Semester Project Master-2

Classification of human vs hard contacts through force sensing

Spring semester : LASA-EPFL

Author:
Hédi Fendri

Supervisors:
Mahdi Khoramshahi
Prof.Aude Billard

December 7, 2020

Contents

1 Introduction and problem statement

Advancement in the field of robotics has transformed the industrial sector. The annual world supply of industrial robots is increasing significantly as shown in Figure 1.1. Traditionally, robots were mainly used in large production lines in the industry, however, more robots appear in smaller production lines, social and service applications. In such domains, roboticists aim at physical cooperation and collaboration between humans and robots. Beside having a seamless and effective interaction, safety becomes the most significant factor in order to ensure that human beings will not be harmed by their robot partners.

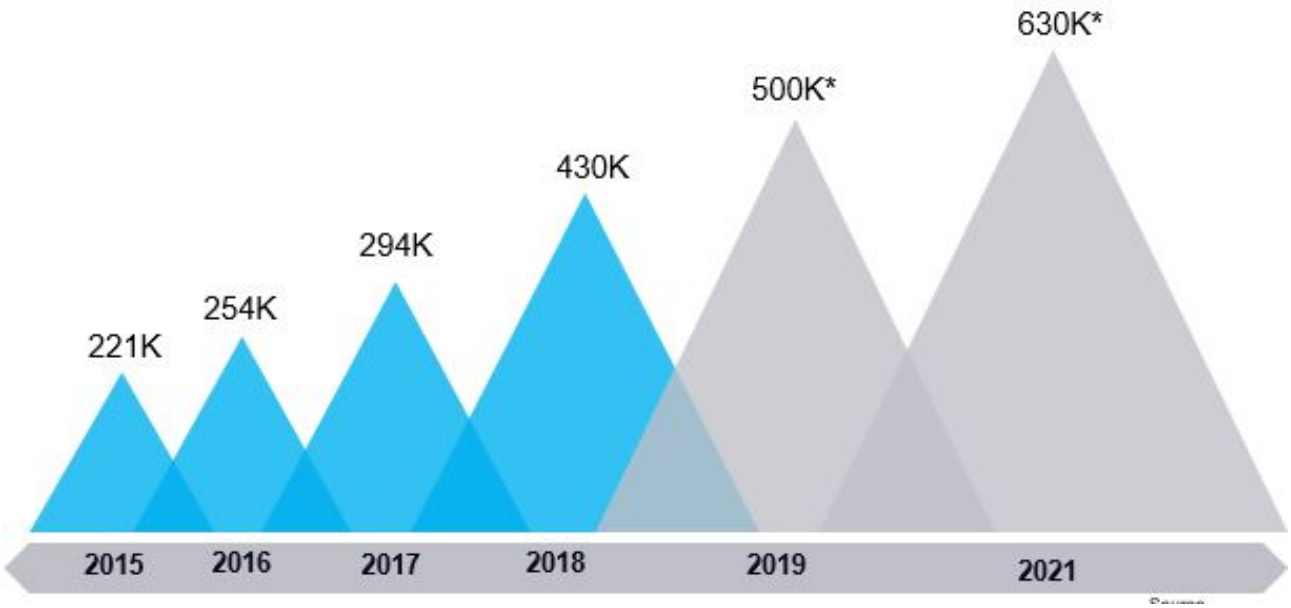

Figure 1.1: Annual worldwide supply of industrial robots between 2009- 2018 and projection between 2018-2021 [1].

For an effective and seamless physical interaction, robots should reduce fatigue and stress, increase human capabilities in terms of force, speed, and precision, and improve in general the quality of life. On the other hand, a human can bring experience, global knowledge, and understanding of the correct execution of tasks. The challenge for robots in this type of application is to understand the intention of the human partners and react accordingly. This communication needs to be in an intuitive way without needing to learn technical details for multiple hours. For instance, the robot adapts its motion according to the interaction forces of the human user. This approach introduces challenges in scenarios when the robot expect interaction forces also from hard surfaces in the environment. Therefore, it is crucial for robot to distinguish between interaction forces of humans and hard surfaces.

In this work, we consider robotic tasks involving contacts with both hard surfaces and humans. For example, consider polishing a surface or drilling an object. The most common approach to solve such a problem is to rely on force-sensing while approaching the surface. The contact is recognized when the sensed forces are passed beyond a threshold. Therefore, the robot begins to exert the desired forces. However, considering cluttered and uncertain environments, including humans, introduce new challenges for the safety of such robotic task. For example, the robot might collide with a human and detects him/her as the expected hard contact. Therefore, following the same strategy, the robots harm the human applying its desired forces. An effective strategy would be to distinguish between hard contact surfaces and accidental contacts with humans. We formulate this as a classification problem in this project.

We start first with a quick overview for related works from the literature in the section 2. We present our method in section 3 followed by the results on simulation and on real robot data in sections 4 and 5. We will also show how can we integrate our proposed method as a feedback to the control loop of the robot in section 6. We finish with conclusion and future work in section 7.

2 Backgrounds and related works

In this chapter, we provide a quick overview for the background and related works. We present the theoretical foundations behind the project from introducing the dynamic system for robotic manipulations 2.1 to estimating parameters using MIT rule 2.3.

2.1 Dynamic system model for robotic manipulations

Let us begin with the dynamic model of the robot we want to control and manipulate in general. The general dynamics of N degrees of freedom (DOF) robotic manipulators is the following [2] :

$$M(q)\ddot{q} + B(q,\dot{q})\dot{q} + G(q) = u_i + u_{env} \tag{2.1}$$

where:

- $q \in \mathbf{R}^N$ represents the joint configuration of the robot end effector.
- $u_i \in \mathbf{R}^N$ represents the internal torque.
- $u_{env} \in \mathbf{R}^N$ represents the external torques for the system. For instance environmental force in general.
- $M \in \mathbf{R}^{N\times N}$ represents the mass matrix
- $B \in \mathbf{R}^{N\times N}$ represents the centrifugal and Coriolis forces.
- $G \in \mathbf{R}^N$ represents the gravity force.

This dynamic model is represented in the joint space. One prefers to have the task space representation of the dynamics. The end effector of the robot in the task space is represented by the vector: $x \in \mathbf{R}^6$. We should then transform equation 2.1 using the first derivative of x:

$$\dot{x} = J(q)\dot{q} \tag{2.2}$$

where $J \in \mathbf{R}^{6\times N}$ represents the Jacobian matrix of the system. This transformation leads to the following equation :

$$M_{ts}(x)\ddot{x} + B_{ts}(x,\dot{x})\dot{x} + G_{ts}(x) = F_c + F_{env} \tag{2.3}$$

where $M_{ts} \in \mathbf{R}^{6\times 6}$, $B_{ts} \in R^{6\times 6}$ and $G_{ts} \in \mathbf{R}^6$ represent the mass , centrifugal and gravity matrices respectively in the task space, $F_{env} \in \mathbf{R}^6$ and $F_c \in \mathbf{R}^6$ represent the internal and environmental forces applied to the system where:

$$u_i = J^T(q)Fc. \tag{2.4}$$

The transformation from the joint space to the task space and the exact definition of the matrices M_{ts}, G_{ts} and B_{ts} are formulated in this work [3].

2.2 Impedance control

Impedance control is an efficient way to compliant control the desired robotic motions. This work is presented in the this work [4]. The structure of the Impedance controller is depicted in Figure 2.1:

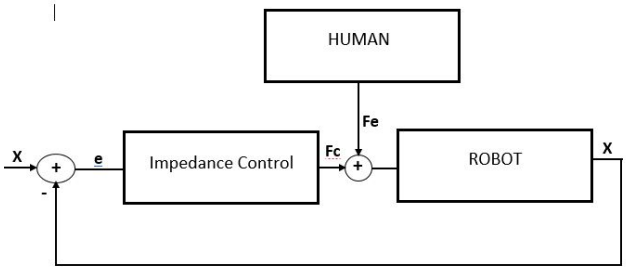

Figure 2.1: Impedance control structure.

The general scheme of a robot controller is composed of a dynamic system to compute the desired motion based on the current state of the robot and a controller to execute the desired motion with stability, optimal and convergence behavior.

We can consider then the DS-based impedance control proposed by Kronander and Billard [5] as follows :

$$F_{imp} = -D(\dot{x_r} - \dot{x_d}) - K(x_r - x_d) \tag{2.5}$$

$$F_{inv} = B_{ts}(x_r, \dot{x_r})\dot{x_r} + G_{ts}(x_r) \tag{2.6}$$

$$F_c = F_{imp} + F_{inv} \tag{2.7}$$

where $F_{imp} \in \mathbf{R}^6$ is the output of the impedance control, $D \in \mathbf{R}^{6\times6}$, $K \in \mathbf{R}^{6\times6}$ are the damping and stiffness matrices and F_{inv} is the force required to compensate for both the centrifugal force and gravity where $B_{ts} \in R^{6\times6}$ and $G_{ts} \in \mathbf{R}^6$ represent the mass , centrifugal and gravity matrices respectively.

The combination of equations 2.7 and 2.3 gives the following equation :

$$F_{ext} = M_{ts}(x)\ddot{x} + D(\dot{x_r} - \dot{x_d}) + K(x_r - x_d) \tag{2.8}$$

We consider no acceleration to the system for simplification so we can model the external (environmental foces) to :

$$F_{ext} = D(\dot{x_r} - \dot{x_d}) + K(x_r - x_d) \tag{2.9}$$

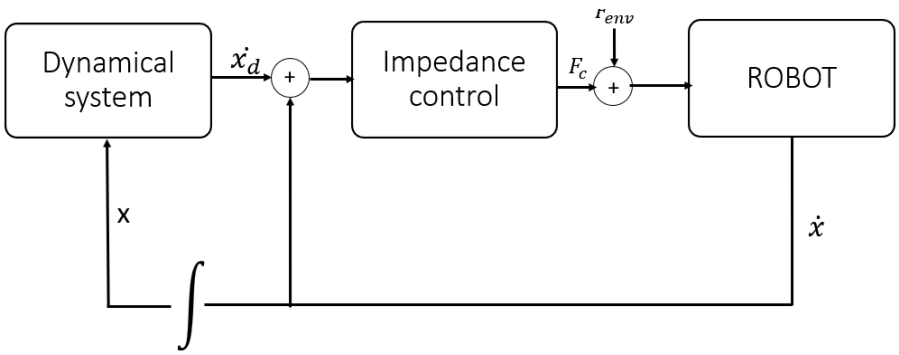

Figure 2.2: DS-based Impedance control [2].

2.3 Parameter estimation using MIT rule

In this section, we are going to present how to estimate a model parameter using adaptive control method from the very general case : adaptive control method using Lyapunov function to a a more simple adaptive law called MIT rule.

2.3.1 General Adaptive control method

The main approach in adaptive control is to first determine a control structure, and derive the error function. The next step is to construct a Lyapunov function based on the error and derive the adaptation law. To explain this, we start with one-dimensional problem dynamics of a robotic system as:

$$m\ddot{x} + b\dot{x} = F_c \tag{2.10}$$

where m is the mass and b is the mechanical damping.
we assume also the environment(including human) in the interaction has the following desired behavior:

$$m\ddot{x}_{env} + b_{env}\dot{x}_{env} + k_{env}x = 0 \tag{2.11}$$

where $b_{env} > 0$ and $k_{env} > 0$ are the unknown damping and stiffness coefficients of the environment. We assume also a simple force controller:

$$F_c = -kx - d\dot{x} \tag{2.12}$$

We want to estimate and adapt the unknown human parameters by trying to minimize the error between the real velocity and the desired velocity (which is generated by dynamical system given its parameters) as explained in the equation 2.13:

$$\dot{e} = \dot{x} - \dot{x}_{env} \tag{2.13}$$

Using the Lyapunov function after deriving the error dynamics, we can derive the adaptation law.
It is given by :

$$\dot{k} = -\epsilon x\dot{e} \text{ and } \dot{d} = -\epsilon \dot{x}\dot{e} \tag{2.14}$$

where $\epsilon \in \mathbf{R}^+$ is the adaptation rate.

2.3.2 MIT rule

The MIT rule, presented in this work [2], is an alternative to find the adaptation law defined in equation 2.14. For this rule we consider the following cost function :

$$J(k, b) = \frac{1}{2}e^2 \tag{2.15}$$

The parameters k and b can be adapted so that the error function can be minimized to zero. For this reason, the change of these parameters is kept in the direction of the negative gradient of the error function J as explained bellow :

$$\dot{k} = -\epsilon\frac{\partial J}{\partial k} \text{ and } \dot{b} = -\epsilon\frac{\partial J}{\partial b} \tag{2.16}$$

The partial derivatives indicate how the error is changing with respect to parameters. For the simple 1-D example above we can write the adaptation rule as follows :

$$\dot{k} = -\epsilon\frac{\partial \dot{x}}{\partial k} \text{ and } \dot{b} = -\epsilon\frac{\partial \dot{x}}{\partial b} \tag{2.17}$$

The MIT rule is preferred to the Lyapunov method because it requires the computation of the error dynamics which is not straightforward. However, this method is locally stable assuming that the adaptation rate ϵ is small. We will use the MIT rule in this project to estimate the parameters of the environment force (see Chapter 3).

3 Proposed method

The main objective of this project is to classify the environment force across the following cases:

- **Passive**: The robot makes contact with a hard surface.
- **Active**: The robot is interacting with a human.
- **Free**: The robot is in free motion and no force is applied to it.

We have as an input vector the dynamics of the robot (speed and position) measured using the joint encoders and the force applied to it measured by a force-torque sensor $I = \begin{bmatrix} F_{env} \\ x \\ \dot{x} \end{bmatrix}$

The Workflow of the theoretical part of the project to achieve this objective is depicted in Figure 3.1.

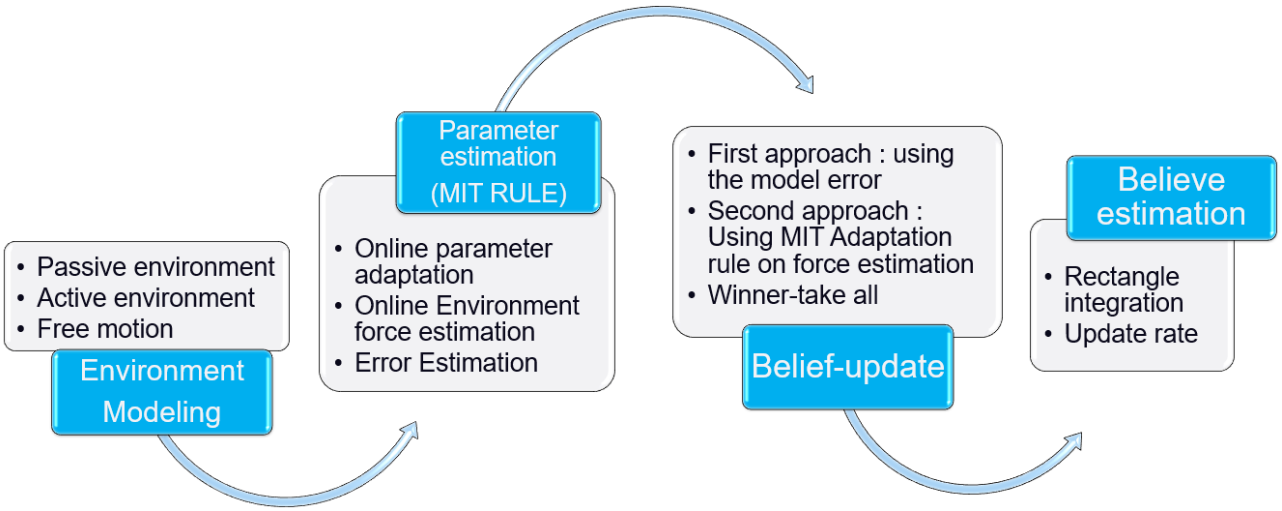

Figure 3.1: Theoretical method workflow: First we start by modeling the environment force with different parameters. These parameters are then estimated using the MIT rule. After adapting the parameters, the error between the estimated force and the measured one is computed to update the beliefs. The different parts of the workflow are discussed in the following sections.

The main idea in this project is to use different models (i.e., passive, active, and free) for the environment and preform the classification based on the model prediction errors. For each model, we adapt the unknown parameters using the MIT rule (see section 3.2) to have a final estimation of the force.
At the end, the predicted errors for each model are converted to beliefs. (see section 3.3)

3.1 Modeling Environment

Let us start first with the modeling of the environment. We consider 3 different models of the environment discussed in the following sections.

3.1.1 Passive environment

The robot executes a predefined task while allowing safe interactions with the environment and tolerating small perturbations.Interaction between environment and robot is described in Figure 3.2.
In the passive model, the contact parameters relate the end-effector position x and velocity v to the interaction force $\tilde{F}_{env}$. $\tilde{b}$ and $\tilde{k}$ are unknown time-varying damping and stiffness matrices of the dynamics,respectively. The force equation of the environment force is then described as follows:

$$\tilde{F}_{env} = -\tilde{k}x - \tilde{b}v \tag{3.1}$$

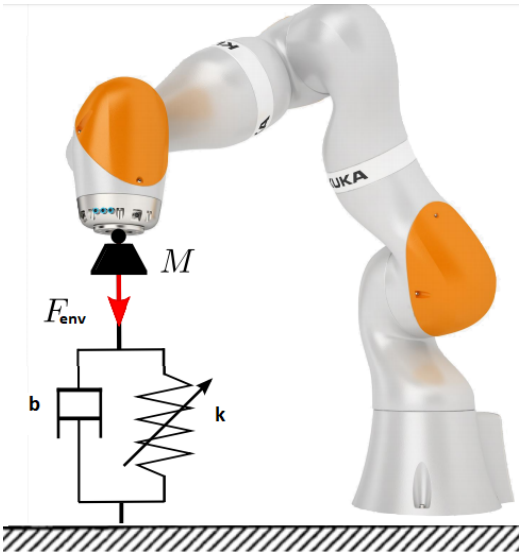

Figure 3.2: Compliant contact situation : Passive environment. The robot executes its predefined task on a surface that acts as a spring damper passive environment.

3.1.2 Active environment

The robot, in this case, should present compliance and following behaviour. In other terms, the human can lead the task of the robot (i.e decides on the desired trajectory). It senses then a constant force from the environment f and some damping b without any stiffness. This model equation is described as follows:

$$\tilde{F}_{env} = -\tilde{f} - \tilde{b}v \tag{3.2}$$

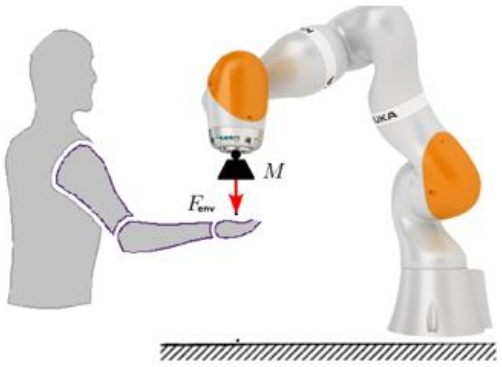

Figure 3.3: Active environment: Robot compliant behaviour. In this case, the robot is interacting with a human.

3.1.3 Pure force model

This model takes only the constant force term as follows:

$$\tilde{F}_{env} = -\tilde{f} \tag{3.3}$$

For all three models, the stiffness, damping, and constant force are unknown. We need then to estimate those parameters to characterize the environment.

3.2 Online parameter estimation

The main approach for the online parameter estimation is to use the adaptation law seen in chapter 2. The main difference in this project is to use the error between the measured force F_m and the estimated one $\tilde{F}_{env}$. The Error function is then described as follows:

$$J(k, b, f) = \frac{1}{2}e^2 \tag{3.4}$$

where $e = F_m - \tilde{F}_{env}$. We can estimate unknown environment parameters by trying to minimize this error function using gradient decent methods described by the MIT rule. We can write then the same equations as 2.17:

$$\tilde{k} = -\epsilon \frac{\partial \tilde{F}_{env}}{\partial k} \tag{3.5}$$

$$\tilde{b} = -\epsilon \frac{\partial \tilde{F}_{env}}{\partial b} \tag{3.6}$$

$$\tilde{f} = -\epsilon \frac{\partial \tilde{F}_{env}}{\partial f} \tag{3.7}$$

where ϵ is the adaptation rate.

After applying the MIT rule for the different methods described in section 3.1, we get the estimations of the following parameters:

- Passive model:

$$\tilde{F}_{env,p} = -\tilde{k}_{passive}x - \tilde{b}_{passive}\dot{x} \tag{3.8}$$

  - model error:

$$e_p = F_m - \tilde{F}_{env,p} \tag{3.9}$$

  - model adaptation:

$$\begin{cases} \tilde{k}_{passive} = -\epsilon e_p x \\ \tilde{b}_{passive} = -\epsilon e_p \dot{x} \end{cases} \tag{3.10}$$

- Active model:

$$\tilde{F}_{env,a} = -\tilde{f}_{active} - \tilde{b}_{active}\dot{x} \tag{3.11}$$

  - model error:

$$e_a = F_m - \tilde{F}_{env,a} \tag{3.12}$$

  - model adaptation:

$$\begin{cases} \tilde{b}_{Active} = -\epsilon e_a \dot{x} \\ \tilde{f}_{Active} = -\epsilon e_a \end{cases} \tag{3.13}$$

- Force model:

$$\tilde{F}_{env,force} = -\tilde{f}_{force} \tag{3.14}$$

  - model error:

$$e_{force} = F_m - \tilde{F}_{env,force} \tag{3.15}$$

  - model adaptation:

$$\tilde{f}_{force} = -\epsilon e_{force} \tag{3.16}$$

Now, we have all model adapted parameters and the corresponding errors between the estimated force and the measured one. Our objective is to know if our robot is interacting with an active or passive environment or simply in a free motion situation. Using the computed error signals, can we update beliefs about the environment? The answer to this question is in section 3.3.

3.3 Passive/active classification

Computing the model errors and the estimated parameters are not enough to classify between active or passive environments. We have to transform this information into beliefs that show the probabilities of being in an active or passive or even in free motion situation.

3.3.1 From error estimation to belief-updates

As a first step, we can compute the adaptation of the beliefs from the model error computed in section 3.2. It corresponds to the variation of the beliefs over the variation of time (derivative). We used two approaches to calculate the belief-updates: the first one using again the adaptation rule or MIT rule the same way as we did to estimate the parameters for each model and the second one by simply considering the beliefs as a Gaussian function with respect to the error model. These two approaches are explained mathematically in sections 3.3.1.1 and 3.3.1.2.

3.3.1.1 MIT Rule approach

Exactly as we did for the online parameter estimation, one can estimate the environment force using a linear combination between the estimated force of the active model and the estimated force of the passive model, the weights correspond to the beliefs of active and passive. Mathematically the estimated force is computed as follows:

$$\tilde{F}_{env} = b_{active}\tilde{F}_{env,a} + b_{passive}\tilde{F}_{env,b} \tag{3.17}$$

where b_{active} and $b_{passive}$ are the beliefs, $F_{env,a}$ and $F_{env,p}$ are the estimated environment forces using the active and passive model respectively. These equations are computed using 3.11 and 3.8. The prior belief is uniform : $b_{active} = \frac{1}{2}$ and $b_{passive} = \frac{1}{2}$.
Using again the MIT rule we can adapt the belief at each time step by trying to minimize the force error function:

$$J(b_{active}, b_{passive}) = \frac{1}{2}(\tilde{F}_{env} - F_m)^2 \tag{3.18}$$

The belief-updates are computed then as follows :

$$\begin{cases} \dot{\hat{b}}_{active} = -\Delta t \frac{\partial \tilde{F}_{env}}{\partial b_{active}} = -\Delta t(\tilde{F}_{env} - F_m)\tilde{F}_{env,a} \\ \dot{\hat{b}}_{passive} = -\Delta t \frac{\partial \tilde{F}_{env}}{\partial b_{passive}} = -\Delta t(\tilde{F}_{env} - F_m)\tilde{F}_{env,p} \end{cases} \tag{3.19}$$

where Δt represents the update rate of the beliefs.

3.3.1.2 Model-error based approach

We can also consider a Gaussian relation between the belief-updates and the corresponding model error. In other terms, if the error goes to zero we will have a maximum belief update. This will allow us to identify the best model. The belief-updates are then computed as follows:

$$\dot{\hat{b}}_{active} = C_1 e^{-\beta_1 \tilde{e_a}} \tag{3.20}$$

$$\dot{\hat{b}}_{passive} = C_2 e^{-\beta_2 \tilde{e_p}} \tag{3.21}$$

$$\dot{\hat{b}}_{free\_motion} = C_3 e^{-\beta_3 \tilde{e_{force}}} \tag{3.22}$$

where the free motion is considered as a zero force.since we consider in this approach the classification between active, passive and free motion, the prior belief in this case is also uniform as follows : $b_{active} = \frac{1}{3}$, $b_{passive} = \frac{1}{3}$ and $b_{free\_motion} = \frac{1}{3}$.

In the second step, these belief-updates for the two approaches are modified based on a winner-take-all process (WTA) that ensures only one increasing belief and 2 decreasing ones.

3.3.2 Winner-take-all algorithm

The winner-take-all algorithm, proposed in this work [6], has two inputs: a vector for the current beliefs and their updates computed based on the error for each case (active,passive or free motion). We should also guarantee that the beliefs are between zero and one and should also sum to 1. Firstly, the case with the greatest update is considered as the winner. We choose randomly one when we have multiple maximums. When this winner is already saturated at 1, no updates are necessary. Secondly, we want to have only one winner with a positive update. This can be done by checking the second biggest update value and calculate its mean with the winner. We subtract then this mean value from all the beliefs updates except the winner to reach our goal. Now we are sure that only the winner has a positive update value. Finally, we ensure that the belief-updates sum to zero to guarantee that the sum of the beliefs stays also constant. To do so, we get all the current updates different to zero and have a negative value and sum them together. The final sum is added to the winner as a positive value. We do this because the beliefs that have negative updates do not influence the process and should be neglected. This is why their contribution is added to the winner to ensure that the sum of the beliefs stays constant.

Algorithm 1 Winner-take-all algorithm

WTA $(B, \dot{B})$ **Input :** Current belief vector B and belief updates vector $\dot{\hat{B}}$
Output: Modified belief updates $\dot{B}$

$w \leftarrow argmax_i \ \dot{\hat{b}}_i \ \forall i$
if $b_w = 1$ **then**
  $\dot{b}_i \leftarrow 0 \ \forall i$
  return $\dot{B}$
end
$v \leftarrow argmax_i \ \dot{\hat{b}}_i \ \forall i \neq w$
$\mu \leftarrow (\dot{\hat{b}}_w + \dot{\hat{b}}_v)/2$
$\dot{b}_i \leftarrow \dot{\hat{b}}_i - \mu \ \forall i$

$S \leftarrow 0$

for i **do**
  if $b_i \neq 0$ *or* $\dot{b}_i > 0$ **then**
    $S \leftarrow S + \dot{b}_i$
  end
end
$\dot{b}_w \leftarrow \dot{b}_w - S$

return $\dot{B}$

3.3.3 Computing beliefs

By now, we calculated the belief-updates using two different approaches. We can then compute the belief by integrating these belief-updates as follows:

$$b_i = b_i + \dot{b}_i \Delta t \tag{3.23}$$

where Δt is the sampling time and $b_i \in \{b_{active}, b_{passive}, b_{free\_motion}\}$

Summarizing, the process to find the probabilities between the three cases is the following :

1. Calculate the belief-updates using equations 3.20 3.21 and 3.22
2. $\dot{B} = WTA(\dot{\hat{B}}, B)$ where B is the current probability
3. $b_i = b_i + \dot{\hat{b}}_i \Delta t$
4. $b_i \leftarrow max(0, min(1, b_i))$ beliefs are saturated between 0 and 1.

3.4 Hyperparameter optimization

Now we can classify the environment between active and passive. This method has 2 hyperparameters that should be tuned by trial and error or optimized.

The hyperparameters of our problem are the following:

- The adaptation rate: ϵ
- the update rate of the beliefs: Δt

Before optimizing our hyperparameters, we have first to define an error metric or score that illustrates the performance of our classification method.

3.4.1 Error metric

As a measure of performance for our classification between active and passive environment, we consider the calculus of the area between the estimated belief curve and the ideal one.

The first curve corresponds to the ideal case when the belief estimation goes very quickly from zero to one following a step function behaviour (i.e., red plot in Figure 3.4) and the second curve corresponds to the estimated believe from our classification method.

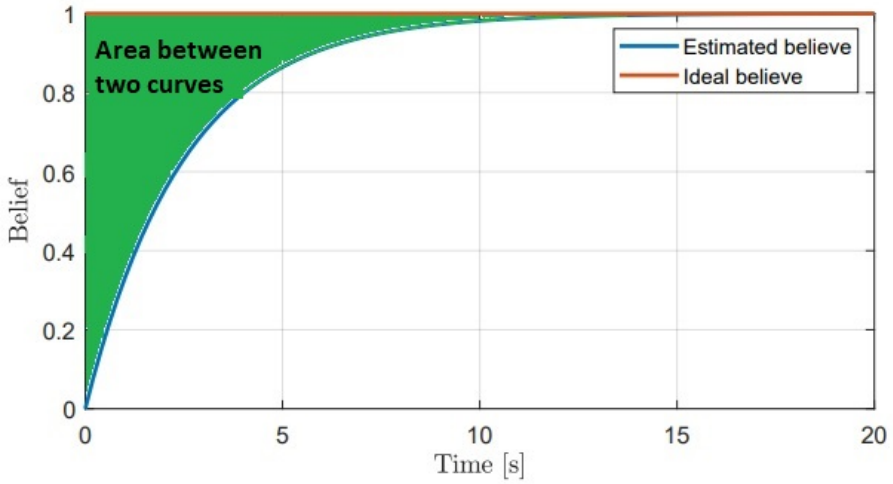

Figure 3.4: Area between the ideal belief and the estimated belief that corresponds to the error metric colored in green.

Mathematically this area can be computed mathematically as follows:

$$\int_0^T (b(t) - b_{ideal}(t))dt \tag{3.24}$$

T is the time of the simulation and b_{ideal} is the Ideal believe estimation. This integral can be computed numerically using cumulative trapezoidal integration. This area will be always positive since the estimated believe will not go superior to one. This area between the two curves represents a good error metric that can illustrate the performance of our classification. If the two curves get close to each other , the area between the curves will be smaller.

One could think about different techniques of hyperparameter tuning:

3.4.2 Simple Grid-search

Taken from the imperative command "Just try everything!" comes grid search, it's a naive but efficient approach of simply trying every possible configuration in a certain range. To optimize our hyperparameters, a grid search over the values of these parameters is performed. I.e., our classifier method is performed, using these parameters. We take the two hyperparameter values that give the small computed area (based on the error-metric discussed above 3.4.1).

3.4.3 Iterative hyperparameter tuning

An alternative approach to grid-search is to consider an optimization problem. We want to minimize the area between the ideal belief and the estimated one by changing these two hyperparameters. This can be done by following the gradient descent for example. Many iterative algorithms exist to obtain the minimum of this two-variables function: min F(update rate, epsilon). We used, for example, a MATLAB function "fminsearch" that try to find a local minimum of this function starting from an initial guess. Depending on the starting guess, this technique will converge to a local minimum. We need then to test the algorithm using different initial points.

The desired hyperparameters should perform well on active and passive environment at the same time. In other terms, the detection of the environment type has to be independent of the choice of our hyperparameters.
We prepare next a dataset giving the estimated beliefs by changing the type of the environment at each time. We can consider different cases of passive,active, and then combined environment when we switch the type from active to passive or passive to active. we make sure that the dataset is balanced between active and passive to avoid biased results. Afterwards, we perform a Nested grid-search.

3.4.4 Nested Grid-search

To be able to have efficient results with the different techniques discussed above, we have to know what is the type of the environment to obtain efficient tuning results. In this case, we cannot find the best combination of hyperparameters online in real robotic applications. One solution to this issue is Nested hyperparameter tuning.
We use our balanced dataset created from different types of environment (Simulated 4.1 or real data using experiments. 5.1) to run our adaptation for all our training data for a given epsilon and update-rate. Then we can compute the area between the estimated belief and the ideal one. Afterwards, we can apply a 3D grid-search by changing the values of our hyperparameters. We obtain then a three-dimensional matrix that contains the error for a couple of hyperparameters as conceptually shown in Figure 3.5.

$$(\text{Epsilon},\ \text{Update rate},\ \ \text{Environment}) \tag{3.25}$$

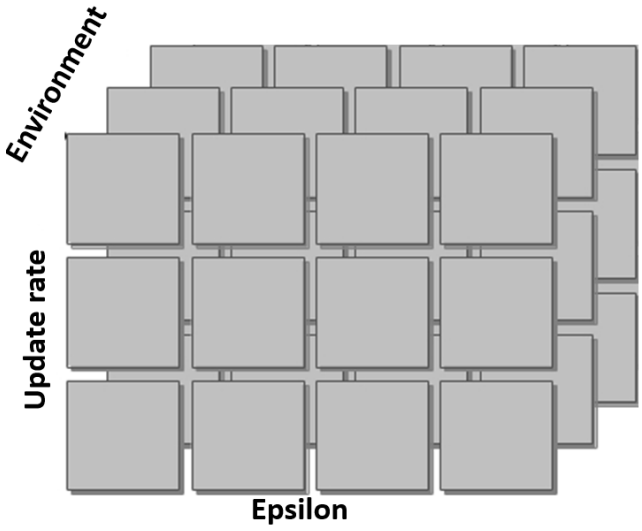

Figure 3.5: Grid search over the hyperparameters epsilon and the update rate.

The computed errors are then averaged over all the experiments (environments used in the training data), reducing the dimensions of the error matrix by one. Optimal hyperparameters are found by choosing those with the least averaged error.

The real pain point of this approach is known as the curse of dimensionality. This means that the more dimensions we add, the more the search will explode in time complexity (usually by an exponential factor), ultimately making this strategy unfeasible. To avoid this, we can run a discrete grid search for different values of our hyperparameters and then use the best result as a starting guess for the iterative process discussed in section 3.4.3 to obtain the global minimum of the loss function.

Summarizing, we used a combination of the different techniques discussed to have the optimal hyperparameter tuning :

1. Building the dataset from simulated or real robot data.
2. Nested hyperparameter tuning on discrete values and a reasonable number of samples.
3. Take the result of the Nested-grid search as a starting guess for the iterative process.
4. Run the iterative process using fminsearch Matlab function.

3.4.5 Hyperparameter tuning results

After applying the method described before for the hyperparameters optimazation, we found that the following hyperparameters give the best result for both passive and active environment.Firstly, we build our dataset from real robot data.This part will be discussed in section 5.1. Secondly, we perform the nested grid-search on discrete values and reasonable number of samples. We interpolate then between these values. The result of this step is depicted in the Figure 3.6.

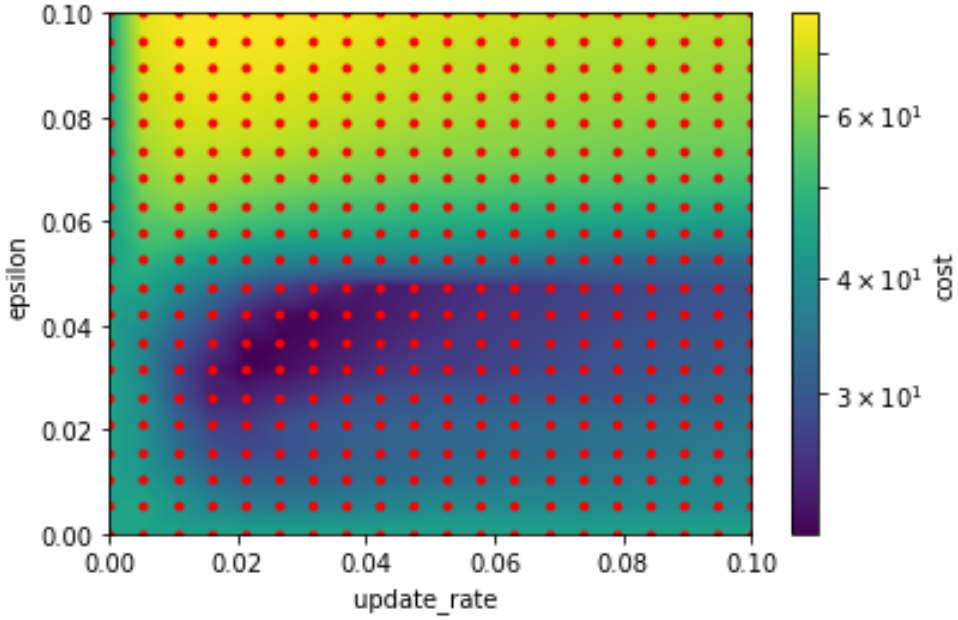

Figure 3.6: Nested Grid search over discrete values of hyperparameters epsilon and update rate. The final result is linearly interpolated.We tested different values. This figure shows a zoom of the best region of interest.

We can see from this figure a very good approximation of the global minimum position. We can easily choose a starting guess for the iterative process : $\epsilon = 0.01$ and update rate=0.01.
We present in the following table the final result of the tuning method after performing the iterative process discussed in the section 3.4.

Hyperparameter	Value
ϵ	0.0367
update rate	0.0224

Table 1: Hyperparameters tuning results: After tuning the hyperparameters discussed, we can find the best hyperparameters that give the lowest error. ϵ is the adaptation rate and the update rate is used to convert the update-beliefs to beliefs.

4 Simulation

For illustrative purposes, we investigate the theoretical part discussed in section 3 for a simple case considering a robot end effector with 1 degree of freedom. This can be done on simulation using MATLAB.

4.1 Implementation

The Dynamics of the robot end effector is straightforward: First, we control the robot's end effector using force impedance control.After adding the environment force F_{env} to this control force F_c, we can compute the dynamics of the robot using Newton's second law as follows :

$$\ddot{x} = \frac{1}{m}(F_c + F_{env}) \tag{4.1}$$

where m represents the end effector mass, F_{env} represents the environment force and finally F_c represents the force impedance control computed as follows:

$$F_c = -k_c(x - x_r) - b_c\dot{x} \tag{4.2}$$

where x_r represents the goal position of the robot. k_c and b_c represent the control stiffness and damping respectively.
Since we don't have any measurements at hands in the simulation, We should then try to simulate the environment force. We assume an active environment is applying a constant force F_0 on the robot with some damping as shown in the following equation:

$$F_m = -F_0 - b_{env}\dot{x} \tag{4.3}$$

where the robot hits the surface at the position and a passive environment reacting as a spring-damper with a force equal to :

$$F_m = -k_{env}(x - x_{surface}) - b_{env}\dot{x} \tag{4.4}$$

where the robot hits the surface at the position $x_{surface}$ and finally a free motion environment with zero force.

The dynamic of the robot is also simulated. From the acceleration derived in equation 4.1, we can deduce the dynamics of the system which are the velocity x and the speed v by simple rectangle integration as follows :

$$\begin{cases} v(t+1) = v(t) + \ddot{x}\Delta t \\ \\ x(t+1) = x(t) + v(t)\Delta t \end{cases} \tag{4.5}$$

where Δt is the update rate. Afterwards, we estimate the models online based on the dynamics of the system and using the mit rule as explained in the theoretical part (see 3.2). The beliefs are computed with two different ways: Gaussian 3.3.1.2 and using MIT rule again 3.3.1.1. Finally we can compute the desired beliefs. The main problem here is that we cannot well detect the free motion using the MIT rule approach as a belief-update.

4.2 Simulation Result

As a first result, let's try to plot the dynamics(Position and speed) of the robot using different types of environment. We can clearly see then a different behaviour from active to passive which can intuitively indicate a possible pattern generation.

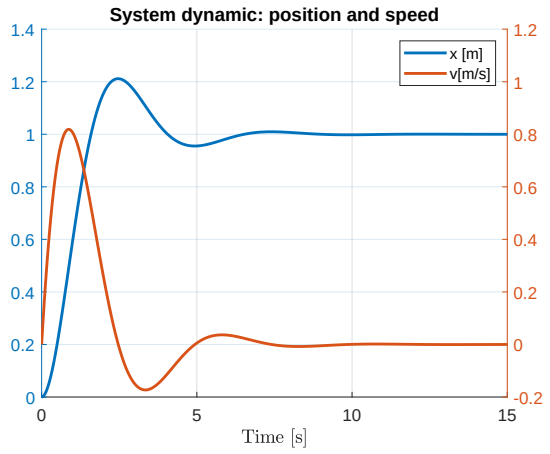

Figure 4.1: End effector dynamics in an active environment

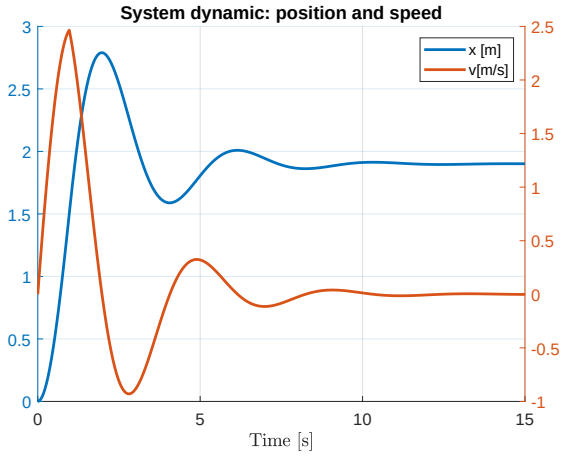

Figure 4.2: End effector dynamics in a passive environment

Using the MIT Rule described in section 3.2 we can adapt the parameters of each model according to the environment which the robot is interacting with. We can see the result of this adaptation in the figure below :

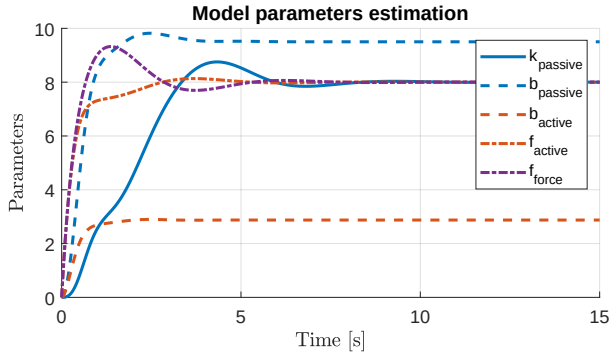

Figure 4.3: Parameters adaptation of active, passive and force models together where the environment is active.

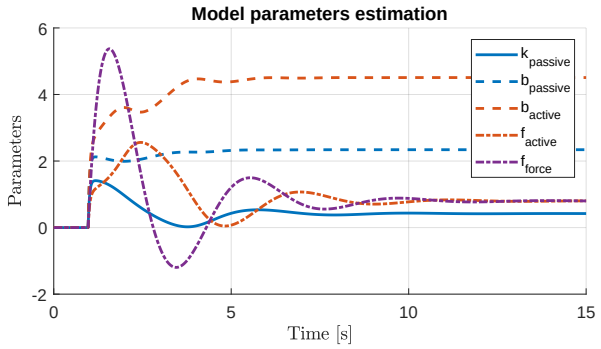

Figure 4.4: Parameters adaptation of active, passive and force models together where the environment is passive.

We can also compute the errors between the simulated force environment and the estimated force environment from the three proposed models when the environment is active and passive:
When the environment is active we have :

$$\begin{cases} F_m = -F_0 - bv \\ F_{env} = -\tilde{f} - \tilde{b}v \end{cases} \tag{4.6}$$

When the environment is passive we have :

$$\begin{cases} F_m = -kx - bv \\ F_{env} = -\tilde{k}x - \tilde{b}v \end{cases} \tag{4.7}$$

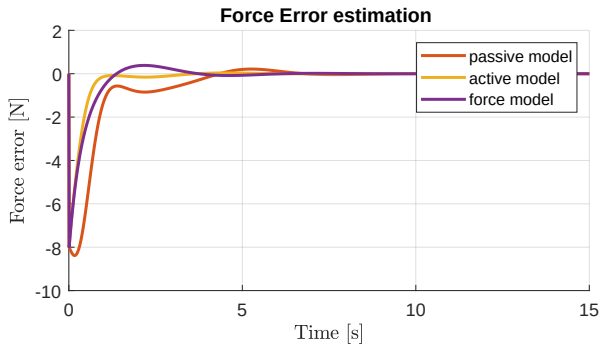

Figure 4.5: Force error estimation using active environment for each proposed model.

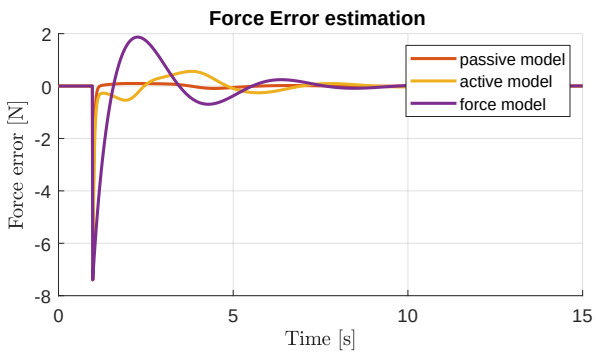

Figure 4.6: Force error estimation using passive environment for each proposed model.

Using an active environment, we can see that the force error of the active model converges to zero and the passive struggles and oscillates a lot before slowly converging to zero.
Despite that the force error estimation of the passive model goes to zero faster using a passive environment, all the errors estimation using the different models converge quickly to zero because the robot at some point find the surface and stop moving (if the position/speed converges to zero everything try also to follow this convergence behaviour). This can cause sometimes small difficulties to detect a passive environment or especially distinguish between passive and free motion. Which means that the error model can be a very good pattern to distinguish between active and passive environment.

From the error estimation, we can compute the belief-updates using two different ways as discussed in section 3.3.1 and finally integrate them to get the beliefs estimation. We present the results for both approaches in section 4.2.2 and 4.2.1.

4.2.1 Model-error based approach

We present in the figures below the different results when using the model-error based approach to update the beliefs.

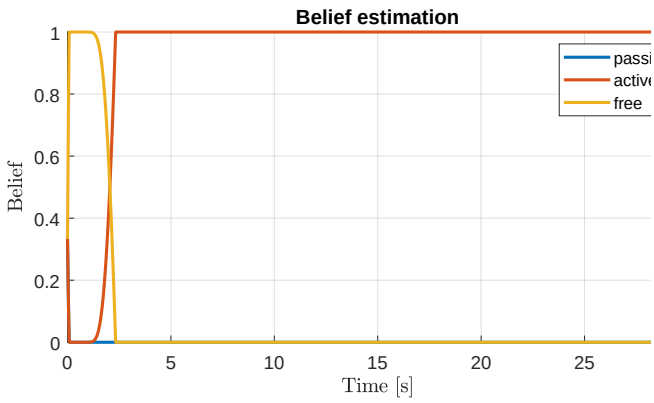

Figure 4.7: Belief estimation using an active environment.

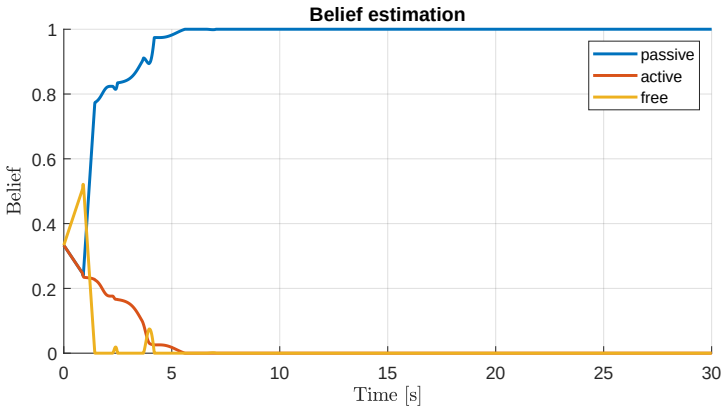

Figure 4.8: Belief estimation using a passive environment.

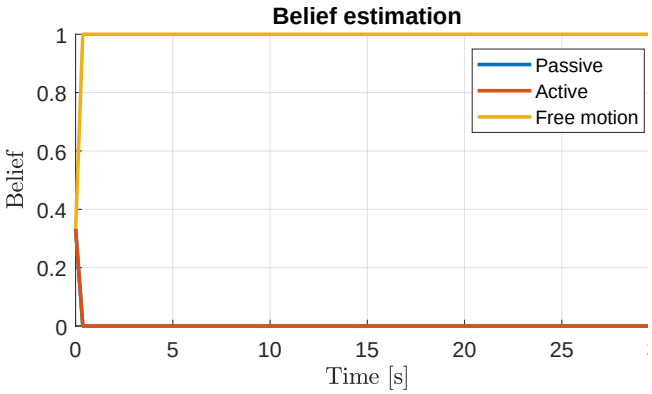

Figure 4.9: Belief estimation when the robot is in free motion.

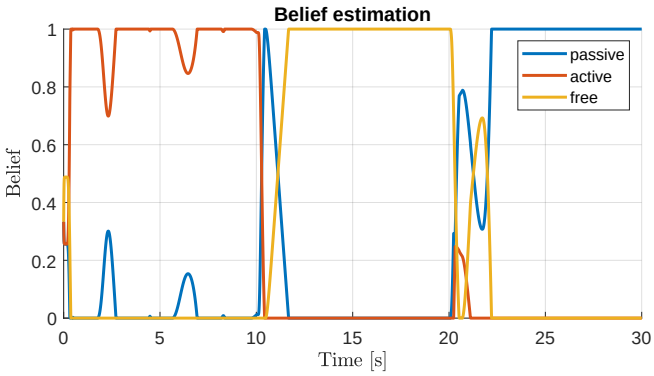

Figure 4.10: Belief estimation using an active environment then passive environment after t=10s.

We see from Figures 4.7,4.8 and 4.9 that we can easily distinguish between active,passive and free motion. We can switch also from an environment to another and see if the system can adapt it's parameters online and change the guess of the environment accordingly.We make here the simulation a bit longer, we change the environment from active to free motion at time 10s and then from free motion to passive at time 20s.The result is depicted in Figure 4.10. We notice that the system is struggling a bit to distinguish between passive and free motion after 10s as we predicted previously.

4.2.2 MIT Rule approach

An intuitive idea to compute the beliefs is to use again the MIT rule based on the estimated error after computing the estimated force with the updated model parameters using also the MIT rule.
Here we show in Figure 4.11, the belief estimation. We change the environment from active to passive at time 10s and then from passive to active again at time 20s.

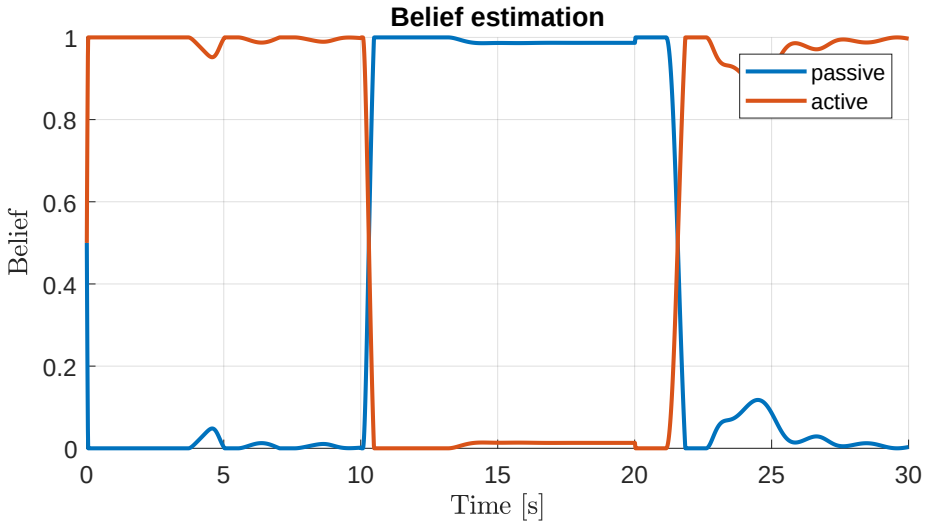

Figure 4.11: Belief estimation using active environment then passive and finally active environment again.

The convergence of this method is very fast compared to model-error based approach. However,it doesn't allow us to detect free motion, it can be then used only for detecting active and passive environments. Overall, both approaches allow us to have a good estimation of the environment's type on simulation. Now we can move to test our method on real robot data.

5 Robotic experiments

We want now to test our implementation on a real robot data. We used the KUKA robot to collect some data for different types of passive and active environments. The robot is only gravity compensated and no control strategy is implemented to manipulate the robot. We measure the speed and the position of the robot end effector using the joint encoders and the force applied to it measured by a force-torque sensor. We manipulate the robot in only one axis. We get at the end our input vector: (x,v,F_{env}).

5.1 Data collection

We start first to collect data with an active environment when a human is interacting with the robot. We show in Figure 5.1 below this first experiment.

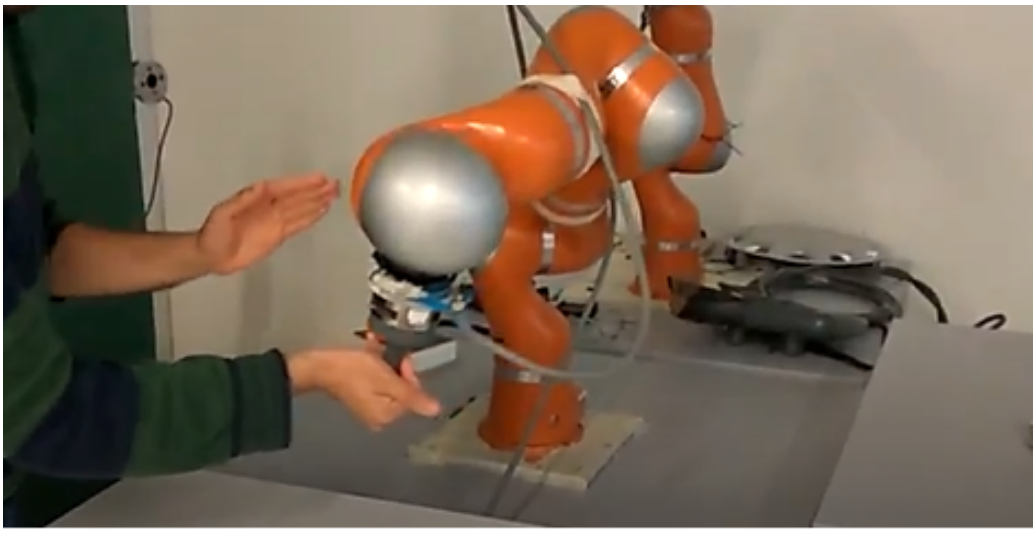

Figure 5.1: Experiment 1: Manipulating the robot and acting as an active environment.

Afterwards, we selected some objects in the lab that could act as a passive environment for the robot as shown in Figure 5.2:

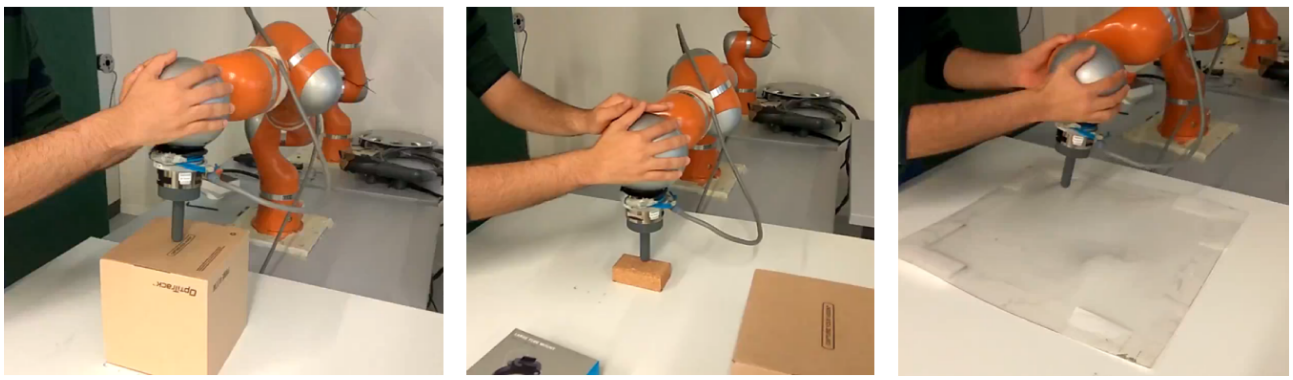

Figure 5.2: Experiments 2,3 and 4: The robot try to interact with some passive environments (some weird objects found in the lab).

We have now a row data that we can put as an input to our implementation on MATLAB instead of putting simulated measurements. This is the only difference from the simulation discussed in section 4.1, in other terms, we are going to adapt the different model parameters based on real data.

We recorded two log files for each type of environments: four logs for active and 6 logs for passive (2 logs for each passive object).

5.2 Results on robot data

Now we have the log files for each type of environment we can parse these files to have our input dataset. For simplicity We consider only the results on the vertical axis to have a 1D situation. We run afterwards our method on these data after tuning our hyperparameters as discussed in section 3.4. Here we will present the result on robot data using the error-based method adaptation.

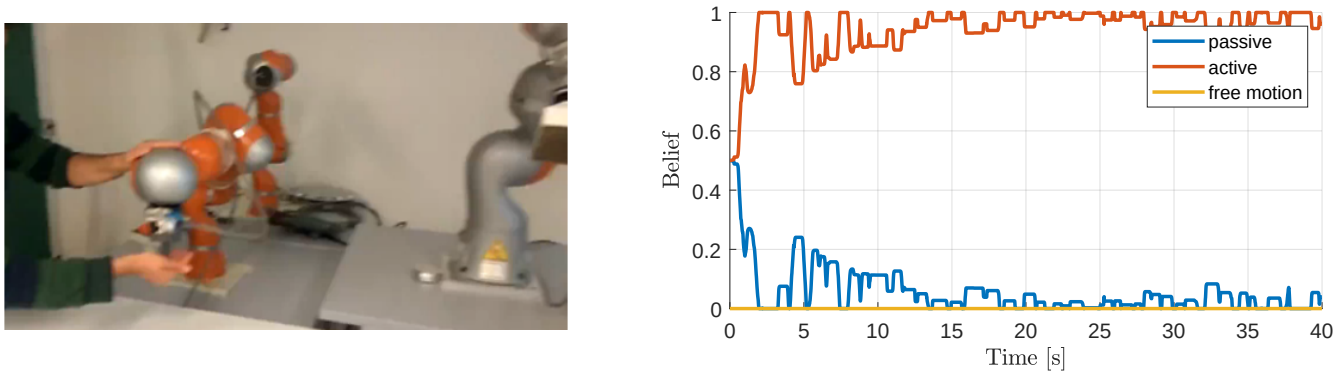

Figure 5.3: Adaptation result in an active environment : Experiment 1. The left image shows the experiment by moving the robot and interacting with it and the right image shows the corresponding belief estimation.

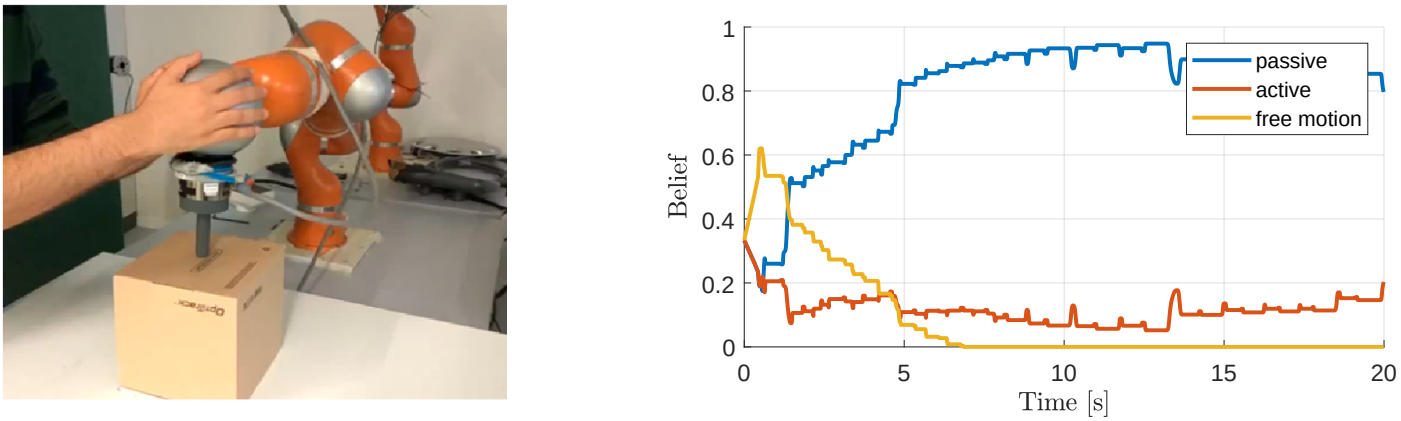

Figure 5.4: Adaptation result in a passive environment : Experiment 2. The left image shows the passive object used and the right image shows the corresponding belief estimation.

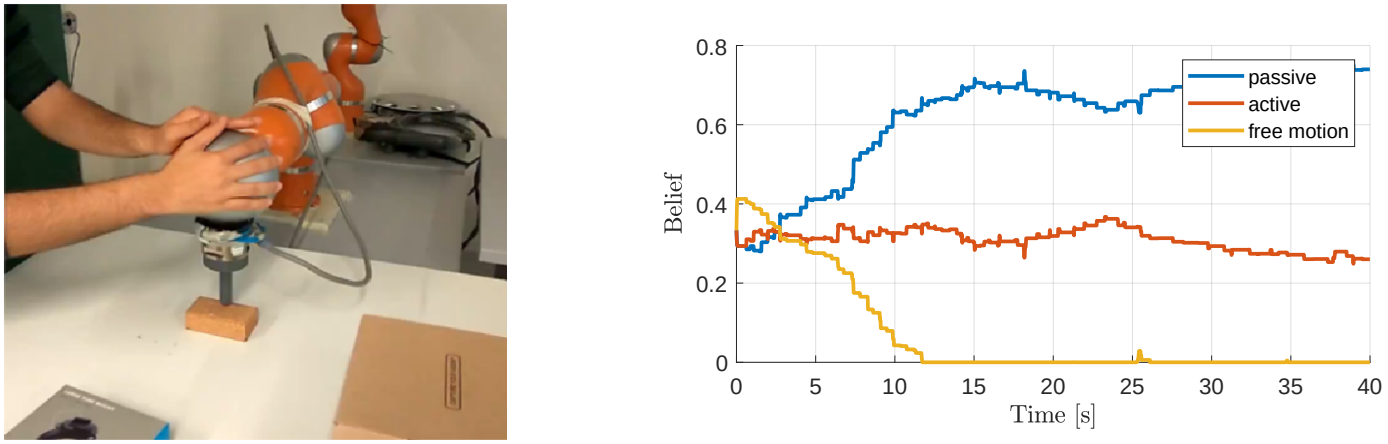

Figure 5.5: Adaptation result in a passive environment : Experiment 3. The left image shows the passive object used and the right image shows the corresponding belief estimation.

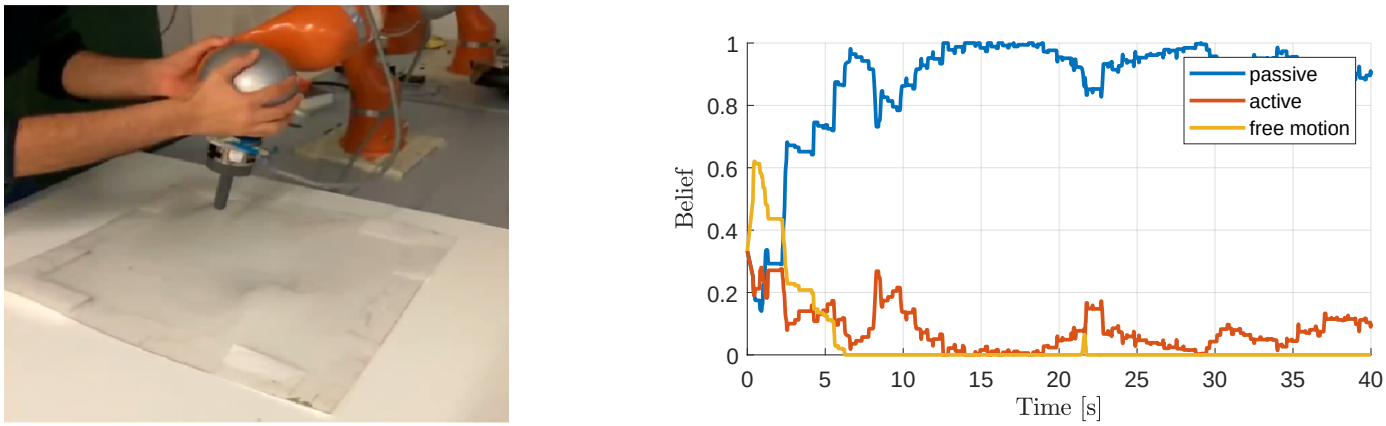

Figure 5.6: Adaptation result in a passive environment : Experiment 4. The left image shows the passive object used and the right image shows the corresponding belief estimation.

6 Using adaptation as feedback

The most important thing now is that the robot should adapt its behaviour when the belief estimation changes from passive to active our the opposite. We Add in this part the belief estimation of the environment as a feedback control to the robot as explained in this Figure 6.1.

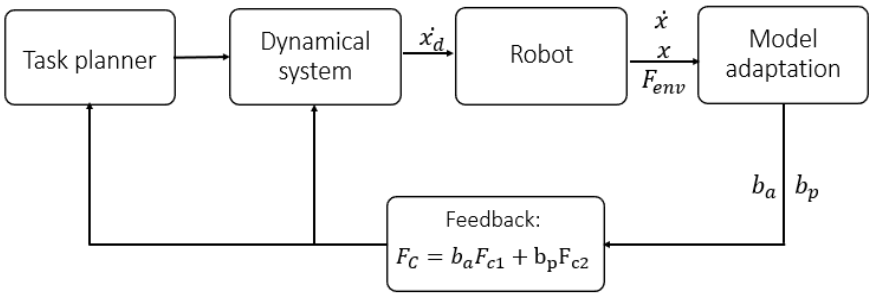

Figure 6.1: The model adaptation as feedback to the impedance control.

We can use the belief estimation to adapt the impedance control of the robot by changing the stiffness of the controller in order to allow the robot to interact compliantly when the human wants to move it. We can suppose for example two different impedance controllers where one has high stiffness and the other one a low stiffness: This means that when the robot is executing its predefined task on the surface it will be very stiff and reject perturbances. In this case, we want to apply the first controller that has higher stiffness. We want also to activate the second controller when the robot have to show a compliant behaviour when it detects an interaction with a human. In this case, we apply the second controller that has a lower stiffness. We can then apply a unique impedance force control to the robot as a weighted sum between the two desired controllers where the weights correspond to the beliefs of active and passive.

We show in the Figure 6.2 how the controller changes its behaviour according to the belief estimation. The speed and position curves at the top show the dynamic of the robot. The surface is at x=1.6m from the origin (the initial position of the robot). We can clearly see that the robot starts going to the desired position and stops to execute the predefined task. When the human starts to interact with the robot at time t=20s, the robot goes back to its initial position quickly. After t=25s, when the robot stops its interaction with the human,it goes again to the surface. We show in the last curve at the bottom the transition in the stiffness of the impedance controller. We see that the stiffness decreases when the believe estimation switches from passive to active.

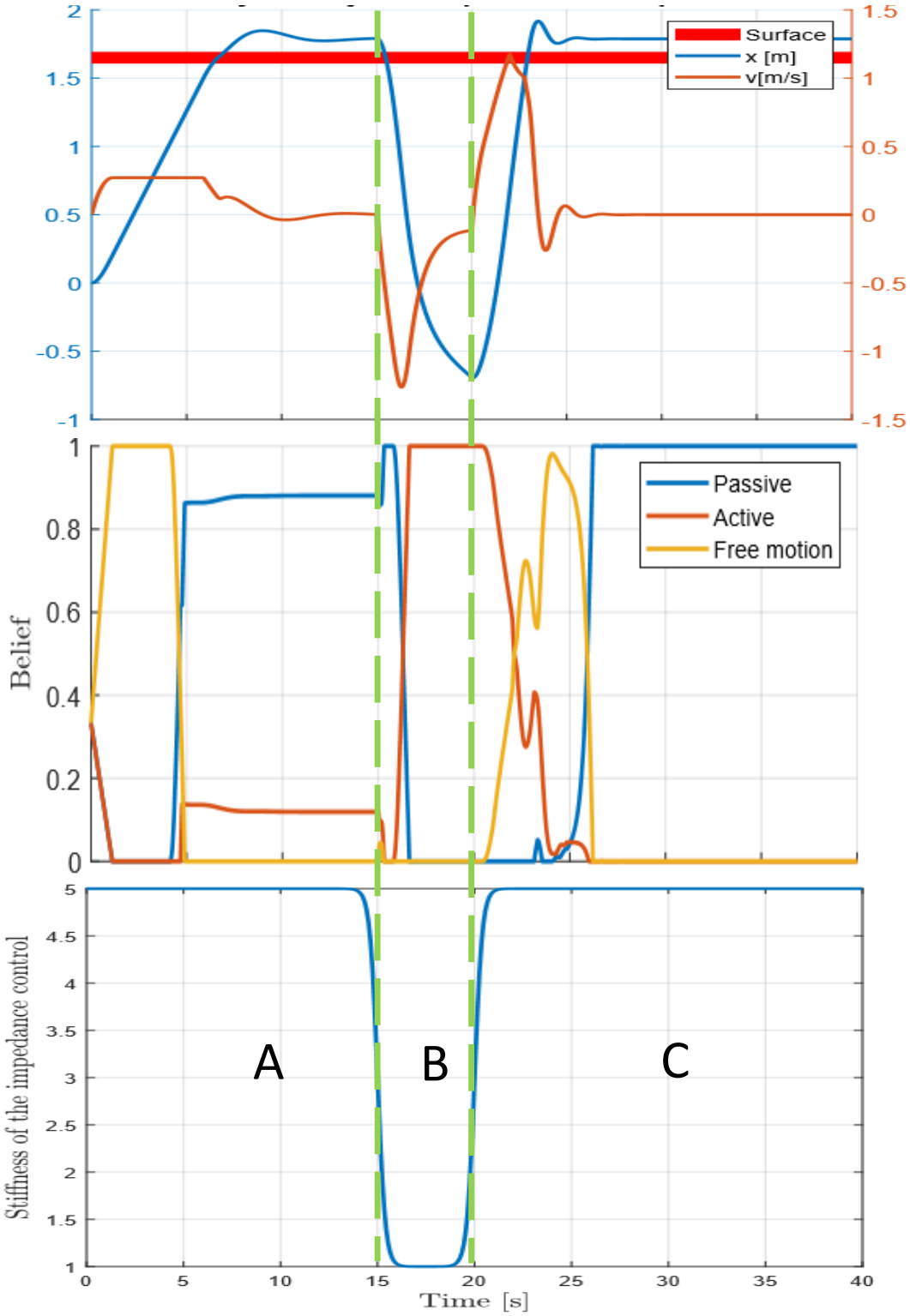

Figure 6.2: This figure shows the feedback behaviour added to the robot. At the beginning (A), the robot goes to the surface to execute its desired force on it : We can see a small period of free motion before hitting the surface at x=1.6m. Afterwards, the robot applies its desired force to the passive environment. At time t=15s (B), a human begins to interact with the robot, the belief estimation switches from passive to active quickly and the robot reacts to this transition trying to change its stiffness accordingly to be compliant to the human. When the stiffness of the impedance control changes from 5 to 1 quickly, the robot stops to execute its desired force and goes away from the surface. When the human stops interacting (C), the robot will try to go to the surface again. We can see again a small period of free motion between 20 and 25s and then the belief estimation becomes passive when the robot reaches the surface.

As we saw previously, the human moves the robot between t=15s at t=20s. We want to compare here the energy consumed by the human to move the robot with and without the adaptation. In other terms, the human will try to move the robot at t=10s with and without adaptation and we will report the difference using the effective power as a metric:

$$P = \int_0^T |Fv| \, dt \tag{6.1}$$

where P is the effective power and T is the duration of the simulation.

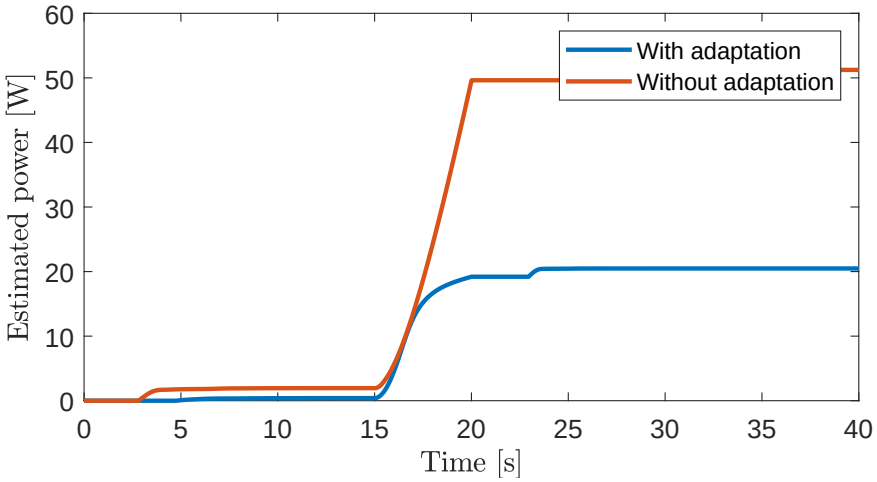

Figure 6.3: Consumed power by the human to move the robot as a cumulative integral of the force times the speed: Before t=15s the robot executes its defines task to a passive environment, the consumed energy stays then zero. After t=15s the human starts moving the robot so the consumed energy starts also increasing to stop at time t=20s when the interaction with the human is finished. Since we compute a cumulative integral, the consumed power stays constant after t=20s.

We want also to see the effect of the hyperparameters on the effective energy consumed to move the robot. This effect is presented in the Figure 6.4.

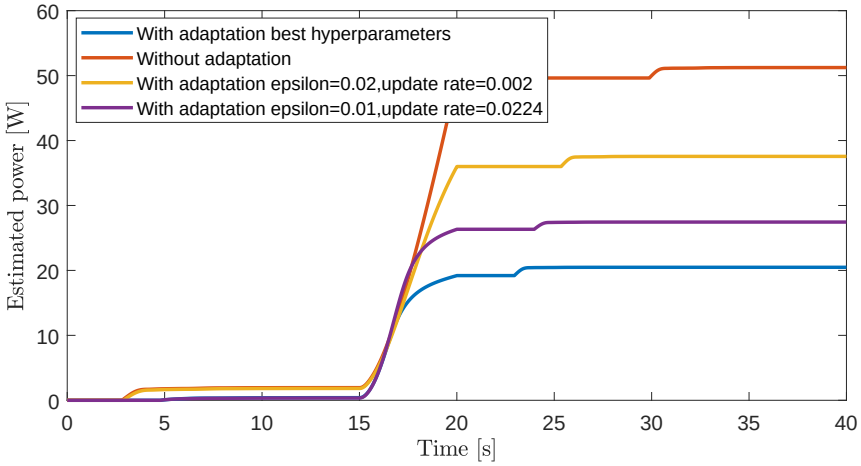

Figure 6.4: Consumed power by a human to move the robot with different adaptation hyperparameters

We can clearly see from Figure 6.4 that the best hyperparameters found in section 3.4 give also the best result in terms of energy consumption when the human tries to move the robot and acting as an active environment. This result is expected since the best hyperparameters ensure a fast and accurate transition between passive and active so the control loop part added will quickly decrease the stiffness of the impedance control and then the human will consume less energy to move the robot away from the surface.

7 Conclusion

In this project, we present an approach to learn the robot to detect which type of environment it's interacting with by adapting the parameters of three models: passive,active and finally pure force using the MIT rule.

After adapting the parameters of each model, we calculate the final estimated force. We tried to convert this into belief-updates using two different approaches: As a first approach, we update the beliefs based on the error between the estimated force from the model and the measured one, We considered a Gaussian relationship between the error and the belief update. As a second approach, we compute the final estimated force as weighted sum between the passive and active models where the weights are the beliefs. The belief-updates are then calculated using again the MIT rule. Finally the result is integrated using an update rate parameter that should be tuned to have at the end a belief estimation of each type of environment.

Both approaches to update the beliefs are satisfactory to detect the difference between passive and active environments. Nonetheless, the update method using MIT rule fails to detect free motion on real robot data. This is due to the noise in the force measurements. It is also important to mention that when the robot stops moving in a passive environment (zero speed and a constant position) it is very difficult to differentiate between passive and free motion. The MIT rule approach considers always free motion as passive environment.

We tune the hyperparameters using a combination between a grid-search and an iterative process using a predefined MATLAB function that can detect local minimums depending on the initial guess. We performed first a nested grid-search based on real robot data and on discrete values of ϵ and update rate. The result of the discrete grid-search is then interpolated between the values to detect a possible range of the global minimum of the loss function. At the end we use a value in this range as a starting guess for the iterative process that complete the task to find the exact global minimum.

We show also that all the adaptation method presented in this work can be integrated into the control loop of the robot. This will allow the human to lead the tasks of the robot online and can ensure a fast and accurate human-robot detection.

This work presents different results of the method on simulation and using also real robot data but in a very simple situation by considering only one degree of freedom of the robot's end effector.
The method presented in this work is based on impedance models. Another approach that can be tested also to be able to classify the environment is to train a machine free algorithm on robot data in a sliding window fashion. In other terms, we can take as input to this classifier a window of 1 second taking the position, the speed and the sensed force as features and apply a machine learning classifier such as support-vector machine or Gaussian mixture models. This can allow us to determine at each second the type of the environment the robot is interacting with.

References

[1] Ifr forecast: 1.7 million new robots to transform the world´s factories by 2020. URL https://ifr.org/ifr-press-releases/news/ifr-forecast-1.7-million-new-robots-to-transform-the-worlds-factories-by-20.

[2] Mahdi Khoramshahi. From human-intention recognition to compliant control using dynamical systems in physical human-robot interaction. page 200, 2019. doi: 10.5075/epfl-thesis-9120. URL http://infoscience.epfl.ch/record/263779.

[3] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, February 1987. ISSN 0882-4967. doi: 10.1109/JRA.1987.1087068.

[4] Neville Hogan. Impedance control: An approach to manipulation. In *1984 American control conference*, pages 304–313. IEEE, 1984.

[5] K. Kronander and A. Billard. Passive interaction control with dynamical systems. *IEEE Robotics and Automation Letters*, 1(1):106–113, Jan 2016. ISSN 2377-3766. doi: 10.1109/LRA.2015.2509025.

[6] Mahdi Khoramshahi and Aude Billard. A dynamical system approach to task-adaptation in physical human–robot interaction. *Autonomous Robots*, 43(4):927–946, 2019.