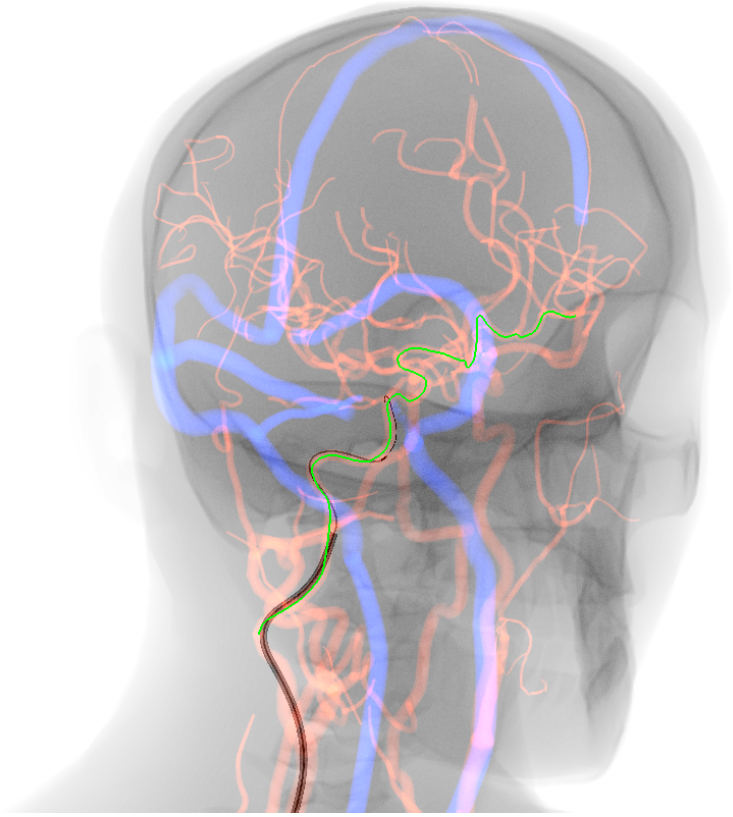

Autonomous Navigation In Real-Time Endovascular Simulation

Using Deep Reinforcement Learning

Master's thesis in Complex Adaptive Systems

NIKLAS MAGNUSSON
LUKAS PETTERSSON

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Autonomous Navigation In Real-Time Endovascular Simulation

Using Deep Reinforcement Learning

Niklas Magnusson, Lukas Pettersson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2026

Autonomous Navigation In Real-Time Endovascular Simulation

Niklas Magnusson, Lukas Pettersson

Supervisor: Martin Ekerstedt, Mentice AB
Examiner: Daniel Midtvedt, Department of Physics

Master's Thesis 2026
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Autonomous navigation with a micro wire and catheter in VIST, with a sheath as support. The tools are colored black, while the followed path is in green.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Autonomous Navigation In Real-Time Endovascular Simulation
Using Deep Reinforcement Learning
Niklas Magnusson, Lukas Pettersson
Department of Physics
Chalmers University of Technology

Abstract

This Master's thesis investigates the use of deep reinforcement learning for autonomous navigation in a simulated endovascular environment. Specifically, a Soft Actor-Critic (SAC) algorithm is employed to train an agent to control a micro guidewire and a micro catheter for path-following tasks.

The simulation environment is based on the VIST simulator, where the agent observes a 23-dimensional state representation capturing relevant path-following information. The agent was rewarded for path following, with the reward function incorporating vessel centerline alignment, penalization of deviation from the path, and progression toward the target. The agent produces continuous translation and rotation commands for the respective tools. Training was conducted on four anatomies over 1600 episodes, each consisting of up to 500 time steps depending on task completion.

The best performing model emerged after 800 training episodes, which achieved a success rate of 94 % on validation data. During testing, it achieved a 96 % success rate on an unseen cerebral anatomy and an 88 % success rate on an unseen liver anatomy. This indicates that the agent learned transferable navigation strategies rather than anatomy specific memorization, which demonstrates that deep reinforcement learning is a viable approach for endovascular navigation in a simulated environment.

Keywords: DRL, SAC, Navigation, Wire, Catheter, Endovascular, VIST, Robotics

Acknowledgements

Firstly, we would like to thank Mentice AB for the opportunity to carry out our Master's thesis at their company. Secondly, we would like to thank our supervisor Martin Ekerstedt for helping us along the way and leading us in the right direction.

Niklas Magnusson, Lukas Pettersson, Gothenburg, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
API	Application Programming Interface
CT	Computed Tomography
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q Network
EVT	Endovascular Thrombectomy
GRPC	Google Remote Procedure Call
LLM	Large Language Model
MDP	Markov Decision Process
PCI	Percutaneous Coronary Intervention
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SOFA	Simulation Open Framework Architecture
VIST	Vascular Interventional System Trainer

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i	Index for critic networks
j	Index for target networks
t	Index for time step

Sets

$\mathcal{D}$	Replay buffer
$\mathcal{A}$	Action space, set of all actions
$\mathcal{S}$	State space, set of all states
$\mathcal{B}$	Mini-batch sampled from the replay buffer

Parameters

γ	Discount factor
τ	Polyak averaging coefficient
ϕ	Actor parameters
θ	Critic parameters
$\bar{\theta}$	Target critic parameters
α	Entropy coefficient
ψ	Angle
η	Learning rate
$\bar{\mathcal{H}}$	Target entropy

Variables

s	State, $s \in \mathcal{S}$
a	Action, $a \in \mathcal{A}$
r	Reward
μ	Mean
σ	Standard deviation
$\mathbb{E}$	Expected value
ξ	Gaussian noise
l	Interpolation distance

Functions

$Q(s, a)$	Action-value function (critic)
J	Objective function
L	Loss function
π	Policy

Contents

List of Figures

List of Tables

List of Algorithms

1
Introduction

This chapter is an introduction to the project. It includes a general background of endovascular procedures including the relevant issues and possible solutions. Furthermore, the section introduces similar work, the project's novelty and limitations.

1.1 Background

Endovascular surgery is an alternative to traditional open vascular surgeries to minimize invasiveness, which results in faster recovery, less blood loss, and ultimately less patient morbidity [1]. Instead of incisions to reach blood vessels during surgery, the procedure is done by puncturing the artery with a small needle and inserting catheters to reach the damaged area [2].

The femoral artery in the leg is the most common choice of access point although it is possible to carry out the procedure from other arteries such as the radial artery in the arm [3]. The tools most commonly used to navigate in these kinds of surgeries are guide wires, catheters and sheaths [4]. The guide wire is the first tool introduced and can come in different tip geometries and thicknesses ranging from straight and 0.010 inches in diameter to J-shaped and 0.052 inches. It can also be substituted during the procedure.

The choice of wire diameter depends mostly on the radius of the vessel it is deployed in [5]. The smaller the vessel is, the thinner and softer the chosen wire has to be. When the wire has accessed the artery, a sheath is introduced over the wire and lastly a catheter [4]. To navigate further, the catheter injects contrast to identify the tool's position relative to the vessel structure. With the wire as a guide, the tools move along the vessels until they reach a location where the sheath can be placed in a secure stationary position. From here, the sheath can act like a safe canal if the tools need to be changed and a stable point for further movement in the vessel tree.

Endovascular surgery is a method used during treatment of multiple types of artery complications. Percutaneous coronary intervention (PCI) is a procedure to treat coronary artery disease [6]. During PCI, a small balloon is guided to a narrowed coronary artery and thereafter inflated to expand the vessel and restore a normal blood flow. After the balloon is withdrawn, a stent takes its place which keeps the coronary artery fully expanded and able to function as normal. Endovascular

thrombectomy (EVT) is a procedure to treat acute ischemic stroke [7]. A catheter is guided through the blood vessels to the blocked artery in the brain and a stent retriever is then used to remove the blood clot in the artery which restores the blood flow.

Endovascular procedures are inherently complex, require specialized training, and are heavily dependent on personal skills and experience [8] [9]. It can be difficult even for experienced clinicians to find the optimal pathway for intervention due to the complexity of the vascular system. Furthermore, endovascular procedures are performed under fluoroscopic guidance, which exposes clinicians to ionizing radiation [10]. Along with being an occupational hazard, exposure prevention necessitates heavy lead clad protective gear, which creates ergonomic problems.

Robot-assisted endovascular surgery is a potential solution to these problems as it could lead to greater efficacy and less radioactive exposure to cardiologists [10]. Algorithms that introduce guiding assistance and semi-autonomous features could also help with decision-making in complex endovascular environments [8]. Using simulation software, such algorithms could be created using trial and error through the process of reinforcement learning [11]. Reinforcement learning consists of many algorithms and implementations. An example is CleanRL, which is a deep reinforcement learning library [12].

1.1.1 Simulation

Mentice AB is a company that has developed a simulated setting for interventional procedures such as endovascular surgery [13]. The software, VIST, is an image-guided procedural virtual reality trainer for clinicians and medical professionals. In the simulated environment, it is possible to navigate with tools such as a catheter or a guide wire through a realistic 3D representation of human anatomies with a 2D x-ray as a guide [14]. The simulation is also able to provide real-time information of the procedure such as position of the tool relative to the environment.

1.2 Aim

The aim of this thesis is to develop an algorithm for autonomous navigation in an endovascular environment using reinforcement learning. The goal is to produce an algorithm that is capable of navigation in a complex environment, such as in cerebral arteries. Furthermore, the thesis will assess the viability and limitations of reinforcement learning for generalizable AI agents in this domain.

1.3 Limitations

- This project is first and foremost focused on navigation. This means that safety considerations will not be taken into account in the algorithm.
- This project assumes certain prerequisites. For example, it relies on a preexisting path finding algorithm and patient anatomy is assumed to be completely stationary.
- Cerebral navigation is the primary focus. As the tools are placed in the carotid arteries, previous navigation from the access point is assumed.
- Only three tools are used during training and validation. A micro wire, a micro catheter and a sheath. The sheath is not a part of the navigation, but placed as support. It is the wire and catheter which are controlled by the AI agent.
- The target points used in training and testing are arbitrary and lack medical motivation. The tools used might not be the optimal real world tool-set for certain target positions.

1.4 Novelty

The novelty of this study is twofold. Firstly, the Mentice VIST software ensures a high fidelity simulation environment. To the knowledge of the authors, there is no previous research where a state of the art simulation as VIST is used in this kind of autonomous navigation task. Secondly, this study is attempting autonomous navigation in complex bifurcation dense environments, whereas previous studies have relied on simple anatomy models.

1.5 Similar works

In the research paper Sim4EndoR, Yao et al explore the viability of different deep reinforcement learning algorithms in an endovascular environment [11]. The research team used vascular phantoms, which were obtained through CT scanners from real anatomies. The study used two different target points and results showed that both DDPG and SAC algorithms reached an 80 percent success rate, in ten trials. Both algorithms successfully navigated to target point A with a 100 % accuracy and to target point B with 60 % accuracy. The paper concludes that future endeavors should concentrate on more sophisticated anatomies with more variability.

Endovascular navigation can also be done through an image to action approach. In the paper by Ziyang Mei et al, a combination of residual neural networks and visual transformers were used to extract features from images [15]. These features were fed into an actor critic algorithm which learned a policy that could navigate a wire to a desired target position.

Another example of successful navigation in a simulated endovascular environment is the research paper by Robershaw et al [16]. The paper aimed to build on the

author's previous work, where successful navigation to the internal carotid artery was achieved with a catheter. The purpose was therefore to navigate with a micro catheter and wire to the middle cerebral arteries using the SAC algorithm. The simulation software used was simulation open framework architecture (SOFA), which is an open source software specifically aimed towards medical simulations [17]. To ensure a high fidelity simulation the attributes of real world tools were tested with a tensile testing machine [16].

The study Multi-Agent Fuzzy Reinforcement Learning with LLM for Cooperative Navigation of Endovascular Robotics also used SOFA as the foundation for the simulation [18]. Tianliang Yao et al used reinforcement learning to train two separate agents to control a wire and catheter respectively. Furthermore, an LLM was used to generate context-aware action generation, which would help the two agents cooperate.

2
Theory

In this chapter the central concepts of the work will be introduced. The section opens with general reinforcement learning theory and jargon, and progresses to detailed theory of the SAC algorithm. Lastly, miscellaneous theory is included.

2.1 Reinforcement learning

Reinforcement learning is an umbrella term for algorithms in machine learning that utilizes an agent to explore an unknown environment, with a scalar reward as feedback [19]. The goal of the agent is to maximize the reward in the long run. The advantage of this approach is that the agent can learn from its environment, which relieves the necessity of decision making for every possible outcome.

The central concepts in reinforcement learning are the agent and the environment [19]. The environment is the world the agent inhabits and interacts with. At every step, the agent sees an observation of the environment, takes an action and is given a reward signal which determines how good or bad the current state is, as can be seen in figure 2.1. The aim of the agent is to maximize this reward and learn different behaviors to reach the goal.

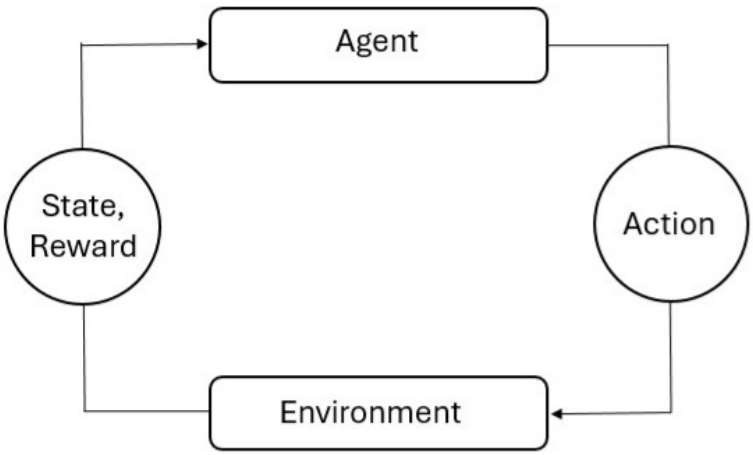

Figure 2.1: Shows the feedback loop between the agent and the environment.

The state S is a representation of the world the RL agent occupies [19]. It contains everything that is important for the problem that is modeled. In deep reinforcement learning, the state is often represented as a vector. From the state, the agent learns to take actions in the environment. The set of actions available to the agent is called the action space, which can be both discrete and continuous.
A policy is a rule which decides the state-action relationship. It is a computable function that for each state outputs an action. For deep reinforcement learning, this function is approximated through neural networks [20]. A policy is usually denoted as π.

Reinforcement learning algorithms can be model-based and model-free [19]. Model-based algorithms require knowledge of the environment. For example, complete knowledge of a system can allow an agent to plan and think ahead. Model-free algorithms learn from the environment. This type of algorithm has no prior knowledge of the environment but learns to estimate it while training. Model-free algorithms are generally what make up reinforcement learning.

2.1.1 Markov Decision Process

A common way to model reinforcement learning is through a Markov Decision process [19], which is formulated as:

$$M = \langle S, A, T, R \rangle$$

Where

- S is the state space, which contains all that is important for each state given the problem.
- A is the action space, which is used to control the state space.
- T is the transition function. By applying an action a_t in the state s_t, the system makes a transition to a new state s_{t+1}.
- R is the reward function, which specifies the reward for a given state or action.

MDPs generally compute optimal policy through a value function [19]. The value function is a measure of how good a state is to be in for the agent, with the measure of good being determined by the expected return. The value of state s_t under policy π is denoted $V_\pi(s_t)$ and is the expected return when starting in state s_t and following policy π thereafter. A fundamental property of the value function is that it satisfies a recursive relationship, which can be defined by the *Bellman equation* as seen in equation 2.1 [19]. The discount factor γ controls if immediate or future rewards are prioritized.

$$V_\pi(s_t) = \sum_{s_{t+1}} T(s_t, \pi(s_t), s_{t+1}) \left(R(s_t, a_t, s_{t+1}) + \gamma V_\pi(s_{t+1})\right) \tag{2.1}$$

Where the optimal policy is defined by the Bellman optimality equation defined in equation 2.2 [19].

$$V^*(s_t) = \sum_{s_{t+1}} T(s_t, \pi(s_t), s_{t+1}) \left(R(s_t, a_t, s_{t+1}) + \gamma V^*(s_{t+1})\right) \tag{2.2}$$

Another optimal state action value function is seen in 2.3 known as Q-functions [19]. These functions are useful because they do not need the transition function. Since T is often unknown, Q-functions are helpful.

$$Q^*(s_t, a_t) = \sum_{s_{t+1} \in S} T(s_t, a_t, s_{t+1}) \left(R(s_t, a_t, s_{t+1}) + \gamma V^*(s_{t+1})\right) \tag{2.3}$$

Although the definition of the Q-function depends on the transition function T, it can be learned in a model-free manner using sampled transitions from the environment [19]. This removes the need for explicit knowledge of the transition dynamics, making Q-learning methods particularly useful in environments where T is unknown.

2.1.2 Learning with model-free algorithms

There are generally two ways to train model-free algorithms, policy optimization and Q-learning. Policy optimization follows a policy $\pi_\phi(a_t \mid s_t)$ which produces an action a_t given the state s_t and aims to optimize the parameter ϕ through gradient descent [21]. This approach is usually on-policy, which means it uses the latest policy and current sampled data to update the parameters.

Q-learning algorithms learn an approximator $Q(s_t, a_t)$ for the optimal action value pair $Q^\star(s_t, a_t)$ [20]. The Q-values are the expected long-term reward of executing an action in a given state and a higher Q-value is indicative of a higher future reward [22]. Q-learning is based on the Bellman optimality equation, which gives the expected return when following an optimal policy for a state and an action, where the optimal action is $a_\star(s) = argmax Q_\star(s_t, a_t)$ [20]. Q-learning learns off-policy, which means it tries to optimize some optimal policy $\pi^\star$, while following another policy π [19].

Traditional Q-learning learns through iterative updates of the Q-value [23]. For many problems there are far too many states to maintain an estimate for each state action relationship. Instead a neural network with parameters θ can be used to approximate the Q-values. This is known as deep Q-networks (DQN) and such Q-values are denoted as $Q_\theta(s_t, a_t)$. While DQN can handle high dimensional observation spaces, it can only handle discrete action spaces. Because DQN relies on finding the action that maximizes the Q-value, continuous action spaces cannot readily be used. This would require an optimization routine at each timestep, which would be computationally infeasible .

To combat this, it is possible to combine policy optimization with Q-learning, for example with an algorithm called Deep Deterministic Policy Gradient (DDPG) [23]. DDPG addresses the problem with continuous action spaces by assuming that the function $Q(s, a)$ is differentiable with respect to the action. This allows the algorithm to avoid an exhaustive search by training a policy network $\pi_\phi(s)$ to approximate the best action. Instead of solving an optimization problem at every step, the agent evaluates the policy network and uses gradient-based updates to improve the policy by following the direction that increases the Q-value.

DDPG is an off-policy algorithm like Q-learning, which is manifested through the *replay buffer* that collects previous experience from the agent [23]. This allows the algorithm to benefit from learning from a large array of uncorrelated transitions. DDPG uses a deterministic policy, where a state is directly mapped to an action $a = \pi_\phi(s)$ [24].

2.1.3 Replay buffer

As mentioned, the replay buffer $\mathcal{D}$ allows Q-learning algorithms to learn *off-policy* [23]. In practice this means that for each iteration, information about the state and actions are saved, which is then sampled from when training. The state action transition is defined as the following tuple.

$$(s_t, a_t, r_t, s_{t+1}, d) \in \mathcal{D}$$

In practical deep reinforcement learning implementations, replay buffers store an explicit *done* flag, which indicates a zero value state known as a terminal state. The done flag tells the agent that there are no further future rewards in the state s_{t+1} [25].

2.1.4 Actor-critic networks

Actor-critic is a common method for continuous action space algorithms like DDPG and SAC. [24]:

- **Actor network** $\pi_\phi(s_t)$: Takes a state s_t, outputs an action a_t. The network learns to find the optimal policy that maximizes future rewards.
- **Critic network** $Q_\theta(s_t, a_t)$: Takes the state and action and determines the Q-value, e.g the expected return from the state action pair while following π. Its purpose is to evaluate the current actor policy.

Actor-critic algorithms also generally utilize target networks [26]. While the target networks are copies of the actor and critic, they lag these networks and therefore have different parameters $\bar{\theta}$. The difference in the parameters stabilizes the training and prevents oscillation and divergence. The target networks are generally updated through the following equation [27]:

$$\bar{\theta} \leftarrow \tau\bar{\theta}(1 - \tau\theta) \tag{2.4}$$

The update is known as polyak averaging, which smoothly updates the target network, as seen in equation 2.4 [23]. The hyperparameter τ is between 0 and 1, but generally $\tau \ll 1$

2.2 Soft Actor-Critic

Soft Actor-Critic (SAC) is a DDPG-style algorithm [28]. It is an off-policy algorithm like DDPG and has an actor-critic structure. While DDPG uses a strictly deterministic policy, SAC uses entropy regularization to balance exploration and exploitation to avoid prematurely converging to a suboptimal policy. The SAC algorithm learns a stochastic policy π, which samples from an action distribution from the actor network. The network outputs a mean and a standard deviation as seen in equation 2.5 [20].

$$\tilde{a}(s_t) = \tanh\Big(\mu_\phi(s_t) + \sigma_\phi(s_t) \odot \xi\Big), \qquad \xi \sim \mathcal{N}(0, I). \tag{2.5}$$

The variable $\tilde{a}_t$ is a deterministic and differentiable sampled action [28]. It is expressed as a function of the current actor network's output mean μ_ϕ and standard deviation σ_ϕ given the state s_t and an independent noise variable ξ. This reparameterization preserves the stochasticity required for exploration while allowing gradients to propagate through the action and enabling efficient optimization of the actor network [29].

2.2.1 Entropy regularization

SAC uses entropy regularization to avoid premature convergence to a suboptimal deterministic policy [28]. The policy is trained to balance expected return and entropy, where a higher entropy corresponds to more randomness in the actions, and therefore more exploration. The entropy contribution appears in the objective through the term $\alpha \log \pi(a_t \mid s_t)$, where $\log \pi(a_t \mid s_t)$ denotes the log-probability density of sampling action a_t given state s_t under the policy π_ϕ. The temperature parameter α determines the relative importance of entropy compared to the expected reward.

The entropy term determines the trade-off between exploration and exploitation [27]. When α is large, the policy becomes more stochastic which leads to more exploration and poor exploitation of the reward signal. For a small α value, the policy learns quickly at first but will become deterministic and potentially converge to a suboptimal local minimum. To overcome this the concept of auto-tuning is introduced, which automatically tunes α to balance exploration and exploitation. Auto-tuning can be achieved by treating α as trainable parameter as seen in equation 2.6 [27]. The function J is called the objective function, which refers to the error that we are trying to minimize [19].

$$J(\alpha) = \mathbb{E}_{\tilde{a}_t \sim \pi_t}\Big[-\alpha \log \pi_\phi(\tilde{a}_t \mid s_t) - \alpha\, \bar{\mathcal{H}}\Big] \tag{2.6}$$

In this equation, $\bar{\mathcal{H}}$ is the entropy target value which is often set to the dimension of the action space $\mathcal{A}$ in order to maintain a suitable level of exploration [27].

This equation is a theoretical objective due to the continuous and high dimensional action and state spaces. As a result, the expected value $\mathbb{E}$ of the right hand side cannot be computed exactly in practice. The right hand side can instead be estimated by sampling a batch $\mathcal{B}$ from the replay buffer $\mathcal{D}$ and calculating the estimated mean value of the equation [30]. This results in a loss function that tunes the parameter α with gradient descent as seen in equation 2.7.

$$L(\alpha) = \frac{1}{|B|} \sum_{(s) \in B} \left[-\alpha \log \pi_\phi(\tilde{a}_t \mid s_t) - \alpha \bar{\mathcal{H}} \right] \tag{2.7}$$

2.2.2 Updating the critic networks

Equation 2.8 shows the soft Q-function. This equation aims to minimize the loss between the neural network approximation $Q_\theta(s_t, a_t)$ with the estimated value $\hat{Q}(s_t, a_t)$, where $\mathcal{D}$ is the replay buffer.

$$\begin{gathered} J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t) \right)^2 \right], \\ \text{where } \hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \, \mathbb{E}_{s_{t+1} \sim p} \left[V_{\bar{\psi}}(s_{t+1}) \right]. \end{gathered} \tag{2.8}$$

The loss function used in practice can be seen in equation 2.9 [31]. Two critic networks are used to reduce Q-value overestimation, which is known to destabilize training [28][32]. Furthermore, replay buffer batch samples $\mathcal{B} \in \mathcal{D}$ are used for the same reason as in section 2.2.1. Here slightly different notation is used for the estimated target y with the inclusion of the done flag as seen in equation 2.10.

$$L(\theta_i) = \frac{1}{|B|} \sum_{(s_t, a_t, r, s_{t+1}, d) \in B} \left(Q_{\theta_i}(s_t, a_t) - y(r, s_{t+1}, d) \right)^2, \qquad i = 1, 2, \tag{2.9}$$

$$y(r, s_t, d) = r + \gamma(1 - d) \left(\min_{j=1,2} Q_{\bar{\theta}_j}(s_t, \tilde{a}_{t+1}) - \alpha \log \pi_\phi(\tilde{a}_{t+1} \mid s_t) \right) \tag{2.10}$$

2.2.3 Updating the actor network

The objective function for the actor is shown in equation 2.11 [28]. The aim of the objective function is to maximize the Q-function while simultaneously maximizing the entropy to find an optimal policy that still encourages exploration.

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D},\, \tilde{a}_t \sim \pi_\phi} \left[\alpha \log \left(\pi_\phi(\tilde{a}_t \mid s_t) \right) - Q_\theta(s_t, \tilde{a}_t) \right] \tag{2.11}$$

To prevent overestimation of the Q-value, the minimum of the two critic estimates is chosen. For the same reason as in section 2.2.1, the expected value is estimated using a batch $\mathcal{B} \in \mathcal{D}$, which is used in the practical loss function shown in equation 2.12. While the states are sampled from the replay buffer, the actions $\tilde{a}$ are sampled from the current policy. This ensures that gradients are computed with respect to the latest policy parameters.

$$L_\pi(\phi) = \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} \left[\alpha \log \left(\pi_\phi(\tilde{a}_t \mid s_t) \right) - \min_{j=1,2} Q_{\theta_j}(s_t, \tilde{a}_t) \right] \tag{2.12}$$

2.3 Adam optimizer

The optimization of parameters is important in many fields in science and engineering and is of particular importance in machine learning, which is sensitive to hyperparameter tuning [33]. Because different hyperparameter values are beneficial in different scenarios, adaptive parameter tuning is introduced. One such method is Adam, which updates parameters by maintaining moving averages for both the gradients and squared gradients. This allows Adam to adapt the parameter based on historical gradient behavior.

$$\theta_t \leftarrow \theta_{t-1} - \eta \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) \tag{2.13}$$

The gradient and squared gradient are used to calculate the *moments* m_t and v_t respectively. The moments are used in the update rule to optimize the parameter as seen in equation 2.13.

2.4 Normalization

When working with gradient based learning such as neural networks, normalization of the input data is important [34]. Because agents in reinforcement learning use neural networks to estimate policies, it is important that all state variables have similar scale. To achieve this Z-score normalization could be used.

$$x' = \frac{x - \mu}{\sigma}$$

μ is the mean, and σ is the standard deviation of the feature.

2.5 Approximation of Bend Energy

A polyline behaves differently depending on how bent it is [35]. To give a metric for the magnitude of the total bend in a polyline, the bend energy can be calculated with use of the curvature k. For a continuous wire rod, the equation for the bend energy is shown in equation 2.14.

$$E = \int \kappa(s)^2 ds \tag{2.14}$$

By the use of approximations, a discrete implementation of bend energy can be derived to the function in equation 2.15 with a comparison of the functions shown in figure 2.2.

$$E \approx \sum_i \psi_i^2 \tag{2.15}$$

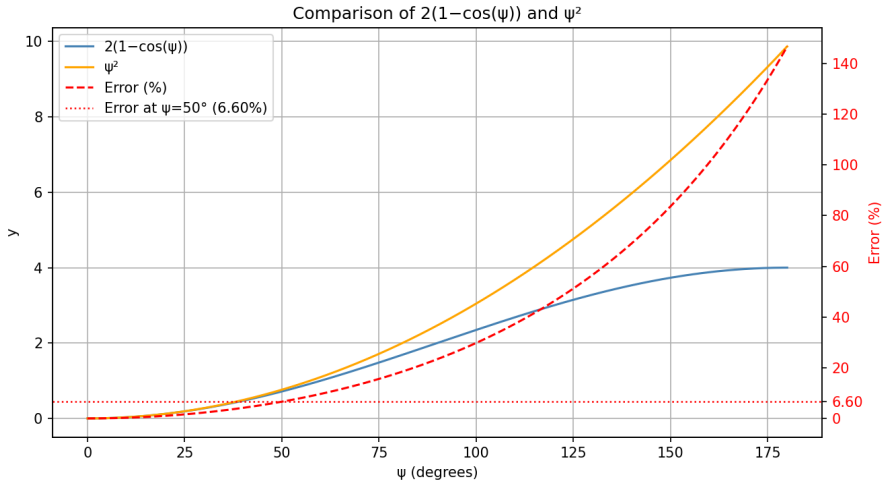

Figure 2.2: Comparison of k^2 and ψ^2 including the error growth and specific error at a 50 degree angle.

3
Methods

The endovascular navigation task was addressed by training an AI agent using the Soft Actor-Critic (SAC) algorithm, based on the CleanRL implementation [12]. The environment was provided by the VIST simulation software, which enabled a realistic vascular representation. The SAC algorithm was implemented using Python.

The agent learned to control a guide wire and catheter to reach predefined target points. A structured state representation based on vessel geometry, tool positioning, and a precomputed path was used, while a multi-component reward function encouraged progression, alignment, and safe navigation. Training was performed on synthetic cases, with validation used to select the best performing model and testing conducted on unseen anatomies to evaluate generalization.

3.1 Experimental setup

The training of the algorithm was conducted on four unique synthetic case anatomies. Each case was used for 100 episodes of training before changing to the next case. When the last case was reached, it started over on the first case and cycled through them again. Furthermore, each case consisted of five predetermined target points. During the 100 episodes, one of the five target points was chosen at random. For each anatomy, the target points had the same initial starting position. The path is calculated from this position to the chosen target position. To illustrate this setup, the paths from case 1 can be seen in figure 3.1.

During the exploration period of 5000 steps, a single target point in the first case was chosen. This target point was positioned closer to the starting location than all other points and prior to any bifurcations, in order to ensure the presence of positive rewards in the replay buffer. It was observed that terminal states in the replay buffer when training started significantly sped up convergence.

Model checkpoints were saved every 50 episodes. After the training was completed, each saved model was tested on two synthetic validation cases with five target points each. To produce more robust statistics, the validation was performed five times on each case. These five runs were then averaged to produce mean statistics for the respective cases.

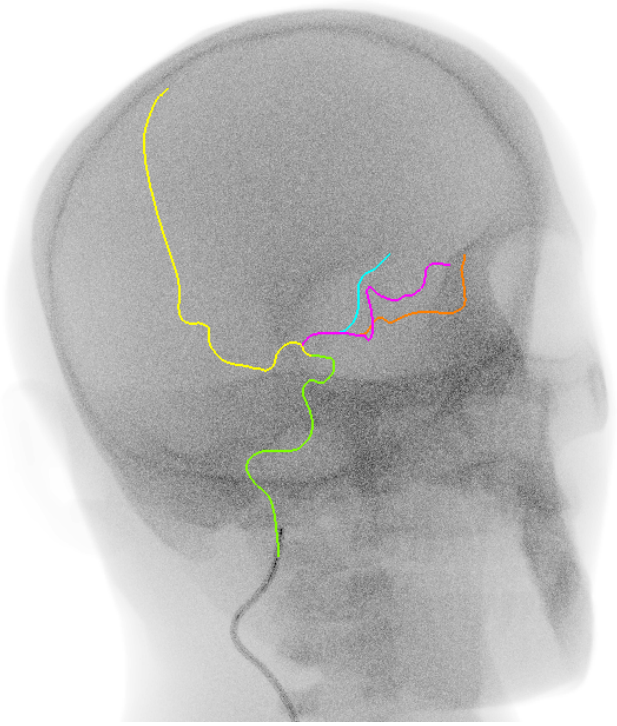

Figure 3.1: Shows the five paths for case 1 along with the tools. All paths have a common start where the green path starts. The target points are at the end of each respective path.

The model with best validation results was chosen for the test set, which consisted of two cases. Case A contained an anatomy modeled from CT-scan data of a real patient. Case B was an anatomy containing the liver structure. Five target points were chosen in each case and each was tested 5 times by the final model.
The training, validation and testing data split can be seen in figure 3.1

	Synthetic/Patient	**Location**	**Cases**	**Target points**
Training	Synthetic	Brain	4	20
Validation	Synthetic	Brain	2	10
Test Case A	Patient	Brain	1	5
Test Case B	Synthetic	Liver	1	5

Table 3.1: Data split overview.

3.1.1 Connecting VIST and Python

While the simulation was running, gRPC calls were used to access the VIST APIs. Through this interface, the state was constructed at each time step. The information was transmitted locally on the same machine, where VIST acted as the host and the Python script as the client.

3.1.2 Initialization of the system and specific modifications

In these steps, the simulation was primed before beginning the training. This included defining hyperparameters, placing the correct tools in the correct positions and computing the target paths as seen in figure 3.1. All cases used the femoral artery as the access point and the tools used can be seen in table 3.2. Since Case B with the liver anatomy contained vessels with larger diameters, the tools chosen for this case are larger as well and can be seen in table 3.3.

Table 3.2: Shows the specific tools used in the simulation during training, validation and testing of Case A

Tool	Dimension (Inch ")	Angle (Degrees °)
Guide wire	0.010	90
Micro catheter	0.014	90
Sheath	0.078	Straight

Table 3.3: Shows the specific tools used in the simulation during testing of Case B

Tool	Dimension (Inch ")	Angle (Degrees °)
Guide wire	0.018	90
Micro catheter	0.025	90
Sheath	0.078	Straight

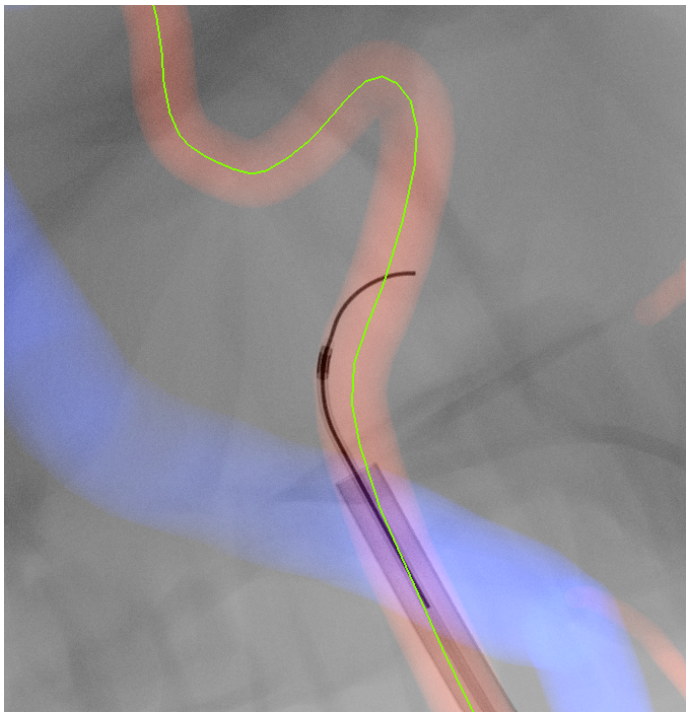

Figure 3.2: Simulation setup used for SAC training. The three tools from table 3.2 can be seen in their initial positions along with the path in green. The wire can be seen protruding from the catheter, which in turn protrudes from the sheath.

The guide wire was designated as the primary navigation tool guiding the catheter. Consequently, only the wire reaching the target was considered a successful navigation, which was chosen to be 0.5 centimeters from the target. The Sheath was only used as support for the guide wire and catheter, as distal navigation without a sheath increases vessel friction, which hinders tool advancement.

3.2 State

The state was constructed using information important for teaching the model to follow a designated path from a starting point to a target point for each episode. The path was obtained via a gRPC call to the VIST software, which computed the shortest path between the two points. The resulting path consisted of points along the centerline of each vessel and was further interpolated with a distance of $l = 0.02\,\text{cm}$ to improve consistency in the state representation. At each step, all state parameters were normalized using a running normalizer, which collected state samples over time to estimate the mean and standard deviation of each state dimension. The full state representation comprised 23 dimensions and is shown in Table 3.4.

Table 3.4: Shows the dimensions each state occupies. The entries marked with *near* are the values at the tool's position, while *future* is defined as 25 interpolated points ahead, which amounts to 0.5 centimeters in the simulation.

Dimension	State
0	vessel_radius_near_wire
1	curvature_near_wire
2	vessel_radius_future_wire
3	curvature_future_wire
4	alignment_wire
5	deviation_wire
6	offset_normal_wire
7	offset_binormal_wire
8	alignment_cath
9	deviation_cath
10	offset_normal_cath
11	offset_binormal_cath
12	wire_movement
13	cath_movement
14	wire_rotation_change
15	cath_rotation_change
16	wire_bend_energy
17	cath_bend_energy
18	distance_between_devices
19–22	previous_action

3.2.1 Directions

The direction in each path point $\mathbf{P}_i$ was computed by normalizing the vector to the next point $\mathbf{P}_{i+1}$ using the formula in equation 3.1

$$\vec{u_i} = \frac{\mathbf{P}_{i+1} - \mathbf{P}_i}{l} \tag{3.1}$$

3.2.2 Curvature

The curvature in each path point $\mathbf{P}_i$ was computed by calculating the length of the vector difference between the direction vector for point $\mathbf{P}_{i+1}$ and $\mathbf{P}_{i-1}$ divided by the distance as shown in equation 3.2

$$\kappa_i = \left\| \frac{\vec{u}_{i+1} - \vec{u}_{i-1}}{2\,l} \right\| \tag{3.2}$$

3.2.3 Alignment

The alignment of the tool relative to the path was computed using the dot product between the direction of the tip of the tool $\vec{u}_{tool}$ and the direction of the closest point i on the path $\vec{u}_i$. The computation is shown in equation 3.3.

$$alignment = \vec{u}_{tool} \cdot \vec{u}_i \tag{3.3}$$

3.2.4 Deviation

The deviation is the distance between the tip of the tool $\mathbf{P}_{tool}$ and the closest point on the path $\mathbf{P}_i$ in space. The computation is shown in equation 3.4

$$deviation = \|\mathbf{P}_{tool} - \mathbf{P}_i\| \tag{3.4}$$

3.2.5 Normal and Binormal deviations

The normal and binormal deviations were computed using the directions of the closest point $\vec{u}_i$ to the tool and next point $\vec{u}_{i+1}$. The normal vector is the change in direction and is shown in equation 3.5. The binormal vector is orthogonal to both the direction vector and the normal vector and is calculated as in equation 3.6.

$$\vec{n}_i = \frac{\vec{u}_{i+1} - \vec{u}_i}{\|\vec{u}_{i+1} - \vec{u}_i\|} \tag{3.5}$$

$$\vec{b}_i = \vec{u}_i \times \vec{n}_i \tag{3.6}$$

To calculate the normal and binormal deviations, the deviation vector $\vec{d}$ from the tip of the tool to the closest point seen in equation 3.7 is used for computing dot products in equations 3.8 and 3.9.

$$\vec{d} = \mathbf{P}_{tool} - \mathbf{P}_i \tag{3.7}$$

$$deviaton_n = \vec{d} \cdot \vec{n}_i \tag{3.8}$$

$$deviation_b = \vec{d} \cdot \vec{b}_i \tag{3.9}$$

3.3 Actions

The agent's possible actions corresponded to real life tool mechanics used in endovascular surgery. The agent outputted a translation and a rotation for the wire and catheter respectively, which is modeled as a four dimensional vector. The actions were capped to a maximum relative translation of 0.3 cm and maximum relative rotation of 0.3 rad.

3.4 Rewards

The algorithm's reward function was the sum of six individual rewards. Each reward corresponded to one aspect of path following which taught the agent to navigate

correctly.

$$
\begin{aligned}
R_{goal} &= 50 \\
R_{progress} &= 10 \cdot progress \cdot alignment \\
R_{deviation} &= -0.7 \cdot (deviation_{wire} + deviation_{catheter}) \\
R_{separation} &= -0.2 \cdot device\ separation \\
R_{bend\ energy} &= -0.005 \cdot bend\ energy_{wire} - 0.002 * bend\ energy_{catheter} \\
R_{step} &= -0.5 \\
R_{total} &= 0.1 \cdot (R_{goal} + R_{progress} + R_{deviation} + R_{separation} + R_{bend\ energy} + R_{step})
\end{aligned}
$$

The rewards were calculated and assigned based on information from both the current and previous states. R_{goal} was given when the wire tip was within 0.5 cm of the target point, which also classified the episode as successful. R_{progress} was computed based on the distance progressed along the path by both the wire and the catheter since the last action, as well as their average alignment with the path. $R_{\text{deviation}}$ combined the deviations of both the wire and the catheter from the path. To avoid incorrect penalties when the tool remained within the correct vessel, individual deviations were only included in the negative reward if they exceeded the vessel radius. $R_{\text{separation}}$ was assigned as a negative reward when the Euclidean distance between the tips of the tools exceeded 6 cm. The full rewards and their constraints are presented in Algorithm 2 in the appendix.

3.5 Algorithm

The algorithm was based on CleanRL's Soft Actor-Critic algorithm. There were however a number of changes needed to adapt the algorithm to VIST. For example, all things concerning the environment were custom made, with the rewards and state being custom implementations. Another modification made was adding a forward bias to the exploration, as movement sampled from a uniform distribution resulted in oscillation from the tools, which filled the replay buffer with irrelevant transitions. The main training loop however remained as per CleanRL. The general steps taken in the algorithm can be seen in algorithm 1.

Algorithm 1 SAC algorithm

Declare hyper parameters
Initialize actor ϕ and critics $\theta_{1,2}$
Initialize targets $\bar{\theta}_{1,2}$
Initialize replay buffer $\mathcal{D}$ and entropy α
for each episode **do**
  for each step **do**
    if $t_{Global} < T_{learning}$ **then**
      Take random action with forward bias
    else
      Take action $a_t \sim \pi_\phi(a_t|s_t)$ to state s_{t+1}
    end if
    Add to $\mathcal{D}$ $s_t, a_t, r(s_t, a_t, s_{t+1}, done)$
    if $t_{Global} > T_{learning}$ **then**
      Sample batch $\mathcal{B}$ from buffer $\mathcal{D}$
      Update $\theta_{1,2}$
      Update $\bar{\theta}_{1,2}$
      if $t_{Episode} \equiv 2$ **then**
        Update ϕ
        Autotune α
      end if
    end if
    $t_{Episode} \mathrel{+}= \text{step}$
    $t_{Global} \mathrel{+}= \text{step}$
  end for
end for

3.6 Network architectures

The actor network consisted of 23 inputs, i.e $\dim S$, which fed into two hidden layers with 256 neurons each as seen in figure 3.3. These hidden layers split into two separate action output layers, which predicted action mean and standard deviation.

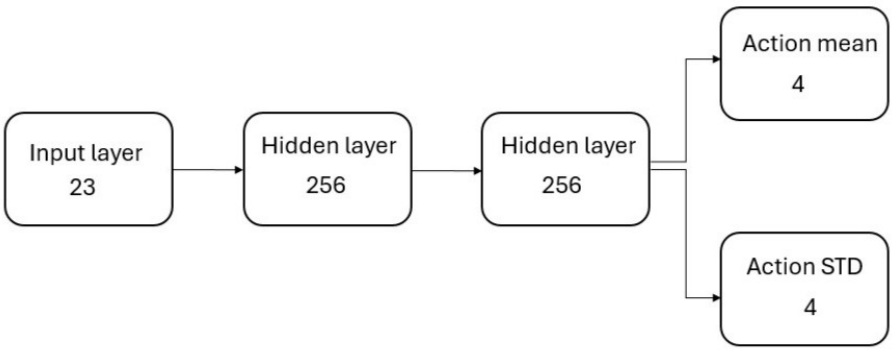

Figure 3.3: Shows the layers of the actor network. The numbers indicate the number of inputs and outputs of each layer respectively.

The critics had input size 27, that is $\dim S + \dim A$. The networks consisted of two hidden layers and one output layer, which outputted a single Q-value as seen in figure 3.4. Both the actor and the critic used ReLU as activation function and the adam optimizer to control the learning rate.

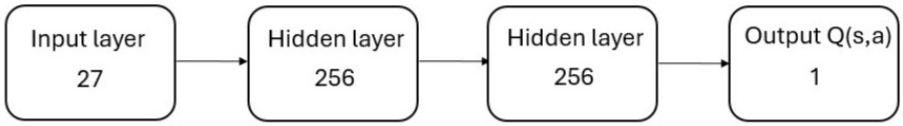

Figure 3.4: Shows the layers of the critic network. The numbers indicate the number of inputs and outputs of each layer respectively.

3.7 Hyperparameters

The hyperparameters used in the algorithm can be seen in table. 3.5.

Table 3.5: Hyperparameters used in the algorithm

Parameter	**Value**
Episode length	500
Episode length validation and testing	1000
Number of episodes training	1600
Learning starts	5000
Replay buffer size	10^5
Batch size	256
Discount factor γ	0.99
Polyak averaging coefficient τ	0.005
learning rate policy η_π	3e-4
learning rate critic η_Q	1e-3
Actor update frequency	Every other step
Target update frequency	Every step
Initial entropy α_0	0.2
Entropy Target $\bar{\mathcal{H}}$	4
State size	23
Action size	4
Action bound	[0.3, 0.3]
Autotune	True
Activation function	ReLU

As many of the hyperparameter values from CleanRL are tested and proven, they were mostly left unchanged. However, there were some changes made to better fit our problem. First off, the max steps per episode was set to 500. This episode length allowed adequate exploration and allowed the agent to reach the target with a margin to spare. It was also short enough that negative rewards were hindered from dominating the replay buffer, as erroneous navigation could lead to incredibly large negative rewards for a large episode lengths. During validation and testing however, the episode length was 1000 timesteps, to avoid premature timeouts.

The learning starts variable was quite low compared to template algorithms. This could be done since the agent would start close to the goal, which means it would fill the replay buffer with terminal states and high rewards during the exploration. This approach was used since vessel navigation is quite difficult and a random policy during exploration is insufficient to find the goal point.

A buffer size of 10^5 was chosen, as it is large enough to capture a wide variety of state action pairs while still being small enough to allow older, less relevant experiences to be discarded over time. This size proved to be a good trade-off.

3.8 Metrics

The models were evaluated using the following metrics.

- Success rate
$$\frac{successful\ attempts}{total\ attempts}$$
- Progression along path
$$\frac{furthest\ point\ reached}{total\ points}$$
- Reward per step
$$\frac{total\ reward}{total\ steps}$$
- Average steps
$$\frac{total\ steps}{number\ of\ episodes}$$

The metrics described above were evaluated for each target in both the validation and test sets. An episode was considered successful if the tip of the guide wire was within 0.5 cm of the target point. Path progression was defined as the furthest point along the path that the wire tip reached while remaining within the correct vessel.

4

Results

4.1 Training results

As seen in figure 4.1, the training achieved high success rate and progress in cycles. It is observed that the case represented by the blue color
performed worse than others with initial declining success rate and progress. Equal performance can be observed for the other regions. The highest rolling average is 100 %, which was achieved approximately at episode 800 and 1600 respectively. The overall success rate across all episodes was 50.9%.

Note that the high early success rate is caused by deliberate adding of terminal states in the replay buffer, as mentioned in section 3.1 . The different colors indicate the different training cases. Also note that the plots show the rolling average over 50 episodes. The red line marks the overall average success rate.

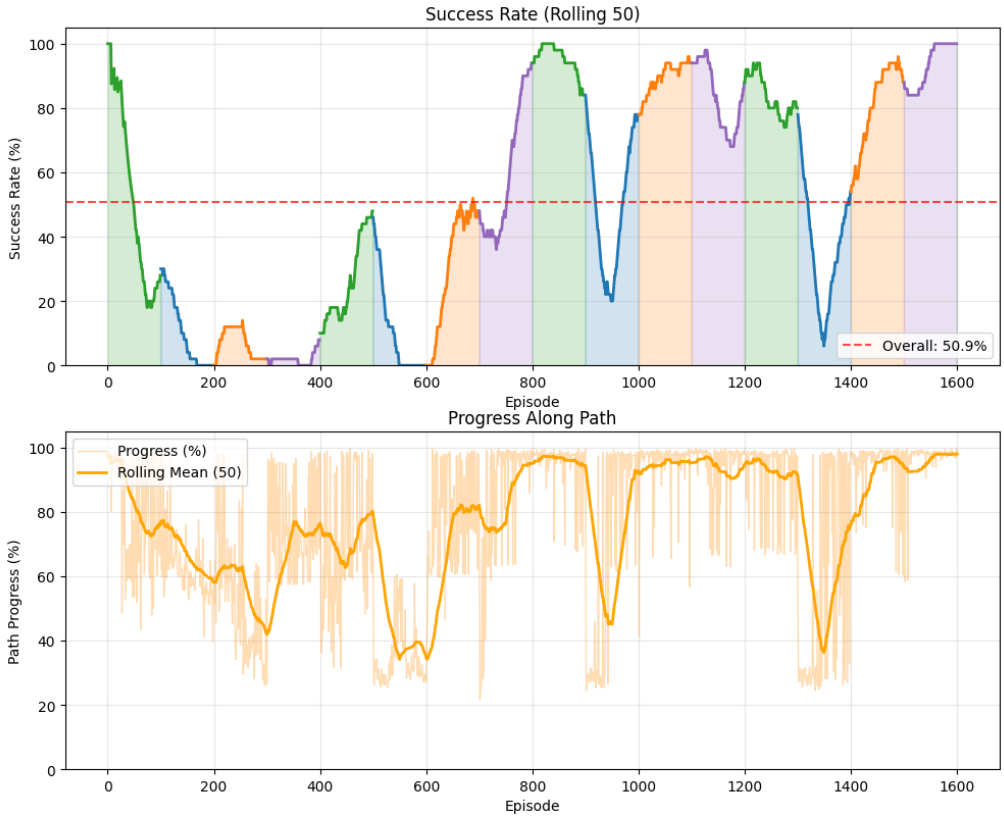

Figure 4.1: The metrics success rate and progress for training.

4.2 Validation results

The validation results show that the model after 800 training episodes produced the best results in all of the measured metrics and was therefore used in the final testing. The model had an average success rate of 94% and an average number of steps taken per episode of 260 as seen in figure 4.2. The average progress and reward per step for the model after 800 training episodes were 93.7% and 0.148 respectively as seen in figure 4.3.

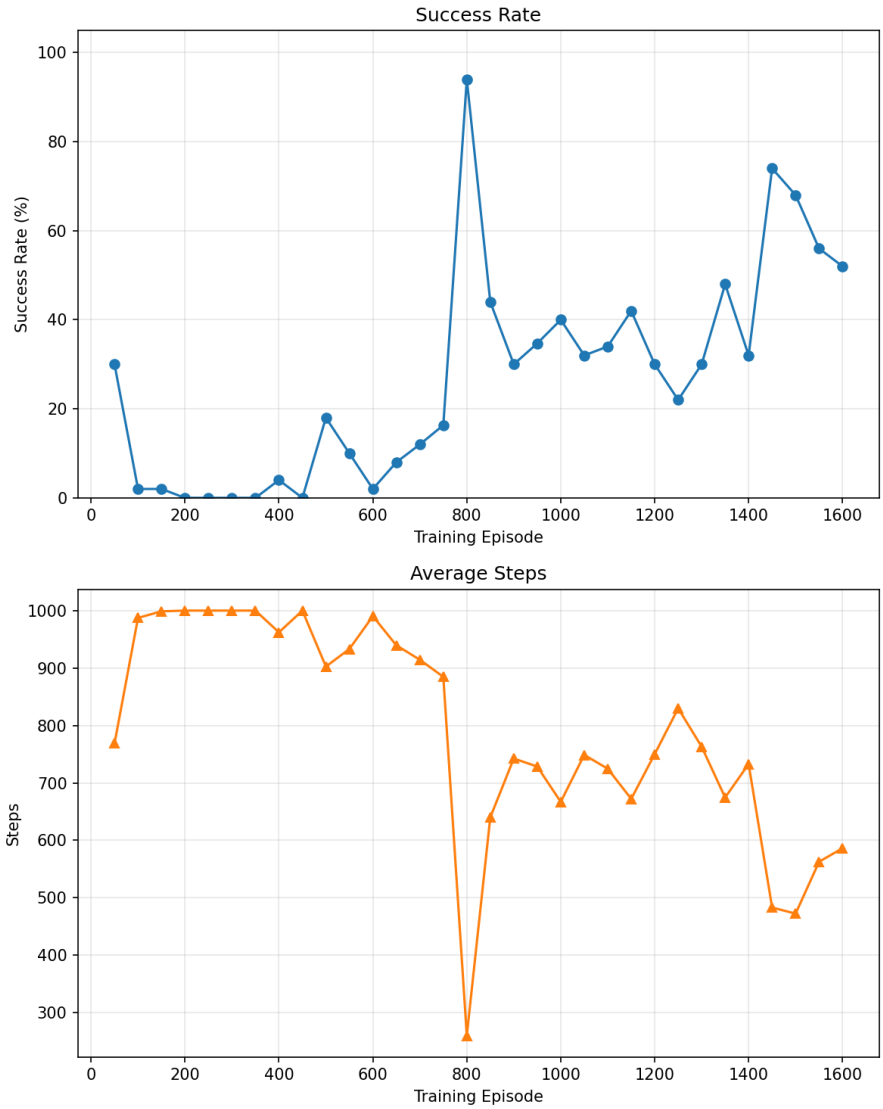

Figure 4.2: The metrics success rate (top) and average number of steps (bottom), evaluated against the number of training episodes.

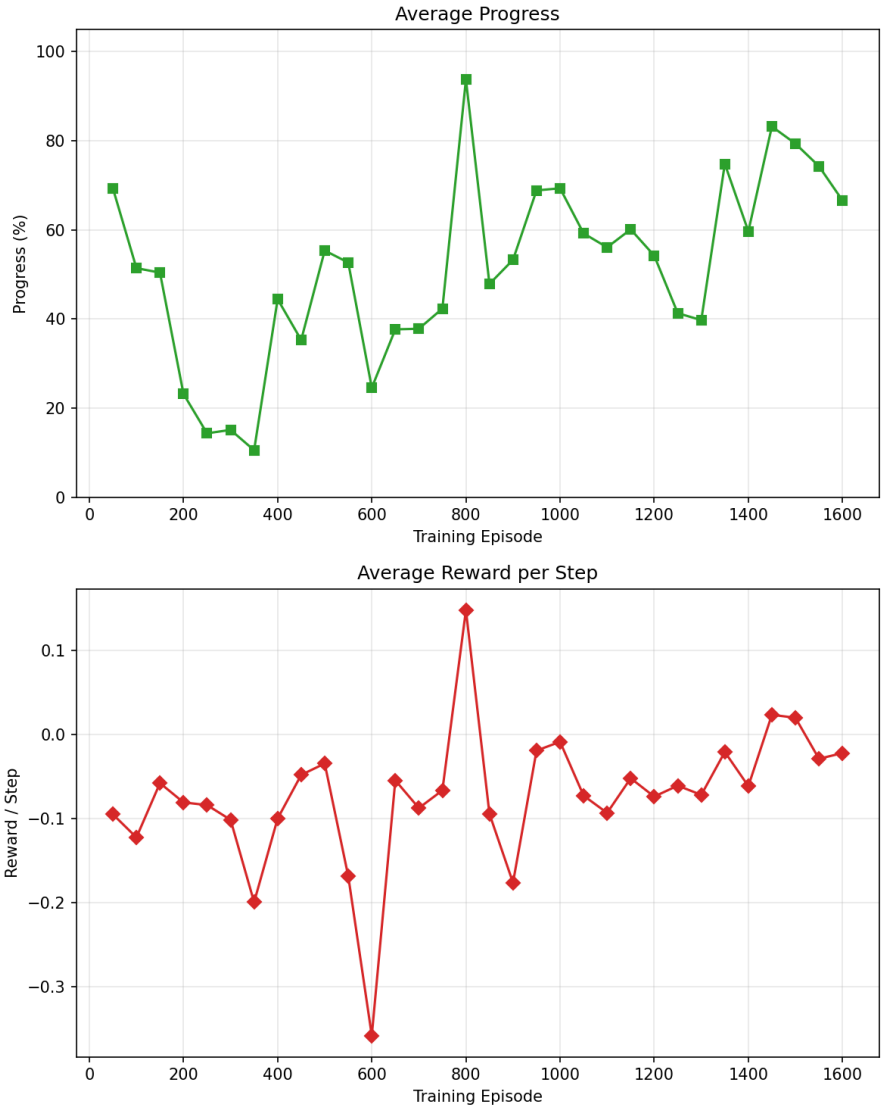

Figure 4.3: The metrics average progress (top) and average reward per step (bottom), evaluated against the number of training episodes.

Metric	Value
Success Rate	96.0%
Avg Progress	$95.3 \pm 4.0\%$
Avg Steps	354 ± 243
Avg Reward per Step	0.128 ± 0.128

Table 4.1: Metrics from Case A containing CT-data from a real patient across five points with five runs each.

4.3 Test results

The model got a 96% success rate on Case A containing data from a CT-scan as seen in table 4.1. The average progress was 95% with an average episode length of 354 steps and average reward per step of 0.061.
The model got a success rate of 88% on Case B containing liver anatomy as seen in table 4.2. The average progress was 92.8% with an average episode length of 421 steps and average reward per step of 0.070.

Metric	Value
Success Rate	88.0%
Avg Progress	$92.8 \pm 10.0\%$
Avg Steps	421 ± 306
Avg Reward per Step	0.070 ± 0.187

Table 4.2: Metrics from Case B containing liver anatomy across five points with five runs each.

5
Discussion

5.1 Discussion of results

During the training, the evaluation on the blue case showed lower performance compared to the other cases as seen in 4.1. The reason for this is not entirely known but the progress plot for the training gives some insight. As soon as the training switched to the blue case the progress dropped to approximately 30%, which might indicate that it introduces challenging navigation in the first third of the paths. After approximately 50 episodes in this case, the model adapts to the environment and overcomes the initial challenges and reaches the targets. When the training switches from the blue case to the next, the newly adapted behavior performs relatively well in the following cases. This changes when reaching the blue case again, where another performance drop occurs. This might indicate that the blue case is important for generalizing the model and learning navigation, but that the blue case destabilizes the performance once the model is overfitted.

The training reached multiple peaks, but this did not result in multiple high performing models, which can be seen by comparing figures 4.1 and 4.2. The most likely explanation is overfitting, that the model learned patterns in the training data and lost the ability to generalize. It is possible that between episode 0 and 800, the model learns to navigate and further training is overfitting to the environment. Since the anatomies are relatively similar, it could be that applying the same actions to each case gives relatively similar results. But when these actions are applied to a new anatomy, or a dissimilar one like the blue case, the actions are not applicable.

To add to this, there is ambiguity to the true highest performer because of the model resolution. Because the validation was time consuming, it was decided to validate every 50th model produced by the training. Figure 4.2 shows the highest performing model within the chosen resolution, but it is entirely possible that an untested model would have performed better.

The test results provide strong evidence of generalization. Although the cerebral anatomy used during testing was unseen during training, the policy still achieved a 96% success rate. This suggests that the agent has learned navigation rather than exploiting specific patterns in the training environments. A potential counterargument is that the training data still contains cerebral vascular anatomy, which could introduce anatomical overlap. However, the training environments were artificially

generated, whereas the test case was reconstructed from real patient CT scans. Furthermore, because the intended application is navigation in cerebral arteries, some similarity in global vessel structure is both expected and necessary for a relevant evaluation. Therefore, while the datasets share general anatomical characteristics, the results indicate that the policy generalizes beyond the specific vessel geometries encountered during training

In contrast to Case A, which evaluated the policy on a cerebral anatomy, Case B consisted of a liver-based vascular anatomy. This vessel structure was outside the distribution of anatomies encountered during training and therefore represents a stronger test of generalization. As shown in table 4.2, the policy achieved a success rate of 88% with an average progress of 92%.

The tools used during training were not suitable for the liver vasculature due to its larger vessel diameters. Therefore, Case B was evaluated using thicker and stiffer tools. Compared to the cerebral anatomy in Case A, the metrics for total success, average progress, and average steps indicate slightly lower performance and higher standard deviations. Nevertheless, the results demonstrate that the policy is still capable of navigating successfully in most cases, although it may require more steps to reach the target. Furthermore, the results in Case B indicate that the model can navigate with more than one specific tool setup, which could broaden the area of use.

5.2 Methodological discussion

There were a number of factors important for the choice of algorithm. The problem required an algorithm capable of handling continuous inputs and outputs, because of the raw data used. In early stages of development the DDPG algorithm was used as it meets the mentioned requirement and certain successes were found. Navigation was possible, but only to one target point at a time as the model failed to generalize. Therefore, it was decided to use the SAC algorithm, which has a number of advantages over DDPG. Firstly, it uses two critic networks to avoid Q-value overestimation, which has a tendency to destabilize training. Secondly, it uses a self tuning stochastic policy, which can balance exploitation and exploration. This means that SAC can better handle local optima and in time converge to a more optimal policy.

Since the SAC algorithm produces a stochastic policy which samples from a distribution, it is worth discussing how the agent can be evaluated. Evaluating on a stochastic policy can be problematic, because each evaluation could lead to different outcomes, which in turn could lead to reproducibility issues. Therefore, when validating and testing the mean action is used, which essentially converts the policy from stochastic to deterministic. This means the results actually reflect the learned policy, rather than lucky stochastic transitions. In a real life setting however, the stochastic policy might be preferred. Because of noise, there will never be a perfect state representation, which means that an action distribution could help the agent navigate when the uncertainty is higher.

5.2.1 State and rewards

The state representation was designed to prioritize path-following behavior rather than explicitly reaching a specific target. In early experiments, target specific features such as distance to target, absolute rotation, and absolute translation were included in the state. However, these parameters led the agent to exploit environment specific patterns instead of learning generalizable navigation strategies. During training on a fixed target point, the agent appeared to associate particular distances with specific actions. As a result, when evaluated on a different anatomy or validation setup, the learned policy did not generalize well. Instead of responding to the current environment, the agent relied on patterns memorized from the training configuration. We also observed that incorporating global features as absolute tool positions and directional vectors reduced generalization performance. These features introduced dependencies on the global reference frame, making it more difficult for the agent to adapt to variations in anatomy and environment.

Since we had no way of guaranteeing that the coordinate systems in the different cases were standardized and every anatomy was unique, we came to the conclusion that the global coordinates acted more as noise to the algorithm than helpful information. Furthermore, the only indication of the state's correctness is experimental results. It is unknown if the state used is valid for the problem, or what parts actually contribute in a meaningful way. It is entirely possible that a small part of the state enables the navigation and that the rest is ignored by the model, or actually hampers its performance.

Our discrete approximation of the bend energy has an error of less than 7% for angles up to 50 degrees, with the error increasing for larger angles. This approximation was chosen deliberately, as the reward term is negative and should impose a stronger penalty for larger bend angles. While smooth bends are natural when the wire and catheter follow the vessel curvature, excessively sharp bends are undesirable and were therefore penalized more heavily.

Similar to the state, the reward function is an experimental setup adapted to the simulation environment. The reward function was created through careful consideration of the state and environment, and through trial and error. There is no previous work done on reinforcement learning in VIST, so the implementations in the project are entirely novel. Therefore, while the reward function is adequate for navigation, it might be too simple or too complex.

In previous works, the reward function is often more simple, but so is the simulation software. It is possible that either the approach in this project is unnecessarily complex, or the more advanced simulation used requires a more nuanced reward signal. However, it can be noted that the reward function was quite brittle and small changes could lead to large changes in performance. It is unclear whether

the chosen design was unstable or if the reward function in general is sensitive to tuning. It is not surprising if the latter is true, as the reward signal is the most important factor in training. Therefore, since successful navigation was achieved, it is indicative of an adequate reward function, even though it might not be the optimal one.

5.2.2 Training, validation and testing

The training and validation set consisted of four and two cases respectively. Since there was no metric to evaluate how different the cases were, each case was assumed to be a unique anatomy. This assumption could be problematic, since it implies there could be a significant overlap between the training and validation cases. It could be argued that this is a non-issue, as the model weights are not updated during validation, which means the agent never actually sees the validation cases. Therefore, this is not data leakage in the traditional sense, but it can still affect the final testing of the model, since the trained model could be overfitted to the validation case by accident. Despite the limited number of cases available, it was for this reason deemed important to have two validation cases, to have some measure of the model's generalization. It could therefore be reasonably assumed that the model with the highest performance on the validation cases would be the most generalized.

Fair evaluation requires unseen data. Since there was some ambiguity concerning the separation between training and validation data, the test data had to be completely separate. The reasoning was that if there is data leakage between the datasets, the testing would be a true indication of the model's path-following ability. This significantly reduces the risk of the final model relying on memorization of anatomical structures present in the training and validation data.

The number of training cases might have been too few. It is unknown if more training cases would have been beneficial, or if too many were used. Using large datasets is typical practice in machine learning, but it might work differently in reinforcement learning. There are three possibilities. First, it is possible that too few cases were used, which would mean that the learned policy was brittle. Second, the number of cases was ideal and led to adequate generalization. Lastly, there might have been too many different cases, which possibly confused the policy and actually reduced performance, because the number of different states and actions destabilized the learning. Perhaps simply training on one anatomy could have resulted in a more stable policy, since the gradient descent might have had a more consistent space to optimize.

It is also interesting to know what combination of number of cases, episodes per cases and total episodes that is optimal. The chosen hyperparameters are arbitrary and a different combination could have achieved greater results. For example, the cycling of cases every 100 episodes might have damaged performance and perhaps staying longer on each case would have stabilized training. It could also be the opposite that staying for a shorter time on each anatomy might have improved results. For

example, choosing a random point and anatomy each episode

5.2.3 Metrics

The most obvious metric when discussing autonomous navigation is the ability to reach a predefined goal, which in this context is called a success rate. However, the binary nature of this metric could be a flaw, where almost successful navigation is categorized as a failure. The progress metric is important for this reason, as it reflects how close the agent gets on average, rather than a simple true or false. An agent that can reach 80 percent of the way to some target can still be very helpful in a clinical environment, as it could potentially reduce work load for surgeons significantly, even if the last distance would be navigated manually. Therefore, it is unfair to discard all navigation attempts that don't reach the target all the way.

Another problem with the success rate metric is brute forcing. The agent has a tendency to coil up for various reasons, usually due to erroneous navigation or handling of the tools. On rare occasions, this will still lead to a successful navigation. While this technically is a success, it is an unacceptable form of navigation. This behavior is however rare and can be detected through the reward per step metric because of the bend energy reward.

5.3 Limitations

Even though VIST is a state of the art simulation software, the anatomies used in training are artificial. Therefore the blood vessels may be too smooth, too symmetric and lack the complexity of real life bifurcations. This could mean that the simulation is not entirely representative of reality, which could lead to biases in the model. If the model is not trained on the imperfections that exist in a real anatomy, the state to action relationship might be entirely broken.

The target points used in training were arbitrarily selected and there is no metric on how well they transfer to real life. Since there is no medical motivation behind the targets, the clinical benefit of the model could be questioned. Also, the absence of an algorithmic safety layer could allow the agent to reach points that would be unreachable in real life without damaging the vessels. Since the blood vessels are impenetrable in the simulation, the model could learn to cheat by using excessive force.
Another important limitation are the bottlenecks in training. Since reinforcement learning requires exploration of the environment and needs to see many combinations of states, actions and rewards, which results in very slow training. Furthermore, there was a lot of overhead with GRPC calls to the simulation, which slowed down each timestep. Slow training limited how many implementations that could be tested, as each training run was a time investment. Therefore, faster training could have led to better outcomes, as more variations of the algorithm could have been tested.

5.4 Future work

In this section, potential future improvements are presented. The suggestions are sorted with more feasible implementations appearing first.

- Future work should have a stronger focus on patient safety. This should first and foremost be implemented by adding wire and catheter tip forces into the state. Tip forces exceeding a threshold should be penalized, to teach the model to navigate safely.

- More data could be beneficial for generalization. For example, an algorithm could be devised to create many datasets from one single anatomy, by creating variation in the anatomy. This could be things like altering the vessel positions, bends or by changing bifurcations.

- It is unclear which state components actually are meaningful to navigation. Future works should try to reduce the size of the state and only include relevant information. This would have to be done through trial and error or by incorporating explainable AI.

- Since the model outputs translation and rotation, this could be directly translated to the input of the electric motors of a surgical robot. Therefore, future works should consider connecting the algorithm to a robot and testing it in real life models of endovascular structures.

- Another consideration is tool selection. The model is only trained on one specific set of tools, while real life endovascular surgery relies on many different set of tools for different situations. The tools used in this project do have their limitations and a truly general algorithm should be able to choose its own tools depending on the current state.

- Since the algorithm relies on the path provided by the simulation, future works should create a separate algorithm to build a path from fluoroscopic images. This would make the algorithm more relevant from a clinical perspective.

6
Conclusion

This thesis investigated the application of deep reinforcement learning for autonomous navigation in an endovascular environment. To achieve this, a reinforcement learning agent was trained in the VIST simulation environment to follow a predetermined path using a guidewire and catheter. The results showed that the highest validation performance was achieved after training for 800 episodes. When evaluated on previously unseen anatomies, the agent achieved a success rate of 96 % in the brain and a 88 % success rate in the liver.

These findings indicate that reinforcement learning is a viable approach for autonomous endovascular navigation in a simulated setting and that the agent demonstrates generalization rather than simple memorization of anatomical structures. Future work should focus on incorporating safety-related constraints, such as limiting tip forces applied to vessel walls, increasing the diversity of training data, and improving transferability to real-world clinical environments.

Bibliography

[1] Roberto G. Aru and Sam C. Tyagi. Endovascular treatment of femoropopliteal arterial occlusive disease: Current techniques and limitations. *Seminars in Vascular Surgery*, 35(2):180–189, April 2022.

[2] UCSF Health. Endovascular surgery, Jul 2025.

[3] Stavros Spiliopoulos, Spyridon Prountzos, Stavros Grigoriadis, Athanasios Diamantopoulos, and Ioannis Paraskevopoulos. Esr essentials: arterial vascular access and closure devices-practice recommendations by the cardiovascular and interventional radiological society of europe. *European radiology*, page 10.1007/s00330–024–11053–3, Mar 2024.

[4] Decker Medicine LLC. Instructional training material. `https://www.deckerip.com/media/3061.pdf`, 2010. Educational instructional document.

[5] Daniel Brandão. Choosing the right guidewire: The key for a successful revascularization. *Art and Challenges Involved in the Treatment of Ischaemic Damage*, Oct 2022.

[6] Sohail Q Khan and Peter F Ludman. Percutaneous coronary intervention. *Medicine*, 50(7):437–444, Jun 2022.

[7] Endovascular thrombectomy (evt) — uf health, 2023. UF Health.

[8] Tianliang Yao, Chengjia Wang, Xinyi Wang, Xiang Li, Zhaolei Jiang, and Peng Qi. Enhancing percutaneous coronary intervention with heuristic path planning and deep-learning-based vascular segmentation. *Computers in Biology and Medicine*, 166:107540, October 2023.

[9] Nikos Werner, Georg Nickenig, and Jan-Malte Sinning. Complex pci procedures: challenges for the interventional cardiologist. *Clinical Research in Cardiology*, 107(S2):64–73, July 2018.

[10] Max Wagener, Yoshinobu Onuma, Ruth Sharif, Eileen Coen, William Wijns, and Faisal Sharif. Features and limitations of robotically assisted percutaneous coronary intervention (r-pci): A systematic review of r-pci. *Journal of Clinical Medicine*, 13(18):5537, September 2024.

[11] Tianliang Yao, Madaoji Ban, Bo Lu, Zhiqiang Pei, and Peng Qi. Sim4endor: a reinforcement learning centered simulation platform for task automation of endovascular robotics, April 2025.

[12] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[13] Us patent 8,083,524 — interventional simulator system. `https://patents.justia.com/patent/8083524`, 2011. Justia Patents. Accessed 2026-01-26.

[14] Us patent 11,464,572 — systems and methods for routing a vessel line such as a catheter within a vessel. https://patents.justia.com/patent/11464572, 2022. Justia Patents. Accessed 2026-01-26.

[15] Ziyang Mei, Siyuan Han, Youchang Xia, Zitong Liao, Wenbo Zhou, Yang Zhao, and Gang Liu. Evegars: End-to-end vision-based endovascular guidewire autonomous reinforcement learning in large-scale simulations from clinical dataset. In *Lecture Notes in Computer Science*, LNCS, pages 211–224. Springer, January 2026.

[16] H. Robertshaw, B. Jackson, J. Wang, et al. Reinforcement learning for safe autonomous two-device navigation of cerebral vessels in mechanical thrombectomy. *International Journal of Computer Assisted Radiology and Surgery*, 20:1077–1086, 2025.

[17] Francois Faure, Christian Duriez, Herve Delingette, Jeremie Allard, Benjamin Gilles, Stephanie Marchesseau, Hugo Talbot, Hadrien Courtecuisse, Guillaume Bousquet, Igor Peterlik, and Stephane Cotin. Sofa: A multi-model framework for interactive physical simulation. In *Studies in Mechanobiology, Tissue Engineering and Biomaterials*, pages 283–321. Springer, January 2012.

[18] Tianliang Yao, Yueqi Xu, Haoyu Wang, Xihe Qiu, Kaspar Althoefer, and Peng Qi. Multi-agent fuzzy reinforcement learning with large language models for cooperative navigation of endovascular robotics. *IEEE Transactions on Fuzzy Systems*, 2025.

[19] Marco Wiering and Martijn van Otterlo, editors. *Reinforcement Learning*, volume 12. Springer, 2012.

[20] Hao Dong, Zihan Ding, and Shanghang Zhang. *Deep Reinforcement Learning*. Springer Nature Singapore, Singapore, 2020. Translation from the English language edition.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[22] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps, 2017.

[23] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[24] ApX Machine Learning. Ddpg algorithm. https://apxml.com/courses/advanced-reinforcement-learning/chapter-3-advanced-policy-gradients-actor-critic/ddpg-algorithm, 2026. Accessed: 2026-02-26.

[25] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning, 2022.

[26] Shengbo Eben Li. *Deep Reinforcement Learning*, pages 365–402. Springer Nature Singapore, Singapore, 2023.

[27] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019.

[28] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.
[29] Diederik Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014.
[30] J. D. Reeve. Biostatistics: Data and models. `https://www.oercommons.org/courseware/lesson/101759/student/`, 2022. Licensed under CC BY 4.0.
[31] Joshua Achiam. Spinning up in deep reinforcement learning. 2018.
[32] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018.
[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
[34] S Gopal, Krishna Patro, and Kishore Kumar. *Normalization: A Preprocessing Stage*.
[35] Johannes Wallner. Note on curve and surface energies. *Computer Aided Geometric Design*, 24(8-9):494–498, Nov 2007.

A

Appendix 1

Algorithm 3 The reward function

▷ Goal reward
if $d_{target} < 0.5$ **then**
    $done \leftarrow$ true
    $r_{goal} \leftarrow 50$
else
    $done \leftarrow$ false
    $r_{goal} \leftarrow 0$
end if
▷ reward
$progress \leftarrow \text{clip}(progress_{wire} + progress_{cath}, -4, 4)$
$total\ alignment \leftarrow \text{clip}(alignment_{wire} + alignment_{cath}, -1, 1)$
$alignment \leftarrow \max(1 + 0.5 \cdot total\ alignment, 0.8)$
$r_{progress} \leftarrow 10 \cdot progress \cdot alignment$
▷ Lateral deviation penalty
$r_{lat} \leftarrow 0$
if $deviation_{wire} > radius$ **then**
    $r_{lat} \leftarrow r_{lat} - 0.7 \cdot deviation_{wire}$
end if
if $deviation_{cath} > radius$ **then**
    $r_{lat} \leftarrow r_{lat} - 0.7 \cdot deviation_{cath}$
end if
▷ Separation penalty
$r_{sep} \leftarrow -0.2 \cdot \max(0, d_{sep} - 6)$
▷ Smoothness penalty
$r_{bend} \leftarrow -0.005 \cdot bend_{wire} - 0.002 \cdot bend_{cath}$
▷ Step penalty
$r_{step} \leftarrow -0.5$
$reward \leftarrow r_{goal} + r_{progress} + r_{lat} + r_{sep} + r_{bend} + r_{step}$
$reward \leftarrow 0.1 \cdot reward$
return $(reward, done)$

Episode	Success	Progress	Steps	Reward	Reward/Step
50	30.0	69.3	768.7	-73.0	-0.095
100	2.0	51.4	987.3	-120.9	-0.122
150	2.0	50.4	998.8	-57.8	-0.058
200	0.0	23.2	1000.0	-80.8	-0.081
250	0.0	14.4	1000.0	-84.0	-0.084
300	0.0	15.1	1000.0	-101.5	-0.102
350	0.0	10.6	1000.0	-199.3	-0.199
400	4.0	44.5	962.2	-96.2	-0.100
450	0.0	35.3	1000.0	-48.1	-0.048
500	18.0	55.4	902.5	-31.2	-0.035
550	10.0	52.7	933.2	-157.2	-0.169
600	2.0	24.6	990.3	-354.3	-0.358
650	8.0	37.7	939.6	-51.3	-0.055
700	12.0	37.8	914.4	-80.0	-0.088
750	16.3	42.3	884.7	-58.8	-0.067
800	94.0	93.7	260.0	38.4	0.148
850	44.0	47.9	640.1	-60.4	-0.094
900	30.0	53.3	742.5	-130.8	-0.176
950	34.7	68.9	728.5	-14.0	-0.019
1000	40.0	69.3	667.0	-5.8	-0.009
1050	32.0	59.2	748.7	-54.5	-0.073
1100	34.0	56.1	724.7	-67.6	-0.093
1150	42.0	60.1	672.2	-35.0	-0.052
1200	30.0	54.2	750.0	-55.4	-0.074
1250	22.0	41.3	829.9	-50.5	-0.061
1300	30.0	39.8	763.0	-54.8	-0.072
1350	48.0	74.8	674.6	-14.1	-0.021
1400	32.0	59.7	732.8	-45.0	-0.062
1450	74.0	83.2	483.0	11.2	0.023
1500	68.0	79.4	472.4	9.3	0.020
1550	56.0	74.3	562.7	-16.3	-0.029
1600	52.0	66.6	586.3	-13.0	-0.022

Table A.1: Shows the results from Validation.

A.0.1 Bend energy approximation

$$E = \int \kappa(s)^2 \, ds \tag{A.1}$$

$$\mathbf{T}_i = \frac{p_{i+1} - p_i}{\|p_{i+1} - p_i\|} \tag{A.2}$$

$$\kappa_i \approx \frac{\|\mathbf{T}_{i+1} - \mathbf{T}_i\|}{\Delta s} \tag{A.3}$$

$$\cos \psi_i = \mathbf{T}_i \cdot \mathbf{T}_{i+1} \tag{A.4}$$

$$\|\mathbf{T}_{i+1} - \mathbf{T}_i\|^2 = 2(1 - \cos\psi_i) \tag{A.5}$$

$$\text{For small angles:} \qquad \cos\psi_i \approx 1 - \frac{\psi_i^2}{2} \implies \|\mathbf{T}_{i+1} - \mathbf{T}_i\| \approx \psi_i \tag{A.6}$$

$$\begin{aligned} E = \int \kappa(s)^2 ds \approx \sum_i \kappa_i^2 \Delta s \approx \sum_i \left(\frac{\|T_{i+1} - T_i\|}{\Delta s} \right)^2 \Delta s = \\ = \sum_i \frac{\|T_{i+1} - T_i\|^2}{\Delta s} \approx \sum_i \frac{\psi_i^2}{\Delta s} \end{aligned} \tag{A.7}$$

$$\text{If all segments have the same length } \Delta s: \qquad E \approx \frac{1}{\Delta s} \sum_i \psi_i^2 \propto \sum_i \psi_i^2 \tag{A.8}$$

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY