Combining Active Learning with Neural Networks for Robot Self-Calibration

Daniel Nobre Chagas

***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.**

***Index Terms*—Robotics, Active Learning, Deep Learning, Deep Active Learning, Calibration.**

I. Introduction

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 the workspace can cause changes in the robots' model.

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. So, since deep learning has already provided good results in the learning domain, it should not be surprising that multiple researches such as [1], [2], [3] 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. 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 would result in an inefficient exploration of the robots workspace during the calibration.

Fig. 1. Visual Representation of iCub Simulator.

In contrast to the standard methods of training deep learning models, active learning aims to select the most useful samples of an unlabeled dataset to reduce the labeling cost while still maintaining performance. 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 [4]. 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.

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 like [5], [6], [7] and [8] can be used to integrate AL with DL frameworks.

A. Problem Statement and Contributions

This work 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 pose of the end-effector.

To solve this problem, this work 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 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. Fig. 2 depicts a flowchart that describes the general process of the proposed framework.

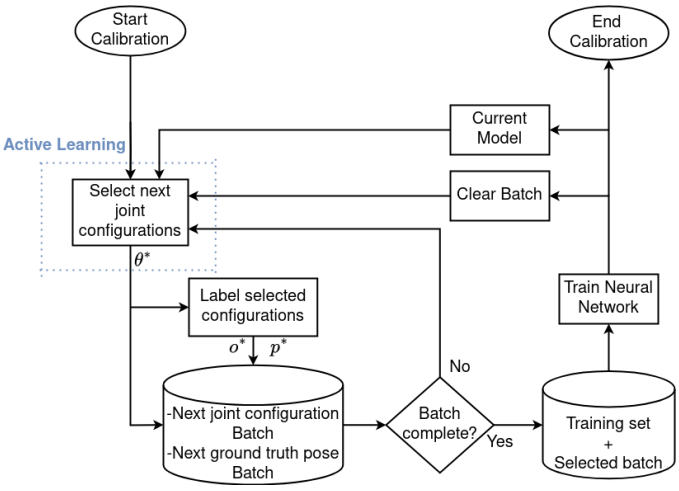


Fig. 2. General calibration flowchart. $\theta*$ is the joint configuration chosen by the AL process at a given iteration while $p*$ and $o*$ are the corresponding position and orientation, respectively.

Fig. 2 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 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 work 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.

II. Related Work

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 [1] where the authors use multiple kinematic chains for self-calibration of the iCub robot.

Focusing on the use of neural networks for the estimation of the kinematics of a robot, in [2] and [3], NN are used to solve the inverse kinematic of robotic arms. 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 DoF. Works such as [9], [10] and [11] employ different types of NN to model the kinematics of soft robots. 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.

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 [12] a general overview of the use of AL in robotics is given. Yu et al [8], [7] proposed new active learning approaches for regression based on the geometric characteristics of the samples, promoting diversity between the selected samples. A similar concept was applied in [13] for 3D object detection in autonomous driving. Some AL approaches are based on the uncertainty of the model and works such as [5] and [14] propose different methods for the estimation of uncertainty in deep learning models. Active learning approaches like the ones proposed in [15] and [16] were used to improve the learning process for kinematic representations for humanoid robots, but lack the flexibility provided by neural networks.

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 [17] gives a general overview of the frameworks, challenges and applications of combining Deep Learning with Active Learning. Works such as [18], [19], [20] and [6] provide some examples on how combining both frameworks can improve the learning performance.

III. METHODS

This section main goal is to provide and overview of the main methodologies used to implement our approach. Section III-A introduces neural networks as the core learning technique for the robot model and, in order to enhance the learning process, section III-B introduces various Active learning techniques based on diversity and uncertainty estimation.

A. 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. Since this work focuses on predicting the position and orientation of the end-effector based on the joint configurations, the problem it aims to solve is a regression task. The framework uses Multilayer Perceptrons (MLP) composed of various layers with multiple nodes. Considering a node with n inputs, its output is given by

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

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.

B. 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 [17]. There are three main AL approaches based on how these samples are generated: i) Pool based, ii) Query-synthesis based and iii) Stream based AL. This work focuses on the first two, comparing the application of both methods to learning the kinematics of iCubs right arm for different AL methods.

In general, AL approaches select the next joint configuration, θ*, to train the NN is computed following

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \boldsymbol{P}}{argmax}\ C(\boldsymbol{\theta}). \tag{2}$$

where $C(\cdot)$ is the utility function that represents the trust in the model. In the case of query-synthesis approaches, $\boldsymbol{P}$ corresponds to the whole workspace of the robot. While, for pool based approaches $\boldsymbol{P}$ is a discrete and finite set given by

$$\boldsymbol{P} = \left\{ \left(\boldsymbol{\theta^{(n)}} \right), n = 1, ..., N \right\}. \tag{3}$$

There are multiple criteria for deciding what samples are more informative, for example, model uncertainty[6], diversity [7], [8], query-by-committee [21] and expected model change [22]. 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 criteria is reflected on the utility function $C(\cdot)$. This work uses three different criterion, two based on diversity and one based on uncertainty.

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

1) Greedy Sampling on input - GSx: Greedy Sampling on Input is proposed by [8] 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. 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.

$$d^x_{nk} = dist\left(\theta_k, \theta_n\right), \tag{4}$$

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

$$dist\left(\theta_k, \theta_n\right) = \left\|\theta_k - \theta_n\right\|, \tag{5}$$

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

$$d^x_n = \underset{k}{min}\ d^x_{nk} \tag{6}$$

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

$$\theta^* = \underset{n}{max}\ d^x_n \tag{7}$$

Summing up, GSx encourages diversity among the selected samples by choosing the samples that are the farthest away from the already selected samples. 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.

2) Improved Greedy Sampling - iGS: Improved greedy sampling iGS was proposed in [7] and considers not only the diversity in the input space, provided by GSx, but also diversity in the output space. GSx doesn't 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.

Considering a similar setup to the one in GSx 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 (4). 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

$$d_{nk}^{y} = \left\| \mathbf{y}_k - f(\theta_n) \right\|, \tag{8}$$

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

$$d_n^{xy} = \min_k \, d_{nk}^{x} d_{nk}^{y}. \tag{9}$$

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

$$\theta^* = \max_n \, d_n^{xy} \tag{10}$$

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.

3) Monte Carlo Dropout - McD: Dropout is a regularization technique for addressing the problem of overfitting when training neural networks which can also be used to estimate model uncertainty. 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.

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 [14] and [6].

As it was shown in [14], 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 models uncertainty of an output i by computing the standard deviation of these predictions.

$$\sigma^{(i)} = \sqrt{\frac{1}{T-1} \sum_{k=1}^{T} (y_k^{(i)} - \overline{y}^{(i)})^2}, \tag{11}$$

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 [6] 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.

IV. Experimental Setup

This section offers a review of the implementation details of this work. Section IV-A introduces the simulator used, sections IV-B and IV-C present the implementation details of the NN model and the AL approaches, respectively. Section IV-D introduces the calibration metrics and, finally, section IV-E presents the calibration framework for the different AL approaches.

A. iCub Simulator

All the work presented is based on simulations of the humanoid robot iCub. The open-source simulator, presented in [23], is used 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.

It uses the middleware YARP [24] (Yet Another Robot Platform) to create a bridge between the user and the simulator. 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 commands to move each of the individual joints of the iCub's kinematic chains.

B. Neural Network Implementation

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

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, which is seven. The output is comprised of the position, defined by three dimensions, and orientation, which is represented by an unit quaternion which is defined by four dimensions, of iCub's right hand. Thus, the output has seven nodes. The full 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. The full model is represented in Fig. 3. The hyperparameters used during the training process are summarized in table I. It is important to

[1]https://github.com/robotology/icub-main
[2]https://pytorch.org/cppdocs/

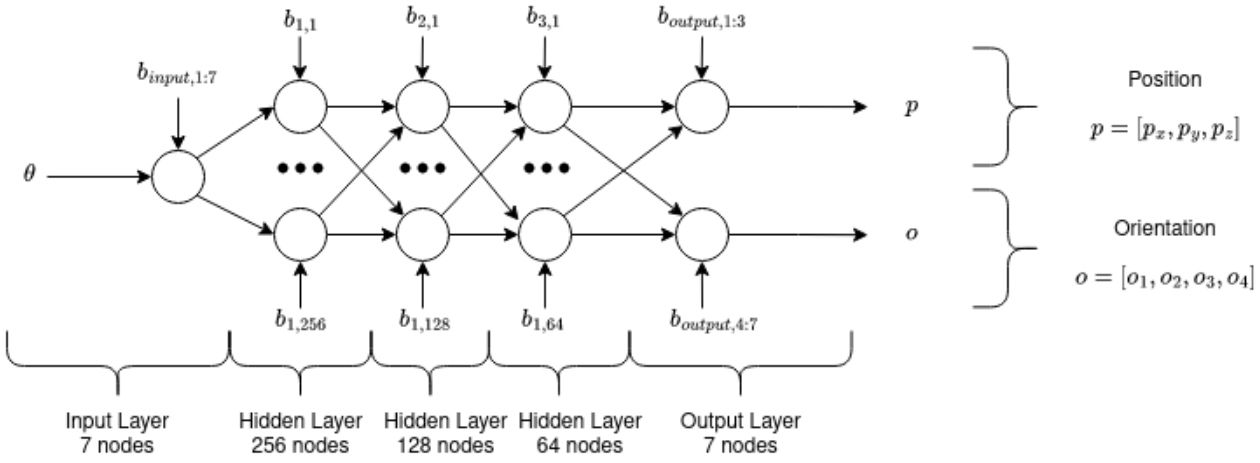


Fig. 3. Compact representation of the NN model used.

note that the dropout rate is only used to estimate the model uncertainty for McD. For learning, the NN uses the L1 loss function and Adam optimizer [25], which is an extension of stochastic gradient descent.

TABLE I
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

$$x_{stand}^{(i)} = \frac{x^{(i)} - \mu_x^{(i)}}{\sigma_x^{(i)}}, \tag{12}$$

and the outputs using

$$y_{stand}^{(i)} = \frac{y^{(i)} - \mu_y^{(i)}}{\sigma_y^{(i)}}, \tag{13}$$

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.

The NN model is also pre-trained using 3000 samples and the hyperparameters summarized in table II. It is important to note that this initialisation is important since, without it, the framework would not be able to converge.

TABLE II
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

C. Active Learning Implementation

Regarding GSx and iGS, we select the first sample randomly to induce variability in different runs of the methods.

Concerning McD we choose to perform $T = 25$ forward passes to estimate the uncertainty of a given sample. It is also important to note that (11) 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.

Pool based AL requires a pool of unlabeled data where the next samples are selected from. 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 work uses two different pools with 10000 and 20000 samples, respectively.

In order to select the most informative samples, query-synthesis AL needs an optimization algorithm to solve (2), so that a new joint configuration can be synthesised, labeled and used to train the NN model. The DIRECT [26] algorithm is used for solving (2). It is a global and deterministic optimization algorithm for minimizing/maximizing multivariate black-box type utility functions with upper and lower bounds on the domain.

D. Comparison Metrics

In order to measure the error of the predictions generated by the proposed models, consider a test dataset composed of

$N = 1000$ samples of randomly generated joint configurations and the corresponding positions and orientations of the end-effector. 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.

1) Average Position Error: 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

$$\frac{1}{N}\sum_{n=1}^{N}\|\widehat{p}_n - p_n\| \ [mm], \tag{14}$$

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

2) Angular Error: Considering, N samples, the angular error is computed using,

$$\frac{1}{N}\sum_{n=1}^{N} arccos\left(|\widehat{q}_n \cdot q_n|\right) \ [rad], \tag{15}$$

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 also important to note that the range of values of this metric is $\left[0; \frac{\pi}{2}\right]$ (radians). The unit quaternion, estimated by the NN can have values larger than one. When this happens, this value is set to one. The values for the quaternion estimatives used in this metric already take this into consideration.

E. 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. There can be slight deviations to the calibration routine represented in Fig. 2.

For pool based AL, the selected sample should be removed from the pool after being selected to avoid it being selected again. Considering GSx and iGS, both methods need to update the pool distances after selecting each sample since they're dependent on the already selected samples geometric location on the input space. This is not true for McD which only needs to compute the uncertainty of all samples in the pool once and then select the M most uncertain samples, where M is the batch size. Regarding query-synthesis based AL, the calibration process is exactly the one depicted in Fig. 2 for all methods.

V. Simulation Results

The results presented in this section correspond to the calibration routines presented for four different joint configuration selection methods: i) Random (R), ii) Greedy Sampling on input (GSx), iii) Improved Greedy Sampling (iGS) and iv) Monte Carlo Dropout (McD). 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, for pool based AL, the results presented use a pool with 20000 samples, while the results for comparing the how the pool size affects the selection methods use two pools, the same with 20000 samples and another with 10000.

A. Pool based Active Learning

For each method, Fig. 4 show the evolution of the average position error and the angular error with respect to the number of AL iterations for a batch sizes of 1000. By analyzing these figures, the first relevant observation is that, in the first iteration, McD increases the error. Afterwards, 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.

From observing Fig. 4, one can conclude that the only AL method to outperform R in learning both the position and orientation is GSx. iGS provides interesting results. Even though it underperforms in comparison to R and GSx when learning position of the end-effector, one can observe that, it is able to outperform R when learning the orientation of the end-effector in the later iterations, 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. 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 (2) 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, of learning the forward kinematics of iCub's right arm using a combination of AL and NN.

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. Fig. 5 show the results for the calibration routines using the AL methods for both pools and for a batch

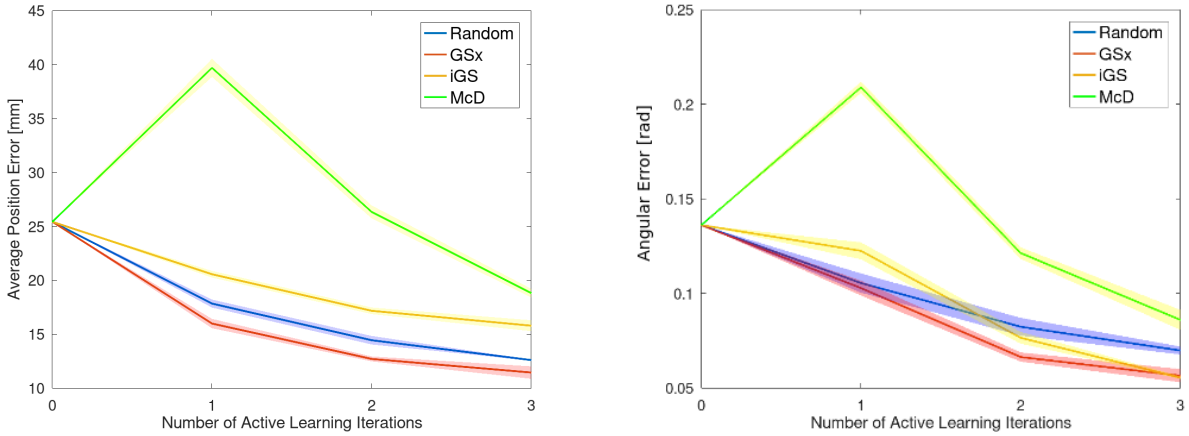
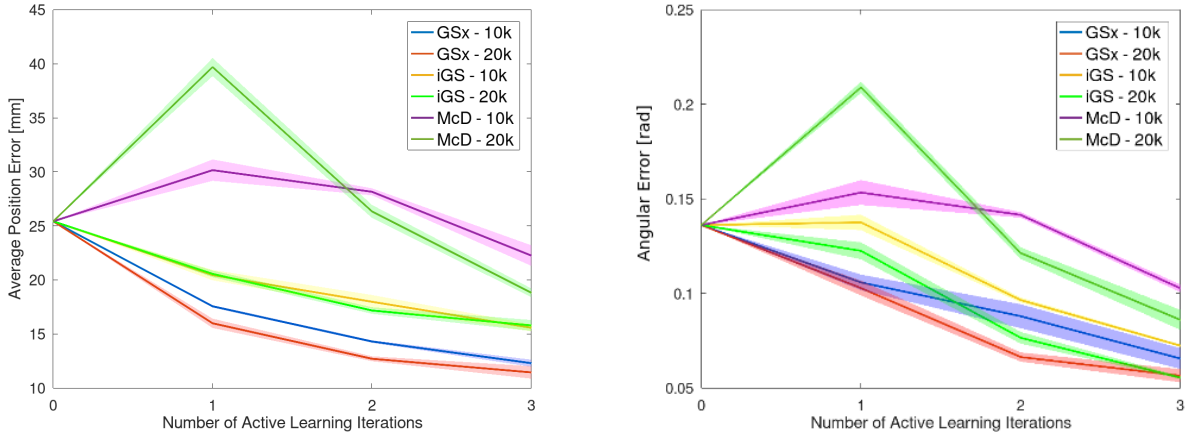


Fig. 4. Average position error (on the left) and angular error (on the right) results for different joint selection methods for the calibration routine for pool based AL and a batch size of 1000. In blue the joint configurations are selected randomly (baseline). In red the joint configurations are selected 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) (b)

Fig. 5. Comparison of joint selection methods for pool sizes of 10000 and 20000 samples and a batch size of 1000 for average position error (on the left) and angular error (on the right). 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.

size of 1000. The legends of the figure indicate the method being used followed by the size of the pool.

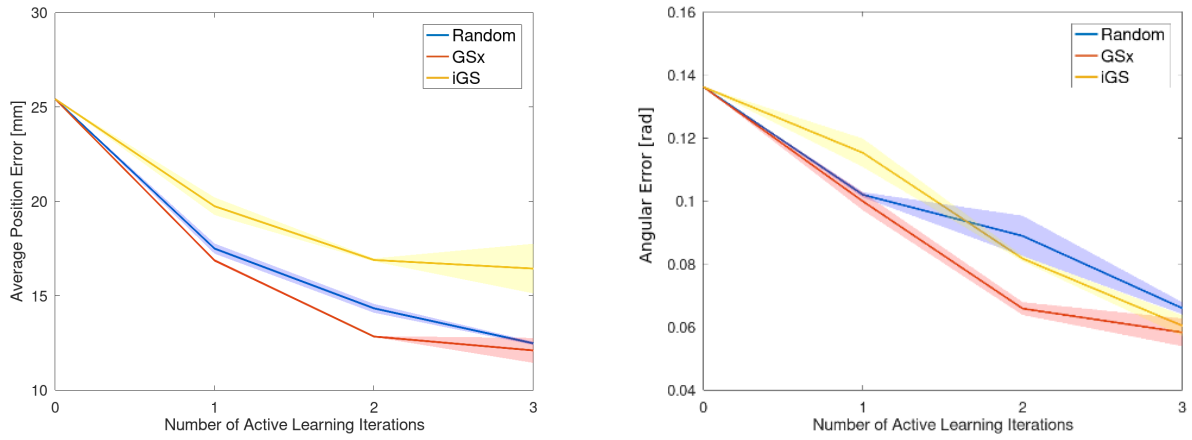
The first observation is that McD initially presents worse results for the bigger pool, but around the second iteration the results for the bigger pool outperform the smaller pool. This is true for both metrics. The other two methods always present better performance for the bigger pool, in comparison to the smaller pool.

Thus, one can conclude that having a bigger pool improves the results for the AL methods. This is intuitive since 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. 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.

B. Query-synthesis based Active Learning

Fig. 6 shows the evolution of both metrics with respect to the number of AL iterations for the selection of a batch with 1000 samples for all methods except McD. This is because McD increased the error infinitely. A reason that explains why this happened resides on the fact that McD could selected similar samples with high uncertainty that provide redundant information.

Performance wise, the rest of the results are very similar to the ones presented in section V-A. GSx is the best sample selection method, being the only one capable of outperforming R and iGS presents the same behavior, performing better when learning the orientation than when learning the position.



(a) (b)

Fig. 6. Average position error (on the left) and angular error (on the right) 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.

C. Pool based Active learning vs Query-synthesis based Active learning

As mentioned before, from comparing the results for pool and query-synthesis based AL, they 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.

Even though, considering the performance, both types of AL can be considered similar, there are other factors that need to be considered when comparing both types of AL to the task at hand. The first is that pool based AL needs a dataset (pool) with a considerable amount of data to train the model that needs to be built before the calibration routine, which is not ideal when compared to query-synthesis AL that is able to produce its samples during the calibration routine. On the other hand, the pool based approach is much faster than query-synthesis since the optimization algorithm can be time expensive. Table III contains the average time needed for the calibration routines of the results for different batch sizes using GSx. 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.

TABLE III
AVERAGE TIME PER CALIBRATION REPETITION FOR GSX.

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

VI. CONCLUSIONS

This work 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 results show that pool and query-synthesis based AL achieved similar performance results, but query-synthesis approaches take twice the time of pool based approaches, for a pool with 20000 samples. The results also show that McD is the worst sample selection method, not being able to learn the forward kinematics of iCub and that iGS it 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 work provides a 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.

Regarding future work, 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 [27].

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 [16]. 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.

REFERENCES

1 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.

2 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.

3 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.

4 Takamitsu Matsubara, K. S., "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.

5 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.

6 Evgenii Tsymbalov, A. S., "Dropout-based active learning for regression," *Lecture Notes in Computer Science*, pp. 247–258, July 2018.

7 Dongrui Wu, J. H., "Active learning for regression using greedy sampling," *Information Sciences*, vol. 474, pp. 90–105, February 2019.

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

9 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.

10 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.

11 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.

12 Annalisa T. Taylor, Thomas A. Berrueta, Todd D. Murphey, "Active learning in robotics: A review of control principles," *Mechatronics*, vol. 77, August 2021.

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

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

15 Ruben Martinez-Cantin; Manuel Lopes; Luis Montesano, "Body schema acquisition through active learning," *2010 IEEE International Conference on Robotics and Automation*, May 2010.

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

17 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.

18 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.

19 SX Yang, M Meng, "An efficient neural network approach to dynamic robot motion planning," *Neural Networks*, vol. 13, pp. 143–148, March 2000.

20 Olov Andersson, Mariusz Wzorek, Patrick Doherty, "Deep learning quadcopter control via risk-aware active learning," *Thirty-First AAAI Conference on Artificial Intelligence*, February 2017.

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

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

23 V. Tikhanoff, P. F. G. M. L. N. F. N., "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.

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

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

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

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