Lab 3 Report
Deep Learning for Robotics

EECS 4421, Intro. to Robotics

Course Instructor: Michael Jenkin

March 22, 2021

Group Composition

Ramavath, Sai Varun	varun333@my.yorku.ca, 213506936
Patel, Henilkumar	henil@my.yorku.ca, 215245707
Rego, Jared	jaredr14@my.yorku.ca, 214843874

4th Year
Electrical Engineering
and Computer Science

Contents

List of Figures

Listings

1 Train the network and generate a model

Training the network can be accomplished by following the steps outlined below:

1. drive robot around roadway,
2. record images of the robot's path,
3. record the movement command sent to the robot for each image,
4. label the images by saving in a folder named forward, left or right based on direction,
5. generate labels based on directory, feed to neural network and observe accuracy.

1.1 Drive the robot

The first step can be achieved by creating a teleoperator node that publishes Twist messages to the robot's cmd_vel topic, as seen in Listing 1. The cv2 library is used in this step to generate a live feed of the robot's camera; any movement commands must be typed while this window is in focus. The depressed key is recorded into a 'key' variable which is then polled to determine the appropriate teleop command to publish. The direction commands are seen in Listing 1, and are standard Twist messages that publish a pre-determined (customizable) speed to each of the x, y and theta channels. The interface shown here has an added function of starting autonomous driving by pressing the 's' key, but this key is mapped to start recording training data for this section.

```
    def _image_callback(self, msg):
        # Initialize the image bridge and display the current camera feed in a window.
        image= self._bridge.imgmsg_to_cv2(msg, "bgr8")
        cv2.imshow("Image", image) # Display the image in a window.
        key = cv2.waitKey(3)

        # An interface to send commands via keyboard input.
        if key == 106:
          self.turn_left()
        elif key == 107:
          self.go_straight()
        elif key == 108:
          self.turn_right()
        # Start autonomous driving with a keyboard input of 's', stop with ' '.
        elif key == 115:
          print(f"Autonomous driving activated!")
          self._autonomous = True
        elif key == 32:
          self._autonomous = False
          self.stop()
          print(f"Autonomous driving deactivated!")
    ...
    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)

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

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

Listing 1: The code in follow_road.py used to perform manual driving.

The roadway was built using a .dae file that was placed in the .gazebo/models folder found in the home directory. The road was created with curbs on either side and a clear center marking for three quarters of the circuit, while one quarter of it was built with gray curbs and no center marking to test the robustness of the neural network; the robot was trained on the 'clear' parts of the roadway.

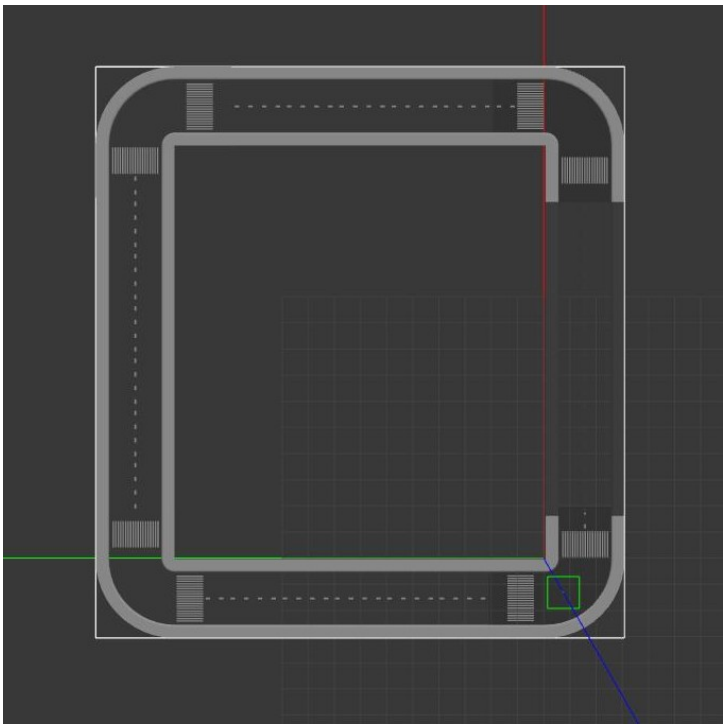


Figure 1: Roadway built to test differential drive robot's autonomous driving capabilities.

1.2 Collect training data

The next three steps use the CV2 Python library to record the robot's current path using the onboard camera. The images are separated into forward, left and right directories in a folder called 'dataset' that is generated in the 'scripts' directory of the cpmr_ch6 project provided for this lab. The code in Listing 2 shows how the standard cpmr_ch6 directory achieved this.

```
    if self._recording and (self._curdir != None):
      dim = min(image.shape[0], image.shape[1])
      oy = int((image.shape[0] - dim) / 2)
      ox = int((image.shape[1] - dim) / 2)
      roi = image[oy:oy+dim, ox:ox+dim, 0:3]
      out = cv2.resize(image, (self._image_size, self._image_size))
      cv2.imshow("Resized", out)
      if self._curdir == DriveByCamera._FORWARD:
        cv2.imwrite(f"{self._output}/forward/image.{self._forward_id:08d}.jpg", out)
        self._forward_id = self._forward_id + 1
      elif self._curdir == DriveByCamera._TURNING_LEFT:
        cv2.imwrite(f"{self._output}/left/image.{self._left_id:08d}.jpg", out)
        self._left_id = self._left_id + 1
      elif self._curdir == DriveByCamera._TURNING_RIGHT:
        cv2.imwrite(f"{self._output}/right/image.{self._right_id:08d}.jpg", out)
        self._right_id = self._right_id + 1
```

Listing 2: The code used for recording training images from the robot's camera feed.

1.3 Process images for training

The extracted images need to be processed into a usable size to ensure the neural network does not take excessive amounts of time to process every image. The training dataset created for the network had 500+ images for each of the forward, left and right directions which were subsampled down to 28x28 images for this purpose. Listing 3 shows how this was implemented in the standard cpmr_ch6 directory. The important thing to note here is the way that labels are enumerated: forward is given a value of 0, left a value of 1, right a value of 2; this will be used in the autonomous driving script to determine the predicted optimal direction based on camera input.

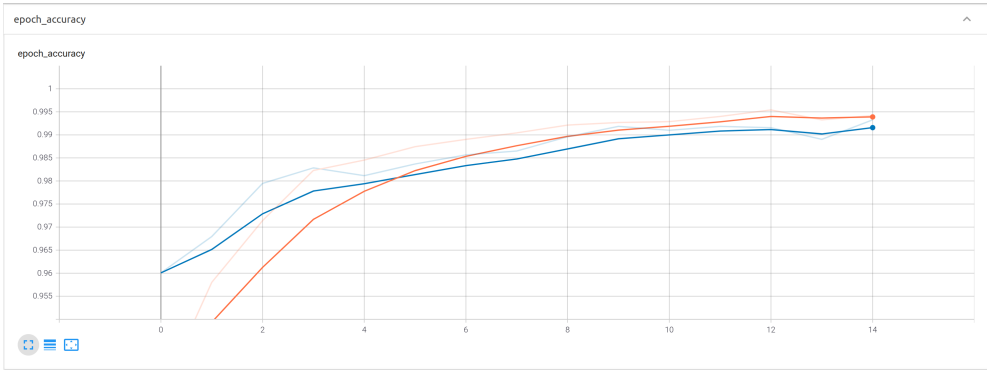
```
for imagePath in imagePaths:
    # load the image, pre-process it, and store it in the data list
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (28, 28))
    image = img_to_array(image)
    data.append(image)
# extract the class label from the image path and update the
    # labels list
    label = imagePath.split(os.path.sep)[-2]
    print(label)
    if label == 'forward':
        label = 0
    elif label == 'right':
        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)
```

Listing 3: The code used for processing training images and generating their labels.

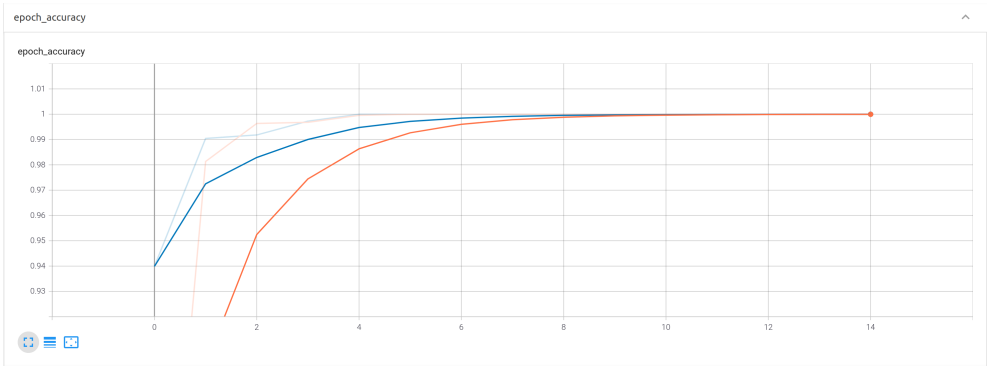
1.4 Training results

Before the training results are provided, it is useful to get an understanding of how the training data was generated; i.e. at which points was the robot instructed to turn or go forward. The speed published to the x, y and theta channels was set to 0.2 m/s and 0.2 rad/s respectively. Initially, the robot was instructed to turn only at the bends in the road, while any small changes needed to correct the robot on the straight sections were not included in the recording. While this worked and provided a high model accuracy, the robot was unable to make those necessary corrections when deviating from the center of the straight sections causing a collision with the curb. The final model includes these corrections in the training data to ensure the robot can handle edge cases well, and as will be seen in the balanced set model results (Section 3), this was quite effective in steering the robot away from the curbs.

The models generated using the unbalanced and balanced training sets showed high accuracy (in the 99% range) which was quite promising before the autonomous driving was tested. The epoch accuracy for both cases is shown in Fig. 2; the most significant difference between the two is the number of epochs required to achieve peak accuracy. The unbalanced set achieves around 100%(!) accuracy within far fewer epochs since it has a higher number of forward images that it can successfully classify, whereas the balanced set has the ability to 'confuse' the neural network which shows in the lower accuracy (99.5%) and higher number of epochs taken to achieve peak accuracy. The model was trained with 500+ images for each of the forward, left and right categories.



(a) balanced set



(b) unbalanced set

Figure 2: Model training accuracy for balanced and unbalanced datasets generated using Tensorboard and the log files created by `train.py`.

2 Create autonomous driving script

The CNN created in the previous section predicts the required robot direction by learning which direction the robot was instructed to travel in the training exercise and applying this knowledge to a live stream of images taken by the robot's camera as it traverses the roadway. The code shown in Listing 4 carries out the model prediction and sets a directional flag based on the output from the model. The flag is then polled and the appropriate direction function was called to carry out the driving. The robot's movement is borrowed from the `drive_by_camera.py` script provided with the cpmr_ch6 project directory, and is shown in Listing 1. Two important points to note in this script are: (1) it employs the 'on-off' or open-loop control method, and (2) the constants 'FollowRoad._FORWARD' etc. were rewritten to represent the model's output as seen in Listing 3.

```
        predict = np.argmax(self._model.predict(im))
        if predict == FollowRoad._FORWARD:
          self.go_straight()
        elif predict == FollowRoad._TURNING_LEFT:
          self.turn_left()
        elif predict == FollowRoad._TURNING_RIGHT:
          self.turn_right()
```

Listing 4: Code in follow_road.py used to perform autonomous driving. The script borrows much of the code from `drive_by_camera.py` and `deploy.py` provided in the project directory. The snippet shown here is the main change made to the code to accomadate autonomous driving.

3 Model driving results

The model driving results can be seen in the videos Balanced Dataset and Unbalanced Dataset. As mentioned in 2, the script utilizes an open-loop control method and therefore the driving seen here can be significantly improved by implementing a closed-loop PID controller. An attempt was made to do this, but this was not successful and provided worse results than open-loop control. The PID controller is being worked on currently to verify that this control method would be better for robot control.

A Appendix

The code and road model created for this lab can be found at the *IntroRobotics_York* Github repository.