Towards a Robot Simulation Framework for E-waste Disassembly Using Reinforcement Learning

Brohus Kristensen, Christoffer; Arentz Sørensen, Frederik; Bjørn Dalgaard Brandt Nielsen, Hjalte; Søndergaard Andersen, Martin; Poll Bendtsen, Søren; Bøgh, Simon

Published in:
29th International Conference on Flexible Automation and Intelligent Manufacturing

DOI (link to publication from Publisher):
10.1016/j.promfg.2020.01.030

Publication date:
2019

Document Version
Publisher's PDF, also known as Version of record

Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 38 (2019) 225–232

Procedia MANUFACTURING

www.elsevier.com/locate/procedia

29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2019), June 24-28, 2019, Limerick, Ireland.

Towards a Robot Simulation Framework for E-waste Disassembly Using Reinforcement Learning

Christoffer B. Kristensen, Frederik A. Sørensen, Hjalte B. Nielsen, Martin S. Andersen, Søren P. Bendtsen, Simon Bøgh*

Fibigerstræde 16, 9220 Aalborg- Denmark

Abstract

The purpose of this paper is to introduce a new framework for training and testing Reinforcement Learning (RL) algorithms for robotic unscrewing tasks. The paper investigates current disassembly technologies through a state-of-the-art analysis, and the basic concepts of reinforcement learning are studied. A comparable framework exists as an extension for OpenAI gym called Gym-Gazebo, which is tested and analysed. Based on this analysis, a design for a new framework is made to specifically support unscrewing operations in robotics disassembly of electronics waste.

The proposed simulation architecture uses ROS as data middleware, Gazebo (with the ODE physics solver) for simulating the robot environment, and MoveIt as a controller. The Gazebo simulation consists of a minimalistic setup in order to stay focused on the architecture and usability of the framework. The simulation world interfaces with the RL-agent, using OpenAI Gym and ROS-topics, which can be adapted to interface with a real robot. Lastly, the work demonstrates the functionality of the system by implementing an application example using a Q-learning algorithm, and the results of this are presented.

Keywords: E-waste; Robotic Disassembly; Reinforcement Learning

* Corresponding author. Tel.: +45 2916 6401.
E-mail address: sb@mp.aau.dk

10.1016/j.promfg.2020.01.030

1. Introduction

A major challenge of today's robotics applications in the context of intelligent manufacturing systems is to be flexible while being efficient in setting up tasks. Often complex, or even simple, robotics applications require the system to be able to adapt to small variations in process execution due to e.g. irregularities in part quality. This is evident when looking into handling electronics waste (e-waste) for a sustainable life-cycle of consumer products. When e.g. mobile phones reach their End-of-Life (EoL) they are regarded as e-waste. As of 2016 44.7 million tonnes of e-waste was produced, of which merely 8.9 millions tonnes (20%) is collected and properly recycled. The remaining 35.8 millions tonnes (80%) is not documented, meaning it is thrown out with the general waste, dumped, traded or recycled under inferior conditions. [1]

Disassembly is the preferred method for reusing and recycling e-waste. However, e-waste can contain highly toxic materials why automated disassembly is highly sought.[2] The implementation of motion planning for an industrial robot in such a use case needs to be able to learn how to automatically cope with the natural variation exhibited in e-waste components, which may potentially be scratched, damaged or missing components.

Since electronic devices differ in size and complexity, creating a single non-destructive automatic disassembly solution, would be a tremendous task. Common for most devices though is that screws are used to fasten components, which makes unscrewing an ideal place to start. But even creating an automatic system for just unscrewing screws is a complex task; both because not all assembly manuals for electronic devices are available, and due to the design complexity of modern electronic devices.

However, [3] presents a technique of measuring and analysing the amount of force and torque humans apply when screwing and unscrewing and have adapted this to an unscrewing robot, with great success as the robot is able to screw and unscrew screws. This gives motivation for continuing researching in this field.

2. Background and related work

When exploring the field of autonomous disassembly, the concept of machine learning frequently appears. Especially reinforcement learning (RL) is an area of increasing interest to the robotics research community. RL is a branch of machine learning and is a computational approach, where a system (agent) learns how to reach a goal. The RL in this paper uses the Markov Decision Process (MDP), which defines the interaction between a learning agent and the environment with regards to actions, states and rewards. [4]

The agent is learning by getting rewarded or penalised. Typically, the agent is rewarded when doing correct steps towards reaching the goal, however rewards and penalties are not always given after each step. The agent picks an action from a specified action space. The action is executed in an environment, which outputs a state, and a reward based on the action taken in the state. The agent should pick actions so that it will at some point reach the goal. The loop of RL can be seen in Fig. 1.

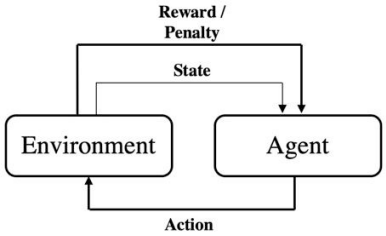


Fig. 1. Reinforcement learning loop: The agent performs an action, which affects the environment. The agent receives a reward based on the executed action and how it affected the environment. Throughout training, the agent tries to maximise the reward.

For each state the agent should pick the action that it expects will give the highest accumulated future reward. For this, the agent uses an algorithm/value function/neural network.

Recent research shows that RL has proven especially competent learning to play and perfecting arcade games [5]. DeepMind, a company now acquired by Google, used Deep Learning combined with RL to train an agent to learn and play 49 Atari 2600 games and in more than half of the games, outperformed a human under the same conditions and the same sensory feedback.

Also in the field of robotics, RL has proven useful. [6] tackles motion planning of an industrial manipulator, by letting it follow a path similar to a wire-loop game, where the ring is not allowed to touch the wire. They implemented it in simulation, and it showed, that it was able to learn the game and the path and be able to follow new paths without needing to learn them.

The people of Erle Robotics have made a framework for testing different reinforcement learning algorithms [7]. Gym-Gazebo serves as an extension to the OpenAI Gym toolkit, and connects the robot simulation tool Gazebo, which is a 3D modelling and rendering tool providing a robust physics engine with Robot Operating System (ROS). ROS is used as the connection between the Gym and the Gazebo simulator. The findings of [7] showed that when using Q-learning and SARSA to train their system, Q-learning was the fastest learning technique.

[7] has tested the Gym-Gazebo extension and tried to implement different learning algorithms. However, the Gym-Gazebo framework proved to be complex to install and set up, and different versions of dependencies conflicted constantly. In addition, it required a thorough knowledge of ROS, Gazebo, Python, etc. Not an easy framework to use without years of experience.

This paper will present a new framework to test RL-algorithms for unscrewing screws with a UR5 robotic manipulator. The framework consists of a simulation environment, controllers for controlling the simulated UR5 robot and screwdriver tool, and an interface for RL-environments. The work presented should provide a framework where it is easy to implement and test different RL-algorithms. The system will be tested using a designed application example and evaluated both based on the results from these tests and based on the documentation and usability of the framework.

3. Methods

The framework is designed to be a user-friendly framework, for others who wants to develop and test RL-environments. It is of main focus that the framework is well documented to make the framework user friendly and easy to continue development on. Fig. 2 gives an overview of what the framework consists of. The Gazebo World is the simulation world, MoveIt is the controller and the yellow ovals are the interface to the framework environment. Combined, these three parts provide the framework. The blue square on the right-hand side is where the user should implement their RL-environment. In this paper an OpenAI Gym RL-environment is implemented for testing the framework. The framework is made in ROS, which means every part of the framework is consisting of ROS-packages, which will be described briefly in the following explanation of each part of the system.

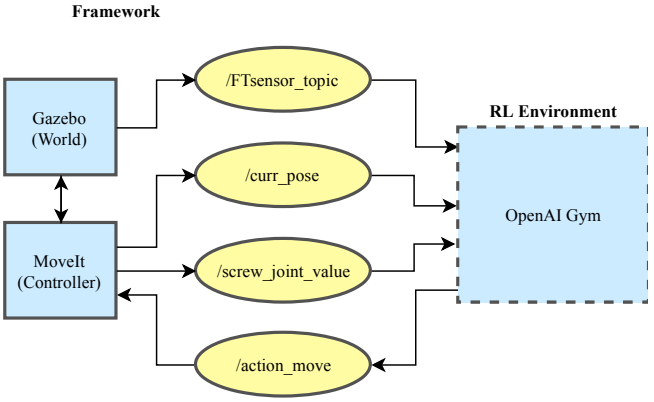


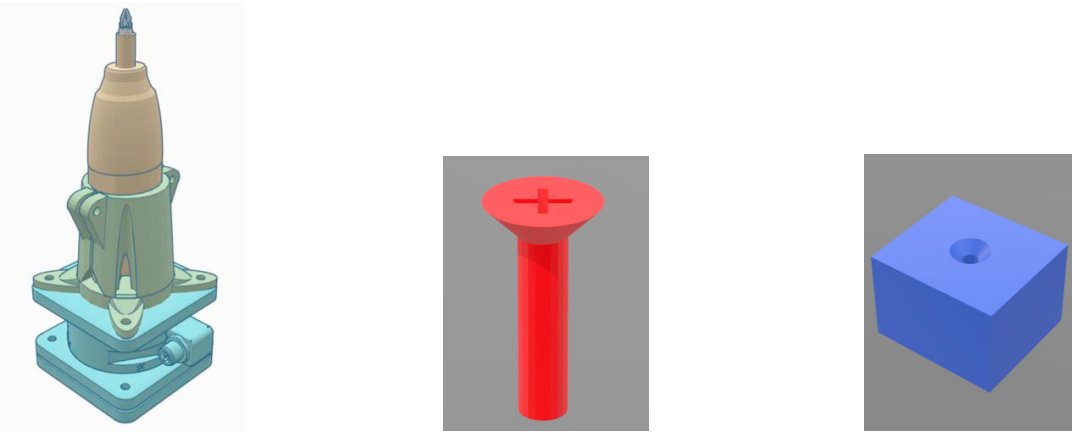
Fig. 2. The RL-Unscrew framework is seen on the left hand side of the diagram. The four ROS-topics in the middle are the interface to the simulation environment. The RL-environment on the right hand side should be designed by the user of the framework.

3.1. Simulation world

The simulation world is built in Gazebo. Gazebo is a robot simulator, which has integration with ROS and high customisation ability [8]. The physics engine used in the simulation world is Open Dynamics Engine (ODE).

3D models of the needed equipment are made as CAD drawings and exported as STL-files. These models provide the physical dimensions and visual look of the world. The screwdriver tool model, shown in Fig. 3a, is consisting of a force-torque sensor (blue), a mount for the screwdriver (green), a cordless Makita screwdriver (yellow) and a screw bit (white).

Models of a screw and screw block are made, which are illustrated in Fig. 3b and Fig. 3c. The screw and the screw block have no thread as the screwing mechanism is implemented with a screw joint in Gazebo. The model of the UR5 robot is provided by a package called Universal Robots [9] which contains URDF description-files for the UR5 robot.



a **b** **c**

Fig. 3. Parts implemented in the simulation world. **(a)** CAD drawing of the complete screwdriver tool used for unscrewing screws. **(b)** 3D-model of the screw implemented in the simulation world. **(c)** Simple screw block where the red screw fits into.

The framework implements a ROS-package for the simulation world. The package includes description files of the screwdriver tool, world and work pieces. Furthermore a this package implements the ROS-topic */FTsensor_topic* on which force-torque sensor data is outputted.

3.2. Controller

The controllers in the simulation is made in MoveIt [10]. MoveIt is a package that makes it possible to set up controllers for robots in Gazebo [10]. MoveIt is used in the framework to set up a position controller, which controls the position of the end-effector, and/or the joint values of each joint.

Two ROS-packages are created for the controller. A package containing YAML-files describing kinematics, joint limits etc., and a package containing scripts for sending move commands to Gazebo. In the lather, ROS-topics for communication is set up. These topics are: /curr_pose, /screw_joint_value and /action_move, and are seen in Fig. 2.

3.3. RL-environment

The RL-environment should be designed by the user of the framework. In the RL-environment, actions, state and rewards are defined as well as necessary functions for making the RL-loop run. This includes functions for being able to reset and sending actions to the simulation environment. A ROS-package has been developed where an application example is implemented. The ROS-package can either be edited in or completely replaced, when using the framework. The application example is designed and implemented to show that the framework is working. In the example, an agent should try to learn how to unscrew screws by the use of the UR5 robot with the screwdriver tool. The actions are defined to be:

1. Move 0.5 millimetre in x+ direction
2. Move 0.5 millimetre in x- direction
3. Move 0.5 millimetre in y+ direction
4. Move 0.5 millimetre in y- direction
5. Move 0.5 millimetre in z+ direction
6. Move 0.5 millimetre in z- direction
7. Unscrew 0.25 radian in counter-clockwise direction

This means, for example, that if the agent picks action 1, the end-effector is moved 0.5 millimetre along the x-axis. The end-effector is fixed, so that it will always point downwards perpendicular to the object containing the screw. The state is defined to be:

- Force-torque sensor output: Force: (z)
- Screw bit joint value (radian)
- Position of the UR5 End-Effector (x,y,z)

The rewards are defined to be:

- A reward of 2 is given if the screwdriver tool is touching something and if the action is unscrewing (action 7). This reward is evaluated after every action. The screwdriver tool is touching something when force in z-direction is lower than 0 (due to the orientation of the z-axis)
- After an episode has ended a reward is given based on the z value of the screwdriver tool position. A low z value results in a higher reward. A maximum of 50 is given by this reward.
- If the screwdriver tool position is outside of the boundary a penalty of -10 is given to the agent.
- Additionally after an episode has ended, and if it has been established that the screw bit has touched the screw, a reward of $30 + z_{value} * 3000$ is given. 3000 is a scaling factor since the movements is in millimetre, but the position is given in meters. A maximum of 150 can be given by this reward.

Q-learning is implemented to estimate Q-values, where the action with the highest Q-value should be picked by the agent. The Q-learning equation is given by Eq. 1 by Chris Watkins's: [11, 4]

$$Q(a_t, s_t) \leftarrow Q(a_t, s_t) + \alpha \left[R_{t+1} + \gamma \max_a (Q(a, s_{t+1})) - Q(a_t, s_t) \right] \quad (1)$$

$Q(a_t, s_t)$ is the Q-function that the system updates based on the state and action at time t. $Q(a_t, s_t)$ on the right hand side of the arrow is the old Q-value. The learning rate α must be between [0;1], and controls how much difference between current and a new Q value is considered.

$$\left[R_{t+1} + \gamma \max_a (Q(a, s_{t+1})) - Q(a_t, s_t) \right] \quad (2)$$

is the temporal difference $TD_t(a_t, s_t)$ where γ is the discount factor and should be between [0;1]. [4]

Different parameters for the Q-learning are set:

- $\alpha = 0.2$ (Learning rate)
- $\gamma = 0.85$ (Discount factor) \\
- $\epsilon = 0.9$ (Start ϵ. This decreases over time)

3.4. Interface

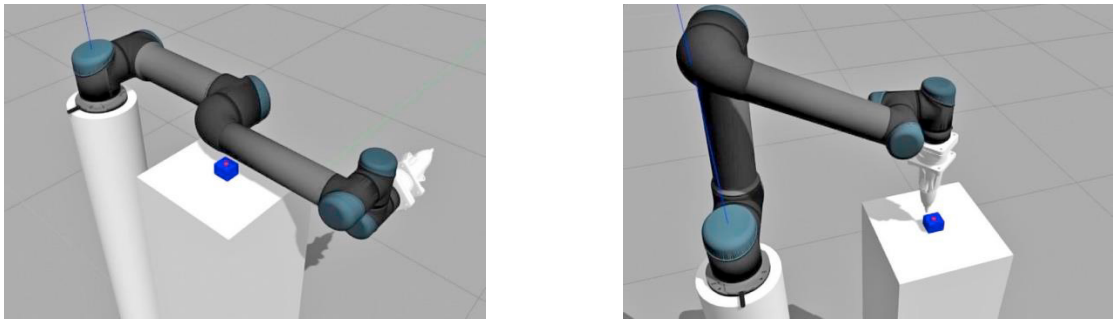
The communication between the framework and the RL-environment is done using ROS-topics as previously explained. Publishers and subscribers are set up and are using the four ROS-topics seen in Fig. 2 from before:

/FTsensor_topic is outputting the force-torque sensor data.
/curr_pose is outputting the current position and orientation of the end-effector.
/screw_joint_value is outputting the orientation of the screw bit in the screwdriver tool.
/action_move is listening for actions, if an action is received it is executed in Gazebo by the controllers.

4. Results

The framework is tested using the application example. The goal of the application example is to implement an RL-algorithm to test if the framework is working as intended. The test is a full integration test where the framework is launched and the application example running as the RL-environment. It will be tested that the system is able to reset, execute actions and get the state.

The test is considered successful if the system can start and complete the training session. It is not a requirement that the agent successfully finds and unscrew the screw for the test to be considered successful. The training session is set to run through 100 episodes and it successfully finished the 100th planned episodes, after 56 minutes and 51 seconds. The training finished but by visual inspection during the training session it was noted that sometimes the robot was seen in positions and orientations that should not be possible, and which the reset function or any defined action should allow the robot to be in. Therefore the reset function and the actions are tested separately.



(a) (b)

Fig. 4 (a) The configuration of the robot after launching the world. (b) The wanted configuration of the robot after the reset service is called.

4.1. Reset function

The reset function is developed as a ROS-service which starts up after the controller has launched. The ROS-service is tested separately where it is called 5 times from different configurations of the robot. One of the tests is displayed in Fig. 4. All 5 times the simulation world is reset, hence the reset function is considered functional as a stand-alone function but is sometimes failing when running with the rest of the framework.

The problem is traced back to the Gazebo reset service not always working correctly, and therefore not a problem regarding the framework.

4.2. The actions

The actions are tested separately, where the start position of the end-effector is noted, and an action is published to the */action_move* topic. The new position is noted and checked with the start position, to see if the end-effector has moved 0.5 mm. This is done 3 times in each direction, ± in both x, y and z.

The results presented in Table 1 shows the average movement in each direction of the three times the action is send. It is close to 0.5 mm along the axis it should move (marked with grey), and close to no movement in the other two directions. The testing of the action is showing that the actions work as intended.

Table 1 Results from action test. Column x, y and z show movement along each axis. Each row is a move command of ± 0.5 mm in the direction noted in the left most column. The cells marked with grey should be ± 0.5 mm and the rest should be zero. The results are averages of three trials.

		Movement in axis		
		x	y	z
Command sent	x+	0.506 mm	-0.030 mm	0.048 mm
	x-	-0.470 mm	-0.031 mm	-0.016 mm
	y+	0.006 mm	0.458 mm	0.004 mm
	y-	-0.012 mm	-0.468 mm	-0.013 mm
	z+	-0.005 mm	0.003 mm	0.489 mm
	z-	0.013 mm	-0.037 mm	-0.558 mm

5. Conclusion and future work

This paper investigated the area of reinforcement learning within the field of automated disassembly technology, and a first and very important step towards a framework for developing and testing RL-algorithms is taken. Automation in all areas of production and disassembly has been the focus for a long time, and especially machine learning is a hot topic these years. Different frameworks exist for testing reinforcement learning algorithms, but specifically with a ROS implementation, only Gym-Gazebo extension was found, and subsequently tested. This experience formed the background for this project; trying to create a simple framework for testing reinforcement learning algorithms in the application of unscrewing. In addition, it was a specific requirement that the documentation should be of high standard. Everything from installation guides, to detailed descriptions of setup and configuration of the environment, as well as tips and tricks, are included in the documentation.
As stated in the results section, this project succeeded in delivering a baseline framework for testing reinforcement learning algorithms. In future work, several things will be developed further. Improvements to the Gazebo simulation will be made. Factors like mass, inertia and frictional forces will be calculated. The framework will be refined, so it becomes even easier to configure actions, states, rewards, and implement new algorithms. The applications could be widened to more and different unscrewing tasks, so different screws can be tested, and a camera can be added for machine vision, as well as other types of sensors; although the strength of this framework lies in its simplicity.

References

[1] C. Baldé, V. Forti, V. Gray, R. Kuehr og P. Stegmann, »The Global E-waste Monitor,« Bonn Geneva Vienna, 2017.

[2] S. Vongbunyong og W. H. Chen, Disassembly Automation: Automated Systems with Cognitive Abilities, Springer, 2015.

[3] D. Mironov, M. Altamirano, H. Zabihifar, A. Liviniuk, V. Liviniuk og D. Tsetserukou, »Haptics of Screwing and Unscrewing for its Application in Smart Factories for Disassembly,« *ArXiv e-prints,* 2018.

[4] R. Sutton og A. Barto, Reinforcement Learning: An Introduction, The MIT press, 2018.

[5] V. Mnih og e. al., »Human-level control through deep reinforcement learning,« *Nature,* 2015.

[6] R. Meyes, H. Tercan, S. Roggendorf, T. Thiele, C. Büscher, M. Obdenbusch, C. Brecher, S. Jeschke og T. Meisen, »Motion Planning for Industrial Robots using Reinforcement Learning,« *Procedia CIRP,* nr. 63, pp. 107-112, 2017.

[7] Z. a. Iker, Lopez, N. Gonzalez, Vilches, V. Mayoral, Cordero og A. Hernandez, »Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo,« *arXiv preprint arXiv:1608.05742,* 2016.

[8] lenka, »V-REP, Gazebo or ARGoS? A Robot Simulators Comparison,« 17 1 2018. [Online]. Available: http://lenkaspace.net/blog/show/120.

[9] F. Messmer, *universal robot Package Summary,* http://wiki.ros.org/universal_robot, 2018.

[10] I. A. Sucan og S. Chitta, *Moveit,* http://moveit.ros.org/, 2018.

[11] P. H. de og E. Kirill, *Artificial intelligence A-Z - A summary paper on the most powerful and popular AI models that exist today,* Super Datascience.