Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea

How a Robot can Learn to Throw a Ball into a Moving Object through Visual Demonstrations

Advisor: Prof. Enrico Pagello
Co-Advisor: Dott. Ing. Stefano Michieletto

Student: Michael Masiero

21 09 2015
Anno Accademico 2014/2015

Abstract

The objective of this thesis is to extend a known probability model for Imitation Learning. Thanks to this model we can teach to a robot how to learn simple tasks, like throwing a ball into a basket, so that it can take into account initial informations related to the surroundings, i.e. the position of the basket can change in time. The objective is to train the probability model through human demonstrations. A RGB-D dataset has been collected by using low cost 3D cameras. This procedure is less accurate with respect to other techniques usually adopted in the field, such as the kinaesthetic motion of the robot. Nevertheless, the selected approach is closer to situations that could append in the everyday behaviours of a service robotic platform. This thesis derives from a previous work based on Robot Learning from Failure (RLfF), where one of the considered parameters was fixed, in particular the basket was positioned in a fixed spot. Once built the new extended probability model, we verify if the model adapts well to the introduced changes, facing the problems that can occur, thanks to a robot (a manipulator robot in our case) too.

Contents

Chapter 1

Introduction

This thesis of Autonomous Robotics aims to study and analyse an approach of Robot Learning from Failures (RLfF), applied to a new and extended situation compared to those already treated. The Robot Learning field, starting from demonstrations, tries to extract autonomously actions and behaviours so that a robot can learn them by itself. RLfF differs from Robot Learning from Demonstrations because it starts from incorrect demonstrations and it tries to distance from them (RLfD does the exact opposite, starting from correct demonstration, it tries to stick to them). Both techniques, pursuing the same objective, are included in the wider field of Machine Learning.

This is an evolution of a precedent work from Rizzi[1][1], that is based on two papers of Grollman and Billard [2] [3]. The previous work was focused on studying and implementing the Donut Mixture Model, the probability model introduced in the first paper of Grollman and Billard, verifying the robustness by teaching a simple action to a robot. Our work starts from here and extend his studies by applying the technique to a more complex situation.

In particular, during this thesis, we worked with a robotic arm (Comau Smart5 SiX[2] [4] [5] [6]) so it can learn how to throw a ball into a basket positioned in a variable spot. This means that, starting from a teaching phase, the robot learns how to change its launches (the positions and the velocities of his joints in time) accordantly to the position of the basket. This behaviour comes from a generated position-velocity trajectory made through the probability model where the information related to the basket position is an input variable. In Rizzi's work, the basket spot was fixed and the base

[1]Alberto Rizzi, Robot learning from human demonstrations: confronto fra DMM e GMM

[2]Comau Smart5 SiX, http://www.comau.com

teaching concept was applied to a humanoid robot (Aldebaran NAO[3]).

This evolution is interesting for three reasons:

- The described technique could not work with more than two degrees of freedom;
- Three degrees of freedom expand the perspective, introducing new limits to be overcome;
- The interaction with a complete different robot could change the point of view (introducing new issues too) for this robot learning technique.

Our work, like it is described in the following chapters, starts from the study and the re-elaboration of the Donut Mixture Model, applied to three degrees of freedom, using synthetic data. Then the research is applied to a real situation, where, starting from collected real data, we teach to the robot how to learn to throw the ball correctly in the basket. To do this, we also need to control the robot in velocity, implementing a plug-in to do so.

1.1 State of Art

One of the most complex cognitive process done from the human beings is the comprehension of what is happening into an environment. This process consists in the abstraction of the situations that are occurring in a specific moment, so the various parts involved in the action can be identified and the informations, useful to understand a new action or to improve the knowledge of a known one, can be extracted. The learning provides the skill to represent in an abstract manner the necessary informations for doing a task correctly, even in case of the appearance of different situations from the ones already seen.

For the robot programming, this is translated into the possibility of adapting a specific behaviour (a movement) to the available information on the real world (the arrangement of the objects in the surroundings). It is necessary to be able to change dynamically the actuation of the commands, maintaining intact a set of limitations that characterize the action in a significant manner.

The Robot Learning from Demonstration (RLfD) [7] [8], known as Imitation Learning too, it is the discipline that take up the actuation of this process through the use of demonstrations. The robot is able to learn a task through a set of demonstrations, usually done by humans. The Imitation

[3]Aldebaran Robotics,https://www.aldebaran.com

Learning is born as an instrument to help the automatization of the manual robot programming in the manufacturer field. The complexity of manual robot programming and the progressive growth of the scientific research in the fields of artificial intelligence and robotics have lead to the creation of new learning tools and techniques. They are more efficient, less complex and allow a better interaction between human and machine.

With this perspective the learning radically changes:

- the user is able to teach a new behaviour through initial demonstrations, i.e. moving directly the robot or controlling it by remote;
- the robot has to understand the task starting from the demonstration dataset to achieve the desired target;
- the learning is based on perception and action and their interaction;
- the relationship between the human and the machine is restricted to an initial phase of learning and building of the cognitive model and to a following phase where the learnt behaviour is checked.

Starting from this point, it is natural to question about what to imitate (learning a skill), how to imitate (encoding a skill), when to imitate and who to imitate.

The study and the analysis of these questions produced various common approaches, and different behaviour encoding, learning models and techniques. The techniques introduced in literature change accordantly to how the limitations are identified and the demonstration represented.
Schaal et al. [9] used a set of elementary movements, known as movement primitives, that applied serially and combined together formed a more complex movement [10] [11] [12]. The success of those techniques is bonded to the choice of a good set of movement primitives and to the "grammar" used to combine them [13] [14] [15]. Akgun et al. [16] extracted specific frames from records, that will form a set of key points of a sequence. This sequence is used to model correctly a given ability, obtained exploring in order the points of the created model, in a similar way for what happens with the cinematographic sequences. Calinon et al. [17] proposed a probabilistic approach to represent the information relying on the Hidden Markov Model (HMM) for coding the movements and on the Gaussian Mixture Regression (GMR) to generalize in a robust way the action. The use of a state model, like HMM, it is particularly adapt for generalizing in a robust manner the variability of a set of demonstrations keeping in consideration the execution time [18] [19] [20]. At the same time, this approach needs a major number

of states compared to other situations where the model was applied, so the use is more complex [21] [22] . Grollman et al. [2] [3], instead, proposed to extract the informations from wrong demonstrations of the task through the Donut Mixture Model (DMM). This model allows to explore the space formed by the set of recorded demonstrations and it adapts flexibly to the variability of the data.

A second fundamental aspect for the RLfD is how the demonstrations are recorded. A first distinction is about the availability of the data: having the whole dataset from the start (batch learning) in opposition with an interactive approach (iterative learning). A more fundamental difference is related to the techniques used for collecting data. In kinaesthetic teaching [23] [24] [25], a human teacher guides physically the robot in the entire execution of a specific task, moving each joint along the desired trajectories. A similar technique is based on the execution of the movements through different simulated sessions of the task until there is the certainty that there are not dangers in the execution of the real task. The use of IMU devices applied to the joints of the body related to the movement [26] permits to collect information on position, velocity and acceleration of the person during the task execution, so we can apply the recorded data to a robot with similar characteristic, like a humanoid one. Another analogue technique provides to apply visual markers on the subject performing the action to record the demonstrations through Motion Capture techniques so that the person movement can be recognized [27] [28]. Recently force sensors are widely used on the robots and it is now possible to utilize atypical devices to collect information about necessary forces and movements to complete a task [29]. All these techniques

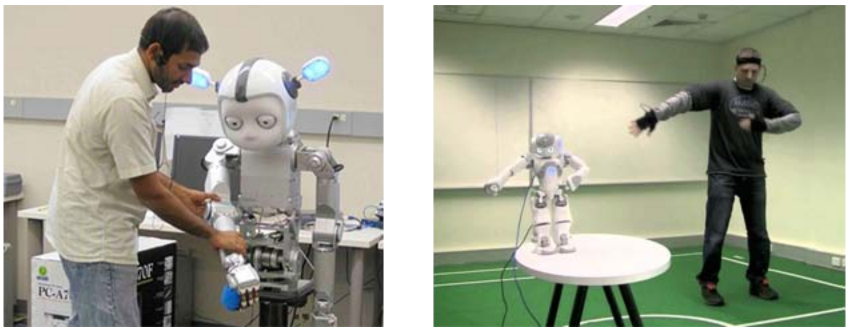


Figure 1.1: Example of teaching techniques. Kinaesthetic teaching on the left, Full Motion Capture on the right.

have the advantage to be very accurate and controlled, so the use of data for learning applications is simple and reliable. To the other side this is the biggest limitation of RLfD, because all this expensive accurate devices are not always available. In fact, they can only be bought in large companies or in some research lab, into economic robots available to the masses. The use of cheap widespread devices, like cameras or 3D sensors could lead to a lot of advantages to RLfD, and some preliminary studies [1] show that exist a wide margin to improve the state of the art under this aspect [30].

This thesis is developed starting from this last statements, choosing to use a low cost camera to collect data. It is certainly a less accurate way to operate, but it is closer to real situations for service robotics.

1.1.1 Reasons behind the choice of RLfF on RLfD

Like we said before, the RLfD principle consists on starting from correct demonstrations and, sticking to them, finding the overall right behaviour. It is fundamental that during this process only correct demonstrations are used, because wrong one could lead to wrong results. At first impact this process may seem very simple, but it carries with it some disadvantages:

- the time needed for the user to learn a task and to teach it to the robot, is variable from user to user and it is directly proportional to the difficulty of the task itself;
- the user need to know perfectly the task that he want to teach. This is not crucial for simple actions, but it is necessary with more complex ones, i.e. for humanoid walking or pick and place situations. In these cases common users are not sufficient and professional ones are required;
- some tasks could be out of range of human skills for their complexity.

For softening these limitations, we can add to the model incorrect demonstrations. This can be done in an improvement phase for the model, to adjust the learning policy. From this point of view, the robot can learn from his errors, but always after that the main model based on correct demonstrations is built.
If we start from incorrect demonstrations, we can take in account the human nature of the users and the possibility to do some mistakes learning phase. Within his work, Grollman has demonstrated that a lot of information can be collected also from the failure attempts. Failures are examples of what the robot does not have to do, they give us information related to the interpretation from the user of how the task has to be done and they constitute an exploration space where the right solution can be found.

1.1.2 Robot Learning from Failure

To teach to the robot what is right and what is wrong to accomplish its task, an autonomous system that learns from failures can be created. The approach is related to Robot Learning from Demonstration (RLfD) and it perfectly suits this situation.

The general steps of this technique are the following:

1. Collect a first set of demonstrations;
2. Build the model;
3. Reproduct the behaviour through the model;
4. Update the model.

The main differences are found on the choice of the model and how it is used. It is fundamental to keep low the probability of following wrong behaviours and to have good exploration skills in the space generated from the various demonstrations.

Adapting to these differences modify the general steps as follows (illustrated in Figure 1.2):

1. Collect a first set of *failed* demonstrations;
2. Build the model;
3. *Generate a new attempt that explores around the known demonstrations*;
4. *Execute the new generated attempt* and update the model;

The last two steps are repeated until we reach a success. In a first moment we followed the first three phases, starting from a synthetic data set through a simulation of the model, then we collected real data and completed each steps of the algorithm.

On practical level, independently from the starting dataset (synthetic or real one), we followed these steps for generating a new attempt:

1. Read the formatted data from file;
2. Generate a Gaussian Mixture Model through an initial phase of KMeans followed from Expectation Maximization;
3. Estimate the exploration factor and create the corresponding Donut Mixture Model from a point of the first mixture;

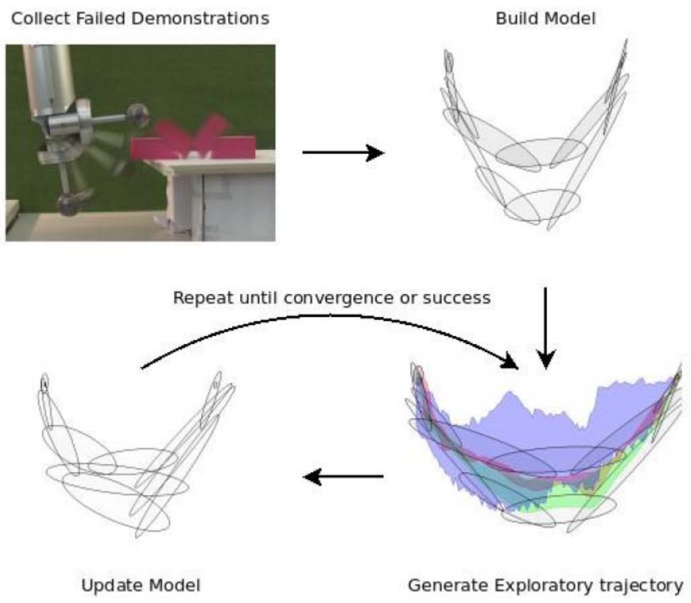

Collect Failed Demonstrations
Build Model
Repeat until convergence or success
Update Model
Generate Exploratory trajectory

Figure 1.2: Steps for the learning from failures

4. Search for a point that will be part of the new attempt from a tuned slice of the generated DMM.

Repeating the last two steps for every X point of the model and putting together the results will lead to a new way of exploring the data (the trajectory that will be sent to the robot).

Chapter 2

Building the probability model

This chapter is focused on describing the various steps necessary for the building of the probability model, core of the learning system. To describe what has been done, each of the following sections concentrates on a specific aspect.

Our work is focused on extending the Donut Mixture Model and the related formulas from two to three degrees of freedom. To be more specific, we added the information related to the position of the basket as a third degree of freedom.

In particular they describe:

- the collection of the initial dataset;
- the Gaussian Mixture Model and its role when creating the DMM;
- the Donut Mixture Model, why it has been chosen for our purposes and how we extracted the exploration trajectory.

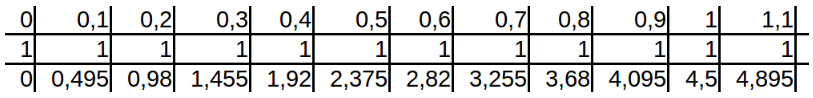
Before proceeding with the description of the model, it is important to underline that, although the we are working with a 3-Dimensional model, it is not always significant to show the 3D representation of what we are talking about, in favour of the associated 2D version with one degree of freedom fixed. This second way of representation it is more intuitive because allows to focus on the current steps without the presence of unnecessary information. In addiction, if we want to represent the model with the associated probability function, we will have to cope with a graphic in 4 dimensions.

We chose to fix the basket position in favour of the other two variables (joint position and joint velocity). In this way we will look at the model from a more interesting perspective than if we chose the basket position as one of the two not fixed variable. Although we used discrete sampled data, the variability of the basket position is a lot more limited than other variables (there are holes between different values) and it is not significant to slice the dataset in other directions because the trajectories associated to the action grow in the position-time or position-velocity domains. In addiction the output is retrieved fixing the basket position at the beginning.

2.1 Dataset

As we said before, the main difference with respect to the precedent works is the additional information of the position of the basket next to the pairs position-velocity associated to the movement. Starting from these data, we aim to find the velocity a joint of the robot has to assume for each angle value crossed during the throwing, related to a particular basket position.

Formally, the dataset $\Xi = \{\tau_s\}_{s=1}^{S} = \{\xi_t,\ z,\ \dot{\xi}_t\}_{n=1}^{N}$ is composed from a collection of trajectories τ_s , formed by triplets, composed by respectively ξ_t position (angle) of the joint, z position of the basket and $\dot{\xi}_t$ velocity of the joint. N $= \sum_{s=1}^{S} T_s$ is the number of data points in the data set.
As we can see in Figure 2.1 and as we just described, we have three rows,



0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1	1,1
1	1	1	1	1	1	1	1	1	1	1	1
0	0,495	0,98	1,455	1,92	2,375	2,82	3,255	3,68	4,095	4,5	4,895

Figure 2.1: Example of a portion of a dataset

each one associated to one degree of freedom. The first one is the angle, expressed in radians. The second one is the basket position, the distance of the basket from the base of the robot, expressed in meters. The last one is the velocity, expressed in radians per second.

In our case we use only one joint of the robot, so we do not need to do distinctions from this point of view. In the end the dataset contains the informations related to various throws to various distance of the basket.

2.2 Gaussian Mixture Model

The Gaussian Mixture Model $\theta = \{K, \{\rho^k, \mu^k, \Sigma^k\}_{k=1}^{K}\}$ is the first used and it is formed from a set of Gaussian component that will be positioned across the data. Respectively each parameters stands for:

- K is the number of Gaussian Component;

- ρ stands for prior, the weight associated to each component (real positive, $\sum_{k=1}^{K} \rho^k = 1$);
- μ is the mean of each component (three dimensional vector);
- Σ is the covariance matrix associated to each component (3x3 positive matrix).

The Gaussian Mixture Model is chosen because we can considerate the relationship between each data point $(\xi, z, \dot{\xi})$ like a non-linear function $\dot{\xi} = f_\theta(\xi, z)$ represented from the model itself. In particular, we can write the associated probability function as follows

$$P_{GMM}(\xi, z, \dot{\xi}|\theta) = \sum_{k=1}^{K} \rho^k N(\xi, z, \dot{\xi}|\mu^k, \Sigma^k) \tag{2.1}$$

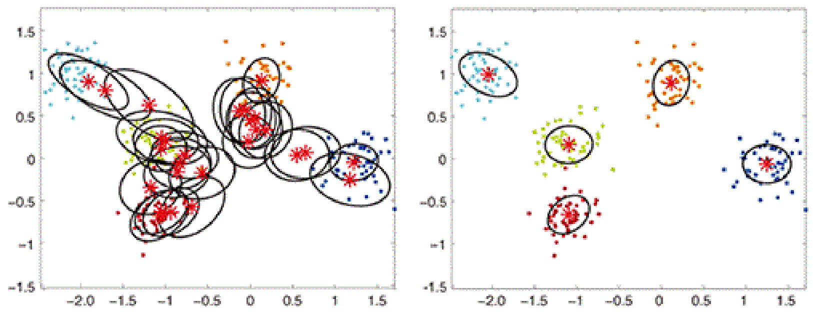


Figure 2.2: Example of Gaussian Mixture Model. The Gaussian Component are placed around the datapoints of the dataset.

In order to correctly build the model, we need to choose the best number of components to estimate each component parameters. On practical side, μ is the center of a Gaussian component and Σ is his extension.

2.2.1 K-means and Extimation Maximization

To obtain the model, we use two algorithms for the positioning of the Gaussian components and the computation of their parameters.

The first one is the K-means algorithm, through which we can do a first positioning of the components, assigning an initial value to each parameters of the model. Thanks to this phase the accuracy of the next step is improved. The initial clusterization can be made in a lot of different ways: the default one is based on random positioning that put all around the area where the data could be sparse the various components. For our purpose, because the trajectories concentrate data points along some lines and not uniformly in all the area, it is preferable to choose a positioning that fits better called PP-CENTERS. In this way we avoid the positioning of Gaussian components in an area where there is no data points.

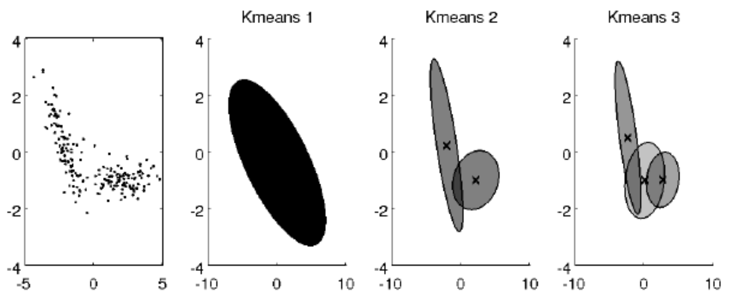


Figure 2.3: K-means algorithm for different values of K.

Next to K-means, we use a second algorithm called EM (Expectation Maximization) that allow to determine the best values for the model between all the possibility (the model that better fits the initial dataset). EM can be divided in two parts: the former (E-step) verify through the calculation of probability if the work of K-means was done well, then with the latter (M-step) the parameters of the model are recalculated using the information from the first step.

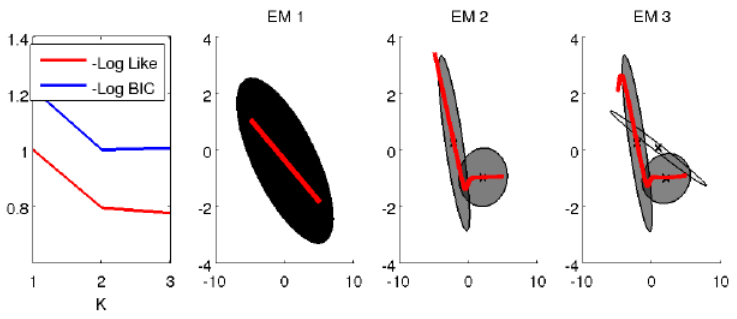


Figure 2.4: EM algorithm for different values of K.

After these two algorithms we have the GMM fully initialized. It is important to point out that this phase can be done with different numbers of Gaussian components. Usually more components means more accuracy for the model, but there is a limit over that we could occur in over-fitting. In addition, as we can see in the Application chapter, there is a chance that if the model has too many components, they could interact too much by themselves.

To estimate the best numbers of components for a dataset, we can use the BIC algorithm (Bayesian Information Criterion), based on

$$BIC = -2 * ln(L) + K * ln(n) \tag{2.2}$$

a function dependent from log-likelihood L and the number K of chosen components (n is the number of data points). With this criterion we need to find the minimum value of the function, associated to a K value. It is important not to use an high value of K because it may induce to a situation of over-fitting.

Built this first model, it is possible to calculate the velocity associated to each position of the joint and of the basket, conditioning respect of them. The model is modified as follows:

$$P_{GMM}(\dot{\xi}|\xi, z, \theta) = \sum_{k=1}^{K} \tilde{\rho}^k(\xi, z, \theta) N(\dot{\xi}|\tilde{\mu}^k(\xi, z, \theta), \tilde{\Sigma}^k(\theta)) \tag{2.3}$$

$$\tilde{\mu}^k(\xi, z, \theta) = \mu^k_{\dot{\xi}} + \Sigma^k_{\dot{\xi}(\xi,z)} \Sigma^{k-1}_{(\xi,z)(\xi,z)}((\xi, z) - \mu^k_{(\xi,z)}) \tag{2.4}$$

$$\tilde{\Sigma}^k(\theta) = \Sigma^k_{\dot{\xi}\dot{\xi}} - \Sigma^k_{\dot{\xi}(\xi,z)} \Sigma^{k-1}_{(\xi,z)(\xi,z)} \Sigma^k_{(\xi,z)\dot{\xi}} \tag{2.5}$$

$$\tilde{\rho}^k(\xi, z, \theta) = \frac{\rho^k N(\xi, z; \mu^k_{(\xi,z)}, \Sigma^k_{(\xi,z)(\xi,z)}}{\sum_{k=1}^{K} \rho^k N(\xi, z; \mu^k_{(\xi,z)}, \Sigma^k_{(\xi,z)(\xi,z)})} \tag{2.6}$$

This new model is a Gaussian Mixture Model too, with different parameters from the first one $\{\tilde{\mu}^k, \tilde{\Sigma}^k, \tilde{\rho}^k\}$ but derived from them and joint position and basket position dependant. It is possible to use this model in case of learning from success, because it guaranteed a smooth movement without a lot of oscillations.

2.3 Donut Mixture Model

The model based only on simple Gaussian components, as we will see soon, does not fit with our objective, because it does not generate new exploratory trajectories.

We need to introduce a new model, called Donut Mixture Model, described by the difference between two Normal distributions

$$D(\vec{x}|\mu_\alpha, \mu_\beta, \Sigma_\alpha, \Sigma_\beta, \gamma) = \gamma N(\vec{x}|\mu_\alpha, \Sigma_\alpha) - (\gamma - 1) N(\vec{x}|\mu_\beta, \Sigma_\beta) \tag{2.7}$$

With $\overrightarrow{x}$ we refer to the triplet formed by the three variables of the model. For our purpose we set the value of γ to 2 ($\gamma > 1$) and we generate a Donut component from the difference of the same Gaussian one ($\mu_a = \mu_b$) (see[1] for further details).
Thanks to these assumptions we can introduce r$_\alpha$ and r$_\beta$ (obtaining $\Sigma_\alpha = \frac{1}{r_\alpha^2}\Sigma$ and $\Sigma_\beta = \frac{1}{r_\beta^2}\Sigma$) to parametrize the model as follows

$$D(\vec{x}|\mu, \Sigma, r_\alpha, r_\beta, \gamma) = \gamma N(\vec{x}|\mu, \Sigma/r_\alpha^2) - (\gamma - 1)N(\vec{x}|\mu, \Sigma/r_\beta^2) \qquad (2.8)$$

Accordantly to this, we can swap each Gaussian component of the conditioned distribution with a Donut function. The probability of the Donut Mixture Model becomes

$$P_{DMM}(\dot{\xi}|\xi, z) = \sum_{k=1}^{K} \tilde{\rho}^k D(\dot{\xi}|\tilde{\mu}^k, \tilde{\Sigma}^k, \varepsilon) \qquad (2.9)$$

where ε is the exploration factor ($0 < \varepsilon < 1$), related to the variance of the model.

There is a link between the variance of the initial dataset and the exploration of the space joint position-basket position-joint velocity

- $\varepsilon = 1$ correspond to the maximum exploration for areas with high variability
- $\varepsilon = 0$ correspond to the minimum exploration for areas with low variability (the behaviour is similar to the Gaussian Mixture Model)

In relation of the input data, the exploration factor ε can be estimated so the model can fit better.

To estimated it, taking into account the conditioning from the joint position and the position of the basket, we can follow the next formulas:

$$\varepsilon = 1 - \frac{1}{1 + ||\tilde{V}[\dot{\xi}|\xi, z, \theta]||} \qquad (2.10)$$

$$\tilde{V}[\dot{\xi}|\xi, z, \theta] = -\tilde{E}[\dot{\xi}|\xi, z, \theta]\tilde{E}[\dot{\xi}|\xi, z, \theta]^T + \sum_k \rho_k(\mu_k \mu_k^T + \Sigma_k) \qquad (2.11)$$

If we want to compare the two introduced probability models, as we see in Figure 2.5, we can put along together a Gaussian component and a family of Donut components, generated changing the value of the exploration factor ε.

[1]Alberto Rizzi, Robot learning from human demonstrations

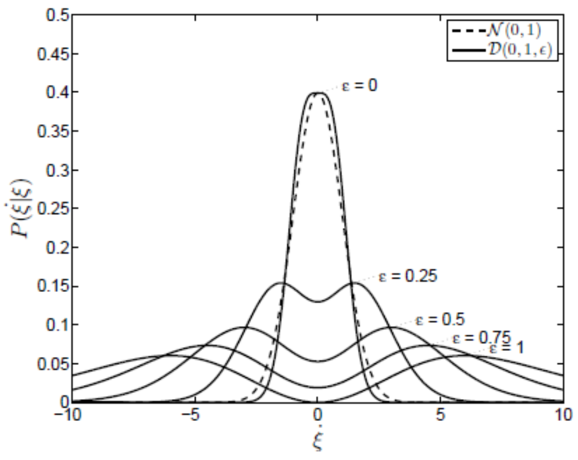


Figure 2.5: Comparison of a Gaussian component with a family of Donut components generated from various value of the exploration factor ε.

The choice of the model fell on DMM because our objective is to generate exploratory trajectories around the initial data (dotted lines in Figure 2.6). When fit to failure data, the mean (solid line in Figure 2.6) may no longer be an appropriate response (output of GMM).

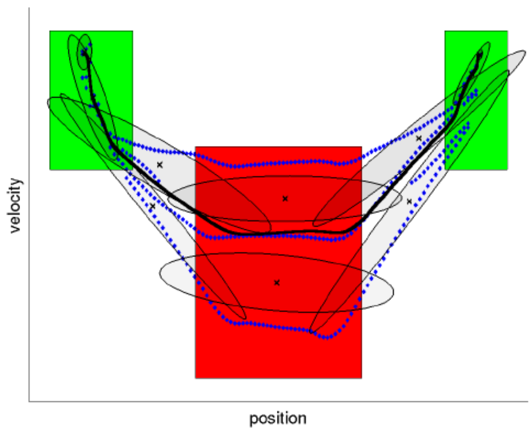


Figure 2.6: We want to mimic the human in areas of high confidence (green) and explore in areas of low confidence(red).

2.4 Optimization

Built the probability model, we need to extract the exploration trajectory that represents the learnt behaviour. The objective of this step is to find the set of points, for each value of joint position, corresponding to the maximum of the PdF (the basket position was fixed from the start). The points set that we obtain from the reiteration of this phase forms the trajectory (new exploration line) that will be send to the robot.

In input to this phase we use a slice generated from the conditioning of the Donut Mixture Model with the fixed basket position and the position of the joint (variable but fixed in each iteration). To find each maximum value we use an optimization algorithm that has as core another algorithm called BGFS2 (efficient version of Broyden-Fletcher-Goldfard-Shannon (BFGS) algorithm). There are a lot of techniques to do this job, but this fits better with our data[2] .

We start to optimize from the mean value because if we start from zero we will arrive (in the major cases) to a local stationary point (that is not interesting). In addiction, usually if we start from the tails of the distribution that are very flat, we incur in a low gradient that block our research. Anyway, we set up a control to check around the initial position, so we do not get stuck onto a wrong value. The optimization algorithm will stop when the difference between successive gradient values are under a tolerance.

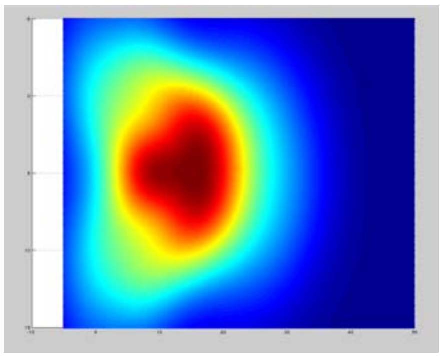


Figure 2.7: View of the PdF from above. We are interest in points with the highest PdF (hotter colors) for each value of joint position.

[2]Alberto Rizzi, Robot learning from human demonstrations

Chapter 3

Communication with the robot

This chapter is focused on describing how we interact with the robot, from how we communicate to how the informations received are processed.
The communication can be summarized as we see in Figure3.1 .

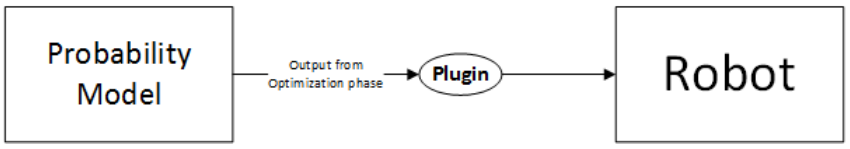


Figure 3.1: Communication scheme

3.1 Reaching the robot

The communication with the simulated robot arm is based on a plug-in developed by IAS Lab. This plug-in allows to visualize the robot in a simulated test space (Gazebo or RViZ) or to communicate directly with the real one.

In ROS, the communication can be done in three ways: by topics, by services or through actions.

With the first method (topic) the client writes specific messages on a topic where the server is listening (subscribed). This method should be used for continuous data streams (like sensor data or robot state). In our case the server is represented by the robot arm. For every new message read on server side (published from the client), the plug-in translates it to the robot so it can move. The subscribed callbacks in the server can receive data once it is available because is the client that decide when data is sent.

The second method (service) is based on the paradigm request/reply. A client can request a service and then wait the reply. Services are defined by *srv* files, that are files where it is wrote the message format. Services should be used for remote procedure calls that terminate quickly (like querying the state of a node). It is not meant to be used for longer running process. This method does not fit well our case because it is based on blocking calls and it does not make sense for the server to be stuck on a request sent from the client when some other request could arrive.

With the third method (action) the client sends an action (basically a request message, like it happen with services) to the server that can capture it using callbacks. The difference is that sometimes, if the service takes too long to execute, the user might want to cancel it. So with this method the service can be pre-empted. This last method permits a better control of the robot because can provide feedback during execution.

More details regarding communication in ROS can be found at http://wiki.ros.org/ROS/Patterns/Communication.

At the beginning the topic method was the only way to communicate with the Comau, and with only a particular message format (only JointPose messages). With this thesis it was implemented a new mode to communicate with the robot arm, using the action method and choosing a message format through which we can have a direct control of the velocity of the joint.

3.1.1 Actions

The communication by actions can be represented with the scheme of Figure 3.2.

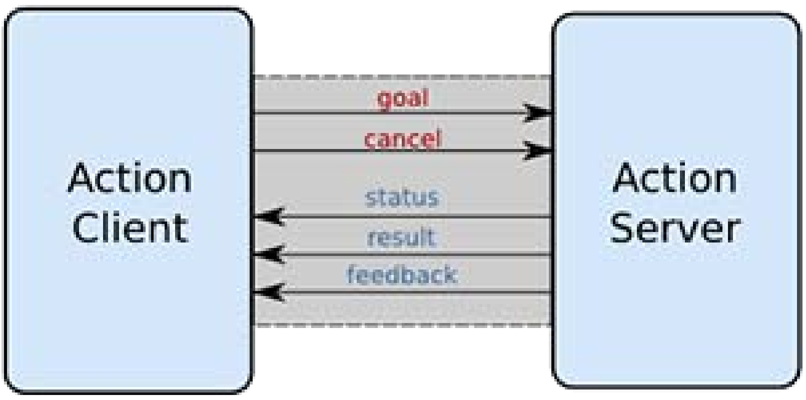


Figure 3.2: Scheme of the action interface

As we said before, the client can send either a goal message or a cancel one, to interrupt what the server is doing. Each time a message is received by the robot (server side), the informations contained were extracted and elaborated by the plug-in and sent to the robot controllers, so that the robot (real or simulated) could move the chosen joint following the computed trajectory, formed by point with informations of the joint position and the computed velocity, related to the position of the basket. In addition, the server communicate through the result if the message was received correctly. In a second time, if it is succeeding to get to the goal, it writes a feedback with its actual state.

3.1.2 JointTrajectory messages

For a direct control of the velocity of the joints of the robot, the choice fell on an approach based on Joint Trajectory Action.

A Joint Trajectory message has this format:

```
Header header
string[] joint_names
JointTrajectoryPoint[] points
```

It is described from a header, from a string vector with the names of the joints inside and a vector of points (each one related to a joint and positioned in the same order like in the joint names vector). Each point has this format:

```
float64[] positions
float64[] velocities
float64[] accelerations
float64[] effort
duration time_from_start
```

As we see a JointTrajectoryPoint is described from a set of one or more positions, with velocities, accelerations and/or effort associated. To a trajectory can be added the information related to the execution time. The dimension is the same for each vector.

3.1.3 Sending an action

For our purpose and for what is elaborated through the estimation maximization and optimization phase, we need to send to the robot only the information related to the position and the velocity of the joint. The other parameters of a joint trajectory message remain unused. For a single message sent to the server we fill the joint names string vector with the six name related to each joint of the robot and the vector points. Each point of the trajectory has only one position and one velocity. For our purpose the information related to unused joints remains constant.

3.1.4 Receiving an action and getting to the goal

On server side, when a message is received, it is not directly sent to the robot, but it is first processed, so the evolution of the trajectory of the joint becomes smooth and controlled step by step.
The first thing that the plug-in does is to extract the informations just received through the message. After the extraction, each point of the trajectory sent is re-published following a particular rate, so the informations can be processed in order without any loss. Thanks to the callback that is listening, the information is caught, elaborated and sent to the robot.
The elaboration consist in a sort of interpolation between a point and its predecessor, so the robot can move smoothly. As we will see in the next chapter, the used joint moves in accord to every couple of joint position - joint velocity. At the end of the trajectory, the joint assume a velocity proximal to zero.

Chapter 4

Application

This chapter is focused on display the results obtained with our implementation. Step by step we will touch each single passage described before.

4.1 Preliminary study

The first section is dedicated to the preliminary studies done before the work with the real data. We started using synthetic demonstrations to verify the correctness of the extended model.

4.1.1 Input data

We decided to gave in input to the model 11 different trajectories. Each one is composed by 50 points and derives from the base one

$$y = \frac{-x^2 + 10x}{2} \tag{4.1}$$

by a different multiplication factor. Each trajectory has associated a constant value of the position of the basket. For the first one z is equal to 1u (symbolic unit). For the others we increment the value of z of one unit, so every trajectory has a position of the basket associated of one units more than the units of the trajectory directly under it.

4.1.2 First positioning of the Gaussian Components

After various tests related to the input data, we estimated by BIC that 2 and 3 is good numbers of components for building the model.
Applying the K-Means and Estimation-Maximization algorithms we obtain

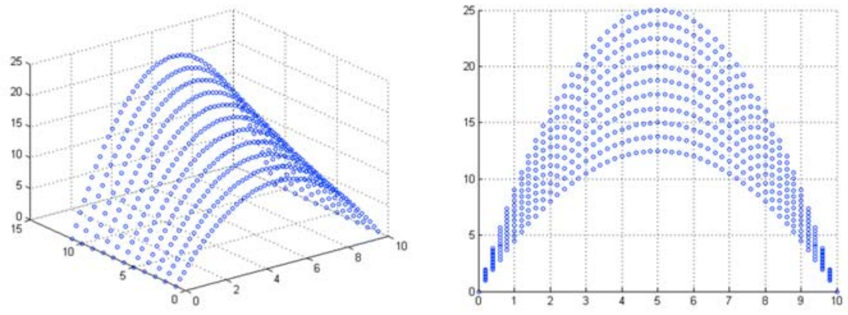


Figure 4.1: Initial synthetic dataset formed by 11 trajectories

a distribution of the 2 and 3 Gaussian Components like in Figure **??** (from joint position - joint velocity perspective)

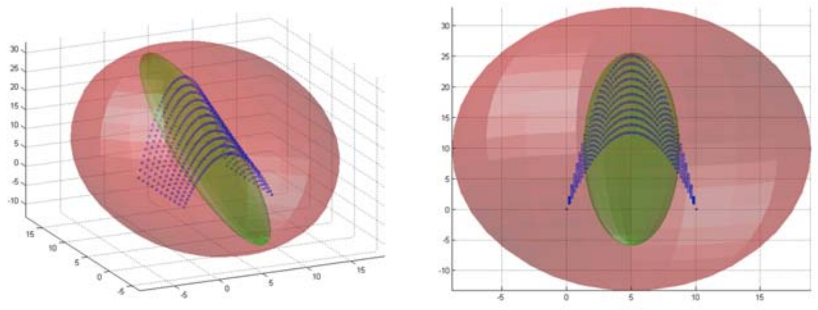


Figure 4.2: Distribution of two Gaussian components around initial data.

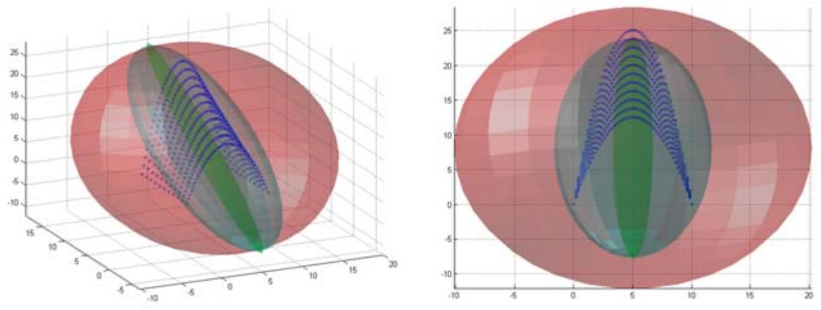


Figure 4.3: Distribution of three Gaussian components around initial data.

4.1.3 Optimization phase

On this phase, chosen a value of the position of the basket, it is iterated the same optimization algorithm for every value of ξ that the joint can assume. Before every new iteration of optimization, the first GMM calculated is adapted to a DMM after the estimation of the exploration factor ε.
In Figure 4.4 we can see an extended photograph of the PdF associated to our model. Each iteration scans a line (represented by red) along which is applied the optimization algorithm.

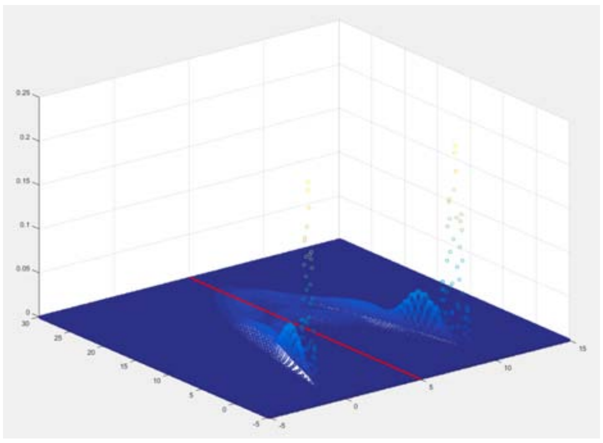


Figure 4.4: Search the maximum value of pdf for ξ=5 and z=8.5

4.1.3.1 Output

The entire algorithm, at his conclusion, create in output a file with the points of the new experimental trajectory and the associated PdF found.

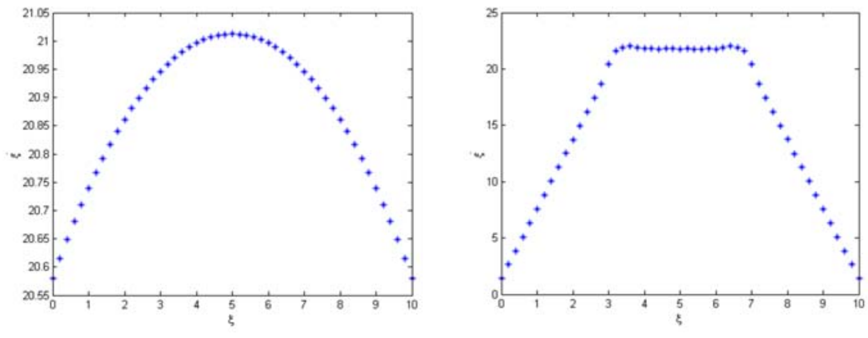


Figure 4.5: Output from the optimization phase, with z = 8.5 units

The associated output, starting from using 2 or 3 Gaussian components, it is a good result but we obtain something better. If we add some other

Gaussian components, although the BIC estimator increases, we can obtain a smoother and more accurate result, as we see in Figure 4.6. This is possible because in this moment we are working with synthetic data, but probably we cannot take actions lightly in the same way with real data.

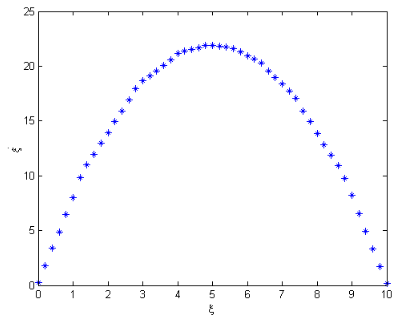


Figure 4.6: Output from the optimization phase, with $z = 8.5$ units

To figure out if the model created behave well, we compared more output, each one derived from a different starting position of the basket. As we see in Figure 4.7 everything go in accord with what we expected. Each generated trajectory grows if the distance of the hypothetical basket increases. It is important to underline that the model behave very well in situation not included in the initial data, as we see in the third graph (in the initial data the maximum distance for the basket is 11 units). At the same time we are aware that this is an ideal situation, because each trajectory is equidistant from the others and has the same rescaled shape. With the real data it is important to keep in mind this observations if some problems come out.

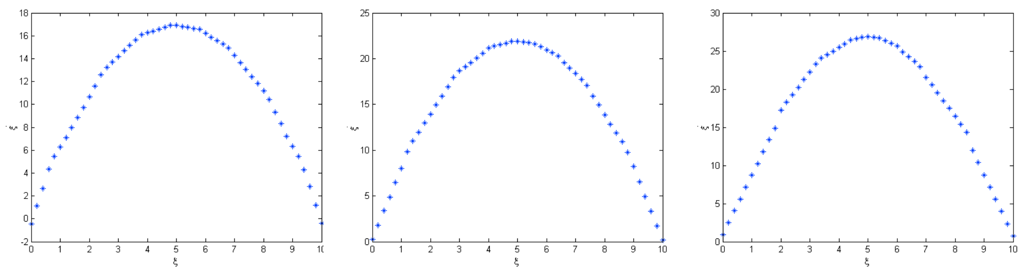


Figure 4.7: Various output related a different position of the basket, respectively 4.5 , 8.5 and 12.5 units.

For every trajectory generated, it is possible to see, if the plugin is active, how the robot moves respecting the value of position (angle assumed) and velocity sent.

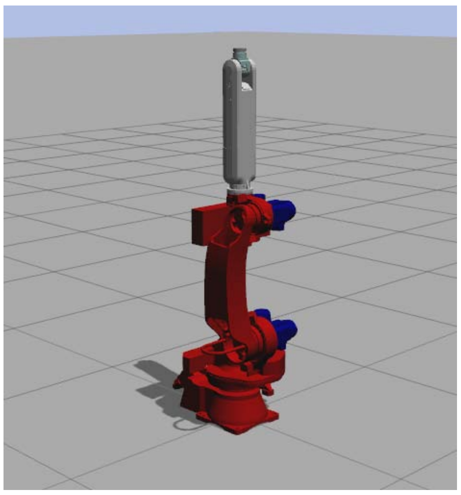


Figure 4.8: Gazebo plugin with the Comau model

For this practical trial we chose the joint number 2, but the model is free to be applied to each joints, always respecting his limits. In addiction, in the real situation we have to give to the robot a starting positioning suitable for throwing.

4.2 Collecting real demonstrations

From the results of the precedent chapter, we can now plan how to do the recording.

Keeping in mind that we need variability in terms of basket positioning and at least ten throws for a single distance for our dataset, we setted up an environment like in Figure 4.9. The red circle represent the throwing spot where the user throws the balls, the green circles are the baskets and the black bars are Microsoft Kinect1.

As we can see in Figure 4.10, we chose to place 12 baskets around a throwing spot. Each basket is on a different line (0, 30, 60 or 90 degrees) and has a different height. These other variables are added for future works, for our purpose we considered only the distance in the throwing.
The maximum distance is chosen following the limits imposed by the cameras. The recording web is setted up by four Kinect1 connected all together in a private network. Before the recording of each throws, thanks to OpenPTrack[1], each camera knows where the others are so, if necessary, the data collected can be unified.

[1]http://openptrack.org/

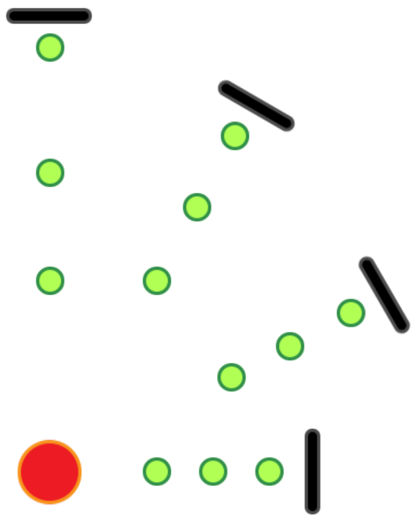


Figure 4.9: Placement of baskets and cameras in the recording environment

The distance of the twelve baskets are 1.0m, 1.5m, 2.0m, 2.25m, 2.5m, 2.75m, 3.0m, 3.25m, 3.5m and 4m. There are three baskets positioned at a distance of 2 meters and with different heights, but, like we said before, it is recorded for future applications.
We asked to 10 users to do four throws for each basket, for a total of 48 throws per user. The only restriction imposed is how the throws need to be done, that is, like in Figure 4.11 from above the shoulder and with the right arm.

To record the movement we choose to listen to three Kinect topics:

- /camera/rgb/camera_info
- /camera/rgb/image_color/compressed
- /tf

The first one gives informations related to the recordings but it is not fundamental. Recording the second one provides compressed image from the video stream that help us in the next elaboration to divide each throws. The last one carries the informations strictly related to the throws, in particular about every joint of the user. For this purpose we need to start a tracker before the start of the recording, that maps the joint of the user from the informations received from the Kinect stream. In this case it is used Nite2Tf, a tool developed by IAS-Lab.

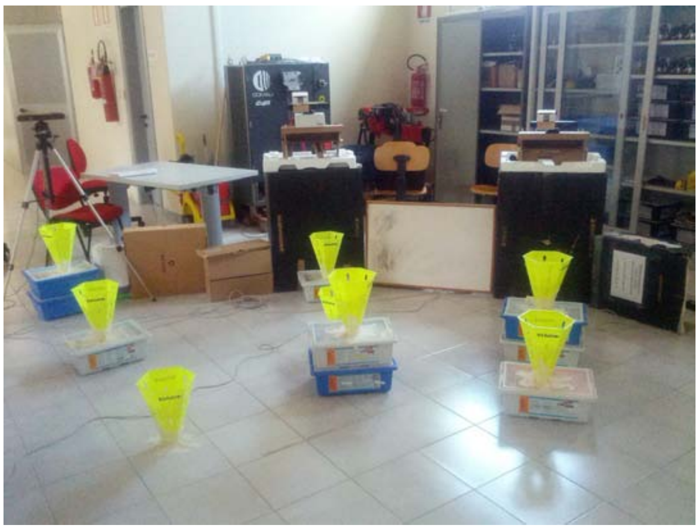


Figure 4.10: Photo of the recording environment

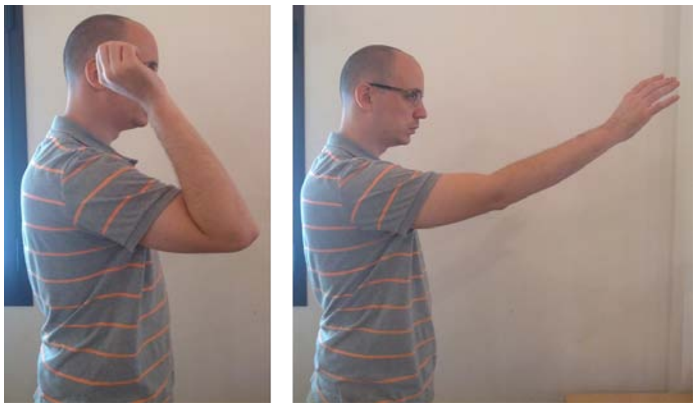


Figure 4.11: Correct movement for the throw

Extracting and elaborating the tf related to the right arm joints allows us to reconstruct the interested movement. In the end it is important to retrieve informations about the orientation of the shoulder, of the elbow and of the hand. As we see in Figure 4.14, from the pairs Shoulder - Elbow and Elbow - Hand we can define two vectors: calculating the rotation in a instant of the first one on the second one gives us the measure of the angle at that time. Thanks to the timestamp associated to each tf message, reiterating this computation for every moment of the throws, provide us of angle-time pairs that forms the entire movement. From the position and the time associated we can now extract the velocity of the arm at that time too.

We recorded all the throws per each line in one bag file. To help the successive elaboration it is simpler to subdivide in smaller files each throw.

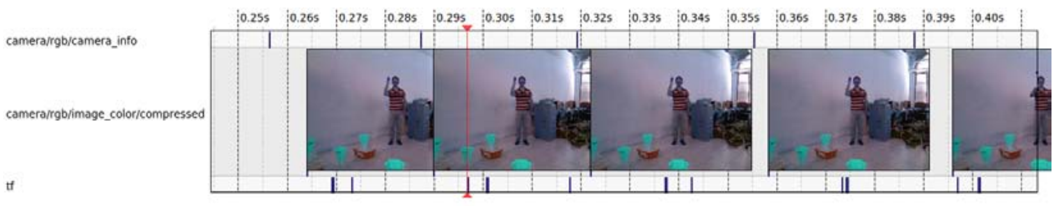


Figure 4.12: Visualization inside the tool rqt_bag of a single throw

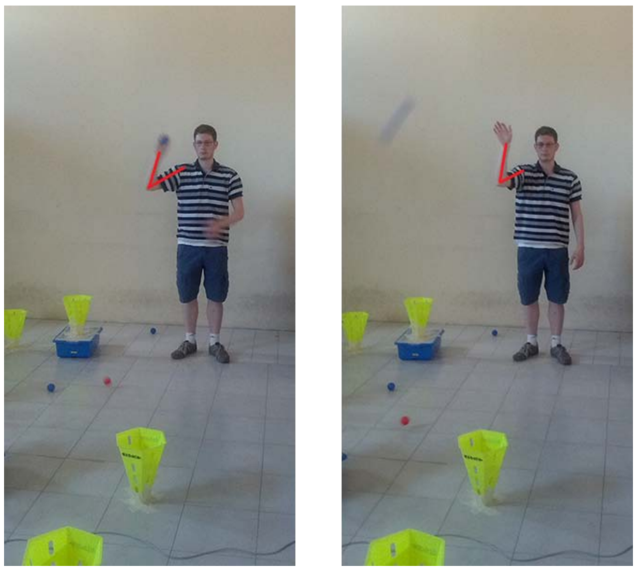


Figure 4.13: Change of the angle during the throw

It is important to define what we considered as a throw. We can choose to consider the full movement (from the starter position to the full extension of the arm) or cut the bag file (the format used in ROS to record data from topics) in correspondence of when the ball is left from the hand. For our purpose the second method it is better, but for a more complete initial dataset we choose the first one and delayed the cutting of the trajectories to a second moment.

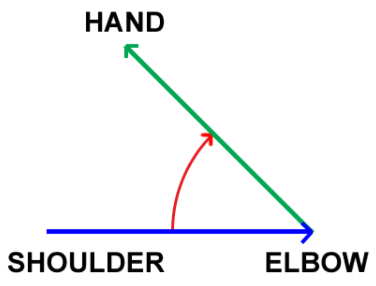


Figure 4.14: Rotation of one vector on the other for angle estimation

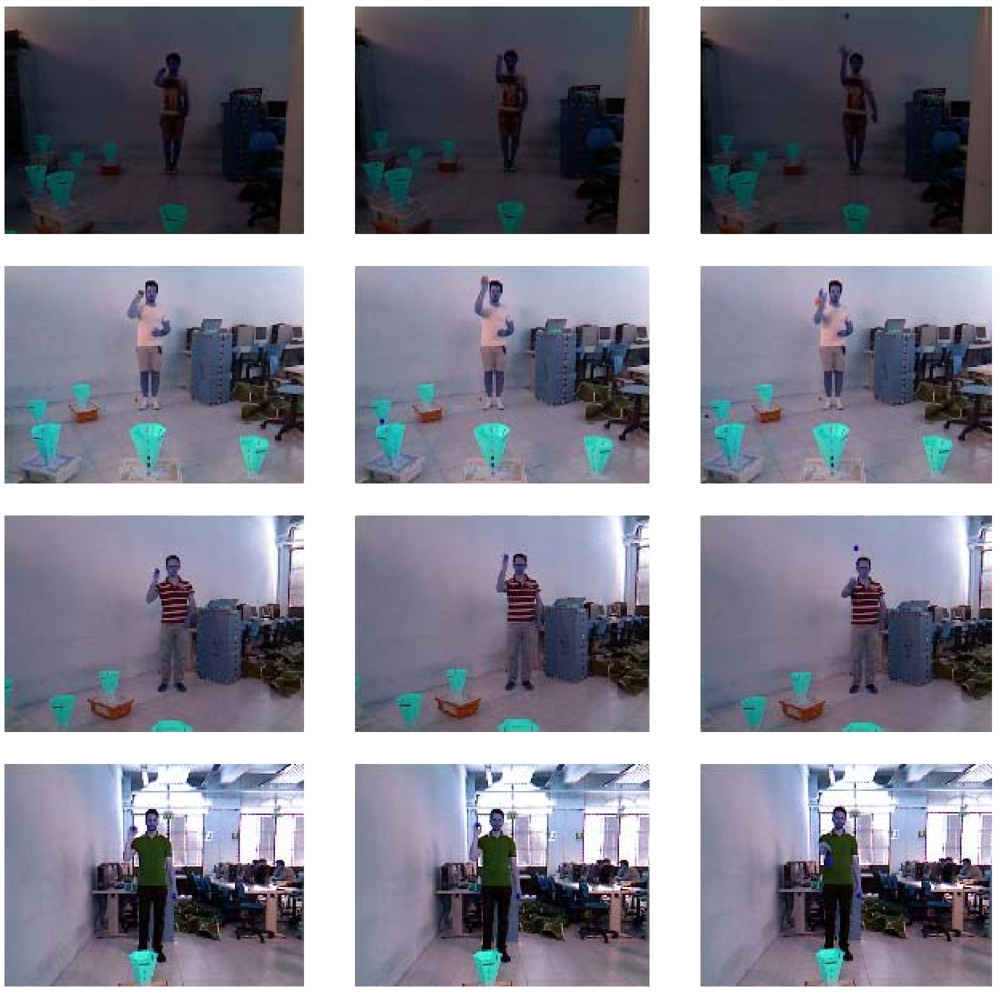


Figure 4.15: Frames of three different instants (start, mid and end of a throw) for each orientation. Each recording has its light condition and different reflex, for example. In addition, each user throws in a different way.

4.2.1 Problems with the data recorded

The utilization of visual demonstrations, like we said in the introduction, brings together with his advantages various problems:

- the camera, and in result the tracker too, could not always recognize the user skeleton (the mapping can disappear or can flicker). The reasons are related to the distance of the camera and to the brightness of the environment. This can be resolved before the start of the recording.
- during the recording the user tracking could be lost or could appear a new skeleton, simply caused from light effects, like reflects or shadows. The solution for both cases is to do again the recording or to elaborate the extracted informations to delete the errors.
- sometimes because of visual obstruction it is not always possible to track all the joints of the user. This is caused from environment obstacle or simply from the user himself (his same body obstructs for example one side). To overcome this situation we need to record the movement from a different angle.

In addition to these problems, some other casualties could appear and we need to face them after the data extraction. If the post-extraction software elaboration does not lead to an acceptable result, thanks to the high number of recordings we can simply decide to discard some throws.

4.2.1.1 Post extraction software elaboration

To create the initial dataset and try to overcome to the various errors we applied some elaboration on software side:

1. *Second skeleton filter:* after the circumscribing of this phenomena, we inserted a barrier that filter the values assumed and that discard points which create a big discontinuity in the trajectory (like high variation of the velocity);
2. *Extraction of values between the global maximum and the global minimum:* at the start and at the end of the movement could be some oscillation in the trajectory, caused from human errors during the separation of the various throws (it is not simple, due to limitations of the recording instrument too, to cut perfectly from the start to the end of a throw without leaving some spurious data). With this extraction we are sure to include the entire throw, from the minimal to the maximal extension of the arm.

3. *Search of the local maxima and local minima:* although we are sure to have included the entire movement, the data extracted until this moment could contain other fluctuations. We need to find the stationary points since the desired trajectory is certainly include between a local minimum and a local maximum.

4. *Extraction of the interval with the highest number of points:* between all the intervals found, we can assume that the trajectory is surely the dataset with more points (it is not like that we found a fluctuation so big that has more points of the entire trajectory).

5. *Acceptance of the trajectories with more than a fixed number of points:* we cannot create a dataset with very sparse data points trajectory. For the correct building of the model we need to have a minimum number of points for each trajectory used, so we discard each trajectory with less than a chosen number of points.

After all this elaboration, we limit the number of trajectories associate to each basket distance to ten (for balancing reasons of the dataset), although we can have more than those.

At this time we have built a dataset with the same format presented in the Section 2.1. In the following sections we present how the model is built starting from real data and the choices done to overcome the problems that will come out.

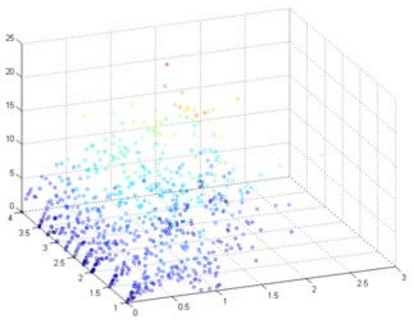


Figure 4.16: Initial dataset with the collected trajectories.

4.3 Donut Mixture Model based on real data

The objective of this section is to verify how the model behaves with real data and, as we see in the following paragraphs, why we need to do various refinement for tuning the data to come closer to the ideal situation of the synthetic data. In each paragraph, alongside the analysis, we show how the model is built and how is the output trajectory. Thanks to the preliminary study we proceed using a value of K equal to 2 or 3, because they are associated to the minimum value of BIC. Until we reach a good result, we choose to keep fixed the input value related to the basket position to 2.5 meters.

4.3.1 Preliminary analysis on the raw dataset

We start from the dataset that came out from the mapping elaboration on the recordings. Before we build a first model, it is interesting to compare side by side the actual real data with the synthetic ones (specifically re-adapted for a significant comparison) .

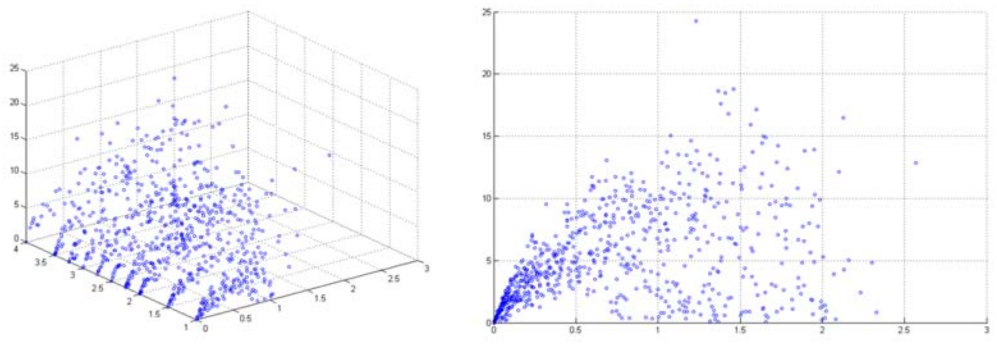


Figure 4.17: Initial dataset from two different point of view.

As we see in Figure 4.17 and Figure 4.18, we can identify straight away some differences (the synthetic dataset is adapted to the real values so the comparison can make sense):

- the shapes of the real trajectories are not symmetric. Someone tends to a parabolic shape, but the right side (points past the vertex) is far more inclined. In addiction, for how a person throws the ball and finishes the movement, the trajectory can instantaneously interrupt without a decreasing side;

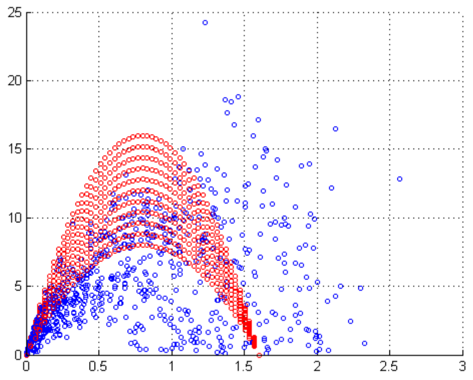


Figure 4.18: Comparison between real data (blue) and synthetic one (red).

- the number of data points for the real data are a lot fewer than in the synthetic dataset;
- in the synthetic dataset the trajectories finish all with zero velocity and all in the same position. The variability is concentrated on the center of the curves. In the real case every trajectory has a different behaviours and the variability is shifted more to the right side.

Taking into accounts all this things will be very important in the next paragraphs to improve the dataset.

In the next paragraphs we place only two or three Gaussian components on the dataset. This choice derives from noticing that with the real data the components are arranged not in a symmetric way and interact too much by themselves. To limit this behaviour we decided to reduce to the minimum the various intersections using only a limited number of components (two and three).

4.3.1.1 DMM on raw data

In Figure 4.19 and 4.20 we can see how the Gaussian components are placed on the dataset with the real data collected.

The first thing that we can notice, for the asymmetric shape and the widespread overall variability, concentrated in particular on the final part, is that the one Gaussian component covers a wider area and the shape of the smaller is more concentrated.

The output (Figure 4.21) is not what we expect and it is very different from the ideal one. This is caused by what we just highlighted. We need to

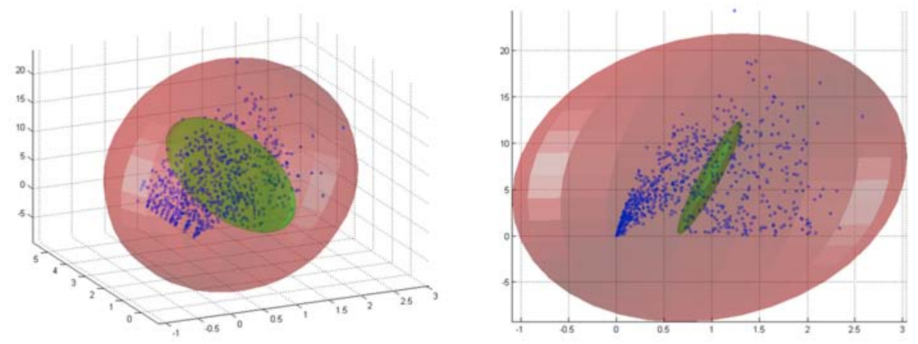


Figure 4.19: Gaussian components positioning on initial dataset (K=2)

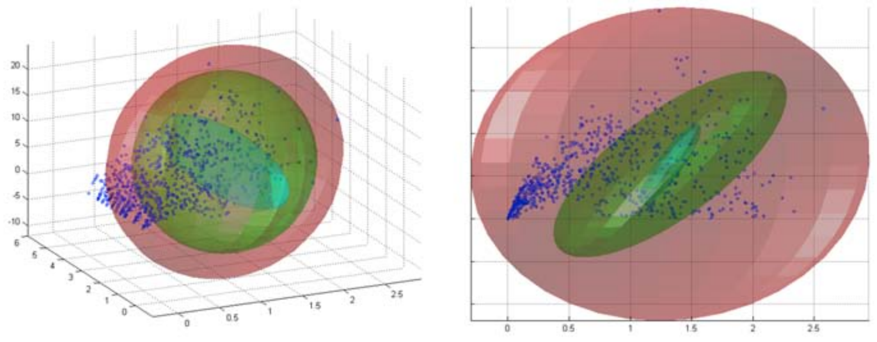


Figure 4.20: Gaussian components positioning on initial dataset (K=3)

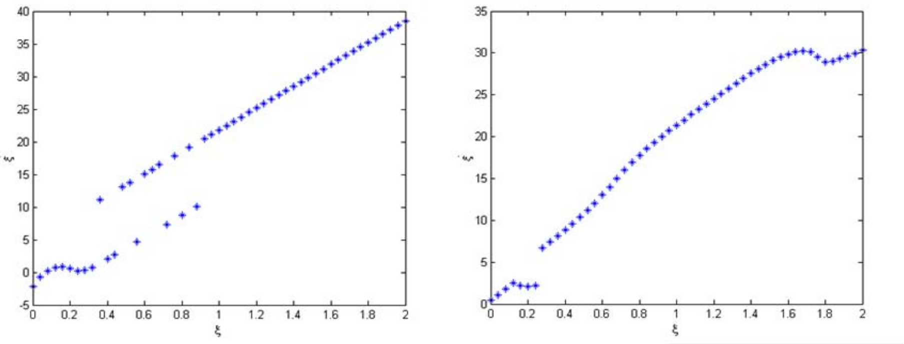


Figure 4.21: Output associated to the initial dataset for a distance z=2.5m (K=2 on the left, K=3 on the right).

intervene to fix the various causes of this result. What impact the most the model is surely the high final variability, because as we see the red Gaussian component is a lot different from the others for both values of K used. In addiction, the components intersect themselves a lot and this is reflected on the evaluation of the pdf on the optimization phase. Because of the high ending variability of the data, the model cannot learn the right behaviour.

Like we said in the preliminary analysis of the raw data, in the followings paragraphs we try to cut the distance between the real data and the synthetic one.

4.3.2 Analysis of the improvement of the model alongside the refinement of the initial dataset

This section is dedicated to the evolution of the initial dataset alongside various solutions adopted.

We start limiting the high variability of the real data in two ways:

- adding a new final point to each trajectory with zero velocity (ordinate). We decided to choose as position associated a value of 10% more than the last position value of a trajectory.
- re-scaling each trajectory so each one can ends in the same point, as in the synthetic data. There are various options for the choice of the rescaling point, as the minimum or the maximum between all trajectories. We opted for the mean one.

Applying these first two solution changes the dataset as in Figure 4.22.

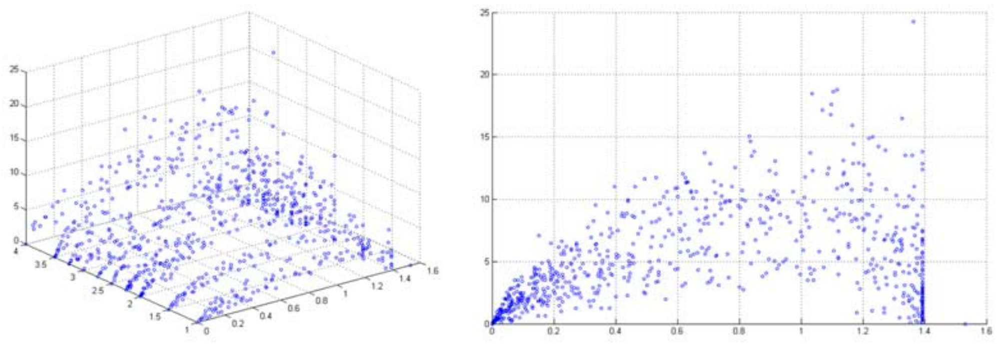


Figure 4.22: Rescaled initial dataset.

As we see the trajectories behaviours are more confined and less variable than in the first case. This should change and improve the components positioning but it is still not sufficient to reach a good output.

Using this data in the model building gives a positioning as in Figure 4.23 and 4.24.

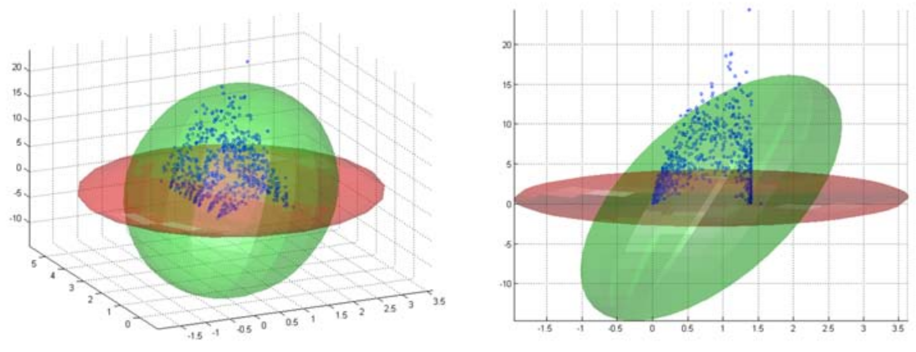


Figure 4.23: Gaussian components positioning on data scaled (K=2)

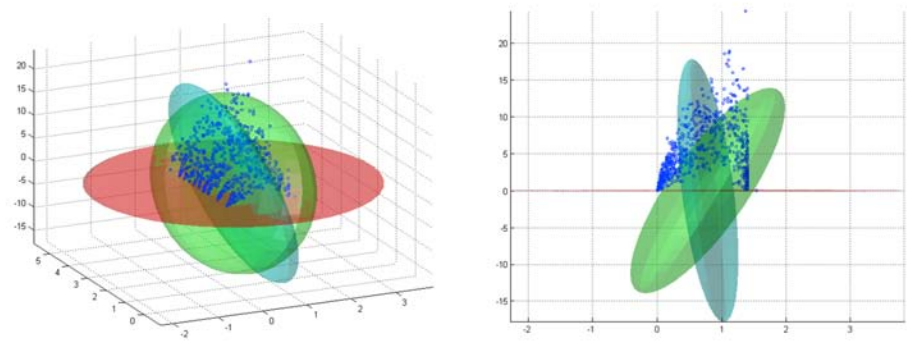


Figure 4.24: Gaussian components positioning on data scaled (K=3)

Giving more regularity to the data it is still not sufficient to overcome the limitations of the initial raw data. The addition of the last point it is not so effective because the dataset remains very sparse. In addition, adding an isolated point at the end of each trajectory compromise the positioning of the Gaussian components, influencing too much and anchoring the entire model, as we see from the output (Figure 4.25) too.

This adjustment is surely important for the improvement of the real data, but for this step we obtain a better result considering only the rescaling of the trajectories (Figure 4.26).

The Gaussian components are better positioned than before and the weight of each one is a bit more balanced. With the overall variability improved, the output regained a bit of sense but the final behaviour is still not

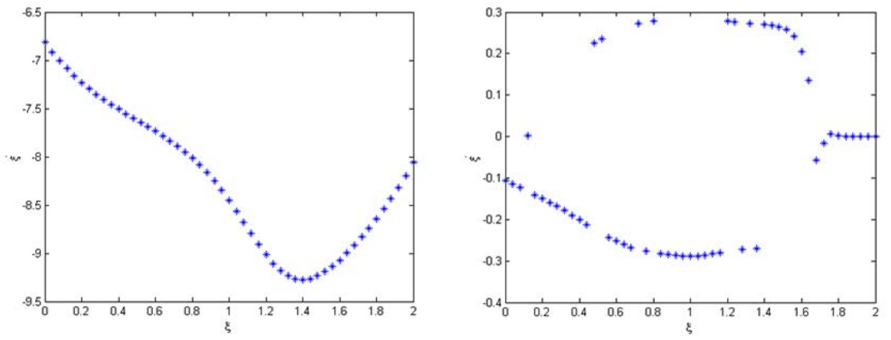


Figure 4.25: Output associated to the scaled dataset for a distance z=2.5m (K=2 on the left, K=3 on the right).

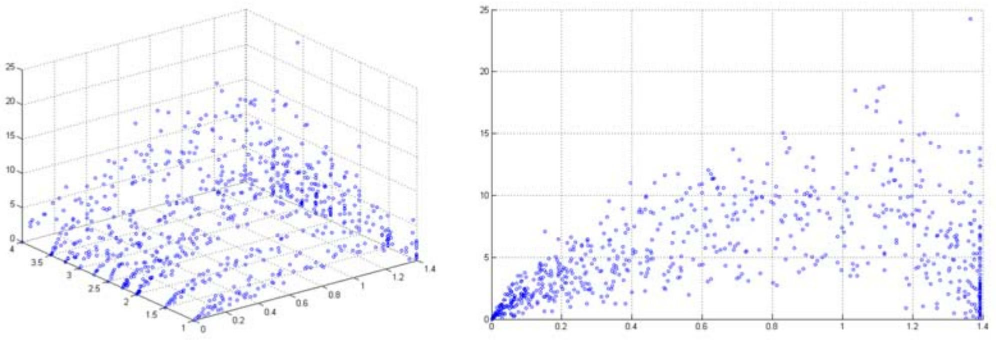


Figure 4.26: Rescaled initial dataset without the add of an ending point.

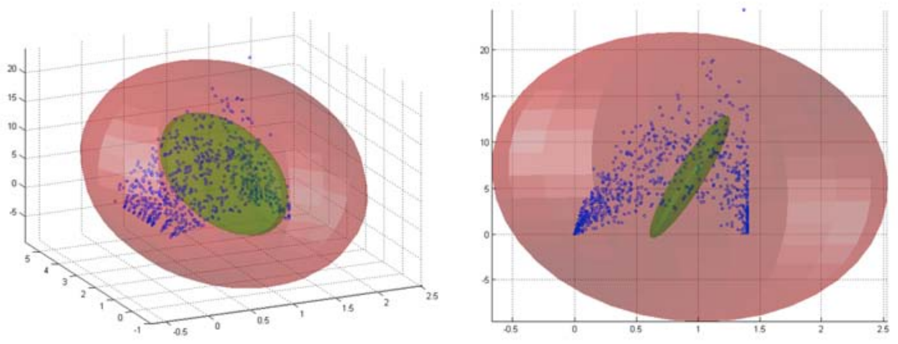


Figure 4.27: Gaussian components positioning on data scaled (K=2)

learned.

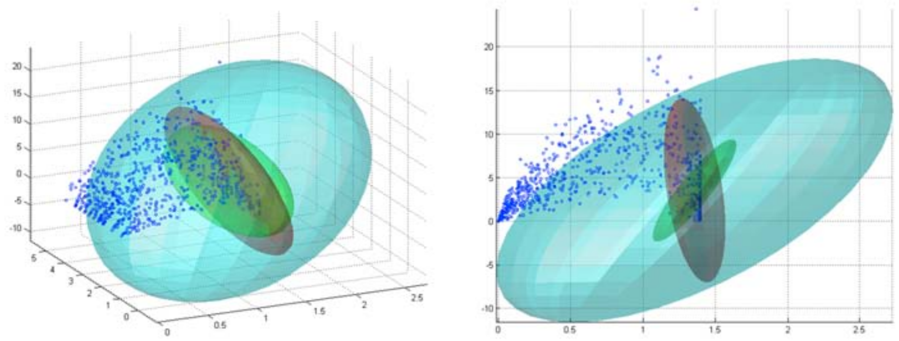


Figure 4.28: Gaussian components positioning on data scaled (K=3)

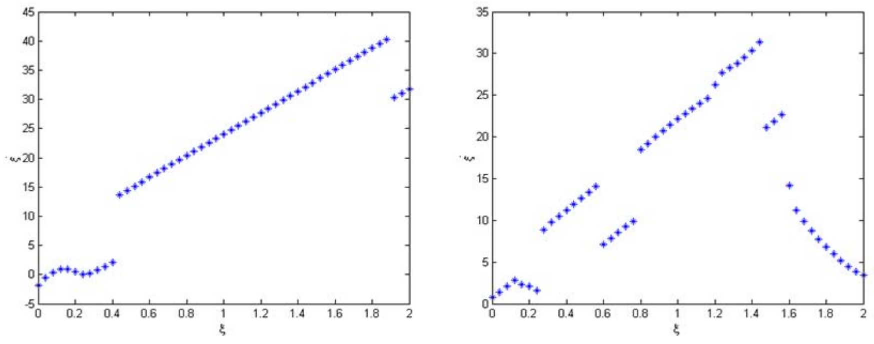


Figure 4.29: Output associated to the scaled dataset without the add of an ending point for a distance z=2.5m (K=2 on the left, K=3 on the right).

To cope with the limited number of points of each trajectory ant to link the last added point to the others, it is necessary to increment them in some way. A solution is to choose an interpolation that works well with this kind of data. Between the various interpolation method, our choice fell on the one that uses for the interpolant splines, a particular polynomial method based on the use of interpolation curves preferred to standard polynomial methods for his small interpolation error even when using low degree polynomials.

Using the spline interpolation the dataset change as in Figure 4.30. The interpolation bring us to lose the initial points of the raw dataset, since the new points belong to a function. To extract the new trajectories, we need to choose a step by which we inspect the associated velocity.

However the side effect of using a curve based interpolation, although we resolved the sparse points problem, is that increases in variability. To solve this new problem it is necessary to insert new anchoring points in the dataset before the interpolation, so the interpolation curve is forced to go through

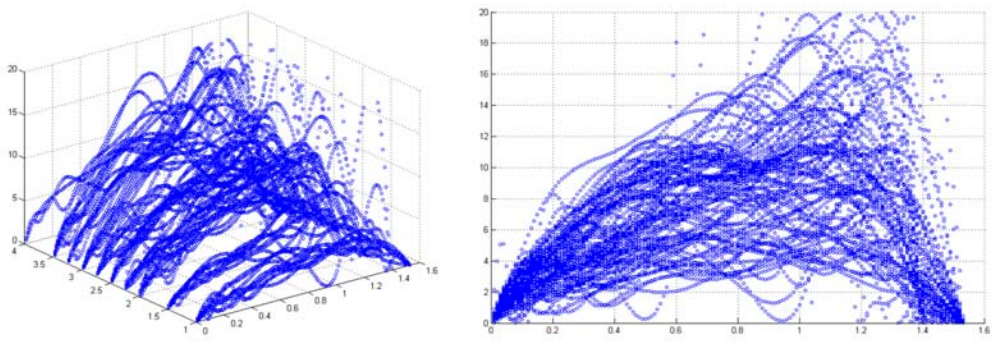


Figure 4.30: Rescaled initial dataset interpolated using splines.

them without wide arch from a point to the next one. We choose to add three average points between each pairs of raw points, in correspondence of 1/4, 1/2 and 3/4 of the position and the velocity of them.

The dataset changes as in Figure 4.31.

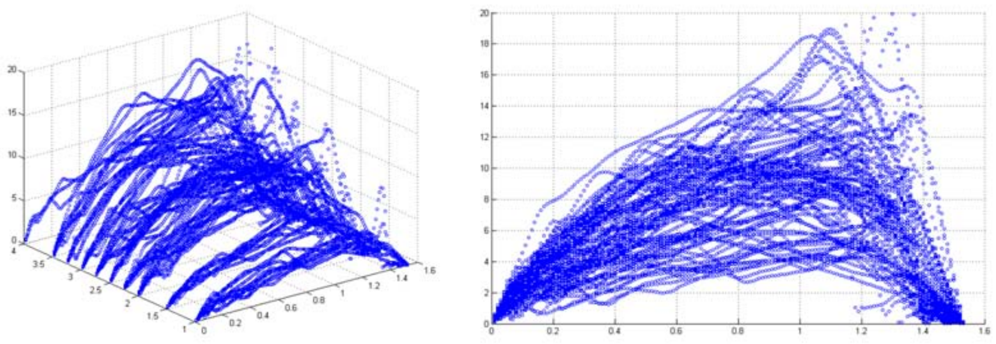


Figure 4.31: Rescaled initial dataset interpolated using splines and with the adding of new points to limit the oscillation of the interpolating functions.

We achieved a state where we overcame the variability problem and the sparse problem. At this point the Donut Mixture Model and the corresponding output looks as in Figure 4.32, 4.33 and Figure 4.34.

The output has surely a smoother behaviour but the high slope of the right side of the trajectories still influence too much the model, making difficult to extrapolate the overall tendency. This is confirmed from the positioning of the red Gaussian component, tighter that the others and stick to the ending side. Starting from this last problem, in the next paragraph we try to take

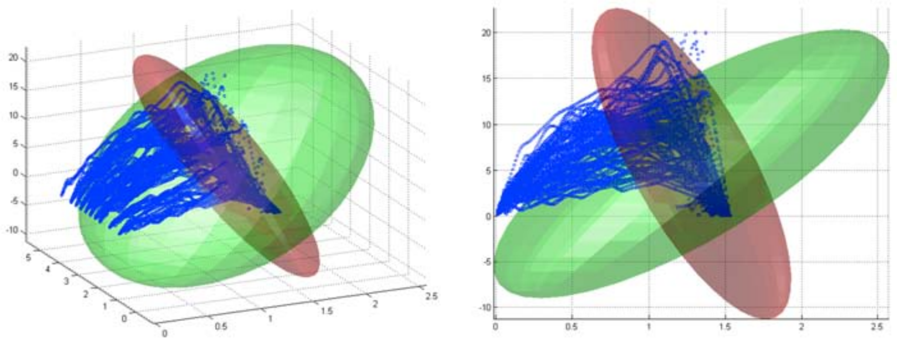


Figure 4.32: Gaussian components positioning on data interpolated through spline (K=2)

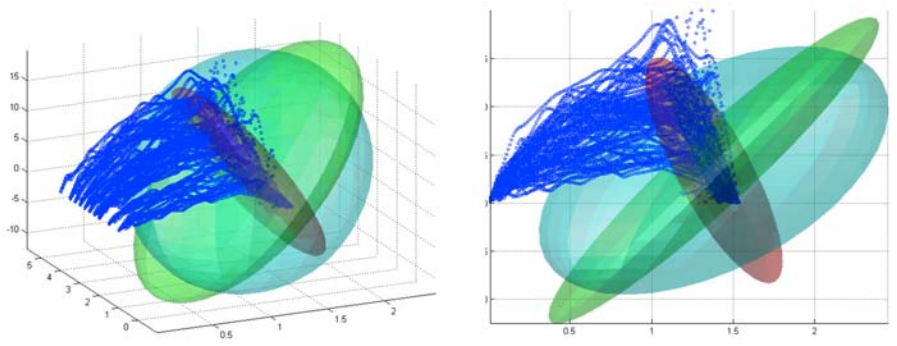


Figure 4.33: Gaussian components positioning on data interpolated through spline(K=3)

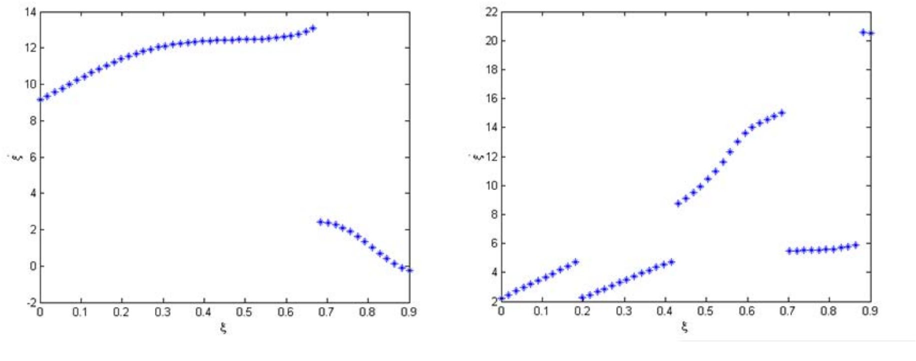


Figure 4.34: Output associated to the scaled interpolated dataset for a distance z=2.5m (K=2 on the left, K=3 on the right).

some new solutions to surpass it.

4.3.3 Further data elaboration

In this paragraph our objective is to find new solutions of different type from what we have already done. Initially we tried to act on the dataset to improve the overall data shape. Achieved a good result, the next step is to do further analysis in terms the intrinsic characteristic of the re-elaborated data. It is important to underline and to recall that at the start of this work we chose to considerer the movements in their entirety and not until the ball is left. This was a good choice on theoretical side for trying to give to the model a tending symmetrical balanced data (to help the Gaussian components positioning too), but for the shape of this particular dataset it is not so relevant also because the weight of the first positioned Gaussian component makes it too much predominant.

From these initial considerations we can now proceed to give to the model only a part of the entire dataset to verify if from only the first part it can learn the right behaviour without any problems. We return to a situation where the last point of the trajectory is effectively correspondent to the instant that the ball is left.

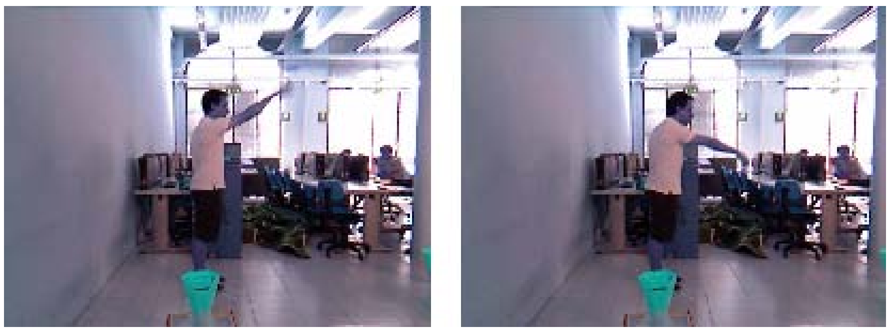


Figure 4.35: Two frame of a throw: when the ball is left and at the end of the movement.

To choose the cutting point from which we eliminate the second part we can proceed with an arbitrary point, with the mean one or we can do some considerations about when, according to the velocity and the acceleration of the trajectories, the ball is probably left.

In a first moment we can choose an arbitrary point, with further elaborations this choice could be more accurate. Cutting in correspondence of 0.9 radians (near the mean) the dataset becomes as in Figure 4.36.

Starting from this new group of input trajectories, the Donut Mixture Model results as in Figure 4.37 and Figure 4.38.

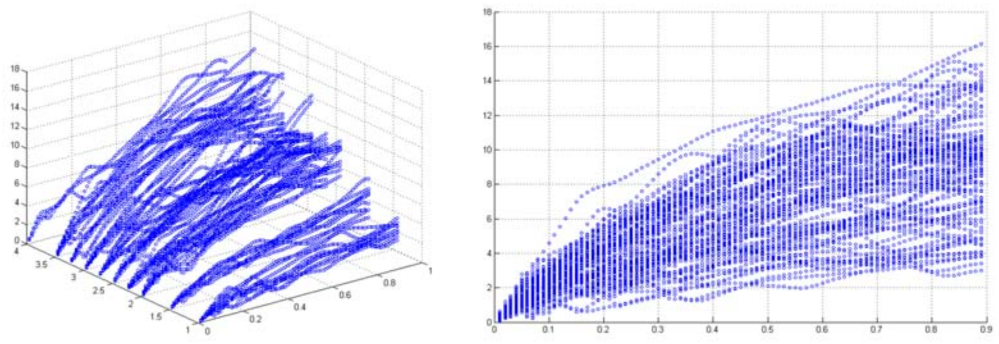


Figure 4.36: Same re-elaborated data as 4.31 cut at 0.9 radians.

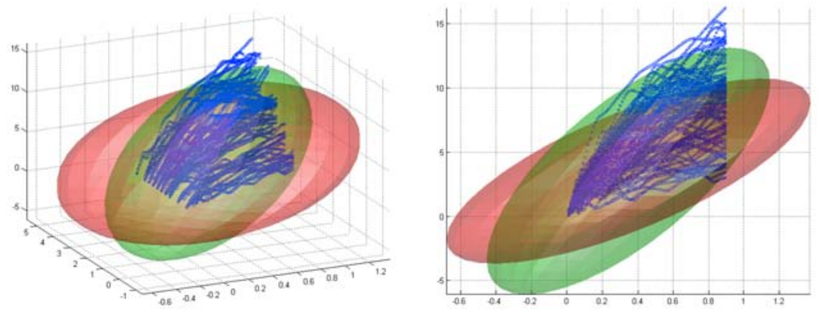


Figure 4.37: Gaussian components positioning for the elaborated data cut at 0.9 radians (K=2).

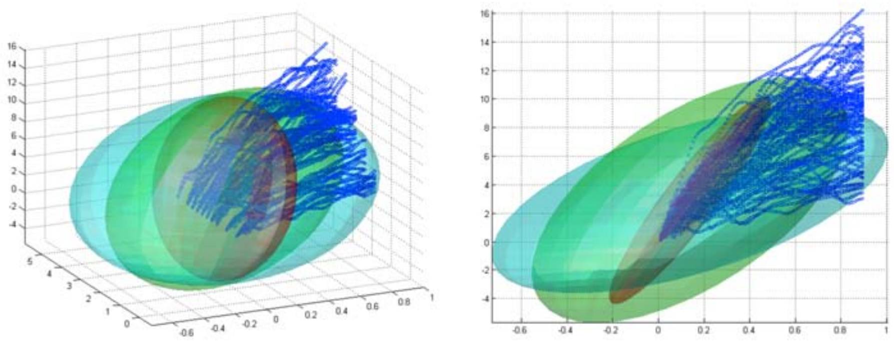


Figure 4.38: Gaussian components positioning for the elaborated data cut at 0.9 radians (K=3).

The associated output for a distance of the basket of 2.5 meters is as in

Figure 4.39.

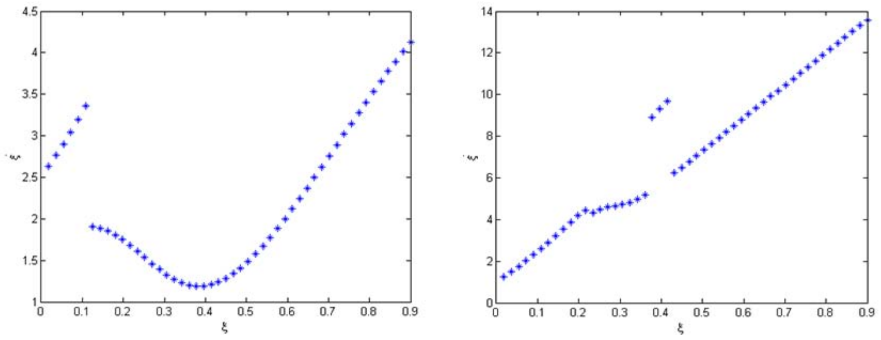


Figure 4.39: Output associated to the cut data for a distance z=2.5m (K=2 on the left, K=3 on the right).

This first result with this new dataset it is good for three Gaussian components. To find out if the model works well, we try to change the basket position so we can at the same time verify the smoothness of the output and if the model learns the right wanted behaviour.

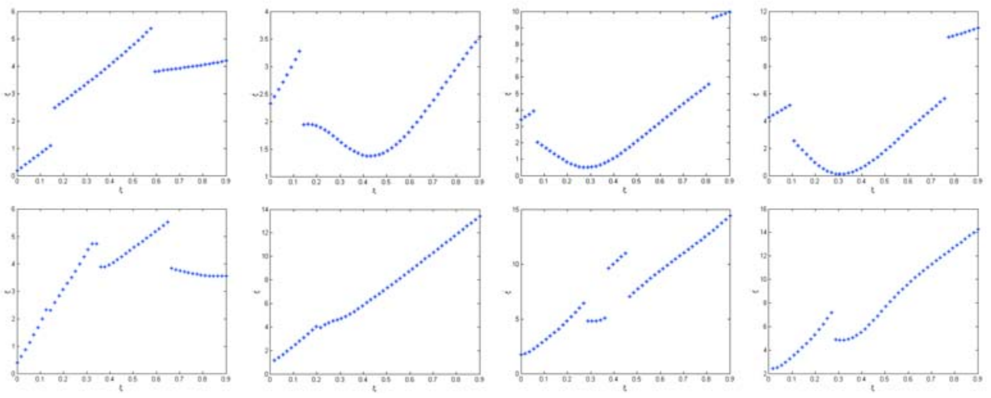


Figure 4.40: Output associated to different basket distances with two (top) and three (bottom) Gaussian components (respectively from left to right 1.5, 2.35, 3.25 and 4 meters).

As we see in Figure 4.40 we are close to a good result but there are still some problems, so we need to do some other steps further. The discontinuity gaps are still too wide and the output change too much if we change the basket position or simply the number of Gaussian components.

The last thing that is left to considerate in this elaboration is the correlation between the trajectories associated to different distances. If take them singularly, as in Figure 4.41, we can see that there are not any particular overall differences (aside the fluctuation related to the spline interpolation).

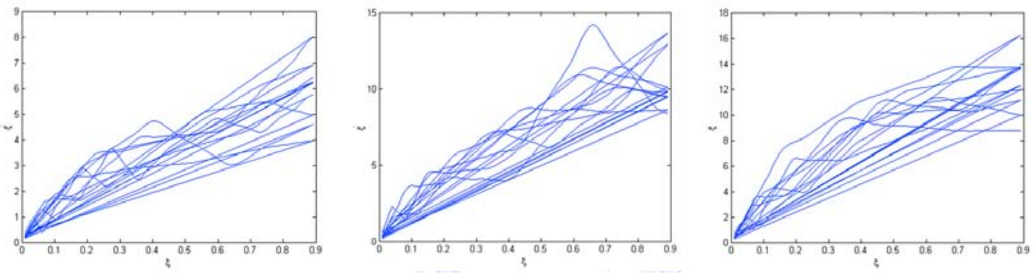


Figure 4.41: Trajectories for some distance (respectively 1.5m, 2.5m and 3.5m).

What we can expect is that to further distances are associated trajectories in a proportional way. In general as we see this statement is respected and taking for example the set of trajectories in correspondence of 2.25 meters and 3.5 meters, the second one are generally over the first one (Figure 4.42).

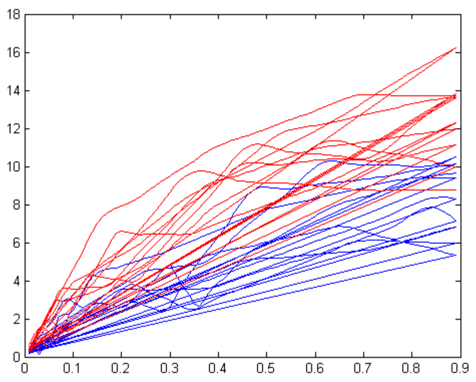


Figure 4.42: Trajectories associated to 2.25m (blue one) and to 3.5m (red one).

The problem is that this consideration is not respected for all of them. We can highlight two different situations:

- singularly the behaviour is correct but differs from the overall trajectories trend. It is the case of the trajectories for the distance of 2.75 meters. The trajectories around are gradually growing but for 2.75 meters there is a drop of the average velocity/position ratio.

- the behaviour associated to a distance follows the global trend but differs too much, for external factor too, from the other trajectories. It is the case of the trajectories of 1, 1.5 and 4 meters. In our case this can be caused from external difficulties related to the throws itself. For example, the baskets at 1 and 1.5 meters are too close to the throwing spot and it is not simple to do a full movement like with the other distances.

In front of this last considerations we can try to remove from the re-elaborated dataset this trajectories set to see if the Gaussian components positioning improves. The new dataset results as in Figure 4.43.

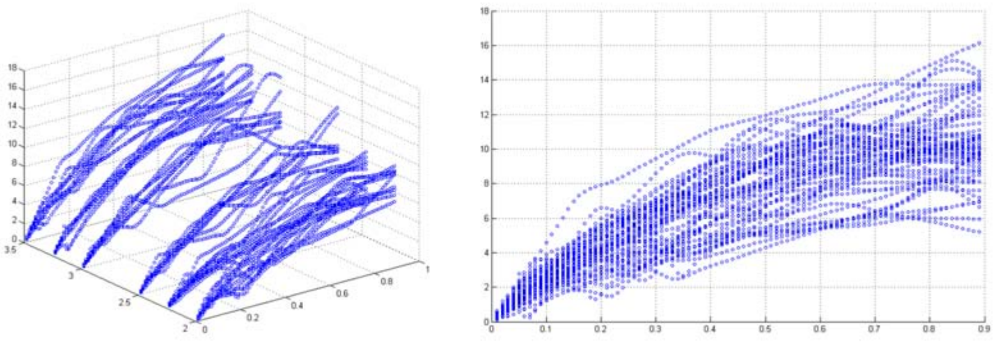


Figure 4.43: Re-elaborated data with some trajectories removed.

As we see the result is globally homogeneous and, as showed in paragraph 4.3.5 where we go deeper with the analysis, the model behaves well.

4.3.4 Analysis on the elaborated dataset with two DoFs

The actions that we took to elaborate furthermore the dataset are strictly related to try to achieve our objective with three degrees of freedom. What we achieved till now, excluding the last consideration related to the distance that have to be excluded, is the results of elaborations done on single set of trajectories associated to a value of basket position. So we can surely assume that at this point Donut Mixture Model behaves well with two degrees of freedom. As follows we present the study parallel for each input basket distance.

The dataset, subdivided for distance, is presented like in Figure 4.44.

As we can see, each trajectories set has a semi-parabolic shape and differs only for the grade of its variability. Using them separately into the DMM

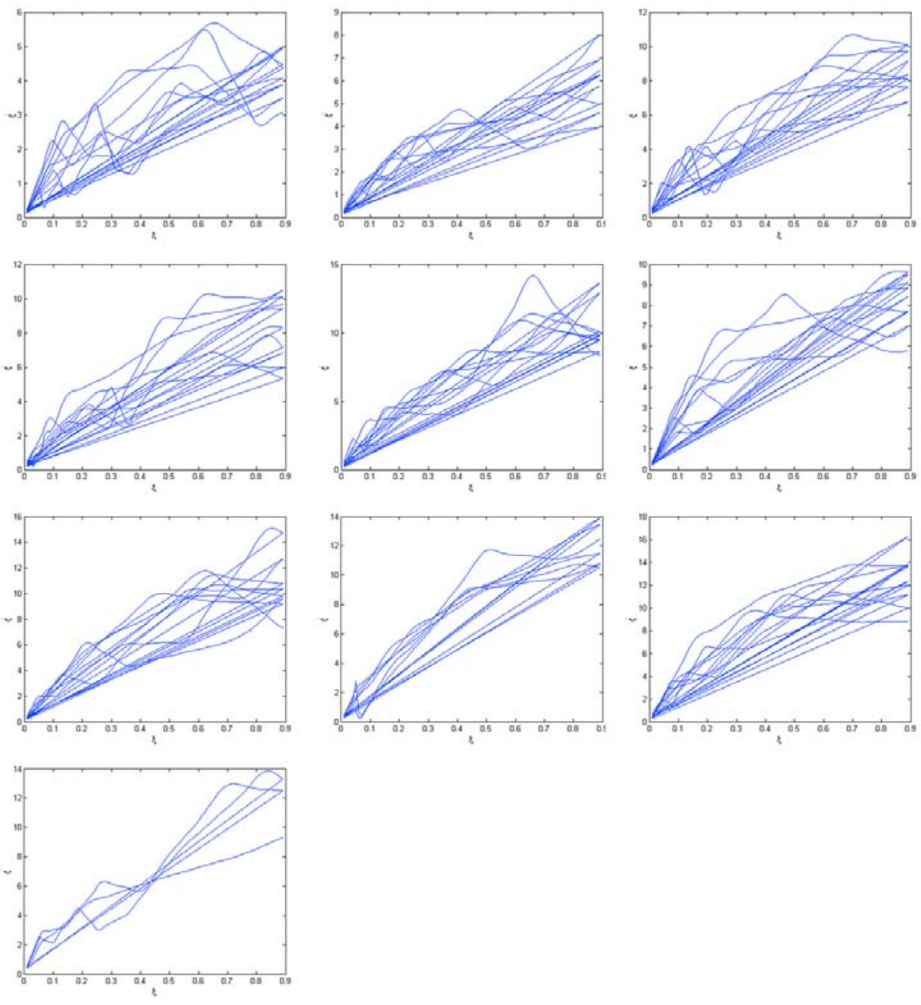


Figure 4.44: Data points elaborated subdivided for each single distance recorded.

gives a similar result for everyone, with a tighter first Gaussian component and a wider one associated to the second part of each set (using two Gaussian components), like we see in Figure 4.45.

For each distance we can see that the first of the two Gaussian components is usually less variable and more close to the data points. The second one instead wraps around the variability. We tested how three Gaussian components are positioned too but we saw that two are sufficient and that there are not any improvements. If we add a third component the positioning variate between each set but it does not add anything significant on the output side. The associated output that comes from the optimization phases, as we see in Figure 4.46, is smooth and it is what we expect.

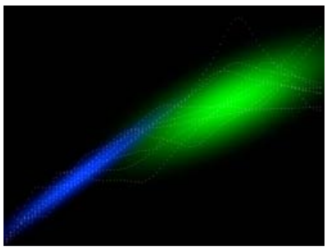


Figure 4.45: Gaussian components positioning for 2 DoFs (K=2)

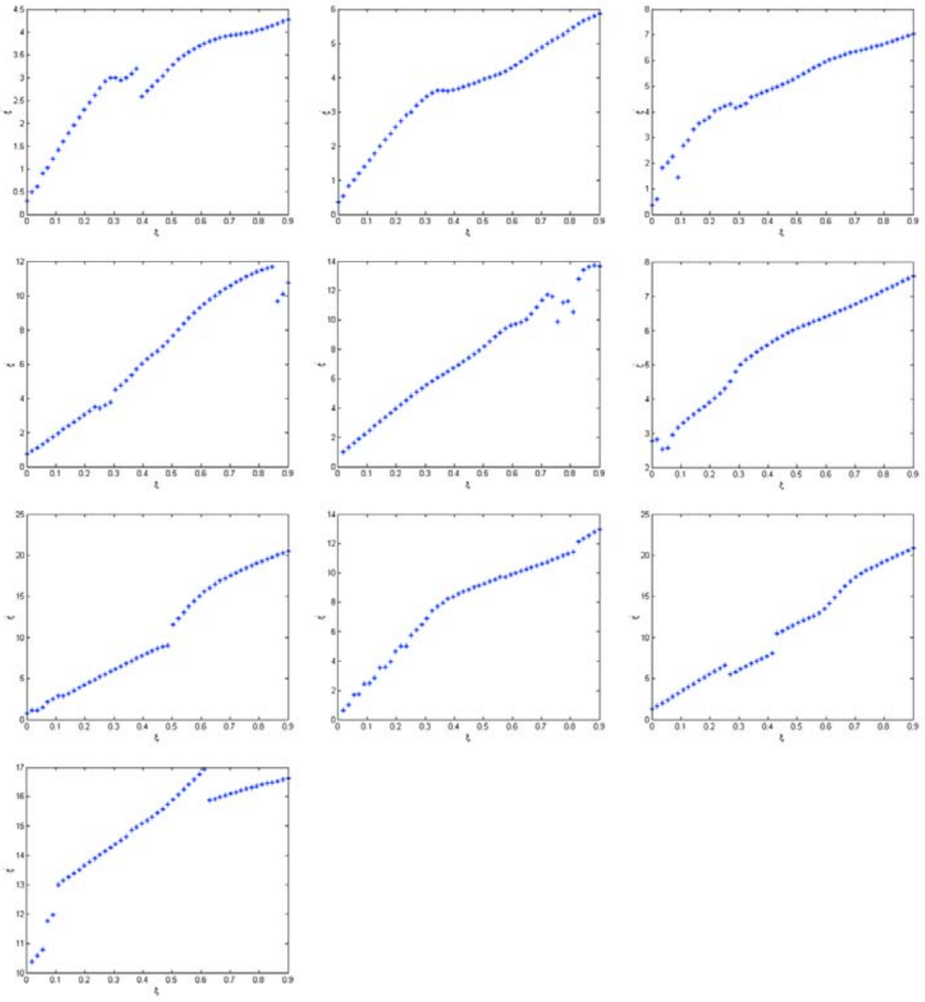


Figure 4.46: Set of output associated to each individual set of data elaborated for every distance.

4.3.5 Conclusive analysis on the model with three DoFs

This last created dataset is the result of numerous actions on the raw data. They can be recapped in the following list:

1. addition of a new final point for each trajectory with zero velocity;
2. re-scaling of all the trajectories to the average final position, so each one can finish at the same time;
3. interpolation of the data points of each trajectory through spline, so the probability model can work with more informations to extract the behaviour;
4. addition of average points, so the oscillations of the curve generated by with the interpolation are limited;
5. cut in half of the complete trajectories, so the probability model can stood better to the correct behaviour;
6. deletion of trajectories associated to some basket distances, so the data points of the dataset can be more homogeneous disposed as possible.

As we saw the last two operations was required to evolve the dataset so the model can be applied to three degrees of freedom. Starting from this, we proceed as we did with the synthetic starting dataset.

Using two and three Gaussian components, their positioning appear like in Figure 4.47 and 4.48.

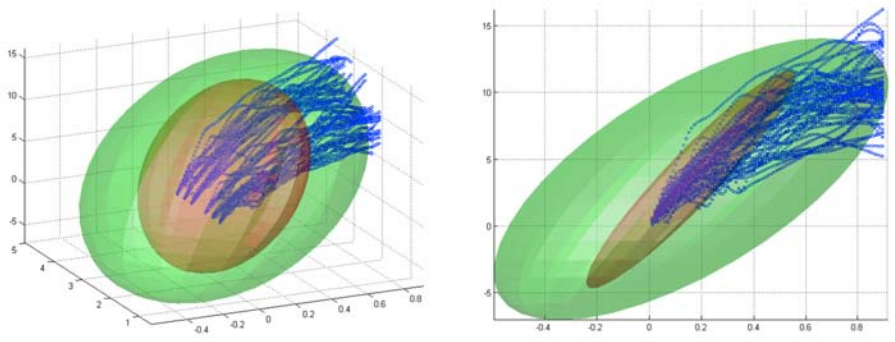


Figure 4.47: Positioning of two Gaussian components in the last elaboration of the initial dataset.

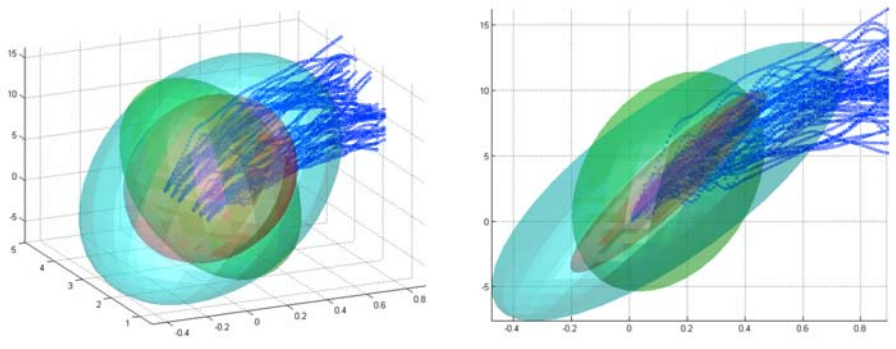


Figure 4.48: Positioning of three Gaussian components in the last elaboration of the initial dataset.

As we see from the side it is very similar to the positioning with only two degrees of freedom, with the difference that the wider components wrap all the data points and not only the last more variable set.

The output associated, always with a distance of 2.5 meters (as we kept in the previous steps) it is smooth (there are only some little discontinuities that do not influence the entire trend) and stick well in its exploration around the data (see Figure 4.49).

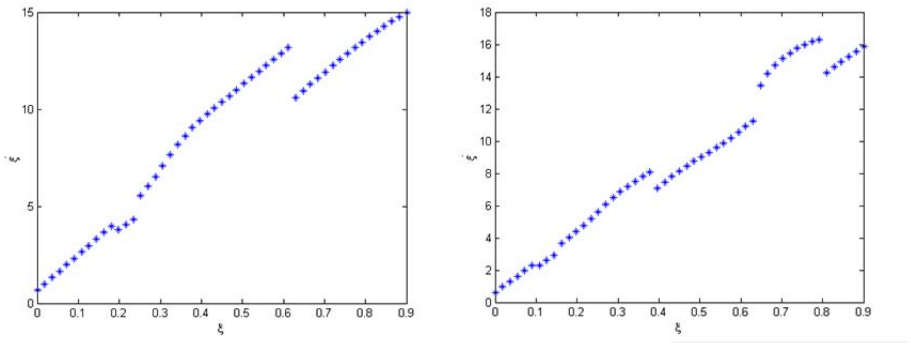


Figure 4.49: Output associated to the last dataset with a distance of 2.5m.

At this point we can now proceed to evaluate if the model has learnt the input behaviour and can adapt well to different input basket position values.

Figure 4.50 confirms that the model has absorbed well the wanted behaviour: to a further distance of the basket corresponds a trajectory with an higher interception with the y-axis and an higher velocity/position ratio. There are still some points of discontinuity, but what is important is that the overall behaviour has sense. In addition we can see that the model adapts well in unknown situations too (without initial information when the model

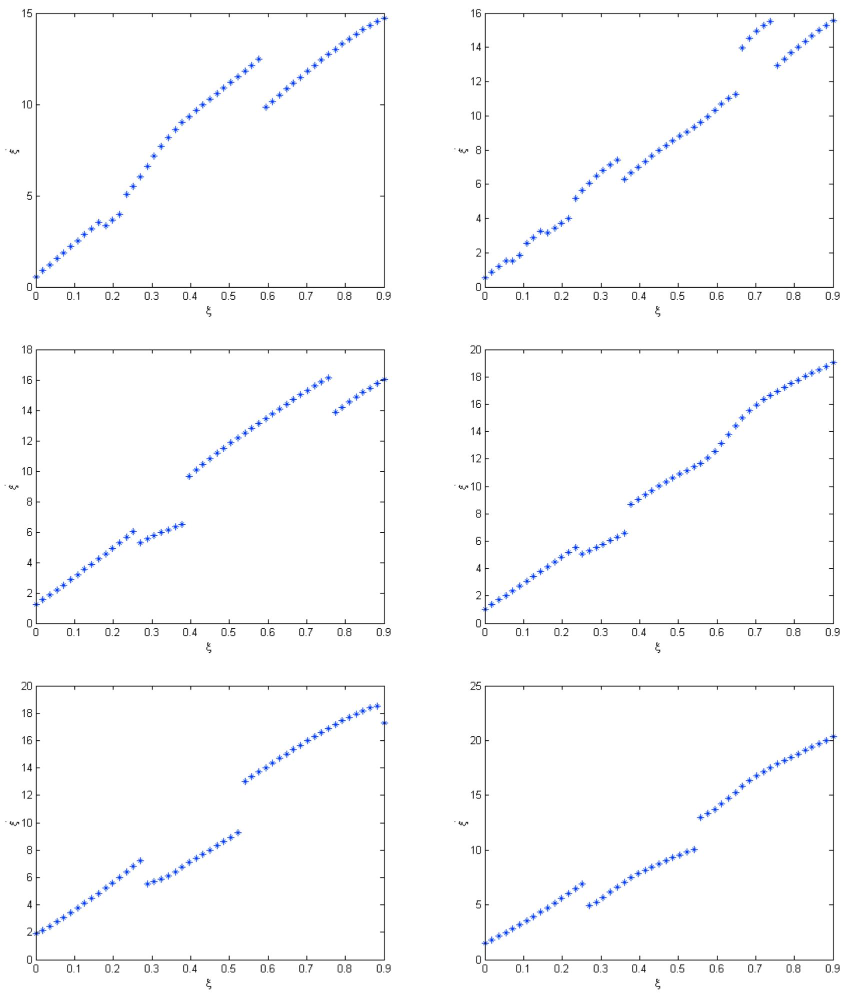


Figure 4.50: Output for two (left side) and three (right side) Gaussian components for distance respectively of 2.35m, 3.25 and 4 m.

is built), like for 2.35 and 4 meters.

This result it is not trivial: if we take the initial raw dataset cut on 0.9 radians and we compare the initial Gaussian component positioning and the output from that dataset with what we obtained we can see the huge differences and, more important, that every subsequent elaboration has lead to a good result that we could not achieve from the start.

If we compare the Gaussian components positioning (Figure 4.47 and 4.48 for the re-elaborated data and Figure 4.51 and 4.52 for the initial dataset) it is immediate to see that in the former case the components are less variable and the single positioning it is better. This is reflected on the output side

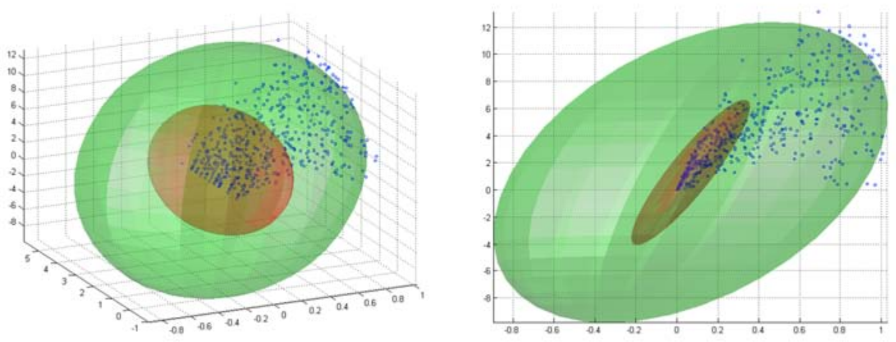


Figure 4.51: Positioning of two Gaussian components on half of initial dataset.

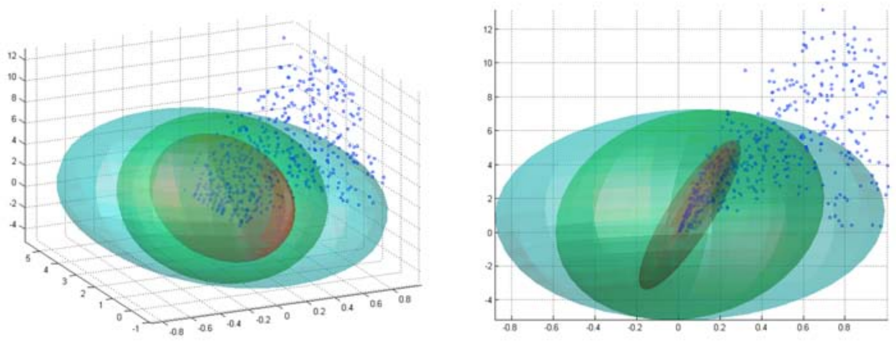


Figure 4.52: Positioning of three Gaussian components on half of initial dataset.

(Figure 4.47 and 4.48 for the re-elaborated data and Figure 4.51 and 4.52 for the initial dataset).

In particular it is important to underline that changing the input basket distance do not determine any interesting effect with the raw data. The model cannot achieve to learn any behaviour and that is confirmed looking at the random variation in the output. On the other side, with the elaborated data, the output trajectory change in height and increase with the associated distance.

4.3.6 Limits of the model

This study has lead to various elaborations in front of various problems that came out. Facing all these matters brought to extrapolate some different thoughts.

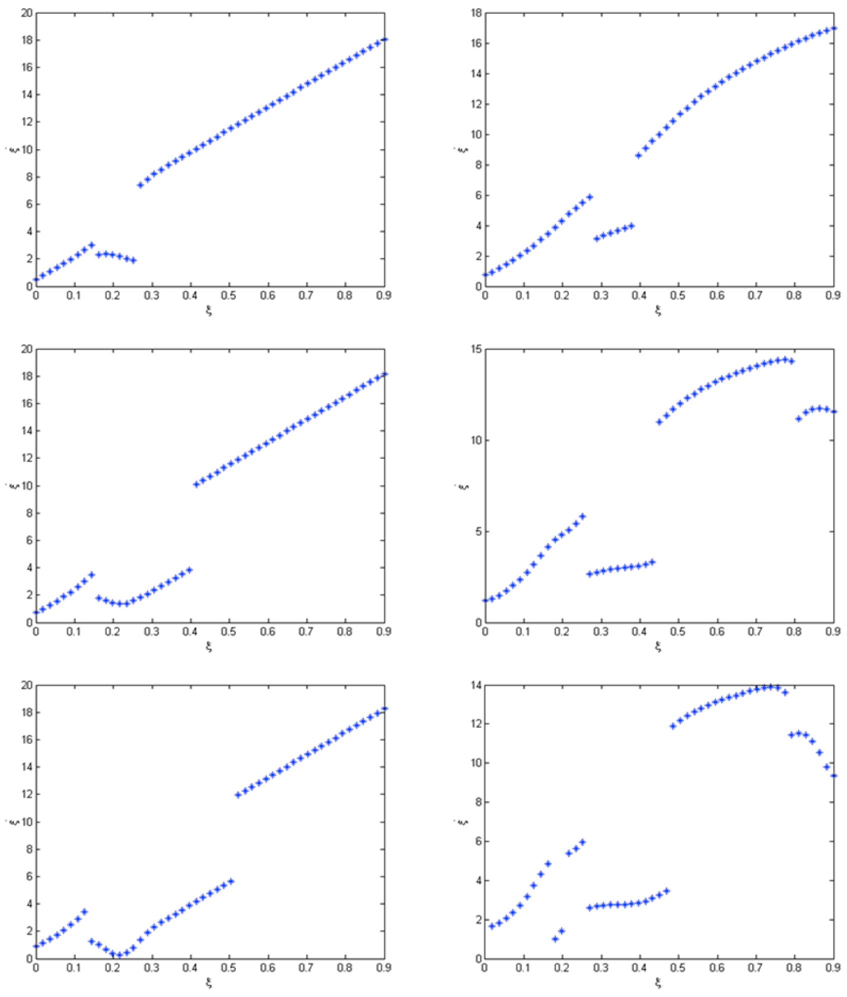


Figure 4.53: Output for two (left side) and three (right side) gaussian component for distance respectively of 2.35m, 3.25 and 4 m. for the raw dataset.

4.3.6.1 Cope with visual data extrapolation and elaboration

The main difficulty working with data extracted from videos is the condition of the information: although it could be correct, usually is dependant to the sampling rate of the camera and, like in our case, it can be very sparse or it can contain informations that we need to discard (noise in the data or simply incorrect informations). In both cases it is necessary to elaborate in some way the initial dataset.

This elaboration is not trivial and human intervention can be needed. It is not immediate to isolate the wrong behaviours, although it is possible to come out with some elaboration, like an estimation of what is correct on the base of the general trend. In our case we found manually that, for example, the second skeleton (produced by the shadow or a reflection or from the environment itself) sticks its tf associated values around the zero and that one of his point can make a huge variation in velocity terms.

Aside all this limitations, a human-like perspective remains the core for future evolutions. For the moment we can eventually use more cameras and melt and compare their signals, in the future the evolutions need to proceed in hardware terms and on the initial automatic elaboration side.

With the next sections, we suppose that all the extraction informations are correct, focusing on problems from a different angle.

4.3.6.2 Manage the data shape and variability

Extracted the trajectories, we took actions to evaluate the data in their entirety, both for a single basket distance and globally with the other cases.

In the first situation the problem was related to which shape was the right one to use. In particular, like in Figure 4.54, both trajectories are smooth and can be associated to a throw, but the robot still cannot decide, during the teaching phase, which one it has to use. A choice is required because the probability model can have some difficulties if we submit to it data with too different shape that limits the learning of the right behaviour. To solve this situation it is possible to try to estimate the shape by a comparison of all the trajectories, choosing the most common one. If the data points of a trajectory are too far from the common shape, the trajectory can be discarded.

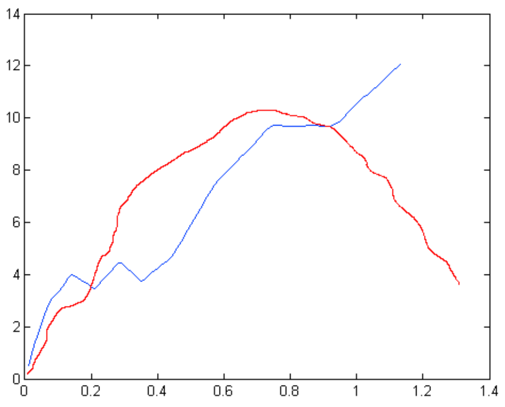


Figure 4.54: Two different kind of trajectories recorded in the collecting phase.

In the second situation, estimated the right shape of the behaviour, it is not instantaneous to accept all of the filtered data. Like it happened with the distance of 2.75 meters, the shape it is the same of the other distances but it does not cope with the overall trend, being lower than what we expected. We cannot accept that trajectories because they can interfere on the learning on the third dimension.

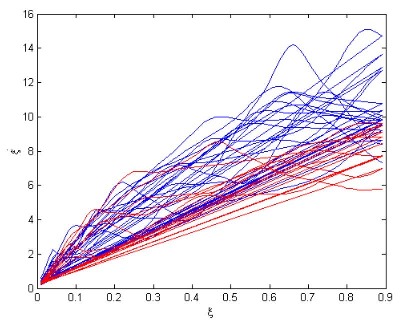


Figure 4.55: Comparison between the trajectories associated to 2.5m and 3m (color blue) and 2.75m (color red).

At the same time subtracting all that block of trajectories it is not ideal, particularly because we create an hole in the middle of the learning dataset.

Either in the two situations, the actions took for the moment were too human dependant. Our objective was related to verify the correctness of the Donut Mixture Model on the third dimension, so at this time it is not very important. In future work there is wide space for improvement on this side.

4.3.6.3 Limitation imposed from the input data

For a correct learning process we need to submit to the probability model dataset that are not very sparse. For the first two dimension this is not a problem and we resolved, if we have limitation on recording side, with an interpolation. On the third dimension the solution is not automatic. Aside of the fact that we can in some way create an interpolation on that side too, for the moment we need to use demonstrations associated to various distance. In addition, together of a discrete number of them, we need to be sure to have (for dimension after the second one) data points not so far from each other.

In our case we created an hole in the middle of the dataset but thanks to the good shape and the good number of data around it, we did not have to face problem in this way. At the same time, for the reasons we have just listed, we needed to remove the trajectories associated to 1.0, 1.5 and 4.0 meters too.

4.4 Robot application

Obtained the functioning model, we can now test the entire cycle sending to the robot the resulting trajectories. Thanks to the implemented plug-in, we can send to the robot the entire resulting trajectory to see if it moves how we want.

On simulation side, this is verified inside Gazebo.

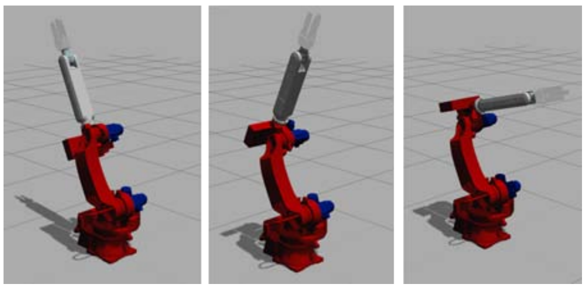


Figure 4.56: Photo of the simulated environment during the movement through a trajectory.

Working with the real robot introduce new problems, in terms of adapting the various dimension (joint position and joint velocity) so there can be a correspondence between the output from the model based on the recorded data and the executed movement. For limitations related to the actual version of the software installed on Comau Smart5 SiX, we need to re-elaborate the trajectory using the velocity not in a direct way but in terms of percentage of a linear joint velocity. In addiction a workaround is needed to send the change of velocity in every instant (point of the output trajectory). To do so we need to make some modifies to the main program and to the message that we send through a python script.

Because of software limitations, we need to sacrifice an information related to a specific joint to insert the velocity of the joint that we are going to use. As we want to use only one joint, we can take advantage of the information, for example of the last joint, inserting in its turn the percentage related to the set linear velocity associated to a joint position. In the main program that receive the message we set the position of the joint sacrificed to a constant value. The script containing the message has the following informations:

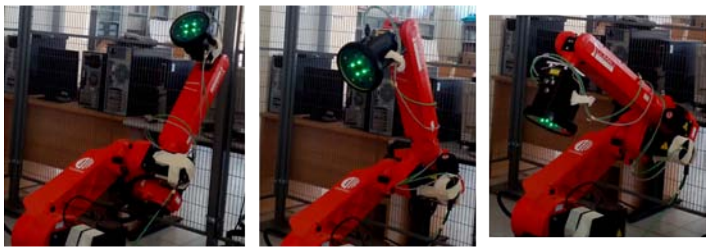


Figure 4.57: Photo of the real robot during the movement through a trajectory.

```
coord_type = "-j"
... (coord_type,'0','-65','-25','0','90','0',...)
... (coord_type,'0','-65','-37.2006','0','90','12.37',...)
                                    .
                                    .
                                    .
```

Initially we need to specify how the robot is controlled. In our case we send joint information and we do not control the robot in Cartesian coordinates. Each instant has associated a row where there are six numbers. The third is to change the position of the joint 2 and the sixth is used for the velocity.

For a correct movement we position the robot in a convenient state before moving the interested joint. In addiction, it is important to underline that if we use a real robot (in particular a non-humanoid one) it is necessary an initial tuning to its movement because there is not a direct correspondence between with ours. In this case this problem it is translated to a proportion between the values of the trajectories that we send and the limit values of the joint 2 and to some tries to find a correspondence between the velocity of the trajectory associated to a specific basket distance and how effectively the robot moves and throws for that distance.

Chapter 5

Conclusion and future development

With the work described in this thesis we presented an Imitation learning model, how this model and the associated input data have been adapted to manage a situation where the degrees of freedom are more than the initial one and they evolved during the time. In particular, we started from a previous study where the objective was to teach through visual demonstrations a simple task by looking at two DoFs and we proceeded extending the situation with another input variable. At the end of this path we can see that the extended approach is applicable and gives us good result that confirm the right behaviour can be learn.

As we introduced during the various elaboration steps, there are a lot space for future extensions. Mainly, we can divide the extensions in two felonies:

- the first one is related to taking into account more complicated tasks, where for example it is taken into consideration the basket motion not only in one direction but all around the throwing spot. In addiction the basket can change in height too;

- the second one is related to finding a more independent way to manage the dataset singularities than human intervention. As we saw our work concentrated in particularly on this topic.

In addiction, the natural conclusion of our elaboration, as we introduced in the first chapter, is related to the last step of the Robot Learning from Demonstrations/Failures, that is the updating step. This phase is important to tune the model perfectly to the robot that we are using, so it can learn not only from human demonstrations but also from its own errors.

Last but not least, with a future version of the software for the specific robot that we used there will be the possibility to control it directly in position-velocity, so it can leave space for a more accurate mapping of the model output trajectory.

References

[1] S. Michieletto et al. Robot learning by observing humans activities and modeling failures. *IROS workshops: Cognitive Robotics Systems (CRS2013)*, 2013.

[2] D. H. Grollman et al. Donut as i do: Learning from failed demonstrations. *IEEE International Conference on Robotics and Automation*, 2011.

[3] D. H. Grollman et al. Robot learning from failed demonstrations. *International Journal of Social Robotics*, 2012.

[4] S. Michieletto et al. Ros-i interface for comau robots. *Simulation, Modeling, and Programming for Autonomous Robots*, 2014.

[5] E. Tosello et al. A learning from demonstration framework for manipulation tasks. *ISR/Robotik 2014; 41st International Symposium on Robotics*, 2014.

[6] S. Michieletto et al. Learning how to approach industrial robot tasks from natural demonstrations. *IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO2013)*, 2013.

[7] B. D. Argall et al. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 2009.

[8] A. Billard et al. Robot programming by demonstration. *Handbook of Robotics*, 2008.

[9] S. Schaal et al. Learning movement primitives. *Robotics Research*, 2005.

[10] A. Ijspeert et al. Trajectory formation for imitation with nonlinear dynamical systems. *IEEE International Conference on Intelligent Robots and Systems (IROS 2001)*, 2001.

[11] A. Ijspeert et al. Learning attractor landscapes for learning motor primitives. *Advances in Neural Information Processing Systems 15*, 2003.

[12] A. Ijspeert et al. Movement imitation with nonlinear dynamical systems in humanoid robots. *International Conference on Robotics and Automation (ICRA 2002)*, 2002.

[13] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 1999.

[14] S. Schaal. Learning robot control. *The handbook of brain theory and neural networks, 2nd Edition*, 2002.

[15] S. Schaal. Arm and hand movement control. *The handbook of brain theory and neural networks, 2nd Edition*, 2002.

[16] B. Akgun et al. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 2012.

[17] S. Calinon et al. Handling of multiple constraints and motion alternatives in a robot programming by demonstration framework. *Humanoid Robots*, 2009.

[18] T. Inamura et al. Acquiring motion elements for bidirectional computation of motion recognition and generation. *Experimental Robotics VIII*, 2003.

[19] D. Kulic et al. Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden markov chains. *Intl Journal of Robotics Research*, 2008.

[20] L. Rozo et al. Force-based robot learning of pouring skills using parametric hidden markov models. *In Proc. of the IEEE-RAS Intl Workshop on Robot Motion and Control (RoMoCo)*, 2013.

[21] S. Calinon et al. Incremental learning of gestures by imitation in a humanoid robot. *In Proc. ACM/IEEE Intl Conf. On Human-Robot Interaction (HRI)*, 2007.

[22] S. Calinon et al. Statistical learning by imitation of competing constraints in joint space and task space. *Advanced Robotics*, 2009.

[23] P. Kormushev et al. Upper-body kinesthetic teaching of a free-standing humanoid robot. *Proc. of the IEEE Intl Conference on Robotics and Automation (ICRA 2011)*, 2011.

[24] P. Kormushev et al. Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input. *RSJ Advanced Robotics*, 2010.

[25] M. Hersch et al. Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transactions on Robotics*, 2008.

[26] A. C. Cifuentes et al. Human-robot interaction based on wearable imu sensor and laser range finder. *Journal Robotics and Autonomous Systems October*, 2014.

[27] C. Stanton et al. Teleoperation of a humanoid robot using full-body motion capture,example movements, and machine learning. *In Proc. of Australasian Conference on Robotics and Automation*, 2012.

[28] H. Dang et al. Robot learning of everyday object manipulations via human demonstration. *In Proc. of IROS 2010*, 2010.

[29] L. Rozo et al. Robot learning from demonstration in the force domain. *Aliht*, 2011.

[30] M. Munaro et al. An evaluation of 3d motion flow and 3d pose estimation for human action recognition. *RSS Workshops: RGB-D: Advanced Reasoning with Depth Cameras*, 2013.

Thanks

The conclusion of this work marks the end of another important path of my life. It was not always easy and steady, there were a lot of moment where I was uphill, but I am sure that if I could go back in time I would choose it again.

I want to thank all the people that accompanied me over these years, starting from my mum that was always by my side and that allowed me to continue to this route. Thanks to anyone's life put me close, who endured my stubbornness and that allowed me to grow as a person and as a future engineer.

In particular I want to thank the IAS-Lab, my advisor prof. Enrico Pagello and my co-advisor dott. Ing. Stefano Michieletto that allowed me to passionate to the beautiful field that is Autonomous Robotics and that took the time to help me with my work and my curiosity.