

Contents

1 Date: September 11th, 2024	2
1.1 Goal: Line Detection From Video	2
1.2 Process	2
1.3 Questions	6
1.4 Random Thoughts	6
2 Date: September 18th, 2024	6
2.1 Goal: ROS robot line following	6
2.2 Process	6
2.3 Questions	12
2.4 Random Thoughts	13
3 Date: October 1st, 2024	13
3.1 Goal: GUI and tracking using SIFT	13
3.2 Process	13
3.3 Questions	20
3.4 Random Thoughts	21
4 Date: October 8th, 2024	21
4.1 Goal: Build a neural network from scratch	21
4.2 Process	21
4.3 Questions	22
5 Date: October 15th, 2024	23
5.1 Goal: Character recognition from images	23
5.2 Process	23
5.3 Questions	30
6 Date: October 29th, 2024	30
6.1 Goal: In sim line following using RL	30
6.2 Process	30
6.3 Questions	31
7 Date: November 5th, 2024	31
7.1 Goal: Cross-entropy inverted pendulum balancer	31
7.2 Process	31
7.3 Questions	34
7.4 Random Thoughts	34

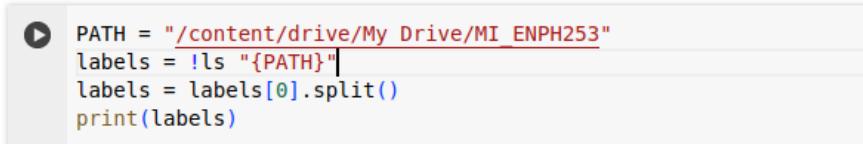
1. Date: September 11th, 2024

1.1. Goal: Line Detection From Video

Figure out Lab 2 as well as what the format of this logbook is going to be. Bear with me while I figure out a style...

1.2. Process

1. I started by trying to follow the Lab 2 Google Colab setup. I got confused when it said “You can download GitHub repos in your Colab instance and run code here.” I’m not sure if that’s useful today.



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

Figure 1: Must update path

2. Then I changed this PATH to .../My Drive/ENPH353 because that’s where my stuff is... Ran the cell, it worked fine.
3. I realized I was being dumb and I need to make a new blank colab...
4. Time to stop and think for a second:
 - (a) First I want to be able to take images from the video they gave us
 - (b) I want to be able to read pixels at a fixed height in the images
 - (c) How do I tell the difference between pixel intensities in OpenCV?
 - (d) Do I need to create a mask?
 - (e) I want to be able to tell the difference between the road and the rest of the video
 - (f) Then I want to find the middle of the road and draw a red dot on it
 - (g) I do that for every single frame and re-export a new video
5. I asked GPT this:

is there a way to screenshot in xubuntu shortcut like windows key shift s in windows

Yes, Xubuntu has a built-in screenshot utility with keyboard shortcuts, similar to the Windows key + Shift + S functionality in Windows.

Here are the common shortcuts for screenshots in Xubuntu:

1. **Print Screen (PrtSc)**: Takes a screenshot of the entire screen and saves it.
2. **Alt + Print Screen**: Takes a screenshot of the currently focused window.
3. **Shift + Print Screen**: Allows you to select an area of the screen to capture (this is similar to Windows' Win + Shift + S).

Figure 2: Secrets to screenshotting unlocked

Now I can screenshot pretty easily.

6. I asked ChatGPT how to upload a file to my Colab. It told me:

```
python
from google.colab import files
uploaded = files.upload()
```

Figure 3: Well that was easy enough

7. I asked ChatGPT how to get frames from the video:

```
python
import cv2
import numpy as np

# Load the video
video_path = 'your_video.mp4' # Replace with your video file
cap = cv2.VideoCapture(video_path)

# Get frame dimensions
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

# Create a video writer to save the output
out = cv2.VideoWriter('output_video.mp4', cv2.VideoWriter_fourcc(*'MP4V'), fps, (fr
```

Figure 4: Extracting still frames from a video

This looked a lil strange because earlier it told me to refer to the video as uploaded and now I'm already specifying the file name... what does this mean? Find out soon...

Browse... raw_video_feed.mp4
raw_video_feed.mp4(video/mp4) - 8560450 bytes, last modified: n/a - 100% done
 Saving raw_video_feed.mp4 to raw_video_feed.mp4

```

import cv2
import numpy as np

# Load the video
video_path = 'raw_video_feed.mp4' # Replace with your video file
  
```

Figure 5: Oh neat you can upload into code blocks in colab

8. I see that the video is called "raw_video_feed.mp4" so I'm guessing that's what I'm supposed to replace it with... Worked.
9. ChatGPT went off when I asked it to code the pipeline for me, so I asked it to explain everything like I'm 5... stay tuned.

```

python

# Doxygen-style comment block
# /**
#  * @brief Process frames to detect the road and mark the center
# *
# * @param frame The current frame of the video
# * @return Processed frame with the center of the road marked
# */

def process_frame(frame):
    # Convert frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Define the height at which to sample pixels (e.g., 3/4th of the frame)
    sample_height = int(0.75 * frame_height)

    # Extract the row of pixels at the defined height
    row_of_pixels = gray_frame[sample_height, :]

    # Threshold the image to differentiate the road (assuming road is darker)
    _, thresh_frame = cv2.threshold(gray_frame, 127, 255, cv2.THRESH_BINARY_INV)

    # Find contours to isolate the road
    contours, _ = cv2.findContours(thresh_frame, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    # Assume the largest contour is the road
    if contours:
        road_contour = max(contours, key=cv2.contourArea)
        # Find the bounding box of the road
        x, y, w, h = cv2.boundingRect(road_contour)
        road_center_x = x + w // 2

        # Draw a red dot at the road's center
        cv2.circle(frame, (road_center_x, sample_height), 5, (0, 0, 255), -1)

    return frame
  
```

Figure 6: Sample implementation for the lab

10. This was a massive output of its process of placing a red dot in the middle of a road... however it's definitely flawed, because it's creating a contour "box" around the road, instead of just reading the value of the bottom... If you try and run this you will get frames that have the red dot way off the path.
11. The next stage is to make it better! I asked it to explain what the box idea was and it actually made sense, I also asked it to show me a sample of a frame being processed and it gave me this:

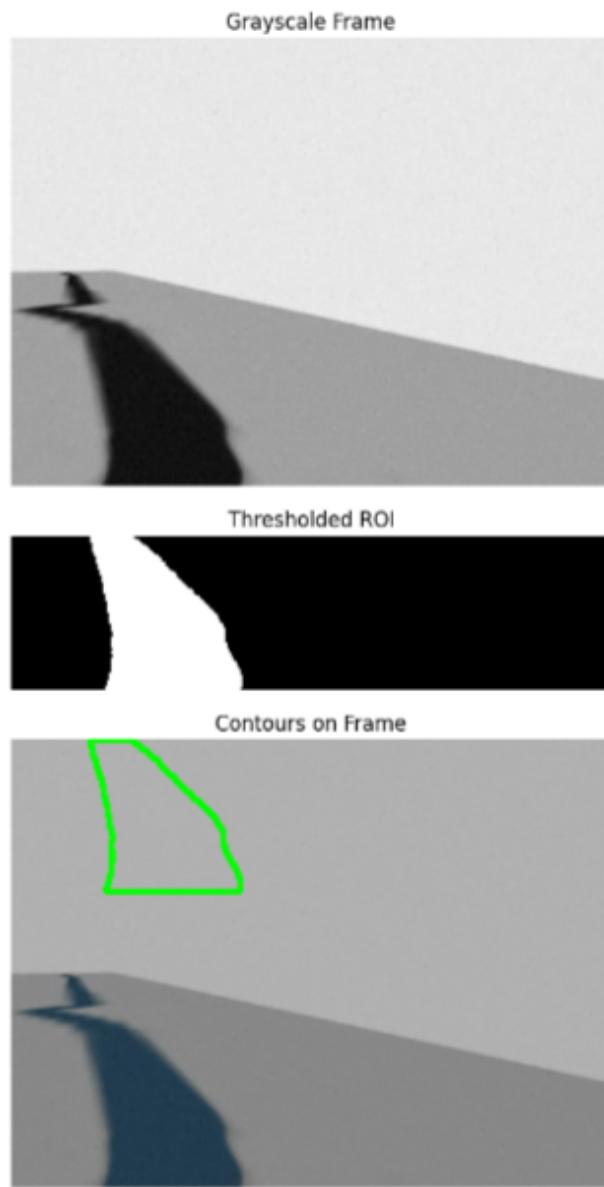


Figure 7: What the...

Now... I know this looks weird. The contour is way too high... I think it's just an issue

with my plotting (not the contour being in the wrong area). I tried for a long time to figure out how to bring it down... didn't succeed.

12. I asked GPT to help me smoothen the red dot, it created a sliding window of center values that updates (to a limit of 3 values) as the video keeps playing.
13. I asked for comments and then I put everything into Colab with some comments and text to explain each stage.
14. Submit!

1.3. Questions

1. How do I get multiple perspectives so I can perform a perspective transform?
2. Why am I using Colab for a project that uses OpenCV?
3. Why isn't my contour plot where it should be?
4. What are other ways to do this?
5. How come my red dot doesn't follow the road when it starts bending weird?

1.4. Random Thoughts

- If you're always trying to say the right things, your whole life is a test [heard this one from Kanye a long time ago].
- Miti says we're at an inflection point in society, I wonder how many times people have said that before about their respective time.
- This is like being thirsty and the only thing you have is a fire hose.

2. Date: September 18th, 2024

2.1. Goal: ROS robot line following

To get a robot simulation run in ROS and to have it line follow a path in the environment.

2.2. Process

1. I realize I never downloaded Python so I'm going to do that now...

```
$ sudo snap install code --classic
```

Found this from the Lab 1 document Miti created.

2. Apparently it was already installed... okay
3. I downloaded an extension called Doxygen Documentation Generator
4. I just made a new directory and launched an instance of ROS

<pre>\$ source ~/.bashrc</pre>	<small>* Rerun the ~/.bashrc script within the current terminal. This will allow the current terminal to load the alias we just added to the ~/.bashrc file.</small>
--------------------------------	--

Figure 8: First time sourcing

5. Lowkey this part makes sense, putting it in the process because it kind of clarifies question 2.
6. I ran step 1.6 and got an output that I didn't really understand so I asked ChatGPT if it understood whether or not the command was successful.



Figure 9: Slowly starting to understand the terminal

7. Cloned Miti's repo into the new directory I made
8. I opened the repo in VS Code and now I'm trying to find how to replace the visual component of the track with a png (`monza.png`)

```

track.material
1 material track{
2   technique
3   {
4     pass
5     {
6       texture_unit
7       [
8         texture monza.png
9       ]
10    }
11  }
12 }

```

Figure 10: Editing the simulation

9. Found it and replaced it.
10. I created a new launch directory, but it broke briefly so I asked GPT why, it was because I didn't have launch flags around my includes. Miti also taught me about ROS security, and how we need to make sure we reference the directory even when we're already in it if we want to run a file.

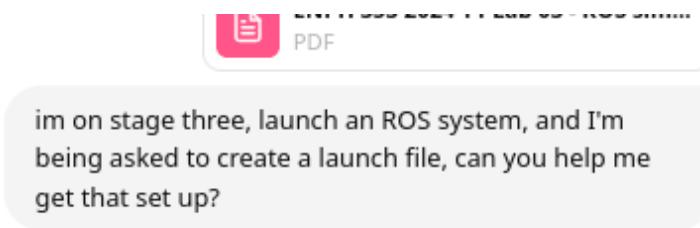


Figure 11: These are the kinds of prompts I used a lot during this class

11. I accidentally zoomed way the hell out... I have no idea how to get back.

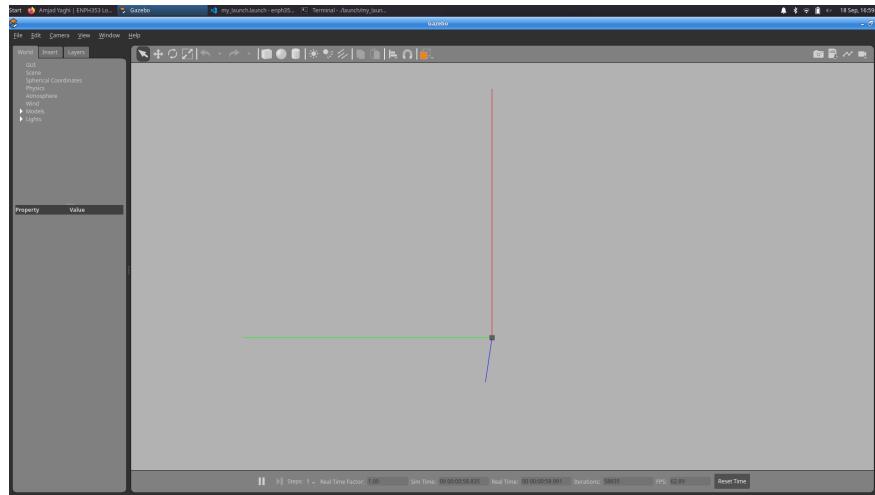


Figure 12: Dear god...

12. NOT HELPFUL!!!!

ok so it worked but now im way zoomed out how do i zoom in on gazebo :sob:



No worries! You can easily zoom in on Gazebo. Here's how you can adjust the view:

1. **Scroll:** Use the **scroll wheel** on your mouse to zoom in and out.
2. **Keyboard Shortcut:**
 - Press **ctrl** + **shift** and use your **mouse scroll wheel** to zoom in and out more smoothly.
3. **Manual View Adjustment:**
 - Hold down the **right mouse button** and **drag** to rotate the camera around.
 - Hold down the **left mouse button** to pan the view.

If you're using a trackpad, the zoom can usually be done by using a **two-finger scroll**.

Let me know if this works! You'll get that zoom level under control in no time



Figure 13: ChatGPT could not save me... I grieve

13. Okay... Check In one complete:

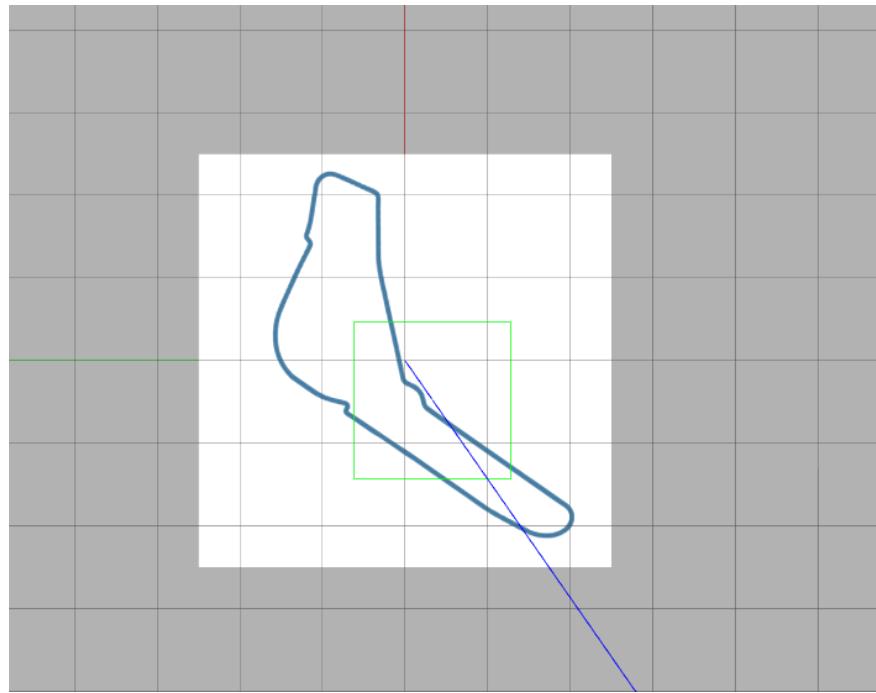


Figure 14: Check point!!!

14. I included the robot launch in the launch file

15. Robot obtained...

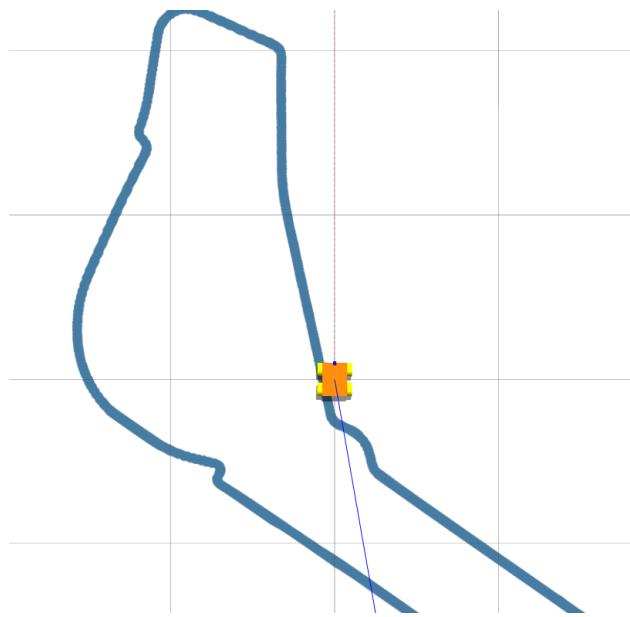
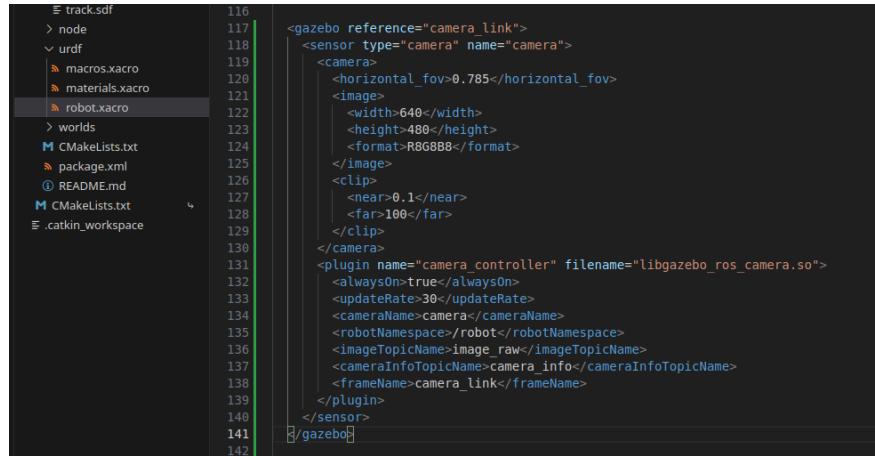


Figure 15: Now you can see the robot on the map

16. I asked GPT to give me a simpler version of the camera plugin found here: [http:](http://)

```
//classic.gazebosim.org/tutorials?tut=ros_gzplugins
```

17. I also pasted the skid steering from the same link into my `robot.xacro`



```
 116 <gazebo reference="camera_link">
 117   <sensor type="camera" name="camera">
 118     <camera>
 119       <horizontal_fov>0.785</horizontal_fov>
 120       <image>
 121         <width>640</width>
 122         <height>480</height>
 123         <format>RGB8B8</format>
 124       </image>
 125       <clip>
 126         <near>0.1</near>
 127         <far>100</far>
 128       </clip>
 129     </camera>
 130   <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
 131     <alwaysOn>true</alwaysOn>
 132     <updateRate>30</updateRate>
 133     <cameraName>camera</cameraName>
 134     <robotNamespace>/robot</robotNamespace>
 135     <imageTopicName>image_raw</imageTopicName>
 136     <cameraInfoTopicName>camera_info</cameraInfoTopicName>
 137     <frameName>camera_link</frameName>
 138   </plugin>
 139 </sensor>
 140 </gazebo>
 141
 142
```

Figure 16: Editing xacro files! We are learning

18. Came across a very strange bug where ros env kept crashing, I just restarted my pc and all was well afterwards... weird
19. I ran `rosrun rqt_image_view rqt_image_view` which unfortunately just gave me this... something is wrong

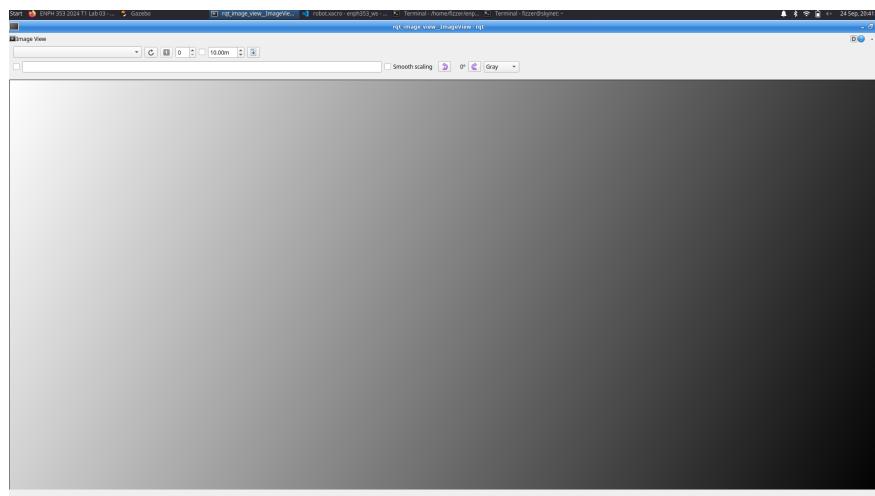


Figure 17: Can't see the simulation space :(

20. I think it has something to do with the simplified camera plug-in I asked GPT to make me so I'm trying the raw one from the same link in step 16.
21. Unfortunately that was not the issue... hmm

22. Great news, it was an issue with the rosrun command given to us, GPT told me to try:

```
rosrun image_view image_view image:=rrbot/camera1/image_raw
```

and it worked.

23. Next up:

```
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"
```

I ran the above in a new terminal with gazebo open and my robot started spinning. Interestingly enough even after terminating the terminal... bro's still spinning.

24. Now I'm following steps GPT is helping me for the implementation of the line following. The first thing it told me to do was nuke my packages... I am listening blindly.
25. Basically all it did was create the python file with the code, but the threshold was way too high
26. Also it created a new launch file called `line_follower.launch`
27. I don't really understand how it created publishers and subscribers yet, will have to do that tomorrow...
28. It basically one shot the whole lab and I just followed instructions and did code review extensively later.

2.3. Questions

1. Why does it matter that I source the ROS workspace?
2. What is going on with the alias section? Why am I appending stuff to the file instead of somewhere else? Is it because the bash file I created is like the base file that has everything we need?
3. Wonder what Mark Twain meant by his “truth is stranger than fiction” quote.

2.4. Random Thoughts

- What if I just take the lab and turn it into step-by-step instructions for me to complete (point and click adventure game pretty much...)
- Miti just said make sure your real time factor is the same on different computers where you are testing integration - pretty sure that'll become very important later
- Seems like I'm not controlling the robot's physics at all which is nice... I was a little scared about that initially, I'm just sending messages using topics.

3. Date: October 1st, 2024

3.1. Goal: GUI and tracking using SIFT

To get a working SIFT algo going.

3.2. Process

1. Opened Qt Designer

2. It begins...

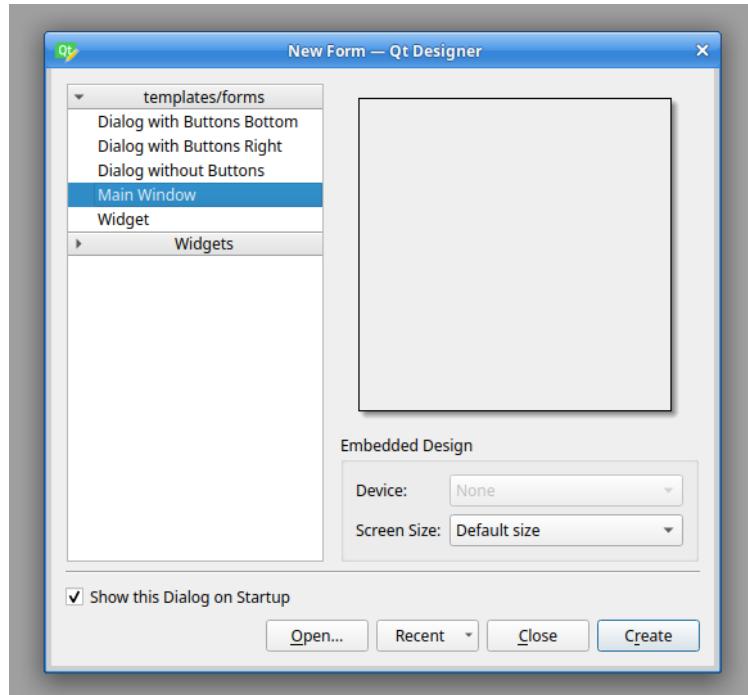


Figure 18: First thing you are greeted with in QT5

3. Add stuff!

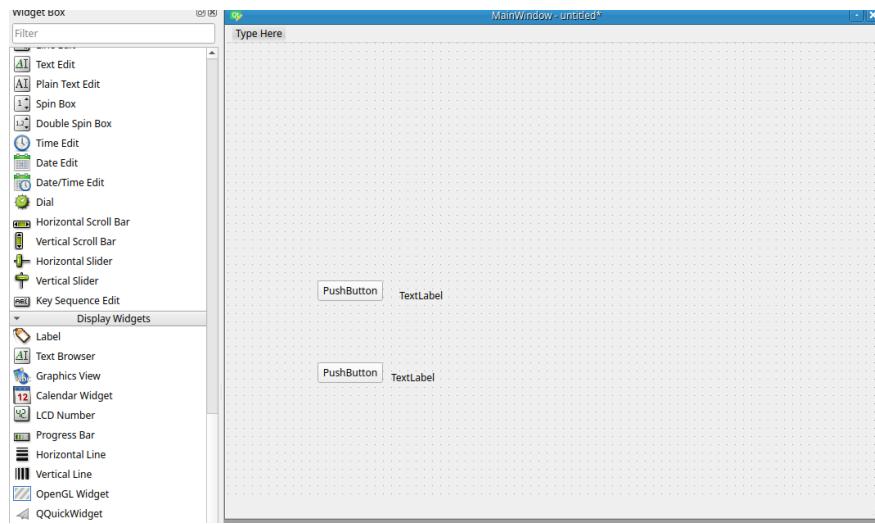


Figure 19: Adding labels and pushbuttons

4. Window tells you what objects you have in your UI

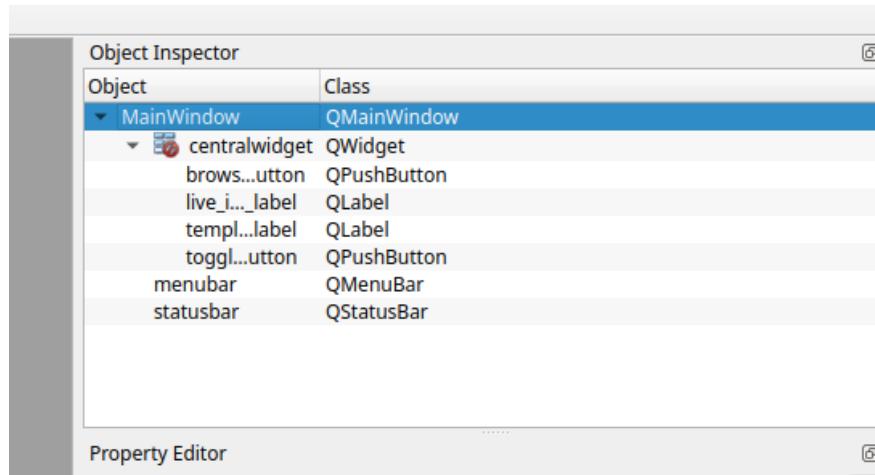


Figure 20: Object Inspector on top right

5. For both labels:

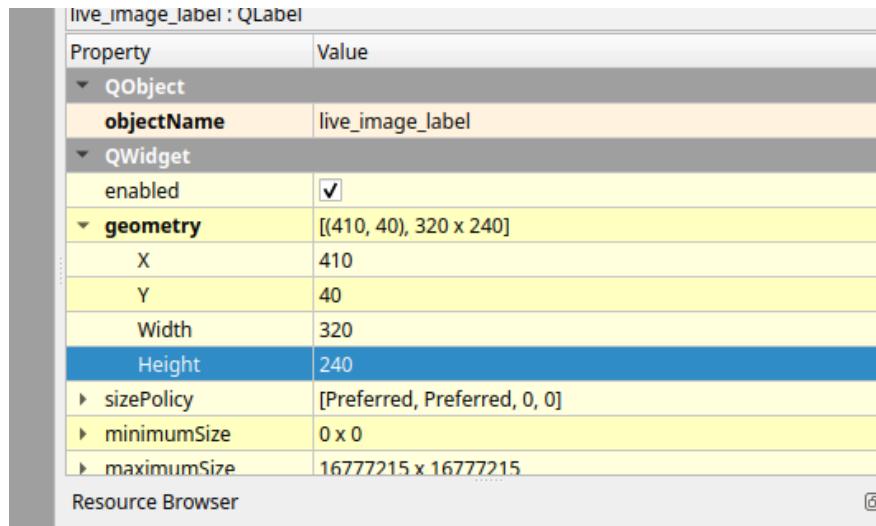


Figure 21: Adjusting size for the labels

6. More adjustments - you can change names too

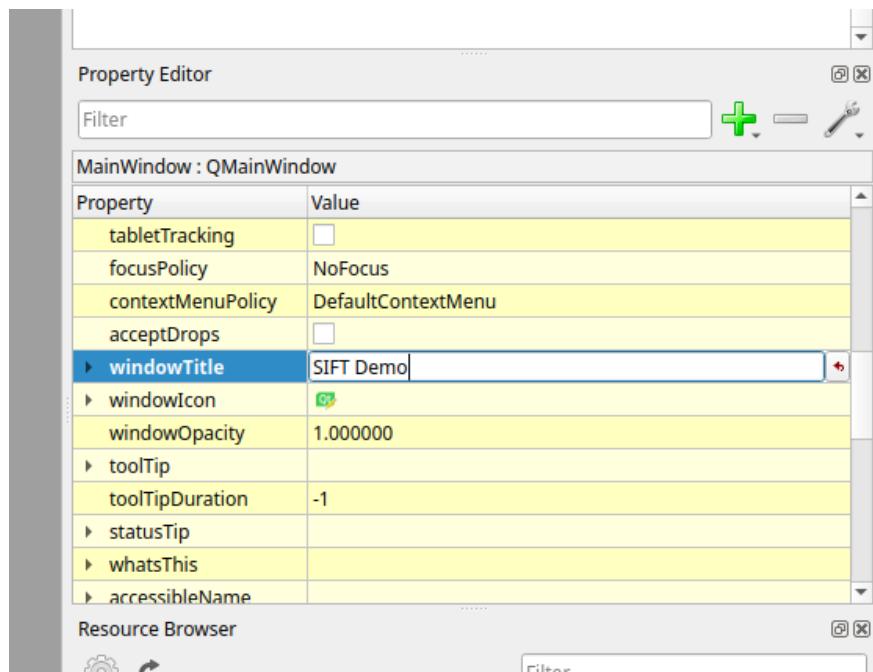


Figure 22: Changing window name to SIFT Demo

7. Make sure your python file is executable!

Command	Notes
\$ cd ~/SIFT_app	Change current directory to the SIFT_app folder.
\$ touch SIFT_app.py	Creates the SIFT_app.py file in the SIFT_app folder.
\$ chmod 700 SIFT_app.py	Changes the permissions for the SIFT_app to be read, write and execute for the owner and none for the group and other users.

Open the "SIFT_app.py" file in your preferred IDE (Visual Studio Code or PyCharm)

Figure 23: Changing permissions using chmod in bash

8. Pasted the code into the python file we were told to make, ran it, got this:

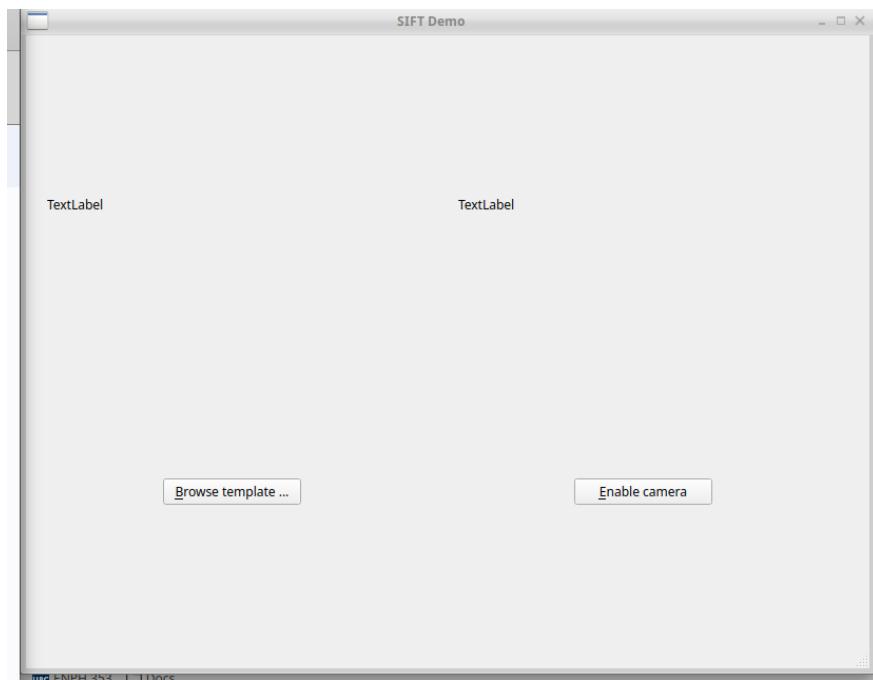


Figure 24: UI pops up when you run the python file

9. Threw this in, tweaked the indentation

```

By adding this connection 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)

```

Figure 25: Provided code for giving buttons functionality

10. Check the camera IDs

8.	To get a live stream showing from our camera we need to first identify what device id for our web camera is. To do this run:
	Command
	\$ sudo apt install v4l-utils

The camera ids are the item order in the list (0 based index). Most often 0 is the right

```

Setting up v4l-utils (1.18.0-2build1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
Processing triggers for man-db (2.9.1-1) ...
fizzer@skynet:~/SIFT_app$ v4l2-ctl --list-devices
Integrated Camera: Integrated C (usb-0000:03:00.3-3):
/dev/video0
fizzer@skynet:~/SIFT_app$ 

```

Figure 26: Two available cameras

11. Watch out for tabs and spaces in python...

12. Pasted the full code given to us in lab 4
13. Taking a break... I have classes now. Will come back and watch videos / use GPT to help me code the SIFT algo.
14. Great I got a basic mapping of keyPoints working, took a little video and posted it on instagram :p
15. I asked ChatGPT to help me draw the links between keypoints
16. Making a new directory where I can get GPT to help out

```

Number of good matches: 2
fizzer@skynet:~/SIFT_app 2024-10-01 19:46:20
$ touch SIFT_app2.py
fizzer@skynet:~/SIFT_app 2024-10-01 19:53:07
$ chmod 700 SIFT_app.py
fizzer@skynet:~/SIFT_app 2024-10-01 19:53:17
$ chmod 700 SIFT_app2.py
fizzer@skynet:~/SIFT_app 2024-10-01 19:53:31

```

Figure 27: Creating a duplicate where I can mess around

- Prompting GPT to help me out with getting the UI-camera integration nice

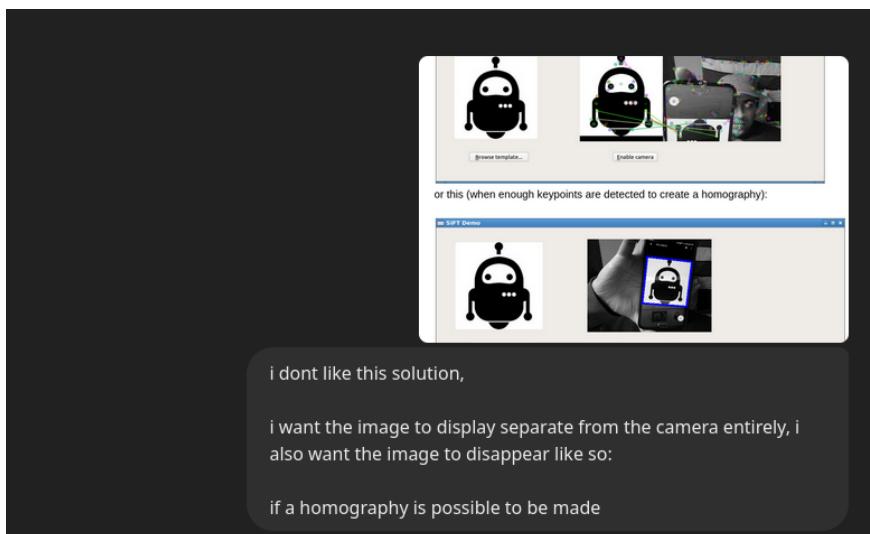


Figure 28: Trying to get help with the layout

- Currently my camera is blocked by the image it's comparing to (lil robot guy not so lil), I asked gpt to help out and also plot the homography
- Didn't work very well, very gross implementation - tried this instead:

Try to keep as much of this original code you can and add the following functionality:

- if enough keypoints are viable to draw a homography, draw it instead of showing the comparison image and the keypoint matches (choose 10 as the threshold)
- if a homography is not possible to draw, display the image we are comparing the camera feed to side by side with the camera (not overlapped)

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

```
from PyQt5 import QtCore, QtGui, QtWidgets  
from python_qt_binding import loadUi
```

Figure 29: The prompt I used to try again

20. This is so funny what the???

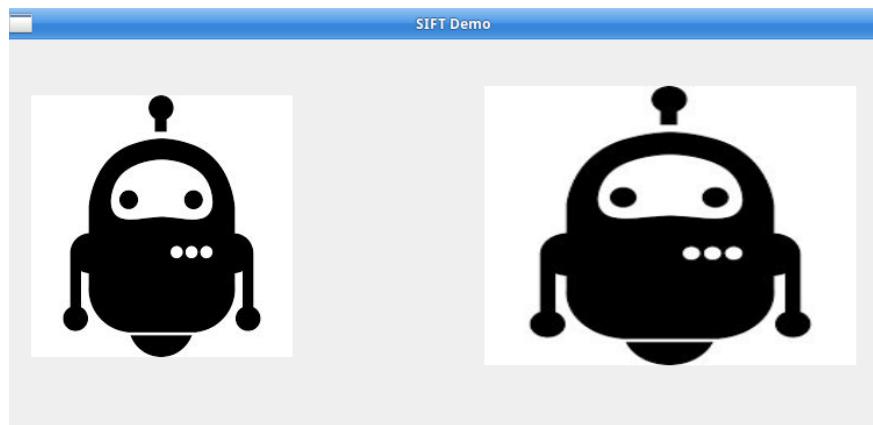


Figure 30: Absolutely failed

21. Im crying of laughter

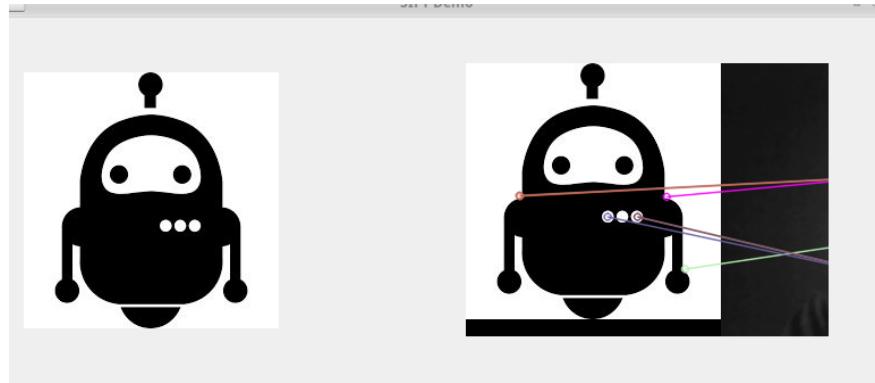


Figure 31: It failed again this is so funny why am I off the screen entirely

22. Its so buggy, can't wait to ask for help tomorrow
23. It works though!!! Submitting this because it meets the requirements but needs slight improvements which I need help with hahaaaa.

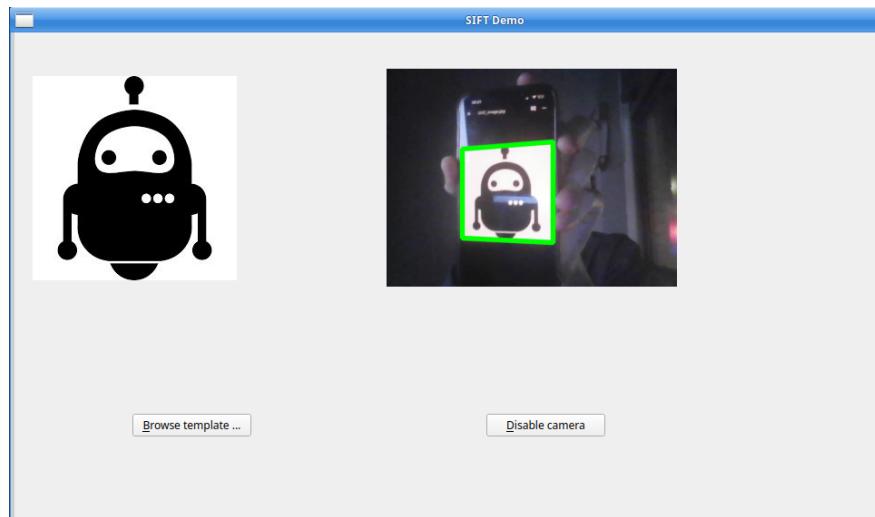


Figure 32: Homography works!

3.3. Questions

1. Why is python so sensitive?
2. Why did the image stretch?
3. How do I make it so the UI allows me to have the keypoints matched without eating up 80% of my camera space?
4. What's with the circles and angles being drawn on the keypoints?

5. How does the homography know how to draw the outline? How would it respond to a triangular or circular image? What about something abstractly shaped?
6. How does my UI know to adjust the label positions when I'm displaying my keypoints (it expands somehow)?

3.4. Random Thoughts

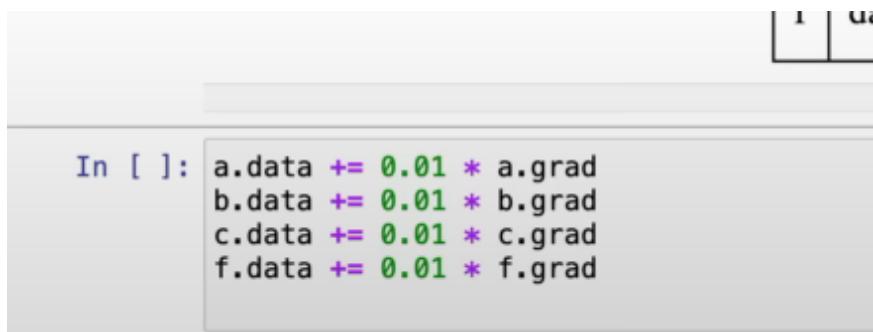
- I find these bugs hilarious for some reason
- Midterm Friday and I decided I was going to ease up on the gas this week... now I have to floor it... who would have thought

4. Date: October 8th, 2024

4.1. Goal: Build a neural network from scratch

4.2. Process

1. I just learned that neural networks are essentially a large set of arbitrary math expressions? Maybe that's not the best way to say it but, I'm trying to apply what I learned from micrograd to neural networks.
2. He just said neural networks and backpropagation can be done with 100 lines of code oh man...



```
In [ ]: a.data += 0.01 * a.grad
b.data += 0.01 * b.grad
c.data += 0.01 * c.grad
f.data += 0.01 * f.grad
```

Figure 33: Datatype and gradients are related

3. Why did he do this? If you increase b in the direction of the gradient you would get a lower value of L overall... he was trying to increase L
4. I can't quite put in words what this is doing

```
out._backward = _backward
```

Figure 34: New python stuff I don't fully understand

5. How is it the accumulation of the gradients that's so beautiful
6. Why didn't he implement radd?
7. I asked gpt to fix my spacing cuz i messed up an indent somewhere (python is brutal)
8. He never implemented radd but you had to...
9. That's amazing you always increase your value with respect to its own gradient if you want to increase your final value
10. Seems like i broke something doing the forward and backward prop he was demonstrating to minimize loss idk what happened am i dividing by 0 or something

```
youd = [n(x) for x in xs]
loss = sum((yout - ygt)**2 for ygt, yout in zip(ys, ypred))
loss

-----
OverflowError                                 Traceback (most recent call last)
<ipython-input-226-5f5d0ee0fe41> in <cell line: 1>()
      1 ypred = [n(x) for x in xs]
      2 loss = sum((yout - ygt)**2 for ygt, yout in zip(ys, ypred))
      3 loss
      4
      5 frames
<ipython-input-189-5547be2b5849> in tanh(self)
      61     def tanh(self):
      62         x = self.data
--> 63         t = (math.exp(2 * x) - 1) / (math.exp(2 * x) + 1)
      64         out = Value(t, (self,), 'tanh')
      65
overflowError: math range error
```

Figure 35: Strange error, trying to figure it out...

4.3. Questions

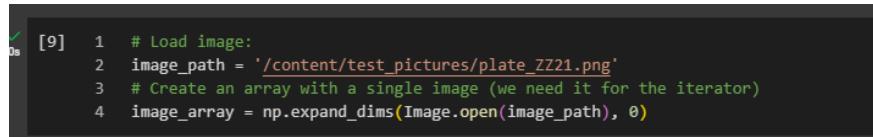
1. What is cross entropy loss and why does it sound so cool?
2. What actually was pytorch... i feel like i just threw it in
3. What happened at the very very end of the video?
4. Is the way I've commented illegal?

5. Date: October 15th, 2024

5.1. Goal: Character recognition from images

5.2. Process

1. Download and make a copy of the license plate generator
2. Asked gpt to help explain what the code we were given does
3. Basically it just draws nice license plates onto the background we downloaded
4. Two characters two numbers
5. for some reason the template given to us was referencing some random path that didn't exist so I changed it to pick out the ZZ21 license plate given to us



```
[9] 1 # Load image:
2 image_path = '/content/test_pictures/plate_ZZ21.png'
3 # Create an array with a single image (we need it for the iterator)
4 image_array = np.expand_dims(Image.open(image_path), 0)
```

Figure 36: I think I am misunderstanding what this template is about but it's okay let's continue

6. Oh im a fool... im not supposed to use the test image set... oops

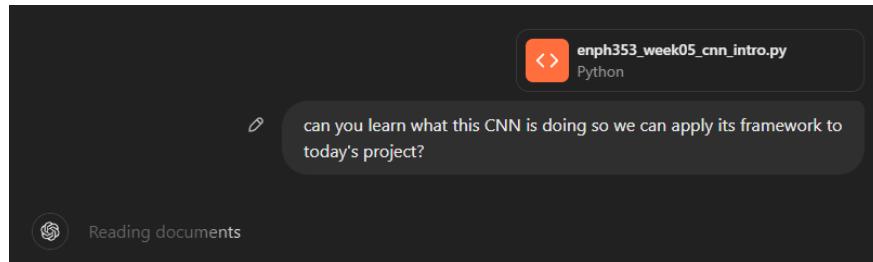
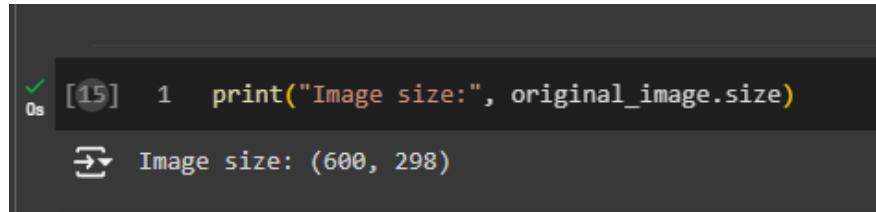


Figure 37: Asking GPT to help me map an existing CNN to one that I wil have to build myself

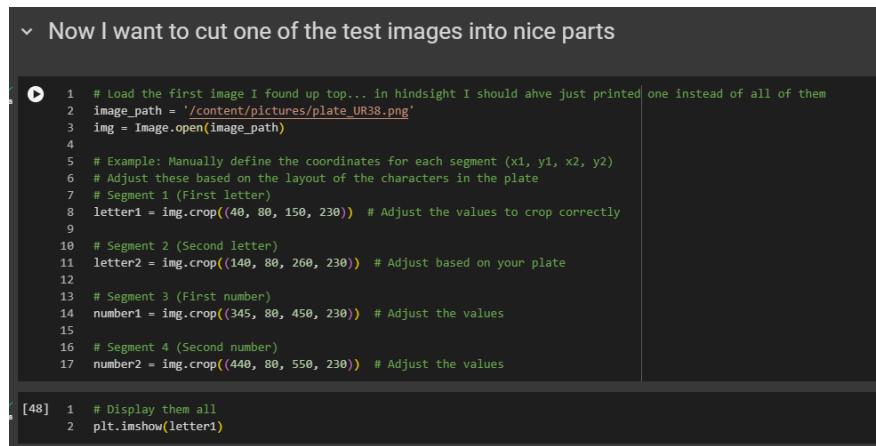
7. I asked gpt to learn what the cats vs dogs notebook was doing so that I can try to get it to help me do the same thing for today's assignment
8. I got the size of my images so that I could see what I need to crop



```
[15] 1 print("Image size:", original_image.size)
[15] 2
[15] 3 Image size: (600, 298)
```

Figure 38: Image sizes printed on the bottom

9. Next step was to list some of the test images I generated so that I could get the cuts for each letter done right



```
Now I want to cut one of the test images into nice parts

1 # Load the first image I found up top... in hindsight I should have just printed one instead of all of them
2 image_path = '/content/pictures/plate_UR38.png'
3 img = Image.open(image_path)
4
5 # Example: Manually define the coordinates for each segment (x1, y1, x2, y2)
6 # Adjust these based on the layout of the characters in the plate
7 # Segment 1 (first letter)
8 letter1 = img.crop((40, 80, 150, 230)) # Adjust the values to crop correctly
9
10 # Segment 2 (Second letter)
11 letter2 = img.crop((140, 80, 260, 230)) # Adjust based on your plate
12
13 # Segment 3 (First number)
14 number1 = img.crop((345, 80, 450, 230)) # Adjust the values
15
16 # Segment 4 (Second number)
17 number2 = img.crop((440, 80, 550, 230)) # Adjust the values

[48] 1 # Display them all
[48] 2 plt.imshow(letter1)
```

Figure 39: Completed hard-coded crop for license plate

10. These were the cutoff values that seemed to work best for the UR38 license plate... kinda hoping it will work nicely for others, I can check real quick
11. (Also I used GPT to help me figure out how to crop images.)
12. It didn't look perfect for another, so i made a minor adjustment
13. I love this, makes understanding this lab very easy... next step is to prepare the data as chat GPT says

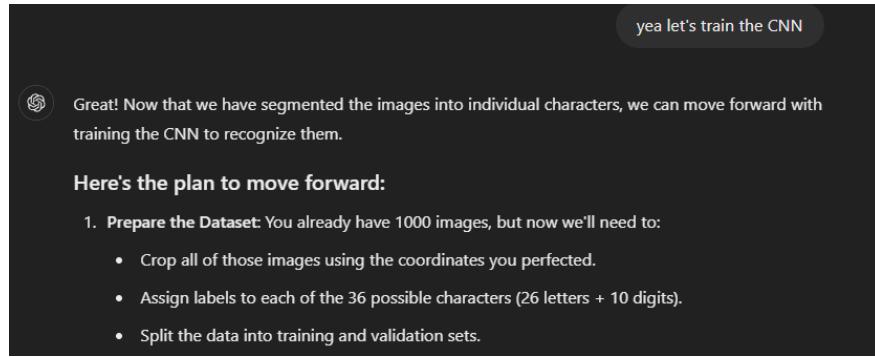


Figure 40: Working side by side with guidance from cgpt

14. soooo elegant, this makes sense

```
# Set up directories
image_dir = '/content/pictures/'
cropped_dir = '/content/cropped_pictures/'

# Make a directory for the cropped images
if not os.path.exists(cropped_dir):
    os.makedirs(cropped_dir)

# List all images in the pictures directory
image_files = os.listdir(image_dir)

# Crop each image
for image_file in image_files:
    # Load the image
    image_path = os.path.join(image_dir, image_file)
    img = Image.open(image_path)

    # Manually define the coordinates for each segment
    letter1 = img.crop((40, 80, 145, 230)) # First letter
    letter2 = img.crop((140, 80, 260, 230)) # Second letter
    number1 = img.crop((345, 80, 450, 230)) # First number
    number2 = img.crop((440, 80, 550, 230)) # Second number

    # Save the cropped images
    base_filename = os.path.splitext(image_file)[0]
    letter1.save(os.path.join(cropped_dir, f'{base_filename}_letter1.png'))
    letter2.save(os.path.join(cropped_dir, f'{base_filename}_letter2.png'))
    number1.save(os.path.join(cropped_dir, f'{base_filename}_number1.png'))
    number2.save(os.path.join(cropped_dir, f'{base_filename}_number2.png'))

    print("Cropped images saved to:", cropped_dir)
```

Figure 41: How to organize each image into a corresponding directory

15. I asked GPT to explain why we need to label our letters and numbers numerically

does the CNN need the labels to be numerical? why

Yes, the CNN needs the labels to be **numerical** because neural networks operate on numbers—they can't work with text or categorical data directly. Let's dive into **why** numerical labels are necessary:

1. Mathematical Computations:

- Neural networks use a series of mathematical operations (like matrix multiplications, activation functions, etc.). These operations require that both the inputs (images) and the outputs (labels) be in **numerical form**.
- For the CNN to compute **error** during training and to adjust the **weights** through backpropagation, it needs to compare the model's predictions (which are also **numerical**) against the numerical labels.

2. One-Hot Encoding for Classification:

In your case, the CNN is trying to classify the characters into one of **36 possible classes** (26 letters + 10 digits). The standard way to handle classification problems in neural networks is to:

1. Convert the labels to numbers.
2. One-hot encode those numerical labels so that the network knows which class is the correct one.

Figure 42: Prompting GPT to explain labels to me

16. it told me to normalize my data for faster compute, but I imagine this step is highly unnecessary since it will be so fast anyway...

```

3 # Process each image and assign labels
4 for cropped_file in cropped_files:
5     # Load the image
6     image_path = os.path.join(cropped_dir, cropped_file)
7     image = cv2.imread(image_path)
8
9     # Normalize the pixel values to [0, 1] (Optional)
0     image = image / 255.0

```

Figure 43: Optionally, divide by 255 to normalize pixel values for faster CNN processing

17. I have an issue where I think I polluted my training set so I'm refreshing the colab to make sure

18. I think I made a mistake refreshing... its not connecting to colab anymore :/

19. My images aren't all the same size so I'm going to crop them a little better

```

1. Letter 1:
• Width = 145 - 40 = 105
• Height = 230 - 80 = 150
• Size: 105x150

2. Letter 2:
• Width = 260 - 140 = 120
• Height = 230 - 80 = 150
• Size: 120x150

3. Number 1:
• Width = 450 - 345 = 105
• Height = 230 - 80 = 150
• Size: 105x150

4. Number 2:
• Width = 550 - 440 = 110
• Height = 230 - 80 = 150
• Size: 110x150

```

Figure 44: Forgot that images need to be same size for CNN processing

20. Massive set-back but eventually i changed the values to something that made sense

```



```

Figure 45: Fixed letter width and height

21. Check out how much data I generated

```

54 print(f"Labels shape: {labels.shape}")
55
→ Data shape: (3968, 150, 110, 3)
Labels shape: (3968, 36)

```

1 Start coding or generate with AI.

Figure 46: Around 4000 images ready to train the CNN!

What Do These Numbers Mean?

1. Data shape: (3968, 150, 110, 3):

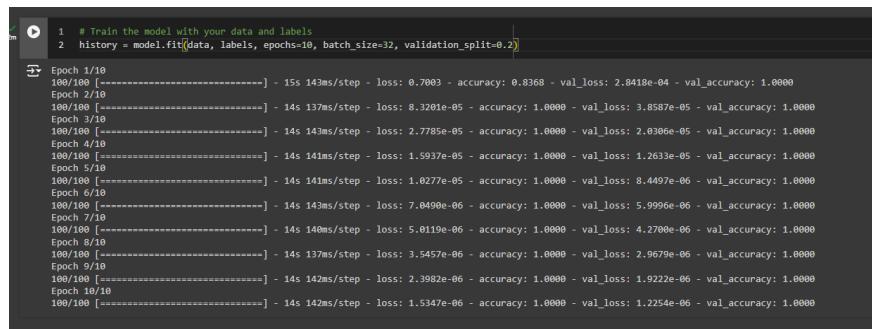
- **3968:** This is the total number of images you have (3968 individual cropped segments).
- **150:** This is the **height** of each image (150 pixels).
- **110:** This is the **width** of each image (110 pixels).
- **3:** This represents the **color channels** (RGB), meaning each image has 3 channels (red, green, and blue).

2. Labels shape: (3968, 36):

- **3968:** This matches the number of images, indicating that each image has a corresponding label.
- **36:** This represents the 36 possible classes (26 letters + 10 digits). The labels are **one-hot encoded**, so each label is a vector of size 36 with a **1** at the index of the correct class and **0**s elsewhere.

Figure 47: Asking for help to understand all the numbers that .shape prints out

22. Oh man I'm cooked for this part, I asked for help with setting up the CNN architecture and I am highly lost
23. Taking a walk while the model trains



```
1 # Train the model with your data and labels
2 history = model.fit(data, labels, epochs=10, batch_size=32, validation_split=0.2)

Epoch 1/10
100/100 [=====] - 15s 143ms/step - loss: 0.7003 - accuracy: 0.8368 - val_loss: 2.8418e-04 - val_accuracy: 1.0000
Epoch 2/10
100/100 [=====] - 14s 137ms/step - loss: 8.3720e-05 - accuracy: 1.0000 - val_loss: 3.8587e-05 - val_accuracy: 1.0000
Epoch 3/10
100/100 [=====] - 14s 137ms/step - loss: 2.7785e-05 - accuracy: 1.0000 - val_loss: 2.0306e-05 - val_accuracy: 1.0000
Epoch 4/10
100/100 [=====] - 14s 143ms/step - loss: 1.5937e-05 - accuracy: 1.0000 - val_loss: 1.2633e-05 - val_accuracy: 1.0000
Epoch 5/10
100/100 [=====] - 14s 141ms/step - loss: 1.0277e-05 - accuracy: 1.0000 - val_loss: 8.4497e-06 - val_accuracy: 1.0000
Epoch 6/10
100/100 [=====] - 14s 141ms/step - loss: 7.8490e-06 - accuracy: 1.0000 - val_loss: 5.9996e-06 - val_accuracy: 1.0000
Epoch 7/10
100/100 [=====] - 14s 140ms/step - loss: 5.0119e-06 - accuracy: 1.0000 - val_loss: 4.2700e-06 - val_accuracy: 1.0000
Epoch 8/10
100/100 [=====] - 14s 137ms/step - loss: 3.5457e-06 - accuracy: 1.0000 - val_loss: 2.0679e-06 - val_accuracy: 1.0000
Epoch 9/10
100/100 [=====] - 14s 142ms/step - loss: 2.3982e-06 - accuracy: 1.0000 - val_loss: 1.9222e-06 - val_accuracy: 1.0000
Epoch 10/10
100/100 [=====] - 14s 142ms/step - loss: 1.5347e-06 - accuracy: 1.0000 - val_loss: 1.2254e-06 - val_accuracy: 1.0000
```

Figure 48: Training the model!

24. These values must be fantastic... surely that loss is amazing, let's see some graphs

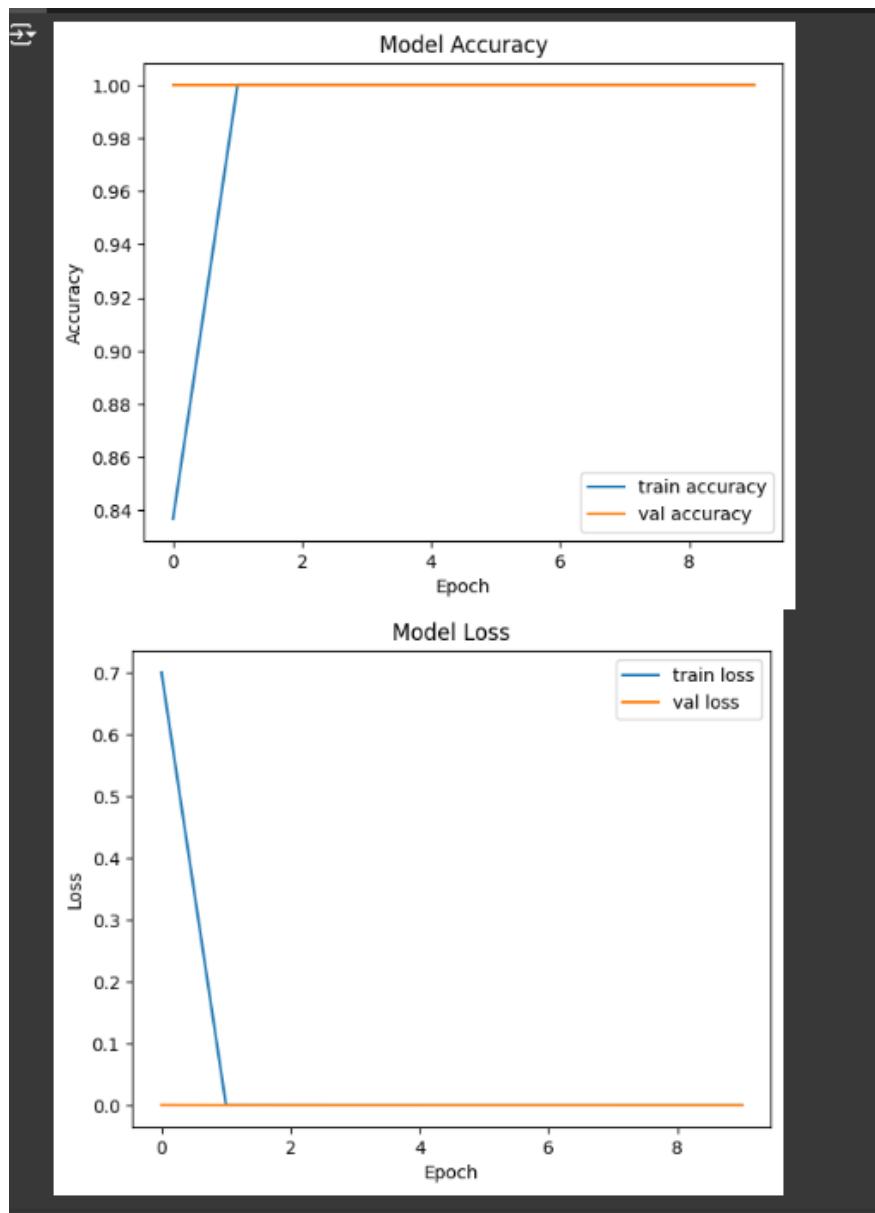


Figure 49: Definitely overfitting the training set

25. HAHAHAAAAAA something is critically wrong
26. Instead of debugging right away I'm going to continue with the lab and work backwards later
27. I asked GPT for help with plotting the losses and confusion matrix
28. Now it's time to debug

5.3. Questions

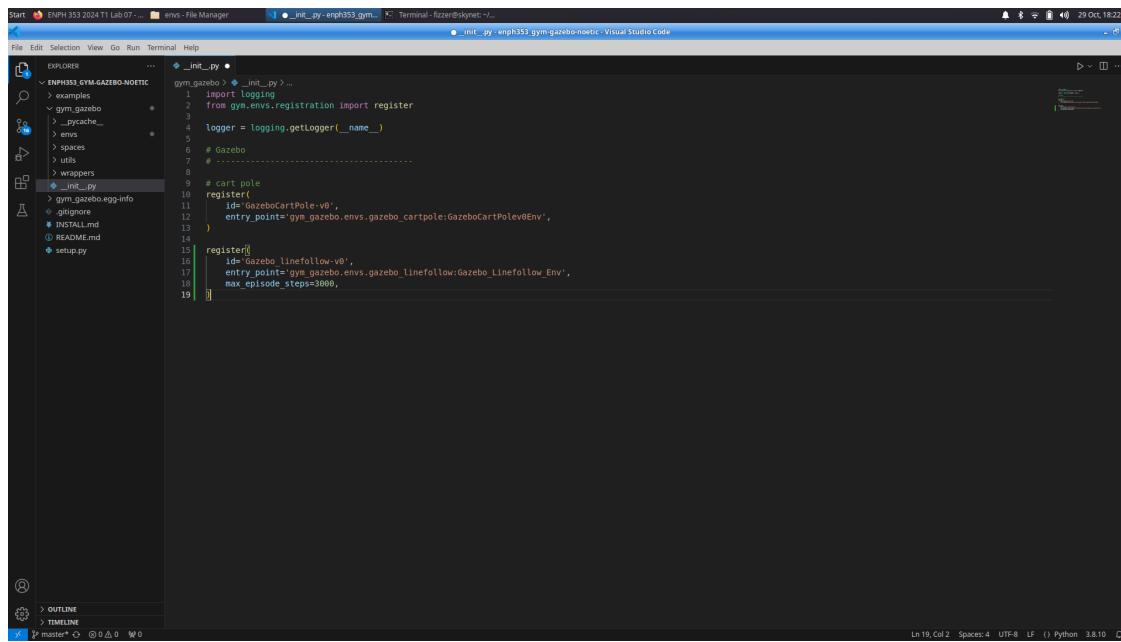
1. Why is my data overfitting so fast?
2. How many epochs should I choose?
3. Why are my val loss and val accuracy immediately set at 1 and 0 respectively?
4. Why does my confusion matrix only have diagonal values?
5. What does density mean in the context of the CNN architecture?
6. What does pooling mean in the context of the CNN architecture?

6. Date: October 29th, 2024

6.1. Goal: In sim line following using RL

To implement a Q-Learning model for the line following robot we implemented earlier.

6.2. Process



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure: ENPH353.gym-gazebo-noetic, containing examples, gym_gazebo, gym_gazebo_noetic, and wrappers. The __init__.py file is selected.
- Code Editor:** Displays the content of __init__.py:

```
gym_gazebo > __init__.py -->
1 import Logging
2 from gym.envs.registration import register
3
4 logger = logging.getLogger(__name__)
5
6 # Gazebo
7 # -----
8
9 # cart pole
10 registered = {}
11 for idc in [GazeboCartPoleEnv,
12             entry_point='gym_gazebo.envs.gazebo_cartpole:GazeboCartPoleEnv'],
13             ]:
14     register(
15         idc=GazeboCartPoleEnv,
16         entry_point='gym_gazebo.envs.gazebo_linefollow:Gazebo_Linefollow_Env',
17         max_episode_steps=3000,
18     )
19 ]
```
- Terminal:** Shows the command "init_py - enph353 gym_gazebo-noetic" and the date "29 Oct, 18:22:54".
- Status Bar:** Shows "Ln 19, Col 2" and "Python 3.8.10".

Figure 50: Must register the line following in the init.py file!

1. cool part

6.3. Questions

1. Why do I need to register the line following?

7. Date: November 5th, 2024

7.1. Goal: Cross-entropy inverted pendulum balancer

Implement a controller that mixes reinforcement learning and deep learning to control an unstable robot.

7.2. Process

1. Alrighty... Let's download the folder given to us.
2. Now I'm going to ask GPT to help me decode all the steps in excruciating detail
3. Noticing PERCENTILE is set to 30... seems awfully low?
4. Learned that we have a hidden amount of layers (I wonder why we call them hidden)...
5. Learned that we have an input space of obs_size, and an output of n_actions.
6. Learned how each episode is stored as a tuple with steps and reward as the parameters.
7. Learned that we're creating another tuple which stores the observation and action taken for any particular step.
8. Noticing iterate_batches is a little confusing.
9. Filter_batch is pretty intuitive, I think I could write it from scratch.
10. Now we've reached the main training loop
11. GPT tells me to set the obs_size and n_actions.

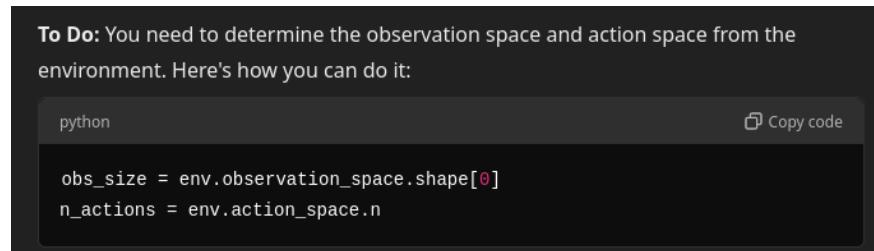


Figure 51: GPT trying to pull me out the depths of confusion

12. Learned about Adam optimizer for changing weights in the CNN.

13. Asked GPT to teach me about the training loop

- **Training Loop:**
 - **Iterate Over Batches:** For each batch of episodes generated by `iterate_batches`, perform the following steps:
 1. **Filter Elite Episodes:**
 - **To Do:** Use the `filter_batch` function to select the top percentile of episodes.
 2. **Zero Gradients:**
 - **To Do:** Before performing backpropagation, clear existing gradients to prevent accumulation.
 3. **Forward Pass:**
 - **To Do:** Pass the elite observations through the network to get action scores.
 4. **Calculate Loss:**
 - **To Do:** Compute the loss between the predicted action scores and the actual actions taken.
 5. **Backward Pass and Optimization:**
 - **To Do:** Perform backpropagation to compute gradients and update the network's weights.

Figure 52: GPT explanation of how the training loop kinda works (and what to do)

- 4. **Calculate Loss:**
 - **To Do:** Compute the loss between the predicted action scores and the actual actions taken.
- 5. **Backward Pass and Optimization:**
 - **To Do:** Perform backpropagation to compute gradients and update the network's weights.

Figure 53: Explanation continued...

14. OK time to test if it runs:

4) Once you are done open a command line terminal and run the following commands to start training your agent:

```
source ~/enph353_gym-gazebo-noetic/gym_gazebo/envs/ros_ws/devel/setup.bash  
cd ~/enph353_gym-gazebo-noetic/examples/gazebo_cartpole  
python3 ./gazebo_cartpole_v0_xentropy.py
```

In another command line terminal start a tensorboard server by running

```
cd ~/enph353_gym-gazebo-noetic/examples/gazebo_cartpole  
tensorboard --logdir runs
```

then open Firefox and navigate to: <http://localhost:6006> to see the training dashboard for the agent.

Figure 54: Class-provided instructions on how to run the simulation and monitor the training

15. All set.

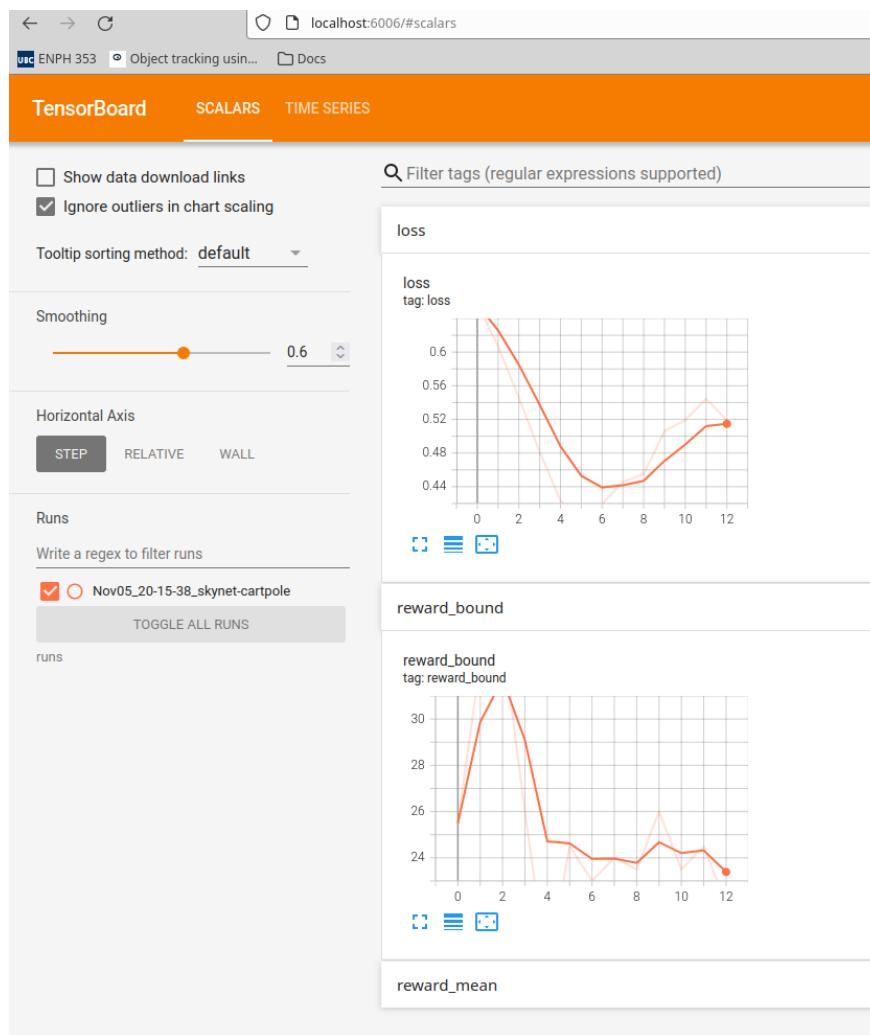


Figure 55: Nice, it's training!

7.3. Questions

1. Why did the lab say that usually we allow the 90th percentile and above to pass through, but in the actual implementation the 70th and above was used?
2. What does it mean to “Convert to tensor”? You know... on that topic, what are tensor cores?
3. Who made the objective function? How is it calculating loss?

7.4. Random Thoughts

- Wait, maybe this is dumb but what is the difference between a step and an action...? I'm sure it'll make more sense in a minute.
- It's not really easy to understand what iterate_batches does, even with help from GPT. I mean on a very high level, it iterates through episodes until a batch is done and then moves on... but there's a lot of meat here.

UNDATED LOGBOOK I USED FOR FINAL PROJECT

▶ sizePolicy	[Preferred, Preferred, 0, 0]
▼ minimumSize	640 x 480
Width	640
Height	480
▶ maximumSize	16777215 x 16777215

label : QLabel	
Property	Value
QLabel	<input checked="" type="checkbox"/> scaledContents

The screenshot shows a terminal window with the following command history:

```
fuzzer@skynet:~/ros_ws$ roscore
fuzzer@skynet:~/ros_ws$ roslaunch robbie_xacro robbie.xacro
[robbie_xacro-1] started with pid [1869]
[robbie_xacro-1] process has died! shutting down ros master
```

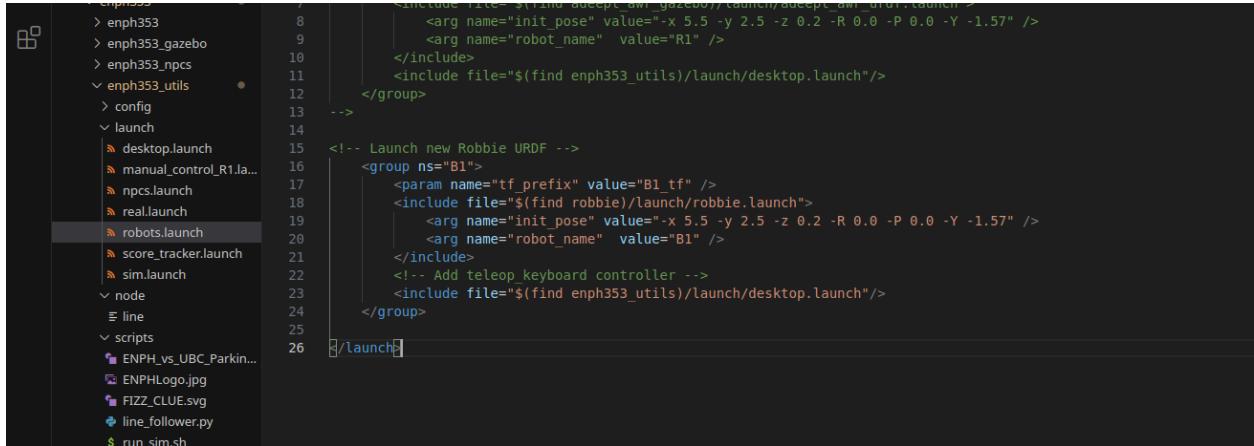
Obviously the wheel diameter is wrong 😞...

```
<rightRearJoint>wheel_RR_joint</rightRearJoint>
<wheelSeparation>0.23</wheelSeparation>
<wheelDiameter>0.08</wheelDiameter>
<robotBaseFrame>base_link</robotBaseFrame>
<torque>20</torque>
<topicName>cmd_vel</topicName>
<broadcastTF>false</broadcastTF>
<plugin>
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

OK SO I RESTARTED

Bro suggested i make a github



```
<include file="$(find accept_urdf_gazebo)/launch/accept_urdf.launch">
    <arg name="init_pose" value="-x 5.5 -y 2.5 -z 0.2 -R 0.0 -P 0.0 -Y -1.57" />
    <arg name="robot_name" value="R1" />
</include>
<include file="$(find enph353_utils)/launch/desktop.launch"/>
```

```
</group>
-->
<!-- Launch new Robbie URDF -->
<group ns="B1">
    <param name="tf_prefix" value="B1_tf" />
    <include file="$(find robbie)/launch/robbie.launch">
        <arg name="init_pose" value="-x 5.5 -y 2.5 -z 0.2 -R 0.0 -P 0.0 -Y -1.57" />
        <arg name="robot_name" value="B1" />
    </include>
    <!-- Add teleop_keyboard controller -->
    <include file="$(find enph353_utils)/launch/desktop.launch"/>
</group>
```

```
</launch>
```

<https://chatgpt.com/c/673f8a0e-cad0-800e-8832-a0466aa4d5b5>

Crucial convo with GPT for future reference

3. Check CMakeLists.txt

Ensure your `controller` package's `CMakeLists.txt` file includes the necessary Qt5 components. Specifically, verify that the following lines are present:

```
cmake
Copy code

find_package(Qt5 REQUIRED COMPONENTS Widgets)

# Include Qt5 Widgets
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)
set(CMAKE_AUTOUIC ON)

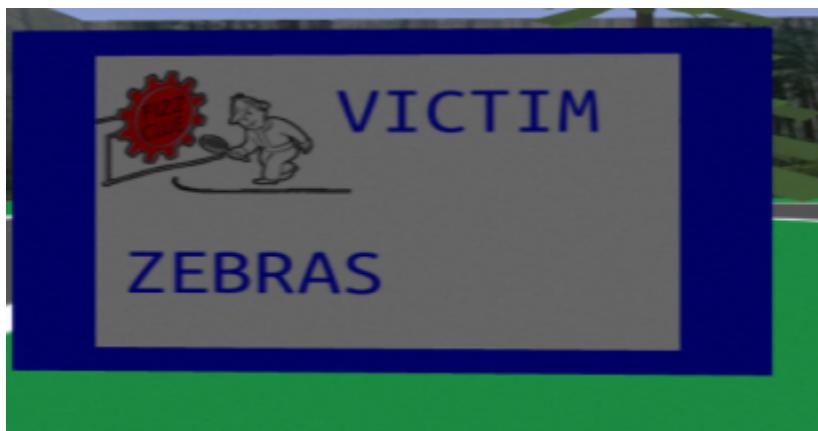
include_directories(${Qt5Widgets_INCLUDE_DIRS})
add_definitions(${Qt5Widgets_DEFINITIONS})

# Add your executable
add_executable(controller_node src/controller_node.cpp)
target_link_libraries(controller_node ${catkin_LIBRARIES} Qt5::Widgets)
```

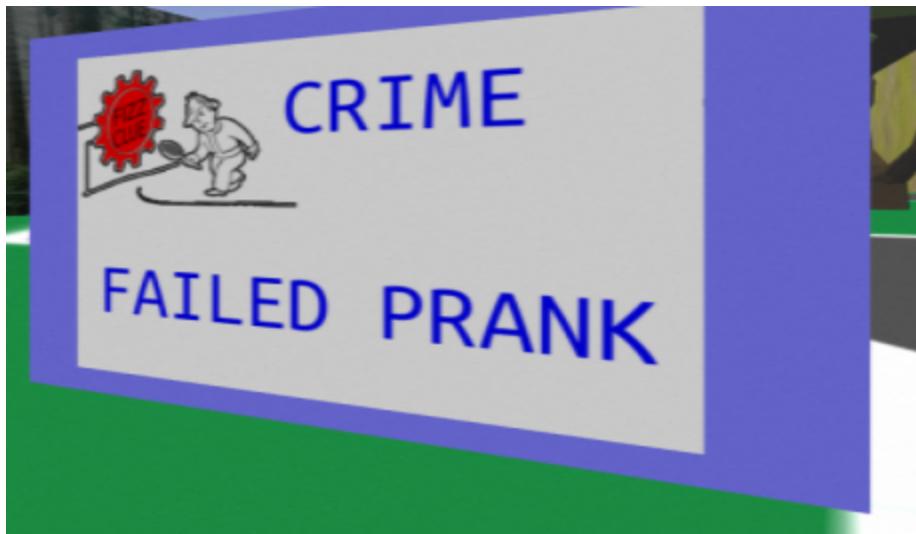
Aight so the cmake and package xml files needed to change for the UI to work for some
I dont really understand what those do but its chill...

UNDATED LOGBOOK I USED FOR FINAL PROJECT

Now i got my wasd working no problem but i want it to be a little better, including make it so you can press the wasd keys on my keyboard to make it work, and also i need them to only work while pressed, rather than toggle on press...



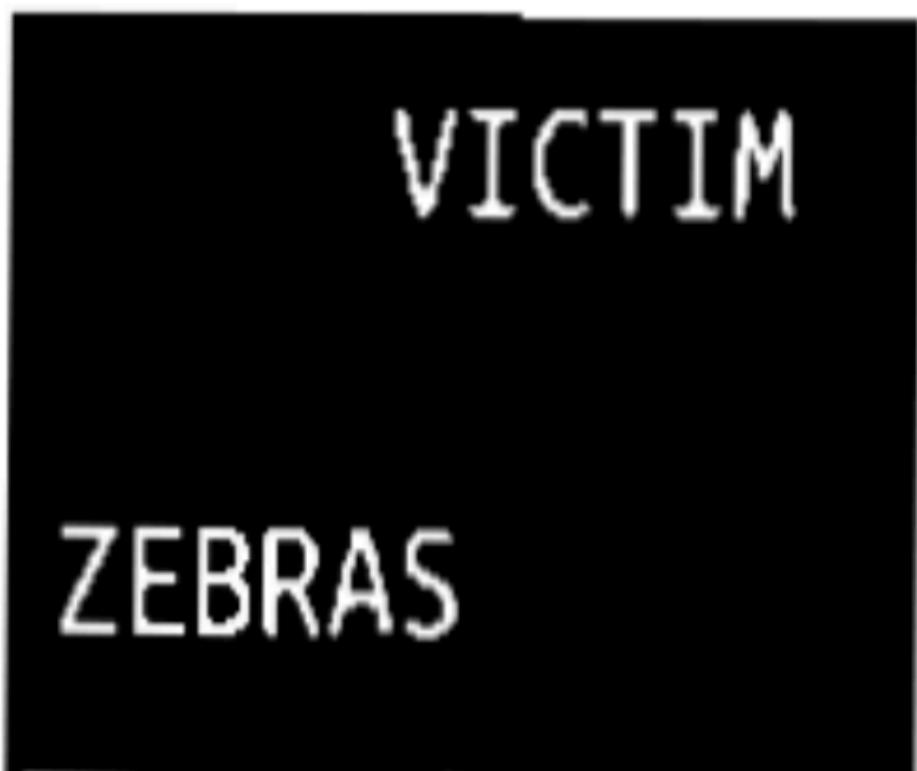
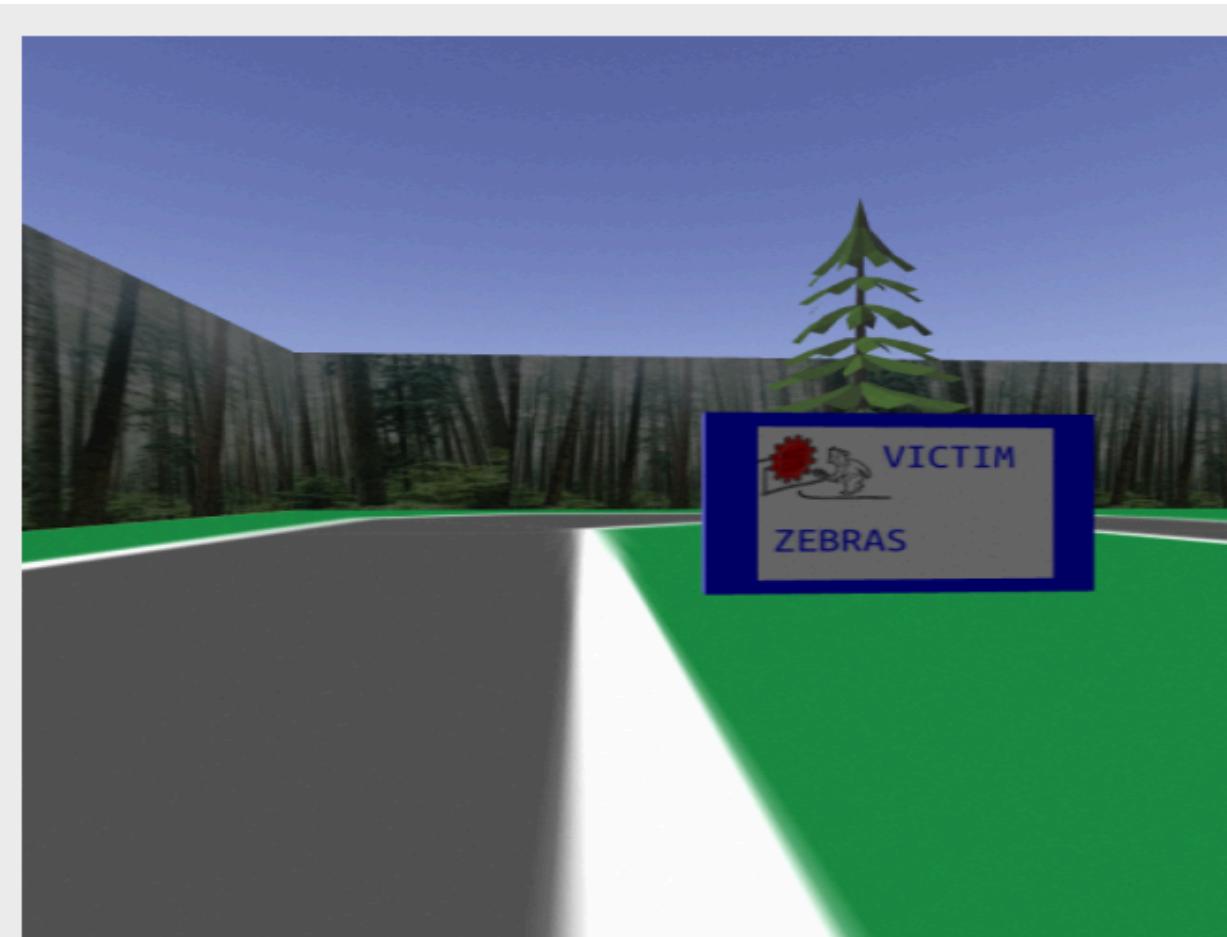
UNDATED LOGBOOK I USED FOR FINAL PROJECT



UNDATED LOGBOOK I USED FOR FINAL PROJECT



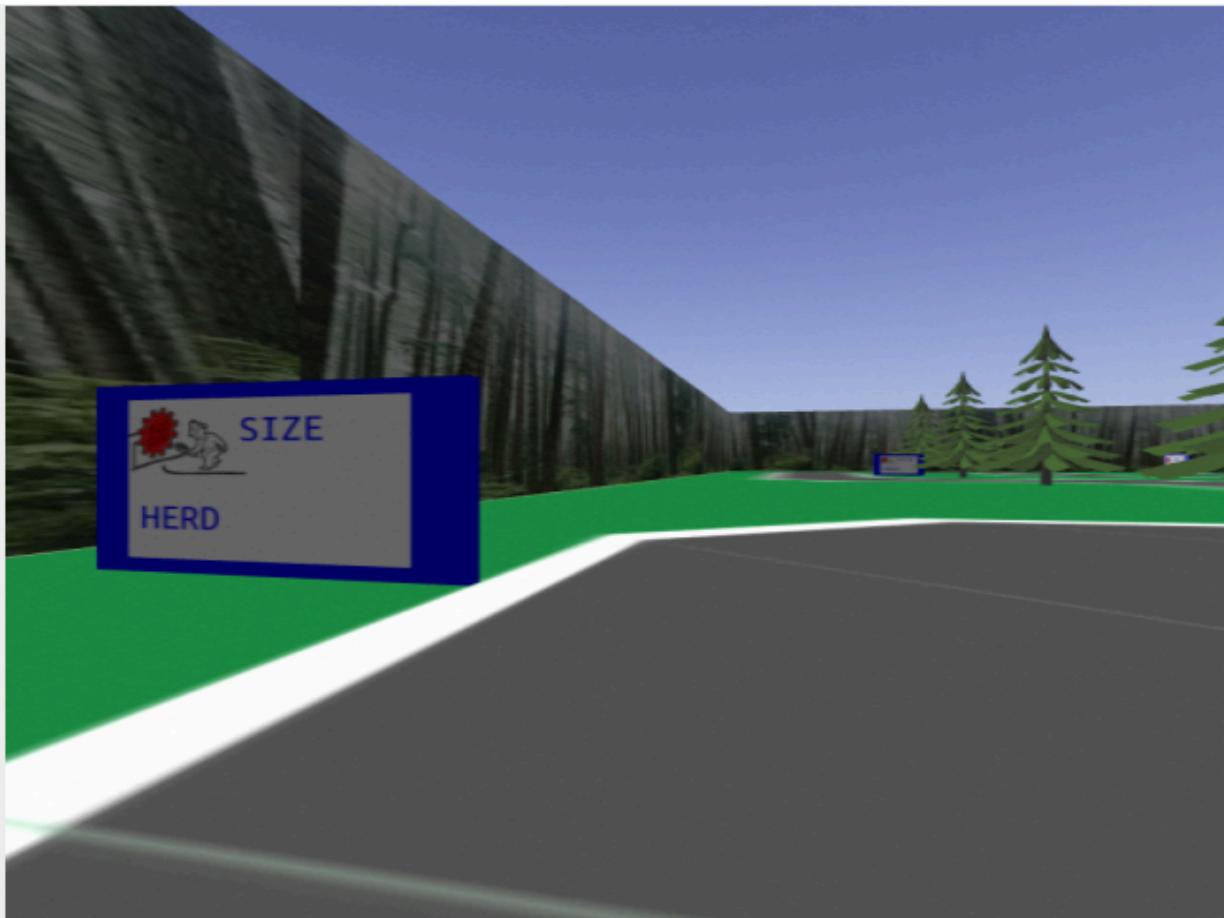
UNDATED LOGBOOK I USED FOR FINAL PROJECT



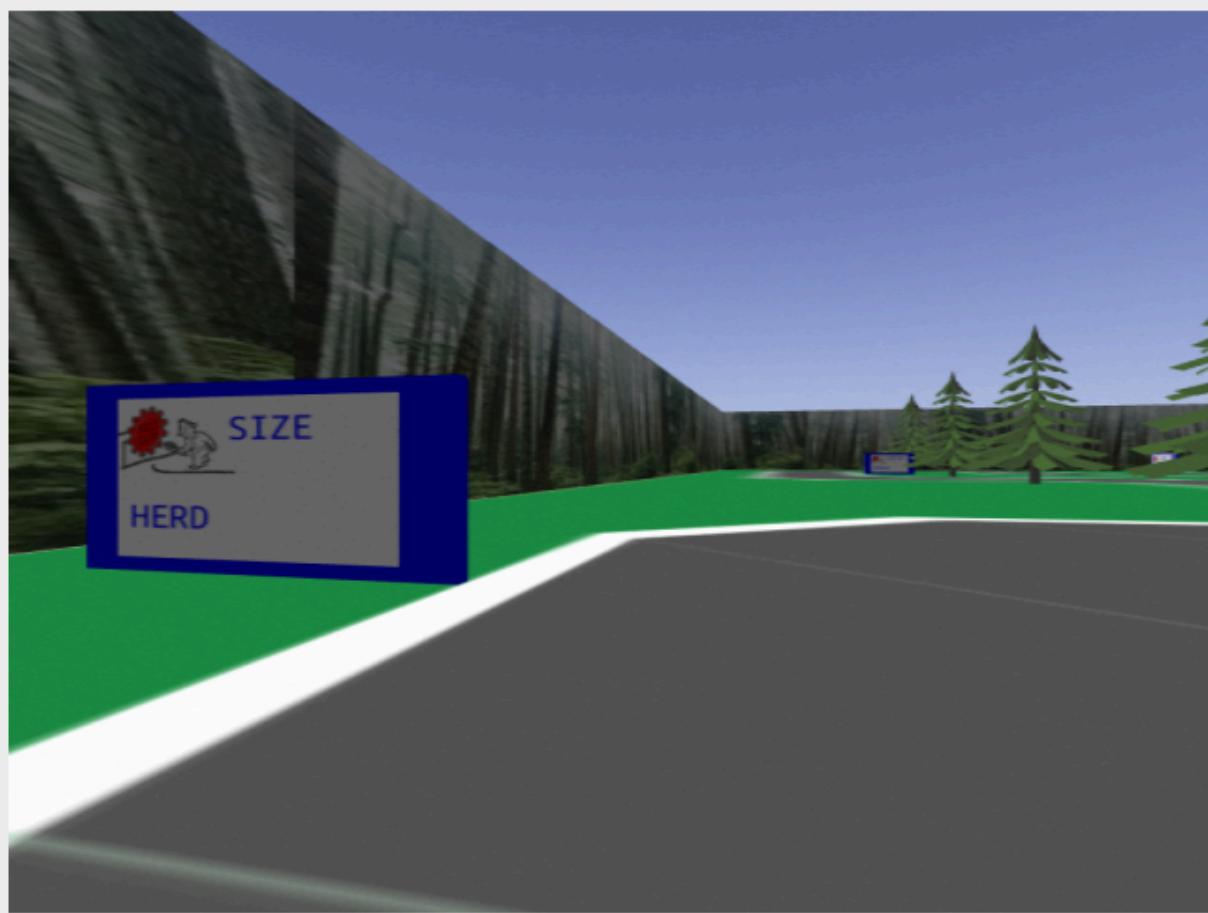
UNDATED LOGBOOK I USED FOR FINAL PROJECT



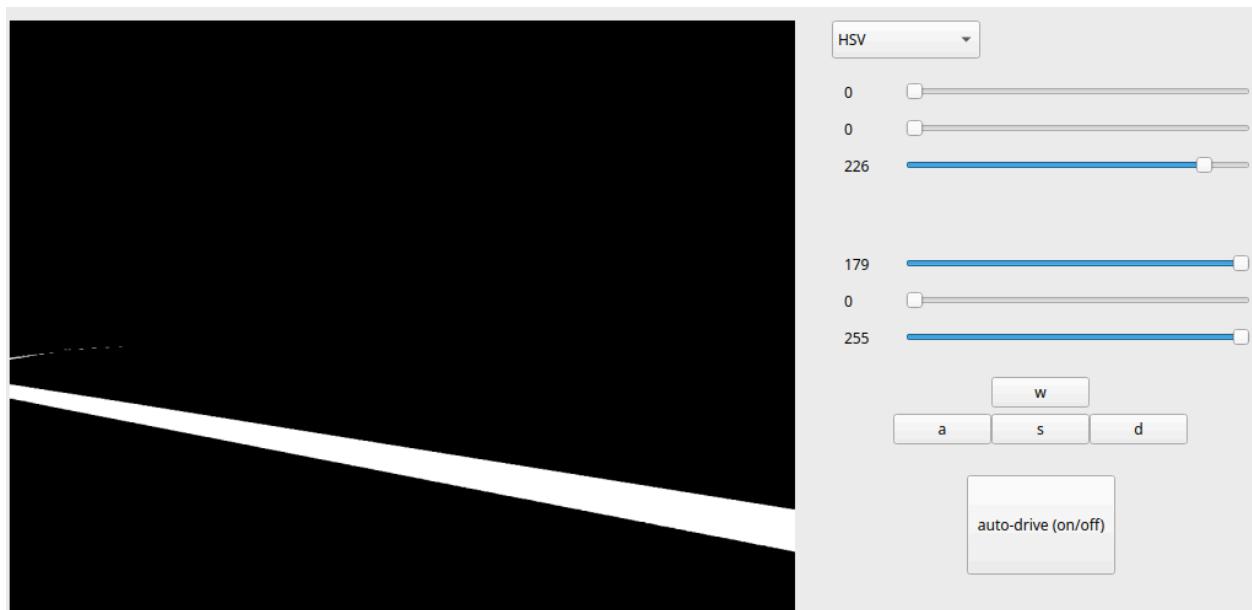
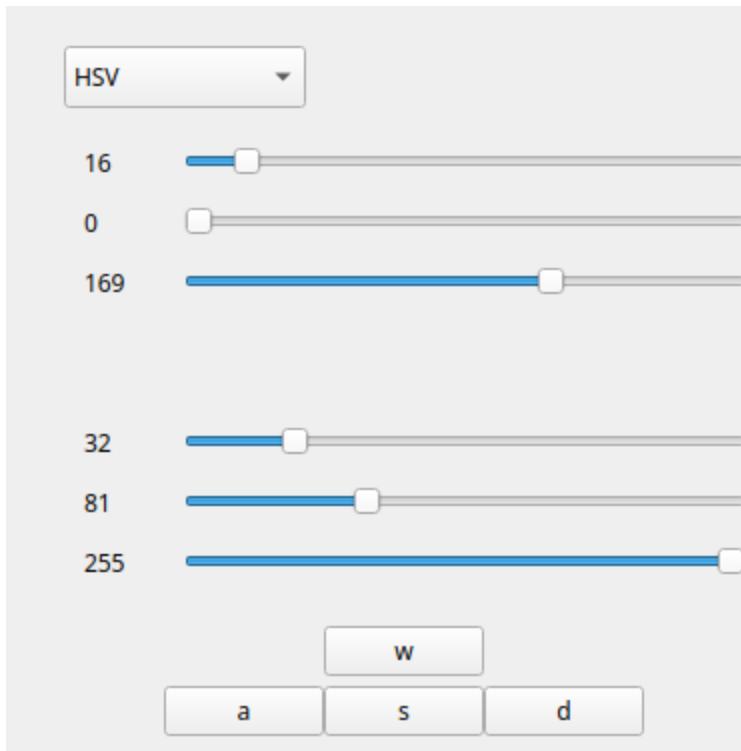
UNDATED LOGBOOK I USED FOR FINAL PROJECT



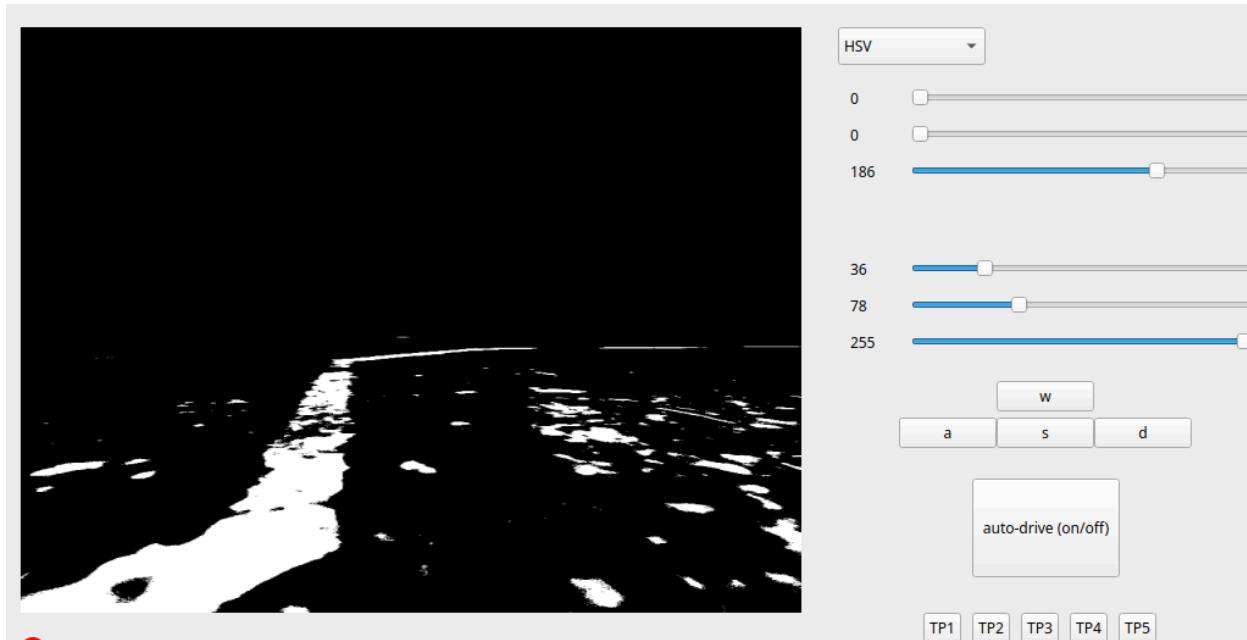
UNDATED LOGBOOK I USED FOR FINAL PROJECT



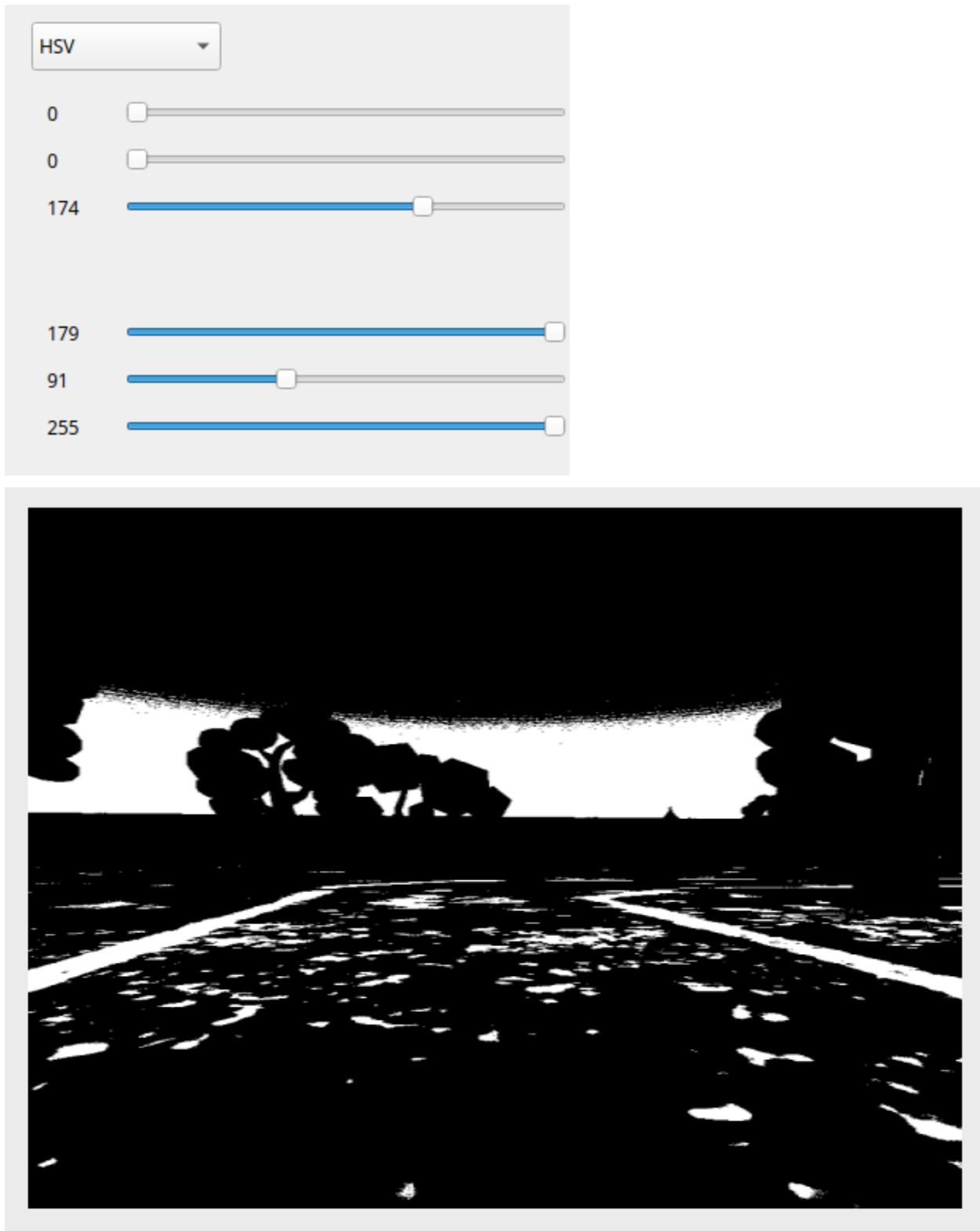
UNDATED LOGBOOK I USED FOR FINAL PROJECT



UNDATED LOGBOOK I USED FOR FINAL PROJECT



UNDATED LOGBOOK I USED FOR FINAL PROJECT



10,10

UNDATED LOGBOOK I USED FOR FINAL PROJECT



fizzer@skynet:~ 2024-12-02 00:28:34

```
$ rosservice call /gazebo/get_model_state "{model_name: 'B1', relative_entity_name: 'world'}
```

header:

seq: 1

stamp:

secs: 100

nsecs: 452000000

frame_id: "world"

pose:

position:

UNDATED LOGBOOK I USED FOR FINAL PROJECT

x: 5.49942880783774

y: 2.504030700996579

z: 0.04000039797012949

orientation:

x: 7.377270669093508e-07

y: -4.3205541389025773e-07

z: -0.7090994033738673

w: 0.7051085279119056

twist:

linear:

x: -3.429385519482691e-07

y: -3.45936087976483e-05

z: -6.961205134532641e-08

angular:

x: -3.3675953025424427e-08

y: -1.695382569344429e-07

z: 2.0465679826048485e-07

success: True

status_message: "GetModelState: got properties"

fizzer@skynet:~ 2024-12-02 00:29:12

\$

here's position one

fizzer@skynet:~ 2024-12-02 00:29:12

\$ rosservice call /gazebo/get_model_state "{model_name: 'B1', relative_entity_name: 'world'}"

UNDATED LOGBOOK I USED FOR FINAL PROJECT

header:

seq: 2

stamp:

secs: 179

nsecs: 510000000

frame_id: "world"

pose:

position:

x: 5.3474887443153785

y: -0.9349569584188745

z: 0.04000037723629943

orientation:

x: 1.7321540120358244e-06

y: 8.610931097589084e-07

z: -0.814923731390067

w: 0.5795682117003553

twist:

linear:

x: -1.9867217578075866e-06

y: -3.4124615815287025e-06

z: -2.5248927044901413e-07

angular:

x: -1.1459745374714128e-06

y: 8.582504550650873e-08

UNDATED LOGBOOK I USED FOR FINAL PROJECT

z: 3.4741473200116976e-06

success: True

status_message: "GetModelState: got properties"

fizzer@skynet:~ 2024-12-02 00:30:31

\$

heres position two

fizzer@skynet:~ 2024-12-02 00:30:31

\$ rosservice call /gazebo/get_model_state "{model_name: 'B1', relative_entity_name: 'world'}"

header:

seq: 3

stamp:

secs: 226

nsecs: 396000000

frame_id: "world"

pose:

position:

x: 4.31305060031579

y: -1.3878525178704242

z: 0.04000050138909643

orientation:

x: 2.8548229122584364e-07

y: 5.904484287748461e-08

z: -0.9992515317144463

w: -0.038683024264505914

UNDATED LOGBOOK I USED FOR FINAL PROJECT

twist:

linear:

x: -4.6083482126018436e-05

y: 4.578524056041447e-06

z: -3.021915736653228e-07

angular:

x: -9.706895705051767e-07

y: -5.44937590917467e-07

z: -3.980972746240802e-06

success: True

status_message: "GetModelState: got properties"

fizzer@skynet:~ 2024-12-02 00:31:18

\$

heres position three

fizzer@skynet:~ 2024-12-02 00:31:18

\$ rosservice call /gazebo/get_model_state "{model_name: 'B1', relative_entity_name: 'world'}"

header:

seq: 4

stamp:

secs: 264

nsecs: 878000000

frame_id: "world"

pose:

position:

UNDATED LOGBOOK I USED FOR FINAL PROJECT

x: 0.6491008198665422

y: -0.9207976055506509

z: 0.04000053773996677

orientation:

x: 5.673153474413703e-07

y: 1.1238297852020603e-07

z: -0.6684896009764731

w: -0.7437214891247805

twist:

linear:

x: 3.6258892883050236e-05

y: 4.6779347039224744e-05

z: 0.0002147694215817508

angular:

x: 0.000118433678818386

y: -7.521498400007287e-05

z: 4.92834007489305e-05

success: True

status_message: "GetModelState: got properties"

fizzer@skynet:~ 2024-12-02 00:31:57

\$

heres position 4

fizzer@skynet:~ 2024-12-02 00:31:57

\$ rosservice call /gazebo/get_model_state "{model_name: 'B1', relative_entity_name: 'world'}"

UNDATED LOGBOOK I USED FOR FINAL PROJECT

header:

seq: 5

stamp:

secs: 302

nsecs: 346000000

frame_id: "world"

pose:

position:

x: 0.6689328884773375

y: 2.01851026363387

z: 0.04000094070292316

orientation:

x: -2.860153819036568e-06

y: 3.005876131366921e-06

z: 0.7085922122246099

w: -0.7056182230905165

twist:

linear:

x: -9.686719334696047e-09

y: -1.0315268016265393e-05

z: -2.2654604084937307e-08

angular:

x: 1.1899176535352882e-08

y: 2.2921365858617846e-07

UNDATED LOGBOOK I USED FOR FINAL PROJECT

z: 1.3864187333922176e-07

success: True

status_message: "GetModelState: got properties"

fizzer@skynet:~ 2024-12-02 00:32:35

\$

here's position 5

fizzer@skynet:~ 2024-12-02 00:32:35

\$ rosservice call /gazebo/get_model_state "{model_name: 'B1', relative_entity_name: 'world'}"

header:

seq: 6

stamp:

secs: 331

nsecs: 98000000

frame_id: "world"

pose:

position:

x: -3.0218217082015695

y: 1.5572375439630923

z: 0.03997891270315111

orientation:

x: -4.068919485243806e-05

y: -9.436718605393764e-05

z: 0.9984824087289902

w: 0.05507148897549148

UNDATED LOGBOOK I USED FOR FINAL PROJECT

twist:

linear:

x: -0.000986809555304847

y: -0.0014373767037252595

z: -0.012317697313816734

angular:

x: 0.0986408972496164

y: -0.04489834128880002

z: -0.0051109325794474

success: True

status_message: "GetModelState: got properties"

fizzer@skynet:~ 2024-12-02 00:33:04

\$

heres position 6

fizzer@skynet:~ 2024-12-02 00:34:10

\$ rosservice call /gazebo/get_model_state "{model_name: 'B1', relative_entity_name: 'world'}"

header:

seq: 8

stamp:

secs: 412

nsecs: 700000000

frame_id: "world"

pose:

position:

UNDATED LOGBOOK I USED FOR FINAL PROJECT

x: -4.301637175421005

y: -2.312349652507272

z: 0.03998563711874584

orientation:

x: -6.732302218612708e-06

y: 0.00012692216855302477

z: -0.05064239775258873

w: -0.9987168424510062

twist:

linear:

x: -0.0006081228219745929

y: -0.00021954653680757505

z: -0.00753131928801298

angular:

x: 0.012700679493104099

y: -0.12287423523921809

z: 0.001318436035590488

success: True

status_message: "GetModelState: got properties"

fizzer@skynet:~ 2024-12-02 00:34:25

\$

heres position 7

fizzer@skynet:~ 2024-12-02 00:34:25

\$ rosservice call /gazebo/get_model_state "{model_name: 'B1', relative_entity_name: 'world'}"

UNDATED LOGBOOK I USED FOR FINAL PROJECT

header:

seq: 1

stamp:

secs: 88

nsecs: 468000000

frame_id: "world"

pose:

position:

x: -1.2089846301463472

y: -1.186840844647

z: 1.8503400563324273

orientation:

x: -0.0057606116672731245

y: -0.013744051998107805

z: -0.0008294302102146539

w: -0.9998886080126218

twist:

linear:

x: 0.0017851634276949322

y: -0.0031014706121645142

z: -0.003942814968645114

angular:

x: 0.20511033665167425

y: -0.0014941832167760828

UNDATED LOGBOOK I USED FOR FINAL PROJECT

z: 0.004200353113679486

success: True

status_message: "GetModelState: got properties"

fizzer@skynet:~ 2024-12-02 00:46:17

\$

heres position 8

fizzer@skynet:~/fizzcomp 2024-12-02 05:54:58

\$ python3 -c "import tensorflow as tf; print('TensorFlow version:', tf.__version__)"

2024-12-02 05:57:18.495692: W tensorflow/stream_executor/platform/default/dso_loader.cc:64]

Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or directory; LD_LIBRARY_PATH:

/home/fizzer/fizzcomp/devel/lib:/opt/ros/noetic/lib

2024-12-02 05:57:18.495714: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.

TensorFlow version: 2.9.3

fizzer@skynet:~/fizzcomp 2024-12-02 05:57:22

\$ python3 -c "import keras; print('Keras version:', keras.__version__)"

2024-12-02 05:57:22.992055: W tensorflow/stream_executor/platform/default/dso_loader.cc:64]

Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or directory; LD_LIBRARY_PATH:

/home/fizzer/fizzcomp/devel/lib:/opt/ros/noetic/lib

2024-12-02 05:57:22.992093: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.

Keras version: 2.9.0

fizzer@skynet:~/fizzcomp 2024-12-02 05:57:24

\$

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

```
import sys
import os
import rospkg
import rospy
from PyQt5 import QtWidgets, uic, QtCore, QtGui
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import cv2
import numpy as np
import datetime

from PyQt5.QtCore import pyqtSignal

# ----- New Imports -----
from gazebo_msgs.srv import SetModelState, SetModelStateRequest
from gazebo_msgs.msg import ModelState
# -----



class ControllerGUI(QtWidgets.QMainWindow):
    # Define a signal that carries the processed image and billCombo selection
    image_update_signal = pyqtSignal(np.ndarray, str)

    def __init__(self):
        super(ControllerGUI, self).__init__()

    # Initialize ROS node
    rospy.init_node('controller_gui_node', anonymous=True)

    # Initialize CvBridge
    self.bridge = CvBridge()

    # Get the path to the 'controller' package
    rospack = rospkg.RosPack()
    package_path = rospack.get_path('controller')

    # Construct the full path to 'developer_tools.ui'
    ui_file = os.path.join(package_path, 'developer_tools.ui') # Adjust if it's in a subdirectory

    # Load the UI file
    if not os.path.exists(ui_file):
        rospy.logerr(f"UI file not found: {ui_file}")
        sys.exit(1)
    else:
        uic.loadUi(ui_file, self)

    # Make movement buttons checkable
    self.move_forward.setCheckable(True)
    self.move_backward.setCheckable(True)
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
self.move_left.setCheckable(True)
self.move_right.setCheckable(True)

# Set 'Raw' as the default option in mainCombo
index = self.mainCombo.findText("Raw")
if index != -1:
    self.mainCombo.setCurrentIndex(index)

# Set 'Raw' as the default option in billCombo
bill_index = self.billCombo.findText("Raw")
if bill_index != -1:
    self.billCombo.setCurrentIndex(bill_index)

# Set up publishers
self.pub_cmd_vel = rospy.Publisher('/B1/cmd_vel', Twist, queue_size=10)

# Connect UI elements to functions
self.move_forward.clicked.connect(self.toggle_move_forward)
self.move_backward.clicked.connect(self.toggle_move_backward)
self.move_left.clicked.connect(self.toggle_move_left)
self.move_right.clicked.connect(self.toggle_move_right)
self.auto_drive_toggle.clicked.connect(self.auto_drive_toggle_function)
self.saveImage.clicked.connect(self.save_image_function)

# Initialize set to keep track of pressed keys
self.pressed_keys = set()

# Movement flags controlled by buttons
self.button_move_forward = False
self.button_move_backward = False
self.button_move_left = False
self.button_move_right = False

# Start a timer to call publish_movement at regular intervals
self.timer = QtCore.QTimer()
self.timer.timeout.connect(self.publish_movement)
self.timer.start(100) # Every 100 ms (10 Hz)

# Subscribe to the image topic
self.image_sub = rospy.Subscriber('/B1/rrbot/camera1/image_raw', Image,
self.image_callback)

# Connect the image update signal to the update_billboard slot
self.image_update_signal.connect(self.update_billboard)
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
# Ensure the window can accept focus and receive key events
self.setFocusPolicy(QtCore.Qt.StrongFocus)

# ----- Added Section: Initialize Sliders and Labels for HSV -----

# Default HSV bounds
self.lower_color = np.array([0, 0, 174])
self.upper_color = np.array([179, 91, 255])

# Set default slider values for lower bounds
self.hSlider.setMinimum(0)
self.hSlider.setMaximum(179)
self.hSlider.setValue(self.lower_color[0])
self.sSlider.setMinimum(0)
self.sSlider.setMaximum(255)
self.sSlider.setValue(self.lower_color[1])
self.vSlider.setMinimum(0)
self.vSlider.setMaximum(255)
self.vSlider.setValue(self.lower_color[2])

# Set default slider values for upper bounds
self.hSlider_2.setMinimum(0)
self.hSlider_2.setMaximum(179)
self.hSlider_2.setValue(self.upper_color[0])
self.sSlider_2.setMinimum(0)
self.sSlider_2.setMaximum(255)
self.sSlider_2.setValue(self.upper_color[1])
self.vSlider_2.setMinimum(0)
self.vSlider_2.setMaximum(255)
self.vSlider_2.setValue(self.upper_color[2])

# Update labels with default slider values
self.hText.setText(str(self.hSlider.value()))
self.sText.setText(str(self.sSlider.value()))
self.vText.setText(str(self.vSlider.value()))
self.hText_2.setText(str(self.hSlider_2.value()))
self.sText_2.setText(str(self.sSlider_2.value()))
self.vText_2.setText(str(self.vSlider_2.value()))

# Connect sliders to their respective update functions
self.hSlider.valueChanged.connect(self.update_hText)
self.sSlider.valueChanged.connect(self.update_sText)
self.vSlider.valueChanged.connect(self.update_vText)
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
self.hSlider_2.valueChanged.connect(self.update_hText_2)
self.sSlider_2.valueChanged.connect(self.update_sText_2)
self.vSlider_2.valueChanged.connect(self.update_vText_2)

# ----- End of Added Section -----


# ----- New Section: Define Teleport Positions -----
self.teleport_positions = {
    'TP1': {
        'position': {'x': 5.49942880783774, 'y': 2.504030700996579, 'z':
0.04000039797012949},
        'orientation': {'x': 7.377270669093508e-07, 'y': -4.3205541389025773e-07, 'z':
-0.7090994033738673, 'w': 0.7051085279119056}
    },
    'TP2': {
        'position': {'x': 5.3474887443153785, 'y': -0.9349569584188745, 'z':
0.04000037723629943},
        'orientation': {'x': 1.7321540120358244e-06, 'y': 8.610931097589084e-07, 'z':
-0.814923731390067, 'w': 0.5795682117003553}
    },
    'TP3': {
        'position': {'x': 4.31305060031579, 'y': -1.3878525178704242, 'z':
0.04000050138909643},
        'orientation': {'x': 2.8548229122584364e-07, 'y': 5.904484287748461e-08, 'z':
-0.9992515317144463, 'w': -0.038683024264505914}
    },
    'TP4': {
        'position': {'x': 0.6491008198665422, 'y': -0.9207976055506509, 'z':
0.04000053773996677},
        'orientation': {'x': 5.673153474413703e-07, 'y': 1.1238297852020603e-07, 'z':
-0.6684896009764731, 'w': -0.7437214891247805}
    },
    'TP5': {
        'position': {'x': 0.6689328884773375, 'y': 2.01851026363387, 'z':
0.04000094070292316},
        'orientation': {'x': -2.860153819036568e-06, 'y': 3.005876131366921e-06, 'z':
0.7085922122246099, 'w': -0.7056182230905165}
    },
    'TP6': {
        'position': {'x': -3.0218217082015695, 'y': 1.5572375439630923, 'z':
0.03997891270315111},
        'orientation': {'x': -4.068919485243806e-05, 'y': -9.436718605393764e-05, 'z':
0.9984824087289902, 'w': 0.05507148897549148}
    },
}
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
'TP7': {
    'position': {'x': -4.301637175421005, 'y': -2.312349652507272, 'z':
0.03998563711874584},
    'orientation': {'x': -6.732302218612708e-06, 'y': 0.00012692216855302477, 'z':
-0.05064239775258873, 'w': -0.9987168424510062}
},
'TP8': {
    'position': {'x': -1.2089846301463472, 'y': -1.186840844647, 'z':
1.8503400563324273},
    'orientation': {'x': -0.0057606116672731245, 'y': -0.013744051998107805, 'z':
-0.0008294302102146539, 'w': -0.9998886080126218}
}
}

# ----- End of New Section -----


# ----- New Section: Set Up Teleport Service Proxy -----
rospy.loginfo("Waiting for /gazebo/set_model_state service...")
rospy.wait_for_service('/gazebo/set_model_state')
try:
    self.set_model_state_service = rospy.ServiceProxy('/gazebo/set_model_state',
SetModelState)
    rospy.loginfo("/gazebo/set_model_state service is available.")
except rospy.ServiceException as e:
    rospy.logerr(f"Service initialization failed: {e}")
    sys.exit(1)
# ----- End of New Section -----


# ----- New Section: Connect TP Buttons to Teleport Functions -----
self.TP1.clicked.connect(lambda: self.teleport_to_position('TP1'))
self.TP2.clicked.connect(lambda: self.teleport_to_position('TP2'))
self.TP3.clicked.connect(lambda: self.teleport_to_position('TP3'))
self.TP4.clicked.connect(lambda: self.teleport_to_position('TP4'))
self.TP5.clicked.connect(lambda: self.teleport_to_position('TP5'))
self.TP6.clicked.connect(lambda: self.teleport_to_position('TP6'))
self.TP7.clicked.connect(lambda: self.teleport_to_position('TP7'))
self.TP8.clicked.connect(lambda: self.teleport_to_position('TP8'))
# ----- End of New Section -----


# ----- New Section: Teleport Function -----
def teleport_to_position(self, tp_name):
"""
Teleports the robot to the specified TP position.

:param tp_name: String name of the TP button (e.g., 'TP1', 'TP2', ...)

```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
"""
if tp_name not in self.teleport_positions:
    rospy.logerr(f"Teleport position '{tp_name}' not defined.")
    return

position = self.teleport_positions[tp_name]['position']
orientation = self.teleport_positions[tp_name]['orientation']

# Create a ModelState message
model_state = ModelState()
model_state.model_name = 'B1' # Ensure this matches your robot's model name in
Gazebo
model_state.pose.position.x = position['x']
model_state.pose.position.y = position['y']
model_state.pose.position.z = position['z']
model_state.pose.orientation.x = orientation['x']
model_state.pose.orientation.y = orientation['y']
model_state.pose.orientation.z = orientation['z']
model_state.pose.orientation.w = orientation['w']
model_state.reference_frame = 'world' # Relative to the 'world' frame

# Create the service request
set_state_request = SetModelStateRequest()
set_state_request.model_state = model_state

try:
    # Call the service to set the model state
    response = self.set_model_state_service(set_state_request)
    if response.success:
        rospy.loginfo(f"Successfully teleported to {tp_name}.")
    else:
        rospy.logerr(f"Failed to teleport to {tp_name}: {response.status_message}")
except rospy.ServiceException as e:
    rospy.logerr(f"Service call failed: {e}")
# ----- End of New Section -----

# ----- Existing Methods Below -----
def toggle_move_forward(self):
    self.button_move_forward = self.move_forward.isChecked()
    if self.button_move_forward:
        self.move_forward.setStyleSheet("background-color: green")
    else:
        self.move_forward.setStyleSheet("")
    # rospy.loginfo(f"Move Forward: {'On' if self.button_move_forward else 'Off'}")
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
def toggle_move_backward(self):
    self.button_move_backward = self.move_backward.isChecked()
    if self.button_move_backward:
        self.move_backward.setStyleSheet("background-color: green")
    else:
        self.move_backward.setStyleSheet("")
    # rospy.loginfo(f"Move Backward: {'On' if self.button_move_backward else 'Off'}")

def toggle_move_left(self):
    self.button_move_left = self.move_left.isChecked()
    if self.button_move_left:
        self.move_left.setStyleSheet("background-color: green")
    else:
        self.move_left.setStyleSheet("")
    # rospy.loginfo(f"Move Left: {'On' if self.button_move_left else 'Off'}")

def toggle_move_right(self):
    self.button_move_right = self.move_right.isChecked()
    if self.button_move_right:
        self.move_right.setStyleSheet("background-color: green")
    else:
        self.move_right.setStyleSheet("")
    # rospy.loginfo(f"Move Right: {'On' if self.button_move_right else 'Off'}")

# Keyboard event handlers
def keyPressEvent(self, event):
    if not event.isAutoRepeat():
        key = event.key()
        if key in [QtCore.Qt.Key_W, QtCore.Qt.Key_A, QtCore.Qt.Key_S, QtCore.Qt.Key_D]:
            self.pressed_keys.add(key)
            # rospy.logdebug(f"Key Pressed: {QtCore.Qt.keyToString(key)}")

def keyReleaseEvent(self, event):
    if not event.isAutoRepeat():
        key = event.key()
        if key in [QtCore.Qt.Key_W, QtCore.Qt.Key_A, QtCore.Qt.Key_S, QtCore.Qt.Key_D]:
            self.pressed_keys.discard(key)
            # rospy.logdebug(f"Key Released: {QtCore.Qt.keyToString(key)}")

# Function to publish movement commands
def publish_movement(self):
    twist = Twist()
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
# Keyboard-controlled movement
if QtCore.Qt.Key_W in self.pressed_keys:
    twist.linear.x += 5.0 # Move forward
if QtCore.Qt.Key_S in self.pressed_keys:
    twist.linear.x -= 5.0 # Move backward
if QtCore.Qt.Key_A in self.pressed_keys:
    twist.angular.z += 3.0 # Turn left
if QtCore.Qt.Key_D in self.pressed_keys:
    twist.angular.z -= 3.0 # Turn right

# Button-controlled movement
if self.button_move_forward:
    twist.linear.x += 1.0
if self.button_move_backward:
    twist.linear.x -= 1.0
if self.button_move_left:
    twist.angular.z += 1.0
if self.button_move_right:
    twist.angular.z -= 1.0

# Publish the twist message
self.pub_cmd_vel.publish(twist)
# rospy.logdebug(f"Published Twist: linear.x={twist.linear.x}, angular.z={twist.angular.z}")

def save_image_function(self):
    """
    Saves the current image displayed on the billboard QLabel to a file.
    """
    try:
        # Retrieve the current pixmap from the billboard
        pixmap = self.billboard.pixmap()
        if pixmap:
            # Get the path to the 'controller' package
            rospack = rospkg.RosPack()
            package_path = rospack.get_path('controller')

            # Define the directory to save images
            save_dir = os.path.join(package_path, 'saved_images')
            os.makedirs(save_dir, exist_ok=True) # Create directory if it doesn't exist

            # Generate a timestamped filename
            timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
            filename = f"billboard_{timestamp}.png"
            file_path = os.path.join(save_dir, filename)
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
# Save the pixmap to the file
if not pixmap.save(file_path):
    rospy.logerr(f"Failed to save image to {file_path}")
else:
    rospy.loginfo(f"Image saved to {file_path}")
else:
    rospy.logwarn("No image to save on the billboard.")
except Exception as e:
    rospy.logerr(f"Error saving image: {e}")

def auto_drive_toggle_function(self):
    # Implement
    pass

# Helper function to outline the largest contour on a binary image
def outline_largest_contour(self, binary_image):
    contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if not contours:
        return None # No contours found

    # Find the largest contour by area
    largest_contour = max(contours, key=cv2.contourArea)

    # Draw the largest contour on the binary image
    outlined_image = binary_image.copy()
    cv2.drawContours(outlined_image, [largest_contour], -1, 255, 2) # White color
(thickness 2)

    return outlined_image

# Helper function to perform inverse perspective transform
def inverse_perspective_transform(self, outlined_image):
    contours, _ = cv2.findContours(outlined_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if not contours:
        return None # No contours found

    # Find the largest contour by area
    largest_contour = max(contours, key=cv2.contourArea)

    # Approximate the contour to a polygon
    peri = cv2.arcLength(largest_contour, True)
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
approx = cv2.approxPolyDP(largest_contour, 0.01 * peri, True) # 2% approximation

if len(approx) != 4:
    # Not a quadrilateral; cannot perform perspective transform
    rospy.logwarn("Largest contour is not a quadrilateral. Skipping perspective transform.")
    return None

# Order the points in consistent order: top-left, top-right, bottom-right, bottom-left
pts = approx.reshape(4, 2)
rect = self.order_points(pts)

# Compute the width and height of the new image
(tl, tr, br, bl) = rect
widthA = np.linalg.norm(br - bl)
widthB = np.linalg.norm(tr - tl)
maxWidth = max(int(widthA), int(widthB))

heightA = np.linalg.norm(tr - br)
heightB = np.linalg.norm(tl - bl)
maxHeight = max(int(heightA), int(heightB))

# Destination points for the perspective transform
dst = np.array([
    [0, 0],
    [maxWidth - 1, 0],
    [maxWidth - 1, maxHeight - 1],
    [0, maxHeight - 1]
], dtype="float32")

# Compute the perspective transform matrix
M = cv2.getPerspectiveTransform(rect, dst)

# Apply the perspective transform
warped = cv2.warpPerspective(outlined_image, M, (maxWidth, maxHeight))

return warped

# Helper function to order points
def order_points(self, pts):
    # Initialize a list of coordinates that will be ordered
    rect = np.zeros((4, 2), dtype="float32")

    # Sum and difference to find top-left and bottom-right
    s = pts.sum(axis=1)
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
diff = np.diff(pts, axis=1)

rect[0] = pts[np.argmin(s)]    # Top-left
rect[2] = pts[np.argmax(s)]   # Bottom-right
rect[1] = pts[np.argmin(diff)] # Top-right
rect[3] = pts[np.argmax(diff)]# Bottom-left

return rect

def image_callback(self, msg):
    # Process mainfeed based on mainCombo selection
    main_selection = self.mainCombo.currentText()

    if main_selection == "Raw":
        try:
            # Convert ROS Image message to OpenCV image
            cv_image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')

            # Convert the image to RGB format
            cv_image_rgb = cv2.cvtColor(cv_image, cv2.COLOR_BGR2RGB)

            # Get image dimensions
            height, width, channel = cv_image_rgb.shape
            bytes_per_line = 3 * width

            # Convert to QImage for mainfeed
            qt_image = QtGui.QImage(cv_image_rgb.data, width, height, bytes_per_line,
QtGui.QImage.Format_RGB888)

            # Scale the image to fit the QLabel while maintaining aspect ratio
            scaled_image = qt_image.scaled(self.mainfeed.size(),
QtCore.Qt.KeepAspectRatio, QtCore.Qt.SmoothTransformation)

            # Set the pixmap of the QLabel
            self.mainfeed.setPixmap(QtGui.QPixmap.fromImage(scaled_image))

        except CvBridgeError as e:
            rospy.logerr(f"CvBridge Error: {e}")

    elif main_selection == "HSV":
        try:
            # Retrieve current HSV bounds from sliders
            lower_h = self.hSlider.value()
            lower_s = self.sSlider.value()
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
lower_v = self.vSlider.value()
upper_h = self.hSlider_2.value()
upper_s = self.sSlider_2.value()
upper_v = self.vSlider_2.value()

# Update lower and upper color arrays
lower_color = np.array([lower_h, lower_s, lower_v])
upper_color = np.array([upper_h, upper_s, upper_v])

# Convert ROS Image message to OpenCV image
cv_image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')

# Convert the image to HSV color space
hsv_image = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV)

# Create a binary mask where the target color is white and the rest is black
mask = cv2.inRange(hsv_image, lower_color, upper_color)

# Apply morphological operations to remove noise and smooth the mask
kernel = np.ones((1, 10), np.uint8)
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel, iterations=2)
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel, iterations=2)

# Display the processed mask
processed_image_display = mask

# Convert processed image to QImage for display
height, width = processed_image_display.shape
bytes_per_line = width
qt_image = QtGui.QImage(processed_image_display.data, width, height,
bytes_per_line, QtGui.QImage.Format_Grayscale8)

# Scale the image to fit the mainfeed QLabel while maintaining aspect ratio
scaled_image = qt_image.scaled(self.mainfeed.size(),
QtCore.Qt.KeepAspectRatio, QtCore.Qt.SmoothTransformation)

# Set the pixmap of the mainfeed QLabel
self.mainfeed.setPixmap(QtGui.QPixmap.fromImage(scaled_image))

except CvBridgeError as e:
    rospy.logerr(f"CvBridge Error: {e}")

else:
    rospy.logwarn(f"Unknown mainCombo selection: {main_selection}")
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
# Optionally, handle other cases or default behavior

# Process billboard based on billCombo selection
bill_selection = self.billCombo.currentText()

try:
    # Convert ROS Image message to OpenCV image
    cv_image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')

    # Convert the image to HSV color space
    hsv_image = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV)

    # Define the lower and upper bounds for the target color (e.g., blue)
    lower_color_bill = np.array([100, 120, 0])
    upper_color_bill = np.array([140, 255, 255])

    # Create a binary mask where the target color is white and the rest is black
    mask = cv2.inRange(hsv_image, lower_color_bill, upper_color_bill)

    # Apply morphological operations to remove noise and smooth the mask
    kernel = np.ones((1, 1), np.uint8)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel, iterations=2)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel, iterations=2)

    # Initialize variables
    processed_image = mask.copy() # Start with the cleaned binary image
    quadrilateral_found = False # Flag for billboard indicator

    if bill_selection == "Raw":
        # Display the cleaned binary image directly
        processed_image_display = mask

    elif bill_selection == "Contour":
        # Outline the largest quadrilateral contour on the cleaned binary image
        outlined_image = self.outline_largest_contour(mask)
        if outlined_image is not None:
            processed_image_display = outlined_image
        else:
            processed_image_display = mask

    elif bill_selection == "Homography":
        # Outline the largest quadrilateral contour on the cleaned binary image
        outlined_image = self.outline_largest_contour(mask)
        if outlined_image is not None:
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
# Attempt inverse perspective transform
warped_image = self.inverse_perspective_transform(outlined_image)
if warped_image is not None:
    processed_image_display = warped_image
    quadrilateral_found = True # IPT successful
else:
    processed_image_display = outlined_image
else:
    processed_image_display = mask

else:
    rospy.logwarn(f"Unknown billCombo selection: {bill_selection}")
    processed_image_display = mask

# Emit the signal with the processed image and bill selection
self.image_update_signal.emit(processed_image_display, bill_selection)

except CvBridgeError as e:
    rospy.logerr(f"CvBridge Error: {e}")

@QtCore.pyqtSlot(np.ndarray, str)
def update_billboard(self, processed_image_display, bill_selection):
    # Convert processed image to QImage for display
    if len(processed_image_display.shape) == 2:
        # Grayscale image
        height, width = processed_image_display.shape
        bytes_per_line = width
        qt_image = QtGui.QImage(processed_image_display.data, width, height, bytes_per_line,
QtGui.QImage.Format_Grayscale8)
    else:
        # Color image (warped_image should be color if IPT was successful)
        processed_image_rgb = cv2.cvtColor(processed_image_display,
cv2.COLOR_BGR2RGB)
        height, width, channel = processed_image_rgb.shape
        bytes_per_line = 3 * width
        qt_image = QtGui.QImage(processed_image_rgb.data, width, height, bytes_per_line,
QtGui.QImage.Format_RGB888)

    # Scale the image to fit the billboard QLabel while maintaining aspect ratio
    scaled_image = qt_image.scaled(self.billboard.size(), QtCore.Qt.KeepAspectRatio,
QtCore.Qt.SmoothTransformation)

    # Set the pixmap of the billboard QLabel
    self.billboard.setPixmap(QtGui.QPixmap.fromImage(scaled_image))
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
# ----- Added Section: Slider Update Functions -----
def update_hText(self, value):
    self.hText.setText(str(value))

def update_sText(self, value):
    self.sText.setText(str(value))

def update_vText(self, value):
    self.vText.setText(str(value))

def update_hText_2(self, value):
    self.hText_2.setText(str(value))

def update_sText_2(self, value):
    self.sText_2.setText(str(value))

def update_vText_2(self, value):
    self.vText_2.setText(str(value))
# ----- End of Added Section -----


if __name__ == '__main__':
    # Initialize ROS node in the main thread if not already initialized
    if not rospy.core.is_initialized():
        rospy.init_node('controller_gui_node', anonymous=True)

    app = QtWidgets.QApplication(sys.argv)
    window = ControllerGUI()
    window.show()
    try:
        sys.exit(app.exec_())
    except rospy.ROSInterruptException:
        Pass
```

TP TIME



END BOARD

fizzer@skynet:~ 2024-12-03 20:13:11

```
$ rosservice call /gazebo/get_model_state "{model_name: 'B1', relative_entity_name: 'world'}"
header:
  seq: 1
  stamp:
    secs: 208
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
nsecs: 170000000
frame_id: "world"
pose:
  position:
    x: -4.063470518828452
    y: -2.2564473800170513
    z: 0.04000018393550679
  orientation:
    x: -2.000162652701596e-07
    y: -5.830896379528323e-07
    z: -0.3377707635826839
    w: -0.9412284054725457
twist:
  linear:
    x: 7.2973450851087e-06
    y: 5.8943899099516665e-06
    z: -8.164412211317728e-08
  angular:
    x: 1.5344223329651384e-07
    y: 1.4127334378487544e-07
    z: -9.24432881558964e-07
success: True
status_message: "GetModelState: got properties"
fizzer@skynet:~ 2024-12-03 20:13:14
$
```

Start:

```
fizzer@skynet:~ 2024-12-03 20:13:14
$ rosservice call /gazebo/get_model_state "{model_name: 'B1', relative_entity_name: 'world'}"
header:
  seq: 2
  stamp:
    secs: 376
    nsecs: 58000000
  frame_id: "world"
pose:
  position:
    x: 0.5527487156990568
    y: -0.12600472106246535
    z: 0.04000056025622947
  orientation:
    x: 2.448884472570069e-07
    y: 1.278603394462025e-07
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

```
z: -0.7071178631556455
w: -0.7070956990437133
twist:
linear:
x: 8.116451582490024e-07
y: 6.492124274429354e-05
z: -3.323202547936527e-07
angular:
x: -2.427257071260534e-06
y: 1.283560883718495e-06
z: -5.485636227147017e-06
success: True
status_message: "GetModelState: got properties"
fizzer@skynet:~ 2024-12-03 20:16:03
$
```

Middle:

```
fizzer@skynet:~ 2024-12-03 20:16:03
$ rosservice call /gazebo/get_model_state "{model_name: 'B1', relative_entity_name: 'world'}"
header:
seq: 3
stamp:
secs: 437
nsecs: 758000000
frame_id: "world"
pose:
position:
x: -4.008629252948557
y: 0.4038460916553116
z: 0.04000060474749287
orientation:
x: 7.154773655035404e-07
y: -1.501085552816514e-06
z: -0.4271649858420221
w: -0.9041736972882035
twist:
linear:
x: 2.3543821966816424e-05
y: 2.572388113532436e-05
z: -6.195886853266245e-07
angular:
```

UNDATED LOGBOOK I USED FOR FINAL PROJECT

x: 2.739456085130211e-06
y: -9.326830447456623e-09
z: 6.524870489909662e-06

success: True

status_message: "GetModelState: got properties"

fizzer@skynet:~ 2024-12-03 20:17:05