Using Deep-Learning Proximal Policy Optimization to Solve the Inverse Kinematics of Endoscopic Instruments

Andreas Schmitz and Pierre Berthet-Rayne, *Associate Member, IEEE*

***Abstract*—There is currently a trend towards small tendon-driven robotic devices targeting endoscopic applications. As these robotic systems aim to navigate within narrow tortuous pathways, their joint arrangement must be longitudinal. Additionally, mechanical constraints such as tendon routing limit the joint range which impacts the overall dexterity of the instruments and complicates the use of iterative inverse kinematics solvers. This article investigates the use of reinforcement learning and proximal policy optimization to solve the inverse kinematics problem of endoscopic instruments while considering joint limits, joint velocity, and whole body configuration. The results show that the proposed approach is able to learn the kinematic model of a 5 degrees of freedom endoscopic instrument and performs better than the Damped Least Square iterative approach in terms of computation time, position error, and orientation error.**

***Index Terms*—Surgical robotics, laparoscopy, medical robots and systems, model learning for control.**

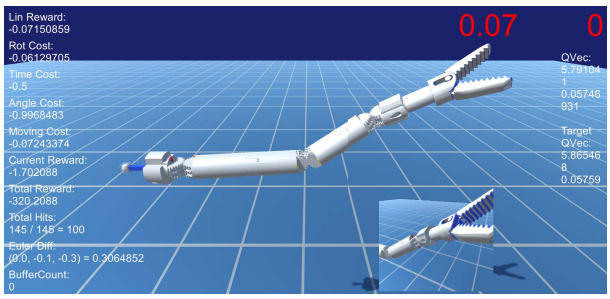

Fig. 1. Interface for learning the inverse kinematics with PPO.

I. Introduction

Minimally Invasive Surgery (MIS) is a surgical technique aiming to improve patient recovery by reducing the amount and size of incisions. Endoscopic surgery is a sub-field of MIS which aims to use natural orifices instead of incisions. Existing endoscopes allow the operator to perform diagnostic and simple therapeutic acts. Unfortunately, complex surgical procedures are limited by the poor stability of the endoscope and the limited dexterity of the instruments, so there is a need for novel systems. In endoscopy, the joint arrangement is longitudinal to allow insertion through narrow channels which limits the joint range and dexterity. Typical iterative Inverse Kinematics (IK) approaches use the Moore-Penrose pseudo-inverse or the Damped Least Square (DLS) [1], but issues such as local minima and joint limits, although addressed in the literature are still open research questions. The use of Reinforcement Learning (RL) is now popular in computing applications, but its applications toward medical robotics is still poorly explored.

Neural networks (NN) can be used to solve the IK of robotic arms. Duka used inverted data pairs generated with forward kinematics (FK) to train a feed-forward NN [2]. The joint configuration was added as an input to the network to help with multiple IK solutions to one target pose [3]. Hasan used a method of NN Inversion to learn the IK problem [4]. The network is trained for the forward dynamics of the robot. The network was inverted to represent the inverse dynamics of the system. Satheeshbabu uses a Deep Deterministic Policy Gradient RL to train the IK of a soft continuum arm [5]. Rolf and Steil, presented an approach to learn the IK of a pneumatic continuum robot arm [6]. They tackled the high dimensions, unknown actuation ranges, and non-stationary system behavior with a method called online goal babbling. They evaluated the method in real-world experiments on a non-stationary robot. In the space of MIS, the use of RL for learning the kinematics or dynamics is less prominent. Chen and Lau investigated learning the IK of a tendon-driven continuum robot [7]. They developed various machine learning approaches, and found that all methods failed to deliver a stable performance. Chen *et al.* used a RL algorithm, called Policy Learning by Weighting Exploration with the Returns, which is a policy gradient RL algorithms [8]. They derived the IK analytically and then used the RL to optimize the parameters. Richter *et al.* used a surgical simulator to train the IK of the da Vinci robot using Deep Deterministic Policy Gradient [9]. Ansari *et al.* demonstrated the use of RL to solve the IK of a planar 6 DoF virtual robot using multi-agent RL for positioning only [10].

In [11] we compared several IK algorithms and showed the need to develop novel IK approaches dedicated to endoscopy that can intrinsically handle joint limits and joint configuration factors in real-time.

In this context, this article proposes to use Proximal Policy Optimization [12] combined with an Artifical NN to control an endoscopic instrument [13] for the i^2Snake robot [14] while considering joint limits.

II. Method

The RL algorithm tries to reach a target, starting from a random workspace position and by changing its joint values. The algorithm is only allowed to use joint values within limits, which is enforced at each step. During the training, the robot can only learn feasible configurations which means the joint limits are learned intrinsically. This allows the algorithm to find a solution to the IK problem within joint limits, which also reduces the search space and speeds up the process. Once the training completed, the model is exported and integrated into the robot control.

The FK determines the tip position of a joint vector q. The instrument used in this article was initially proposed in [15]. The FK is calculated with the modified Denavit-Hartenberg (DH) convention. The instrument's DH table is shown in Table I where "rolling" represents a rolling-joint with radius a_r. θ_{off} is the joint angle offset. The end-effector length being 7.09mm along the final z axis. The RL

Manuscript received July 13, 2020; revised October 10, 2020 and October 26, 2020; accepted November 3, 2020. Date of publication November 17, 2020; date of current version February 22, 2021. This article was recommended for publication by Associate Editor F. Rodriguez y Baena and Editor P. Dario upon evaluation of the reviewers' comments. *(Corresponding author: Andreas Schmitz.)*

The authors are with the Hamlyn Centre for Robotic Surgery, Imperial College London, London SW7 2AZ, U.K. (e-mail: a.schmitz16@imperial.ac.uk).

Digital Object Identifier 10.1109/TMRB.2020.3038536

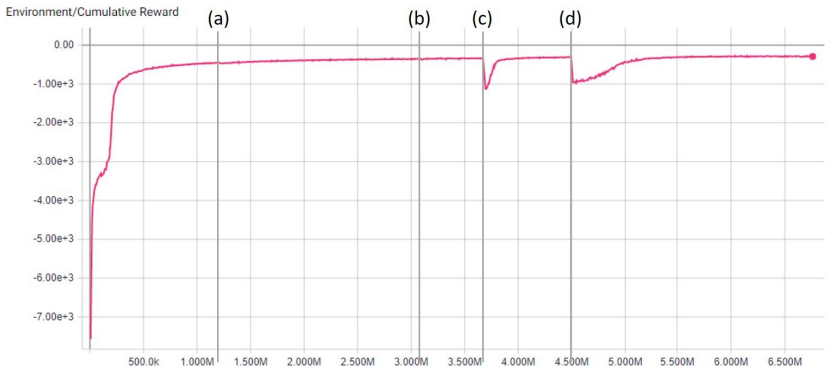

Fig. 2. The learning curve of the network. At time points (a)–(d) the curriculum switched to a new set of parameters. The learning process took approximately 20 hours.

TABLE I
INSTRUMENT DH PARAMETERS

a	α	d	θ	type	a_r	θ_{off}	joint limits
0	0	0	pi/2	prismatic			0 - 20
0	pi/2	0	pi/2	rolling	2.85		-0.88 - 0.88
12.27	pi/2	0	0	rolling	2.85		-0.88 - 0.88
7	pi/2	0	0	rolling	2.85	pi/2	-0.88 - 0.88
0	pi/2	3.04	pi/2	rolling			-0.88 - 0.88

algorithm uses the FK to create random targets while ensuring they are reachable.

A. Reinforcement Learning

The principle of RL is the optimization of a reward function of an actor within an environment. An environment e is a description of the system as a tuple of $e = (\vec{s_1}, S, \vec{o}(\vec{s}), A, r(\vec{s}), d(\vec{s}), t : (\vec{s}, \vec{a}) \rightarrow \vec{s'})$ with S being the set of possible inner states, defined by a set of state variables, a start state $\vec{s_1} \in S$ as a valid instance of the state space, and $o(\vec{s})$ a set of observations, which the actor can see. Further it has a set of actions A, which can be discrete or continuous and describe how the system reacts to the actor's choices. It results in a reward $r(\vec{s})$ for the new state, the observation $o(\vec{s})$ of the new state, a terminated state $d(\vec{s})$ which defines if the environment is in a final state, and a transition function t which leads to a new state $\vec{s'} \in S$. The basic learning principle is Q-Learning [16]. The central learning principle is:

$$Q^{new}(\vec{s}_t, \vec{a}_t) = Q^{old}(\vec{s}_t, \vec{a}_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_a \left(Q(\vec{s}_{t+1}, \vec{a}) \right) - Q^{old}(\vec{s}_t, \vec{a}) \right) \quad (1)$$

with r_t the reward at time step t, discount factor γ, learning rate α, and the expected return $\max_a(Q)$. This Q-Learning function defines the expected reward for each state $\vec{s}_t$ choosing the action $\vec{a}_t$. To train this function at each learning step, the change in the reward plus the expected future rewards is added to the old expected reward. This ensures the function converges towards the desired reward function.

An advanced version of this RL is Proximal Policy Optimization (PPO) [12] which we use in this article. There are multiple implementations and libraries for RL, such as tf-agents and ml-agents. In this article, ml-agents (Unity Technologies, USA) is used which works closely together with the Unity graphics engine which we used for simulation and validation. It is based on Tensorflow (Google, USA) and incorporates PPO. The environment is implemented as a derivative of an "Agent" class which has a function for all parts of the set e.

B. Development of the Learning Method

The proper learning of a robotic model is a challenging task for the following reasons:

- *Parameters:* RL has a high number of adjustable parameters. Each parameter has a functional range. If one parameter is out of this range, the training may not succeed. Further, dependencies between these parameters exist, e.g., learning rate and number of learning steps.
- *Training Time:* The learning time increases exponentially with factors such as model complexity or target precision.
- *Cost Function:* The target area in which the actor receives a reward is important. A large target area can be used during preliminary experiments to accelerate the learning time, but the training time will grow exponentially as the target area becomes smaller.
- *Task Definition:* Often the task definition is not possible to solve. The best way is to create an user controlled instance of the task and try to solve it manually. Problems in the task definition will be recognized easily this way.

To overcome these challenges, it is important to start with a simple task definition and increase the complexity progressively. The target area should be relatively large in the beginning.

C. Environment Definition

This section introduces the definition of the key components of the RL environment.

1) State: The arm design consists of a prismatic joint, 3 geared rolling-joints, a distal-roll, for a total of 5 DOF + gripper, as shown in Figure 1. All gear joints have the same joint limit of 50°. The q-vector is the joint angle vector. It is used to calculate the tip position and to determine the error in position and orientation.

2) Target: The target T defines the position and orientation where the tip of the arm needs to go. The target is a 4×4 homogeneous transformation matrix:

$$T = \begin{pmatrix} R & P \\ \vec{0} & 1 \end{pmatrix} \quad (2)$$

where R is a 3×3 rotation matrix, P is a 3×1 position vector, and $\vec{0}$ is the vector $(0, 0, 0)$. For training purposes, T is calculated from a random $\vec{q}$-vector within joint limits. This ensures that the target is always reachable. The target has an area around it, where it is counted as being reached. This area is defined by a maximal positional error and a maximal rotational error.

3) Reset: The environment's reset function defines the new target and initializes the robot arm with a random $\vec{q}$-vector within joint limits. The random starting position together with the random target

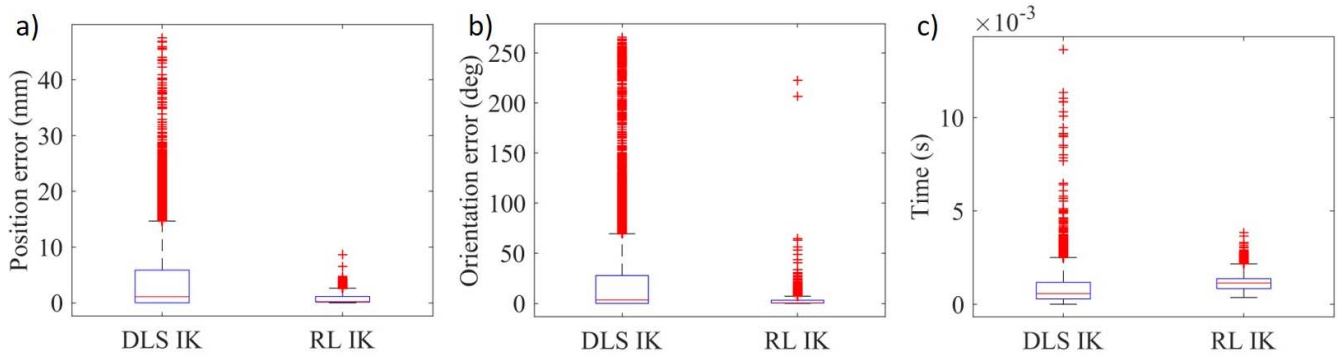

Fig. 3. Box plot statistical comparison between DLS and RL using the position error, orientation error and time as metrics.

position ensures that all {start, target} combinations are learned. This helps the algorithm to learn the robot's workspace.

4) Action: The action space in this environment is a continuous action vector $\vec{a} \in A = [-1, 1]^N$ with the length of the number of joints N. Most RL algorithms and especially PPO can only handle continuous action vectors from -1 to 1. To transform an action vector $\vec{a}$ from A-space into a $\Delta\vec{q}$ in the $\vec{q}$-vector space $Q = \mathbb{R}^N$ it is multiplied by a step size: $\Delta q_j = a_j \cdot \epsilon_j$ for each joint $j \in [0, N]$ with $\vec{a} \in A$ and step size for each joint $\epsilon \in \mathbb{R}^N$. The new $\vec{q}$ of the environment state is calculated by modifying the current $\vec{q}$ vector: $\vec{q}_{t+1} = \vec{q}_t + \Delta\vec{q}$. The step size can be seen as the precision of a step.

5) Reward: The reward function rewards the agent following an action. It is possible to have all the reward only in the last step while all other step rewards are zero. Since this is a very flat reward function it is difficult for the algorithm to find a solution and learn the optimal policy. Hence, the reward function used is the distance between the tip and the target. It is defined by multiple factors, which consist of:

- The position cost, defined as the norm of the distance of the target and the current tip position.
- The rotational cost is defined as the orientation error between start and goal as a Euler vector. The cost is the euclidean norm of the Euler vector.
- Time cost which is a constant ($= 0.5$).
- Angle cost, which penalizes sharp bends as a quadratic function of all angles combined. This ensures smooth bends instead of moving one joint to its to full extent. This is especially important with tendon-driven robots.

6) Observations: The observations are the set of variables the agent can see. They are the input parameter of the NN, while the actions are the output. In this case, the observations are a vector of 13 float values consisting of:

$\vec{q}$-vector (5), Position error (3), Position error norm (1), Orientation error (3), Target reached boolean (position error norm $<$ than **goal_size**, orientation error norm $<$ than **goal_rot**) (1).

7) Curriculum: With RL, it sometimes happens that a problem is too simple to learn and the algorithm finds a solution quickly but suboptimal. The use of a curriculum allows to overcome this challenge by changing the model parameters and thus increasing the required precision after a predefined amount of training episodes.

8) Model Parameters: The parameters used for the RL are described below. The parameters' values were assigned experimentally.

- *Goal_Size:* This parameter is part of the curriculum and determines the size of the goal area in the Cartesian space in millimetres. During the learning, a range between 10mm and 0.1mm was used.
- *Goal_Rot:* This parameter determines the size of the goal area in rotational degrees. During learning with curriculum a range between 10.0° and 0.1° was used.
- *Step_Size:* This controls the joint angle increment between steps and hence the velocity. This was set to 0.1°.
- *Trans_Step_Size:* This controls the prismatic joint increment. This value was set to 2.0 mm.
- *Max_Steps_Agent:* The maximal number of steps performed during learning for one episodes. This restricts the time the arm has to reach the goal. This was set to 500.
- *Moving_Cost_Factor:* This specifies the cost of instrument movement. It is defined as the movement distance multiplied by this factor and limits unnecessary movement, e.g., overshooting. It was set to 10.
- *Time_Cost:* This is a constant added at each time step and penalizes long exploration time. This was set to 0.5.
- *On_Goal_Num:* This parameter specifies the number of steps the agent needs to stay on the goal before it counts as being reached to prevent oscillations. It was set to 20.
- *Angle_Cost:* Angle cost determines the cost for the bending angle at each joint. This is used to get smooth joint configurations. It was set to 1.
- *Rot_Cost_Factor:* Rotation cost factor weights the orientation error. This parameter was set to 0.2.
- *Reward:* For the reward all costs are summed up. The rotation cost is scaled by the **rot_cost_factor** in the sum.

III. Evaluation

A. Integration

The network was trained offline until a good precision and success rate was reached which took 20 hours and 6.7M training episodes, see Fig. 2. The learned Tensorflow model was integrated into the control system to calculate the IK for the simulated robot. After training, the model is exported to a Tensorflow frozen network and the Tensorflow library loads the exported model. The component receives the target position from the master device and scales it to the right reference frame, calculates the joint values, and sends it to the robot control component. Although the solution generated by the network is expected to remain inside the joint limits, it is verified that no joint limits are exceeded and that the target is reached. The latter check ensures that no untrained joint values are used. If the target position is in an area around the current tip position, only the last $\vec{q}$-vector is used, otherwise an intermediate step of the path is transmitted.

B. Results and Discussion

To validate the RL implementation 10,000 random target points were generated. The targets are either reachable or non-reachable.

The targets are created from feasible random joint vectors which are then scaled up by 20%. During the evaluation, 3 metrics were saved: position error, orientation error, and computation time. The RL algorithm was compared to a DLS iterative solver [11] implemented with the EndoRob library [17]:

$$\dot{q} = J^T \left(JJ^T + \lambda^2 \mathbb{I} \right)^{-1} \vec{e} \tag{3}$$

with J the Jacobian matrix of the robot, $\vec{e}$ the tracking error, and λ a non-zero damping constant. The results are shown in Fig. 3 which shows a statistical box plot of the position and orientation error, and computation time. The red line shows the median. The blue box displays the interval of the first quantile and the third quantile, holding 50% of the data. The span of the grey interval shows where most of the data is, the red crosses being outliers.

The main advantage of DLS is that no learning is involved and hence this approach can easily be applied to different kinematics, while RL needs to be retrained for any kinematic change of the robot. The main limitation of DLS is its inability to enforce joint limits intrinsically which resulted in increased position and orientation errors as the joint values were bounded to the joint range. With RL, joint limits reduce the search space which makes the training of the NN more efficient. Regarding the computation time, RL is more consistent in computation time than DLS which varies significantly. This makes RL better suited for precise real-time control scheduling. During robot control with RL, the network is repeatedly called until the target is reached or after a predefined number of steps. If the number of steps is exceeded the search for an IK solution counts as failed and the robot will not move. This never appeared during the evaluation and was only implemented as a safety mechanism.

Regarding the step size (0.1°), it was found that this parameter has a significant impact on the learning quality and time. If the step size is too large, the algorithm cannot find a solution, because the arm is not moving precisely enough and results in oscillations. If the step size is too small, the algorithm cannot find a solution because the arm is too slow to get to the target or the algorithm would need too long to find a solution. It has been observed that the larger the target area is, the larger the step factor can be. Further, the environment needs different step size factors for different joint types, since the $\vec{q}$-vector has different scales for prismatic and rotational joints.

IV. Conclusion

This article proposed to use RL and PPO to solve the IK of a 5 DOF endoscopic instrument while considering joint limits. The results show that RL is faster, more consistent than DLS and can handle joint limits intrinsically. Additionally, more constraints can be implemented easily with RL such as limited joint velocity or smooth joint configuration. It is expected that further development in RL and longer training times will further increase the overall performance and will improve the real-time teleoperation of endoscopic robots.

References

[1] S. R. Buss. (2009). *Introduction to Inverse Kinematics With Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods*. [Online]. Available: https://www.academia.edu/download/34612234/iksurvey.pdf

[2] A.-V. Duka, "Neural network based inverse kinematics solution for trajectory tracking of a robotic arm," *Procedia Technol.*, vol. 12, pp. 20–27, Dec. 2014.

[3] Y. Kuroe, Y. Nakai, and T. Mori, "A new neural network learning of inverse kinematics of robot manipulator," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 5, 1994, pp. 2819–2824.

[4] A. A. M. Isa, H. M. A. A. Al-Assadi, and A. T. G. Hasan, "Positioning control of an under-actuated robot manipulator using artificial neural network inversion technique," in *New Developments in Artificial Neural Networks Research*, 2011, pp. 207–220.

[5] S. Satheeshbabu, N. K. Uppalapati, T. Fu, and G. Krishnan, "Continuous control of a soft continuum arm using deep reinforcement learning," in *Proc. 3rd IEEE Int. Conf. Soft Robot. (RoboSoft)*, 2020, pp. 497–503.

[6] M. Rolf and J. J. Steil, "Efficient exploratory learning of inverse kinematics on a bionic elephant trunk," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 6, pp. 1147–1160, Jun. 2014.

[7] J. Chen and H. Y. K. Lau, "Learning the inverse kinematics of tendon-driven soft manipulators with K-nearest neighbors regression and Gaussian mixture regression," in *Proc. Int. Conf. Control Autom. Robot.*, 2016, pp. 103–107.

[8] J. Chen, W. Xu, H. Ren, and H. Y. K. Lau. *Inverse Kinematics Refinement Based on Reinforcement Learning for a Wire-Driven Flexible Surgical Manipulator*. [Online]. Available: https://www.academia.edu/36110036/

[9] F. Richter, R. K. Orosco, and M. C. Yip, "Open-sourced reinforcement learning environments for surgical robotics," 2019. [Online]. Available: arXiv:1903.02090.

[10] Y. Ansari, E. Falotico, Y. Mollard, B. Busch, M. Cianchetti, and C. Laschi, "A multiagent reinforcement learning approach for inverse kinematics of high dimensional manipulators with precision positioning," in *Proc. IEEE Int. Conf. Biomed. Robot. Biomech.*, 2016, pp. 457–463.

[11] P. Berthet-Rayne, K. Leibrandt, G. Gras, P. Fraisse, A. Crosnier, and G.-Z. Yang, "Inverse kinematics control methods for redundant snakelike robot teleoperation during minimally invasive surgery," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 2501–2508, Jul. 2018.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: arXiv:1707.06347.

[13] A. Schmitz, S. Treratanakulchai, P. Berthet-Rayne, and G.-Z. Yang, "A rolling-tip flexible instrument for minimally invasive surgery," in *Proc. Int. Conf. Robot. Autom.*, May 2019, pp. 379–385.

[14] P. Berthet-Rayne *et al.*, "The i^{2}snake robotic platform for endoscopic surgery," *Ann. Biomed. Eng.*, vol. 46, no. 10, pp. 1663–1675, 2018.

[15] A. Schmitz, P. Berthet-Rayne, and G.-Z. Yang, "Endoscopic bimanual robotic instrument design using a genetic algorithm," in *Proc. IEEE/RSJ Int. Conf. Intell. Rob. Syst.*, Macau, China, 2019, pp. 2975–2982.

[16] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Comput. Sci. Dept., King's College, Univ. Cambridge, Cambridge, U.K., 1989.

[17] P. Berthet-Rayne. (2018). *EndoRob, a Multi-Thread Cross-Platform Library for Endoscopic Robot Control*. [Online]. Available: http://takskal.free.fr/EndoRob/