax>
        </bndbox>
    </object>
</annotation>
```

- Once done all images should place in the right directory such that it can generate records file to be trained by the model.
- Create Dataset Composition/Blueprint

ID	Position	Distance(cm)	Offset X-axis(cm)	Angle (5 degree)	Angle (10 degree)	Light Source
1	LEFT	25	None	None	None	2
2	LEFT	25	1	None	None	2
3	LEFT	25	-1	None	None	2
4	LEFT	25	None	Left	None	2
5	LEFT	25	None	Right	None	2
6	LEFT	25	None	None	Left	2
7	LEFT	25	None	None	Right	2
8	LEFT	25.5	None	None	None	2
9	LEFT	24	None	None	None	2
10	LEFT	23	None	None	None	2
11	LEFT	25	None	None	None	1
12	LEFT	25	None	None	None	0
13	LEFT	25	2	None	None	2
14	LEFT	25	-2	None	None	2

Repeat the same for middle and right position 14 x 3 = 42 sets

Compute for blur and contrast, the cumulative of image is 42 x 3 = 126 sets

- Blur (Gaussian - kernel size 3 by 3)

```
cv.blur(image,(3,3))
```

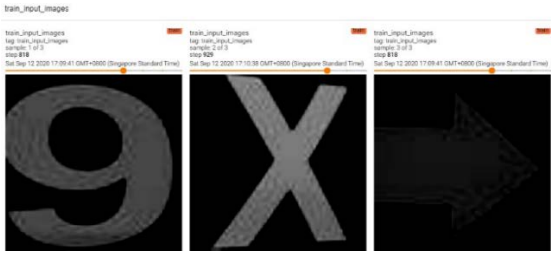
- Contrast (alpha = 2 and beta = 0)

```
def change_brightness(img,alpha, beta):
    return cv.addWeighted(img, alpha, np.zeros(img.shape, img.dtype),0, beta)
```

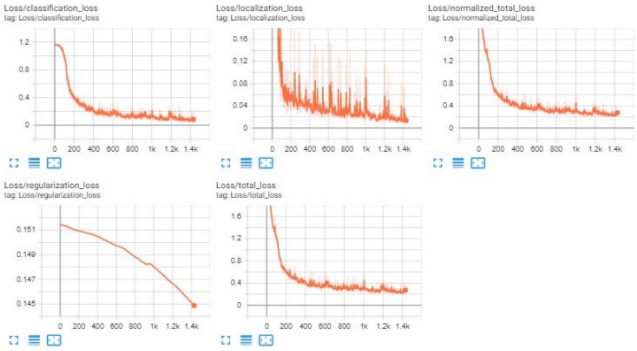
Repeat for all 15 classes which is 126 x 15 = 1890 sets in total

Training Model

- Consideration
    - We follow the recommendation of splitting our dataset into 80:20 trainset and testset respectively with our train dataset. We divide them randomly to ensure bias-free.
    - After some trial and error, we found out that 4000 epochs are the best value to prevent underfitting or overfitting the model.
- Implementation
    - Using Tensor Board, we can visualize how the model learnt and display them statistically
    - Learnt Input Images



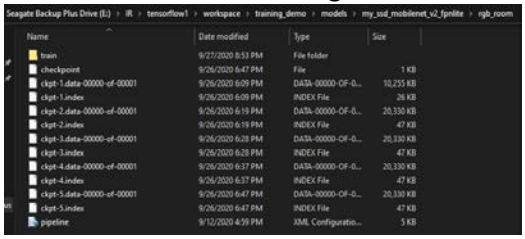
    - Model's Loss Graph



    - Terminal Console

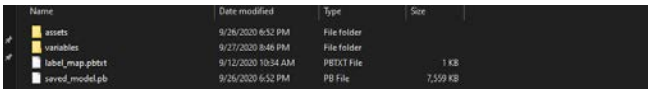
```
I0912 12:55:12.536207 11484 model_lib_v2.py:649] Step 1100 per-step time 0.609s loss=0.221
INFO:tensorflow:Step 1200 per-step time 0.624s loss=0.213
I0912 12:56:12.272427 11484 model_lib_v2.py:649] Step 1200 per-step time 0.624s loss=0.213
INFO:tensorflow:Step 1300 per-step time 0.579s loss=0.207
I0912 12:57:11.694021 11484 model_lib_v2.py:649] Step 1300 per-step time 0.579s loss=0.207
INFO:tensorflow:Step 1400 per-step time 0.579s loss=0.196
I0912 12:58:09.941260 11484 model_lib_v2.py:649] Step 1400 per-step time 0.579s loss=0.196
```

    - The loss value ideally converge point is around 0.150 to 0.200 range and we stopped training model once it reached 4k epochs.
    - Note: The model can be further train as long the correct check points are in place.



Deploy Model

- After running exporting python script, the exported-models directory should consist of saved models and its necessities files as shown below.



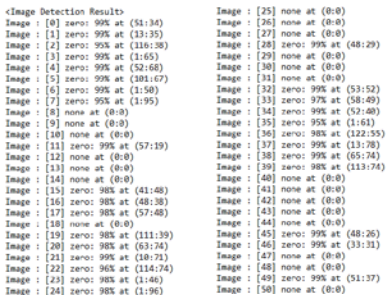
- The exported file can be place into Image Recognition Workspace Directory under 'my_mobilenet_model' folder as shown below to be use for testing and evaluation.

Experiments Consideration

- We need to ensure each class is tested equally for a fair test
- We use the noisiest environments with and without target to determine how good our model.
- Test set
  - 
  - Total of **760** test set
    - The image size is equivalent to a partition image size for our models
  - **740** images, 49 sets of images per class consist of:
    - Different distances
    - Different angle
    - Different environments
    - Different position
    - Note: A portion of the images of the above factor that expecting it to be failed to detect
  - **20** images are noise images
  - The Model Use
    - RGB_Card (1890 Images, 126 per class)
    - RGB_Room (1890 Images, 126 per class)
    - RGB_Mix (3780 Images, 252 per class)
    - Gray_Card (1890 Images, 126 per class)
    - Gray_Room (1890 Images, 126 per class)
    - Gray_Mix (3780 Images, 252 per class)
  - The Model Process Differences
    - 'Card' refers to crop after contour cropping process
    - 'Room' refer to crop without contour cropping process
    - 'Mix' refer to mixing both 'Card' and 'Room' datasets
    - Model that use Grayscale as dataset, capture image require conversion to gray image before object detection process

Results

Abstract result.txt



Models	Average Accuracy	Detected	Ideally	Correctness	Noise
RGB_Card	93.34%	321/740	500/740	100%	None/20
RGB_Room	97.52%	478/740	500/740	100%	None/20
RGB_Mix	91.17%	353/740	500/740	100%	None/20
Gray_Card	92.09%	301/740	500/740	100%	None/20
Gray_Room	95.44%	432/740	500/740	100%	None/20
Gray_Mix	92.98%	320/740	500/740	100%	None/20

Note:

The accuracy is derived by the mean of all detected object confidence rate. Among 740 target images, 240 Images are not expected to detect anything as it is not train for it.

Evaluations

We are proud to say that our model can generalize the object quite well as is no error in terms of correctness and noise. This shows that the image did not detect any image wrongly. Besides that, Gray Models detected less objects as compare to RGB Models, the possible reason for such phenomenon is probably due to loss of color information. Apart from that, the accuracy works better on 'room' dataset, the reason is most 'card' dataset fail due to inconsistent of contour process. Finally, 'mix' Model does not seem to perform that well than expected too.

Final Decision Model: **'RGB_Room' model** is use for our challenge.

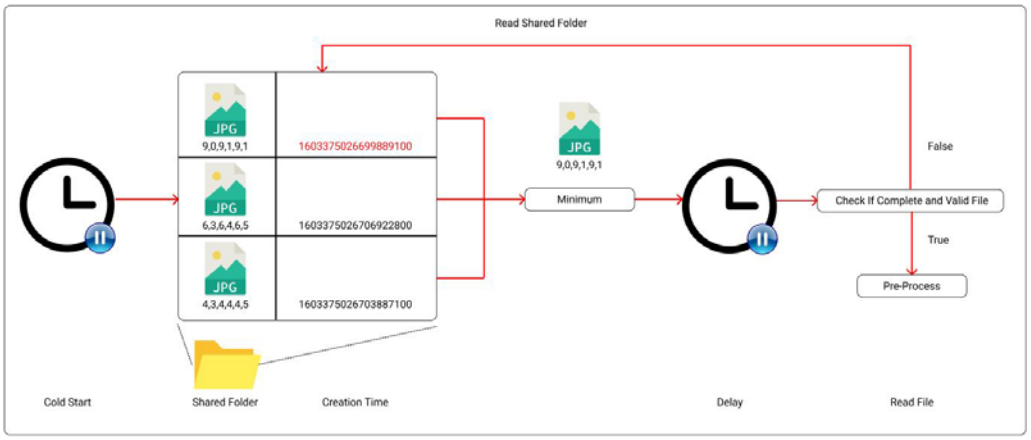
RPI-PC

Introduction

We begin by answering question about what we require to know. Based on our understanding how can we achieve the objective. Generally, RPI served as a communication hub to transfer data between different modules. We start off getting RPI's PiCamera and guideline given by the school such as OpenCV. Next, we decide on where to run our model. We start off with RPI only approach where model is run on it and then move on with RPI-PC approach as we realize RPI is not feasible due to following reason.

During run time testing we encounter crashing and non-responsive behavior from RPI. This impose a high risk for RPI process, consider the fact there will be other thread concurrently running for other components of the robots. Thus, we decide to use PC and execute our model instead.

Communication or Data Transfer



We will briefly touch about the data reading from RPI and data to send/received using TCP/IP Protocol.

Consideration

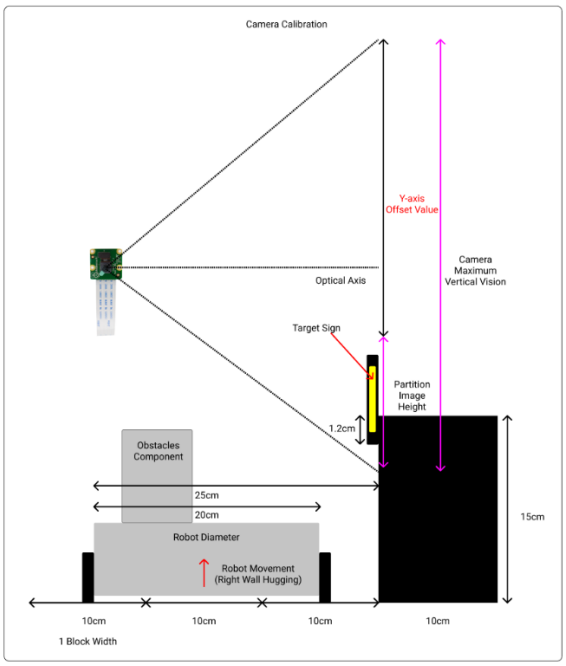
- No conflict in read and write.
- All unused resources should be freed, to prevent access denied error.

Implementation

- Pre-Requisite
  - Samba 4.13.0. It is a server/client architecture, it allows host to use a share file in the same network for client(s) to use. In our context RPI is host, PC is client(s)
- Reading Process
  - Cold start
    - As image takes time to populate the shared file as robots need to move and wait for capture command. Therefore, we decide to put a cold start to prevent read/write error. This operation is executed at the start of the exploration once.
  - Creation Time
    - Our filename consists of coordinates of left, middle and right position of the image in the arena, by default, the read order is depending on filename, this will result in inconsistency when reading the file and live map will update randomly. Thus, there is a need to read from creation time such that the order is First Come First Serve (FCFS) basis
    - Next, we choose the earliest creation time base on FCFS principle, then we put the data on hold again to prevent reading too early.
    - We need to ensure that the file is complete or valid. We discard the file if it is not true.
      - Occasionally, discard fail happens, a solution was to keep track of non-discarded able file, skip them when we re-read the directory.
    - Once the file is read, we pass the data for pre-process
  - Result
    - This solution was implemented at the very late stage of our MDP lifecycle as we did not suspect this issue at the beginning and took us many trial and error before we discovered it. Thus, we did not manage to find the optimal delay time and cold start. We put the safer timing which is 45 seconds cold start and delay time of 1 second.

Image Detection and Recognition

One-Time Calibration



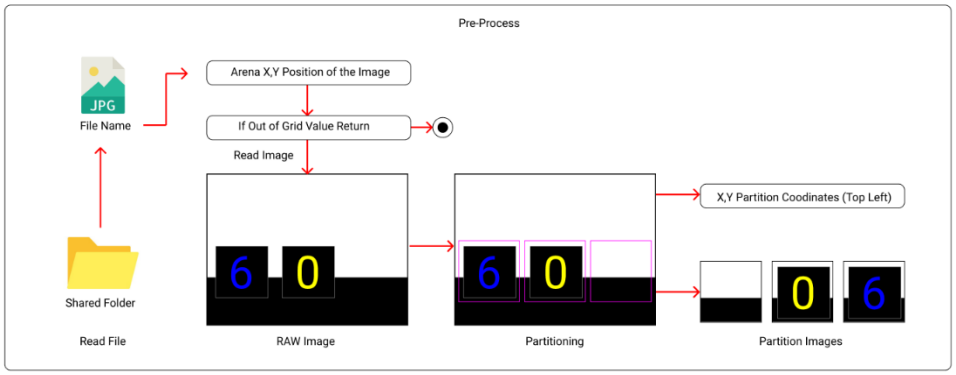
To capture all features of the images correctly, we need to choose our y-offset for cropping carefully. This help us to adjust our camera's height freely should we encounter occlusion by other means.

Implementation

By doing calibration, we can find out the y-offset values that allow us to see all features of the targets sign systematically. We use this feature called Contour to find the y-offset value. It utilizes image curve joining all the continuous point along the boundary with the same intensity level.

Note: It is recommended make sure there is no other feature in the background as shown below, this is because contour algorithm may priorities other contour as the maximum circumference which may pick up the wrong features.

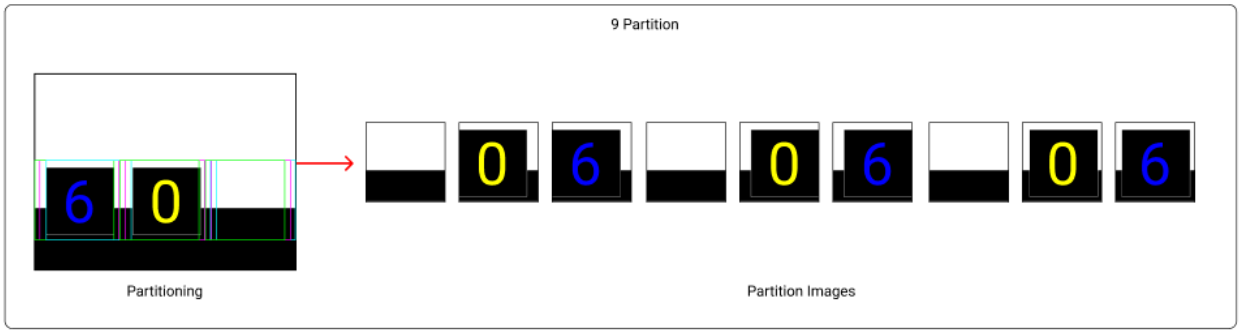
Preprocess



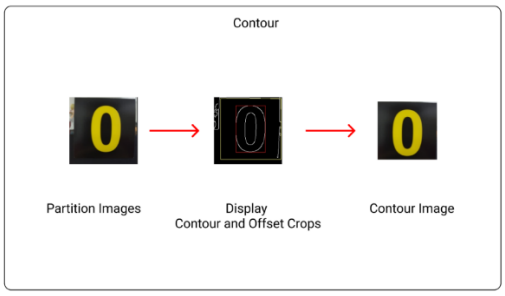
Consideration

- Our process images must be like our dataset images to yield highest accuracy.
- Minimize Background Noise with cropping mechanism.
- Ease of identify coordinates

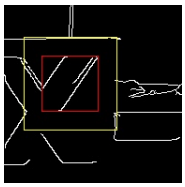
Discarded Features



It offers a wider range of detection. However, it introduces noises as more partition images is process, the likelihood of detecting a noise target. It also takes longer time to process as most of the images are have no target signs, it will process all 9 partitions in worst case most of the time. The process time was an issue during our trail run as the robot finish exploration while our image processing have yet to finish. Thus, this feature was not considered at the end.



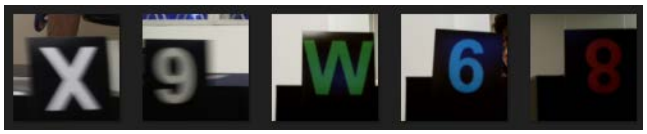
Contour help us further minimize the noise by using the highest contour in our image which is our target signs. To begin process contouring, the images must be converted to binary image to return best outcome. We use canny edge detection before running contour. Based on the result, contour is not perfect as it is unstable to use. A case as shown below. Thus, this feature was not considered at the end.



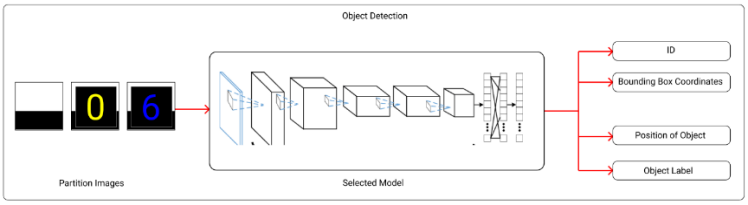
Implementation

- If the 'jpg' image is taken outside arena (out of grid), it will not be processed and discarded immediately.
- Otherwise, the image will be partitioned into 3 crops and 3 top left coordinates will also be stored for plotting process.

Result



Detection



Consideration

- Accuracy threshold finding the optimal value such that minimize noise while remain highest accuracy.
- Separate loading model and execute model as load time takes about 15~200 seconds, it subjective to GPU's capability.
- Each crop image must not have more than 1 object detection, we use the highest score of the detected objects.
- Partition images not detecting any object will be classify as label as 'none' and ID as '0'

Implementation

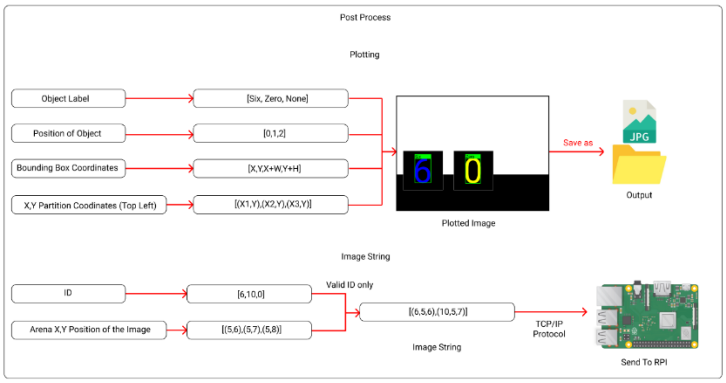
- Partition Images will be process in an order to the position of the RAW image (left, mid, right).
- The detection process will return 4 variables for further process.
    - **Object ID** to be use for image string.
    - **Bounding box** consist of top left, top right, and bottom left coordinate value respectively. This is use for plotting purpose.
    - **Object Position** is to keep track of which position to plot on the RAW image.
    - **Object label** is used for plotting.

Result


(This is for illustrate purpose, not plotted in this process)

We set our optimal detection threshold to 90% and above.

Postprocess

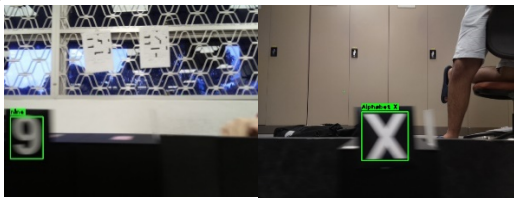


Consideration

- The right datatype and format image string for RPI's receiver
- The plotting visibility and correct bounding box and label, the label must match the ID

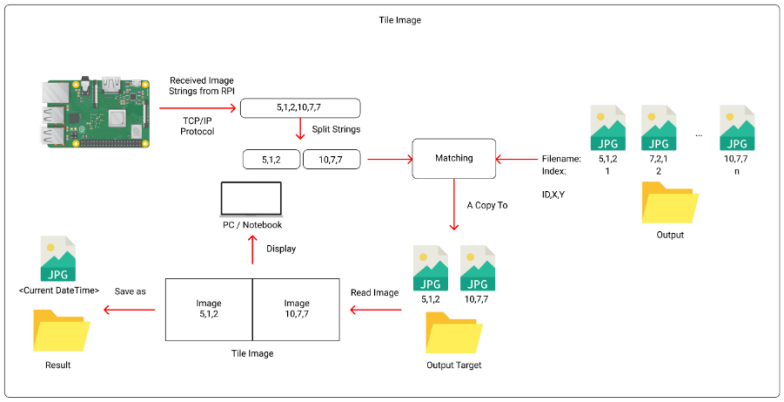
Implementation

- Plotting
    - Bounding box, require 3 inputs attributes
        - **Object position** such that we know to plot left, mid or right
        - **Bounding Box coordinates**, this is to ensure the size of the box is plotted onto the target signs
        - **X, Y Partition coordinates**, it acts as a reference point to where the bounding box to be plotted and at which position.
    - The label, require 1 attribute
        - **Object label**, this is to label on top of the bounding box correctly and another filled box will be plotted at the back to ensure the text is visible regardless of the background. The filled box size will proportional to the label size.
    - The plotted images will then be saved into an output folder for further process.
- Image String
    - It requires 2 inputs which follows the rule needed for Nexus 7 to display the right image string as the message box.
        - **Object ID**
        - **Image position coordinate in the arena** left, mid and right respectively.
    - The valid IDs will be accepted and recompose into image string format to be ready for send out to RPI for Nexus 7 to received.
    - Result
        - Plotting



        - Image String
            1.
```
Image String Output:
9,4,19
```
            2.
```
Image String Output:
13,14,11
```

Tile



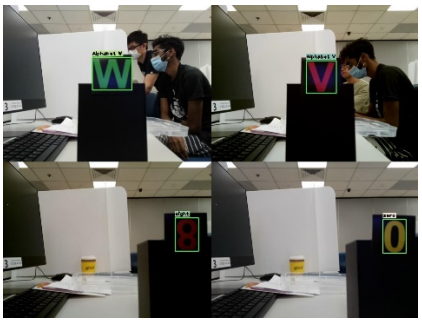
Consideration

- Ensure image string is correct and exist in our output folder images' filename
- Received correct image strings from RPI and tile accordingly

Implementation

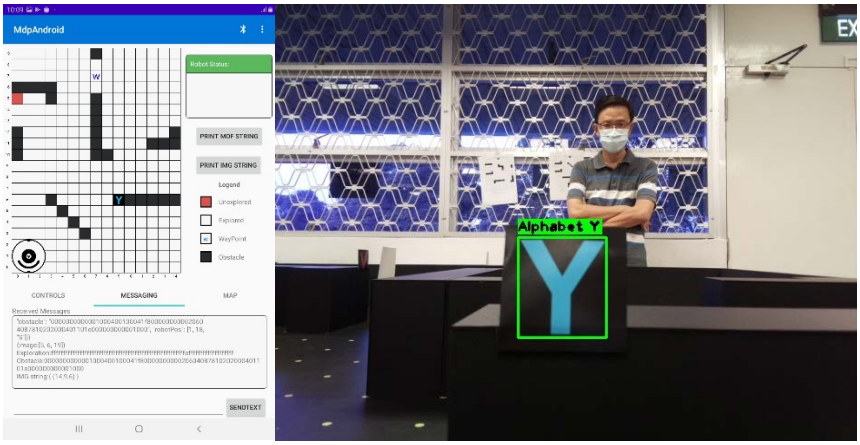
- This process trigger when the exploration end, the Nexus 7 will send a final image strings consist of the approved image strings after their filtering process.
- The image strings received will need to break down into <ID,X,Y> form, this data will then traverse with our output folder and create a copy to output target folder. This process is repeat until there are no more image strings.
- The output target folder contains the shortlisted image to be tiled and the tile image will be placed in result folder to be ready for examiner to verify.

Result



Leaderboard

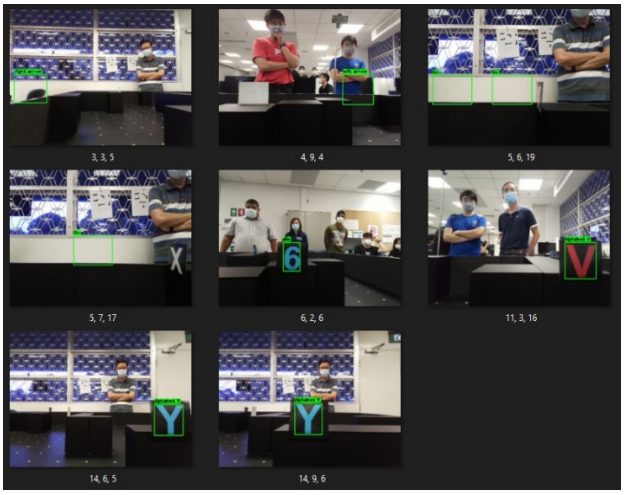
Result



1 correct image (Alphabet Y) and coordinate

Remarks

Unfortunately, we didn't manage to process all the image within 5mins+ before fastest path begins. The final output with all detected images as shown below. If we process finish within 5mins+. Our expected result would be 2 correct images (Alphabet V and Y) and coordinates after filtering from Nexus 7.



Room for Improvement

1. Using a different model such as YOLO.
2. Using better filtering process to reduce noise further.
3. Better capture imaging during turning movement for Arduino Team.
4. Less performance dependency from other robot components such as Algorithm and Android.
5. Reduce image process delay down time.

Conclusion

All in all, we would like to thank everyone for the supports especially our teammates during this tough and exciting journey. Despite all the issues we faced, we still manage to make it this far! Great Teamwork! Bravo! Credits goes to, Image Recognition Team: Lin Yue and Sam.