Survey of Various Learning Methodologies for Robotic Assembly Tasks

Siddhant Keshkar
Computer Science
University of Colorado Boulder
sisu7408@colorado.edu

***Abstract*—The purpose of this report is two-fold. First of all, It discusses the WRS 2020 assembly challenge. Then it looks into the currently implementation methods using behavior trees and then suggested methodologies for learning in behavior trees. The second part of the paper presents a set of algorithms that can be used for improving robotic assembly tasks using reinforcement learning, imitation learning, neural networks and deep reinforcement learning. Finally, the performance of these algorithms are compared using the OpenAI Fetch pick up and place environment.**

***Index Terms*—Behavior trees, Reinforcement learning, Imitation learning, Neural Networks, Deep Reinforcement Learning**

I. Introduction

Over the last decade, Behavior Trees (BTs) have become a very important tool in the design of game AI, and are widely appreciated for their modularity and reactivity. BTs provide a more intuitive approach than previous techniques such as hierarchical state machines, which often required complex data structures producing poorly structured code when scaled up.

A challenge for robotics, is the need to engineer a reward function that not only reflects the task at hand but is also carefully shaped to guide the policy optimization. The necessity of cost engineering limits the applicability of Reinforcement Learning in the real world because it requires both RL expertise and domain-specific knowledge. But, it is not applicable in situations where we do not know what admissible behaviour may look like. It is therefore of great practical relevance to develop algorithms which can learn from unshaped reward signals, e.g. a binary signal indicating successful task completion.

II. Problem: Assembly Challenge

For the WRS 2020 competition, participating teams compete on assembling model products that contain the technical elements necessary for assembling industrial products and other goods as quickly and accurately as possible. In order to respond to new production demands, it is also required to reconfigure the robot systems for the new products in agile and lean manners.

III. Background

Robotic manipulation system used is a programmable and adaptable system for various tasks common to industrial setting and inspired by WRS 2020 Industrial Assembly Challenge [14], which includes an assembly of standard, commercially available industrial parts into 2D and 3D assemblies. Model workflow we used can seen in the figure 1. Model includes central code repository represented by RM Studio that combines UR5 Arm control code, Opto-force sensor code and Smart Hand Gripper control code. We discuss the various learning methods that can be used for robotic assembly tasks.

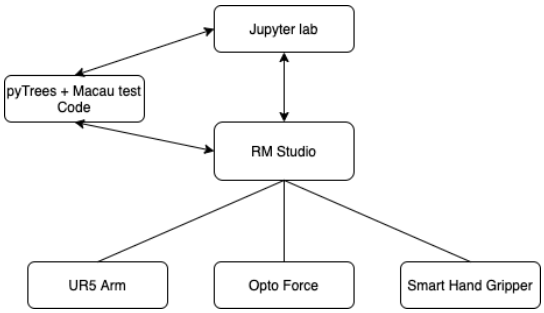


Fig. (1) Code Infrastructure for assembly challenge

A. Behavior Trees

A BT is a directed rooted tree where the internal nodes are called control flow nodes and leaf nodes are called execution nodes. For each connected node we use the common terminology of parent and child. The root is the node without parents; all other nodes have one parent. The control flow nodes have at least one child. [5]

A BT starts its execution from the root node, that generates signals called Ticks with a given frequency. These signals allow the execution of a node and are propagated to one or several of the children of the ticked node. A node is executed if and only if it receives Ticks. The child immediately returns Running to the parent, if its execution is under way, Success if it has achieved its goal, or Failure otherwise.

In the classical formulation, there exist two main categories of control flow nodes (Sequence and Fallback) and two categories of execution nodes (Action and Condition).

The Sequence node routes the Ticks to its children from the left until it finds a child that returns either Failure or Running, then it returns Failure or Running accordingly to its own parent. It returns Success if and only if all its children return Success. Note that when a child returns Running or

Failure, the Sequence node does not route the Ticks to the next child (if any). The symbol of the Sequence node is a box containing the label "→".

The Fallback node routes the Ticks to its children from the left until it finds a child that returns either Success or Running, then it returns Success or Running accordingly to its own parent. It returns Failure if and only if all its children return Failure. Note that when a child returns Running or Success, the Fallback node does not route the Ticks to the next child (if any). The symbol of the Fallback node is a box containing the label "?".

When an Action node receives Ticks, it executes a command such as moving the agent. It returns Success if the action is successfully completed or Failure if the action has failed. While the action is ongoing it returns Running.

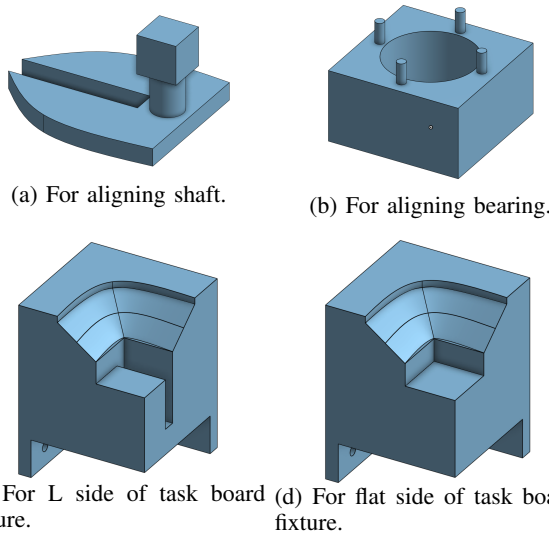
When a Condition node receives Ticks, it checks a proposition. It returns Success or Failure depending on if the proposition holds or not. Note that a Condition node never returns a status of Running.

IV. Implementation for WRS 2020

We looked through the previous challenge code and found that it was lacking scalability, reusability, and parallelization. The tasks for the assembly challenge can be divided into three parts: First, Part recognition and grasping, Second, Parts fitting and nut screwing and Third, flexible part assembly that need to be performed in a sequence. Hence, we implemented the grasping and fitting tasks using behavior tree. The behavior tree implementation was done with python jupyter labs using the pyTrees library. I started with the analysis and error resolution of tiltInsert and spiralInsert sequence nodes of the current implementation of the behavior tree. Then, I implemented the grasp pick up and place node with the reference of the drill holster and unholster node. These algorithms currently use hard-coded thresholds in the force, torque and location domains to detect critical points in the assembly.

Robotic assembly tasks requires support for stability and alignment of different parts. This is achieved using a fixed bench for base setup and 3d printed custom parts. I have created various 3d models using the onshape online tool to bolster the aligning of bearing with housing, shaft and task assembly board fixture. This was achieved after two to three design iterations till the final product fits the requirements. These designs can be seen in figure 2.

Fig. (2) Final 3D Printed model designs



(a) For aligning shaft.

(b) For aligning bearing.

(c) For L side of task board fixture.

(d) For flat side of task board fixture.

V. Learning in Behavior Trees

A. Neural Networks for Behavior Trees

Safety guarantees of an emergency subtree can be combined with the performance guarantees of a model based subtree and the efficiency of a machine learning subtree. An example for which is shown in the figure 3. If the safety constraint is in risk of being violated the Emergency subtree is invoked. If safety is ok, the BT checks if the current execution time is ok, that is, if there is reason to believe that the learning subtree is not going to complete the task in time, or at all. If the execution time is not ok, the previously designed model based subtree is invoked. Finally, if both of the conditions above are satisfied, the learning subtree is executed. [7] Switching between subtrees like this might induce undesired behaviors where one subtree counteracts another one. Also, this method might lead to deadlocks where the different controllers work against each other.

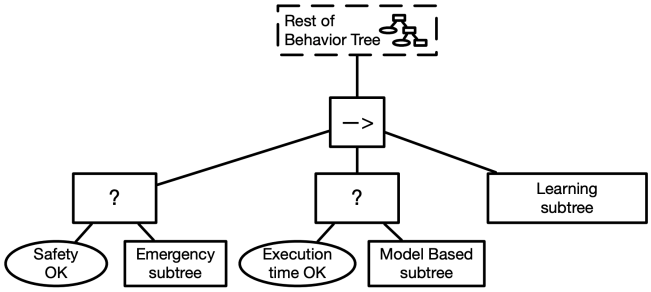


Fig. (3) Emergency subtree for safety, Model based subtree for execution time and Learning subtree when neither run into problems.

B. Q-Learning for behavior trees

Q-Learning is a Reinforcement Learning(RL) algorithm that creates and maintains a table of values which estimate the utility of taking an action in a state. The agent is given a function associating states with a predetermined reward. The algorithm then feeds rewards back to state-action pairs that lead to reward states creating gradually improving utility estimates. The algorithm begins with a BT as input. The tree is analysed to find the deepest Sequence nodes. These nodes are identified as actions for the RL stage. These actions are used in an offline Q-learning phase to generate a Q-value table. The table is then divided into sub-tables by action and the highest valued states for the action are extracted into the Q-Condition nodes within the BT. The Condition nodes in the input BT are

then replaced with the Q-Condition nodes. Finally, the BT's topology is reorganized by sorting each node's child by their maximum Q-value, which provides AI designers with a more optimized permutation of the BT. [1] A drawback of the QL-BT algorithm is its reliance on correct Q-values. The validity of the Q-values relies on an appropriate reward function and an effective learning phase providing accurate utility estimates for each state-action pair. Values can be improved by a longer training period because this is an offline pre-processing step and does not cause performance issues at run-time.

C. Hierarchical Reinforcement Learning with behavior trees

Hierarchical RL decomposes a reinforcement learning problem into a hierarchy of subtasks such that higher-level parent-tasks invoke lower-level child tasks as if they were primitive actions. This method uses a Learning Node which is created with a Composite and an Action node, in which a Q-Learning algorithm is embedded to perform a local learning, without affecting how other nodes work. This framework is related to Hierarchical Reinforcement Learning, being a specialization of the Options framework, thus ensuring convergence of nested learning nodes, allowing them to be interrupted before the task is completed and allowing the use of intra-option learning for more complex models. [3]

VI. Algorithm Designs for robotic manipulation Tasks

A. Hindsight Experience Replay(HER)

Dealing with sparse rewards is one of the biggest challenges in RL. Hindsight Experience Replay(HER) allows sample-efficient learning from rewards which are sparse and binary and therefore avoid the need for complicated reward engineering. It can be combined with an arbitrary off-policy RL algorithm. The key insight that HER formalizes is what humans do intuitively, which is learning that taking an action at a state not corresponding to reaching a specific goal is instead achieving a different one. So if we wanted to achieve this goal to begin with, instead of the one that we set out to achieve originally, By doing this substitution, the reinforcement learning algorithm can obtain a learning signal since it has achieved some goal. If we repeat this process, we will eventually learn how to achieve arbitrary goals, including the goals that we really want to achieve. [4]

B. Demonstration Based Hindsight Experience Replay(D-HER)

This method uses demonstrations to overcome the exploration problem in robotic settings and successfully learn to perform long-horizon, multi-step robotics tasks with continuous control. Because, finding a non-zero reward is exponentially more difficult with increasing task horizon or action dimensionality. Using demonstrations over Deep Deterministic Policy Gradient along with Hindsight Experience Replay, provides a significant speedup over RL on simulated robotics tasks. [9]

C. Reinforcement learning with Behavior Cloning(RLBC)

This Method uses RL approach for task planning which can learn to combine basic skills using behavior cloning. This requires neither intermediate rewards nor complete task demonstrations during training. Method uses vision-based task planning in challenging settings with temporary occlusions and dynamic scene changes. Finally, it undergoes training of basic skills from few synthetic demonstrations by exploring recent CNN architectures and data augmentation. [12]

D. Relational Reinforcement Learning(RRL)

This method uses a graph-based relational architecture that overcomes absence of inductive biases for transferring knowledge from simpler to complex tasks and enables learning of complex tasks when provided with a simple curriculum of tasks with increasing numbers of objects. Despite using step-wise sparse rewards, this method is more data efficient and outperforms the some of the methods that use human demonstrations. [10]

E. Deterministic Generative Adversarial Imitation Learning (DGAIL)

This method integrates deterministic off-policy RL with generative adversarial network. It allows the robot to implement the grasping task rapidly by learning the reward function from the demonstration data. Firstly, the discriminator is used to learn the reward function from demonstrations, which can guide the generator to complete the robot grasping task. Secondly, the deep deterministic policy gradient method is used as the generator for learning action policy on the basis of discriminator. The demonstration data is also input into the generator to ensure its performance. [11]

VII. Evaluation Framework

For evaluating the performance of the discussed algorithms we have used the OpenAI fetch pick up and place simulation environment. The fetch manipulation environments are based on an existing hardware robot to ensure that the challenges we face correspond as closely as possible to the real world. These environments use a 7-DOF Fetch Robotics arm which has a two-fingered parallel gripper. The robot is simulated using the MuJoCo [13] physics engine. Robotic tasks are designed as a "goal", for example can be the desired position of the puck in the slide task. These environments use a sparse reward of -1 if the desired goal was not yet achieved and 0 if it was achieved (within some tolerance). [8]

Three fetch environments discussed are as follows:

1) Pushing: In this task a box is placed on a table in front of the robot and the task is to move it to the target location on the table. The robot fingers are locked to prevent grasping. The learned behaviour is a mixture of pushing and rolling.
2) Sliding: In this task a puck is placed on a long slippery table and the target position is outside of the robot's reach so that it has to hit the puck with such a force

that it slides and then stops in the appropriate place due to friction.

3) Pick-and-place: This task is similar to pushing but the target position is in the air and the fingers are not locked.

VIII. Results and Conclusion

RL in behavior-based agents provides adaptiveness to physical or virtual agents while respecting the constraints modeled by the expert. D-HER is able to solve tasks that are not completely solvable by either RL or behavior cloning alone and turns out to perform better than the demonstrator policy. With RLBC, the learned policy exhibits zero-shot generalization which successfully adapts to previously unseen configurations without any further training. DGAIL can effectively train from the demonstration data in different states of the task. It can complete the robot grasping task without environmental reward quickly and improve the stability of the training process. The algorithms are trained for 80000 episodes of 50 steps each. The figure 4 shows success rate comparison of the discussed algorithms for the Pick and Place Task.

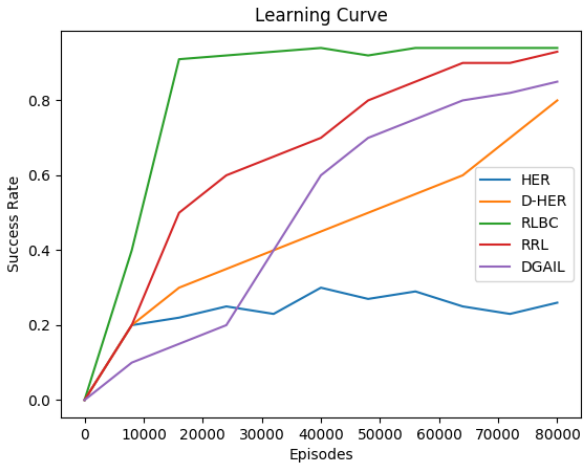


Fig. (4) Learning Curve Comparison

HER is used as a benchmark to compare with other algorithms. We can see that RLBC is able to learn faster as compared to D-HER among the demonstration based algorithms. RRL performs better than DGAIL from a learning perspective.

For BTs, performance measures can be computed using techniques such as success/failure probabilities and execution times, for the plans encoded and executed by them. For a Stochastic BT, the probabilistic performance measures of the basic action controllers are provided. Discrete Time Markov Chains (DTMC) can be used to aggregate these measures from one level of the tree to the next. The recursive structure of the tree enables step by step propagation of such estimates from the leaves (basic action controllers) to the root (complete task execution). [2]

IX. Acknowledgements

I would like to thank Dr. Nicolaus Correll for giving me an opportunity to work for this independent study and guiding me throughout the study. I would also like to thank James Watson for training me and providing feedback for the project. I would also like to thank the whole Correll lab team for great discussions.

References

[1] Dey, Rahul, and Chris Child. "QL-BT: Enhancing behaviour tree design and implementation with q-learning." 2013 IEEE Conference on Computational Inteligence in Games (CIG). IEEE, 2013

[2] Colledanchise, Michele, Alejandro Marzinotto, and Petter Ögren. "Performance analysis of stochastic behavior trees." 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2014.

[3] de Pontes Pereira, Renato, and Paulo Martins Engel. "A framework for constrained and adaptive behavior-based agents." arXiv preprint arXiv:1506.02312 (2015).

[4] Andrychowicz, Marcin, et al. "Hindsight experience replay." Advances in neural information processing systems. 2017.

[5] Colledanchise, Michele, and Petter Ögren. Behavior trees in robotics and AI: An introduction. CRC Press, 2018.

[6] Martens, Chris, Eric Butler, and Joseph C. Osborn. "A Resourceful Reframing of Behavior Trees." arXiv preprint arXiv:1803.09099 (2018).

[7] Sprague, Christopher Iliffe, and Petter Ögren. "Adding neural network controllers to behavior trees without destroying performance guarantees." arXiv preprint arXiv:1809.10283 (2018).

[8] Plappert, Matthias, et al. "Multi-goal reinforcement learning: Challenging robotics environments and request for research." arXiv preprint arXiv:1802.09464 (2018).

[9] Nair, Ashvin, et al. "Overcoming exploration in reinforcement learning with demonstrations." 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018.

[10] Li, Richard, et al. "Towards Practical Multi-Object Manipulation using Relational Reinforcement Learning." arXiv preprint arXiv:1912.11032 (2019).

[11] Lu, Jiahao, et al. "Accomplishing Robot Grasping Task Rapidly via Adversarial Training." 2019 IEEE International Conference on Real-time Computing and Robotics (RCAR). IEEE, 2019

[12] Strudel, Robin, et al. "Learning to combine primitive skills: A step towards versatile robotic manipulation."

[13] E. Todorov, T. Erez and Y. Tassa, "MuJoCo: A physics engine for model-based control," 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, 2012, pp. 5026-5033, doi: 10.1109/IROS.2012.6386109.

[14] von Drigalski, Felix, Christian Schlette, Martin Rudorfer, Nikolaus Correll, Joshua C. Triyonoputro, Weiwei Wan, Tokuo Tsuji, and Tetsuyou Watanabe. "Robots assembling machines: learning from the World Robot Summit 2018 Assembly Challenge." Advanced Robotics (2019): 1-14