Collaborative Robotics : Using Programming by Demonstration and Dynamical Systems for Robot Learning
Spring Semester Project

Rohun Tripati
Supervised by: Prof Eric Nyiri
Prof Olivier Gibaru

June 18, 2014

Abstract

This report presents the methodology used to create a model for robot learning using Programming by Demonstration (PbD). Gaussian Mixture Modeling(GMM) is used to create Gaussian functions based on the given set of demonstrations. The motion learned is modeled using a time-invariant nonlinear Dynamical System(DS) and global asymptotic stability of the motion is ensured by placing sufficient conditions on the fixed law. This report introduces each of the key areas of the project and a step by step implementation of each. To conclude, we discuss the results obtained and the future directions for this project.

Contents

1 Introduction

Collaborative robotics is about humans and robots working together in the same space. For this we require robots that can work in human-like work-space without safety fencing. The recent introduction of compliant robots into the robotics market has opened up a host of new human-centric research possibilities in robot learning. Robots are no longer put behind fences (as in large manufacturing plants) and so there is a need to increase their capability to work collaboratively with users. With the fast development of these new robot technologies, one key advancement for robot learning by imitation is to encode the learned skills in probabilistic and statistical models.

Programming by Demonstration(PbD), also known as learning by imitation, is a subset of Supervised Learning that examines methods by which a robot learns new actions according to guidance provided by humans. In PbD, mapping between the state of the robot and the action to be taken at that state, is learned from a set of demonstrations and examples provided by the teacher. The teacher can be a human or robot demonstrator. For using PbD one does not require the expert knowledge in this mapping and thus it has major advantages as it allows non-robotics-experts to interact with robots. Furthermore, demonstrating to the robot is advantageous, as learning by demonstration is a very intuitive medium for communication for humans.
Each demonstration in this report is defined as a set of points(trajectory) to be followed by the robot arm. In section 3, we will describe the problem's two fundamental phases, gathering the examples and deriving the mapping from such examples.
We have used point to point motion to describe the path to be followed, as it has many advantages. For example, it can be used to break down complex tasks to simpler ones for easy mapping. Typical example of point to point movements is, as demonstrated in the image, of picking and placing an object at a different position.

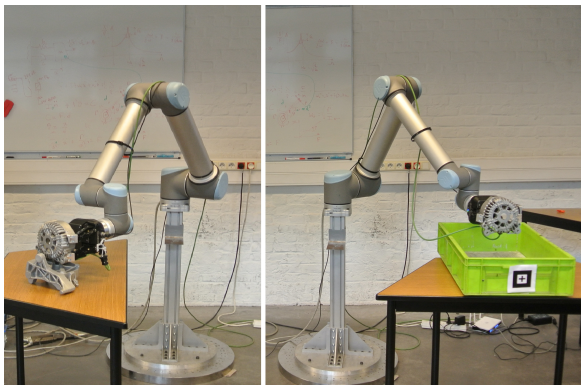

Figure 1: Pick and place, an example of point-to-point motion

1.1 Formalism

We model the complete state of the system using an un-ambiguous state variable $\xi \in R^d$, where R^d is the state space of the robot. For example, ξ can store the 3D Cartesian position of the end-effector at a given instant or the joint angles of the robot and so on. For example for a Cartesian 2D system, the state variable encodes as $\xi = \begin{bmatrix} x \\ y \end{bmatrix}$.

The demonstrations that we supply to the system are governed by the following fixed law.

$$\dot{\xi} = f(\xi) + \epsilon \tag{1}$$

Here f is a nonlinear continuous and continuously differentiable function with a single equilibrium point ξ^*. ϵ represents a zero mean additive Gaussian noise. This noise can have origin in sensor inaccuracies and/or inaccurate demonstrations by the teacher. From these given demonstrations we wish to obtain a noise-free estimate as given by :

$$\dot{\xi} = \hat{f}(\xi) \tag{2}$$

Based on the above fixed law, there are two important notes:

- The above equation will generate trajectories that do not intersect, even if the original demonstrations did intersect, as according to the equation, for each state only one action can be executed by the robot.
- The motion of the system is uniquely determined by its state ξ. Notice that the motion is not dependent in any way on the time of execution and can easily overcome temporal perturbations.

θ denotes the set of parameters of $\hat{f}$. These are modeled using Gaussian Mixture Modeling(GMM) as will be described in section 3.

1.2 Dynamical Systems

Dynamical Systems(DS) are used to describe the system as it enables a robot to adapt its trajectory instantly in the face of perturbations. A Dynamical System embeds all possible solutions to reach a target into a single function, the fixed law described above, Eq(2). There are two main kinds of perturbations, spatial perturbations, which are sudden displacements of the robot arm in its work-space or of the target position, and temporal perturbations, which are related to delays in execution. We place constraints on the DS to ensure global asymptotic stability and thus the model can overcome spatial and temporal perturbations. Figure 2 is an illustration of how the learned model overcomes spatial perturbations.

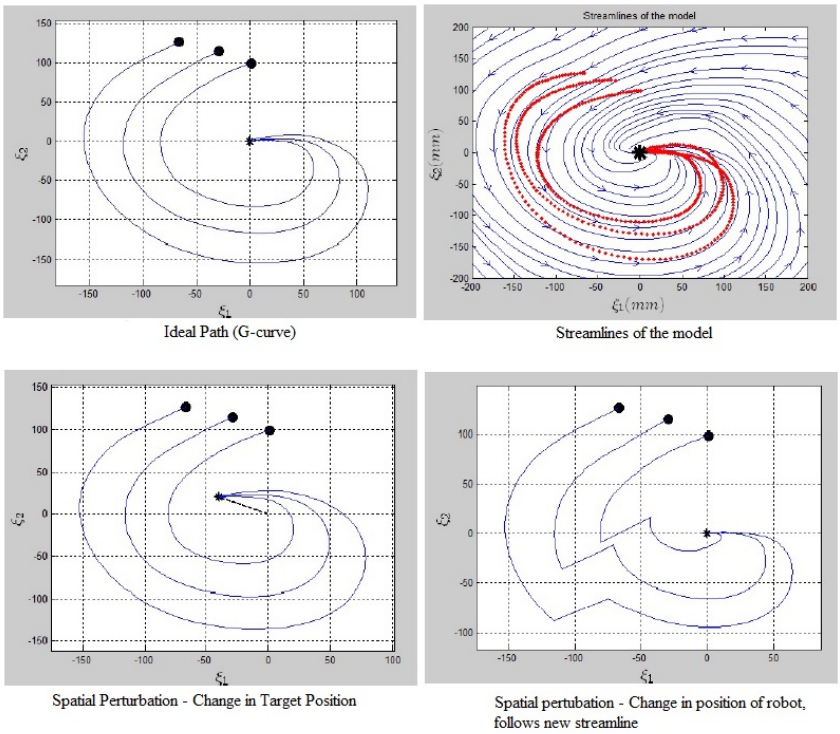

Figure 2: An illustration of how the model deals with spatial perturbations

1.2.1 Global Asymptotic Stability

Theorem 1 *A system is said to be Globally Asymptotically stable(GAS) if for every trajectory $\xi(t)$, from any starting state $\xi_0 \in R^d$, we have $\xi(t) \to \xi^*$ as $t \to \infty$.*
A system is said to be Locally Asymptotically stable(LAS) near or at ξ^* if there exists a subspace D, $D \subset R^d$, such that from any starting state $\xi_0 \in D$, for every trajectory $\xi(t)$ we have $\xi(t) \to \xi^*$ as $t \to \infty$.
Here, ξ^* is the equilibrium state such that $\hat{f}(\xi) = 0$ and R^d is the state space.

To ensure Global Asymptotic Stability for the model learned in our implementation, we will place sufficient conditions. This will be discussed in section 4. The remainder of this report is structured as follows. Section 2 reviews related works in the field of PbD and learning robot motions via statistical modeling. Section 3 describes the methodology we follow and the mathematical basis of the Gaussian Mixture Model. In Section 4 we will elaborate on the Global Asymptotic Stability of the model and constraints placed on the model to ensure the same. Section 5 is devoted to methodology used to communicate with the robot. Section 6 introduces the cost functions and Section 7 discusses the NLopt implementation. Section 8 covers Results and Experiments and finally Future Work and Conclusion are covered in sections 9 and 10 respectively.

2 Literary Review

2.1 Programming by Demonstration (PbD)

Programming by Demonstration (PbD) is the general category of algorithms in which the policy that is used for mapping states to actions-taken is based on demonstration data provided [2]. The main principle of robot PbD is that end-users can teach robots new tasks without explicit programming. Consider, for example, a robot that is required to pick and place an object. Individually this motion requires many different motions, such as reaching the initial point, picking the object, traversing the path required and then placing the object at the target position.

Traditional approaches to robot-control model domain dynamics and are mathematically based models. These models require expertise to program and develop and for thorough testing at each step. This is nontrivial for a robot learner executing actual motions in the real world. Another issue is that of errors or new circumstances arising after the robot has been programmed and probably deployed. Its a very costly process for the robot to be recalled or taken out of service and reprogrammed, considering the new picture. These drawbacks make Programming by Demonstrations(PbD) a very easy choice for policy modeling.

Also referred to as learning by imitation, PbD is based on the the very intuitive way in which humans learn, by demonstration and new methods are being developed by which skills can be transmitted to a robot. PbD covers a broad range of applications and is particularly of interest in cases where robots and human interact in a collaborative way. Research on Programming by Demonstration(PbD) has received great attention in the last decade since it can serve a useful methodology for intuitive robot programming, even by general users without robotics expertise. In various new developments in this field, demonstrations were provided either by teleoperating the robot, or by vision/motion sensors recordings of the user doing the task. The recent hardware and software developments towards compliant and tactile robots are now allowing the user to physically interact with the robot to transfer or refine skills.
Key issues in PbD include the problem of what to imitate, how to imitate and Interface for Demonstration.

2.1.1 What to imitate

What to imitate relates to the problem of determining which aspects of the demonstration should be imitated. For a given task, certain observable properties of the environment may be irrelevant and safely ignored. In the problem that we have dealt with in this report, we assume that the features that are to be imitated are present as the invariant in the demonstration data and we focus on the next problem, how to imitate.

2.1.2 How to imitate

How to imitate consists of methods that determine how the robot will actually perform the learned behaviors when solving the 'What to Imitate' problem. Often, a robot cannot act exactly the same way as a human does, due to physical limitations of the robot. For example, a six-DOF robot arm does not possess the same flexibility as a Human arm . We will discuss how we imitate motion throughout the next section, Methodology and also discuss other statistical methods used to solve the imitation problem in the next subsection.

2.1.3 Interface for Demonstration

The interface used to convey the demonstrations to the robot can influence the motion learned by the robot. Directly recording human motions using image processing and other methods for tracking the human motion is the first kind of interface. The performance of this interface depends in a large part on the receiving sensor's capabilities. Kinesthetic teaching is the newer form of teaching, where the robot is physically guided through the task by the humans. This has been made possible with newer technologies in collaborative robotics. This method also helps overcome the correspondence problem as the demonstrations can be made from the point of view of the robot. Our implementation of this method is discussed in section methodology.

2.1.4 Issues faced

PbD is a relatively young field and there is still a lack of a proper metric to compare algorithms in this field. Most of the research is tested on only one robot and as a result these techniques are often customized to one domain and don't represent a general solution.
Another Issue face is of data-set limitations for PbD. For example, an robot arm might be provided with demonstrations regarding approaching a target from one direction. However during execution, cases might arise in which the robot arm begins from the opposite direction and instead of reaching the target in the shortest possible way, it might loop around to the other side as shown. As discussed in the SEDS model [6], a possible solution is to re-train using new demonstrations as shown.

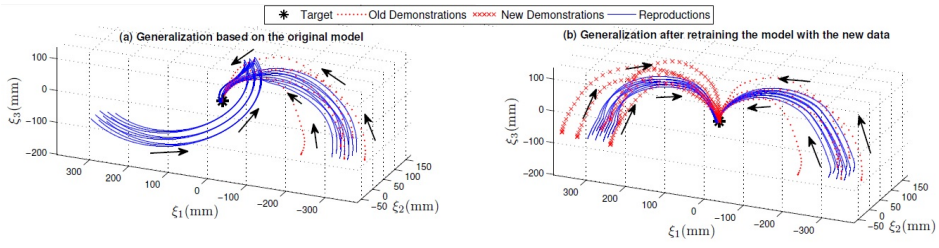

Figure 3: Case of limited dataset

2.2 Models using Statistical Estimation

In this section we cover prevalent models that use statistical approaches for robot learning. The traditional approach in trajectory planning is generally to interpolate or approximate the desired path using a class of polynomial functions and generate a sequence of time based control set points for control of the end effector from the initial point to the target. Spline Decomposition is useful and quick but gives poor estimate of nonlinear trajectories and is time-based thus can be affected by temporal disruptions. As covered in the SEDS model [6], many approaches using Dynamical Systems (DS) have been implemented for robot motion planning. Some common approaches to statistical estimation are Gaussian Process Regression(GPR), which locally finds an optimal model of $\hat{f}$ by maximizing the likelihood that the complete model represents the data well, Locally Weighted Projection Regression (LWPR),which achieves nonlinear function approximation in high dimensional spaces even in the presence of redundant and irrelevant input dimensions using locally linear models, spanned by a small number of uni-variate regressions in selected directions in input space and finally, Gaussian Mixture Regression(GMR) in which the parameters of the Gaussian Mixture are optimized through Expectation Maximization. However all these above mentioned algorithms do not provide a model that is globally asymptotically stable and is rarely even locally asymptotically stable and thus has a tendency to be attracted to spurious attractors. This is illustrated in Fig [4-6].

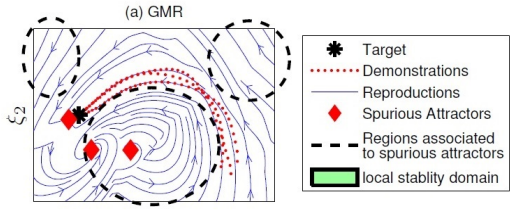

Figure 4: Spurious attractors in Statistical Models - (a)

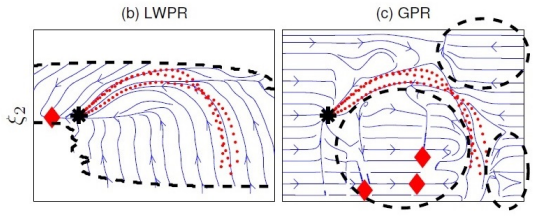

Figure 5: Spurious attractors in Statistical Models - (b)

Stable Estimator of Dynamical Systems [6] ensures global asymptotic stability by placing sufficient constraints on the model. It creates a time-invariant dynamical system to describe the motion and can overcome spatial and temporal perturbations in its

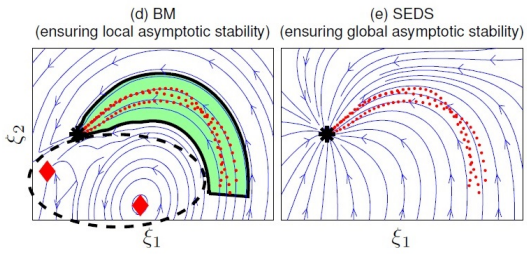

Figure 6: Spurious attractors in Statistical Models - (c)

motion. Another approach based statistical modeling is Binary Merging (BM) [4], to construct a mixture of Gaussian functions so as to ensure local asymptotic stability at the target, hence the model can only applied in a region close to demonstrations as can be seen in the figure 6 above. However, this work still relied on determining numerically the stability region and had a limited region of applicability. The Dynamic Movement Primitives (DMP) offers a way by which a non-linear DS can be estimated while ensuring global stability at an attractor point. Global stability is ensured through the use of linear DS that takes precedence over the non-linear modulation to ensure stability at the end of the motion. However DMP is time-dependent. The choice between using DMP or SEDS to model a motion is application dependent. For example, when the motion is intrinsically time dependent and if only a single demonstration is available, one may use DMP to model the motion. In contrast, when the motion is time-independent and when learning from multiple demonstrations, one may opt to use SEDS. We will discuss the statistical model used by us in the Methodology section.

3 Methodology

In this section we will discuss our implementation model for robot motion learning. First, the mathematical basis for the project will be discussed and then the step by step methodology followed by us.

3.1 Gathering Demonstrations

For gathering each demonstration we manually direct the robot as shown in Figure 7 over the path required. During the demonstration, via TCP connection we record the points(state) of the robot at regular intervals. This can be 2 or 3 dimensional Cartesian coordinates, or Joint-space coordinates.
We collect about 3-5 demonstrations for learning the motion. Each demonstration is modified to have the same number of points and re-arranged into the format required by the program to train the model. The whole process ensures that the demonstrations are made from the point of view of the robot, thus solving the correspondence problem that could arise in PbD.

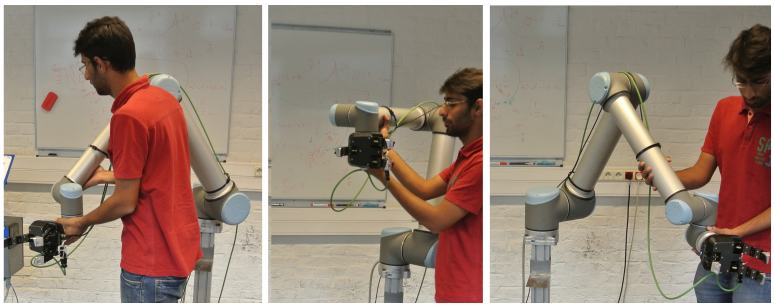

Figure 7: A demonstration for a 3D parabolic motion

3.2 Schematics of Code - Pre-process

Pre-process is the first part in the code which deals with handling the input data. The data provided to the system should follow a specific format, with each demonstration arranged one under another and each having the same number of points. In pre-process part we first calculate the first time derivatives $\dot{\xi}^{t,n}$ for each point in the demonstration data $\xi^{t,n}$ using the equation that follows, $\forall t = 1 \dots T$ and $\forall n = 1 \dots N$, where T is the total number of points in each demonstration and N is number of demonstrations.

$$\dot{\xi}^{t,n} = \xi^{t,n} - \xi^{(t+1),n} \quad \forall t = 1 \dots (T-1)$$

$$\dot{\xi}^{t,n} = 0 \quad \forall t = T$$

We then shift the target position $\xi^*$to the same state, the origin state(for example, in a 3D Cartesian coordinates that is (0,0,0))and all states in each demonstration are shifted accordingly. Thus the system has the origin state as the target position. During the motion however, we can shift the target position as required by simply subtracting the required target coordinates from the present state and supplying that state to the system. This flexibility allows us to easily deal with an abrupt change in the target position, as discussed later.

3.3 Schematics of Code - Initialize

As we have covered in earlier sections, from the demonstrations we must develop a Dynamical System with the fixed law :

$$\dot{\xi} = \hat{f}(\xi) \tag{3}$$

We model $\hat{f}$ using Multivariate Gaussian Mixture Model(GMM), which is a probabilistic model to represent clusters in the total data points or demonstration points provided and represent each cluster by a Gaussian function θ. These Gaussian functions become the unknown parameters of the function $\hat{f}$.
We select the number of Gaussian functions(K) manually or using Bayesian Information Criteria, which is a method that penalizes large increase in the number of parameters when it only offers a small gain in the likelihood of the model, for finding an optimal number of components(K) in the model. BIC is discussed later in the report.

Each Gaussian function, $\theta^k \; \forall k = 1 \ldots K$, is described by its prior π^k, its mean μ^k, and its co-variance $\sum^k$. The mean and the co-variance matrix of Gaussian k are defined by :

$$\mu^k = \begin{pmatrix} \mu_\xi^k \\ \mu_{\dot{\xi}}^k \end{pmatrix} \qquad \Sigma^k = \begin{pmatrix} \Sigma_\xi^k & \Sigma_{\xi\dot{\xi}}^k \\ \Sigma_{\dot{\xi}\xi}^k & \Sigma_{\dot{\xi}}^k \end{pmatrix}$$

Figure 8 is an illustration of how input data is transformed into information stored in a GMM model. The first image is depicts the data and the second one presents the learned GMM model comprising of 4 Gaussians.

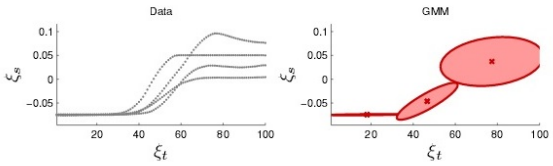

Figure 8: Output of 4 Gaussians from data

In this section of the code we initialize the priors, mus and co-variances of each Gaussian, providing an initial guess. First we run the K-means Clustering algorithm on the data set. K-means aims to identify k number of clusters in n given data points, where each data point belongs to the cluster with the nearest mean. We set K-means to initialize K number of clusters, the number of Gaussian functions we require. We have used K-means implementation of external mathematics library OpenCV for C++ for this step.
Then we run the Estimation Maximization(EM) on the output from K-means algorithm. EM algorithm is an iterative method for finding maximum likelihood estimates of parameters in statistical models, where the model depends on unobserved latent variables. The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.
In section 6, we will see the log-likelihood model used for EM and also the modification of priors, mus and co-variances as required by the initial guess to satisfy the constraints placed on the model for global asymptotic stability.

We represent each point in the demonstration data as $(\xi^{t,n}, \dot{\xi}^{t,n})$, $\forall t = 1 \ldots T$ and $\forall n = 1 \ldots N$, where N is number of demonstrations and T is number of points in each demonstration. For each point the probability density function associated with it is,

$$P(\xi^{t,n}, \dot{\xi}^{t,n}; \theta) = \sum_{k=1}^{K} P(k) P(\xi^{t,n}, \dot{\xi}^{t,n}; k) \tag{4}$$

$$\begin{cases} P(k) = \pi^k \\ P(\xi^{t,n}, \dot{\xi}^{t,n}; k) = \mathcal{N}(\xi^{t,n}, \dot{\xi}^{t,n}, \mu^k, \sum^k) \\ \text{is the conditional probability density function.} \end{cases}$$

We obtain the following equation for $\dot{\xi}$ after using the posterior mean method on it, which minimizes the posterior expected value of a loss function (i.e., the posterior expected loss).

$$\dot{\xi} = \sum_{k=1}^{K} \frac{P(k)P(\xi|k)}{\sum_{i=1}^{K} P(i)P(\xi|i)} (\mu_{\dot{\xi}}^k + \sum_{\dot{\xi}\xi}^k (\sum_{\xi}^k)^{-1} (\xi - \mu_{\xi}^k)) \tag{5}$$

Using a change of variable as follows :

$$\begin{cases} A^k = \sum_{\dot{\xi}\xi}^k (\sum_{\xi}^k)^{-1}. \\ b^k = \mu_{\dot{\xi}}^k - A^k \mu_{\xi}^k \\ h^k(\xi) = \frac{P(k)P(\xi|k)}{\sum_{i=1}^{K} P(i)P(\xi|i)} \end{cases}$$

We obtain the following equation after substitution:

$$\dot{\xi} = \hat{f}(\xi) = \sum_{k=1}^{K} h^k(\xi)(A^k \xi + b^k) \tag{6}$$

This is the equation we use for determining the action(velocity) of the robot depending on the present state(co-ordinates) of the robot's end-effector. To ensure Global Asymptotic Stability, we shall place some constraints on the Eq(6) as discussed in the next section.

Each $h^k(\xi)$ term represents the nonlinear weight of the k^{th} Gaussian at that state and makes the function nonlinear. This term allows the function to be flexible enough to model a variety of motions. An example of the effects of the weight term $h^k(\xi)$ are shown in figure 9. It illustrates the effect of three Gaussian functions and their influence on a 1D path. The path followed tends towards the center or mean of each Gaussian locally as it passes by the Gaussian. The lines shown through each Gaussian follows the linear dynamics $A^k\xi + b^k$ with μ^k as the center.

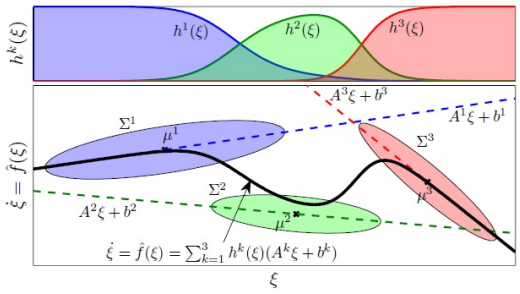

Figure 9: 1D motion in presence of 3 Gaussian functions

4 Ensuring Stability

To ensure global asymptotic stability for Eq(6) we apply the following constraints on it.

Theorem 2 *A trajectory defined by Eq(6) above is globally asymptotically stable at the target ξ^* in R^d if:*

$$\begin{cases} b^k = -A^k \mu^k_{\xi^*} & \forall k = 1 \dots K \\ A^k + (A^k)^T \prec 0 & \forall k = 1 \dots K \end{cases} \tag{7}$$

Where $\mu^k_{\xi^*}$ is the mean at the final state.
$(A^k)^T$ is the transpose of the A^k and $\prec 0$ implies negative definiteness

Proof of theorem 2:
We prove theorem 2 above using Lyapunov Stability Theorem as presented in the SEDS model [6].

Theorem 3 *A dynamical system determined by the function,*

$$\dot{\xi} = f(\hat{\xi})$$

is globally asymptotically stable at the point ξ^* if there exists a continuous and continuously differentiable Lyapunov function $V(\xi) : R^d \rightarrow R$ such that:

$$\begin{cases} V(\xi) > 0 & \forall \xi \in R^d \quad and \quad \xi \neq \xi^* \\ \dot{V}(\xi) < 0 & \forall \xi \in R^d \quad and \quad \xi \neq \xi^* \\ V(\xi^*) = 0 \quad and \quad \dot{V}(\xi^*) = 0 \end{cases} \tag{8}$$

Methodology used in proof: We will present a Lyapunov function that satisfies the above conditions for our case under the two constraints mentioned in theorem 2, thus proving that under the above two constraints, the system is Globally Asymptotically Stable.

Note that in our case the Lyapunov function V is just a function of ξ as $\dot{\xi}$ can be expressed as it can be expressed directly in terms of ξ using $\dot{\xi} = f(\hat{\xi})$. Consider a Lyapunov function $V(\xi)$ of the following form:

$$V(\xi) = \frac{1}{2}(\xi - \xi^*)^T(\xi - \xi^*) \quad \forall \xi \in R^d$$

The function above is a quadratic function and it will have positive value $\forall \xi$ and thus it satisfies Eq(8a). Taking the first time derivative of $V(\xi)$ with respect to time, we obtain:

$$
\begin{aligned}
\dot{V}(\xi) &= \frac{dV}{dt} \\
&= \frac{dV}{d\xi} \times \frac{d\xi}{dt} \\
&= \frac{1}{2}\frac{d}{d\xi}((\xi - \xi^*)^T(\xi - \xi^*)) \\
&= ((\xi - \xi^*)^T\dot{\xi} \\
&= ((\xi - \xi^*)^T f(\hat{\xi}) \\
&= ((\xi - \xi^*)^T \sum_{k=1}^{K} h^k(\xi)(A^k\xi + b^k) \\
&= ((\xi - \xi^*)^T \sum_{k=1}^{K} h^k(\xi)(A^k(\xi - \xi^*) + (A^k\xi^* + b^k)) \\
&= ((\xi - \xi^*)^T \sum_{k=1}^{K} h^k(\xi)(A^k(\xi - \xi^*)) \quad \ldots (A^k\xi^* + b^k) = 0 \\
&= \sum_{k=1}^{K} h^k(\xi)((\xi - \xi^*)^T(A^k(\xi - \xi^*)) \quad \ldots A^k + (A^k)^T \prec 0 \\
&< 0 \quad \forall \xi \in R^d \quad \textit{and} \quad \xi \neq \xi^*
\end{aligned}
$$

Thus the Eq(8b) is satisfied for the above Lyapunov function as $h^k(\xi) > 0 \quad \forall \xi \in R^d$ and using the constraint $A^k + (A^k)^T \prec 0$ .A symmetric $n \times n$ real matrix M is said to be negative(positive) definite if $z^T Mz$ is negative(positive) for every non-zero column vector z of n real numbers. For Eq(8c), we simply substitute $\xi = \xi^*$ into the Lyapunov function.

$$V(\xi^*) = \frac{1}{2}(\xi^* - \xi^*)^T(\xi^* - \xi^*) = 0$$

$$\dot{V}(\xi^*) = \sum_{k=1}^{K} h^k(\xi^*)((\xi^* - \xi^*)^T(A^k(\xi^* - \xi^*)) = 0$$

Thus we have demonstrate that the ODE given by $\dot{\xi} = f(\hat{\xi})$ is Globally Asymptotically Stable, provided the constraints in Eq(7) are respected.

4.1 Overcoming perturbations

4.1.1 Spatial Perturbation

Spatial perturbation is defined as a spatial displacement of the robots end-effector caused by an external source. In case of spatial perturbation of the robot, the velocity to be followed by the robot is re-calculated according to the fixed law and it follows a new path to the target. Another interesting form of spatial perturbation that arises is when

the **target point** is displaced. For example, when during a pick-and-place operation the user interacting with robot shifts the box in which the object has to be placed.
The model learned above can modify its path, in case of change in target position, in a manner similar to the first kind of spatial perturbation. During the training part of the model, the origin state is kept as the target position and this allows us to shift the model easily when the target position is changed by shifting the origin to the new target.

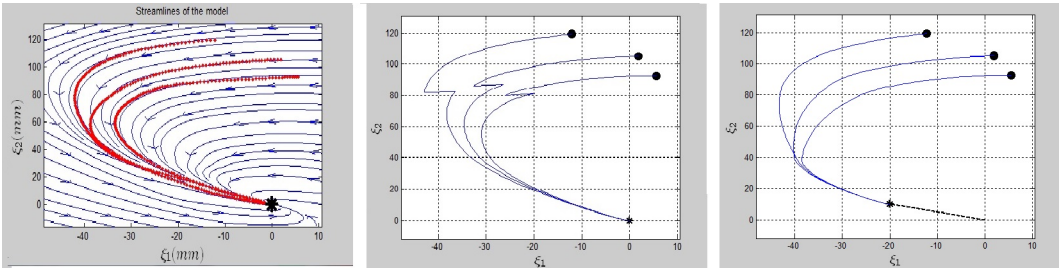

Figure 10: Illustration of spatial perturbations

The first image above is the demonstrations and the stream lines learned by the model for a C-curve. The second and third illustrate the model adapting the path for different cases of spatial perturbations. The first is when the robots end-effector is displaced and the second is change in final point.

4.1.2 Temporal Perturbation

Temporal Perturbation of the model is generally associated with delays in the execution of the motion. These might be due to variety of reasons such as limitations in robot capabilities and/or an inconsistent connection between the robot and the program computer. These perturbations are handled easily by the model as the actions performed by the robot depend only on the state of the robot and are time-invariant and thus the perturbation is overcome by re-calculating the velocity at the state reached after the perturbation.

5 Communicating with UR10

After simulation on MATLAB was completed, the model trained is implemented on a Universal Robots-UR10 robot. For interacting with the robot we used a separate module/project named ”UR10-sending-receiving”. The connection to the robot was established via a Transmission Control Protocol(TCP) connection, which provides reliable delivery protocol between two computers connected on Local Area Network(LAN). In this module we created different modes for communicating with the robot. One can shift between different modes on the program by pressing the correct input key on the computer running the server for the connection. For example, using a set of predefined keys, the user of the program can increase or decrease the speed of execution of motion in real-time.

5.1 Reception

The first mode is ”reception”, which was used to save the received data points from the robot. This mode is used to store demonstrations of as many dimensions as required for training of the robot model and for facilitating reception from the robot at the program end, we have a thread running in an infinite loop claiming the points sent by the robot. Via different keys, one could start and stop the saving process. For example, one needs to disable the storing process when the robot is brought back to the initial position, and start it again for the next demonstration.

5.2 Sending

The second and more important mode of this module is 'Learning'. In this mode the motion learned by the program **offline** is used to execute the motion that was demonstrated by the teacher. In the 'Learning' mode at every iteration the program receives the present state of the robot. Then, using the statistical model learned offline, it calculates the required velocity of the robot and this is then sent to the robot. This loop, called the policy execution, is illustrated in the image below. The rate of sending to robot is once every 33 milliseconds and has been determined via experimentation to avoid overflowing of the buffer at the receiver's(robot's) end. In this mode we also

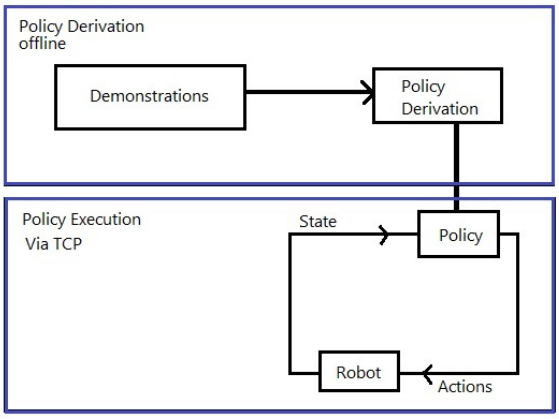

Figure 11: Sending Module

experimented with features such as change of final point/target point of the motion and interacting with the robot-hand attached at the end of the end-effector, for example for pick-and-place kind of motions.

5.3 Integration with Kinect

Microsoft Kinect is a line of motion sensing input devices by Microsoft. Based around a webcam-style add-on peripheral, it enables users to control and interact with their

console/computer through a natural user interface using gestures. Via a named pipe connection (a named pipe is a named, one-way or duplex pipe for communication between the pipe server and one or more pipe clients) we connected to a program that utilized a Kinect device placed at the base of the robot to calculate the distance between the robot and Users/humans in the robot's work space and this distance is used to slow down or stop the robot while in the 'learning' phase, if any user came to close to the robot. This addresses the major issue regarding safety around collaborative robots, that work without safety fencing.

6 Optimization - Learning the Motion

We implement cost functions as described in Stable Estimator of Dynamical Systems (SEDS) model [6], for optimizing the parameters. Using the above obtained initial guess, we apply two different Cost Functions to optimize the parameters. SEDS solves the optimization problem under the constraints required for stability.

6.1 Log-Likelihood Based

The first is a Log-likelihood based cost function. Here θ represents the parameters or the Gaussian functions that are being optimized. $J(\theta)$ is the cost function whose value has to minimized. N is total number of demonstrations and T is total number of points in each demonstration. τ represents all the data points, $\tau = \sum_{n=1}^{N} T^n$. $P(\xi^{t,n}, \dot{\xi}^{t,n}|\theta)$ is given by Eq(4) is the probability density function for data point $(\xi^{t,n}, \dot{\xi}^{t,n})$.
The cost function $J(\theta)$ is as follows

$$min_\theta J(\theta) = -\frac{1}{\tau} \sum_{n=1}^{N} \sum_{t=0}^{T^n} logP(\xi^{t,n}, \dot{\xi}^{t,n}|\theta) \tag{9}$$

The constraints on the cost function are as follows. The next section will address each constraint in detail.

$$\begin{cases} b^k = -A^k \mu_{\xi^*}^k & \forall k = 1 \dots K \\ A^k + (A^k)^T \prec 0 & \forall k = 1 \dots K \\ 0 \leq \pi^k \leq 1 & \forall k = 1 \dots K \\ \sum^k \succ 0 & \forall k = 1 \dots K \\ \sum_{k=1}^{K} \pi^k = 1 \end{cases}$$

6.2 Mean Square Error Implement

The second algorithm is Mean Square Error based model. It minimizes the difference the action(speed) predicted by the model and the data obtained. The cost function is as follows

$$min_\theta J(\theta) = \frac{1}{2\tau} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \|\hat{\dot{\xi}}^{t,n} - \dot{\xi}^{t,n}\|^2 \tag{10}$$

Where the symbols θ, N, T and τ are as explained for Eq(9). $J(\theta)$ is the cost function to be minimized. $\dot{\xi}^{t,n}$ is calculated in initialize and $\hat{\dot{\xi}}^{t,n}$ is calculated directly from Eq(6).

$$\dot{\xi} = \hat{f}(\xi)$$

As, discussed in the SEDS model [6], both these cost functions present Non-Linear Programming(NLP) problems. The SEDS model utilizes Successive Quadratic Programming approach relying on L-BFGS(Limited-memory Broyden-Fletcher-Goldfard-Shanno)to optimize the cost function. Our implementation is derivative free and will be discussed in the NLopt section. Notice, however that it is not ensured that one obtains the globally optimal solution every time one optimizes as the problem is non-convex. On the contrary, changing the initial step size(discussed in the NLopt section) for each NLopt optimization leads us to a different solution(local minima) every time.

7 Implementing the Optimization - NLopt

For optimization we have used NLopt optimization library. This library address general nonlinear optimization problems of the form :

$$min_{x \in R^n} f(x)$$

where f is the objective function and x represents the n optimization parameters. NLopt also has algorithms for solving optimization problems with m non-linear inequality constraints(called NLP, non-linear programming) of the type :

$$fc_i(x) \leq 0 \quad \forall i = 1 \dots m$$

7.1 Adding the constraints

This subsection deals with the process of adding each of the constraints to the optimization problem. The first two constraints are related to ensuring Global Asymptotic Stability of the model.

$$\begin{cases} b^k = -A^k \mu^k_{\xi^*} & \forall k = 1 \dots K \\ A^k + (A^k)^T \prec 0 & \forall k = 1 \dots K \end{cases}$$

For the first constraint we have set the target point to the origin state and thus,

$$b^k = -A^k \mu^k_{\xi^*} = 0$$

$$\implies \mu^k_{\dot{\xi}} - A^k \mu^k_{\xi} = 0$$

The above equality constraint was added to the optimization problem for each Gaussian for each dimension.
The second constraint was relating to the negative definiteness of the $A^k \forall k = 1 \dots K$. A symmetric $n \times n$ real matrix M is said to be negative(positive) definite if $z^T M z$ is negative(positive) for every non-zero column vector z of n real numbers. Here z^T denotes the transpose of z. An equivalent definition is, A symmetric $n \times n$ real matrix M is said

to be negative(positive) definite if all its eigenvalues are negative(positive).
To add this constraint we added inequality constraints requiring that all eigenvalues of the symmetric matrix $A^k + (A^k)^T$ be negative. The last three constraints were required for a feasible statistical model.

$$\begin{cases} 0 \leq \pi^k \leq 1 & \forall k = 1 \dots K \\ \sum^k \succ 0 & \forall k = 1 \dots K \\ \sum_{k=1}^{K} \pi^k = 1 \end{cases}$$

The third constraint was pretty straight forward as the priors π^k represent probability of each Gaussian and have to have values between 0 and 1. We applied this step by bounding the parameters π^k, which is a feature in NLopt.
The forth constraint implies that all co-variance matrices should be positive definite which is their property and was implemented in a similar fashion to how the second constraint was implemented above. Initially we had included the complete the co-variance matrix of each Gaussian and had to include equality constraints on the parameters to keep the matrix symmetric Later, however we reduced the parameters by including only the upper triangular matrix as reduced the parameters used and the constraints required for the optimization.
The last constraint is also a property of the statistical model and implies that sum of probabilities of all the Gaussians equals 1. This was implemented similar to the first constraint using the equality constraint option in NLopt.
We have experimented with three of the algorithms implemented in this library, the first is COBYLA.

7.2 COBYLA

COBYLA stands for Constrained Optimization BY Linear Approximations and its implementation in NLopt is a derivation of Powell's implementation of the COBYLA (Constrained Optimization BY Linear Approximations) algorithm for derivative-free optimization with nonlinear inequality and equality constraints, by M. J. D. Powell [8]. COBYLA constructs successive linear approximations of the objective function and constraints via a simplex of n+1 points (in n dimensions), and optimizes these approximations in a trust region at each step.

We started with COBYLA as it was one of the few implementations in NLopt that supports nonlinear equality constraints. Also, COBYLA is a local derivative free optimization and as we are yet to implement the derivatives of the cost functions mentioned above with respect to the parameters(prior π^k, its mean μ^k, and its co-variance $\sum^k$) as covered in [5] it was the primary choice.

7.3 BOBYQA

BOBYQA denotes Bound Approximation BY Quadratic Approximation. This algorithm is a modified implementation by NLopt derived from the BOBYQA subroutine

of M. J. D. Powell. BOBYQA performs derivative-free bound-constrained optimization using an iteratively constructed quadratic approximation for the objective function [9]. The NLopt BOBYQA interface supports unequal initial-step sizes in the different parameters (by the simple expedient of internally re scaling the parameters proportional to the initial steps), which is important when different parameters have very different scales. However, COBYLA outperforms BOBYQA and because BOBYQA constructs a quadratic approximation of the objective, it may perform poorly for objective functions that are not twice-differentiable.

7.4 AUGLAG

AUGLAG stands for Augmented Lagrangian algorithm. In the NLopt implementation of this method, it develops a Lagrangian comprising of the cost(objective) function and nonlinear inequality/equality constraints as described in [1, 3]. The function generated is essentially the objective function plus a 'penalty' if any of the constraints is violated. This modified objective function is then passed to another optimization algorithm with no nonlinear constraints. The subsidiary optimization algorithm that we used is L-BFGS as used in the SEDS model [6]. If the constraints are violated by the solution of this sub-problem, then the size of the penalties is increased and the process is repeated; eventually, the process must converge to the desired solution. However this algorithm did not produce any positive results when compared to the MATLAB implementation of L-BFGS.

7.5 Initial Step

NLopt has a feature to select the initial step size that the optimizer uses to begin optimization. For derivative-free local-optimization algorithms, the optimizer must somehow decide on some initial step size to perturb x by when it begins the optimization. This step size should be big enough that the value of the objective changes significantly, but not too big if you want to find the local optimum nearest to x. By default, NLopt chooses this initial step size heuristically from the bounds, tolerances, and other information, but this may not always be the best choice. We tried systematically to change the initial step size used in different optimizations. The results are discussed in the section 8.

7.5.1 Repeated optimizations

Another concept that we experimented with was to optimize in a loop the solutions we obtain for the parameters from the previous optimization. One of NLopt's stopping criteria is the maximum number of iterations and so for models that needed a large number of iterations and time to optimize we tried to repeated optimizations by optimizing for a moderate number of iterations multiple times and changing the initial step at each iteration.

7.6 Parameters

The parameters of the model used are the Gaussian functions and their corresponding priors π^k, mus μ^k and sigmas $\sum^k$. The selection of the optimal number of parameters and their dimensionality are discussed in this subsection.

7.6.1 Bayesian Information Criteria

For selecting the number of parameters(number of Gaussians - K) for the optimization we either manually choose K or employ Bayesian Information Criteria(BIC). While optimizing, as one increases the number of parameters, it increases the computational overhead and also increases the time taken to train the model. After particular value of K, increasing K does not provide significant change in the value of the loss function and for finding this optimal value of K, we use BIC. BIC provides us with an appropriate trade-off between decreasing the loss function and complexity of the model(number of parameters used).

$$BIC(\theta) = \tau \times J(\theta) + \frac{n_p}{2} log(\tau) \tag{11}$$

Where $J(\theta)$ is the log-likelihood based cost function, τ is the total number of points in the data set provided and n_p is the total number of free parameters.

7.6.2 Initialization of parameters

As discussed in the subsection Initialize in section 3, we use Estimation Maximization(EM) algorithm for providing initial values for the optimization. The complete implementation to make the initial parameters in accordance with the constraints as is required for stability as discussed in [6] is as follows:

- Run EM on the input data and store the result in π^k, μ^k and the $\sum^k$, $\forall k \in 1 \dots K$
- Define $\hat{\pi}^k = \pi^k$ and $\hat{\mu}^k_\xi = \mu^k_\xi$
- Modify the co-variance matrices as follows to respect the constraints on the model:

$$\begin{cases} \hat{\sum}^k_\xi = \mathbf{I} \times abs(\sum^k_\xi) \\ \hat{\sum}^k_{\dot{\xi}\xi} = -\mathbf{I} \times abs(\sum^k_{\dot{\xi}\xi}) \\ \hat{\sum}^k_{\dot{\xi}} = \mathbf{I} \times abs(\sum^k_{\dot{\xi}}) \\ \hat{\sum}^k_{\xi\dot{\xi}} = -\mathbf{I} \times abs(\sum^k_{\xi\dot{\xi}}) \end{cases} \qquad \forall k \in 1 \dots K$$

  where the $\times$ here represents product of term with corresponding term in other matrix and $abs()$ represents the absolute function and $\mathbf{I}$ is the identity matrix.

- Finally, compute $\hat{\mu}^k$ such that it satisfies Eq(7a).

$$\hat{\mu}^k = \hat{\sum}^k_{\dot{\xi}\xi} (\hat{\sum}^k_\xi)^{-1} (\hat{\mu}^k_\xi - \xi^*)$$

Return : Initial parameter set = $\hat{\pi}^k, \hat{\mu}^k, \hat{\sum}^k$

7.6.3 Dimensionality of parameters

The number of parameters used by our implementation for each Gaussian is 1 for the prior, $2d$ for the mu, taking into account μ_{ξ}^{k} and $\mu_{\dot{\xi}}^{k}$ and $2d(d+1)$ sigma parameters, as we only store the upper diagonal matrix of each co-variance matrix(co-variance matrices are symmetric) and the size of the co-variance matrix is $2d \times 2d$ for each Gaussian. Here d is the number of dimensions in which the model is being trained.

Thus the total number of parameters used for each Gaussian is $(1+3d+2d^2)$. Finally the total number of parameters required by our implementation are : $(1+3d+2d^2)K$, where K is the total number of Gaussians being used for optimization. Thus, in our implementation the number of parameters increases linearly with the number of Gaussians used and quadratically with the number of dimensions the model is trained in. This implementation of dimensionality is not optimal, however, as is discussed in SEDS model [6], which uses only as many parameters as required for computation. For example, MSE requires a lesser number of parameters than likelihood.

8 Experiments and Results

In this section we will compare the results obtained from the C++ code utilizing the NLopt library. The algorithms which will be compared will be COBYLA and BOBYQA. We will also compare the results to the MATLAB implementation provided withe SEDS package [6], which utilizes Successive Quadratic Programming (SQP) approach relying on L-BFGS to solve the constrained optimization problem.

The first successful simulation comprised a model trained using Likelihood cost function and using COBYLA(Likelihood-COBYLA) as the algorithm. In the figure on the left is the result from SEDS-Likelihood model and on the right is Likelihood-COBYLA model.

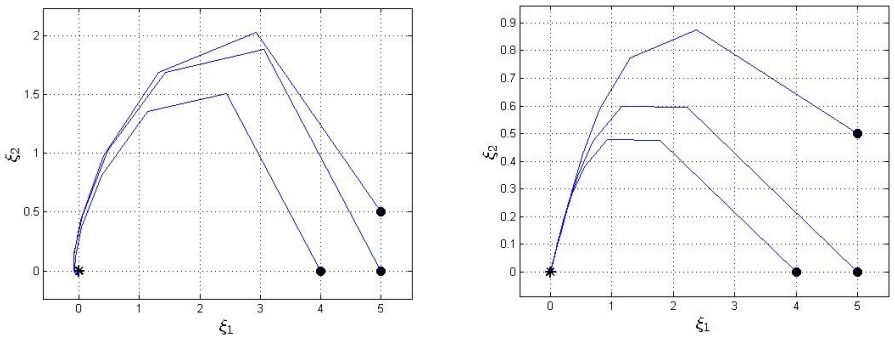

Figure 12: Simulation with three demos, five data points each and using one Gaussian to learn the model. On the right is SEDS-Likelihood and on the left is Likelihood-COBYLA

8.1 Comparing Algorithms - MSE

Working with the cost function Mean Square Error, both MSE-COBYLA and MSE-BOBYQA models gave comparable results as can be seen in simulations in figure 13. MSE-COBYLA model uses MSE based cost function with COBYLA algorithm and similarly MSE-BOBYQA uses BOBYQA algorithm.

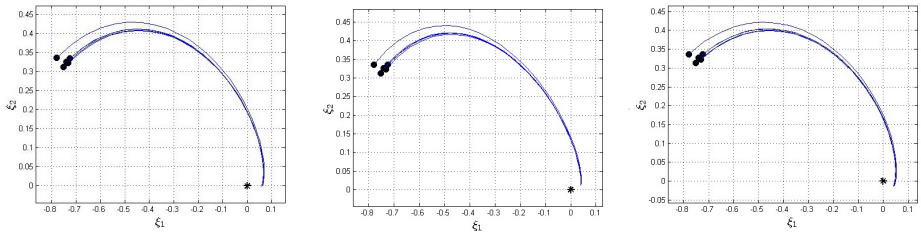

Figure 13: Simulation with four demos and using one Gaussian to learn the model. On the left is SEDS-MSE, then MSE-COBYLA and finally MSE-BOBYQA

It must be noted that the model always converges. In the above simulations it appears that the motion did not converge as the simulation runs for a limited number of iterations and the motion had not converged in simulation yet. However, for all the trained models, the robot motion always converges.
Overall, the algorithm that performs the best is MSE-COBYLA, as seen in figure 13 above. The MSE-BOBYQA at times performs better then MSE-COBYLA but is an unreliable system as it fails to train in some rare situations (whereas MSE-COBYLA always produces a valid solution).

8.2 Comparing Algorithms - Likelihood

Likelihood-COBYLA is the combination of the Likelihood cost function and COBYLA algorithm. This algorithm was highly reliable and almost always produced a valid solution. It however was generally out performed by MSE-COBYLA as seen in the simulation in figure 14. This was different from some of the results obtained in [6], as in that article, the results claim SEDS-Likelihood performs better than SEDS-MSE. One possible factor could be the use of SQP in the SEDS model and derivative-free optimization in our C++ model. Also, when compared to the L-BFGS implementation in the SEDS MATLAB code, the L-BFGS algorithm has a much larger initial step and a much faster rate of convergence to a local minimum, which is rarely the same minima reached by our C++ implementation. Likelihood-BOBYQA, which is the combination of the Likelihood cost function and the BOBYQA algorithm for optimization, rarely succeeded in training a stable model. One possible explanation could be that BOBYQA was not explicitly developed in the NLopt implementation to deal non-linear inequalities even though it handles them well in the case of MSE-BOBYQA.

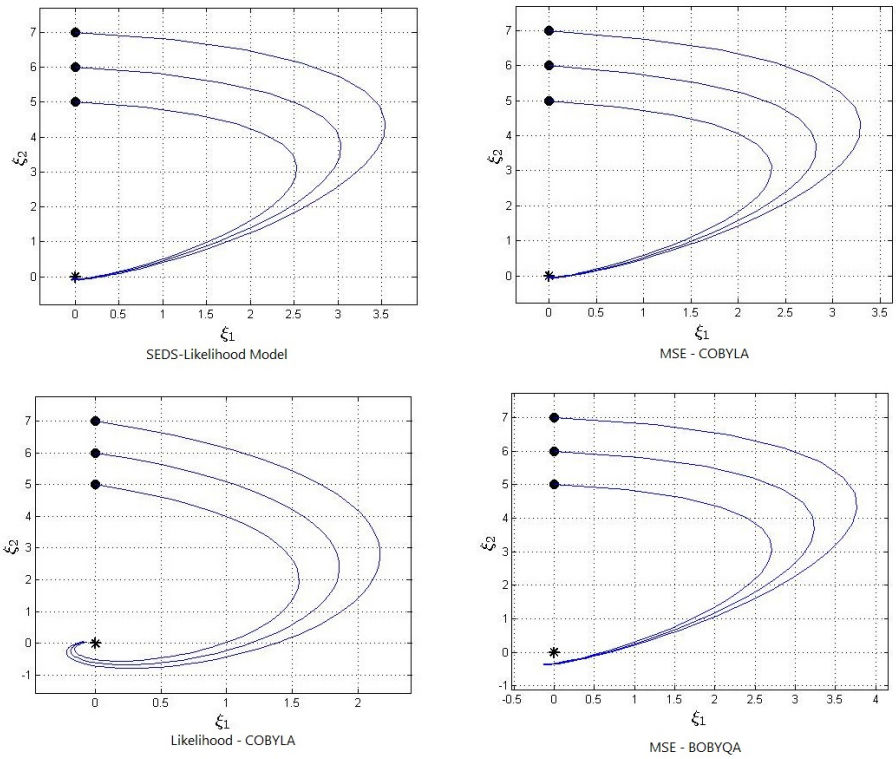

Figure 14: Simulation with three demos in 2D Cartesian space and using one Gaussian to learn the model.

8.3 Initial step

As mentioned earlier in the report, NLopt offers a feature to change the initial step used for the algorithm. Setting Likelihood-COBYLA as the default algorithm for these tests, we changed the initial step in an attempt to obtain better precision. Simulation results are shown in figure 15. However due to lack of a proper system to compare the results and due to erratic results we were unable to ascertain a generalized optimal initial step for all optimizations.

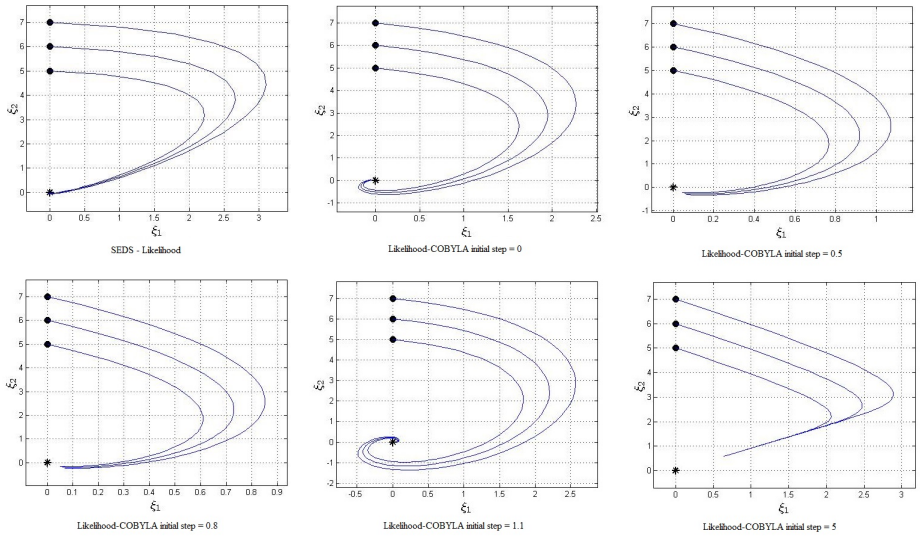

Figure 15: Simulation with three demos in 2D Cartesian space and using one Gaussian to learn the model. Increasing step size till a point improves the result and then deteriorates it. However this point is not general for different motion and we were unable to find a way to ascertain the most appropriate step size

9 Future Work

This project opens up many horizons on which work can be pursued. Here is a list of a few possible future applications of this project.

- Obstacle Avoidance as implemented in [7]. This system takes as input a Dynamical System that we have created in this project and modifies it such that the stability conditions of the original DS are preserved. It modulates the robot space to avoid paths through the obstacle.
- Presently, we modify the input data obtained from robot during the demonstrations manually to fit the input conditions required for training. For example, each demonstration should have the same number of points. However, this might cause us to loose data and is a tedious process that takes more time. One possible solution is to apply L1 interpolation onto each demonstration of the data set and then selecting as many data-points as required from the resulting curve. This will improve the division of points resulting in uniform velocity during the motion, as presently there is a larger density of points near the end of the motion and thus the motion slows down drastically as it approaches the end.
- Reachability : Another issue is to identify the limits of the working space of the robot one is experimenting on and how to reach the target when the path to reach from an initial point to a target position, both within the working space of

the robot, traverses through one or more points outside the working space of the robot(outside the reachability).

- The present model that we have trained cannot include loops in the motion learned. One solution discussed in [6] is to encode along with the state of the motion, also the velocity of the motion in a new state variable $\phi = \begin{bmatrix} \xi \\ \dot{\xi} \end{bmatrix}$. And the resulting 2nd order ODE, can be transformed into first order dynamics thus obtaining a model similar to that we have created in this project.

- Implementing this algorithm in joint-space. This algorithm has been developed to be used in any dimensional space, Cartesian space as well as Joint-space. Its implementation in joint space had produced favorable results in simulation but did not test well on the robot, mainly because the code that has been developed to interact with the robot(UR10-sending-reception) conveyed speed to the robot in Cartesian coordinates and was not developed to convey angular velocities as required here.

- An important step is development of a proper metric to compare results obtained from different optimizations. Lack of a proper metric to compare the results caused a problem of properly judging between different settings that are used to run each optimization. For example, different algorithms or different number of iterations or different initial step. The value of minf, the output value of the minimization which is the value of the cost function, did rarely have any relation to the results obtained from simulation and on the robot. The only way to distinguish between results was by observing the results on the simulator or on the robot and this took more effort to compare the solutions obtained. A better metric will also help evaluate change in results with different initial step and repeated optimization.

- Multi-Robot demonstration learning would be another interesting future work. This aspect requires cooperation of multiple robots and faces a major issue in action coordination. For example, for the pick and place example we visited earlier, for the model we have used to train in 3D Cartesian space, through inverse kinematics, the end-effector of the arm was free to rotate and thus coordination with another robot would require a slave arm which adapts to change in rotational axis of the master arm.

10 Conclusion

In this report we have presented a method based on Dynamical Systems approach to learn robot motion which is independent of time and globally asymptotically stable. The motion trained using this method is governed by a fixed law $\dot{\xi} = \hat{f}(\xi)$ and is stable to temporal and spatial perturbations. Our method of learning is part of a broad category of algorithms known as Programming by Demonstration(PbD). In PbD, models are trained based to demonstration data provided. We have used Gaussian Mixture Modeling(GMM) to model data. GMM is one of many statistical models used

for robot motion learning. We have also covered Lyapunov Stability Condition and used it in our model to ensure global asymptotic stability. In the results section we compared the performance of the different algorithms. MSE-COBYLA is the best performing algorithm, being more precise and reliable than Likelihood-COBYLA and MSE-BOBYQA. We also compared performance for Likelihood-COBYLA algorithm at different initial steps. Finally in the last section, we discussed future directions in which this project can be applied and need of a proper metric to compare PbD algorithms and solutions.

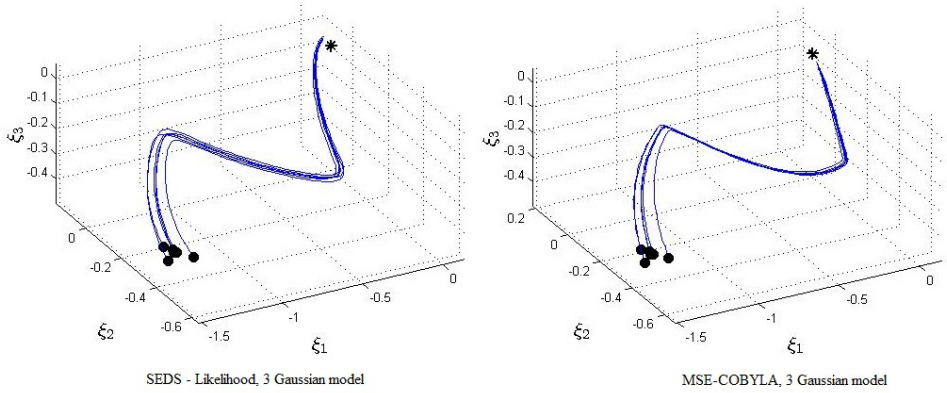

Figure 16: Simulation of 3D S-Shape using 3 Gaussian functions

References

[1] Nicholas I. M. Gould Andrew R. Conn and Philippe L. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds, 1991.

[2] Manuela Veloso Brett Browning Brenna D. Argall, Sonia Chernova. A survey of robot learning from demonstration, 2008.

[3] J. M. Martnez E. G. Birgin. Improving ultimate convergence of an augmented lagrangian method, 2008.

[4] S. Mohammad Khansari-Zadeh and Aude Billard. M: An iterative algorithm to learn stable non-linear dynamical systems with gaussian mixture models, 2010.

[5] S. Mohammad Khansari-Zadeh and Aude Billard. The derivatives of the seds optimization cost function and constraints with respect to its optimization parameters, 2011.

[6] S. Mohammad Khansari-Zadeh and Aude Billard. Learning stable non-linear dynamical systems with gaussian mixture models, 2011.

[7] S. Mohammad Khansari-Zadeh and Aude Billard. A dynamical system approach to realtime obstacle avoidanc, 2012.

[8] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation, 1994.

[9] M. J. D. Powell. The bobyqa algorithm for bound constrained optimization without derivatives, 2009.