Model Predictive Control for Dynamic Cloth Manipulation: Parameter Learning and Experimental Validation

Adrià Luque, David Parent, Adrià Colomé, *Member, IEEE*,
Carlos Ocampo-Martinez, *Senior Member, IEEE*, and Carme Torras, *Fellow, IEEE*

***Abstract*—Robotic cloth manipulation is a challenging problem for robotic systems. Textile items can adopt multiple configurations and shapes during their manipulation. Hence, robots should not only understand the current configuration of the item but also be able to predict its future possible behaviours and perform real-time control during manipulation. This paper addresses the problem of indirectly controlling the configuration of certain points of a textile object, by applying actions on other parts of it through the use of a Model Predictive Control (MPC) strategy. MPC allows to foresee the behavior of indirectly controlled points, while satisfying physical/operational constraints. This is done by first identifying the optimal control signals that may constitute the desired future cloth configuration. After that, a dynamic model of the item will be used and sensor data will allow to update the belief on the object's state and close the loop. This paper investigates how grasping the upper corners of a square piece of cloth can allow to track a reference trajectory of the pieces' lower corners. To do so, we propose and validate a linear cloth model that allows solving the MPC optimization problem in real time. Reinforcement Learning (RL) techniques are used to learn the parameters of the proposed cloth model and to tune the resulting MPC. The full control scheme was implemented and executed in a real robot, obtaining accurate tracking results in adverse conditions.**

***Index Terms*—cloth manipulation, model predictive control, reinforcement learning, robot perception**

I. INTRODUCTION

ROBOTS have become a key component for increasing the productivity in industry since the 20th century. Furthermore, nowadays robots are starting to be part of domestic environments. In both situations, we can find deformable objects like textiles. Until now, most robotics research has focused on rigid object manipulation. However, the textile industry today encounteres major technological difficulties in automating parts of their production processes. A simple task such as taking a cloth garment from a rack and putting it in a box for shipping is still an unsolved problem which the industry is trying to find a solution for. Highly deformable objects represent a major challenge due to their physical properties: their shape and appearance can continuously vary during manipulation. Therefore, the robot needs to understand the cloth current configuration and also predict how its action will change the cloth state.

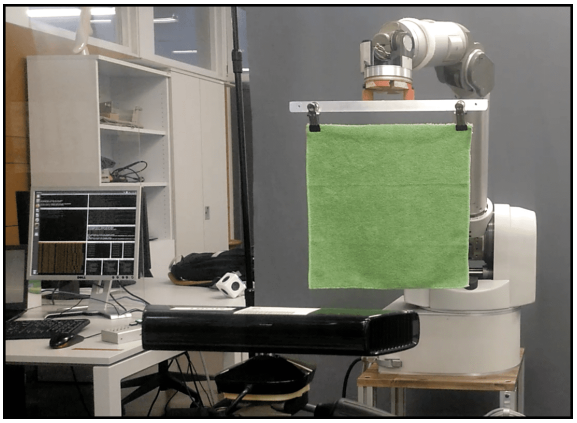

Fig. 1: Picture of the setup used in the real-world experiments

In a previous work [1], Dimensionality Reduction (DR) techniques applied to motion characterization together with Reinforcement Learning (RL) were successful at learning to fold a polo shirt. However, the authors realized how sensitive the output of the action was to any perturbation on the initial conditions, resulting in large noise when mapping a robot motion parametrization to the reward function to be optimized in the training phase. Feed-forward models have also been used in robotic cloth manipulation [2]. The control action results from the sum of the outcome of the Inverse Dynamical Model (IDM) of the robot: the torque necessary to maintain a tuple of position, velocity and acceleration, and a PID compensating term that accounts for the model deviation from ground truth and external disturbances, such as cloth dynamics. While feed-forward controllers allow for smaller gains and thus more compliance, they loose precision with larger unmodelled external disturbances. Ideally, one would include the IDM of the manipulated object so as to compensate for its dynamics. While in [3], an IDM model for a rope is learned through CNNs after 60k training interactions, learning such models for a cloth garment would be more sample-expensive and, therefore, it is probably not the best option for cloth manipulation. Nevertheless, dynamic knowledge of the manipulated object needs to be included in the control loop. Otherwise, the behavior of the cloth may drift away from the desired one, with the controller being unable to compensate such error, as a model-free, reactive controller can struggle

This work was developed in the context of the project CLOTHILDE ("CLOTH manIpulation Learning from DEmonstrations"), which has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programmee (Advanced Grant agreement No 741930). The work is also supported by Project PID2020-118649RB-I00 funded by MCIN/ AEI /10.13039/501100011033.

All authors are with the Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Barcelona, Spain. {aluque, dparent, acolome, cocampo, torras}@iri.upc.edu. C. Ocampo-Martinez is also with the Automatic Control Department, Universitat Politècnica de Catalunya - BarcelonaTECH, Barcelona, Spain. carlos.ocampo@upc.edu.

Implementations available at https://github.com/Alados5/{mpc_node/mpc_vision/tfm_matlab}

to infer the necessary change in commands to correct the cloth's behaviour. This fact hinted at the need to include some kind of predictive behavior and prior knowledge in learning cloth manipulation. The natural next step is to use a control technique that makes the robot proactively modify its motion according to the predictions made based on a cloth model.

A suitable technique along this line is Model Predictive Control (MPC) [4]. The idea of MPC is to use a mathematical model of the plant to be controlled together with an optimization algorithm. This algorithm looks for the best possible control inputs according to a previously defined cost function in a finite time horizon [5]. The model is used to predict the future states along that time horizon. In addition, constraints in the related optimization problem can be specified in a quite straightforward way. Bhardwaj et al. [6] recently used MPC to control a robot under certain restrictions (e.g.: joint limits, collision avoidance, etc.) given its dynamic model. In our case, the plant to be modelled and controlled is a cloth garment. Cloth models are represented as high dimensional systems [7]. These dimensions are required for properly describing cloth behavior, but they make the control problem harder. In literature, different cloth models that simulate the internal dynamics of clothes are available [8]. The models consist in solving large systems of equations. However, the high cost of solving these systems limits their utility for real-time applications [9]. Modeling the dynamics of woven fabrics is a complex problem widely studied in computer graphics. Most models available in the literature need to be highly nonlinear to properly describe a realistic behavior [10]. These nonlinearities will affect the solving speed of an optimization problem. In other works like [11], the cloth model is included in the MPC design but the solution takes hours of computation.

Cloth manipulation has been a research challenge with successful cases such as the PR2 robot folding towels [12], where a vision-based grasping point detection algorithm is presented. The robot begins by picking up a dropped towel on a table, goes through a sequence of vision-based re-grasps and manipulations and finally stacks the folded towel in a target location. Despite the impressive results, it takes the robot almost 25 minutes to recognize the different states and complete the task. More recently, Yan et al. [13] used latent representations to learn the dynamic models of non-rigid objects and used them for planning sequences of actions in order to obtain a desired state of the manipulated object. The manipulation is, however, not real-time in the sense that the models learned are used for predicting the outcomes of the actions, rather than real-time control. In [11], the authors presented a technique to synthesize dexterous manipulation of cloth for physics-based computer animation. An optimization problem is formulated to find the commanding forces of the hand so that the cloth follows the reference motion. However, the computational cost is prohibitive for any real-world scenario. Moreover, in 2018 Erickson et al. [14] presented an MPC approach that allows a robot to reduce the force exerted during assisted dressing. Nonetheless, the cloth mathematical model was not included in their approach to the problem. In the present work, we are interested in incorporating the cloth model in the controller so that the robot always considers the motion of the cloth caused by the movement of its end-effector. Recent *fast folding* literature [15] focuses on the perception and action planning speed, rather than online correction of the actions. Applications such as folding cloth [16], the authors used their knowledge about the physics of a folding cloth task [17]. They pre-trained a neural network in simulation using domain randomization for robustness and detected a particular corner of the cloth being folded to correct the robot's motion accordingly. However, the method requires extensive training (in simulation) for each task to be learned. In our case, the reference motion of the cloth will be represented by the desired motion of certain *interest points*. To the knowledge, MPC of cloth manipulation or a task-independent framework for real-time robotic control as addressed in this article is not found in the literature. Several intrinsic features of such a control strategy are quite convenient when performing automatic cloth manipulation, e.g., the natural handling of uncertainty given by the benefits of the online optimization (fact that makes the control law time-varying) and the explicit consideration of physical/operational constraints. It is well known that MPC could suffer from the potentially high computational burden. However, this paper proposes a predictive control strategy able to run in real time and with the suitable stability/feasibility conditions no matter the considered complex setup.

The use-case of this paper is the design, simulation and final implementation on a real setup of a closed-loop control strategy aimed to improve robotic cloth manipulation, by using an MPC that includes a dynamic cloth model which satisfies both physical and operational constraints. This controller finds the optimal control inputs (motions of the *grasped points*) that yield minimum tracking error, predicting the behavior of the interest points using a cloth Control-Oriented Model (COM). This model, built and validated with captured evolved trajectories of a real cloth piece, focuses on describing the motion of the aforementioned interest points accurately when given the control signals (motions of the grasped points), in contrast with other cloth dynamic models that try to describe the behavior of the entire piece accurately. RL techniques are used in order to validate the model and find the optimal tuning to reduce tracking error. These contributions are validated in experiments executed in a real robot in real time.

This paper is structured as follows: Section II presents the cloth manipulation problem. Then, Section III explains our proposed solution: the linear cloth model and the design of the MPC controller. Section IV reports on the case study, specifying differences between simulations and real setup. Section V present the results, including model validation, controller tuning, simulations and trajectory tracking in a real setup. Finally, Section VI draws conclusions of the achieved results and considers future work directions.

II. PROBLEM STATEMENT

The main application of this paper is to control the movement of a piece of cloth so that certain parts of such cloth, the *interest points*, track a reference trajectory, by controlling other *grasped points*. The proposed framework serves as a proof of concept showing the potential of applying MPC to cloth

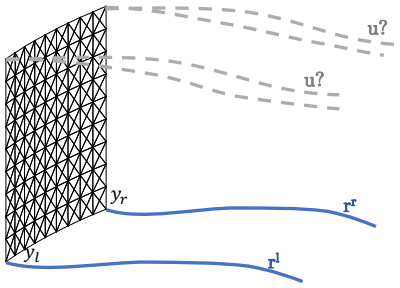

Fig. 2: Representation of the trajectory tracking problem

manipulation. We do not include collision scenarios, leaving this aspect as further work. We assume the robot is holding the piece of cloth in the air with two pinch grasps. The MPC will generate changes in the position (x, y, z coordinates) of the two upper corners, which are our grasped points $\boldsymbol{u} \in \mathbb{U} \subset \mathbb{R}^6$ (see Fig. 2), where $\mathbb{U} = \{\boldsymbol{u} \in \mathbb{R}^6 : \underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \overline{\boldsymbol{u}}\}$, with $\underline{\boldsymbol{u}}$ and $\overline{\boldsymbol{u}}$ being the minimum and maximum control signals, respectively. This situation can also be adapted to a single robot arm with a rigid link between the grasped points.

In most manipulation applications, the positions of all the points of the piece of cloth are not relevant for its control, and the efforts can focus in tracking *interest points*, which have a known relationship with the grasped ones, for example a dynamical model. In our case, defining the trajectory of both lower corners $\boldsymbol{y} = \{\boldsymbol{y}_r, \boldsymbol{y}_l\}^T \in \mathbb{Y} \subset \mathbb{R}^6$ is enough to generate a reference trajectory to track, namely $\boldsymbol{r} = \{\boldsymbol{r}_r, \boldsymbol{r}_l\}^T \in \mathbb{Y} \subset \mathbb{R}^6$ (see Fig. 2), where $\mathbb{Y} = \{\boldsymbol{y} \in \mathbb{R}^6 : \underline{\boldsymbol{y}} \leq \boldsymbol{y} \leq \overline{\boldsymbol{y}}\}$, with $\underline{\boldsymbol{y}}$ and $\overline{\boldsymbol{y}}$ being the minimum and maximum position values for each lower corner, respectively. Controlling the position of the lower corners by only moving the upper corners (the grasped ones) is not a simple task, specially when they are far from each other. Dynamic relationships among all the points of the piece of cloth are mostly nonlinear and depend on both the position and velocity of the other points of the cloth. Therefore, we chose to use MPC to foresee the effects of the robot actions on the cloth.

MPC uses a dynamics model to predict the future behavior of the system states (and then outputs) along a time-ahead horizon, while physical and operational constraints are satisfied. In the context of this paper, the predictive control strategy requires a mathematical model to predict the cloth behavior. This model must resemble the real cloth statically and dynamically. Multiple theoretical cloth models can be found in the literature, but most of them are highly nonlinear and not appropriate for real-time control - considering that the MPC must integrate a segment of the trajectory at every timestep using such model- but suitable for simulation. Therefore, we propose to obtain a linear cloth model, faster but accurate enough, such that the designed MPC strategy can be used in real time, assuming a possible loss of realistic system evolving dynamics in extreme circumstances (e.g., fast dynamics). This reduced model is described in detail Section III. Thus, as previously explained, our idea is to use a realistic Simulation-Oriented Model (SOM) to represent the real cloth in simulation, while the simpler but accurate enough Control-Oriented Model (COM) is to be used for control purposes.

Although the control strategy has a certain degree of inherent robustness, and the controller outputs an optimal solution to its optimization problem, its formulation includes several parameters, weights and alternatives that might help reaching minimum tracking errors. We propose the usage of RL techniques to learn the best controller structure and its optimal tuning.

To sum up, the problem consists in finding the optimal sequence of control inputs ($\boldsymbol{u}$) via MPC to manipulate a cloth, making the interest points ($\boldsymbol{y}$) track a desired reference ($\boldsymbol{r}$) while satisfying the physical and operational constraints. To this end, we will find a fast COM for real-time implementation, able to reproduce the system dynamics to be controlled, then use RL to learn its parameters and validate the model against the behavior of real cloth pieces. The optimal tuning of the controller together with RL will be found to minimize tracking error in closed-loop.

III. PROPOSED SOLUTION

The three subsections that follow define the cloth COM to use in our predictive controller (Section III-A), the control optimization problem (Section III-B), and specific real-time requirements (Section III-C).

A. Control-Oriented Model Definition

We propose to use the simplest –and cheapest to evaluate– cloth model: a mass-spring-damper system [18]. This model should be able to reproduce the system dynamics to be controlled afterwards, while maintaining a high degree of accuracy wrt. the real cloth. While there are many ways of building dynamics models, the one presented here is a computationally efficient modification of the typical spring-damper system, that allows the MPC controller to find the proper sequence of future actions to perform. We use the $\mathcal{L}1$ norm for the inter-vertex distance, which makes the dynamics of the cloth a linear system, and define the initial elongation of the cloth so as to consider the cloth's own weight. As the camera will provide feedback on the cloth dynamic evolution, a simple model is enough for predicting the cloth's immediate future behaviour. The resultant COM has been validated against a complex nonlinear model reported in [19], reaching suitable performance with bounded errors less than 10%.

Here, the cloth is treated as a system of particles (nodes) interconnected with spring-dampers. Usually, mass-spring models use three types of connections between nodes to give them more realism [18]: structural springs, shear springs and flexion springs. Our goal is not the most realistic model but a fast one that describes the lower-corners behavior along a prediction horizon H_p. Note that, whilst we want such a model to fit the behavior of the lower corners as realistically as possible, the behavior of rest of the mesh is only enforced indirectly. Hence, we are only going to use structural springs as shown in Figure 3 with a mesh of N nodes. In a general case, $N = n_r \times n_c$, with n_r and n_c being the number of nodes per row and column, respectively, but in a square mesh, we have $N = n^2$.

The system in Figure 3 can be represented as a graph of N nodes belonging to a set $\mathcal{I} = \{1, 2, ..., N\}$ and a set of edges $\mathcal{E}$

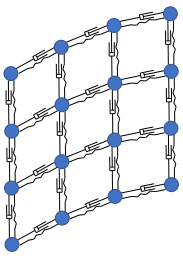

Fig. 3: Mass-spring-damper system with structural springs ($N = n^2 = 16$)

that represent the springs connecting those nodes. Moreover, a set of neighbouring nodes related to the i-th node is defined as $\mathcal{N}_i = \{j : (i,j) \in \mathcal{E}\}$, where its cardinality is $|\mathcal{N}_i| = \rho$. In our case, we only use structural springs –which yield a sufficiently proper results and save computational costs–, therefore, $\rho = 4$ for the interior points of the mesh. For each particle $i \in \mathcal{I}$, we can compute the resulting force of the springs $\boldsymbol{S}_k^i$ in time step $k \in \mathbb{Z}_{\geq 0}$ as

$$\boldsymbol{S}_k^i = \sum_{\mathcal{N}_i} -K \frac{\|\boldsymbol{d}_k^j\| - l_0^j}{\|\boldsymbol{d}_k^j\|} \boldsymbol{d}_k^j,$$

where K is the spring stiffness, $\boldsymbol{d}_k^j \in \mathbb{R}^3$ is the vector pointing from the neighbor $j \in \mathcal{N}_i$ to the particle $i \in \mathcal{I}$ and $l_0^j \in \mathbb{R}_{>0}$ is the initial length of the spring connecting that j neighbor with the i-th particle. The resulting force of the dampers $\boldsymbol{D}_k^i$ is calculated as $\boldsymbol{D}_k^i = \sum_{\mathcal{N}_i} -b(\boldsymbol{v}_k^i - \boldsymbol{v}_k^j)$, where b is the damping constant and $\boldsymbol{v}_k \in \mathbb{R}^3$ is the linear velocity of a particle. Moreover, $i \in \mathcal{I}$ and $j \in \mathcal{N}_i$.

In addition, each particle is affected by the force of gravity $\boldsymbol{G}_k^i = -m^i \boldsymbol{g}$, where m^i is the mass of the particle $i \in \mathcal{I}$ and $\boldsymbol{g}$ is the gravity constant. By adding these three terms, we compute the total force $\boldsymbol{F}_k^i$ applied to the particle as $\boldsymbol{F}_k^i = \boldsymbol{S}_k^i + \boldsymbol{D}_k^i + \boldsymbol{G}_k^i$. Once we have computed all the forces, we can use Newton's Second Law to compute the acceleration as $\boldsymbol{a}_k^i = \boldsymbol{F}_k^i / m^i$.

Having the accelerations of the nodes, we can integrate them over time to obtain the positions and velocities. We describe this process in the next subsections.

1) Integration of the model: To integrate the acceleration and obtain the position and velocity of the nodes, we can use either implicit or explicit integration methods. The former are stable but require solving large systems of equations [9]. The high cost of solving these systems limits their utility for real-time applications. Instead, explicit integration methods are fast but could have stability problems. To avoid such issues, we can put enough damping $\boldsymbol{D}_k^i$ in the system so that the energy decreases in a single time step [20]. We propose to use explicit Euler's integration method with a given time step Δt to update positions and velocities similarly as in [9], with:

$$\begin{aligned} \mathbf{p}_{k+1}^i &= \mathbf{p}_k^i + \Delta t \mathbf{v}_k^i \\ \mathbf{v}_{k+1}^i &= \frac{dt}{m} \mathbf{v}_k^i + \Delta t \mathbf{a}_k^i. \end{aligned}$$

2) Linearizing the spring force dynamics: Even if we have proposed a fast spring-damper model, the spring force vector $\boldsymbol{S}_k^i$ is nonlinear since it depends on a quadratic norm of state variables multiplied by other state variables. This norm computes the spring length in three dimensions along $k \in \mathbb{Z}_{\geq 0}$ as $\|\boldsymbol{d}_k^j\| = \sqrt{(p_{x,k}^i - p_{x,k}^j)^2 + (p_{y,k}^i - p_{y,k}^j)^2 + (p_{z,k}^i - p_{z,k}^j)^2}$. Instead of computing the $\mathcal{L}2$-norm, we propose to calculate it as an $\mathcal{L}1$-norm to linearize it by creating three linear springs, one for each direction of the space. Using this norm, we propose to associate different stiffness constants for each direction k_x, k_y, k_z so as to preserve the different behavior in each direction, i.e.,

$$\boldsymbol{S}_k^i \approx \sum_{\mathcal{N}_i} \begin{bmatrix} -k_x(p_{x,k}^i - p_{x,k}^j - l_{0x}^j) \\ -k_y(p_{y,k}^i - p_{y,k}^j - l_{0y}^j) \\ -k_z(p_{z,k}^i - p_{z,k}^j - l_{0z}^j) \end{bmatrix}.$$

With this transformation, we have a whole linear COM that can be expressed through a state-space realization as

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{f}_{ct}, \\ \boldsymbol{y}_k &= \boldsymbol{C}\boldsymbol{x}_k, \end{aligned} \tag{1}$$

where $\boldsymbol{x} \in \mathbb{X} \subset \mathbb{R}^{6N}$ is the vector of the cloth states (position p and velocity v in x, y, z of nodes $i \in \mathcal{I}$), $\boldsymbol{u} \in \mathbb{U} \subset \mathbb{R}^6$ is the control input vector, $\boldsymbol{y} \in \mathbb{Y} \subset \mathbb{R}^6$ is the output vector (positions of the lower corners) and $\boldsymbol{f}_{ct} \in \mathbb{F} \subset \mathbb{R}^{6N}$ is the vector of constant forces applied to each node: gravity and natural spring length force. Moreover, $\boldsymbol{A} \in \mathbb{R}^{6N \times 6N}$, $\boldsymbol{B} \in \mathbb{R}^{6N \times 6}$, $\boldsymbol{C} \in \mathbb{R}^{6 \times 6N}$ are the system state-space matrices and $\mathbb{X} = \{\boldsymbol{x} \in \mathbb{R}^{6N} : \underline{p}_x \leq p_x \leq \overline{p}_x,\ \underline{p}_y \leq p_y \leq \overline{p}_y,\ \underline{p}_z \leq p_z \leq \overline{p}_z,\ \underline{v}_x \leq v_x \leq \overline{v}_x,\ \underline{v}_y \leq v_y \leq \overline{v}_y,\ \underline{v}_z \leq v_z \leq \overline{v}_z\}$, where $\underline{p}_i$ and $\overline{p}_i$ represent the minimum and maximum position values for each direction, respectively, whereas $\underline{v}_i$ and $\overline{v}_i$ are the corresponding minimum and maximum velocities. Note that the resultant COM in (1) is, in fact, not a linearization of the dynamics equations, but it is obtained by substituting the terms that correspond to nonlinear dynamics: the spring elongation forces.

We previously discussed that the addition of the damping term guarantees the stability of the system: If we have different spring constants for each direction, we also need different damping constants to reduce the energy in each direction. Similarly as before, we compute $\boldsymbol{D}_k^i$ as

$$\boldsymbol{D}_k^i \approx \sum_{\mathcal{N}_i} \begin{bmatrix} b_x(v_{x,k}^i - v_{x,k}^j) \\ b_y(v_{y,k}^i - v_{y,k}^j) \\ b_z(v_{z,k}^i - v_{z,k}^j) \end{bmatrix}.$$

3) The super-elastic problem: Elasticity is the major drawback of the mass-spring cloth model, as it might stretch under its own weight [18]. Two solutions are found in the literature: making the springs stiffer or applying a maximum deformation rate. The former has its limits, as it can unstabilize the model. With the latter, the elongation of the springs is corrected if it is over a 10% of the initial length, making the model nonlinear, and thus the computations slower.

To solve the super-elastic problem, we propose to shorten the initial length of the linear springs in the vertical direction in simulation by Δl_{0z}. This results in an equilibrium in the vertical direction when gravity is applied, avoiding the mesh stretching under its own weight (see Fig. 4).

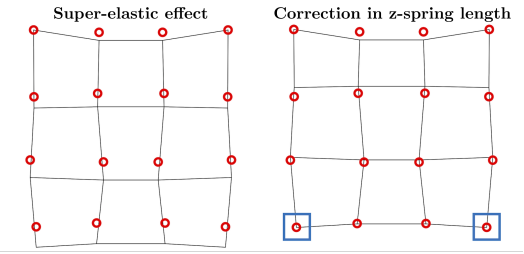

Fig. 4: Mesh positions showing the super-elastic problem (left) and the result of applying Δl_{0z}, correcting it (right)

B. Control Strategy

The main objective of a tracking MPC strategy is to minimize the tracking error while satisfying the system constraints. Closed-loop stability can be guaranteed if the initial state is inside the feasibility region $\boldsymbol{x}_0 \in \mathbb{X}$ and the evolution of the state trajectories remain inside the polytope defined by the system/operational constraints. The controller has been obtained considering the following elements:

1) Cost function: The goal is to find a sequence of control inputs, i.e., the positions of the controlled cloth points, namely

$$\boldsymbol{u}_k^s \triangleq \{\boldsymbol{u}_0, \dots, \boldsymbol{u}_{H_p-1}\}, \tag{2}$$

along a prediction horizon H_p such that the lower right and left nodes $\boldsymbol{y} \in \mathbb{Y}$ follow the desired trajectories $\boldsymbol{r} \in \mathbb{Y}$ as accurately as possible. Moreover, we also want to obtain smooth trajectories, reducing noise and sudden changes in acceleration. To this end, reducing the changes in consecutive control signals or *slew rates* (i.e., $\boldsymbol{\Delta u}_i = \boldsymbol{u}_i - \boldsymbol{u}_{i-1}$) instead of the control inputs has been proven to yield better results (Section V). The final multi-objective cost function at used at each time step $k \in \mathbb{Z}_{\geq 0}$ is

$$J_k = \sum_{i=0}^{H_p-1} \|\boldsymbol{y}_{k+i+1|k} - \boldsymbol{r}_{k+i+1}\|^2_{\boldsymbol{Q}} + \|\boldsymbol{\Delta u}_{k+i|k}\|^2_{\boldsymbol{R}}, \tag{3}$$

where the notation $(k+i|k)$ refers to the prediction at time step $(k+i)$ based on measurements made at time step k and integrating the dynamics with the control commands given by the MPC controller for the next i steps. For $i = 0$, $\boldsymbol{\Delta u}_{k|k} = \boldsymbol{u}_{k|k} - \boldsymbol{u}_{k-1|k}$. This $\boldsymbol{u}_{k-1|k} = \boldsymbol{u}_{k-1}$ is the control input applied in the previous step, and must be given as an initial condition. Equation (3) will be optimized to find $\boldsymbol{\Delta u}_{k|k}$, and $\boldsymbol{u}_{k|k} = \boldsymbol{\Delta u}_{k|k} - \boldsymbol{u}_{k-1|k}$ Note that $(\boldsymbol{Q}, \boldsymbol{R}) \in \mathbb{R}^{6\times 6}$ are the weighting matrices of suitable dimensions that prioritize each term of the multi-objective cost function. Note that, since the proposed control approach is implemented at a supervisory level, the displacements generated by the predictive controller (control signals) are the setpoints to be sent to a regulatory level (lowest control layer). There, PID controllers (default robot controllers) are used to convert these displacements of the grasping points to robot torques. Therefore, robot torques lie out of the scope of this paper.

2) System constraints: The robot arms that manipulate the cloth have restrictions of velocity $\boldsymbol{v}_{max}$ and workspace $\mathbb{U}$, providing bounds for the search of the optimal computed control input.

3) Optimization problem formulation: Based on the receding horizon strategy [5] and considering a fixed prediction horizon H_p, the goal is to obtain the input sequences that minimize the tracking errors of the two lower corners while satisfying all constraints. The controller is based on the solution of the following discrete-time open-loop optimization problem (OOP):

$$\min_{\boldsymbol{u}_k^s} J_k, \tag{4a}$$

subject to

$$\boldsymbol{x}_{k+i+1|k} = \boldsymbol{A}\boldsymbol{x}_{k+i|k} + \boldsymbol{B}\boldsymbol{u}_{k+i|k} + \boldsymbol{f}_{ct} \tag{4b}$$
$$\boldsymbol{y}_{k+i|k} = \boldsymbol{C}\boldsymbol{x}_{k+i|k} \tag{4c}$$
$$\boldsymbol{x}_{k|k} = \boldsymbol{x}_k \tag{4d}$$
$$\boldsymbol{u}_{k-1|k} = \boldsymbol{u}_{k-1} \tag{4e}$$
$$\boldsymbol{x}_{k+i+1|k} \in \mathbb{X} \subseteq \mathbb{R}^n \tag{4f}$$
$$\boldsymbol{u}_{k+i|k} \in \mathbb{U} \subseteq \mathbb{R}^m, \tag{4g}$$

$\forall i \in [0, H_p - 1]$, where $\boldsymbol{u}_k^s$ is the sequence of control signals in (2). Besides, $\mathbb{X}$ represents the subspace of the physical/environmental constraints that can potentially affect the cloth state, and $\mathbb{U}$ is a ball centered around $\mathbf{u}_k$ with radius $H_p \cdot 1$cm, meaning a maximum change of 1 cm per timestep in the robot's end effector position.

Assuming the OOP (4) is feasible, there exists an optimal sequence solution $\boldsymbol{u}_k^{s,*} \triangleq \{\boldsymbol{u}^*_{k|k}, \boldsymbol{u}^*_{k+1|k}, ..., \boldsymbol{u}^*_{k+H_p-1|k}\}$. The applied control is $\boldsymbol{u}_k^{MPC} \triangleq \boldsymbol{u}^*_{k|k}$, $(i = 0)$, ignoring the rest of the sequence. The entire process is repeated at the next time instant k.

4) Weighting matrix tuning: The $\boldsymbol{Q}$ matrix in (3) is the responsible of penalizing the tracking errors for each coordinate of the lower corners, while $\boldsymbol{R}$ penalizes sudden changes and quick fluctuations in the control signals.

An adaptive tuning of the $\boldsymbol{Q}$ matrix is proposed. In each time step, we compute the distance vector between the current position and that of the reference at the end of the prediction horizon, finding the main direction $\boldsymbol{\beta}$ in which the lower corners move along H_p. By normalizing this vector, the final adaptive weighting matrix $\boldsymbol{Q}_a$ is computed as

$$\boldsymbol{\beta}_k = \frac{|\boldsymbol{r}_{k+H_p} - \boldsymbol{y}_{k|k}|}{\|\boldsymbol{r}_{k+H_p} - \boldsymbol{y}_{k|k}\| + \epsilon}, \quad \boldsymbol{Q}_{a,k} = \text{diag}\left[\boldsymbol{\beta}_k\right], \tag{5}$$

where an ϵ term has been included to avoid numerical problems when the corner is exactly at the reference. We must note that this adaptive term changes in each time step, as is made explicit by the k subscript.

This adaptive computation has been studied as a way to compute the optimal tuning along with different constant matrix structures. The results of are shown in Section V.

C. Real-Time Control

To work in real time, the controller needs to have a constant output rate with an updated control signal on every time step (sampling time T_s). It was observed that sometimes, depending on initial conditions, optimizations could take longer than the maximum allowed time (one step), thus slowing down the

executions. To avoid this, the solver was changed to run in parallel, and return not only $\boldsymbol{u}^*_{k|k}$ but the entire sequence $\boldsymbol{u}^{s,*}_k$ of H_p control signals. This way, the solver can be called with new initial conditions on every time step, but if an optimization takes longer than T_s to finish, the controller can output a sub-optimal solution coming from the most recently obtained control sequence. This can be done for a maximum of H_p steps, the length of each sequence, thus a hard time limit was added to cancel optimizations running for too long. In the end, a maximum time of $T_s H_p/4$ was set empirically, ensuring there are always at least four optimizations with different initial conditions in one prediction horizon.

D. Stability analysis

As previously mentioned, the closed-loop stability can be ensured when the system states remain inside the polytope described by the system constraints (starting from a feasible region, $x_0 \in \mathbb{X}$). However, in a real scenario the reference may change without a predefined deterministic law and therefore stability and feasibility are not guaranteed. To deal with this problem, [21] proposes an MPC formulation for tracking which ensures recursive feasibility and asymptotic stability when the reference value changes. This solution is based on using a reference governor and a predictive controller. The main idea of the reference governor is to introduce an artificial reference r^a computed to guarantee that the current state is inside the domain of attraction while tending to the reference r. To achieve that r^a tends to r, a term that penalizes the deviation $l(\boldsymbol{r}^a, \boldsymbol{r}) = \|\boldsymbol{r}^a - \boldsymbol{r}\|^2_{\boldsymbol{T}}$ is added in the cost function, where $\boldsymbol{T}$ is a weighting matrix. The reference r is generated with an exogenous model

$$\begin{aligned} \chi_{r,k+1} &= f_r\left(\chi_{r,k}, \boldsymbol{u}_{r,k}\right), \\ \boldsymbol{y}_{r,k} &= h_r\left(\chi_{r,k}\right), \end{aligned}$$

where $\chi_r \in \mathbb{X} \subset \mathbb{R}^{6N}$ is its vector of cloth states, $u_r \in \mathbb{U} \subset \mathbb{R}^6$ is its control input vector and $y_r \in \mathbb{Y} \subset \mathbb{R}^6$ is its output vector. Moreover, $f_r : \mathbb{X}_r \times \mathbb{U}_r \to \mathbb{X}_r$ and $h_r : \mathbb{X}_r \to \mathbb{Y}_r$ are nonlinear functions describing the cloth dynamics. This model is used in open-loop with given inputs u_r to obtain a reference $r = y_r \in \mathbb{Y}$. Notice that, at this point, the cost function in (3) becomes

$$J_k = \sum_{i=0}^{H_p-1} \|\boldsymbol{y}_{k+i|k} - \boldsymbol{r}^a_{k+i}\|^2_{\boldsymbol{Q}} + \|\boldsymbol{\Delta u}_{k+i|k}\|^2_{\boldsymbol{R}} + \|\boldsymbol{r}^a_{k+i} - \boldsymbol{r}_{k+i}\|^2_{\boldsymbol{T}}, \tag{6}$$

in order to include the objective of reference tracking to ensure recursive feasibility of the closed-loop scheme. Before formulating the optimization problem, we define the following sequences along a prediction horizon H_p: $\boldsymbol{\chi}^s = \{\boldsymbol{\chi}_1, \boldsymbol{\chi}_2, ..., \boldsymbol{\chi}_{H_p}\}$, $\boldsymbol{u}^s = \{\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_{H_p-1}\}$, $\boldsymbol{r}^{a,s} = \{\boldsymbol{r}^a_1, \boldsymbol{r}^a_2, ..., \boldsymbol{r}^a_{H_p-1}\}$ and $\boldsymbol{y}^s = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_{H_p-1}\}$. Then, the controller is based on the solution of the following discrete-time open-loop optimization problem (OOP):

$$\min_{\boldsymbol{u}^s, \boldsymbol{r}^{a,s}} J_k, \tag{7a}$$

subject to

$$\boldsymbol{\chi}_{k+1+j} = \boldsymbol{A}\boldsymbol{\chi}_{k+j} + \boldsymbol{B}\boldsymbol{u}_{k+j} + \boldsymbol{f}_{ct} \quad \forall j \in [0, H_p - 1], \tag{7b}$$

$$(\boldsymbol{\chi}_{k+1+j}, \boldsymbol{u}_{k+j}) \in \mathbb{X}^{H_p} \times \mathbb{U}^{H_p} \quad \forall j \in [0, H_p - 1], \tag{7c}$$

$$\boldsymbol{r}^a_{k+j} \in \mathcal{R}^{H_p} \quad \forall j \in [0, H_p - 1], \tag{7d}$$

$$(\boldsymbol{\chi}_{H_p}, \boldsymbol{r}^a_{H_p}) \in \Gamma = \mathbb{X} \times \mathcal{R}, \tag{7e}$$

where

$$\mathcal{R} = \{\boldsymbol{r} : (\boldsymbol{x}_r, \boldsymbol{u}_r) \in \mathbb{X}_r \times \mathbb{U}_r, f_r(\boldsymbol{x}_r, \boldsymbol{u}_r) = \boldsymbol{x}_r, \boldsymbol{y}_r = \boldsymbol{r}\}$$

and Γ is a terminal constraint set.

Assuming that the OOP (7) is feasible for $\boldsymbol{\chi} \in \mathbb{X}$, i.e., $\mathbb{U}^{H_p} \neq \emptyset$, there exists an optimal sequence solution $\boldsymbol{u}^{s,*}_k \triangleq \{u^*_{0|k}, u^*_{1|k}, ..., u^*_{H_p-1|k}\} \in \mathbb{U}^{H_p}$, and then the receding horizon philosophy [5] sets the MPC law $u_{\text{MPC},k} \triangleq u^*_{0|k}$ and ignores the computed control inputs from $k = 1$ to $k = H_p - 1$, repeating the whole process at the next time instant k.

To guarantee the stability of the presented MPC strategy, the following assumptions are considered.

Assumption 1: The pair ($\boldsymbol{A}$,$\boldsymbol{B}$) from (1) is stabilizable.

Assumption 2: The weighting matrices $\boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{T}$ are positive definite.

Assumption 3: Γ is an invariant set for tracking system (1). As shown in [22], a possible choice of the terminal constraint is $\Gamma = \{(\boldsymbol{y}, \boldsymbol{r}^a) : \boldsymbol{y} = \boldsymbol{r}^a\}$.

Then, the following result can be stated.

Theorem 1: (Limon et al. [21]) Consider that Assumptions 1, 2 and 3 hold. Consider that the target steady state $\boldsymbol{r}$ is admissible. Then, for any feasible initial state $\boldsymbol{\chi}_0 \in \mathbb{X}$, the proposed MPC controller asymptotically steers the system to $\boldsymbol{r}$ in an admissible way.

Proof: The formal proof follows the one presented in [21] taking into account the use of the terminal constraint set Γ from Assumption 3. ■

Remark 1: The target steady state $\boldsymbol{r}$ is admissible because it is obtained in simulation from the SOM.

Remark 2: Using the pair ($\boldsymbol{A}$,$\boldsymbol{B}$), we ensure closed-loop stability with the linear system (COM). Since we use the COM in a bounded prediction horizon and since the COM is validated with the nonlinear model (SOM), closed-loop stability with the SOM is also guaranteed.

Thus, stability of the closed-loop system can be guaranteed by using the artificial target $\boldsymbol{r}^a$, which differs from the reference $\boldsymbol{r}$ in order to guarantee recursive feasibility and finally converges to $\boldsymbol{r}$ to enforce asymptotic stability.

IV. CASE STUDY DESCRIPTION

This section starts with the specific changes that had to be incorporated to the previously defined solution for the used setup with only one robotic arm, described in Section IV-A. Then we divide the study in two, with simulations in Section IV-B and the real setup in Section IV-C.

A. Application to Single-Arm Manipulation

The presented formulation in (7) assumes the evolving of the two controlled corners are independent, as they are when using two different robot arms to pick them. In the considered case study, only one robot is used, and thus the upper corners are linked together with a rigid piece. This fact means they always keep a constant distance, adding a new quadratic constraint to the problem in (7). This new considered constraint changes such a problem from a linear programming to a Quadratically Constrained Quadratic Programming (QCQP) problem, but even if it is more complex, convexity is kept [23], and computations can be fast. Therefore, the new formulation of (7) is written as follows:

$$\min_{\boldsymbol{u}^s, \boldsymbol{r}^{a,s}} J_k, \tag{8a}$$

subject to

$$\boldsymbol{\chi}_{k+1+j} = \boldsymbol{A}\boldsymbol{\chi}_{k+j} + \boldsymbol{B}\boldsymbol{u}_{k+j} + \boldsymbol{f}_{ct}, \tag{8b}$$
$$(\boldsymbol{\chi}_{k+1+j}, \boldsymbol{u}_{k+j}) \in \mathbb{X}^{H_p} \times \mathbb{U}^{H_p}, \tag{8c}$$
$$\boldsymbol{r}^a_{k+j} \in \mathcal{R}^{H_p}, \tag{8d}$$
$$(\boldsymbol{\chi}_{H_p}, \boldsymbol{r}^a_{H_p}) \in \Gamma = \mathbb{X} \times \mathcal{R}, \tag{8e}$$
$$\left\| d^{uc}_{k+i+1|k} \right\|^2 = L^2, \tag{8f}$$

for all $j \in [0, H_p - 1]$, where d^{uc} is the distance vector between the upper corners.

The control signals obtained with the described MPC controller are the displacements of the upper corners of the cloth in one time step. However, to move one robot, we need a pose of its Tool Center Point (TCP), which must be computed with the available data. The position of the TCP can be obtained by taking the absolute positions of the upper corners of the cloth, $\boldsymbol{u}^{abs}$, computing the midpoint and adding a constant offset (Δh) introduced by the rigid piece that links both corners together. This offset is always in the Z_L axis of the local cloth base, so it must be transformed to global coordinates before being added as follows:

$$\begin{aligned} \Delta \boldsymbol{p}^L &= \begin{bmatrix} 0 \\ 0 \\ \Delta h \end{bmatrix} \rightarrow \Delta \boldsymbol{p} = \boldsymbol{R}^W_L \cdot \Delta \boldsymbol{p}^L, \\ \boldsymbol{p}_{TCP} &= \frac{1}{2} \begin{bmatrix} u^{abs}_1 + u^{abs}_2 \\ u^{abs}_3 + u^{abs}_4 \\ u^{abs}_5 + u^{abs}_6 \end{bmatrix} + \Delta \boldsymbol{p}. \end{aligned} \tag{9}$$

The orientation of the TCP is obtained with the same local reference frame, but inverting the Z_L axis to point away from the robot and into the cloth, as it is a convention in robotics. The other two axes are swapped for practical convenience, resulting in $R^W_{TCP} = [Y_L \ X_L \ \text{-}Z_L]$.

B. Simulation

The proposed control scheme is shown in Figure 5. In simulation, instead of using only two models, one inside the controller (the COM) and one to simulate the real cloth (the SOM), we add a third model as an intermediate step, namely "Backup" Simulation-Oriented Model (B-SOM). This is motivated by the real setup, where the feedback signal from the real cloth is the output of a computer vision algorithm, which is the slowest part of the scheme. Therefore, it takes multiple time steps to send an updated signal. This issue raises the necessity of having such a B-SOM model that can *i)* provide feedback to the control system if there is none at a certain step, and *ii)* can compensate, in the robotic application, for the time delay in processing the camera data.

Both linear models (COM and B-SOM) follow the equations presented in Section III, but they can have different sizes. The nonlinear model SOM substitutes the real cloth in simulation and is the one presented in [19], where a dynamics validation between their model and real cloth is performed (errors lower than 6%). Computationally speaking, this model employs finite elements to discretize the Lagrangian of the mechanical system (kinetic energy minus potential energy) but not the equations of motion. This way, Euler-Lagrange equations are obtained as a system of ordinary differential equations (ODE) instead of a partial differential equations (PDE), making integrations faster. Even then, computations are slower than required for a real-time application, and this nonlinear model cannot be used as COM, and is used only in simulation.

The nonlinear model can be discretized with a mesh of 10×10 nodes to achieve realistic behavior [19]. For the linear models, it was found that a mesh of 4×4 was enough for the considered trajectory tracking problem, as shown in Section V. The proposed solution to be able to use different mesh sizes together is to make the smaller ones be sub-meshes of the larger ones. This can be done in square meshes when the side sizes n follow $(n_L - 1) = p(n_S - 1)$ for some proportion $p \in \mathbb{Z}^+$. Knowing larger mesh sizes increase computational time, simulations were executed with sizes $n = 10$ and 13 for the nonlinear model, and $n = 4$ and 7 for the linear ones, testing cases where COM and B-SOM had different sizes or the same one. It can be checked how a mesh of $n = 4$ can be extracted from one of $n = 7$ and, in turn, this can be extracted from a larger mesh of $n = 13$.

To evaluate the trajectory tracking performance, we define two Key Performance Indicators (KPIs). The first one is related to the tracking error, and can be obtained first computing the Root Mean Squared Error (RMSE) of the obtained output over time, which results in a 6-dimensional vector $\boldsymbol{e}$. Then this vector is split into the two corners, to compute the norm of each error and the final average, i.e.,

$$\boldsymbol{e} = \text{RMSE}(\boldsymbol{y} - \boldsymbol{r}), \tag{10a}$$
$$\text{KPI}_1 = \frac{1}{2} \left(\|\boldsymbol{e}^r\| + \|\boldsymbol{e}^l\| \right), \tag{10b}$$
$$\text{KPI}_2 = \bar{\tau}. \tag{10c}$$

The second KPI is the average computational time per step τ, without counting the time needed to simulate the nonlinear model. This is used to check if optimizations are completed within a time step.

These two KPIs were evaluated executing the same 3D trajectory with a range of different prediction horizon H_p values to find a value that yielded low errors with times under T_s. The results for the case $T_s = 0.01$ s and $n = 4$ for both linear models are shown in Figure 6. The vertical blue line represents the threshold where errors are 10% larger

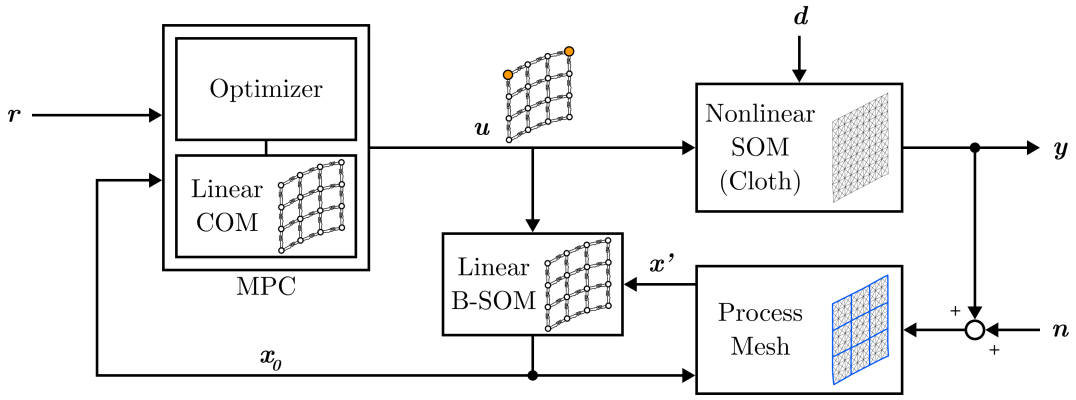

Fig. 5: Closed-loop control scheme. The nonlinear SOM substitutes the real cloth in simulation, and the linear B-SOM acts as a fast backup while the real feedback is processed, which in the real setup can take several time steps

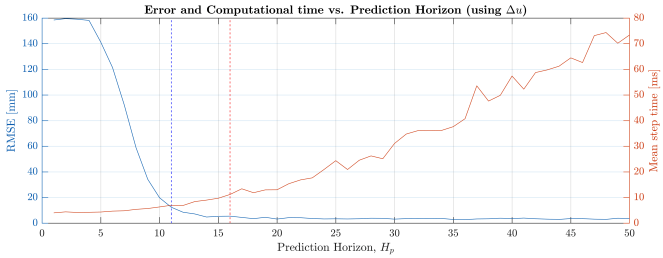

Fig. 6: KPIs depending on H_p, for $T_s = 10$ ms and $n = 4$

than the minimum value found at $H_p = 50$ steps. The red vertical line corresponds to the moment where times go over T_s. The window between these lines, $H_p \in [11, 16]$ represents the range of possible values with optimal results. In the end, $H_p = 15$ was chosen for experiments in these conditions, as it presented a smaller error within a tractable computational cost.

It is worth noting $T_s = 0.01$s is the most restrictive value among the tested ones. Larger time steps allow for larger prediction horizons. On the other hand, larger mesh sizes ($n = 7$) are slower, meaning this analysis had to be done for each specific case, leading to similar results.

C. Experimental Setup

The final closed-loop control scheme was implemented using ROS Kinetic [24]. Figure 7 shows the full diagram indicating the different nodes involved. The messages that are published at a fixed rate (one per time step) are marked with a solid line, and with dashed lines all which are not. The separation between dashes is proportional to how slow these messages are published, with the optimizer being the fastest among them (it can finish within one time step or take longer, depending on initial conditions). The **Vision Node** used needs around 100ms to process data and publish the mesh positions, thus its output and the output of the processing node are the slowest ones. The contents of each message are also specified, with $\boldsymbol{p}_V^C$ being the positions of all nodes expressed relative to the reference frame of the camera, $\boldsymbol{x}_V^W$ being the state vector obtained with Vision data in world base, and $\mathcal{P}_{TCP}^W$ is the pose of the TCP (position and quaternion) in global coordinates.

The **RT Node** processes the feedback data and updates the B-SOM state, the additional backup model needed to have updated data between real feedback samples, as discussed before. Every time step, the variable of initial parameters, $\boldsymbol{P}_0$, is updated and published to the **Optimizer Node**. The theoretical rate of the Opti Node is also set to $1/T_s$, but optimizations can take longer than that to complete. Once done, the full sequence of control inputs, $\boldsymbol{U}_{Hp} = \boldsymbol{u}^{s,*}$, is published and saved on the RT Node.

The **Cartesian Controller Node** is used to transform from a pose in Cartesian space to torques in joint space is the one developed in [25], wrapped as a ROS node. The hardware includes a 7-DoF Barrett WAM robot, used in all experiments with a custom gripper to hold the cloth by its upper corners (see Figure 8), and a Kinect camera to obtain RGB-D images as real feedback data.

These images were processed with the **Vision Node** [26], obtaining all node positions for a mesh of any given size. As mentioned, this process is the slowest step in the scheme, at rates around 0.1s, creating the need of the B-SOM included in the RT Node.

The **Processing Node** transforms the vector of positions before closing the feedback loop. First, positions are expressed relative to the camera, so a change of base was performed. The camera reference frame can be obtained with available data, as we can obtain the pose of the end-effector from both references: using Forward Kinematics (FK) from the robot, and adding the known constant offset from the upper central point of the cloth that the camera is seeing. This can be written as

$$T_C^W = T_E^W \cdot T_C^E = T_E^W \cdot \left(T_E^C\right)^{-1}, \tag{11}$$

where W indicates the world/robot reference frame, C the camera and E the end-effector one. Transform T_E^C is computed in a calibration process as an average inside a time window (set experimentally to 20 s), as the camera has a noisy output, with mesh positions varying slightly from frame to frame even with the cloth being completely still. After calibrating, T_C^W is saved for the experiments.

Besides this change of reference, feedback data also had to be filtered. With the **Vision Node** being the slowest part of

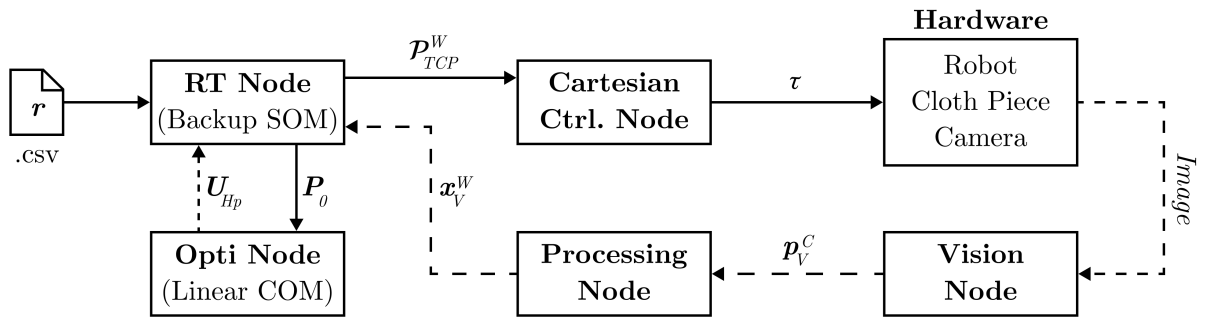

Fig. 7: ROS Diagram of the implemented control scheme, with nodes and connections between them

the system, waiting for the next sample to filter the current one would add a delay between four and ten steps, which was deemed too much and thus discarded. From the existing online filtering options using only current and past data points, the most commonly used ones are Moving Average (MA) filters [27], with several variants depending on the weights of the points considered. The disadvantage of using these filters is that their output corresponds to an average of the last several points, which in an actual movement means the filtered output lags behind the real position of the mesh, pulled back by the previous positions, adding a form of delay.

The considered variants of MA filters were Simple (SMA), Weighted (WMA) and Exponential (EMA). The same set of experiments was executed for all three variants and different weight values, as well as using no filter. The best results ($\text{KPI}_1 < 6$ cm, $\text{KPI}_2 < T_s$) were obtained using EMA, i.e.,

$$\boldsymbol{y}_f(k) = \alpha \boldsymbol{y}(k) + (1-\alpha)\boldsymbol{y}_f(k-1), \tag{12}$$

with $\alpha = 0.66$, also obtained empirically. Therefore, this was the selected filter for all experiments. Other approaches such as [28] report novel strategies to be used in the design and implementation of filters for our context, showing promising results that can be incorporated within the cloth manipulation setup as further research in the future.

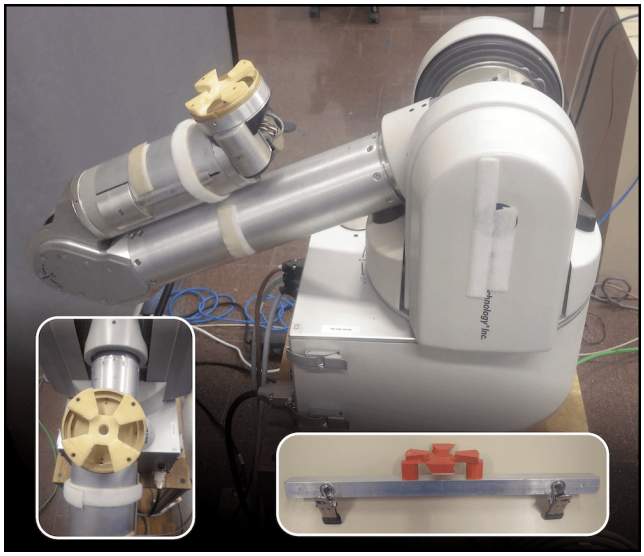

Fig. 8: Picture of the WAM used in the real setup, with details of its end-effector and piece that connects to the cloth

The filtered feedback data is sent back to the MPC node. To avoid spurious and unreliable data affecting the system, a final filtering step discards data points with delays or distances to the simulated B-SOM over certain thresholds. In the real implementation, the B-SOM satisfies another important function when real data is received. The incoming sample can be received on the RT node with a significant delay with respect to when it was captured (t_c). Setting this data directly as the current initial state can lead to unsuccessful tracking, hence the need to update it from t_c to current time. This is done with another instance of the B-SOM simulating the required steps with a history of previously applied control signals until the current step, yielding $\boldsymbol{x}_V$. Finally, the B-SOM state vector $\boldsymbol{x}_{SOM}$ is updated with the new sample with

$$\boldsymbol{x}_{SOM} \leftarrow W_V \boldsymbol{x}_V + (1 - W_V)\boldsymbol{x}_{SOM}, \tag{13}$$

where W_V is the Vision weight, analogous to an observer gain, added to reduce the effect of the remaining noise to the MPC and obtain smoother evolved trajectories.

As a summary of all the steps the captured Vision data must go through to close the loop, we show Algorithm 1.

V. RESULTS

All simulations were performed using CPU power only on an i7-8550U @1.80GHz with 8GB RAM using the optimization toolbox CasADi [29] in MATLAB. All real experiments were executed with the described setup, using ROS Kinetic (written in C++). All executions were done with a square piece of cloth of 30×30 cm.

A. Model Validation

The linear model presented in (1) has seven parameters to tune in order to follow the same behavior as a real cloth: spring stiffness k_x, k_y, k_z, damping b_x, b_y, b_z and the Δl_{0z} length previously introduced. It was found that these parameters depend on mesh size and sampling time T_s. Therefore, a combination of parameters was found from training data for each tested situation.

First, we gathered data of real cloth (open-loop executions, without MPC) following a set of trajectories, to ensure movement in all directions of space. After being filtered and regularized in time, these trajectories were used to find the parameters of the linear model via black-box optimization. We used a direct policy search method called Relative Entropy

Algorithm 1 Closing the loop with Vision feedback data

Require: Camera publishing RGB-D images
for each image captured at time t_c **do**
  Use Vision Node to get the mesh positions $\boldsymbol{p}(t_c)$
  Apply EMA to obtain $\boldsymbol{p}_f(t_c)$
  Order the node positions left to right, bottom to top
  Apply a base change from camera to world reference
  Add velocities to obtain full state vector
  Publish mesh using acquisition time stamp, $\boldsymbol{x}_V(t_c)$
  Receive mesh on MPC Node: callback function
  Compute delay $\Delta t = t - t_c$
  if $\Delta t > \Delta t_{max}$ **then**
    Discard feedback data
    Exit callback function
  else
    Update data using B-SOM $\lceil \Delta t / T_s \rceil$ steps, get $\boldsymbol{x}_V(t)$
    if $\|\boldsymbol{x}_V(t) - \boldsymbol{x}_{SOM}(t)\| > \Delta d_{max}$ **then**
      Discard feedback data
      Exit callback function
    else
      $\boldsymbol{x}_{SOM}(t) \leftarrow W_V \boldsymbol{x}_V + (1 - W_V)\boldsymbol{x}_{SOM}$
    end if
  end if
  Exit callback function
end for

Policy Search (REPS) algorithm [30], which assumes a Gaussian distribution over the parameters and iteratively converges towards a better solution by maximizing the expected reward while keeping the Kullbach-Leibler divergence between the updated distribution and the previous one bounded. As a reward, we use a function penalizing the squared errors in node positions for the entire mesh, but with higher weights for the lower corner nodes, as they will be the output to be tracked. This parameter optimization process is performed prior to the solution of the optimization problem in (8) related to the predictive controller. Therefore, the parameters of the model are not changing online during execution, preventing the loss of recursive feasibility.

Learning experiments were conducted to learn the model parameters for square meshes of sizes $n = 4$ and 7, and for time steps of $T_s = 10,\ 15,\ 20$ and 25 ms, obtaining results for all eight combinations. Figure 9 shows the evolving of the lower corner positions for the learnt linear model ($n = 4,\ T_s = 15$ ms) and the real cloth (dashed black line). We can see how both evolving trajectories have the same behavior, with a final RMSE of 1.5 cm.

Even though errors increase with $n = 7$ and larger time steps due to small oscillations that try to capture the nonlinear behavior of the real cloth, the errors are always within the same order of magnitude, and the lower corner evolving trajectories of the linear model have the same behavior as the real ones. Therefore, the model in (1) is validated with a real cloth, with parameter combinations for all studied cases.

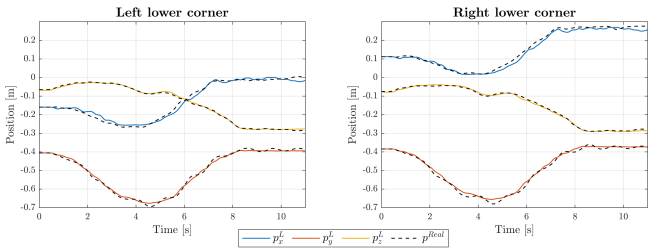

Fig. 9: Lower corner evolving trajectories of the learnt linear model ($n = 4,\ T_s = 15$ ms) and the data gathered from the real cloth (dashed line)

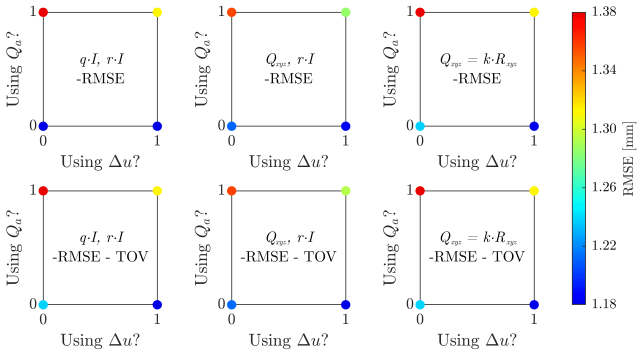

Fig. 10: RMSE obtained with all the considered structure options

B. Controller Structure and Tuning

Given the amount of parameters and weights present in the predictive controller, REPS was also used in closed-loop simulations to find its optimal structure and tuning. All experiments were made to learn the weight values inside matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ depending on different conditions, with the following options:

- Using the adaptive weight in (5) or not to obtain $\boldsymbol{Q}$.
- Minimizing the control signals $\boldsymbol{u}$ or the slew rates $\boldsymbol{\Delta u}$.
- Three different structures of matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$: scalars times the identity ($q\boldsymbol{I}, r\boldsymbol{I}$), with different weights per coordinate only for $\boldsymbol{Q}$ ($\boldsymbol{Q}_{xyz}, r\boldsymbol{I}$), or with both having different weights per coordinate, but with the matrices being proportional to each other ($\boldsymbol{Q}_{xyz} = k\boldsymbol{R}_{xyz}$).
- A reward function only penalizing tracking RMSE or also adding a cost for computational times over the considered step time (TOV $= \max(\bar{\tau}/T_s - 1,\ 0)$).

Altogether results in a total of 24 learning experiments executed. We can organize their results in groups with the same weight structure and reward function, and make comparisons purely based on the use of $\boldsymbol{\Delta u}$ and $\boldsymbol{Q}_a$, as shown in Figure 10.

The worst results are always obtained using $\boldsymbol{Q}_a$ and $\boldsymbol{u}$ regardless of weight structure and reward function, and the opposite selection, with a constant $\boldsymbol{Q}$ and $\boldsymbol{\Delta u}$, yields the best outcome. This is the reason behind using $\boldsymbol{\Delta u}$ in (8).

Differences in computational time are minimal, with only a slight time increase when using $\boldsymbol{\Delta u}$. Knowing how time measurements are sensitive to memory and CPU usage, and obtaining roughly the same results with both reward functions, the second one, with TOV, was discarded for the following experiments.

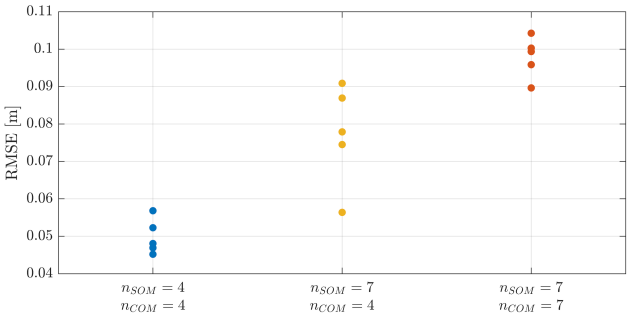

Fig. 11: Obtained RMSE using different linear model sizes

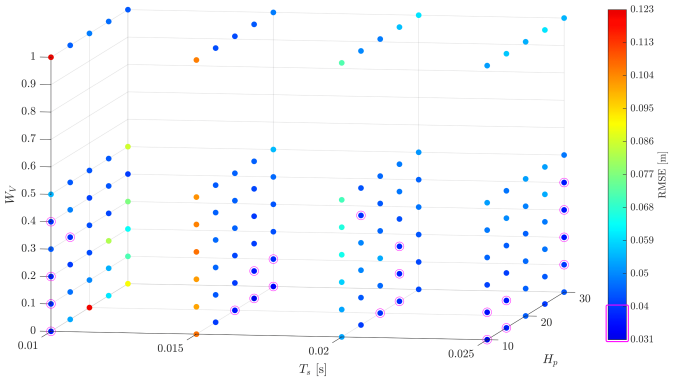

Fig. 12: Results of all executed experiments depending on T_s, H_p, W_V

The final tracking error is approximately the same regardless of the chosen weight structure (less than 0.5% difference between best and worst), and the minimum errors are actually obtained with the first and simplest option. This means that the more complex structures, with weights depending on direction, do not adapt better to the trajectory leading to smaller errors. Experiments on two other trajectories confirmed this tendency, which, together with simplicity, pushed us to select the first structure, with just two weights, leaving only one degree of freedom to learn as the final controller tuning: the proportion between $\boldsymbol{Q} = q\boldsymbol{I}$ and $\boldsymbol{R} = r\boldsymbol{I}$.

Proceeding with more learning experiments, it was observed that the tracking errors decreased with low $\boldsymbol{R}/\boldsymbol{Q}$ ratios, until a certain threshold where $\boldsymbol{R}$ is too small and the system can unstabilize. This behavior was found to be consistent across all tested trajectories and conditions, with the threshold only varying slightly between them.

This led us to set a unique tuning to guarantee the best performance in any given situation. The idea is getting as close as possible to the limit without crossing it for any case. A possible combination that works safely for all considered trajectories is $\boldsymbol{R}/\boldsymbol{Q} = 0.2$ (e.g., $q = 1,\ r = 0.2$). It is worth noting that, even for the case where the threshold had the lowest ratio, using 0.2 increased the error less than 1 mm, which does not represent a great sacrifice in performance.

C. Trajectory Tracking Results

With the scheme successfully tested in simulation, real experiments were conducted with the conditions explained in Section IV. To obtain the best trajectory tracking, an experimental analysis was conducted with two parts. The first one compared results with different linear model sizes in the same set of situations, while the second analyzed the effects of the remaining control parameters ($T_s,\ H_p,\ W_V$) on tracking performance.

1) Linear model sizes: Given that we have two available sizes and two linear models, and that the COM must be simpler or the same as the B-SOM, there are three possible combinations. Executing them in five different situations, we obtained the results shown in Figure 11. It is clear how larger mesh sizes yield worse results, meaning the increase in computational time is more important than the improvement in accuracy. Therefore, size $n = 4$ was selected for both linear models in the following experiments.

2) Control analysis: The range of T_s values analyzed comes from the obtained parameters through learning, being 10, 15, 20 and 25 ms. The prediction horizon H_p can take any arbitrary ($\mathbb{Z}^+$) value, but thanks to the executed simulations, we have an indicative range with relatively low errors without increasing computational times over the limit. As the conditions in the real setup are different with regards to timing and feedback data, horizons were tested from 10 to 30 steps, in increases of 5 (10, 15, 20, 25 and 30). Finally, the weight W_V in (13) was tested in increases of 0.1 from 0 to 0.5. While higher values were also tested, the majority of combinations resulted in unsuccessful executions. This results in a total of 120 finished real-robot experiments under the same conditions except for these three parameters. After they were completed, however, some additional changes were made to test executions with higher W_V. They only worked after reducing the Cartesian controller gains and using a filter with $\alpha = 0.5$, but all cases with $W_V = 1$ finished correctly. This makes a total of 140 experiments, with all their results shown in Figure 12. Given the conditions for $W_V = 1$ had to be different from the rest, and that the obtained results were consistently worse than their $W_V = 0.5$ counterparts, no values of W_V were tested in between to keep them separated and focus on the 120 executed with the same conditions. It is worth noting that the worst result within these 120 experiments had an RMSE of 12.1 cm, thus adding the additional experiments does not change the color scale significantly. We have also marked all results within the best 10% of errors, i.e., with an RMSE lower than 4 cm with a magenta circle.

With these results, we can see how a low T_s combined with a high H_p does not produce correct tracking. During execution, these experiments produced several timeouts on the optimizer, trying to keep up with a very fast rate while predicting a lot of steps into the future. Other issues were detected during the execution of all experiments with $T_s = 10$ ms and $W_V \geq 0.3$, where feedback data was consistently being discarded either for it being too much into the past (could not be updated) or too distant to the simulated state of the B-SOM. This means that these experiments actually have an effect of the feedback data closer to $W_V = 0$ than their actual value, as most of the time the real data was unusable. Furthermore, a smaller time step results in an increased difference between the rate at

which the control signals are computed and sent to the robot and the rate at which the Vision node outputs feedback data, being around 10 times slower in these conditions. This makes computed errors less reliable, as there are fewer captured points to compare with the reference. With all this, even if we have some results with really good tracking errors using $T_s = 10$ ms, it is clear that this sampling time is too fast for the majority of cases, and must be avoided to obtain the best possible tracking results. For both $T_s = 15$ ms and $T_s = 20$ ms, a horizon of 10 steps is too short to track correctly, regardless of W_V. This effect disappears with $T_s = 25$ ms, having optimal results with the shortest H_p too, which means that it is a problem related to total prediction time and maximum allowed time for the optimizer.

We can also see a tendency of errors increasing with higher W_V, with some optimal results being obtained without considering the Vision feedback at all. This is a product of noise, present even after filtering. In the executed experiments, there were no strong rotations, sudden movements, offsets, nor other disturbances (e.g., wind or human actions), and the B-SOM always started in the exact same position as the real cloth, making its simulated evolution an accurate one without any added noise. Of course, $W_V = 0$ means the control loop is not closed, and cannot be applied in a general scenario, where external forces or initial deviations can make the B-SOM state have an unreliable evolution, and the real feedback would have to correct its state. Unfortunately, with the current camera and Computer Vision algorithms, this comes at the cost of updating the initial state of the MPC with noisy data, which can increase optimization times. A general application of this scheme would need a camera with a faster refresh rate, more precision, not fixed in place to enable more movements and orientations without losing the cloth, and a fast and reliable algorithm to obtain an updated mesh.

All in all, even discarding cases with $T_s = 10$ms, $W_V = 0$ and 1, and the cases with $H_p = 10$, there is no delimited region with optimal results, but a tendency towards longer prediction horizons as the time step increases and the MPC module has more time to compute. Even then, the remaining cases, 65 different combinations, yielded errors lower than 5.5cm, which is not far from the previously considered threshold, and an acceptable error considering the large range of options it includes, as well as the precision of the camera and Vision algorithms used, in the order of centimeters.

3) Final tracking performance: Choosing $T_s = 20$ ms, $H_p = 25$ and $W_V = 0.2$ as an example of a combination that yielded optimal tracking, with an RMSE of only 3.8 cm, we can plot the evolved trayectories of all corners, as shown in Figure 13. We can see how some noise persists, but the reference is tracked accurately.

Trajectories with changes in orientation can also be tracked. In the real setup, we were limited by the fixed camera, as the Vision algorithm infers the mesh of the cloth when the majority is visible, with the best results being in a fully frontal perspective. Nevertheless, Fig. 14 shows a trajectory ending with a rotation of $45°$ followed by a counter rotation of $90°$.

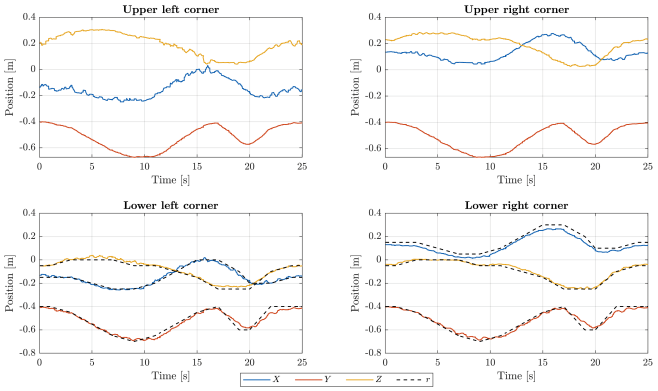

Fig. 13: Cloth corner evolved trajectories for $T_s = 20$ ms, $H_p = 25$, $W_V = 0.2$

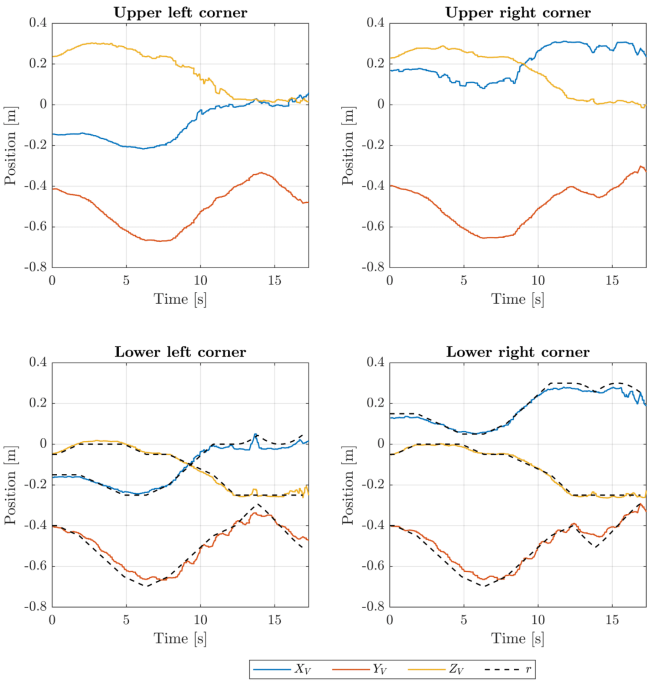

Fig. 14: Trajectory tracking with oscillating motion wrt frontal view.

D. Experiments with Disturbances

Two different cases were studied: blocking the camera and applying forces to the robot arm during execution.

1) Blocking the camera: During execution, a human walked between the camera and the cloth, covering its view for about two seconds on two occasions. The obtained results are shown in Figure 15, where shaded areas correspond to moments where the camera was blocked.

Sudden changes in captured mesh positions are caught before updating the state of the linear models and discarded, so even if we see them in the Vision data, they do not affect the controller. This is checked with the obtained TCP evolved trajectories, smooth even during these moments. This can be achieved thanks to the B-SOM inside the controller, which simulates the real evolution of the cloth during the moments where the vision feedback cannot provide updated data.

2) Human-robot interaction: The used Cartesian controller allows movements in the null space of the WAM without of-

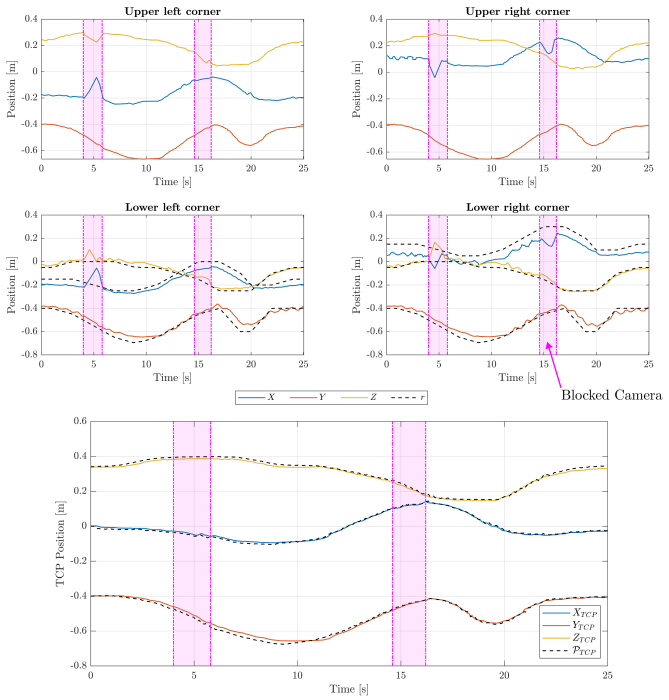

Fig. 15: Cloth corners and TCP evolved trajectories with camera blocking

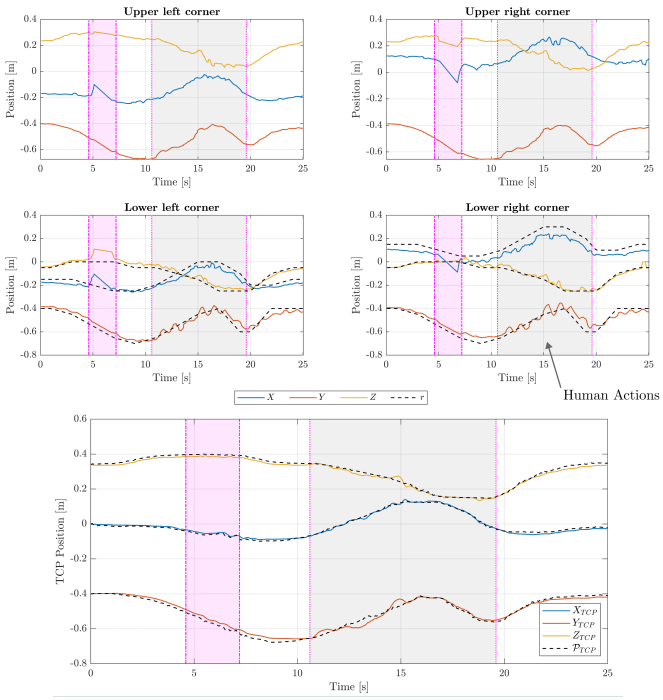

Fig. 17: Cloth corners and TCP evolved trajectories with human interaction

fering much resistance [25]. If a human moves the elbow joint, while theoretically it would not change the TCP pose, during a real experiment there are slight displacements and forces that are transmitted from the arm into the cloth, producing disturbances that are picked up by the camera.

In fact, the captured experiment started with a person walking in front of the camera as before, and after moving the elbow joint, the rigid piece connecting the upper corners of the cloth was pressed down on the right side, causing a slight rotation and a vibration when released. This situation can be seen in Figure 16.

The results of this experiment are shown in Figure 17. The shaded region in magenta corresponds to the interval when the camera was blocked, as in the previous experiment, while the region shaded in gray corresponds to the time when a human agent was interacting with the robot arm.

Blocking the camera has the same effects as before. Moving the elbow of the arm (from around 11 s to 15 s) produces slight movements in the TCP, which are also visible in the upper corners, and are propagated to have a greater effect on the lower corners due to the non-rigid nature of the cloth. When the rigid piece between the upper corners is pressed and released around the 15 s mark, we can see how it creates a sudden disturbance, very clear in the right upper corner, which adds to the oscillations already present in the lower corners. Even in these situations, we can see how the reference trajectory is tracked correctly. In real experimentation, we had to add a security layer which, in case the difference from the mesh estimated from vision and the belief in the linear model is too large, filters out the vision feedback. This was added to prevent aggressive control reactions to large errors or flickering in the mesh feedback, and limits the amount of interaction the robot allows during trajectory tracking. A test experiment was performed in which the authors held back a lower corner of the cloth while the robot was moving and, while the robot started to pivot its gripper to compensate for it, the safety layer was then triggered and the robot ignored the deviation from there on, working in open loop.

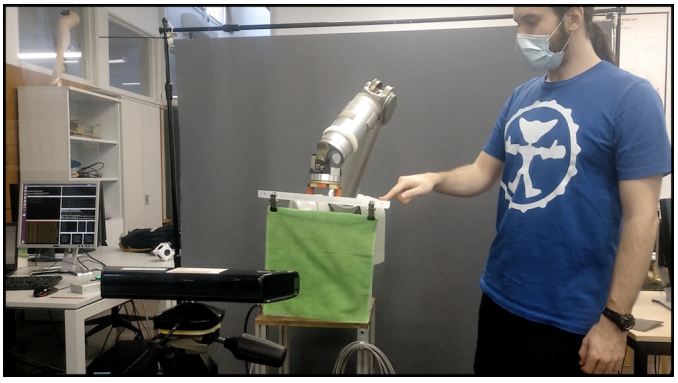

Fig. 16: Disturbance created by a person poking the cloth

With these results, we can say that the developed implementation has proven to work in demanding conditions, and even under the effects of an external agent creating disturbances while keeping a good trajectory tracking performance. These last two experiments yielded RMSEs of 5.5 and 5.6 cm, respectively, once the incorrect data obtained when the camera was partially obstructed (magenta regions) had been removed.

E. Error Analysis

A final set of experiments was conducted with a different trajectory, which was also printed in real scale to record videos showing the obtained tracking behavior. One of these

videos can be found in the supplementary material. These experiments were conducted to test the control scheme in situations outside the ones used for testing and obtaining the optimal parameters, and to do an analysis of all the possible error sources, quantifying each one. From the components of the used control scheme (Figure 7), the ones that can introduce significant errors are i) the MPC, ii) the Cartesian Controller, iii) the physical robot, and iv) the Vision feedback, both from the actual camera limitations and the processing algorithm.

To evaluate the errors introduced by each of these components, we can compare the following data: i) the input reference, which is a sequence of lower corner positions, ii) the MPC output, a TCP pose from which we can obtain the corresponding desired TCP positions, iii) the actual position of the TCP at all times obtained via IK, and finally, iv) real Vision data, processed to obtain positions of the entire mesh every instant, and the lower corners in particular. To compare TCP (ii & iii) and lower corner positions (i & iv), we can add a fixed offset of the initial distance between them. This is an approximation, as the cloth is flexible and their relative position can change, but these differences are reduced on slow trajectories, as the tested one.

The new trajectory consists in a sinusoidal movement in the X-axis while moving closer to the camera at a constant height, followed by a linear movement to return to the initial position. Figure 18 shows a comparison of the four aforementioned signals for the lower right corner in one execution. The MPC output (plus a fixed offset) follows the reference trajectory closely in situations of constant velocity, and tends to smooth out changes in direction. Part of this behavior can be explained due to its predictive nature, seeing future points after the direction change, and also given the flexible nature of the linear cloth model (COM), which will continue going in the previous direction due to inertia for a short time after the change. Even without considering these factors, the worst error is of 4.8 cm, and happens in one of the sharp changes in X. The RMSE between these two trajectories is only 1.7 cm.

Additionally, we can compare the errors due to orientation, given that the reference trajectory has the TCP pointing down. Of course, we cannot say a change in rotation produces an error of the angle times the entire length of the cloth, as it is a non-rigid object, but we have a rigid link between the TCP and the cloth of 9 cm that can produce some position error due to changes in orientation. In this case, the MPC output follows the reference almost perfectly, with maximum errors of 0.1 mm in position due to orientation errors.

Next, we can obtain the error between the desired TCP pose and the real one, also saved throughout the experiment. In Figure 18, we can see how these two trajectories (blue and magenta) are close, with some discrepancies in the Z-axis and especially on the change of direction in the Y-axis. The maximum position error between the two is of 2.6 cm, with an RMSE of 1.7 cm. In the plots, we can see how in the most critical points (turns), the two mentioned errors are additive, meaning the actual TCP position is even further to the input reference. With regards to changes in position due to orientation errors, here they are more notable, with a maximum of 1 cm and an RMSE of 5 mm.

Finally, the last signal corresponds to the processed data from the Vision algorithm. While we could expect this signal to deviate from the TCP evolved trajectories due to the flexible nature of the cloth piece, ideally being between the last signals and the input reference, here we are also adding the errors produced by the camera itself, with its maximum resolution and frame rate, and by the subsequent algorithm to detect and extract the positions of all the mesh nodes, and as we can see in Figure 18 (red line), as in all previous experiments, the obtained data is very noisy. Besides that, it is clear how it is also unreliable in the Z-axis, as the TCP does not change in this direction, the cloth is always extended vertically, but we get an evolution with constant changes. This difference being the most severe in this axis can be due to the table placed directly under the cloth with the reference trajectory, to serve as a visual indicator for the videos, which might affect the detection process. In the X and Y axes, we can see how the reference is tracked correctly, with the largest deviations being in the direction changes. While in X the trajectory seems to be between the TCP and reference, in Y we always have it further from the reference, adding onto the previous errors.

The final RMSE of the experiment (from reference to Vision data) is of 4.1 cm, but the error is not constant or distributed evenly, as it reaches maximums of 4.4, 5.7 and 5.8 cm in X, Y and Z, respectively. While these errors can seem large at first, they are the result of adding several sources together, and we do not have the exact position of the lower corners without camera noise and error, making difficult the isolation of the error due to the predictive controller. For example, for the worst case in the Y-axis, the difference between reference and MPC output is 1.4 cm, as shown in Figure 19 (in blue), while the remaining 4.3 cm come from other sources (Cartesian controller and physical actuators in red, and not predicted cloth behaviors and Vision error together in yellow). Additionally, this difference is a conservative measure, as we are adding a constant offset to the TCP position and not accounting for the inertia and non-rigid behavior of the cloth, which, in theory, would help make the real evolution be closer to the input reference. We can see how this is not the case here, but as another example, in the X-axis, the captured evolution is closer to the reference than the MPC output. This is what we would expect in good tracking, as the MPC must consider these behaviors and correct them, and the TCP evolution must not be a perfect copy of the reference trajectory with a constant offset, as it would be with a rigid piece. In this case then, we can only remove the error between the desired and actual TCP pose (around 1 cm) and say that the remaining 3.4 cm of error are shared between MPC and Vision processing.

All in all, without separating error sources, we obtain values of KPI_1 similar to the ones obtained before, and this new analysis proves that the errors come from different parts of the control scheme and are added together, and how the Predictive Controller itself is not the main source of error.

The trajectory must be slow enough to adapt to the framerate of the camera and Vision algorithm, which is not part of the contribution of this paper. A new set of experiments was carried out to compare results using no Vision feedback ($W_V = 0$), which means the controller relies completely on the

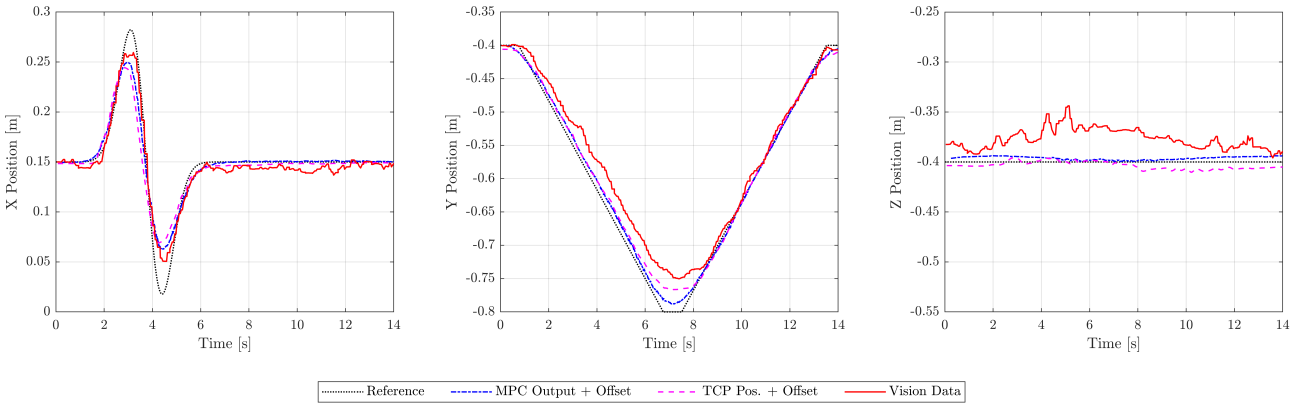

Fig. 18: Obtained trajectories of all relevant signals in the control scheme, showing the different error sources

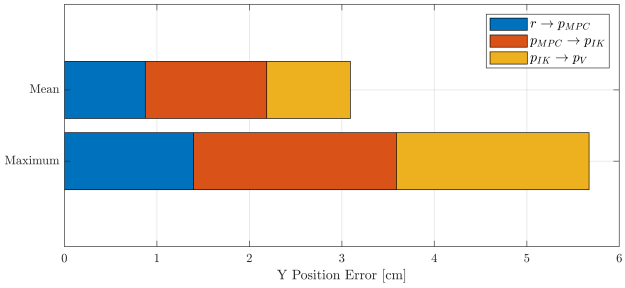

Fig. 19: Mean and maximum errors in the Y-axis of the right lower corner, and their distribution in different sources

simulation of its model to update the initial states. The MPC error was consistently larger in open loop, but the vision error is dominant and makes it hard to assess the types of motions for which the degradation is statistically significant; this would require an in-depth analysis using a less error-prone vision procedure, which is left to future work, as mentioned in the Conclusions section.

F. Comparison

Evaluating the performance of our proposed cloth manipulation method is currently hindered by the lack of literature on real-time closed-loop cloth manipulation. Using a classical controller like PID could be a straightforward solution, but a PID controller cannot integrate the dynamical model, including physical/operational constraints, and motion predictions. As a result, a direct comparison is not possible.

A potentially similar approach using MPC techniques applied to cloth manipulation can be found in [11], where the goal of the authors is to have position precision on the entire cloth. However, whereas they solve the optimization problem in several minutes, we solve it in milliseconds as we only focus on the lower corners' positions. Another existing technique for cloth manipulation is presented in [12], where a PR2 robot folding towels is presented. However, the cloth manipulation is static or quasi-static, as in the majority of the existing literature. By contrast, our approach is to dynamically manipulate cloth by incorporating a dynamic cloth model in the control system. A different dynamic manipulation approach is presented in [31], complementary to the approach presented here. It tries to learn cloth movement without a previous model using RL. However, the learning requires thousands of simulated samples, and the policies generated for specific tasks would require further real-robot samples to transfer from simulation to reality.

VI. CONCLUSIONS

In this paper, we propose an MPC framework to track reference trajectories of certain indirectly controlled parts of a deformable object, given an approximate model of its dynamics and a camera that can update the belief of the object's state in real time. Our proposed method is generalistic as regards the task to be performed, as it is given by a reference trajectory of the indirectly controlled points. Moreover, our controller is robust against camera occlusions, thanks to the dynamics model. We applied this framework to a robotic cloth manipulation task, in which the lower corners of a square cloth garment are controlled by the motion of its upper corners. To the best of the authors' knowledge, this is the first time that an MPC using a cloth model is used in real time for this kind of manipulation. Moreover, we have developed a linear cloth model that is shown to be an accurate approximation of real cloth and, therefore, is suitable to be used as a Control-Oriented Model in the MPC strategy.

Both the linear cloth model and the predictive controller have been improved via RL. The parameters of the model have been validated against the behaviour of a real cloth in several scenarios, and to enable its use inside the MPC, as well as learning the optimal structure and tuning of the controller.

Simulations showed accurate 3D trajectory tracking results, yielding mean errors in the order of millimeters for a cloth of 30×30 cm, and solving the optimization problem in near real time. After the simulations, a full closed-loop control scheme was implemented in a real setup, including MPC, a Cartesian controller, and Vision feedback and processing. Working with a real environment raised additional issues, like noise and timing, which had to be addressed. The final implementation runs at a constant rate fixed by the chosen T_s, it filters the output noise and has fail-saves against optimizations taking too long and vision data being too unreliable (with a B-SOM serving as a reference). Tracking errors are larger, to the order of centimeters, in the real robot, but they include new sources of error only present in the real setup, such

as sensor noise, actuator tolerances, and Cartesian controller inaccuracies. Even with disturbances, the considered errors are around the 5 cm, and an error analysis shows its principal source is not the Predictive Controller, even when considering a conservative approach.

The outcome of this paper can serve as a proof of concept of a new methodology for robotic cloth manipulation: using MPC with cloth dynamics prediction in real-time. A key aspect that allows for such real-time capability is the fact that, unlike other approaches [11], we do not need to predict an accurate evolution for the entire cloth mesh, but rather few relevant points (e.g., lower corners plus the grasped points). There are, however, aspects that can be further developed in the future:

- While the cloth models proposed can be changed according to the user/task demands and situations, *finding a general expression for the parameters* of the linear model could improve the method described in this paper. We have seen how these parameters depend on both the sampling time T_s and the mesh size n, and found the values for some combinations through learning, but an interesting step could be to try to find relations between these two conditions and the resulting parameters, and with enough time and data, even a model that can be used for any combination within a given range of values.
- Applying *Online Learning*. The tuning and controller structure found have proven to be the optimal ones for the tested cases, but for a more general application, a learning algorithm could use recent data during executions to adapt the parameters of the controller to their optimal values for the task at hand.
- Change the *Vision setup*. The current vision procedure clearly limits the aplicability of the method, thus we are currently devising a vision strategy to improve the quality of the feedback and, therefore, allow the robot to carry out cloth manipulation tasks at higher velocities.
- *Model external dynamics*. Other dynamic events such as air resistance were left out of this work, but could be included in the future by using disturbance estimators or by complementing the robustness statement approach already used for compensating/rejecting the effect of the wind on the cloth behavior.

REFERENCES

[1] A. Colomé and C. Torras, "Dimensionality reduction for dynamic movement primitives and application to bimanual manipulation of clothes," *IEEE Transactions on Robotics*, pp. 602–615, 2018.

[2] A. Colomé, A. Planells, and C. Torras, "A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5649–5654, 2015.

[3] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine, "Combining self-supervised learning and imitation for vision-based rope manipulation," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2146–2153, 05 2017.

[4] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, 2017.

[5] J. M. Maciejowski, *Predictive control: with constraints*. Pearson education, 2002.

[6] M. Bhardwaj, B. Sundaralingam, A. Mousavian, N. D. Ratliff, D. Fox, F. Ramos, and B. Boots, "Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation," in *5th Conference on Robot Learning (CoRL)*, vol. 164, pp. 750–759, 2022.

[7] X. Man and C. C. Swan, "A mathematical modeling framework for analysis of functional clothing," *Journal of Engineered Fibers and Fabrics*, vol. 2, no. 3, pp. 10–28, 2007.

[8] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson, "Physically based deformable models in computer graphics," in *Computer Graphics Forum*, vol. 25, pp. 809–836, Wiley Online Library, 2006.

[9] D. Baraff and A. Witkin, "Large steps in cloth simulation," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 43–54, 1998.

[10] X. Hu, Y. Bai, S. Cui, X. Du, and Z. Deng, "Review of cloth modeling," in *ISECS International Colloquium on Computing, Communication, Control, and Management*, vol. 4, pp. 338–341, 2009.

[11] Y. Bai, W. Yu, and C. K. Liu, "Dexterous manipulation of cloth," in *Computer Graphics Forum*, vol. 35, pp. 523–532, Wiley Online, 2016.

[12] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel, "Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding," in *2010 IEEE International Conference on Robotics and Automation*, pp. 2308–2315, IEEE, 2010.

[13] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto, "Learning predictive representations for deformable objects using contrastive estimation," in *Conference on Robot Learning CoRL*, vol. 155, pp. 564–574, 2021.

[14] Z. Erickson, H. M. Clever, G. Turk, C. K. Liu, and C. C. Kemp, "Deep haptic model predictive control for robot-assisted dressing," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4437–4444, IEEE, 2018.

[15] Y. Avigal, L. Berscheid, T. Asfour, T. Kröger, and K. Goldberg, "Speedfolding: Learning efficient bimanual folding of garments," in *2022 IEEE/RAS International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–8, 2022.

[16] J. Hietala, D. Blanco-Mulero, G. Alcan, and V. Kyrki, "Learning visual feedback control for dynamic cloth folding," in *2022 IEEE/RAS International Conference on Intelligent Robots and Systems (IROS)*, pp. 1455–1462, 2022.

[17] V. Petrík and V. Kyrki, "Feedback-based fabric strip folding," in *2019 IEEE/RAS International Conference on Intelligent Robots and Systems (IROS)*, pp. 773–778, 2019.

[18] X. Provot *et al.*, "Deformation constraints in a mass-spring model to describe rigid cloth behaviour," in *Graphics Interface*, pp. 147–154, Canadian Information Processing Society, 1995.

[19] F. Coltraro, J. Amorós, M. Alberich-Carramiñana, and C. Torras, "An inextensible model for the robotic manipulation of textiles," *Applied Mathematical Modelling*, vol. 101, pp. 832–858, 2022.

[20] C. Zhou, X. Jin, and C. C. Wang, "Efficient and stable simulation of cloth undergoing large rotations," *Computing in Science & Engineering*, vol. 10, no. 4, pp. 30–40, 2008.

[21] D. Limón, I. Alvarado, T. Alamo, and E. F. Camacho, "MPC for tracking piecewise constant references for constrained linear systems," *Automatica*, vol. 44, no. 9, pp. 2382–2387, 2008.

[22] D. Limón and T. Alamo, "Tracking model predictive control," in *Encyclopedia of Systems and Control*, 2019.

[23] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[24] Stanford Artificial Intelligence Laboratory, "Robot Operating System - Kinetic Kame." https://wiki.ros.org/kinetic.

[25] D. Parent, A. Colomé, and C. Torras, "Variable impedance control in cartesian latent space while avoiding obstacles in null space," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9888–9894, IEEE, 2020.

[26] M. Arduengo, C. X. Zheng, A. Colomé, and C. Torras, "Cloth Point Cloud Segmentation." https://github.com/MiguelARD/cloth_point_cloud_segmentation, 2021.

[27] S. W. Smith, "Chapter 15 - moving average filters," in *Digital Signal Processing* (S. W. Smith, ed.), pp. 277–284, Boston: Newnes, 2003.

[28] Y. Zhao, X. He, J. Zhang, H. Ji, D. Zhou, and M. G. Pecht, "Detection of intermittent faults based on an optimally weighted moving average t2 control chart with stationary observations," *Automatica*, vol. 123, 2021.

[29] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, 2018.

[30] J. Peters, K. Mulling, and Y. Altun, "Relative entropy policy search," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[31] R. Jangir, G. Alenyà, and C. Torras, "Dynamic cloth manipulation with deep reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4630–4636, IEEE, 2020.