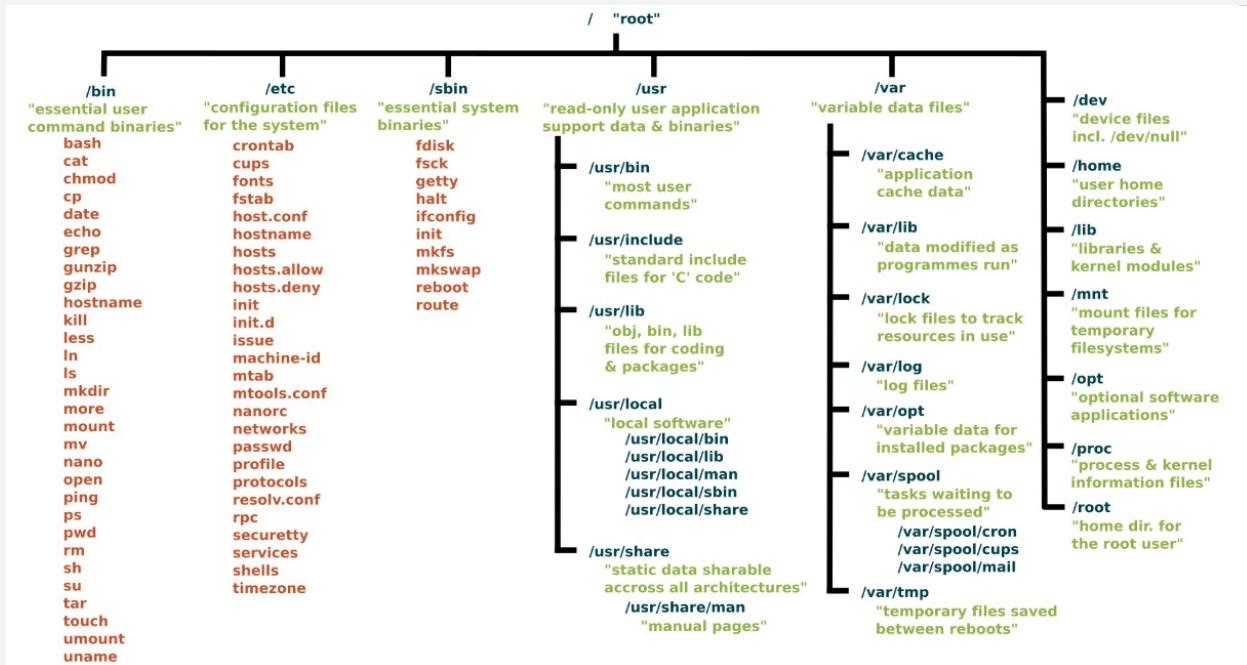


ENPH 353 Logbook

LAB 1 - Jan 14th

OVERVIEW AND COMMANDS



There are so many commands to be explored, one thing that I am doing is setting up an Anki Deck which I shared with classmates over Discord. Based off the command line tutorial Miti sent: <https://www.youtube.com/watch?v=oxtRxtrO2Ag> I will add commands to it as they come up as something that I would like to add to my memory recall.

Notes on setting up Linux on SSD

I decided to use ZSH for the terminal instead of bash to have a more powerful autocomplete option while learning some of the commands on Linux. Some of the differences in validating gazebo involves accessing different configuration file extensions than the bash version.

VALIDATION for ZSH:

```
source ~/enph353_gym-gazebo-noetic/gym_gazebo/envs/ros_ws/devel/setup.zsh
cd ~/enph353_gym-gazebo-noetic/examples/gazebo_cartpole
python gazebo_cartpole_v0.py
```

I also made some adjustments to the current desktop by using GNOME which
ADJUSTING WITH GNOME:

XDG_CURRENT_DESKTOP=GNOME gnome-control-center

htop - interactive process monitor that will be useful for competition

use the f4 command to filter out for gazebo

These problems can be resolved using the Python config file that Miti sent, will look into it:

<https://note.nkmk.me/en/python-pip-install-requirements/>

Notes on Backing Up System

As part of the process of deciding to run dual boot I had to resolve a system for recovering my computer setup if something bad happens. Linux offers a lot of power to the user but with great power comes great responsibility. As a beginner I don't find it so bad that Linux has a lot of different aspects to learn, but rather it can be an unforgiving learning environment.

- Back ups are setup on a 6 TB MyPassport from WD found in parents basement. Differential backups are being used which takes up a bit less space. Although after a certain number of them a fresh base image is made again. I have still yet to test the recovery option from ARCONIS which is the free license I have.
- True backup: on double copies on 2 different systems. If I am in a professional position with my computer I would consider paying additional \$\$ to backup into a cloud solution as a backup to the physical backup. My life is uninteresting enough that security is not a concern.

-Linux Back up- The 128 GB high speed USB is likely the best solution, since the larger Windows OS installation is on the WD external drive already. Mixing Windows and Linux partitions on my backup device seems like a terrible plan. Will look into some tools for doing this.

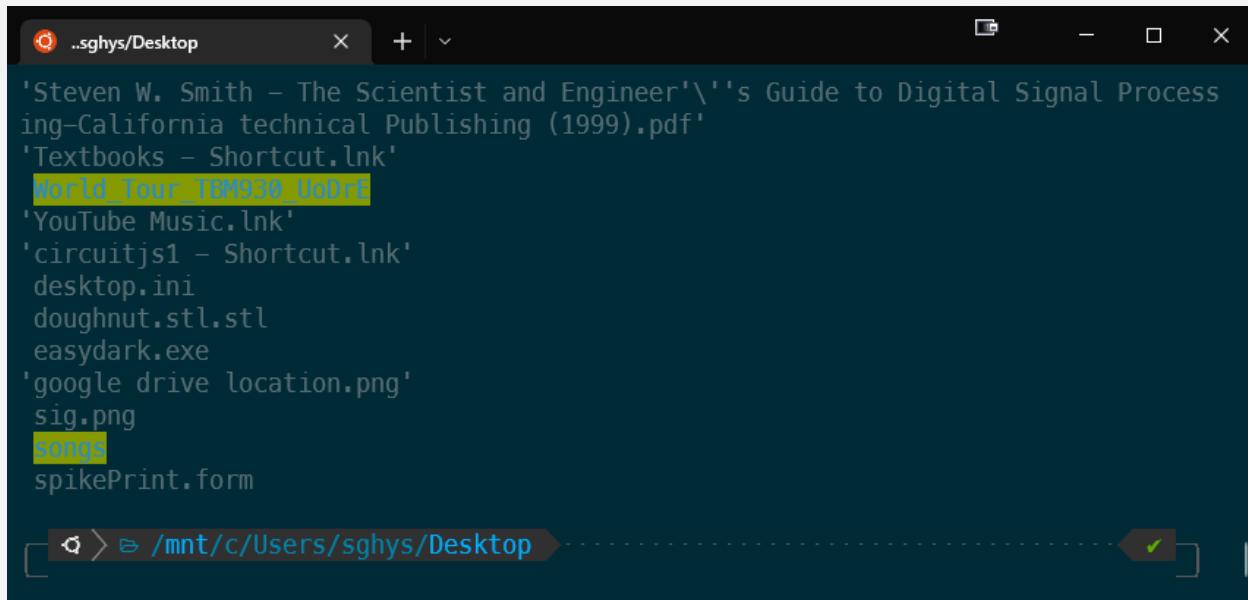
TODO

Setup backups on USB

https://sourceforge.net/p/clonezilla/discussion/Clonezilla_live/thread/a325efbc/

ZSH Shell

I started out with a command line tutorial video that recommended using zsh, which I combined with oh-my-zsh and powerlevel20k. I am quite happy with the visual results that it gives.



A screenshot of a terminal window titled '.sghys/Desktop'. The window shows a list of files and folders in the current directory: 'Steven W. Smith - The Scientist and Engineer''s Guide to Digital Signal Processing-California technical Publishing (1999).pdf', 'Textbooks - Shortcut.lnk', 'World_Tour_TBM930_UoDrF', 'YouTube Music.lnk', 'circuitjs1 - Shortcut.lnk', 'desktop.ini', 'doughnut.stl.stl', 'easydark.exe', 'google drive location.png', 'sig.png', 'songs', and 'spikePrint.form'. The 'songs' folder is highlighted with a yellow background. The bottom of the window shows a command line prompt: [] > [] /mnt/c/Users/sghys/Desktop [] . The path '/mnt/c/Users/sghys/Desktop' is highlighted in blue, indicating it is the current working directory.

<https://github.com/romkatv/powerlevel10k>

It requires working with the .zshrc config files instead of .bashrc

Target:

Eventually I can make a good .bashrc config file with all of my preferences. For now, time resources are very limited with high course loads. It is something to keep in mind for the future.

Preparation for Week 2 Jan 16, 2023

The [Course overview + the shell · the missing semester of your cs education](#) youtube video was watched for more understanding of the Linux shell. The shell is something that is going to be important in the context of the course project but also for future ambitions in software development. Some of the commands from the video were added to the ongoing AnkiDeck that I have

<https://ankiweb.net/shared/info/28677449>

Lab 2 - Jan 20, 2023

Google Collab

Logging in to collab using my regular sghyseli@student.ubc.ca Drive account. When using the Chrome Browser I ran into the issue where it would only open the Drive associated with the current Chrome user. Also Shell command can be run using the ! flag but to actually change directories being used by the shell a %cd should be used as found on this stackexchange post:

The screenshot shows a Stack Exchange post with the following content:

use

193 `%cd SwitchFrequencyAnalysis`

to change the current working directory for the notebook environment (and not just the subshell that runs your ! command).

you can confirm it worked with the `pwd` command like this:

`!pwd`

further information about jupyter / ipython magics:
<http://ipython.readthedocs.io/en/stable/interactive/magics.html#magic-cd>

I'm running into errors trying to set the PATH:

The screenshot shows a Jupyter Notebook cell with the following code:

```
PATH = "/content/drive/My Drive/MI_ENPH253"
labels = !ls "{PATH}"
labels = labels[0].split()
print(labels)
```

An error message is displayed below the code:

```
IndexError Traceback (most recent call last)
<ipython-input-58-ed87cc50c2d1> in <module>
      1 PATH = "/content/drive/My Drive/MI_ENPH253"
      2 labels = get_ipython().getoutput('ls "{PATH}"')
----> 3 labels = labels[0].split()
      4 print(labels)

IndexError: list index out of range
```

At the bottom of the cell, there is a button labeled "SEARCH STACK OVERFLOW".

Jan 22, 2023

Reviewing an intro to Google Collab now, some shortcuts:

- | | |
|--|------|
| <ul style="list-style-type: none">• Click the Play icon in the left gutter of the cell;• Type Cmd/Ctrl+Enter to run the cell in place;• Type Shift+Enter to run the cell and move focus to the next cell (adding one if none exists); or• Type Alt+Enter to run the cell and insert a new code cell immediately below it. | Auto |
|--|------|

completion:

Completions can be opened again by using **Ctrl+Space**.

The grid widget looks pretty awesome for quickly organizing outputs:

https://colab.research.google.com/notebooks/widgets.ipynb#scrollTo=QKk_E6-QRVPW

Use execution history and scratch cells!

Example of Doxygen syntax and conventions

```
Copy code
```

```
#!/usr/bin/env python

## @file
# This is a sample script that demonstrates how to use Doxygen-style
comments
# in a Python script.

## @brief A simple function that adds two numbers
#
# This function takes in two arguments, adds them together, and returns
the result.
#
# @param a The first number to add
# @param b The second number to add
# @return The sum of the two input numbers
def add_numbers(a, b):
    """Add two numbers together and return the result"""
    return a + b

## @brief A simple function that multiplies two numbers
#
# This function takes in two arguments, multiplies them together, and
returns the result.
#
# @param a The first number to multiply
# @param b The second number to multiply
# @return The product of the two input numbers
def multiply_numbers(a, b):
    """Multiply two numbers together and return the result"""
    return a * b
```

The manual can be found at <https://doxygen.nl/manual/docblocks.html>

<https://www.doxygen.nl/manual/starting.html>

 Copy code

```
"""
@brief Saves a video from a numpy array of frames

This function takes a numpy array of frames, frames per second and the
output video path as inputs

and saves the video in the specified location. The function uses OpenCV
to define codec, create video writer and write to the video file

It also raises an error if the frames array passed is not filled or not
2 or 3 dimensional

@param frames (numpy.ndarray) : A numpy array containing the frames of
the video

@param fps (float) : The frames per second of the video

@param pathString (str) : The path of the output video file

@return None
"""
```

Working on Line Following in OpenCV Jan 23, 2023

I am not so familiar with Google Collab and there are so many new tools to use that I am feeling overwhelmed. A compromise that I decided to make that will not detract from learning objectives is to have Chat GPT write me a starter template for the project so that I can get focused on completing the lab in time while seeing an example of a properly written template.

I also found a guide on medium that is explaining some of the basic steps involved in setting up the video processing:

<https://odsc.medium.com/the-beginners-guide-for-video-processing-with-opencv-4e39795b942a>

I also am slightly rusty with Python so I wanted to get a good stub setup for the first function that I am making to work on line detection. ChatGPT is helpful in this respect to get going and start focusing more on the computational aspects instead of spending hours learning new syntax.

 Copy code

```
def line_detection(frame):
    """
    Perform line detection algorithm on the given frame
    @param frame: the frame to perform line detection on
    @return: the processed frame with the line highlighted
    """
    # Your code here
    pass
```

I ran into a specific problem where **denoising all of the frames is really computationally expensive**. The solution was to use dilation followed by erosion referencing the lab lecture slides. The effect was to produce very fast denoising.

https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html

```
#!/usr/bin/env python

"""

Doxygen documentation for this code is provided in the following
format:

@param explanation of the parameter
@return explanation of what the function returns
@fn explanation of what the function does
"""

# Import required libraries
import cv2
import numpy as np

# Load raw_video_feed.mp4
cap = cv2.VideoCapture("raw_video_feed.mp4")

while(cap.isOpened()):
    # Read frame
    ret, frame = cap.read()

    # Perform line detection algorithm
    # Your code here

    # Highlight line on video
    # Your code here

    # Output video with highlighted line
    # Your code here

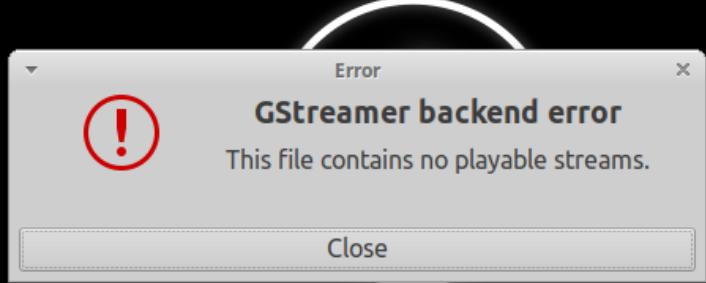
    # Exit video on 'q' key press
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    # Release video capture
    cap.release()
    cv2.destroyAllWindows()
```

I have gotten some black and white line/image processing done but I can't save the frames to video. I am calling it a night for now:

```
param explanation of the parameter
Media Playback Audio Video Tools Help
#
# i
# f
# i
#
# C
# i
# i
#
# e
#
# L
#
# print(width,height)

# Define codec and create video writer
fourcc = cv2.VideoWriter_fourcc('F','M','P','4')
out = cv2.VideoWriter("/content/drive/My Drive/Colab Notebooks/output.mp4", fourcc, 20.0,
```



Jan 24, 2023

One of the main issues with this was that I was using improper codec and that by default the writer expects a color frame which has 3 dimensions.

```
# Define codec and create video writer

fourcc = cv2.VideoWriter_fourcc(*'mp4v')

out = cv2.VideoWriter("/content/drive/My Drive/Colab Notebooks/output.mp4",
fourcc, 20.0, (width,height), isColor=False )
```

Will allow for black and white images to be processed

I've isolated video reading and writing into single functions that serve the purpose of what I need so that I don't run into these issues again. Doxygen for the stubs via GPT

Lab Debrief

Lab 3 ROS

Joystick Package:

<http://wiki.ros.org/joy>

The system setup is fine with some modifications made in order to maintain compatibility with the .zsh shell that I'm using. The file structure for the ROS environment is shown below:

```
source ~/enph353_ws/devel/setup.zsh
```

File Structure ROS

```
└── CMakeLists.txt
└── launch
    └── robot.launch
└── media
    └── materials
        ├── scripts
        │   └── track.material
        └── textures
            ├── monza.png
            └── spa.png
└── models
    └── track
        ├── model.config
        └── track.sdf
└── node
    └── README.txt
└── package.xml
└── README.md
└── urdf
    ├── macros.xacro
    ├── materials.xacro
    └── robot.xacro
└── worlds
    └── 353_ros_lab.world
10 directories, 14 files
```

Folder	Notes
launch	here we define the scripts to launch our ROS applications
media	here we store images, textures and material scripts that we use in our models
models	here we store the 3D definition of the models that we use in our simulated world
urdf	here we store the definition of the robots used in our simulated world
worlds	here we store the definition of our simulated world

ROS Launcher and Commands Summary

\$ source353	* Run the source command for the enph353 workspace.
\$ code	Move to lab directory
\$ tree	Display Contents of Directory
\$ roslaunch enph353_ros_lab my_launch.launch	

Documentation:

<http://wiki.ros.org/roslaunch/Commandline%20Tools>

Tutorial:

<http://www.clearpathrobotics.com/assets/guides/noetic/ros/Launch%20Files.html>

roslaunch is an important tool that manages the start and stop of ROS [nodes](#). It takes one or more `.launch` files as arguments.

1.1 Launch syntax

Most roslaunch commands require the name of a launch file. You can either specify the file path of the launch file, or you can specify a package name and launch file in that package, e.g.

```
$ rosrun rosrun rosrun example.launch
```

or

```
$ roscd rosrun  
$ rosrun example.launch
```

`rosrun <package-name> <launch-filename> [args]`

Launch `<launch-filename>` located in `<package-name>`, e.g.:

```
$ rosrun rosrun ros_tutorials talker_listener.launch
```

`rosrun` will find a file with the matching name inside the specified package and run it.

`rosrun <launch-file-paths...> [args]`

Launch the file(s) specified by relative or absolute paths, e.g.

```
$ rosrun rosrun pr2_robot/pr2_bringup/pr2.launch
```

`rosrun - [args]` **New in Indigo**

Launch the `rosrun` XML passed as standard input, e.g.

```
$ rosrun rosrun package generate_launch | rosrun -
```

`-p port`

If you launched `roscore` on a different port using the `-p` option, you need to pass in the `-p` flag to `rosrun` as well, e.g.

```
$ rosrun rosrun -p 1234 package filename.launch
```

This will dynamically override the port setting in your `ROS_MASTER_URI`.

Robot Creation

URDF Specifications for XML:

<http://wiki.ros.org/urdf/XML>

- [robot](#)
- [link](#)
- [joint](#)
- [gazebo plugins](#)

Using macros:

[XACRO tutorial](#)

Adding Plugins

There was quite a bit of grief from not understanding the process and naming conventions. When adding robot parts it is important to check the documentation for links and joints. Use <gazebo> to encapsulate the simulation portions of the robot plugins. Inside the parameters list it is important to check that links and joints and name calls are correct or else it just throws a segmentation fault in terminal without much to work with.

To check that your camera plugin is working first check that an image_raw topic is being published when you launch the Gazebo world.

Command	Notes
\$ rostopic list	You should see a set of robot/camera/image_raw topics listed.

Then with the Gazebo world running in a terminal open an additional terminal tab in the current terminal window (Ctrl+Shift+T) and run the display image ROS Qt app.

Command	Notes
\$ rosrun rqt_image_view rqt_image_view	You should see an image of the playing surface from the point of view of the robot. You might have to select the robot/camera/image_raw topic from the drop-down menu.

To check that the skid steer driver plugin is working similarly to the camera plugin, launch the Gazebo world (if it is not already running) and first check that a cmd_vel topic is published.

Command	Notes
\$ rostopic list	You should see a cmd_vel topic listed.

Then we have a couple different ways to check that the robot is drivable. From a new terminal run one of the two commands below:

Command	Notes
\$ rostopic pub /cmd_vel geometry_msgs/Twist "linear: x: 0.0 y: 0.0 z: 0.0 angular: x: 0.0 y: 0.0 z: 1.0"	NOTE: This is a fairly long command. After you type 'rostopic pub /cmd_vel' hit Tab multiple times for the built-in command completion app to fill in the missing details. After running this command the robot in the Gazebo world should start spinning CCW.
\$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py	This command will start the teleop_twist_keyboard plugin from the ROS package with the same name. The teleop_twist_keyboard plugin converts keyboard inputs into ROS /cmd_vel commands. Follow the instructions on the screen.

I added an alias to the ~/enph353_ws/devel/setup.zsh for teleop keyboard control that will make it easier:

TELEOP- for keyboard control

CAMFEED-for camera access

Packages and Nodes

A package is a folder under source that is a collection of nodes. The nodes are Python files that can be

called by the launcher or at the command line.

In the launch file I have put

```
<node name="image_processor" pkg="test_package" type="image_processor.py" />
```

This has the package name, node name, and the file to load for the node. **Importantly, the .py file requires executable chmod +x status before it can run.**

```
#!/usr/bin/env python3
import random

import rospy
from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist

def image_callback(image_data):
    # Process image data here
    # ...
    rospy.loginfo("Image height: {}, Image width: {} "\ 
                 .format(image_data.height, image_data.width))
    # Publish velocity message
    velocity_msg = Twist()
    velocity_msg.linear.x = .1
    velocity_msg.angular.z = .1
    velocity_pub.publish(velocity_msg)

rospy.init_node('image_processor')
velocity_pub = rospy.Publisher("cmd_vel", Twist, queue_size=10)
image_sub = rospy.Subscriber("rrbot/camerl/image_raw", Image, image_callback)
rospy.spin()
```

To test and launch use

```
$ roslaunch test_package move_robot.launch
```

Code Debrief so Far:

rospy library always is going to need an import. Next we have objects that are representing the data structure used to send images and control instructions. The subscriber function is given a rostopic to listen for data/ When it gets a data input it calls the passed function with the datatype

Importantly for debugging and outputting data, use rospy.loginfo()

```
rostopic echo rosout      ;   rqt_console
```

Either command will display the output with rqt_console giving a streaming view

To see the network graph:

```
$rosrun rqt_graph rqt_graph
```

The node interaction between the publisher and subscriber would make a lot more sense in a properly encapsulated class structure. This way the image callback is able to access the publishing object:

Yes, you can encapsulate this code in a class and guard it with a `__name__` statement. This can help to organize your code and make it more modular.

For example:

```
ruby Copy code
class ImageProcessor:
    def __init__(self):
        self.velocity_pub = rospy.Publisher("cmd_vel", Twist, queue_size=10)
        self.image_sub = rospy.Subscriber("rrbot/camera1/image_raw", Image,
                                         self.image_callback)

    def image_callback(self, image_data):
        rospy.loginfo("Image height: {}, Image width: {}".
                      format(image_data.height, image_data.width))
        velocity_msg = Twist()
        velocity_msg.linear.x = .1
        velocity_msg.angular.z = .1
        self.velocity_pub.publish(velocity_msg)

if __name__ == '__main__':
    rospy.init_node('image_processor')
    image_processor = ImageProcessor()
    rospy.spin()
```

This code defines a class `ImageProcessor` which has a constructor that initializes the publisher and subscriber, and a method `image_callback` which is the callback function for the subscriber. The `if __name__ == '__main__':` statement ensures that the code inside it will only run when the script is run directly, and not when it is imported as a module in another script.

Jan 29, 2023

Summary and Lab Completion

A working model has been made some important information from the lab is summarized below:

\$ source353	The command has been updated to include new aliases and automatically goes to the pwd of ros, running below command
\$ cd ~/enph353_ws/src/enph353_ros_lab	Move to lab directory
\$ roslaunch enph353_ros_lab my_launch.launch	This is to launch the main launcher for ROS which will handle the simulation environment and create the communication network between packages and nodes
\$ roslaunch test_package move_robot.launch	We can also launch individual packages while ROS is launched by invoking the package name and launch file
Aliases added to source353:	TELEOP - for keyboard control CAMFEED -for camera access
rostopic echo rosout ; rqt_console	Importantly for debugging and outputting data, use rospy.loginfo(en

Lab 4 - Neural Networks

Feb 2, 2023

GUI Setup

Qt designer is a GUI based GUI Designer that allows for the graphical layout of various object types similar to the JAVA SWING library. It produces a scripting file that can be used in conjunction with Python libraries in order to instantiate a window that has the layout and labeling of objects created in the Designer.

Main Documentation Page:

The Qt model uses the publish-subscribe design model common to other frameworks such as ROS. The documentation for the communication framework between the UI Objects can be found at:

SIGNALS and SLOTS mechanism

Example of implementation:

6. In our case to we connect the button "clicked" signal to our browse_button slot function by adding one connect line to our class constructor as below:

```
01 def __init__(self):
02     super(My_App, self).__init__()
03     loadUi("./SIFT_app.ui", self)
04
05     self.browse_button.clicked.connect(self.SLOT_browse_button)
```

and then adding the slot function to our My_App class as below:

```
01 def SLOT_browse_button(self):
02     dlg = QtWidgets.QFileDialog()
03     dlg.setFileMode(QtWidgets.QFileDialog.ExistingFile)
04     if dlg.exec_():
05         self.template_path = dlg.selectedFiles()[0]
06
07     pixmap = QtGui.QPixmap(self.template_path)
08     self.template_label.setPixmap(pixmap)
09     print("Loaded template image file: " + self.template_path)
```

Here the dlg variable is used to instantiate a File Dialog object which we then run by calling the dlg.exec_() function. The dialog event loop will block our app event loop for the extent the File Dialog object is running. The File Dialog returns the absolute path of the selected file. We use this path to load the image file and display it.

The buttons can be connected to a function that is activated upon clicking on them, similar to the MATLAB GUI system that I used on The Launch simulator that I made last year. SLOT_browse_button is used as the function call.

It uses a library function to call a File browser to select a path. Then the pix map creates a pixel mapping of the png which is updated to the template_label object from our GUI

Implementing SIFT

- Key points are identified in the image with an associated descriptor vector. If we can find multiple keypoint associations through the feature descriptors we can match two images.
- Locally distinct keypoints are found using a difference of gaussian blurs. Usually it is done

over images of several different resolutions or scales.

- Descriptor vectors look at the local neighborhood of a found keypoint. Gradients in local regions are collected into histograms. The histogram has 45 degree bins for angle directions and they are compiled together to form regional descriptors

RANSAC

- Random Sample Consensus: Dealing with data affected by outliers
- Valuable for sensor data that has noise or outlier
- Separates data into inliers and outliers
- Sample two data points and see how many other points are in agreement with it, select the model that has the largest number of inliers, then those points are all classified as inliers and that model is selected
- 1. Sample subset of data points, treat them as inliers
- 2. Compute the model parameter
- 3. Score the model based on the subset select, then use the solution with the highest score

Coding Strategy

It is important to be able to convert between various image representations and datatypes. For finding keypoints a CV2 image should be first converted to grayscale using `cv2.cvtColor`, a static object is made that can perform keypoint + descriptor matches on an image automatically and output them generally as an np array.

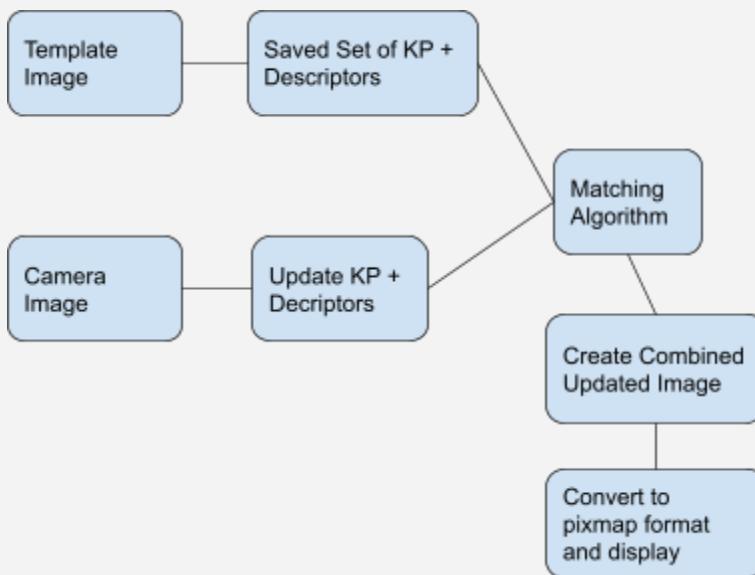


Image importation and camera image processing has been abstracted to functions along with a UI for displaying. The main focus is on processing the descriptors to match keypoints for the image. Initial tests were conducted on the simpler brute force ORB method before trying more complex matching schemes.

Masking Matches for Descriptors:

We use the FLANN algorithm that is provided by openCV to efficiently go through and process the nearest two neighbors for each of the descriptors that we have. More about the parameters for matcher:

tell me more about the dictionary data that is being passed as parameters

The two dictionaries being passed as parameters to the cv2.FlannBasedMatcher object are the index parameters and search parameters.

index_params dictionary is used to specify the algorithm to be used for nearest neighbor search and its parameters. Here, the algorithm used is FLANN_INDEX_KDTREE (0) and the number of trees to use is set to 5.

search_params dictionary is used to specify additional search parameters such as the number of times the trees in the index should be recursively traversed (checks). Here, the number of times the trees in the index should be recursively traversed is set to 50.

These parameters can be tuned for better performance based on the specific use case.

How does the number of trees affect performance and results

The number of trees in FLANN (Fast Library for Approximate Nearest Neighbors) algorithm affects the performance and accuracy of nearest neighbor searches.

More trees generally lead to higher accuracy, but also increase the time required to perform the search. Each tree helps to partition the data in a different way, resulting in a more comprehensive view of the data and thus more accurate nearest neighbor search results. However, having too many trees can result in longer search times, so it is a trade-off between accuracy and speed.

The optimal number of trees depends on the size of the dataset and the desired accuracy, and it is often a matter of trial and error to find the best value for a specific use case.

```

def match_template_query(self):
    # https://www.tutorialspoint.com/how-to-implement-flann-based-feature-matc
    matches = self.matcher.knnMatch(self.desc_template, self.desc_query, k=2)
    # Need to draw only good matches, so create a mask
    matchesMask = [[0,0] for i in range(len(matches))]
    # ratio test to remove bad matches 0.6 is the threshold
    for i,(m,n) in enumerate(matches):
        if m.distance < 0.6*n.distance:
            matchesMask[i]=[1,0]

```

The `matches` variable that is returned by the `cv2.FlannBasedMatcher.knnMatch()` function is a list of `cv2.DMatch` objects.

A `cv2.DMatch` object represents a single match between two keypoints, one from the query image and one from the template image. This object has several attributes that describe the match, such as `distance` (the distance between the two keypoints), `queryIdx` (the index of the keypoint in the query image), and `trainIdx` (the index of the keypoint in the template image).

So we have a list of **cv2.DMatch** data type and we are checking the distance field on them to perform a comparison test.

Implementing Homography

https://docs.opencv.org/3.4/d1/de0/tutorial_py_feature_homography.html

This takes matches and uses matrix transform to match it with the overall shape of the object of interest (square)

Lab 5 Neural Network Building

Feb 9, 2023

1) Nomenclature

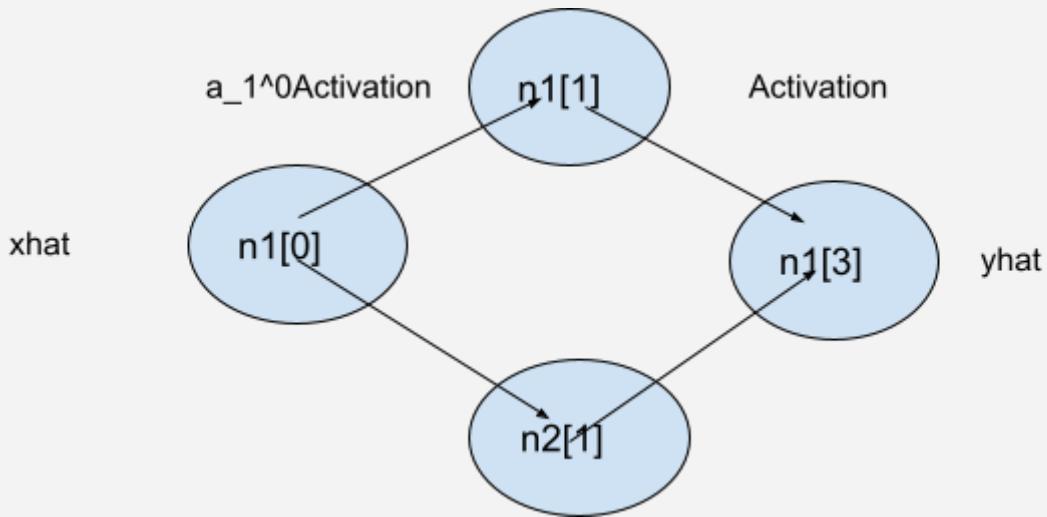
- a) Data X: Function input, Data Y: Expected Output

The data pipelines all needs to be of the same dimension whether it is vectors or images for consistency in the network and to facilitate the matrix transforms.

Example

X = {Array[16x32],Array[16x32],Array[16x32],Array[16x32],.....}

Y = {Vector[0,3],Vector[0,3],Vector[0,3],Vector[0,3],....}



0 is input level and then hidden layers in between leading to output

Each link has an associated coefficient and potentially an activation non linear function applied to it.

$$\text{e.g. } Z = \sum_i a_j^{i-1} w_j^i + b \text{ activates to } Z = a(z)$$

All of the w values are for weights assigned to one of the links

We also have the partial derivatives wrt to each of the weights that are used to produce a gradient descent.

Algorithm

1. Instantiate weights and biases ~ not random
2. Shuffle data as pairs of (x_i, y_i)

3. Forward propagation to get output \hat{y}
4. Back propagation -> Compute loss $(y - \hat{y})^2$ regression
or a binary classification

$$y \log y - (1 - y) \log(1 - y)$$

5. Update weights and biases

From the computed partial derivatives in the loss function $w_{i+1} = w_i - \alpha \frac{dg}{dw}$

The alpha value is the learning rate (It is basically the Eulerian time-step)

6. The forward back propagation cycle is applied over all of the training data in the set.

Then the data is shuffled into a different order until:

- Loss Plateaus
- Accuracy threshold reached
- Validation and training accuracy diverge

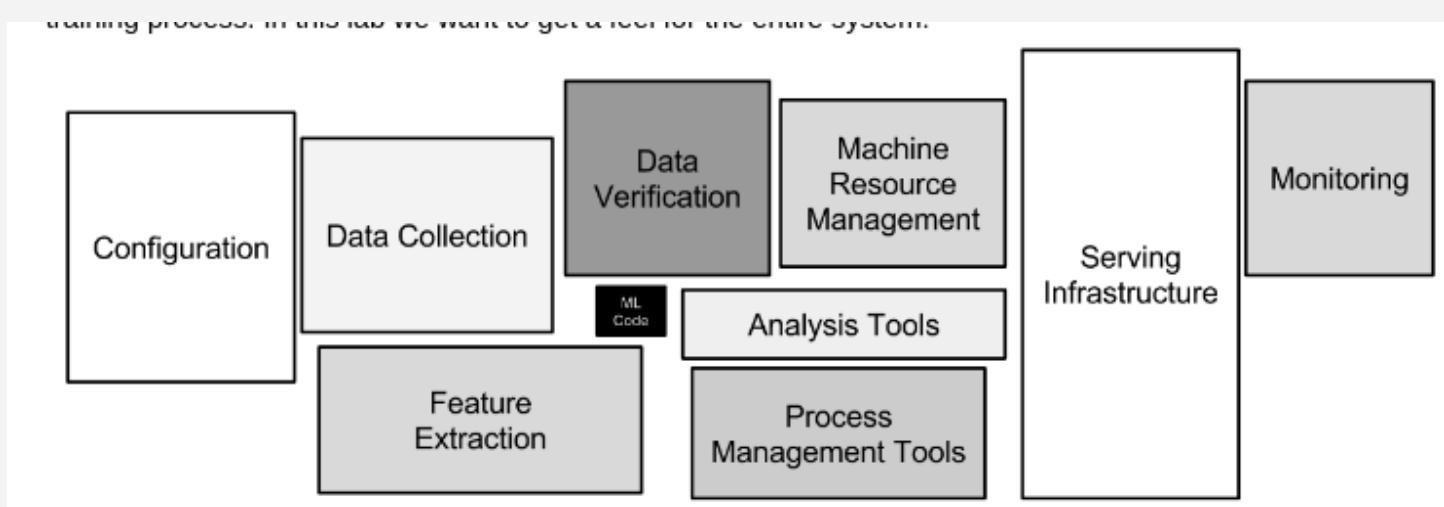
Building a Neural Network

Karptathy Tutorial followed, comments are included in the Jupyter Notebook.

Lab 6 Building CNN

Feb 17, 2023

Transfer Learning could be a useful tool for character recognition



This lab is going to focus in the ML code for implementing a character recognition module

We first need to set up a training data set. For this lab the set is randomized license plate images but something better can likely be sourced for the competition, like a set that is pulled from driving around the simulation environment.

Setting Up the Plate Data and Validating

Plate Processing

The license plates are generated using a random generator that was already provided:

<https://colab.research.google.com/drive/1J7emUeKgChHLKc8nDa3yLBbi-Sonf35V?usp=sharing>

Initially I was going to try and do a structured approach to generating the plates but it seemed like a headache to think of all the different ways to make data evenly distributed. An easier method is to generate troves of data which is not computationally expensive and then trim it as needed.

To start, 1000 plates have been generated and placed into a folder with information about the content contained in the filename convention Plate_AA00.png. This is the plate repository but not the training set, the training set will consist of individual characters. The plates needed slicing that was empirically found by examining the slicing results. A positional indicator is added to each sliced file to indicate where on the plate it came from. This is the raw data set of processed slices.

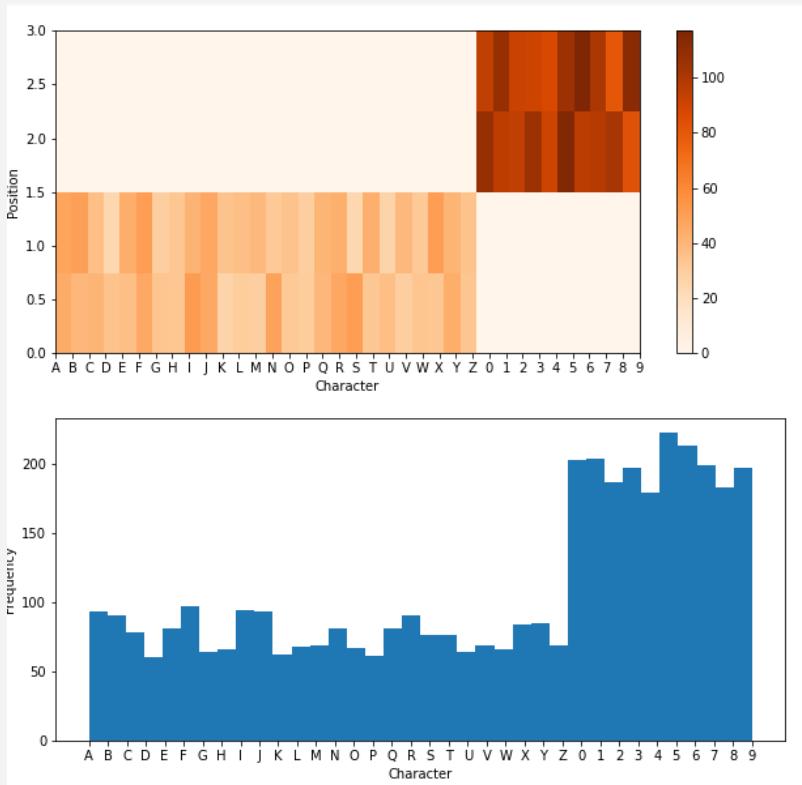
Each of the slices has all the relevant information contained in the naming conventions used for the file. The slices are loaded in and the file strings are parsed to get both the position and character that is matching. We have 3 Data sets

1. The images
2. The characters
3. The positions

The Character set requires further processing into one hot vectors. They really represent the dimensionality of the solution space, where it appears we are working on the unit hyper-sphere and pointing in the most probable direction. The characters fully matching represent the cardinal points on the sphere. There are some delicate and sometimes annoying things with Python typing that make inspection of data crucial using type and shape commands.

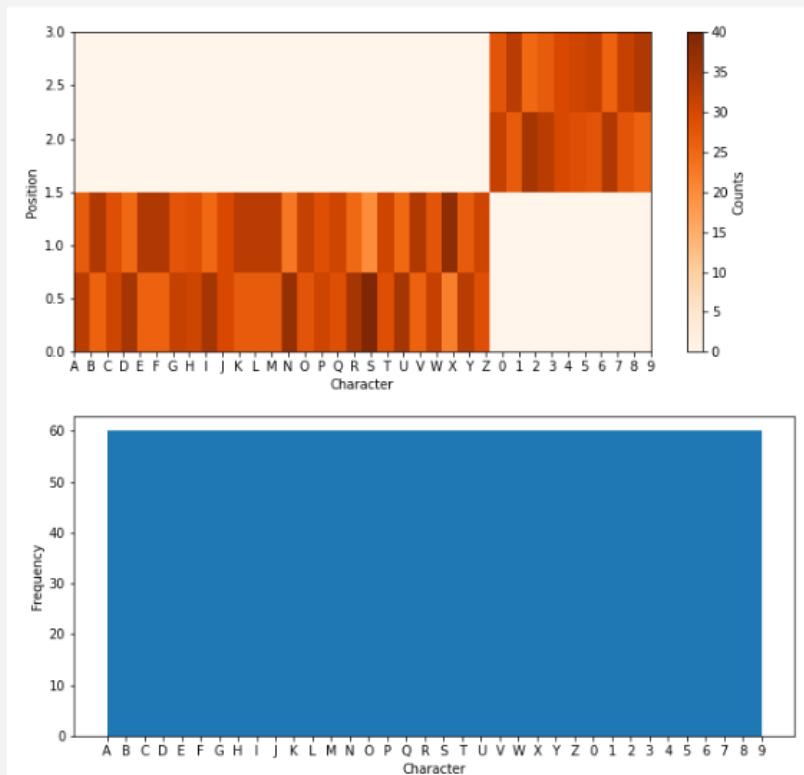
Finally we display the results for human reading and validation as well as the histogram. I elected to make a 2D histogram because it seems like position should matter for reducing bias.

Results:



There are far more digits than letters which is not very good for balancing out set, the next step is to selectively remove and reduce some of these entries to get to a normalized state.

A normalization is applied by generally binning all of the characters and flattening the histogram to the minimum but, then slicing our original data set to a balanced set. The new plot is as follows:



We now have about 60 of each character and a balanced data set to begin the training with

CNN Training

The homogenized data set is then further processed into a training and validation set. We use a character dictionary to ensure that the split occurs at the level of each data type to preserve homogeneity. Later we can add extra samples as needed to particular characters but this is the base set

Run local notebook:

```
jupyter notebook --NotebookApp.allow_origin='https://colab.research.google.com' --port=8888  
--NotebookApp.port_retries=0
```

one epoch = one forward pass and one backward pass of all the training examples

batch size = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.

number of iterations = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Google Collab is very annoying with continuous crash and resets.

I have trained the same CNN countless times and wasted many hours of time on the task. The solution is permanence for the model that has been computed through a JSON

[https://machinelearningmastery.com/save-load-keras-deep-learning-models/#:~:text=Save%20Your%20Neural%20Network%20Model%20to%20JSON&text=This%20can%20be%20saved%20to,the%20symmetrical%20load_weights\(\)%20function.](https://machinelearningmastery.com/save-load-keras-deep-learning-models/#:~:text=Save%20Your%20Neural%20Network%20Model%20to%20JSON&text=This%20can%20be%20saved%20to,the%20symmetrical%20load_weights()%20function.)

```
# serialize model to JSON  
  
model_json = conv_model.to_json()  
  
with open("/content/drive/My Drive/Colab Notebooks/model.json", "w") as json_file:  
    json_file.write(model_json)  
  
# serialize weights to HDF5  
  
conv_model.save_weights("/content/drive/My Drive/Colab Notebooks/model.h5")  
  
print("Saved model to disk")
```

```
from tensorflow.keras.models import Sequential, model_from_json

# load json and create model

json_file = open('/content/drive/My Drive/Colab Notebooks/model.json', 'r')

loaded_model_json = json_file.read()

json_file.close()

loaded_model = model_from_json(loaded_model_json)

# load weights into new model

loaded_model.load_weights("/content/drive/My Drive/Colab Notebooks/model.h5")

print("Loaded model from disk")
```

I made some images that are cut differently to test with but the dataset for this model is very limited. It does not appear to be overtrained and it is reading a test set of plates perfectly– it is even a stretch to apply a confusion matrix in this case

Summary of choices for the NN



Sure, here's an explanation of each layer in the model and the reasons for selecting them:

1. `Conv2D` layer with 32 filters, kernel size of (3, 3), and ReLU activation: This is the first convolutional layer in the model. It has 32 filters to learn 32 different feature maps from the input image. The kernel size of (3, 3) is small enough to capture local features, while the ReLU activation helps introduce non-linearity to the model.
2. `MaxPooling2D` layer with pool size of (2, 2): This layer performs max pooling on the output of the first convolutional layer. It helps reduce the dimensionality of the feature maps while preserving the most important information.
3. `Conv2D` layer with 64 filters, kernel size of (3, 3), and ReLU activation: This is the second convolutional layer in the model. It has 64 filters to learn more complex features from the input image.
4. `MaxPooling2D` layer with pool size of (2, 2): This layer performs max pooling on the output of the second convolutional layer.
5. `Conv2D` layer with 128 filters, kernel size of (3, 3), and ReLU activation: This is the third convolutional layer in the model. It has 128 filters to learn even more complex features from the input image.
6. `MaxPooling2D` layer with pool size of (2, 2): This layer performs max pooling on the output of the third convolutional layer.
7. `Conv2D` layer with 128 filters, kernel size of (3, 3), and ReLU activation: This is the fourth convolutional layer in the model. It has 128 filters to learn even more complex features from the input image.
8. `MaxPooling2D` layer with pool size of (2, 2): This layer performs max pooling on the output of the fourth convolutional layer.
9. `Flatten` layer: This layer flattens the output of the fourth max pooling layer, converting it into a 1D vector.
10. `Dropout` layer with rate of 0.5: This layer randomly drops out 50% of the connections between the previous layer and the next layer during training, helping to prevent overfitting.
11. `Dense` layer with 512 units and ReLU activation: This is a fully connected layer with 512 units, which takes the output of the previous layer as input. The ReLU activation helps introduce non-linearity to the model.
12. `Dense` layer with 36 units and softmax activation: This is the output layer of the model, which has 36 units, one for each character class. The softmax activation function is used to convert the outputs of the previous layer into a probability distribution over the 36 classes.

The reasons for selecting these layers are as follows:

- The convolutional layers are used to extract features from the input image, with increasing complexity as the depth of the network increases.
- The max pooling layers are used to reduce the spatial dimensions of the feature maps, while keeping the most important information.
- The dropout layer is used to prevent overfitting by randomly dropping out connections between the layers during training.

When a `conv2D` layer has 64 filters, it means that the layer is configured to learn 64 different feature maps from the input data. Each filter in the layer applies a convolution operation to the input, resulting in a feature map that highlights a specific pattern or feature in the input.

For example, in an image classification task, the first convolutional layer might learn low-level features such as edges and corners, while the subsequent layers learn higher-level features such as shapes and textures. Each filter in the layer learns a different feature map, which is then combined to form a representation of the input data that is used for classification.

Having more filters in a convolutional layer allows the model to learn a larger variety of features from the input data, which can result in better performance in the task at hand. However, using more filters also increases the number of parameters in the model, which can lead to overfitting if not properly regularized. Therefore, the number of filters used in a convolutional layer is typically chosen based on the complexity of the task and the size of the input data.

Tips for Setting up NN and training

For tracking all the different models that have been used and their performance use this additional software.

<https://wandb.ai/site>

Lab 7 QLearning Mar 2, 2023

Exploration vs Exploitation Policy

Use exploration behavior early on in training in order to gain new information or strategies. The model should change to a greedy or exploitation based one as the training progresses. Epsilon

Mar 5, 2023

Starting again on the lab–

Q-learning: update equation

Given a representation of environmental **states s** and possible **actions a**, how do we converge towards the optimal Q_{ij} value for each action a_j in each state s_i .

$$Q(s_t, a_j) = Q(s_t, a_j) + \alpha \{ R_{t+1} + \gamma \max_j [Q(s_{t+1}, a_j)] - Q(s_t, a_j) \}$$

"Error" = set-point Q_s - current value

R_{t+1} reward for new state s_{t+1}

$\max_j [Q(s_{t+1}, a_j)]$ reward for best action available from new state s_{t+1}

where α is the learning rate

$\alpha = 0$ no learning

$\alpha = 0.9$ learning (fast but can be unstable)

and γ is the discount factor

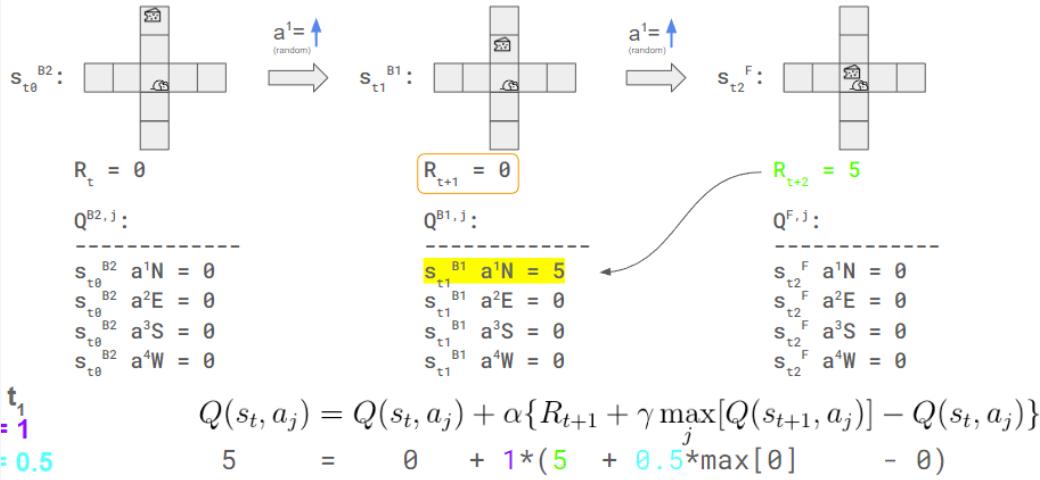
$\gamma = 0$ consider only current rewards

$\gamma = 1$ assign long-term rewards equal importance to current rewards

```
def learn(self, state1, action1, reward, state2):
    """
    @brief updates the Q(state,value) dictionary using the bellman update
    equation
    """
    #     Q(s1, a1) += alpha * [reward(s1,a1) + gamma* max(Q(s2)) - Q(s1,a1)]
    # Find Q value for current state-action pair, gives zero for unknown pair
    current_q = self.getQ(state1, action1)
    max_next_q = max([self.getQ(state2, a) for a in self.actions])

    # Update Q for (state1, action1) (use discount factor gamma for future rewards)
    new_q = current_q + self.alpha * (reward + self.gamma * max_next_q - current_q)
    self.q[(state1, action1)] = new_q
```

1st rollout: propagate reward to previous state-action



Lab documentation Links

[OpenAI's Gym](#)

[Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo](#)

I had an issue with getting the environment setup because my user name is not fizzer, but it was resolved. If it happens again the problem was in the stack call. Going back to the `~/enph353_gym-gazebo-noetic/examples/gazebo_linefollow_ex/gazebo_linefollow_ex.py` It has a filepath that includes 'fizzer' – go back one file call in the stack to change the culprit.

I simply imported my image processing from earlier to get the library of functions needed to complete the todo. We use bins to rescale the **xcoord** down to a value from 0 to 9 and update the state array.

I conveniently built in a no line sentinel value into the function in Lab 3 so that can be used to flag the condition. Fairly quick to do it all.

The implementation for the qlearn dictionary management class was implemented, we have a self updating model. But connecting all the pieces is very confusing or disorienting. I will ask Andrew for some help on Tuesday. I understand the overall concepts of what is going on but overall it feels like too many things that are new at once and I have lost my bearings a bit.

I got some feedback on Tuesday to change one function of code after reviewing it all. I also compared against Hod and found it to be functionally similar. Running the learning algorithm takes time to see if it is working or not. After a half hour of running it was still not working.

Thoughts on Lab 7

I don't feel like I was able to access help for the lab very well. It was the first instance in the course I can think of where I required help, so maybe I am not good at asking for it. On a Thursday when I am

feeling underslept and the lab is particularly noisy as an environment, sitting for 1.5 hours to get help on a problem is a bit of a slow-cooker. I am reluctant to try on lab 8 after what was a multi-day bad experience.

It is not frustrating to have delays in reaching an objective or to not have code work, I thrive on hard problems. It is *stressful* to not have things work and spend a bunch of time on them and fall behind on other coursework for other classes. This is the primary difference between school and work for me.

At the end of term feedback I will leave some suggestion for improvement, using a priority queue for requesting help would greatly improve the system from my vantage point.

Lab 8 Deep Reinforcement Learning

Mar 20, 2023

This is the last lab, hopefully it goes well. To get familiar with the topic again I read all the slides, the lab and watched tensorboard video. All of the packages were installed successfully

Environment

Looking at the `/gazebo_cartpole_v0.py`

File we can see some of the details about the implementation

1. The action space has two options, increase velocity by .2 or decrease by -.2

```
self.action_space = spaces.Discrete(2)

    # Take action

    if action > 0.5:

        self.current_vel += 0.2

    else:

        self.current_vel += -0.2
```

2. Next we have the observation space where we can see that the state is defined as follows:

We have a balancing link with feedback on the state. It is a bit surprising that the x-position is also tracked but it makes sense since Miti brought it up as a potential issue that the balanced state is tilted which will cause the cart to accelerate off to infinity. But surprisingly these have been set to zero in the last two lines of code. This may have been deliberate?

```
# Define state

x = self.data.position[1]

x_dot = self.data.velocity[1]

theta = math.atan(math.tan(self.data.position[0]))
```

```

        theta_dot = self.data.velocity[0]

        state = [round(x, 2), round(x_dot, 1), round(theta, 2), round(theta_dot,
0)]

```

3. The reward is a very simple policy where 1 is given for a continuation state and 0 is given for a termination state.
4. The conditions for reset of the environment are going off the track in either direction which is accomplished by monitoring the x.pos +- 15

There is also a +- 12 degree threshold for the angle theta so that if the pole tips by too much then it resets

Cross Entropy

After downloading the file, the next step is to have a review of the code before doing todo. A Tensorboard notice came up in VSCode so it seems automatically enabled is an option for it when the tensorboardX import is detected.

The tensor board runs at

<http://localhost:6007/>

Wish is important to note since it differs from lab instructions

```

# REMINDER:

# as the last layer outputs raw numerical values instead of

# probabilities, when we later in the code use the network to predict

# the probabilities of each action we need to pass the raw NN results

# through a SOFTMAX to get the probabilities.

```

Defining the Neural Net Class

The Net class is a basic neural network. There is a linear layer that is the input and is the same size as observations. It maps linearly to the 1228 hidden size for the layer then undergoes a relu activation. It introduces non-linearity into the network to allow for learning more complex patterns

Episode rewards and steps

Nest there are some variables for storing the episodes, each step is further defined as an observation

and action, so this gives net reward and the series of actions and observations that lead to the reward

Iterate Batches

This handles the training episodes by iterating through the environment reset then passing the observations as a NN entry.

- The output goes through softmax returning probability of an action from the agent.
- It needs to be converted out of tensor back to NP array and only the probability distribution instead of some other information on gradients we wont use.
- The CNN output is then used to make a probabilistic choice for the next action to take.
- The simulation ticks forward by one step
- Reward is processed and added to the total for the current episode, and the step taken is also added.
- The process resets or terminates depending on if the episode is finished or requires more steps still.

Filter_Batch

One iteration has been done the filtering is implemented

- Extract all of the rewards from the iterations and then use that data to form a percentile cutoff mark
- Identify the mean reward to see how well the overall training is progressing.
- The pairs of observations and actions from the elite sessions are pooled together, then converted to tensors so that they are readable by the CNN
- The calculated values are returned

Code completion

I asked chat GPT to educate me a little on the codebase since I am unfamiliar with Pytorch and have a lot of questions. It is faster than searching up the documentation and for a brief overview like this I think that the output is fairly reliable:

...

Now, let's discuss the optimizer specifically:

1. Optimizer: In the code, the optimizer used is the Adam optimizer, which is a popular optimization algorithm for training deep learning models. It is an extension of stochastic gradient descent (SGD) that adapts learning rates for each parameter individually. The optimizer is responsible for updating the neural network's parameters based on the gradients computed during the backward pass.
2. Zeroing gradients: Before each training iteration, it is essential to zero the gradients of the optimizer using `optimizer.zero_grad()`. This is done because PyTorch accumulates

gradients by default during each backward pass. Zeroing the gradients ensures that the optimizer updates the model's parameters based on the gradients computed only for the current iteration.

3. Forward pass: During each iteration, the neural network performs a forward pass on the elite episodes' observations, generating action scores. These action scores are then passed through a softmax function to get the probability distribution of actions.
4. Calculating loss: The CrossEntropyLoss function computes the loss between the predicted action probabilities and the actual actions taken in the elite episodes. The loss function combines softmax and negative log-likelihood loss in one expression, making it suitable for this problem.
5. Backward pass: The loss is then backpropagated through the network using `loss_v.backward()`. This computes the gradients of the loss concerning each parameter in the neural network.
6. Updating parameters: After the backward pass, the optimizer updates the neural network's parameters based on the computed gradients by calling `optimizer.step()`. This adjusts the weights and biases of the network, helping it to learn the optimal policy for balancing the cart-pole system.

...

Net class defines the NN architecture

We set up our environment variables and then iterate through many episodes through a training loop:

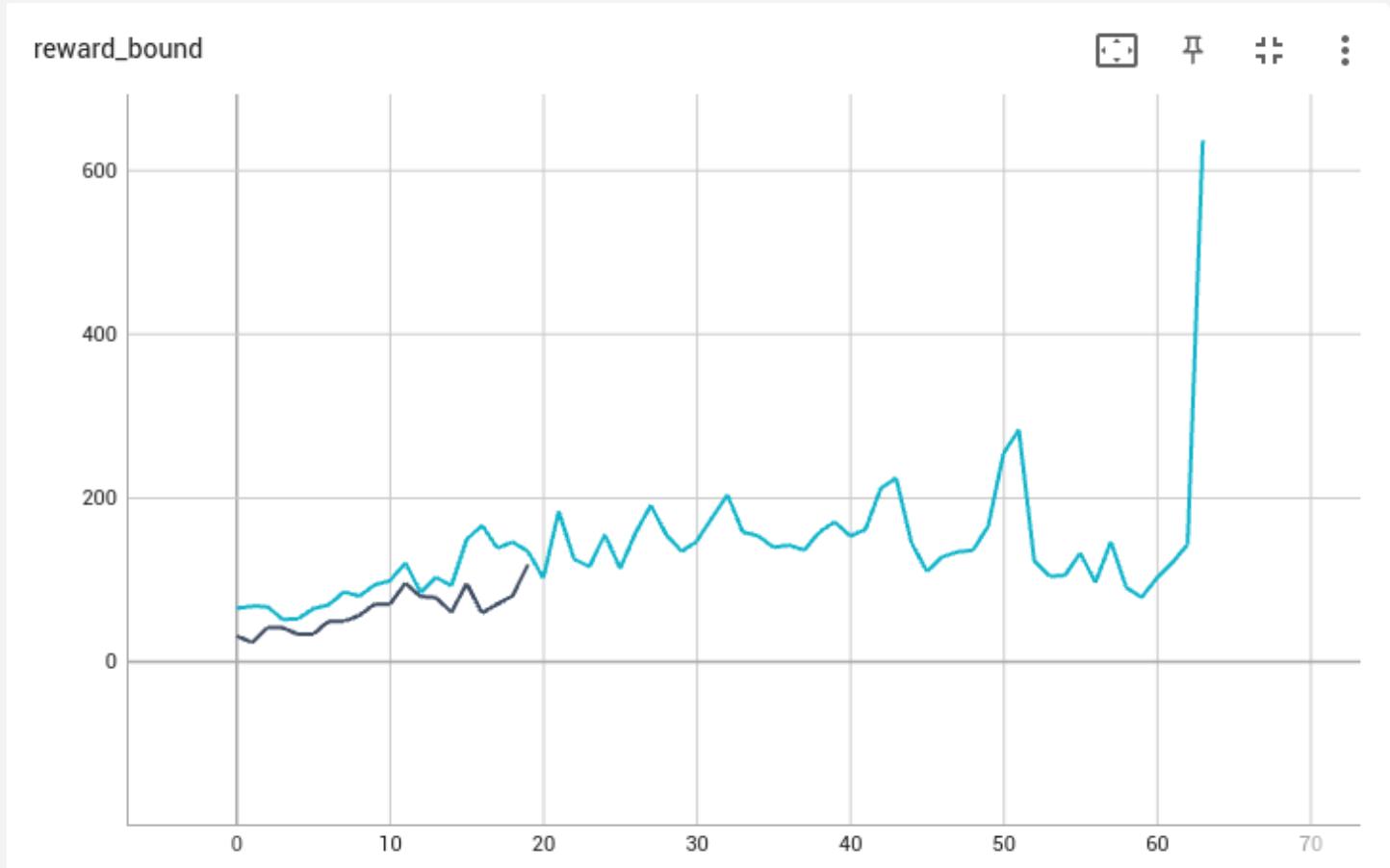
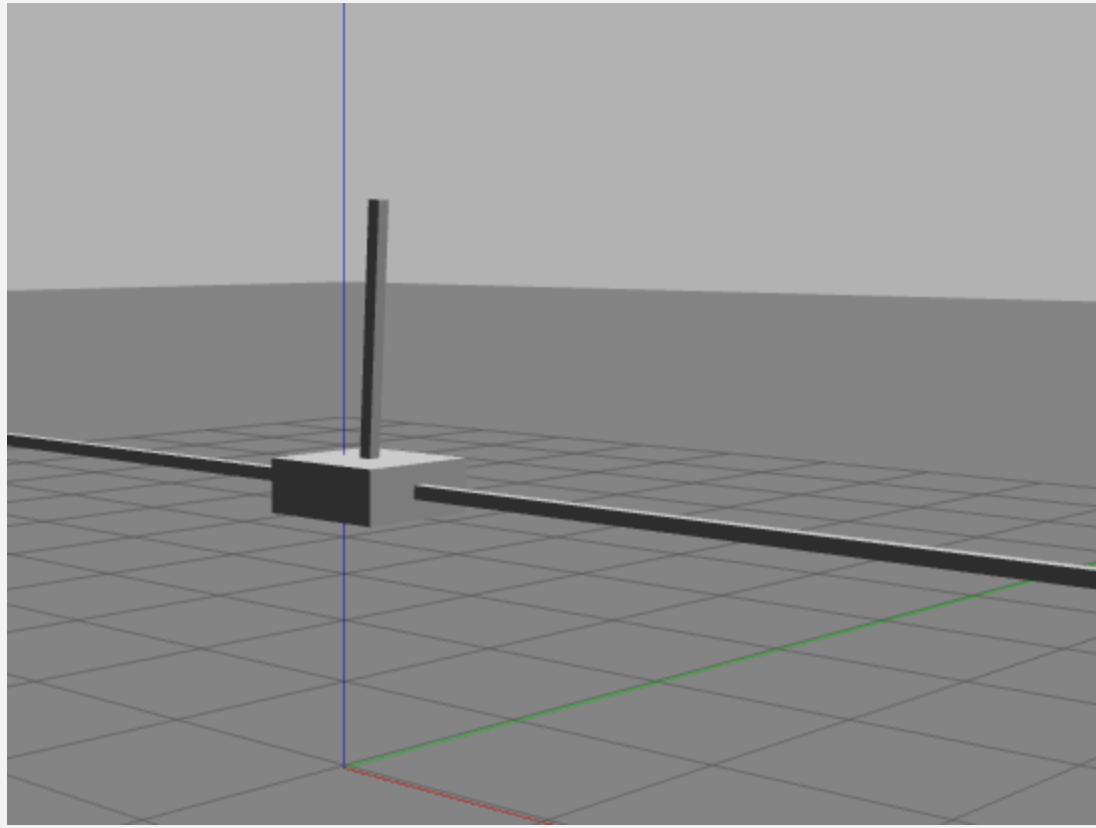
The optimizer can adjust hyper parameters based on the NN gradients computed. We need to zero the gradients at each step because PyTorch accumulates the gradients by default and we want to optimize always based only on the current iteration

The percentile and parameters are easily accessed at the top of the .py file. So we use this version of PERCENTILE for example.

To zero the accumulated gradients we actually need to zero out the optimizer.

Training Results

The cartpole appears to be training well for the simple 2 state case. I thought it would be a good exercise to also try and keep it closer to the center, so I added those states back in and altered the reward to have a preference for the center with a linear reward addition for the proximity to the center. A fair bit of adjustment would need to be made to find the right parameters that balance the desired behavior



Competition Environment

<https://docs.google.com/document/d/1TPA9V2gT7fbuTZKShv3O9NNrVftjPwqFouQBc0W17P4/edit#>

Setting Up The Environment Feb 25, 2023

Our team decided to get the joystick control running for the competition environment. I have also started to adjust the source file and install the local files on my system.

Key Commands

sourceComp	runs the source file for competition terminal environment
edit_source	once source is run, this opens up a nano editor to adjust commands
teleop	Keyboard control in the sim
camfeed	Camera feed window from the sim
gatherplates	roslaunch enph353_plate_detection gather_plate_images.launch

```
alias scriptsfolder='cd ~/ros_ws/src/2022_competition/enph353/enph353_utils/scripts'
```

- 1) Start the simulated world:

```
cd ~/ros_ws/src/2022_competition/enph353/enph353_utils/scripts  
./run_sim.sh -vpg
```

The available flags for the simulation are:

Option	Description
-v	spawn vehicle
-p	spawn pedestrians
-g	generate new license plates

- 1) Start the score tracking app Open a new tab in the current terminal window by pressing Ctrl+Shift+T

The new tab will open in: ~/ros_ws/src/2022_competition/enph353/enph353_utils/scripts
Launch the score tracking app:

```
./score_tracker.py
```

Questions – How to not startup with teleop?

<http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick>

Starting a joy nodeL

Ros Joy

Ros Joy is able to read commands from the XBOX controller. For testing purposes we can do the following:

Find the devices available:

```
$ ls /dev/input/
```

The joystick will be a device js0, js1 etc

We need to give permissions to the device to write, see:

<http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick>

Start Roscore and then run the joy node

```
$ roscore
$ rosparam set joy_node/dev "/dev/input/jsX"
$ rosrun joy joy_node
```

In a new window we can verify that the commands are registering for it:

```
$ rostopic echo joy
```

The commands are not in the right format to make a twist message and for this reason we need to add a new teleop_twist_joy package. The package has configuration files that can be accessed from the main ros directory, the path is

```
cd /opt/ros/noetic/share/teleop_twist_joy/
```

From there the launch and config files should be edited for the default controller being used. I also added the joy config and input to the launch file for my system:

```
<launch>
```

```
<node pkg="teleop_twist_joy" name="teleop_twist_joy" type="teleop_node"
output="screen">

<rosparam command="load" file="$(arg config_filepath)" />

</node>

</launch>
```

So it is important to note that any future changes to the system are going to need to be handled from this directory.

IMPORTANT:

Configuration has been set so that teleop will launch ros joy so it is a single action to launch both.. this is done from within the project environment in the manual_control launch file. It is all handled under the R1 name space. This makes it so that the default cmd_vel publishing topic for teleop_twist_joy is now published to R1/cmd_vel as desired, since R1 is the robot.

Actually it turns out the entire thing has been pre-setup in the directory.

- > run_sim.sh creates many launch sequences as a script. One of which calls robot.launch.
- > robot.launch contains the launch description for two robots, one of which is R1, the information in there is name spaced “R1” from that point onwards
- > From that name space, desktop.launch is initialized as an included file.

Inside of that file is where the joy control launch is made.

We specify a joy config in the config folder as the teleop_twist_joy.yaml file
This contains the button mapping and the enable button for the project, it can be edited to make adjustments. I changed it to separate the velocity and turn joysticks for more precise control.

Trying to set up a camera picture button for driving around Feb 26, 2023

I am getting strange errors– trying to open some type of yaml. I need to ask Miti for help in setting up the folder and .py script and to find out why it will not open

I am working with the emph353_plate_detection package

Now I am debugging using `rospy.loginfo()` and the python script that was installed.

Can also print messages to console using `print` if the flag

```
<node pkg="enph353_plate_detection" type="gather_plate_images.py"  
name="gather_plate_images" output="screen" />
```

`output="screen"` is used in the launch file.

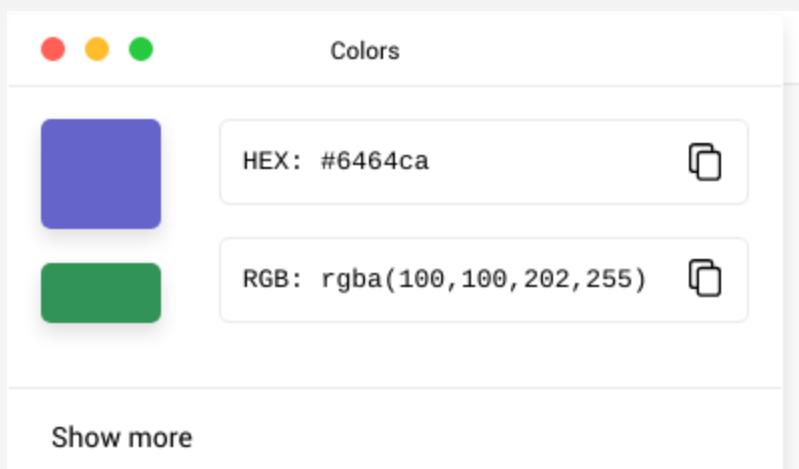
It is registering button inputs and sending the messages to the console, but no picture is saving.

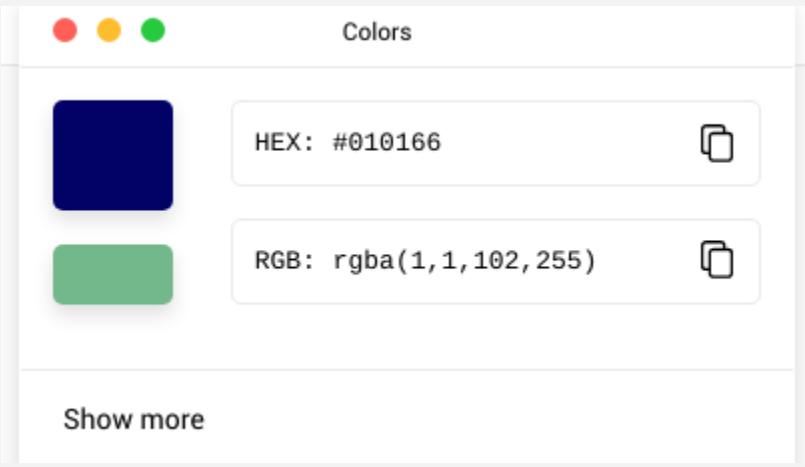
A linux file search found it saving to the wrong folder. Issue has been corrected.

Trying to set up a machine vision feed Feb 27, 2023

Now I am trying to add a feature to the node that allows for live image processing to test out some different theories on how to best preprocess data for license plates with live control.

Ask Miti about issues with package importing / catkin build in the gather plate images module





Looking at these shades it seems like a threshold could be made based off of the blue being at least 80 higher than other colors

Another idea is to filter for the narrow window around these color values.

Questions for Miti

I want to position some cameras with topics in all of the locations around the map.

- Ask for location info on the original camera used on robot
 - Build cameras in the positions as needed
- Check how to interface with the plate generator data such as the csv, have python node start up and wait 5 seconds before snapping photo processing on all 8 positions on launch

Ideas for Automated Plate Data Gathering Mar 5, 2023

I think it is possible to run a python script that will handle shutting down and restarting the sim environment.

- Have the script generate slightly shifted positions for the cameras. Make 8 cameras spawn in locations near the plates.
- Make sure that the plate generator boots a new csv file each time – another option is to alter it myself.
- Wait for 5 seconds to stabilize environment. Snap a picture from at each position and record the position and plate letters into the saved file name all in one folder with a time tag to prevent duplicates.

Shutdown Gazebo and reboot it again with some amount of delay allowed for

Easiest way: Spawn 8 robots with 8 name spaces at the 8 positions. This ensures the same camera ect



Left is positive Y-axis, Up is positive X-axis

View the model pose using

```
rostopic echo /R1/odom -n 1
```

```
rostopic echo /R1/odom | grep -A 11 'pose:' | head -n 11
```

pose:

position:

x: 0.998491705849964

y: 1.101117016096782

z: 0.0

orientation:

x: -0.03698471617180094

y: 0.0009158864489996162

z: 0.999143909921056

w: -0.01851321661324041

```
<arg name="init_pose" value="-x 1 -y 1.1 -z 0.5 -R 0.0 -P 0.0 -Y 3.14" />
```

Todo, fill this list with positions

Distance	Coords	P1	P2	P3	P4	P5	P6	P7	P8
Close	X1	-1.17							
	Y1	-1.09							
Med	X2	-1.13							
	Y2	-1.13							
Far	X3	-1.01							
	Y3	-1.20							
	Angle1								
	2								
	3	0 to -.3							

You can use the ``rostopic echo`` command to view the data on a ROS topic from the command line. Here is an example of how to use ``rostopic echo`` to view the pose information on the `/R1/odom` topic:

```
bash
```

 Copy code

```
rostopic echo /R1/odom
```

This command will print the `Odometry` messages as they are received on the `/R1/odom` topic to the console. The messages will include the pose information of the robot, including its position (``pose.pose.position``) and orientation (``pose.pose.orientation``) in the world frame.

You can also use the `-n` option with `rostopic echo` to limit the number of messages that are printed to the console. For example, the following command will print the latest 10 messages received on the `/R1/odom` topic:

```
bash
```

 Copy code

```
rostopic echo /R1/odom -n 10
```

Note that `rostopic echo` will continue to print messages to the console until you stop it by pressing `Ctrl+C`. If you only want to view a single message and then exit, you can use the `rostopic echo -n 1` command.

Implementing Data Scraping Mar 6, 2023

Before investing too much time into the ideas that I have for gathering a large data set, I set out to answer questions about what is feasible for camera capture automated with a bash script.

The first step is to set up a new launch file for robots– robots_many

```
<group ns="P2">  
  <param name="tf_prefix" value="P2_tf" />
```

```

<include file="$(find adeept_awr_gazebo)/launch/adeept_awr_urdf.launch">

    <arg name="init_pose" value="-x -1.01 -y -1.20 -z 0.5 -R 0.0 -P 0.0 -Y
-.15" />

    <arg name="robot_name" value="P2" />

</include>

<include file="$(find enph353_utils)/launch/desktop.launch"/>

</group>

<!-- bootup the camera scraping node --&gt;

&lt;arg name="nrobots" default="2" /&gt;

&lt;include file="$(find plate_data_generation)/launch/plate_data.launch"&gt;

    &lt;arg name="nrobots" value="$(arg nrobots)" /&gt;

&lt;/include&gt;
</pre>

```

Here the idea is that we can specify a parameter for the number of robots, and we use the convenience of name space to assign each one a parking number. I tested some of the coordinates scraped from odometer to position a robot spawn in front of the plate. The rqt_viewer confirms that this method works.

I then tried to spawn many robots and can see that it does not create an issue. The reality is that it is probably possible to easily spawn 100. Since they are all duplicates, running the joystick module will send messages to all of them at the same time with some hilarious results, an army of synchronous robots.

Now that is established that cameras can be positioned in this way– the next step is prototyping the ability to snap photos from them after launch. Each topic is indexed with a number for P1, P2, etc. This task is large enough that a new package makes sense to organize all of the data scraping activities. A new package **/home/sghyseli/ros_ws/src/plate_data_generation** is made using catkin, with a python file and permissions set. When launching many robots, the launch file should automatically call this so that it is automated on reset.

```

<!-- bootup the camera scraping node -->

<arg name="nrobots" default="2" />

<include file="$(find plate_data_generation)/launch/plate_data.launch">

<arg name="nrobots" value="$(arg nrobots)" />

</include>

```

For testing it was best to check the package was installed correct by running it with no arguments and a roslog output to flag it was called. After reaching this checkpoint, I added an robots flag to be able to alter that parameter from the source where it is being edited, the number in the launch file. The parameter is able to flag to the python script how many topics and photos it needs to do.

The next step was to use a basic nrobots=1 parameter to test that a single image can be captured. An initial delay allows the robots to settle before snapping a photo. Using the callback function was initially not successful for limiting to just one photo. Instead

```
msg = rospy.wait_for_message(topic, Image)
```

will take only one image msg and apply commands to it.

By putting this in a for loop, it is possible to cycle through all of the topic numberings and to save a png that is based on the topic so that the position is saved.

One missing feature is to be able to search the .csv that contains the plate numbers to add that to the file name. The filenames also have a date and time stamp to help with any debugging and to ensure unique identifiers for raw data images.

One last piece missing is to have a bash script that will open and close gazebo and take pictures overnight on automation. I wasn't very familiar with bash commands so I used some help from ChatGPT to make the script. One challenge is that the script calls another one, so care has to be made to stop the execution. Eventually the script is working based on a test with two robots and 3 loops. This is a proof of concept that it is possible to take 8 snapshots in less than a minute and to run that overnight. In ten hours this amounts to 4800 raw data images.

TODO:

Get the .csv tags for the plates into the file names

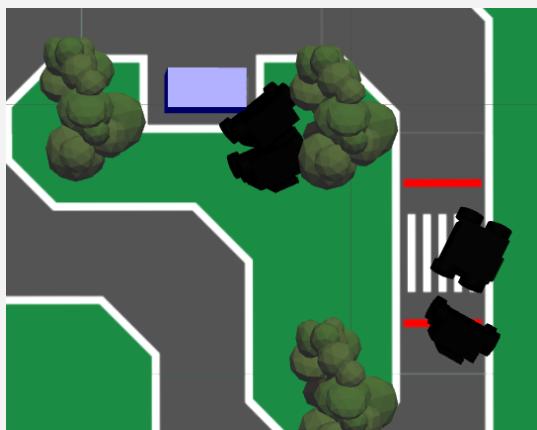
Add the full 8 robots and positions to the launch file

Specify an image size to use for the processed plates before the CNN

Right now I already have a suitable input to the CNN, it would be easier to work with smaller image sizes so it makes sense to save raw images as a backup but to also run the plate detection on each robot camera image from inside the data scraper. Then it can save a raw image and also save a perspective transformed image in a standard file dimension. That can then be further preprocessed into data for training the CNN. With this setup it is feasible to have 10s of thousands of training images to work with so I expect to have very good results.

Mar 9, 2023

Some work tonight on changing the color of the robot. For data gathering it is not normal for the robot to have another bot in the FOV, for this reason I changed the color to black so that it does not interfere with the plate detection algorithm



/home/sghyseli/ros_ws/src/2022_competition/adeept_awr/adeept_awr_description/urdf/adeept_awr.urdf
f

has some minor materials edits to account for this.

Plates csv file scraping

The plates file is csv and stores the plates generated for labeling of the images to be captured

/home/sghyseli/ros_ws/src/2022_competition/enph353/enph353_gazebo/scripts/plates.csv

So we need to fish these out in our data engine.

The data has been connected to labels but now I notice that the plates are not changing on boot up

I even set the flag in the shell script for opening gazebo to explicitly always generate plates be true, but it seems to simply not call on startup. The work around is to explicitly run it in the automated bash script for data gathering, failsafe method.

Now all the black robots load up in the script– enough time needs to be allocated for ros to shutdown completely. It actually takes more time to shut down than to gather data. If a shutdown is premature it will mess up all future iterations of the script so it is best to be conservative with the timing on it.

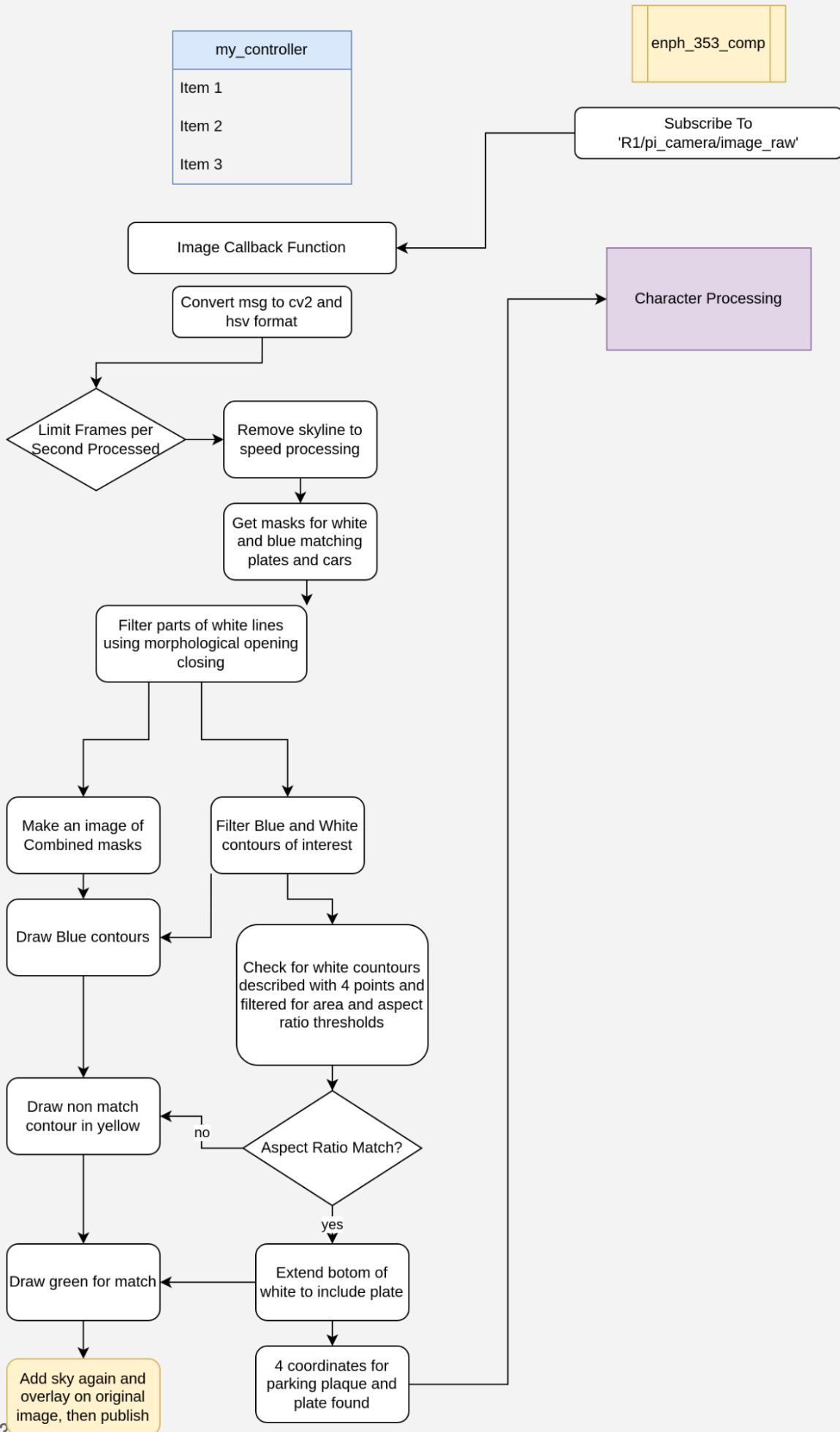
Notes on Comp Ideas

`ImageDataGenerator()` from `keras.preprocessing.image` to randomly distort the images based on parameters provided, resize all images to 20x20 pixels

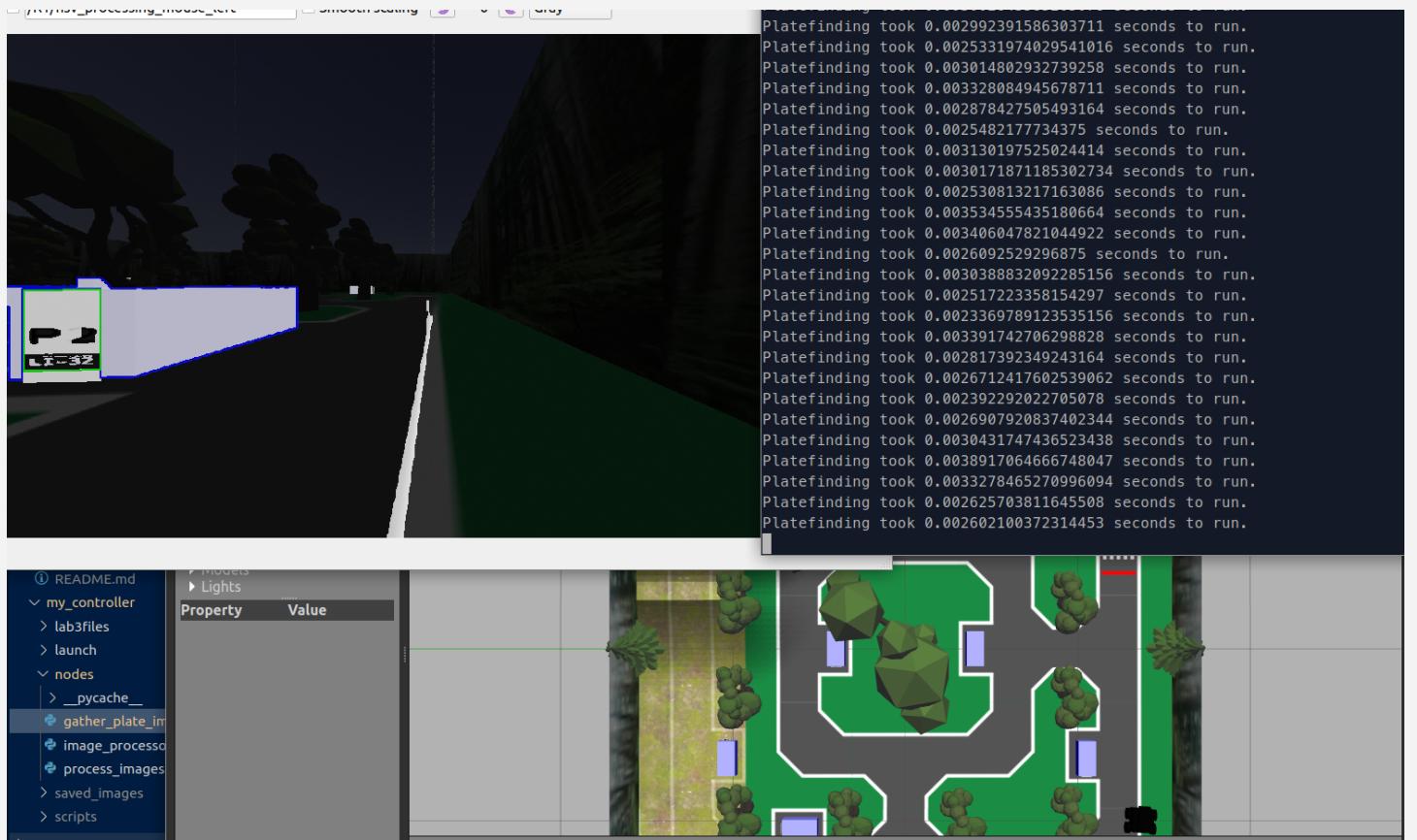
Refactoring Plate Filter and Mapping Data Flow Mar 9, 2023

Now that many tools have been prototyped it is time to clean up the code, do some refactoring, and diagram some flow charts.

The Plate Detection data flow is as follows:

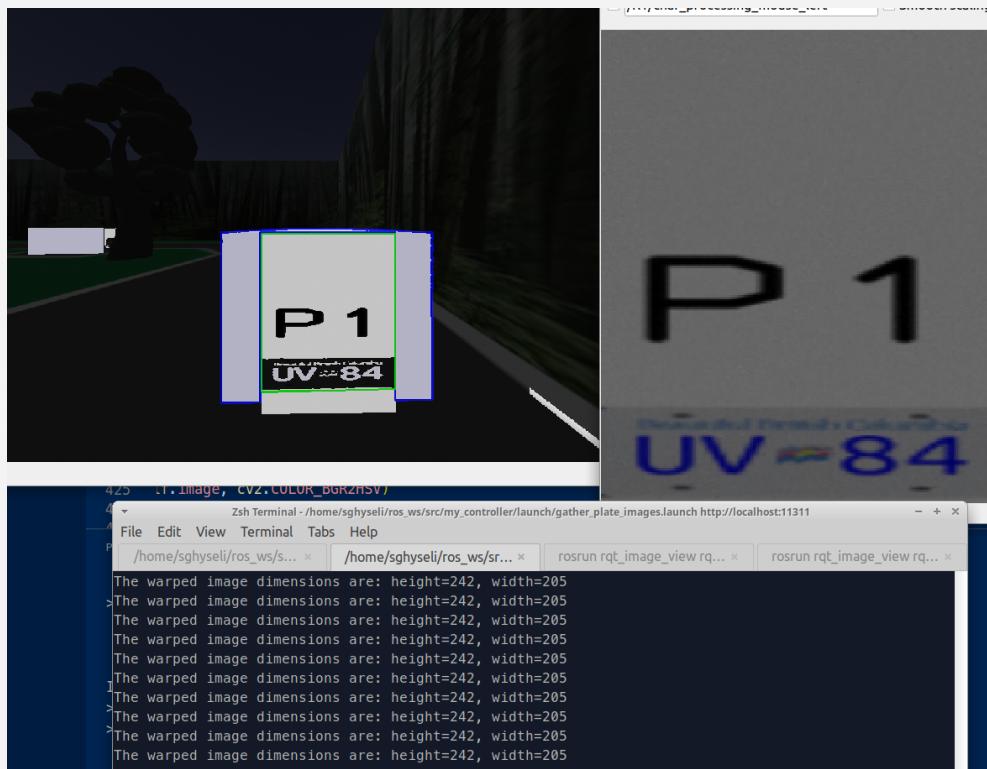


The refactoring is complete and the code is more modular and readable with some performance increases. The time to filter out and retrieve a plate position is 3ms which is relatively fast for image processing. Subsequent steps will be done at much lower resolution so this is a key area to optimize when the data input is quite large.



Some of these functions have ported modified to the data engine so that pictures of warped plates can be save directly which will greatly ease the image processing. smaller and more manageable sample images can be processed this way if the preprocessing happens overnight during capture.

To standardize the data photos, a height and width of 240x200 are chosen as this represents the undistorted view: from face-on



Chat GPT can do some of the heavylifting for such a specific and testable task

SCSS

Copy code

```
# Define a fixed output size for the warped image
OUTPUT_SIZE = (600, 300)

# Extract the corner points
topleft, topright, bottom_right, bottom_left = rect

# Determine the width and height of the rectangle to warp to
width = max(np.linalg.norm(topleft - topright), np.linalg.norm(bottom_left - bottom_right))
height = max(np.linalg.norm(topleft - bottom_left), np.linalg.norm(topright - bottom_right))

# Calculate the scaling factors for the width and height
scale_x = OUTPUT_SIZE[0] / width
scale_y = OUTPUT_SIZE[1] / height

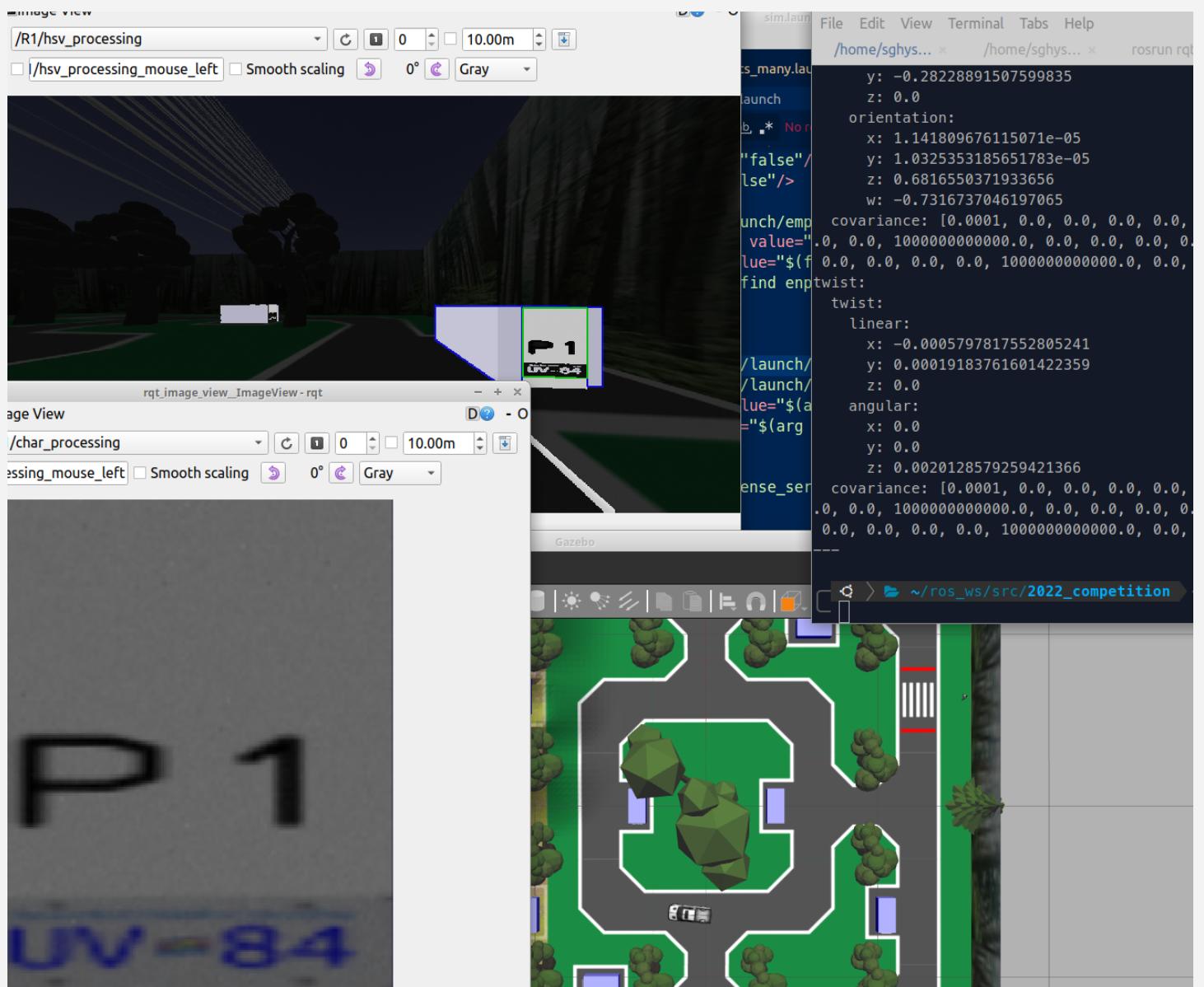
# Scale up the width and height by the scaling factors
width *= scale_x
height *= scale_y

# Define the target points for the perspective transformation
target = np.array([[0, 0], [width, 0], [width, height], [0, height]], dtype=np.float32)

# Get the perspective transform matrix and warp the image
M = cv2.getPerspectiveTransform(rect, target)
warped = cv2.warpPerspective(img, M, OUTPUT_SIZE)
```

Now the updates are added to the data engine, the process is tested with an adjustment back to RGB for the png and the results are good. For some reason the contour filtering was applying trouble for edge cutoff condition, but the data engine pictures will not be set up in such border cases so there is no issue removing the conditional check.

Now to finish the data set for positions

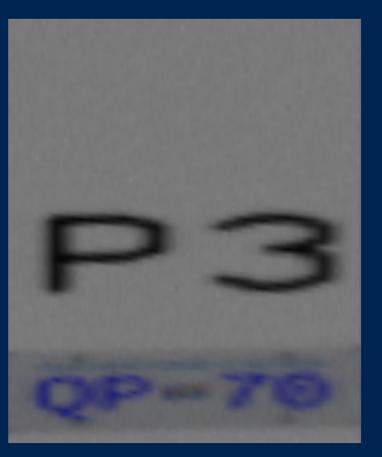


Distance	Coords	P1	P2	P3	P4	P5	P6	P7	P8
far	X1	-1.26	-1.19	1.14	1.09	-0.2116	-1.108	-0.397	0.547
	Y1	.28	-1.073	-1.20	1.208	1.216	1.152	0.6046	-0.539
Med	X2	-1.2674	-1.083	1.22	1.019	-0.255	-1.19	-0.3101	0.391
	Y2	-0.3013	-1.169	-1.00	1.266	1.227	1.028	0.6103	-0.6035
close	X3	-1.2659	-0.99	1.238	0.914	-0.342	-1.225	-0.2837	0.2987
	Y3	-0.3415	-1.20	-0.94	1.265	1.239	0.9002	0.5968	-0.6091
	Angle1	right 4.71	5.5 to 5.4	.96	2.5	3.14	4.2	.35	4.35
		to 4.5	5.9	1.4	2.7	3.14	4.4	0	3.5
				1.57	3.14	3.14	4.7	0	3.14



Up is 0, Left is $\pi/2$, Down is π , right is $3\pi/2$

- ☒ P2_UM50_20230309_212439.png
- ☒ P2_VP53_20230309_212515.png
- ☒ P2_VY81_20230309_212019.png
- ☒ P2_YF03_20230309_212134.png
- ☒ P2_YZ09_20230309_212210.png
- ☒ P3_BJ89_20230309_212247.png
- ☒ P3_DC86_20230309_212324.png
- ☒ P3_D009_20230309_212210.png
- ☒ P3_EX57_20230309_212056.png
- ☒ P3_OF30_20230309_212401.png
- ☒ P3_QA41_20230309_212134.png
- ☒ P3_QP70_20230309_212019.png
- ☒ P3_RC25_20230309_212553.png
- ☒ P3_WV33_20230309_212515.png
- ☒ P3_YO55_20230309_212439.png
- ☒ P4_DW41_20230309_212401.png
- ☒ P4_RA36_20230309_212134.png



The positions were time consuming to set up but they should be a lot easier to increment closer to the plates on each round. The settings are on relatively far images right now that have distortions and blur.

Mar 13, 2023 Collecting Parking Spot Numbers

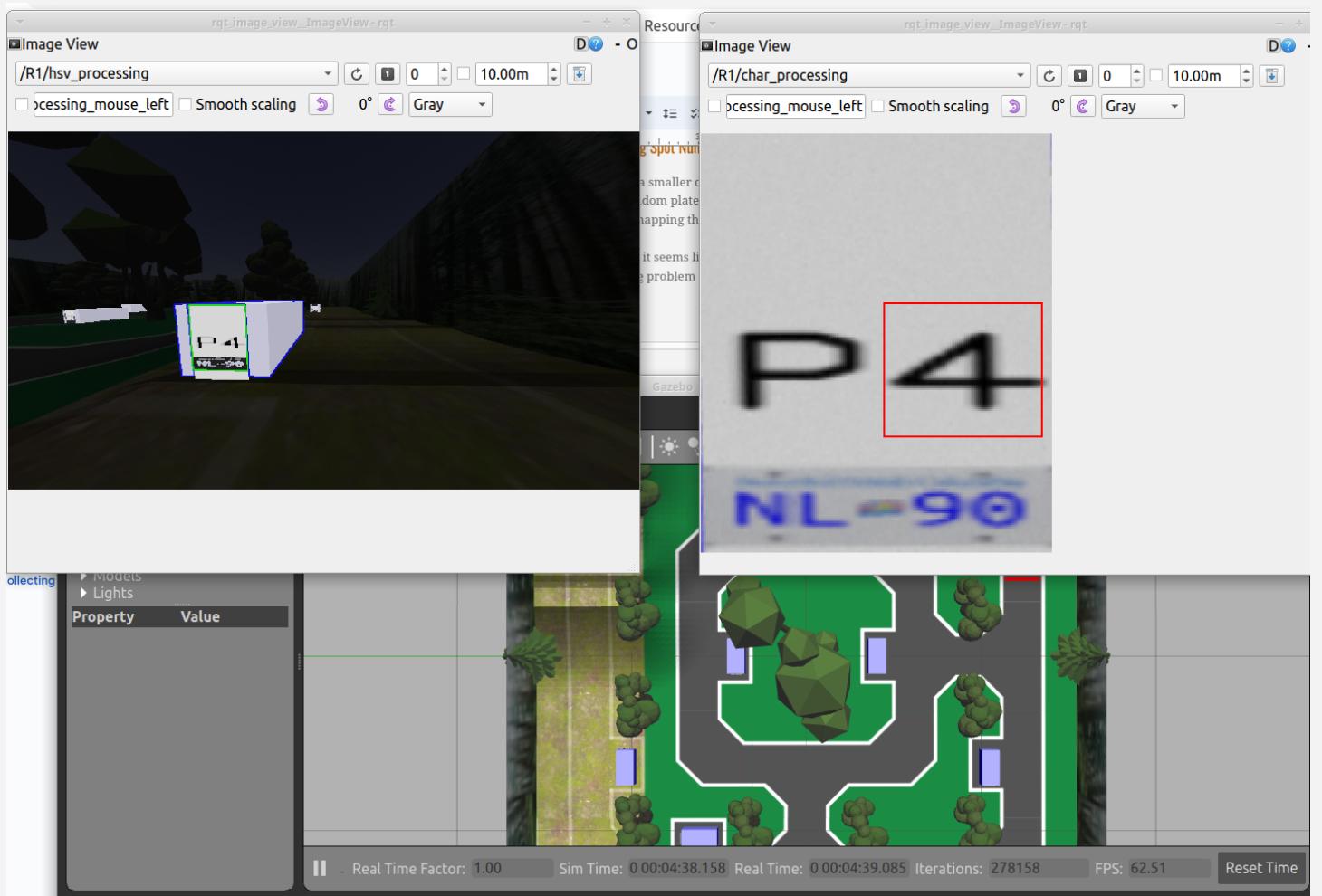
The parking spot numbers are a smaller data set and are generally easier to process due to the size and clarity. They dont rely on a random plate generator and so we can capture a wide sample of data by hand by driving with joy and snapping the homographic images manually.

At this stage of the competition it seems like a good problem to learn some of the tools with before moving on to the more advance problem of sorting the license plates. In addition, by having parking

number recognition working reliably, it can be combined with a self-driving robot to generate labeled plate images automatically. The plate label can be identified automatically from the image if the parking spot is recognized at the time of the image generation.

A button is added to joy for manual images.

A large selection of images was taken manually from all angles from each plate, now it is time to process images. By drawing a red box onto areas of interest it is possible to define the region that will have the parking ID #:



The box is tested from all angles on all placards and it matches the number positioning well:
Parameters are kept relative to allow image resizing

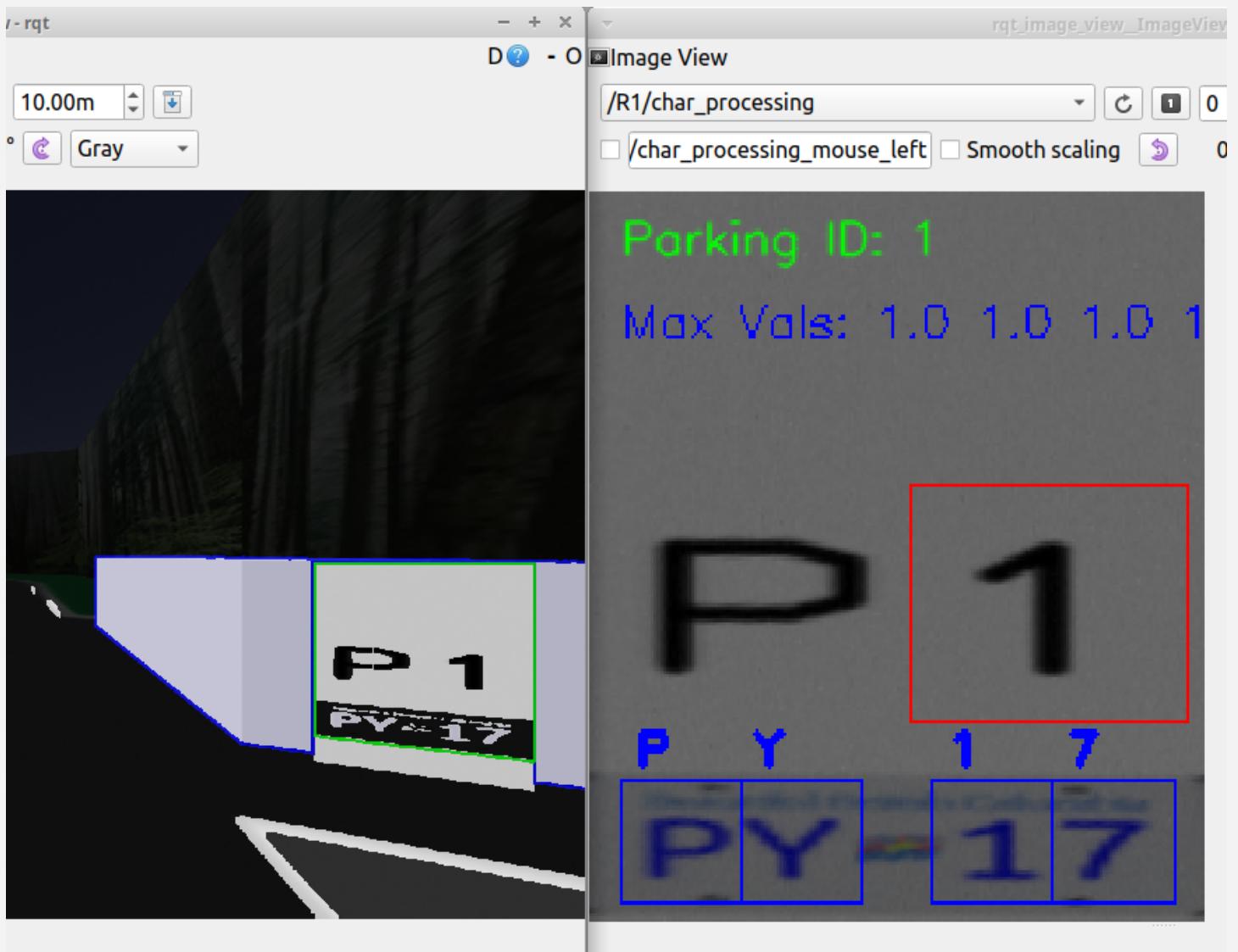
```
def drawSlicePoints(self):
    # Check the aspect ratio of the warped image
    warped_height, warped_width, _ = self.warped.shape

    # Draw a red box onto the image
    line_color = (0, 0, 255) # red color in BGR format
    line_thickness = 1 # thickness of the lines in pixels

    # Calculate the positions of the horizontal and vertical lines
    x1 = int(0.52 * warped_width)
    x2 = int(0.97 * warped_width)
    y1 = int(0.4 * warped_height)
    y2 = int(0.72 * warped_height)

    # Draw the horizontal lines
    cv2.line(self.warped, (x1, y1), (x2, y1), line_color, thickness=line_thickness)
    cv2.line(self.warped, (x1, y2), (x2, y2), line_color, thickness=line_thickness)

    # Draw the vertical lines
    cv2.line(self.warped, (x1, y1), (x1, y2), line_color, thickness=line_thickness)
    cv2.line(self.warped, (x2, y1), (x2, y2), line_color, thickness=line_thickness)
```



Building Data Set in Jupyter

Now that the digits are collected, we can process the data. The first step in notebook is to slice all the images down to digits. We maintain the original resolution on the saved files, it is best to keep the raw data for now and experiment with what is going to work. The exact same slicing points as the simulator are used.

Mar 14, 2023

The next steps with the data is to train using weights and biases to track the different models. With only 8 inputs and a desire for computational speed, I believe that the small set of training data can be used in combination with a smaller CNN to get good results, since the environment is somewhat static being digital. The code from Lab6 was scraped and adjusted to a local notebook for processing.

Some design decisions made in processing the data.

- We want speed so reducing the input size and complexity is a good idea. The images are reduced to a 20x25 size in greyscale to accomplish this goal.
- The best way to test the CNN is with an actual run in the simulation environment so there is no test data set in the Jupyter Notebook. Instead the CNN is imported into the controller.

Tensorflow, Keras, and Nvidia CUDA packages all had to be installed. This took a fair bit of debugging and is not a trivial environment to setup correctly. It was imperative to downgrade numpy to an earlier version to fix an error that was happening involving a datatype.

Mar 15, 2023

Some test runs were conducted to gain familiarity with WandDB. Chat GPT provided some good assistance with the syntax required to keep the data for the CNN training singular without repetition. This is accomplished through a configuration data type that holds all of the different parameters for training that can then be shared both with the training in keras and with wandDB.

The 2D convolution requires there to be a third layer in the data, even if it is only trivially 1. The max pooling size also has to be reduced from the Lab_6 values because of the reduced size of the input.

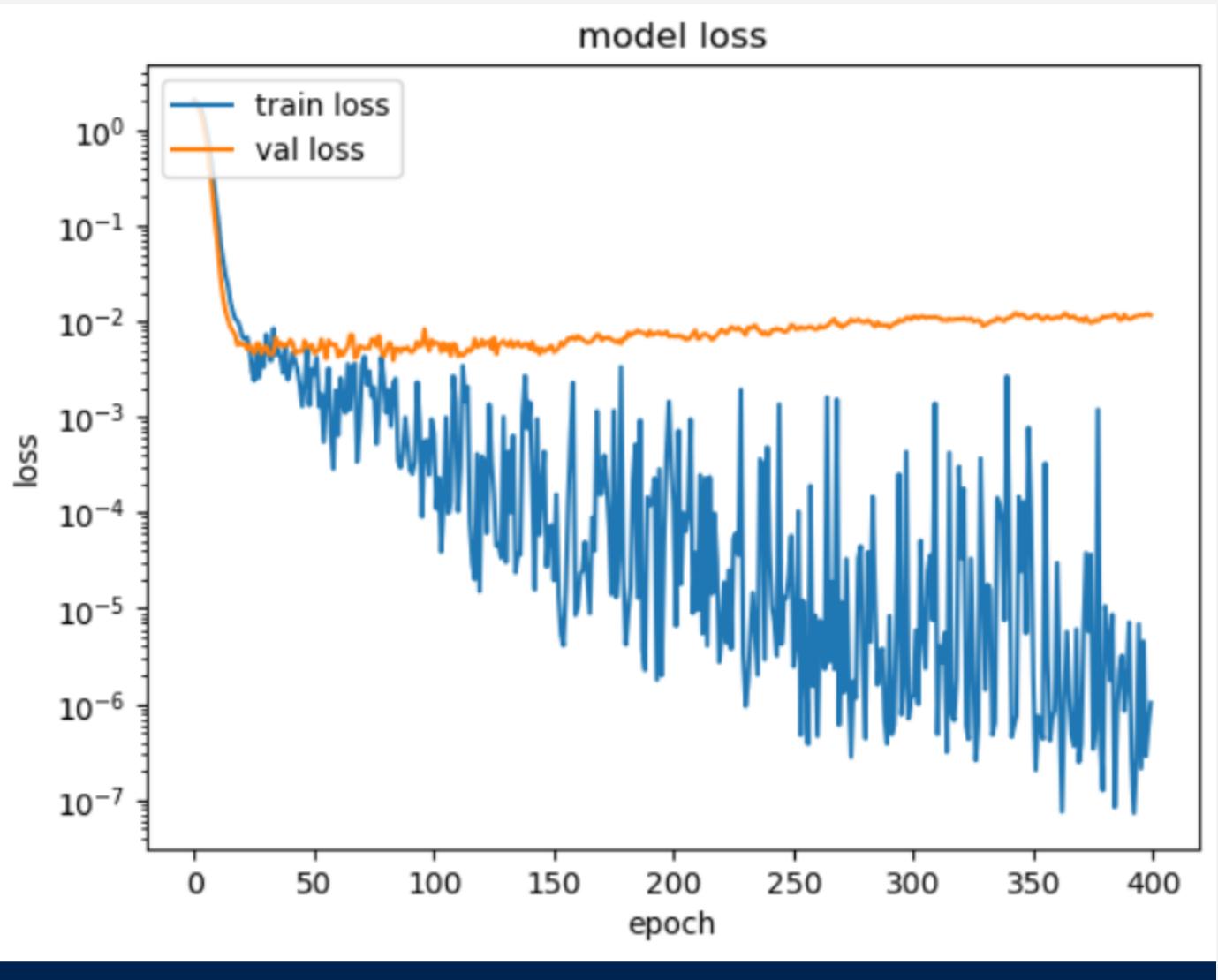
For the live testing code, the network is imported into the controller model. The same image preprocessing on the training data is copied over to the live version that will run in ROS. That includes remapping to greyscale, trimming down to 20x25, redefining as an np array, normalizing to 0->1 from the 0->255 standard of greyscale, then reshaping the array so that it has the 3rd layer on it. Pasting the code into chatGPT makes for a good check on the syntax and some code feedback. It made some improper suggestions that I argued against with clarifications, now the code is ready for testing.

A new function is also added from a modified parking ID overlay box. the slice points are drawn onto the homography image for debugging purposes but we also can draw the Y_prediction onto the top left of the extracted placard to get a live prediction for debugging. Hopefully all the components will work together in some of the test runs!

It works!

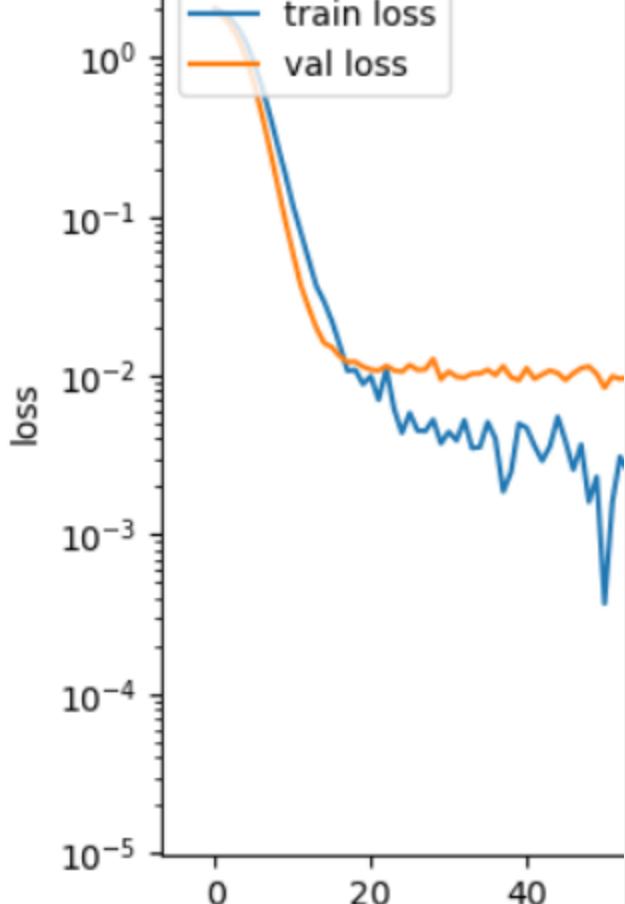
Initial testing shows a very good response to all plates. But the processing time is 12ms

I tried a simpler network for 400 epochs, it displays a clear signature of over-training:



It looks like about 35 epochs is enough for training. This time lets save intermediate weights.

Tinkering with a model more, lets try the 20th epoch of this one



Training and Cutting the Characters

The same process is used to find slice points, using a blue box overlay. I used Chat GPT to help with this very cumbersome task that has repeat lines

To draw two blue rectangles that represent the left 1/3 and right 1/3 of the current red rectangle, you can modify the x-coordinates of the vertical lines as follows:

python

 Copy code

```
# Check the aspect ratio of the warped image
warped_height, warped_width, _ = self.warped.shape

# Draw a blue box onto the image
line_color = (255, 0, 0) # blue color in BGR format
line_thickness = 1 # thickness of the lines in pixels

# Calculate the positions of the horizontal and vertical lines
x1 = int(0.05 * warped_width)
x2 = int(0.95 * warped_width)
y1 = int(0.8 * warped_height)
y2 = int(0.97 * warped_height)

# Calculate the positions of the new vertical lines for the left and right rectangles
x1_left = x1
x2_left = int(x1 + (x2 - x1) / 3)
x1_right = int(x2 - (x2 - x1) / 3)
x2_right = x2

# Draw the horizontal lines
cv2.line(self.warped, (x1_left, y1), (x2_right, y1), line_color, thickness=line_thickness)
cv2.line(self.warped, (x1_left, y2), (x2_right, y2), line_color, thickness=line_thickness)

# Draw the vertical lines for the left rectangle
cv2.line(self.warped, (x1_left, y1), (x1_left, y2), line_color, thickness=line_thickness)
cv2.line(self.warped, (x2_left, y1), (x2_left, y2), line_color, thickness=line_thickness)
```

A decent set of slice points is found, now it is time to try and slice some training images

I went through the other cleanup steps, next I want to compare various potential prefilters and see which ones will work best for preprocessing. I asked chatGPT to take the original data display function and then modify it to show the original data beside the stubs for 3 preprocessing functions.

and the preprocessed datasets side by side using a slider. For now, I kept the `preprocess_image1`, `preprocess_image2`, and `preprocess_image3` functions the same as the original `preprocess_image` function. You can update them later to apply different preprocessing steps.

```
python Copy code

import matplotlib.pyplot as plt

# Apply the preprocessing functions to the dataset
X_dataset_preprocessed1 = [preprocess_image1(img) for img in X_dataset_raw]
X_dataset_preprocessed2 = [preprocess_image2(img) for img in X_dataset_raw]
X_dataset_preprocessed3 = [preprocess_image3(img) for img in X_dataset_raw]

# Display images in the training data set.
def displayImage(index):
    fig, axes = plt.subplots(1, 4, figsize=(20, 5))

    axes[0].imshow(X_dataset_raw[index])
    axes[0].set_title("Original")

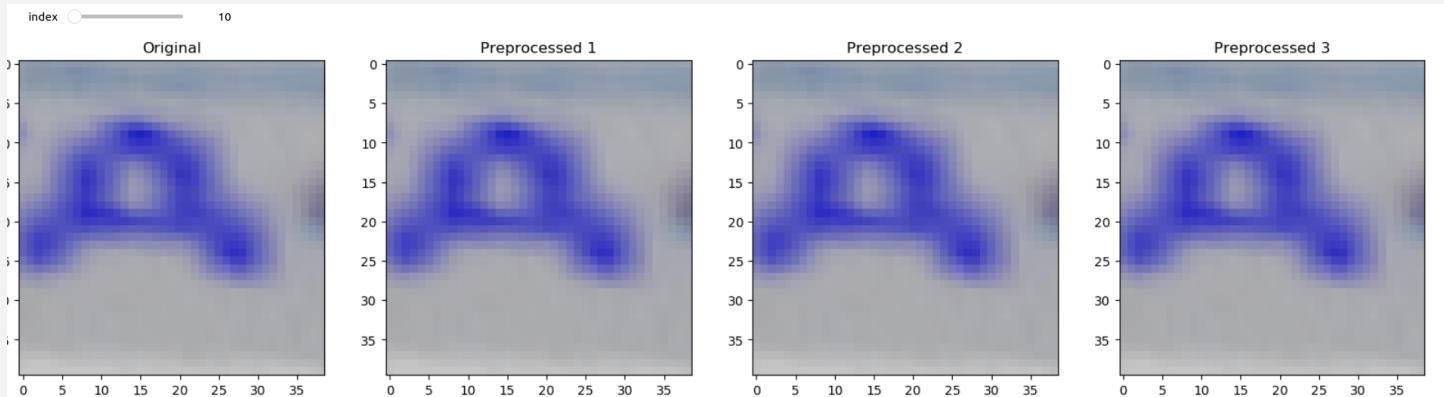
    axes[1].imshow(X_dataset_preprocessed1[index])
    axes[1].set_title("Preprocessed 1")

    axes[2].imshow(X_dataset_preprocessed2[index])
    axes[2].set_title("Preprocessed 2")

    axes[3].imshow(X_dataset_preprocessed3[index])
    axes[3].set_title("Preprocessed 3")

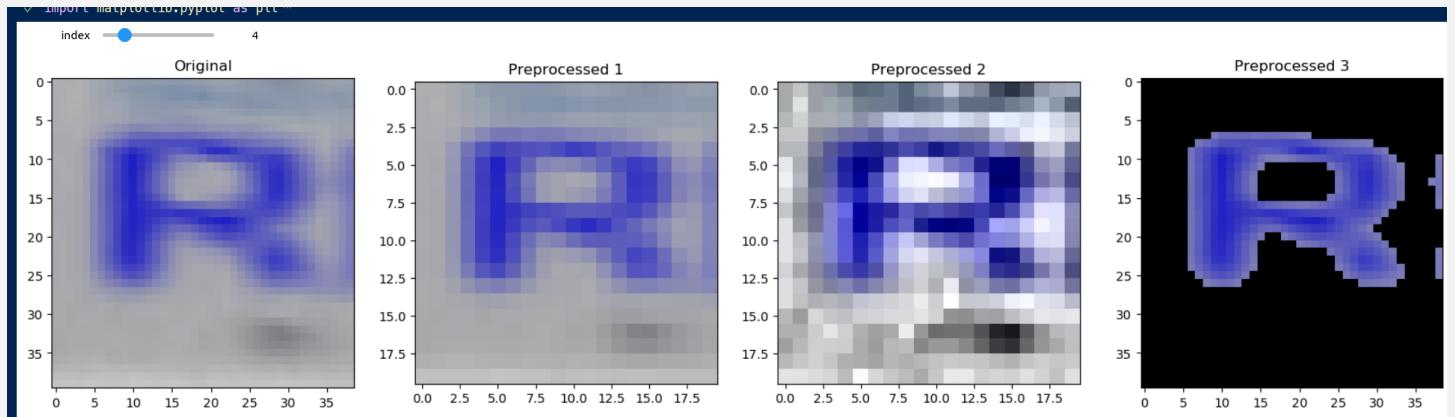
    plt.show()

interact(displayImage,
         index=ipywidgets.IntSlider(min=0, max=len(X_dataset_raw)-1,
                                    step=1, value=10))
```

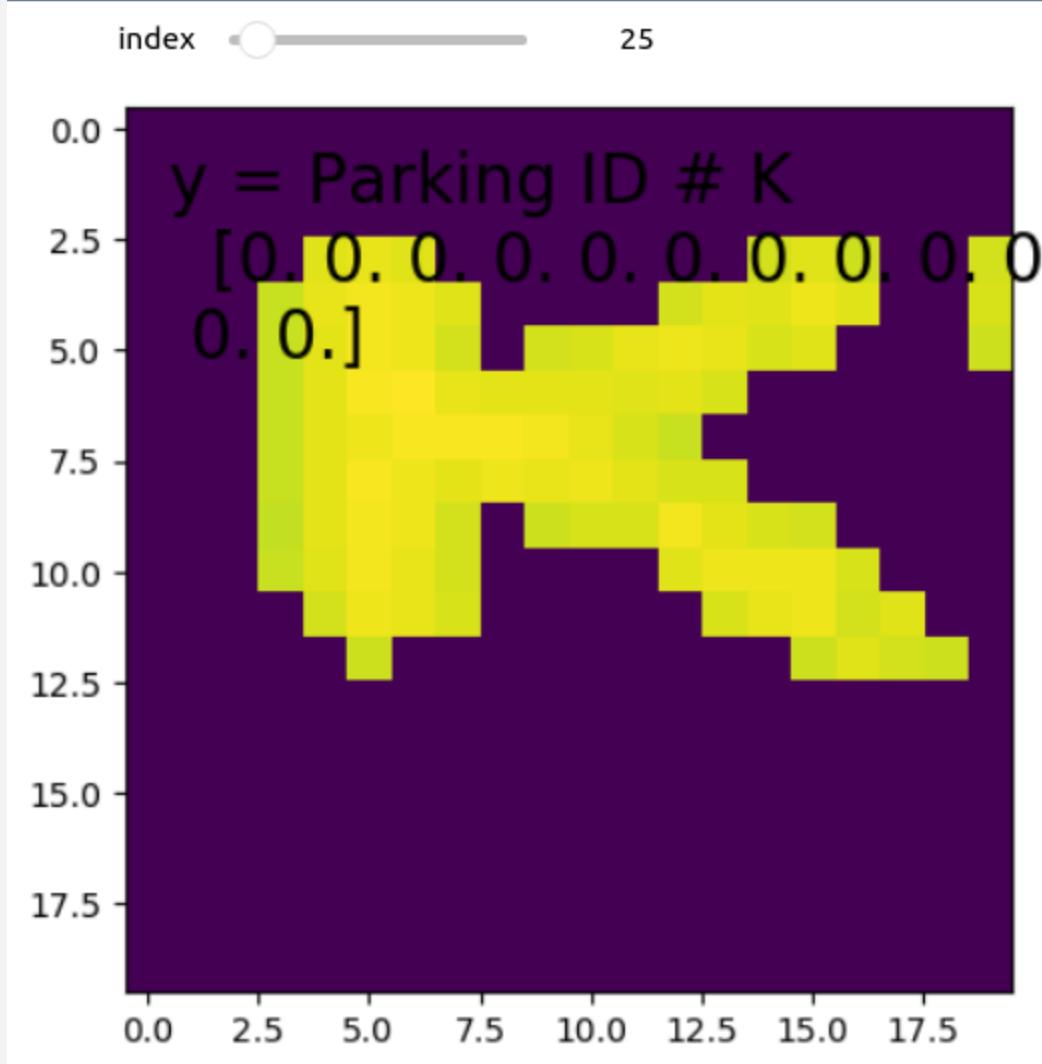


The obvious next step is to reduce to greyscale and put a light threshold on the images

I tried Gaussian, reduced resolution, CLAHE, but the good old blue hsv filtering from the placard recognition works amazing.



I picked a model that masks out the blue grayscale using hsv, it is 20x20 image size.



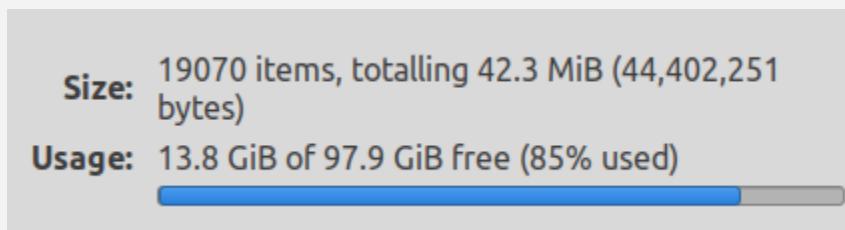
By staying in greyscale all the code from earlier can be slightly modified improving the workflow. For setting it all up, only a small batch of 250 letters are used, later The 10,000 plate images can be imported from the overnight sessions.

Mar 16, 2023

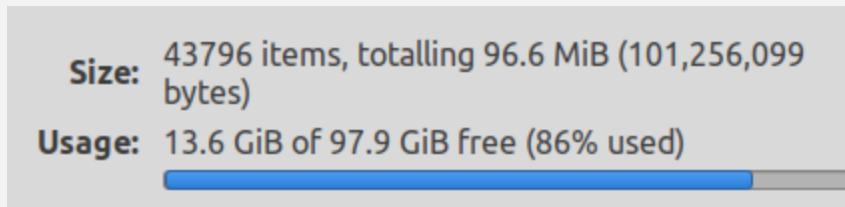
Last night produced about 6000 images but the raw images are large enough to cause concerns about overfilling the hard disk so I removed the saving of those since I am quite happy with homography snapshots that are only 200x240

The next idea that I have is to preload the plates with AA 11 style images for all 8. Then go into the system and use the automatic gathering from all angles on the plates. This will augment the data that was already collected.

It took about 2 hours of time but it was possible to rewrite the python plate generator to take in non-random values. The joystick and homography capture was used to get a pass from all angles on every plate with every letter and number in every position. In short- all of the data space has been covered, all letter combinations and also all letters from all angles. I am almost certain that this data set is failsafe and it contains ~20,000 characters.



With the digits intact, the cleaned data is:

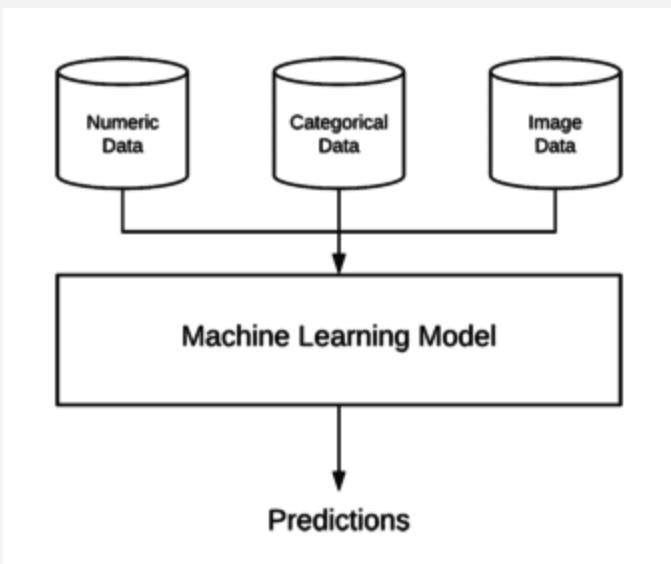


I am quite proud of this data set that is 'live' from the exact search and slice algorithm that is onboard the robot. This means we train on data that is identical to the test-set!

Rather than train two CNNs, I wish to try and inject the positional data to the Keras model. It seems like there may be a way to do this efficiently:

<https://pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/>

From the site:



This is actually a common scenario in ML applications

If you were to use Keras to visualize the model architecture it would look like the following:

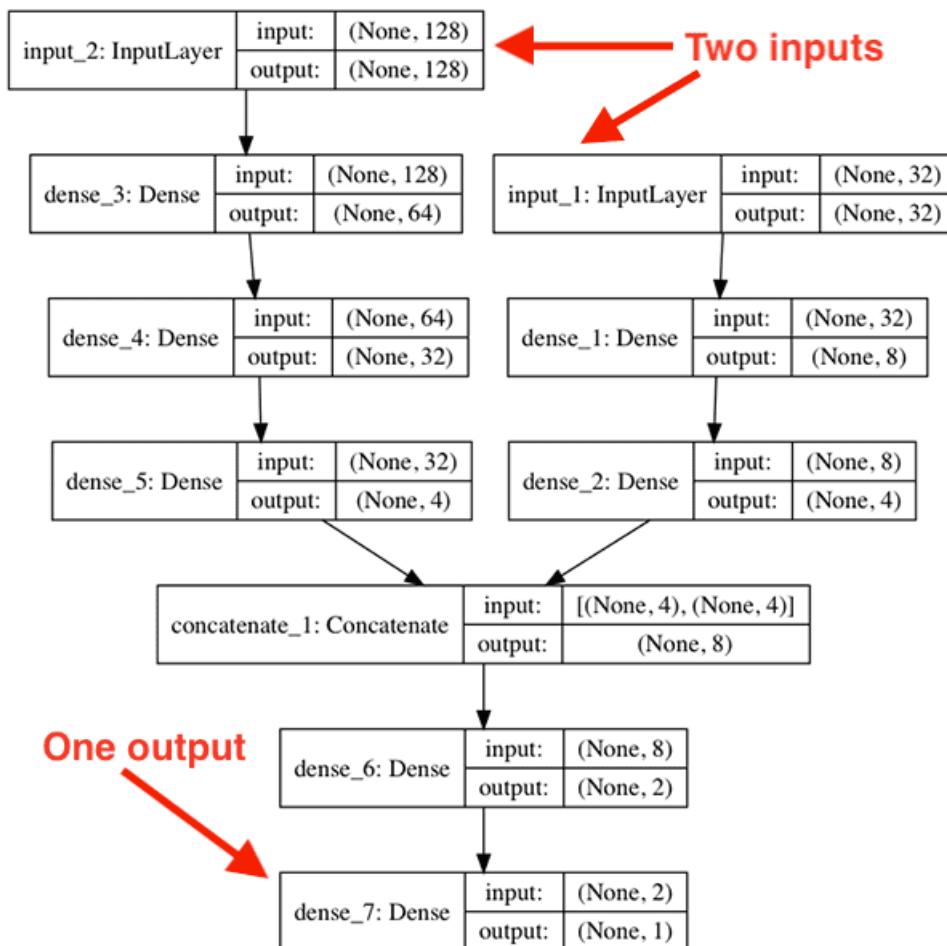


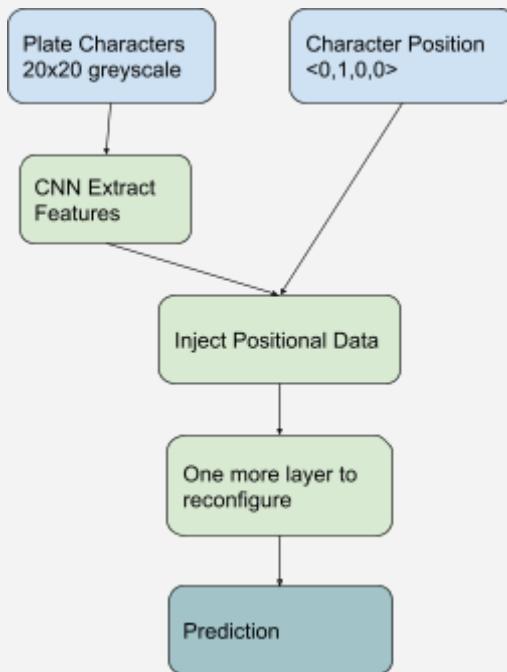
Figure 3: This model has two input branches that ultimately merge and produce one output. The Keras functional API allows for this type of architecture and others you can dream up.

A good example from the site shows the parallel paths.

In fact this situation is perfect for trying out a functional architecture because we are not going to be using parallel CNNs and simply injecting a one-hot positional vector at the end stages

```
# define two sets of inputs
inputA = Input(shape=(32,))
inputB = Input(shape=(128,))
# the first branch operates on the first input
x = Dense(8, activation="relu")(inputA)
x = Dense(4, activation="relu")(x)
x = Model(inputs=inputA, outputs=x)
# the second branch operates on the second input
y = Dense(64, activation="relu")(inputB)
y = Dense(32, activation="relu")(y)
y = Dense(4, activation="relu")(y)
y = Model(inputs=inputB, outputs=y)
# combine the output of the two branches
combined = concatenate([x.output, y.output])
# apply a FC layer and then a regression prediction on the
# combined outputs
z = Dense(2, activation="relu")(combined)
z = Dense(1, activation="linear")(z)
# our model will accept the inputs of the two branches and
# then output a single value
model = Model(inputs=[x.input, y.input], outputs=z)
```

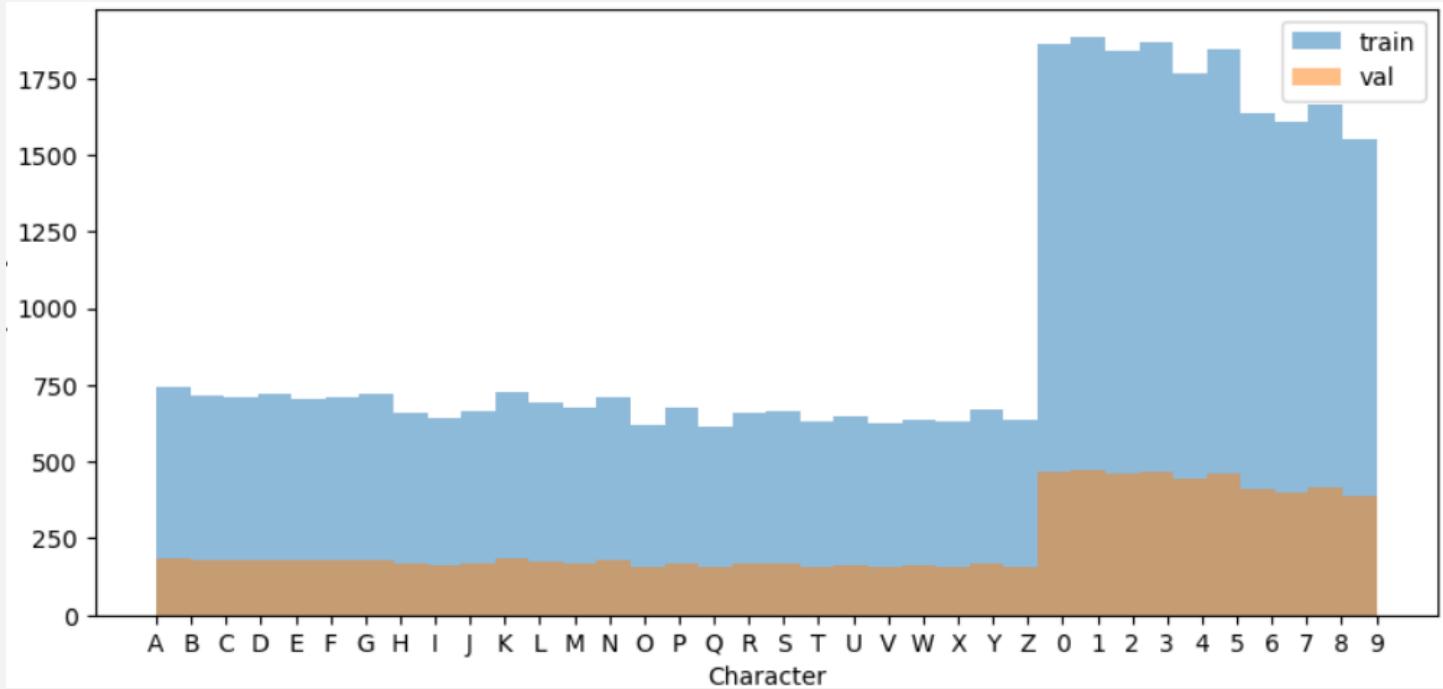
So the planned architecture to implement is:



There is no preprocessing required for the character position. I'd like to try one last layer before activating to allow for the high level features to be reconfigured according to position. We could also

try injecting at an earlier point even.

The training set is built with a position 1-hot vector and the characters mixed:

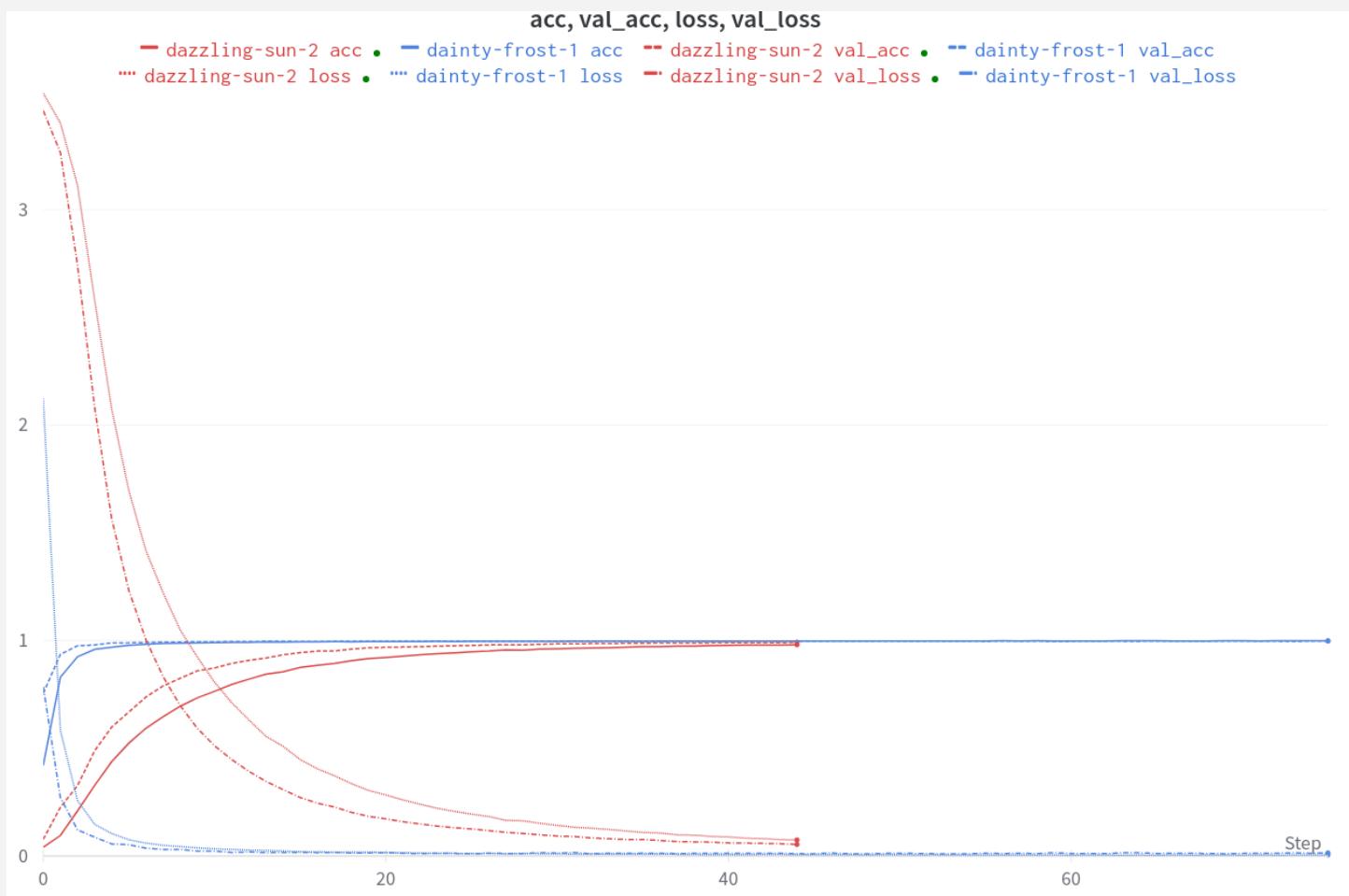


The digits are overweight seemingly, but the split between the two data types of letters and digits is equal, We probably dont need to truncate digits but it is something that can be revisited

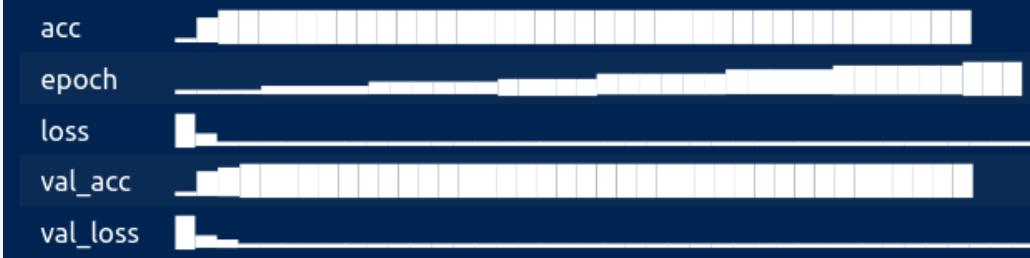
Training Results

I was blown away by the results of the first network. It is hitting .9977 accuracy and that is on a validation set that is not even human-readable for many instances. It can read extremely blurred and distorted characters. It only took 40 training Epochs with a batch size of 64, so I lowered the learning rate by a factor of 10 and will try to see if the accuracy can be improved at all. Here is a comparison of the two models training with the lowered learning rate in red showing much slower convergence.

The reality seems that I went completely nuclear on data acquisition and tool-set building, spending weeks on machine vision techniques and datascraping. The data cleaning and architecture takes a half day to research and set up, then the actual training is down to ~15 minutes with GPU acceleration



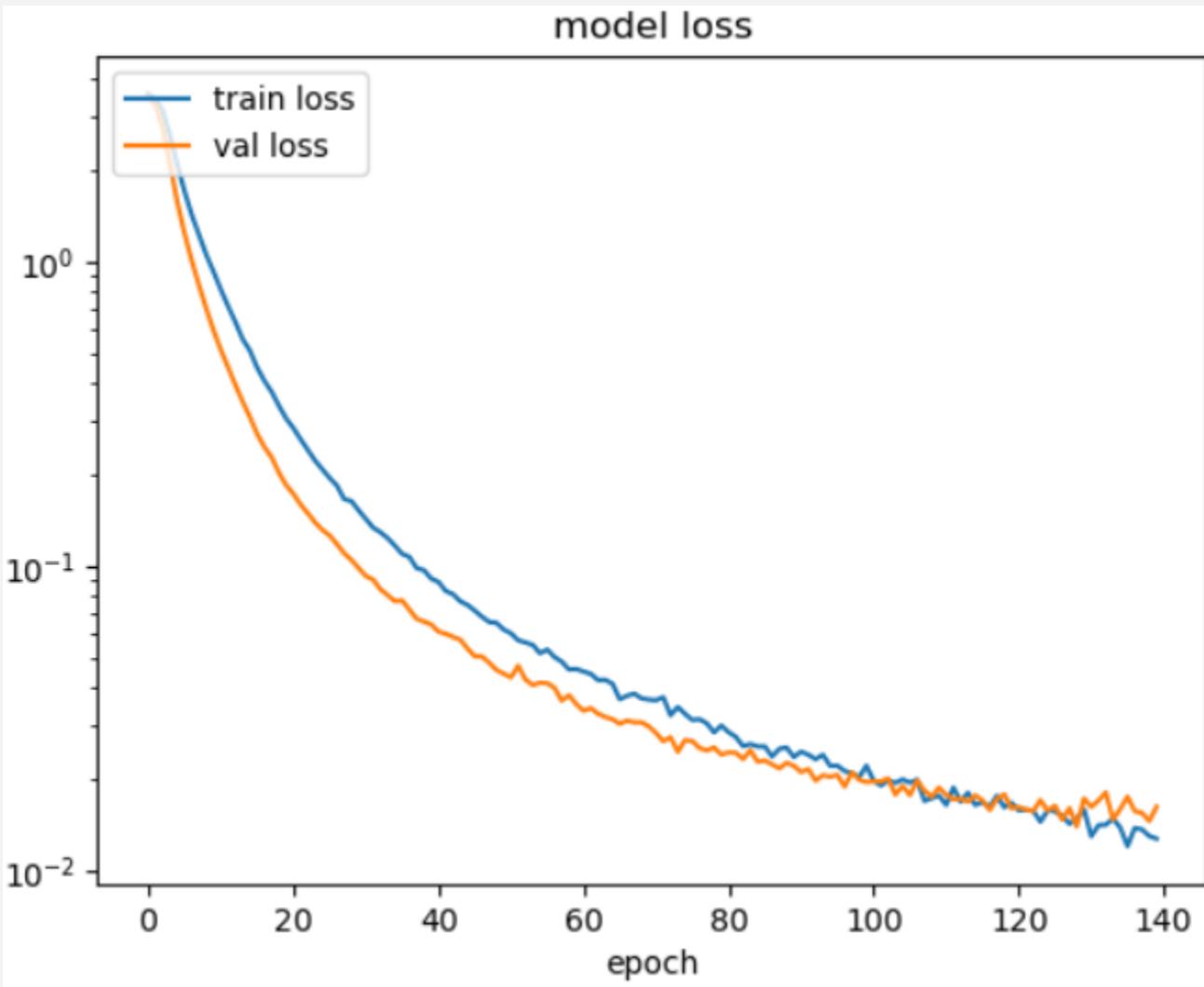
Run history:



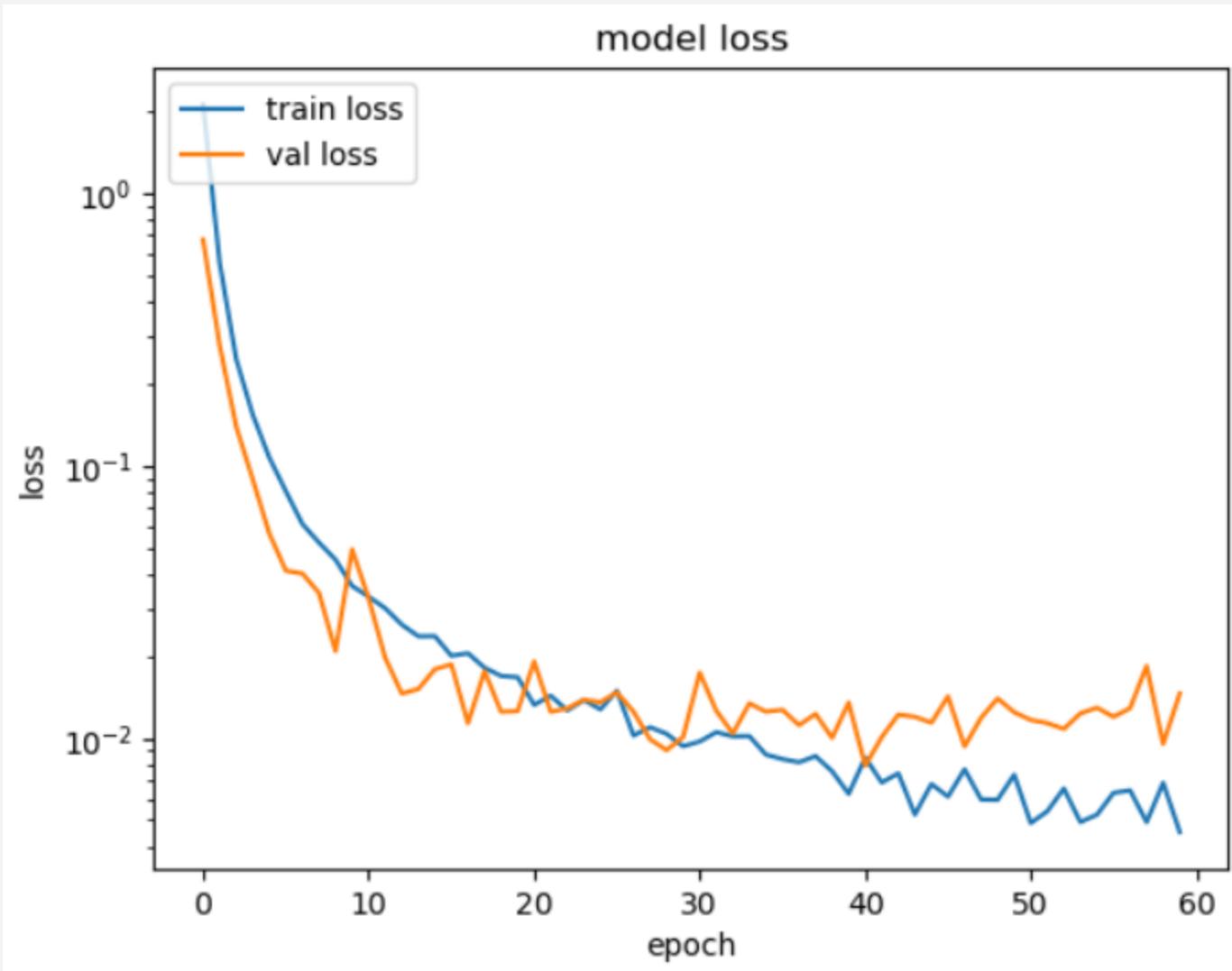
Run summary:

acc	0.99877
best_epoch	54
best_val_loss	0.00995
epoch	75
loss	0.00499
val_acc	0.99795
val_loss	0.01424

The second run with slower learning stopped at 140 epochs but it appears slightly undertrained so more training without resetting weights is in order.



The faster training rate was actually much more preferable. The trick is to save weights from every epoch, then find the best accuracy in validation and use that set of weights. It should only take about 40 epochs to reach 998/1000 accuracy and that is for a very messy training data set. Excited to try out the model live.



Epoch 40 is the winner, it is saved to epoch 41 with 1 indexing

Integrating into Live CNN

All of the image transforms from the CNN training are copy pasted over. In the future on such a project, it would be best to have a python file that contains the functions accessed by both to ensure concurrency. Any slight deviation from the trained preprocessing will cause problems.

Case in point, the sim data is BGR instead of RGB, this made it appear to be a failure until the mistake was modified since the sliced characters differ in color coding between the simulation environment and the PNGs that were loaded for training

The predictions can run in parallel which offers a huge advantage over using separate models. Feeding 4 characters seems to take the same time as feeding one, so it is optimal to use the mixed data CNN.

A plate overlay for debugging is added, everything is packaged up and very smooth

On a wanted list is to integrate parking ID into the CNN as well, but that is not needed at this time and more of a todo. But the same CNN could be used with an extra position added forming mixed data

Mar 17, 2023

Handling error values and determining best match

It is an optimization problem to prevent the CNN from reading false matches. One Idea that I have for converging a prediction is to take the most common read value at the end of the driving session. The results can be stored in a dictionary which is a familiar datatype for me. However I was not sure how to implement it in python when there are actually 3 data types involved:

- ParkingID
- Plate Reading
- Number of times read

The Parking ID is clearly the key. I asked GPT-4 for some help:

"I have a CNN reading plates, it mostly gets them right, but sometimes wrong. But it reads a lot of them over time.

I want to have a dictionry to store the results for 8 different locations. I have a very reliable ID locator that will put the result into the correct bin.

What I want is to take a string of 4 characters, for example "AA56" that has a matching parking ID, for example '4'

Then each time I read "AA56" at "4" add that to a counter that counts how many times that it has been read. There will be occasional erroneous values like "AAS6" , those are tracked too. At the end of the session I want to look at parking ID 4 and then take the value that was read the largest number of times, inducing that it is the correct one"

Something important when doing this is to understand the code so I look through it to make a breakdown. We can see that parking data is a dictionary of dictionaries. There is a constructor that is given for the outer dictionary, the constructor indicates that the keyed data type is another dictionary that stores integers for the keys. The effect is a 2D key system for location and plate with a stored integer value.

```
python                                         Copy code

from collections import defaultdict

# Initialize the dictionary
parking_data = defaultdict(lambda: defaultdict(int))

def add_plate_reading(plate, location):
    """
    Add a plate reading to the parking_data dictionary.

    Args:
        plate (str): The plate string, e.g. 'AA56'
        location (int): The parking location ID, e.g. 4
    """
    parking_data[location][plate] += 1

def get_most_common_plate(location):
    """
    Get the most common plate at a specific location.

    Args:
        location (int): The parking location ID

    Returns:
        str: The most common plate
        int: The count of the most common plate
    """
    plates_counts = parking_data[location]
    most_common_plate, count = max(plates_counts.items(), key=lambda x: x[1])
    return most_common_plate, count
```

f

Refactoring of the Plate Detection Mar 18, 2023

After some discussion with Hod it is decided that the plate reader should sit within its own node. The current node can be used and then the control node is launched separate, they both remain in the same package

The refactoring was done in a step by step process that involves both separating the CNN functionality completely and also splitting it into a main .py file with vision class to handle all the machine vision functions that may be reused for control.

The launch file was remade to launch both nodes at once, then functions are transported over and refactored one at a time. At intermediate points the partial refactor is run and then rqt_viewer

directly compares the old and new version to ensure that the functionality remains unchanged. After a few hours of careful evaluation of the code, a much more polished version of the CNN is ready and the final steps are to remove all CNN functions from the controller.

The testing results are very promising, with split nodes the CNN plate recognition activity does not disturb the control node.

Mar 19, 2023

Spent a few hours today updating the github readme to be more descriptive of the project. The work done so far has been explained to Hod and some helpful tools and aliases documented. all of the CNN and vision work has documentation now so that it is easy to understand. Finally the work is added to the master branch via a pull request. This was several hours of polishing work, but now I can step away from the project for a week or two and it will be equally easy to understand after.

Revisiting Image Collection

Mar 21, 2023 The CNN for the plates likely needs a better data set. One of the issues is that I was not able to get sufficient randomization for the plate gathering engine that was automated so many images were near-duplicates creating issues with the training and validation sets having lot of similar samples. I am going to try and collect more variety again by running Specific plate generation and having the parking ID CNN automatically label the plates.

The implementation works quite well and if enough samples are taken at a particular plate, it is easy to identify if a wrong plate has been inserted because it will be a deviating label from the rest of them. One strategy to employ is to focus on the plates that are misidentified and then take snapshots from around those vantage points. In this way it is similar to the methods of the confusion matrix but also incorporates the vantage point and other factors instead of just the letter labels.

Some old code from previous commits was spliced together to get a working data engine. It is not the tidy refactored version, but it works. The data engine is kept in the my_controller package but with a separate launch file.

I should also sort the placards by data source so that I can organize them and potentially exclude some portions. The sorting was done and some addition Q and O images have been added. The number of digits versus characters is more balanced. Finally some data augmentation is added. Mainly for shifting around of the image and a small amount of Gaussian blur. The new model trainged well in just 23 epochs, testing is done in simulation

Machine Vision for Faded White Line

sdf

1. Adaptive Thresholding: Instead of using a global threshold value, adaptive thresholding sets different thresholds for different regions of the image based on local characteristics. This can help account for the variable lighting and shadows.
2. Color Space Transformation: You mentioned using HSV filtering. You might also want to experiment with different color spaces, such as LAB or YCrCb, which can sometimes provide better separation between the line and the background.
3. Edge Detection: Use edge detection algorithms like Canny or Sobel to identify the boundaries between different regions in the image. This can help emphasize the line, making it easier to detect.
4. Hough Transform: This can be used to detect straight lines in an image, even if they're partially obscured or distorted. After applying edge detection, use the Hough Transform to find the best candidates for the faded white line.
5. Machine Learning: Train a custom machine learning model, such as a convolutional neural network (CNN), to identify the line. You'll need a labeled dataset of images with the line annotated. This approach can be more robust to variations in lighting and perspective than traditional image processing techniques.
6. Morphological Operations: Apply morphological operations like dilation and erosion to emphasize the line and remove noise from the image.

Robot Controller

Mar 24, 2023

So far this documentation has only covered technical portions of the project, but there is an unforeseen challenge outside of the technical aspects that has arisen. Due to the lack of development of a robot controller I have started on the task

I've worked on machine vision for both the regular and dirt track portions of the course. The pedestrian crossing with the red patch should introduce an 'interrupt' in the service routine of the automatic control causing the robot to pause completely. Detection of the pedestrians should continue until a crossing is found and then the robot can move forward rapidly to the other side of the crossing by unpausing the automatic navigation routine.

After the first crossing the dirt track area should use a different machine vision mask that will filter out the faded lines and continue navigation, then cross the second crossing.

The reading of the 6th plate will trigger a desire to stay to the left. The plates read can be sent to the controller via a topic since plate detection is contained in a separate node. The left turn tendency will enter into the inner loop, there may need to be a control sequence to wait for the truck to pass. After it passes the truck can be followed and a right moving tendency is taken after plate 7 is read. Once plate 8 is read the vehicle can stop there and send all the plates to the competition topic.

Question

Do I just start developing a backup plan as a contingency? This seems really counter productive to me, but I also can't compel somebody else to do work that they don't feel like doing. I think I can work more on the architecture of the robot and I will implement a state system that is ready to accept a control algorithm.

Mar 27, 2023

I still haven't gotten any news back about developments in the robot controller so I've decided to at least take care of the bare minimum. When time trials come if there is nothing produced by my team mate then I can at least pass on my own efforts and then have a discussion about it after.

Problems with wiping competition environment

After reinstalling from .git the competition environment, it is no longer operating the license plate generation properly. I tried to do catkin build, etc but it did not work and the catkin clean command was not working. To cut losses I have made a backup of the two additional packages that were in the work space and wiped the whole folder. I will follow the environment setup steps again and hopefully plate gen is working again? Otherwise I can't debug the new code for reporting license plates very well.

No idea what the problem was, the fix was an overwrite from a previous copy of the gazebo folder. Should ask Miti about it.

Now testing the output to the licence plate topic and it is automated updates

```
data: "Galadin,bowser,1,HJ36"
```

The output to the topic is correct and it updates after each plate read.

As a time check I put a wrapper on the publishing function to make sure it won't use up time resources while reading plates, it doesn't.

```
Publishing plate reading for location 1 took 0.000 seconds

Location 1: Most common plate is ['H', 'J', '3', '6'] with a count of 103
Location 2: Most common plate is N/A with a count of 0
Location 3: Most common plate is N/A with a count of 0
Location 4: Most common plate is N/A with a count of 0
Location 5: Most common plate is N/A with a count of 0
Location 6: Most common plate is N/A with a count of 0
Location 7: Most common plate is N/A with a count of 0
Location 8: Most common plate is N/A with a count of 0
```

Getting ready for the time trial

Start and stop have been implemented, very easy to do. The plates are also sending correctly, but I cant view them because the score_tracker is not working. Will ask for help tomorrow.

The most basic cheater method I can think of for the time trial is to prerecord a sequence. I asked chatGPT if there is a way to record a sequence of commands to cmd_vel so will try that



To record and replay a series of commands published to a rostopic, you can use the `rosbag` tool, which allows you to record and replay ROS messages. Here's a step-by-step guide to record and replay a 5-second segment of the `R1/cmd_vel` topic:

1. Record the `R1/cmd_vel` topic:

Open a new terminal and run the following command to start recording messages published to the `R1/cmd_vel` topic for 5 seconds:

```
bash
```

Copy code

```
rosbag record -0 cmd_vel_5s.bag -d 5 R1/cmd_vel
```

This command will create a file called `cmd_vel_5s.bag` containing the recorded messages.

2. Replay the recorded messages:

To replay the recorded messages, first, make sure your robot control node is running and subscribed to the `R1/cmd_vel` topic. Then, run the following command in a new terminal:

Copy code

```
rosbag play cmd_vel_5s.bag
```

This command will publish the recorded messages from the `cmd_vel_5s.bag` file to the `R1/cmd_vel` topic, mimicking the exact control sequence you made by hand with the controller.

Remember that you need to have your robot simulation or the real robot running and subscribed to the `R1/cmd_vel` topic to replay the recorded messages properly.

So this was highly successful. I noticed that the recorded path is not completely faithful in replication, but it is quite astounding how well it records the smooth inputs from the joystick. This will work for the competition to at least position the robot at times. It could for example get the first two license plates well before a handover to controller until pedestrian crossing.

Mar 28, 2023

Talking with Miti, the score tracker has been fixed but I can't use a bag command. So the solution is going to be timed commands to the command vel topic for hard coding. To not waste time on something that will not be useful for later stages of the development, I figure it will be useful to abstract motion into forward distances and turn radius based on speed and time. With the abstraction the desired turn angle and distance can be set in the hard coding.

The result works well to get to the first pedestrian crossing. This allows now for testing of a pedestrian detection and interrupt method in the controller, since there has still been no development work done on an AI robot controller with less than 48 hours before time trials.

I am implementing a state machine for control that will allow for different sequences to be developed in parallel.

Right now the hard-coding is able to get to the second pedestrian pretty consistently. This is promising as a backup plan for the competition to avoid scoring zero points. The experimental channel for vision is opened and some prototyping for the pedestrian is in order.

Pedestrian Crossing Ideas

The pedestrian can be handled by tracking its position. The red lines are really easy to find and use as interrupts, that part was trivial with a red mask and techniques adapted from the plate homography earlier.

Next step in finding a solution is to examine the machine vision space. Asking the question: how would I describe what I need to an alien that has no idea of semantic properties of objects.

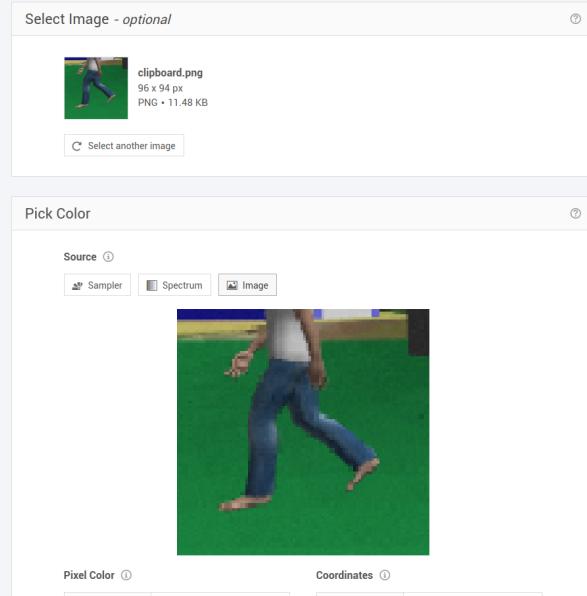
The pedestrian moves with a periodic motion left to right, if we can track this motion we can track when it is at the extreme of a position and then floor it. (It is a really aggressive pedestrian!)

Next step is to find how to track it, hsv and contours have been very useful so far. Taking a view at the hsv spectrum we see:

2

The shirt matches the white paint but the pants stand out much better. One Idea would be to track the interruption of continuity on the white crossing lines. We could also use the back red line as a split to find where it is interrupted by the ped. We could also use signal processing on the time variance of the image, from a still image, it should be possible to track changes.

The easiest solution can be tried first, then I would consider tracking time variance in the image



The pedestrian pants are uploaded to a color picker
rgb(51, 86, 121)

We can try an hsv mask on this now.

With a bit of tinkering I am satisfied that there is statistically significant data to work with:

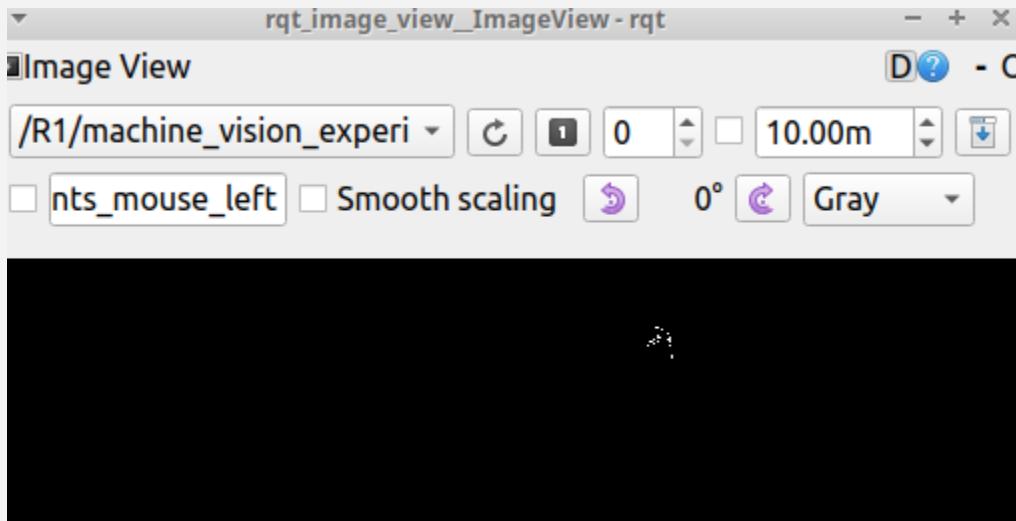
```
# Define the RGB values for the blue color of parked cars
rgb_blue = (51, 86, 121)

# Convert the RGB values to HSV color space
hsv_blue = cv2.cvtColor(np.uint8([[rgb_blue]]), cv2.COLOR_RGB2HSV)[0][0]

# Argument order is Hue, Saturation, Value
# Define the lower and upper bounds for the blue color
hsv_lower_blue = np.array([hsv_blue[0] - 5, 100, 50])
hsv_upper_blue = np.array([hsv_blue[0] + 5, 255, 150])

# Create a mask for the blue color
```

```
mask_blue = cv2.inRange(img, hsv_lower_blue, hsv_upper_blue)
```



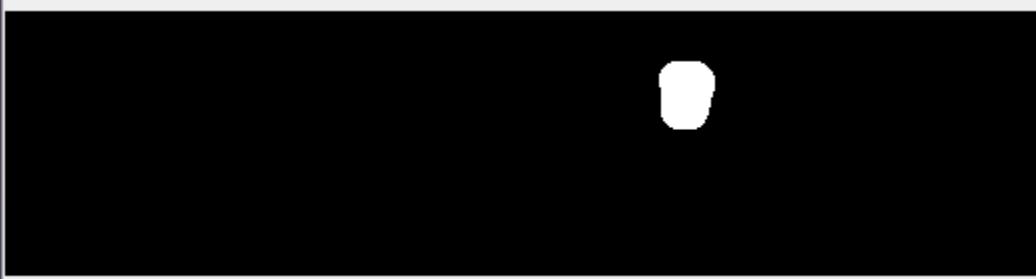
It looks very trivial but it is unique to the pedestrian and these data points can be cleaned and processed with erosion then dilation. The erosion removes small noise from other blueish sources. The dilation is done at an extreme level to get a blob

```
# Create a structuring element for erosion
kernel_erode = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))

# Apply erosion on the mask_blue
mask_blue_eroded = cv2.erode(mask_blue, kernel_erode, iterations=1)

# Create a structuring element for dilation
kernel_dilate = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (50, 50))

# Apply dilation on the eroded mask_blue
mask_blue_processed = cv2.dilate(mask_blue_eroded, kernel_dilate, iterations=1)
```



..

So the weak but *accurate* data is amplified giving a perfect contour to track across the vision space. The min/max of the center can be extrapolated and then now we are working with simple positional data that is reliable.

The easiest solution was quickly prototyped to determine feasibility using known tools, with some backup ideas in mind. It showed promise so the data was enhanced, and then tested at both crossings. This vision filter detects pedestrians from halfway across the map. It works well with the hardcoding in testing and will likely be valuable for interrupting an autonomous agent.

Thoughts After Hardcoding

After doing the hardcoding I can see some patterns emerge that could be used in the competition.

Some ideas:

1. PID is actually a classic agent and the white lines are very well defined, why not use it? I'd like to experiment with an offset white line to one side on the right
2. The plate detection needs a bit of tweaking for some letters and especially the ones that are located on the dirt track.
3. Hard code is actually very viable to get all the way to the first pedestrian crossing.
4. There is a specific texture that is in the distance at the end of the dirt track. I think I can hardcode crossing 2nd crossing and then latch onto that texture in the distance to produce smooth sailing through that section. It could prove more reliable than PID on faded lines.
5. Relatch onto a white line on the other side, then implement PID following on the truck.

Mar 31, 2023

Developing Realignment Methods and Prototyping ‘Exploit’ Methods

The viability of hardcoding on segments of the course was somewhat surprising after time trials. The

main issue with the coding is the summation of errors over time that compound. A realignment method would fix this issue and the more that I think about it, the more I can see multiple patterns for recognition that are not related to the road. I've had the chance to explore making a CNN with license plates already, I understand the principles for machine learning controller agents– but it is two weeks from the competition and the cutoff window for trying higher risk methods is closing.

The scope of the competition environment is not necessarily as complex as it seemed at first. The outer track sections visually differ and have pedestrian crossings on some, but now that pedestrian crossings have been separately handled, I can see that the four sides of the outer track are all nearly identical from a 2D perspective. The question to ask now is, what level of complexity is actually required for an agent and what is the simplest solution path available?

Some factors that I have considered are:

1. I have gotten very fast at developing machine vision tools
2. The controlled simulation environment is very static compared to the real world, ML agents offer advantages in tackling *new* scenarios, but those advantages are not needed for the comp environment.
3. A more deterministic routing would enhance the plate detection, the robot will always see the plates from nearly the same path.
4. The road is mostly straight stretches on the outer track
5. The simulation is full of straight lines that can be used for alignment purposes, the lighting never changes

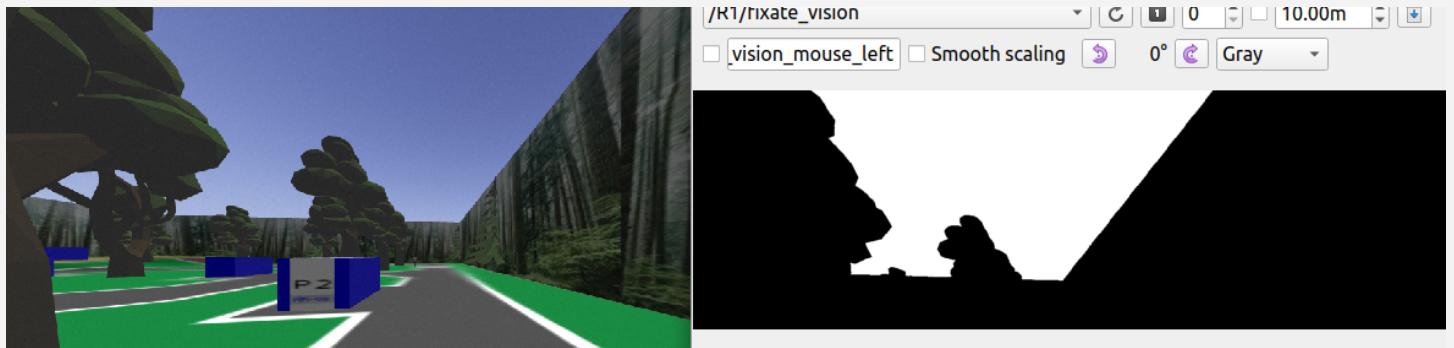
The Exploit

Given all of these factors the first idea that I had in mind was looking at doing SIFT on the forest texture at the end of the road in the simulation environment. We observe that the corner texture is always the same 3 standing trees. The difficulty is that the texture is also repeated to the left again, SIFT is also computationally intensive and I am not sure about its viability when using it as a trigger for recentering the robot. One observation is that the texture is never rotated whatsoever and only has perspective transform from distance and yaw, so there are surely simpler computational methods for it.

Using a turn realignment on the end of the road will also solve the error from path distance differences in the hard code, because if the robot runs long or short, it still should always realign to drive to the same end point on the road.

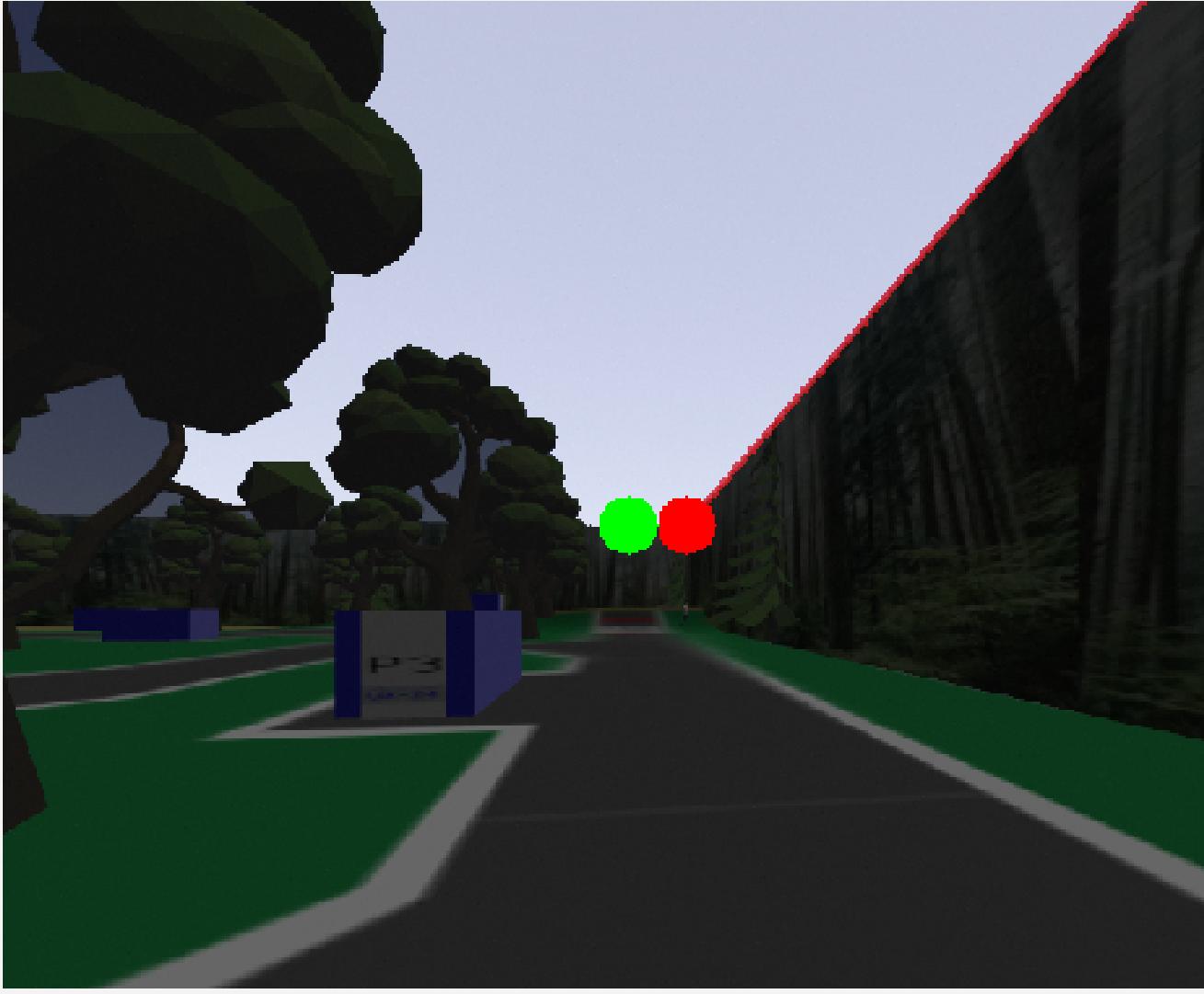
Looking to the stars

Traditional navigation techniques for sailing would use the stars at night with the north star indicating the compass directions. During the day there were no stars available and only the sun, so some amount of dead reckoning was necessary to estimate the distance and direction traveled. With an accurate clock, both latitude and longitude eventually could be determined depending on the weather conditions. This same approach can be used here. Dead reckoning with periodic realignment– and the simplest attack on the problem is the sharp lines from the sky contour.



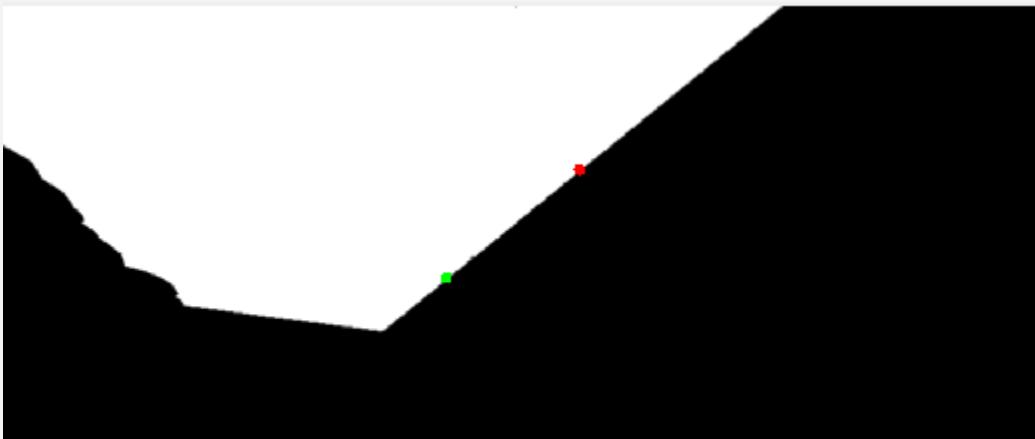
The color picker + hsv filter tuning that was used for plates and pedestrian detection is reused here. It is basically copy paste of previous code in the code base. The largest contour is always going to be on the right. The perspective and compatibility for all four corners was tested to make sure it is viable. The desired goal is to pinpoint the vertex forming the corner of the skybox, which is very near the end of the road.

Given that it is two lines forming it the first naive approach was a failure, making the contours polygonal does not release the desired vertex, even with more image conditioning. So the next step to try is some type of line filter. The line filter has trouble with diagonal lines due to the staircase pattern of pixelation and small elements of noise. In addition occasionally there is the tip of a tree silhouette poking through into the contour disrupting the line. Often the line wouldn't continue until the vertex. A bit of stubborn headbanging ensued, trying to tweak the tool to do the job, but it was not satisfactory due to issues with reliability but also the enormous computational load from the probabilistic line detection.

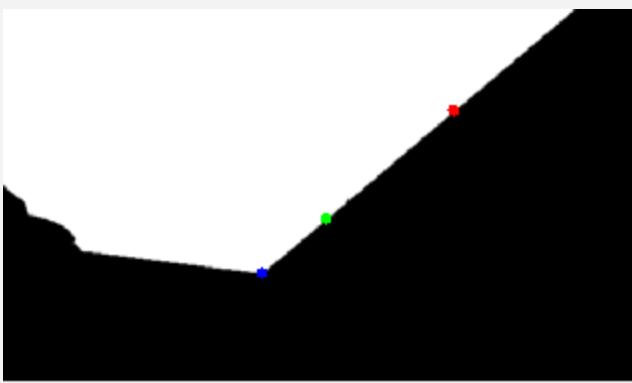


Some energy was expended in trying to improve the situation but it brings up the question, what is the initial problem we are solving? It is to find the vertex– this should not be so difficult, it is plain as day to the human eye. Instead of using prepackaged tools from OpenCv, the logical step is to find it manually.

Starting from the largest contour after some inspection is is clear that there will always be two points on the right side that fall on the line. They can be found by sweeping inward from the right side at two heights, after cutting out the sky:

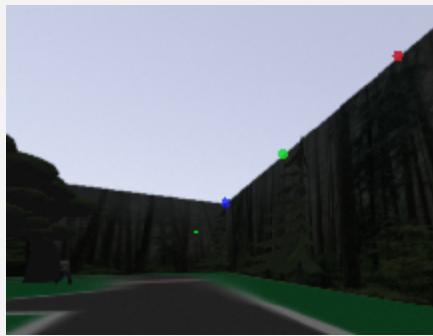


The first step to confirm the conclusion and that the points are found is by drawing them on a debug console. Once I was satisfied with that, the line can be extended by fitting a line to the two points. We run downwards until hitting black again. To avoid false flags due to noise, we sweep a path that is some threshold to either side of the line path searching for white and terminate when it is all black:

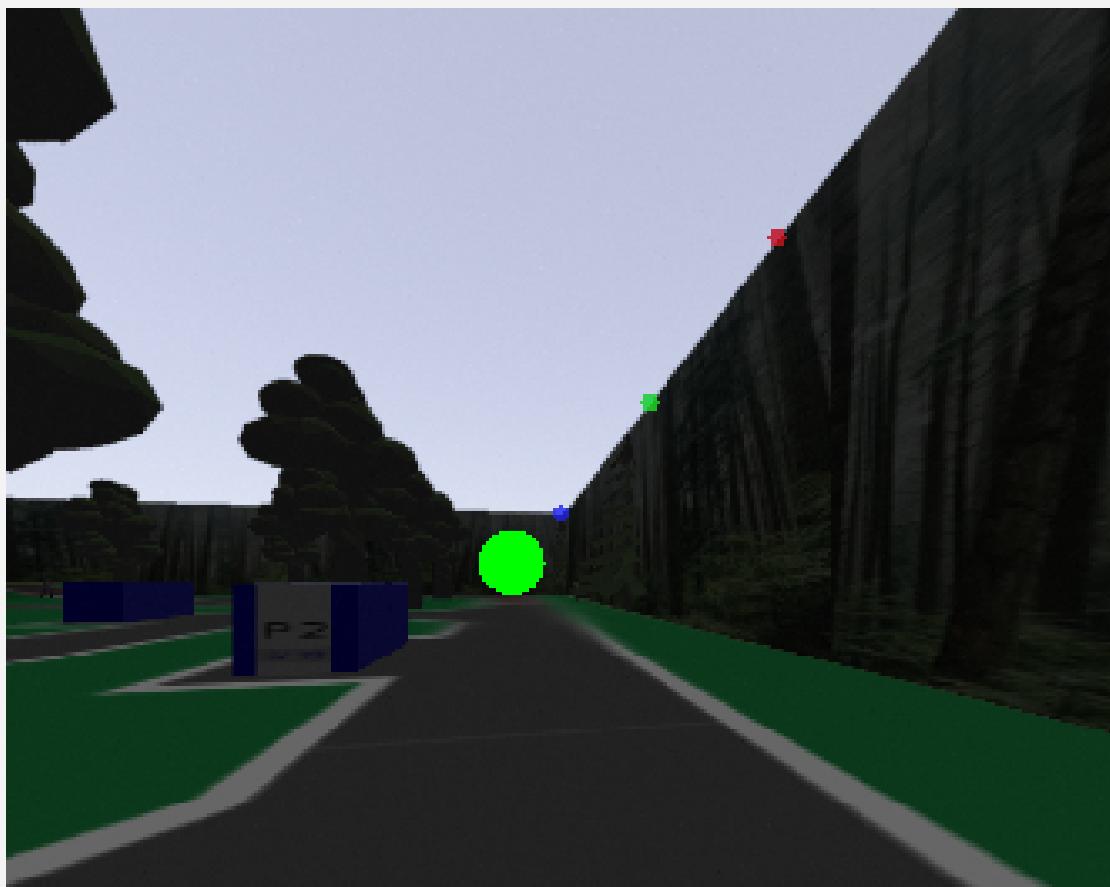


The vertex is found and tuning on the algorithm performed by inspecting the blue dot at all four corners under competition conditions.

The next step is to overlay the mask onto the raw image we start with to check relative distance from the blue dot to where we want to go:



The extension point in green is a fixed distance away. More clever math could make it indicate end of path based on other cues, but we are always searching from the other end of the road. It is a consistent view point, and all that matters is that the dot is aligned when facing the correct position head-on:



The circle is a radius of 20 pixels, it helps to indicate how severe the error tolerance is. The coordinate of the pixel is also exported with the debug image.

Instead of PID, we only need to latch onto this point once in a function called ‘fixate()’

fix·ate

1.

cause (someone) to acquire an **obsessive** attachment to someone or something.

"she has for some time been fixated on photography"

2.

TECHNICAL

direct one's eyes toward.

"subjects fixated a central point"

Instead of PID, we only need to latch onto this point once in a function called 'fixate()'

It is a simple while-loop, the condition for it is that the camera is turned to the right of the desired point, so we hardcode understeering in the turns, then fixate runs at a diminished rotation speed to get precision down to ~3 pixels of the desired point when the stop command is given to twist.

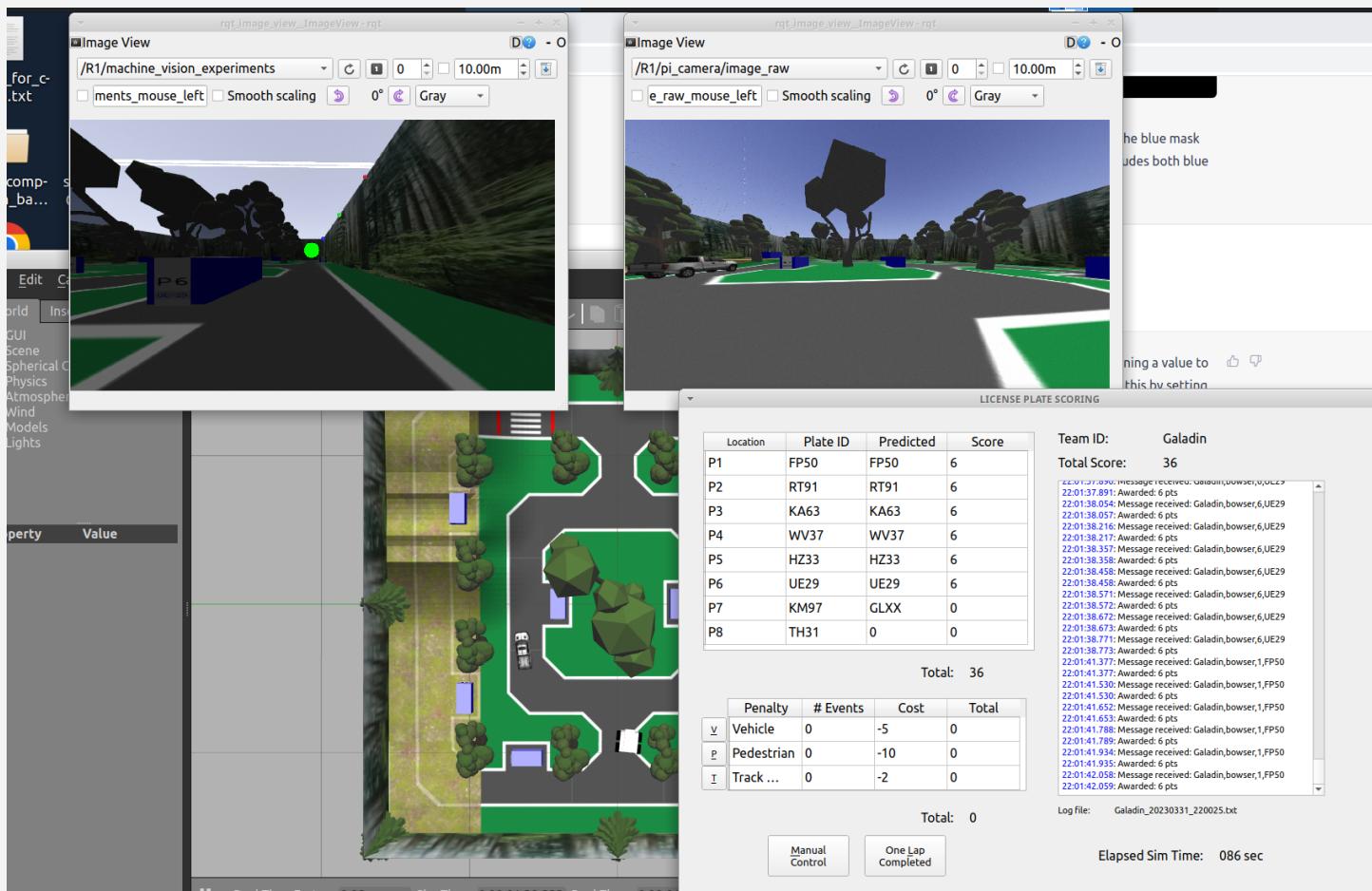
Because the robot runs perfectly true and straight, this is a viable option. If the robot controls had some element of gaussian noise or drift built in, then it would correct such strategies from being possible and require more consistent feedback points along the course.

The fixation searches to align the setpoint (green dot) with the exact center column of the camera, ensuring alignment. The function can accept a different setpoint for tuning if needed.

The hardcoded functions are updated to include turning while moving forward, this allows for avoiding the stop start of single value twist messages. After reducing times at pedestrian crossings, the implementation does full laps, reading all 6 plates and is positioned to wait for the truck, all within 45 seconds of competition start. For me personally, I am impressed by just how effective this method is.

For the inner loop the task is small enough that I think a proper PID feedback can be placed on the inner line with a mask over the blue and white values to get parked cars and the road lines. Then we detect the edge, instead of contour centroids, and PID off of that. I think that would be a very successful strategy.

Once the full control sequence is running, the training set for plates can be updated with captures from the exact competition path, and I expect this will give 100% accuracy and that the competition can be completed in about 1min 20sec or so.





To let your PID agent run for 30 seconds and then terminate it, you can make the following changes to your code:

1. Add an attribute to your `PIDAgent` class to track whether it should continue running or stop.
2. Modify your `image_callback` function to check the status of the attribute before executing.
3. Create a separate function to stop the agent and send the stop robot message.
4. Use a timer to call the stop function after 30 seconds.

Here's the modified code:

```
python
import threading

class PIDAgent:
    def __init__(self):
        # ...
        # Add this attribute to control the agent's running state
        self.should_run = True

    def image_callback(self, data):
        # Check if the agent should continue running
        if not self.should_run:
            return

        # ...

    def stop_agent(self):
        self.should_run = False
        # Send the stop robot message
        twist = Twist()
        twist.linear.x = 0
        twist.angular.z = 0
        self.cmd_vel_pub.publish(twist)

    # ...
```

Copy code

Regenerate response

One minute full score achieved Apr 1, 2023

Location	Plate ID	Predicted	Score
P1	UC19	UC19	6
P2	RY96	RY96	6
P3	NH46	NH46	6
P4	ZS85	ZS85	6
P5	OC64	OC64	6
P6	LO82	LO82	6
P7	ML00	ML00	8
P8	HP48	HP48	8

Total: 52

	Penalty	# Events	Cost	Total
V	Vehicle	0	-5	0
P	Pedestrian	0	-10	0
I	Track ...	0	-2	0

Total: 0

Manual Control

One Lap Completed

Team ID: Galadin

Total Score: 52

```

19:04:47.555: Message received: Galadin,bowser,7,ML00
19:04:47.555: Awarded: 8 pts
19:04:51.795: Message received: Galadin,bowser,8,2SB5
19:04:51.797: Awarded: 0 pts
19:04:51.926: Message received: Galadin,bowser,8,HP48
19:04:51.927: Awarded: 8 pts
19:04:51.938: Team total: 52 pts
19:04:52.361: Message received: Galadin,bowser,8,HP48
19:04:52.362: Awarded: 8 pts
19:04:52.490: Message received: Galadin,bowser,8,HP48
19:04:52.491: Awarded: 8 pts
19:04:52.622: Message received: Galadin,bowser,8,HP48
19:04:52.623: Awarded: 8 pts
19:04:52.768: Message received: Galadin,bowser,8,HP48
19:04:52.769: Awarded: 8 pts
19:04:54.535: Message received: Galadin,bowser,8,HP48
19:04:54.536: Awarded: 8 pts
19:04:54.669: Message received: Galadin,bowser,8,HP48
19:04:54.670: Awarded: 8 pts
19:04:54.809: Message received: Galadin,bowser,8,HP48
19:04:54.809: Awarded: 8 pts
19:04:54.907: Message received: Galadin,bowser,8,HP48
19:04:54.908: Awarded: 8 pts
19:04:55.015: Message received: Galadin,bowser,-1,PA55
19:04:55.016: Awarded: 8 pts
19:04:56.353: Message received: Galadin,bowser,-1,PA55
19:04:56.364: Timer stopped: 59.328sec sim time (real time: 60sec).

```

Log File: Galadin_20230401_190355.txt

Elapsed Sim Time: 058 sec

The inner track PID controller Apr 3, 2023

It was implemented very quickly because PID was already set up in the line following lab. At first I made a threshold for the white and blue areas. I tried to process the image into something more reliable for PID but the solution was not obvious. Then I remembered picking up the road surface earlier when I was trying to detect plates. The HSV is changed to find road surface and contours.

The turns look like they will cause a problem with the robot exiting. I took the prototype PID from the lab and setup a new agent object in a separate .py script. Before doing PID though, I checked to see if it is plausible by displaying the xcoord of the centroid of the largest contour over the entire image of the

camera. Then I used the controller to follow the signal and see where it leads. This was very insightful for path detection and it looked promising.

PID was adjusted with only P, A new launch file allows for booting up the agent on its own for testing. It worked right away after just a bit of tuning. After finishing the tuning some work was done to add the agent to the last machine state of the robot. Now all that is missing is a truck detection sequence.

PID on fixate

Now there is a chance to optimize even more. The fixate function is not reliable if the turn overshoots and it needs to run slow. The solution to this problem is to implement a simple PID that can be tuned so that maximum speed response is achieved with the robot realignment. In the original context of coding fixate I included at set point without even thinking about PID, but it is very conveniently placed in the code ready to go.

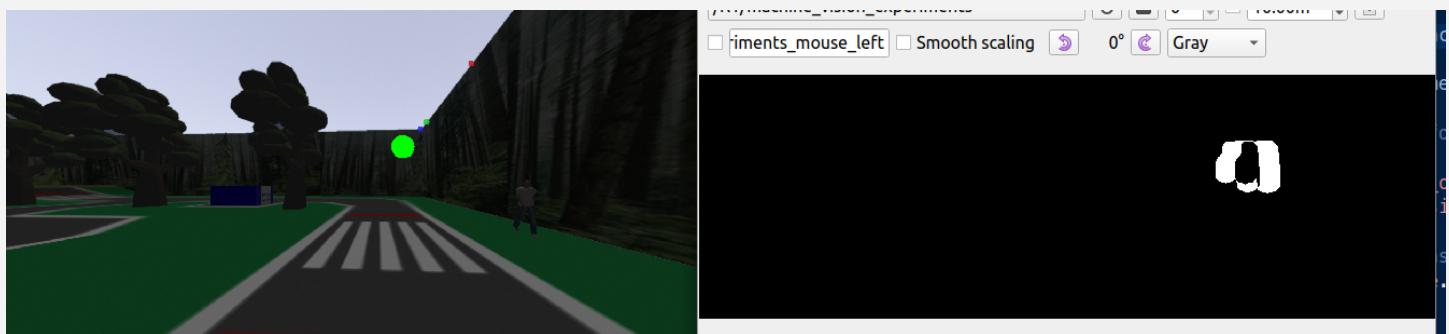
Apr 4, 2023

Tonight I have been working on getting fixate working better. Now it can rotate from either side and the decision making is handled by PId, the error tolerance for the final position can also be lowered now. I am very happy with the results, it has increased both speed and reliability.

Pedestrian Crossings Optimized via Time-Differential Framing of Pedestrian Pants

The pedestrian crossings are a very slow point now. I changed it so that it waits only for 2 extrema instead of 3. The ultimate solution is to go as soon as possible. I checked the pedestrian again and notice a very key pattern. It stands still and then rotates before crossing. So rotation and crossing indicate that it is not safe to cross. My idea now is to redo the detection to search for consecutive **frames** instead of positions. This will indicate if the crossing can be jumped early and greatly improve the track times.

That was very fast to implement since the bitwise image was already available. U implemented a last image saved field. Then the rotation in the pants shows up very clear, it is actually one of the coolest machine vision techniques I have found so far. It is super fast to compute the non-zero pixels with another cv2 function cv2.countNonZero to quantify the changes. The streaming data is checked in the console through print, it works perfect!



Above is the moving pedestrian



This is the spinning pedestrian, also shows up clearly

When the value is below 10 then the pedestrian is still **and** not about to move.

With some testing in place, it has saved 10-15% of the time which is a huge gain. There are now **35-40 second runs possible**, not accounting for the truck detection which still needs to be implemented. This is extremely satisfactory and beyond what I could have ever expected. The hardcoding segments allow for perfect optimization and speed on the license plate drive bys that would simply not be possible with an autonomous controller that has too much variation built in

Apr 4, 2023

Project benchmark as of April 4th is under 40 seconds. The plate recognition could potentially use a bit of tuning with a training enhancement set that can be gathered from the control path.

LICENSE PLATE SCORING

Location	Plate ID	Predicted	Score
P1	EA04	EA04	6
P2	DS32	DS32	6
P3	KV48	KV48	6
P4	GI84	GI84	6
P5	YL27	YL27	6
P6	CY04	CY04	6
P7	FV15	FV15	8
P8	OM22	OM22	8

Total: 52

	Penalty	# Events	Cost	Total
V	Vehicle	0	-5	0
P	Pedestrian	0	-10	0
I	Track ...	0	-2	0

Total: 0

Manual Control

One Lap Completed

Team ID: Galadin

Total Score: 52

```

11:53:24.837: Awarded: 8 pts
11:53:24.967: Message received: Galadin,bowser,8,OM22
11:53:24.967: Awarded: 8 pts
11:53:25.098: Message received: Galadin,bowser,8,OM22
11:53:25.098: Awarded: 8 pts
11:53:25.243: Message received: Galadin,bowser,8,OM22
11:53:25.243: Awarded: 8 pts
11:53:25.370: Message received: Galadin,bowser,8,OM22
11:53:25.371: Awarded: 8 pts
11:53:25.500: Message received: Galadin,bowser,8,OM22
11:53:25.500: Awarded: 8 pts
11:53:25.575: Message received: Galadin,bowser,-1,PA55
11:53:25.576: Timer stopped: 36.572sec sim time (real time: 36sec).
11:53:25.639: Message received: Galadin,bowser,8,OM22
11:53:25.640: Awarded: 8 pts
11:53:25.774: Message received: Galadin,bowser,8,OM22
11:53:25.774: Awarded: 8 pts
11:53:25.906: Message received: Galadin,bowser,8,OM22
11:53:25.907: Awarded: 8 pts
11:53:26.036: Message received: Galadin,bowser,8,OM22
11:53:26.036: Awarded: 8 pts
11:53:26.172: Message received: Galadin,bowser,8,OM22
11:53:26.173: Awarded: 8 pts
11:53:26.313: Message received: Galadin,bowser,8,OM22
11:53:26.313: Awarded: 8 pts
11:53:26.436: Message received: Galadin,bowser,8,OM22
11:53:26.437: Awarded: 8 pts

```

Log File: Galadin_20230404_115248.txt

Elapsed Sim Time: 035 sec

Truck Detection



Pixel Color ⓘ

Coordinates ⓘ

Pixel Color

#383838

Pixel X 174

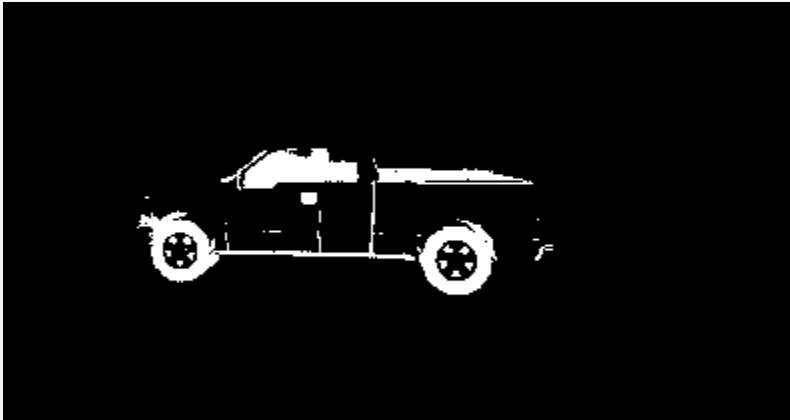
Magnifier

Zoom in

Zoom out

Pixel Y 77

rgb(56, 56, 56)



Recycling the image filtering from the road surface, the black windows and black tires come out the strongest, this is a good image to use for differencing on.

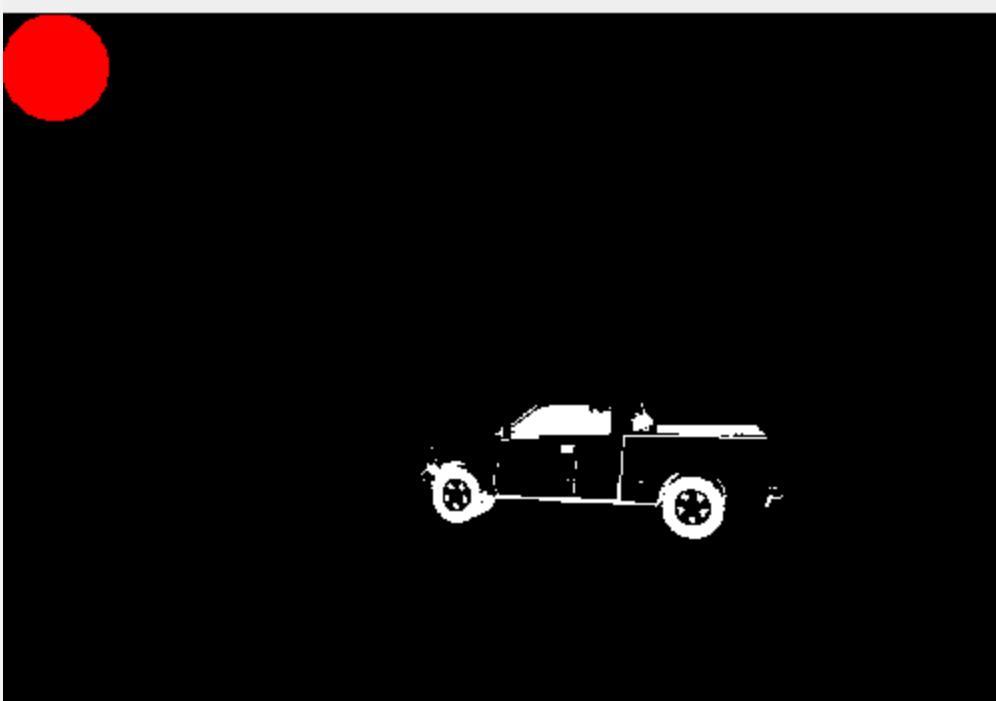
Next step in the tool building flow is to view the differences and send the value to a console for live observation.

```
img = vision.thresholdTruck(self.image)
reading = cv2.countNonZero(img)
print(reading)
|
if(img is not None):
    self.publish_to_experimental_vision(img)
```

The truck is always in motion, there is no need for diff, we can just count the amount of white pixels directly from one frame. The truck reads at under 100 when on farside and at 10000 when close, this should be a sufficient trigger.

I select 10,000 to start with and now print out “NO GO” when the threshold is hit. I can test this feedback from different positions and angles to find a suitable threshold.

Even easier than print is a red or green light



Now I want to only red-light if the truck is about to cut the robot off, but I would like to experiment with faster inner-loop PID. So we cut out much of the right side and then lower the threshold to get a left biased stop call.

An extra blackout box to the right obstructs the false flag area of the camera view:



This is ready for processing and testing.

More Testing

Testing indicates the the robot still enters in after the truck too soon to avoid collision before P8. Rather than slow the robot down, the vision is fine-tuned with another blackout square. Only the front and left parts of the view are read for the truck allowing for higher sensitivity and a greater delay before the robot enters. This has worked well after nmore testing, I consider the truck issue to be closed.

Last steps, optimizing the PID inner loop for faster speed

The PID is retuned with driving much faster. One think that strikes me is how effective this is as an agent even for the outer loop. I think it could have been a viable option to explore in the early months of the competition. The new PID is deployed and it works great. The speed is set so that it will not collide with the truck if it has enough of a head start and it will wait for the truck if it is in the interference zone.

The track time is at around 31 seconds now, it seems not very productive to optimize much more. It would be better to consolidate gains and test for reliability so that there is no surprise at the competition

That is about a wrap on things. Some minor todos are to tune the truck and inner loop pid. A confusion override for letter-digit confusion could be implemented, I saw one plate misread due to Z=2.

But in fact the retraining of model C4 has completely taken care of this, the plates are always classified correct now.

Location	Plate ID	Predicted	Score
P1	YE31	YE31	6
P2	UT22	UT22	6
P3	TT74	TT74	6
P4	OJ08	OJ08	6
P5	HV81	HV81	6
P6	VB73	VB73	6
P7	UF51	UF51	8
P8	OD26	OD26	8

Total: 52

	Penalty	# Events	Cost	Total
V	Vehicle	0	-5	0
P	Pedestrian	0	-10	0
I	Track ...	0	-2	0

Total: 0

Manual
Control

One Lap
Completed

Team ID: Gadalin

Total Score: 52

```

23:15:29.509: Awarded: 8 pts
23:15:29.310: Team total: 44 pts
23:15:30.446: Message received: Gadalin,bowser,7,UF51
23:15:30.447: Awarded: 8 pts
23:15:30.571: Message received: Gadalin,bowser,7,UF51
23:15:30.572: Awarded: 8 pts
23:15:30.701: Message received: Gadalin,bowser,7,UF51
23:15:30.702: Awarded: 8 pts
23:15:33.534: Message received: Gadalin,bowser,8,IQ08
23:15:33.535: Awarded: 0 pts
23:15:34.538: Message received: Gadalin,bowser,8,IQ08
23:15:34.547: Awarded: 0 pts
23:15:34.809: Message received: Gadalin,bowser,8,IQ08
23:15:34.810: Awarded: 0 pts
23:15:34.935: Message received: Gadalin,bowser,8,OD26
23:15:34.936: Awarded: 8 pts
23:15:34.937: Team total: 52 pts
23:15:35.075: Message received: Gadalin,bowser,8,OD26
23:15:35.075: Awarded: 8 pts
23:15:35.200: Message received: Gadalin,bowser,8,OD26
23:15:35.201: Awarded: 8 pts
23:15:35.344: Message received: Gadalin,bowser,8,OD26
23:15:35.345: Awarded: 8 pts
23:15:35.357: Message received: Gadalin,bowser,-1,PA55
23:15:35.358: Timer stopped: 29.333000000000002sec sim time (real time: 29sec).
23:15:35.471: Message received: Gadalin,bowser,8,OD26
23:15:35.472: Awarded: 8 pts

```

Log file: Gadalin_20230406_231505.txt

Elapsed Sim Time: 028 sec

Retraining CNN on Path Data

Apr 5, 2023

The time has come to retrain CNN and get a higher reliability from it. To do so, I made a new branch of the repo and then made some direct edits into the controller and plate reader.

The plate reader borrows some code from earlier plate generation in that it makes a saved images folder and then saving the homography is inserted into the code base where it normally appends a plate value to the dictionary. It was fairly quick to setup with some small changes from code drift over time. The code base has started to become a bit bloated as I am finding little relics from earlier iterations. I think this is a normal process when doing so much prototyping. The main repo can be

refactored at some point to give it a cleaner appearance if there is time.

Main Notes

The plates are being picked up from too far away. This was a design feature to maximize plate data at an earlier stage of the competition but now the path of motion is much more certain. I reduced the contour threshold so that the plate readings that do come in are more clear and can train on this data set.

Bash Scripting

The bash script is modified to launch the robot_controller launch and it is verified to be saving correctly. I added a spin away from plate 8 at the end to avoid overcollecting on it. The plan now is to gather a large set of representative data from the competition path. This can be used to train a new CNN on the same model as earlier, but hopefully more focussed. I plan to level out the data from both letters and plate position if possible. I even would like to process the parking ID first and then feed that into the CNN as an 8-value 1-hot along with the character position.

This will require some restructuring in the Jupyter notebook but should yield some very good results I think.

Reflection

I feel like I wasted a lot of time on a plate gathering engine earlier and gathering all the plates when it would have been way more simple to do it this way. I am not sure if it is from being more familiar with the tools that it is so easy now. Was it a necessary step to painstakingly find 8 camera positions to snap photos overnight when now I have a superpowered bot? The parking ID readings are so accurate that I will go through and check after to make sure that the IDs read came out in order or for any odd looking photos.

Let it run overnight...

Apr 5, 2023

10,000 Plates are gathered, a manual verification process involved checking the Parking ID versus the label to ensure no mislabeling. A few poisonous samples were found and removed. Next I want to balance the plate evenly across all parking IDs randomly. The easiest way to get the job done fast is to request a bash script from ChatGPT:



I have a folder with 10000 labeled images in it. Every filename starts with P# where the # is an integer character from 1 to 8. I want to randomly select an equal number of plates from each-- i.e. P1,P2,P#,.....

Count the value that has the minimum number of plates and then randomly select that quantity of plates from all 8 options so there is an even spread across the 8 types and they are randomized

I am in linux, plates are in

/home/sghyseli/gadalin/cnn_trainer/cnn_alpha/placards/live_plates/



You can use the `find` command along with `shuf` and `head` to accomplish this task. First, let's count the minimum number of plates for each type (P1, P2, ..., P8) and store the minimum count:

bash

Copy code

```
min_count=$(for i in {1..8}; do find /home/sghyseli/gadalin/cnn_trainer/cnn_alpha/placards/
```

Now, you can randomly select an equal number of plates from each type and store the result in a new folder, e.g.,

'/home/sghyseli/gadalin/cnn_trainer/cnn_alpha/placards/selected_plates/'

bash

Copy code

```
mkdir -p /home/sghyseli/gadalin/cnn_trainer/cnn_alpha/placards/selected_plates/
for i in {1..8}; do
    find /home/sghyseli/gadalin/cnn_trainer/cnn_alpha/placards/live_plates/ -type f -name "P$done
```

Easy

I run all of the previous cells from the Jupyter notebook and then look at also restructuring the CNN

for incorporating the positional information earlier in the features of the CNN. It gets inserted after the first max_pooling where the primitive features lie. This gives some room for the positional parameters to exert influence in the CNN architecture

```
# Define CNN architecture
# Input has been shaped to (35, 35, 1) to match the shape of the image
# We hope to get our first subfeatures using two layers of convolution and max pooling
image_input = Input(shape=(35, 35, 1), name='image_input')
conv1 = Conv2D(32, (3, 3), activation='relu')(image_input)
conv2 = Conv2D(64, (3, 3), activation='relu')(conv1)
maxpool1 = MaxPooling2D((2, 2))(conv2)

# Inject the positional data after the first max-pooling layer
positional_input = Input(shape=P_dataset.shape[1], name='positional_input')
positional_reshaped = Dense(maxpool1.shape[1]*maxpool1.shape[2]*maxpool1.shape[3], activation='relu')(positional_input)
positional_reshaped = Reshape((maxpool1.shape[1], maxpool1.shape[2], maxpool1.shape[3]))(positional_reshaped)

concatenated_features1 = concatenate([maxpool1, positional_reshaped])

conv3 = Conv2D(128, (3, 3), activation='relu')(concatenated_features1)
conv4 = Conv2D(256, (3, 3), activation='relu')(conv3)
maxpool2 = MaxPooling2D((2, 2))([conv4])

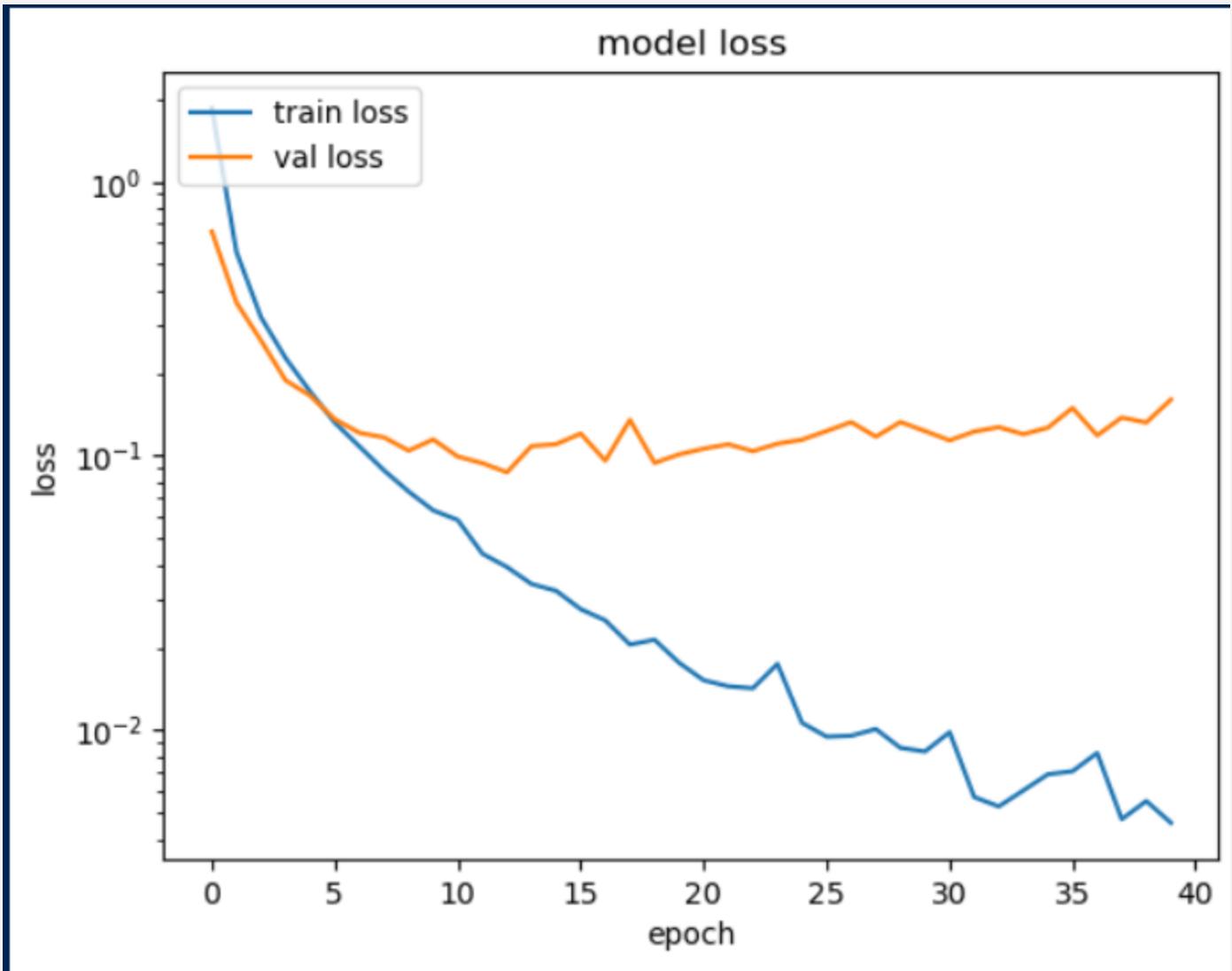
flatten = Flatten()(maxpool2)
dropout1 = Dropout(0.5)(flatten)
dense1 = Dense(256, activation='relu')(dropout1)
dense2 = Dense(128, activation='relu')(dense1)

# Add the final classification layer
num_classes = 36 # Set the number of classes for your character recognition task (A-Z and 0-9)
final_output = Dense(num_classes, activation='softmax', name='final_output')(dense2)

# Create the new model with the two inputs and the final output
model = Model(inputs=[image_input, positional_input], outputs=[final_output])
```

It does not alter the input or outputs which means the codebase does not need to be changed

Epoch 13 is chosen as the right balance to avoid over-training



I call this configuration **ModelC1**

Search					1-10 ▾ of 14	<	>
Name	Project	State	Created	User			
andorian-enterprise-10	Alpha_Chars	finished	16 minutes ago	sghyseli			
easy-wave-9	Alpha_Chars	finished	2 weeks ago	sghyseli			
exalted-water-8	Alpha_Chars	finished	2 weeks ago	sghyseli			
gallant-dawn-7	Alpha_Chars	scrapped	2 weeks ago	sgbussell			

It is andorian on WandB

The new structure and training was a failure:

```
Location 1: Most common plate is ['V', 'P', '9', 'B'] with a count of 8
Location 2: Most common plate is ['L', 'X', '8', '4'] with a count of 7
Location 3: Most common plate is ['W', 'K', '2', '6'] with a count of 8
Location 4: Most common plate is ['M', 'P', '5', 'B'] with a count of 4
Location 5: Most common plate is ['W', 'M', '6', '3'] with a count of 6
Location 6: Most common plate is ['W', 'F', 'Z', 'T'] with a count of 8
Location 7: Most common plate is ['W', 'H', '9', '4'] with a count of 5
Location 8: Most common plate is ['E', 'E', '1', 'B'] with a count of 30
```

So either the data set is too small that was used, or the previous model structure with a later addition of position info was correct.

I also added some of the AA11 plates gathered with the joystick to increase the size of the set

ModelC2 - is trained only on the path data gathered so far

ModelC3 - is trained on the AA11 data and the image enhancement is more similar to before

Verdict Model C2

The CNN having positional injection earlier in the layering was the key mistake. The new CNN on the same set of data that uses some of the image enhancements from before (shifting images and 2,2 gaussian blur) along with the earlier model has a stellar performance. It is not misclassifying any of the plates. I am extremely satisfied with the result and think that it is competition ready. As a benefit it even is allowing to optimize some of the plate sweeps more, removing time from the runs. Now the time is down to 32-36 seconds.

I also modified the bash script to run repeated test runs with the correct rqt_viewer windows open. This greatly speeds up the testing process for multiple runs.

The next area for improvement is going to be the inner loop and the truck.

Increasing Resolution and Training C4

There are still some plates being misread, so I think there is a capacity to train at a higher resolution that is closer to that of the original image. The 35x35 images are hard for my human eye to decipher compared to the raw images. I could also look into a slower learning rate with the large set of data, and also changing the image processing even.

The slower learn rate and higher resolution worked successfully, the plate reads are ready for competition. All the runs appear to be full score now and there are no character misreads or confusion. I tracked 10 full score runs in a row now, so 80/80 plates

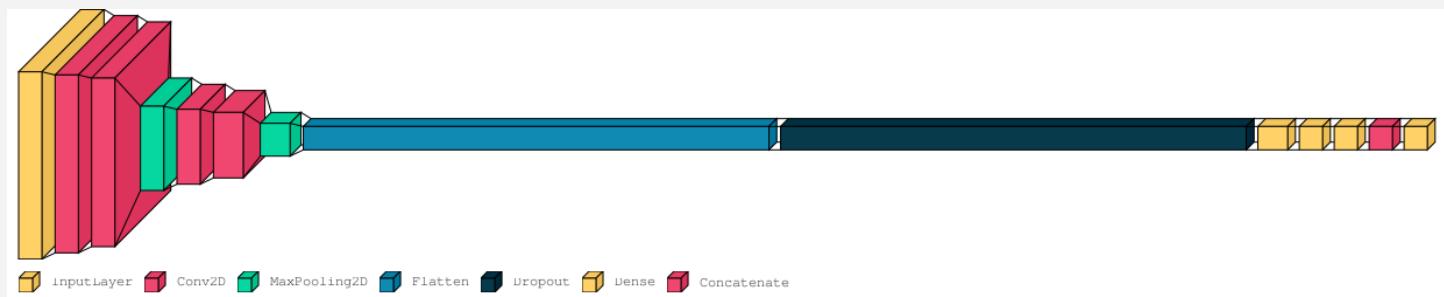
This is going to work for a full score run on the competition day. The scratch work from the separate branch is ported over to simon-controls where the final model is being built. A final rerun in the notebook to get a fresh data modeling of the training

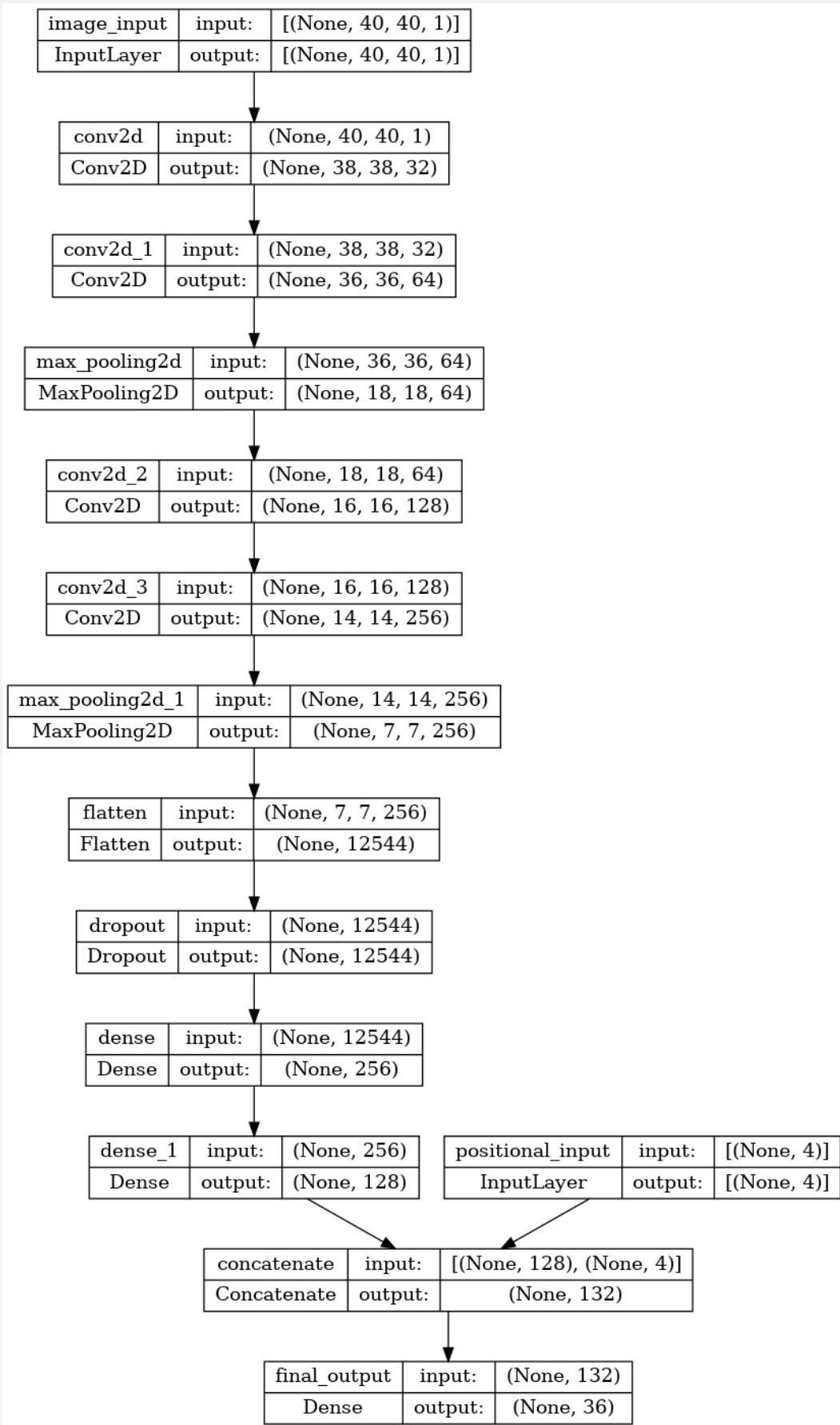
I created this model based on some pieces from the character recognition lab with some added complexity in the layers. The main principle is to have two max pooling steps, one at low level and one at higher level features. Flatten shapes the input ready for dense layers and dropout adds robustness to the training by varying which inputs to the dense layers are trained on. Two dense layers allow for the neural network interaction of features. A final dense layer brings the positional data into the picture with the 128 final extracted features for a final softmax activation.

`image_input`: This Input layer accepts input images with a shape of (40, 40, 1), where 40x40 is the image size, and 1 represents a single grayscale channel.

1. `conv1`: The first Conv2D layer applies 32 filters of size (3, 3) with a ReLU (Rectified Linear Unit) activation function. This layer is responsible for detecting low-level features, such as edges and corners, in the input image.
2. `conv2`: The second Conv2D layer applies 64 filters of size (3, 3) with a ReLU activation function. This layer detects higher-level features, such as shapes and textures, by combining the low-level features from the previous layer.
3. `maxpool1`: The first MaxPooling2D layer reduces the spatial dimensions of the previous layer's output by taking the maximum value within non-overlapping 2x2 regions. This operation reduces computational complexity and helps the model become more invariant to small translations in the input image.
4. `conv3`: The third Conv2D layer applies 128 filters of size (3, 3) with a ReLU activation function. This layer detects even more complex features by combining the outputs of the previous layers.
5. `conv4`: The fourth Conv2D layer applies 256 filters of size (3, 3) with a ReLU activation function. This layer further refines the complex features learned in the previous layers.
6. `maxpool2`: The second MaxPooling2D layer further reduces the spatial dimensions of the previous layer's output, again using a 2x2 pooling window.
7. `flatten`: The Flatten layer reshapes the output from the last max-pooling layer into a one-dimensional vector, preparing the data for the fully connected layers (Dense layers).

8. `dropout1`: The Dropout layer randomly sets a fraction (0.5) of the input units to 0 during training, helping to prevent overfitting by reducing the reliance on any single feature.
9. `dense1`: The first Dense layer consists of 256 neurons with a ReLU activation function, which processes the high-level features extracted by the previous layers.
10. `dense2`: The second Dense layer consists of 128 neurons with a ReLU activation function, further processing the high-level features.
11. `positional_input`: This Input layer accepts the positional data (P_dataset) as additional input to the model.
12. `combined_features`: The concatenate operation combines the outputs from the previous Dense layer (dense2) and the positional_input layer. This step allows the model to utilize both image-based features and positional information for character recognition.
13. `final_output`: The final Dense layer has 36 neurons, corresponding to the number of classes (A-Z and 0-9). It uses the softmax activation function, which calculates the probabilities for each class. The highest probability corresponds to the predicted character.
14. `model`: The Model function creates a new model with the two inputs (image_input and positional_input) and the final output. This combined model is then used for training and character recognition.

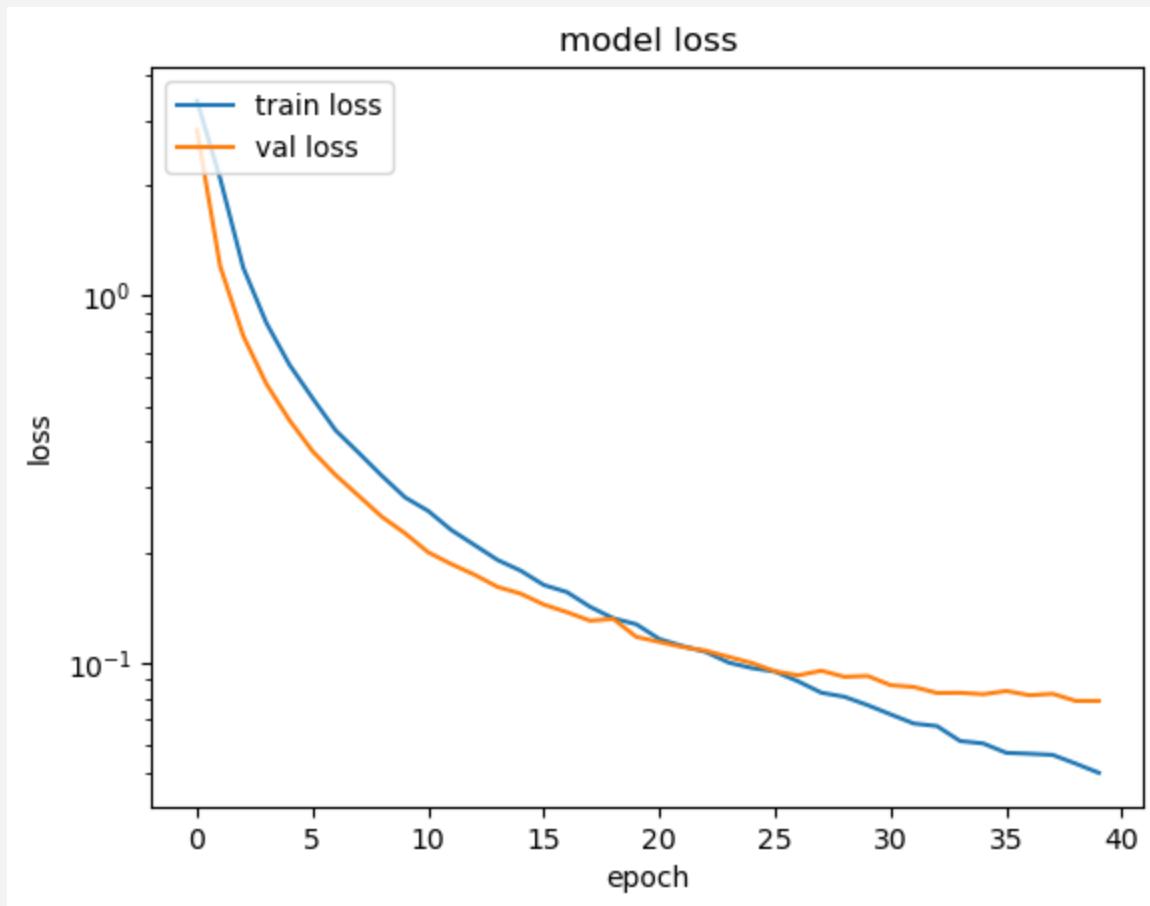




Notes on Model Testing

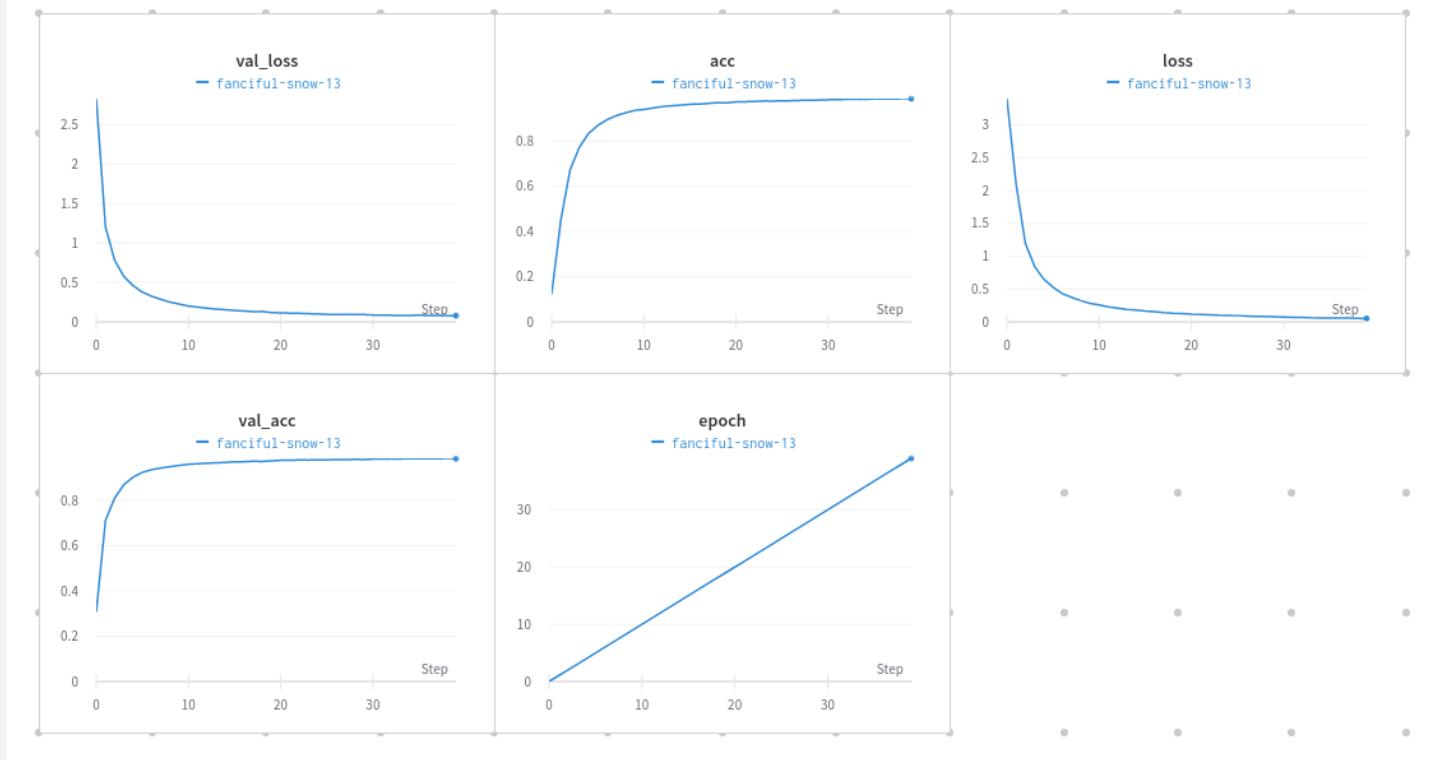
Rather than have a test set, a decision was made to test directly in the simulation. A simulation run will read 100 images and so about 400 characters, the output is displayed to the console via a function to show the most frequent reading for each parking ID. Patterns of confusion in earlier models such as between 'O' and 'Q' were apparent from noting down on a scrap pad plate recognition failures. Only once the final driving agent was built would it make sense to test specific plate images from that specific path to validate the use. ModelC4 trained off this exact type of data, so once it was built the driving agent was mostly finished and the results show no confusion.

For reference to a confusion matrix, please see Lab6 where one was implemented.



Model was retrained, Epoch 40 is selected

▼ Section 1



Final Parking ID Model:

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 18, 23, 16)	160

max_pooling2d (MaxPooling2D (None, 9, 11, 16) 0

)

conv2d_1 (Conv2D) (None, 7, 9, 32) 4640

max_pooling2d_1 (MaxPooling (None, 3, 4, 32) 0

2D)

flatten (Flatten) (None, 384) 0

dropout (Dropout) (None, 384) 0

dense (Dense) (None, 128) 49280

dense_1 (Dense) (None, 8) 1032

=====

Total params: 55,112

Trainable params: 55,112

Non-trainable params: 0

