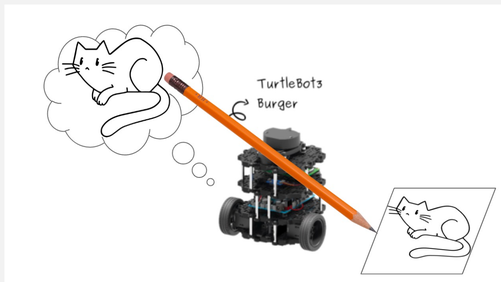
Reproducing a Line Drawing with a Turtlebot3 Sketcher

Team #2 Sujeburger

20180069 Dong Joo Kim, 20180491 Jae Hyun Lee, 20210584 Youngwoon Cheong, 20180645 Yoojin Cho

Project Repository

Problem Definition



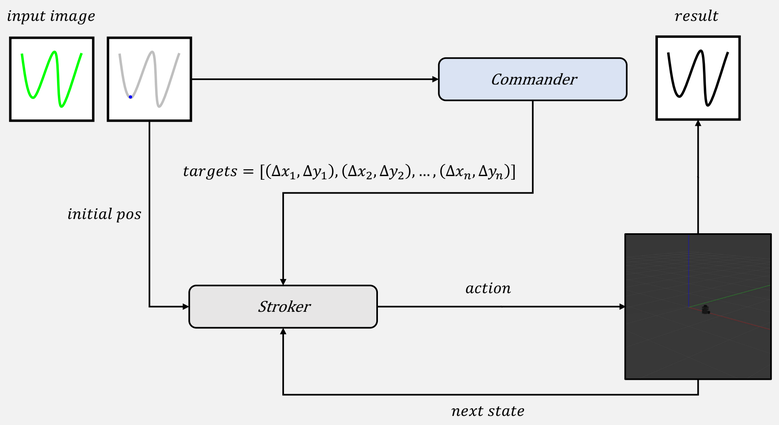
Tracing the given path is one of the most fundamental tasks for modern robots. For example, autonomous vehicles need to follow the traffic precisely. Traditional robot control programs use inverse kinematics, which can be laborious. In this study, we used machine learning to make our robot agent, Turtlebot3, trace the given path. We design model architecture and measured our model performance. Finally, we visualized the model's output to verify the model works well.

Related Work

Previously, the research "From Scratch to Sketch" [1] studied line reproduction using a robot arm with a gripper. They presented a hierarchical framework that integrates a high-level stroker-generating process (Commander) and a low-level robot control process (Stroker) through reinforcement learning. Two models learn separately during the training process. Furthermore, in the deployment process, models are combined such that Commander generates strokes, and the Stroker controls the robot to follow those strokes. In this research, we tried to extend the research by applying the line reproducing to a vehicle.

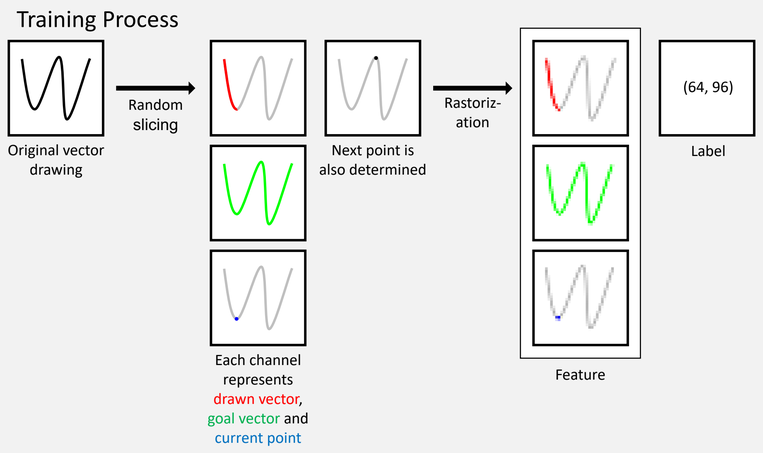
Method

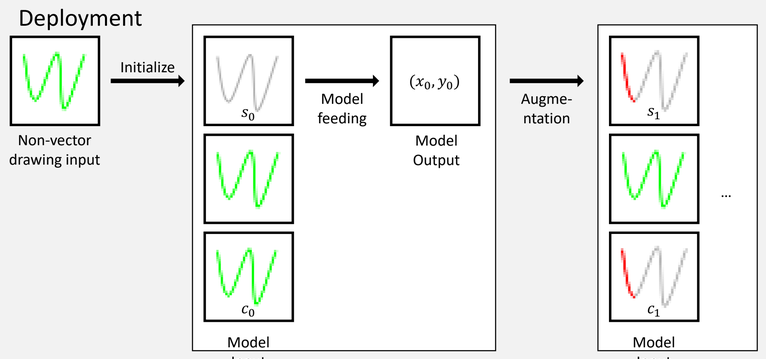
A. Hierarchical Model



Our solution adapts the hierarchical structure, which consists of Commander and Stroker. To illustrate the process, given an input image, the Commander determines which sequences of moving actions would optimally reproduce the given image. This command set is directly forwarded to the Stroker and performs the commands by manipulating the robot's angular and linear velocities so that the Turtlebot can move around the canvas. Both Commander and Stroker are trained independently to prevent dependency between the models.

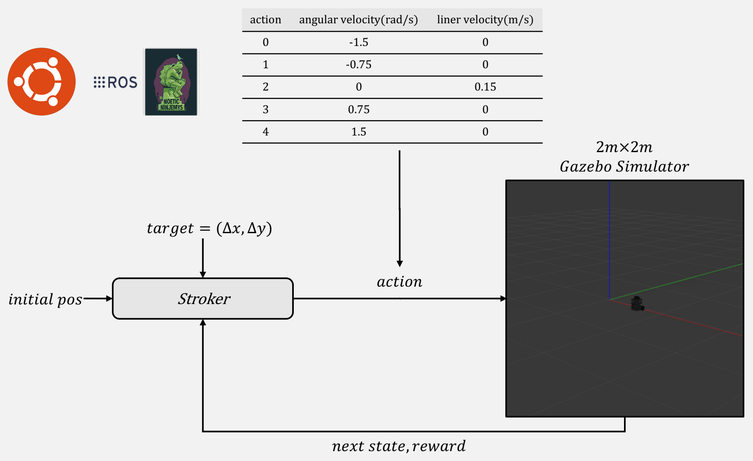
B. Commander





Given the already-drawn image $\overrightarrow{P_1 \cdots P_k}$, the goal image $\overrightarrow{P_1 \cdots P_n}$, and the current location of the robot P_k, Commander was trained to predict the next point to move P_{k+1}. In deployment, Commander is asked to find all points that the robot should go through to reproduce the goal image. The model starts from three initial inputs: the goal image, an empty image which only includes the starting point, and the current point. It continues to find next point to move until it reaches near the goal point or it steps over the step number limit.

C. Stroker



action	angular velocity(rad/s)	liner velocity(m/s)
0	-1.5	0
1	-0.75	0
2	0	0.15
3	0.75	0
4	1.5	0

We have formulated Stroker's MDP as above. The state is defined as $s = (x_{agent}, y_{agent}, \theta_{agent}, x_{target}, y_{target}, \theta_{target}, v, \omega)$. These features describe the positions of the agent, target, and the most recently chosen action. Stroker's action is the robot agent's linear and angular velocity. which is discretized into several combinations: turn left, go straight, turn right, for convenience. The reward formulation was adopted and modified from the Turtlebot3 navigation tutorial provided by Robotis as below.

$$reward = \begin{cases} 200, \text{if agent has reached the goal} \\ -100, \text{if agent has gone past the goal} \\ -200, \text{if agent goes out of canvas} \\ reward_{rot} \cdot reward_{dist}, \text{otherwise} \end{cases}$$

Briefly, larger rewards are given to actions that align the agent with the target's orientation and move the agent closer to the target.

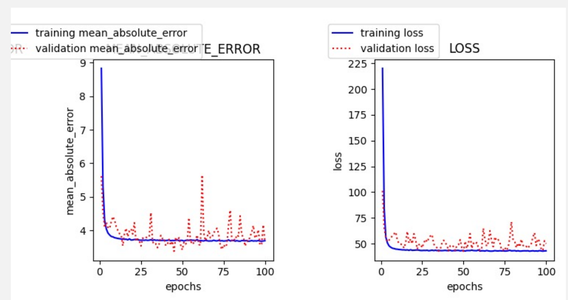
Experiments and Results

A. Training

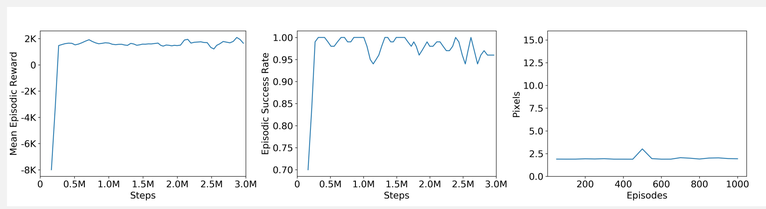
For Commander, we conducted supervised learning using a subset of Google's *Quick, Draw!* Dataset [2], while ResNet [3] is used as a network. Mean-squared error is selected as a loss function, while mean-absolute error is selected as the metric, representing the average error distance between the predicted and actual points on the bitmap.

For Stroker, we used reinforcement learning with Gazebo environment. DQN implementation provided by stable-baselines3 framework was used to train the Stroker. We used the DQN model architecture presented by Mnih, V et al [4]. We randomly generated the agent's orientation and target position in the training process on each reset of the environment.

B. Component Results

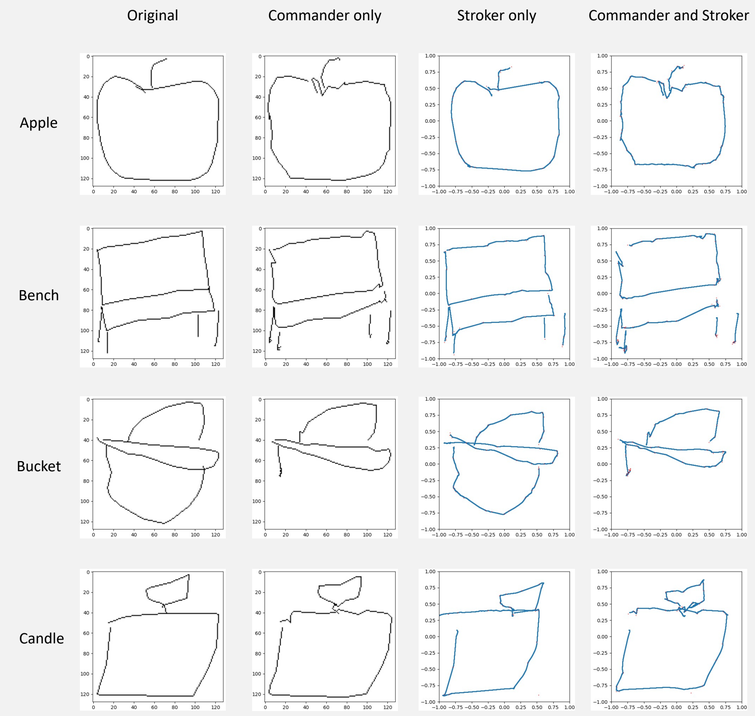


The above plots illustrate the measured metric of the Commander during the training process. The average training MAE is lower than 4, meaning that the predicted and label points have an error of 4 pixels on average in 128 x 128 size imaginary canvas.



The above plots illustrate the measured metric of the Stroker during the training process. Mean episode reward increases as training proceeds while most learnings were done in the first 250K steps. The episodic success rate, the proportion of episodes that terminated by reaching the target, is approximately 100% after training. Pixel difference, the Euclidian distance between the agent's final position and target position in pixels, is within only 2 pixels.

C. Integration Results



Since the resemblance between input and output drawings is hard to define, instead, we have visualized some of the results. Here, we compare carefully-selected four drawings. From left to right, each image represents input, Commander-reproduced, Stroker-reproduced, and integrated model-reproduced images.

Future Works

Several future advancements can be made in this study. To begin with, when the input image is complicated or has a cross between the strokes, the Commander model shows poor performance. This problem is possibly caused by the limitation of the model, ResNet, which cannot preserve past contexts. This problem is likely to be solved by adopting a Transformer-like model. Furthermore, we used discretized action space for convenience, but for delicate control, continuous action space is needed. This can be achieved by implementing another RL method than DQN, such as soft actor-critic. Moreover, real-time integration of Commander and Stroker is required. In the current process, Stroker can operate after every Commander's calculation ends. This process can be pipelined to work simultaneously.

Citations

[1] Lee, G., Kim, M., Lee, M., & Zhang, B. (2022). From Scratch to Sketch: Deep Decoupled Hierarchical Reinforcement Learning for Robotic Sketching Agent. arXiv. https://doi.org/10.1109/ICRA46639.2022.9811858

[2] quickdraw.withgoogle.com

[3] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. arXiv. https://doi.org/10.48550/arXiv.1512.03385

[4] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236