TÉCNICO LISBOA

Combining Active Learning with Neural Networks for Robot Self-Calibration

Daniel Nobre Chagas

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Alexandre José Malheiro Bernardino
Prof. Plinio Moreno Lopez

Examination Committee

Chairperson: Prof. Joao Manuel de Freitas Xavier
Supervisor: Prof. Alexandre José Malheiro Bernardino
Member of the Committee: Dr. Pedro Emanuel Antunes Ouro Vicente

November 2023

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

Firstly, I would like to express my deepest gratitude to my family, especially my parents, Anabela Chagas e José Chagas, for all the love and education throughout my life, for all the unconditional support and for making sure that I had everything I need. I would also like to thank my grandparents for all the support and for always believing in me.

Special thanks to my thesis supervisors Alexandre Bernardino and Plinio Moreno for all the guidance you provided which was crutial for the conclusion of this thesis.

I am deeply indebted to Alexandre Gabriel, Bruno Estêvão, Gonçalo Mestre, João Campião and Tiago Jesus for all the years of friendship, fun times and all the unforgettable memories.

Words cannot express my gratitude to my friends ”Vices” and Jéssica for all the emotional support, good laughs and unforgettable moments from the last few years.

Many thanks to karma's staff for the friendship and providing the best work environment. Lastly, I'd like to thank everyone who left a significant mark throughout this journey and helped me becoming who I am today.

Abstract

The calibration of robots can be laborious, expensive and time consuming. Non-industrial robots, like humanoid and soft robots, have complex kinematic chains with several Degrees of Freedom (DoF) that are difficult to model. It is important that the robot is continuously calibrated to keep its productivity and adaptation to new environments. Active Learning (AL) is a decision-making process that chooses the most informative observations to reduce the amount of samples needed to learn a model. Using a Neural Network (NN) to learn the kinematic model of a robot gives more flexibility than using the traditional Denavit-Hartenberg (DH) parameters. In this thesis we propose a framework combining the learning capabilities of deep learning with the sample selection efficiency of active learning for humanoid robots' self-calibration. This work was implemented in a simulation environment and uses the 7-DoF right arm of the iCub simulator. The framework is tested for three different active learning approaches: i) Greedy Sampling on Input (GSx), ii) Improved Greedy Sampling (iGS) and iii) Monte-Carlo Dropout (McD). The performance of these approaches are compared with a random sample selection baseline. The results show that using Greedy Sampling on input improves the learning performance of the neural network, outperforming random sampling and the other approaches.

Keywords

Robotics, Active Learning, Deep Learning, Deep Active Learning, Calibration.

Resumo

A calibração de robôs pode ser laboriosa, cara e demorada. Robôs não industriais, como robôs humanoides e flexíveis, têm cadeias de cinemática complexas com diversos graus de liberdade que são difíceis de modelar. É importante que o robot seja continuamente calibrado para manter a sua produtividade e adaptação a novos ambientes. Aprendizagem Ativa é um processo de decisão que escolhe as observações mais informativas para reduzir o número de amostras necessárias para aprender um modelo. Usando uma rede neuronal para aprender o modelo de cinemática de um robô oferece mais flexibilidade do que usando os tradicionais parâmetros de Denavit–Hartenberg (DH). Nesta tese propomos um framework combinando as capacidades de aprendizagem de aprendizagem profunda com a eficiência de seleção de amostras de Aprendizagem Ativa para a auto-calibração de um robô humanoide. Este trabalho foi implementado num ambiente de simulação e usa o braço direito de 7 graus de liberdade do simulador do iCub. O framework é testado para três diferentes abordagens de Aprendizagem Ativa: i) Amostragem Ambiciosa na entrada, ii) Amostragem Ambiciosa melhorada e iii) Monte Carlo Dropout. Os desempenhos destas abordagens são comparados com seleção aleatoria das amostras. Os resultados mostram que usando amostragem ambiciosa na entrada melhora o desempenho da aprendizagem da rede neuronal superando a amostragem aleatória e as outras abordagens.

Palavras Chave

Robótica, Aprendizagem Ativa, Aprendizagem Profunda, Aprendizagem Ativa e Profunda, Calibração.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AL	Active Learning
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DH	Denavit-Hartenberg
DL	Deep Learning
DNN	Deep Neural Network
DoF	Degrees of Freedom
GNN	Graph Neural Network
GSx	Greedy Sampling on Input
iGS	Improved Greedy Sampling
McD	Monte-Carlo Dropout
MCDUE	Monte-Carlo Dropout Uncertainty Estimation
MLP	Multilayer Perceptrons
NN	Neural Network
RNN	Recurrent neural networks

1

Introduction

Contents

Robots rely on models of their bodies to accomplish most of their tasks and how well they perform relates directly to their calibration. The calibration of the parameters of these models is a fundamental pillar where all the robots' motion is built on. However, the calibration process can be very difficult, expensive and time consuming. This is especially true for soft robots and non-industrial robots where flexibility exists and that can be deployed in uncertain environments, making their default models less precise. So it is important that the robot is able to perform self-calibration regularly to improve productivity, since irregular conditions like gear backlash, changes in the environment or in the workspace can cause changes in the robots' model. For this reason it is important that the robot is able to update its models in a safe, fast and efficient way, so that it can quickly adapt to the changes in its workspace without losing productivity.

Human newborns, in their first few months of life, also try to learn how their bodies work through sensorimotor information, performing multiple motions with their arms and legs and observing the result [5], a similar learning process occurs when an amputee is adapting to its prosthesis. This learning process is what allow humans to adapt to the environment and conditions surrounding them and it is analogous with self-calibration in robotics described before. So, creating robots that continuously use this methodology to update their kinematics, adapting to the new conditions exposed to them, would increase their autonomous efficiency in a wide variety of tasks.

In recent years, Deep Learning (DL) made unprecedented breakthroughs in various challenging tasks from multiple research areas. This is due to its powerful learning capabilities. Regarding automation, it has transformed the manual design of features in machine learning to facilitate automatic extraction. This powerful automatic feature extraction ability is also why it has showed unrivaled advantages in many fields. So, since deep learning has already provided good results in the learning domain, it should not be surprising that multiple researches such as [6–8] have extended the use of Neural Networks to learn the kinematic models of a robot, exhibiting very accurate results. Another advantage of using Neural Networks in the estimation of the robots' model in comparison to the traditional methods like the estimation of Denavit-Hartenberg (DH) parameters is that it offers more flexibility. It also allows it to represent a bigger number of functions, even being able to estimate the kinematics of soft robots that are really complex. Although Deep Neural Networks can provide very accurate estimates of a model, it is very expensive to train them since they require a lot of data. This data usually has similar samples that provide redundant information to the model, which, in the case of learning a robot's model, would result in an inefficient exploration of the robots workspace during the calibration.

In contrast to the standard methods of training deep learning models, Active Learning (AL) aims to select the most useful samples of an unlabeled dataset to reduce the labeling cost as much as possible while still maintaining performance. In other words, the main idea of active learning is to maximize the learning performance by reducing the labeled training instances and picking the most informative

data that it can learn. Let us consider the practical example of a robotic hand with tactile sensors that generates models of objects through tactile exploration. If, instead of randomly picking positions to touch the object, it touches positions that generate more information, like vertices or edges, it will be able to generate the object's model faster and more efficiently. A similar work using active learning was made in [9]. This is mainly done by associating a different cost with the learning task, which can be based, for example, on the uncertainty of the learning model or the diversity of the selected samples. So, active learning is a decision-making process that uses manual or automatic methods to design models with high performance feature extraction capabilities. All of this also makes active learning a great way to increase the performance when the robot is in an unknown environment, which is a desired characteristic for self-calibration, allowing the robot to rapidly adapt to the new environment. Researches like [1] have also proven that using Active learning can also reduce the amount of movement during calibration.

Active learning and deep learning present complementary advantages, so, by combining them it is expected to achieve superior results regarding performance and number of iterations. Some methods in the literature can be used to integrate AL with DL frameworks, a good example was introduces in [10] and [11], that allow the estimation of uncertainty in deep learning models that can be used in Active Learning. Another good example was introduced in [3] and [2], which introduce an AL method based on the geometric characteristics of the samples, promoting diversity between the selected samples, which can be directly employed to any Neural Network (NN).

1.1 Problem Statement and Contributions

The robot calibration problem consists on learning of a set of parameters which allows the description of the motion of the robot in the real world. To be more specific, this thesis focuses on the kinematic calibration problem, which consists of learning a set of parameters that allows the mapping of a joint configuration for the robotic arm to the position and orientation of its hand. To learn these parameters, the learning model requires multiple samples of the joint configurations and the corresponding position and orientation of the end-effector.

To solve this problem, this thesis proposes a flexible self-calibration framework that makes use of active learning methods to improve the sample selection, providing more informative samples to enhance the learning performance of a neural network. The framework is able to estimate the robot's hand pose (position and orientation) from a joint configuration of its arm. So, the framework assumes that there are enough sensors to accurately observe the robots' movements for executing the calibration. Figure 1.1 depicts a flowchart that describes the general process of the proposed framework.

Figure 1.1 provides the keys steps of the calibration for a query-synthesis based approach. It starts by selecting the next joint configuration, using active learning, then labels the configuration and adds it

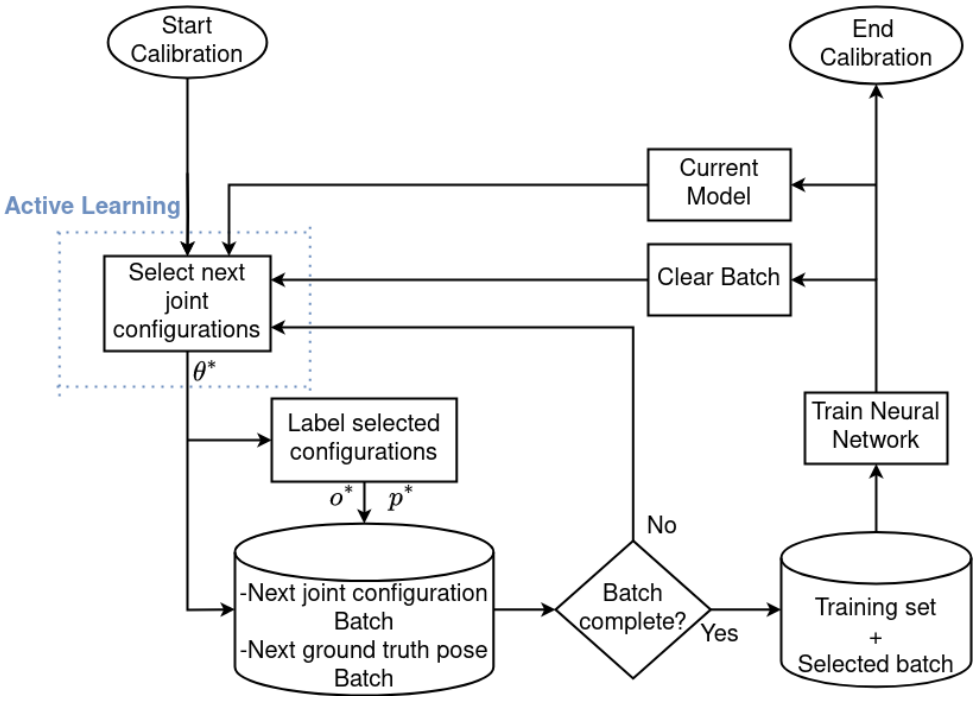


Figure 1.1: General calibration flowchart containing the key steps of the calibration process. θ^* is the joint configuration chosen by the AL process at a given iteration while p^* and o^* are the corresponding position and orientation, respectively.

to the batch. It repeats this process until the batch has the intended size and then adds it to the training set. This training set is used to train the neural network that learns the forward kinematics. Finally, it clears the batch and repeats the whole process for the intended number of iterations. The calibration process for pool based approaches is very similar. The main differences are that the configurations are selected and then removed from the pool.

The framework will be tested for three different active learning approaches: Greedy Sampling on input, Improved Greedy Sampling and Monte Carlo Dropout. A comparative study between the three approaches and random sampling selection will be presented to assess the effectiveness of each sampling selection criteria. This work also studies the performance of the framework using pool based and query-synthesis based AL approaches. Pool based AL chooses the samples from a pool of unlabeled data while query-synthesis based AL generates its own queries for labelling.A comparison between pool based AL and query-synthesis AL for all the previously mentioned approaches will also be presented. Finally, regarding pool based AL, this thesis presents a study to compare how the size of the pool affects the AL approaches. All comparisons mentioned will evaluate how effectively these methods can reduce position and orientation errors while learning the forward kinematics of iCub.

1.2 Thesis Outline

The structure of this thesis is organized as followed:

Chapter 2 - State-of-the-Art: provides a brief overview of the state-of-the-art in the fields of robot calibration, active learning for regression tasks, uncertainty estimation for neural network models, and the integration of deep learning with active learning frameworks.

Chapter 3 - Methods: Presents a thorough examination of the methodologies employed within this thesis. Focusing on two main key elements: robot's model and learning process.

Chapter 4 - Experimental Setup: Offers a review of the implementation details of this thesis. Introduces the iCub simulator, reports on implementation details of the neural network and the active learning methods, presents the comparison metrics and shows a visual representation of the calibration routine.

Chapter 5 - Results: Presents the simulated results.

Chapter 6 - Conclusion and Future Work: Draws conclusion from the presented results and presents suggestions for future work.

2

State-of-the-art

Contents

This chapter offers an overview of the state of the art methods regarding robot calibration, active learning for regression tasks, uncertainty estimation for neural network models and the frameworks combining deep learning with active learning.

2.1 Robot Calibration

There is a considerable amount of research focusing on robot calibration. Multiple new techniques were developed using different learning methods for a plethora of robot models. A good example, was proposed in [6] where the authors use multiple kinematic chains for self-calibration of the iCub robot. They combine the kinematic chains of both arms and both eyes in different ways to compute all the DH parameters of every joint, getting the information needed through either touch sensors and the cameras in the robot's eyes. Making a comparison between self-observation, combining the chains of one arm and one eye, self touch by using both arms only, stereo self-observation, both eyes and one arm, and combining all kinematic chains together. The last one outperformed the former in both end-effector position estimation, although it needed around 100 poses to get to an error of one millimeter, and DH parameter estimation.

Focusing on the use of neural networks for the estimation of the kinematics of a robot, in [7] a simple neural network is used to compute the inverse kinematics of a planar three-link manipulator. Using the position and orientation of the end-effector from the forward kinematics as input and the joint angles and the outputs. A more robust approach was presented in [8], where the authors compare the use of a traditional Artificial Neural Network (ANN) design with a new proposed design to solve the inverse kinematics of a 6-DOF robotic arm. The traditional design has the position and orientation of the end effector (6 variables) as its input, 10 hidden layers and the 6 angles of the joints as an output, while the design they proposed includes the feedback of the current joint configuration, having the previous 6 variables as input plus 6 new inputs for the current joint configuration and maintaining the number of hidden layers and outputs. Regarding experimental results they conclude that the new proposed ANN presented higher accuracy while minimizing the position error and using around half of the epochs of the traditional ANN.

Besides articulated robots, another type of robot where it is interesting to use neural network as a learning model for the calibration is the soft robots. Traditional kinematic models, for example DH-parameters, were developed for robots composed by rigid joints and links, therefore cannot be used for soft robots. So, there is a need of generic input-output models like neural networks to learn their kinematic models. The NN can also provide flexibility to deal with their infinite Degrees of Freedom (DoF). So, to model the kinematics of a silicone soft robot, the authors in [12] proposed the use of a Multilayer Perceptrons (MLP), receiving as input the actuation of its four actuation cables and computing the posi-

tion of its end-effector (top point of the robot). Although this method required collecting enough data for the purpose of largely covering the robot's workspace, which is obviously time-consuming. In an homogeneous way, [13] proposed the use of a Recurrent Neural Network for the estimation of the kinematics of a pneumatic soft finger. Similarly, in [14] the authors make use of a Graph Neural Network (GNN) to model the kinematics of a robotic soft hand considering that the system in composed by simple interacting parts connected by edges and considering the system is order invariant. Since the soft robot systems requires self-calibration due to their complex dynamics the authors consider a sensorimotor learning approach where the robot will perform multiple experiments with their body, observing the effects of actuation signals on the robotic soft-hand and allowing it to learn its own kinematics. They test the framework on a simulation setup, testing different baseline network architectures and showing that the proposed model outperforms them in problems involving different number of fingers in the robots hand. Even though these approaches achieve accurate representations of the robots' models, using AL could improve their performances and the reduce number of samples required to learn the model.

2.2 Active Learning

Active learning has been extensively researched for classification task while only in recent year some works were proposed to improve the quality of AL in regression tasks. In [15] a general overview of the use of Active Learning is given for multiple areas of robotics like search and rescue, localization and mapping and imitation learning while also referring to how important it is for the robot to learn its own dynamics. For this it shows the comparison between using active or passively learning the Koopman operator by turning of a rotor of a quadrotor vehicle mid-flight and showing that using active learning the robot can learn its own dynamics and can recover faster than using passive learning.

Yu et al [2] proposed a new active learning approach for regression based on passive sampling. This means that, unlike most AL approaches that try to sample the most informative samples with respect to a learning function, it focuses on getting the most informative samples based on their geometric characteristics. This approach chooses the samples that is the farthest away to the already chosen samples, providing diversity to the selected batch. Following its own work, [3] proposes a new approach that takes into consideration not only the diversity of the input samples but also the diversity in the output space. Following a similar concept [16] applied diversity based AL to 3D object detection in autonomous driving, proving that it outperformed both random sample selection and uncertainty based AL.

Some AL approaches are based on the uncertainty of the model, so an important part of these type of approaches resides on uncertainty estimation. Loquercio et al [10] propose a novel framework for uncertainty estimation in Deep Learning, based in Bayesian Belief networks and Monte Carlo sampling. This framework shows some appealing properties that enables it to be better integrated into robotic

systems. It is agnostic to the network's architecture and its task, it does not require any changes in the optimisation or learning process and it can be applied to already trained networks. The model considers the two main sources of uncertainty involved in deep learning: data uncertainty and model uncertainty. Using an Assumed Density Filtering network and the sensor noise to compute the data uncertainty and Monte Carlo samples to compute the model uncertainty. This framework outperformed other state-of-art methods for uncertainty estimation in various tasks like End-to-End Steering Angle Prediction, Object Future Motion Prediction, Object Recognition and Closed-Loop Control of a Quadrotor. The main limitation of the framework is that in order to generate the total uncertainty it needs multiple network forward passes for each input. Another approach is taken in [17] where the authors show that the use of dropout in inference time is a Bayesian approximation of the Gaussian processes and build a probabilistic interpretation of dropout allowing to obtain model uncertainty out of existing deep learning models.

An active learning based approached is proposed in [18] to learn the kinematic chains of a six degrees-of-freedom (dof) arm from the humanoid robot Baltazar. The learning process is done using Recursive least squares, which also provides the estimation of the posterior uncertainty of the parameters that are used in the active learning algorithms to compute the most informative observations. They verified that this framework outperformed the baseline methods in both simulation and in experiments with the real robot by obtaining faster results. In a similar fashion, Cunha et al [1] propose a cost sensitive active learning approach to estimate the Denavit-Hartenberg (DH) parameters for the 7Dof arm of humanoid iCub robot to prioritise movement efficiency. The method adds a term to the cost function of active learning to penalise the movement in between the previous observation and the next most informative one. The main objective is to try to improve the performance of the calibration for humanoid robot's arm to reduce the arm movement comparing to a traditional active learning method without compromising learning accuracy of the body schema. The use of these methods reduces the arm movement by half while maintaining iteration performance. However, both methods lack the flexibility provided by neural networks.

2.3 Neural Networks with Active Learning

The combination of active learning with neural networks is expected to reduce the amount of data used for training while maintaining the learning performance, having a great research potential since it is expected to improve the overall computational performance for the training phase. Ren et al [19] gives a general overview of the frameworks, challenges and applications of combining Deep Learning, which requires a huge amount of data to train models and extract high-quality features, with Active Learning, which uses the least amount of data to try to maximize the model's performance, into Deep Active

Learning that has a huge potential of improving the labeling cost of DL while maintaining its strong learning ability.

Zhou et al [20] proposes an active trajectory generation framework to train Deep Neural Network (DNN) to represent the robots inverse dynamic model. They identify the most informative DNN output, using the uncertainty estimates of the DNN to send to the system, encouraging it to explore where the learning model is uncertain. They show that this method is faster to train the DNN, requiring a lower number of training trajectories to learn the inverse model.

In [21] the authors use neural networks to plan the paths for both a mobile robot and a multi-joint robot manipulator while avoiding collision with static and moving obstacles. Similarly, [22] presents a risk-aware active learning procedure, allowing the use of booth trajectory-optimization and deep learning software. This process assigns higher weight to dangerous regions, so that they can be trained by the DNN to optimize the trajectories of a quadcopter. Experimentally, they observed that the Neural Network approximations are 50 times faster than the trajectory optimizer, without neglecting the safety requirements. They also concluded that the procedure avoids collisions for multiple moving objects.

Following the work of Gal et al [17], Tsymbalov et al [11] proposed an active learning algorithm for regression problems based on uncertainty estimation from stochastic dropout outputs of the neural network. Making use of Monte-Carlo Dropout Uncertainty Estimation (MCDUE) they create an AL framework using dropout at inference time to generate stochastic predictions to estimate the variance of said predictions, evaluating the uncertainty of unlabeled data. The proposed approach outperformed methods like random sampling and max-min baseline.

3

Methods

Contents

This chapter's main goal is to provide an in-depth review of the methodologies used in this thesis. It is divided into two main sections: the robots model and the learning process.

Robots use mathematical models to describe their movement in order to interact with the environment. Therefore, it is important that the models are well defined in order to improve the robots performance. Section 3.1 is responsible for specifying the key elements and parameters to model the robot. The section starts by defining the main reference frames for the robot and outlining the process for computing the robots kinematics which are responsible for establishing its motion. Sections 3.1.3 and 3.1.4 provide two conventions for obtaining the the end-effectors pose. The first is Denavit-Hartenberg convention that, as explained in chapter 4, is used to build the dataset to train our model. The latter describes quaternions and how they can be used to express rotations. These are used to represent the orientation of the end-effector.

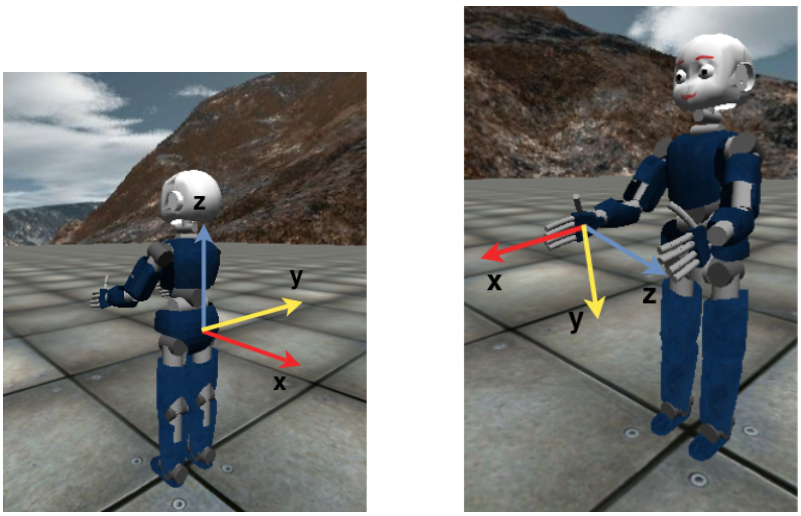
Now that the techniques for modelling the robot were defined, one needs to specify how the robot is going to learn this model. Section 3.2.1 introduces neural networks as the core learning technique for the robot model. It also defines Adam as the optimization method for training the neural networks along with specifying the key elements of its initialization and update steps. Additionally, it references dropout as regularization method and introduces the idea of using dropout for model uncertainty estimation. Finally, in order to enhance the learning process, section 3.2.4 introduces various Active learning techniques based on diversity and uncertainty estimation.

3.1 Robots Model

3.1.1 Reference Frames

A rigid body is described in space by its position and orientation with respect to a reference frame. So, in order to model our robotic arm, two reference frames are defined: one to define the pose of any object in reference to the robots root reference frame and the other to define poses with respect to the robots right hand. The first is defined as the root reference frame $\{R\}$, composed by the orthogonal axis $\{x_R, y_R, z_R\}$, whose origin is located in the middle of the robot in between the two legs. The second is defined as the end-effector frame, $\{E\}$, whose origin coincides with the center of the palm of the robots right hand, composed by the orthogonal axis $\{x_E, y_E, z_E\}$. Regarding the root frame $\{R\}$, the z_R axis is parallel to gravity but points up, towards the head, the x_R axis points behind the robot and the y_R points laterally and is chosen according to the right hand rule. Regarding the end-effector frame $\{E\}$, the x_E axis points in the direction of the middle finger, z_E axis towards the front of the palm of the hand, while the y_E axis is chosen similarly to y_R.

For better visualisation of both the frames, previously described, Figure 3.1 depicts these frames and how their axis are disposed.



(a) Robot root reference frame, $\{R\}$

(b) End-effector reference frame, $\{E\}$

Figure 3.1: Reference Frames.

3.1.2 Kinematics

A robot manipulator, such as the robotic arm used in this thesis, is composed of rigid bodies affixed by joints that can be either prismatic or revolute. So the robotic arm can be seen as a kinematic chain connecting, in one end, the root frame, $\{R\}$, and, in the other end, a movable end-effector,$\{E\}$, that allows the robot to interact with the environment. Since the movement of this end-effector is a conjoint effort of the movement of the individual joints that compose this chain, in order to manipulate objects in the environment, the relationship between the joint actuation and the position and orientation of the end-effector must be known. This section is based on [23] and [24], where more detailed information can found, regarding kinematics.

Robot kinematics studies the motion of the robot with respect to a fixed reference frame, ignoring external forces or torques that cause the movement of the robot. In other words, the kinematics of a robot define the position and orientation of the end-effector as the function of the robots joint configurations and vice versa. Robot kinematics are divided into two main problems: forward kinematics and inverse kinematics. Their relationship is represented in Figure 3.2. The forward kinematics are easier to solve and maps the pose of the end-effector as a function of the joint configuration, with respect to the root reference frame. This means that a joint configuration corresponds to a single pose of the robots hand. While the inverse kinematics are much more difficult to solve due to singularities and deals with the inverse problem. In this case there can be multiple solutions since the same pose can usually be obtained with different joint configurations. The number of these solutions can vary with the number

of joints. In this thesis the focus will be on forward kinematics of iCub right arm, composed of seven revolute joints.

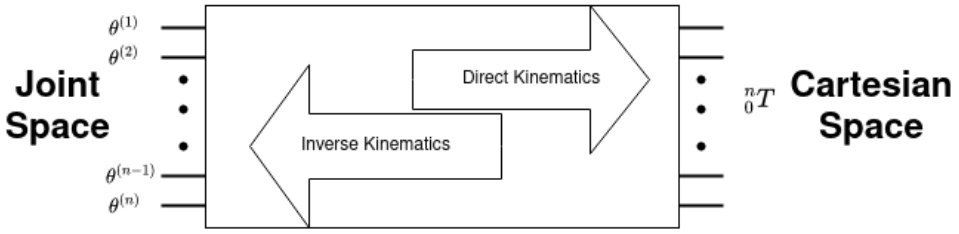


Figure 3.2: Schematic representation of the relationship between direct and inverse kinematics

Considering a point, p, in the Cartesian space that represents a rigid body, if we consider two different frames, for example $\{R\}$ and $\{E\}$, the point's coordinates will be different depending on which frame we are taking into consideration. This should be intuitive if we consider, for example, that the robot has this point in its hand, with its arm fully stretched, where its pose with respect to $\{E\}$ coincides with the origin of the reference frame, while with respect to $\{R\}$ it does not. The transformation of the points pose from one frame to the other can be obtained by applying a translation and a rotation. The relation between the pose of the point with respect to $\{R\}$, ${}^{R}p$, and the pose of the point with respect to $\{E\}$, ${}^{E}p$, is given by

$$ {}^{E}p = {}^{E}_{R}P + {}^{E}_{R}R\,{}^{R}p \tag{3.1} $$

where ${}^{E}_{R}P$ is the position of the origin of frame $\{R\}$ with respect to frame $\{E\}$, representing the translation between the frames, and ${}^{E}_{R}R$, is the rotation matrix, responsible for the rotation between the frames, given by

$$ {}^{E}_{R}R = \left[{}^{E}_{R}\mathbf{x} \quad {}^{E}_{R}\mathbf{y} \quad {}^{E}_{R}\mathbf{z} \right], \tag{3.2} $$

where ${}^{E}_{R}\mathbf{x}$, ${}^{E}_{R}\mathbf{y}$ and ${}^{E}_{R}\mathbf{z}$ contain the description of the 3 coordinate axis of frame $\{R\}$ in frame $\{E\}$. This transformation is represented in Figure 3.3.

Instead of these two operations we can simplify it into a single transformation given by

$$ {}^{E}\tilde{p} = {}^{E}_{R}T\,{}^{R}\tilde{p}, \tag{3.3} $$

using the homogeneous representation of p:

$$ \tilde{p} = \begin{bmatrix} p \\ 1 \end{bmatrix}. \tag{3.4} $$

So ${}^{E}_{R}T$ becomes a 4×4 matrix given by

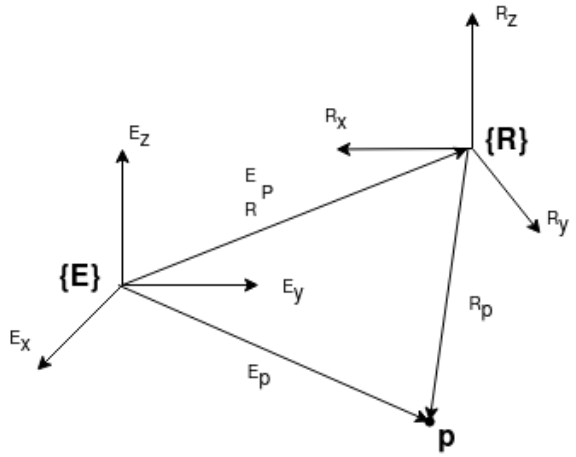


Figure 3.3: Representation of point p in different reference frames

$$ {}^{E}_{R}T = \begin{bmatrix} {}^{E}_{R}R & {}^{E}_{R}P \\ \mathbf{0^T} & 1 \end{bmatrix}. \tag{3.5} $$

As it was said before, a robot manipulator is composed of multiple joints connected by links, so, if we consider that there is a reference frame in each individual frame, we can compute the transformation between the root frame of the robot and the frame of its end-effector as the sequence of transformations between the frames of sequential joints. Considering the kinematic chain, of a robot manipulator with n joints, numbered from 1 to n and where 0 represents the root frame of the robot, it is important to note the following relationships

$$ {}^{1}\tilde{p} = {}^{1}_{2}T\,{}^{2}\tilde{p} $$
$$ {}^{0}\tilde{p} = {}^{0}_{1}T\,{}^{1}\tilde{p} $$
$$ {}^{0}\tilde{p} = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}\tilde{p} $$

then

$$ {}^{0}_{2}T = {}^{0}_{1}T^{1}_{2}T. \tag{3.6} $$

So the transformation matrix between any two joints is given by the multiplication of the transformation matrix of sequential joint, which is represented in Figure 3.4. Thus, the transformation matrix between joint n and the root reference frame is given by

$$ {}^{0}_{n}T(\theta) = {}^{0}_{1}T(\theta^{(1)}){}^{1}_{2}T(\theta^{(2)})...{}^{n-2}_{n-1}T(\theta^{(n-1)})\,{}^{n-1}_{n}T(\theta^{(n)}), \tag{3.7} $$

where $\theta = \left[\theta^{(1)}, \theta^{(2)}, ..., \theta^{(n-1)}, \theta^{(n)}\right]$ is the vector containing the values of each joint.

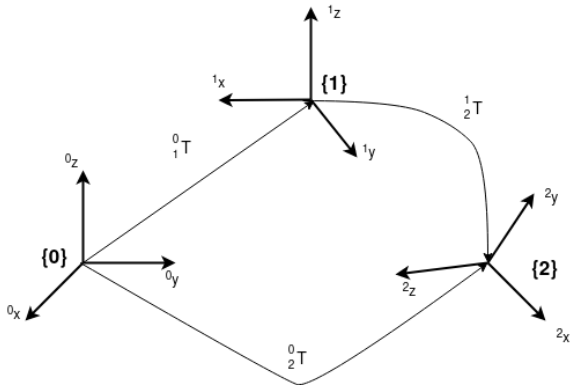


Figure 3.4: Representation of a sequence of transformations between difference reference frames

3.1.3 Denavit-Hartenberg Convention

The Denavit-Hartenberg Convention provides a consistent way of computing the transformation matrix between two consecutive joints in a kinematic chain only using 4 parameters. This notation allows a description of the manipulator that can be used in different algorithms to find the kinematics solution and it requires the reference frames to be attached to the joints. For more information about this topic, see [24].

Once all reference frames are establish, the pose of frame i with respect to frame $i-1$ is defined by the following parameters:

a_{i-1} Distance between the origins of the reference frames or, in other words, length of the link.

α_{i-1} Angle between axis z_{i-1} and z_i around axis x_i or, in other words, link twist.

d_i Coordinate of the origin of the reference frame in joint i, along z_{i-1} or, in other words, link offset.

θ_i Angle between axis x_i and x_{i-1} around z_i axis or, in other words, joint angle.

Knowing these parameters is enough to build a transformation matrix for a transformation between links i and $i+1$,

$$ {}^{i-1}_{i}T = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8} $$

where s_{α_i} and c_{α_i} represent, respectively, the sine and cosine of α_i. It is also important to note that the parameter o_i is the only variable depending on the type of joint.

3.1.4 Quaternions

Quartenions are a four-dimension extension of the complex numbers. So, in a similar fashion to the complex numbers, quaternions are composed by a real and an imaginary part and can be represented

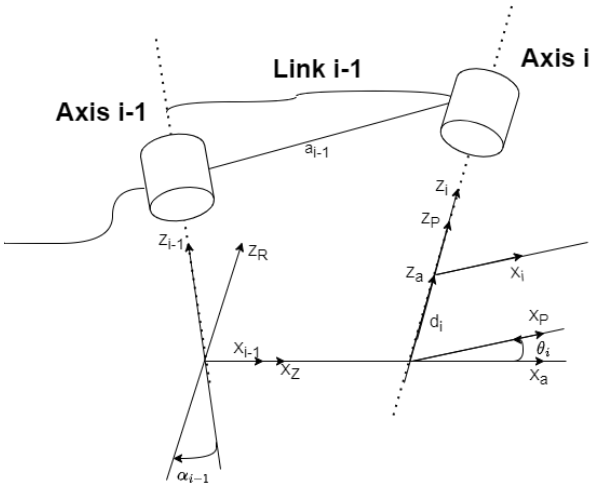


Figure 3.5: Denavit-Hartenberg parameters for two sequential joints in a kinematic chain.

as

$$q = q_0 + \boldsymbol{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}, \tag{3.9}$$

where q_0 is a scalar, composing the real part, and $\boldsymbol{q} = (q_1\boldsymbol{i}, q_2\boldsymbol{j}, q_3\boldsymbol{k})$ is a vector that represents the imaginary part. So i, j and k are imaginary numbers that respect the following rule:

$$\boldsymbol{i^2} = \boldsymbol{j^2} = \boldsymbol{k^2} = \boldsymbol{i\,j\,k} = -1. \tag{3.10}$$

The multiplication of quaternions is associative but not commutative and besides property (3.10), it also respects the following properties

$$\begin{aligned} \boldsymbol{i\,j} &= \boldsymbol{k} = -\boldsymbol{j\,i}, \\ \boldsymbol{j\,k} &= \boldsymbol{i} = -\boldsymbol{k\,j}, \\ \boldsymbol{k\,i} &= \boldsymbol{j} = -\boldsymbol{i\,k}. \end{aligned} \tag{3.11}$$

The quaternion multiplication is the base to representing rotation using unit quaternions, which will be explained in the next section. So, considering the previous quaternion q and another quaternion

$$p = p_0 + \boldsymbol{p} = p_0 + p_1\boldsymbol{i} + p_2\boldsymbol{j} + p_3\boldsymbol{k}, \tag{3.12}$$

the multiplication of quaternions can be expressed using the cross product and the inner product as the following compact form

$$pq = p_0q_0 - \boldsymbol{p}\cdot\boldsymbol{q} + p_0\boldsymbol{q} + q_0\boldsymbol{p} + \boldsymbol{p}\times\boldsymbol{q} \tag{3.13}$$

And, since a quaternion can be interpreted as a complex number with three imaginary variables instead of one, the complex conjugate of q can be defined as

$$q^* = q_0 - \boldsymbol{q} = q_0 - q_1\boldsymbol{i} - q_2\boldsymbol{j} - q_3\boldsymbol{k} \tag{3.14}$$

For a deeper understanding on the algebra behind quaternions see [25] and [26].

3.1.4.A Quaternions as a rotation representation

Even though quaternions are four-dimension vectors, they can be used to represent three-dimension rotations. They even show some advantages over rotation matrices and Euler angles. First, they require less memory, in opposition to a 3×3 rotation matrix, since a quaternion can represent the rotation using only 4 scalars. Its computation is also faster since, for composing two rotations, the quaternions only need to compute 16 scalar multiplications while using rotation matrices requires 27. Quaternions also avoid problems like Gimbal lock, that can occur when using Euler angles, which causes the loss of one DoF when two of the rotation axis are parallel. Finally, quaternions can also avoid the issues of numerical precision and normalization that arise when trying to interpolate between two rotation matrices. This section is based in [26],where more detailed information can be found regarding this topic.

Consider a unit quaternion, q, where

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1, \tag{3.15}$$

this implies that there is an angle, $\theta \in [0, \pi]$ that satisfies the following properties

$$cos^2(\theta) = q_0^2 \tag{3.16}$$

$$sin^2(\theta) = \|\boldsymbol{q}\|^2 \tag{3.17}$$

this also means that $cos(\theta) = \pm q_0$ and $sin(\theta) = \|\pm\boldsymbol{q}\|$. Allowing us to write the quaternion as the Euler's formula for complex numbers

$$q = cos(\theta) + \boldsymbol{u}sin(\theta), \tag{3.18}$$

where $\boldsymbol{u} = \frac{\boldsymbol{q}}{\|\boldsymbol{q}\|}$ is an unit vector and can be interpreted as the axis around which the rotation will occur, while θ is the rotation angle.

Now lets consider any vector $\boldsymbol{v} \in \mathbb{R}^3$ and any unit quaternion

$$q = cos(\frac{\theta}{2}) + \boldsymbol{u}sin(\frac{\theta}{2}),$$

we define the operator

$$L_q(\boldsymbol{v}) = q\boldsymbol{v}q^* \tag{3.19}$$

$$= (q_0^2 - \|\boldsymbol{q}\|^2)\boldsymbol{v} + 2(\boldsymbol{q}\cdot\boldsymbol{v})\boldsymbol{q} + 2q_0(\boldsymbol{q}\times\boldsymbol{v}) \tag{3.20}$$

where q^* is the complex conjugate of q. Applying this operator to v is equivalent to rotating the vector around $\boldsymbol{u}$ with a rotation angle of θ. So this allows us to represent rotations with a single unit quaternion.

It is also important to note that any given rotation can be represented by two quaternions. If we have a quaternion $\boldsymbol{q}$, that represents a given rotation, the $-\boldsymbol{q}$, will also represent the same rotation. So, in this thesis, we always consider quaternions with positive q_0.

3.2 Learning Model

3.2.1 Neural Networks

Neural Networks are universal estimators that can be modelled to generate predictions or classify information in countless research fields by attempting to build learning models that simulate the human brain. NN are composed by multiple layers of interconnected nodes and can have many shapes and sizes depending on the tasks they are designed to solve. These tasks are separated into two main groups, regression and classification tasks. Classification tasks are machine learning problems that involves the categorization or classification of information into predefined classes based on features from the input data. An example of a classification task consists of having pictures of either cats or dogs and classifying the ones with dogs and with cats. Regression tasks are problems where the model is intended to predict continuous numerical values based on the inputs. On contrary to classification tasks, that intend to assign data points to specific classes, in regression tasks the goal is to find a relationship between the inputs and the outputs. A good example of a regression task is predicting housing prices based the on property features like its area. Since this thesis focuses on direct kinematics, or in other words, predicting the position and orientation of the end-effector based on the joint configurations, the problem it aims to solve is a regression task. There are multiple architectures of NN like Recurrent neural networks (RNN), useful for time series forecasting, or Convolutional Neural Network (CNN), mainly used in image classification, but this thesis focuses on Multilayer Perceptrons (MLP).

MLP are feedforward NN. They're also the most basic type of NN usually composed of an input layer, any number of hidden layers and an output layer. It is important to note that any layer can have multiple nodes, depending on the task at hand. This type of NN can solve both classification and regression

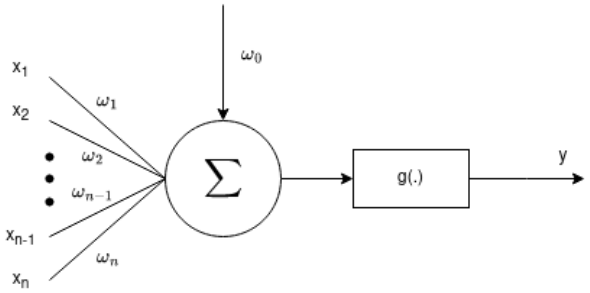


Figure 3.6: Representation of a node or neuron.

tasks. Lets consider a single node with n inputs, represented in Figure 4.2, its output is given by

$$y = g\left(\omega_0 + \sum_{i=1}^{n} x_i \omega_i\right) \tag{3.21}$$

where $g(\cdot)$ is the activation function, ω_0 is the bias, x_i and ω_i are, respectively, the input and the weight i with $i = 1, ..., n$. Since the MLP is composed by multiple nodes, it represents a nonlinear transformation from the input space to the output space $y = f(x, \omega)$ controlled by the weights ω. Considering that the inputs are the joint configurations and the outputs are the orientation and position of the robots hand, then a MLP can be trained to solve the kinematics of a robot. It is important to note that this is possible because the kinematic model of a robot is and injective function. Instead learning the DH parameters of the robot, a NN can simply learn its weights. This is represented in Figure 3.7.

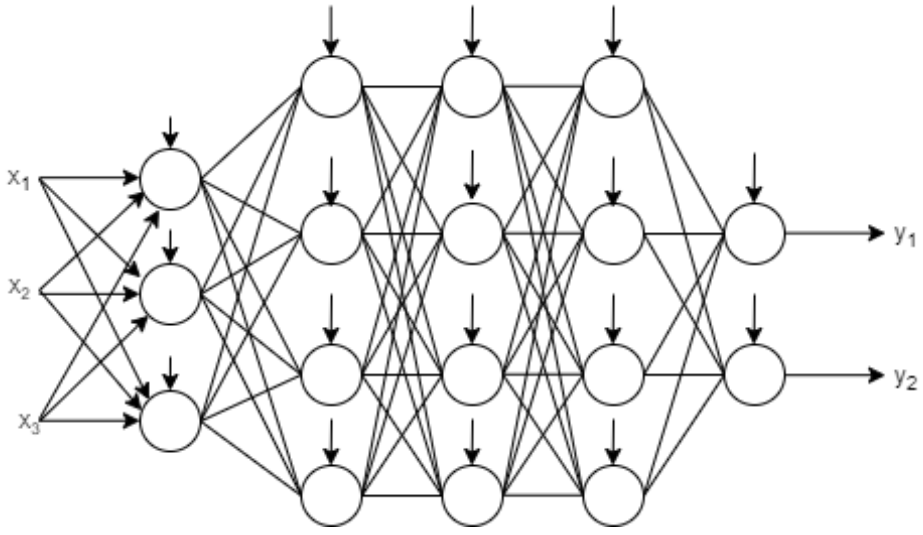


Figure 3.7: Example of a MLP with three inputs and two outputs. It has three hidden layers with four nodes each. The vertical arrows represent the bias of each node.

In order to obtain a good estimator for a specific task, the model needs to train its weights. Since this thesis focuses on supervised learning, a training set containing labeled data is used for training. This

training set will be used to minimize a given loss function. In this thesis the lost function is the L1 loss, given by

$$L(y, \widehat{y}) = |y - \widehat{y}|\,, \tag{3.22}$$

where y is the ground truth, from the training set, and $\widehat{y}$ is the prediction of the model for the input, x, corresponding to y. This means that a training set containing n samples can be defined as

$$T = \left\{ \left(x^{(k)}, y^{(k)} \right), k = 1, ..., n \right\}. \tag{3.23}$$

Most optimization methods used to update the weights use the gradients of the loss function with respect to the networks weights. The Backpropagation algorithm can compute these gradients. Considering the single node in Figure 4.2, and the following equations

$$\begin{aligned} s &= \omega_0 + \sum_{i=1}^{n} \omega_i x_i \\ y &= g(s), \end{aligned} \tag{3.24}$$

by applying the chain rule of differentiation, the gradient is given by

$$\frac{\partial L}{\partial \omega_i} = \frac{\partial L}{\partial s} \frac{\partial s}{\partial \omega_i} = x_i \epsilon, \tag{3.25}$$

where

$$\epsilon = \frac{\partial L}{\partial s} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial s} = g'(s) \frac{\partial L}{\partial y} \tag{3.26}$$

3.2.2 Adam optimizer

Adam, explained in detail in [27], is an extension of stochastic gradient descent. It is a stochastic optimization algorithm that only requires first-order gradients with little memory requirements. It is also straightforward to implement and computationally efficient. Adam combines the advantages of Adaptive Gradient Algorithm (AdaGrad), which improves the performance on problems with sparse gradients, and Root Mean Square Propagation (RMSProp), which works well on online and non-stationary problems, making it a very popular and widely used optimization algorithm for training machine learning models.

3.2.2.A Initialization

Adam uses exponential moving averages of the gradient (m) and the squared gradient (v) to update the parameters, the weights of the neural network, at a given time step, θ_t. These moving averages can also be denominated as the first and second moment vectors, respectively, and should be initialized with a vector of zeros. The time step, t, should also be initialized as zero.

Adam uses four hyperparameters that need to be defined before the update step.

α The learning rate controls the step size for parameters update.

β_1 Exponential decay rate for the first moment estimates (m). $\beta_1 \in [0, 1)$

β_2 Exponential decay rate for the second moment estimates (v). $\beta_2 \in [0, 1)$

ϵ Small constant for numerical stability

Good default setting for these hyperparameters are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The values were tested in various machine learning problems by the authors in [27].

3.2.2.B Update

The update stage starts with updating the time step, $t = t + 1$. Then, considering $f(\theta)$ the stochastic objective function with parameters θ, or in other words, the loss function of the NN with parameters θ, the algorithm computes the gradients of the objective function at the given time step using back propagation defined in (3.25).

$$g_t = \nabla_\theta f_t(\theta_{t-1}) \tag{3.27}$$

The following step is to update the biased moments estimate for both moments, following

$$\begin{aligned} \widehat{m_t} &= \frac{m_t}{1 - \beta_1^t} \\ \widehat{v_t} &= \frac{v_t}{1 - \beta_2^t}, \end{aligned} \tag{3.28}$$

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \end{aligned} \tag{3.29}$$

then compute the bias-corrected moments, $\widehat{m_t}$ and $\widehat{v_t}$, following

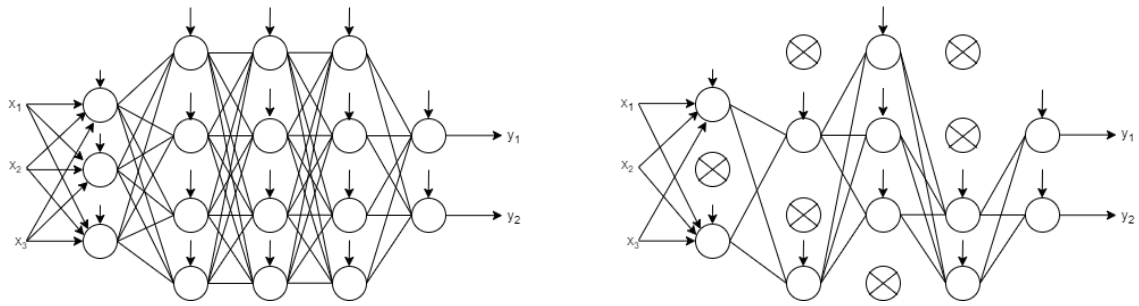
$$\begin{aligned} \widehat{m_t} &= \frac{m_t}{1 - \beta_1^t} \\ \widehat{v_t} &= \frac{v_t}{1 - \beta_2^t}, \end{aligned} \tag{3.30}$$

finally, the update step is given by

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon} \tag{3.31}$$

3.2.3 Dropout

Dropout is a regularization technique for addressing the problem of overfitting when training neural networks. The main idea behind it is to randomly drop some nodes, and their connections, during training in order to prevent them to co-adapt. This also builds a big ensemble of ”thinner” NN that can have their predictions averaged by using the main neural network at test time. So, dropout possesses two main advantages. It prevents overfitting and provides an efficient way of approximately combining multiple different NN architectures. The latter, in most cases, improves the models efficiency. Figure 3.8 presents a visual representation of dropout. For a more detailed explanation about this method see [28].



(a) Neural network before applying dropout

(b) Neural network after applying dropout

Figure 3.8: Dropout NN model. **Left**: Standard NN model with 3 hidden nodes. **Right**: Example of a ”thinned” network after applying dropout to the network on the right. The crossed out nodes represent neurons that have been dropped.

The nodes that are randomly going to be dropped are selected randomly with a dropout rate p, which is the probability of retaining a node. This hyper parameter is usually defined per layer of the network, meaning all the nodes in a specific layer have the same probability to be dropped but different layers can have different dropout rates. Note that dropout can be applied to all layers, except the output layer. It is also important to note that dropout should only drop nodes during training time and not during test time, otherwise the model would not benefit from the ensemble of networks used to train the main network. As it will be explained in section 3.2.4.C, dropout can be used at test time to estimate model uncertainty.

3.2.4 Active Learning

Active learning is a decision-making strategy used to improve the efficiency of learning processes by aiming to select the most useful samples from unlabeled data and hand it over to an oracle (e.g., human annotator) for labeling, so as to reduce the cost of labeling as much as possible while still maintaining performance [19]. There are three main AL approaches based on how these samples are generated. Pool based AL chooses the samples from a pool of unlabeled data. Stream based AL receives samples

or small batches of samples sequentially and decides whether to label or discard it. Finally, Query-synthesis based AL generates its own queries for labelling. This section focuses on both pool based AL and query-synthesis based AL, since this thesis compares the application of both these methods to learning the kinematics of iCubs right arm.

There are multiple criteria for deciding what samples are more informative, for example, model uncertainty [11], diversity [2, 3], query-by-committee [29] and expected model change [30]. The choice of criteria is important when trying to solve an AL problem, since choosing the wrong sampling method can lead to worse performance than randomly selecting samples. This thesis focuses on uncertainty and diversity based approaches.

Stating the problem for both query-synthesis and pool based AL, one must first define a proper utility function, $C(\cdot)$, to choose the most informative samples. In this specific problem, the sample is a joint configuration, θ, that, using the real kinematics of iCub, will be labeled, generating the corresponding pose of the end-effector. The same utility functions will be used in both approaches. First, considering the Query-synthesis approach, the next joint configuration, θ^*, chosen to be queried to train the NN is computed following

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{argmax}\ C(\boldsymbol{\theta}). \tag{3.32}$$

In the case of pool based AL, the computation of the next joint configuration is very similar. But first, one need to define a pool with N joint configurations

$$\boldsymbol{P} = \left\{ \left(\boldsymbol{\theta^{(n)}} \right), n = 1, ..., N \right\}. \tag{3.33}$$

The next joint configuration is given by

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \boldsymbol{P}}{argmax}\ C(\boldsymbol{\theta}). \tag{3.34}$$

It is important to note that the random selection baseline selects samples using a random uniform distribution.

3.2.4.A Greedy Sampling on Input

Greedy Sampling on Input (GSx) is proposed by [2] for regression problems. It is a passive sampling technique that does not require model predictions to select the next sample. Instead, it selects the samples purely based on its geometric characteristics in the feature space. So it is completely independent from the regression model and has low computational cost.

Being a greedy approach, it chooses the sample which is farthest away from the previously selected

samples. It is important to consider that this approach was mainly designed for pool based AL. So, let us consider that K samples have already been selected, stored in a dataset $\boldsymbol{T}$ and removed from the pool. To select the next sample the algorithm starts by computing the distances from all the previously selected points θ_k to all the samples in the pool θ_n.

$$d_{nk}^{x} = dist\left(\theta_k, \theta_n\right), \tag{3.35}$$

where $k = 1, ..., K$, $n = 1, ..., N - k$ and $dist(\cdot)$ defines the distance. The x in d_{nk}^{x} means that these distances are on the input space. This thesis uses the euclidean distance:

$$dist\left(\theta_k, \theta_n\right) = \left\|\theta_k - \theta_n\right\|, \tag{3.36}$$

then computes the shortest distances from the pool to the already selected samples:

$$d_{n}^{x} = \min_{k} d_{nk}^{x} \tag{3.37}$$

finally, GSx chooses the samples with the largest distance to the selected samples:

$$\theta^{*} = \max_{n} d_{n}^{x} \tag{3.38}$$

Summing up, GSx encourages diversity among the selected samples by choosing the samples that are the farthest away from the already selected samples. The pseudo-code for this algorithm applied to pool base AL is represented in 3.1. It is important to note that this method can also be applied to query-synthesis based AL by selecting samples from the whole robot workspace using an optimization algorithm.

Algorithm 3.1: GSx algorithm adapted from [2]

Data: Unlabeled pool dataset P, labeled dataset T, sample size K.
Result: Unlabeled batch B of size K.
$B = \emptyset$. Let k be the number of the current iteration. $k = 0$;
for $k = 0, ..., K$ **do**
  Calculate the minimum distance between each sample in P and $\left\{T \cup B\right\}$ using (3.37);
  Select the sample from P with the largest minimum distance x_k;
  $P = P - \left\{x_k\right\}$;
  $B = B \cup \left\{x_k\right\}$

3.2.4.B Improved Greedy Sampling

Improved greedy sampling Improved Greedy Sampling (iGS) was proposed in [3] and considers not only the diversity in the input space, provided by GSx, but also diversity in the output space. GSx does not take into consideration how important the features are. When considering the diversity of the outputs, iGS aims to consider feature weighting while avoiding when this weighting proves unreliable. Similarly to GSx, iGS was devised for pool based AL.

Considering a similar setup to the one in section 3.2.4.A but also considering that the already selected features were labeled, iGS starts to compute the distances from all the previously selected features to all the samples in the unlabeled pool following (3.35). Then it computes the distances between labels of the selected samples y_k and the predictions from the regression model $f(\theta_n)$ of all the samples in the labeled data

$$d_{nk}^{y} = \|\mathbf{y}_k - f(\theta_n)\|\,, \tag{3.39}$$

where $\mathbf{y}_k$ is the label of the $k$ith selected feature and $f(\theta_n)$ is the prediction of the regression model to the sample n from the pool. Then it computes d_n^{xy} following

$$d_n^{xy} = \min_k \, d_{nk}^{x} d_{nk}^{y}. \tag{3.40}$$

Finally, iGS selects the configuration with maximum d_n^{xy}:

$$\theta^* = \max_n \, d_n^{xy} \tag{3.41}$$

In summary, iGS chooses the farthest sample from the previously selected in both the input and the output to achieve a well-balanced diversity in the chosen samples. It is important to note that this approach requires an initial regression model to be implemented. The pseudo-code for this algorithm applied in pool based AL is represented in 3.2.

3.2.4.C Uncertainty based AL - Mc Dropout

As referenced before, dropout can be used to compute model uncertainty. Mc Dropout combines the ensemble building characteristics of dropout with Monte Carlo sampling to estimate the uncertainty of a NN model. By using dropout at inference time one generates stochastic predictions that can be used to compute various statistics such as mean and variance of said predictions. For a better understanding on how to use dropout to compute model uncertainty, see [17] and [11].

As it was shown in [17], dropout can be seen as a Bayesian approximation. So in practice this means that one can perform T stochastic forward passes through the NN model, for the same input, while using dropout. This will generate various slightly different predictions. One can determine the

Algorithm 3.2: iGS algorithm adapted from [3]

Data: Unlabeled pool dataset P of size N, labeled dataset T, sample size K.
Result: Labeled batch B of size K.
$B = \emptyset$. Let k be the number of the current iteration. $k = 0$;
for $k = 0, ..., K$ **do**
  for $n = k, ..., N$ **do**
    Calculate the distances between each sample in P and $\{T \cup B\}$ using (3.35);
    Calculate the distances between the prediction from the regression model for each sample in P and the labels of $\{T \cup B\}$ using (3.39);
    Compute d_n^{xy} using (3.40);
  Select the sample, x_k, from P with the highest value of d_n^{xy}, ;
  $P = P - \{x_k\}$;
  Label the x_k following the process described in section 4.2.1.A to obtain y_k;
  $B = B \cup \{(x_k, y_k)\}$;

models uncertainty of an output i by computing the standard deviation of these predictions.

$$\sigma^{(i)} = \sqrt{\frac{1}{T-1}\sum_{k=1}^{T}(y_k^{(i)} - \overline{y}^{(i)})^2}, \tag{3.42}$$

where y_k is the k stochastic prediction and $\overline{y}$ is the mean of these predictions and $i = 1, ..., 7$ is the number of outputs of the model. This method for uncertainty estimation was introduced in [11] and it is called Monte-Carlo Dropout Uncertainty Estimation (MCDUE). So, if a sample has a high uncertainty, means that the model is less confident about its prediction of that sample. Thus, the next joint configuration to be sampled should have the highest uncertainty so the expected improvement should be as large as possible.

4

Experimental Setup

Contents

This chapter offers a review of the implementation details of this thesis. The use of simulation in robotics is vital for testing and validating different approaches for specific robot models. So section 4.1 introduces the simulator of iCub used in this thesis. Then, since this thesis focuses on how AL can improve the learning capabilities of neural networks, sections 4.2 and 4.3 present the implementation details of the NN model and the AL approaches, respectively. The first, starts by introducing the library used to implement the model. Then it describes the process of building labeled and unlabeled datasets and explains the labeling process of unlabeled data. Afterwards, it focuses on the architecture of the NN model and introduces the hyperparameters used during training. Finally, it shows the initialization of the model by displaying how the NN is pre-trained. Section 4.3 starts by explaining implementation particularities of each of the AL approaches used. Afterwards it explains the process behind the construction of the pool datasets for pool based AL. Since query-synthesis based AL requires a global optimization method for non-differentiable functions, section 4.3.2 introduces the DIRECT optimization algorithm, explaining in detail its main steps.

In order to compare the different approaches used in this thesis, one needs comparison metrics. So, section 4.4 introduces the average position error, to analyze the performance of the proposed approaches when learning the position of the end-effector, and the angular error to compare the learning process of the orientation. Finally, section 4.5 presents the calibration framework for the different AL approaches, both for pool and query-synthesis based AL.

4.1 iCub Simulator

All the work presented in this thesis is based on simulations of the humanoid robot iCub. The open-source simulator, presented in [31], is used in this thesis to implement our approach. The simulator is an accurate representation of the real iCub. It replicates its sensors, actuators and dynamics, being able to accurately represent the robot's motion for various joint configurations while respecting the joints' range of motion.The limits that represent this range of motion are summarized in table 4.1. It also provides a visual representation of the robot and its movements. This representation can be visualized in Figure 4.1.

The simulator uses the same software infrastructure and process of communication as the physical robot. It uses the middleware YARP [32] (Yet Another Robot Platform) to create a bridge between the user and the simulator. It is responsible for connecting the data, introduced by the user, with the simulator via multiple connection types (tcp, udp, multicast, local, MPI, mjpg-over-http, XML/RPC, tcpros, ...). The iCub library[1] offers interfaces to interact with simulator either by issuing commands or receiving information. It reads proprioceptive information about the joints and enables the transmission of com-

[1] https://github.com/robotology/icub-main

Figure 4.1: Visual representation of iCub's simulator.

mands to move each of the individual joints of the iCub's kinematic chains. The DH parameters for iCubs right arm kinematic chain are summarized in table 4.2. It is important to note that only the links 3 to 9 compose the right arm, since links 0 to 2 are part of the robots torso so they're stationary. But all nine links are part of the kinematic chain that connects the robots root frame to its end-effector frame.

Table 4.1: Joint limits for iCubs right arm.

Link	0	1	2	3	4	5	6
Lower bound [°]	-94.5	0	-36.2	15.3	-90	-90	-19.8
Upper Bound [°]	9.4	160.8	79.5	105.8	90	0	39.6

Table 4.2: Denavit-Hartenberg parameters for the iCubs right arm kinematic chain. These are the parameters for the simulator.

Link	0	1	2	3	4	5	6	7	8	9
a_i $[mm]$	32	0	-23.3647	0	0	-15	15	0	0	62.5
d_i $[mm]$	0	-5.5	-143.3	-107.74	0	-152.28	0	-137.4	0	16
α_i [°]	90	90	90	90	-90	-90	90	90	90	0
θ_i [°]	0	-90	-105	-90	-90	-105	0	-90	90	180

4.2 Neural Network Implementation

The PyTorch C++ API[2] was used to implement the neural network used in this thesis. This API provides a C++ library for GPU and CPU tensor computation and automatic differentiation for gradient computation.

[2]https://pytorch.org/cppdocs/

It also provides high level building blocks for neural network applications, being used for state of the art machine learning research in high performance environments. It is also important to note that C++ can help improve the performance of ML models by running them faster and more efficiently.

4.2.1 Dataset Construction

Neural networks use datasets with labeled data to train their models. It is important that these datasets are well constructed with the correct labels. All the data used in this thesis was computationally generated with the purpose of learning the forward kinematics of the iCub. So, this subsection focuses on the process behind the assembly of said datasets and on how to label each sample of these datasets.

Firstly, it is important to define that there are two types of dataset used in this thesis: unlabeled and labeled. The first contains only the joint configurations of the robot, while the latter also contains the corresponding pose of the end-effector. Considering an unlabeled dataset with n samples, one can define it following

$$T = \left\{ \left(\boldsymbol{x}^{(k)} \right), k = 1, ..., n \right\}, \tag{4.1}$$

where $\boldsymbol{x}^{(k)}$ is a joint configuration sample from the dataset. A labeled dataset is defined in (3.23).

To construct an unlabeled dataset with n samples, one generates n random joint configurations, using an uniform distribution between the limits for each joint. These limits are summarized in table 4.1. To construct a labeled dataset, first one constructs the unlabeled dataset and then label each sample.

4.2.1.A Labeling process

The labeling process uses the Denavit-Hartenberg convention to compute the direct kinematics of the robot. The Denavit-Hartenberg Convention provides a consistent way of computing the transformation matrix between two consecutive joints in a kinematic chain only using 4 parameters. This notation allows a description of the manipulator that can be used in different algorithms to find the kinematics solution and it requires the reference frames to be attached to the joints. For more information about this topic, see [24].

Once all reference frames are establish, the pose of frame i with respect to frame $i-1$ is defined by the following parameters:

a_{i-1} Distance between the origins of the reference frames or, in other words, length of the link.

α_{i-1} Angle between axis z_{i-1} and z_i around axis x_i or, in other words, link twist.

d_i Coordinate of the origin of the reference frame in joint i, along z_{i-1} or, in other words, link offset.

θ_i Angle between axis x_i and x_{i-1} around z_i axis or, in other words, joint angle.

Knowing these parameters is enough to build a transformation matrix for a transformation between links i and $i+1$,

$$ {}^{i-1}_{i}T = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2} $$

where s_{α_i} and c_{α_i} represent, respectively, the sine and cosine of α_i. It is also important to note that the parameter o_i is the only variable depending on the type of joint.

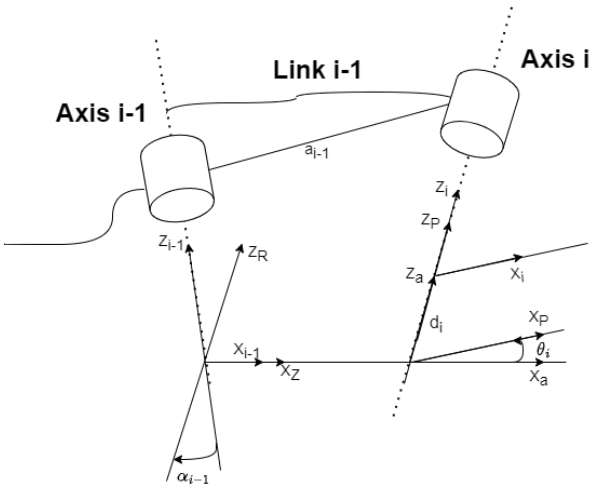


Figure 4.2: Denavit-Hartenberg parameters for two sequential joints in a kinematic chain.

Regarding the labeling process and considering a single joint configuration, $\boldsymbol{\theta}$, the first step of the labeling process consist on building the transformation matrices, using the Denavit-Hartenberg convention, with equation (4.2), using the DH parameters from table 4.2. After computing all the transformation matrices between consecutive joints, one uses these matrices to compute the transformation matrix that represents the transformation between the root reference frame and the end-effector reference frame for configuration $\boldsymbol{\theta}$, following (3.7). In other words, one uses the real DH parameters to compute the forward kinematics for $\boldsymbol{\theta}$ and obtains the position and the rotation matrix that represents the orientation of the end-effector. Finally, one needs to convert the rotation matrix to a quaternion. For this, consider a rotation matrix

$$ R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, $$

and the quaternion

$$ q = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}. $$

One can compute the quaternion from a rotation matrix following

$$q_0 = \frac{1}{2}\sqrt{r_{11} + r_{22} + r_{33} + 1}, \tag{4.3}$$

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} sgn(r_{32} - r_{23})\sqrt{r_{11} - r_{22} - r_{33} + 1} \\ sgn(r_{13} - r_{31})\sqrt{r_{22} - r_{33} - r_{11} + 1} \\ sgn(r_{21} - r_{12})\sqrt{r_{33} - r_{11} - r_{22} + 1} \end{bmatrix}, \tag{4.4}$$

where $sgn(x) = 1$ for $x > 0$ and $sgn(x) = -1$ for $x < 0$.

So, summarizing the labeling process, firstly one applies the forward kinematics to the unlabeled sample, to get the position, then converts the rotation matrix to the corresponding quaternion. It is important to note that computing the forward kinematics uses, not only the 7 joints from the robot's arm, but also 3 from the robot's torso. For the torso we consider zero degrees of rotation for all its joints.

4.2.2 Architecture and hyperparameters

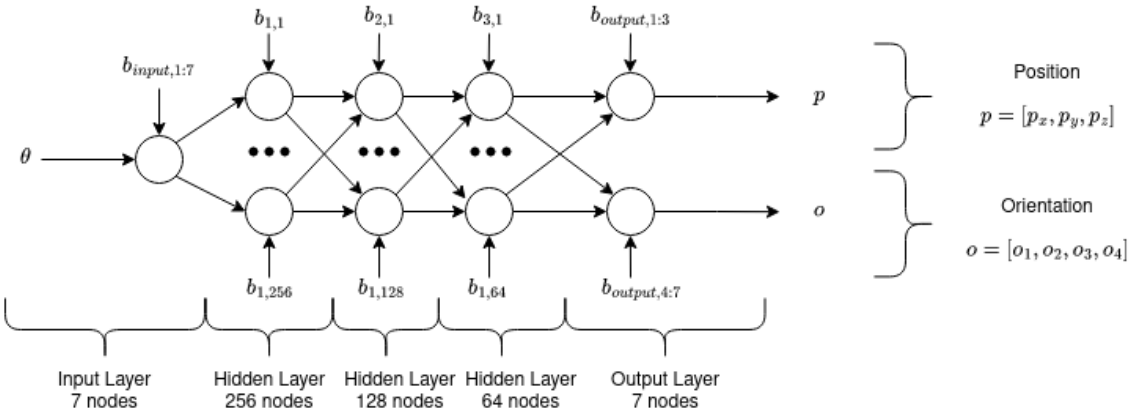


Figure 4.3: Compact representation of the NN model used.

As mentioned in section 3.2.1, the architecture of a NN should be adapted according to the task at hand. Since the model is meant to learn the forward kinematics of a seven DoF robotic arm, the number of inputs should correspond to the number of joints that compose the arm. In this case we have seven joints, represented in Figure 4.4. The output is comprised of the position and orientation of iCub's right hand. The position can be defined by three dimensions, while the orientation is represented by a quaternion, so it is defined by four dimensions. Thus, the output should have seven nodes. Considering the whole model, the network is composed of a input layer with 7 nodes, followed by three hidden layers with 256, 128 and 64 nodes, respectively, and, finally, an output layer with 7 nodes. Every node, from a given layer, is fully connected with all the nodes of the next layer. The full model is represented in Figure 4.3.

Besides the size of the NN, there are other hyperparameters that need to be defined for the training process. For a fair comparison of the results for the multiple methods tested in this thesis, the hyper-

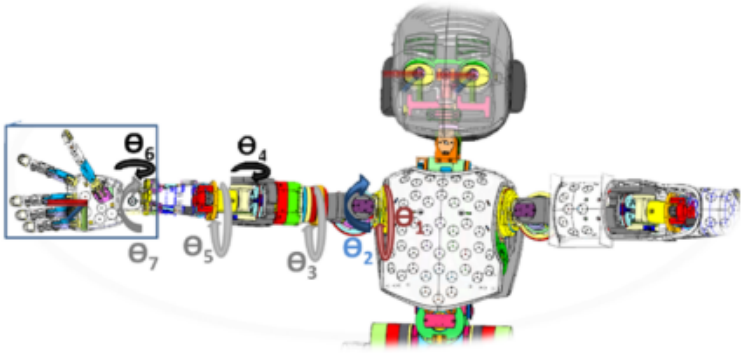


Figure 4.4: Visual representation of iCub's joints and the direction of their rotation. Figure from [1]

parameters used are the same for all the results presented in section section 5.These hyperparameters are summarized in table 4.3. The optimization method used is the Adam optimizer with a learning rate of 10^{-7}. The batch size is 10 and the training is iterated for 1500 epochs. Finally, as mentioned in section 3.2.3, the NN uses dropout for model uncertainty estimation with a dropout rate of 0.5.

Table 4.3: Neural networks hyperparameters.

Network Size	(7,256,128,56,7)
Learning Rate	10^{-7}
Batch Size	10
Epochs	1500
Dropout Rate	0.5

It is also important to note that the NN uses standardized inputs and outputs. For this we use a dataset of 1000 randomly generated joint configurations and the corresponding poses and compute the mean μ and standard deviation σ for each input and for each output. Thus, one uses this data to standardize the inputs using

$$x_{stand}^{(i)} = \frac{x^{(i)} - \mu_x^{(i)}}{\sigma_x^{(i)}}, \tag{4.5}$$

and the outputs using

$$y_{stand}^{(i)} = \frac{y^{(i)} - \mu_y^{(i)}}{\sigma_y^{(i)}}, \tag{4.6}$$

where $i = 1, ..., 7$ since both the outputs and inputs have 7 dimensions. The standardization allows one to re-scale all inputs and outputs to the same order of magnitude which makes the learning process easier. This is specially important for the outputs since the orientation and the position have different orders of magnitude, so the NN would rather update its weights to better learn the position, since this would reduce the loss way more than focusing on the orientation where the difference between its predictions and the ground truth is much smaller. So the normalization helps balance the learning process between

the position and orientation of the end-effector. It is also important to note that the NN can predict any part of the quaternion to be greater than one, which can not happen since we use the unit quaternion to represent the rotation. Thus, when this happens, our approach considers that prediction to be 1.

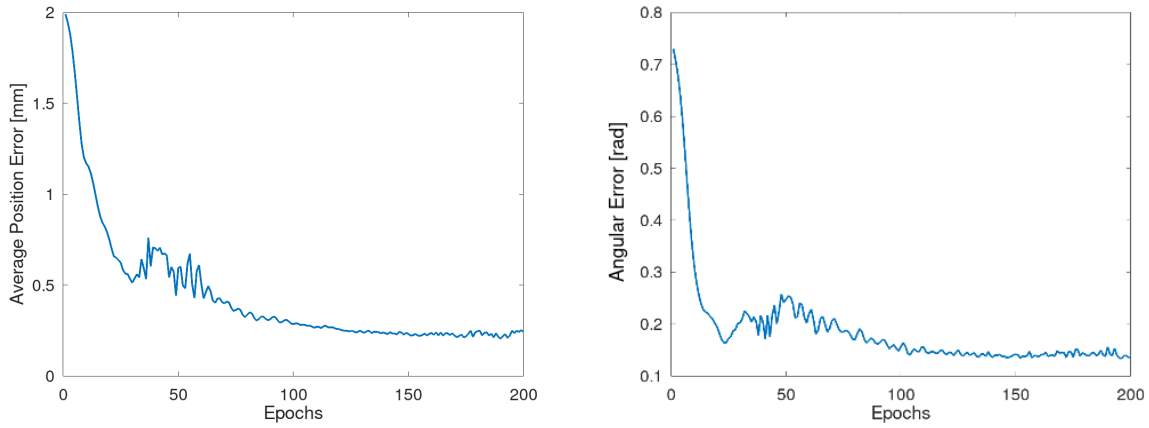
4.2.3 Initialisation

Since the focus of this thesis is to study the effect of combining AL with NN on the calibration of robots, the NN model is pre-trained so that the results are closer to reality. This training is represented in Figure 4.5. It is important to note that this initialisation is important since, without it, the framework would not be able to converge.

To train the model, a dataset with 3000 samples was used. The hyperparameters are summarized in table 4.4.

Table 4.4: Neural networks hyperparameters for pre-training the model.

Network Size	(7,256,128,56,7)
Learning Rate	10^{-5}
Batch Size	25
Epochs	200
Dataset Size	3000



(a) Average position error in millimeters with respect to the number of epochs during initialisation training.

(b) angular error in radians with respect to the number of epochs during initialisation training.

Figure 4.5: Pre-training of the NN model with using a dataset with 3000 samples.

4.3 Active Learning Implementation

As mentioned before, this thesis aims to improve the calibration of the robot in real-time situations, so it uses AL methods to accelerate the learning process. This section is responsible for introducing the implementation and particularities of the Active learning methods proposed.

Unlike the model uncertainty based AL approach, implemented via Mc Dropout, the proposed diversity based approaches, greedy sampling on input (GSx) and improved greedy sampling (iGS), are mainly based in the location of the samples in the feature space. This is not entirely true for iGS, since it balances the importance of the sample location with the prediction performance from the regression model. But, the main key information to take away is that both these models take into consideration the distance of the selected samples to the samples that were already picked. This raises the question: how does one select the first sample? In this thesis, we select the first sample randomly to induce variability in different runs of the methods

Monte Carlo sampling methods usually use a considerable amount of samples in order to solve problems in a deterministic way. As mentioned in section 3.2.4.C, Mc Dropout uses Monte Carlo sampling to estimate model uncertainty. And since the proposed approach focuses on the real-time calibration of the robot, the amount of samples usually used in Monte Carlo sampling is unfeasible. For example, if one chooses to perform 1000 forward passes to compute the uncertainty of a single sample while using a pool with 20000 samples, where it needs to repeat the process for every sample, the computation cost is unfeasible with real-time applications. So we choose to perform $T = 25$ forward passes to estimate the uncertainty of a given sample. It is also important to note that (3.42) gives the model uncertainty per output, so to get the general model uncertainty we simply sum the uncertainty of all outputs. This is reasonable since we compute the uncertainty for the normalized outputs which means that all outputs are scaled to the same order of magnitude.

4.3.1 Pool

Pool based AL requires a pool of unlabeled data where the next samples are selected from. By following the unlabeled dataset construction process described in 4.2.1, one is able to construct a pool of joint configurations that are uniformly distributed allover the feature space. So, if the pool has a considerable amount of samples, it will be a good representation of the feature space. Taking this into consideration, and since we also want to study how the size of the pool affects the results of the proposed methods, this thesis uses two different pools with 10000 and 20000 samples, respectively.

4.3.2 DIRECT Optimisation Algorithm

In order to select the most informative samples, query-synthesis AL needs an optimization algorithm to solve (3.32), so that a new joint configuration can be synthesised, labeled and used to train the NN model. Due to the non-differentiable attribute of some cost functions and to the real-time property of the calibration task, this thesis requires a global optimization algorithm with low computational cost that provides real-time solutions. The DIRECT algorithm is a global and deterministic optimization algorithm for minimizing/maximizing multivariate black-box type cost functions with upper and lower bounds on the domain. It is also offers low computational complexity while also balancing local and global searches. NLopt library[3] offers an implementation of the DIRECT algorithm, along with the implementation of a few other optimization algorithms. The DIRECT algorithm was first introduced in [4], which, along with [33], should be consulted for a deeper understanding of the algorithm.

The DIRECT algorithm divides the input space into multiple hyper-rectangles, sampling their center, with the key idea of eventually isolating the global minimum into a small region of the input space, allowing the method to deal high-dimensional spaces. So, one can divide the method into two main steps:

- Identify a set of potentially optimal hyper-rectangles.
- Divide the previously identified hyper-rectangles an sample their centers.

These are repeated for a set number of iterations or until a termination criteria is verified. This criteria can vary between the size of the rectangle to a threshold for the cost function. In this thesis, the optimization ends if it exceeds one second. Figure 4.6 represents an example of two iterations of the DIRECT algorithm for a two dimensional problem.

It is also important to note that, for higher dimensional spaces, DIRECT needs to normalize the search space into the n-dimensional unit hyper-cube. In this thesis, considering the fact that each angle of a joint configuration, θ_i, is limited by an lower bound, a_i, and an upper bound, b_i, this normalization is given by

$$\widehat{\theta_i} = \frac{\theta_i}{b_i - a_i}, \tag{4.7}$$

where $\widehat{\theta_i}$ is the normalized value of the angle. The following sections 4.3.2.A and 4.3.2.B explain in detail how to identify the potentially optimal hyper-rectangles and how to divide said hyper-rectangles, respectively. For this, one considers a general minimization problem with a cost function $C(\theta)$. The pseudo-code for the DIRECT algorithm is represented in the end of this subsection in algorithm 4.1.

[3] https://nlopt.readthedocs.io/en/latest/

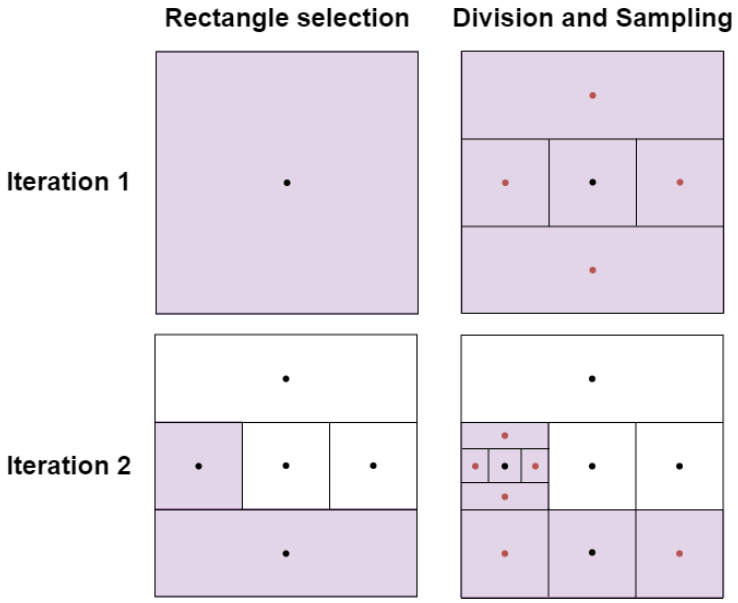


Figure 4.6: Example of two iteration of DIRECT algorithm in two dimensions. The selected potentially optimal rectangles are represented in pink while the sampled centers of the new rectangles are represented by red dots.

4.3.2.A Identification of potentially optimal hyper-rectangles

Considering that the n-dimensional unit hyper-cube has been divided into m hyper-rectangles. Let $\boldsymbol{c}$ be the center of a hyper-rectangle and d the distance between its center and its vertices. A hyper-rectangle is said to be potentially optimal if there exists a $K > 0$ such that

$$f(\boldsymbol{c}_j) - Kd_j \leq f(\boldsymbol{c}_i) - Kd_i, \quad for\ all\ i = 1, ..., m, \tag{4.8}$$

$$f(\boldsymbol{c}_j) - Kd_j \leq f_{min} - \epsilon\,|f_{min}|\,, \tag{4.9}$$

where ϵ is a small constant, usually between 10^{-3} and 10^{-7}, and f_{min} is the lowest value of the cost function know until that iteration.

The first inequality has the most impact when defining if a hyper-rectangle is optimal or not. Figure 4.7 represents a diagram that helps to visualize how a set of hyper-rectangle are defined optimal by the first condition. On the other hand, the second inequality is responsible for stopping the DIRECT algorithm from wasting time in function evaluations for very small rectangles where the expected improvement is close to none.

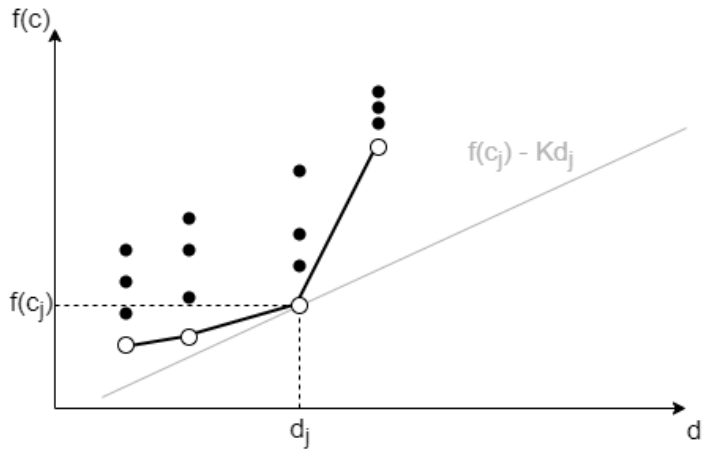


Figure 4.7: Identification of potentially optimal hyper-rectangles diagram. Each hyper-rectangle is represented by a black dot while the optimal ones are represented by a empty dot.

4.3.2.B Division of hyper-rectangles

The division of the hyper-rectangles starts with sampling the center of the next hyper-rectangles. This sampling is always done along the dimensions of highest length. The sampled points are defined by

$$\boldsymbol{c} \pm \delta e_k,\ k = 1, ..., N \tag{4.10}$$

where $\boldsymbol{c}$ are the coordinates of the center of the rectangle being divided, δ corresponds to a third of its highest length, e_k is the unit vector for the kth dimension containing the highest length and N is the number of dimensions which have the highest length. For a better visualisation, consider the example with two dimensions, n=2, in Figure 4.8 that represents the two possible divisions of the initial unit square. Since initially one has the unit square, both dimensions have the same length. This means there the algorithm samples two points vertically and two points horizontally, the sampled points are represented in blue. Now there are two ways to divide the unit square, first horizontally and then vertically or the other way around. The decision on how to subdivide the square depends on

$$\omega_k = min\left\{f(\boldsymbol{c} + \delta e_k), f(\boldsymbol{c} - \delta e_k),\right\} \tag{4.11}$$

which is the best function values along the dimension k. The algorithm starts by dividing along the dimension with the smallest ω. Then it splits the rectangle containing $\boldsymbol{c}$ along the dimension with the next smallest ω, and so on until all dimensions were divided. This makes it so that the best sampled point remains in a bigger rectangle which increases the attractiveness of searching near points with good function values, since bigger rectangles and preferred for sampling.

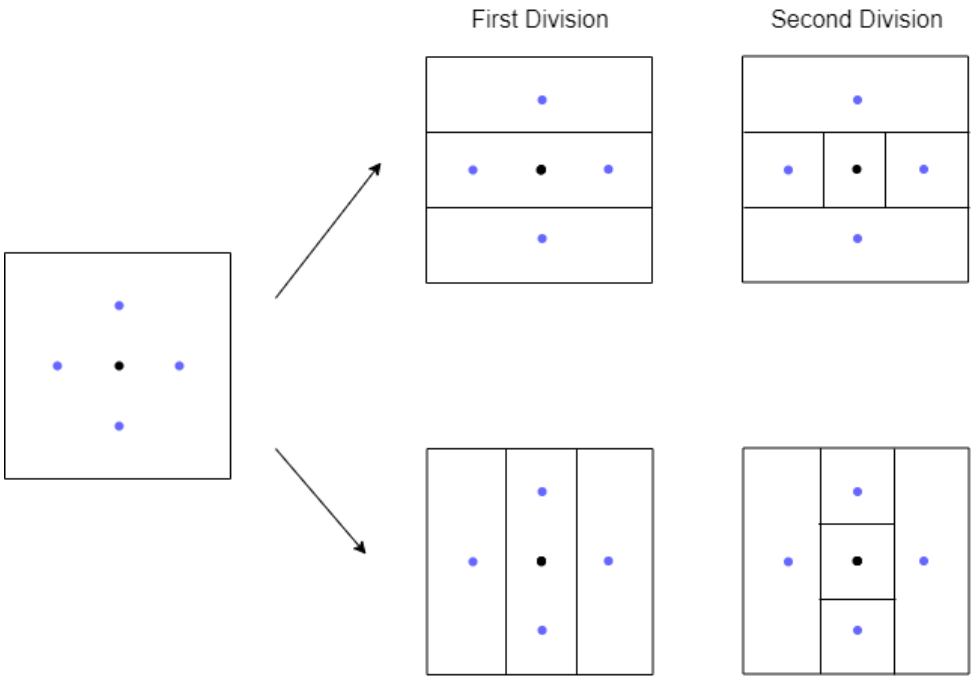


Figure 4.8: Example of sampling and division of rectangles in direct algorithm for two dimensions.

4.4 Comparison Metrics

For the purpose of comparing the advantages and analysing the performance of the proposed methods, one must define comparison metrics. These metrics are what allow the critical evaluation and construction of a comparative study between the different methods. This section introduces and examines each metric used for comparing the performance of our framework with baseline methods.

In order to measure the error of the predictions generated by the proposed models, consider a test dataset composed of 1000 samples of randomly generated joint configurations and the corresponding positions and orientations of the end-effector. These samples were generated using a uniform distri-

Algorithm 4.1: DIRECT algorithm adapted from [4]

Let c_1 be the center of the unit hyper-cube and evaluate $f(c_1)$. Set $f_{min} = f(c_1)$. Let I be the number of iterations and m the number of sampled points. Set $m = 1$.;
for $i = 0, ..., I$ **do**
  Identify the set S of potentially optimal hyper-rectangles as described in section 4.3.2.A;
  while $S \neq \emptyset$ **do**
    Select any $j \in S$;
    Determine where to sample and how to divide the hyper-rectangle j following the process described in section 4.3.2.B;
    Update f_{min};
    Set $m = m + \triangle m$, where $\triangle m$ is the number of new points sampled ;
    Set $S = S - \{j\}$.

bution following the process explained in section section 4.2.1. For each sample of this dataset, the corresponding errors are computed, and the overall errors, for both position and orientation, are formed from the mean of all the errors. For an easier and more general representation of the used metrics, the number of samples considered is N.

4.4.1 Average Position Error

This metric is used to evaluate the performance of the position predictions from the model. It provides the average deviation of the estimated position from the real reference position. It is one of the most used metrics in robotics application that involve the estimation of positions and its computation involves comparing a set of position estimates to their real position and the averaging the results.

So, considering N samples, the average position error uses the euclidean distance between the estimate and the ground truth and averages the results. Its expression is given by

$$\frac{1}{N}\sum_{n=1}^{N}\|\widehat{p}_n - p_n\| \ [mm]\,, \tag{4.12}$$

where $\widehat{p}_n$ represents the estimated position and p_n the real position.

4.4.2 Angular Error

This metric is used to evaluate the performance of the orientation predictions from the models. It uses the inner product between quaternions to represent the angle deviation of the rotations represented by two quaternions.

Considering, again, N samples, the angular error is computed using,

$$\frac{1}{N}\sum_{n=1}^{N} arccos\left(|\widehat{q}_n \cdot q_n|\right) \ [rad]\,, \tag{4.13}$$

where $\widehat{q}_n$ is the estimated quaternion, representing the estimated orientation, q_n is the quaternion that represents the real orientation and $\cdot$ represents the vector inner product, not the multiplication. It is important to make this distinction since the multiplication of quaternions generates another quaternion. It is also important to note that the range of values of this metric is $\left[0; \frac{\pi}{2}\right]$ (radians).

4.5 Calibration Routine

Our approach focuses on the calibration of the right arm of the iCub by combining the learning capabilities and quality feature extraction of NN with the efficient sample selection of AL. The calibration

routine runs in a loop where, in every iteration a new batch of data is chosen and added to the training set to train the NN model in order to learn the kinematics of the robot. These iterations can be different for different AL approaches and whether one uses a pool of data (pool based AL) or generates its own samples (query-synthesis based AL). This section focuses on explaining how these iterations work for the different approaches.

4.5.1 Pool based Active Learning Setting

For pool based active learning one needs to solve (3.34), which means one needs to apply the utility function of the used approach to all the samples in the pool. It is important to note that the calibration routine for the diversity based AL approaches is different from the routine of the model uncertainty based AL approach. This difference resides on iGS and GSx needing to update the longest distances from the samples in the pool to the already selected samples every time a new joint configuration is added to the selected samples. Instead, Mc Dropout only needs to compute the uncertainty of every sample of the pool once. In this subsection, consider that the number of samples acquired in each iteration it N.

Regarding GSx and iGS, the iteration starts by updating the distances of the pool samples following (3.38) and (3.41), respectively, then selects the joint configuration from the pool with the highest uncertainty by solving (3.34) with the respective utility functions. Afterwards it removes these samples from the pool, labels it, following the labeling process explained in section 4.2.1.A, and adds both the configuration and the corresponding pose to the batch containing the next selected data. Then, since it is the first sample of this batch, it updates the distances of the pool but only taking into consideration the newly selected sample. Since we store the previous distances of all the pool values, one applies the respective utility functions between the pool samples and the new sample and only update these distances if they are higher than the previous. Even thou this process requires more memory, it reduces the computation time. The whole process is repeated until the batch achieves the intended size. When it does, the batch is added to the training set which is then used to train the neural network model, which is the last step of the iteration. This process is represented in the flowchart represented in Figure 4.9.

When using Mc Dropout, the calibration routine does not need to update the uncertainty of the samples in the pool every time it selects a new sample since its utility function is purely based on the predictions of the learning model. The iteration starts by computing the uncertainty of every sample in the pool following (3.42). Afterwards, it selects the n samples with higher uncertainty, removes them from the pool, labels them following the explained in section 4.2.1.A and adds both the selected joint configurations and the corresponding poses to the training set. This training set is then used to train and update the neural network, finishing the iteration. The described framework can be visualized in the flowchart in Figure 4.10.

Thus, the main difference between the calibration routines described, is the fact that the first needs

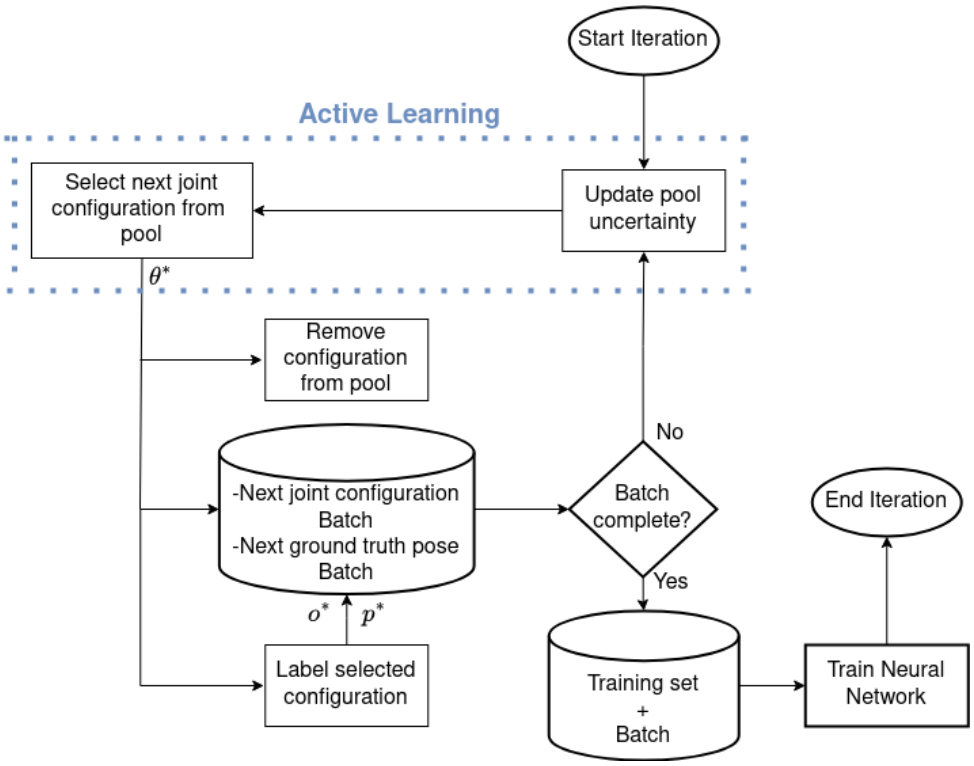


Figure 4.9: Iteration flowchart for the calibration routine using GSx or iGS in pool based AL. θ^* is the joint configuration chosen by the AL process at a given iteration while p^* and o^* are the corresponding position and orientation, respectively.

to update the distances of the pool samples every time a new sample is selected, creating a loop that runs N times. While the latter only needs to update the uncertainty once. Even though the first has to compute the distances for the pool multiple times, the computation time for both methods is around the same. This is due to the fact that Mc Dropout has to do 20 forward passes through the neural network for each sample, balancing the computation time of the batch selection process. It is important to note that this process is quite fast, being the training of the neural network that occupies most of the computation time during each iteration. It is also important to note that, when not using, AL the samples are randomly selected and removed from the pool.

4.5.2 Query-synthesis based Active Learning Setting

Query-synthesis based AL, unlike pool based AL, does not selects its samples from a pool of data, it generates its own samples. Thus one needs a optimization algorithm to solve (3.32), for the different utility functions: (3.38), (3.41) and (3.42). This algorithm is the DIRECT optimisation algorithm, introduced in section 4.3.2. So, regarding the calibration routine for the query-synthesis based AL, the iteration starts by selecting a sample using DIRECT and the corresponding utility function, depending on what AL method is being used, which is then labeled, following the process described in section 4.2.1.A, and

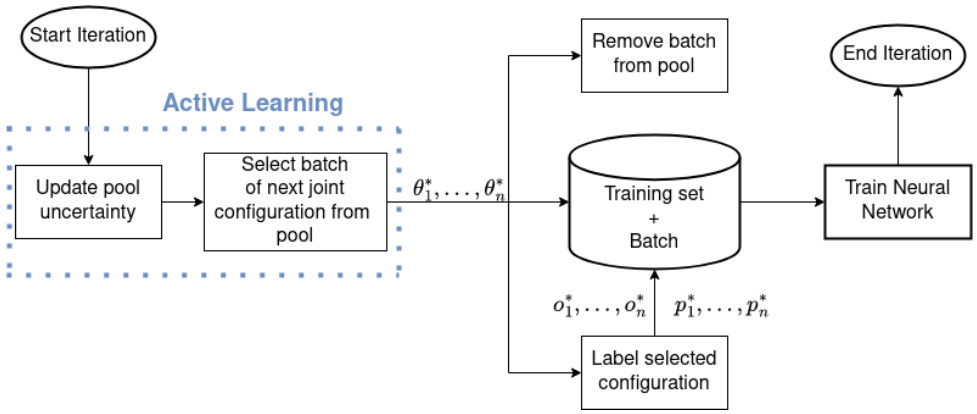


Figure 4.10: Iteration flowchart for the calibration routine using Mc Dropout in pool based AL. θ^* is the joint configuration chosen by the AL process at a given iteration while p^* and o^* are the corresponding position and orientation, respectively. The subscripts in these variables correspond to the number of the given sample in the corresponding batch.

added to the batch of the next joint configurations and corresponding poses. This process is iterated until the batch has the intended size. At this point, similarly to how the previously described routines ended, this batch is added to the training set which is then used to train and update the NN model. The flowchart representing this routine can be visualized in Figure 4.11. Also, when not using AL, the batches of data are chosen randomly using an uniform distribution.

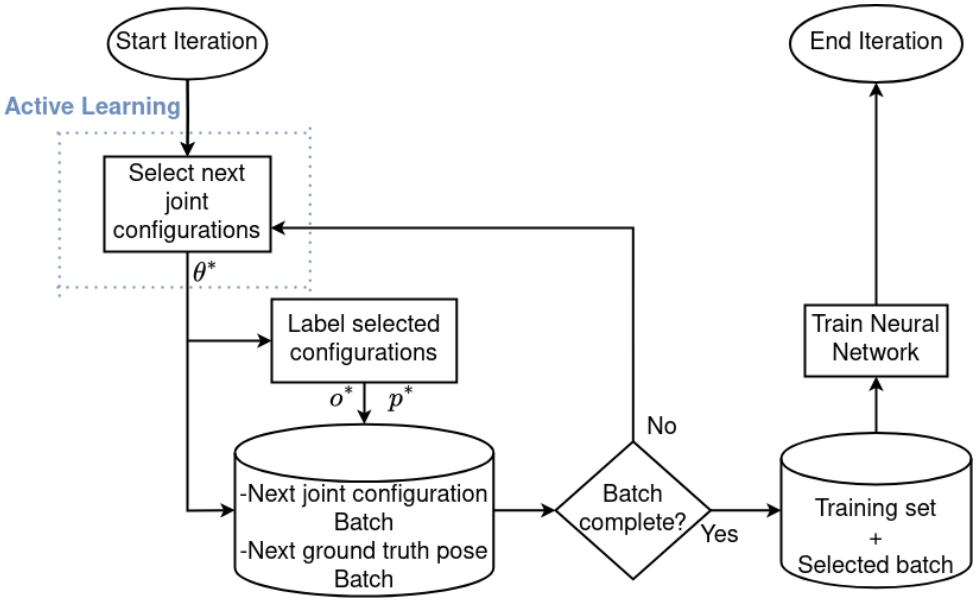


Figure 4.11: Iteration flowchart for the calibration routine for query-synthesis based AL. θ^* is the joint configuration chosen by the AL process at a given iteration while p^* and o^* are the corresponding position and orientation, respectively.

5

Simulation Results

Contents

This chapter's objective is to provide the results for the proposed framework of the calibration of iCub. It starts by showing the results for pool based AL in Section 5.1 while providing an explanation for these findings. Still regarding the pool based AL, Section 5.1 provides an analysis on how the size of the pool influences the AL methods results. Section 5.2 displays the results for query-synthesis AL. Throughout that chapter, for both pool and query-synthesis AL, the results also provide a study for how the size of the batch of selected samples affects the different methods. Finally, Section 5.3 provides a general comparison between the results for pool and query-synthesis based AL.

The results presented in this chapter correspond to the calibration routines presented in section 4.5 for three different active learning methods and random sampling. These methods and their utility functions are summarized in table 5.1. The first methods is the Random selection and serves as baseline for comparison with the other proposed methods. The second and third are Greedy Sampling on input and improved Greedy Sampling, respectively, and are both diversity based active learning methods. As mentioned before, since these methods are based on a pool already selected configurations, the first sample for both of them is selected randomly. The last method, for joint selection, is Monte Carlo Dropout which is a uncertainty based AL method and uses a dropout rate of $p = 0.5$ which was found to achieve the best results. It is also important to note that, during the uncertainty estimation phase of the method, we do 25 forwards passes for each joint configuration. We found that when doing the more forwards passes the results would get slightly better but it was also more time consuming, and since the objective is to calibrate the robot in real time we choose to only do 25 forward passes.

Table 5.1: Joint configuration selection methods.

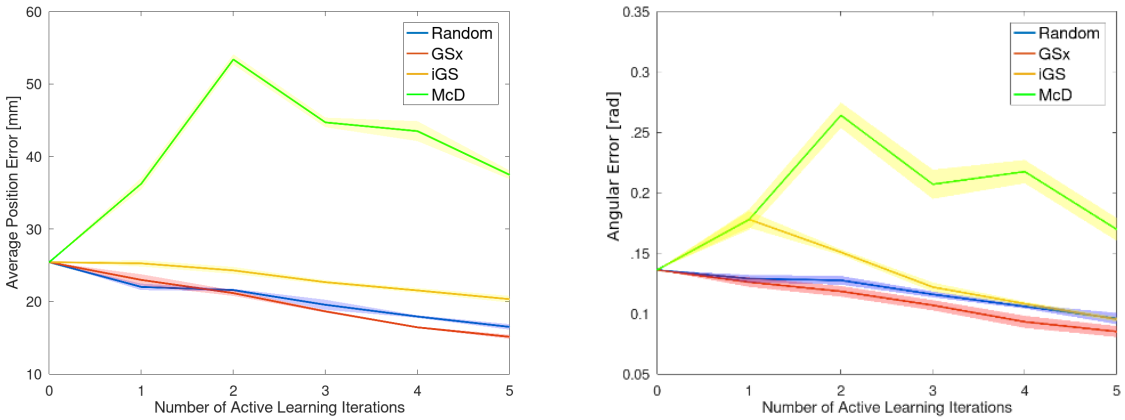
Method	utility function
Random (R)	Uniform Random selection
Greedy Sampling on input (GSx)	(3.38)
Improved Greedy Sampling (iGS)	(3.41)
Monte Carlo Dropout (Monte-Carlo Dropout (McD))	(3.42)

To study the effect of the batch size on the proposed framework we present the results for three different batch sizes: 250, 500 and 1000. For the two smaller batches, the calibration routine runs for 5 iterations while for 1000 configurations runs for 3 iterations since the learning model would start to overfit for more iterations. Each iteration is responsible for selecting a new batch of data, add it to the training set and train the neural network. Due to the stochastic nature of NN, the results displayed correspond to an average of 5 repetitions of the calibration processes. It is also important to note that 250 was the minimum size of the training set that allowed the NN to learn the forward kinematics of iCub, not being able to converge to a solution of iCub's kinematics for a smaller batch size.

5.1 Pool based Active Learning

The results in this section were obtained following the calibration process described in Figure 4.10, for McD, and Figure 4.9, for GSx and iGS. The selection process for R is also described at the end of section 4.5.1. All the results presented use a pool with 20000 samples.

For each method, Figure 5.1, 5.3 and 5.5 show the evolution of the average position error and the angular error with respect to the number of AL iterations for batch sizes of 250, 500 and 1000, respectively. By analyzing these figures, the first relevant observation is that, in the first iterations, McD increases the error, in both metrics. Even for higher batch sizes, after diverging during the first iterations, it starts to properly learn the kinematics of the robot, decreasing both metrics, but not fast enough to compete with the other selection methods. It is also important to note that, for the later AL iterations, the uncertainty estimation for the selected samples were higher than for the first iterations. This provides the intuition that, when the learning model is more uncertain of one joint configuration, that sample is more useful for training the neural network.



(a) Average Position error in millimeters with respect to the number of active learning iterations.

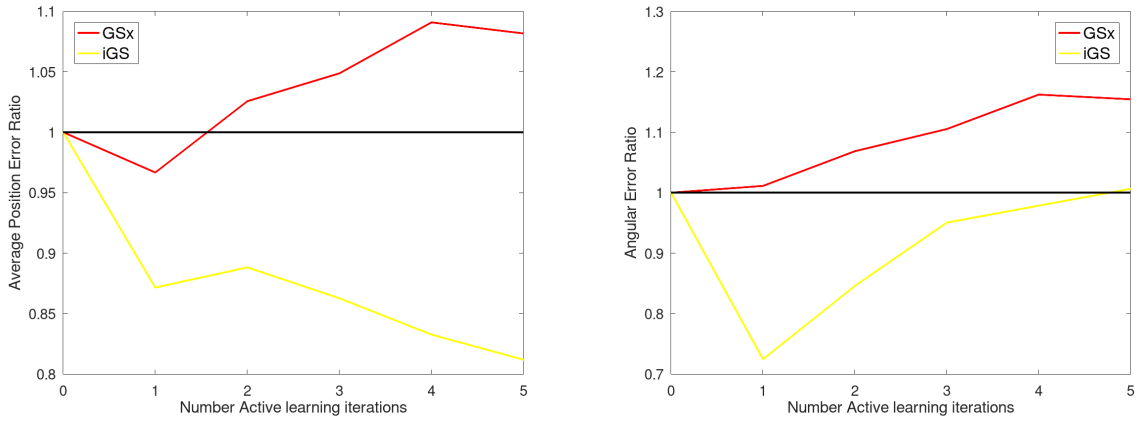
(b) Angular error in radians with respect to the number of active learning iterations

Figure 5.1: Results for different joint selection methods for the calibration routine for pool based AL and a batch size of 250. In blue the joint configurations are selected randomly (baseline). In red the joint configurations are selected using GSx. In yellow the joint configurations are selected using iGS. In green the joint configurations are selected using MC Dropout. The faded areas correspond to the standard error of the mean of the respective method.

To provide a quantitative comparison between GSx and iGS opposed to R, Figure 5.2, 5.4 and 5.6 show the ratios of both metrics of the AL methods with the metrics of R for batch sizes of 250, 500 and 1000, respectively. The values in the these figures are computed following

$$ratio = \frac{e_R}{e_{AL}}, \tag{5.1}$$

where e_R is a metric of method R and e_{AL} is a the same metric when using one of the AL approaches (GSx or iGS). These metrics can either be the average position error of the angular error. Ratios higher than 1, above the black line, mean that the respective method outperformed R. Due to the bad performance of McD, in comparison with the other selection methods, we decide to not include its results in ratio figures.



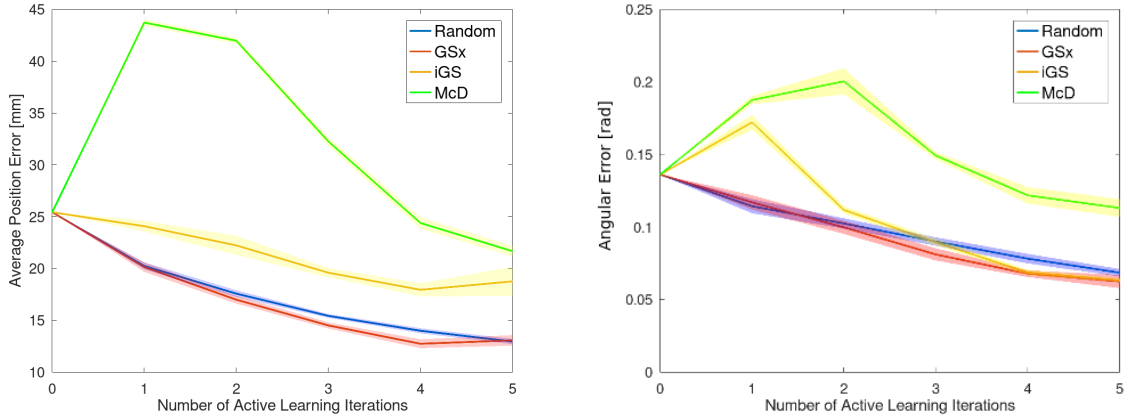
(a) Ratio of Average Position error with respect to the number of active learning iterations.

(b) Ratio of Angular error with respect to the number of active learning iterations

Figure 5.2: Comparison of different joint selection methods for pool based AL and a batch size of 250. Ratio of the errors for the AL methods in comparison with Random Selection. Ratio bigger than 1 (above black line) shows the superiority of the method opposed to random selection.

From observing Figure 5.1, one can conclude that the only AL method to outperform R in learning both the position and orientation is GSx, only outperformed by random in the first iteration for the position. When increasing the batch size to 500 (figure 5.3) we see that GSx learns the forward kinematics faster than R, always outperforming it. When increasing, even more, the batch size to 1000 (figure 5.5), GSx learns the kinematics even faster than R, needing less iterations to achieve higher ratios. This increase in performance with the batch size is to be expected since, for higher batch sizes, the AL methods contains more informative samples with less iterations. For example, in the third iteration for a batch size of 250, the training set has 750 joint configuration samples (and the respective poses), while for a batch size of 500 it has double the samples (1500) and, for a batch size of 1000, it has four times the samples (3000). Let us consider, for example, the iteration when the average position error ratio for GSx achieves 1.1, for 250 batch size it is in the fifth iteration (1250 samples in the training set), for 500 it is around the third iteration (1500 samples in the training set) and for 1000 it is in the first iteration (1000 samples in the training set). But this is in the direct comparison between GSx and R, while it is also important to consider that, in general, all methods improve their preformances when increasing the batch size being able to achieve lower errors with less iterations. This result is also to be expected since the more data a

NN uses to train a model, the better its performance.



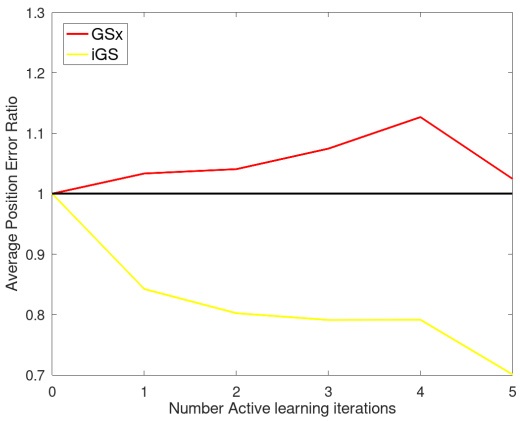
(a) Average Position error in millimeters with respect to the number of active learning iterations.

(b) Angular error in radians with respect to the number of active learning iterations

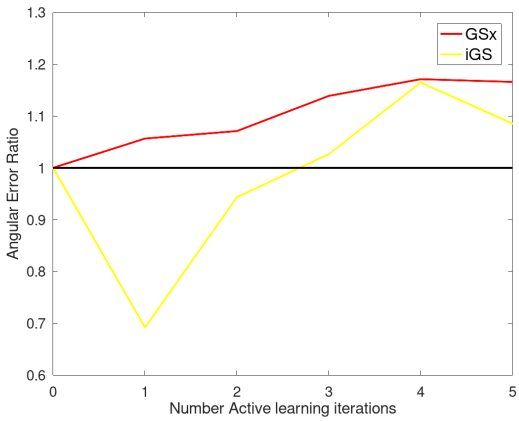
Figure 5.3: Results for different joint selection methods for the calibration routine for pool based AL and a batch size of 500. In blue the joint configurations are selected randomly (baseline). In red the joint configurations are selected using GSx. In yellow the joint configurations are selected using iGS. In green the joint configurations are selected using MC Dropout. The faded areas correspond to the standard error of the mean of the respective method.

The most interesting results come from the iGS selection method. Even though it underperforms in comparison to R and GSx when learning position of the end-effector, one can observe that, for higher batch sizes, it is able to outperform R when learning the orientation of the end-effector in the later iterations. Regarding the orientation, iGS initially diverges for the smaller batch sizes, but, when it has enough samples, it is able to learn the orientation faster than R and, for a batch size of 1000 it is even able to outperform GSx in the last iteration. Comparing GSx and iGS, the main difference between the two methods is that iGS also takes into consideration the distance between the prediction of the learning model and the ground truth. So, one can conclude that also considering the distances in the output space, instead of just in the input space, deteriorates the learning process of the position while improving the performance of learning the orientation, for enough samples.

Thus, one can conclude that, for pool based AL, the best sample selection method for learning the forward kinematics of iCub is GSx, being the only method in this study that outperforms random selection. One can also conclude that diversity based AL outperforms uncertainty based AL for this specific task and that the latter is not a good selection criteria for the task at hand. It is also important to note that GSx is the only sample selection method that does not use predictions from the NN in the utility function. So, one could conclude that AL methods that use predictions of the NN in the utility function when solving (3.34) are worse than random selection of the samples, when learning the forward kinematics of iCub. It is important to note that these conclusions are for the specific experimental context,



(a) Ratio of Average Position error with respect to the number of active learning iterations.



(b) Ratio of Angular error with respect to the number of active learning iterations

Figure 5.4: Comparison of different joint selection methods for pool based AL and a batch size of 500. Ratio of the errors for the various AL methods in comparison with Random Selection. Ratio bigger than 1 (above black line) shows the superiority of the method opposed to random selection.

of learning the forward kinematics of iCub's right arm using a combination of AL and NN.

5.1.1 Pool size

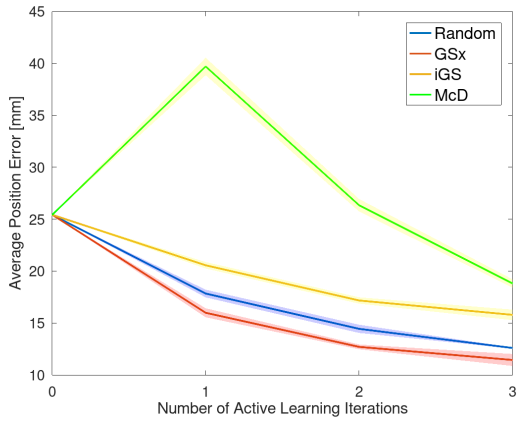
In this section the results of how the size of the pool affect the performance of each AL method are presented. For comparison, we use two different pools of sizes 10000 and 20000 and present the results of GSx, iGS and McD for both pools and for batch sizes of 250, 500 and 1000. Figures 5.7, 5.8 and 5.9 show the results for the calibration routines using the AL methods for both pools and for batch sizes of 250, 500 and 1000, respectively. The legends of these figures indicate the method being used followed by the size of the pool.

The first observation is that McD, for every batch size, initially presents worse results for the bigger pool, but around the second iteration the results for the bigger pool outperform the smaller pool. This is true for both metrics.

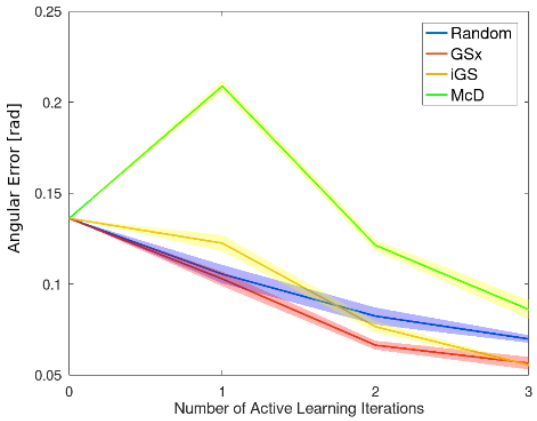
Regarding iGS, using the bigger pool always outperforms the smaller pool for position. In the case of the orientation, for the two smaller batch sizes, using the bigger pool starts to outperform the smaller pool around the second iteration. But in general the bigger pool provides better results for both metrics.

Finally, regarding GSx, for 250 and 500 batch sizes the results for both pools are pretty similar for both metrics but the bigger pool presents slightly better results. For the 1000 batch size the bigger pool clearly presents better results for both metrics

Thus, one can conclude that having a bigger pool usually improves the results for the AL methods.

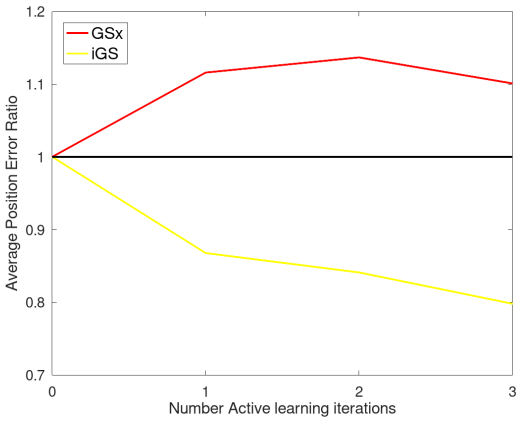


(a) Average Position error in millimeters with respect to the number of active learning iterations.

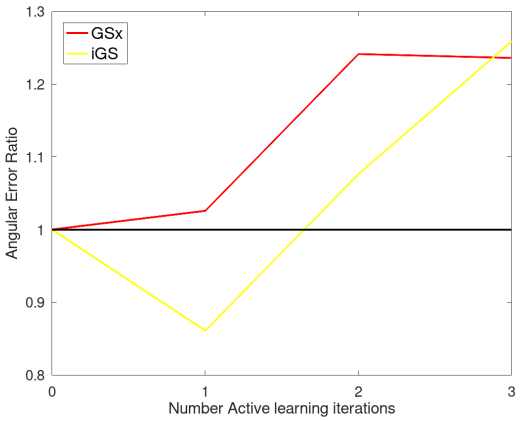


(b) Angular error in radians with respect to the number of active learning iterations

Figure 5.5: Results for different joint selection methods for the calibration routine for pool based AL and a batch size of 1000. In blue the joint configurations are selected randomly (baseline). In red the joint configurations are selected using GSx. In yellow the joint configurations are selected using iGS. In green the joint configurations are selected using MC Dropout. The faded areas correspond to the standard error of the mean of the respective method.

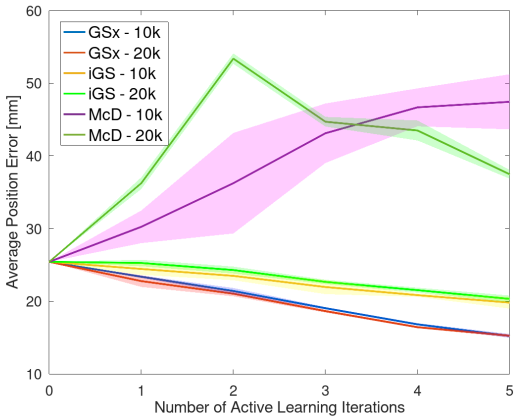


(a) Ratio of Average Position error with respect to the number of active learning iterations.

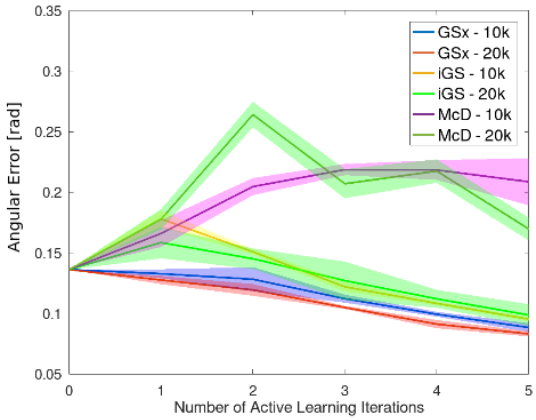


(b) Ratio of Angular error with respect to the number of active learning iterations

Figure 5.6: Comparison of different joint selection methods for pool based AL and a batch size of 1000 Ratio of the errors for the various AL methods in comparison with Random Selection. Ratio bigger than 1 (above black line) shows the superiority of the method opposed to random selection.

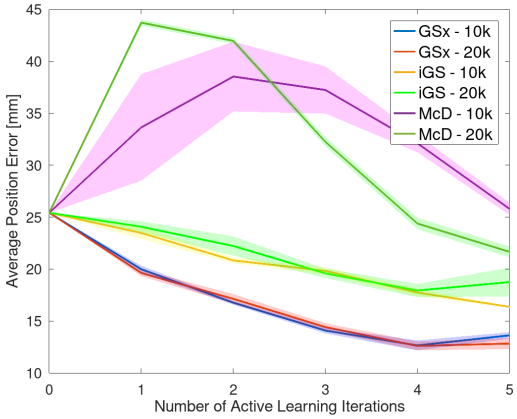


(a) Average Position error in millimeters with respect to the number of active learning iterations.

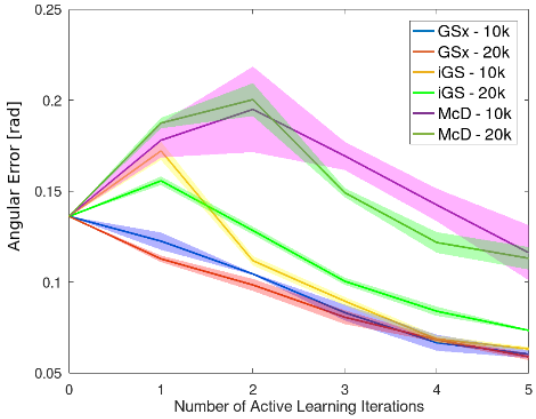


(b) Angular error in radians with respect to the number of active learning iterations

Figure 5.7: Comparison of joint selection methods for pool sizes of 10000 and 20000 samples and a batch size of 250. The legend indicates the selection method being used followed by the size of the pool. The faded areas correspond to the standard error of the mean of the respective method.

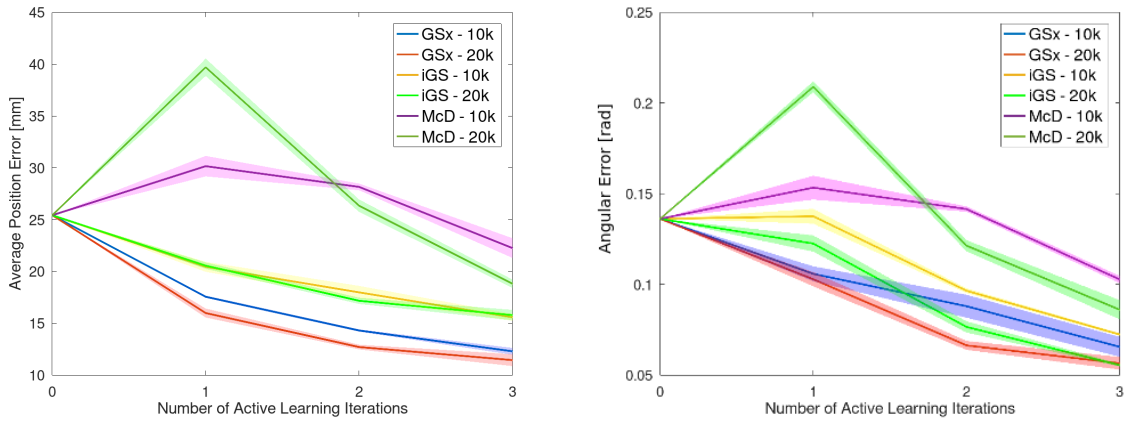


(a) Average Position error in millimeters with respect to the number of active learning iterations.



(b) Angular error in radians with respect to the number of active learning iterations

Figure 5.8: Comparison of joint selection methods for pool sizes of 10000 and 20000 samples and a batch size of 500. The legend indicates the selection method being used followed by the size of the pool. The faded areas correspond to the standard error of the mean of the respective method.



(a) Average Position error in millimeters with respect to the number of active learning iterations.

(b) Angular error in radians with respect to the number of active learning iterations

Figure 5.9: Comparison of joint selection methods for pool sizes of 10000 and 20000 samples and a batch size of 1000. The legend indicates the selection method being used followed by the size of the pool. The faded areas correspond to the standard error of the mean of the respective method.

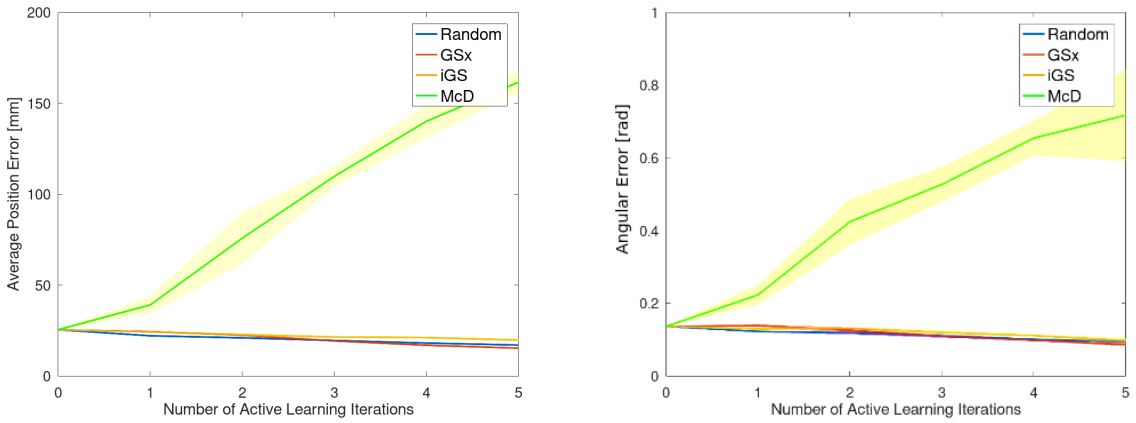
This is intuitive since the pool is built by generating random joint configurations using an uniform distribution, so, having a bigger pool usually means that the pool has a larger diversity of samples, representing more of the input space. This means that, if the pool is big enough, it can fully represent the input space, allowing the active learning methods to fully explore the input space. It is also true that, if the pool is big enough, it will contain more redundant information which can be destructive for methods like McD. For example if you have 100 samples that are pretty similar and one of them has the highest uncertainty, then the other 99 samples should also have high values of uncertainty. If the batch size is 250, it is possible to pick those 100 samples in the same batch which only a few will provide important information, because the rest will be redundant information. On the opposite hand, having a pool with low amounts of samples means that the pool only represents a restricted part of the entire input space, so it should be intuitive that, in this case, having a bigger pool should provide better results for the AL methods.

5.2 Query-synthesis based Active Learning

The results presented in this section were obtained following the calibration process described in Figure 4.11 for all the AL sample selection methods. For R, the joint configurations are chosen randomly using an uniform distribution that respects the lower and upper bounds of every joint.

Figure 5.10 shows the evolution of both metrics with respect to the number of AL iterations for the selection of batch with 250 samples. It is clear that McD increases the error when trying to learn the kinematics of iCub. The results for McD were the same for higher batch sizes, so, in order to study the

other methods, Figures 5.11, 5.13 and 5.15, show the evolution of both metrics for the methods R, GSx and iGS for batch sizes of 250, 500 and 1000, respectively. There might be two explanations for the results of McD. The first was mentioned in section 5.1.1, meaning that the DIRECT algorithm would chose samples that are very similar because they have higher values of uncertainty. This is possible since it is a deterministic optimization method. But, since the utility function uses dropout, which offers some randomness to the uncertainty estimations, the optimization methods does not chose always the same solution. This does not mean that it wont choose similar solutions. The second reason resides on the fact that the stopping criterion for optimization algorithm (stops after running for one second) might not be enough to reach a proper solution. But increasing this value would go against the objective of this thesis of calibrating the robot system in real time. Specially since one second for each sample is already a considerable amount of time.



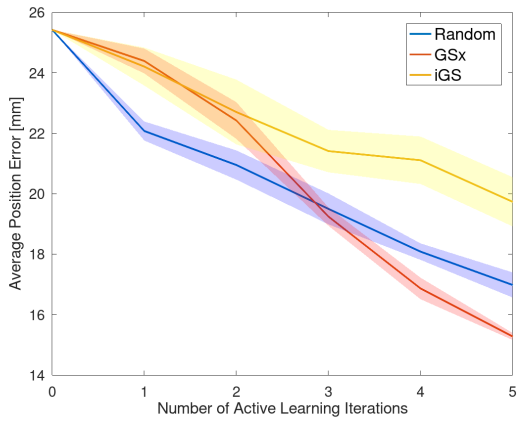
(a) Average Position error in millimeters with respect to the number of active learning iterations.

(b) Angular error in radians with respect to the number of active learning iterations

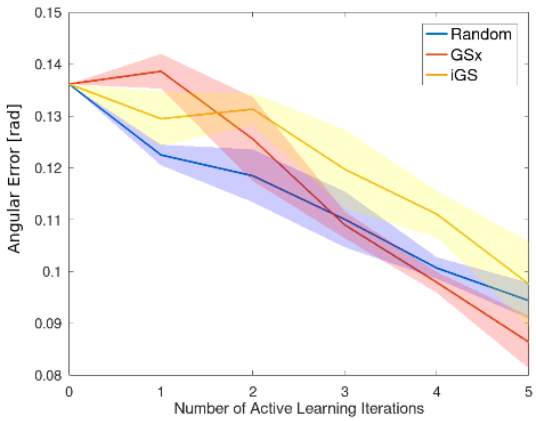
Figure 5.10: Results for different joint selection methods for the calibration routine for query-synthesis based AL and a batch size of 250. In blue the joint configurations are selected randomly (baseline). In red the joint configurations are selected using GSx. In yellow the joint configurations are selected using iGS. In green the joint configurations are selected using MC Dropout.

Similarly to the results shown in section 5.1, to provide a quantitative comparison between the AL approaches in opposition to R, Figure 5.12, 5.14 and 5.16 show the ratios, computed using (5.1), for batch sizes of 250, 500 and 1000 respectively. In general, on can identify that the results for the baseline R are very similar to when applying R using a pool.

From Figure 5.11, one can verify that GSx outperforms R from the third iteration for both metrics, while when using a batch of 500 samples (5.13) it always outperforms R and the same is also true for 1000 batch size (5.15). The increase of performance with the batch size analyzed in section 5.1 holds true for GSx using query-synthesis based AL.

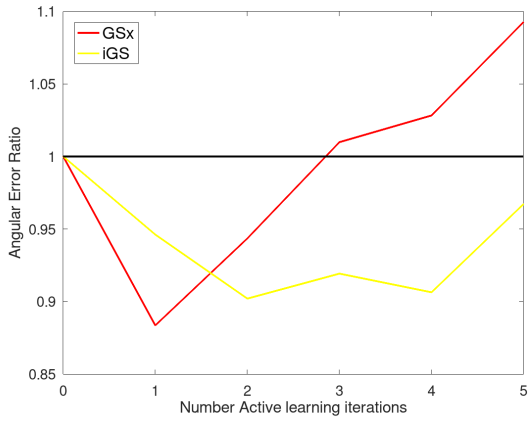


(a) Average Position error in millimeters with respect to the number of active learning iterations.

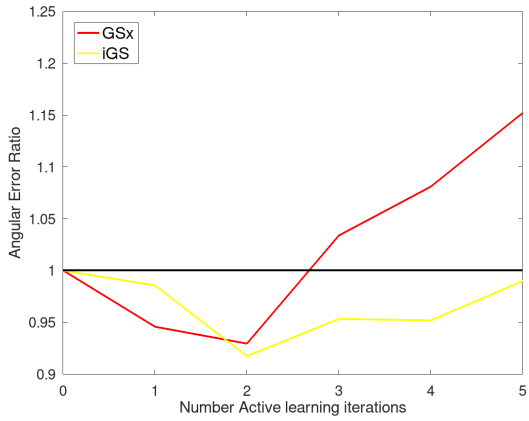


(b) Angular error in radians with respect to the number of active learning iterations

Figure 5.11: Results for different joint selection methods for the calibration routine for query-synthesis based AL and a batch size of 250. In blue the joint configurations are selected randomly (baseline). In red the joint configurations are selected using GSx. In yellow the joint configurations are selected using iGS. The faded areas correspond to the standard error of the mean of the respective method.



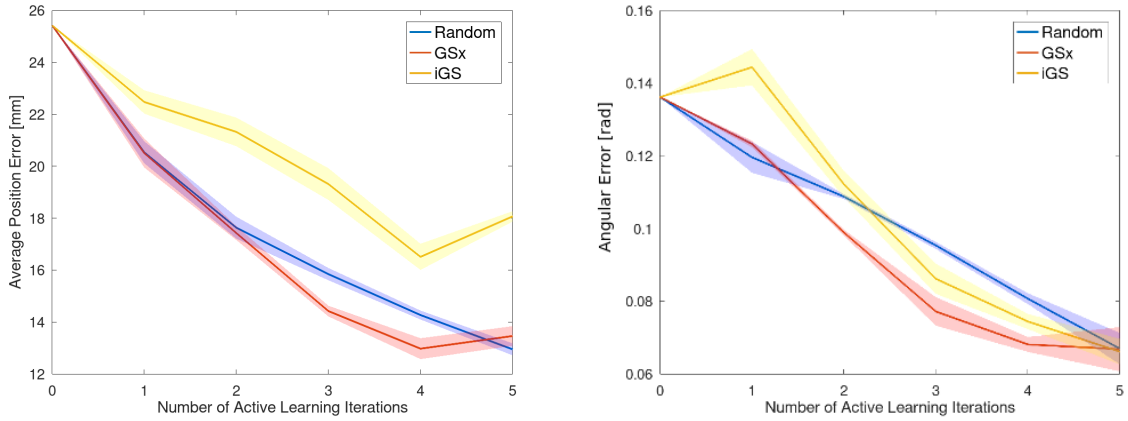
(a) Ratio of Average Position error with respect to the number of active learning iterations.



(b) Ratio of Angular error with respect to the number of active learning iterations

Figure 5.12: Comparison of different joint selection methods for query-synthesis based AL and a batch size of 250. Ratio of the errors for the AL methods in comparison with Random Selection. Ratio bigger than 1 (above black line) shows the superiority of the method opposed to random selection.

When using iGS, the results are pretty similar to the ones in section 5.1, the method clearly underperforms R and GSx when learning the position of the end-effector but manages to outperform R when learning the orientation, with the difference that it seams to perform better for smaller batch sizes when compared to the results using pool based AL.



(a) Average Position error in millimeters with respect to the number of active learning iterations.

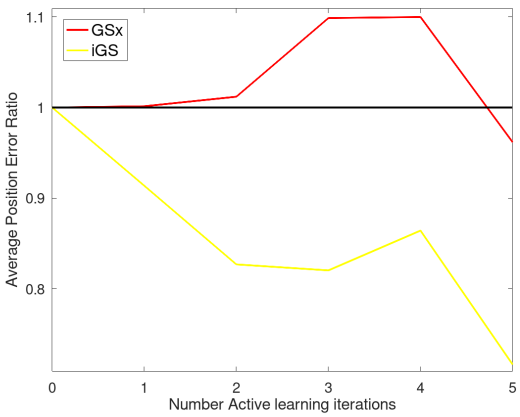
(b) Angular error in radians with respect to the number of active learning iterations

Figure 5.13: Results for different joint selection methods for the calibration routine for query-synthesis based AL and a batch size of 500. In blue the joint configurations are selected randomly (baseline). In red the joint configurations are selected using GSx. In yellow the joint configurations are selected using iGS. The faded areas correspond to the standard error of the mean of the respective method.

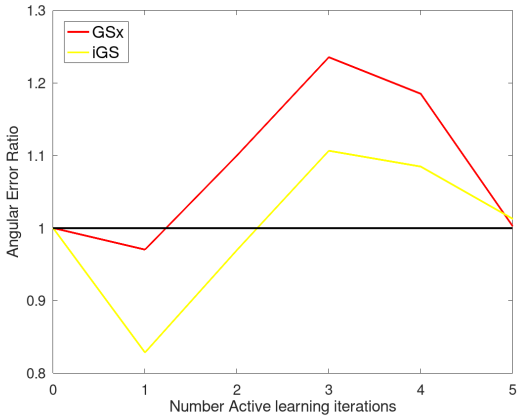
Thus, one can conclude that, for query-synthesis AL, the best sample selection method for learning the forward kinematics of iCub is, again, GSx being the only method that is able to outperform R. One can also conclude that McD does not use a good criteria for the selection of the most informative samples, for this specific task, since it is unable to learn the forward kinematics of the robot. So the overall results when using query-synthesis based AL are quite similar to the results when using pool based AL.

5.3 Pool based Active learning vs Query-synthesis based Active learning

As mentioned before, from comparing the figures with the average position errors and angular errors for the respective batch sizes for pool and query-synthesis based AL, the results are quite similar, not having a significant quantitative difference. All the AL methods also have the same qualitative trend when compared to R, for query-synthesis and pool based AL. McD is the worst selection method, while iGS it is better at predicting the orientation than the position and GSx is the only joint configuration selection method that is able to outperform R.

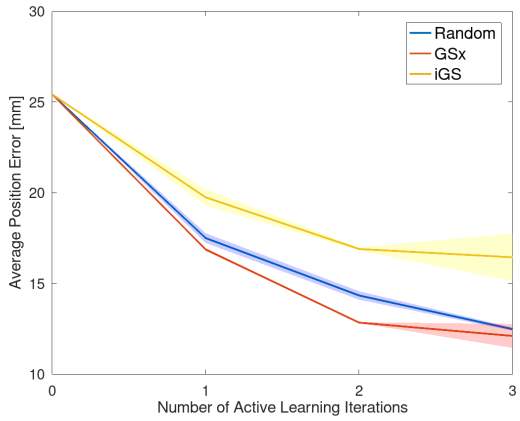


(a) Ratio of Average Position error with respect to the number of active learning iterations.

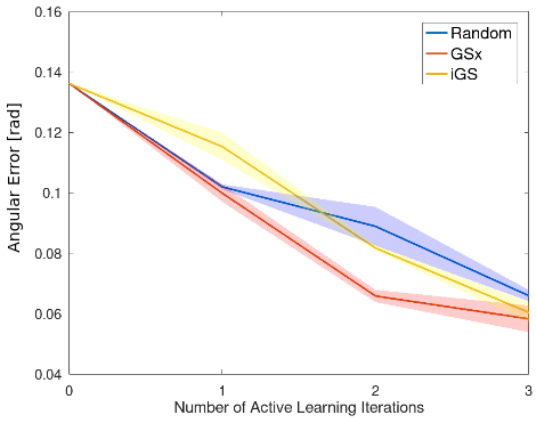


(b) Ratio of Angular error with respect to the number of active learning iterations

Figure 5.14: Comparison of different joint selection methods for query-synthesis based AL and a batch size of 500. Ratio of the errors for the AL methods in comparison with Random Selection. Ratio bigger than 1 (above black line) shows the superiority of the method opposed to random selection.

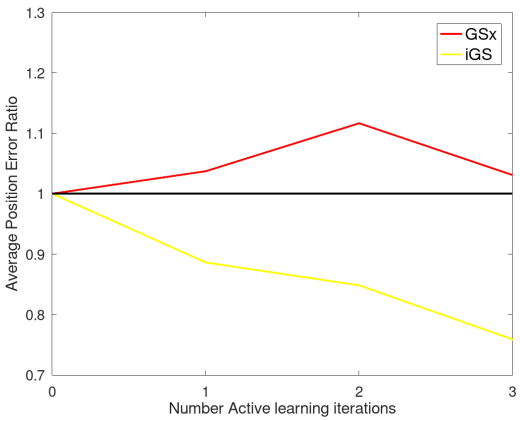


(a) Average Position error in millimeters with respect to the number of active learning iterations.

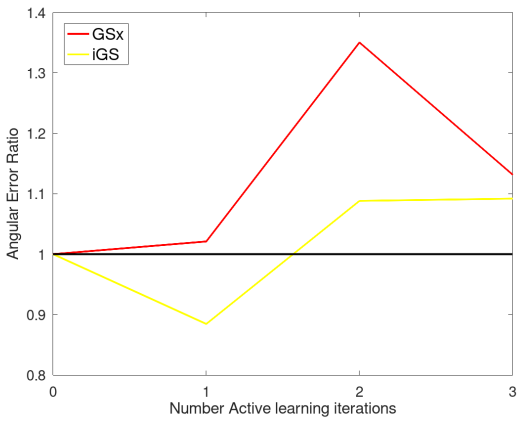


(b) Angular error in radians with respect to the number of active learning iterations

Figure 5.15: Results for different joint selection methods for the calibration routine for query-synthesis based AL and a batch size of 1000. In blue the joint configurations are selected randomly (baseline). In red the joint configurations are selected using GSx. In yellow the joint configurations are selected using iGS. The faded areas correspond to the standard error of the mean of the respective method.



(a) Ratio of Average Position error with respect to the number of active learning iterations.



(b) Ratio of Angular error with respect to the number of active learning iterations

Figure 5.16: Comparison of different joint selection methods for query-synthesis based AL and a batch size of 1000. Ratio of the errors for the AL methods in comparison with Random Selection. Ratio bigger than 1 (above black line) shows the superiority of the method opposed to random selection.

Even though, considering the performance, both types of AL can be considered similar, there are other factors that need to be considered. The pool based approach is much faster than query-synthesis since the optimization algorithm can be time expensive specially for higher batch sizes. Table 5.2 contains the average time needed for the calibration routines of the results for GSx described in this chapter. It is important to note that the main temporal difference between both methods reside in the sampling selection process, since for pool based AL this process is almost instant and the NN training takes the same time for both approaches. Thus, it is hard to identify which approach is the best. If time is not a constraint, then query-synthesis approaches are naturally the better fit for this specific task. But if one needs a faster solution to the problem, using pool based AL is more appropriate.

Table 5.2: Average time per calibration repetition for GSx.

Batch Size	AL iterations	Time per calibration for Pool based AL	Time per calibration for Query-synthesis based AL
250	5	22m28s	41m09s
500	5	42m34s	82m00s
1000	3	34m12s	82m06s

6

Conclusion and Future Work

Contents

This thesis proposed a framework for the calibration of 7-DoF right arm of iCub that combines the sample selection efficiency of Active learning methods with the learning capabilities of neural networks in order to improve performance. The framework was proposed for two types of AL, pool and query-synthesis based AL, which achieved similar performance results. This thesis also presents a comparison between multiple criterion for using AL in this specific task. The first is McD, an uncertainty based approach that uses MCDUE to estimate the uncertainty of a given input sample and select the samples with higher uncertainty as the most informative to train the NN model. The second is GSx, a diversity based approach that focuses on the geometric characteristics of the input samples, choosing the sample that is the furthest away from the previously selected samples in order to promote diversity in the input space. Finally, the third is iGS, a diversity based approach that chooses the samples that are furthest away from the previously selected samples in both the input and output space.

The results show that McD is the worst sample selection method, not being able to learn the forward kinematics of iCub. This might be because the method selects similar samples that have high uncertainty but provide redundant information when training the NN model, meaning that the selected samples are very close to each other in the input space and lack diversity. The results also show that iGS is not an appropriate AL method to improve the learning process for this task since, even though it is able to learn the forward kinematics of the arm, it is outperformed by random sampling selection. Finally, GSx is the only selection method that is able to outperform random selection when learning both the position and orientation of the end-effector. Thus, one can conclude that diversity seems to be a more suitable selection criteria when learning the forward kinematics of iCubs right arm.

Considering pool based AL, this thesis provides a small study on how the size of the pool affects the performance of the selection methods. The results show that, in general, using a bigger pool provides better results. The main reason for this is that a bigger pool is able to better represent the entirety of the input space of the task. This means that the sampling selection methods can be more precise when choosing a sample.

When comparing pool based AL and query-synthesis based AL, the results show that, both approaches present, in general, similar performances, except McD which increases the error for query-synthesis based AL. But there are other characteristics about these methods that are worth comparing. Pool based approaches require a pre-existing pool with a considerable amount of samples which is not optimal for this type of tasks when compared to query-synthesis that is able to produce its own samples. This also means that query-synthesis has access to the entirety of the input space when selecting a sample opposed to the pool approach that is restricted to the samples in the pool, which might only represent a part of the input space. Finally, when using a pool with 20000 samples, pool based approaches are faster when compared with query-synthesis AL which take almost double the time than pool based approaches when learning the forward kinematics of iCubs right arm.

6.1 Future Work

This section provides possible improvements to the proposed framework and potential research directions that may be interesting following this work.

Firstly, concerning the neural network model, the architecture can be improved to reach smaller values of error, since minimum position errors of $11mm$ and orientation errors of $3.43°$ are decent but not ideal. The loss function could also be adapted for the task of learning the position and orientation of a robot, for example following a similar approach to [34].

It could also be interesting to make the calibration autonomous using vision, instead of the DH parameters. By having the robot move its arm, according to a joint configuration, and using vision to sample the hand's position and orientation to build the datasets used for training the robot, similar to the work in [1]. This would remove the need of using the real DH-parameters of the robot in the labelling process, allowing the robot to select the most informative samples and sampling them based on its movement. This would make the calibration process more desirable in real-life situations.

Furthermore, it would also be interesting to apply the framework to the real robot to study how the framework performs in real world. Studying the effect of this framework applied to soft robots would also be interesting, since these type of robots would probably benefit the most from using a neural network to estimate their kinematics due to their complex kinematic chains with infinite DoF.

Bibliography

[1] Gonçalo Cunha, Pedro Vicente, Alexandre Bernardino, Ricardo Ribeiro, Plínio Moreno, "Online body schema adaptation through cost-sensitive active learning," January 2021.

[2] H. Yu and S. Kim, "Passive sampling for regression," *2010 IEEE International Conference on Data Mining*, pp. 1151–1156, December 2010.

[3] J. H. Dongrui Wu, Chin-Teng Lin, "Active learning for regression using greedy sampling," *Information Sciences*, vol. 474, pp. 90–105, February 2019.

[4] B. E. S. D. R. JONES, C. D. PERTTUNEN, "Lipschitzian optimization without the lipschitz constant," *JOURNAL OF OPTIMIZATION THEORY AND APPLICATION*, vol. 79, p. 157–181, October 1993.

[5] C. Von Hofsten, "An action perspective on motor development," *Trends in cognitive sciences*, vol. 8, no. 6, pp. 266–272, 2004.

[6] Karla Stepanova; Tomas Pajdla; Matej Hoffmann, "Robot self-calibration using multiple kinematic chains—a simulation study on the icub humanoid robot," *IEEE Robotics and Automation Letters*, vol. 4, pp. 1900 – 1907, February 2019.

[7] Adrian-Vasile Duka, "Neural network based inverse kinematics solution for trajectory tracking of a robotic arm," *Procedia Technology*, vol. 12, pp. 20–27, October 2014.

[8] Ahmed R. J. Almusawi, L. Canan Dülger, and Sadettin Kapucu, "A new artificial neural network approach in solving inverse kinematics of robotic arm (denso vp6242)," *Computational Intelligence and Neuroscience*, August 2016.

[9] K. S. Takamitsu Matsubara, "Active tactile exploration with uncertainty and travel cost for fast shape estimation of unknown objects," *Robotics and Autonomous Systems*, vol. 91, pp. 314–3262, May 2017.

[10] Antonio Loquercio, Mattia Segu and Davide Scaramuzza, "A general framework for uncertainty estimation in deep learning," *IEEE Robotics and Automation Letters*, vol. 5, pp. 3153 – 3160, April 2020.

[11] A. S. Evgenii Tsymbalov, Maxim Panov, “Dropout-based active learning for regression,” *Lecture Notes in Computer Science*, pp. 247–258, July 2018.

[12] Gang Zheng, Yuan Zhou, Mingda Ju, “Robust control of a silicone soft robot using neural networks,” *ISA Transactions*, vol. 100, pp. 38–45, May 2020.

[13] Thomas George Thuruthel, Benjamin Shih, Cecilia Laschi, Michael Thomas Tolley , “Soft robot perception using embedded soft sensors and recurrent neural networks,” *Science Robotics*, vol. 4, January 2019.

[14] João Damião Almeida, Paul Schydlo, Atabak Dehban and José Santos-Victor, “Sensorimotor graph: Action-conditioned graph neural network for learning robotic soft hand dynamics,” *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, July 2021.

[15] Annalisa T. Taylor, Thomas A. Berrueta, Todd D. Murphey, “Active learning in robotics: A review of control principles,” *Mechatronics*, vol. 77, August 2021.

[16] S. D. L. C. T. J. K. J. Zhihao Liang, Xun Xu, “Exploring diversity-based active learning for 3d object detection in autonomous driving,” *arXiv preprint arXiv:2205.07708*, p. 521–566, May 2022.

[17] Y. Gal and Z. Ghahramani, “Dropout as a bayesian approximation: Representing model uncertainty in deep learning,” in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[18] Ruben Martinez-Cantin; Manuel Lopes; Luis Montesano, “Body schema acquisition through active learning,” *2010 IEEE International Conference on Robotics and Automation*, May 2010.

[19] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, Xin Wang, “A survey of deep active learning,” *ACM Computing Surveys*, August 2020.

[20] Siqi Zhou and Angela P. Schoellig, “Active training trajectory generation for inverse dynamics model learning with deep neural networks,” *2019 IEEE 58th Conference on Decision and Control (CDC)*, December 2010.

[21] SX Yang, M Meng, “An efficient neural network approach to dynamic robot motion planning,” *Neural Networks*, vol. 13, pp. 143–148, March 2000.

[22] Olov Andersson, Mariusz Wzorek, Patrick Doherty, “Deep learning quadcopter control via risk-aware active learning,” *Thirty-First AAAI Conference on Artificial Intelligence*, February 2017.

[23] F. M. Mordechai Ben-Ari, “Kinematics of a robotic manipulator,” *Elements of Robotics*, vol. 1, p. 267–291, October 2017.

[24] L. V. G. O. Bruno Siciliano, Lorenzo Sciavicco, *Robotics - Modelling, Planning and Control*. Springer, 2009, vol. 91.

[25] R. Goldman, "Understanding quaternions," *Graphical Models*, vol. 73, p. 21–49, March 2011.

[26] Y.-B. Jia, "Quaternions and rotations," *Com S*, vol. 477, no. 577, p. 15, 2008.

[27] J. B. Diederik P. Kingma, "Adam: A method for stochastic optimization," *3rd International Conference for Learning Representations*, January 2017.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 56, pp. 1929–1958, October 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[29] J. J. R. . R. D. K. Robert Burbidge, "Active learning for regression based on query by committee," *Intelligent Data Engineering and Automated Learning*, p. 209–218, December 2007.

[30] A. F. O. M. B. B. J. D. Christoph Käding, Erik Rodner, "Active learning for regression tasks with expected model output changes," *British Machine Vision Conference*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52289772

[31] P. F. G. M. L. N. F. N. V. Tikhanoff, A. Cangelosi, "An open-source simulator for cognitive robotics research: the prototype of the icub humanoid robot simulator," *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, vol. 1, p. 57–61, August 2008.

[32] P. F. Giorgio Metta and L. Natale, "Yarp: Yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, March 2006.

[33] J. R. R. A. M. Donald R. Jones, "The direct algorithm: 25 years later," *Journal of Global Optimization*, vol. 79, p. 521–566, August 2020.

[34] A. K. R. Cipolla, "Geometric loss functions for camera pose regression with deep learning," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6555–6564, 2017.