Real-time simulation of hydraulic components for interactive control of soft robots

Alejandro Rodríguez, Eulalie Coevoet, Christian Duriez

Real-time simulation of hydraulic components for interactive control of soft robots

A. Rodríguez[1] and E. Coevoet[2] and C. Duriez[2]

***Abstract*— In this work we propose a new method for online motion planning in the task-space for hydraulic actuated soft robots. Our solution relies on the interactive resolution of an inverse kinematics problem, that takes into account the properties (mass, stiffness) of the deformable material used to build the robot. An accurate modeling of the mechanical behavior of hydraulic components is based on a novel GPU parallel method for the real-time computation of fluid weight distribution. The efficiency of the method is further increased by a novel GPU parallel leveraging mechanism. Our complete solution has been integrated within the open-source SOFA framework. In our results, we validate our simulation with a fabricated silicone cylinder and we demonstrate the usage of our approach for direct control of hydraulic soft robots.**

I. Introduction

The salient feature of soft robots is their compliant nature. As consequence of their compliance, soft robots exhibit a large number of passive degrees of freedom, leading to a higher operational flexibility even for underactuated configurations, and an intrinsic robustness to uncertainty. Due to these features, soft robots provide a real alternative to rigid robots for different environments and tasks, mainly involving safety conditions, or certain levels of uncertainty [1], [2].

In order to benefit from these features, advanced control systems must be developed. Unlike their rigid counterparts, the configurations and motions of soft robots are ruled by the deformable mechanics of their underlying soft structure and the properties of their base materials, thus a key component in the control pipeline is a precise yet interactive modeling of this compliant behavior. Additionally, a reliable simulation framework also provides a platform to design and validate prototypes prior to fabrication. Classic deformable solids simulation allows to predict the pose of a robot for a given state of its actuators. However, the inverse problem arises when aiming for a direct control of soft robots: find the necessary actuation to reach a desired robot pose. The fundamental interactive control capabilities of *point-to-point movement* or *path tracking* among others require a solution to this inverse problem in real time [3].

Duriez [4] introduced a novel method for simultaneous simulation and control of soft robots in an interactive manner based on an inverse kinematics problem using the Finite Element Method (FEM). This framework was further improved by Largilliere et al. [5], efficiently solving the inverse problem through a quadratic-programming (QP) algorithm on the constraint space. The complete framework is integrated within SOFA [6] and enables the design, simulation and interactive control of soft robots piloted by cable actuators controlled by servo-motors and/or pneumatic actuated internal cavities. As evidenced by recent works, hydraulic actuated soft robots exhibit a higher force output, and allow a higher frequency of actuation change [7], increasing the range of applicability of soft robots. Hydraulic actuation has also been proven effective for medical environments [8] among other fields [9].

In this work, we propose a real-time motion planning generation scheme in task-space for hydraulic actuated soft robots, allowing for both rapid prototyping and direct control or hydraulic soft robots. The correct and efficient modeling of hydraulic components arises as a necessary piece of this scheme, and requires a computationally demanding estimation of the fluid weight distribution on the model depending on the current configuration. We present a GPU parallel method for efficient fluid weight distribution inside dynamic cavities, along with a complete modeling of the dynamic behavior of hydraulic components. Moreover, we present a simple yet efficient leveraging mechanism for the computation of *irregular-parallel workloads*, which is applied to further increase the efficiency of the proposed method. We have integrated our entire solution within the Soft Robot framework of SOFA and we have validated our weight distribution algorithm with a real soft model.

II. FEM-QP soft robot control

We aim to develop a model for hydraulic components to be integrated within the FEM-based soft robot control framework proposed in [5]. In their work, cable and pneumatic actuators are handled interactively, but the computational complexity of estimating the fluid weight on the structure of the robots prevented their use within the framework.

Let us first recall the principles of the framework, which serves as base for our work. A soft robot is regarded as a FEM model accounting for its structure and material properties, a set of actuators and an arbitrary number of control DOFs in the form of end effectors placed on the structure. The configuration of the robot at a given time is obtained by solving the static equilibrium between the internal non-linear stress forces of the structure $f(x)$, the external and gravity loads f_{ext} and the contributions of each actuator $J_a^T \lambda_a$, yielding

$$f(x) = -f_{ext} - \sum_a (J_a^T \lambda_a), \tag{1}$$

with J_a^T the direction of the effort applied by the actuator on the FEM nodes and λ_a the contribution of the actuator.

[1]University of Granada, Spain, alejandrora@ugr.es
[2]INRIA, University of Lille 1, France

We compute a linearization of the internal forces at each time step i of the simulation

$$f(x_i) \approx f(x_{i-1}) + K(x_{i-1})dx, \tag{2}$$

where $K(x)$ is the tangent stiffness matrix that depends on the current position of the FEM nodes, and dx is the difference between consecutive positions in time $dx = x_i - x_{i-1}$.

To enable a direct control through motion planning, the value of λ_a is unknown, and depends on the input desired constraints δ_e for the end effectors. Thus, a three-step strategy is followed.

Step 1 A configuration x^{free} of the model without actuators influence is found by solving Eq. 1 with $\lambda_a = 0\ \forall a$. For the end effectors, also coupled to the model through J_e^T, a constraint violation δ_e^{free} is found.

Step 2 The actuators contribution λ_a that minimizes the violation efficiently found by projecting the mechanics into the smallest possible constraint space $\delta_e = \sum_a (J_e K^{-1} J_a^T \lambda_a) + \delta_e^{free}$ and solving the inverse QP problem

$$\begin{aligned} &\min_{\lambda_a} \ \left\| 1/2 \delta_e^T \delta_e \right\| \\ &\text{s.t. } A \geq b \end{aligned} \tag{3}$$

with A and b the constraint matrices on actuators, such as limits on volume growth.

Step 3 The model configuration is corrected

$$x = x^{free} + \sum_a (K^{-1} J_a^T \lambda_a). \tag{4}$$

Non-actuating constraints, such as cables with fixed length or internal air chambers can also be introduced in the system, with their corresponding λ_a set to fixed values.

Within this framework, hydraulic components add two constraints $J_p^T \lambda_p$ and $J_w^T \lambda_w$, corresponding to the pressure term and the fluid weight term respectively. The pressure term is equal to that of the pneumatic components, but the fluid weight term is complex to obtain because it requires an accurate computation of the fluid weight distribution on a given configuration. We address this problem on Section III.

Moreover, both the pressure and the fluid weight terms must be merged prior to performing the optimization routine. Otherwise, they would be treated as independent constraints and lead to an incorrect simulation. We address this issue in Section IV.

III. Fluid weight distribution

The first step for a correct modeling of hydraulic cavity components is an accurate computation of the added fluid weight distribution on the cavity surface. Although this is an easy task on analytical shapes, the typical piecewise FEM models use unstructured triangle or quad meshes for their geometrical description, and the exact computation of the weight distribution in those cases become a highly complex geometrical problem.

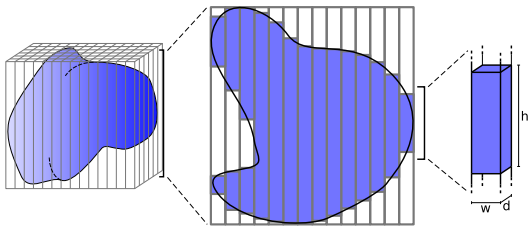

Fig. 1. Our method discretizes the fluid volume inside the cavity using a grid of regular columns covering the cavity geometry as shown on the left. A 2D lateral view of a slab of the computed grid is shown on the center. The parameters used to compute the volume of one column section are shown on the right. Note that in the presence of non-convex cavities, one column may generate more than one weight contribution.

Instead, we address this as a *mesh-on-grid* discretization problem. Two major advantages motivate this choice: we can easily control the trade-off between accuracy and computational cost independently of the particular model by setting the grid size, and more importantly, the problem becomes very amenable to parallel computation, allowing us to obtain very accurate approximations in real time, as we demonstrate later in this work.

A schematic depiction of our approach is shown in Fig. 1. Intuitively, we discretize the bounding box of the cavity into regular fluid columns with fixed cross section area $A = w \cdot d$. Then the problem is reduced to compute the height h of each *column section* bounded by the cavity and distributing its weight, easily computed per column as $W = w \cdot d \cdot h \cdot \rho \cdot g$, with ρ the fluid density and g the gravity force. Of course, for non-convex geometries a single column may produce several column sections contributing to different parts of the cavity (take for instance the 2D simplified case depicted in Fig. 1, center) and thus our algorithm also addresses these scenarios, as we explain below.

W.l.o.g., let us assume cavities defined by a closed triangular mesh. Our algorithm is composed by several consecutive steps:

Step 1 The mesh is transformed to fit a discrete 2D grid with arbitrary resolution. The grid is parallel to the XZ plane with with its normal parallel to the gravity force in the Y direction, and corresponds to the cross section of the grid of 3D columns.

Step 2 The mesh is partitioned in two groups: top triangles and bottom triangles, attending to whether their normals point upwards or downwards.

Step 3 The intersections between the top triangles and the columns are computed. For each candidate column-triangle pair, a barycentric test is performed to check whether the column center intersects the triangle. The barycentric test can be performed in 2D using the projected triangle on the grid plane. In case of intersection, the barycentric coordinates are used to compute the height where the intersection occurs. At the end of this stage, a list of intersecting heights per column is obtained.

Step 4 Similar to the previous step, the intersections between the bottom triangles and the columns are computed. The only difference is that for each intersection, not only the

height is stored, but also the triangle *id* and the barycentric coordinates of the intersection point.

Step 5 Finally, per each column, the top-intersections and the bottom-intersections are sorted and matched to compute the actual height of the different column sections. Then, the weight force of each column section is distributed to the nodes of its associated triangle, attending to the barycentric coordinates of the intersection to perform the distribution.

Our algorithm is very close in nature to the layered depth image (LDI) method [10], which is used for similar purposes in the computer graphics field and could likely be adapted to its use for our purpose. Our method however is specifically designed to our problem and thus avoids unnecessary rendering-related steps present in the LDI method, but more importantly, the LDI method relies on the fixed graphics pipeline while ours is standalone, becoming thus more easily adaptable to different platforms and languages.

A. Parallel leveraging

We have tested our algorithm running on CPU, but, as we shown in our experiments, only small grid sizes can be used under strong time restrictions, thus obtaining inaccurate estimations. However, we can achieve much more accurate results in the same time using a GPU computation. In fact, all the stages of our algorithm are highly parallelizable, thus we have implemented our algorithm using CUDA.

Steps 1, 2 and 5 are trivially computed in parallel by launching one thread per mesh vertex, triangle and column respectively.

Steps 3 and 4 on the other hand require a more complex parallel leveraging. By computing in parallel the axis aligned bounding box (AABB) of each triangle in the grid space we easily discard many columns per triangle, greatly reducing the candidate column-triangle pairs. The computation of each column-triangle pair candidate is independent of each other and theoretically they could run in parallel. However, the presence of triangles of different sizes leads to an uneven number of candidate columns per triangle, turning into an *irregular-parallel workload* problem [11]. This is actually an ubiquitous problem in a variety of disciplines, and different approaches have been proposed attending to the specificities of each context.

Our case has a special resemblance with a well known issue in the computer graphics field, since a very similar problem arises in the triangle rasterization stage, which is central to the rendering pipeline. This stage was historically performed by the fixed hardware implementation of the GPUs, however, some recent software approaches aiming to provide a higher flexibility have been presented. A very easy approach consists on launching one thread per triangle that computes all the candidates for that triangle [12], [13], unfortunately, a very poor performance is observed for highly uneven configurations. Other approaches provide complex scheduling pipelines to dispatch pair candidates in a parallel manner [14], [15]. Although these approaches provide a stable behavior, their complex nature adds a scheduling overhead that hurts performance, most evidently when handling very homogeneous cases, and also complicates their implementation, although this is regarded as a minor issue.

We propose a new approach which is as simple as the triangle-parallel approaches, but provides a fairly stable behavior in all cases without this resulting in a heavy computational overhead.

After computing the AABBs of the top or bottom triangles, we compute the average number c of columns per AABB. We then assign $k = \lfloor \alpha c \rfloor$ threads to each triangle (ensuring $k \geq 1$), and we perform a single GPU call of $k \cdot n$ parallel threads, with n the number of triangles. The α parameter allows to control the trade-off between parallel threads and memory read operations as will become apparent later. Through simple arithmetic using the threads *ids* and the already computed triangle-specific AABBs, we efficiently assign a subset of the candidates of each triangle to its assigned threads as we show in Algorithm 1, thus avoiding the construction of complex indirection structures required by other methods.

Algorithm 1 Leveraging kernel

$i_{tri} \leftarrow \lfloor ThreadID/k \rfloor$
triangle $\leftarrow$ *trianglesArray*[i_{tri}]
BBSize $\leftarrow$ *triangle.AABBMax*$-$*triangle.AABBMin*
offset $\leftarrow$ *triangle.AABBMin*
$i_{cur} \leftarrow ThreadID \mod k$
while $i_{cur} <$ *BBSize.x* $\cdot$ *BBSize.y* **do**
  i_{col}*.x* $\leftarrow$ *offset.x*$+i_{cur} \mod$ *BBSize.x*
  i_{col}*.y* $\leftarrow$ *offset.y*$+(i_{cur}/$*BBSize.x*$) \mod$ *BBSize.y*
  Compute column$-$*triangle pair candidate*
  $i_{cur} \leftarrow i_{cur}+k$
end while

This approach is very easy to implement with a negligible overload, yet it greatly homogenizes the workload among the threads. Additionally, this scheme also suits well the modern GPU programming paradigm since neighboring threads access the same triangle information thus memory read operations are efficiently performed, and each thread computes several column-triangle candidates for the same triangle thus this information only needs to be read once per thread. In our experiments we found the best outcome for $\alpha = 0.25$ given enough triangles, i.e., in the order of several thousands.

Fig. 2 shows the behavior of this approach in two very simple cases with three even and three uneven triangles respectively. In the first case the average number of candidates per triangle is 21, thus $k = 3$ and 9 threads are launched. In the second case, the average number of candidates per triangle is 20.3, thus $k = 5$ and 15 threads are launched. In both cases, all the threads are launched in a single GPU call, addressing all the column-triangle candidates without the need of additional indirection structures. In the first case, the workload on the threads is perfectly balanced. In the second case, the thread workload remains uneven, but the computational burden of each triangle is evenly split among

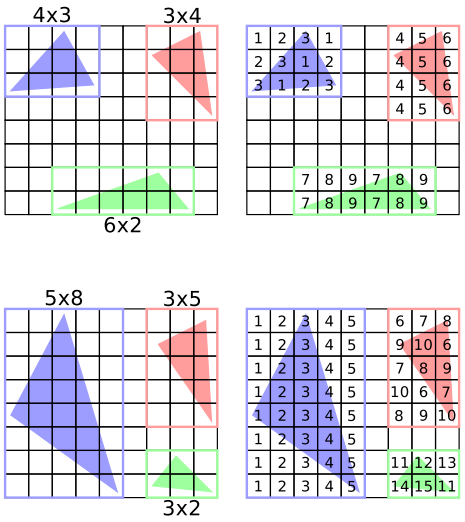

Fig. 2. Two simple cases with an even (top) and uneven (bottom) set of triangles. After computing the triangles AABBs (left), the average number of candidates per triangle is obtained and the number of threads to launch per triangle is computed. All the threads are launched in a single GPU call, which compute all the column-triangle candidates (right).

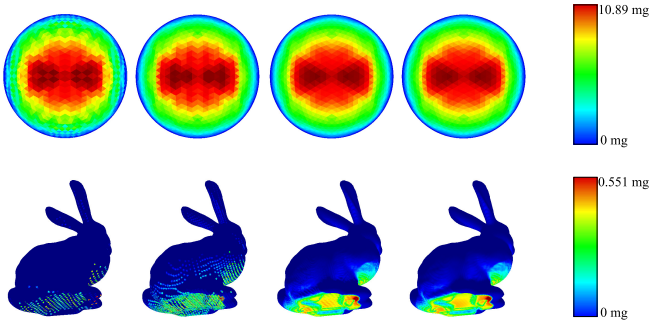

Fig. 3. Results of our weight distribution algorithm for different grid sizes (from left to right, 30^2, 50^2, 200^2 and 500^2) applied on a spherical cavity with 5120 triangles (top) and a bunny-shaped cavity with 90760 triangles (bottom). The nodal weight distribution is shown using a heatmap ranging from dark blue to dark red. Note that a distribution pattern appears due to the meshing of both the sphere and the bunny.

its assigned threads, homogenizing the average workload with a negligible overload.

B. Performance and accuracy evaluation

We have tested our fluid weight distribution in order to evaluate the performance of the algorithm and the accuracy achieved. We use two models with different topological complexity: a convex sphere model, created with $1cm$ radius and 5120 triangles, and a non-convex bunny model of $2cm$ height with 90760 triangles (Fig.3). We have applied our method with different grid sizes to validate the volume estimation and test its performance using both the CPU and the GPU implementations using an Intel Xeon W3550 system equipped with a Nvidia GeForce GTX 960. The results are summarized in Table I.

The theoretical volume of a sphere with $r = 1cm$ is $V = \frac{4}{3}\pi = 4.1887cm^3$, however our mesh is not analytic and its exact volume computed using the signed volume method [16]

TABLE I

VOLUME ESTIMATION OBTAINED AND TIME REQUIRED BY BOTH CPU AND GPU IMPLEMENTATIONS OF OUR METHOD FOR TWO CAVITY MODELS USING SEVERAL GRID SIZES.

	Sphere			Bunny		
Grid size	Volume (cm^3)	GPU (ms)	CPU (ms)	Volume (cm^3)	GPU (ms)	CPU (ms)
30^2	4.1835	0.29	0.63	1.1414	0.65	3.97
50^2	4.1828	0.31	0.93	1.6709	0.75	7.07
100^2	4.1804	0.37	2.24	1.7154	0.84	9.57
200^2	4.1799	0.57	7.63	1.7164	1.06	15.64
500^2	4.1798	0.86	41.77	1.7165	1.27	54.16
1000^2	4.1798	2.65	160.75	1.7165	2.79	186.66
1500^2	4.1798	5.29	358.81	1.7165	4.90	394.95

is $V = 4.1798cm^3$. Although the cavity volume estimation is very good for all the grid sizes, the distribution is not well approximated for low resolution grids, as seen in Fig. 3 (top), however, for the 200^2 grid both the volume estimation and the distribution estimation are very precise, and for larger grids the improvement is negligible. It is also worth noting that the meshing of the cavity plays an important role during the distribution process since larger triangles receive a higher amount of fluid weight.

For the bunny model we also compute the exact volume using the signed volume method, obtaining $V = 1.7165cm^3$. In this case, a very low grid resolution leads to a poor volume estimation, but for the 100^2 grid the cavity volume is accurately approximated. The distribution estimation requires higher grid resolutions as seen in Fig. 3 (bottom), mainly due to the increased number of triangles of the mesh, reaching a very precise estimation for the 500^2 grid. Again, meshing patterns appear during the distribution process.

Summing up, the method converges to the real solution by increasing the grid size. Naturally, the computation time also increases and the CPU version remains interactive only for low resolution grid sizes, however, the GPU version achieves very accurate estimations for both cases in less than 2 milliseconds, enabling its use for real-time simulation environments.

IV. UNIFIED HYDRAULIC CONSTRAINT

Hydraulic actuation applies two types of load on the deformable structure of the robot: pressure inside the cavity and additional weight distribution created by the fluid (computed thanks to the method presented above). Yet, in practice, these two terms are coupled when one activates the actuator by adding / removing the liquid. In the case of hydraulic components, this equates to differentiating the pressure and fluid weight terms w.r.t. the amount of contribution. Consequently, both terms must be coupled in a single constraint defining the behavior of the hydraulic component, and the function governing this coupling has a highly non-linear nature. Our goal is to keep the same formulation $J_a^T \lambda_a$ (see equation 1) for the total contribution of hydraulic actuators. In the following, we present how this term is obtained.

The derivative of pressure forces is easy to obtain since it only depends on the geometry of the cavity walls. $p =$

$\frac{F}{A}$, thus we define $J_p^T \lambda_p = pA$. Therefore, for a given configuration, we build J_p^T as $(J_p^T)_i = \sum_{t,i \in t} \frac{S_t n_t}{3}$, with n_t the normal and S_t the area of a triangle incident on the i^{th} node. With this definition, J_p^T is homogeneous to a force and λ_p represents the intensity of the pressure.

The derivative of the weight distribution with respect to the volume change, on the other hand, is complex because it depends on the cavity shape deformation. This deformation depends on the equilibrium between the pressure forces exerted from inside the cavity (which in turn depend on the geometry of the cavity walls as explained above), and the opposing stress forces exerted by the material surrounding the cavity (which in turn depend on the material properties and the current geometry configuration of the soft robot).

This is in fact a highly non-linear relation, but it can be linearized around the current configuration, assuming small weight variations. This assumption is valid since the soft-robot control pipeline is recomputed every few milliseconds, thus changes between iterations are indeed small. We also validate this hypothesis experimentally in this work. We define $J_w^T = \frac{\partial W}{\partial v}$ with W the fluid weight and v the cavity volume. For a given configuration, we compute the weight distribution per node w_i as explained in Sec. III and set the corresponding J_w entries with $\frac{w_i}{v_c}$, with v_c the current cavity volume. λ_w represents then the cavity volume change.

Lastly, we need to merge both pressure and weight terms because they are in fact coupled for hydraulic components and otherwise they would be treated as independent constraints during the optimization routine. As we explained earlier, $K^{-1} J_a^T \lambda_a = \Delta x$, thus $K^{-1} J_p^T \lambda_p$ gives the nodal displacements due to the pressure term. These displacements can be regarded as extrusion lengths for a given surface shape with area a, extruded along the surface normal direction, and would thus produce a volume change $a \cdot \Delta x$. Since this surface information is already stored in J_p, we linearize this relation as $\lambda_w = J_p K^{-1} J_p^T \lambda_p$ that couples both terms. Therefore, we merge both terms into a single hydraulic constraint λ_a with $J_a^T = J_p^T + J_w^T J_p K^{-1} J_p^T$. Again, this linearization remains accurate only for small changes from a given configuration and must be updated every iteration.

V. Results and Validation

We have integrated our solution within the SOFA soft robot framework [5]. To validate our model, we fabricated an empty silicone cylinder, we fill the cavity both with air and water and inflate it with an extra 15 ml of content in each case. We compare the obtained results with the simulated version of the same setup, using a FEM model with 375 nodes, setting the constitutive law of the material from mechanical experiments carried out in [4] and computing the fluid distribution inside the cavity with a grid of 200^2 cells. The final results are shown in Fig. 4, including the inclination angle of the hanging tip of the cylinders in the different configurations as a measure of deviation. Our model reproduces the behavior of its fabricated counterpart with an angle deviation under 2.5 degrees in every case, indicating that the forward simulation including our fluid weight distribution method achieves accurate and realistic estimations.

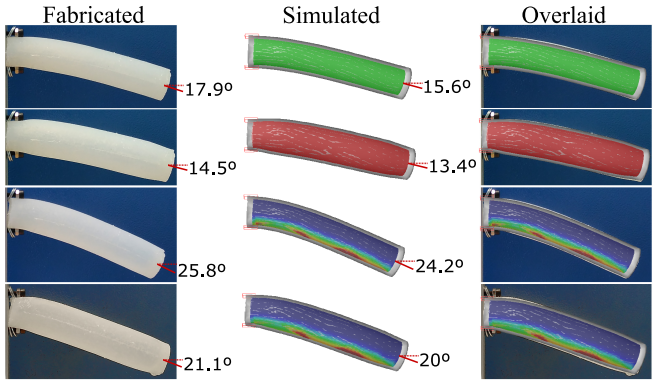

Fig. 4. Comparison of the fabricated silicone cylinder and the simulation of the same setup. From top to bottom, filled with air in rest state, inflated with 15 ml, filled with water in rest state and inflated with 15 ml. The inclination angle of the hanging tip is shown for each case.

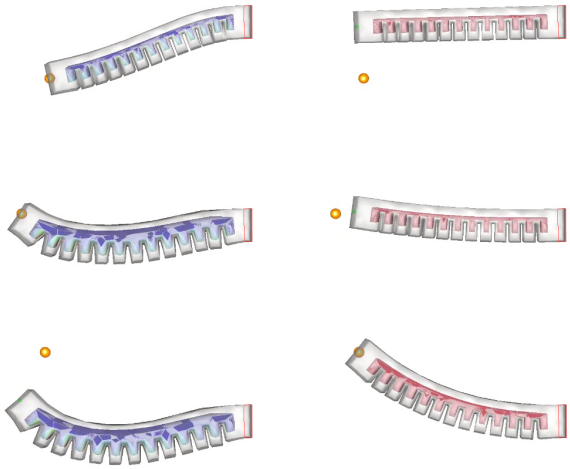

Fig. 5. Comparison of a soft finger operated by an hydraulic actuator (left) and a pneumatic actuator (right) for the same desired positions (yellow sphere). The models exhibit very different ranges of actuation, but our framework is able to reach the closest possible configuration in each case.

For the validation of our inverse problem modeling, we simulate the direct control of an actuated soft finger, similar to those used for the design of soft grippers [2]. To further emphasize the differences between pneumatic and hydraulic actuated soft robots, we compare both actuators on the soft finger as shown in Fig 5. The user interactively inputs the desired position of an end effector placed on the tip of the finger and the inverse problem is solved using the constraints built as explained in Section IV. As it is clearly depicted, the range of possible configurations is very different in both cases. The bending due to the weight of the fluid allows the tip to reach lower targets, and it counters the bending due to volume change, allowing to reach far horizontal targets, but on the other hand, it prevents the tip to reach higher targets. The models are accurately actuated to reach the desired configuration, or reach the closest possible configuration when the desired position is unreachable.

We also test a more complex simulation of a hanging soft

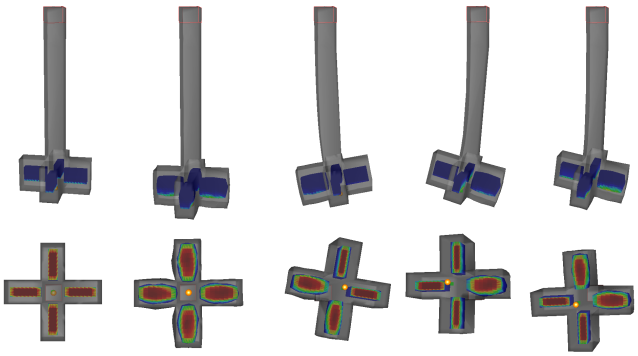

Fig. 6. Different configurations of a hydraulic actuated soft pendulum, shown from the front (top row) and from below (bottom row). The motion planning framework operates the actuators in real time to reach the desired position (yellow sphere) which is controlled by the user.

pendulum operated by four hydraulic actuators, shown in Fig 6, where the inflatable cavities can modify the weight of the bottom part and the center of gravity. Again, the user inputs the desired position of the end effector placed at the bottom of the pendulum. The fluid distribution inside each cavity is accurately computed with a grid of 200^2 cells, and the computation of the four cavities takes less than 2.1 ms.

The fluid weight is correctly considered during the motion planning process, allowing the pendulum to extend and contract along its vertical (Fig 6, first and second columns) and bend towards any direction in the horizontal plane (Fig 6, third to fifth columns) by inflating one or more of the cavities.

VI. Conclusions and Future Work

In this work we have presented an online simulation and motion planer for hydraulic actuated soft robots. The motion planer is based on an inverse solver that has been integrated within the SOFA framework. The main contributions are a novel method for accurate real-time computing of fluid weight distribution and a model for the dynamic behavior of hydraulic actuated cavities in soft robots. Moreover, we propose a novel GPU parallel leveraging algorithm for the efficient computation of irregular-parallel workload problems, increasing the efficiency of our fluid weight distribution approach. Our experiments and results show that the weight distribution algorithm provides very accurate estimations, and both the weight distribution and the dynamic behavior model of the hydraulic components are computed interactively, enabling the direct control of hydraulic soft robots.

Our simulation matches the real behavior of a simple fabricated specimen and, although we can successfully control a simulated hydraulic soft robot, we would like to apply our solution to control a fabricated soft robot for further validation of the simulation accuracy as future work. We also believe that a joint actuation of hydraulic and other components may increase the capabilities of soft robots and should be explored. Another interesting line for future work is the inclusion of contact with the environment in the motion planner, which could achieve an automatic avoidance of contact with obstacles or even consider them to modify the range of configurations of the robots. Moreover, we believe that our parallel leveraging algorithm may be of use in other contexts, such as software rasterization algorithms. We would like to perform a comparison with other current leveraging strategies and evaluate its performance and benefits.

Acknowledgements

This work is supported by the FPU 2012 program of the University of Granada and by the project TIN2014-60956-R of the Spanish Ministry of Economy and Competitiveness.

References

[1] A. D. Marchese, R. K. Katzschmann, and D. Rus, "Whole arm planning for a soft and highly compliant 2d robotic manipulator," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 554–560.

[2] B. S. Homberg, R. K. Katzschmann, M. R. Dogar, and D. Rus, "Haptic identification of objects using a modular soft robotic gripper," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 1698–1705.

[3] A. D. Marchese, K. Komorowski, C. D. Onal, and D. Rus, "Design and control of a soft and continuously deformable 2d robotic manipulation system," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2189–2196.

[4] C. Duriez, "Control of elastic soft robots based on real-time finite element method," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3982–3987.

[5] F. Largilliere, V. Verona, E. Coevoet, M. Sanz-Lopez, J. Dequidt, and C. Duriez, "Real-time control of soft-robots using asynchronous finite element modeling," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 2550–2555.

[6] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik, and S. Cotin, "Sofa: A multi-model framework for interactive physical simulation," in *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*. Springer, 2012, pp. 283–321.

[7] R. K. Katzschmann, A. D. Marchese, and D. Rus, "Hydraulic autonomous soft robotic fish for 3d swimming," in *Experimental Robotics*. Springer, 2016, pp. 405–420.

[8] K. Ikuta, H. Ichikawa, and K. Suzuki, "Safety-active catheter with multiple-segments driven by micro-hydraulic actuators," in *Medical Image Computing and Computer-Assisted InterventionMICCAI 2002*. Springer, 2002, pp. 182–191.

[9] M. De Volder and D. Reynaerts, "Pneumatic and hydraulic microactuators: a review," *Journal of Micromechanics and microengineering*, vol. 20, no. 4, p. 043001, 2010.

[10] B. Heidelberger, M. Teschner, and M. H. Gross, "Real-time volumetric intersections of deforming objects." in *Proceedings of Vision, Modeling, Visualization (VMV)*, vol. 3, 2003, pp. 461–468.

[11] S. Tzeng, A. Patney, and J. D. Owens, "Task management for irregular-parallel workloads on the gpu," in *Proceedings of the Conference on High Performance Graphics*. Eurographics Association, 2010, pp. 29–37.

[12] F. Liu, M.-C. Huang, X.-H. Liu, and E.-H. Wu, "Freepipe: a programmable parallel rendering architecture for efficient multi-fragment effects," in *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM, 2010, pp. 75–82.

[13] K. Fatahalian, E. Luong, S. Boulos, K. Akeley, W. R. Mark, and P. Hanrahan, "Data-parallel rasterization of micropolygons with defocus and motion blur," in *Proceedings of the Conference on High Performance Graphics 2009*. ACM, 2009, pp. 59–68.

[14] C. Eisenacher and C. Loop, "Data-parallel micropolygon rasterization," *Eurographics 2010 Short Papers*, pp. 53–56, 2010.

[15] S. Laine and T. Karras, "High-performance software rasterization on gpus," in *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. ACM, 2011, pp. 79–88.

[16] C. Zhang and T. Chen, "Efficient feature extraction for 2d/3d objects in mesh representation," in *Image Processing, 2001. Proceedings. 2001 International Conference on*, vol. 3. IEEE, 2001, pp. 935–938.