EECS 4421

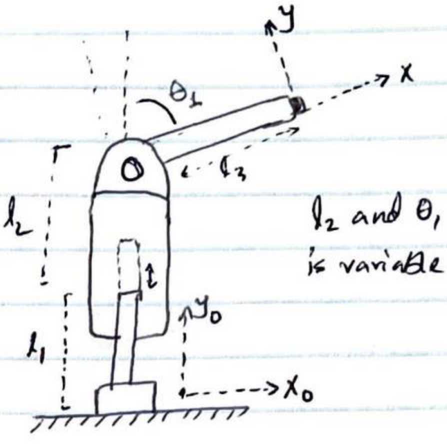
Assignment 2 Submission

Name: Mahfuz Rahman

Student No.: 217847518

Assignment 2
Mahfuz Rahman
217847518

1. Transformation between the tool frame and the base frame for the following robots:



1.1) This robot has two main joints,
↳ a prismatic joint that can extend along the y_0 axis. The length l_2 is added to the fixed length l_1.

↳ a revolute joint, that moves about the orthogonal z-axis by θ_1

So, the transformation matrix, from Base to prismatic joint is:

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & (l_1+l_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

the transformation matrix from prismatic joint to the revolute joint:
↳ here the revolute joint rotates by θ_1 around z-axis.

So,

$$T_2 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

the transformation matrix from Revolute Joint to the Tool frame:

$$T_3 = \begin{bmatrix} 1 & 0 & 0 & l_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

→ Having derived the transformation matrix for all the joints/frames, we can combine them to get the transformation from the base frame to the tool frame.

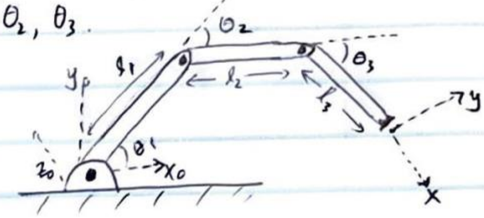
$$T_4 = T_1 \times T_2 \times T_3$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & (l_1+l_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & l_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & (l_3 \cos\theta_1) \\ \sin\theta_1 & \cos\theta_1 & 0 & (l_3 \sin\theta_1 + l_1 + l_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

1.2) For this robot, there exists,

* three revolute joints, that rotate by $\theta_1, \theta_2, \theta_3$.
* Three link lengths, l_1, l_2, l_3



Therefore,

↳ transformation matrix for base to first joint:

* here, there is a rotation by θ_1 (z-axis) and translation by l_1.

$$\therefore T_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & l_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & l_1\sin\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

↳ transformation matrix for, first joint to second joint:

* here, there is a rotation by θ_2 about the z-axis and, translation by l_2.

$$T_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & l_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & l_2\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

↳ Similarly, transformation matrix for, second joint to third joint (tool frame):

* here, there is a rotation by θ_3 about the z-axis and translation by l_3:

$$T_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & l_3\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & l_3\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

↳ Combining all the transformations gives us the transformation from the base frame to the tool frame:

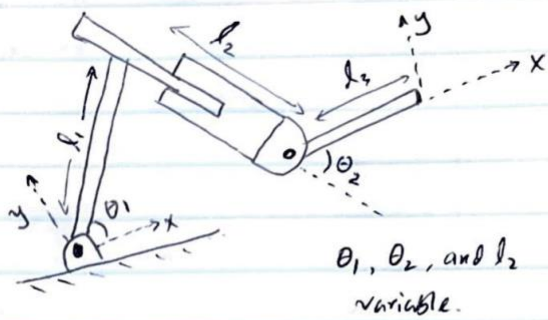
$$T_4 = T_1 \times T_2 \times T_3$$

$$= \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & l_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & l_1\sin\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & l_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & l_2\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & l_3\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & l_3\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$\cancel{\cos\theta_1\cos\theta_2 - \sin\theta_1\sin\theta_2 \quad -\cos\theta_1\sin\theta_2 - \sin\theta_1\cos\theta_2 \quad 0}$

1.3 For this robot, there exists:

* base joint, that rotates by θ_1
* prismatic joint, that extends by l_2.
* final joint, that rotates by θ_2.



θ_1, θ_2, and l_2 variable.

↳ Therefore, the transformation matrix from the base joint to the prismatic joint is:

* here there is a rotation by θ_1 and translation by l_1:

$$T_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & l_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & l_1\sin\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

↳ Transformation from the first joint to the prismatic joint:

* here the prismatic joint extends by l_2 with no rotation along the x-axis.

$$T_2 = \begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

↳ Transformation from prismatic joint to tool frame:

* here the revolute joint rotates by θ_3 around the z-axis and a translation by l_3.

$$T_3 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & l_3\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & l_3\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

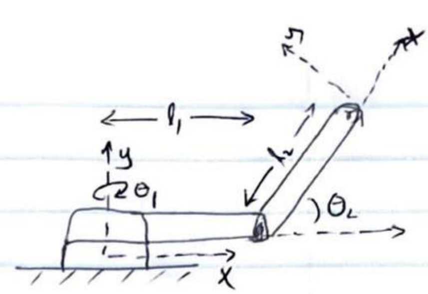
↳ Combining all the transformations, we get the final transformation from the base frame to the tool frame.

$$T_4 = T_1 \times T_2 \times T_3$$

$$= \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & l_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & l_1\sin\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & l_3\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & l_3\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & l_2\cos\theta_1 + l_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & l_2\sin\theta_1 + l_1\sin\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & l_3\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & l_3\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_1\cos\theta_2 - \sin\theta_1\sin\theta_2 & -\cos\theta_1\sin\theta_2 - \sin\theta_1\cos\theta_2 & 0 & \cos\theta_1(l_3\cos\theta_2 + l_2 + l_1) - l_3\sin\theta_2\sin\theta_1 \\ \sin\theta_1\cos\theta_2 + \cos\theta_1\sin\theta_2 & -\sin\theta_1\sin\theta_2 + \cos\theta_1\cos\theta_2 & 0 & \sin\theta_1(l_3\cos\theta_2 + l_2 + l_1) + l_3\sin\theta_2\cos\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



1.4 For the non-planar robot, there exists:

* first revolute joint that rotates around the y-axis. The link that is attached is of length l_1 and rotates out of the x-y plane.

* second revolute joint the rotates around the z-axis on the x-y plane. The link attached is of length l_2.

↳ Therefore, the transformation matrix from the base joint to the first revolute joint:

* here, there is rotation θ_1 around the y-axis and translation by l_1.

$$T_1 = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & l_1\cos\theta_1 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_1 & 0 & \cos\theta_1 & l_1\sin\theta_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

↳ transformation matrix from the ~~base join~~ first joint to second joint:

* here, there is rotation by θ_2 around the z-axis and translation by l_2 on x-y plane.

$$T_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & l_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & l_2\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Thus, to get the overall transformation, from the base frame to the tool frame, we combine the transformation T_1 and T_2.

$$T_3 = T_1 \times T_2 = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & (l_2\cos\theta_1) \\ 0 & 1 & 0 & 0 \\ -\sin\theta_1 & 0 & \cos\theta_1 & (l_1\sin\theta_1) \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & (l_2\cos\theta_2) \\ \sin\theta_2 & \cos\theta_2 & 0 & (l_2\sin\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_1 \cos\theta_2 & -\cos\theta_1 \sin\theta_2 & \sin\theta_1 & l_2 \cos\theta_1 \cos\theta_2 + l_2 \cos\theta_1 \\ \sin\theta_2 & \cos\theta_2 & 0 & l_2 \sin\theta_2 \\ -\sin\theta_1 \cos\theta_2 & \sin\theta_1 \sin\theta_2 & \cos\theta_1 & -l_2 \sin\theta_1 \cos\theta_2 + l_1 \sin\theta_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

=

Answer 2

The world was constructed by adding a road plane model within Gazebo. The following steps were performed to ensure the environment suited the training and testing requirements:

1. **Road Plane Model**: A road plane model was added to the world to simulate a track for the robot to follow. The ground plane was removed to ensure the robot remained on the road plane.
2. **Camera Setup**: The robot was equipped with a forward-facing camera that captured the scene ahead as it moved along the road. This camera feed was downsampled to 28x28 grayscale images to reduce data size and processing overhead.
3. **Data Capture**: Images were captured based on key commands representing specific actions: "go straight," "turn left," and "turn right."

The resulting dataset contained approximately 20,000 images with approximately 85 MB of disk space, and the distribution as follows:

- **Forward**: Most images were collected for the "forward" motion, as the robot's primary trajectory followed the center of the road.
- **Left and Right Turns**: Roughly 5,500 images each were captured for "turn left" and "turn right" motions.

The training process involved using a convolutional neural network (CNN) architecture to classify images into three categories: go straight, turn left, and turn right. The default model architecture included:

- **Two Convolutional Layers**: The first layer had 20 filters of size 5x5, and the second layer had 50 filters of size 5x5. Each convolutional layer was followed by ReLU activation and 2x2 max-pooling layers.
- **Fully Connected Layer**: A dense layer with 500 neurons and ReLU activation.
- **Output Layer**: A softmax output layer for multi-class classification into three classes.

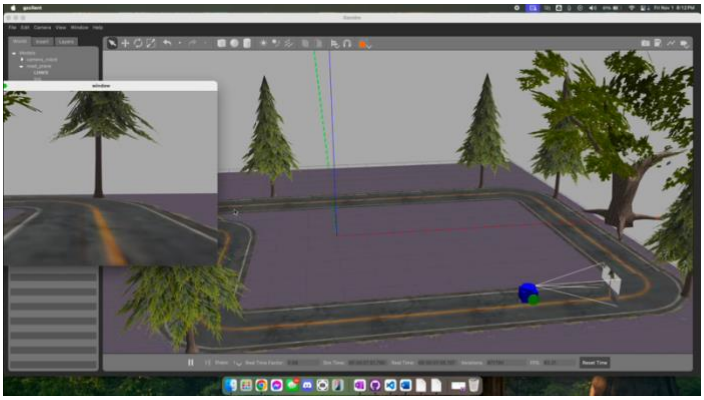
The initial tests were conducted in an obstacle-free environment:

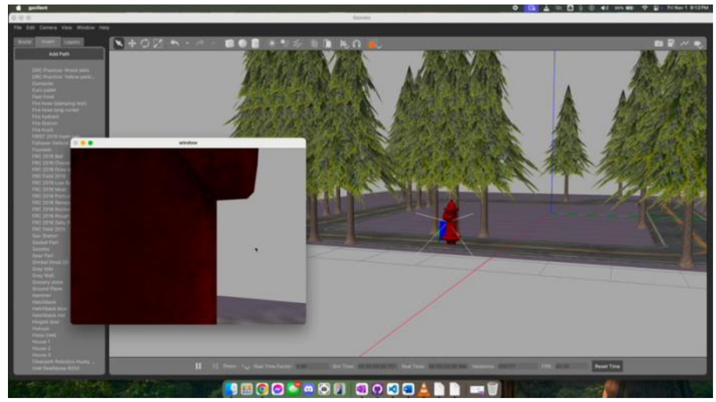
- The trained model successfully navigated the road without veering off the path. Please refer to the attached video for this demonstration.

To test the model's robustness, I introduced pine and oak trees along the roadside, sometimes with roots or obstacles directly in the robot's path. In these cases:

- The robot performed well as long as obstacles were not directly in its path.

- When obstacles (tree roots or a fire hydrant) were placed directly in front, the model failed to navigate around them, resulting in crashes. Thus, the model is sensitive to the path being clear and may not generalize to situations with unexpected objects directly on the road.

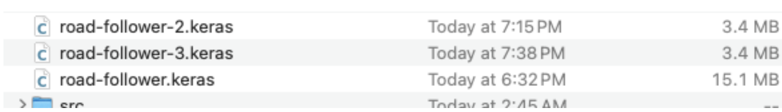




I then experimented with several modifications to the model architecture to improve performance and reduce model size (road-follower-2.py and road-follower-3.py):

1. **Added Dropout Layers**: Dropout was added after the fully connected layer to help prevent overfitting.

2. **Reduced Convolutional Filter Size**: Reducing the filter size and number slightly decreased the model size without significantly impacting performance. The default model was approximately 15.1 MB whereas the resulting models with my architecture modifications were approximately 3.4 MB.



road-follower-2.keras	Today at 7:15 PM	3.4 MB
road-follower-3.keras	Today at 7:38 PM	3.4 MB
road-follower.keras	Today at 6:32 PM	15.1 MB
src	Today at 2:45 AM	--

3. **Increased Depth for Feature Extraction**: I increased the depth of convolutional layers in another variation to capture more detailed features. This model achieved better performance in obstacle-free environments but was less efficient in terms of processing time.

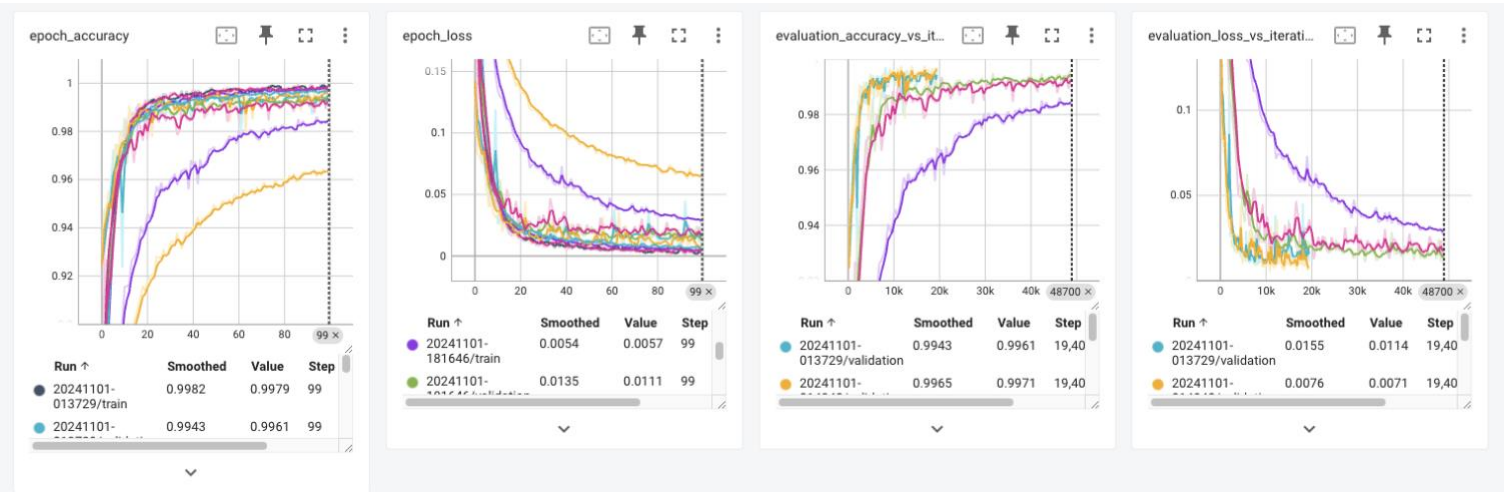
Each model was evaluated based on the robot's ability to follow the road accurately in Gazebo simulations, both with and without obstacles. The final model configurations were as follows:

- **Model 1**: Basic architecture, larger in size, performed well in obstacle-free environments.

- **Model 2**: Increased depth; good performance but slightly higher computational cost.

- **Model 3:** Dropout layers added; reduced model size and slightly better generalization

Screenshots of the training process for all the models:

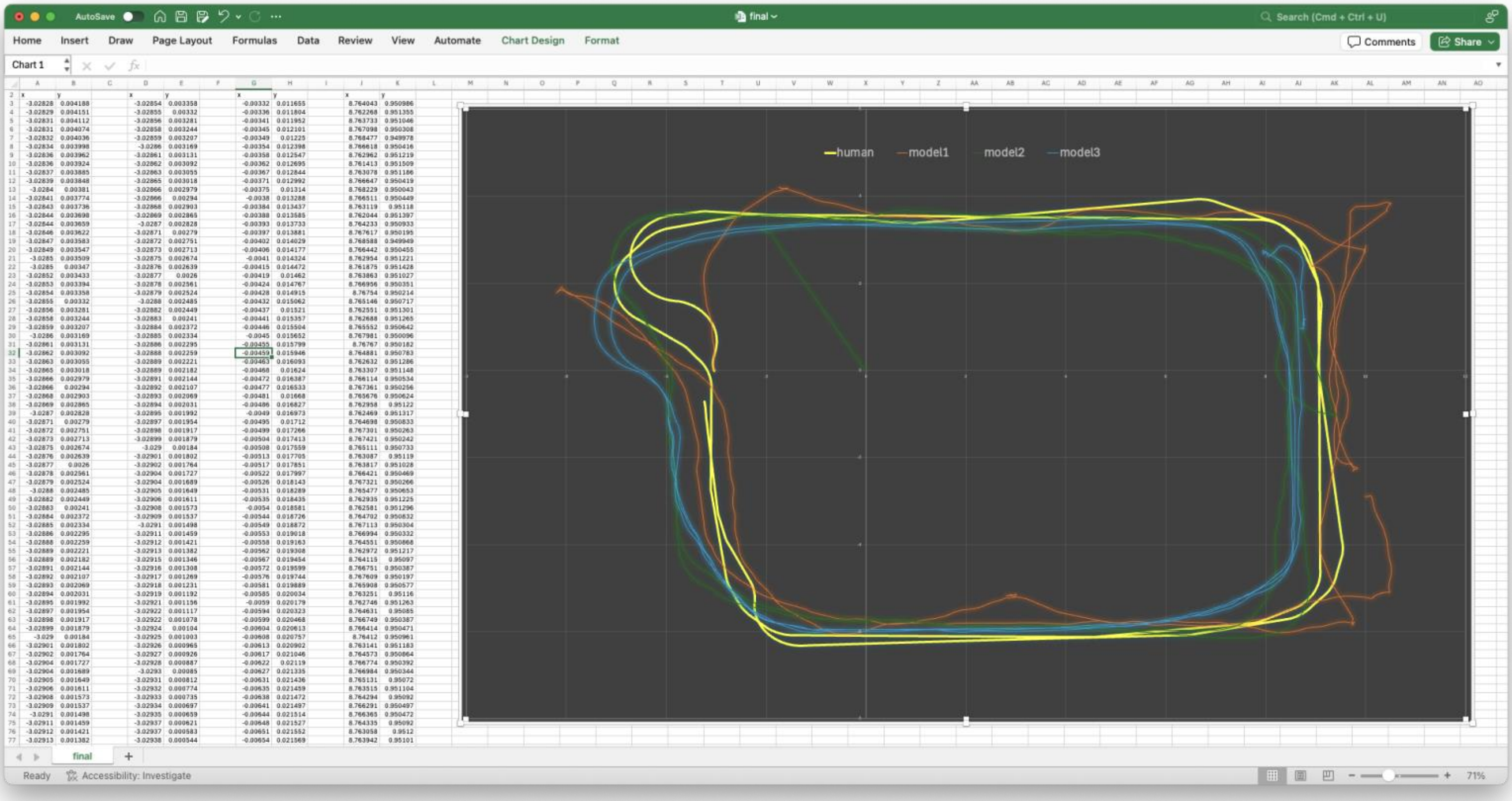


5. Performance Evaluation

To objectively evaluate each model's performance, I tracked the robot's pose using odometry data while the robot followed the track in both human-controlled and model-controlled runs. The odometry data captured the robot's position (x and y axis) over time. By plotting these trajectories on an excel sheet, I compared how closely the models replicated the human-controlled path. Please see screenshot below.

Human-Controlled Path: Served as the ground truth for evaluating deviations in the model's path.

Model Paths: Deviations were observed for each model, especially when encountering obstacles or sharper turns. Model 3 had the best overall performance but at a slightly higher computational cost. By testing and modifying architectures, I found that adding dropout layers and adjusting convolutional parameters could improve generalization and reduce model size without a significant performance drop. It is evident that model3 performed closely to that of manual human driving which was set as the ground truth.


AutoSave
final
Search (Cmd + Ctrl + U)
Home
Insert
Draw
Page Layout
Formulas
Data
Review
View
Automate
Chart Design
Format
Comments
Share
Chart 1
human
model1
model2
model3
final
Ready
Accessibility: Investigate
71%

Below are the rest of the code as follows:

Code for auto_drive_by_road.py

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # disable information messages
from packaging import version
import tensorflow as tf
from tensorflow.keras.utils import img_to_array
from tensorflow.keras.models import load_model
import math
import numpy as np

import cv2
from cv_bridge import CvBridge, CvBridgeError
import rclpy
from rclpy.node import Node
from rclpy.parameter import Parameter
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Image

class AutoDriveByLine(Node):
    def __init__(self):
        super().__init__('drive_by_line')
        self.get_logger().info(f'{self.get_name()} created')

        self.declare_parameter('image', "/mycamera/image_raw")
        self.declare_parameter('cmd', "/cmd_vel")
        self.declare_parameter('odom', "/odom")
        self.declare_parameter('rate', 10)

        # Added more models
        #self.declare_parameter('model', "road-follower.keras") # added extension .keras
        #self.declare_parameter('model', "road-follower-2.keras") # added extension .keras
        self.declare_parameter('model', "road-follower-3.keras") # added extension .keras
```

```python
        self.declare_parameter('x_vel', 1.0)
        self.declare_parameter('theta_vel', 1.0)
        self.declare_parameter('image_size', 28)


        self._image_topic = self.get_parameter('image').get_parameter_value().string_value
        self._cmd_topic = self.get_parameter('cmd').get_parameter_value().string_value
        self._rate = self.get_parameter('rate').get_parameter_value().double_value
        self._model = load_model(self.get_parameter('model').get_parameter_value().string_value)
        self._x_vel = self.get_parameter('x_vel').get_parameter_value().double_value
        self._theta_vel = self.get_parameter('theta_vel').get_parameter_value().double_value
        self._image_size = self.get_parameter('image_size').get_parameter_value().integer_value

        self.create_subscription(Image, self._image_topic, self._image_callback, 1)
        self._pub = self.create_publisher(Twist, self._cmd_topic, 1)

        self._bridge = CvBridge()
        self._auto_driving = False

# Changes by Mahfuz Rahman

        self._odom_data = []  # List to store odometry data
        self.create_subscription(Odometry, self.get_parameter('odom').get_parameter_value().string_value,
self._odom_callback, 1)


    def _odom_callback(self, msg):
        # Capture position and orientation from the odometry message
        position = msg.pose.pose.position
        orientation = msg.pose.pose.orientation
        self._odom_data.append((position.x, position.y, orientation.z))

    def save_odom_data(self, filename):
        with open(filename, 'w') as f:
            for pos in self._odom_data:
                f.write(f"{pos[0]}, {pos[1]}\n")
        self.get_logger().info(f"Odometry data saved to {filename}")

    def _image_callback(self, msg):
```

```
    image = self._bridge.imgmsg_to_cv2(msg, "bgr8")
    cv2.imshow('window', image)
    key = cv2.waitKey(3)

    if self._auto_driving:
        if key == 125:
            self.get_logger().info(f"Auto driving ending")
            self._auto_driving = False
        image = cv2.resize(image, (self._image_size, self._image_size))
        im = img_to_array(image)
        im = np.array(im, dtype="float") / 255.0
        im = im.reshape(-1, self._image_size, self._image_size, 3)
        id = np.argmax(self._model.predict(im))
        if id == 0:
            self.turn_left()
        elif id == 1:
            self.go_straight()
        else:
            self.turn_right()
    else:
        if key == 106:
            self.turn_left()
        elif key == 107:
            self.go_straight()
        elif key == 108:
            self.turn_right()
        elif key == 32:
            self.stop()
        elif key == 113:
            self.get_logger().info(f"Closing node")
            exit(0)
        elif key == 120:
            self.get_logger().info(f"Auto driving starting")
            self._auto_driving = True


def _command(self, x_vel, theta_vel):
    twist = Twist()
```

```
        twist.linear.x = x_vel
        twist.angular.z = theta_vel
        self._pub.publish(twist)

    def go_straight(self):
        self._command(self._x_vel, 0.0)

    def turn_left(self):
        self._command(self._x_vel, self._theta_vel)

    def turn_right(self):
        self._command(self._x_vel, -self._theta_vel)

    def stop(self):
        self._command(0.0, 0.0)

def main(args=None):
    rclpy.init(args=args)
    node = AutoDriveByLine()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

    node.save_odom_data("database.csv")
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Code for road-follower-2.py (Added a Convolutional layer)

```
# Changes by Mahfuz Rahman
# Additional Convolutional Layer
```

```python
#
# This network is based on the Line Follower Robot using CNN by Nawaz Ahmad
# towardsdatascience.com
#
from packaging import version
import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import img_to_array, to_categorical
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard
from imutils import paths
import numpy as np
import argparse
import random
import cv2
import os
from datetime import datetime

from sklearn.model_selection import train_test_split


class LeNet:
  @staticmethod
  def build(width, height, depth, classes):
    # initialize the model
    model = Sequential()
    inputShape = (height, width, depth)
# first set of CONV => RELU => POOL layers
    model.add(Conv2D(20, (5, 5), padding="same",
      input_shape=inputShape))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
# second set of CONV => RELU => POOL layers
    model.add(Conv2D(50, (5, 5), padding="same"))
```

```
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    # changes my Mahfuz Rahman
    # Additional Convolutional Layer
    model.add(Conv2D(50, (3, 3), padding="same"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# first (and only) set of FC => RELU layers
    model.add(Flatten())
    model.add(Dense(500))
    model.add(Activation("relu"))
# softmax classifier
    model.add(Dense(classes))
    model.add(Activation("softmax"))
# return the constructed network architecture
    return model


dataset = './trainImages/'
# initialize the data and labels
print("[INFO] loading images...")
data = []
labels = []

# grab the image paths and randomly shuffle them
imagePaths = sorted(list(paths.list_images(dataset)))
random.seed(42)
random.shuffle(imagePaths)
# loop over the input images
for imagePath in imagePaths:
    # load the image, pre-process it, and store it in the data list
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (28, 28))
    image = img_to_array(image)
    data.append(image)
# extract the class label from the image path and update the
```

```
    # labels list
    label = imagePath.split(os.path.sep)[-2]
    print(label)
    if label == 'left':
        label = 0
    elif label == 'forward':
        label = 1
    else:
        label =2
    labels.append(label)
# scale the raw pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)

# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data,
    labels, test_size=0.25, random_state=42)
# convert the labels from integers to vectors
trainY = to_categorical(trainY, num_classes=3)
testY = to_categorical(testY, num_classes=3)

logdir = "logs/scalars/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=logdir)


# initialize the number of epochs to train for, initial learning rate,
# and batch size
EPOCHS = 100
INIT_LR = 1e-3
BS = 32
# initialize the model
print("[INFO] compiling model...")
model = LeNet.build(width=28, height=28, depth=3, classes=3)
opt = Adam(learning_rate=INIT_LR)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])
```

```
# train the network
print("[INFO] training network...")
H = model.fit(trainX, trainY, batch_size=BS,
    validation_data=(testX, testY),# steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS, verbose=1,
    callbacks=[tensorboard_callback])

# save the model to disk
print("[INFO] serializing network...")
model.save("model2.keras")
```

Code for road-follower-3.py (Added dropout layer):

```
# Changes by Mahfuz Rahman
# Additional Convolutional Layer and Dropout layers


#
# This network is based on the Line Follower Robot using CNN by Nawaz Ahmad
# towardsdatascience.com
#
from packaging import version
import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Activation, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import img_to_array, to_categorical
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.layers import Dropout # new
from imutils import paths
import numpy as np
import argparse
import random
import cv2
```

```
import os
from datetime import datetime

from sklearn.model_selection import train_test_split


class LeNet:
  @staticmethod
  def build(width, height, depth, classes):
    # initialize the model
    model = Sequential()
    inputShape = (height, width, depth)

# changes my Mahfuz Rahman

# first set of CONV => RELU => POOL layers
    model.add(Conv2D(20, (5, 5), padding="same",
      input_shape=inputShape))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    model.add(Dropout(0.25))  # Added dropout layer

# second set of CONV => RELU => POOL layers
    model.add(Conv2D(50, (5, 5), padding="same"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    model.add(Dropout(0.25))  # Added dropout layer


    # Additional Convolutional Layer
    model.add(Conv2D(50, (3, 3), padding="same"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Dropout(0.25))  # Added dropout layer


# first (and only) set of FC => RELU layers
```

```
    model.add(Flatten())
    model.add(Dense(500))
    model.add(Activation("relu"))
    model.add(Dropout(0.5))  # Added dropout layer

# softmax classifier
    model.add(Dense(classes))
    model.add(Activation("softmax"))
# return the constructed network architecture
    return model


dataset = './trainImages/'
# initialize the data and labels
print("[INFO] loading images...")
data = []
labels = []

# grab the image paths and randomly shuffle them
imagePaths = sorted(list(paths.list_images(dataset)))
random.seed(42)
random.shuffle(imagePaths)
# loop over the input images
for imagePath in imagePaths:
    # load the image, pre-process it, and store it in the data list
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (28, 28))
    image = img_to_array(image)
    data.append(image)
# extract the class label from the image path and update the
    # labels list
    label = imagePath.split(os.path.sep)[-2]
    print(label)
    if label == 'left':
        label = 0
    elif label == 'forward':
        label = 1
    else:
```

```
        label =2
    labels.append(label)
# scale the raw pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)

# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data,
    labels, test_size=0.25, random_state=42)
# convert the labels from integers to vectors
trainY = to_categorical(trainY, num_classes=3)
testY = to_categorical(testY, num_classes=3)

logdir = "logs/scalars/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=logdir)


# initialize the number of epochs to train for, initial learning rate,
# and batch size
EPOCHS = 100
INIT_LR = 1e-3
BS = 32
# initialize the model
print("[INFO] compiling model...")
model = LeNet.build(width=28, height=28, depth=3, classes=3)
opt = Adam(learning_rate=INIT_LR)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])

# train the network
print("[INFO] training network...")
H = model.fit(trainX, trainY, batch_size=BS,
    validation_data=(testX, testY),# steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS, verbose=1,
    callbacks=[tensorboard_callback])

# save the model to disk
```

```
print("[INFO] serializing network...")
model.save("model3.keras")
```