Towards Automated Suturing of Soft Tissue: Automating Suturing Hand-off Task for da Vinci Research Kit Arm using Reinforcement Learning

by

Vignesh Manoj Varier

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Robotics Engineering

by

May 2020

APPROVED:

_____

Dr. Gregory S. Fischer, Advisor

_____

Dr. Loris Fichera, Thesis Committee Member

_____

Dr. Jing Xiao, Thesis Committee Member

_____

Dr. Adnan Munawar, Thesis Committee Member

Abstract

Successful applications of Reinforcement Learning (RL) in the robotics field has proliferated after DeepMind and OpenAI showed the ability of RL techniques to develop intelligent robotic systems that could learn to perform complex tasks. Ever since the use of robots for surgical procedures, researchers have been trying to bring some sort of autonomy into the operating room. Surgical robotic systems such as da Vinci currently provide the surgeons with direct control. To relieve the stress and the burden on the surgeon using the da Vinci robot, semi-automating or automating surgical tasks such as suturing can be beneficial. This work presents a RL-based approach to automate the needle hand-off task. It puts forward two approaches based on the type of environment, a discrete and continuous space approach. For capturing a unique suturing style, user data was collected using the da Vinci Research Kit to generate a sparse reward function. It was used to derive an optimal policy using Q-learning for a discretized environment. Further, a RL framework for da Vinci Research Kit was developed using a real-time dynamics simulator - Asynchronous Multi-Body Framework (AMBF). A model was trained and evaluated to reach the desired goal using model-free RL techniques while considering the dynamics of the robot to help mitigate the difficulty in transferring trained model to real-world robots. Therefore, the developed RL framework would enable the RL community to train surgical robots using state of the art RL techniques and transfer it to real-world robots with minimal effort. Based on the results obtained, the viability of applying RL techniques to develop a supervised level of autonomy for performing surgical tasks is discussed. To summarize, this work mainly focuses on using RL to automate the suture hand-off task in order to move a step towards solving the greater problem of automating suturing.

Acknowledgements

I cannot begin to express my thanks to everyone who has helped and supported me in becoming the person I am today. I would like to extend my deepest gratitude to Dr. G.S. Fischer, the chairman of my committee. I have only been able to pursue this project because of the faith and profound belief he has shown in me and I couldn't have asked for a more supportive advisor for my thesis. I discovered my love and passion for surgical robotics through a guest lecture provided by Dr. L. Fichera, who has been a great teacher and advisor throughout my masters tenure. I would especially like to thank my mentor Dr. A. Munawar, who has always helped me push my limits and provided the inspiration required for me to pursue this project. I would like to express my deepest appreciation to my committee member Dr. J. Xiao who has provided me with an extensive professional guidance and taught me a great deal about scientific research.

I am deeply indebted to all my family members, for their unwavering love and support. Most importantly, my inspiration, my role models, my motivation and without whom I would have never got here, my loving and supportive father and my late mother. They have always encouraged me in all of my pursuits and inspired me to follow my dreams.

Special thanks to all the AIM lab team members particularly Dhruv, Farid, Nathaniel, Shreyas, Nuttaworn, Vishnu, Katie for their invaluable insight, contribution and assistance throughout my time working at the lab. My roommates Suraj and Nishan for listening, encouraging and being tolerating at home. All my WPI robotics peers who have given me valuable advice and cleared all my doubts with patience, especially Tyagaraja, Chinmay, Nathalie, Akshay, Vineeth, Amey, Achyutan and Aditya. Everyone at WPI who has helped make my time at WPI memorable. To all my friends scattered around the world, thank you for your messages, well-wishes, video calls and being there whenever I needed a friend.

My sincere thanks also goes to all my Medtronic colleagues, especially Jiqi Cheng, Elizabeth Gage and Ankur Agrawal for giving me an opportunity to work at Medtronic Surgical

Robotics division.

This work is supported by the National Science Foundation (NSF) through National Robotics Initiative (NRI) grant: IIS-1637759 and NSF AccelNet grant-1927275. The research work involved using computational resources supported by the Academic Research Computing group at WPI. I would also like to thank Dhruv Mishra for his help in designing some of the graphics shown in the thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the advent of technology, many enhancements have occurred in the healthcare industry. Innovations in surgical field has helped save many human lives, such as the introduction of laparoscopic tools which changed the kind of instruments being used by the surgeons to operate on patients. With about 1.2 million successful surgeries performed world-wide in 2019 using the da Vinci Surgical Systems [32], Robot-Assisted Surgeries (RAS) have become increasingly prevalent in clinical environments [11, 21]. RAS have been successful in performing a wide range of complicated surgical procedures such as coronary artery bypass, skull-based surgeries, cardiac surgeries, cholecystectomies, and organ transplants [8, 42, 75]. In recent years there has been a widespread use of MIS due to their benefits to the patient recovery. MIS is basically a type of surgery where minimal amount of surgical incisions are made on the operating body. It helps reduce the trauma to the body and results in faster recovery time. Busch *et al.* [9] quantitatively compared robotic surgery using the da Vinci with conventional laparoscopy and concluded superiority of surgeries performed using the da Vinci for Minimally Invasive Surgeries (MIS). One caveat was a higher learning curve as compared to traditional open cavity surgery [62, 79]. The difficulty associated with restricted view of the operating surface and the amount of dexterity required to operate, paved the way for using robots to perform surgeries. Due

to the high accuracy, precision and repeatability provided by a robot, surgical robotics has seen immense growth in the medical field. Over the past few decade, the da Vinci Surgical System® created by Intuitive Surgical™, Sunnyvale, CA , has helped pave way for surgical robotics. Recently, many companies have been working on bringing out their surgical robotics platform such as Verily Life Sciences™ and Johnson and Johnson™ are collaborating on the development of a surgical robot through Verb Surgical™, Medtronic™ acquired Mazor Robotics, Covidien and recently invested a lot on digital surgery, Auris Surgical Robotics™, Cambridge Medical Robotics™ and TransEnterix Surgical™ are the other major companies. Most of the companies would be launching there product soon and there are other companies which are working on partial autonomous surgical robots such as Activ Surgical [44].

A typical surgical procedure requires the team supporting the surgeon to perform numerous tasks. Some of these tasks include; prepare patient for the operation, perform the pre-op assessment, provide anesthetic induction, ensure all instruments and gauze are accounted for and check if the post-operative care plan requires changes [43]. One of the main goals of surgical robotics is to reduce the burden on the surgeon while providing them with added functionality. This could be accomplished through ergonomic robot interface and intuitive robot control. Over the years there has been a lot of development in the robotic and computer guided surgical systems. The use of surgical robots help reduce the tremors from surgeons hands, provide more dexterity and provide real time visual feedback using endoscopes [63]. It enables the surgeons to move more precisely within the body and helps them access difficult to access regions [54]. Since surgical robots use MIS techniques, the patients have to stay less time at hospital and there would be reduction in possible complications [71].

The da Vinci robot is one of the most well-known surgical robots on the market today. It is used for a variety of different surgeries ranging from general surgery to cardiac operations,

Figure 1.1: Operator using the dVRK system to perform hand-off suturing task

each of which would require a large incision in the body without a surgical robot such as the da Vinci robot. With the smaller incision, the surgeries can take quite a long time to complete and can result in the surgeon feeling tired due to the long hours of operation [51]. To relieve the stress and the burden on the surgeon using the da Vinci robot, semi automating or automating surgical tasks such as suturing can be beneficial [36, 48].

To provide an overview of the thesis, this work proposes a proof of concept implementation of Reinforcement Learning (RL) to assist in teleoperated suturing hand-off task. An introduction to surgical robotics along with the surgical task being considered is provided. The problem of suturing is first broken down into sub-tasks and of these sub-tasks, the task of needle hand-off is studied. Through an extensive literature review, few of the reasons found that point towards automating surgical tasks using learning techniques are lack of trained and experienced surgical assistants, need for reduction in clutching frequency for surgeons, eliminating heuristic tuning, ability to process high dimensional data, reduce cognitive load on surgeons, reduce procedural time, converting an instrument into an

intelligent system, compensate for the error in perception system and utilizing simulations for collection of large amounts of data [16, 20, 36, 38, 48, 73, 88]. More supporting information and advantages have been provided in the later sections of this chapter along with a summary of prior work in this field. Initially, the problem was simplified by considering it as a discrete space problem and later, a continuous space approach is explored. For the discrete space problem, user trajectories were collected and an algorithm was designed to generate sparse rewards to learn a unique suturing style. Q-learning was then implemented with this generated reward function to derive an optimal policy. Based on the state of the tool-tip, a trajectory was computed to complete the needle hand-off task. The user defined trajectories were collected from the Master Tool Manipulator (MTM) of the dVRK and the Patient Side Manipulator (PSM) is commanded to reciprocate the learned trajectory using Q-Learning. To verify the implementation of the discrete algorithm, three optimal policies were derived from a single user trajectory (reference trajectory). Each policy was used to compute a trajectory which were quantitatively compared to the reference trajectory. To ascertain the robustness of the algorithm, the initial states were distributed away from the reference trajectory's initial state and trajectories were calculated from a single policy. An illustration showing consecutive passes of the hand-off task was also presented. For the continuous space problem, a RL framework was developed for dVRK PSM using a real-time dynamic simulator. The two algorithms implemented are Deep Deterministic Policy Gradient (DDPG) and Hindsight Experience Replay (HER) with DDPG. A module was also developed to create RL compatible expert demonstrations data using the data collected from dVRK. Finally, a conclusion from the results was drawn and presented, along with the advantages and drawbacks of using these techniques.

Through this work, one would successfully be able train a surgical robot to perform automated hand-off task which can help move a step forward into automating and standardizing suturing to ease the burden on surgeons. Automating suturing will reduce the surgeons

time in the operating room, causes less stress on them during the surgery, and causes the surgeon to be less tired [77]. With the surgeon spending less time in the operating room, the cost of the operation will decrease, and the surgeon will have more time to attend to other patients [48]. Also, due to long hours of surgery, suturing could be performed differently between different surgeons and the technique and quality of suturing can vary between each surgeon [83]. Therefore, one of the possible results that could be achieved through further development of this work and eventual automation of suturing would be to help set a standard for suturing technique [38].

1.1 Surgical Robotics

Over the past few decades there has been a lot of improvement in the medical robotics field. One of the best applications found for robots has been in the medical surgery field due to the improved accuracy and repeatability provided by a robot. The use of robotics in medical field could be traced back to 1985, when PUMA 560 was employed for medical surgery [55]. The real breakthrough was, when a robot was developed that specifically performed prostate surgeries. During this time another system was being developed named ROBODOC by Integrated Surgical Supplies at California, USA. ROBODOC was designed to assist in Hip Replacement surgeries and got the approval from Food and Drug Administration (FDA) to perform limited clinical procedures [54]. The use of Tele-Operated surgery started through a collaboration between Ames Research Center at National Air and Space Administration (NASA) and Stanford Research Institute [72]. The US military saw this as an opportunity to utilize robots for surgeries in battle grounds to heal the wounded soldiers. The researchers developed Tele-Operated tool for surgical procedures and was called Mobile Advanced Surgical Hospital. It could be controlled by expert surgeons from a remote location. This system performed successful animal

trials, but never did any human trials [71]. Meanwhile, another surgical robot being developed was ZEUS. The ZEUS system had two main robotic arms for laparoscopic and thoracoscopic surgery and a third arm for endoscopy. The endoscopy arm was voice controlled and called Automated Endoscopic System for Optimal Positioning Robotics System and the system got a FDA approval in 2001. During the same time when ZEUS system was being developed, Integrated Surgical licensed the research project by SRI and completely modified the system along with lot of redesigns. It ended up becoming the most advanced Tele Operated Surgical Robot at that time [72]. It was called the da Vinci Surgical System. The company later renamed itself from Integrated Surgical to Intuitive Surgical.

The dVRK [39] system is an open-source mechatronics system based on the first generation da Vinci surgical robot developed from Intuitive Surgical Inc. The dVRK consists of manipulator arms from retired clinical da Vinci surgical systems donated by Intuitive Surgical to universities and research groups to expand the areas of research in the field. It contains a pair of MTM and PSM arms. The research kit integrated with a core set of libraries called "Computer Integrated Surgical Systems and Technology" (CISST) provides basic libraries for math, multi-threading and data logging. These libraries were utilized by the Surgical Assistant Workstation (SAW) system architecture to provide an application for general medical robots using teleoperation. An additional layer of interface is present using Robot Operating System (ROS) for communication between a Linux computer and the hardware. The CISST libraries and Surgical Assistant Workstation (SAW) software allow for the control of the two PSMs, an Endoscopic Camera Manipulator (ECM) through the two MTMs and foot pedal tray [11]. The MTM consists of 7 active joints and the PSM has six degrees of freedom with a gripper attached at the end. The foot pedal tray consists of camera and position clutches for moving the camera and to re-position the MTM's workspace respectively.

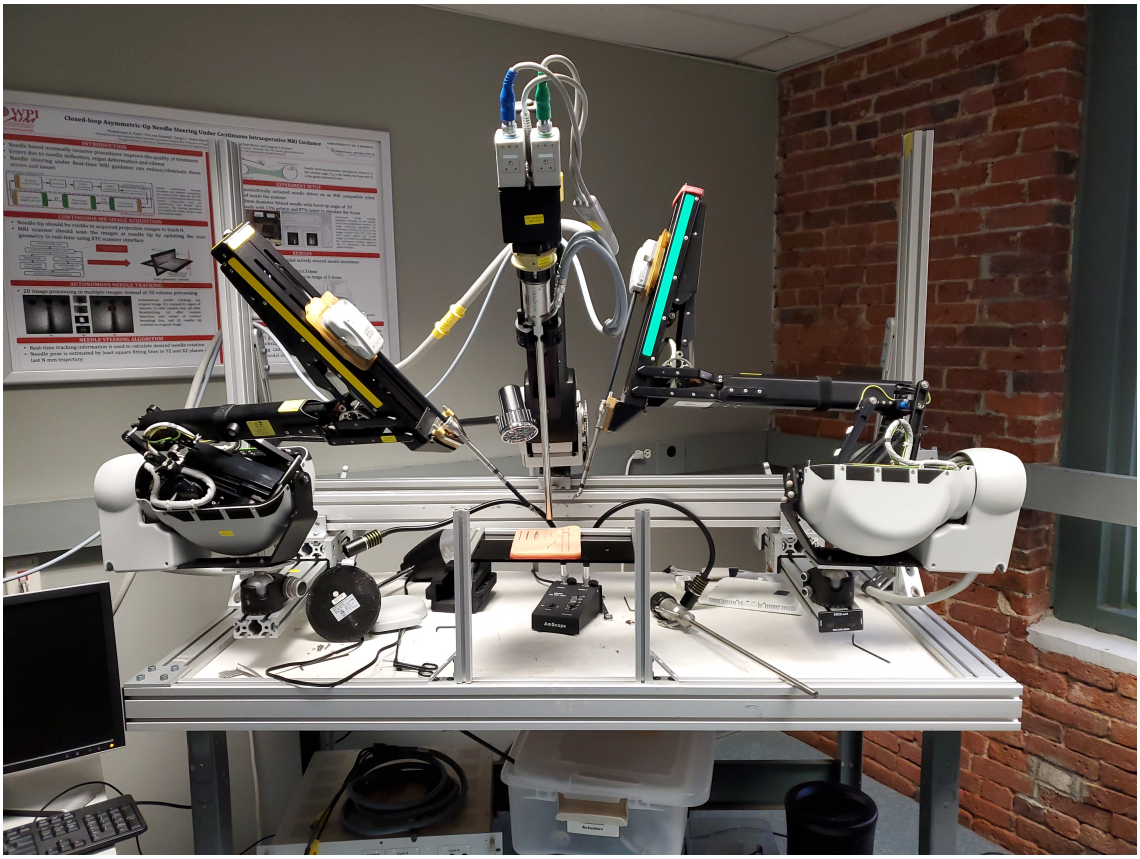

Figure 1.2: da Vinci Research Kit system with a pair of Patient Side Manipulators (PSM) and an Endoscopic Camera Manipulator (ECM)

1.2 Suturing

Suturing is the process where body tissue is held together after a surgery with a needle and thread. An ideal suture at the end of the surgery, should be simple and fast for the surgeon to perform. A suture procedure requires precise approximation of the distance from the wound edge that allows it to heal properly [53]. It primarily consists of two main tasks, stitching and knot tying, with compounded sub-tasks such as needle aligning and tissue handling. In order to improve the performance of robotic suturing while still keeping the surgeon in charge, the process of collaborative suturing can be distributed into sets of automated and manual tasks. By automating certain sub-tasks of suturing, promising results can be achieved through clutch-less suturing, a step which can allow

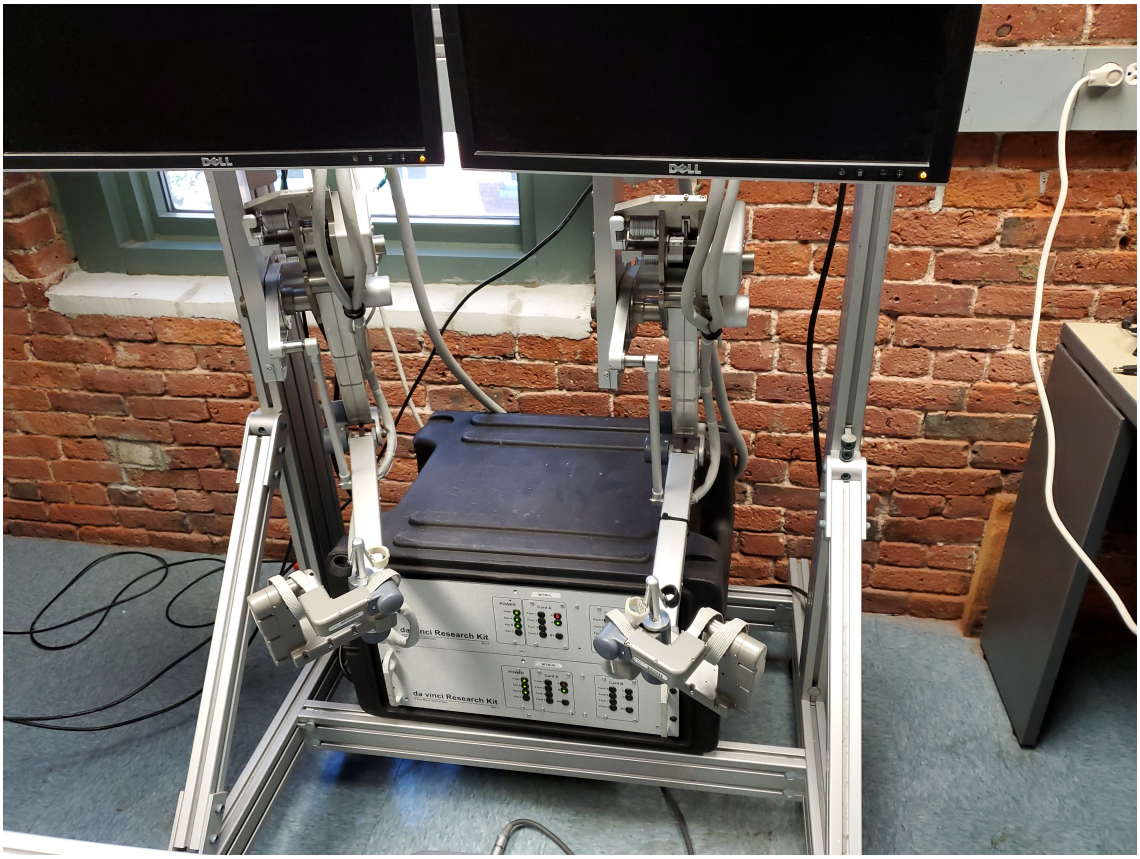

Figure 1.3: da Vinci Research Kit system Master Tool Manipulators (MTM)

faster movements and reduce fatigue on the surgeon during teleoperated surgeries [61].

Figure 1.5 describes the repetitive flow of robotic suturing and a breakdown of tasks considered in this work (Figure 1.6 shows PSM images depicting the breakdown of suture tasks). The tasks mentioned generalize gestures obtained from the JIGSAWS public data set [25] - a database of action recognition and skill assessment for automated surgery. Since the targeted application is that of collaborative suturing, a consistent task definition convention can be assumed for the remainder of the thesis. Tasks with M_i denote teleoperated tasks performed by the operator. Tasks with A_i denote automated tasks performed by the methods described in this work.

M_1 Inserting the needle through the tissue with PSM arm (PSM_M).

Figure 1.4: da Vinci Research Kit system User Foot Pedal

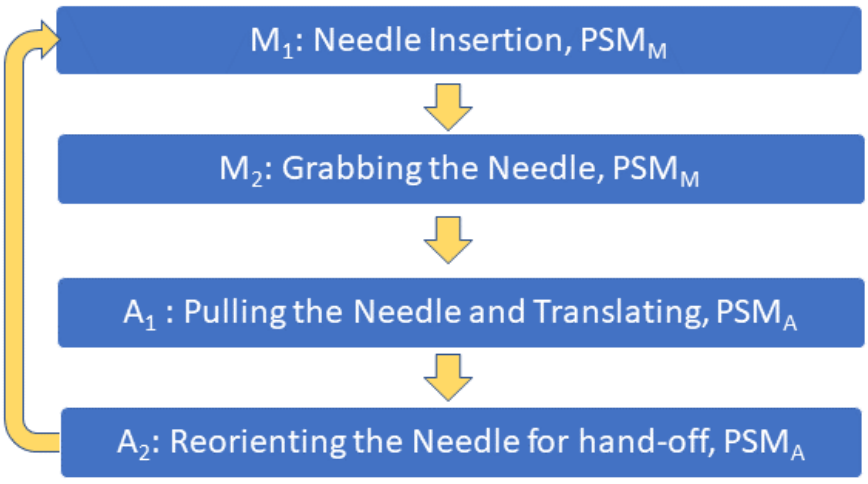

Figure 1.5: Breakdown of suturing tasks and convention assumed

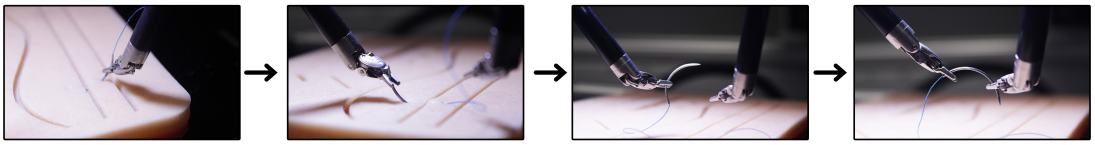

Figure 1.6: Visual representation of the tasks being performed in this paper: M_1 - needle insertion performed by PSM_M; M_2 - grasping the Needle manually performed by PSM_A, A_1 - pulling the needle and translating it, performed by PSM_A, and A_2 - reorient the needle and hand-off task performed by PSM_A. M_i indicates all manual tasks and A_i indicates all automated tasks. ($i \in$ 1,2)

M_2 Grabbing the needle with the PSM arm (PSM_M).

A_1 Tightening the suture and translating to the next suture point.

A_2 Aligning the needle with PSM_A's jaw for completing hand-off.

1.3 Problem Definition

Several robotic systems have been developed for minimally invasive surgery [10] [44]. In most of the surgical robots available in the market, the surgical procedures are still entirely performed by human surgeons. The procedures requiring high skill levels such as suturing, knot tying and tissue dissection etc. are still dependent on the expertise of surgeons because the input received from the surgeons are imitated by the robotic system through computer control [73]. Suturing plays an important role in MIS and there are many technical challenges associated with it due to the complexity [20, 82]. Most research work focus on imitating the surgeon or manual trajectory planning to perform suturing, as a result the system barely mimics the user movements and still depends on the performing surgeon's skill level.

Recently, deep RL has shown potential of applying RL to real-world applications. The advantages of applying RL include helping novice surgeons improve their surgical skills

through training, reducing clutching frequency during teleoperation, circumventing errors that occur in model creation due to unreliable estimation from the perception system, dealing with high dimensional data and being able to improve user's trajectory [38, 73, 88]. This thesis presents a RL-based approach to automate suturing. A collaborative operation between a surgeon and the robot is studied. For performing automated robotic suturing, suturing task is analyzed and a few RL algorithms are implemented and tested for comparisons. Taking inspiration from observations found during this research, the goal is to develop a RL framework using a real-time dynamic simulator for the dVRK system that could perform collaborative laparoscopic procedures with surgeons. It could be achieved by automating a sub-task performed by one of the robot's arm to reduce the strain on surgeons leading to a standardized suturing hand-off technique (not varying based on surgeon's level of training and practice), and possibly augment the robotic system efficiency by reducing the operation time [77].

- ***Problem 1:*** Identifying optimal trajectory for completing Task A_1.

Given a distinct value function of the environment, an optimal policy is to be computed to provide actions to the PSM to perform the hand-off task autonomously.

- ***Problem 2:*** Addressing the varying suture styles of surgeons.

Surgeons are known to have varying suture styles. To accurately assist surgeons during procedures it is required to reciprocate their respective suture styles.

Assumption: Tasks M_1 - M_2 have been completed before hand-off task. The needle has been identified by the dVRK endoscope and the PSM_A has grabbed the needle in an orientation with the normal vector parallel to the needle profile, and the approach vector normal to the needle [Figure 4.3].

1.4 Motivation

The surgeon is responsible for many different things including diagnosis, preoperative process preparing both the patient and the team, the surgery, and postoperative checkups with the patient. Literature review shows that most, if not all, surgeons work long tiring hours performing surgery [14, 59, 83]. These surgeries include a variety of tasks, most of which can become complicated or tedious [77]. In the operating room, only the surgeon executes these complicated tasks causing consistent effort for them. The preoperative process consists of making sure the operating team is properly preparing the operating room, getting the patient ready, making sense out of the preoperative images, and marking the incisions. Once the surgery begins, the surgeon makes the necessary incisions and performs the surgery all while making sure the operating team is working at their best, and finally sutures the wound. By the time the surgeon reaches the end of the surgery, there is a possibility that the surgeon is tired and might not be able to maintain a good quality of suture [48]. On average, for a newly trained surgeon on the da Vinci robot, it takes approximately 5 minutes to complete a single stitch [12]. However, more experienced surgeons would perform this task much faster, it still takes a some amount of time. Another problem faced nowadays is the lack of trained and experienced surgical assistants, leading to increasing work load on expert surgeons [38]. All this points towards the need to automate surgical interventions which can lead to more efficiency in the operation theatres. Teleoperated surgeries with the da Vinci can be particularly laborious for novice surgeons due to the repetitive use of *clutching* - an action used to reorient the Master Tool Manipulator (MTM) without moving the Patient Side Manipulator (PSM). A previous user study [55] conducted at Worcester Polytechnic Institute on using the MTM to manipulate grippers in a small work space created in a simulation environment showed an average variation in a user's clutching frequency between 1 and 11 times, and an average path length coverage

of 15.7cm - 356cm. Due to the robot's mechanical structure, there are some constraints present on the freedom of motion on the user's end. Therefore, surgeons waste reasonable amount of time re-configuring the robot during the operating procedure [14, 73]. Fontanelli *et. al.* [20] found that while performing suturing task using da Vinci robot, user's had to use both the arms many times to perform hand-off task before each stitch. It all leads to increased operating time, fatigue, higher cognitive loading and drop in performance [73] [Figure 1.7].

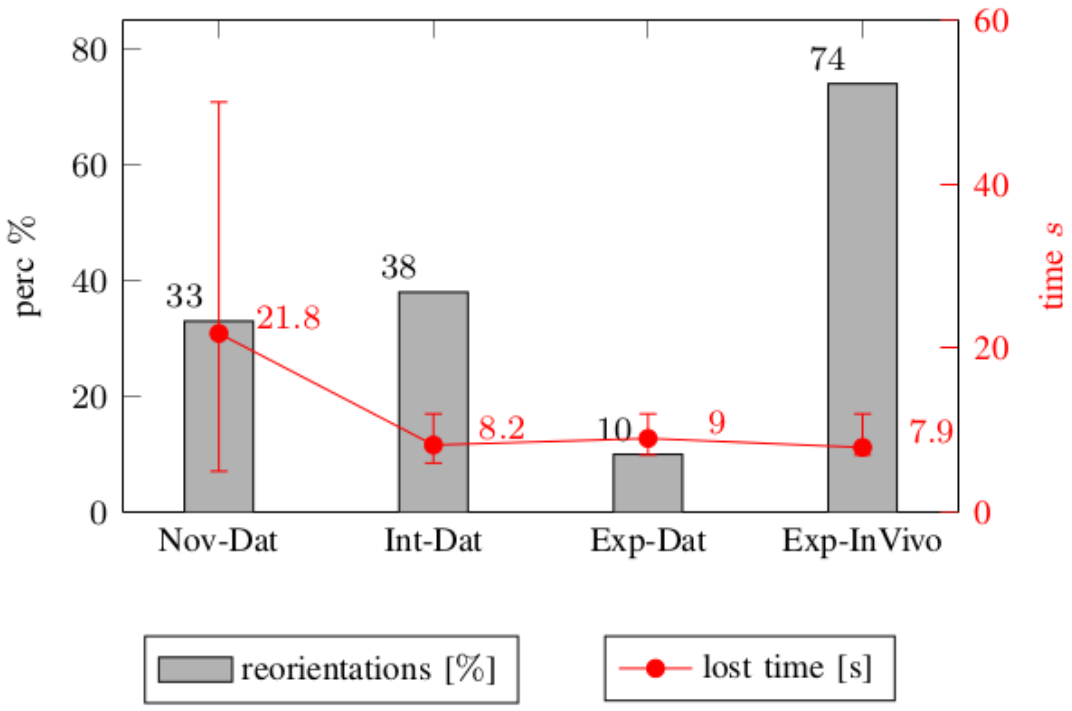

Figure 1.7: Percentage of stitches requiring needle reorientation and average/-variance of the time lost for reorienting the needle. Data provided by the JIGSAWS dataset (-Dat) and real procedures (-InVivo) performed by novice (Nov-), intermediate (Int-) and expert (Exp-) surgeons. [20]

Machine learning techniques can process large amounts of data and aid in extracting surgical skills, learn through expert human demonstrations and transfer it to surgical robot

to make it an intelligent system. It could help change the role of surgical robot from being an instrument that enhances surgeon dexterity to a more crucial role involving helping the surgeon. These systems could perform the trivial tasks and the surgeons could reduce or ease the the burden on them allowing the surgeon to not worry too much about the trivial tasks and save their time and energy [38, 44]. Due to this increase in time and energy, the surgeons would be able to treat more patients during their scheduled hours [77]. The surgeon spending less time in the operating room would decrease the cost of a single surgery for the patients.

Currently, there are no set standards on the quality of sutures. Even though the quality of suturing has a large effect on the patients healing and recovery process [2]. It has been proven that for larger stitch lengths, over 5cm, there is a much greater chance of a post-operative infection at any period of healing [34]. The eventual automation of suturing could standardize the quality of the process. A benefit of this work is that the robot could potentially aid or give feedback to the surgeon while suturing is being performed giving the surgeon the opportunity to correct his motions and react accordingly [41]. Through repetitive training using the learnt trajectory as reference trajectory, it can aid in improving the skills of the surgeon [82]. Better accuracy reduces the chance of the stitch being too close to the wound edge as well as larger stitch lengths preventing the chance of infections [20].

There are many different ways to make a robot perform a task or follow a trajectory. The trajectories obtained using common methods such as set point tracking, would need to be reconstructed each time the goal position is varied which can be tedious. Another problem faced in robotic applications is the presence of possible errors or uncertainty in the modeling of the environment because of the robot perception system providing inaccurate estimation of poses or due to sensor noise. It can result in unreliable transition dynamics, i.e. when an action a is taken from state s, if it is supposed to reach state s',

but if the transition dynamics is unreliable, there are chances it can end up reaching some state s'') [40]. Therefore, a goal based planning algorithm such as RL could be helpful to overcome these challenges [5].

With the introduction of deep learning into RL field, it could be beneficial to robotics due to the freedom in formulating complex RL problems. Renaudo *et. al.* [68] showed that there is a similarity between how mammals learn and concepts used by RL. It depicted that mammals learn through a combination of two behaviors, goal oriented and habitual behaviors. Initially, a task is performed repeatedly using a goal oriented approach and once we have learnt how to perform the task, it becomes habitual to us. RL being similar in concept, it could help develop intelligent systems that are flexible. RL algorithms build an internal model of the environment, plan an action to take based on the changes occurring in the environment, to avoid wasting time planning after learning how to perform a task.

Robotic applications can differ from locomotion to manipulation, which require tedious heuristic tuning due to high complexity of robotic systems. Therefore, applying RL in robotics would need large number of interactions with the environment. For overcoming this problem, robotics researchers have tried to take advantage of using simulations for running large number of iterations because it would be impractical to run large number of iterations on real-world robots. Especially, if the systems are delicate or if it's a costly control system, unexpected joint movements could be dangerous to the robotic platform. Along with all that, it would also be a very time consuming task. Simulations help overcome the issue of fast data collection. If the dynamics of the robot system is computed and modeled accurately in simulations, it could avoid any errors during transfer of model to real-robot. An ideal simulation should model the robot accurately to make the task of transferring learnt model to real world robot easy and hassle-free [86].

In March 2019, Richter et. al. [69] released the first open source RL environment for surgical robotics. It modeled the dVRK PSM in V-REP. In order to run the simulation in

real time, they didn't consider the dynamics of the dVRK system. The reasons mentioned for not considering dynamics included speeding up the simulation time, difficulty in modeling dynamics of the PSM arm and avoiding instability. But, due to the possibility of the robot undergoing wear and tear, the dynamics of the robot could differ and transferring the model to a real robot could require modifications and manual tuning of parameters [40]. Therefore, it could be beneficial for the RL framework to include the dynamics of the robot system [55].

1.5 Contributions

Sometimes it can be hard to find solutions to problems and provide good research contributions. But a good problem would be the one that requires making good research contributions for solving the problem. First, a way to automate suturing hand-off task along with capturing a user's unique suturing style is provided. Another challenge faced while applying learning techniques to robotics is the need to make efficient use of the data collected which can be time consuming and expensive at times. This work also tries to address this issue by utilizing the advantages of simulation. The second approach explores using a real-time dynamic simulation to collect and train robust learning algorithms (model-free deep RL) to reach a desired goal within a user-defined error margin.
This work would be the first to develop a real-time open source RL framework for the dVRK system. The term real-time indicates the computation and consideration of dynamics of the robot, while training and testing the robot in simulation. It combines two separate problems of creating the required RL framework and automate hand-off task in order to help solve a greater problem of automating suturing. The results obtained could help enhance the research on automating surgical tasks. The major contributions could be summarized as:

- Develop the first real-time open source Reinforcement Learning Framework for da Vinci Research Kit (dVRK) using AMBF dynamic simulator
- Automate Suturing Hand-off task using Q-Learning and utilize Learning from Demonstrations technique for creating a custom reward function to learn a unique suturing style
- Implement Model-Free Reinforcement Learning techniques such as Deep Deterministic Policy Gradients (DDPG), Hindsight Experience Replay (HER) with DDPG for performing PSM reach task
- Provide module for creating RL compatible Expert data from collected dVRK ROS data
- Provide a qualitative comparison of results between different Reinforcement Learning agents
- Contribute in the development of AMBF dynamic simulator through upgradation of relevant files and fixing minor issues
- Collection of collaborative suturing data using the da Vinci Research Kit

1.6 Outline of Thesis

In this chapter, an introduction to surgical robotics and the motivation behind problem being considered was detailed along with brief overview of previous works. Chapter 2 is an introduction to the world of RL. The starting few sections help the reader get used to the terms and techniques used in RL. The next few sections explain the algorithms used for inverse RL and how deep learning has been been integrated with RL to form the state of the art algorithms. Chapter 3 provides an introduction into the simulation environment

used for this work. Chapter 4 describes the methodology followed throughout this work. Initially, the problem was simplified by considering it as a discrete space problem. Then a solution for continuous space system is presented by developing a RL framework for dVRK PSM using a dynamic simulator. It also includes a section which discusses the dynamic simulator being used for this work and parameters that could be varied to interface with a learning model. Chapter 5 discusses the results obtained through application of various algorithms on the dVRK PSM. Chapter 6 discusses the results obtained and describes the future work and the applications of the proposed work. Chapter 7 is the Appendix section which contains some miscellaneous information.

1.7 Publication

The initial discrete space RL approach is written as a research paper titled "Collaborative Suturing: A Reinforcement Learning Approach to Automate Hand-off Task in Suturing for Surgical Robots " and is submitted for review to the 29th IEEE International Conference on Robot and Human Interactive Communication, RO-MAN 2020. It's submitted for the special session topic "Towards intelligent and natural Human-Robot Interaction in Medical Robot applications".

1.8 Prior Work

One of the earliest notable work towards automating trivial surgical tasks was put forward by Kang *et. al.* [37]. They suggested computing a simple predetermined path and then execute the computed steps without updating the trajectory for a different environment. Over time, researchers started exploring Artificial Intelligence based techniques to automate surgical tasks. EndoPAR system was developed at Technical University of Munich which could perform knot-tying task using recurrent neural networks [50]. In order to improve

the performance of robotic suturing while still keeping the surgeon in charge, Padoy *et. al.* [61] distributed the process of collaborative suturing into sets of automated and manual tasks. By automating certain sub-tasks of suturing, the authors presented promising results demonstrating clutch-less suturing, a step which can allow faster movements and reduce fatigue on the surgeon during teleoperated surgeries. Kassahun *et. al.* [38] summarized the need and advantages of using machine learning techniques for developing intelligent systems that can perform autonomous surgical actions. Research groups from around the world have investigated techniques to automate surgical tasks employing some iteration of the following concepts, Learning from Demonstration (LfD) [61], Iterative Learning Control [83], Gaussian Mixture Model [67], Deep Reinforcement Learning (DRL) [81], Sequential Convex Optimization [74], Visual Servoing [35], etc. The two main approaches followed are complete autonomous suturing and collaborative suturing. One of the state of the art techniques which uses a vision-based system to automate suturing completely was put forward by Leonard *et. al.* [44, 45]. Once the surgeons specify the placements of each stitch, the robot automatically performs the suturing task with better accuracy and shorter time than a surgeon performing the same tasks using laparoscopic tools. It utilizes the Smart Tissue Anastomosis Robot (STAR) surgical robot, a special suturing tool, force sensors and vision system for performing suturing on planar phantoms. They also showed the effects of using Near Infrared Fluorescence Imaging as markers for guiding the vision systems used in STAR surgical robot. Even though this approach presented great results, it required additional attachments and is only compatible with the STAR surgical system. Sen *et. al.* [74] automated the suturing task completely using sequential convex optimization technique and designing a needle holder. But it removed the surgeon from the loop and gave complete control to the robot. One of the problems faced by autonomous agents nowadays is the low acceptance rate from common people [38]. There are also a lot of legal and ethical issues in bringing autonomous agents into the operating room [88].

Therefore, a collaborative approach could be more suitable, where the surgeon retains complete control and the trivial surgical tasks are performed by the robot during a surgery. Padoy *et. al.* [61] demonstrated through a LfD technique that collaborative tasks between operators and RAS were possible. Recorded demonstrations by operators were provided as inputs to a Dynamic Time Warping (DTW) algorithm to learn the trajectory using the dVRK [61]. Another approach followed by a group at Germany was a mix of Gaussian process regression, DTW, force control and LfD techniques to achieve automation of surgical tasks [59]. But one of the shortcomings mentioned by them was the performance depended on the quality of learned demonstrations. As future work, they suggested investigating RL to provide autonomous improvement in the performance of automated motions. In 2019, Richter *et. al.* [69] developed the first open-sourced RL environment for surgical robotics using V-REP. It helped open up the surgical robotics field to RL researchers worldwide. They were successfully able to show the transfer of a trained model to physical dVRK system for automating the task of debris removal with few modifications. The model was trained using HER for two tasks - reach a desired goal and pick up an object. However due to difficulty in modeling dVRK PSM dynamics and for real-time computation, they didn't account for the dynamics of the robot.

Chapter 2

Background Knowledge

RL is a semi-supervised machine learning model. It learns the behavior or the set of actions to be taken from each state based on the interaction between an agent and the environment. The agent is the one who takes an action based on the rewards received and the environment is the world (virtual or real) where the agent performs the actions. The agent tries to achieve or reach the goal state by maximizing the rewards. The problem is formulated as a Markov Decision Process (MDP). For example, collecting maximum points in an arcade game or in a game of chess, one can learn to take a preferred action from each position in the board. The concept of RL is inspired from how humans learn to perform a task as a child. For example, getting a candy for behaving in a good manner and get a scolding for bad behaviors. Similarly, when the agent takes a wrong decision or action, it is penalized and it's rewarded for the right decisions. RL algorithms have been successful in beating world champions in games such as Go. Since then the RL field has been developing and performing a variety of tasks. RL uses Markov decision processes (MDP) as it's base framework to define the interaction between an agent and environment. All MDP's follow the Markov Property which states that the conditional probability distribution of future states is only dependent on the current state and is independent of the previous states [87]. A MDP is defined as a tuple (S, A, P, R, γ), where the terms mean the following:

- **S**: finite set of states
- **A**: finite set of actions
- **P**: state transition probability matrix (probability of ending up in a state after taking a particular action from a state)
- **R**: reward function
- γ: discount factor, ($0 < \gamma \leq 1$)

Then, a policy π is a probability distribution over actions given states. It's the likelihood of taking any action when an agent is in a certain state. An advantage of RL techniques is that they can make corrections to the original prediction as new information is gained during the interaction with the environment. The major terms that would be used commonly in the upcoming sections include agent, environment, state space, action space, rewards, policy and episode. In a RL problem, an agent is the one who makes the decision regarding what action to take next from the present state. An environment could be defined as a model created of the task to be performed in the world (real or simulation). An agent interacts with the environment and receives reward for certain actions while trying to reach the terminal state. State space is defined as the set of all possible configurations an agent can have in a given environment. Action space could be defined as set of all possible steps that an agent can take to transition between two states. Reward is the numerical signal received by the agent for taking an action in the environment. A policy defines the learning agent's way of behaving at a given time. A policy is a mapping from perceived states of the environment to actions to be taken when in those states. Finally an episode includes all the states visited by an agent between the agents initial state and terminal state.

2.1 Reinforcement Learning

Based on the problem, one can assign immediate rewards or give delayed rewards for certain actions. In certain scenarios, short term rewards are preferred over long term rewards and vice versa. This concept of short and long term rewards is suited to solve problems in a real world environment and are expected to perform well in problems where one specific action has to be chosen from vast number of actions while keeping in mind the final goal. The problem can be solved by using mathematical optimization techniques to maximize the reward collected. The two main ways to solve a RL problems are:

- Model-Based Reinforcement Learning
- Model-Free Reinforcement Learning

2.1.1 Model-Based Reinforcement Learning

Model-Based algorithms use a constructed model of the environment to predict or find the next state of an episode (either provided by the user or created after few interactions with the environment in simulation). All the interactions are done with the constructed model of the environment and not in the real environment. The two main ways of solving a model-based RL problem are using policy and valued based techniques. As an initial approach, value based technique was explored in this work because value based can be terminated after running a certain number of iterations compared to policy based where it's run until an optimal policy is found [80].

- **Policy based:** It is also called policy iteration. The algorithm keeps changing the policy directly after each iteration or receiving inputs from the environment. Initially, a randomized policy is assumed and after each iteration, the policy is changed to get a better policy. In each iteration, the algorithm tries to find a better policy,

until an optimal policy is found. A number of iterations of policy evaluation and improvement steps occur. Since one of the assumptions for RL included that we are dealing with finite MDP's, this process should theoretically converge to a optimal policy over a finite number of iterations.

- **Value based:** It is also called value iteration. One of the problems with policy iteration is that it can take a lot of iterations for the algorithm to finally converge to an optimal policy. The policy evaluation step can be reduced to occur only at the first sweep of policy iteration and this is called as value iteration. Theoretically, value iteration also can take infinite number of iterations to converge to an optimal value function v_π [80]. But the algorithm can be stopped from proceeding further after a given number of iterations or when the value function value for a state doesn't change over a specified threshold. Value iteration consists of one iteration of policy evaluation and improvement. Algorithm 1 summarizes the steps followed for solving a RL problem using value iteration [80] utilizing the terms defined in the previous section (δ is error margin set by user).

Algorithm 1: Value Iteration Algorithm

Assign V a random value for initialization, ex:V(s) = 0, for all s $\in \boldsymbol{S}^+$
while $\Delta < \delta$ **do**
  $\Delta \leftarrow 0$
  for *each* $s \in \boldsymbol{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end
end
Compute a deterministic policy, π*, which satisfies*
  $\pi(s) = \underset{a}{argmax} \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')]$

Dynamic Programming

Dynamic Programming (DP) is a technique to solve complex problems by simplifying them into sub-problems, solving the sub-problems individually and finally combining the results obtained from each sub-problem to get the solution for the initial problem. It can be used to find an optimal policy for a given model of the environment (needs to be constructed as a MDP). Since a MDP uses Bellman equation, DP algorithms can be used to solve a MDP [80]. Bellman equation can be used for recursive decomposition and it also breaks down an optimal value function into finding the optimal behavior for each consecutive steps followed by the agent. For the model to be created, the algorithm needs to be provided with the transition probabilities for each state and rewards received. Classical DP algorithms form the theoretical foundations for understanding different RL algorithms. But DP algorithms have many drawbacks due to their assumption of the model being considered as perfect and are known to be computationally expensive. As a result, they have limited applications in real life.

$$\begin{aligned} V^*(s) &= \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \\ &= \max_a \sum_a P^a_{ss'}[R^a_{ss'} + \gamma V^*(s_{t+1})] \end{aligned} \tag{2.1}$$

or

$$\begin{aligned} Q^*(s,a) &= E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \\ &= \max_a \sum_a P^a_{ss'}[R^a_{ss'} + \gamma V^*(s_{t+1})] \end{aligned} \tag{2.2}$$

$\forall s \in S,\ \ \forall a \in A(s),\ \ \forall s' \in \mathrm{S}^+$

The action value function q_π to follow a policy π and take an action A_t from state S_t can be calculated using Equation 2.3.

$$q_\pi = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \quad (2.3)$$

Where $\mathbb{E}_\pi$ represents the expected value of the numerical terms, G_t represents the expected reward received by the agent from the current state s, γ represents the discount factor, R_{t+k+1} represents the reward received from the state after interacting with the environment, a indicates the action taken. Once the agent estimates the value functions for each state, it chooses the path leading to the goal state by selecting the maximum value of rewards Equation 2.4 and repeats itself until the goal state is reached [80].

$$\begin{aligned} v_*(s) &= max_\pi v_\pi(s) \quad \forall s \in S \\ q_*(s,a) &= max_\pi v_\pi(s,a) \quad \forall s \in S, \ \ \forall a \in A(s) \end{aligned} \quad (2.4)$$

2.1.2 Model-Free Reinforcement Learning

Model-Free techniques learn from direct interactions with the environment (it can be real world or simulation world). Contrary to model-based techniques, model-free techniques don't construct it's own version of the model of the environment. It performs the actions from a given state in the environment, collects the reward and updates the expected value of the state. In model-free techniques, a prediction is made about the expected reward for following a policy $\pi[a|s]$ and then the policy is changed in a way to maximize the rewards received by the agent. There are two types of policy learning methods:

- **Off-Policy:** Off-Policy methods learn different policies for controlling the behavior of agent in the environment and estimation of expected value of a state
- **On-Policy:** On-Policy methods learn the value of the policy that is used by the agent to make decisions in the environment.

In on-policy learning techniques, the algorithm changes the policy which is being used to interact with the environment. Policy evaluation and improvement occurs directly on the policy being followed by the agent. But in off-policy learning, the policy evaluation and improvement occurs on a different policy called the target policy, while the policy being followed by the agent is different policy called behaviour policy. The target policy may be stochastic or deterministic, but behavior policy is stochastic. Bootstrapping in RL means the value of a state is updated based on an estimation and not the exact value. Few of the common model-free techniques are Monte Carlo approach and Temporal Difference Learning (TD). In this work, TD learning technique was explored due to the advantages of utilizing bootstrapping.

Temporal Difference Learning

In TD learning, the current estimate is found using the previously learned estimate. It is a combination of DP and Monte Carlo learning techniques, it learns through experiences without the need for a model of the environment and also uses bootstrapping.

- Learn an optimal behaviour without needing a model of the environment dynamics
- Since it learns from interacting with the environments, it is suitable for problems which can be trained in a simulation.
- An advantage over Monte Carlo learning is that it doesn't wait till the end of an episode for updation of values, its updated at every time step.

$$Q_*(S,A) \longleftarrow Q_*(S,A) + \alpha[R + \gamma max_a Q_*(S',a) - Q_*(S,A)] \tag{2.5}$$

where R is the reward received when moving from the state S to the next state S' and α is the learning rate, γ represents the discount factor ($0 < \alpha, \gamma \leq 1$), . Once the agent

estimates the value functions for each state, it chooses the path leading to the goal state by selecting the maximum value of rewards and repeats itself until the goal state is reached. Two commonly used TD learning techniques are SARSA and Q-learning. SARSA is an on-policy TD control technique and Q learning is an off-policy TD control technique. In this work Q-learning was explored and the procedure to implement Q-learning is described in algorithm 2.

Algorithm 2: Q-learning (off-policy TD Learning)

Algorithm parameters: step size $\alpha \in$(0,1], small $\epsilon > 0$

Initialize

Q$_*(s, a)$, for all$s \in S^+, a \in A_*(s)$, arbitrarily except that $Q_*(terminal, .) = 0$

for *each episode* **do**

  Initialize S

  for *each step of episode* **do**

    Choose A from S using policy derived from Q$_*ex : \epsilon - greedy$

    Take action A, observe R, S'

    Q$_*(S, A) \longleftarrow Q_*(S, A) + \alpha[R + \gamma max_a Q_*(S', a) - Q_*(S, A)]$

    S $\longleftarrow S'$

  end

  if *S is terminal* **then**

    break

  end

end

2.2 Learning from Demonstrations

Real world problems are usually very complex and have a large state space. Recent research has shown that learning from demonstrations could play a vital role in the development of complex systems which are operating on state of the art learning algorithms [58, 61]. It

can sometimes become very difficult for a user to specify the reward function in a way that the algorithm is able to learn. Also for certain problems, the task is so complex that it is very unlikely that the algorithm would just randomly stumble upon the goal (chances of that occurring are usually very low) [31]. It leads to the algorithm getting trained without ever getting to complete an episode successfully or even see the goal. This can negatively effect the training of the model since the model is not getting to see any positive outcomes and is trained only on negative rewards. Therefore, this problem of exploration can be solved by providing the algorithm with expert demonstrations [57]. It helps guide the robot during the initial trials and helps improve the training outcome [56]. Currently, a lot of research is going on regarding how to best utilize expert demonstrations to help the training process [85]. Certain works have shown improved performance and faster learning rate when the algorithm is provided with some demonstrations [1, 57, 83]. It also helps the agent perform an action in a way which is very similar to the expert [64]. In this thesis, initially Inverse Reinforcement Learning (IRL) was explored for incorporating demonstrations and later a module was developed which could help convert the recorded rosbag data (a set of tools for recording from and playing back to ROS topics) into a format suitable for providing as inputs to RL algorithms.

2.2.1 Inverse Reinforcement Learning

The concept of IRL revolves around the algorithm predicting the optimal policy an agent should follow such that the rewards is maximized. One of they key factors of an effective RL algorithm is the reward function because it decides how well the algorithm would perform. Therefore, the selection of a good reward function is vital because the reward function varies depending on the task being considered. The reward function could be represented by a linear function or sometimes even a very complex function. Unless it's a simple problem or the person has a previous experience with a similar problem, it is very

difficult for someone to find the right reward intuitively. IRL is one of the ways of finding the reward function which is tailor made for the desired expert trajectory. Based on the expert data provided, IRL finds a reward function which can help the agent learn a custom trajectory that needs to be followed. The concept was introduced by Ng *et. al.* [58] and there have been few other major work on this field including using the Maximum Entropy IRL [89], Deep Maximum Entropy IRL [87]. In this work, initially the Maximum Entropy IRL algorithm was studied and implemented for creating a custom reward to learn a unique trajectory. Later a sparse reward function was used as the final result for the discrete RL problem.

Maximum Entropy Inverse Reinforcement Learning

Maximum entropy works on the principle that each trajectory being followed has equal probability of occurring, but trajectories with higher regard are exponentially more preferred. Therefore, the algorithm tries to maximize the rewards by preferring exponentially more to visit the states and actions followed by the expert. Ziebart *et. al.* [89] put forward the algorithm for computing the state visitation frequency as shown in algorithm 3. A task such as suturing requires various complex actions. For automating the hand-off task, a custom reward function can help the agent learn the motion to be followed. Once the features defining each state is selected, the features of each state are linearized using a set of weights θ. Equation 2.6 depicts the probability of following a trajectory ζ using a parametrized set of weights (θ). It shows that paths with equivalent rewards have equal probabilities, but the paths which give higher rewards are exponentially more preferred.

$$P(\zeta_i|\theta) = \frac{1}{Z(\theta)} e^{\theta^T f_{\zeta_i}} = \frac{1}{Z(\theta)} e^{\Sigma s_j \in \zeta_i^{\theta^T f_{s_j}}} \tag{2.6}$$

where Z(θ) is the partition function for the reward function weights θ, s represents the states, f_{s_j} represents the features of the states for a trajectory ζ. The algorithm tries to

minimize the difference in value of features between the expert demonstrations and the computed optimal policy for a given set of weights θ. Equation 2.7 depicts how θ is chosen such that it maximizes the likelihood of that specific trajectory occurring.

$$\theta^* = \underset{\theta}{argmax} L(\theta) = \underset{\theta}{argmax} \sum_{examples} logP(\zeta_i|\theta) \tag{2.7}$$

$$\nabla L(\theta) = \widetilde{f} - \sum_{\zeta} P(\zeta|\theta, T) f_\zeta = \widetilde{f} - \sum_{s_i} D_{s_i} f_{s_i} \tag{2.8}$$

The maximized value of θ is found using gradient descent (Equation 2.8). For this problem, the gradient can be defined as the difference between expected feature counts and expert users feature counts (represented by D_{s_i} as shown in algorithm 3).

Algorithm 3: Algorithm for Computing State Visitation Frequency [89]

Backward pass

1. Set $Z_{s_i,0} = 1$

2. Recursively compute for N iterations

$$Z_{a_{i,j}} = \sum_k P(s_k|s_i, a_{i,j}) e^{reward(s_i|\theta)} Z_{s_k}$$
$$Z_{a_i,j} = \sum_{a_{i,j}} Z_{a_i,j}$$

Local action probability computation

3. $P(a_{i,j}|s_i) = \frac{Z_{a_{i,j}}}{Z_{s_i}}$

Forward Pass

4. Set $D_{s_i,t} = P(s_i = s_{initial})$

5. Recursively compute for t = 1 to N

$$D_{s_i,t+1} = \sum_{a_{i,j}} \sum_k D_{s_k,t} P(a_{i,j}|s_i) P(s_k|a_{i,j}, s_i)$$

Summing Frequencies

6. $D_{s_i} = \sum_t D_{s_i,t}$

Algorithm 4: Maximum Entropy IRL Algorithm

Input: Expert Trajectories (D), MDP (S,A,T,γ), Observations (s)

Result: Optimized weights for computing rewards obtained from each state

1. Randomly initialize the weights θ

2. Load and parse through the expert demonstrations D

3. Solve for Optimal policy $\pi(a, s)$ w.r.t reward r(τ)

4. Find State Visitation Frequency $P(s|\theta)$ using algorithm algorithm 3

5. Compute the gradient ∇_θL = $\frac{-1}{m}(\sum_{s \in \tau_d} \frac{dr(s_d)}{d(\theta)} + \sum_{s \in \tau} P(s|\theta, T) \frac{dr(s)}{d\theta})$

6. Update θ with gradient step using ∇_θL

7. Repeat from Step 2 until defined number of iterations

2.3 Combining Deep Learning and Reinforcement Learning

2.3.1 Deep Reinforcement Learning

Applying artificial intelligence to real world problem is a very challenging task because even a small variation in the environment can cause a drastic change in the results. The variations in conducting the trial can result in the algorithm performing poorly or give a completely false output [28]. For example, a small change in brightness of an image can produce erroneous results, a change in camera orientation can also cause vision systems to produce wrong results. Our brains are capable of separating out data and account for the variations that occur in the environment, but a computer would not be able to filter the errors. In most of the cases, the only way to process such raw data is using sophisticated machines [60]. Traditional learning techniques such as machine learning, representation learning etc. can not account for such small variations in the data because it is very difficult to represent the problem as it is to solve the problem [84].

Deep learning solves this problem by introducing representations which enable the computer to build complex concepts out of simpler concepts. A typical example of a deep learning model is the feedforward deep network, or also called as multilayer perceptron (MLP). A multilayer perceptron is just a mathematical function that maps a set of input values to output values [26]. A complex function is simplified and expressed as a combination of simpler functions. The depth of the algorithm decides the number of steps that a computer can learn from analyzing the data. The complex mapping between the input and output data is converted into a series of nested simple mappings and each layer contains one of those simple mappings. The data is provided as input to the first layer called the input layer. The number of nodes in the input layer depends on the number of features of

the input data. Then a series of hidden layers extract the desired information from the input data. It is called hidden layers because the values computed at the nodes in each layer is not provided by the user, instead it is learned by the algorithm after a certain number of successive passes through the data. In RL, an agent needs to learn a task through a number of trial and error without inputs from a human operator. In 2015, DeepMind demonstrated that when deep learning is combined with a reinforcement learning system, the algorithm's performance improves and is able to learn complex tasks such as learning how to play Atari video games and even achieve human level performance [52]. Deep learning has significantly improved the performance of RL in the robotics field [40].

In robotic applications the dimensionality and size of the state space is very large. As a result, it becomes difficult to define a value function or reward for individual states in the state space. A possible way to compute the value or rewards of each state would be by utilizing the features of a state space (states with similar features get same values or rewards). The value function could be found using simple function approximation techniques. The feature representation, f, is usually hand-crafted by the user (segmentation, manually defined distance metrics). The choice of model used for function approximation is vital and plays an important role on the ability of the algorithm to figure out a relationship between the state feature vector f and user preference. For simple scenarios, one could map the state to reward or feature using a simply weighted (θ) linear combination of feature values ($g(f, \theta) = \theta^T f$) [58]. But for most of the real world problems the optimal value or reward function can not be accurately approximated by a linear model and can result in huge errors. A solution could be to use a deep parametric architecture to approximate the value or reward functions [87]. It could help improve the accuracy of nonlinear function approximation. A real-world problem can be very complicated and tough to be solved using a RL algorithm. For example, real world scenarios include many agents interacting with each other within the same environment. One of the possible ways to solve such

complex problems is by using the concept of deep learning. Deep learning can be used to represent a relationship between two entities as a complex function. Therefore, a great way to solve real world problems using RL is through integration with deep learning called Deep Reinforcement Learning (DRL). One of the basic RL techniques that uses neural network to parameterize the Q value function is Actor-Critic Methods [80].

2.3.2 Actor-Critic Methods

Actor-Critic (A2C) methods combine value and policy-based methods to find the optimal policy. An A2C contains two main components that run in parallel, an actor and a critic. The agent uses an actor for a policy based approach which learns by interacting with the environment and uses a critic network to evaluate the quality of actions predicted by the actor [27]. Using the combinations of two networks is more stable than value based methods and can also help speed up the training process. Algorithm 5 shows the steps involved in solving a problem using A2C (α_θ, α_w are the learning rates for actor and critic).

Algorithm 5: Actor-Critic Algorithm

Initialize parameters s, w, θ, α_θ, α_w; sample a~$\pi_\theta(a|s)$

for *t = 1 to T* **do**

Sample reward r_t ~$R(s,a)$ and next state s'~$P(s'|s,a)$

Sample the next action a'~$\pi_\theta(a'|s')$

Update the policy parameters: $\theta \leftarrow \theta$ + $\alpha_\theta Q_w(s,a) \nabla_\theta \log \pi_\theta(a|s)$;

Find the TD error to correct action-value at time t:

$$\delta_t = r_t + \gamma Q_w(s',a') - Q_w(s,a)$$

Use the previous computed value to update the parameters of Q function:

$$w \leftarrow w + \alpha_w \delta_t \nabla_w \mathbf{Q}_w(s,a)$$

Assign values: $a \leftarrow$ a'ands $\leftarrow$ s'

end

2.3.3 Deep Deterministic Policy Gradient

The most commonly used techniques to find an optimal policy is using Q value function. The agent estimates the value of taking a particular action from a state and tries to follow the path which maximizes the reward. But as mentioned in previous section, a huge drawback of the above technique is that it is not practical to be applied in real world problems due to large states spaces [47]. Even though neural networks could be used to estimate the value functions, the agent has to estimate the value function before finding an optimal policy. For real world scenarios where the environments are continuous and complex, it is time consuming and computationally expensive to find a value for each state in the environment [66]. Since, the main goal of RL is to get the maximum reward, finding and evaluating policies can be more advantageous. Also, classical RL techniques are not suited for environments having large action spaces and continuous spaces [40]. Techniques such as policy gradient can be used in such cases where the optimal policy can be found by evaluating the policy at each step. Deterministic policies take a state and return a single action, whereas stochastic policies take a state and return the probability of taking an action. For robotic applications, the robot requires continuous action inputs from the algorithm.

Deep Deterministic Policy Gradient (DDPG) solves the above problem and provides the robot with a continuous set of actions which can be directly applied to the robot. The policies formed are deterministic and not stochastic in nature. Similar to A2C method, DDPG also uses two neural networks to estimate the value of a function and the estimated Q function to select the optimal policy. For maintaining stability during the learning process, it uses a target network. Target network uses two more neural networks, one for prediction and another for estimating the target. Therefore, DDPG consists of 4 neural networks running in parallel. The target network is updated through soft-updates which means the regular network is slowly mixed with the target network (updated by a very

small amount). The actor network weights are updated using policy gradients and the critic network weights are computed using TD error. Similar to deep Q-network, DDPG also uses the concept of replay buffer to store state transitions and samples them randomly [52]. Algorithm 6 illustrates the steps required to implement DDPG [47].

Algorithm 6: Deep Deterministic Policy Gradient Algorithm [47]

Randomly initialize critic network $Q(s, a|\theta^Q)$ and action $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q\prime} \longleftarrow \theta^Q, \theta^{\mu\prime} \longleftarrow \theta^\mu$
Initialize replay buffer R
for *episode = 1, M* **do**
  Initialize a random process N for action exploration
  Receive initial observation state s_1
  for *t=1, T* **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathbb{N}$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathbb{R}$
    Sample a random mini-batch of N transitions $(s_t, a_t, r_t, s_{t+1})$from$\mathbb{R}$
    Set $y_i = r_i +' (s_{i+1}, \mu'(s_{i+1}|\theta^{\mu\prime})|\theta^{Q\prime})$
    Update critic by minimizing the loss: L = $\dfrac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a \, Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \, \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q\prime} \longleftarrow \tau\theta^Q + (1-\tau)\theta^{Q\prime}$$
$$\theta^{\mu\prime} \longleftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu\prime}$$

  end
end

2.3.4 Hindsight Experience Replay

For many real-world problems, designing a reward function can get very tedious and complex. Instead of going through a complex reward shaping technique, it would be very convenient if the user could just provide a reward at the end for accomplishing the goal. For example, give the agent +1 reward if it is able to lift the cube up or reach the terminal position. But this gives rise to another issue, the state space for many real-world problems is usually very large and it becomes very difficult for an agent to explore the whole state space efficiently. Therefore, the agent doesn't receive a reward until the goal is accomplished. For certain models, it can take millions of iterations before the agent can receive a reward because the chance of randomly stumbling upon the goal is very low. Consequently, the model ends up getting poorly trained and the ratio of positive rewards to negative rewards received is negligible. To solve this problem, researchers at OpenAI came up with a technique called Hindsight Experience Replay (HER) [3]. It utilizes the concept of replay buffer which was introduced by authors of deep Q-network [52]. They suggested the idea of adding fictitious data to the replay buffer by assuming the result to be different if the goal position would have changed. The authors suggested the following modifications in the replay buffer:

During an episode the agent is trying to reach goal state (G) from an initial state (S) after taking a series of actions (a_i). But the agent fails to reach the goal state (G) and ends up reaching a state (S') at the end of an episode (as shown below).

$(S_o, G, a_o, r_o, S_1), (S_1, G, a_1, r_1, S_2), ...(S_n, G, a_n, r_n, S')$

where r represents the reward received at the i^{th} episode after taking an action a_i from an initial state S_k . Now, a modification is made in the replay buffer. It's assumed that from the start, the agent's plan was to reach the goal (S'). This assumption leads to the outcome that if the goal state had been S' ($G = S'$), then the actions followed by the agent during that episode was correct and a positive reward would be received. Therefore, along

with the actual set of observations stored in the replay buffer, we add a fictitious set of observations (as shown below).

$(S_o, S', a_o, r_o, S_1), (S_1, S', a_1, r_1, S_2), ...(S_n, S', a_n, r_n, S')$

This technique proved to be novel and helped improve the rate of convergence of policy gradient based algorithms such as DDPG. The idea is very similar to how humans learn to accomplish a goal even from failed attempts. Therefore, however complex the environment or reaching the goal state is, the algorithm is provided with some positive experiences. Algorithm 7 describes the steps for implementing HER [3].

Algorithm 7: Hindsight Experience Replay [3]

Given:

- an off-policy RL algorithm *A*
- a strategy *S* for sampling goals for replay
- a reward function r: S X A X G $\rightarrow R$

Initialize A
Initialize replay buffer R
for *episode = 1, M* **do**
  Sample a goal g and an initial state s_o
  for *t = 0, T - 1* **do**
    Sample an action a_t using the behavioral policy from A:
      $a_t \leftarrow \pi_b(s_t||g)$
    Execute the action a_t and observe a new state s_{t+1}
  end
  for *t = 0, T - 1* **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition $(s_t||g, a_t, r_t, s_{t+1}||g) in R$
    Sample a set of additional goals for replay G:= S(current episode)
    for *g' $\in G$* **do**
      r' := r(s_t, a_t, g')
      Store the transition ($s_t||g', a_t, r', s_{t+1}||g') in R$
    end
  end
  for *t = 1, N* **do**
    Sample a minibatch B from the replay buffer R
    Perform one step of optimization using A and minibatach B
  end
end

Chapter 3

Simulation Environment

The recent advancements in the field of artificial intelligence and various learning techniques has lead to an increase in research towards automating mundane tasks. In order to deal with complex systems such as a surgical robot, a real-time dynamics simulation called Asynchronous Multi-Body Framework (AMBF) was developed by Munawar *et. al.* at AIM lab, WPI [55]. The spawned robots can be controlled using various input devices including the MTM's of the dVRK. It can be used to simulate and train various surgical robots.

AMBF is a multi-body framework that offers real-time dynamic simulation of multi-bodies (robots, free bodies etc.) coupled with real-time haptic interaction via several haptic devices (such as dVRK Manipulators and Razer Hydras). The robots spawned in simulation can be controlled using an inbuilt Python client. The users have the freedom to load any existing robot models or create their own robots using Blender or Solidworks. The creator of AMBF has added few existing complex robot models using Blender. It includes a model of the dVRK system (PSM, ECM and MTM) which was used for this work. The robot model configuration file can be used to specify all the information regarding the robot such as damping, parent links, number of joints, type of joints, type of link, relative positions of link, joint limits, PID gains for tuning the controller etc. It uses Yet Another

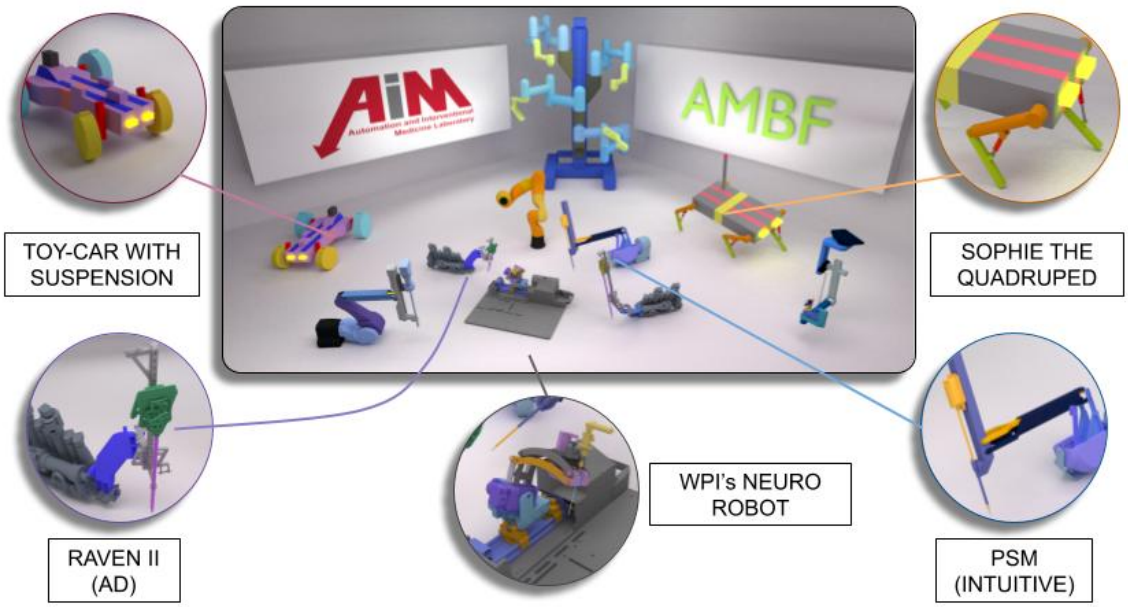

Figure 3.1: Examples of some robots simulated in AMBF [55]

Markup Language (YAML) to store and transmit configuration data files because of which the robot configuration files are readable to the user. AMBF can be used for training neural networks or RL techniques in real-time. It was developed using external tools such as an extended version of CHAI-3D, BULLET-Physics etc. Since the output can be computed in real-time, it enables AMBF to be interfaced with learning tools such as Keras, TensorFlow, Gym etc. Multiple robots can be loaded and trained at the same time in AMBF. The Python client provides control over the simulated bodies using afState (for reading current states) and afCommand (to move to body). The communication between the algorithm and AMBF can occur through Robot Operating System (ROS). Many complex robots can be simulated as shown in Figure 3.1. Figure 3.2 shows the dVRK system being simulated in AMBF. It shows the an ECM, PSM and MTM being simultaneously simulated in real time. The AMBF GitHub Wiki page can be referred for explanations on how to use various features of AMBF. It also provides users with examples to illustrate the implementation structure.

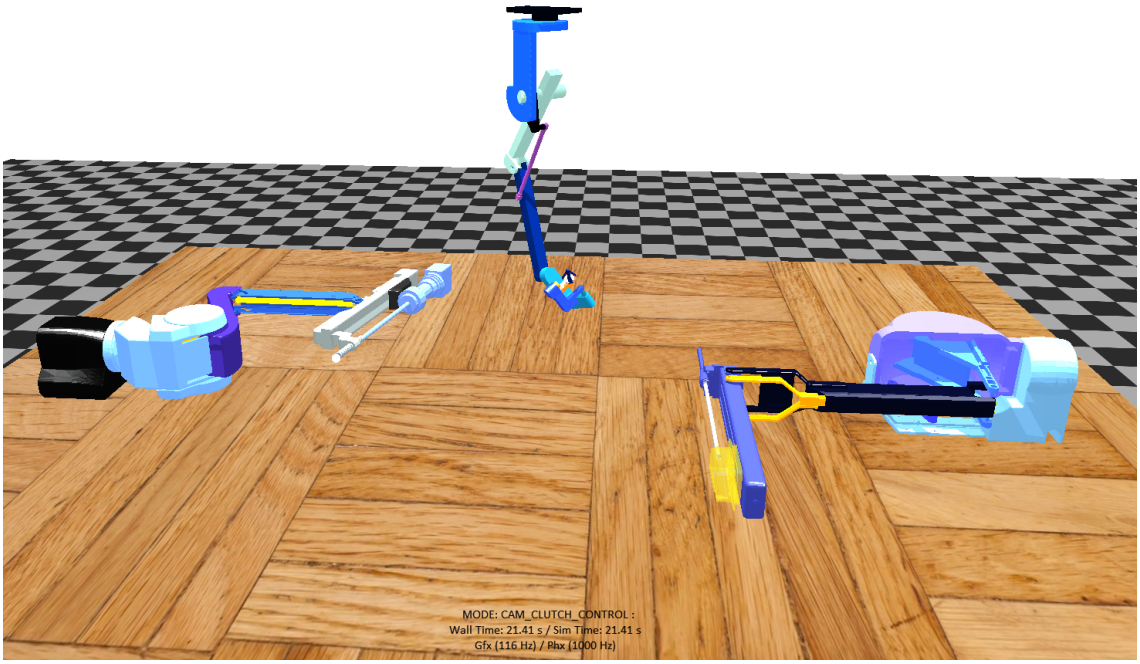

Figure 3.2: dVRK system simulated in AMBF (ECM (left), MTM (center), PSM(right))

Chapter 4

Methodology

One of the main principles followed for this work was to first get familiar with the concepts by implementing a basic example. Once a good foundation of the basics was achieved, a highly constrained form of the true problem was implemented. If satisfactory results was achieved from the previous step, one by one each constraint was removed. As a result the desired output can be achieved without much confusion, since it would be easier to pin point the exact reason for the error and could be dealt in a more systematic way rather than having to deal with all the problems at the end. Hence, the work is divided into two parts based on the type of environment - discrete and continuous space system. Initially, a classical RL approach was followed where the environment was discretized and a simple PSM environment model was created following the Gym structure [7] (the term simple indicates no rendering or visualization was implemented). Later, a continuous space system approach is provided using a dynamic simulator for training and evaluation. The initial discrete space RL approach is written as a research paper titled "Collaborative Suturing: A Reinforcement Learning Approach to Automate Hand-off Task in Suturing for Surgical Robots " and is submitted for review to the 29th IEEE International Conference on Robot and Human Interactive Communication, RO-MAN 2020.

To provide an overview of the upcoming subsections, initially the PSM environment was discretized and formulated as a grid world problem. Data was collected using the dVRK of users performing a simple running suture. The user data was processed and studied, based on which the parameters for the environment was selected. Using the collected expert demonstrations data, Maximum Entropy IRL technique was used to create a custom reward function using value iteration. Due to unsatisfactory results, a model-free RL technique Q-learning was implemented using the created PSM environment. For learning a custom style of hand-off task, sparse reward was used to create a custom reward function for Q-learning. The results achieved were impressive and as a next step forward, focus was shifted to continuous space environment. The lack of real-time RL environment for surgical robots lead to the development of a RL framework for surgical robotics using AMBF dynamic simulator (accounts for dynamics during training and evaluation). It computes the outputs in real-time and is compatible to be interfaced with state of the art RL algorithms. A wrapper was created for interfacing AMBF with standard RL modules such as Stable-baselines [30] and a PSM environment was developed which follows the Gym structure [7]. Finally, the PSM robot model in AMBF was used to perform training and testing using HER [3] and DDPG [47]. To solve the problem of exploration inefficiency, a module to generate RL compatible expert data was created for implementing techniques such as Generative Adversarial Imitation Learning (GAIL) [31] in the future. The obtained results are discussed in the next chapters.

4.0.1 Data Collection

As part of the work for discrete RL research paper, a set of 3 trials of collaborative suturing data was collected. For collecting initial data, the dVRK system was used. This experiment was conducted using the dVRK system and a 5/0 nylon monofilament black suturing

Different Users Collected Suturing Data (Start and End points)						
X, Y, Z coordinates (in mm) / S. no	X_{start}	X_{end}	Y_{start}	Y_{end}	Z_{start}	Z_{end}
User 1	-10	5	55	55	-135	-125
User 2	-130	-5	58	60	-142	-125
User 3	-4	10	52	50	-137	-130

Table 4.1: Table displaying the starting and ending points of different users while they were performing suturing hand-off task

needle manufactured by Assorted Surgical Sutures. A suture pad sold as part of Suture Practice Kit by Kenley was used as the base to perform suturing. The users involved in the data collection process were members of the team, no external users were involved in this experiment. Since the purpose of this work is to learn a particular user's style rather than generalizing the whole population, only a small number of users were required for the study. The MTM was used to control the PSM arm during the suture needle hand-off task. The recorded data included the PSM pose and joint position data, the clutching frequency of the operator, and the MTM pose and joint positions. A python script implementing the $StateToIndex$ function was created to process the operator specified trajectories and to generate sets of state-action pairs with appropriate discretization of action value ranges and concurrent states. The function $StateToIndex$ converts each point P_i in the processed array to it's associated index between 0 to $gridsize^3$. The $gridsize$ decides the number of points present in the grid world. In this work, the $gridsize$ is set to 11 and therefore a total of 1331 points are present in the grid world. For example, if the $gridsize$ is set to 3, a total of 27 points would be present in the grid world. Suture data of various suture styles was collected and the starting and ending points is tabulated as shown in Table 4.1.

4.0.2 Reinforcement Learning Model: Discrete Space

Initial steps included discretizing the whole state space, formulating the problem as a MDP and simplify the problem as a Two-dimensional (2D) and Three-dimensional (3D)

Grid world problem. In order to overcome the drawbacks of discretizing a system, many modifications were tried and implemented (explained in the following sections). In the following section, the methodology followed for discrete space system is described and then the problem specific implementations of the RL model is detailed.
In 2016, OpenAI Gym toolkit was launched for advancing RL research [7]. To run and train the agent, an environment which follows the structure of a gym environment was created. The main functions of a gym environment include:

- **Init Function**:

  For a given environment, the observation and action space is initialized in this function along with all the other required variables.

- **Step function**:

  The step function returns the next state (S') that the agent reaches when an action (A) is taken from state (S). It returns four values:

  - Observation: The new state the agent has reached.
  - Reward: The reward received by the agent for taking that action from the previous state.
  - Done Flag: It tells the user if the agent has reached the goal position.
  - Info: It is usually left blank, but could be used to provide any debugging information.

- **Reset Function**

  An episode is reset either to a predetermined state or randomly reset to any state present in the state space. The starting position of the agent, all the variables of the function are initialized again for a new episode. Depending on the user, an episode

could be reset if the agent reaches the terminal state or exceeds maximum number of iterations or crossed the time limit or any other user defined conditions.

- **Render Function**

  This function can be used to visualize the output. Gym provides few inbuilt models of environments such as Bipedal walker, Lunar Lander, Cheetah etc.

A MDP environment following the class structure as shown above was developed for the PSM in Python. Then to become familiar with RL algorithms and creating custom environments, the problem was formulated into a 2D grid world. The PSM was considered to be the agent and the environment was everything viewed by the endoscope camera of dVRK. All the code was implemented using Python and relevant learning modules. Once it was successfully implemented, all the individual functions was studied including the main concepts involved in the implementation of the algorithm, later a 3D world problem was attempted.

After creating a working 3D world PSM environment, few of the factors that need to be decided included the level of discretization and number of parameters required to define the state of the robot. If the problem was discretized into very small values, it would help make the trajectory smooth, but would make it computationally infeasible. A wider spaced discrete world would not be ideal because it could result in jerks during the robot movements. For a discretized system, the parameters required to define a state should be a minimalistic representation because based on the size of observations, the calculations needed to be computed at each iteration increases drastically and could end up becoming computationally expensive [38].

For finding the ideal discretization factor, the factors considered was the collected users data, required computational resources and the largest step the PSM could move without resulting in jerky movements. The smallest change in end-effector position of dVRK PSM

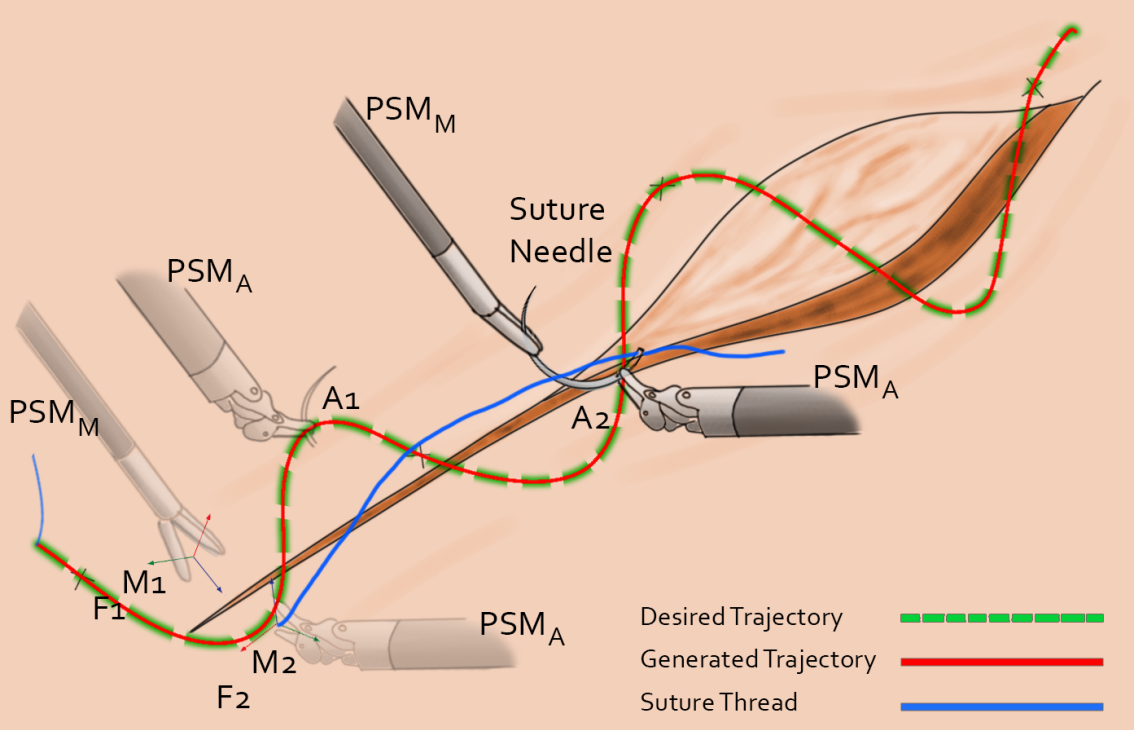

Figure 4.1: Demonstration of the entire operation

that could made through cartesian control is 1mm [11]. Based on Table 4.1, it can be seen that the difference between starting and ending positions of Y axis is very low. Hence, the discretized grid values could be set to 1mm or 2.5mm or 5mm or 10mm etc. The final decision on the grid size could be made based on the computational resources required for the algorithm to compute the output. The ideal scenario would have been a grid size with consecutive values 1mm. For the first trial, the data was processed such that the values were rounded off to the nearest 1mm value and was tested on the created simple PSM environment (later, it was found to be computationally expensive and the value was increased to 5mm).

As mentioned in the introduction section, the dVRK PSM's are being controlled using the CISST-SAW system architecture [10]. The PSM can be controlled using the dVRK-ROS Python client by providing the inputs as individual joint positions or robot end-effector

position. For controlling the PSM in joint space control, 7 values would be required to define the position of each joint whereas in cartesian control only 6 values would be required to uniquely define the robot tip position. Since RL algorithms can end up becoming computationally expensive, a minimalist representation of the robot would be preferred because of which cartesian space was chosen to define the robot state space.

A 3D grid world with a total of 6 values representing the state of the agent was implemented. The PSM was controlled by providing the end-effector positions and orientations. With 3 states representing the X, Y, Z position values and the other 3 states representing the orientation of the PSM end-effector (α, β, γ). Due to the large number of state spaces resulting from the different combinations of states and actions, it was computationally impractical to get the outputs with the resources available. It would require special super computers to compute such high dimensional data [38]. It was concluded that the problem being considered was too complex to be solved at one shot, and required simplifying the problem further more. The main issue was the large number of parameters defining a state of the robot. From the collected user data, it was found that the needle could be held in the same orientation during hand-off task movement and then the PSM arm could be reoriented after reaching the desired position. The above assumption made it possible to separate the position and orientation parameters into two different problems. The RL algorithm finds the path to follow for reaching the desired position and the PSM arm orientation could be set based on the user controlled PSM arm's orientation. By doing this, the number of combinations of the states was reduced significantly and was converted into a simpler problem.

The next part to decide was how to provide the agent with rewards. Since a RL algorithm is driven using a reward function, to make the agent follow a specific trajectory a custom reward function would need to be designed [76]. The trajectory followed by an user can vary and it would be very tedious to manually create a custom reward function for each

individual. In early 2000's Ng *et. al.* introduced the concept of IRL. After the initial introduction of the concept of apprenticeship learning, several researchers worked on IRL and developed different IRL algorithm. Based on the literature review, one of the best suited IRL algorithms for finding a custom reward function was found to be Maximum Entropy IRL. The next step was to find an optimal policy. Initially, value iteration technique was used to solve the discrete RL problem. Maximum Entropy IRL utilized the collected user data as expert demonstrations to calculate the expected state visitation frequency of the user. Since, the problem is being considered as a 3D discrete grid world problem, the next parameter to set was the size of the grid. Again to simplify the problem, a grid size of 3 was selected to start testing the algorithm (state space = action space = 27). The total number of actions can be found by the formula a^s, where 'a' represents the total number of actions at each state and 's' represents the total number of parameters defining a state of the model [80]. Even though this basic implementation worked quite well for a grid size of 3, when the grid size was increased to a size 11 (state space = 1331, action space = 27), the agent was unable to reach the desired terminal position. The agent ended up shifting between two consecutive states to claim the reward [4]. Even after several trials of reward shaping and varying parameters, the agent was unable to learn a trajectory. Therefore, it was found that value iteration lacked exploration and was not suitable for solving this problem [4].

One of the solutions found to resolve this issue was to use model-free RL techniques. It uses function approximation techniques to find the Q values of a state. The algorithm runs the MDP for a number of episodes randomly and calculates the quality of each state. Over a large number of episodes, the algorithm will converge to the optimal solution. Q-learning algorithm was selected because Monte Carlo learning only updated the values at the end of an episode, which can cause drop in performance (agent might keep following a completely wrong path until the episode ends). The code was written in Python and

tested for a grid size of 3. The solution converged, but an increase in grid size caused the algorithm to consume a lot of time to give an output. After some research related to optimizing the code and modifying the code to run in parallel, the performance improved and the agent finally was successful at learning the operator's trajectory.

The following sections describe the key elements of discrete RL model considered in this work.

State Space

The agent is the PSM and the environment is everything viewed by the endoscope camera of dVRK. The PSM end-effector position can be completely defined by providing the position and orientation of the end-effector with respect to the base frame. The position values was defined using cartesian coordinates. The PSM end-effector position was defined using X, Y, Z positions with respect to base frame and orientation values were defined using roll, pitch, yaw rotations (α, β, γ). It was a minimalistic way of representing the PSM tip position to reduce the computational load on the computer system.
The state space was set to a subset of the PSM arm's workspace. A lower and upper limit was defined for the PSM end-effector position. It is represented as a 3D vector with all the three values representing cartesian position. Therefore, the agent's state could be defined to be the vector $s = [x, y, z]$ and the state space was defined as $S = \{s \quad | \quad x \in (-30mm, 20mm), y \in (25mm, 75mm), z \in (-140mm, -90mm)\}$.

Action Space

The main aim of RL is to select the best action from a state in the environment. In this model, actions is defined as the change in PSM end-effector X, Y, Z positions. For a discrete model, the action space is also a discrete set of values. The action values were

selected appropriately to reduce jerk and ensure a smooth transition from a state to another state. The action space for the robot was designed to ensure all possible configurations out of a total of $[-ve, 0, +ve]^3 = 27$ actions from the current state, where 0 indicates no change in position value, $-ve$ and $+ve$ indicate a unit decrease and unit increase in the position value respectively. The action space was defined as A = {[δq_1 ,δq_2, δq_1] | δq_1 $\in \{-5mm, 0mm, 5mm\}$,$\delta q_2 \in \{-5mm, 0mm, 5mm\}$,$\delta q_3 \in \{-5mm, 0mm, 5mm\}\}$.

Transition

The discrete model was considered as a deterministic problem because once the PSM arm is commanded to reach a position (S') from an initial state (S) after taking an action (A), the PSM arm moves to the commanded position (S') without any uncertainty, i.e. S' = S + A. In other words, these are environments where each state-action pair would result in a unique resulting state. Since it's a deterministic system, the transition probability is equal to one.

Rewards

For the initial trial using Maximum Entropy IRL with value iteration, the reward function created was a set of weights for the complete state space which represented the rewards received at each state. As the final solution to the discrete RL problem, sparse rewards was used with Q-learning. The algorithm designed to generate an appropriate reward function is shown in algorithm 8. The data collected from the dVRK was discretized into an array of points $P_i(x, y, z)$ in the 3D grid world. Figure 4.2 shows the computed reward function using Maximum Entropy IRL (left) and Sparse Rewards (right). The size and color of the dots indicate the rewards at their corresponding states. In the plot generated by Sparse rewards, equal rewards given to the user visited states ($normalizedthreshold < 0.1$) and higher reward at the goal state can be seen ($normalizedthreshold > 0.1$). This helped

the Q-learning algorithm to track the visited states before reaching the goal state. In comparison, no such relation could be deduced in the Maximum Entropy IRL plot, which made it harder for the value iteration to track the desired trajectory.

Algorithm 8: Sparse Reward Generation

Result: Sparse rewards generated along the trajectory drawn by the operator

$Reward_{goal}$ much greater than $Reward_{visited}$

$Reward_{goal} \gg Reward_{visited} > Reward_{other}$

Input: $StateToIndex(array(x, y, z))$

Output: $Rewards[gridsize^3]$

while r_{id} *in* $Rewards$ **do**

  if r_{id} == $goal$ **then**

    $Rewards[r_{id}] = Reward_{goal}$

  else if r_{id} == $visited$ **then**

    $Rewards[r_{id}] = Reward_{visited}$

  else

    $Rewards[r_{id}] = Reward_{other}$

  end

end

Policy

Once the algorithm finds the optimum policy, it can be used to control the PSM. The action which provides the highest reward would be selected from each state the agent visits. When the best action is selected at each state, it is repeated until the goal state is reached. As a result, the robot follows the desired trajectory. The trajectory can be represented as, $Q = \{q = [q_1, q_2, q_3] \quad | \quad q_1 \in (-30mm, 20mm), q_2 \in (25mm, 75mm), q_3 \in (-140mm, -90mm)\}$.

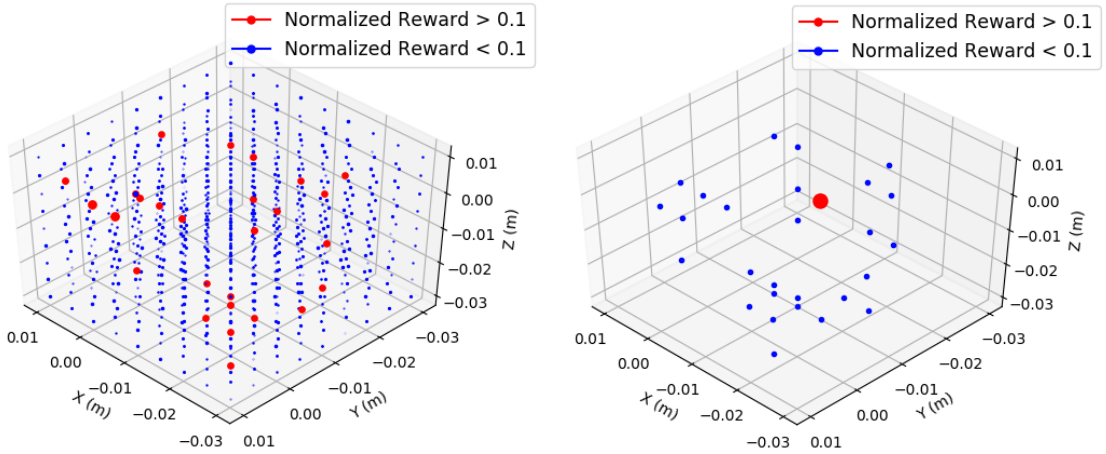

Figure 4.2: Comparison of reward functions obtained for an user using IRL with value iteration (left) and sparse reward with Q-learning (right). The size and color of the dots indicate the rewards received by the PSM for visiting the corresponding states in the grid world.

Episode Termination Conditions

Upon the completion of each iteration, the agent's expected value of a state was evaluated and stored to be used to calculate the expected reward. The following conditions warranted termination:

- To ensure the PSM doesn't keep moving around in the environment indefinitely, a maximum number of steps was assigned to terminate an episode (set to 100)
- During learning phase it is possible that the PSM can get stuck at a position. Therefore, the maximum amount of time allotted to the PSM to complete an episode is set to 30 minutes.
- Task completion (when the PSM reaches the goal state)

Realigning PSM_A Jaw

After gathering the suture data, the user indicated a preference for the PSM_A's tip to approach the PSM_M's tip orthogonally, and have the normal vector of PSM_A's jaw be parallel to the approach vector of PSM_M's jaw to help make the hand-off task easier [Figure 4.3].

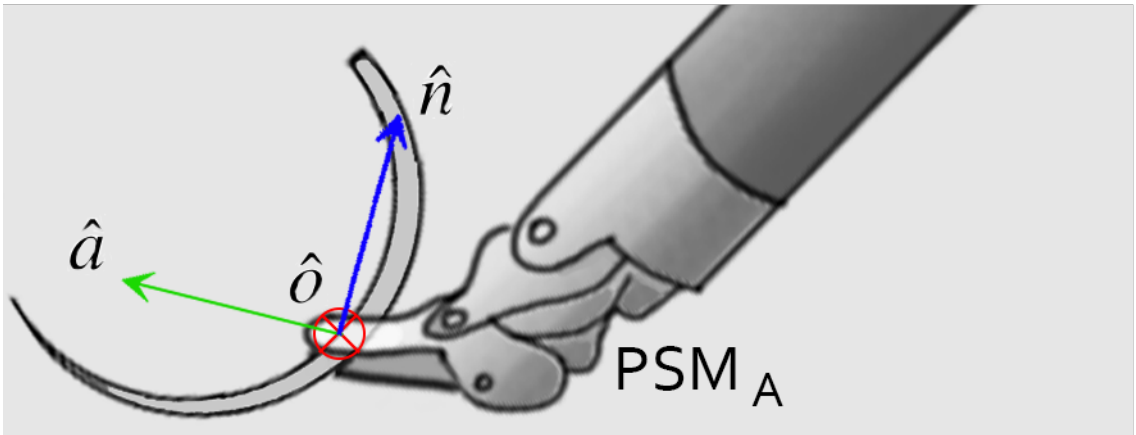

Figure 4.3: Transformations between World, PSM_M, PSM_A and Needle frames.

While researching about RL techniques which could be used to scale discrete RL model to a larger size, it was found that for models having large state spaces deep Q-learning would be better suited technique [52]. Deep Q-learning (DQN) uses deep learning to approximate the function. A set of weights is optimized using Q value found from Bellman equation. It uses back propagation to adjust the weights and takes sample from a set of transitions stored as replay memory. Replay memory is used to store a set of transitions from state S to S' when an action A is taken. The deep learning algorithm takes random samples from replay memory to avoid overfitting of the data. But, due to the large size of action space for this problem, DQN was not the ideal solution to the problem and other

techniques needed to be explored [47].

For robotic applications involving large action spaces, continuous action space system was the most suitable approach. The continuous action space RL technique which was best suited for robotics was Deep Deterministic Policy Gradient (DDPG) [47]. Few years ago, OpenAI created an open-source RL module called OpenAI Baselines, which had python implementations of state of the art RL algorithms [15]. Even though it helped accelerate work in the RL community, it lacked documentation and was never updated regularly. Therefore, users found it difficult to resolve the bugs in the code and implementing it on a custom environment became extremely complex. A year later, a fork of OpenAI Baseline was introduced called Stable-Baselines [30]. It solved many of the problems by providing documentation, frequent updation and assistance for bugs in the code. It became one of the best RL learning module available online because of the explanations provided for different algorithms and had a modular code which could be further edited by the user. Therefore, Stable-Baselines was used in this work for training the PSM arm. After finalizing upon an established and resourceful open-source RL module, the next problem to consider was the requirement of continuous action space systems to run large number of trials to learn a policy. It would be impractical to run millions of iterations on a real robot, therefore simulation environments needs to be used to train the model. The trained RL model could be used to predict the output for each state and control the real robot. While working on this thesis, the most commonly available simulators included Gazebo, V-REP etc. As mentioned in the introduction section, it would be beneficial to have a simulator which accounts for the dynamics of system and helps train in real-time. The real-time indicates the computation and consideration of dynamics of the robot, while training and testing the robot in simulation. All the simulations mentioned above are mainly meant for general purpose applications and not meant for real-time dynamic computations or medical robots. The only real time open-source dynamic simulator available was AMBF. For proceeding

ahead, a Python wrapper needed to be developed which would act as an interface between AMBF and the RL algorithm.

4.0.3 Asynchronous Multi-Body Framework (AMBF)

This section explains the important features present in AMBF which could be useful for training RL models. Recent research by Wang *et. al.* presented a technique to accurately calculate the dynamics of the dVRK system using convex optimization [86]. Therefore, the problem of finding accurate dynamics model of the dVRK PSM is solved because their work has been included in AMBF by the creator to provide an accurate dynamic model. Before, the PSM YAML file is loaded into AMBF, the proportional–integral–derivative (PID) gain values of individual arms needed to be tuned before it could be used for training (to reduce the error between commanded and reached joint positions).

Few of the important terms that needs to be understood which are related to learning models have been explained in this section. The Wall Time denotes the System Clock (Wall Clock) in seconds, Simulation Time denotes the time of the physics simulation in seconds (for real-time, wall time is equal to simulation time) and Step throttling is a Boolean that can be used to enable/disable physics simulation throttling. Amongst the arguments that can be passed while launching AMBF, the following factors could be used to help give control over AMBF functioning while interfacing with learning models:

- **Dynamic Loop Frequency:** It helps control the frequency of underlying physics solver loop.
- **Simulation Speed Factor:** It can be used to change the simulation speed (by default it is equal to 1).

- **Fixed Time Step:** It makes the simulation wait for a given time before the physics simulation computes the next step.

The dynamic loop frequency can be used to set the number of smaller iterations the simulation computes for a given unit time step. For example: When the dynamic loop frequency is set to 1000Hz. It breaks down 1 second into 1000 small parts and the physics solver computes the output 1000 times in 1 second.

In AMBF, the wall time is independent of the simulation time. For real-time applications, depending on the value of dynamic loop frequency set by the user, AMBF breaks each second into that many smaller time iterations. The simulation time denotes the time being followed by the physics simulation which may or may not be equal to wall time. Simulation time will be equal to wall time unless there is a change in simulation speed factor from the default value or if the Fixed Time Step is chosen such that the simulation runs in non real-time. The simulation speed factor is responsible for making AMBF compute it's iterations faster. For example: If the simulation speed factor is set to 2, then the simulation computes all the calculations at twice the normal speed. The following example provides a more detailed example regarding the actual computations that differ.

For example: If the arguments dynamic loop frequency and simulation speed factor is set to 1000 and 2 respectively. The above arguments denote that the user wants AMBF simulation time step to be twice the unit wall time step. Therefore, while computing the physics, a unit simulation time step would be divided into 1000 smaller iterations and for half an unit time step in wall time, technically the number of iterations computed would be 1000.

Now, the Step throttling is a special feature added in AMBF to run the simulation based on the rate of messages transmitted by the learning algorithm. If the Step throttling Boolean is set to True, then AMBF will remain in throttling mode until an input is received from the user's code to run the next iteration. During throttling, the simulation performs the

calculations on a parallel thread and continues normal execution once an input is received. It could be seen as a way to make AMBF adapt to the learning algorithms data transmission speed.

The Fixed Time Step can be seen as a buffer which makes AMBF wait for a fixed time step before the physics simulation computes the next step. While training it's better to set it to one for speeding up the computation (1 second) and during evaluation it can be set back to the default value.

Depending on the application, the Step throttling can be set to True or False . When it is set to False, AMBF and user's code will run independently of each other. As a result, both of the applications are not dependent on each other to proceed to next iteration. If the simulation speed factor is being changed from the default value, Step throttling should be set to False. Otherwise, the simulation will not compute the calculations faster and make the effect of simulation speed factor to be null because AMBF will give control over to user's code for starting next iterations. The following example can help explain the implementation.

For example: If the arguments dynamic loop frequency, fixed time step and simulation speed factor is set to 100, 1 and 5 respectively and Step throttling is disabled. This would make AMBF wait for one second to solve next physis loop and then compute 100 smaller iterations for a unit simulation time step and would repeat this 5 times before a unit time step is increased in wall time.

Another important aspect required in RL is to be able to run multiple instances of the environment simultaneously (to enable training and evaluation in parallel). This feature is also provided to the user by enabling them to run multiple instances of AMBF. Multiple instances of AMBF could be created by running multiple roscore (it is a pre-requisite for any ROS-based system and consists of a collection of nodes and programs). Multiple roscore's could be run by specifying the Port Number along with roscore using the flag

'p' (example: roscore -p 11315). One thing that would need to be ensured is that AMBF is running the same instance of roscore which could be achieved by sourcing the ROS environment variables (such as ROS_MASTER_URI). Now, to avoid any confusion regarding which instance of AMBF the user's code is accessing, AMBF provides the user with an option to provide a custom namespace to each instance of AMBF using the flag 'ns' (example: –ns /test/) . The above steps could help user run multiple training at the same moment to help utilize time better. Multiple instances of AMBF might not run smoothly on a low configuration computer system. For such scenarios, AMBF could be run headless to reduce the graphics requirements.

4.0.4 Reinforcement Learning Model: Continuous Space

The AMBF Python client can be used to control the robot, but few modifications is required to interface AMBF with the RL learning modules. One of them included upgrading the AMBF Python client to Python 3 because the creator had written the AMBF Python Client for Python 2.7. It was required because as of January 1^{st} 2020, Python 2 was deprecated and the standard RL modules required the AMBF client to be Python 3 compatible. The other requirement was to create a wrapper to act as a bridge between the RL algorithm and the spawned dVRK system in AMBF simulator. Finally, a compatible gym based environment for the dVRK system needed to be developed.

As part of this work, after a few modifications and changes, the AMBF Python Client was upgraded to Python 3, a wrapper was created to interface the learning module and dVRK PSM arm spawned in AMBF. Also, a gym compatible environment was developed for dVRK PSM.

This section provides more details about the created PSM environment for AMBF. In continuous action space systems, the state and action space are defined between a lower and upper limit value. Other major parameters that need to be set for robot safety is joint

limits and workspace limit values. The joint limit values could be set based on the actual dVRK PSM joint limits or it could set to a user defined value. The workspace limit values can be set by the user depending on the size of workspace required for the application. The robot can be controlled through joint position control, cartesian position control or torque control. Torque control is not ideal for this application because during training as part of exploration the algorithm outputs random values. Therefore, a set of random joint torque values can make the robot become unstable and cause the robot to go haywire (because of robot dynamics). Therefore, the best options for controlling the PSM is using joint or cartesian space control.

Figure 4.4 shows the workflow followed to train a model using the PSM environment. Depending on the RL technique selected, the PSM environment would need to be loaded. It is because algorithms such as HER require a goal based gym environment (observation space is a dictionary), whereas DDPG uses the default gym environment structure (observation space is an array). Different modules need to be imported such as the base algorithm, type of policy, noise etc. because algorithms like HER use DDPG as the base algorithm. Stable-Baselines also allows the users to define various model parameters like the actor learning rate, number of training steps, number of exploration steps etc. Similar to common gym environments available, the environment needs to be built using the make command. For this case, building the environment creates an instance of AMBF Python client which gives the algorithm access to control the bodies spawned in the simulation. Before starting training, the model needs to be reset to ensure all the variables are initialized and a sample goal position is generated. The next step would be to start training the model. It is possible to either load a previously trained model and continue training or start training a new model from scratch. Finally, the model is saved for evaluation and testing the efficiency of the output.

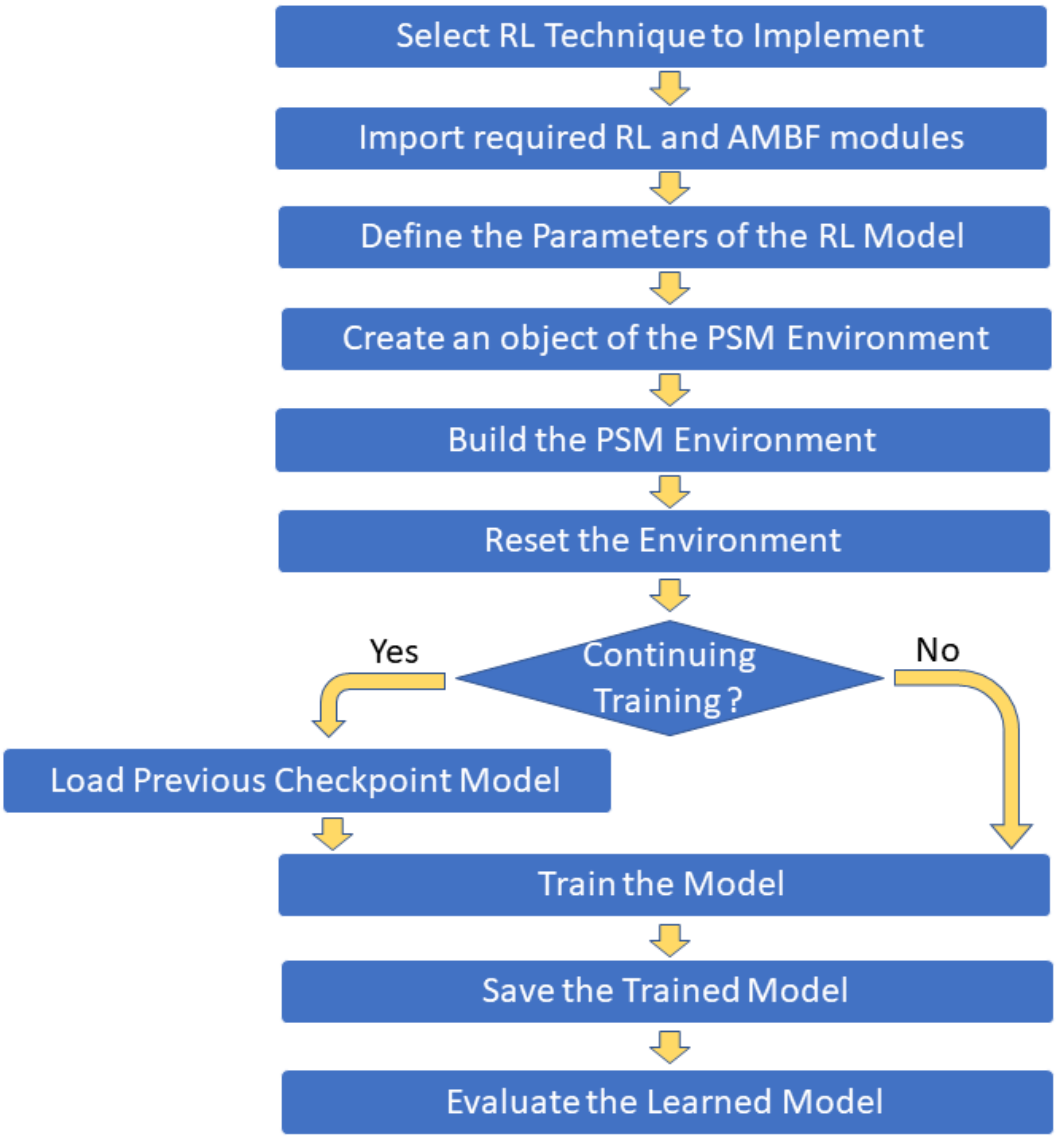

Figure 4.4: Workflow for creating and training Model for the PSM Environment

For solving the exploration inefficiency of continuous space RL techniques, a module was created to convert the recorded dVRK ROS data into input for RL algorithm. The user demonstrations could be used to train algorithms such as GAIL (subsection 7.0.1). The ROS data is converted into a Python dictionary which consists of observations, actions, rewards, episode returns and episode starts. The difference between consecutive

Algorithm 9: Function Definitions for AMBF PSM Environment

function `Init(`*action space limit, joints to control, goal position range, position error threshold, goal error margin, joint limits, workspace limits, enable step throttling*`)`**:**
- Create an instance of AMBF Python client
- Create an object handle of the robot
- Define the sction and state space for the problem
- Set the desired joints to control
- Initialize all the required variables

function `Reset()`**:**
- Reset the robot to home position (preset by user)
- Generate a goal position using random sampling
- Return the current position of the robot
- **return** Observation

function `Step(`*action*`)`**:**
- Clip action values within defined Limits
- Get current joint positions and velocity
- Compute forward kinematics of the robot
- Apply the action in cartesian space
- Check if resulting state is within set cartesian limits
- Compute inverse kinematics
- Check if computed joint positions are within set joint limits
- Set the robot to commanded joint positions
- Update all observations
- Compute the reward received for resulting state
- **return** Observation, Reward, Done, Info

function `Reward(`*achievedgoal, desiredgoal, info*`)`**:**
- Compute distance between desired goal and currently achieved goal
- **if** *Sparse Reward* **then**
  - **if** *Computed Distance* < *Goal Error Margin* **then**
    - Reward = 1
  - **else**
    - Reward = -1
  - **end**
- **end**
- **if** *Continuous Reward* **then**
  - Reward = 1 - $\frac{ComputedDistance \times 0.5}{MaximumDistancewithinWorkspace}$
- **end**
- **return** Reward

observations and actions can be set by the user depending on the application. It could help speed up training and help train the PSM arm for complex tasks such as picking up a suturing needle etc. [31, 57].

The two continuous space RL techniques selected to explore was DDPG and HER with DDPG. For implementing DDPG, it required the environment to follow a Gym environment structure and needed the environment to inherit the main Gym environment. But for HER with DDPG, the structure of the environment had to be modified because it needs the environment to follow the Gym goal environment structure. The observation structure for goal based Gym environment is:

- **Observation**: It stores the current observations of the state.
- **Achieved Goal**: It stores the current result by the agent.
- **Desired Goal**: It stores the final goal the agent is supposed to achieve.

Gym Fetch environment was used as an example for creating the PSM environment. The best way to achieve good results in RL is to simplify the complex tasks into simple tasks and later combine different trained model for to complete the whole task. As mentioned in previous sections, since focus is on the hand-off task, the model was trained to reach a desired goal position with some error margin. The PSM environment is made up of many functions which are interdependent on each other and algorithm 9 describes the steps occurring in the four major functions: init, reset, step and reward.

- Init

  The basic elements of the model such as action and state space is defined inside this function. For ensuring the safety of the robot, joint and work space limit needs to be set for the robot. The action space can be defined based on the maximum value by which the end-effector position can change without producing a jerk. The goal

is generated by adding a random number to the each of the end-effector positions and the goal position range defines the maximum range of these random values. During the training process, the algorithm outputs the action values at a very high rate because of which an error can occur between the commanded and reached joint positions of the robot. The goal error margin is the maximum error that is allowed to occur between the commanded and reached positions to continue to the next iteration. It also creates instances of all the class objects required for AMBF to function.

- Reset

  The reset function sets the robot back to the home position defined by the user. It also samples a random goal for the upcoming episode. The user could select amongst the three options for choosing the initial state of the robot (based on the requirements):

  - Home position of the robot
  - Use previous achieved state as the new initial state
  - Random position within the workspace limits

- Step

  Based on the current observations, an action is predicted using the current learnt policy. The predicted action value is kept between the action space limit values to limit the maximum change in end-effector position. The resulting joint positions and end-effector positions is computed using the Forward and Inverse Kinematic function provided for the dVRK PSM. A clipping function then ensures the computed joint values are within the joint limits of the PSM arm. A separate function was used to ensure the robot reaches the commanded joint positions within a permitted error value. It returns the update positions of the robot along with the reward for taking that particular action and some debugging information.

- Reward

  Based on the distance between the current position and the desired position of the end-effector, the agent receives a reward value. In case of sparse reward, the agent only receives a positive reward once it reaches the goal position. But in the case of continuous rewards, the agent receives a steady set of rewards based on the distance from goal position. The closer the agent is to goal, the higher is the reward received by the agent.

The render function was not required for this work because AMBF was running in parallel and the rendering would occur in real-time through AMBF. The robot can be controlled using joint or cartesian control. If joint control is implemented, the environment uses only the forward kinematics function, whereas cartesian control uses forward and inverse kinematics functions. Currently, the environment's inbuilt parameters that can be observed include position and orientation of the end-effector (in total 6 values, roll-pitch-yaw representation [78]), joint velocities and joint positions. Later, support for more entities such as joint acceleration, end-effector velocity etc. can be provided or if required the user could compute and add any other other observations. An important point to keep in mind was to ensure the spawned robot is able to reach the commanded positions before a new command is received. It is achieved by running a loop which provides the robot some time to reach the commanded value with some margin for error (in this work it is set to 10mm and 7.5mm). During this work, the inverse kinematics function utilized functions from the module PyKDL. Therefore, to use cartesian space control, PyKDL would need to be installed for Python 3. The installation of PyKDL has been known to cause lot of errors and as a result there is work going on currently to create a stand alone module for AMBF which includes those functions. Regardless, until then the steps required to install PyKDL for Python 3 is summarized in the AMBF wiki page under the RL section. A maximum number of iterations to run is set by the user. Along with the total steps for a

model, Stable-Baselines allows users to provide specific values for rollout, training and evaluation. During the rollout phase, the agent keeps exploring and filling up the replay buffer with state transition parameters. Once the maximum number of rollout steps is surpassed, the model starts to train by randomly sampling from the replay buffer. Then, it is always a good idea to use an evaluation environment to test the learnt policies at a regular interval. It can help make sure the model is performing as expected and the progress can be monitored in a systematic manner (algorithm 10). The wrapper was successfully created and tested using DDPG and HER with DDPG. The progress during training was logged using TensorBoard.

Algorithm 10: Steps for Trained PSM Model

Result: Trained PSM Model
Select the type of Action and Parameter Noise
Define the Model Parameters
Reset the Environment
while $steps \leq$ *total steps* **do**
  Rollout
  for $number\ of\ rollout\ steps$ **do**
    Predict action based on current policy
    Call Step Function
    Store the Transition in Replay Buffer
    Increment Number of Steps
    if $reached\ goal\ state$ **then**
      Update Rollout Episode Parameters
      Increment Rollout Episode Number
      Reset the Environment
    end
  end
  Training
  for $number\ of\ training\ steps$ **do**
    Sample from Replay Buffer
    Train the underlying A2C Models
    Update Actor and Critic Losses
    Update Target Network
  end
  Evaluation
  for $number\ of\ evaluation\ steps$ **do**
    Predict Action based on Learnt policy
    Call Step Function
    if $reached\ goal\ state$ **then**
      Update Evaluation Episode Parameters
      Increment Evaluation Episode Number
      Reset the Environment
    end
  end
  Log all the Model Parameters using TensorBoard
  Save the Model
end

Chapter 5

Results

5.0.1 Discrete Space RL Problem Results

The discrete space RL results present in this section is written as a research paper titled "Collaborative Suturing: A Reinforcement Learning Approach to Automate Hand-off Task in Suturing for Surgical Robots " and is submitted for review to the 29th IEEE International Conference on Robot and Human Interactive Communication, RO-MAN 2020.

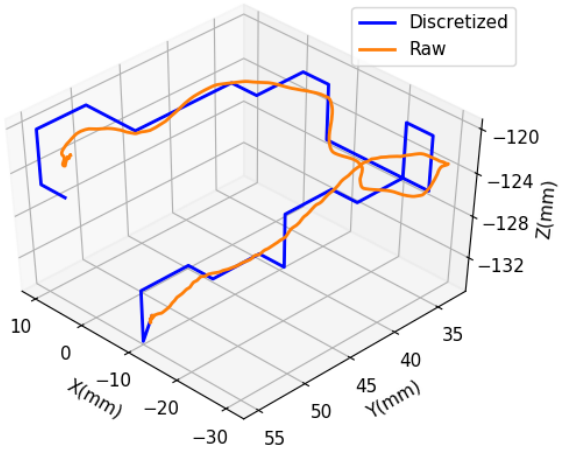

Figure 5.1: Raw and discretized input trajectory for one of the user's data

For finding the ideal discretization factor, the factors considered was the collected users

data, required computational resources and the largest step the PSM could move without resulting in jerky movements. The grid size was set to $11 \times 11 \times 11$ and all the values were rounded off to nearest 5mm value because the data loss observed at this set of values was minimal (without compromising the performance of the algorithm). Figure 5.1 shows that the discretized trajectory manages to retain the features of the desired raw input trajectory of the user.

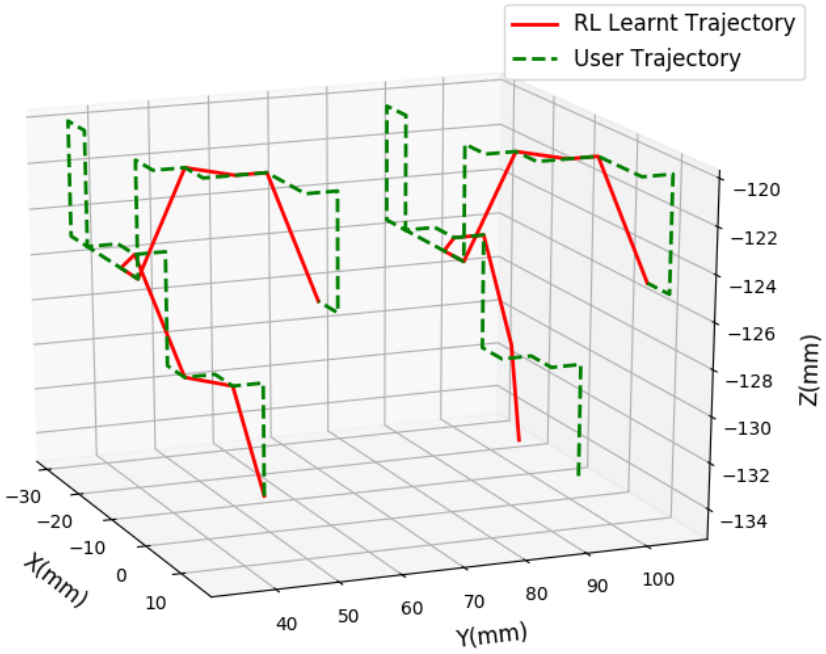

Figure 5.2: Application of learnt RL policy for successive A_1 and A_2 tasks

Given the required suture points are computed beforehand and provided to the algorithm, the same model can be used to perform repeated suturing hand-off task (A_1, A_2) by aligning and stacking the grid world with respect to the goal position as shown in Figure 5.2. This is possible since the agent has learnt the policy to follow the user trajectory, even if there is a change in initial position. Since the exploration factor in Q-learning is dependent on the random numbers generated by the system, different iterations of the algorithm generated a slightly new policy. Each policy differs in state-action pairs, but as it can be seen from Figure 5.3, the agent was still able to successfully learn the user's

trajectory.

Mean			Standard Deviation		
X	Y	Z	X	Y	Z
2.857	1.488	0.774	3.388	2.286	1.808

Table 5.1: The mean and standard deviation of the dissimilarity between the average of the three learnt trajectories [Figure 5.3] and a single user trajectory (all values are in mm)

To show the effectiveness of the technique presented, the similarity of the generated trajectory and the user defined trajectory was measured using DTW [70]. To establish a metric, Manhattan distance was used as the distance measure. DTW is a common method to compare the similarity between two sets of data which vary in length or time. Table 5.1 shows the mean [2.857mm, 1.488mm, 0.774mm] and standard deviation [3.388mm, 2.286mm, 1.808mm] of three learnt policies shown in Figure 5.3 from the user defined trajectory. For reference, perfectly similar trajectories would have a 0 mean and 0 deviation from the reference trajectory. Figure 5.3 also shows the deviations of the learnt policies from the user trajectory, where the black dashed lines symbolize the DTW produced mapping from the policy to the user trajectory. From the Figure 5.3 it can be seen that the learnt trajectory was not able to visit each point of the user trajectory. The RL learnt trajectory circumvented few of the loops present in the user trajectory which resulted in DTW mapping many points in user trajectory to the same point in learnt trajectory. Root-Mean-Square Error (RMSE) was used to evaluate the *goodness* of the generated trajectories. The RMSE was found to be [0.0044mm, 0.0027mm, 0.0020mm] for the three trajectories shown in Figure 5.3 with reference to the user trajectory.

Figure 5.4 shows the agent following the user's trajectory with a deviated initial state. The user's initial position was [-10mm, 55mm, -135mm] and the value of different initial state tried for the learnt agent ranged between [-25mm, -5mm] along the X-axis, [50mm,

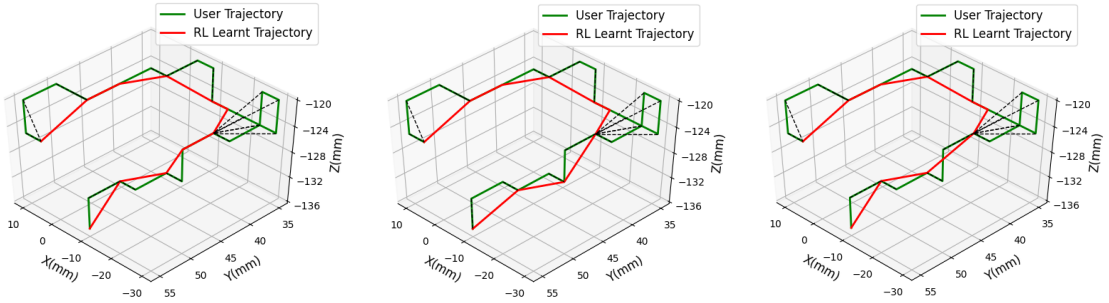

Figure 5.3: Learnt RL and discretized user trajectory for 3 different computed policies with DTW to measure similarity

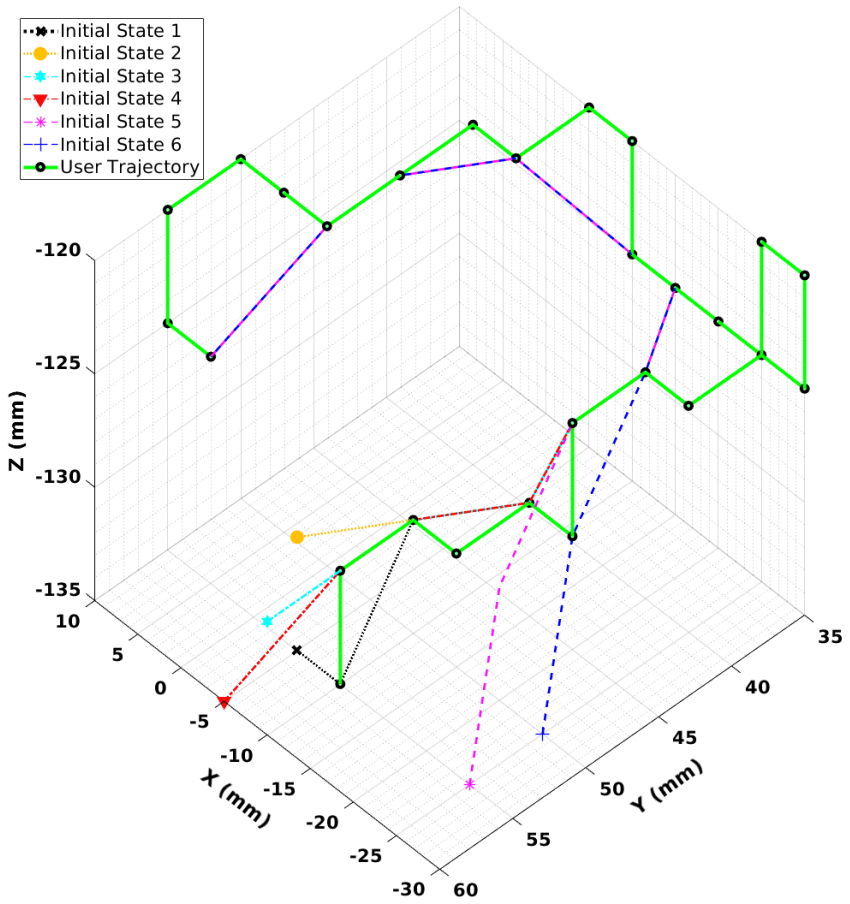

Figure 5.4: Trajectory followed by PSM after change in the initial state based on the learnt RL policy

60mm] along the Y-axis and [-130mm, -135mm] along the Z-axis. As shown in the figure, the agent was successful at learning the user's trajectory even after significant deviations

in initial state of the agent.

5.0.2 Continuous Space RL Problem Results

The training of RL policies was done on a system which had a Ryzen 7 3700X processor and NVIDIA GeForce RTX 2070 SUPER graphics card. The outputs obtained from DDPG and HER with DDPG depend on the user defined error margin set for reaching the goal position and workspace limits.

As proof of concept, a total of 3 models were trained:

- **Model 1:** DDPG (goal error margin=10mm)
- **Model 2:** HER with DDPG (with goal error margin=10mm)
- **Model 3:** HER with DDPG (with goal error margin=7.5mm)

Figures 5.8, 5.5 and 5.10 depict the success rate of the models when the maximum number of steps allowed in an episode is varied. For Model 1 and 2, the margin of error for reaching the goal was set to 10mm and the workspace limits was set to $S = \{s|x \in (-60mm, 55mm), y \in (-50mm, 60mm), z \in (-200mm, -90mm)\}$. For Model 3 the margin of error for reaching the goal was set to 7.5mm and the workspace limits was set to $S = \{s|x \in (-40mm, 30mm), y \in (-30mm, 40mm), z \in (-200mm, -91mm)\}$. The parameters used for training the models have been detailed in Appendix (subsection 7.0.3). The success rate was calculated by finding the number of times the PSM reached the goal state out of 20 trials. It was trained for a total of 4 million steps and the success rate was calculated for 3 iterations. From the Figure 5.8 it can be seen that once the maximum number of steps is increased over 250 steps, the PSM arm is always able to reach the goal state successfully for Model 2. During the training process the progress was tracked using TensorBoard. The variation in rewards received for each model during training

can be seen in Figures 5.6, 5.9 and 5.11. The rewards received for the models using HER increased significantly after a few thousand steps whereas the rewards can be seen constantly fluctuating for the DDPG model. The steps required for PSM to reach the goal position varied depending on the maximum number of steps allowed per episode (Table 5.2, 5.3, 5.4). For most of the cases, the average number of steps required by the model trained with HER are less compared to the DDPG model.

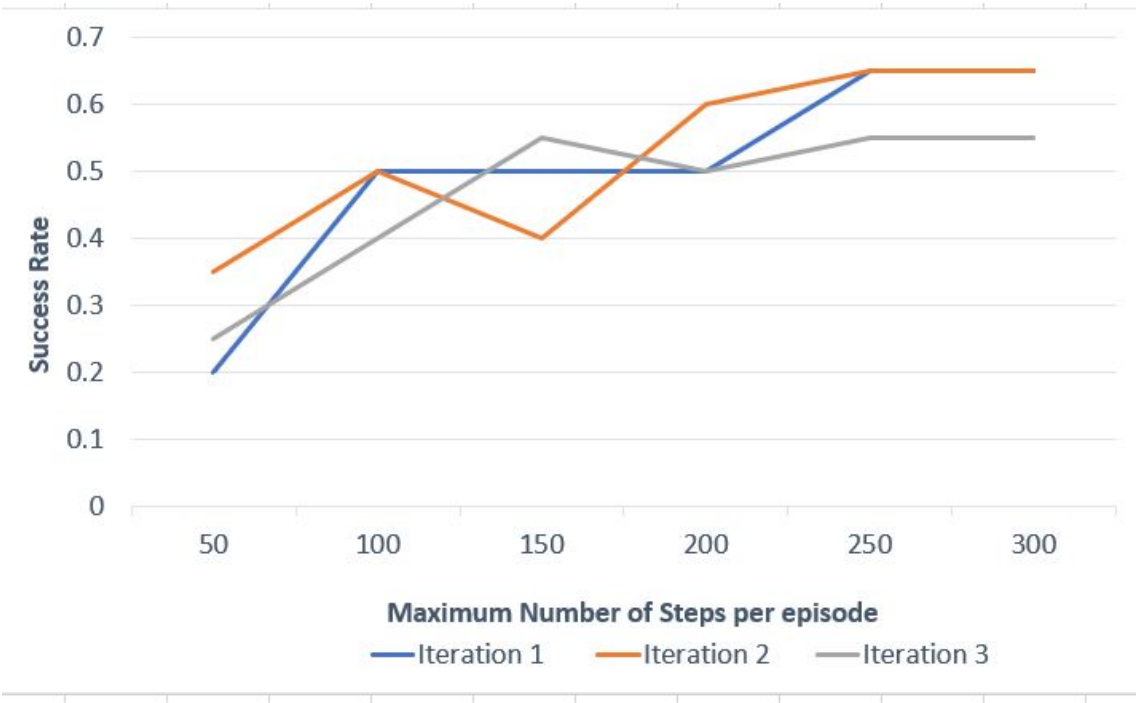

Figure 5.5: Success Rate of 3 Iterations of PSM Reach Task for different Maximum Number of Steps per Episode (Model 1). Each iteration consisted of 20 trials to reach the goal position.

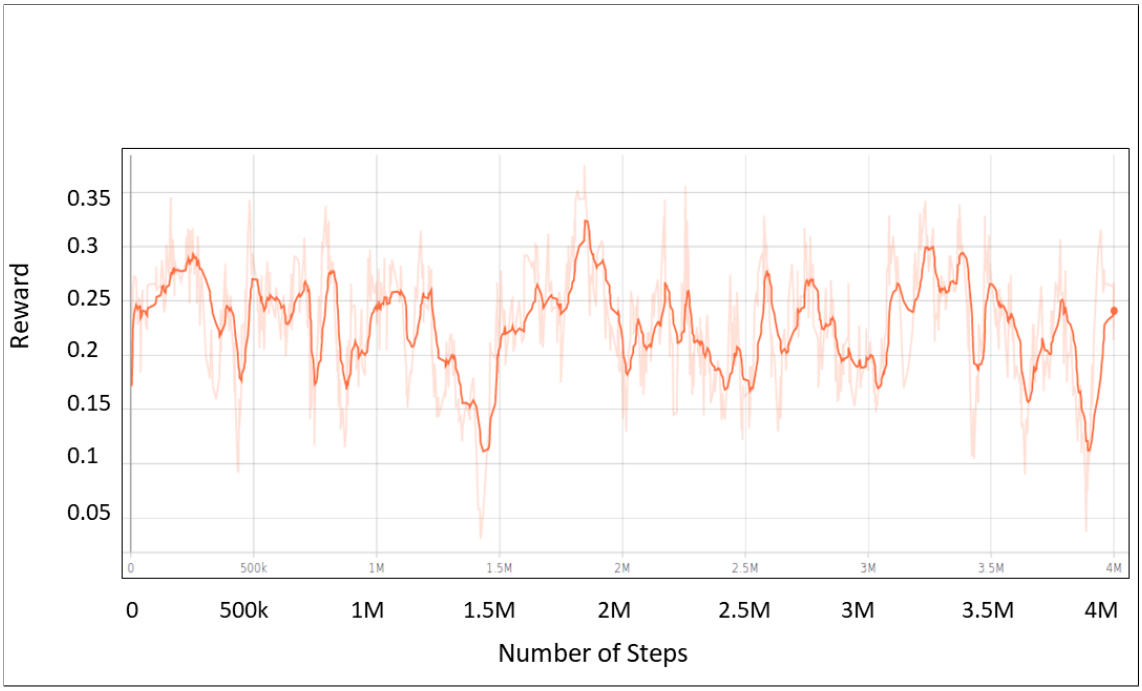

Figure 5.6: Rewards received by PSM over 4 million steps during Model 1 training

Average Number of Steps to Reach Goal Position	**Maximum Number of Steps per Episode**					
	50	**100**	**150**	**200**	**250**	**300**
Iteration 1	23.25	27.8	43.2	72.4	85.4	99.46
Iteration 2	21.14	39	64.75	70.08	100.7	120.38
Iteration 3	14.25	34.75	48.81	62.7	84.57	104.36

Table 5.2: The average number of steps required for PSM to reach the goal position for different maximum number of steps per episode for 3 iterations (Model 1). Each iteration consisted of 20 trials to reach the goal position.

Average Number of Steps to Reach Goal Position	**Maximum Number of Steps per Episode**					
	50	**100**	**150**	**200**	**250**	**300**
Iteration 1	17.62	27.53	43.93	54.4	59.75	48.55
Iteration 2	20.33	19.81	44.41	60	35.65	36.2
Iteration 3	18.23	28.8	38.6	36.7	47.5	29.45

Table 5.3: The average number of steps required for PSM to reach the goal position for different maximum number of steps per episode for 3 iterations (Model 2). Each iteration consisted of 20 trials to reach the goal position.

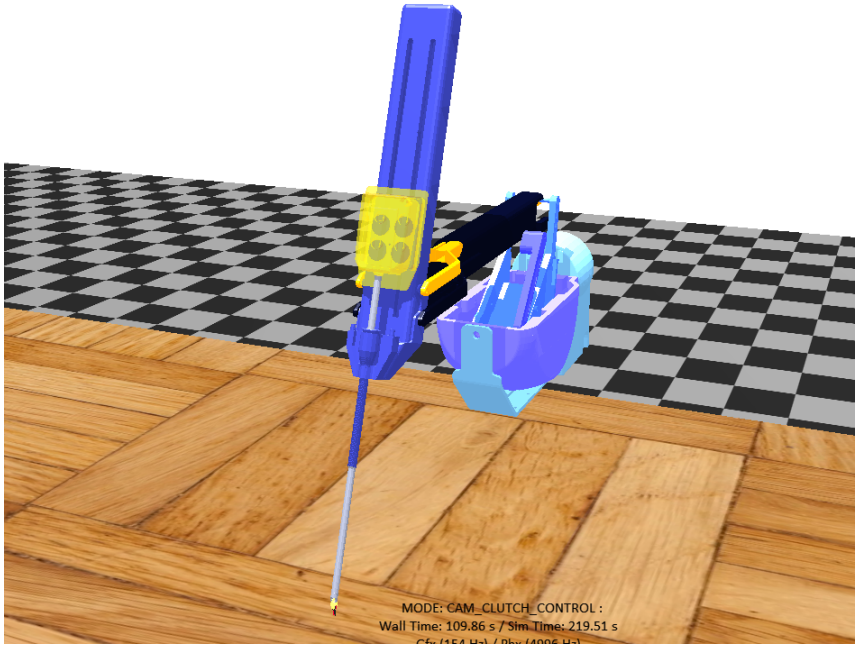

Figure 5.7: dVRK PSM during training in AMBF

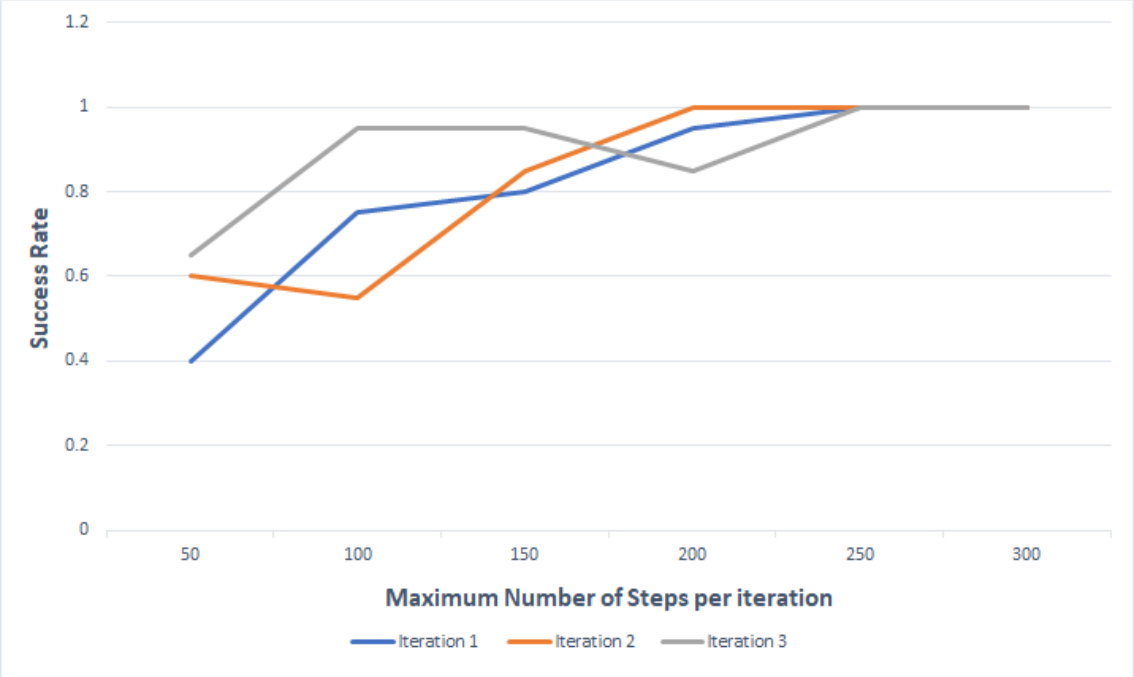

Figure 5.8: Success Rate of 3 Iterations of PSM Reach Task for different Maximum Number of Steps per Episode (Model 2). Each iteration consisted of 20 trials to reach the goal position.

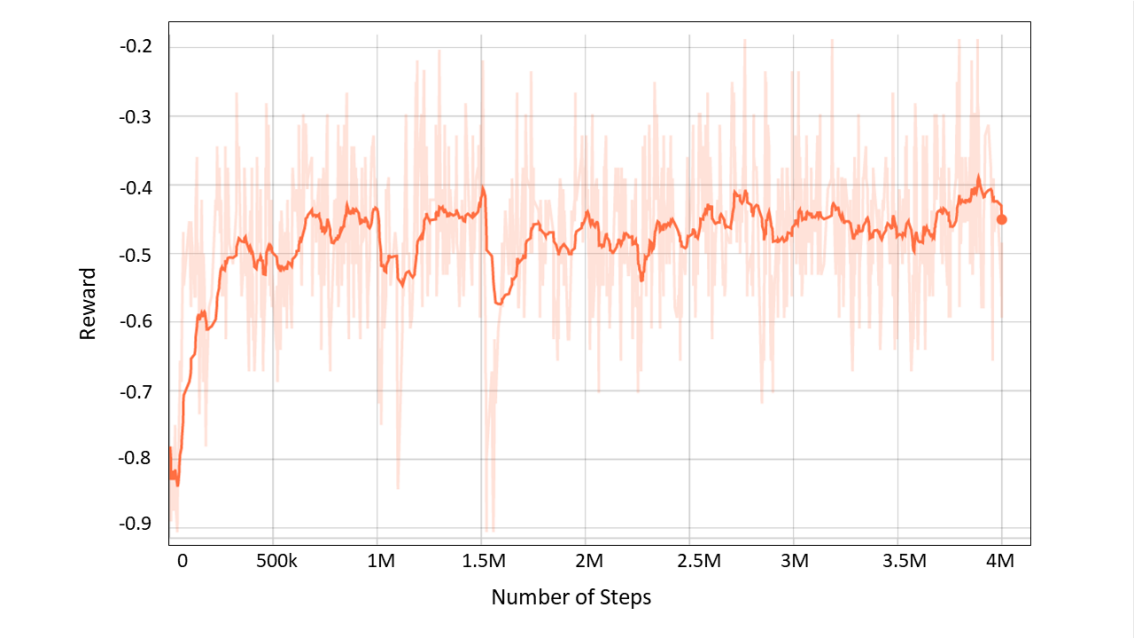

Figure 5.9: Rewards received by PSM over 4 million steps during Model 2 training

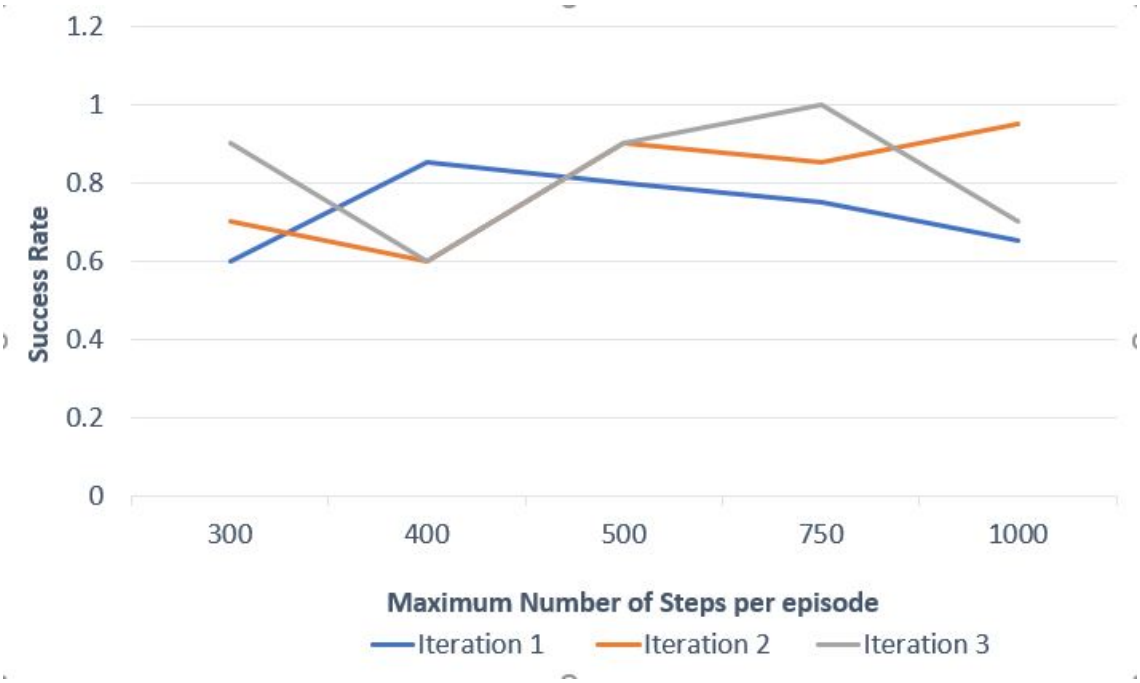

Figure 5.10: Success Rate of 3 Iterations of PSM Reach Task for different Maximum Number of Steps per Episode (Model 3). Each iteration consisted of 20 trials to reach the goal position.

Average Number of Steps to Reach Goal Position	Maximum Number of Steps per Episode				
	300	**400**	**500**	**750**	**1000**
Iteration 1	41.9	33.52	65.31	72	75.4
Iteration 2	43.85	35	35.6	30.29	72.92
Iteration 3	51.33	34.16	43.27	19.05	75.68

Table 5.4: The average number of steps required for PSM to reach the goal position for different maximum number of steps per episode for 3 iterations(Model 3). Each iteration consisted of 20 trials to reach the goal position.

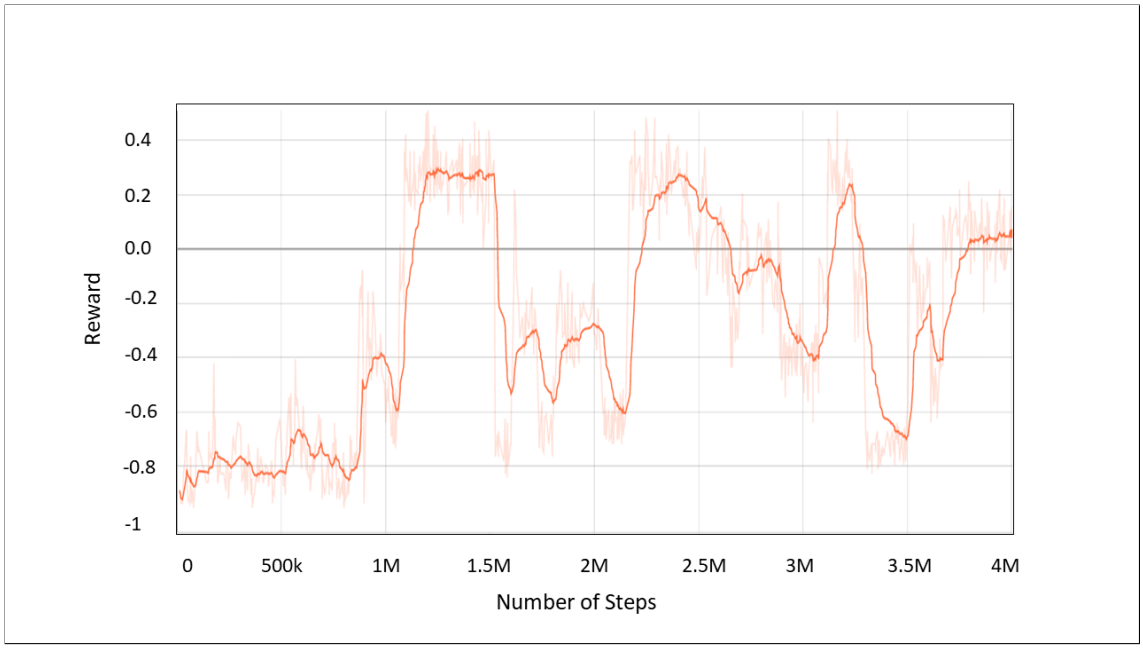

Figure 5.11: Rewards received by PSM over 4 million steps during Model 3 training

Chapter 6

Discussion and Future Work

6.0.1 Note on the Results

The discrete space results show the robustness of learnt discrete RL algorithms and repeatability of the model for successive hand-off tasks. The results from Figure 5.4 demonstrate that the agent was able to follow a trajectory close to that of the user despite varying the initial position. This is a relevant scenario since the needle may not present itself at the same position over each step. The user's trajectory was successfully learned and reproduced on the dVRK to perform multiple passes of the hand-off task with varied start points, with a similarity mean of 1.706mm of the learned trajectory from the user's trajectory, and a cumulative standard deviation of 2.494mm over the trial conducted by the user and the robot. From the Figure 5.3 it can be noticed that learnt trajectory seemed to be more smooth than user trajectory and circumvented the loops present in the user trajectory. It could be because RL is usually used to find an optimized short route to the terminal state. This work used a custom reward function for tracking reference trajectory which possibly resulted in the algorithm finding a balance between optimized route and tracking of trajectory. Depending on the use case, it could be beneficial since it could eliminate unwanted motions and find an alternate optimized route. Another way to

eliminate unwanted movements could be collect multiple trials of the same user and use the average of all trials for creating the custom reward function. As part of the continuous space approach, a real-time RL framework was developed for dVRK PSM using AMBF simulator. The modular PSM environment was tested for goal reach task using DDPG and HER with DDPG. The PSM arm was successfully able to reach the desired position with a success rate of 100% for episodes which allow 250 or greater steps and goal error margin of 10mm. Also, a module was provided to convert collected dVRK ROS data into a format which can be provided to RL algorithms. From Figures 5.6, 5.9 and 5.11 it can be seen that there are points during training when the performance drops suddenly. A possible reason for that could be the instability that occurs in DDPG due to sensitivity to hyper-parameters [49]. Based on the goal error margin and workspace limits set for each iteration, a compromise needed to be made for the solution to converge. It is because when the agent has to explore a new environment from scratch, it can be a non-trivial task leading to erroneous results or deceptive gradients [13, 46]. Better results could have been achieved using these techniques, if a solution was known and only fine tuning was required. The PSM model needed to be trained many times, each time a parameter was varied and the output was evaluated. Many of the iterations did not converge and it might be because techniques such as DDPG rely on the random perturbations for exploring the state space. The type of action noise provided to the algorithm can be a simple Normal Gaussian action noise or Ornstein-Uhlenbeck Action Noise [47] etc. The best suitable action noise can vary depending on the environment and a wrong selection of noise can lead to non-efficient exploration of states [22]. However, the results do show the potential of RL techniques in learning specific tasks which in future could lead to automation of surgical tasks. There are still a lot of problems that RL researchers need to solve for overcoming the limitations of applying RL to a real-world problems. Through this work, by automating the hand-off task, clutch-less suturing can be performed and the user can ignore the workspace constraints

of the PSM_A and RL researchers could use the developed framework to train the dVRK system for various complex tasks. The further additions of the described future work would enable researchers to utilize this framework to train the PSM arm to perform trivial surgical tasks. The results obtained from using this framework could be significantly improved, if various researchers could provide the details of the parameters used by them for training their models (creating a database of parameters for a robot performing specific task).

6.0.2 Applications

The automation of suturing hand-off task could lead to the following applications:

- Due to the high dexterity required for suturing tissue with the da Vinci, obtaining certification to be a trained da Vinci user mandates certain requirements [18]. The methods mentioned could be employed to train novice users by replicating a reference trajectory of the hand-off task performed by an expert user [9].

- Further modifications to the presented methods could be performed to allow haptic feedback to jog the novice user through the trajectory and/or score the novice user's trajectory by evaluating the standard deviation and mean from the expert user's trajectory [55, 73].

- Since robotic suturing is a complex task, a novice user could start practicing suturing using only one arm. Once a good expertise is gained in using a single MTM arm, then the user could start controlling both the da Vinci arms [16].

- The skill level of each user can vary, therefore it could be used for skill analysis which in turn could be used for teaching or training [19].

- A rating scale or a checklist could be developed for performing each surgical task and help standardize evaluation.

- The data collected could also be used for time action and motion analysis of users depending on their response time [5].

- Integration with virtual reality setups and overlaying trajectory data could really help with visualization of movements [38].

6.0.3 Drawbacks

The major drawback of employing discrete RL techniques is that follow Bellman's Optimality equation (Dynamic Programming etc.) is the curse of dimensionality [40]. As the number of states and actions increase, deriving the optimal policy becomes computationally exhaustive. Therefore, the algorithm presented is not scalable and requires limited observations at each state. For a system with s states and a actions, a total of a^s computations are performed to obtain the optimal policy at that state for one iteration [80]. Exploration inefficiency is one of the biggest problems faced by continuous action space RL techniques [65]. Finding an efficient exploration technique can be very hard for new problems and the output might not converge. In order to make the algorithms to converge to a goal and successfully apply RL for practical application, immense experience and familiarity in dealing with similar problems is required. Another possible drawbacks of using off-policy RL techniques such as DDPG is that it is known to suffer from a phenomenon called extrapolation error (when the values of state-action pairs are computed incorrectly and can lead to unrealistic values) [24]. Implementing state of the art RL techniques is still not a trivial task and requires lot of tuning and testing for real world applications [29]. The other major drawbacks of using autonomous agents for practical surgical applications include the low acceptance rate from patients, ethical and legal issues [88].

6.0.4 Future Work

- Look into possible workarounds for improving discrete space such as moving towards employing Deep RL techniques such as Deep Q-network (DQN) or exploring techniques such as multi-step look ahead Q-learning [17].

- At the moment, the developed PSM environment is not compatible to be used with cloud computing resources. A basic docker image version of AMBF has been created by an AIM lab team member, but it has not been tested yet. It could help improve the training process and utilize the high performance clusters available online without running into any installation issues.

- Currently, all the outputs shown for continuous space system is run using a single environment for rollout, training and evaluation. Even though multiple instances of AMBF can be run simultaneously, training multiple instances of the PSM environment needs to be tested.

- Implementation of new RL algorithms such as Twin Delayed DDPG (TD3) could help obtain better results [23].

- Researchers have shown [57] the advantages of using learning from demonstrations techniques to learn an initial policy. A module was developed for creating RL compatible expert data using collected dVRK ROS data, but not utilized in this work. A larger user study involving users performing suturing hand-off and picking up a suturing needle could be conducted. Then it can be provided as an input to GAIL algorithm [31] (can help achieve grasping tasks by improving the training process).

- For implementing grasping in AMBF, a proximity sensor needs to be attached to the end of the PSM. The creator of AMBF is working on adding that feature. Once

the feature is added to AMBF, it could be used to train pick and place task using the created PSM environment.

- As a proof of concept, only dVRK PSM arm support is present currently. Support for more robots could be provided by creating the required forward and inverse kinematic function for each robot.

- Similar to RViz a marker feature needs to be added to AMBF for visualizing goal positions.

- The final results obtained from HER needs to implemented and tested on the physical da Vinci Research Kit.

- Another topic to explore is utilizing parameter noise for exploration rather than action noise. [65].

6.0.5 Conclusion

Initially, an algorithm to generate a sparse reward function was designed and a successful implementation of Q-learning was presented. Trajectories obtained from three learnt policies were compared to the user defined trajectory. A RMSE of [0.0044mm, 0.0027mm, 0.0020mm] was observed and the robustness of the algorithm was calculated using multiple trajectories with varying initial positions from a single policy. Then, a continuous action space approach was presented by developing a real-time RL framework for dVRK system using AMBF simulator. A PSM environment (gym compatible) was created and tested using DDPG and HER with DDPG. For testing the environment, PSM was made to reach random goal positions within the user defined workspace limits and goal position error margin. The results obtained was tested for 3 iterations with each iteration consisting of 20 trials to reach the goal position within the set error margin. For scenarios, when the

maximum steps can be greater than 250 per episode, the PSM model trained using HER with DDPG was successful at reaching the desired positions with a success rate of 100%. In the end, a qualitative analysis of all the results was presented along with the drawbacks and the plans for future work. Bringing autonomous systems into an operating room is definitely not feasible yet. A lot of research, repeated trials, testing and approval from FDA is required before any kind of automation could be used for surgical procedures. But, the emerging research in Artificial Intelligence techniques provide hope that some sort of autonomous agents would be developed in near future to help surgeons.

Chapter 7

Appendix

7.0.1 Generative Adversarial Imitation Learning

Another approach of utilizing expert demonstrations is to directly learn a policy based on the data collected. Unlike IRL where a complex reward function is computed, the algorithm rather tries to replicate the movements of the expert. The movements of the user can be replicated by learning a policy directly and indirectly. Behavioral cloning (BC) is a direct method where the problem is considered as a supervised learning problem. It is a commonly used technique and is usually easy to implement. The shortcomings of BC include requirement of large amounts of data, suffers from compounding error caused by covarient shifts (there is a difference between training and testing dataset in distribution of input variables) and is not good at capturing the intuition of the user. GAIL is an indirect model-free imitation learning technique which in some cases has even outperformed expert behaviors due to its RL based learning approach [31]. It has shown good results in dealing with complex high-dimensional environments. The main difference between GAIL and IRL is that, GAIL learns a policy from the data whereas IRL learns the reward function from the data. The concept of GAIL is similar to Generative Adversarial Networks (GAN), where there is a combination of two neural networks called the Generator and

Discriminator. Ho *et. al.* used Trust Region Policy Optimization to train the generator network and the discriminator is trained using the collected expert data. The goal is similar to general GAN problems, the discriminator should not be able to differentiate between collected expert data and generator's computed data.
The steps to train a GAIL can be listed as follows:

- Collect expert trajectories
- Sample data from the expert trajectories
- Optimize the Policy π_θ
- For a given π_θ, maximize the discriminator to estimate the divergence
- Update the policy π_θ

Equation 7.1 shows the objective function which GAIL tries to optimize. The terms *f* represents the discriminator output, π represents the policy of reinforcement learning, E depicts the expectation values. The RL policy π tries to minimize the divergence. And the term H(π) = $E_\pi[-log\pi(a|s)]$ is the γ-discounted causal entropy of policy π (maximum causal entropy IRL finds a cost function that gives low cost for following an expert policy and high cost for following any other policy) [6].

$$\min_{\pi}\max_{f} E_{x,u\sim\rho_{\pi^*}} log(f(x,u)) + E_{x,u\sim\rho_{\pi}} log(1-f(x,u)) \tag{7.1}$$

7.0.2 dVRK PSM Denavit–Hartenberg Parameters

Figure 7.1 and Figure 7.2 show the Denavit–Hartenberg (DH) frame assignments for the dVRK PSM. The Table 7.1 shows the DH parameter values for PSM. l_{RCC} = 0.4318 m
l_{tool} = 0.4162 m
$l_{Pitch2Yaw}$ = 0.0091 m

Frame	Joint Name	Joint Type	a	α	D	θ
1	Outer Yaw	1	0	π/2	0	$q_1+\pi/2$
2	Outer Pitch	1	0	-π/2	0	$q_2-\pi/2$
3	Insertion	2	0	π/2	$q_3 - l_{RCC}$	0
4	Outer Roll	1	0	0	l_{tool}	q_4
5	Wrist Pitch	1	0	-π/2	0	$q_5-\pi/2$
6	Wrist Yaw	1	$l_{Pitch2Yaw}$	-π/2	0	$q_6 =\pi/2$
7	End Effector	0	0	-π/2	$l_{Yaw2Ctrlpnt}$	0

Table 7.1: DH Parameter Table for PSM [11, 33, 39]

$l_{Yaw2CtrlPnt}$ = 0.0102 m

q_1 to q_6 are the joint variables

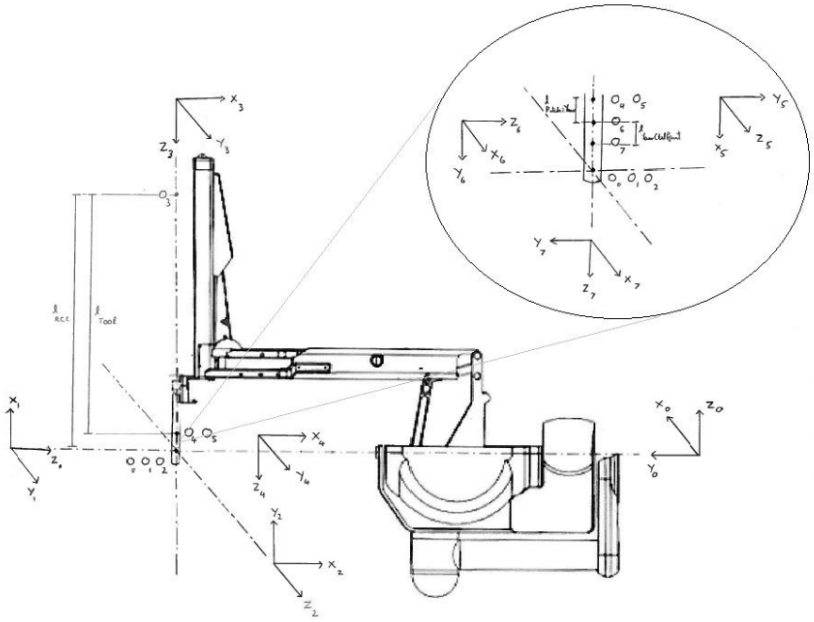

Figure 7.1: PSM DH Frame Assignments [11, 33, 39]

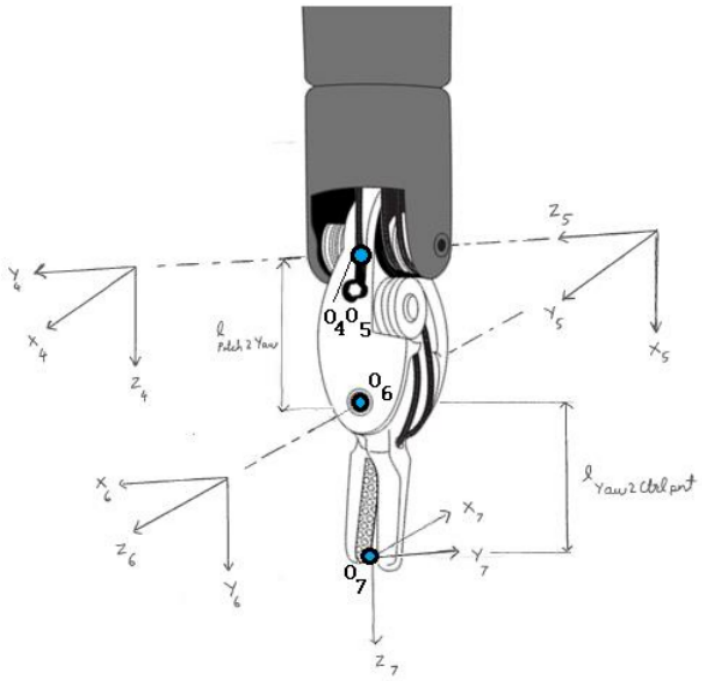

Figure 7.2: PSM Instrument DH Frame Assignments [11, 33, 39]

7.0.3 Stable-Baselines HER with DDPG Model Parameters

The parameters used for training the HER with DDPG (goal error margin=10mm) model are:

- **Action Noise:** Ornstein Uhlenbeck Action Noise with noise standard deviation set to 0.2
- **Actor and Critic Learning Rate:** 0.001
- **Train Steps:** 300
- **Rollout Steps:** 100
- **Gamma:** 0.95
- **Observation Range:** (-1500mm, 1500mm)

- **Random Exploration Factor:** 0.05
- **Normalize Observation:** True
- **Critic L2 Regularization:** 0.01
- **Buffer Size:** 100000
- **Batch Size:** 128
- **Sampled Goal Factor for HER:** 4
- **Goal Selection Strategy for HER:** Future
- **Action Space Limits** (-50mm, 50mm)

The parameters used for training the HER with DDPG (goal error margin=7.5mm) model are:

- **Action Noise:** Ornstein Uhlenbeck Action Noise with noise standard deviation set to 0.2
- **Actor and Critic Learning Rate:** 0.001 and 0.0001
- **Train Steps:** 500
- **Rollout Steps:** 350
- **Gamma:** 0.95
- **Observation Range:** (-1250mm, 1250mm)
- **Random Exploration Factor:** 0.05
- **Normalize Observation:** True
- **Buffer Size:** 100000

- **Batch Size:** 128
- **Sampled Goal Factor for HER:** 4
- **Goal Selection Strategy for HER:** Future
- **Action Space Limits** (-50mm, 50mm)

The parameters used for training the DDPG (goal error margin=10mm) model are:

- **Action Noise:** Ornstein Uhlenbeck Action Noise
- **Actor and Critic Learning Rate:** 0.001
- **Train Steps:** 300
- **Rollout Steps:** 150
- **Gamma:** 0.95
- **Observation Range:** (-1500mm, 1500mm)
- **Random Exploration Factor:** 0.05
- **Normalize Observation:** True
- **Buffer Size:** 100000
- **Critic L2 Regularization:** 0.01
- **Batch Size:** 128
- **Action Space Limits** (-50mm, 50mm)

Bibliography

[1] AGRAWAL, A. S. Automating endoscopic camera motion for teleoperated minimally invasive surgery using inverse reinforcement learning. 120.

[2] ALKADI, S., AND STASSEN, L. Effect of One-Suture and Sutureless Techniques on Postoperative Healing After Third Molar Surgery. *Journal of Oral and Maxillofacial Surgery 77*, 4 (Apr. 2019), 703.e1–703.e16.

[3] ANDRYCHOWICZ, M., WOLSKI, F., RAY, A., SCHNEIDER, J., FONG, R., WELINDER, P., MCGREW, B., TOBIN, J., ABBEEL, P., AND ZAREMBA, W. Hindsight Experience Replay. *arXiv:1707.01495 [cs]* (Feb. 2018). arXiv: 1707.01495.

[4] ARORA, S., AND DOSHI, P. A survey of inverse reinforcement learning: Challenges, methods and progress.

[5] BANI, M. J., AND JAMALI, S. A New Classification Approach for Robotic Surgical Tasks Recognition. *arXiv:1707.09849 [cs]* (July 2017). arXiv: 1707.09849.

[6] BLOEM, M., AND BAMBOS, N. Infinite Time Horizon Maximum Causal Entropy Inverse Reinforcement Learning. 6.

[7] BROCKMAN, G., CHEUNG, V., PETTERSSON, L., SCHNEIDER, J., SCHULMAN, J., TANG, J., AND ZAREMBA, W. OpenAI Gym. *arXiv:1606.01540 [cs]* (June 2016). arXiv: 1606.01540.

[8] BUMM, K., WURM, J., RACHINGER, J., DANNENMANN, T., BOHR, C., FAHLBUSCH, R., IRO, H., AND NIMSKY, C. An automated robotic approach with redundant navigation for minimal invasive extended transsphenoidal skull base surgery. *Minimally invasive neurosurgery : MIN 48* (07 2005), 159–64.

[9] BUSCH, C., NAKADATE, R., UEMURA, M., OBATA, S., JIMBO, T., AND HASHIZUME, M. Objective assessment of robotic suturing skills with a new computerized system: A step forward in the training of robotic surgeons. *Asian Journal of Endoscopic Surgery 12*, 4 (2019), 388–395.

[10] CHEN, Z., DEGUET, A., TAYLOR, R., DIMAIO, S., FISCHER, G., AND KAZANZIDES, P. An Open-Source Hardware and Software Platform for Telesurgical Robotics Research. 11.

[11] CHEN, Z., DEGUET, A., TAYLOR, R. H., AND KAZANZIDES, P. Software Architecture of the Da Vinci Research Kit. In *2017 First IEEE International Conference on Robotic Computing (IRC)* (Taichung, Taiwan, Apr. 2017), IEEE, pp. 180–187.

[12] CHIU, H.-Y., KANG, Y.-N., WANG, W.-L., TONG, Y.-S., CHANG, S.-W., FONG, T.-H., AND WEI, P.-L. Gender differences in the acquisition of suturing skills with the da Vinci surgical system. *Journal of the Formosan Medical Association 119*, 1 (Jan. 2020), 462–470.

[13] COLAS, C., SIGAUD, O., AND OUDEYER, P.-Y. GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms. *arXiv:1802.05054 [cs]* (Sept. 2018). arXiv: 1802.05054.

[14] D'ETTORRE, C., DWYER, G., DU, X., CHADEBECQ, F., VASCONCELOS, F., DE MOMI, E., AND STOYANOV, D. Automated Pick-Up of Suturing Needles for Robotic Surgical Assistance. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (Brisbane, QLD, May 2018), IEEE, pp. 1370–1377.

[15] DHARIWAL, P., HESSE, C., KLIMOV, O., NICHOL, A., PLAPPERT, M., RADFORD, A., SCHULMAN, J., SIDOR, S., WU, Y., AND ZHOKHOV, P. OpenAI Baselines.

[16] DULAN, G., REGE, R. V., HOGG, D. C., GILBERG-FISHER, K. M., ARAIN, N. A., TESFAY, S. T., AND SCOTT, D. J. Developing a comprehensive, proficiency-based training program for robotic surgery. *Surgery 152*, 3 (Sept. 2012), 477–488.

[17] EFRONI, Y., TOMAR, M., AND GHAVAMZADEH, M. Multi-step greedy policies in model-free deep reinforcement learning.

[18] ESTES, S., GOLDENBERG, D., WINDER, J., JUZA, R., AND LYN-SUE, J. Best practices for robotic surgery programs. *JSLS 21(2):e2016.00102* (01 2017).

[19] FARD, M. J., AMERI, S., AND ELLIS, R. D. Skill assessment and personalized training in robotic-assisted surgery. *CoRR abs/1610.07245* (2016).

[20] FONTANELLI, G., SELVAGGIO, M., BUONOCORE, L., FICUCIELLO, F., VILLANI, L., AND SICILIANO, B. A new laparoscopic tool with in-hand rolling capabilities for needle reorientation. *IEEE Robotics and Automation Letters PP* (07 2018), 1–1.

[21] FONTANELLI, G. A., FICUCIELLO, F., VILLANI, L., AND SICILIANO, B. Modelling and identification of the da Vinci Research Kit robotic arms. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Vancouver, BC, Sept. 2017), IEEE, pp. 1464–1469.

[22] FORTUNATO, M., AZAR, M. G., PIOT, B., MENICK, J., OSBAND, I., GRAVES, A., MNIH, V., MUNOS, R., HASSABIS, D., PIETQUIN, O., BLUNDELL, C., AND LEGG, S. Noisy Networks for Exploration. *arXiv:1706.10295 [cs, stat]* (July 2019). arXiv: 1706.10295.

[23] FUJIMOTO, S., HOOF, H. V., AND MEGER, D. Addressing function approximation error in actor-critic methods. *CoRR abs/1802.09477* (2018).

[24] FUJIMOTO, S., MEGER, D., AND PRECUP, D. Off-Policy Deep Reinforcement Learning without Exploration. *arXiv:1812.02900 [cs, stat]* (Aug. 2019). arXiv: 1812.02900.

[25] GAO, Y., VEDULA, S. S., REILEY, C. E., AHMIDI, N., VARADARAJAN, B., LIN, H. C., TAO, L., ZAPPELLA, L., BEJAR, B., YUH, D. D., CHEN, C. C. G., VIDAL, R., KHUDANPUR, S., AND HAGER, G. D. JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS): A Surgical Activity Dataset for Human Motion Modeling. 10.

[26] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. The MIT Press, 2016.

[27] GRONDMAN, I., BUSONIU, L., LOPES, G. A. D., AND BABUSKA, R. A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. 17.

[28] GUO, X. Deep Learning and Reward Design for Reinforcement Learning. 117.

[29] HENDERSON, P., ISLAM, R., BACHMAN, P., PINEAU, J., PRECUP, D., AND MEGER, D. Deep Reinforcement Learning that Matters. *arXiv:1709.06560 [cs, stat]* (Jan. 2019). arXiv: 1709.06560.

[30] HILL, A., RAFFIN, A., ERNESTUS, M., GLEAVE, A., KANERVISTO, A., TRAORE, R., DHARIWAL, P., HESSE, C., KLIMOV, O., NICHOL, A., PLAPPERT, M., RADFORD, A., SCHULMAN, J., SIDOR, S., AND WU, Y. Stable Baselines.

[31] HO, J., AND ERMON, S. Generative Adversarial Imitation Learning. *arXiv:1606.03476 [cs]* (June 2016). arXiv: 1606.03476.

[32] INTUITIVE SURGICAL, I. 2019 annual report, 2019.

[33] INTUITIVE SURGICAL, I. User guide for project: The da vinci research kit, 2019.

[34] ISRAELSSON, L. A., AND JONSSON, T. Suture length to wound length ratio and healing of midline laparotomy incisions. *BJS (British Journal of Surgery) 80* (1993), 1284–1286.

[35] JACKSON, R. C., AND CAVUSOGLU, M. C. Needle path planning for autonomous robotic surgical suturing. In *2013 IEEE International Conference on Robotics and Automation* (Karlsruhe, Germany, May 2013), IEEE, pp. 1669–1675.

[36] JANSEN, R., HAUSER, K., CHENTANEZ, N., VAN DER STAPPEN, F., AND GOLDBERG, K. Surgical retraction of non-uniform deformable layers of tissue: 2d robot grasping and path planning. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2009), pp. 4092–4097.

[37] KANG, H. ROBOTIC ASSISTED SUTURING IN MINIMALLY INVASIVE SURGERY. 183.

[38] KASSAHUN, Y., YU, B., TIBEBU, A. T., GIANNAROU, S., METZEN, J. H., AND POORTEN, V. Surgical Robotics Beyond Enhanced Dexterity Instrumentation. 15.

[39] KAZANZIDES, P., CHEN, Z., DEGUET, A., FISCHER, G. S., TAYLOR, R. H., AND DIMAIO, S. P. An Open-Source Research Kit for the da Vinci R Surgical System. 6.

[40] KOBER, J., BAGNELL, J. A., AND PETERS, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research 32*, 11 (Sept. 2013), 1238–1274.

[41] LAMATA, P., GÓMEZ, E., SÁNCHEZ MARGALLO, F., FÉLIX, L., DEL POZO GUERRERO, F., AND USÓN, J. Tissue consistency perception in laparoscopy to define the level of fidelity in virtual reality simulation. *Surgical endoscopy 20* (09 2006), 1368–75.

[42] LANFRANCO, A. R., CASTELLANOS, A. E., DESAI, J. P., AND MEYERS, W. C. Robotic surgery: a current perspective. *Annals of surgery 239 1* (2004), 14–21.

[43] LAVELY, H. T. The responsibilities of a surgeon. *Journal of the Tennessee Medical Association 66*, 2 (Feb. 1973), 115–117.

[44] LEONARD, S., SHADEMAN, A., KIM, Y., KRIEGER, A., AND KIM, P. C. W. Smart Tissue Anastomosis Robot (STAR): Accuracy evaluation for supervisory suturing using near-infrared fluorescent markers. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (Hong Kong, China, May 2014), IEEE, pp. 1889–1894.

[45] LEONARD, S., WU, K., KIM, Y., KRIEGER, A., AND KIM, P. Smart tissue anastomosis robot (star): A vision-guided robotics system for laparoscopic suturing. *Biomedical Engineering, IEEE Transactions on 61* (04 2014), 1305–1317.

[46] LIESSNER., R., SCHMITT., J., DIETERMANN., A., AND BÄKER., B. Hyperparameter optimization for deep reinforcement learning in vehicle energy management. In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,* (2019), INSTICC, SciTePress, pp. 134–144.

[47] LILLICRAP, T. P., HUNT, J. J., PRITZEL, A., HEESS, N., EREZ, T., TASSA, Y., SILVER, D., AND WIERSTRA, D. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]* (July 2019). arXiv: 1509.02971.

[48] LIU, T., AND ÇAVUŞOĞLU, M. C. Optimal Needle Grasp Selection for Automatic Execution of Suturing Tasks in Robotic Minimally Invasive Surgery. *IEEE International Conference on Robotics and Automation : ICRA : [proceedings] IEEE International Conference on Robotics and Automation 2015* (May 2015), 2894–2900.

[49] MATHERON, G., PERRIN, N., AND SIGAUD, O. The problem with ddpg: understanding failures in deterministic environments with sparse rewards. *ArXiv abs/1911.11679* (2019).

[50] MAYER, H., GOMEZ, F., WIERSTRA, D., NAGY, I., KNOLL, A., AND SCHMIDHUBER, J. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2006), pp. 543–548.

[51] MENDES, K. D. S., ROZA, B. D. A., BARBOSA, S. D. F. F., SCHIRMER, J., AND GALVÃO, C. M. Organ and tissue transplantation: responsibilities of nurses. *Texto & Contexto - Enfermagem 21*, 4 (Dec. 2012), 945–953.

[52] MNIH, V., KAVUKCUOGLU, K., SILVER, D., GRAVES, A., ANTONOGLOU, I., WIERSTRA, D., AND RIEDMILLER, M. Playing Atari with Deep Reinforcement Learning. 9.

[53] MOY, R. L., WALDMAN, B., AND HEIN, D. W. A review of sutures and suturing techniques. *The Journal of Dermatologic Surgery and Oncology 18*, 9 (Sept. 1992), 785–795.

[54] MUNAWAR, A. Implementation of a Surgical Robot Dynamical Simulation and Motion Planning Framework. 126.

[55] MUNAWAR, A., AND FISCHER, G. S. An Asynchronous Multi-Body Simulation Framework for Real-Time Dynamics, Haptics and Learning with Application to Surgical Robots. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Nov. 2019), pp. 6268–6275. ISSN: 2153-0858.

[56] MURALI, A., SEN, S., KEHOE, B., GARG, A., MCFARLAND, S., PATIL, S., BOYD, W. D., LIM, S., ABBEEL, P., AND GOLDBERG, K. Learning by observation for surgical subtasks: Multilateral cutting of 3D viscoelastic and 2D Orthotropic Tissue Phantoms. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (Seattle, WA, USA, May 2015), IEEE, pp. 1202–1209.

[57] NAIR, A., MCGREW, B., ANDRYCHOWICZ, M., ZAREMBA, W., AND ABBEEL, P. Overcoming Exploration in Reinforcement Learning with Demonstrations. *arXiv:1709.10089 [cs]* (Feb. 2018). arXiv: 1709.10089.

[58] NG, A. Y., AND RUSSELL, S. Algorithms for Inverse Reinforcement Learning. In *in Proc. 17th International Conf. on Machine Learning* (2000), Morgan Kaufmann, pp. 663–670.

[59] OSA, T., SUGITA, N., AND MITSUISHI, M. Online Trajectory Planning and Force Control for Automation of Surgical Tasks. *IEEE Transactions on Automation Science and Engineering 15*, 2 (Apr. 2018), 675–691.

[60] O'MAHONY, N., CAMPBELL, S., CARVALHO, A., HARAPANAHALLI, S., HERNANDEZ, G. V., KRPALKOVA, L., RIORDAN, D., AND WALSH, J. Deep Learning vs. Traditional Computer Vision. In *Advances in Computer Vision*, K. Arai and

S. Kapoor, Eds., vol. 943. Springer International Publishing, Cham, 2020, pp. 128–144. Series Title: Advances in Intelligent Systems and Computing.

[61] PADOY, N., AND HAGER, G. D. Human-Machine Collaborative surgery using learned models. In *2011 IEEE International Conference on Robotics and Automation* (Shanghai, China, May 2011), IEEE, pp. 5285–5292.

[62] PALEP, J. Robotic assisted minimally invasive surgery. *Journal of Minimal Access Surgery 5* (2009), 1–7.

[63] PATIL, S., AND ALTEROVITZ, R. Toward automated tissue retraction in robot-assisted surgery. In *2010 IEEE International Conference on Robotics and Automation* (Anchorage, AK, May 2010), IEEE, pp. 2088–2094.

[64] PENG, X. B., ABBEEL, P., LEVINE, S., AND VAN DE PANNE, M. DeepMimic: example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics 37*, 4 (Aug. 2018), 1–14.

[65] PLAPPERT, M., HOUTHOOFT, R., DHARIWAL, P., SIDOR, S., CHEN, R. Y., CHEN, X., ASFOUR, T., ABBEEL, P., AND ANDRYCHOWICZ, M. Parameter Space Noise for Exploration. *arXiv:1706.01905 [cs, stat]* (Jan. 2018). arXiv: 1706.01905.

[66] POPOV, I., HEESS, N., LILLICRAP, T., HAFNER, R., BARTH-MARON, G., VECERIK, M., LAMPE, T., TASSA, Y., EREZ, T., AND RIEDMILLER, M. Data-efficient Deep Reinforcement Learning for Dexterous Manipulation. 12.

[67] REILEY, C. E., PLAKU, E., AND HAGER, G. D. Motion generation of robotic surgical tasks: Learning from expert demonstrations. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology* (Buenos Aires, Aug. 2010), IEEE, pp. 967–970.

[68] RENAUDO, E., GIRARD, B., CHATILA, R., AND KHAMASSI, M. Respective Advantages and Disadvantages of Model-based and Model-free Reinforcement Learning in a Robotics Neuro-inspired Cognitive Architecture. *Procedia Computer Science 71* (2015), 178–184.

[69] RICHTER, F., OROSCO, R. K., AND YIP, M. C. Open-Sourced Reinforcement Learning Environments for Surgical Robotics. *arXiv:1903.02090 [cs]* (Jan. 2020). arXiv: 1903.02090.

[70] SALVADOR, S., AND CHAN, P. Toward accurate dynamic time warping in linear time and space. *Intell. Data Anal. 11*, 5 (Oct. 2007), 561–580.

[71] SATAVA, R. M. Surgical robotics: the early chronicles: a personal historical perspective. *Surgical laparoscopy, endoscopy & percutaneous techniques 12 1* (2002), 6–16.

[72] SATAVA, R. M. Robotic surgery: from past to future—a personal journey. *Surgical Clinics of North America 83* (2003), 1491–1500.

[73] SELVAGGIO, M., E, A. M. G., MOCCIA, R., FICUCIELLO, F., AND SICILIANO, B. Haptic-guided shared control for needle grasping optimization in minimally invasive robotic surgery. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), pp. 3617–3623.

[74] SEN, S., GARG, A., GEALY, D. V., MCKINLEY, S., JEN, Y., AND GOLDBERG, K. Automating multi-throw multilateral surgical suturing with a mechanical needle guide and sequential convex optimization. In *2016 IEEE International Conference on Robotics and Automation (ICRA)* (Stockholm, Sweden, May 2016), IEEE, pp. 4178–4185.

[75] SEPEHRIPOUR, A., GARAS, G., ATHANASIOU, T., AND CASULA, R. Robotics in cardiac surgery. *Ann R Coll Surg Engl. 2018;100(Suppl 7):22–33.*.

[76] SERMANET, P., XU, K., AND LEVINE, S. Unsupervised perceptual rewards for imitation learning. *CoRR abs/1612.06699* (2016).

[77] SPEIDEL, S., KROEHNERT, A., BODENSTEDT, S., KENNGOTT, H., MÜLLER-STICH, B., AND DILLMANN, R. Image-based tracking of the suturing needle during laparoscopic interventions. R. J. Webster and Z. R. Yaniv, Eds., p. 94150B.

[78] SPONG, M., HUTCHINSON, S., AND VIDYASAGAR, M. *Robot Modeling and Control*. Wiley, 2005.

[79] STEGEMANN, A. P., AHMED, K., SYED, J. R., REHMAN, S., GHANI, K., AUTORINO, R., SHARIF, M., RAO, A., SHI, Y., WILDING, G. E., HASSETT, J. M., CHOWRIAPPA, A., KESAVADAS, T., PEABODY, J. O., MENON, M., KAOUK, J., AND GURU, K. A. Fundamental Skills of Robotic Surgery: A Multi-institutional Randomized Controlled Trial for Validation of a Simulation-based Curriculum. *Urology 81*, 4 (Apr. 2013), 767–774.

[80] SUTTON, R. S., AND BARTO, A. G. *Reinforcement learning: an introduction*, second edition ed. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, 2018.

[81] THANANJEYAN, B., GARG, A., KRISHNAN, S., CHEN, C., MILLER, L., AND GOLDBERG, K. Multilateral surgical pattern cutting in 2D orthotropic gauze with deep reinforcement learning policies for tensioning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)* (Singapore, Singapore, May 2017), IEEE, pp. 2371–2378.

[82] THIYAGARAJAN, M., AND RAVINDRAKUMAR, C. A Comparative Study in Learning Curves of Two Different Intracorporeal Knot Tying Techniques. *Minimally Invasive Surgery 2016* (2016), 1–7.

[83] VAN DEN BERG, J., MILLER, S., DUCKWORTH, D., HU, H., WAN, A., XIAO-YU FU, GOLDBERG, K., AND ABBEEL, P. Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations. In *2010 IEEE International Conference on Robotics and Automation* (Anchorage, AK, May 2010), IEEE, pp. 2074–2081.

[84] VARGAS, R., MOSAVI, A., AND RUIZ, R. Deep Learning: A Review. preprint, MATHEMATICS & COMPUTER SCIENCE, Oct. 2018.

[85] VECERIK, M., HESTER, T., SCHOLZ, J., WANG, F., PIETQUIN, O., PIOT, B., HEESS, N., ROTHÖRL, T., LAMPE, T., AND RIEDMILLER, M. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *arXiv:1707.08817 [cs]* (Oct. 2018). arXiv: 1707.08817.

[86] WANG, Y., GONDOKARYONO, R., MUNAWAR, A., AND FISCHER, G. S. A Convex Optimization-Based Dynamic Model Identification Package for the da Vinci Research Kit. *IEEE Robotics and Automation Letters 4*, 4 (Oct. 2019), 3657–3664.

[87] WULFMEIER, M., ONDRUSKA, P., AND POSNER, I. Maximum Entropy Deep Inverse Reinforcement Learning. *arXiv:1507.04888 [cs]* (Mar. 2016). arXiv: 1507.04888.

[88] YIP, M., AND DAS, N. Robot Autonomy for Surgery. *arXiv:1707.03080 [cs]* (July 2017). arXiv: 1707.03080.

[89] ZIEBART, B. D., MAAS, A., BAGNELL, J. A., AND DEY, A. K. Maximum Entropy Inverse Reinforcement Learning. 6.