Software Architecture For Deformable Linear Object Manipulation: A Shape Manipulation Case Study

Manuel Zürn[1], Markus Wnuk[2], Armin Lechler, Alexander Verl
manuel.zuern@isw.uni-stuttgart.de
Institute for Control Engineering of Machine Tools and Manufacturing Units
Stuttgart, Baden-Württemberg, Germany

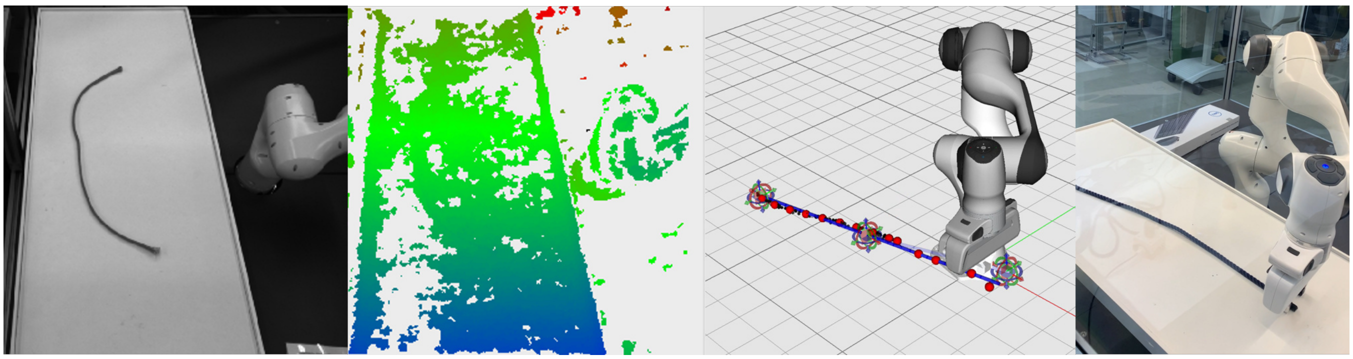


Figure 1: Key components for deformable linear object manipulation: left two images - perception of the stereo camera. An observer filters the point cloud and calculates target points shown in picture three as red spheres. After estimating a grasping point, the robot control plans a trajectory and executes it to shape the deformable linear object.

ABSTRACT

Deformable linear object manipulation is challenging due to their high dimensional configuration space and their underactuated nature when manipulated by a robotic gripper. Due to the complexity of the task, robotic manipulation relies on sensors and computationally demanding models, which end up in multiple different software components interacting with each other. Research in deformable object manipulation usually focuses on modeling, planning or control, without focusing on a software architecture. This paper presents a novel software architecture for deformable linear object manipulation. The software architecture includes components for deformable linear object manipulation, namely perception-, observation-, robot control-, planning-, communication- and decision component. On top of these components, a layered software architecture consisting of a decision layer, a skill layer and a functional layer is presented The proposed concept aims to be a blueprint for a unified software architecture satisfying the requirements of robotic systems to achieve deformable linear object manipulation. The validation of the software architecture is done in a case study of an autonomous shape manipulation task, where one robot and a stereo camera shape a deformable linear object to a predefined desired shape. This use case is inspired by an automated cable routing process, which today in the industry is still mainly handled manually and therefore offers a vast potential for automation.

[1] The research leading to this publication has received funding from the German Research Foundation (DFG) as part of the International Research Training Group "Soft Tissue Robotics" (GRK 2198/1).
[2] Funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany´s Excellence Strategy – EXC 2075 – 390740016.

RoSE '22, May 21–29, 2022, Pittsburgh, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

CCS CONCEPTS

• **Computer systems organization** → *Robotic autonomy*; • **Software and its engineering** → *Use cases.*

KEYWORDS

Software architecture, Deformable linear object, Robotic manipulation, Autonomous control

ACM Reference Format:
Manuel Zürn[1], Markus Wnuk[2], Armin Lechler, Alexander Verl. 2022. Software Architecture For Deformable Linear Object Manipulation: A Shape Manipulation Case Study. In *RoSE '22: 4th International Workshop on Robotics Software Engineering, May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

One major challenge in manipulating deformable objects lies in their high-dimensional configuration space. While the configuration of a rigid object can be described by knowing their translation and orientation, deformable objects also need a description of their configuration space. Therefore, deformable objects need new software solutions for perception, planning, and control to autonomously fulfill specific tasks for manipulating the deformable object.

Due to the additional complexity introduced by the object's deformation to the manipulation task, deformable object manipulation requires even more decision-making than rigid object handling. Therefore, further challenges emerge, which mainly manifests in new software requirements of the system. As many researchers create solutions for individual research areas within the interdisciplinary research field of deformable object manipulation, each researcher relies on their specific software implementation. This leads to many solutions without specific interfaces for general deformable linear object (DLO) manipulation, required for comparing the different developed algorithms. Strategic software design with common software engineering tools helps in reducing the complexity of deformable manipulation while being able to build modular and more reusable algorithms [3, 20].

The contribution of this paper is a software architecture for DLO manipulation. Research in DLO manipulation is often focused on individual components without considering an architecture combining the requirements from the different research fields of modeling, planning, control, and manipulation. For DLO manipulation, investigating software architectures lead to insights about the interfaces between the components, which helps in focusing on the individual component, enhances comparability, and leads to modular component design. Modular component design leads to more efficient programming, as complexity reduces using smaller programs, which also benefits the debugging process. The contributed architecture, therefore, presents the components and interfaces needed for a complex manipulation task of autonomously shaping a DLO.

This paper is structured as follows: Section 2 highlights relevant research for the components used for DLO manipulation and points out that the developed solutions are difficult to compare and reuse as they lack generic software interfaces. Section 3 elaborates the challenging problem for autonomous shape manipulation, which is then used to extract the requirements for the architecture design. The software concept in section 4 introduces the components and connections required for the DLO manipulation problem. After introducing a general software concept, section 4 also introduces a specific implementation of the shape manipulation case study, consisting of three software layers. The software layers range from high-level control in the decision layer, over a skill layer with individual executable skills, to a low-level control in the functional layer. Section 5 evaluates the presented software architecture in a case study for autonomous shape manipulation, first by validating the different skills of the skill layer and afterwards validating the decision layer in three real-world shaping experiments. Section 6 discusses the framework, concluding that software-driven development benefits reusability and generic implementations. Reducing component complexity through modularization and interface definition enhances programming efficiency and allows for comparing different algorithms.

2 STATE OF THE ART

In the vast research field of deformable object manipulation, this paper focuses on robots shaping deformable objects to a desired shape. To the best of our knowledge, current research for DLO manipulation does not consider specific software architectures. Therefore, the presented research about deformable object manipulation is mainly from a control engineering perspective. Recent research especially covers two different shape manipulation areas.

The first area investigates how two manipulators or one manipulator and one mount can continuously minimize the error between desired shape and the current shape. This is called shape control, where the object's current state gets returned by a 2D or 3D camera. Depending on the model, there exist different visual servoing approaches [2, 12, 14–16, 21, 23]. One approach for shape control focuses on model based controllers, e.g. [12, 14, 16, 23], while others use model free controllers [15]. The usual representations for visual servoing approaches are block diagrams, which show the relationship of the control loop needed to minimize the individual shape error function. Shape control research neglects grasping the objects or regrasping for more complex manipulation.

The second area is complex shape manipulation. Complex manipulation considers grasping and regrasping the deformable object. Regrasping can be useful for cloth folding [17], packing a DLO in a box [9], or trying to knot a DLO with two manipulators [19]. In general, regrasping the deformable object deals with the complex problem of finding suitable grasp positions. Lee at al. [8] used a learning approach to train the robot a pick-and-place action sequence for shaping a DLO. They used images with goal positions and needed 1,000 samples of real-world data in order to perform their shape manipulation task.

Current research in DLO manipulation is driven from a control perspective, which usually relies on block diagrams of the control loop or flow charts for shape manipulation. Systems used for DLO manipulation are custom-made and hand-coded by the individual researchers in the area, tailored for a particular manipulation task. The specific task considered is often a low-level task [7], which does not consider a higher abstract layer of the system. These systems are often roughly outlined in a schematic without addressing the software architecture used to implement specific connections between the software components. Specific details about the used components and interconnections from a software concept perspective are missing, although a software architecture would allow easier reuse of developed algorithms, benefiting interchangeablility and comparability. The use of basic software engineering tools would benefit the comparability of algorithms, e.g., a strategy design pattern [4] which can be used for generic interfaces.

Therefore, this paper presents a software architecture for DLO manipulation with its components and connections. The software architecture is then used in a three-layered architecture inspired by [1] to autonomously manipulate a DLO on a table with a robotic gripper, observed by a 3D stereo camera.

3 PROBLEM FORMULATION

Shape manipulation deals with the problem of finding a set of actions in order to transfer a DLO from an initial shape to a goal shape, shown in grey and green in Figure 2.

As DLOs have a high configuration space, it is usually not enough to perform just one action to reach the target shape. The action sequence of one action is defined as follows:

(1) Estimate a grasping position $\mathbf{T}_{W \to TP}$
(2) Move to grasping position
(3) Grasp

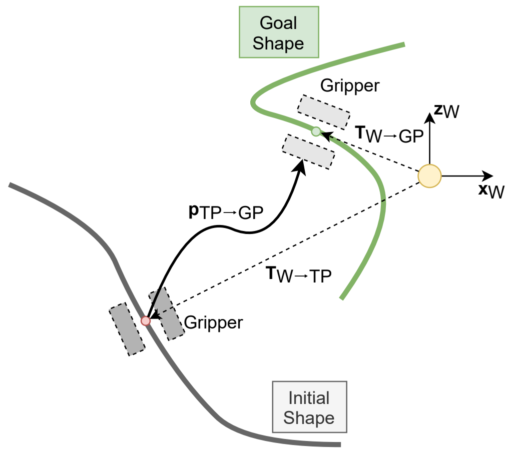


Figure 2: Problem formulation of the autonomous shape task. The sketch illustrates the autonomous shape manipulation task by showing the top view on the table of Figure 1.

(4) Plan a path $\mathbf{p}_{TP\rightarrow GP}$ to a goal position
(5) Move along the path
(6) Release grasp

Note that a robotic gripper can also push or grasp with different forces. To keep it simple, pushing or different grasping forces are not considered in the case study.

Each item in the enumeration can be seen as a skill. The skill to estimate grasping points on the DLO requires the evaluation of sensor data. The sensor signals offer information about the shape of the DLO in the sensor coordinate system, which then has to be transformed to a unified world coordinate system $\mathbf{x}_{\mathrm{W}}, \mathbf{z}_{\mathrm{W}}$.

Vision-based sensors are used for locating a DLO. For 3D information, the shape has to be estimated out of point cloud data from a 3D camera. After estimating the current shape, a grasp point has to be chosen, such that the motion afterwards reduces the overall error from the current shape to the goal shape. As there are an infinite amount of possible grasping points on the DLO, this is a rather challenging problem.

The following components emerge through the problem formulation to achieve the action sequence.

- A robot control component to execute trajectories and grasp objects.
- A model representation of the DLO to define a goal shape and to estimate grasping points of the current shape.
- A camera in order to perceive the current shape.
- An observer which calculates the current model state out of the sensor signals of the camera.

4 SOFTWARE ARCHITECTURE

A software concept for DLO manipulation has to include multiple components in order to change and adapt it quickly to new research contributions. To design a software architecture, all components and connections of the system have to be defined [22]. Therefore, at first, the key components for DLO manipulation are extracted from the requirements and listed before the connections between them are shown. The specific implementation of the code is done using the software architecture, which links requirements and code [5].

One component of the software architecture is a robot control component. The robot or the robots are needed in order to manipulate the DLO. As the motion of the robots is subject to errors, it is necessary to access the robot state for an optional control loop. Another component of the software architecture is the perception component. It is needed as soon as the state of the DLO is not precisely known. To interpret the signals of the perception component and translate it into the state of the model, a model component and an observer component are needed. The observer component calculates the state out of the signal of the perception component combined with the old or initial state of the model component. A model component can be used for planning specified DLO motions, as well as grasp point calculation, trajectory calculation, or goal shape definition. Furthermore, a communication component is needed, and optionally a viewer component. The viewer is connected to all data provided by the components, which makes it suitable for logging and recording data. A particular requirement about DLO manipulation is that, in contrast to rigid object manipulation, planning and control are difficult to separate. On the one hand, local minima in the configuration space result in conventional optimization-based control methods getting stuck. On the other hand, global planning is computational expensive due to the high dimensional shape approximations of the DLO, hence it is not feasible in real-time control cycles. Therefore, interleaving planning and control is necessary [10]. To meet this requirement, a decision component is introduced. The decision component can decide whether to access planning or control mode.

These components can be developed independently as soon as the interfaces are defined. The component diagram of the presented software architecture is shown in Figure 3.

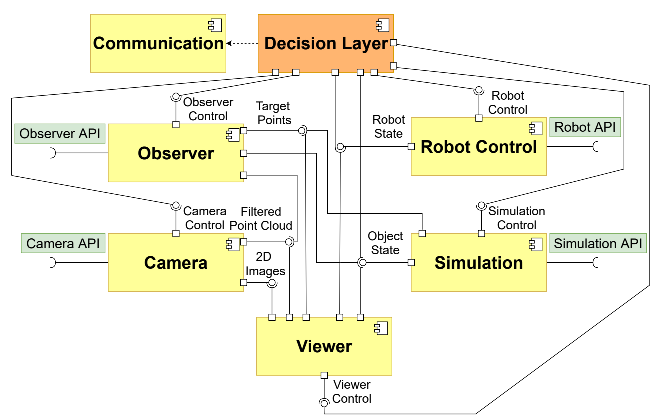


Figure 3: UML component black box diagram of the proposed software architecture.

4.1 Specific component design for the shape manipulation

The perception unit of the component diagram is chosen as a camera, while the model component of the architecture is a simulation.

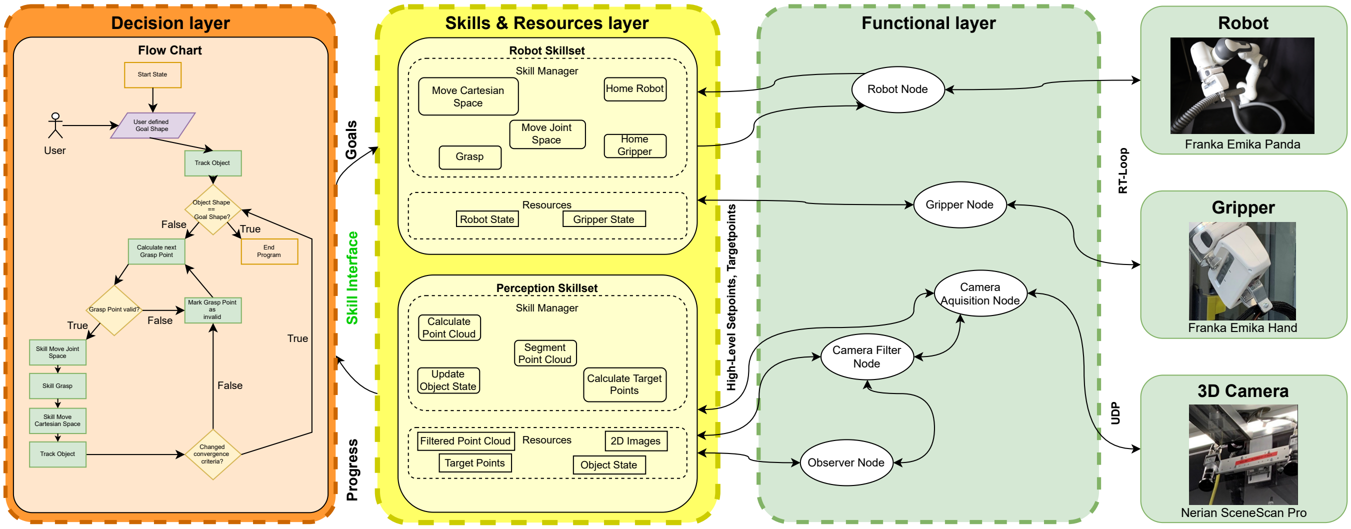


Figure 4: Specific implementation of the used three-layer software architecture for autonomous shape manipulation. It illustrates the hardware of the case study connected with the functional layer, the flow chart of the decision layer for high-level abstract task formulation, and the individual skills of the skill layer. Further details of the flow chart are shown in Figure 6.

Cameras are cheap and the most popular choice in recent literature for state-feedback of DLOs. With the simulation, it is possible to predict future states of the object and satisfy constraints of the object to be more robust to outliers of the camera component. The camera component accesses the camera application programming interface (API) of a stereo camera. For interchangeable registration algorithms, the observer uses an observer API which is implemented using a strategy design pattern [4]. Three different observers are implemented in order to be able to choose different strategies for different scenarios. Structure preserve registration [18], a self-organizing map [24] as well as the coherent point drift method [11].

The overall software concept includes five processes. The main process is used as a decision layer to start all sub-processes and to execute skills on the robot, which is connected to the robot API. Furthermore, the decision layer controls each component and executes skills. The camera process receives images from the camera API, filters the images and creates a filtered point cloud, and publishes both. The observer process subscribes to point clouds of the camera and the old or initial state from the simulated model and afterwards calculates target points. The simulation is used to drag the simulated model to the calculated target points of the observer and publishes then the updated object state. The viewer visualizes the point cloud, the robot's current configuration, and the target points to check if the algorithms work as intended.

4.2 Software Layers used for Autonomous Shape Manipulation Architecture

To reduce complexity and build up upon a common practice in software engineering, we divide the task into a decision layer, a skill layer, and a functional layer [1, 6, 13], shown in Figure 4.

The decision layer is used to formulate an abstract task that must be fulfilled to achieve the shape manipulation task. Our flow chart is structured as follows: A user can interactively choose a goal shape by drag and drop using the interactive frames. The interactive frames are then used in simulation to apply a force on the dragged bodies. Note that there are no restrictions in choosing a goal position. However, as the simulation applies forces onto the simulated object, the simulation constrains the movement to the predefined DLO model.

One heuristic that can be used to decide the grasping point is discretizing the object and afterwards calculating the largest distance between the discretized initial shape and the discretized goal shape, which returns one specific grasping point.

After grasping, a path is calculated to move the grasped point to the goal point. The path can then be calculated by the skill layer and commanded to the robot control node. As different trajectories with different velocity profiles may result in different goal shapes of the DLO, the goal shape has to be observed again in order to validate the correct shape of the object.

One way to reach a goal configuration is to execute the above action sequence iteratively. This is implemented in the decision layer. After setting the goal positions, motions are performed until a tracking threshold is reached. This is shown in the flow chart in Figure 4.

The skill and resources layer includes the high-level interface for operating the perception and robot skillset. Simulation with the model of the DLO, as well as camera and observer component of Figure 3 are included in the perception skillset, whereas the communication component handles the interface between decision layer and skill layer. The skill layer includes functionality to move the robot, to move the gripper, to calculate and segment point clouds, and to calculate target points, which are shown as red spheres in Figure 5.

The functional layer is the low-level interface to the connected hardware. Connected is a Franka Emika Panda robot with seven degrees of freedom (DOF) over the Franka control interface [1] with self-written motion planning and control, as well as a Nerian SceneScan Pro stereo camera connected over the camera API [2].

4.2.1 Interface between software layers. The interfaces between the software layers enable their communication. As all individual components shown in Figure 3 are connected with the decision layer, the decision layer acts as a master. For executing functions on the separate processes, the decision layer publishes the component name, the skill name, and optional arguments. See Algorithm 1.

Algorithm 1: Decision layer communication to skill Layer.

```
Input  : String component c, String skill s and Arguments args
Output : Boolean success
try:
    sArgs ←serialize(args);
    msg ← c, s, sArgs;
    sendMessage(msg);
    success ← True;
catch ConnectionError:
    success ← False;
end
return success
```

This message is then used to control and execute the different skills of the skill layer. Furthermore, it is used to configure the filters of the camera to extract the model of the environment. Each component is then reading the message and executing the skills, see Algorithm 2. This allows for distributed components, so that each component can run in its own process.

Progress is detected of the decision layer by subscribing on the object state, see also Figure 3.

The skill layer is separated from the functional layer to have independent hardware choices. A high-level interface in the form of abstract functions can be called directly from each component at the functional layer. The functional layer includes the specific APIs used to communicate with the hardware.

5 CASE STUDY OF AUTONOMOUS SHAPE MANIPULATION

This section consists of first presenting the different skills and afterwards evaluating the architecture by solving a shape manipulation problem formulated in Figure 2.

5.1 Evaluation of the Skill layer

Each skill is first evaluated separately in the evaluation section before running the decision layer. The different skills are shown in Figure 5. First, the point cloud is calculated from the perception unit and afterwards loaded into the simulation environment. To segment the environment from the DLO, a brightness filter and a box filter are used. Afterwards, the point cloud should only contain outliers and points of the DLO. To calculate the target points, the observer interface is used, which calculates target points from previously known positions and the captured point cloud, shown in Figure 3. A PD controller is used in the simulation to track the object to the calculated target points. Afterwards, the viewer generates three interactive frames for the operator to shape a custom goal shape. The grasp point will be calculated, and the robot will be commanded to move to the grasping point. After grasping, the robot has to move to the goal point to establish the user-defined shape.

[1]https://github.com/frankaemika/libfranka

[2]https://nerian.com/support/documentation/api-doc

Algorithm 2: Skill execution interface of the components.

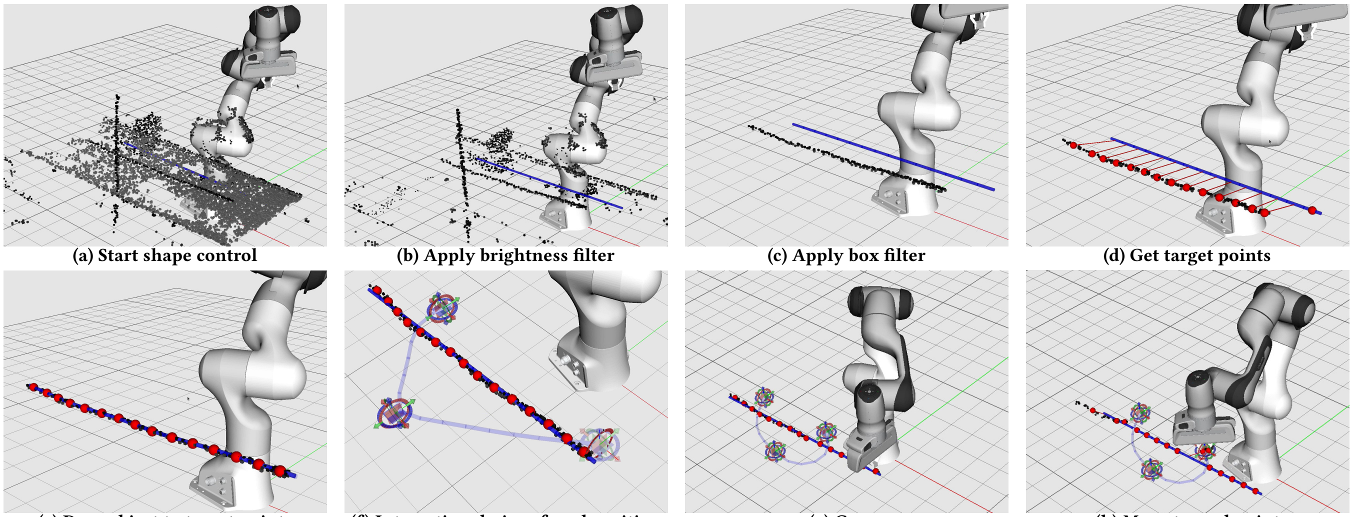
```
Input  :
Output : Boolean success
try:
    msg ← recvMessage();
    c, s, sArgs ← msg args ← unserialize(sArgs);
    try:
        if c = thisComponent then
            if s = executableSkill then
                skill ← getSkill(s);
                skill(args);
                success ← True;
            else
                success ← False;
            end
        else
            success ← False;
        end
    catch SkillError:
        success ← False;
    end
catch ConnectionError:
    success ← False;
end
return success
```

5.2 Evaluation of the decision layer

The flow chart for the shape manipulation task is shown in Figure 6 and in the decision layer in Figure 4. After starting the program, the user inputs a desired goal shape. Three movable coordinate systems define the goal shape, see Figure 5 (f). After setting the goal shape, the simulated DLO object gets tracked to the observed target points. If the object shape equals the desired goal shape, the program ends. The skill compares the desired shape with the current shape. The resulting error can then be compared to an empirically chosen convergence threshold, where values below the threshold indicate convergence.

After calculating the grasp point, which is farthest away from its goal point, the grasp point is checked. If the grasp point is valid, the robot grasp- and moving sequences start, otherwise the grasp point is marked invalid, and the next best grasp point is selected.

The robot skills consist of moving in joint space to avoid robotic singularities, grasping, and moving in cartesian space for the pick- and place operation. Checking the convergence criteria shows if the robot has manipulated the DLO. If the convergence criteria do not



(a) Start shape control (b) Apply brightness filter (c) Apply box filter (d) Get target points

(e) Drag object to target points (f) Interactive choice of goal position (g) Grasp (h) Move to goal point

Figure 5: Sequence of skill executions, repeat (d),(e),(g),(h) until converged.

change, it marks the grasp point as invalid. Otherwise, all invalid marked grasping points reset.

The sequence of comparing the object shape to the goal shape starts again, which reduces the shape error iteratively.

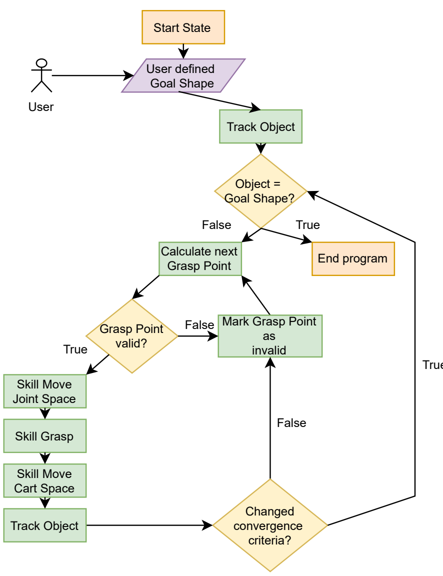
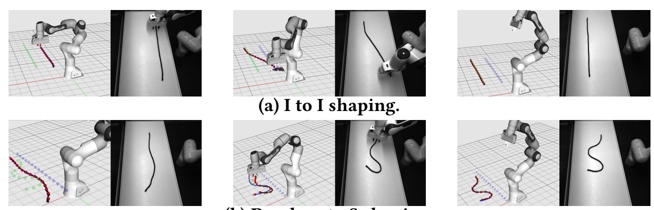


Figure 6: Flow chart of the autonomous shape manipulation task.

5.3 Shape manipulation experiments

Five experiments validate the software architecture. Three shown in Figure 8 by choosing a user-defined shape, as well as two experiments with a predefined shape shown in Figure 7. The number of individual actions for Figure 7, as well as the convergence criteria and convergence value are shown in Table 1.



(a) I to I shaping.

(b) Random to S shaping.

Figure 7: (a) I to I shaping, starting at the blue dots and ending at the green dots. (b) Random start to S shaping.

The corresponding live images of the stereo camera are shown on the right of each simulation image. Red spheres indicate target positions calculated by the observer, while the simulated DLO is indicated in blue. Black cubes represent the segmented point cloud calculated from the stereo camera images.

Table 1: Number of actions until the DLO reaches its final shape determined by a root mean square error < 1.2cm, see Figure 7.

	I to I shaping	Random to S shaping
Actions	5	30
RMSE	0.85 cm	0.95 cm
Convergence Criteria	1.2 cm	1.2 cm

Before doing the experiments, the convergence threshold has to be set. Choosing a high convergence threshold results in high shape errors, whereas a low convergence threshold can result in an endless refinement loop, as no break conditions for the robot are implemented. This results out of the stiffness of the DLO. Manipulation on one end can increase the shape error of the other end through the applied manipulation forces, as it always affects the whole shape of the DLO. To avoid this, the convergence threshold is determined empirically in the experiments and is set to a reasonable low shape error, see Figure 8.

In the first experiment, the DLO lies down in a random but not overlapped position. After tracking, the user input was to shape approximately an "I" shape. The final result shows some slight deviation on the bottom, which results in an incomplete point cloud from the box filter. The second experiment was to shape an "S" shape out of an "I" starting configuration. The third experiment was to shape an "I" shape out of a "S" starting configuration. Despite each manipulation task requiring the robot to recognize different object shapes, determine varying grasping positions for each shape, and follow different trajectories, these tasks could all be solved by the same flow chart in the decision layer. This shows how the abstraction of the presented software architecture from the underlying functional and skill layer to the high-level decision layer facilitates DLO manipulation and extends applicability for different manipulation scenarios.

6 CONCLUSION

This paper presents a software architecture for DLO manipulation. It is used in three autonomous shape manipulation experiments with a skill-based 3-layer software architecture. The presented three-layer software architecture helps in reducing the complexity by modularizing different skills. Table 1 proves that the programming complexity stays the same even if the number of actions to achieve the desired shape increase. These skills are implemented separately, making it possible to verify the correct skills individually without testing the whole action sequence. As the decision layer makes it possible to shape a DLO, it is further possible to move the decision layer to a skill. If further development requires a skill to shape a DLO, the future decision layer would be able to build up upon the current development, which makes it reusable by design.

For validation of the software architecture, an autonomous shape manipulation demonstration was implemented. The demonstration allows the creation of high-level flow charts for an abstract formulation of goals. At the same time, skills can be implemented and verified separately with a connected functional layer as a low-level hardware interface. The presented demonstration validates the architecture design with various shape manipulation tasks.

Other skills like pushing the DLO would improve the shape demonstration, as pushing does not have the same impact on the shape as grasping. In the future, it will be investigated how the presented architecture can be used for more complex manipulation tasks. Considering the high potential in wire harness assembly or routing scenarios, more work will bridge the gap from academic to industrial examples.

ACKNOWLEDGMENTS

The research leading to this publication has received funding from the German Research Foundation (DFG) as part of the International Research Training Group “Soft Tissue Robotics” (GRK 2198/1). Funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany´s Excellence Strategy – EXC 2075 – 390740016.

REFERENCES

[1] Alexandre Alborne, David Doose, Christophe Grand, Charles Lesire, and Augustin Manexy. 2021. Skill-Based Architecture Development for Online Mission Reconfiguration and Failure Management. In *3th International Workshop on Robotics Software Engineering (RoSE'21)*. Vol. 3.

[2] Miguel Aranda, Juan Antonio Corrales Ramon, Youcef Mezouar, Adrien Bartoli, and Erol Özgür. 2020. Monocular Visual Shape Tracking and Servoing for Isometrically Deforming Objects. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 7542–7549. https://doi.org/10.1109/IROS45743.2020.9341646

[3] Donald Bell. 2004. UML basics: The component diagram. *IBM Global Services* (2004).

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1997. *Design Patterns Elements of Reusable Object-Oriented Software.*

[5] David Garlan. 2000. Software architecture: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*. 91–101.

[6] José Carlos González, José Carlos Pulido, and Fernando Fernández. 2017. A three-layer planning architecture for the autonomous control of rehabilitation therapies based on social robots. *Cognitive Systems Research* 43 (2017), 232–249. https://doi.org/10.1016/j.cogsys.2016.09.003

[7] Rafael Herguedas, Gonzalo López-Nicolás, Rosario Aragüés, and Carlos Sagüés. 2019. Survey on multi-robot manipulation of deformable objects. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 977–984. https://doi.org/10.1109/ETFA.2019.8868987

[8] Robert Lee, Masashi Hamaya, Takayuki Murooka, Yoshihisa Ijiri, and Peter Corke. 2022. Sample-Efficient Learning of Deformable Linear Object Manipulation in the Real World Through Self-Supervision. *IEEE Robotics and Automation Letters* 7, 1 (2022), 573–580. https://doi.org/10.1109/LRA.2021.3130377

[9] Wanyu Ma, Bin Zhang, Lijun Han, Shengzeng Huo, Hesheng Wang, and David Navarro-Alarcon. 2021. Action Planning for Packing Long Linear Elastic Objects into Compact Boxes with Bimanual Robotic Manipulation. arXiv:2110.11652 [cs.RO]

[10] Dale McConachie, Mengyao Ruan, and Dmitry Berenson. 2020. Interleaving Planning and Control for Deformable Object Manipulation. In *Robotics Research*, Nancy M. Amato, Greg Hager, Shawna Thomas, and Miguel Torres-Torriti (Eds.). Springer International Publishing, Cham, 1019–1036.

[11] Andriy Myronenko and Xubo Song. 2010. Point set registration: coherent point drift. *IEEE transactions on pattern analysis and machine intelligence* 32, 12 (2010), 2262–2275. https://doi.org/10.1109/TPAMI.2010.46

[12] David Navarro-Alarcon and Yun-Hui Liu. 2018. Fourier-Based Shape Servoing: A New Feedback Method to Actively Deform Soft Objects into Desired 2-D Image Contours. *IEEE Transactions on Robotics* 34, 1 (2018), 272–279. https://doi.org/10.1109/TRO.2017.2765333

[13] Issa A.D. Nesnas, Anne Wright, Max Bajracharya, Reid Simmons, and Tara Estlin. 2003. CLARAty and challenges of developing interoperable robotic software. In *Proceedings / 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*. IEEE Operations Center, Piscataway, NJ, 2428–2435. https://doi.org/10.1109/IROS.2003.1249234

[14] Jiaming Qi, Guangfu Ma, Jihong Zhu, Peng Zhou, Yueyong Lyu, Haibo Zhang, and David Navarro-Alarcon. 2021. Contour Moments Based Manipulation of Composite Rigid-Deformable Objects With Finite Time Model Estimation and Shape/Position Control. *IEEE/ASME Transactions on Mechatronics* (2021), 1–12. https://doi.org/10.1109/TMECH.2021.3126383

[15] Romain Lagneau, Alexandre Krupa, and Maud Marchal. 2020. Automatic Shape Control of Deformable Wires based on Model-Free Visual Servoing. (2020).

[16] Kenta Tabata, Hiroaki Seki, Tokuo Tsuji, Tatsuhiro Hiramitsu, and Masatoshi Hikizu. 2020. Dynamic manipulation of unknown string by robot arm: realizing momentary string shapes. *ROBOMECH Journal* 7, 1 (Dec. 2020). https://doi.org/10.1186/s40648-020-00187-w

[17] Daisuke Tanaka, Solvi Arnold, and Kimitoshi Yamazaki. 2021. Disruption-Resistant Deformable Object Manipulation on Basis of Online Shape Estimation and Prediction-Driven Trajectory Correction. *IEEE Robotics and Automation Letters* 6, 2 (2021), 3809–3816. https://doi.org/10.1109/LRA.2021.3060679

[18] Te Tang and Masayoshi Tomizuka. 2019. Track deformable objects from point clouds with structure preserved registration. *The International Journal of Robotics Research* (2019), 027836491984143. https://doi.org/10.1177/0278364919841431

[19] Te Tang, Changhao Wang, and Masayoshi Tomizuka. 2018. A Framework for Manipulating Deformable Linear Objects by Coherent Point Drift. *IEEE Robotics and Automation Letters* 3, 4 (2018), 3426–3433. https://doi.org/10.1109/LRA.2018.2852770

[20] Richard N. Taylor and Andre van der Hoek. 2007. Software Design and Architecture The once and future focus of software engineering. In *Future of Software Engineering (FOSE '07)*. 226–243. https://doi.org/10.1109/FOSE.2007.21

[21] Bao Thach, Brian Y. Cho, Alan Kuntz, and Tucker Hermans. 2021. Learning Visual Shape Control of Novel 3D Deformable Objects from Partial-View Point Clouds. *CoRR* abs/2110.04685 (2021). arXiv:2110.04685 https://arxiv.org/abs/2110.04685

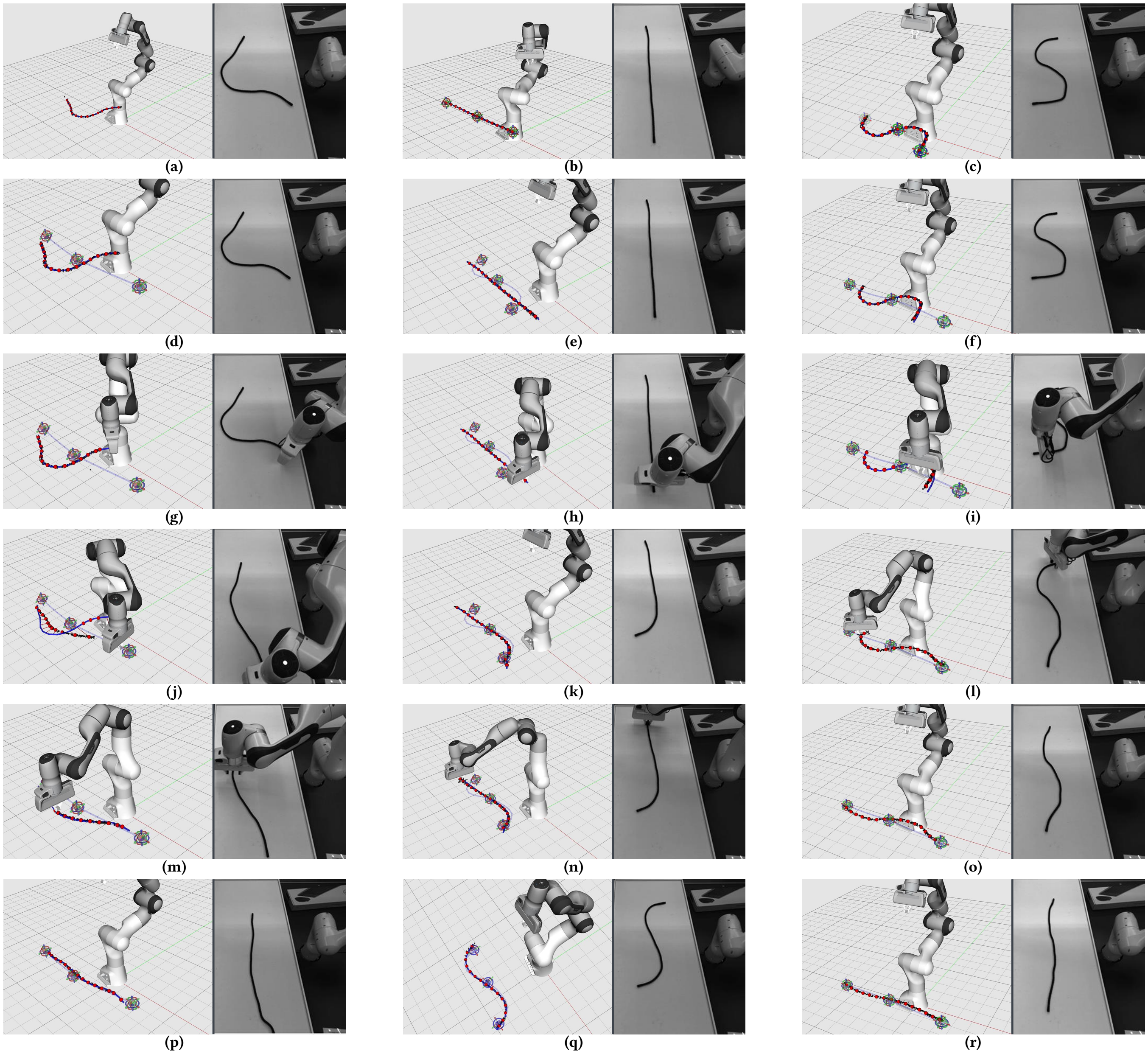


Figure 8: Three experiments shaping a DLO through multiple grasps. A single image contains the simulation including the loaded point cloud, target points and model of the DLO, as well as the synchronized robot and the goal shape. The right image is the associated 2D image of the left camera. (a), (b), (c): Start tracking. (d), (e), (f): Define the target shape. (p), (q), (r): final state. All other images are states between the three experiments.

[22] Mohammad Hadi Valipour, Bavar Amirzafari, Khashayar Niki Maleki, and Negin Daneshpour. 2009. A brief survey of software architecture concepts and service oriented architecture. In *2009 2nd IEEE International Conference on Computer Science and Information Technology*. 34–38. https://doi.org/10.1109/ICCSIT.2009.5235004

[23] Mingrui Yu, Hanzhong Zhong, and Xiang Li. 2021. Shape Control of Deformable Linear Objects with Offline and Online Learning of Local Linear Deformation Models. *CoRR* abs/2109.11091 (2021). arXiv:2109.11091 https://arxiv.org/abs/2109.11091

[24] Manuel Zürn, Markus Wnuk, Christoph Hinze, Armin Lechler, Alexander Verl, and Weilang Xu. 2021. Kinematic Trajectory Following Control For Constrained Deformable Linear Objects. In *International Conference on Automation Science and Engineering*. Vol. 17. 1701–1707.