Software Design Requirements

for

Reinforcement Learning Environment for Robotic Suturing

Group 20: Walee Attia (wattia1), Jocelyn Hsu (jhsu37), Jihoon Kim (jkim620)

Mentors: Dr. Anqi Liu, Dr. Adnan Munawar, Dr. Manish Sahu, Dr. Peter Kazanzides

May 11, 2023

Contents

1 Introduction

1.1 Purpose

Reinforcement learning (RL) is a machine learning framework involved in creating artificial agents that fulfill various complex problems. Surgical robots have opened the door to surgical task automation, which has piqued the interest of RL research. However, no robust framework exists for RL tasks in surgical robotics environments. We developed an OpenAI Gym environment based on an autonomous robotic suturing simulator with benchmark algorithms to pave the way for future surgical automation. Integrating the surgical robotics simulator with the OpenAI Gym framework will enable the comparison of different reinforcement learning algorithms' (of varying complexity) abilities to perform autonomous suturing and streamline the process for more complex tasks in the future.

1.2 Project Scope

The AccelNet Surgical Robotics Challenge, which served as the inspiration for this project, is broken down into 3 challenge tasks.

- The first challenge involves finding the needle. This includes developing an algorithm to identify the pose of the suturing needle within the simulation given the 3D model of the needle and camera calibration. Given that a computer vision approach is more suited for the nature of this problem and the provided data, we decided to not focus our efforts directly on this challenge (our emphasis is on development of the Gym environment compatible with reinforcement learning solutions).

- The second challenge involves grasping the needle and driving through the tissue. Given the initial ground-truth pose of the needle and the poses of the entry target and exit on the suturing phantom, the goal is to move the seven DOF needle driver towards the needle at the right angle, grasp it, and drive the needle through the tissue until the needle tip exits on the other side of the suturing phantom. Note that the grasp pose is not restricted so long as the desired suturing outcome is achieved. This challenge is the focus of our project, as all tasks in this challenge will be trained with our OpenAI Gym environment.

- Lastly, the third challenge involves suturing the phantom. This includes develop an algorithm that repeatedly drives the needle through the suturing phantom from the entry point to the associated exit point. The left needle driver interacts with

the right needle driver by pulling the needle after it goes through the hole and giving it back to the right needle driver. This should be done for every pair of entry and exit points. We leave this challenge as future work that builds on the OpenAI-Gym-compatible environment and benchmarking done for challenge 2.

2 Surgical Robotics Challenge Environment

2.1 Challenge Requirements [1]

The 2021-2022 AccelNet Surgical Robotics Challenge is a simulation platform featuring two seven degrees-of-freedom (DOF) needle drivers and a controller based on the da Vinci Surgical System, a suturing phantom, and a needle with suture. The challenge is based on the Asynchronous Multi-Body Framework (AMBF) simulator [2], which is a real-time dynamics simulator for robots. The RL approach to robotic suturing focuses on Challenge 2, using the needle driver to grasp the needle and drive the needle through the suture phantom.

2.2 Functional Requirements

In order to complete a custom environment compatible with OpenAI Gym, the environment must function as a wrapper class for the base OpenAI gym.env. The simulation will be connected to the AMBF simulator and ROS components of dVRK. The environment should initialize all the AMBF simulation components, which will then be connected as clients to the corresponding ROS objects. For Challenge 2 specifically, this will encompass creating the simulation manager, the scene, 2 patient side manipulators (PSM), 1 endoscopic manipulator (ECM), and a needle. Due to various components, each in their own coordinate frame, the environment should support frame transformations across different objects. Additionally, the environment should contain methods that the reinforcement learning algorithm will require, namely `step()`, `reset()`, `render()`, and `close()` methods.

2.3 Design

The OpenAI Gym environment developed, SRCEnv, is currently comprised of piecewise reward functions for `Grasp`, with compatibility for future `Insert` and `Target` tasks (Fig. 2.1). The overall environment contains the `step()`, `reset()`, `render()`, and `close()` methods, and each task will have its own `reward()` function implemented that modifies the robot's goal while executing the task.

`Grasp`

The `Grasp` task will be responsible for handling the initial grabbing of the needle from the surface of the surgical table. Given the positions of the needle and the robotic arm,

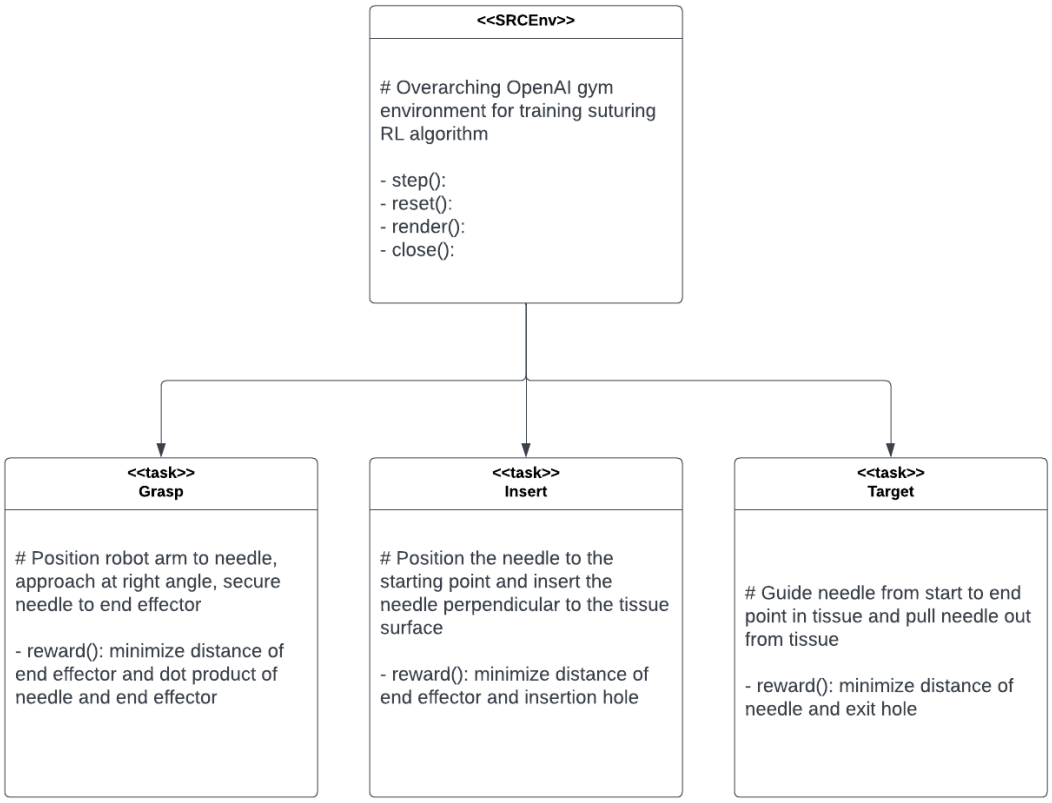


Figure 2.1: Planned Environment Architecture

this environment will involve moving the robotic arm to the coordinates of where the needle is located, rotating the joint of the arm such that it is perpendicular to the needle, and securing the needle to the end effector of the robotic arm.

`Insert`

The `Insert` task will move the needle already grasped in the robotic arm to the starting position provided. Once at the right position and angle, the robot will puncture the needle through the surface of the tissue.

`Target`

The `Target` task guides the needle embedded in the tissue to the target end point. The needle will then be pulled through the end point to complete the single suture.

Method development

For the overall SRCEnv, the following methods have been developed:

- `step()`
  - Adjusts the robotic arm position based on the calculated action and returns the observation and reward resulting from the action. The robotic arm position can be backtracked to a previous state if a failure mode is encountered, such as dropping the needle after grasping.
  - Parameters:
    - `self`: robotic arm position, needle position
    - `action`: change in robotic arm position
  - Returns
    - `observation`: robotic arm position, needle position
    - `reward`: calculated reward score
    - `terminated`: `True` if task is accomplished, `False` otherwise
    - `truncated`: `True` if failing to accomplish task within certain number of iterations, `False` otherwise
    - `info`: additional diagnostic information
- `reset()`
  - Sets all variables of the environment to its initial state.
  - Parameters:
    - `self`: robotic arm position, needle position
    - `seed` (optional): randomization seed for replication, if needed
  - Returns
    - `observation`: initial robotic arm position, initial needle position
    - `info`: additional diagnostic information
- `render()`
  - Visualize the state of the environment from the agent's perspective.
  - Parameters:
    - `self`: robotic arm position, needle position
  - Returns: None
- `close()`
  - Completes the simulation.
  - Parameters:

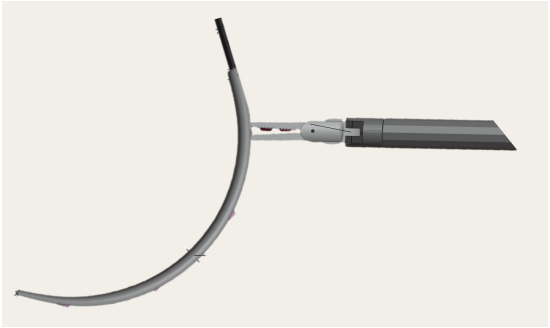
* `self`: robotic arm position, needle position

– Returns:

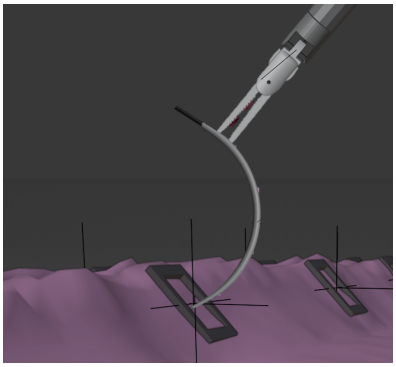
* `Env.unwrapped`: raw state of the environment

2.4 Reward algorithm

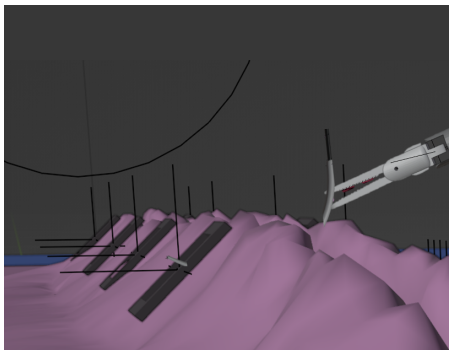
To position the PSM correctly in order to accomplish Challenge 2, we have determined the following positions to be optimal positions for Grasp, Insert, and Target.



(a) Grasp



(b) Insert



(c) Target

Figure 2.2: Optimal PSM Positions

Distance: Grasp

For grasping the needle with the PSM, the goal is position the PSM at the base of the needle.

In order to minimize this distance, the reward function computes the magnitude of p_{res}, where $p_{res} = p_{needle} - p_{psm} = (x, y, z)$. The magnitude of p_{res} can then be computed with the Euclidean norm, such that $|p_{res}| = \sqrt{x^2 + y^2 + z^2}$. The reward distance function for grasp then returns $|p_{res}|$.

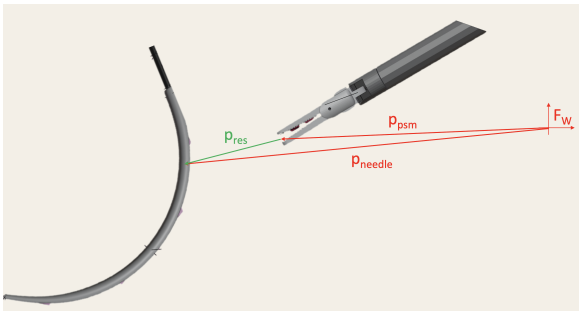


Figure 2.3

Angle: Grasp

To approach the needle at the correct angle, the PSM must be as close to perpendicular to the needle as possible.

To achieve this, we define p_1 as the vector perpendicular to the PSM end-effector jaw and p_2 as the vector parallel to the PSM end-effector jaw. Similarly, n_1 is the vector tangent to the base of the needle, and n_2 is the vector perpendicular to the base of the needle. To ensure that p_1 is parallel to n_1 and p_2 is parallel to n_2, the cross products $p_1 \times n_1 = 0$ and $p_2 \times n_2 = 0$. The reward angle function for grasp then returns then sum of the cross products, $p_1 \times n_1 + p_2 \times n_2$.

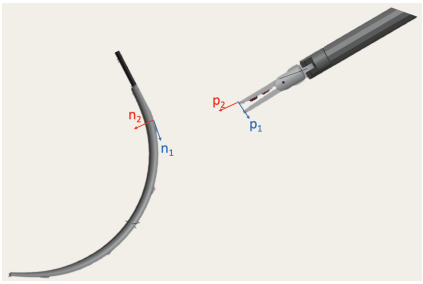


Figure 2.4

Distance: Insert & Target

For guiding the needle to the starting hole and driving the needle through the tissue to the target hole, the goal is position the needle as close to the center of the hole as possible.

Similar to the reward distance function for the grasp, the reward function computes the magnitude of p_{res}, where $p_{res} = p_{hole} - p_{needle} = (x, y, z)$. The magnitude of p_{res} can then be computed with the Euclidean norm, such that $|p_{res}| = \sqrt{x^2 + y^2 + z^2}$. The reward distance function for both insert and target then returns $|p_{res}|$.

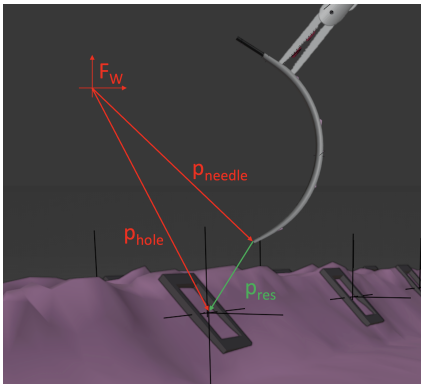


Figure 2.5

Angle: Insert & Target

For optimal suturing, the needle should be guided perpendicularly through the insertion hole and out the target hole.

To achieve this, we define n_1 as the vector tangent to the tip of the needle and n_2 as the vector perpendicular to the tip of the needle. h_1 is the vector pointing perpendicularly down the surface of the skin at the insertion/target hole, and h_2 is the vector tangent to the insertion/target hole and surface of the skin. To ensure that n_1 is parallel to h_1 and n_2 is parallel to h_2, the cross products $n_1 \times h_1 = 0$ and $n_2 \times h_2 = 0$. The reward angle function for grasp then returns then sum of the cross products, $n_1 \times h_1 + n_2 \times h_2$.

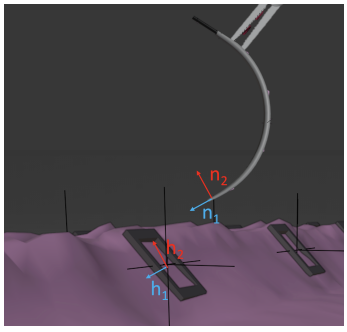


Figure 2.6

3 Reinforcement Learning

Reinforcement learning involves an agent interacting with an environment to learn how to take actions that maximize a cumulative reward signal [3]. The agent learns through trial and error, gradually improving its policy for taking actions based on the feedback it receives from the environment in the form of rewards or penalties.

The RL framework consists of an agent, an environment, a set of states S, a set of actions A, a reward function R, transition probabilities P, and a discount factor $\gamma \in [0, 1]$. The agent is responsible for taking actions in the environment based on its current policy, which maps states to actions. The environment provides feedback to the agent in the form of rewards or penalties, which the agent uses to update its policy. The state of the environment is typically represented as a vector of features that captures relevant information about the current situation.

The agent's goal in RL is to learn a policy that maximizes the cumulative reward over time. The cumulative reward for a step at time t can be computed with

$$R_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.1}$$

This is typically achieved through the use of a value function or a Q-function, which estimates the expected cumulative reward of following a given policy from a given state or state-action pair, respectively. The optimal value function $V^{\pi *}$ can be updated using the Bellman equation, where

$$V^{\pi *} = \max_a \{R(s, a) + \gamma \{P(s'|s, a\} V^{\pi *}(s')\} \tag{3.2}$$

which expresses the expected reward in terms of the current maximized reward and the expected future reward given the future state s'.

3.1 Functional Requirements

With the SRCEnv, we now have the ability to train reinforcement learning (RL) agents on this environment. To demonstrate the ability to seamlessly take advantage of OpenAI Gym, we will be implementing current state-of-the-art RL algorithms on the second challenge of the Surgical Robotics Challenge, which requires the RL agent to grasp the needle, then drive the needle to and through the suturing phantom, all at optimal target locations and angles. The RL algorithm will take advantage of the `step()` function to input the action decided by the RL agent, and the `reset()` function to reset the environment for the multiple epochs it will be trained on. We will be able to easily visualize and debug the training and evaluation process with the `render()` function to properly assess at the RL algorithm's performance.

3.2 Design

In the design of our RL algorithms for the Surgical Robotics Challenge, we will be using two state-of-the-art techniques, Deep Deterministic Policy Gradient (DDPG) and Hindsight Experience Replay (HER).

Deep Deterministic Policy Gradient [4]

The DDPG algorithm is a model-free, off-policy actor-critic algorithm that is specifically designed for continuous control problems, making it well-suited for robotic control tasks. The algorithm is an extension of the popular Q-learning algorithm used in RL that operates in continuous action spaces by learning a deterministic policy function that maps states to actions. The key idea behind DDPG is to use deep neural networks to approximate the value function and the policy function, allowing the algorithm to handle high-dimensional state and action spaces.

In DDPG, the actor network is responsible for selecting actions based on the current state, while the critic network evaluates the chosen actions and the resulting state transition. The actor network is trained using a deterministic policy gradient, which is the gradient of the action-value function with respect to the policy parameters. Parameters can be optimized by minimizing the loss

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}[(Q(s_t, a_t|\theta^Q) - y_t)^2] \tag{3.3}$$

where

$$\begin{aligned} &y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q) \\ &\beta \text{ is the stochastic policy} \\ &\rho^\beta \text{ is the discounted policy} \end{aligned} \tag{3.4}$$

The critic network is trained using temporal difference (TD) learning, which involves updating the estimate of the action-value function based on the difference between the observed reward and the predicted reward.

Hindsight Experience Replay [5]

HER is an extension of the DDPG algorithm that was specifically designed to address the challenge of sparse rewards in RL, which is a common problem in many robotics applications. The key idea behind HER is to modify the reward function during training so that even unsuccessful episodes can contribute useful information to the learning process. This is achieved by relabeling the original goal with the achieved goal in hindsight and using this modified goal to compute the reward.

In HER, a modified replay buffer is used to store not only the original transitions experienced by the agent, but also the modified transitions that result from relabeling the goal. During training, the agent samples transitions from this buffer and uses them to update the policy and value functions. By relabeling the goals in hindsight, HER is

able to turn unsuccessful episodes into successful ones, providing the agent with more opportunities to learn and reducing the sparsity of the reward signal.

HER can be applied to RL agents learning a set of goals G. For all $g \in G$, there exists a state s such that the agent's goal g is achieved, or $f_g(s) = 1$. For some batch of iterations B, the states and their associated actions that achieved transitions between states can be stored in a replay buffer. After updating the replay buffer R, R is sampled such that the policy can be optimized based off the history of states and actions, regardless of the rewards of those actions.

3.3 Implementation, Training, and Evaluation

To implement, train, evaluate the DDPG and HER algorithms described above, we will be taking advantage of Tianshou [6], which is an RL platform based on PyTorch that supports OpenAI Gymnasium. This Python API allows for a seamless connection with our developed SRCEnv since both are developed on the Gymnasium environment. Tianshou has shown to have state-of-the-art benchmark performance on existing Gymnasium environments, particularly the MuJoCo Benchmark, which is a corpus of 3D robotics RL environments. Tianshou is the only RL platform that supports the newly released version of Gymnasium and is regularly updated and maintained.

Tianshou API Overview [6]

The Tianshou API decouples the RL algorithms into several components, each with its distinct functionality. The components are laid out as such:

1. `init`: This function initializes the policy.
2. `forward`: This function computes actions based on given observations.
3. `process_fn`: This function preprocesses data from the replay buffer. All algorithms have been reformulated as replay-buffer-based algorithms.
4. `learn`: This function learns from a given batch of data.
5. `post_process_fn`: This function updates the replay buffer based on the learning process. For example, the prioritized replay buffer requires updating the weight.
6. `update`: This function acts as the main interface for training. It sequentially calls the process_fn, learn, and post_process_fn functions.

The API's design effectively modularizes the reinforcement learning process, streamlining the development and integration of various algorithms into the project.

4 Other

4.1 Instructions for running

Our code repository is located on GitHub (https://github.com/jocelynhsu8/SRC-gym). During set-up, please download ROS (http://wiki.ros.org/noetic/Installation/Ubuntu) and AMBF (https://github.com/WPI-AIM/ambf) with their respective instructions. To run the Surgical Robotics Challenge [1], please refer to https://github.com/collaborative-robotics/surgical_robotics_challenge. After cloning and setting up all packages, please run the following to set up the SRCEnv:

Setting up SRCEnv

```
cd SRC-gym/gym-env/src/scripts
pip3 install -e .
```

After installation and set-up, training can be run with the following commands:

Terminal 1:

```
roscore
```

Terminal 2:

```
cd ~/ambf/bin/lin-x86_64
./ambf_simulator -g 0 --launch_file ~/SRC-gym/gym-env/src/launch.yaml
-l 0,1,3,4,14,15 -p 120 -t 1 --override_max_comm_freq 120
```

Finally, to train the SRCEnv:
Terminal 3:

```
cd ~/SRC-gym/gym-env
python3 PSM_herddpg.py
```

Bibliography

[1] C. R. T. (CRTK), "2021-2022 accelnet surgical robotics challenge." [Online]. Available: https://collaborative-robotics.github.io/surgical-robotics-challenge/challenge-2021.html

[2] A. Munawar, Y. Wang, R. Gondokaryono, and G. S. Fischer, "A real-time dynamic simulator and an associated front-end representation format for simulating complex robots and environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Nov. 2019. [Online]. Available: https://doi.org/10.1109/iros40897.2019.8968568

[3] D. Silver, "Introduction to reinforcement learning with david silver." [Online]. Available: https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015. [Online]. Available: https://arxiv.org/abs/1509.02971

[5] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," 2018. [Online]. Available: https://arxiv.org/abs/1707.01495

[6] J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, Y. Su, H. Su, and J. Zhu, "Tianshou: A highly modularized deep reinforcement learning library," *Journal of Machine Learning Research*, vol. 23, no. 267, pp. 1–6, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1127.html