Master Thesis

Comparison of Parametric and Non-parametric Learning Methods for the Inverse Dynamics Modeling of the Arm of the iCub Humanoid Robot

Chalikonda Prabhu Kumar

Master in Computer Vision and Robotics (MSCV4)
University of Burgundy, France

Host by

Istituto Italiano di Tecnologia, iCub Facility
Genova, Italy

Supervised by : Giorgio Metta

A Thesis Submitted for the Master Degree in Computer Vision and Robotics (Mscv 4)

· 2014 ·

Abstract

In recent years, learning models from data has become an interesting tool for robotics. Acquiring accurate models of dynamical systems is an essential step in many applications for example safe operation in unstructured dynamic environments. Aanalytical models for robot dynamics often perform suboptimally in practice due to various non-linearities and difficulty of accurately estimating the dynamic parameters. Machine learning techniques are less sensitive to these problems and therefore they are an interesting alternative for modeling robot dynamics. The subject of my thesis is comparision of parametric and non-parametric learning methods for the inverse dynamics modeling. Non-parametric methods will approximate the function describing the relationship between joint trajectories (i.e. joint positions, velocities and accerlations) and the joint torques of the rigid body dynamic model. The qualitative measure for the predicition is Root Mean Square Error (RMSE) for forces and torques in three dimensions and influence of the data supplied (i.e. in Batch or Incremental) on RMSE error. In parametric models, we aim to identify the small set of significant parameters of the equations whose form is given to the learning method.

Contents

List of Figures

Acknowledgments

I would like to thank for all faculty who thaught me during two years of my master course in both UB and UdG. I am really greate ful to my family & happy who gave me support during this two years of study. It is my pleasure to work in IIT, iCub Facility under Prof. Giorgio Metta. I would like thank Silvio & Raffaello PhD students at IIT, helped me during my thesis time and clarifying everything with great patience.

Chapter 1

Introduction

Current trends in robotics research are mainly aimed at designing and implementing systems able to perform advanced tasks in unstructured environments, possibly along with humans. For example, robots used in applications ranging from domestic chores, healthcare, entertainment, dexterous and safe robots able to autonomously carry out tasks in unstructured environments. These environments often cannot be predicted in all relevant details, and therefore many researchers have contributed to conceive robots that are able to adapt to changing conditions and to learn by themselves from their experience. Traditional approach to robotics, consisting of precisely characterizing the robot, its environment, and the desired task, is rendered infeasible in this novel setting.

The field of robotics is facing significant challenges in the development of a future generation of robots, both in terms of suitable mechanics and morphology as well autonomous and intelligent behaviors. In order to overcome these situations the study of robots is necessary, that are able to perform advanced tasks, changing conditions and to learn by themselves. This strategy is particularly opposite, If the environment cannot be foreseen in all relevant details, either due to its complexity and dynamic change. Learning mechanisms allow advanced robots to act successfully under such circumstances and have shown better performance with respect to traditional analytic approaches in a variety of applications.

Machine learning may allow avoiding that all possible scenarios need to be programmed, but rather learned by the system during operation. In the literature, there are attempts in creating learning algorithms which enable the robot to learn autonomously complex skills ranging from task imitation to motor control to interacting with human actors. In fact, learning is not easy to implement in machines, there are wide variety of learning algorithms like supervised, unsupervised, reinforcement learning, etc. Every learning algorithm is used for the specific purpose. For example, reinforcement learning requires more trials and data that one can generate in life time of a robot, and black box imitating learning can reproduce the desired behavior. Thus, it is an essential to study how the basic, underlying mechanisms of the world can be learned. This kind of approach is commonly referred to as model learning.

In robotics research, methods to learn models from data have become interesting tools, as they allow straightforward and accurate model approximation. There are many motivations for rising this interests, mainly because it is hard to obtain accurately analytical models due to complexity of modern robot systems and their presence in unstructured and uncertain environments. Model learning can be a useful alternative to manual pre-programming, because the model is estimated directly from measured data. Unknown non-linearities are taken into account, while they are neglected

by the standard physics-based modeling to a larger state space and to adapt the models for time dependent changes, on-line learning of such models is necessary. Generally, a model possesses essential information about the system and it describes the influence of an agent on this system. Thus, modeling of a system inherently connected with the question, how this model can be used to manipulate i.e. to control the system. Model learning is an efficient tool in many scenarios such as inverse dynamics control, inverse kinematics, robotic manipulations, autonomous navigation and robot locomotion.

We are interested in building a model of the inverse dynamics of a robot, followed by learning. Learning model varies depending on the application. Learning the inverse dynamics in Non-Parametric and Parametric modelling. From theoritical point of view, we are interested in learning the same thing: the robot dynamics. In Non-parametric modeling we don't estimate exactly the parameters of the commom Newton- Euler or Lagrangian formulation of the dynamic equation rather we aim to get an overall accurate representation of such equation as output (black-box learning). This means the realtionship between joint trajectories (i.e. joint positions, velocities and accelerations) and joint torques. While on the other hand for parametric modeling instead, we aim at the learning parameters of the equation whose form is given to the learning method. In this parametric models we identify the smallset of significant parameters (i.e. mass, center of mass & inertial matrix). To accomplish these tasks machine learning algorithms, are viable solution. Using GURLS package for learning algorithms especially in Non parametric modeling for both linear and non linear models.

Chapter 2

Problem Definition

The demand for robotics is increasing day by day, especially with reference to preprogrammed robotic systems which mean to act in a structured environment. In designing, robots more efficiently and for safe operations in unknown environment is becoming an challenging task in robotics & interest in this domain is increasing day to day. Aiming to this, design of robots are made to learn by themselves from their experience in order perform certain tasks in an unknown environments. Robot learning is a research field at the intersection between machine learning and robotics. Robot learning studies the techniques allowing a robot to obtain novel skill or to adapt to the environment through learning algorithms. Learning includes rich varieties of techniques that use previous experience to enable more effective performance. Depending on the task, the designer must decide what should be learned, when learning should occur, computational means to implement learning, and how much a priori knowledge should be supplied. Learning can be carried out either through autonomous self exploration or through the guidance (for example learning by imitation). Skills that are targeted by learning algorithms include sensorimotor skills such as locomotion, grasping, active object categorization and interacting with humans such as joint manipulation of objects with an human peer etc. The purpose of this learning and adaptation will allow the robot to make accurate predictions for the safe operations in an unknown environment. The purpose of accurate predictions in both internal and external environment are necessary for the safe behavior and skillful control. In order to get accurate predictions machine learning techniques are viable solution in implementation.

In reality, robots should be designed in such a way to act safely in dynamic unstructured environments. To avoid damage,special sensors are usually applied for identifying interactions occur in the environments. Classical approaches to manipulation exploit a force/torque (FT) sensor placed on the end-effector, where most of the interactions can be identified. There is a drawback in using this configuration like measuring of external forces acting on the other parts of the arm is not possible with this configuration. Taking its weight and size into account it is not possible to place the sensor on the end-effector. An alternative solution exists to overcome these drawbacks that is to place the sensor at the base of the manipulator or along the kinematic chain (e.g. after shoulder, as in iCub). By embedding the F/T sensor after shoulder, it can measure both internal and external forces, and depending on gravitational, Coriolis and inertial forces. This design allows the robot to detect interactions with the environment not on the end-effector for performing tasks like touching and grasping an object, but also the whole arm for example hitting obstacles, being stopped by human agents during motion. To detect the external contribution of the forces, the manipulator dynamics

must be compensated, i.e. internal forces must be known, modeled or estimated.

Multiple approaches are used to estimate these internal forces. Namely, functional estimation and model based estimation. In functional estimation,the analytical model describing the physics of the system or its significant properties. Model-based estimation relies on the availability of model, it can be done only if there is an availability of kinematic and dynamic parameters are availabe. For this purpose in the literature rigid multi-body dynamic modeling is generally used and some or all parameters are identified, to improve the model accuracy. In this way, overall model accuracy is primarily limited by effects, which model doesn't take into account like gearbox, backlash. To obtain the approximate internal dynamic model, supervised machine learning algorithms are the solution from a set of training samples. This approach is preferred when explicitly modeling all possible non-linear effects are high. The main drawback of this supervised machine learning approach is the need for a rich and significant training set. In addition to this, it is necessary to perform offline training, due to the computational requirements of this learning method. In contrast, model-based approach only needs to identify a small set of significant parameters; this identification technique requires much fewer data and computational resources and therefore they can perform efficiently in online.However, due to complexity of the positional error of robots it is common practice to approximate the error rather than modelling it explicity by developing parametric models. Functional approximation theory is a well established mathematics disicipline, which provides a variety of approximate models and methods. For most of practical applications they are generally assumed to be adequate.

My primary work is to investigate supervised learning methods for the estimation of internal forces in a robotic arm, which is equipped with a six-axis F/T sensor inside the kinematic chain. The qualitative measure for the prediction is the root mean square error (RMSE) for the forces and torques in three dimensions. In our context, the minimum amount of external forces that the robot can detect is directly proportional to the magnitude of this estimation error. In fact, this value is important for safe operation and interaction with the environment. We know machine learning methods are advantageous, one of them is to get benefit from larger data sets; while on the other hand model-based techniques should be more insensitive to the size of the training set. Furthermore, we worked on how data supplied either in batch or incremental, and how it will show the effect on predictions. This is important in the field of humanoid robotics where smooth motions are prefearble.

Second part of my thesis is parametric modeling i.e. to get the inertial parameters of iCub humanoid left hand. Since we do not have ground truth values for these inertial parameters i.e. mass, center of mass and inertia matrix, so I focused on how the regularization parameter will influence the inertial parameters. The motivation for regularization is to prevent overfitting. From non-parametric modeling we discuss in later sections optimal regularization parameter will reduce rmse error. With this evidence, using accurate regularization parameter we can obtain inertial parameters closer to the true ones.

Chapter 3

State of the Art

Robotics research are mainly aimed at designing and implementing systems able to perform advanced tasks in unstructured environments, possibly along with human peers.If the F/T sensor is placed along the kinematic chain proposed in [39], this configuration F/T sensor measures both external and internal forces, the latter being the ones depending on gravitational, Coriolis and inertial forces[6].This solution allows the robot to detect interactions that occur in the surrounding not only on the end-effector, but on the whole arm. To detect this external contribution of forces acting on the robot the internal forces must be known, estimated or modeled. There are several approaches to estimate these internal forces. Functoinal estimation can be done using an analytical model which describes the physics of the system. Analytical models for robot dynamics will perform suboptimally in practice due to the presence of various non-linearities in the system. Machine learning techniques are less sensitive to these problems, therefore they are interesting alternative solution for these problems. In this we are approximating the function describing the relationship between joint trajectories $q, \dot{q}, \ddot{q}$ [6] and joint torques τ. It can be seen as regression problem, however we have to estimate the relation between the inputs and outputs. To approximate the function for linear and non-linear models different approaches can be used proposed in the literature. In this main objective is to find the optimal function that will approximate the overall accurate representation of such function [1][2][3]. In this learning method that combines a least square algorithm with linear and non-linear models. In Linear models depending on the data there may be possiblity of overfitting or underfitting. To prevent from overfitting there should be a tradeoff between inputs and outputs. This parameters is called as regularization parameter λ. For selecting this parameter there are different approaches in machine learning field [7].The optimal value for λ gives accurate predictions. To esure the λ is optimal, lease squares error is reasonable choice[1]. If this system of linear equations is well posed this gurantees unique optimal solution, this technique proposed by Tikhonov[8]. Since then regularized optimization is used in many fields [9],[10],[11],[1]. Accuracy of the predictions depends on how data was supplied to the learning algorithm. In batch leanrning the covariance matrix will be stable, performance of the algorithm is promising in most of the cases. In case of incremental learning, there will be numerical instability in covariance matrix, because sherman morrison update[22] will use the round off errors.

In reality linear models are restricted or limited in use. To deal with non-linear system mapping the input features into feature space. In order to map this Hilbert is the best choice because of the possibility of linear product of spaces. There are different types of non linear feature mapping known as basis functions, used in practice like

Radial Basis Function, Polynomial basis function, splines, etc [15]. These are computationally inefficient. So kernel trick was proposed by Aizerman[16], it is computationally fast and able to use infinite dimensional features. To get the optimal solution for this dual formulation representor therom is used proposed by Wahba [17] generalized by scholkopf[18].Most widely used kernels are polynomial and RBF kernel, because of high computational cost & less capable to deal with large dataset they are limited. Rahimi & Techt[19] demonstrated that RBF and shift invariant kernels can be approximated to an arbitrary precision using finite dimensions random features mapping. However, they utilized Bochner's theorem[20]. In addition to this any kind of kernels can be used in non-linear system if they satisfies mercer's condition, because they avoids explicit expression for non linear mapping. In incremental learning non-linear case using Sherman Morrison is not a feasible in reality, due to its numerical instability [22]. This problem is alleviated to large extent by updating the Cholesky factor[23] also known as QR algorithm in the field of adaptive filtering[21]. The regularization parameter is to prevent singularity of the covariance matrix by increasing its eigen values by Lo Gerfp[24]. Subsequent rank update increase the smallest eigen values which is popularly known as interlacing theorem[25].

The other model-based approach depends on the availability of a model of the robot [40] and is used only if the kinematic and dynamic parameters are known. For this purpose rigid multi-body dynamic modeling is generally used and some or all parameters can be indentified [41]. Model-based approaches need to find a small set of significant parameters. These parameters can be estimated using arms joint torques and forces along with positions, velocities & accelerations of joint[26][27]. Using spatial vector algebra (i.e. 6D vectors) the advantage is that they cut the volume of algebra[28][29]. The inverse dynamics of general kinematic tree is calculated using Recursive Newton Euler Algorithm[29].For torque estimation Fumagalli et al [34] proposed a theoretical framework that exploits embedded force/torque sensors to estimate internal and external forces acting on the floating base dynamics link with multiple branches. Recursive Newton-Euler steps can prove that relocation relies solely on inertial parameters[35] & inertial parameters space affects the dynamic equation [36].By considering only torque and fixed base to obtain the base parameters were presented by Kawasaki et al [37].More general approach to determine the identifiable subspace, with no assumptions on the regressor structure or on kinematic was proposed by Gautier [38].

Chapter 4

Methodology

4.1 Non-Parametric Modeling

4.1.1 Overview

Machine learning is termed as field of study that computers the ability to learn without being explicitly programmed by *ArthurSameul*. There are many kinds of machine learning algorithms: like supervised, unsupervised, reinforcement learning, recommender systems etc. For my thesis I am using supervised learning algorithms. Figure 4.1 shows the overview in the field of machine learning and, in blue, the areas involved in the present thesis.

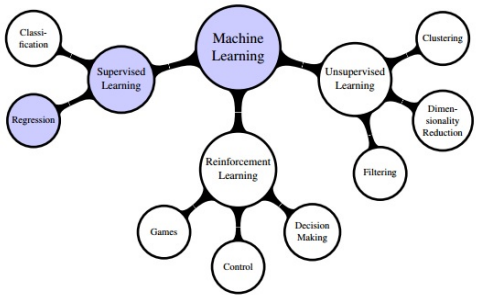

Figure 4.1: overview within the field of machine learning

Brief overview of learning algorithm is as follows and depicted in Figure 4.2 . Assume the learner is given following data.

1. Set of input features $X_1, X_2....X_n$;
2. Set of output variables $Y_1, Y_2.....Y_n$;
3. Set of training samples, where the values of input features and target features are given for each example
4. Set of test samples, where the value of each input feature is given

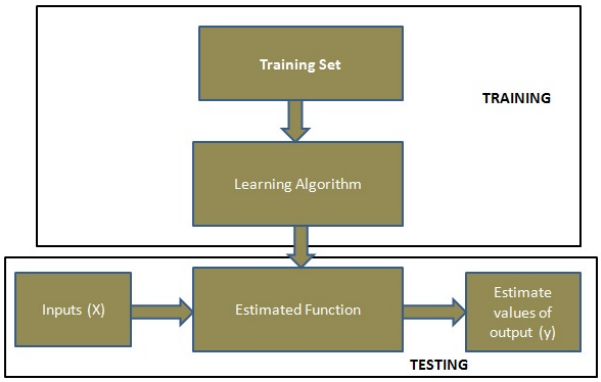

Figure 4.2: learning algorithm in brief

The goal in machine learning[1] is to use these observations to learn a function

$$f : \mathcal{X} \to \mathcal{Y}$$

such that $f(x)$ is a good approximation of the output y for an arbitrary input x. Obviously, in order to find such a function we must require that the collected observations are actually representative for possible future observations. In statistical learning theory, this is guaranteed by requiring that the observations are independent and identically distributed (i.e. according to a fixed but unknown distribution P on $(\mathcal{X} \times \mathcal{Y})$. This joint distribution of inputs and outputs can be decomposed as

$$P(y, x) = P(y|x)P(x)$$

One can recognize two phases in generating samples : first, an input x is generated according to marginal/prior probability $P(x)$; second, the corresponding output y is generated according to the conditional probability $P(y|x)$ given the input x. These probabilities imply that (1) control over generation of samples is not possible (2) mapping from inputs to outputs is stochastic rather than deterministic.

Learning methods use these observations to produce a function/hypothesis that captures the relationship between inputs and outputs, or in formal terms

$$x \times y \to f$$

Approximating performance of a function f is measured at the hand of non-negative loss function $L(y, f(x))$, where x and y are inputs and outputs. The optimal function $f*$ can therefore be defined as the one that minimizes the loss over the weighted space of inputs and outputs. The idea is to find a function f, that minimizes the expected risk Vapnik, 1995 [2] and MLCC2014 [3], defined as

$$R_{exp}(f) = \int x \times yL(y, f(x))dP(y, x) \tag{4.1}$$

From the above equation direct minimization of the integral is not efficient and infeasible, since the joint distribution $P(y, x)$ is unknown. The only available information regarding the process is the set of observations. It is reasonable

to use average loss on these observations.

$$R_{emp}(f) = \frac{1}{m}\sum_{i=1}^{m} L(y_i, f(x_i)) \tag{4.2}$$

which is known as the empirical risk. Although this function clearly minimizes the empirical risk in eq 4.2 it will almost surely perform poorly in terms of expected risk in eq 4.1. This phenomenon is known as overfitting, in which a learning method produces a function that models the noisy observations too closely. The problem of overfitting can be alleviated by restricting the optimization to a limited function space F. To prevent overfitting F can be choose sufficiently small to get satisfactory results. Unfortunately, identifying an appropriate space F requires intimate knowledge of $P(y, x)$ and generalization performance may suffer if F is chosen too small or restrictive. It is therefore more desirable to choose F sufficiently large and to penalize complex functions, which is depicted in the graph below Figure 4.3

We can do one thing in this case, we can minimize the regularized risk functional

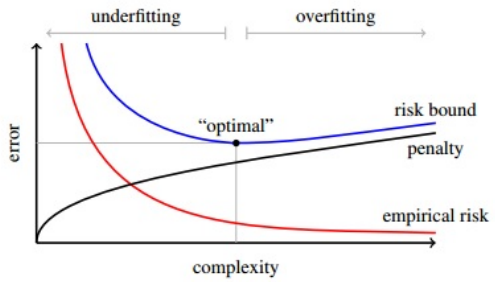

Figure 4.3: overfitting and underfitting of model complexity

$$R_{reg}(f) = R_{emp}(f) + \lambda G(f) \tag{4.3}$$

such that f belongs to F. Where $G(f)$ is a regularizer that penalizes the complex function and λ is the a constant that regularizes the tradeoff between both terms. There is a broad range of choice for choosing the regularization term $G(f)$. Two common choices are L_1 and L_2 regularization, defined respectively as $||f_1||$ and $||f_2||^2$ and explained in the *Chapter* 4 . Typically learning is the creation and representation to make predictions based on the input features of new samples.

4.1.2 Factors to be Considered Before Modeling

Before modeling the data one should consider other factors [5], like

- *Heterogeneity of the data :* If the input feature vector consists of different kinds of features like discrete, continuous attributes which are supposed to be classification and regression problems, it is easy to apply for some algorithms. Many algorithms, including linear regression, logistic regression, neural networks etc are require that the input features be numerical and scaled to similar ranges.

- *Redundancy in data :* If data is highly correlated the learning algorithms will perform poorly because of numerical instabilities. To know efficiency the learning frameworks, the best way is basically choosing the data which is independent and identically distribute or to randomize the data and to observe the performance or else by imposing some form of regularization.

- *Presence of interactions and non-linearities :* If each of the features provide an independent contribution to the output, then algorithms based on linear functions (e.g. linear regression, logistic regression etc) and distance functions (e.g. Nearest neighbor methods, support vector machine with Gaussian kernels) generally perform well. Linear methods can also be applied, it is must important to specify manually the interactions when using them.

4.1.3 Comparision between Batch and Incremental Learning

Real world problems involve the challenging context of data streams, this means algorithm should be able to learn from a theoretically infinite stream of examples using limited time and memory. These should be done such a way, that are able to predict at any point of time. Two traditional approaches are there to provide the data for learning algorithm, namely: batch and incremental. Data that arrive may be continuous or potentially infinite. To learn the model using inverse dynamics series of experiments are done with continuous data stream or whole data at once. In other words they are called as incremental or batch learning.

Batch learning can be termed as one time go. In this we send all the data (i.e. including testing data) at once to train the model,and then this model is used for testing. In batch learning the algorithm keeps the systems covariance matrix constant while the prediction step is performed on each sample of new input. Batch learning can also termed as off-line learning.

There are different approaches for incremental/on-line learning in the literature,They are: Batch-incremental or instance incremental. Traditional batch learning method done on batches of the data: every new samples form a batch and when that batch is complete; it is given to a learner for training. There are some drawbacks in using the batch-incremental, require to specify the batch size, are forced to delete trained models to make for new ones and it cannot learn from the most recent sample until a new batch is full. Instance-Incremental learning are truly incremental in the sense that they learn from each training samples as it arrives. Having instamce incremental nature it is often to choose over batch-incremental. In incremental learning constantly updating its weights, its error calculation uses different weights for each input sample. With this continous update in covariance matrix this makes system unstable, this leads to more error in predicitons. There are some approaches to overcome this will be discussed in *Chapter* 4.2.

4.1.4 Learning Models

Regression is a fundamental technique for learning associations between motor actions and their corresponding sensors responses. These associations are at the center of action and perception of the robot and therefore a core component for its control and behavior. In the literature there are two types of models that are commonly learned : Namely forward models and inverse models. Forward models predict the perceptual changes associated with motor actions. For example, forward kinematic models compute the position of manipulator for a given joint configuration. In contrast, inverse models attempt to predict the actions that results in desired perceptual changes. For example, inverse kinematics model thus compute the joint configuration that is required to attain a desired pose. In addition to

forward and inverse models, there are also methods which is combination of both. The basic idea behind this model is combination of both forward and inverse models.The information encoded in the forward model can help to resolve the non-uniqueness, i.e. the ill-posedness, of the inverse model. For example, Given a joint configuration q, the task space position x can be determined exactly (i.e. forward kinematic model is well defined), but there may be many possible joint configurations q for a give task space position x (i.e. inverse model could have infinitely many solutions and their combination is not straightforward).

Inverse models, such as inverse dynamics models are frequently used in robotics by Spong et al., 2006 [5]. There are two primary reasons for prominently featuring this particular type of internal model. Firstly, the availability of inverse dynamics models is important for compliant control and contact detection, which are crucial techniques for safe robot operation in human environment. Secondly, dynamic modeling is a well posed supervised learning problem opposed to kinematics, because of the advantage in measuring the desired output is direct in using force/torque sensor. It is explained in the *chapter* 1, i.e. in problem definition placing of force-torque sensor mounted on the arm of the iCub humanoid robot. The Figure 4.4 depicted demonstrates a proximal force torque sensor mounted in the left arm of the iCub humanoid robot. The relationship between manipulator configurations (i.e. joint positions, velocities and

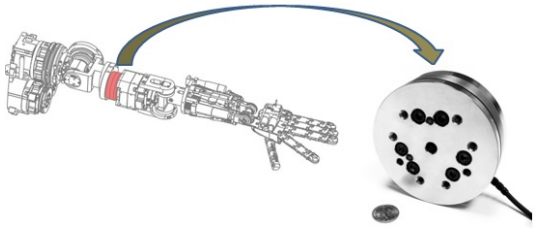

Figure 4.4: Position of proximal force/torque sensor (indicated in red and original sensor) in the left arm of the iCub humanoid

accelerations) and force/torque can be described analytically using the following rigid body dynamics formula below,

$$\tau = D(q)(\ddot{q}) + C(q, \dot{q})(\ddot{q}) + g(q) \tag{4.4}$$

Where D,C and g are the inertial, Coriolis and gravity terms respectively [6]. To determine various kinematic and dynamic parameters accurately, above equation is limited to apply on real life robots. Apart from this many humanoid robots (iCub) have significant additional nonlinear dynamics beyond the rigid body dynamic model, such as actuator dynamics (e.g. due to gearboxes), routing of cables and protective covers. In this we have to predict the force and torques using this dynamic information which are in hand i.e. joint positions, velocities and acceleration.

The big picture Figure 4.5 shows the structure of non-parametric modeling explained to learn the model in batch and incremental for both linear and non-linear models explained in *Chapter* 4.1.

4.1.5 Linear Modeling

Linear regression is a statistical procedure when the relationship between the variables can be described as linear. This means predicting the value of a dependent variable from an independent variable. Linear Models are a family of model-based learning approaches that assume the output y can be expressed as a linear algebric combination of the input attributes $x_1, x_2, ...x_n$. The input attributes $x_1, x_2, ..., x_n$ are expected to be numeric and the output is

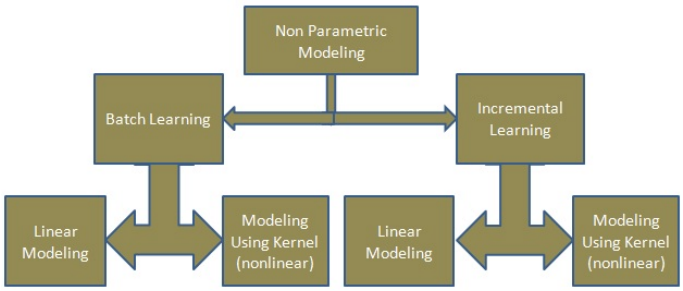

Figure 4.5: big picture for non-parametric modeling with batch and incremental learning

expected to be numeric as well. There different set of methods like ordinary least squares, ridge regression that which are intended for regression in which the target value is expected to be a linear combination of the input variables. In mathematical notation, if y is the predicted value.

$$y = w_1 x_1 + w_2 x_2 + ... \tag{4.5}$$

The whole objective of the training phase is to learn the weights $w_1, w_2, ...$ by minimizing the empirical loss function. Gradient descent is the classical technique for solving this problem with the general idea of adjusting $w_1, w_2, ...$ along the direction of the maximum gradient of the loss function. To avoid overfitting, regularization (L_1 and L_2 norms) is used to penalize large values of $w_1, w_2, .., w_n$. L_1 adds up the absolute value into the loss function, while L_2 performs the squares of the components of w into the loss function. L_2 norm promotes smoothness by penalizing wild oscillations. The differences in their properties are L_1 regularization is computationally inefficient on non-sparse cases whereas L_2 regularization is computationally efficient due to having analytical solutions which is not simple. In L_1 regularization sparse outputs and built-in feature selection is possible, $L1$ is used to enforce sparseness of the solution but in L_2 regularization non sparse outputs and no feature selection is possible.

The strength of linear modeling is that it has very high performance in both predicting and learning. The stochastic gradient descent based learning algorithm is highly scalable and can be handled incrementally. The weakness of linear modeling is assumption of linearity in output with respect to the intput features, which is often false. Therefore, an important feature engineering effort is required to transform each input feature , which usually requires a domain expert. To deal with Linear models via GURLS package. GURLS stands for Grand Unified Regularized Least Squares. GURLS is a least squares based library for state of the art supervised learning algorithms. In non-parametric modeling using GURLS there are some tasks that has to explained before actual implementation of pipeline. To accomplish the task which is different from linear modeling implementation. Depending on the user interest, the task sequence can be given in the pipeline. In this section explaining the common tasks used in parametric and non-parametric modeling that can be implemented on the GURLS and followed by the Batch learning for linear models and incremental learning for linear models. Details of GURLS usage can be found in *Appendix*.

Common tasks used for implementation of linear modeling

- **Splitting Dataset:**

  This task is explicitly aimed at splitting the data into training and validation sets. In the training phase we present data along with the outputs to train the model. Validation set is used in order to estimate how accurately

our model has been trained in general. In fact it actually depends on the size of the data, the value we are interested in predicting, inputs and, also to estimate the model. Validation comes before testing, this can also been seen as pretesting phase. It would be interesting to perform the same test with N runs and averaging the error.

It will be interesting to know how we can split the dataset for training and validation sets. In general it will be useful splitting entire data set into three part one part i.e. 50% of whole data set for training the model, second part i.e. 25% of data set for validating the model and other 25% for testing the model. splitho in GURLS is used for this purpose. By default it uses value 1 for training and validation sets. It is easy to customize the choice for example if splitho portion is specified as 0.6. It uses 60% for training the model and 40% for validation.

- **Parameter selection:**

  Regularization refers to a process to solve an ill-posed problem or to prevent overfitting. This plays a key role in the learning pipeline i.e., regularization parameter is chosen by specifying the approach we are interested in.

  [7] Cross validation is a model evaluation method that is better than residuals. The problem with residual evaluations is that they do not give indication of how well the learner will do when it is asked to make new predictions for data it has not already seen. One way to overcome this problem is to not use the entire data set when training a learner. Some of the data is removed before training begins. Then, when training is done, that data was removed can be used to test the performance of the learned model on new data. This is the basic idea for a whole class of model evaluation methods called cross validation.

  K-fold cross validation is one way to improve over the holdout method. The idea is to divide the dataset into k subsets, and it is repeated k times on holdout method. Each time, one of the k subsets is used as the test set to compute the average error on all subsets. The other $k-1$ subsets are put together to form a training set. Then average error across all k is computed. The advantage of this method is that it matters less how the data is divided. Every data point in a test set exactly once, and in a training set $k-1$ times. The variance of the resulting estimate is reduced as k is increased. The only drawback of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation for training only. A variant of this method is to randomly divide the data into test and training set k different times. The advantage of doing this cross folds is we can get the overall empirical error with differnt sets. The regularization parameter associated to lowest empirical error is chosen for learning.

  In the initial experiments considered in the present work, lambda is set to a default value (i.e. 1). For computing lambda, we first specify a hyperparameter selection approach (e. g. LOO or HO explained in this section *b* & *c*). Then, some initial guesses for lambda are made and the chosen selection algorithm is run for each guess to compute the empirical error associated to it. Finally, the lambda with the lowest empirical error is selected. Later, we repeat the process for the given number k of cross folds (i.e. used 5 for experiments *chapter* 5) and we select the overall best lambda associated with the lowest empirical error . It is easy to customize the number of cross folds in GURLS.

  In order to get the regularization parameter using GURLS there are three approaches, namely:

  a. *fixlambda* in which the regularization parameter is set to default i.e. one (1),

  b. *loocvprimal*, which performs parameter selection when the primal formulation of RLS is used by leave one out

approach. The function approximator is trained on all the data except for one point and a prediction is made for that point. As before the average error is computed and used to evaluate the model. The evaluation given by leave-one-out cross validation error is good, but it is very expensive to compute. Fortunately, locally weighted learners can make leave-one-out predictions just as easily as they make regular predictions. That means the computing Leave-one-out cross validation takes no more time than computing the residual error and it is a much better way to evaluate models.

c. *hoprimal*, which performs parameter selection when the primal formulation of RLS is used by hold out approach. This is the simplest kind of cross validation. The dataset is separated into two sets, called training set and test set. The function approximator fits a function using the training set only. Then the function approximator is asked to predict the output values for the data in testing set (it has never seen these patterns before). The errors it makes are accumulated to give the mean absolute test set error, which is used to evaluate the model. The advantage of this method is that it is usually preferable to the residual method and takes no longer to compute. However, its evaluation can have high variance. The evaluation may depend heavily on which data points end up in the training set and which end up in the test set, and thus the evaluation may be significantly different depending on how the division is made.

- **Optimizer**

The objective of optimizer is to minimize the hyper parameters, so the hypothesis/function is close to outputs for our training samples. For most of the regression problems squared error is the reasonable choice. It Interesting to use the L_2 norm, because of properties it has for both the loss function and the regularization terms, such that

$$G(f) = ||f||^2 \tag{4.6}$$

and

$$L(y, f(x)) = (y - f(x))^2 \tag{4.7}$$

Let us consider this configuration to explain this, the constraining F to the set of linear functions of the form

$$f(x) = \langle w, x \rangle \tag{4.8}$$

where w is a weight vector. Inserting the squared loss and L_2 regularization in equation 4.3 results in a convex optimization i.e. inverted bell shape doesn't have any local optima. Thus, guarantees unique solution which is global optima for problem of w, for which the objective function is given by

$$J(w, \lambda) = \frac{\lambda}{2}||w||^2 + \frac{1}{2}\sum_{i=1}^{m}(y_i f(x_i))^2 = \frac{\lambda}{2}||w||^2 + \frac{1}{2}||y - Xw||^2 \tag{4.9}$$

The equation 4.9 uses matrix notation by defining an m x n matrix input samples $X = [x_1, x_2, ..., x_m]^T$ and an m-dimensional vector of outputs $y = [y_1, y_2, ..., y_m]^T$. The additional factor is solely for mathematical convenience and does not affect the optimal solution. Setting partial derivative of J with respect to w to zero, such that

$$\partial J/\partial w = w(\lambda I + X^T X)X^T y$$

$= 0$, we obtain the optimal solution given by

$$argmin_w J(w, \lambda) = (\lambda I + X^T X)^{-1} X^T y \tag{4.10}$$

Note that this system of linear equations is well posed if $\lambda > 0$, this guaranteeing a unique optimal solution. This technique, known as Tikhonov regularization, was originally proposed to improve condition of ill-posed linear systems of equations by Tikhonov and Arsenin, 1977 [8]. Since then, the regularized optimization problem in eq 4.10 has been introduced in statistical community as ridge regression by Hoerl and Jennard, 1970 [9]. An equivalent formulation with respect to artificial neural networks (ANNs) is known as weight decay by Hinton, 1989 [10], while it has also been studied extensively within the framework of regularization networks (Rns) by Poggio and Girosi, 1990[11]; Girosi et al., 1993[12]. In the remainder, this algorithm will be refereed to as linear RLS following approach of Rifkin et al., 2003 [13] , to emphasize (1) the link to both regularization and the least squares method and (2) that it has used successfully for both regression and classification problems.

Obtaining the solution for equation 4.10 requires solving an n-dimensional system of linear equations. Explicit inversion of covariance matrix $(\lambda I + X^T X)$ is usually not necessary. The overall time complexity for training is cubic in the number of input features and linear in the number of samples. Once an optimal weight vector w has been found, the function in equation 4.9 can be used to predict the output for a given input vector x. It is clear from equation 4.9 that the time complexity per prediction is linear and independent of the number of training samples m.

The output variable y has thus far been assumed scalar for simplicity; however, it is relatively easy to consider the case of a p -dimensional output vector y (Bishop, 2006 [14]). Substituting the vector variant in 4.9 and 4.10, we observe that the weight vector becomes an $n \times p$ weight matrix W, where each column contains the weight vector corresponding to one of the p regression problems. The main advantage of this formulation is of computational nature: solving equation 4.10 is dominated by inversion of the covariance matrix. Since this matrix does not depend on the output variables, the inversion step can be easily used to solve multiple regression problems simultaneously at negligible additional cost. Obviously, a requirement is that all problems share identical input variables X and regularization parameter λ.

- **Prediction**

  Once training of data is finished, prediction of output is performed in this task. This computes the predictions of the linear estimator stored (matrix of coefficient vectors of rls estimator) computed with primal formulation of RLS on the samples passed in the input data matrix i.e. test dataset.

- **Performance Measure**

  After predicting the outputs, now we can compare the predicted outputs with ground truth obtained from force/torque sensor.We can measure the performance of learning algorithm through rmse measurement 4.11 between predicted outputs and ground truth obtained from the actual force-torque sensor measurements.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(diff_t)^2} \tag{4.11}$$

where $diff_t$ is the difference between original vales and predicted value & n is number samples used for testing set.

4.1.6 Batch Learning Implementation Using GURLS For Linear-Models With Results

4.1.6.1 Linear Model with different approaches

In batch learning as discussed above we will feed all the data at once to learning algorithm. After training was done prediction are made for test dataset. It is easy to implement in GURLS. The GURLS pipeline Figure 4.6 depicts necessary tasks for linear modeling. In GURLS as our first job is to train the model which includes splitting the dataset

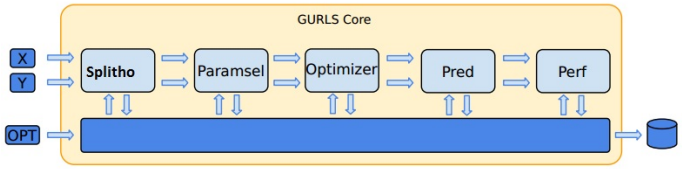

Figure 4.6: GURLS pipeline for implementation of linear modeling

into training and validation set, parameter selection depending on the approach. In next phase we will test data where output is predicted and performance measure of the learning algorithm is obtained. Each task was discussed in the earlier section. As we are using batch learning using the optimizer in this case covariance matrix is constant because training is done only once for all the samples.

For doing experiments on the linear modeling 80% of the dataset is used for training set and 20% of the data is chosen as testing set. In experiments, splitho is set to 0.4 this means 60% of training dataset used for training and 40% is used for validation set. The results in Figure 4.7 obtained for batch learning for linear models with different parameter selection.
For our convenience in the whole thesis three bar graphs are shown in one. First bar graph is rmse error of forces ($units : Newton$) between predicted and actual measurement from the F/T sensor, follwed by rmse error for torques ($units : Newton - meter$) and computational time ($in\ Seconds$) for both training and testing on vertical axis. On horizontal axis number of samples including testing and approach used for parameter selection.

Results in Figure 4.7 are plotted for rmse error over experiments (i.e., with different tasks used in GURLS). From the above results we can see that rmse error is same for force (i.e. blue, green and red), torques (cyan, magenta and yellow) in all experiments this is because of regularization parameter is different in all the cases, but covariance matrix is stable and well posed from Tikhonov et al[8] . When we try to implement these experiments in real time it is important to consider the computational time as well. The third plot is computational time over different experiments. In practical experiments it is one of the important factor to be considered, most of the cases training time (brown) as testing time (gray) is same in all experiments except using Leave-one-out cross validation the reason was discussed earlier. It is evident from Figure 4.7 experiment using holdout approach performs well because rmse error is same for three experiments but less in computational time when compared to other two experiments and which gives optimal value for regularization parameter.

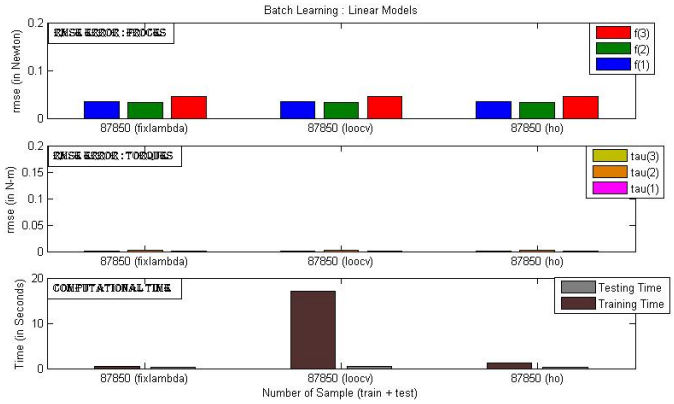

Figure 4.7: Results of Linear models with different approaches in Batch Learning

4.1.6.2 Experiments in Linear Modeling

Performance of the learning algorithm depends on many factors, one of them is number of samples we used for training. As samples increase rmse error will decrease (i.e. inversely proportional to rmse error), but computational time increases when samples increase. To justify this, experiment is performed on the linear models for batch learning with different number of samples for training. From previous experiments result we observe holdout approach for selecting λ performs well when compared to others taking computational time into account. So, experiment 3 parameters are used on different number of samples for training to get rmse error as low as possible. Figure 4.8 show this justification for this experiment.

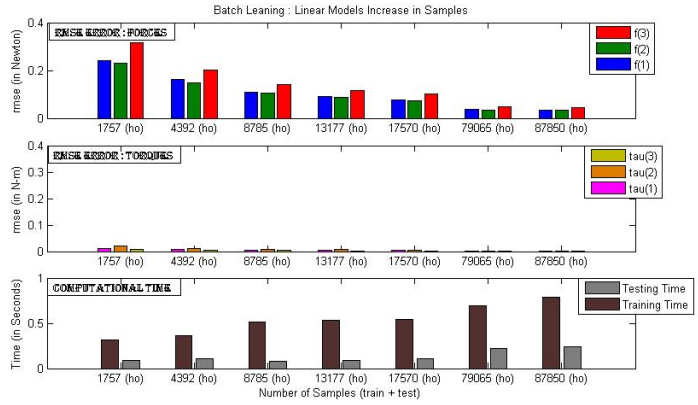

Figure 4.8: Batch Learning for linear models increase in samples error get reduced

From Figure 4.8 it is clear that rmse error is reduced as number of samples are increase but we are penalized by time. For exp no 1, 2% of whole dataset was chosen in this 80% was for training and 20% for testing which is same for all experiment. Increase in the size of the dataset 5%, 10%,..,100% (i.e. whole dataset). One of the drawback in learning algorithms where we cannot deal with huge amount of dataset, even though we have huge data it is hard to learn all the data because it take lot of time for training.

4.1.7 Incremenatal Learning For Linear Models

In *section* 4.1.3 we stated the formal definition of incremental learning. The idea behind this for implementation is, Initially, by taking reasonable amount of samples (above 1000) for training and parameter selection , then we will update model by training each sample. In incremental learning we can do retraining after updating the model, for the experiments it is not considered because of it is computationally cost. In retraining once model was updated we will do training again on all samples. In Incremental leaning covariance matrix will be updated when training model is updated with new sample. This makes covariance matrix numerically unstable in most of the cases. Figure 4.9 demonstrates the general idea of incremental with retraining.

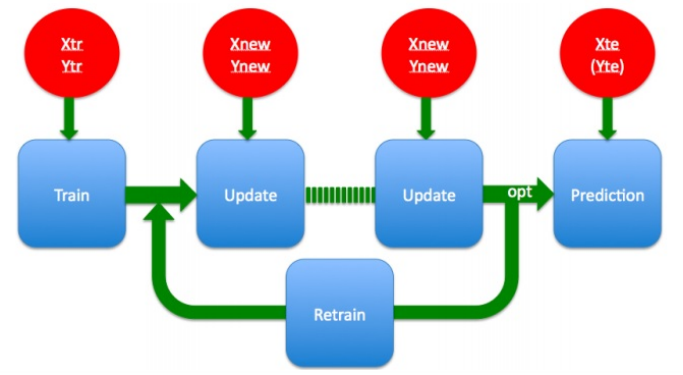

Figure 4.9: General idea of incremental learning with retraining

Our motivation is to work in reality on humanoids, for this reason incremental learning is preferred over batch learning because we can model the data when we are getting the continuous stream of data from the humanoid iCub. Wrappers in GURLS make it easy for implemention of this task. GURLS pipeline for implementation of incremental learning for linear models is shown in Figure 4.10

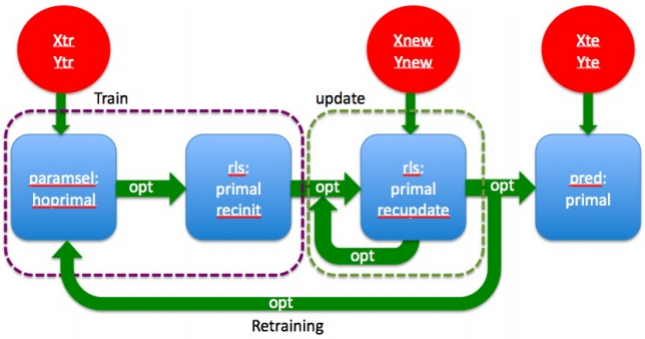

Figure 4.10: GURLS pipeline for implementation of incremental learning for linear models

From the above Figure 4.10 it is clear that initially some samples are needed for parameter selection and training. Later when new samples are added the model get updated. In the experiments for initial training (batch) 2500 samples are chosen for parameter selection and training. It is always better to take reasonable amount of samples at least 1000 as initial batch otherwise there regularization parameter might not be accurate. Once training was done with all the

new samples then we can perform prediction step. Due to computational cost retraining is avoided in the experiments. Figure 4.11 shows the experimental results for Incremental learning for Linear models. From Figure 4.11 we can see

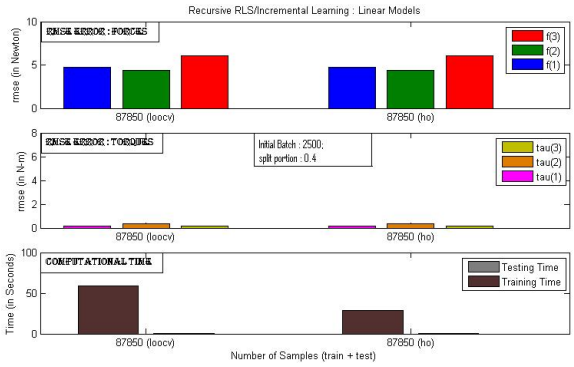

Figure 4.11: Results of incremental learning for linear models

rmse error is same for both the experiments, but exp 1 is penalized more with time when compared to the other because we are using leave-one-out approach for parameter selection. Error is increased by 20 times When compared to Batch learning, because incremental learning is numerically unstable when compared to the batch learning. This instability is due to updating the model when new sample is used for training and using sherman morrison update.

4.1.8 Non-Linear Models

In non-linear modeling, modeling of observed data by a function which is non-linear combination of the modeled parameters that depends on independent variables. In real life problems linear functions are too restrictive, which are often found to be non-linear. To deal with non-linear models, best way is to apply linear algorithms on non linear problems to map the input features into feature space. With this approach input-output relation will turn to be in linear form in feature space i.e. Hilbert space. To fulfill this we use kernel in this case.

Linear inner product spaces with a few additional mathematical details are called as Hilbert spaces. It is a very popular in applied mathematics, due to structure and they are easy to use. Some important definitions are necessary to know

Definition 1 : Inner product on a real vector space '$\mathcal{H}$' is a mapping $< *, * > : \mathcal{H} \times \mathcal{H} \rightarrow R$ satisfying for all vectors $x, y, z \in \mathcal{H}$ and all scalars $u, v \in R$

a. $< ux + vy, z >= u < x, z > +v < y, z >$

b. $< x, uy + vz >= u < x, z > +v < y, z >$

c. $< x, y >=< y, x >$

d. $< x, x >= 0$ with equality only if x = 0.

Definition 2 : A vector space $\mathcal{H}$ endowed with an inner product $\langle ., . \rangle$ and corresponding norm $\|x\| = \langle x, x \rangle^{1/2}$ such that$\mathcal{H}$ is complete with respect to this norm is called a Hilbert space. A Hilbert space might be infinite dimensional, but just like in $\mathbb{R}^n$ .we can say about independent sets, orthogonality and bases. In this we can also project, decompose, linearly transform and anything else as in euclidean space. Most practical computations in Hilbert space get down to ordinary linear algebra.

Let us consider map function ϕ: $\chi \rightarrow \mathcal{H}$ where $\mathcal{H}$ is Hilbert space which is feature space.
On substituting ϕ in4.8, we get

$$f(x) = \langle w, \phi(x) \rangle \tag{4.12}$$

optimal solution for 4.10 turns to

$$argmin(w) J(w, \lambda) = (\lambda I + \Phi^T \Phi)^{-1} \Phi^T y \tag{4.13}$$

where $\Phi = \phi(x)$. This means the algorithm operates explicitly in the feature space and consequently $w \, \epsilon \, \mathcal{H}$. There are different types of non-linear feature mapping also known as basis functions, which are used in practice like Radial Base Functions, Polynomial basis functions[15].

The time complexity of eq 4.13 is cubic in the number of features and may therefore be computationally not feasible if huge number of basis functions is made as a choice. To overcome this issue we can use kernel trick. In addition to this it allows linear methods to be applied in infinite dimensional feature spaces. This kernel trick was originally introduced in the pattern recognition community by Aizerman et al, in 1964 [16] and it is widely used in machine learning in SVM's. The kernel trick is a mathematical tool which can be applied to any kind of algorithm which depends on dot product between two vectors. Kernel functions are replaced wherever dot product is used. Non-linear algorithms are equivalent to their linear originals operating in the range space of a feature space ϕ i.e. Hilbert space explained earlier in this section. However, because kernels are used, the ϕ function does not need to be ever explicitly computed. This is tightly desirable, as we noted previously, because this higher dimensional feature space could even be infinite dimensional and thus infeasible to compute under certain conditions. In addition to this there are is no restriction on input vector.

Tasks used in Non-Linear Models

- **Splitting Dataset :**

  Splitting of dataset was explained in the linear modeling *Section* 4.1.5 , However it same for both Linear and Non-Linear models. Splitting dataset is to divide dataset into training dataset used to train the model and validation set to know performance of our learning algorithm before testing.

- **Parameter Selection :**

  In non-linear modeling we are using kernels, because provide bridge from linearity to non-linearity for algorithms which can be expressed in terms of dot products. We observed in linear model we only need regularization parameter λ becasue we are using primal formulation there. While coming to the non-linear model we need other adjustable kernel parameter σ, this play a key role in the performance of the kernel.

  In this we have three types of parameter selection approaches

  a. *Fixsiglam :* With this choice we are setting regularization parameter and kernel adjustable parameter sigma to 1.

  b. *Siglam :* This perform parameter selection i.e. both lambda and sigma when dual formulation of RLS is used by leave-one-out approach.

c. *Siglamho :* For parameters lambda and sigma is performed when dual formulation of RLS is used by holdout approach.

Leave-one-out cross validation or holdout approach was explained clearly in linear modeling *section* 4.1.5.

- **Kernel Selection :**

I.Representer theorem in General and for RLS :

First we should consider the dual formulation of RLS algorithm, before this we need to know about representer theorem. Rewriting equation 4.9,

$$w_T = \sum_{i=1}^{m} (X_i)^T c_i \tag{4.14}$$

The idea of the representer theorem is depicted in Figure 4.12 where $x_1, x2, ...x_n$ are the inputs in the training

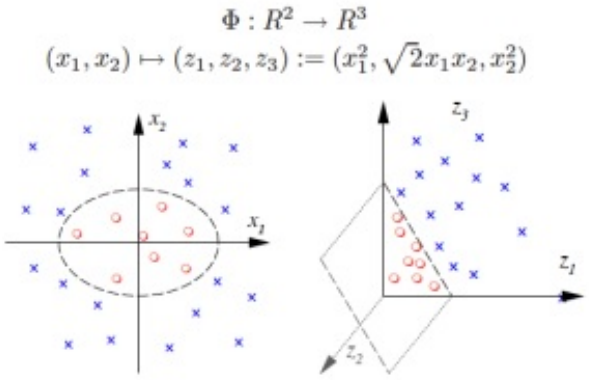

Figure 4.12: Representer theorem overview

set and $c = (c_1, c_2, ...c_n)$ a set of coefficients. The above result is an instance from the representer theorem. The optimal solution for equation 4.13, we can rewrite w as

$$w = \tfrac{1}{\lambda}\Phi^T(y - \Phi w) = \Phi^T\alpha = \sum_{i=1}^{m} \alpha_i \phi X_i \tag{4.15}$$

showing that w can be written as linear combination of the training samples. Following that m-dimensional coefficient vector α is given by

$$\alpha = \tfrac{1}{\lambda}(y - \Phi w) = (K + \lambda I)^{-1} y \tag{4.16}$$

where $K = \Phi\Phi^T$.

To obtain optimal solutions for this dual representation we need m dimensional system of linear equations, as opposed to an n dimensional system for primal formulation. The above alternative formulation has one of the most attractive features is computationally advantageous in training when $m < n$. Apart from this, training

occurs within the inner product. The matrix K is called the kernel matrix and is symmetric and positive semi-definite. It is interesting to know which kernel functions
$K : \chi \times \chi \to \mathcal{R}$ to an inner product in a feature space $\mathcal{H}$. The answer is mercer's condition, because it avoids explicit expressions for non-linear mapping.

One of the advantage in using representer theorem is solution for the given problem depends on the inputs only through inner product. The function K is often called as kernel and to be admissible it should behave like inner product. To be precise, it should be symmetric, positive definite i.e. kernel matrix K should be positive semi-definite for any set of input points. This means kernel matrix doesn't contain non-negative eigen values if they are positive semi-definite. In addition to this, main use of positive semi definite kernel ensures that the optimization problem will be convex and solution is unique. It is easy to check symmetry but it is somehow tricky. With the given admissible kernel, it is possible to construct a corresponding Hilbert space $\mathcal{H}$, which is known as Reproducing Kernel Hilbert Space (RKHS). The representer theorem, originally by Wahba et al,1990 [17] and generalized by Scholkopf et al, (2001) [18], guarantees that optimal solution in the possible infinite dimensional RKHS can be expressed as a finite dimensional kernel expansion.

II.Selection of Kernel :

It is interesting to know how to choose kernel among many types. A key property is that linear algorithms are performs in implicit feature space. The assumption for this kind of problem is actually linear in feature space induced by kernel. With this assumption, it is necessary to use appropriate kernel for the learning problem under consideration. In fact, it is difficult to say which kernel is best in performance, there are particular kernels that have been shown to perform well on a wide variety of practical learning problems. Most widely used families of kernels are polynomial and RBF kernel. In experiments linear kernel, RBF kernel and approximating Gaussian kernel with random features are used. In the later section these kernels are explained in detail.

a. Linear Kernel :

The linear kernel is the simplest kernel function among all. It is given inner product with an optional constant.

$$K(x, y) = x^T y + c \tag{4.17}$$

Kernel algorithms using a linear kernel are often equivalent to their non-linear counter part, in other words linear kernel is equivalent to linear modeling. For example KPCA with linear kernel is the same as standard PCA.

b. RBF Kernel :

Short form of Radial Basis Function kernel is RBF kernel. Gaussian kernel is an example for RBF kernel.

$$K(x, y) = exp^{(-\frac{\|x-y\|^2}{2\sigma^2})} \tag{4.18}$$

Above equation 4.18 can also be implemented using
$K(x, y) = exp^{(-\gamma\|x-y\|^2)}$.

The adjustable parameter sigma plays a major role in performance of kernel and one should be careful while tuning this parameter. If it is overestimated, the exponential will behave almost linearly and higher dimensional

projections will start loosing their non-linear power. In case, if it is underestimated the function will lack regularization and decision boundary will be highly sensitive to noise in training data.

c. Gaussian kernel approximation with random features :

Even though polynomial and RBF kernel are widely used there in machine learning but they are restricted to use in reality because high computational time & unable to deal with large number of input features. To justify these, polynomial kernels are computationally infeasible when the degree of the polynomial or the original input dimensionality n is moderately large. Explicit computation of the feature mapping for the RBF kernel is intractable, since the corresponding feature space is infinite dimensional. Random features are a trick to speed up supervised learning algorithms, so they are able to handle large datasets. In the literature, Rahimi and Techt (2008a) [19] demonstrated that the RBF kernel and other shift invariant kernels can be approximated to an arbitrary precision using finite dimensional random features mapping. Their approach utilizes Bochner's theorem, which relates positive definite functions, among which admissible kernel functions, to Fourier transforms using finite Borel measure.

A function K on $\mathcal{R}^d$ is the Fourier transform of a finite positive Borel measure μ if and only if it is positive definite and continous. (Bochner, 1933 [20]). In brief, this theorem states that shift-invariant kernel function $K(x_i, x_j) = K(x_i - x_j)$ can be described as the Fourier transform of a unique measure μ. Here μ is probability density function. Then the kernel can be estimated randomly sampling features according to μ. Combining both these techniques, finally we get

$k(x_i - x_j) = \int_{\mathbb{R}^d} e^{-i\omega^T(x_i - x_j)\mu(\omega)\mathrm{d}\omega}$

$= \mathbb{E}_\omega\ [e^{-i\omega^T}(x_i - x_j)\ ](if \int \mu(\omega)\mathrm{d}\omega = 1)$

$= \mathbb{E}_\omega\ [\cos(\omega^T(x_i - x_j)]$

$= \mathbb{E}_\omega\ [\cos(\omega^T x_i)\cos(\omega^T x_j) + \sin(\omega^T x_i) sin(\omega^T x_j)]$

$= \mathbb{E}_\omega\ [z_\omega(x_i)^T z_\omega(x_j)]$ where

$$z_\omega(x) = [\cos(\omega^T x), \sin(\omega^T x)]^T \tag{4.19}$$

Inner product $\langle z_\omega(x_i)^T z_\omega(x_j)\rangle$ gives an unbiased estimate of any shift-invariant kernel $k(x_i, x_j)$, spectral frequency ω is drawn according to its corresponding measure μ. Corresponding probability density function μ can be obtained by computing IFT (Inverse Fourier Transform). Probability density function for the isotropic RBF kernel is Gaussian and it suffice to sample $\omega \sim N(0, 2\gamma I)$. Figure 4.13 is example of 1-dimensional RBF kernel and its approximation using these random features.

Gaussian kernel approximation using random features is computationally efficient when compared to other methods and it can easily handle large dataset and high dimensional dataset.

- **d. Optimizer :**

For Non linear modeling RLS dual formulation is used explained earlier in the kernel section. The optimal solution for equation 4.13, we can rewrite w as

$w = \frac{1}{\lambda}\Phi^T(y - \Phi w) = \Phi^T\alpha = \sum_{i=1}^{m} \alpha\phi X_i$ showing that w can be rewrite as linear combination of the training samples. Following that m-dimensional coefficient vector α is given by

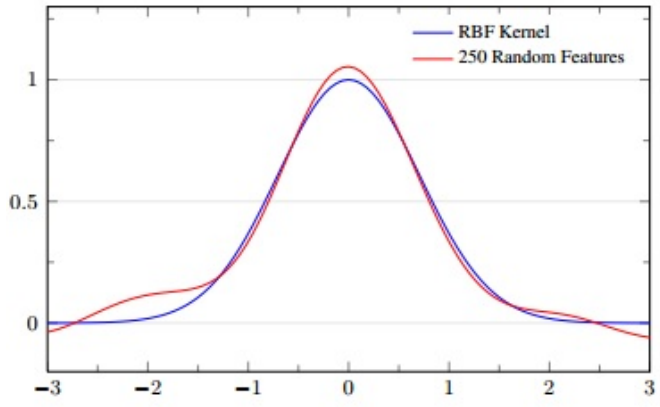

Figure 4.13: RBF Vs approximating Gaussian Kernel with Random Features

$\alpha = \frac{1}{\lambda}(y - \Phi w) = (K + \lambda I)^{-1} y$
where $K = \Phi\Phi^T$. For optimal solution for this dual representation we need m dimensional system of linear equations, as opposed to an n dimensional system for primal formulation. The above alternative formulation has one of the most attractive feature is computationally advantageous for training when $m < n$ and training occurs within inner product. Matrix K is called the kernel matrix and is symmetric positive semi-definite because it satisfies mercer's condition.

Parameters which include regularization parameter λ and the kernel parameter γ or σ , optimization of these parameters is known as model selection/ hyper parameter optimization. Unfortunately, most of learning algorithms are formulated to result in a convex optimization, hyperparameter optimization is typically non-convex and having multiple local minima. Most widely used approach, for optimizing hyperparameters is to empirically evaluate the performance of predefined number of configuration using cross validation, because it give convex optimization with inverted bell shape gurantees in unique optimal value. Alternatively, we can estimate performance using K-cross folds for cross validation, it is presented in *Chaper* 5

- **e. Predicting Kernel :**

  This task compute kernel matrix between training points and testing points. It must use dual formulation.

- **f. Prediction :**

  Matrix $K = \Phi\Phi^T$ obtained above can be described component wise as $K_{jj} = \langle \phi(x_i), \phi(x_j) \rangle$ and the prediction function can be written as

$$f(x) = \langle w, x \rangle = \langle \Phi^T \alpha, \phi(x) \rangle = \sum_{i=1}^{m} \alpha_i \langle \phi(x_i), \phi(x) \rangle \tag{4.20}$$

  From the above equation 4.20 we can observe that inner product $\mathcal{H}$ is required to compute the RLS dual solution. Kernel trick exploits this observation by directly specifying a kernel function. This avoids explicit mapping of the input samples into feature space.

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x) \rangle_{\mathcal{H}} \tag{4.21}$$

In other way computed the predictions of the regressors stored in gram matrix on the samples passed in the X matrix.

- e. Performance Measure:

  Performance measure between predicted output and ground truth is obtained by computing rmse error between them, same as in Linear modeling *Section* 4.1.5.

4.1.9 Batch Learning Implementation Using GURLS For Non-Linear Models With Results

4.1.9.1 Non-Linear Model with Different Approaches

In batch learning using kernel with different pipeline should be used. As discussed earlier in the drawbacks of linear and RBF kernel they are infeasible in large number of features, which need more time and memory. So for initial experiments small set of data set is used which 2% (1757 samples for training and testing) of whole dataset. The pipelines for Linear, RBF kernel and Gaussian approximation are shown in Figure 4.14

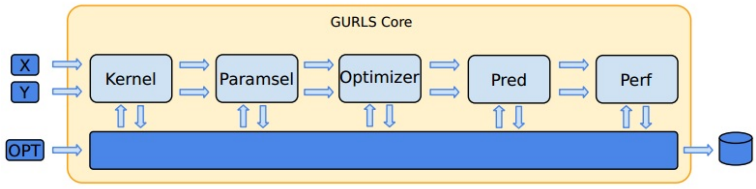

Figure 4.14: GURLS pipeline for implementation of non-linear models with linear kernel

After specifying dataset split portion, kernel task should be done and followed by others as shown in the figure. However, Linear kernel is almost equivalent to linear modeling but for implementation it uses dual formulation instead of primal.

The kernel method i.e. using RBF kernel is different from linear kernel pipeline is depicted in Figure 4.15 In the

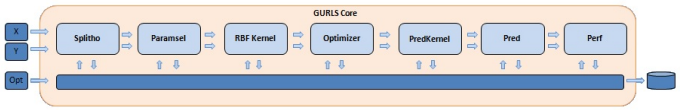

Figure 4.15: GURLS pipeline for implementation of non-linear modeling with RBF and RF kernel

experiment, small dataset is used i.e. 2% of whole dataset. For training, 1405 samples (i.e.80% of dataset chosen) and for testing 352 samples are used. RMSE error with the different kernels, with different parameter selection approach is depicted in the Figure 4.16 For convenience, on horizontal axis which kernel was used (i.e. L is Linear, RBF kernel & RF is approximating Gaussian kernel with random features) is shown. In this small dataset was used in Exp No 1-6 (i.e. first 6 results with 1757 samples for both training and testing) and for Exp No 7 (i.e. last with 70280 samples for training and 17570 samples for testing) whole dataset was used, because in this Gaussian kernel was approximated using random features is used as kernel. Having some advantages over other kernel methods, efficiently

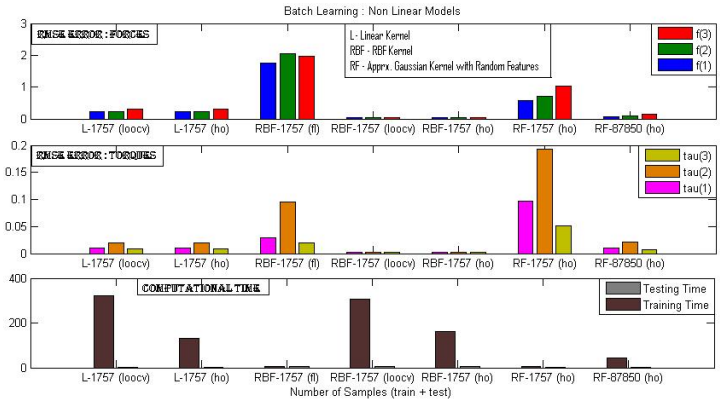

Figure 4.16: Results of batch learning for non-linear models

able to deal with high dimensions & fast in computation. In Exp 1 & 2, Learning was made using Linear kernel with different parameter selection approaches i.e. leave-one-out cross validation and holdout approach are used. For Exp 3 RBF kernel was used by fixing regularization parameter lambda and kernel parameter sigma to 1, as a result with non-optimal values of parameters error is more when compared to other methods. Exp 4 & 5 RBF kernel was used using different approaches i.e. leave-one-out cross validation and holdout approach for selecting parameters. For this Exp 4 & 5 it is clearly evident that computational time is high, when compared with other kernels. Using different approaches for getting optimal parameters results in increase in computational time.

This experiment was done explicitly to show Gaussian kernel approximation using random features are feasible in non linear models over other methods, I used whole dataset which is 70280 (i.e. 80%) for training and 17570 (i.e. 20%) for testing. This gives good observation on the error & computational time. Computational time when dealing with whole dataset is 4 times (i.e. around 40 seconds) more than with linear kernel for 2% of dataset. Testing time is almost same in every kernel method. While on the other hand rmse error is 1.5 times high with small dataset (i.e. 2% of whole dataset) over other RBF kernel method & 1.25 times more than linear methods, but for other kernel methods we used very small dataset. If we use large dataset for Linear and RBF kernels, error will too less, which is negligible almost equivalent to ground truth values. Among Linear and RBF kernel, RBF kernel performs well regarding rmse error. In real time applications it is not possible to do this because our motivation is to design an algorithm to reduce computational time & memory, it would take hours or may be days to predict the values & huge memory to compute them. With this results, there is a strong evidence that approximating Gaussian kernel outperform when compared to other kernel methods when we are dealing high dimensional large dataset.

4.1.9.2 Experiments on Non Linear Model

Other kernel methods are not efficient to deal with high number of samples & increase in features etc, because of their drawbacks like computational time and memory, they are limited in reality.Figure 4.17 shows results when experiment was done on increase in dataset, Increase in number of samples computational time increases and error get reduced. This experiment is conducted for RBF kernel, from above Figure 4.17 it shows among linear and RBF kernel methods, rmse error of RBF kernel is less. Figure 4.17 clearly indicates with increase in dataset size, rmse error gets reduced and computational time increases. For exp no 1, 2% samples of whole dataset followed by exp no 2, with 5% and exp

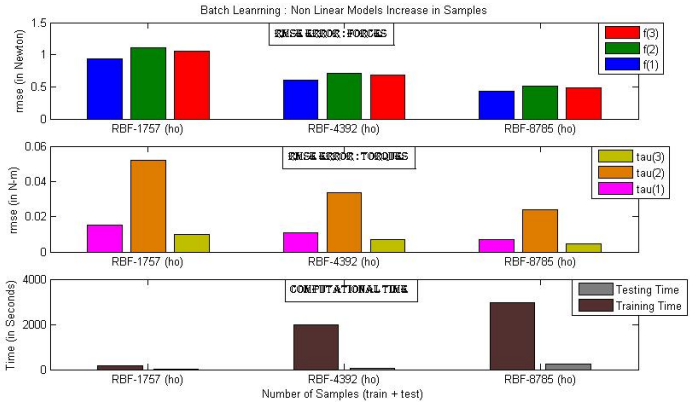

Figure 4.17: Increase in samples error get reduce for non-linear models

no 3, with 10% of dataset.

4.1.10 Incremental Learning For Non-Linear Models

General idea of incremental learning was explained in the *section* 4.13 & *section* 4.17. In Linear models *Figure* 4.09 shows the outline of incremental learning. We can see from the results *Figure* 4.11, Incremental learning for Linear models is numerically unstable, so it leads to more error. So now the idea is to make the system stable by making efficient updates when the new sample arrives. With this RLS as base algorithm was not incidental, since this algorithms allow efficient and exact updates of the solution using linear algebra. There are two approaches used for incremental learning with efficeint updates namely : Sherman-Morrison update and Cholesky update.

4.1.10.1 Sherman-Morrison Update :

In the earlier section optimal solution is

$$w = (\lambda I + \Phi^T \Phi)^{-1} \Phi^T y = A^{-1} b \tag{4.22}$$

Now the solution is decomposed in terms of a $D \times D$ matrix $A = \lambda I + \Phi^T \Phi$ and D dimensional vector $b = \Phi^T y$. When new sample (x_t, y_t) arrives at time step t can described as

$$\Phi_t = \begin{bmatrix} \Phi_{t-1} \\ \phi_t \end{bmatrix} \; and \; y_t = \begin{bmatrix} y_{t-1} \\ y_t \end{bmatrix} \tag{4.23}$$

where $\phi_t = \phi(x_t)$
Using these update rules, matrix A in equation 4.22 can formulated recursively as

$$A_t = \lambda I + \Phi_t^T \Phi_t = \lambda I + \Phi_{t-1}^T \Phi_t + \phi_t \phi_t^T = A_{t-1} + \phi_t \phi_t^T . \tag{4.24}$$

From this we can say that each additional sample has rank-1 update. Similarly, the update of vector b can be described recursively as

$$b_t = \Phi_t^T y_t = \Phi_{t-1}^T y_{t-1} + \phi_t y_t = b_{t-1} + \phi_t y_t. \tag{4.25}$$

From above equations we can draw conclusion recursive update rules are no longer necessary to store previous samples in memory instead of $A(t)$ and $b(t)$ are necessary at each time step. For the 4.25 formulation for inverse of $A(t)$ for every update, it takes more time to compute it is around $O(D^3)$. To overcome this time constraints we can update A^{-1} directly, which is known as recursive least squares in the field of signal processing (Sayed, 2008 [21]). In the literature, it is straight forward method to perform a rank -1 update of the inverse of a matrix is the Sherman Morrison formula (Hager, 1989 [22]), such that

$$A_t^{-1} = (\lambda I + \Phi^T \Phi)^{-1} = (A_{t-1} + \phi_t \phi_t^T)^{-1} = A_{t-1}^{-1} - \frac{A_{t-1}^{-1} \phi \phi^T A_{t-1}^{-1}}{1 + \phi^T A_{t-1}^{-1} \phi} \tag{4.26}$$

Updating the inverse of matrix A reduces the time complexity of adding a sample to an existing model to $O(D^2)$ and only requires the $D \times D$ inverse matrix A_t^{-1} and D-dimensional vector $b(t)$ to be stored in memory. The recursion is completed by setting the initial configuration $A_o^{-1} = \frac{1}{\lambda} I$ and $d_o = 0$. The simple and recursive update procedure is very popular in many applications. The main disadvantage of explicitly updating the inverse matrix are sensitivity to roundoff errors.

4.1.10.2 Cholesky Update :

Sherman-Morrison problem can be alleviated in large extent by updating the Cholesky factor of A instead (Golub and Loan, 1996 [23]), known as the QR algorithm in the field of adaptive filtering (Sayed, 2008 [21]).The upper triangular Cholesky factor R that satisfied for each step t

$$A_t = \mathcal{R}_t^T \mathcal{R}_t \tag{4.27}$$

Note that $\mathcal{R}(t)$ is full rank and guaranteed to be unique, since $A > 0$ for $\lambda > 0$. Following earlier notation, we are interested in the rank update problem

$$\mathcal{R}_t^T \mathcal{R}_t = A_t = A_{t-1} + \phi_t \phi_t^T = \mathcal{R}_{t-1}^T \mathcal{R}_{t-1} + \phi_t \phi_t^T \tag{4.28}$$

Where the initial matrix $\mathcal{R}_o = \sqrt{\lambda} I$. This equates to the problem of finding $\mathcal{R}_t$ given the previous Cholesky factor $\mathcal{R}_{t-1}$ and new training sample ϕ_t. This special form of rank update can be computed by reformulating the problem as

$$\tilde{\mathcal{R}}_{t-1}^T \tilde{\mathcal{R}}_{t-1} \tag{4.29}$$

Where the temporary $(D+1) \times D$ matrix

$$\tilde{\mathcal{R}}_{t-1} = \begin{bmatrix} \mathcal{R}_{t-1} \\ \phi_t^T \end{bmatrix} \tag{4.30}$$

$\mathcal{R}_{t-1}$ is considered as upper triangular, matrix $\tilde{\mathcal{R}}_{t-1}$ can be illustrated

$$\begin{bmatrix} r & r & r & r \\ 0 & r & r & r \\ 0 & 0 & r & r \\ 0 & 0 & 0 & r \\ \text{ø} & \text{ø} & \text{ø} & \text{ø} \end{bmatrix} \tag{4.31}$$

Here dimension $D = 4$ is choosen for demonstration purpose. It follows that the updated Cholesky factor $\mathcal{R}_t$ can be obtained by introducing zeros in the last row of $\tilde{\mathcal{R}}_{t-1}$. Standard technique id to introduce zeros in a matrix is using given rotation (Golub & Loan, 1996 [23]), which perform a rotation in the plane spanned by two coordinate axes. In this case, a total of D rotations needed to zero the elements in the last row, where each rotation related to the plane $(D+1, i)$ for $1iD$. As last row elements contains zeros and removed from $\tilde{\mathcal{R}}_{t-1}$ such that the updated cholesky factor $\mathcal{R}_t$ still remains. The weight vector w_t can then obtained using back and forward substitution. Update of cholesky factor and solving w might be high, the computational time is same to rank updates of inverse matrix. However computational cost is related to incremental update of cholesky factor is prefered to increase stability of system.

For the initial matrix $\mathcal{R}_o$ or A_o regularization parameter is only taken into account and does not enter in subsequent updates. The effect of regularization parameter will decrease as we increase the number of training samples is seen in previous experiments. However, note that the regularization parameter is to prevents singularity of the covariance matrix by increasing its eigenvalues (Lo Gerfp at al., 2008 [24]). This is needed at the early stages of the algorithm when Φ is not full column rank. Subsequent rank updates increase the smallest eigenvalue of $\mathcal{R}$,which is popularly known as interlacing theorem (Horn and Johnson 1986 [25]),therefore reducing regularization in later stages. We see in the previous sections regularization parameter is to prevent from overfitting. An argument from this perspective is that overfitting is less likely as the number of training samples increases. From the results in Figure 4.18 we can see that

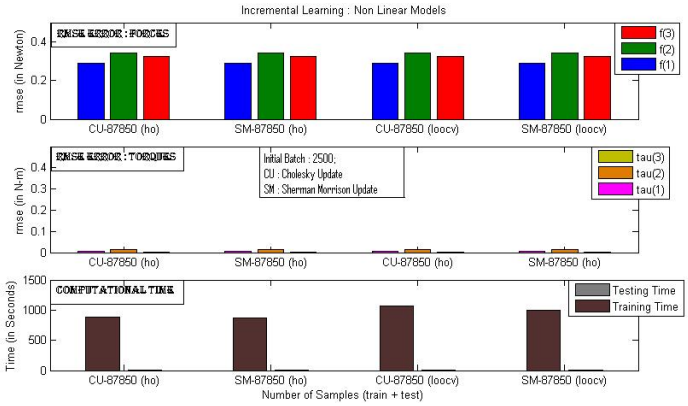

Figure 4.18: Results for incremental learning for non-linear models

rmse error for both the approaches looks same. However, there is some change in precision values in Sherman-Morrison update and cholesky update remains stable.

4.2 Parametric Modeling

Dynamic model of the robot is required to control or simulate its motion. This model is characterized by ten inertial parameters of each link, which are known as standard inertial parameters. These parameters can be estimated using the arms joint torques and forces along with the positions,velocities and accelerations of the joint [26][27]. However iCub humanoid is mounted with joint force-torque sensor. Due to redundancy of this representation, there are infinite sets of standard parameters which satisfy the dynamic model. Due to computational cost of the dynamic model and facilitate identification process, a minimum set of inertial parameters are enough, so called base parameters. These estimated inertial parameters can be used in many applications like force control, object recognition and pose estimation. The intuition behind this experiment is how regularization parameter influence in identifying the inertial parameter which is explained in the later sections.

4.2.1 Overview

- *I Spatial notation overview:*
  A rigid body has 6 DOF, now the idea is to use 6D vectors instead of two 3D vectors. Some roboticists already use this and using this 6D vectors in practice is more common. 6D vectors come in various forms. In this section I present brief overview of this representation, which are called spatial vectors. By using this 6D vectors we can cut the volume of algebra, instead of having two different 3D vectors for example to describe force we need 3D vector in the same way for acceleration as well, with this representation we have two equations of motion for each body. By using 6D vectors, we will pair up corresponding 3D vectors and equations. Which leads to immediate result more compact in notation involving fewer quantities and equations. However, if anyone thinks the idea of 6D vectors are notations for organizing 3D vectors is missing half the point. In fact 6D vectors are tools though but they have their own mathematical properties. This will solve the problems directly and at higher level of abstraction and thinking in 6D, which is easier than it sounds.Using this spatial vectors, formulation of a problem more succinctly, solve it more quickly and in less steps, will be able to present results more easy to others.

  Spatial vectors are six-dimensional vectors which describes the motion of the rigid bodies and the forces acting on them. Spatial vectors are used for various problems in robot kinematics and dynamics. To illustrate the power of spatial vectors, we shall consider the class of robots having branched connectivity. This class includes legged robots, humanoids and multi-fingered grippers and traditional serial robot arms, however it doesn't include robot kinematics loops, such as parallel robots. To cope up with the degree of generality, we shall take model based approach this means the robot mechanism that describe by means of a standard set of quantities stored in model structure, and the equations and algorithms. Spatial vector algebra and for more details one can found in Featherstone[28][29].

  Lets see an example use of spatial vectors.The 6D spatial vectors are part of two different vector spaces, depending on whether they are twists like spatial velocities, accelerations etc or wrenches like spatial forces/torques, momentum etc.
  M^6vector space of motion spatial vectors and F^6 vector space of force spatial vectors. Between these two vectors different vector spaces a scalar product is defined: if $f \forall F^6$ is the force acting on the body and $V \forall M^6$ is the velocity of the body $f.v$ is the power exerted by the wrench on the body. For more details refer to [28].

- *II. Floating base notation with newton euler algorithm & floating base dynamics equation :*
  For using spatial vectors notation described in the earlier section and from ref [3,4] the expression of the dynamics of a floating base rigid body tree, it is important to know the joints and links of a tree. In the Figure **??** rigid body structure as graph having links and joints is shown. In this rigid body tree is examined, so it is not possible

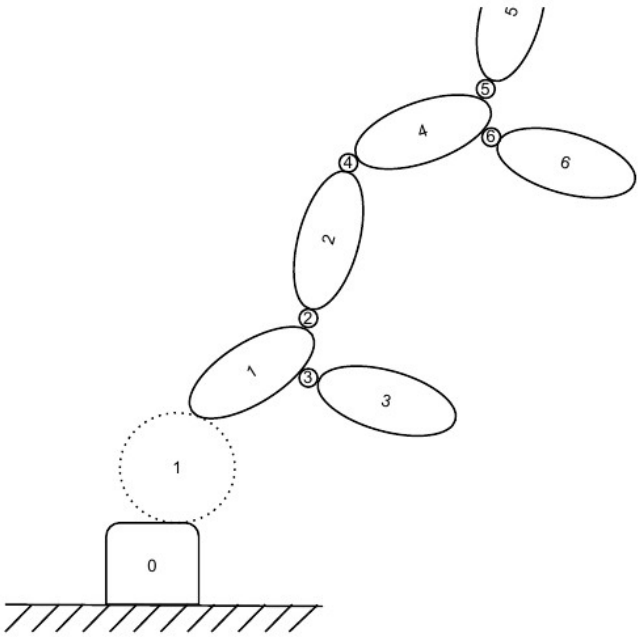

Figure 4.19: Schematic example of floating base rigid body tree

to have loops in the connectivity graph. For the image shown above tree consists of number of bodies denoted as N_b. In this one is chosen as base link, it is represented as 1. This link is root of the tree which doesn't have any parent links and links that are connected to this are considered as children. We can represent the links number in arbitrary way respecting the rule that any successor link is assigned a higher number than its predecessor. To use this structure lets represent some notation to this tree. $P(i)$ is the parent of the link i, $\mu(i)$ are the set of children for link i, $v(i)$ are the set of bodies in the subtree starting from link i, $V_e(i)$ is the set of bodies on the path from the base link to the link i.

All these definitions hold for floating base rigid body tree, but it could be convenient to consider structure as a fictitious fixed base structure to use all the results that hold for fixed base structure. For our example fixed base O is considered as the part of the tree as the parent of the floating base 1. These two are connected by fictitious 6 DOF joint, that for consistency with the rest of definitions, is called 1 as in Figure 4.19 . If the rigid body is completely connected and it is composed of bodies N_b, then joints are $N_b - 1$. They are numbered from 2 to N_b, so that each joint i connected the body i to its parent link. For our convenience superscript indicates frame of reference which is omitted in notation for example ${}^{i}\mathrm{a}^{g}$ i.e. a^g is expressed in frame i. Joint i can be considered as motion constraint between the coordinate frame $P(i)$ and the coordinate frame i. $\mathbf{q_i}$ is the vector which is configuration of each joint i, whose dimensions are equal to degree of freedom of joint. For revolute joint

quantity $\mathbf{q_i}$ is scalar. Transformation matrix ${}^{i}X_{P(i)}(i.e\ T)$ depends on the joint type and joint configuration. Important quantity to defined by the joint model is the joint motion subspace matrix S. For example motion subspace of joint i is defined as

$$S_i\dot{q}_i = V_{p(i)} \tag{4.32}$$

Here V_i is the spatial velocity of the body i, $V_{p(i)}$ spatial velocity of the body $p(i)$. Both V_i and $V_{p(i)}$ consist of angular and linear part.

Recursive Newton Euler Algorithm

The inverse dynamics of general kinematic tree can be calculated in three steps

1. Calculating velocity and acceleration of each body in the tree.
2. Calculating the forces required to produce these accelerations.
3. Calculate forces transmitted across the joints from the forces acting on the bodies.

These are the steps of the recursive Newton-Euler algorithm[29]. The equations of the each step are as follows

– *step 1:* Let V_i and a_i be the velocity and acceleration of body i. The velocity of any body in a kinematic tree can be defined recursively as the sum of the velocity of its parent and velocity across the joint connecting it to its parent. Thus

$$V_i = V_{p(i)} + S_i\dot{q}_i \tag{4.33}$$

V_o are the initial values i.e. 0. The recurrence relation for accelerations is obtained by differentiating this equation, resulting in

$$a_i = a_{p(i)} + S_i\ddot{q}_i + S_i\dot{q}_i \tag{4.34}$$

where a_o initial acceleration which is 0. Iterating this formulae with i ranging from 1 to N_b will give the velocity and acceleration of each body in the tree.

– *step 2:* f_i^b be the net force acting on body i. This force is related to the acceleration of body i by equation of motion

$$f_i^b = I_i a_i + v_i \times^* I_i v_i \tag{4.35}$$

– *step 3:* From *Figure* 4.20 let f_i be the force transmitted from body $P(i)$ to body i across joint i, and let f_i^x be the external force acting on body i. External forces can be model a variety of environment influences like force fields, physical contacts and so on. They are regarded as input to the algorithm; that is their values are assumed to be known. The force on body i is then

$$f_i^B = f_i + f_i^x - \sum_{j\epsilon P(i)} f_j \tag{4.36}$$

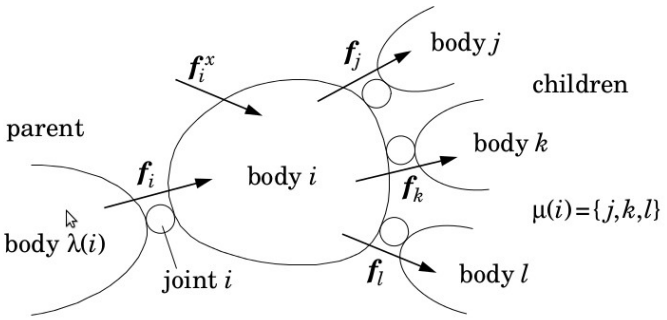

Figure 4.20: Force acting on body i to demonstrate RNEA

This can be rearranged to give a recurrence relation for the forces

$$f_i = f_i^B - f_i^x + \sum_{j \epsilon P(i)} f_j \tag{4.37}$$

Successive iteration of this formula, with i ranging from N_b down to 1, will calculate spatial force across every joint in the tree. No longer use of initial formula as $\mu(i)$ is the empty set for any body with no children. Now we have spatial force across each joint, it remains only to calculate generalized forces at the joints, which are given by

$$\tau_i = S_i^T f_i \tag{4.38}$$

All this equations describe the recursive Newton Euler algorithm in its simplest form. In this gravitational force is not taken into account. If gravitational force is taken into account or acting on the body then it should separate from the total force exerted by the body i.e in equation 4.37 which turns as

$$f_i = f_i^B - f_i^g - f_i^x + \sum_{j \epsilon P(i)} f_j \tag{4.39}$$

we can write $f_i^B - f_i^g$ in one term then

$$f_i^B g = f_i^B - f_i^g = I_i a_i + V_i \times^* I_i V_i - I_i^i a_g = I_i(a_i^{a_g}) + V_i \times^* I_i V_i \tag{4.40}$$

Where $a_i^{a_g} = a_i -^i a_g$ is called as proper acceleration.

- *III FLOATING BASE DYNAMICS*

The dynamic equation of a floating base of rigid body tree with N_b links can be written as [28][30][31]

$$\begin{bmatrix} H_{11} & H_{1*} \\ H_{*1} & H_{**} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_* \end{bmatrix} + \begin{bmatrix} b_1 \\ b_* \end{bmatrix} + \begin{bmatrix} g_1 \\ g_* \end{bmatrix} = \begin{bmatrix} 0 \\ \tau \end{bmatrix} + \sum_{i=1}^{N_b} \begin{bmatrix} {}^1X_i^* \\ J_i^T \end{bmatrix} f_i^x \tag{4.41}$$

In the above equation 4.41 q_1 is joint position for the 6 DOF virtual joint connecting floating base to a fixed reference frame, q_* position of real joints of the rigid body tree, H is the inertial matrix b is the bias force considering Coriolis and centrifugal effects, g is the bias force considering gravitational effects, τ the joint

considering gravitational effects f_i^x the external wrench acting on the link i & J_i is the Jacobian for link i.

The vector τ and f^x represents the torques and forces acting on rigid body tree. These torques differ from those exerted by the joint motors, because friction dissipates some power.

4.2.2 Description of the Regressor's Structure

Vectorization of Inertial Parameters
For each link i of rigid body structure, the inertial parameters are ten which are shown below

m_i mass of link i, C_i position vector of the center of mass of link i, $\bar{I}_{oi}$ 3D rotational inertia matrix of link i, The above quantities are with respect to reference frame. It is common to find 3D rotational inertia of the link expressed on the same axis of the frame of reference i, but with respect to a pole fixed to the center of mass of the link, expressed as $\bar{I}_{Ci}$ in [28], because this choice leads to a slightly less complicated dynamic equation. However, these two representation of inertia matrix related to parallel axis theorem

$$\bar{I}_{Oi} = \bar{I}_{Ci} + m_i C_i \times C_i \times^T \tag{4.42}$$

In order to obtain a set of inertial parameters that effect linearly the dynamic equations, two choices are made to prefer $m_i c_i$ first moment of mass instead of C_i and $\bar{I}_{Oi}$ over $\bar{I}_{Ci}$. The concise notation $\phi_i \epsilon R^{10}$ for the inertial parameter of the link i can be:

$$\phi_i = \begin{bmatrix} m_i \\ m_i c_i \\ vech(\bar{I}_{Oi}) \end{bmatrix} \tag{4.43}$$

The $Vech$ operator [32] turns a symmertrical matrix into a vector:

$$vech\left(\begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}\right) = \begin{bmatrix} I_{xx} \\ I_{xy} \\ I_{xz} \\ I_{yy} \\ I_{yz} \\ I_{zz} \end{bmatrix} \tag{4.44}$$

Considering the spatial inertia [28] I_i it is possible to define the spatial $vech$ operator, indicated with $vech^s$:

$$vech^s(I_i) = \phi_i \tag{4.45}$$

The matrix $\omega\bullet$ is defined such that $\omega \bullet vech(\bar{I}) = \bar{I}\omega$: where

$$\omega\bullet = \begin{bmatrix} \omega_x & \omega_y & \omega_z & 0 & 0 & 0 \\ 0 & \omega_x & 0 & \omega_y & \omega_z & 0 \\ 0 & 0 & \omega_x & 0 & \omega_y & \omega_z \end{bmatrix} \tag{4.46}$$

In the same way $V\bullet^S$ is defined by

$$V \bullet^S vech^s(I) = I * V \tag{4.47}$$

It is important to know about net wrench regressor to describe various structure of regressor. The net wrench acting on a single link which depend only on dynamical parameters of the link :

$${}^i f_i^B = I_i a_i + V_i \times^* I_i V_i =^i Ai\phi_i \tag{4.48}$$

We described all notations earlier

$${}^i A_i = a_i \bullet^s + V_i \times^* V_i \bullet^s = \begin{bmatrix} 0 & -(\bar{a}_i + \omega_i \times \bar{V}_i)\times & \dot{\omega}_i \bullet + \omega_i \times \omega_i \bullet \\ (\bar{a}_i + \omega_i \times \bar{V}_i) & \dot{\omega}_i \times + (\omega_i \times)(\omega_i \times) & 0 \end{bmatrix} \tag{4.49}$$

similarly we can define gravitational wrench acting on link [28].

From the above equations we can write dynamics equation of a floating base robot equation 4.41 can rewrite as

$$Y\phi = \begin{bmatrix} Y_1 \\ \begin{bmatrix} Y_2 \\ \dots \\ Y_{N_j} \end{bmatrix} \end{bmatrix} \phi = \begin{bmatrix} Y_1 \\ Y_\tau \end{bmatrix} \phi = \begin{bmatrix} 0 \\ \tau \end{bmatrix} + \sum_{i=1}^{N_B} \begin{bmatrix} {}^1 X_i^* \\ J_i^T \end{bmatrix} f_i^x \tag{4.50}$$

the dynamic regressor Y is computed by the base regressor and the torque regressor details are found from [28] & [33].

To calculate the regressor Y_1 we write from the float base dynamics equation 4.50 using set of wrench regressor :

$${}^1 f_1 = \sum_{i=1}^{N_b} ({}^1 X_1^* ({}^i f_i^{B_{a_g}} -^i f_i^x)) \tag{4.51}$$

combining equation 4.41 & 4.51 yields to

$$\sum_{i=1}^{N_b} ({}^1 X_i^*)^i A_i \emptyset_i = \sum_{i=1}^{N_b} ({}^1 X_i^*)^i f_i^x = Y_1 \emptyset \tag{4.52}$$

So we gather from this equations

$$Y_1 = [\ {}^1 X_1^* ({}^1 A_1^{a_g}), {}^2 X_2^* ({}^2 A_2^{a_g}), \dots, {}^1 X_{N_b}^* ({}^{N_b} A_{N_b}^{a_g})\ ] \tag{4.53}$$

For torque estimation Fumagalli et al [34] proposed a theoretical framework that exploits embedded force/torque (F/T) sensors to estimate internal and external forces acting on the floating base dynamics link with multiple branches. Theoretically, he proposed framework allows to virtually relocate the available F/T sensors anywhere along the chain. Recursive Newton-Euler steps can prove that relocation relies solely on inertial parameters, velocities and accelerations of rigid links in between the real and virtual sensor details are from [35]. The idea here is the floating base structure is divided into subchains obtained by cutting the original structure at the level of F/T sensor. Then each subchain can be mathematically treated as an independent floating base structure for dynamic equation 4.50, where wrench measure at the F/T sensor can be considered as an external wrench.

4.2.3 Parameters

Given the dynamic regressor Y, it is known that only subspace of the $\mathbb{R}^{10}$ inertial parameters space affects the dynamic equation [36]. This subspace can be called as identifiable subspace for regressor Y, written as I_Y and defined as orthogonal complement of the non identifiable subspace N_Y :

$$N_Y = \emptyset \epsilon \mathbb{R}^{10N_b} : Y(q, \dot{q}, \ddot{q})\emptyset = 0 \forall q, \dot{q}, \ddot{q} I_Y = N_Y^{\perp} \tag{4.54}$$

The above definition assumes unconstrained motion of kinematic tree, so that all possible values of q,$\dot{q}$, $\ddot{q}$ are admissible. It can be useful to know the identifiable subspace for only a subset of the domain of the regressor, for example express the concept of identifiable subspace using only static with zero velocity and acceleration act of motions. $D \subseteq \mathcal{R}^{N_j} \times \mathcal{R}^{N_j} \times \mathcal{R}^{N_j}$

$$N_Y^D = \left\{ \emptyset \epsilon \mathbb{R}^{10N_b} : Y(q, \dot{q}, \ddot{q})\emptyset = 0 \forall (q, \dot{q}, \ddot{q}) \epsilon D \right\} \tag{4.55}$$

$$I_Y^D = N_Y^{D^{\perp}} \tag{4.56}$$

The projections of the real parameters on a basis of I_Y are usually called base or minimal inertial parameters. However, it is common in literature to consider only torque measurements or kinematic trees with fixed base, so called base parameters are usually a projection on a basis for the identifiable subspace I_{Y_*}, considering base link as fixed one. By considering only torque and fixed base to obtain the base parameters were presented by Kawasaki et al [37]. More general approach to determine the identifiable subspace, with no assumptions on the regressor structure or on kinematic was proposed by Gautier [38]. This method has drawback of producing only a numerical representation of the identifiable subspace. But in practical to calculate a base for the identifiable subspace of a given regressor, it is possible to use the generic numerical algorithm by Gautier [38]. In the next section idea of Gautier algorithm is presented,

Gautier Algorithm

Given $f = Y(q, \dot{q}, \ddot{q})\emptyset$ where $Y(q, \dot{q}, \ddot{q})$ regressor function is known. N random acts of motion are then generated q_i, $\dot{q}_i$, $\ddot{q}_i$ where i = 1,2...,N. The Y_N matrix can be defined:

$$Y_N = \begin{bmatrix} Y(q_1, \dot{q}_1, \ddot{q}_1) \\ Y(q_2, \dot{q}_2, \ddot{q}_2) \\ ... \\ Y(q_N, \dot{q}_N, \ddot{q}_N) \end{bmatrix} \tag{4.57}$$

For sufficiently random samples and large N, then

$$N_Y = ker Y_N \tag{4.58}$$

, consequently $I_Y = ker Y_N^{\perp} = span(Y_N^{\top})$.
So the problem of finding I_Y is reduced to the easier problem of finding span of $Y_N^{\top}$. Using SVD (Singular value decomposition), the Y_N matrix can be decomposed as

$$Y_N = U \sum V^T \tag{4.59}$$

where U is a $N_{n_m} \times N_{n_m}$ orthogonal matix, V is a $10N_b \times 10N_b$ orthogonal matrix and S is a $N_{n_m} \times 10N_b$ diagonal matrix,with singular values of Y_N on the diagonal, ordered by decreasing magnitude. In case if Y_N matrix is not full rank, some singular values are 0, and decomposition can be written as:

$$Y_N = U \begin{bmatrix} \sum_1 & 0 \\ 0 & \sum_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^\top \\ V_2^\top \end{bmatrix} \tag{4.60}$$

where $\sum_2 = 0$. Then it is easy to see the columns of V_2 from the base of $ker(Y_N)$:

$$span(V_2) = ker(Y_N) N_{Y_N} \tag{4.61}$$

As V is an orthogonal matrix, we have

$$\begin{bmatrix} V_1^\top \\ V_2^\top \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \tag{4.62}$$

$V_2^\top V_1 = 0$, $span(V_1) = span(V_2)^\top$ with this we can say

$$span(V_1) = N_Y^\perp = I_Y \tag{4.63}$$

Then the columns of V_1 as calculated through SVD from Y_N from the basis of the identifiable subspace of $Y(q, \dot{q}, \ddot{q})$. We define this matrix whose columns are a basis of I_Y as b and we have $b = V_1$. For numerical reason, all the singular values calculated in SVD routine on a real machine are different from the floating point value 0.0. A tolerance threshold is then needed to decide when a singular value is sufficiently little to be considered zero. From [par ref 11] the formula used for determining a suitable value adaptively based on the matrix M_N

$$tol = N_{n_m} \times \varepsilon \times \sigma_1 \tag{4.64}$$

where N_{n_m} is the maximum dimension of the Y_N matrix, ε is the machine epsilon and σ_1 is the largest singular value of the Y_N matrix.

4.2.4 Parameters Estimation

Initial parameters estimation is accomplished by collecting sample measurements if kinematic quantities i.e. joint positions, velocities and acceleration and forces acting in robot i.e. external forces and joint torques. The samples measurements of kinematic quantities are then used to compute regressor that linearly related to inertial parameters to the force measurements. This regressor could be the complete dynamic regressor Y in equation 4.50. In caase if only joint measurements are unavailable, only the base link regressor Y_1 from equation 4.53. Alternatively, if only joint positions and torques measures are available, it could be preferable to avoid the numerical calculations of joint accelerations so energy regressor must use from [28]. Independently from the method used to obtain a measure quantity that depends linearly in inertial parameters , considering all samples $f_i \epsilon \mathbb{R}^{n_m} i = 1, 2, ..., Nm$ it is possible to write a

regressot for all the available measurements in single equation

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} \emptyset = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix} \tag{4.65}$$

Also if the collected samples are sufficiently exciting the equation 4.65 is almost always intermediate in $\emptyset$. So infinite least square solutions exist. As only the elements of identifiable subspace can be identified, to avoid numerical issues it is possible to compute directly the base parameters $\emptyset_b = B^\top \emptyset$,
where B is a matrix whose columns are a base for I_Y, that can be calculated from an arbitrary structure using Gautier algorithm explained in earlier section.

$$\begin{bmatrix} Y_1 B \\ Y_2 B \\ \vdots \\ Y_N B \end{bmatrix} \emptyset_b = \begin{bmatrix} Y_1^b \\ Y_2^b \\ \vdots \\ Y_N^b \end{bmatrix} \emptyset_b = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix} \tag{4.66}$$

Using this equations only the base parameters, that are a projection on a base of identifiable subspace I_Y can be estimated. A projection on this subspace is sufficient to obtain many quantities of interest in robotics that solely depends on the inertial parameters. The equation 4.65 can be used to estimate the parameters $\emptyset_b$ using least squares presented in Non parametric modeling part. As we are using incremental update here there will be numerical instability to over this issue cholesky update was preferable than sherman morrison update.

4.2.5 GURLS Implementation For Inertial Parameters

For identifying inertial parameters, the body of the robot should be static and only the left arm should be dynamic (i.e. movement in left arm). By doing this other part of the body weight doesn't conflict with the left arm in estimating inertial parameters. Since the robot arm is embedded with the F/T sensor. An estimation algorithm is derived from Newton-Euler equations,and used the base force sensor measurements and manipulator (i.e. iCub left arm) joint positions and velocities. No direct measurement of the arm's joint torque or force are required. Adaptive filtering algorithm eliminates the need for difficult to measure joint accerlation.

we see the in *Section* 4.1 how the regularization parameter λ effects in RMSE error. In order to prevent from overfitting regularization parameters is used. For this experiment leave-one-out and holdout approach was choosen for getting optimal λ value. In non-parametric modeling we see that using different approaches to obtain the λ show some difference in error. The main advantage of rigid body dynamics[6] model is that it provides a global and unique relationship between joint trajectories $q, \dot{q}, \ddot{q}$ and the torques τ. With this evidence, we pose an interest in regularization parameter λ effects on inertial parameters.

Earlier it was used $\lambda = 1$, comparing the inertial parameters obatianed with this default (i.e. 1) value there is some change in inertial parameters obtained through Leave-one-out approach & holdout approach.The regularization parameter for both approaches is same in linear model, so their inertial parameters will be same. However, using dynamic regressor the predictions for the force-torque measurements are accurate with actual F/T sensor measurement values. This because the regressor matrix computed using Gautier algorithm is well posed apart from this we are using

Inertial Parameters	Fixed lambda (set to 1)	Leave-one-out cross validation/Holdout approach
1	-0.0201	-3.1769
2	-0.2628	1.5349
3	0.15506	-1.6876
4	0.19480	1.4711
5	-0.3954	-3.4833
6	0.06456	-1.5095
7	0.09521	-0.4808
8	0.04200	3.0720
9	-0.02018	1.2524
10	-0.02855	1.5048
11	0.0722	-0.1308
12	0.7530	0.8666
13	-0.2044	-1.5486
14	-0.2757	0.6470
15	-0.4010	-1.1930
16	0.3315	1.1051
17	0.05401	-2.2087
18	0.04743	-0.51895
19	-0.0146	0.4882
20	0.0609	-0.8363
21	0.0945	0.1272
22	-0.0022	0.8602
23	-0.0281	-0.1876
24	-0.1728	-0.7472
25	0.1638	0.2713
26	-0.0250	0.0719
27	-0.0691	-0.4890
28	0.0307	0.3191
29	0.0947	0.0811
30	0.2171	0.1429
31	-0.0758	-0.4715
32	-0.1379	-0.1846
33	-0.1116	0.0089
34	-0.2156	-0.2190
35	1.2101	1.2108
36	2.6189	2.61800
37	-69.2751	-69.25190
38	26.9123	26.9222
39	11.6651	11.6703

Figure 4.21: Inertial Parameters obtained for iCub left arm

linear model. In this we are using incremental learning with cholesky update instead of sherman morrison update.

Since we don't have ground truth we cannot compute RMSE error. In this altogether we have 39 inertial parameters which include mass, center of mass, inertia matrx, F/T sensor offsets, cad offsets etc. In fact we need only 10 inertial parameters to characterize the dynamic model of the robot. Gauiter[38] proposed a solution some inertial parameters which remains unchange are the standard parameters they can be omitted.

It is not appreciated to plot them because we are not sure which we are ineterested in & which are offsets. For our convenience they are plotted to show the change in inertial parameters with different approach. From the *Figure* 4.22 we can see that there is change in inertial parameters.It is evident that taking the advantage of λ that prevent from overfitting to give accurate prediction in non-parametric modeling, we can say that optimal λ yields to more accurate in obtaining inertial parameters than assumed one (i.e. fixlambda : 1 in our case). From results plotted we can see that there is change in inertial parameter with different λ, apart from this parameters from 20 are remains stable we can consider them as standared parameters this can be omitted. However, there are 20 more parameters, finally we are interested in 10 inertial parameters (i.e. they are mass, centre of mass and inertia matrix) need to work in this direction to obtain them.

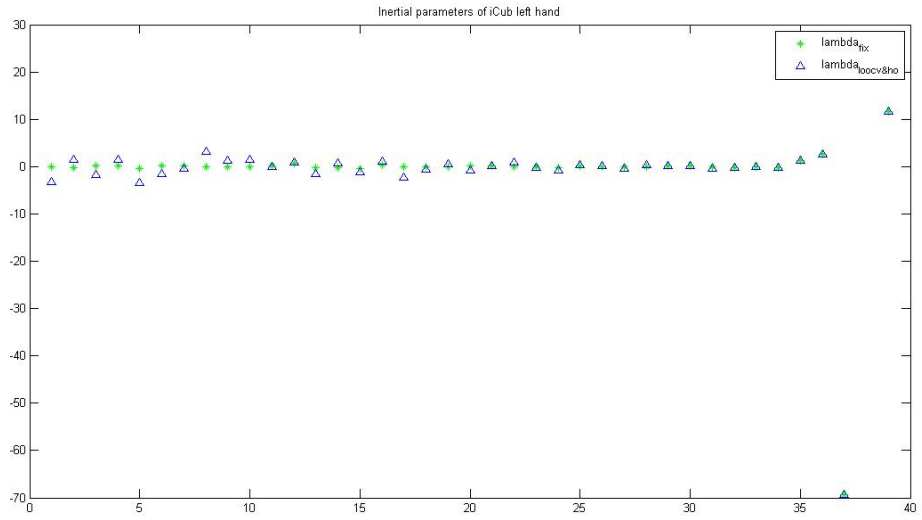

Figure 4.22: Effects of regularization parameter on Inertial parameters estimation

Chapter 5

Experimental Results

There are some parameters which yields to improve in results and sometimes they may be worst because not selecing appropriate parameters. Investigated on the tuning parameters, for this series of experiments are done.

5.1 Batch Learning Experiments

- **A. Change in Cross Folds :** The idea is to divide training set into training and validation set explained in *Section* 4.1.5.To summarize, In this training set is divided into K-cross folds, where K is specified on our interest. This training set is divided into into K portions, $K-1$ portions are used for training and K_{th} portion is used for validation set. This K_{th} portion is changes depending on the K values specified. Once empirical error is computed, regularization parameter and kernel parameter associated with the lowest empirical error is chosen for testing. The advantage of cross folds can be observed in holdout approach because loocv has cross validation due to this there will not much improvement in results. *Figure* 5.1 show idea of cross folds.

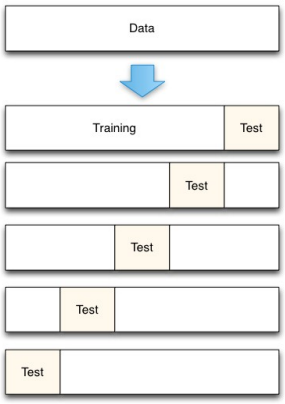

Figure 5.1: General Overview of Cross folds

This is tested is done on RBF kernel with different parameter selection approach i.e. leave-one-out cross validation and holdout approach with cross folds 5. *Figure* 5.2 shows the rmse error for forces and torques with change

in cross folds. From there results in *Figure* 5.2 we can see that error is reduced when compared with the cross

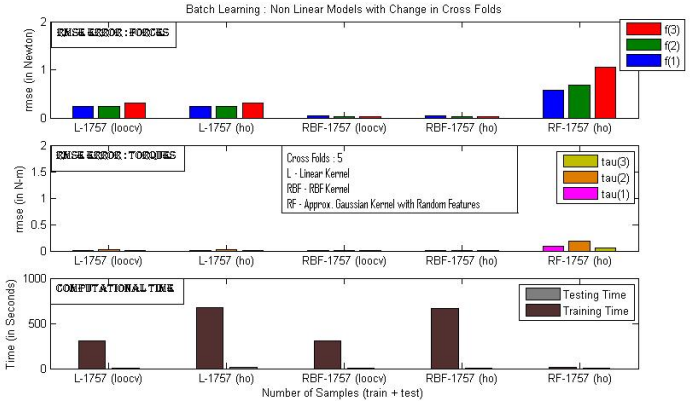

Figure 5.2: Change in Cross folds for Non Linear Models

folds 5 over 1 in *Figure* 4.16 from *Section* 4.1.9. Parameter selection approach also plays key role in this, computational time for leave-one-out cross validation is more when compared to the holdout method. Even though error is decreased by 0.3 times but the computational time is more because computing empirical error for each cross fold is time consuming.The error with cross folds 1 in *Figure* 4.16 & 5 in *Figure* 5.2 there is no much difference in leave-one-out approach.

- **B. Change in Number of samples, Cross folds and limiting guess of parameters :** The intution of doing this experiments, if we limit the guesses for hyper parameters they yiled to worst results. We see in the earlier section number of samples increases error will reduce but it is inversely proportional to computational time. So, I limited the guess to 3 for the parameters, which is set to 20 for regularization parameter lambda and 25 for kernel parameter sigma by default. By limiting the guess there is possibility of both increase or decrease of results because for regularizarion & kernel parameter we cannot promise the optimal value. To overcome this I used cross folds 5. Results shown below with samples size 5% of whole dataset, cross folds 5 and parameters guess limited to 3.

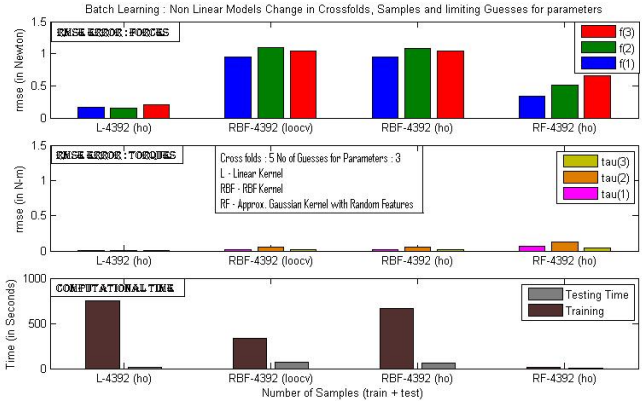

Figure 5.3: Change in Samples,Cross folds and limiting guesses for Non Linear Models

The above results may be acceptable but it is not good idea to limit the guesses for regularization and kernel parameter. In the experiment, guesses are limited to 3 from 25 & 20. In this case if good and accurate parameter values doesn't fall in the 3 guess then we can see worst results ever. This experiment is not good to conduct but the intuition behind this experiment is limiting of guess may do favor in reducing computational time but one cannot be confident to say results will improve. It depends incase fortunately accurate parameter values fall in the guesses specified.

- **C. Linear Model Vs Non Linear Model :** In reality linear models are restricted, we are interested in Non linear models. This is the comparison between linear models and Non linear models. *Figure* 5.3 shows the results for linear models and Non linear models with same amount of dataset and all parameters are set to default. It is evident from the image, Non-linear models error is almost same as linear models with different

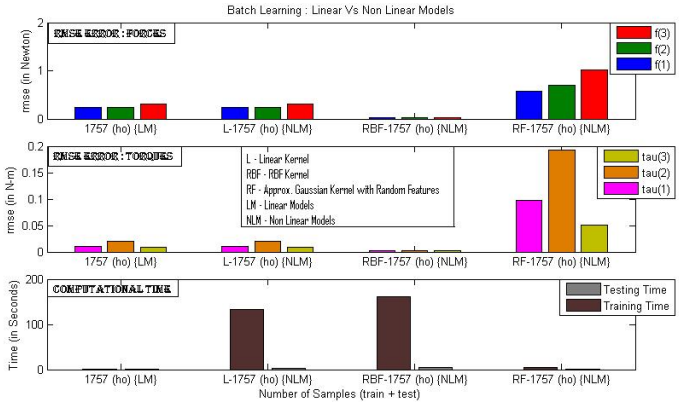

Figure 5.4: Linear Models Vs Non Linear Models in Batch Learning

kernel methods. But Gaussian kernel approximation with random features is slightly more than other methods and linear models. In the above experiment RBF kernel outperforms but high in computational time. We are interested in Gaussian kernel approximation with random features because it can deal with high dimensions and huge dataset which is necessary in reality with less computational time.

5.2 Incremental Learning Experiments

- **A. Change in Number of Features :** This experiement is for Incremental Learning with Sherman Morrison and Cholesky update discussed in *Section* 4.1.10. In reality most of the systems are Non linear, so we are interested in Non linear models over linear models. In non linear models we are interested in kernel method to get accurate predictions, which can deal with huge data, high dimension and computationally efficient. For this reason we prefer, Gaussian kernel approximation using random features with an efficient update and able to make system numerically stable, experiment was conducted on increasing random features . *Figure* 5.4 shows the increase in random features in incremental learning for Sherman Morrison and Cholesky update. For our convinence on the horizontal axis clearly indicated the number of samples used, approach used for getting regularization parameter and which method we are using for the experiment (i.e. SM is Sherman Morrison & CU is Cholesky Update). From the results in *Figure* 5.4, few observations are made, the number

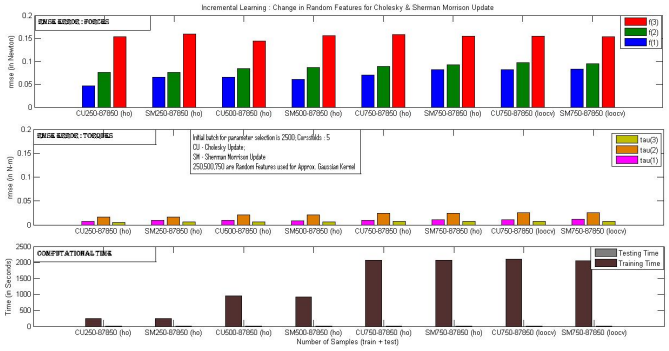

Figure 5.5: Change in Number of Random Feature for Non Linear Models with Sherman Morrison and Cholesky updates

random features increases there is small change in rmse error for Sherman Morrison update when used different approaches, but computational time is inversely proportional to number of random features. One of the strength of cholesky update is it is almost stable in all cases when compared with Sherman Morrison update. However, in reality lowest number of random features will be made as choice with Cholesky update because of its stability and computationally efficient. Last two bars are used leave-one-out cross validation approach for parameters selection with high number of random features 750, so there is small change in error with Cholesky update, it is also evident if we closely observe in all the cases Sherman Morrison update is changing in all the experiments with small change in rmse error..

Chapter 6

Conclusion

In recent years, learning models from data has become an interesting tool for robotics.In this comparision of non-paramateric & parametric learning methods for inverse dynamics models are examined. In non-parametric model approximating of the function that describes the relationship between inputs (i.e. joints position, velovity and accelrations) and outputs (i.e. forces and torques). This was performed for both linear and non-linear models. In real life robots, linear systems are too restrictive, apart from this the covariance matrix is well posed so it performs better in batch learning but leads to high error in incremental learning due to instabilities in covariance matrix. In most of the applications incremental learning is prefered than bacth learning. Due to this issues in linear models are not efficient in robotics for safe operation in unstructured environment. However, non-linear models provide an efficient solution towards this context. All the family of kernels like RBF, polynomial, etc will not perform well because they are restricted with number of input features, needs lots memory and computational time. To alleviate this issue in non-linear models, Gaussian kernel approximation with random features sounds as an alternative solution for non-linear models. These are used in real life with less number of random feature like 250 gives better almost accurate predicitions with few seconds of computational time even with large amount of data. Due to sensitivity to roundoff in error Sherman-Morrison update is numerically unstable, to overcome this cholesky update is preferred. It is stable even with increase or decrease in random feaatures, high dimensional data etc. With more number of features we will penalised by computational time.

In parametric learning method our aim is to identify the small set of significant parameters i.e. mass, center of mass & inertia matrix using Gautier algorithm. Due to presence of cad offsets & F/T offsets it not possible to categorize them for results obtained. However, investigated on regularization parameter effects estimational parameters. Since we don't have ground truth we cannot say they are accurate one. From theoritical point of view, role of regularization parameter is to prevent overfitting of data, this will give accurate predicitons in output. In addition to this using Newton-Euler algorithm we are getting accurate predictions of forces and torques which are almost equal with this evidence we can say that inertial parameters identified are close to true ones.

This are the techniques that are used in robotics for safe operation. However learning is common in both but context is different. Strongly motivated to obtain the accurate inertial parameters (i.e. mass, center of mass and interia matrix). This techniques are used in object recognition, safe operations in unknown environments, interacting with people and many more. In future to extend the work, in combining both the methods (i.e. parametric and

non-parametric models) for robot tracking and control.

Appendix

This are the basic tutorials that are used to install the GURLS Libraries and to get experience with the ICUB environment via simulator and real robot as well.

GURLS : GURLS stands for the Grand Unified Regularized Least Squares is a software library for regression and (multi-class) classification based on the Regularized Least Squares (RLS) loss function. The library takes advantages of some favorable properties of regularized least squares algorithm and is tailored to deal with particular with multi-category/multi-label problems. The packages comprises of useful routines to perform automatic parameter selection can handle computation with very large matrices by means of both memory-mapped storage and distributed task execution. For more information about GURLS algorithms, Documentation & Installation* of GURLS (GURLS++), BGURLS(BGURLS++) and Recursive RLS [42]. The algorithms used by GURLS is explained briefly in State of the Art earlier.

Installation and basic tutorials to get familiar with libraries (started with demo). The GURLS manual will serve better for this [43]. In fact there are separate versions for C++ and Matlab users. One can go according to their choice.

In GURLS library there are separate modules to deal big data sets (big learning scenario). The idea of the algorithms is same as GURLS in order to deal big data sets it relies on simple interface developed ad-hoc and called Gurls Distributed Manager (GDM). For more information refer to complete manual [44].

ICUB INSTALLATION:-

The Installation* of all the dependences and sources were done from Icub Wiki [45]. After installation of the modules and simulator (TO make sure all installations are done correctly with the help of 'check your installation'), one can proceed with the basic tutorials which are necessary to know and to get familiar with Icub environment. The basic tutorials gives the idea of basic compilation, visual processing, to control in the operational space [46]. Depending on the interest, one can go with the specific modules given in the tutorials.

*Installation :- All the installations are done in Ubuntu 12.04 LTS.

Bibliography

[1] Arjan Gijsberts,Incremental Learning for Robotics with constant update complexity (2011), Ph. D. Thesis, Italian Institute of Technology and University of Genoa.
[2] Cortes, C. & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20:273297.
[3] http://lcsl.mit.edu/courses/mlcc/
[4] http://en.wikipedia.org/wiki/Supervised_learning
[5] Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2006). Robot Modeling and Control. Wiley.
[6] Siciliano, B. & Khatib, O. (2008). Springer Handbook of Robotics. Springer.
[7]http://www.cs.cmu.edu/~schneide/tut5/node42.html
[8] Tikhonov, A. N. & Arsenin, V. Y. (1977). Solution of Ill-Posed Problems. V.H. Winston and Sons, Washington D.C.
[9] Hoerl, A. E. & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 12:5567.
[10] Hinton, G. E. (1989). Connectionist learning procedures. Artificial Intelligence, 40:185234.
[11] Poggio, T. & Girosi, F. (1990). Networks for approximation and learning. Proceedings of the IEEE, 78(9):14811497.
[12] Girosi, F., Jones, M., & Poggio, T. (1993). Priors stabilizers and basis functions: From regularization to radial, tensor and additive splines. Technical report,Massachusetts Institute of Technology.
[13] Rifkin, R., Yeo, G., & Poggio, T. (2003). Regularized least squares classification. In: Advances in Learning Theory: Methods, Model and Applications, volume 190, pages 131154. VIOS Press.
[14] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer-Verlag New York,Inc.
[15] http://crsouza.blogspot.it/2010/03/kernel-functions-for-machine-learning.html
[16] Aizerman, M. A., Braverman, E. M., & Rozoner, L. I. (1964). Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control,25:821837.
[17] Wahba, G. (1990). Spline Models for Observational Data, volume 59 of CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM).
[18] Scholkopf, B. & Smola, A. J. (2001). Learning with Kernels: Support Vector Machines, Regularoization, Optimization, and Beyond. MIT Press.
[19] Rahimi, A. & Recht, B. (2008a). Random features for large-scale kernel machines. In: Advances in Neural Information Processing Systems 20, J. Platt, D. Koller, Y. Singer, & S. Roweis, ed.,pages 11771184. MIT Press.
[20] Bochner, S. (1933). Monotone funktionen, stieltjessche integrale und harmonische analyse. Math-ematische Annalen, 108:378410. 10.1007/BF01452844.
[21] Sayed, A. H. (2008). Adaptive Filters. Wiley-IEEE Press.

[22] Hager, W. W. (1989). Updating the inverse of a matrix. SIAM Rev., 31:221239, 1989.
[23] Golub, G. H. & Loan, C. F. V. (1996). Matrix computations (3rd ed.). Johns Hopkins University Press.
[24] Lo Gerfo, L., Rosasco, L., Odone, F., Vito, E. D., & Verri, A. (2008). Spectral algorithms for supervised learning. Neural Computation, 20:18731897.
[25] Horn, R. A. & Johnson, C. R. (1986). Matrix Analysis. Cambridge University Press.
[26] An, C., Atkenson, C. and Hollerbach, J., "Estimation of Inertial Parameters of Rigid Body Links of Manipulators," Proc. of the 24th IEEE Conf. on Decision & Control, pp.990-995, Florida, December 1985.
[27] Goldebberg, A., Apkarian, J. and Smith, H., " An Approach to Adaptive Control of Robot Manipulators Using the Computed Torque Technique," ASME Journal of Dynamic Systems, Measurement, and Control, Control vol.111, pp. 1-8, 1989.
[28] R. Featherstone. A Beginners Guide to 6-D Vectors (Part 1). Robotics Automation Magazine, IEEE, 17(3):83 94, sept. 2010.
[29] R. Featherstone. Rigid body dynamics algorithms, volume 49. Springer Berlin:, 2008.
[30] K. Ayusawa, G. Venture, and Y. Nakamura. Identification of humanoid robots dynamics using floating-base motion dynamics. In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pages 28542859. IEEE, 2008.
[31] M. Mistry, S. Schaal, and K. Yamane. Inertial parameter estimation of floating base humanoid systems using partial force sensing. In Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on, pages 492 497, dec. 2009.
[32] Harold V. Henderson and S. R. Searle. Vec and vech operators for matrices, with some uses in jacobians and multivariate statistics. Canadian Journal of Statistics,7(1):6581, 1979.
[33] Silvio Traversaro, Andrea Del Prete, Riccardo Muradore, Lorenzo Natale & Francesco Nori Inertial Parameter Identification Including Friction and Motor Dynamics 2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2013),Atlanta,Georgia, USA; 10/2013.
[34] Matteo Fumagalli, Serena Ivaldi, Marco Randazzo, Lorenzo Natale, Giorgio Metta, Giulio Sandini, and Francesco Nori. Force feedback exploiting tactile and proximal force/torque sensing. Theory and implementation on the humanoid robot iCub.Autonomous Robots, 33(4):381398, 2012.
[35] M. Randazzo, M. Fumagalli, F. Nori, L. Natale, G. Metta, and G. Sandini. A comparison between joint level torque sensing and proximal F/T sensor torque estimation: implementation on the iCub. In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, pages 41614167. IEEE, 2011.
[36] John Hollerbach, Wisama Khalil, and Maxime Gautier. Model Identification. In Bruno Siciliano and Oussama Khatib, editors, Springer Handbook of Robotics, pages 321344. Springer Berlin Heidelberg, 2008.
[37] H. Kawasaki, Y. Beniya, and K. Kanzaki. Minimum dynamics parameters of tree structure robot models. In Industrial Electronics, Control and Instrumentation, 1991. Proceedings. IECON 91., 1991 International Conference on, pages 1100 1105 vol.2, oct-1 nov 1991.
[38] M. Gautier. Numerical calculation of the base inertial parameters of robots. Journalof Robotic Systems, 8(4):485506, 1991.
[39] S. Dubowsky G. Liu, K. Iagnemma and G. Morel. A base force/torque sensor approach to robot manipulator inertial parameter estimation. In Robotics and Automation, 1998. ICRA 1998. Proceedings of the 1998IEEE International Conference on, 1998.

[40]L. Sciavicco and B. Siciliano,Modeling and control of robot manipulators,MacGraw-Hill, 1996.

[41] J. Swevers, C. Ganseman, D.B. Tukel, J. De Schutter, and H. Van Brussel.Optimal robot excitation and identification.IEEE Trans. on Robotics and Au-tomation, 3(5):730740, 1997.

[42] `https://github.com/LCSL/GURLS`

[43] `https://github.com/LCSL/GURLS/wiki/3-User-Manual#wiki-User_Manual`

[44] `https://github.com/LCSL/GURLS/wiki/3-User-Manual`

[45] `http://wiki.icub.org/wiki/Linux:Installation_from_sources.`

[46] `http://wiki.icub.org/iCub/main/dox/html/icub_tutorials.html`