Robots, one step ahead of us: Prediction of human movements

Gonçalo Alexandre Mesquita Albuquerque Castilho
Instituto Superior Técnico
goncalocastilho@tecnico.ulisboa.pt

ABSTRACT
The ability of predicting human movements allows for improvements in environments where both humans and robots work. These improvements might be in terms of their direct interaction or in order to avoid getting in the way of a human’s action. This work proposes the development of a system that is able to predict human movements and trajectories both online and offline. The system uses Probabilistic Motor Primitives to represent the learned movements and will use Dynamic Time Warping to scale movements at different speeds and phases from those learned. Furthermore, the system classifies complete movements and incomplete movements (prediction of an unfinished movement). Lastly, the results of the system both online and offline were presented and discussed as well as a comparison with a simple euclidean distance predictor was made, where the system proved to be better than the euclidean distance predictor.

Keywords
Probabilistic Motor Primitives, Prediction, Trajectory, Dynamic Time Warping, Classification of movements

1. INTRODUCTION

More and more humans must, in some manner, interact with robots in their everyday life. Be it either in a work environment, like a factory that has multiple robots manufacturing some sort of product, or in a more personal environment like our homes, where we already have robots that clean our home autonomously. Robots have come to a point where they share the same spaces as humans.

Applying robotics to certain tasks leads to improvements in the overall quality or efficiency of the tasks, but in collaborative tasks between robots and humans the way they interact is crucial for this improvements. To improve these interactions this work aims at developing an intelligent system that is able to learn and perceive how a Human performs in a collaborative task. Using this knowledge the system will then predict the actions of the human.

Prediction belongs in multiple scenarios as a complement to the execution of any task with collaboration, even becoming a need in certain situations.

Two persons performing a collaborative task intrinsically adapt to one another during its execution, making predictions about their partner’s intentions and movements along the way. This is an aspect of collaboration that can greatly improve efficiency in a workspace, as well as confidence between peers. At first glance this appears as something natural and that we do in our everyday life. In reality, this is not that simple. It is not easy to predict an action of a human but, why can’t this be translated to human robot interaction? There is already proven work when it comes to Humans predicting Robot actions, as well as work on making a Robot more predictable [2] [3] but, in this case, the idea is to give a robot the ability to predict human actions using knowledge acquired through interaction, just like two humans would do.

A prediction will be defined as a classification of a type of movement before it’s conclusion. In order to do this, the system will need to learn different types of movements for different targets. Knowing this, it will, using incomplete trajectories, perform the classification of these trajectories and therefore predicting the type of the movement. A problem that exists with predicting incomplete movements is the question of different speeds while executing exactly the same movement and trajectory. Since we don’t know at what percentage of the trajectory we are in then we cannot now for certain how to compare with the learned movements. Not only this but, we also do not know to which phase of the learned trajectory belongs the partial trajectory we receive.

1.1 Contributions

This work contributed with a novel framework that uses Probabilistic Motor Primitives to predict and anticipate the goal of human motions during human-robot interaction. Given a set of potential known targets for the movements of the human, the robot can recognize the movement being performed and predict its goal after observing only a portion of the movement.

This work also contributed with an approach that using the Dynamic Time Warping algorithm can increase robustness to the recognizing of motions with different speeds of execution and that have an unknown phase. This allows the robot to perform the recognition of a movement even if it is being performed at a speed that is different from that it learned. The DTW algorithm can be used with both complete and partial movements.

In Summary:

- Framework based on ProMP that given a set of known targets can predict, after observing an incomplete movement performed by the human, which is the goal target of said movement;
- Approach using the DTW algorithm that allows the system to recognize movements despite them having different speeds of execution in regard to the learned movements;
• Approach using the DTW algorithm that allows for the system to recognize movements without knowing beforehand the phase of the incomplete movement observed

1.2 Outline

The next section, section 2 presents background about the Dynamic Time Warping algorithm and Motion Primitives which helps understand the concepts along the document, as well as discusses related work which uses both of these approaches. Section 3 presents the solution for the prediction system as well as the approaches to solve the problems that were encountered. In Section 4, the results of the system are presented and discussed. Finally, in Section 5 and Section 6 the conclusion, future work and system limitations are presented.

2. RELATED WORK

2.1 Dynamic Time Warping

The Dynamic Time Warping algorithm is a crucial part of the system proposed in this work. In overview, this is an algorithm that given two time series, which may vary in speed, measures the similarity between them. This similarity is measured by calculating an optimal match between the two time series.

The DTW algorithm can be applied to temporal sequences of any type of data be it video, audio, graphics or motion, as is the case of this work, as long as it can be represented as a linear sequence.

Besides the measure of the similarity between two time series, the DTW algorithm also produces a “warping path” from which the two series may be aligned in time.

The DTW algorithm in order to calculate an optimal match between two time series, X and Y, it has to “warp” the sequences non-linearly in time. This “warp” consists in stretching or shortening the one of the sequences in order to find the best match. The best match is calculated in terms of a distance measure between the warped X and the original Y.

In terms of implementation of the DTW algorithm its pseudo-code is presented in algorithm 1. The algorithm receives two time series X, Y of sizes n and m and returns a matrix S of size n x m containing the distances between the pairs of points X[i] and Y[j] being i and j indexes of the vectors X and Y respectively.

The optimal warp path is defined by the path through S in which the cost is minimal (see algorithm 2). To calculate the optimal warp path the algorithm initializes an empty array which will be our warp path. It will start by appending the pair (0,0) corresponding to the starting indexes and will continue to append pairs (i,j) in which i is an index in time series X and j is an index in time series Y. The algorithm traverses the cost matrix S starting from entry (0,0), until entry (n,m). The algorithm will see each neighbor S[i+1,j], S[i, j+1] and S[i+1, j+1] and append to the warp path the pair of indexes which present the minimal value in S. It will then continue from the indexes it appended until i=n and j=m.

In [11] the authors present an algorithm for Dynamic Time Warping on multi-dimensional time series (MD-DTW). With this algorithm the authors use all dimensions to find the best synchronization between time series.

Data: Let X=(a_1,...,a_n), Y=(b_1,...,b_m) be 2 time series of size n and m

//Initialize the two dimensional array S to store the similarity measures of the differences
//between time series X and time series Y
S := array[n, m];
for i := 0 to n do
  for j := 0 to m do
    S[i,j] := ∞;
  end
end
S[0,0] := 0;
// Using a pairwise comparison, incrementally fill the similarity array with the differences between
//both time series. This differences will define the warp path. for i := 0 to n do
for j := 0 to m do
  cost := d(X[i], Y[j]); S[i,j] := cost +
          minimum(S[i-1,j], //increment
                   S[i,j-1], //decrement
                   S[i-1,j-1]); //match
end
end
return S[n,m]

Algorithm 1: Pseudo code for the Dynamic Time Warping algorithm (adapted from [4])

Data: Let S = array[n,m] be the two dimensional array containing the similarity measures of the differences between two time series X[n] and Y[m]

warp_path := [];
i := 0;
j := 0;
warp_path.append((0,0));
while i < n and j < m do
  if minimum( S[i+1, j] , S[i, j+1], S[i+1, j+1] )
  == S[i+1, j] then
    warp_path.append( (i+1, j) );
    i := i+1;
  end

  if minimum( S[i+1, j] , S[i, j+1], S[i+1, j+1] )
  == S[i, j+1] then
    warp_path.append( (i, j+1) );
    j := j+1;
  end

  if minimum( S[i+1, j] , S[i, j+1], S[i+1, j+1] )
  == S[i+1, j+1] then
    warp_path.append( (i+1, j+1) );
    i := i+1;
    j := j+1;
end
return warp_path

Algorithm 2: Pseudo code for the optimal warp path
They use this algorithm for gesture recognition of signs from the standard vocabulary of Sign Language of the Netherlands. These gestures are modeled as multi-dimensional time series where the number of measurements is the dimensionality of the series and the number of time instances its length.

The MD-DTW algorithm is composed of 4 steps. The first is to normalize each dimension of both time-series we want to synchronize separately to a zero mean and unit variance. Then there is an optional step which is smoothing each dimension with a Gaussian filter. Then the algorithm fills the M x N distance matrix, where M and N are the lengths of each time-series, respectively, using the following equation, where K is the number of dimensions:

$$D(i, j) = \sum_{k=1}^{K} |A(i, k) - B(j, k)|$$  \hspace{1cm} (1)

In the end the algorithm used the distance matrix calculated to find the best synchronization with the regular DTW algorithm.

Using the MD-DTW to synchronize movements they were able to classify gestures.

In [8], similar to the goal of this work the authors classify human motion, and use Dynamic Time Warping to solve the problem of the time and speed differences between motion sequences. They focus on automatic classification of an unknown motion sequence using known motion sequences from a database.

The DTW is applied to warp the motions recorded to match against the motions recorded. This use is similar to the approach discussed in section 3.2.2.

This application of the DTW algorithm to solve the problem of trajectories being warped in time and speed was used in [7]. In this work they used ProMP’s to perform action recognition and human-robot movement coordination. ProMP’s need to have all the trajectories aligned in relation to a common time scale in order to work. Using the DTW they were able to align in time the trajectories they receive with the ones used to train the ProMP’s.

In [10] they proposed an alternative to the Dynamic Time Warping algorithm. The purpose behind this work was to present an approximation of the DTW algorithm which has linear time and space complexity. According to the authors, DTW has a quadratic time and space complexity which limits its application to small time series data sets. Having a DTW algorithm which is able to have linear time and space complexity makes it possible to expand its application to experiments with larger time series data sets.

The authors approach to their problem of developing a linear complexity version of the DTW algorithm was done by avoiding the brute-force dynamic programming approach of the DTW algorithm, and instead using a multilevel approach. They first sample the time series to a very low resolution and find a warp path in this lower resolution. Then they project the warp path onto incrementally higher resolution time series continuously until the full warp path for the entire time series is found.

In a more specific manner, the multilevel approach used in this work is decomposed in three key operations, Coarsening, Projection and Refinement.

In the Coarsening, the time series is shrunk in to a smaller time series that represents the same curve as accurately as possible but with fewer data points.

In the Projection, operation a minimum-distance warp path is found at a lower resolution, and then this warp path is used as an initial prediction for a higher resolution’s warp path.

Last, in the Refinement, the warp path obtained at a lower resolution is refined through local adjustments.

They test their algorithm’s accuracy by measuring the error of a warp path which is given by the following equation:

$$ErrorOfWarpPath = \frac{approxDist - optimalDist}{optimalDist} \times 100$$  \hspace{1cm} (2)

They evaluated the FastDTW algorithm against two other existing approximate DTW algorithms, Sakoe-Chuba bands and data abstraction. After their test they could conclude that the FastDTW was better than both of the other two algorithms by a big margin and despite the radius value of the algorithm. The FastDTW achieved a maximum error of 19.2% when the radius was 0 and that reduced to 1.5% with the radius equal to 10.

In terms of execution time, the FastDTW is proven faster than the DTW algorithm.

2.2 Motor Primitives

Motor Primitives are the basis behind the approach used in this thesis. In this work we use Probabilistic Motor Primitives in order to learn human movements in the form of Motor Primitives.

To address how these form a good approach to learn movements we will talk about some related work about Motor Primitives in this section.

In [6], the authors goal is to find reinforcement learning techniques that can be applied to Motor Primitives in the context of learning high-dimensional motor control tasks. The motor primitives are used in policy search for their “PoWER” algorithm. They describe their motor primitive framework as two coupled differential equations where they have a canonical system \( \ddot{y} = f(y, z) \) with movement phase \( y \) and possible external coupling to \( z \), as well as a nonlinear system \( \ddot{x} = g(x, \dot{x}, y, \theta) \) which yields the current action for that system. Both systems are made to have the right properties in order to be useful for the desired class of motor control problems. One big advantage that is addressed by the authors is that the linearity of the function \( g \) enables Motor Primitives to be well-suited for imitation learning and reinforcement learning algorithms.

To perform the experiments to benchmark their "PoWER" system the authors had to learn Motion Primitives of the movements needed for certain actions, one of which was the Ball-in-a-Cup game. Their system state was described in terms of joint angles and velocity of the robot and Cartesian coordinates of the ball. The actions were joint space accelerations where each of their seven joints was represented by a motor primitive.

The authors in [5] used attractor landscapes to learn motor primitives of rhythmic movements.

They demonstrated that using attractor landscapes they were able to learn motor primitives for various movement skills using imitation learning. This was done by fitting joint-angle trajectories with control polices, with one control policy per degree of freedom. Then they could use the control policies to replay the learned movements in a hu-
manoid robot using an inverse dynamics controller to track the trajectories generated by the control policies.

In [12], the authors have as their goal the generalization of example trajectories to new situations not observed during training. To achieve this they needed a representation suitable for robot control. They represented every trajectory with parameters that described characteristics of the task as, for example, the position of the goal target and the positions of the points of a trajectory.

The approach presented in this paper uses Dynamic Movement Primitives as their motor representation. This enables the generalization of DMP’s to new situations using the goal of the task and the training set of movements. Since the generalized trajectories are encoded as DMP’s it is possible to apply modifications to the dynamic system in order to account for unforeseen perturbations, such as obstacles, which may not have been a part of the training set. The use of DMP’s makes it possible to also specify constraints and weights to the generalized movements with their respect to the importance of the task. All of the training data can be obtained either by kinesthetic guiding or from human demonstration, as how it is done in my work.

Tests on discrete movements of reaching, grasping, and ball-throwing. They were able to conclude that their approach was suitable if the example trajectories smoothly transitioned as a function of query points. They also concluded that the generalization of the trajectories using DMP’s allowed for the exploit of the advantages I described earlier of modifying and conditioning a generated generalized movements, given any unforeseen perturbation before execution time.

As we have seen in the former related work, Motor Primitives can be used as a representation for a variety of types of movements and applied to various different algorithms. Daniel et al. [1] adapted the children’s game of Thether ball for a robot application. In their problem the robot had to adapt to a varying position of the ball without being explicitly programmed for that. Therefore it had to have the ability to adapt its movement given the conditions of its environment.

Their approach extends the Relative Entropy Policy Search algorithm to the hierarchical policy case. They apply this algorithm in combination with Dynamic Movement Primitives. At the same time, they learn a gating network, which selects between primitives given the current context, and the policies of the primitives, which specify the robot’s actual actions.

Each single DMP represents a movement plan for an episode, in an episodic learning case for motor skills, applied for many single-stroke motor skills. A DMP uses a second order linear dynamical system which is modulated by a learnable non-linear function \( f(z; w) \). where \( z \) is a phase variable of the movement. The function \( f(z; w) = \Phi(z)^T w \) is non-linear in the phase variable \( z \) but linear in its parameters \( w \). The parameters \( w \) correspond to the weights and these define the shape of the movement and are learned through imitation learning. Each joint has a different learnable function \( f \) and therefore has different parameters \( w \).

Given the related work presented so far we can conclude that Motor Primitives are a representation of movements that are adaptable to many algorithm’s and techniques. Not also is it very adaptable, it is flexible as it allows for a system to manipulate the learned movements in order to adapt to unforeseen obstacles or constraints when reproducing the movements. The results of using Motor Primitives in works that learn movements or trajectories were also positive since the papers researched proved to have been able to effectively learn the movements and even generalize, reproduce a new movement from the learned one and also classify new movements. Therefore we believe this is a good approach to learn the movements in this work.

3. PREDICTION OF MOVEMENTS

In this work, the core problem being addressed is the challenge of predicting the goal of a human movement. These predictions are done based on movements performed by humans while knowing a set of possible goals and trajectories performed to reach these goals. The system performs predictions based on incomplete movements, performing this prediction before the movement has reached its goal.

To perform a prediction, first the system has to be able to recognize and classify the type of an observed movement from those learned. Having recognized and classified the movement then the system performs the prediction of the goal of the movement.

These predictions are robust to differences in the speed of the movements observed, that is to say that, the speed of the observed movements does not need to be the same as the learned movements. The system is also robust to observing different phases of the movement, as for example, observing a movement that corresponds not to the beginning of a learned movement but instead to a motion in between the full movement from the learned starting position to the goal position.

In the following sections the solution approach will be explained. To learn the human movements Probabilistic Movement Primitives where used. To recognize and classify the movements we used Multivariate Normal Distributions. In order to bring robustness to speed and phase differences we used the Dynamic Time Warping algorithm.

3.1 Learning Human Movements

Many authors use Motor Primitives to allow robots to learn movements[6][5][12][1]. In this work a probabilistic approach of the MP concept was used, that maintains a distribution over trajectories[9].

The goal behind the use of Probabilistic Movement Primitives is to be able to learn a representation of a human’s movement. This is performed using multiple demonstrations of a certain class of movements.

For the purposes of following the usual nomenclature in the bibliography we will refer to each joint or Cartesian coordinates of a movement as a degree of freedom (DOF).

We will denote \( Y(t) = \{(y_1(t), y_2(t), \ldots, y_n(t))\}_{t=0,\ldots,T} \) as the trajectory of a movement having \( n \) DOF’s and where \( y_1(t) \) is the position of DOF \( 1 \) in the time instant \( t \). We will use a linear regression with \( m \) Gaussian basis functions \( \psi \).

We can then represent the state vector \( Y(t) \) as:

\[
Y(t) = \{(y_1(t), y_2(t), \ldots, y_n(t))\}_{t=0,\ldots,T} = \{(\psi_1(t), \psi_2(t), \ldots, \psi_m(t))\}_{t=0,\ldots,T} \ast w + \epsilon_y,
\]

where \( \psi_m(t) \) is a \( m \) dimensional time-dependent basis vector, \( \epsilon_y \) is a zero-mean i.i.d. Gaussian noise and \( w \) is a matrix
3.2 Classifying a new movement

3.2.1 Dealing with movements at different speeds and phases

When performing a task with a set of possible movements a person performs each movement at his own pace, defining its own speed for the execution of the movement. When two persons perform the same type of movement and we compare the two movements it is very common that, assuming the trajectories are exactly the same, the speed of the execution differs. This can make a time step by time step comparison impossible without first scaling the movements to the same scale.

But even having every movement in the same scale we still have a second problem of knowing the phase of the movement for incomplete movements. Because the robot can’t know exactly when a human begins a movement it may not always receive the reading of a partial movement from its beginning, which then will introduce the problem of not knowing which phase of the movement this reading belongs to.

In order to deal with the first problem, the Dynamic Time Warping algorithm was used to scale the perceived movement to the same scale of the learning data set used to train the system. Using this we can obtain a scaled version of the perceived movement and then classify the movement from the scaled version.

For the second problem, the solution became easier given that it was possible to obtain the scaled version of the movement and therefore, speed was not an issue anymore. Having every movement on the same scale gives the possibility of comparing the movements time step by time step. To find the phase of the movement a sort of rolling window algorithm is applied, in which we place the movement over the learned trajectory and slide until an optimal match is found.

3.2.2 Using the Dynamic Time Warping for the scaling

As was discussed in 3.2.1 Dynamic Time Warping is used in this system to scale a movement into the scale of the learned movements. The DTW algorithm receives both the movement the human executed and a learned trajectory and tries to obtain an optimal alignment between the movement and the learned trajectory.

As was explained in section 2.1, the DTW algorithm is used to compare two temporal sequences (which may vary in speed) in terms of their similarity, as well as giving an error measurement on the basis of the match between both sequences. For the purposes of this experiment the algorithm will compare two human movements. In a more general approach, the DTW is a method that calculates an optimal alignment between the two given temporal sequences. [10] [8]

The optimal alignment is done by “warping” non-linearly one of the sequences by stretching or compacting it along its time axis to determine a measure of their similarity. Their similarity is measured in terms of the distance between one temporal sequence and the warped versions of the other. The algorithm then aims to find the minimum-distance warp path [figure 3] that can be constructed from the two time series.

To solve the problem of the scaling of the trajectory in this thesis, the cost matrix was used to construct a scaled version of the received movement. This is done by creating a vector which will be populated by the values of the received movement but at the scale of the learned movement.

The DTW algorithm is run having as arguments a learned trajectory and an observed trajectory. The distance measure used in this system to calculate the similarity between both trajectories is the euclidean distance. The DTW algorithm used is the ”FastDTW” [10] algorithm in Python which can be obtained in https://pypi.python.org/pypi/fastdtw. The
algorithm returns two arguments, a cost matrix which we referred earlier as being used to construct the scaled version of a received movement, and also returns a distance which corresponds to the similarity measure of the algorithm. This distance will be later used in the calculation of the probability of the movement belonging to a class using the multivariate normal distributions.

Let’s define $Y$ as the learned trajectory, $X$ as the new unscaled trajectory, $j$ and $i$ being indexes of each of the trajectory vectors respectively and $D[j, i]$ as a pair in the warp path cost matrix. The warp path is traversed through each pair $D[j, i]$ storing the previous value of $Y[j]$ and $X[i]$. Each time there exists a "jump", which consists of both $j$ and $i$ having changed in value since the last stored value, then the value $X[i]$ is appended to the vector of the scaled trajectory. This is repeated until there are no more pairs in the warp path. In the end the new vector will contain a scaled version of the trajectory received.

### 3.2.3 Using Multivariate Normal Distributions for the classification

Having learned the MP’s of different classes of human movements we can then begin to classify new movements given their trajectories.

To do this, the learned MP’s were approximated to multivariate normal distributions (MVND). Since the movements learned are depicted in 3 DOF’s, the movement is modeled in 3 multivariate normal distributions, one for each DOF.

The MVND of a k-dimensional random vector $x = [X_1, X_2, ..., X_k]^T$ can be written in the following notation:

$$x \sim N_k(\mu, \Sigma)$$

with k-dimensional mean vector $\mu$ and $k \times k$ covariance matrix $\Sigma$:

$$\mu = E[x] = [E[X_1], E[X_2], ..., E[X_k]]^T,$$

$$\Sigma =: E[(X - \mu)(X - \mu)^T] = [\text{Cov}[X_i, X_j]; 1 \leq i, j \leq k].$$

Using this type of distributions, the probability of a movement belonging to a certain class of movements is calculated using the distance between the mean trajectory of a learned movement and the new movement with the following:

$$P(\text{DOF}) = e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

Having calculated the probability for each of the DOF’s for a certain class of movements, the average probability is calculated and compared with the average for every other learned class. The class with the highest probability is the class to which it is assumed the new movement belongs.

But in the case in which the movement is to different from any learned movement, and therefore has a low probability...
Figure 4: Example of the rescaling with the DTW. In this example the red trajectory corresponds to the learned trajectory and the blue trajectory corresponds to a partial version of the learned trajectory but executed at half the speed. The green trajectory corresponds to the rescaling after applying the DTW.

In the best class, we compare the probability with an empirically defined threshold. This comparison is to avoid classifying movements without enough certainty. In the case the probability is lower than the threshold, the algorithm does not return any class, but instead returns a null object.

Due to the DTW’s ability to rescale movements given a learned movement, it might happen in some cases that the rescaled version fits a learned movement that is in fact different from the observed movement and therefore the calculation of the probability could return a high probability for a class despite it not being the correct class. In order to avoid these mismatches in classification, the distance [section 3.2.2] output returned from the DTW algorithm is used to refine the classification. Having \( p_a \) and \( p_b \) as the probability of the observed movement being of class \( a \) or \( b \), respectively, and assuming \( p_a \) is the best possible class seen so far with a DTW distance of \( dist_a \) then the decision of the movement being of class \( a \) or \( b \) will be done with the following in algorithm 3:

Using algorithm 3 we are able to avoid misclassifications due to a rescale of a movement with a low similarity comparatively to the learned movement.

**3.2.4 Determining the phase of partial movements using a rolling window**

In sections 3.2.2 and 3.2.3 it was discussed how DTW is used to scale movements being performed at different speeds, and was also discussed how the classification of the movements was made using multivariate normal distributions.

When it comes to incomplete movements, in which the system has only seen a partial movement by the human, the multivariate normal distributions can not be applied directly. First, the movement has to be scaled to the learned movement’s scale using the DTW. The DTW can scale even incomplete movements as is shown in figure 4. After having scaled the movement, the probability of the the movement belonging to the distribution of the learned movement is calculated, but because the phase of the movement is unknown this probability can not be calculated directly. To solve this problem we calculate the probability using a rolling window. This rolling window consists in sliding the partial movement over the time axis in order to find the best fit, which is the fit that returns the highest probability [figure 5]. Having the highest probability we then apply all of these methods to the other learned trajectories and compare the probabilities to return a classification of the partial movement.

The DTW can also calculate the phase of the movement but in movements which are cyclic this could lead to errors. For example, if we have a movement that forms a trajectory, like a cosine function, then the trajectory will repeat itself. If we only see a portion of the trajectory it will match with any of the cycles of the cosine function, but since this is applied to a multi dimensional scenario, if this cosine function only refers to one of multiple DOF’s it could have a mis-

**Data:** Let \( Y \) and \( X \) be the learned movement and observed movement respectively and \( bestClass = a \), \( bestDist = dist_a \) and \( bestProb = P_a \). \( currentDist \), \( rescaledMovement := DTW(Y,X) \) \( currentProb := P_b \) \( currentClass := b \) \( percentageTolerance := (bestDist - currentDist) / bestDist \)

**Algorithm 3:** Pseudo code for the selection of the best class

**Figure 5:** Example of the rolling window trying to find the best match for the blue trajectory. In this case the algorithm will see the right box as a better match than the left and center box because the right one will return a higher probability when applying the Multivariate Normal Distributions.
match with the phase returned by the DTW respective to another DOF where the trajectory is non-cyclic. In order to avoid this we use the rolling window and find the best match having all of the DOF’s in consideration at the same time.

4. EXPERIMENTS AND RESULTS

To test the system with real trajectory data, an OptiTrack (http://www.optitrack.com/) system was used. The setup of the system included 6 cameras mounted on the ceiling in a closed office. The motion data was captured using a ROS (Robot Operating System) node.

The OptiTrack system is a system capable of recording 3D motion data. This is done by using multiple synchronized cameras installed around the target, or targets, in which, each camera captures 2D images and then using triangulation and the images captured, a 3D position the target(s) is computed.

One of the main perks of using the OptiTrack for this work is its capability of capturing all types of human body and rigid body movements, also including subtle and fast motions. Not only can it track rigid body movements, it can also track props along with skeleton data, which is required due to the tools and items that will be present in the experiment. The OptiTrack works with its own software, the Motive, as well as a camera SDK. This system also supports the use of markers which can be useful to mark specific objects and landmarks in the environment. For the duration of the tests performed a marker attached to the wrist of the human was used.

4.1 How the trajectories are captured

For the offline and online tests, with the exception of the test with synthetic trajectories, all of the trajectories used are recorded from real human movements.

The setup used to record these trajectories consisted in a work area where 6 OptiTrack cameras formed the capture area for the movements. The movements were done on top of a table which had 5 points marked, all of which are visible in figure 6. The five points were S, Box, LT, LB and BC. All of the recorded trajectories had the common starting point, point S. We performed 4 types of trajectories in which the user would move its hand from point S to one of the 4 other points, S to LT, S to Box, S to LB and S to BC. For each training set the system received the same amount of trajectories for each type of trajectory in order to avoid differences in sample complexity between the types of trajectories. We recorded a total of 1116 trajectories performed from a total of 4 different persons. The trajectories used for the training of the system were scaled in order to have the same time length. All other recorded trajectories that were used for training were not scaled.

We recorded 2 training sets and one test set. The 2 training sets are, one with 800 trajectories, having 200 trajectories of each type and the other one contains 200 trajectories, 50 of each type. These two different training sets differ mainly in sample complexity and were used to test the performance differences caused by sample complexity. The learning set consisted in 116 trajectories, 27 for S to Box, 30 for S to BC, 29 for S to LT and 30 for S to LB. The number of test trajectories for each type of trajectory is not the same because after the running of the tests and results it was found that 3 S to Box and 1 S to LT trajectories where recorded incorrectly and therefore discarded from the test set.

For the Online Tests the trajectories were captured using the same setup but in real time. The OptiTrack system captured a point every 0.01 seconds and sends it to the predictor which appends it to the observed trajectory vector. It then performs a prediction given what it has seen until the moment. The system resets its observed trajectory vector each time it does not record any significant movement (a difference of 5 cm) after 0.3 seconds. During the Online Tests the system cannot capture any information of the percentage of the complete trajectory it has observed. Due to this limitation the online tests does not feature the results of the predictions at a given trajectory percentage as is done in the offline tests.

4.2 Synthetic trajectory Tests

To test the performance of the system in a scenario with less noise than the real scenario tests with synthetic trajectories were performed. Those trajectories received perturbations in order to generate different trajectories for the training and also for the test trajectories. This was done by applying random noise to the training trajectories. For the test trajectories the random noise could be 3 times more than on the training set, to generate the possibility of generating trajectories that were significantly different from the training set.

In table 1 the results of the test are shown.

For the test the system was trained with a total of 1200 trajectories, 200 for each type of trajectory. The test was performed using 200 trajectories, 40 of each type. To test the system’s performance with partial trajectories the 200 trajectories were cut in order to feed the system partial versions of the trajectories.

The results show that for a case in which the noise of the learning and test set is very low the system is able to, despite the percentage of trajectory observed, perform predictions with an average accuracy of 93.8%. This accuracy corresponds to the percentage of correct predictions in a given test set and a given percentage of trajectory. Having these high results in terms of accuracy rate with this scenario led us to conclude that the system is performing correctly the predictions, which allow us to perform the tests with real trajectories with confidence.

We can also conclude that the capability of performing a correct prediction is not the same for every type of trajectory. We can see in table 1 that Trajectory 1 is the one with the lowest correct number of predictions for the trajectories of that type and that for example, for Trajectory 3 the system always predicts correctly the trajectory. This is due to the DTW being able to rescale an observed trajectory to a scale in which it returns a high probability of matching with the MVND. Since the trajectories of type Trajectory 3 are the most distinct from the other 4 trajectories (even though using the rescale of the DTW), since the noise applied to the trajectories is very low the trajectories of type Trajectory 3 remain always very similar to the learned trajectories and therefore the MVND always returns a higher probability for this Trajectory, in relation to the other 4 types which explains the high accuracy rate for Trajectory 3.

4.3 Offline Tests

In this section the results of the prediction system in an
Figure 6: Test setup. In the image we can see the 4 targets used for the movements, LT, LB, Box and BC. All the movements start at point S. Labeled with 1 is the Optitrack marker used. It is attached to a wristband that will be placed on the user’s wrist.

<table>
<thead>
<tr>
<th>Percentage of trajectory</th>
<th>Trajectory 1</th>
<th>Trajectory 2</th>
<th>Trajectory 3</th>
<th>Trajectory 4</th>
<th>Trajectory 5</th>
<th>Total accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>100%</td>
</tr>
<tr>
<td>90%</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>100%</td>
</tr>
<tr>
<td>80%</td>
<td>31/40</td>
<td>35/40</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>93%</td>
</tr>
<tr>
<td>70%</td>
<td>33/40</td>
<td>35/40</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>94%</td>
</tr>
<tr>
<td>60%</td>
<td>31/40</td>
<td>37/40</td>
<td>40/40</td>
<td>30/40</td>
<td>40/40</td>
<td>94%</td>
</tr>
<tr>
<td>50%</td>
<td>28/40</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>34/40</td>
<td>91%</td>
</tr>
<tr>
<td>40%</td>
<td>28/40</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>23/40</td>
<td>84.5%</td>
</tr>
<tr>
<td>30%</td>
<td>28/40</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>40/40</td>
<td>94%</td>
</tr>
</tbody>
</table>

Table 1: Synthetic trajectory test results. In the column Total accuracy are presented the results for the percentage of correct predictions given all of the test trajectories seen for a certain percentage of trajectory observed (column percentage of trajectory).

The offline test had the system learning 4 different possible trajectories, as was discussed in section 4.1, being so that 2 pairs of the trajectories had similar initial trajectories in order to create a challenge for the system.

The training set and test set are disjoint. The training set consists of 800 trajectories, 200 for each learned trajectory. The test set consists of 116 complete trajectories with 30 for each learned trajectory. To test the accuracy of the algorithm while receiving partial trajectories what was done was a cut to the test trajectories, related to the percentage of the trajectory that would be tested. This way using only the 116 trajectories it was possible to test the system’s accuracy while receiving 100%, 90%, 80%, 70%, 60%, 50%, 40% and 30% of a trajectory.

In table 2 the results are shown:

Analyzing the results we can see that the system is able to perform predictions with a fairly high accuracy rate of 71.6% when the percentage of the observed trajectory is of 60%. Looking at the overall accuracy of the predictions performed by the system in the interval of 60% to 70% of the trajectory complete we can consider that having 71.6% and 89.7% falls in the expectation that using the ProMP’s we are able to perform early prediction with a good accuracy rate.

Looking at the performance of the Dynamic Time Warping algorithm, it allowed for the algorithm to perform predictions on incomplete movements as the system was able to receive incomplete trajectories and still classify those tra-
<table>
<thead>
<tr>
<th>Percentage of trajectory</th>
<th>Accuracy for each type of trajectory</th>
<th>Total accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S to Box</td>
<td>S to BC</td>
</tr>
<tr>
<td>100%</td>
<td>27/27</td>
<td>30/30</td>
</tr>
<tr>
<td>90%</td>
<td>26/27</td>
<td>27/30</td>
</tr>
<tr>
<td>80%</td>
<td>25/27</td>
<td>27/30</td>
</tr>
<tr>
<td>70%</td>
<td>26/27</td>
<td>28/30</td>
</tr>
<tr>
<td>60%</td>
<td>26/27</td>
<td>5/30</td>
</tr>
<tr>
<td>50%</td>
<td>8/27</td>
<td>0/30</td>
</tr>
<tr>
<td>40%</td>
<td>0/27</td>
<td>0/30</td>
</tr>
<tr>
<td>30%</td>
<td>0/27</td>
<td>0/30</td>
</tr>
</tbody>
</table>

Table 2: Offline test results. In the column Total accuracy are presented the results for the percentage of correct predictions given all of the test trajectories seen for a certain percentage of trajectory observed (column percentage of trajectory).

We can also see that the capability of performing a correct prediction is not the same for every type of trajectory, as what was seen and discussed in section 4.2. In this offline test, the system at 60% of the observed trajectory only classifies correctly 5 out of 30 tested movements for the S to BC trajectory while doing only 8 wrong classifications in the rest of the other 86 tests for the other types of trajectories. For S to BC the initial part of the learned movements are similar to S to Box (see figure 9 in the appendix), which given only a partial observation of the movements can lead to mistakes in the classification as the MVND for S to Box and S to BC will both return a high probability and the distance outputted by the DTW will not be very different between both types of movements.

When the percentage of the trajectory falls below 60% the system becomes somewhat inaccurate, having around 30% accuracy when the trajectory is ≤40% and only 44.8% of accuracy at 50% of the trajectory. Although this shows that the algorithm is unable to predict with confidence when the observed trajectory is at ≤50%, this is understandable as the pairs of trajectories S to Box - S to BC and S to LT - S to LB share similar initial trajectories which as was explained above, will return similar probabilities from the MVND and similar distances from the DTW.

4.4 Comparison with an euclidean distance predictor

In order to evaluate the system against a heuristic based predictor tests with an euclidean distance predictor were performed. This predictor, given the last position seen predicts the possible target given the distance to it.

The test was done using the same test data set used in the offline tests.

We can see from table 3 that the euclidean distance predictor is worst than the predictor using the DTW and the ProMP’s. This can be explained by the fact that the euclidean distance predictor only takes into account the current position of the hand and not the trajectory it took for the form of the trajectory. Since some of the trajectories of the test data are not direct (they do not form a trajectory that follows the shortest possible path from the starting point to the target point) and instead form a curvy trajectory, using only the euclidean distance to the learned points will not give a correct answer with a relatively confident margin until the trajectory is reaching its end (80% or more of the trajectory).

4.5 Performance differences caused by sample complexity

To evaluate the sample complexity of the system the first tests were performed with a reduced amount of training trajectories. In the version of the final tests there were 800 total trajectories (200 for each type of trajectory). In this reduced version the system learned from only 200 trajectories (50 for each type of trajectory) which results in 75% less trajectories than the final version.

In table 4 are the results of the test and in figure 8 the evolution of the accuracy of the system with the two different sample complexities is plotted.

Looking at the results obtained in table 2 and comparing them with the results in table 4 it can concluded that the version with 800 learning trajectories outperforms the version with 200 learning trajectories overall. It is also possible to conclude that below 50% of the observed trajectory the sample complexity becomes less important as both versions have similar accuracy rates. Nevertheless, there exists a big gap between the two versions in the interval of 90% to 60% of the observed trajectory, which also coincides with where the system is most accurate.

Given these results, the conclusion is that the system is
Percentage of trajectory | Accuracy for each type of trajectory | Total accuracy |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>27/27 30/30 29/29 30/30</td>
<td>100%</td>
</tr>
<tr>
<td>90%</td>
<td>27/27 27/30 26/29 28/30</td>
<td>93.1%</td>
</tr>
<tr>
<td>80%</td>
<td>20/27 26/30 24/29 14/30</td>
<td>72.4%</td>
</tr>
<tr>
<td>70%</td>
<td>15/27 22/30 24/29 10/30</td>
<td>61.2%</td>
</tr>
<tr>
<td>60%</td>
<td>15/27 15/30 0/29 14/30</td>
<td>37.9%</td>
</tr>
</tbody>
</table>

Table 3: Offline test results of the euclidean distance predictor

Percentage of trajectory | Accuracy for each type of trajectory | Total accuracy |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>22/27 30/30 29/29 29/30</td>
<td>100%</td>
</tr>
<tr>
<td>90%</td>
<td>17/27 30/30 28/29 29/30</td>
<td>94.8%</td>
</tr>
<tr>
<td>80%</td>
<td>11/27 29/30 29/29 26/30</td>
<td>81.9%</td>
</tr>
<tr>
<td>70%</td>
<td>0/27 29/30 28/29 1/30</td>
<td>50%</td>
</tr>
<tr>
<td>60%</td>
<td>0/27 7/30 26/29 13/30</td>
<td>39.7%</td>
</tr>
<tr>
<td>50%</td>
<td>0/27 0/30 13/29 25/30</td>
<td>32.8%</td>
</tr>
<tr>
<td>40%</td>
<td>0/27 0/30 22/29 14/30</td>
<td>31.0%</td>
</tr>
</tbody>
</table>

Table 4: Offline test results with only 200 trajectories in the learning set

Figure 8: Graphic representing the evolution over the percentage of the trajectory that is given to the system. In blue we have the system with 800 learned trajectories and in red we have the system with 200 learned trajectories.

4.6 Online Tests

In this section the results obtained in an online state will be presented and discussed. The online tests were performed using movements captured in real time by the Optitrack system with the marker visible in figure 6. These movements were all performed from the same starting point, point S. 4 different types of movements where performed, S to Box, S to BC, S to LT and S to LB. All of the positions are visible and marked in figure 6.

In these tests the system has no information about the percentage of the full trajectory captured in any moment, as well as any dimension of the movement vectors captured by the Optitrack system is not controlled or set. Therefore the system has to rely on its ability to scale the captured movements to the same size as the ones in the training data set and also it has to detect the phase of the movements automatically.

The system is able to classify any of the 4 movement types plus a “None” label that is returned whenever the system does not have enough certainty of its prediction to perform a prediction.

To perform the test 200 trajectories were performed, 50 of each type. The system is able to perform one or more predictions during a movement. Due to the prediction being performed asynchronously (to avoid not capturing any position with the Optitrack) the system takes a variable time to perform a single prediction. So, for the same type of movement and the same duration of time the system may perform a different number of predictions along the movement until its conclusion.

We define a prediction for this test as a classification of the type of trajectory seen until any given time or percentage of observed trajectory.

The confusion matrix for this test is presented in table 5. In this confusion matrix are presented all of the predictions performed along each of the 200 trajectories performed. These predictions were not all performed with the same percentage of observed trajectory. All of the predictions were made before the conclusion of the movement and after the reading of the first 3 points of the trajectory by the Optitrack. It is possible to confirm that the number of true positives is significantly larger than the false positives in every type of movement.

The system was unable to predict a type of movement a total of 28 times in 637 predictions. All of these predictions happened in the beginning of the movement, where due to a lack of points captured, or a not yet defined trajectory the system was unable to classify the trajectory observed with enough certainty. This certainty was defined empirically using a threshold variable for which the system had to have a
certain probability of a trajectory being of a given type for it to give a prediction. This variable was set to 0.8 in which the MVND had to return a probability of more than 80% for the value to be accepted.

In S to LT and S to LB movement types, respectively, there is a more common misclassified movement. In S to LT the majority of misclassified movements are of type S to Box, and in S to LB the majority of misclassified movements are of type S to LT. This occurs because the initial predictions are made without enough data points to perform a correct prediction with high confidence, also, the beginning of the movement is common or similar. For example, the S to LB movement shares a similar initial trajectory with S to LT which generates both a high probability of the movement being of the type S to LT or S to LB. This initial prediction inaccuracy is coherent with the results obtained in the Offline tests and is visible in the Online test results in table 6 as well.

The precision for S to LT and S to LB are the lowest and fall below 70%. This can be explained due to the similarity between both movements, which can lead to uncertainty in the early predictions of both movements. In the case of the S to BC movements, the precision is of 94.1% which is by far the highest prediction. This value is very high because although it shares some points of the average trajectory with S to Box it’s trajectories are more direct and therefore easier for the system to predict. This results show the same limitation shown in the offline tests where we saw that the system had problems predicting partial trajectories that are very similar to more than one learned trajectory.

5. CONCLUSION

This work addressed the problem of predicting the actions of a human. To solve this problem a solution using Probabilistic Motor Primitives was proposed, that could later be applied to a robotic system. This solution was divided in two main sections. First the learning of the human movements and second the classification of a new movement.

To classify a new movement it was necessary to deal with movements at different speeds and phases, as well as partial movements. This was solved by using the Dynamic Time Warping algorithm for the speed and a rolling window for the phase. This proved capable of solving the problem both for real and synthetic data.

To perform the classification, Multivariate Normal Distributions were used. These were able to give a probability of a given trajectory belonging to a given distribution. Using this probability it was possible to infer on which class the trajectory belonged.

To benchmark the system’s capability to perform predictions, 4 types of tests were done: Synthetic trajectory tests, Offline Tests, Online Tests and a comparison with an euclidean distance predictor. It was also possible to conclude that the system’s performance is sensible to different sample complexities in which, the higher the complexity the better.

Last, it was concluded that the system outperforms a heuristic predictors based on the euclidean distance to a target.

6. SYSTEM LIMITATIONS AND FUTURE WORK

The future work for this prediction system starts by creating a scenario with a robot using this system and a human to see if there are significant improvements to the efficiency of a task with this addition. Also with this scenario, measuring the humans trust and confidence in the robot, knowing it has the ability to make predictions could lead to further conclusions. Furthermore it would be interesting to test the system in a scenario where the robot doesn’t work as an assistant to the human but instead as its partner in an assembly task having both the same tasks.
In terms of limitations of the system its major limitation is that early predictions have a low accuracy rate which makes it difficult to decide when the robot should accept the prediction.

7. ACKNOWLEDGMENTS

I would like to thank my advisors, Professor Francisco Melo and Professor Manuel Lopes for their patience, guidance and counsel throughout the development of my thesis. Their support was very important to keep me on track and with hope to finish my work.

8. REFERENCES


APPENDIX

Figure 9: In blue we have the learned trajectory S to BC, used in the tests. In green we have the learned trajectory S to Box used in the tests. In red we have an example of one of the test trajectories of type S to BC.