Deep Turtle Control

Michael Farrell, Skyler Tolman

***Abstract*— abstract here**

I. Introduction

II. Classical Approach

III. Network Approach

The end goal of our project was to get the turtlebot to follow a track purely using an image input to a neural network. This section describes the architecture and training methods used to train a control network for the turtlebot.

A. Network architecture

We experimented with multiple architectures, including a simple convolutional neural network (CNN), and a ResNet [**?**], but we eventually decided to use the DenseNet as our base architecture [**?**].

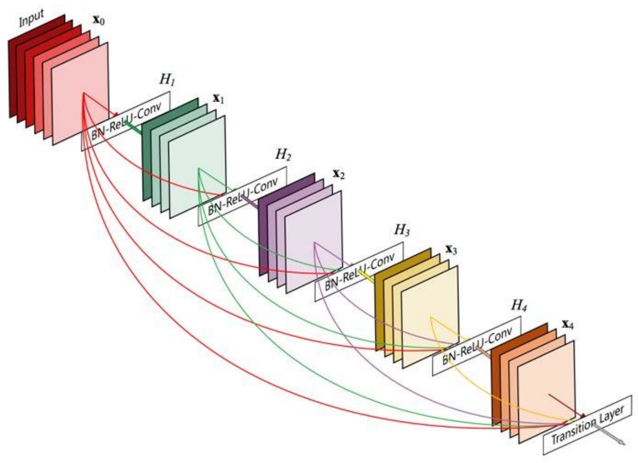

Fig. 1. DenseNet architecture

The Densenet is an network architecture with densely connect convolutional blocks, as seen in Figure 1. The DenseNet architecture is unique because of the densely connected skip connections across multiple convolutional blocks, which allows the architecture to re-use features throughout the network. Accordingly, it has achieved better results on benchmarks such as ImageNet with a smaller memory footprint.

We used a pre-trained model for the DenseNet convolutional blocks, and added three linear layers with dropout to train the turtlebot control using the DenseNet features.

B. Network Training

We used supervised ”imitation” learning to train the control network. We collected data by driving the turtlebot around multiple tracks on different ground surfaces and matched RGB camera images to angular velocity command outputs. We initially used the classical vision control method to gather data on a single track, then diversivied the data by manually driving the turtlebot around other varied tracks and surfaces.

Using the image-command pairs, we trained the neural network to predict the appropriate command for each input image. We trained the network on about 7 epochs of 15,000 images.

C. Continous Control

Ideally, we would like the network to be able to predict a continous control output given an input image. We allowed the network output to range from -1 to 1 radians per second. A simple way to limit the control effort was to use a hyperbolic tangent function as the final activation on our linear layer backend.

After multiple attempts at training the network to predict a continouous output, we were able to get some success, but the turtlebot had a hard time making sharper turns and would often drive straight off the course instead of making the turns. In general, the control was underactuated.

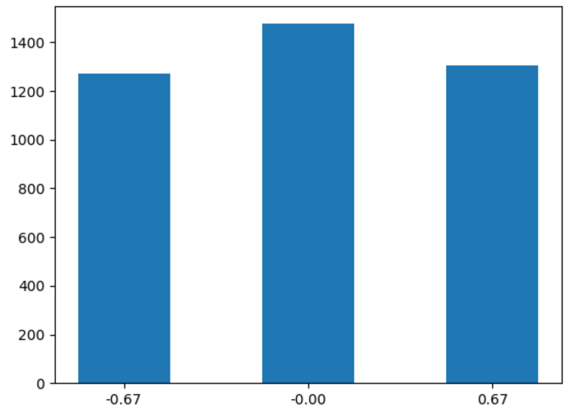

Fig. 2. Histogram of the omega commands for the training data used.

We assume that the issue was a data problem, so we plotted a rough distribution of the data, as seen in Figure 2. The data is centered about mean and has a larger proportion of commands close to zero than it does further away. It seems as though the network was able to quickly decrease the loss by driving the control close to zero, causing the turtlebot to default to slower angular rates. The problem was then magnified because the turtlebot would fail to turn sharply and place itself in a situation that it had not seen before and could not recover from.

To overcome this issue, we decided to try using an architecture with discrete outputs and treat it as a classification problem. Some other potential solutions to our problems could have included augmenting the data to include more off-center scenarios or to gather data with more variety, including the corner cases where the turtlebot needs to recover from poor control or poor initial conditions.

D. *Discretized Control*

As a simple solution to the difficulties we faced when using continous control, we decided to discretize the control by binning the allowable omega values and using the midpoint of each bin as the representative omega value for that bin. We used 5 bins to keep the classification problem simple, but still have enough control authority to navigate sharper turns. With five bins, the allowable control outputs were -0.8 (hard left), -0.4 (left), 0.0 (straight), 0.4 (right), and 0.8 (hard right). Neural networks are often better suited to classification problems and by using a cross-entropy loss function instead of the mean squared error loss we had used for the continous output, the network lost the incentive to center commands near zero, but instead was required to classify each image in the correct command bin.

This approach proved to be much more successful and we were able to replicate and eventually surpass the performance of the classical method.

IV. Results

We were pleased to be able to succesffully implement autonomous control of a turtlebot using only visual data. The classical approach served as a meaningful learning experience and also a useful tool in collecting data to train the neural network approach, which was our end goal.

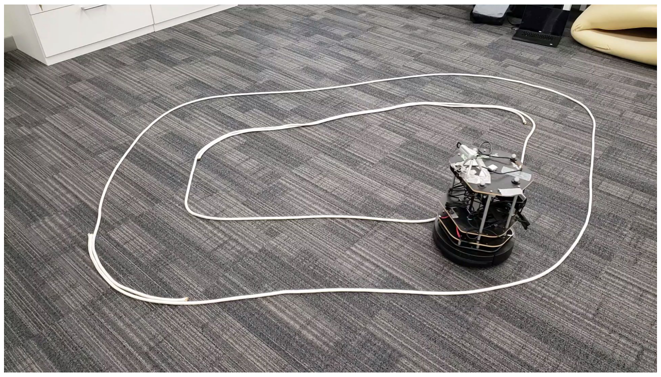

Fig. 3. Snapshot of the turtlebot successfully following a track using end-to-end deep learning-based control.

Some of the main advantages we saw of using the neural network control as opposed to the classical method was the ability to implicitly encode variability and robustness into the control simply by training on more data in varied environments. For example, the classical method, which depended heavily on proper segmentation, needed to be re-tuned for each ground surface and lighting condition, whereas the neural network was able to learn how to extract the proper features in multiple environments without the need to explicitly retune or revise the network.

It was also interesting to note the information the neural network was able to learn. As a simple metric of determing what the network deemed important, we took the gradient of the output with respect to the input image and, after normalizing the image, were able to visualize which pixels contributed most strongly to the output. As seen in Figures 4 through 6, the brightest pixels in the gradient image corresponds to the areas where the rope is seen in the input image. Note specifically how the neural network learned to identify the rope specifically, not just white objects, as the light reflection in Figure 6 is filtered out and does not contribute much to the output. These glare spots were difficult for the classical method to handle, which conveys a clear advantage of the network-based control over the classical control.

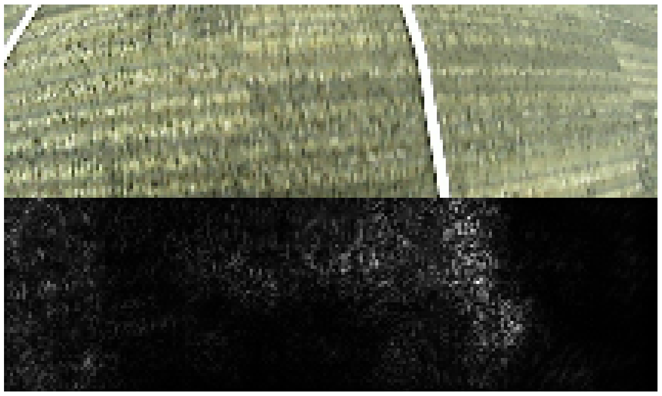

Fig. 4. Top: Original image; bottom: Class saliency for rope on carpet data. Shows the gradient is highest near where the rope appears in the original image (top)

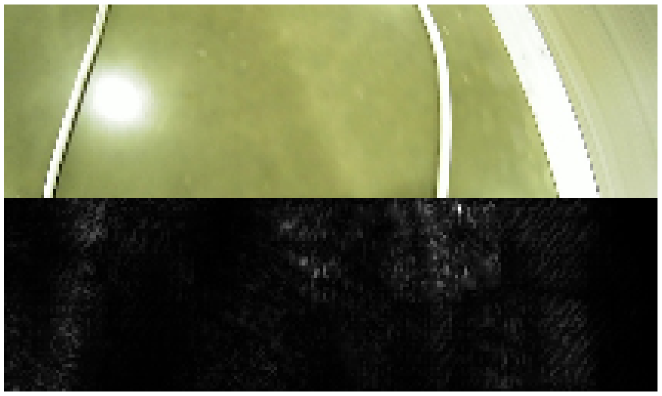

Fig. 5. Class saliency on reflective concrete flooring.

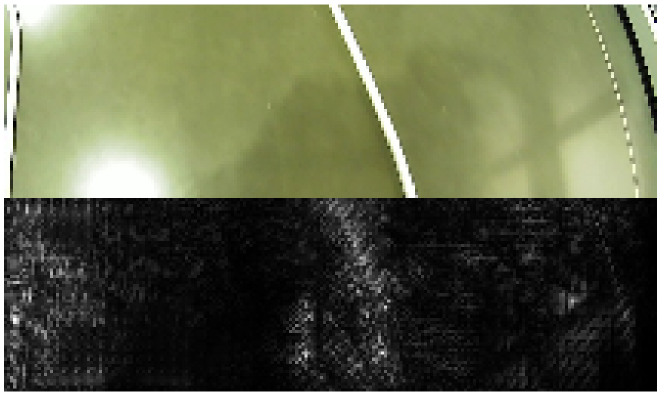

Fig. 6. Class saliency on reflecitve concrete floor with mulptiple glare spots. Show that the network learned to filter out bright contours that do not correspond with rope.

V. Future Work

Although the control was sufficient to navigate a simple track multiple times, there is definitely room for improvement. One main disadvantage of our current method is that the discretized cotrol is somewhat jerky and it could potentially limite the robots ability to make sharper turns on more difficult tracks if the avaiable discrete values were insufficient to make the turn. Some possible improvements to overcome this could be to use more bins to better approximate continuous control, or possibly to better collect and augment the data used to train the continous control output to see if we can produce effective control without discretization.

We would also like to find ways to extend the results of this project beyond the contrived example of following rope lanes to something with real-world usefulness. This could include training the network to use more natural features to navigate such as hallways, aisles, existing tile markings, or sidewalks. This would allow the network to leverage it's full potential to interpret high level features rather than just simple lines. Another extension that could improve the usefulness of this control method could be to integrate the autonomous control with SLAM (simulataneous localization and mapping) or some other mapping algorithm to not only navigate the real world, but also to collect data about the environment along the way.

This turtlebot control project proved to be a useful opportunity for us to apply deep learning and computer vision principles to a non-trivial problem and to gain some real-world experience implementing these technologies on a hardware platform.