Assignment 1: ROS Services and Launch Files

What to do

- Create a node named 'array_server' with a ROS2 service that accepts an array of fixed length (length of 8 elements) as its request.
- The service should return the average of the requested array as its response.
- Create a client node named 'array_client' that requests the average of the following array from your server: [10, 0, 5, -20, 13, 7, -9, -6]
- You should create a custom srv file for your communication between the client and the server.
- You should create two launch files to launch the client and the server nodes.

What to submit

Submit your src folder in a zip file. In addition, you should include screenshots of your client and servers running on two terminals simultaneously.

Assignment 2: Motion detection

What to do

1. You will have the video file intersection.mp4 in the videos folder. Split the video into images at 5 frames per second. Use the Linux program '$ffmpeg$'. Let's call this the dataset.

2. Write a Python file to process the generated images. In your Python file, read the first image from the dataset as background $B[0]$. Convert $B[0]$ to grayscale.

3. Write a loop to read all the other images in grayscale one by one. Each image you read will be $X[k]$. Apply the following equation.

$$B[k+1] = B[k] + c(X[k] - B[k]), \tag{1}$$

   where $B[k]$ is the current background image, $B[k+1]$ is the updated background, $X[k]$ is the new image you read in the loop, and $(X[k] - B[k])$ is the difference between the current background image and the image you read.

4. The monadic operation $c(.)$ is described below:

$$c(x) = \begin{cases} \sigma, & x > \sigma \\ x, & -\sigma \le x \le \sigma \\ -\sigma, & x < -\sigma \end{cases}$$

5. Use Python *numpy* library's *clip* function to easily apply the above function to your image $(X[k] - B[k])$.

6. Inside your loop, 1. At every iteration, save the image $(X[k] - B[k])$ in a separate folder, 2. concatenate the images $(X[k] - B[k])$ at every iteration in a variable and call it motion M, where

$$M[k+1] = M[k] + (X[k] - B[k]).$$

7. M[0] can be an empty image with only white-color pixels.

8. After the end of your loop, save both the updated background and the motion variable as two separate images.

9. Use ffmpeg to concatenate your individual motion images into a video.

10. (optional) Develop a colorized version of the motion image.

What to submit

Submit the background image, the motion image, the motion video, and Python code.

Assignment 4: Implementing CNN

Padding

Padding adds extra border pixels (typically zeros) around the input map before convolution, preventing edge information loss and controlling output size.

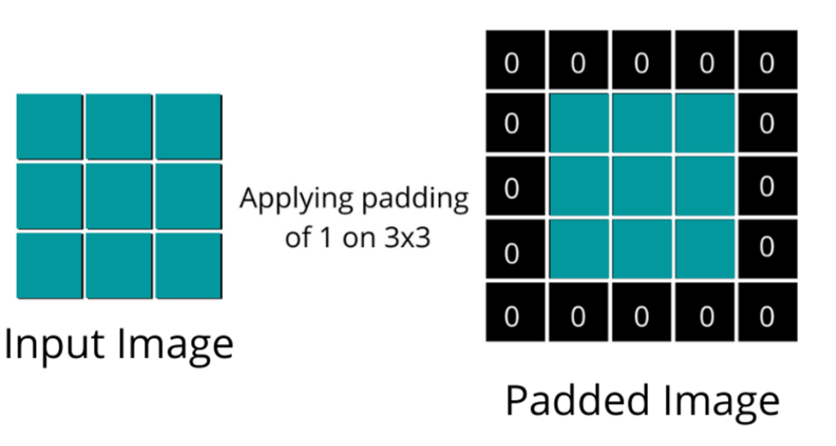

Figure 1: Padding

Strides

In convolutional neural networks, strides define the step size by which the filter (kernel) moves across the input feature map during convolution.

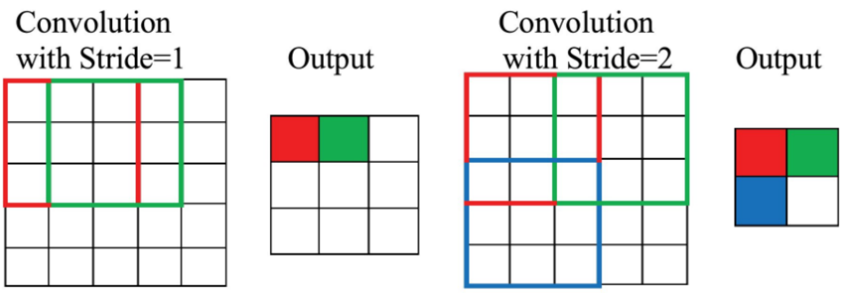

Figure 2: Strides

The following formula lets you map the output to the input with strides and padding.

$$\text{Output Size} = \left\lfloor \frac{\text{Input Size} + 2 \times \text{Padding} - \text{Kernel Size}}{\text{Stride}} \right\rfloor + 1$$

- Input Size: The height or width of the input feature map.
- Padding: The number of zero-padding layers added to the input borders.
- Kernel Size: The height or width of the convolutional filter.
- Stride: The step size by which the filter moves.
- $\lfloor \cdot \rfloor$: The floor function, rounding down to the nearest integer.

Implement the following CNN in Figure 3 using Tensorflow. Make the following assumption:

- Number of the classes at the output layer are 10.
- Assume the activation function is 'ReLu' for all the layers, but 'softmax' for the output layer.
- Assume the pooling layers are max pooling with 3x3 layers.

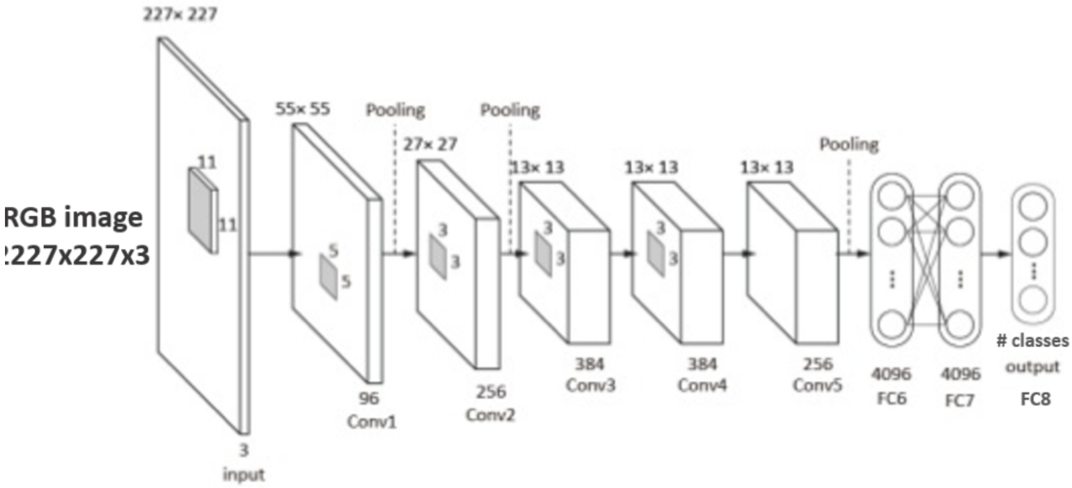

Figure 3: CNN

Final Project

Objective: At the end of the final project, you will have successfully implemented a YOLO object detection model on a mobile robot that controls via Raspberry Pi.
Goal: You are a member of the special police in the NYC. Successfully navigate the robot through the maze to identify the bad guys.

Training Steps

- Segment the video and tag them to their respective classes. Construct your Train set, Validation set, and Test set. I prefer this simple program, but there may be better options. **You are not allowed to use the first 2 minutes of the video for training or testing.**
- You will tag two objects : vehicles and bad-guys
- Train a YOLO by Ultralytics on the tagged images.
- Extract the weights in an appropriate format.

Implementation Steps

- Construct the GoPiGo robot and attach the Raspberry Pi. Connect the camera module to the Raspberry Pi.
- You can burn a base Ubuntu image on the RPi, connect the RPi to the internet, install ROS2 through this, and GoPiGo controls through this. **The alternative way would give you control over the libraries you install, hence it would help you debugging if needed**.
- Install YOLO through Ultrlytics.
- Copy your trained weights to the RPi for inference.
- Write a Python script to read the serial images from the camera. Broadcast them via a ROS2 topic / service.
- Write another Python script to implement the YOLO inference. That script should also subscribe to the ROS2 topic / service to receive the images.
- Improve your Python scripts to recognize AprilTags and calculate the distance between the robot and the Tag.
- A third Python script performs GoPiGo controls depending on the results provided by the inference procedure.

Progress Tracking

- You have the option to work either by yourself or in groups of two. This is the signup link.
- You should have a GitHub repository to host your code. Your progress and contributions are tracked via github. If you decide to keep your repository private, you should grant me (chandima-ccsu) access to your repository.

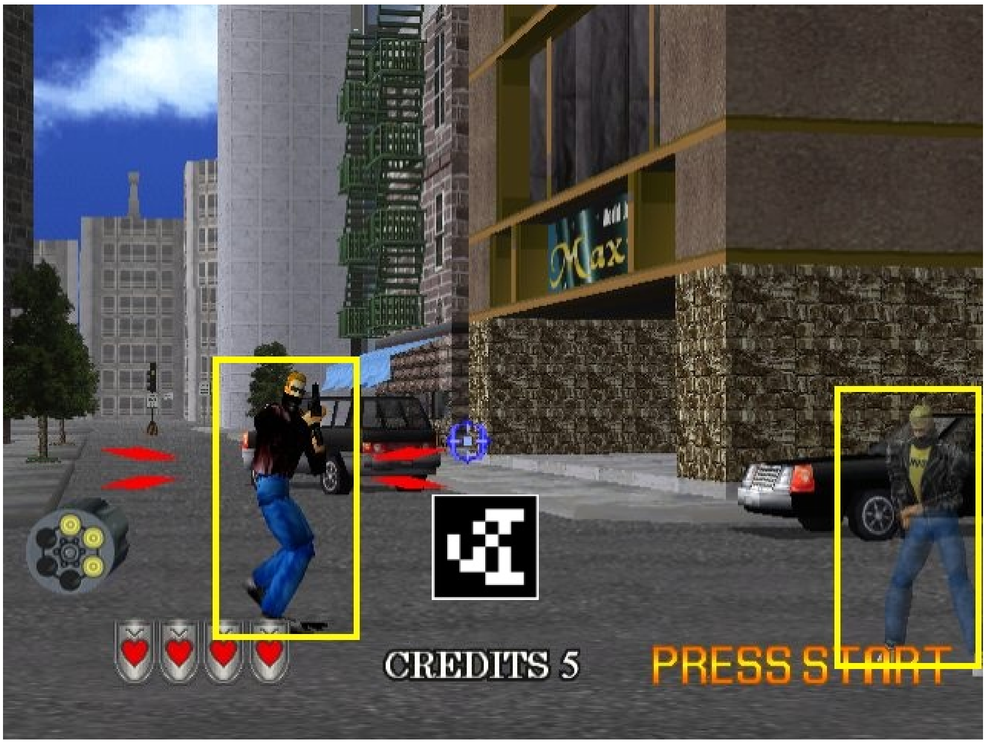

Figure 4: Inference time image with an AprilTag.

Grading (subject to adjustments)

Total available points: 50.

- Successfully Train a YOLO model (10pts).
- Implement the YOLO model on the RPi and correctly perform inference on test images (10pts).
- Successfully implement ROS2 Nodes for image communication and message passing on RPi (10pts).
- Successfully control the GoPiGo robot through the RPi (5pts).
- Successfully reading the AprilTags (5pts).
- A GitHub repository with a reasonable commit history, a README.md file with results of each of the above steps, and instructions to operate the robot (5pts).
- Working demonstration video (5pts).