Learning Vision-Based Tactile Skills for Robotic Architectural Assembly

Lernen kamerabasierter taktiler Fähigkeiten für die architektonische Robotermontage

Master-Thesis von Alymbek Sadybakasov aus Bischkek/Kirgisistan

Februar 2019

TECHNISCHE UNIVERSITÄT DARMSTADT

Learning Vision-Based Tactile Skills for Robotic Architectural Assembly
Lernen kamerabasierter taktiler Fähigkeiten für die architektonische Robotermontage

Vorgelegte Master-Thesis von Alymbek Sadybakasov aus Bischkek/Kirgisistan

1. Gutachten: Prof. Ph. D. Jan Peters
2. Gutachten: M.Sc. Boris Belousov

Tag der Einreichung: 19.02.2019

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Alymbek Sadybakasov, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum / Date: Unterschrift / Signature:

_____ _____

Abstract

Assembly tasks require rich feedback that enables tracking and adjustment of task execution parameters to unforeseen changes in the environment. Due to recent breakthroughs in computer vision, many architectural systems use video and depth cameras to guide assembly through sensory feedback. However, visual feedback alone is insufficient for performing manipulation tasks that crucially depend on contact with external objects and require force estimation at contact points. Although conventional force-torque sensors can be used to extend robot capabilities in this direction, they tend to be expensive.
In this thesis, we investigate the feasibility of employing an inexpensive vision-based alternative in closed-loop control scenarios – a variant of the FingerVision sensor that leverages the advances in computer vision to compensate for the lack of problem-specific hardware. We develop and evaluate a range of feedback controllers that use various modalities provided by the FingerVision sensor as input. In particular, by combining tactile feedback with real-time gripper and robot control algorithms, we demonstrate grasp adaptation, object shape and texture estimation, slip and contact detection, force and torque estimation. Furthermore, we show that the data delivered by the sensor is of sufficient quality to enable learning of auxiliary tactile skills – mapping directly from contact sensations to the force applied at an object in contact and to the viscosity and granularity of a substance being stirred by means of a spoon. Finally, the proposed tactile feedback controllers and learned skills are combined together to demonstrate applicability and utility of tactile sensing in collaborative human-robot architectural assembly tasks.

Zusammenfassung

Architektonische Montagen erfordern umfangreiches Feedback, mit dem die Nachverfolgung und Anpassung von aufgabenspezifischen Parametern an die unvorhergesehene Umgebungsänderungen möglich ist. Viele architektonische Systeme verwenden dank den jüngsten Durchbrüchen im Bereich maschinelles Sehen Video- und Tiefenkameras, um die Montagen mit Hilfe vom Sensorfeedback zu steuern. Feinmanipulationen erfordern jedoch die Einschätzung von Kontaktkräften, die von Kraft- bzw. Drehmomentsensoren vorgenommen werden. Zwar können die Roboterfähigkeiten mit den üblichen Kraft- bzw. Drehmomentsensoren erweitert werden, deren Herstellungskosten bleiben tendenziell hoch.
In dieser Arbeit untersuchen wir die Möglichkeit, eine kostengünstige Kamera-basierte Alternative in Regelungsszenarien einzusetzen - eine Variante des FingerVision-Sensors, die die Fortschritte in der Computer Vision ausnutzt, um den Mangel an problemspezifischer Hardware zu kompensieren. Durch die Kombination aus taktilem Feedback mit Echtzeit-Greifer- und Robotersteuerungsalgorithmen demonstrieren wir die Greifanpassung, die Einschätzung der Objektform- bzw. -text, die Schlupf- bzw. Kontakterkennung und die Einschätzung von Kräften bzw. Drehmomenten. Ferner zeigen wir, dass die vom Sensor gelieferten Daten von ausreichender Qualität sind, um das Erlernen zusätzlicher taktiler Fähigkeiten zu ermöglichen - das Abbilden von Kontaktempfindungen auf die Kraft, die an einem in Kontakt befindlichen Objekt angelegt wird, sowie auf die Viskosität und Granularität einer Substanz, die mittels eines Rührwerks gerührt wird. Schließlich werden die vorgeschlagenen taktilen Feedback-Steuerungen und die erlernten Fähigkeiten kombiniert, um die Anwendbarkeit und die Nutzbarkeit der taktilen Wahrnehmung für kollaborative Mensch-Roboter-Montage zu demonstrieren.

Acknowledgments

I would like to express my very great appreciation to Boris Belousov for offering me this topic and his support throughout this thesis. Boris always provided me with valuable comments and guided me towards finishing this thesis. Many thanks should be given to Bastian Wibranek from the Digital Design Unit group at TU Darmstadt for his contribution to this thesis and who supported me from the architectural point of view.
I would also like to acknowledge Jan Peters for giving me the opportunity to work on the real robots during the integrated projects and this thesis.
Special thanks to Christian Betschinske for working together on making the FingerVision.
I would like to thank my mother for her enormous support during my studies. This long journey would not be possible without her.
My great gratitude goes to my lovely fiancée. Her love, patience and support always inspired me to go beyond the limits.

Contents

Figures and Tables

List of Figures

List of Tables

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
2D	Two-Dimensional
3D	Three-Dimensional
API	Application Programming Interface
CCD	Charge-Coupled Device
DDU	Digital Design Unit group (TU Darmstadt)
GAN	Generative Adversarial Network
HSV	(Hue, Saturation, Value)-color model
JSON	JavaScript Object Notation
LBFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm
MLP	Multi-Layer Perceptron
MSE	Mean-Squared Error
PC	Personal Computer
PID	Proportional-Integral-Derivative controller
robcom	ROBot COMmunication interface
ROS	Robot Operating System
UI	User Interface
VAE	Varitional AutoEncoder

List of Symbols

Notation	Description
μ_{pq}	Central moments of order $(p+q)$
$\mathbf{f}$	Contact force vector estimated by FingerVision
$\boldsymbol{\tau}$	Contact torque vector estimated by FingerVision
L	External input for the leaky-integrator controller
f_ϵ	Force threshold
M^{pq}	Geometric moments of order $(p+q)$
H	Histogram value
M^{00}	Image area
$\{\overline{x}, \overline{y}\}$	Image centroid
α	Leakage factor for the leaky-integrator controller
$\{x_d, y_d\}$	Position shift of the d-th blob on the image plane
s_d	Size shift of the d-th blob on the image plane
t	Time step
τ_ϵ	Torque threshold

List of Operators

Notation	Description	Operator
cov	Covariance matrix of a matrix A	cov(A)
vec	Vectorization of a matrix A	vec(A)

1 Introduction

Assembly tasks in architecture rely on rich sensor feedback for compensation tolerance to allow the use of poorly described building elements. Feedback is commonly provided by external video or depth cameras. Additionally, force and tactile sensors enable tracking and adjustment of task execution parameters to unforeseen changes in the environment. However, the manufacturing costs of traditional tactile sensors tend to be high. At the same time, a recent breakthrough in computer vision brought a new class of tactile sensors that employ a camera as the sensing unit. Such vision-based tactile sensors provide estimations of both contact forces and tactile feedback such as slip and geometrical information, while remaining cheap at the same time.
In this thesis, we explore the usage of vision-based tactile sensors in architectural assembly tasks. As the first step, we enhance the design of a vision-based tactile sensor FingerVision [1] according to the architectural requirements. We improve the quality of the contact force estimation by integrating a Kalman filter into the blob tracking algorithm. We prepare the usage of FingerVision for architectural assembly by introducing a class of tactile-feedback controllers and evaluating them in the experiments with a KUKA arm with 7 degrees of freedom. Moreover, we introduce two novel tactile skills through exploiting the capabilities of FingerVision - mapping the contact forces onto real force values of a pressing movement and learning to distinguish between various types of substances through stirring. Finally, we demonstrate the first usage of a vision-based tactile sensor in architectural assembly. For this purpose, we build a small scale model in which a robot moves building elements into the place of highest load on the roof. Naturally, the changes in load distributions can be identified by utilizing the contact force estimation. We evaluate the usage of FingerVision in the assembly task in which the wall elements need to be placed inside a miniature model of a building. The introduced tactile-feedback controllers were sufficient for building a working demonstration in which a human collaborates with the robot and takes advantages of using a tactile sensor for architectural assembly.

1.1 Related Work and Contribution

We provide a brief overview of conducted work in three related research areas - vision-based tactile sensors, tactile-feedback and learning tactile skills, as well as robotics in architecture. We additionally outline our contribution to each mentioned research area.

1.1.1 Vision-Based Tactile Sensors

The force sensor is a device that determines physical properties of the objects in the world through contact with them [2]. Thus, a typical sensor is based on projecting the physical domain into electrical domain, whereas the projection can be divided into four main types: capacitive, piezo-resistive, piezoelectric and optic [3]. However, a production-ready force sensor based on the first three mentioned principles tends to be too expensive. In contrast, optical-based, or more precisely, vision-based sensors employ the camera as a sensing mechanism which reduces the manufacturing costs significantly. Another significant advantage of recent vision-based sensors is that they bring additional tactile information, such estimation of the slippage, which is extracted using computer vision algorithms.
One of the early attempts at building a vision-based force sensor used the camera to measure the light intensity of the flexible contact medium which was painted with a reflecting color [4]. They could examine the magnitude and distribution of contact forces by measuring the changes in light intensities. Later, the contact forces were estimated using a CCD camera which tracked the shift of markers with different colors [5, 6]. Recent breakthrough in computer vision enabled to estimate not only contact forces but also other tactile sensing such as slip of an object or its orientation between fingers. FingerVision [1] and GelSight [7] are examples of such vision-based sensors as they both provide estimations of contact forces and tactile information. The contact medium of FingerVision is a soft silicon layer in which the small black spheres are inserted and the shifts of these spheres yield the estimation of the contact force during grasping movements. The tactile information is maintained by employing computer vision algorithms. In this thesis, we extend their work by adapting the design of FingerVision to architectural tasks and improving the methods that estimate contact forces and tactile information.

1.1.2 Tactile-Feedback Control and Learning Tactile Skills

Tactile-feedback robot controllers are closed-loop controllers that exploit the information from tactile sensors to control the robots. Several attempts have already been made at incorporating the tactile data into robot control. The tactile data is commonly provided by traditional force-torque sensors. For instance, the tactile data from a piezo-resistive force sensor was integrated into dynamic movement primitives to control the robot in manipulation tasks [8]. Exploratory movements, such as applying normal force to a surface, can be planned by incorporating biomimetic tactile sensors into closed-loop control [9]. We, instead, exploit a vision-based tactile sensor for learning various skills such as recognizing collisions of grasped objects with external obstacles or following grasped and ungrasped objects. A tactile-feedback Cartesian controller serves as a closed-loop robot control.
The grasping method proposed in [10] evaluates the visual and tactile feedback before planing the grasping movement. Furthermore, object grasping can be stabilized by employing a leaky-integrator controller [11]. We extend their work by proposing a method which integrates the tactile information provided by FingerVision into the leaky-integrator controller for controlling the Sake gripper. The resulting tactile-feedback leaky-integrator controller is used in different grasping scenarios such as object handover and finding minimum effort for pulling up an object. Instead of adapting the grasping force using a simple control strategy [12], our method can be utilized in various grasping scenarios in a more general fashion.
Visual information can be utilized not only in robot control, but also for learning other tactile skills. For instance, [13] learned the viscosity of various liquids to discriminate between them by measuring changes of the liquids‘ surfaces that were captured with a depth camera, e.g. in order to adjust the pouring velocity. We two novel methods that utilize a vision-based tactile sensor for learning auxiliary tactile skills. The first method introduces a classification model which distinguishes between specific substances through stirring them, whereby the tactile information from FingerVision is utilized to train the classifier. The second method proposes a regression model maps the contact forces onto real force value of a pressing movement. Evaluations prove the efficiency of both methods.

1.1.3 Robotics in Architecture

Using robots for architectural tasks gains a lot of attention. Robotic fabrication systems with sensor feedback bridge the material and digital worlds, enabling new concepts within the field of construction [14, 15]. Real-world feedback in robotic fabrication was used, e.g. to capture human gestures on-site to construct architectural elements [16], assemble wooden rod structures based on force-torque sensors [17] and to generate robotic motion control based on gestures and sensor inputs [18]. The real-time modification of a robotic fabrication can be allowed by providing construction information from the integrated force sensors [19], depth-cameras [20], or motion-capture markers mounted on a reciprocating electric tool [21]. Still, no work has demonstrated the advantages of using vision-based tactile sensors for architectural tasks. Therefore, we demonstrate the first usage of a vision-based tactile sensor, enabled by a novel combination of two types of information provided by the tactile sensor - surface deformation and slippage estimation. We build a small scale model in which a robot can move a vertical element into the place of highest load. Instead of defining desired locations for the building elements, the load placed on the top surface serves as the guiding medium. The tactile sensor used in the assembly task is our implementation of FingerVision which has been designed according to the requirements of the assembly task.

1.2 Structure

The structure of this thesis is as follows. Chapter 2 describes the design of FingerVision as well as the algorithms for extracting the tactile information. Chapter 3 introduces a new class of tactile-feedback robot controllers and conducts experimental evaluations of the proposed controllers. Moreover, it proposes new methods for learning to map contact forces estimated by FingerVision onto real pressing forces, and distinguishing between various types of substances through stirring. Chapter 4 evaluates the usage of FingerVision in an architectural assembly task by combining the low-level tactile-feedback controllers in a closed-loop control strategy. Chapter 5 outlines the thesis and discusses possible directions for future work.

2 Vision-Based Tactile Sensor FingerVision

Tactile sensing has been a promising technology in grasping scenarios for decades. Unlike traditional force and tactile sensors, FingerVision uses a cheap and easily accessible video camera to detect deformations of a touch layer to infer not only tactile information and contact forces but also object properties. Recent advances in computer vision made it possible to create algorithms that process the data coming from the camera at an acceptable speed for using in robot controllers.
The assembly task from architecture required a special design of the FingerVision sensor for mounting on our gripper. In addition, several modifications and improvements to the original computer vision algorithms were introduced to accommodate for the challenges of the presented assembly environment.
In this chapter, we first describe the making process of an improved FingerVision. Next, we describe the original force detection algorithm and an additional improvement using a Kalman filter. Finally, we explain the concept of proximity vision which is a collective term denoting both slippage and object properties extracted through computer vision.

2.1 Making FingerVision

The initial prototype of FingerVision was produced by utilizing materials and sketches presented in [1]. However, several modifications had to be done to the design to use the sensor in the architectural manipulation tasks. This section provides the manufacturing process of the improved FingerVision.

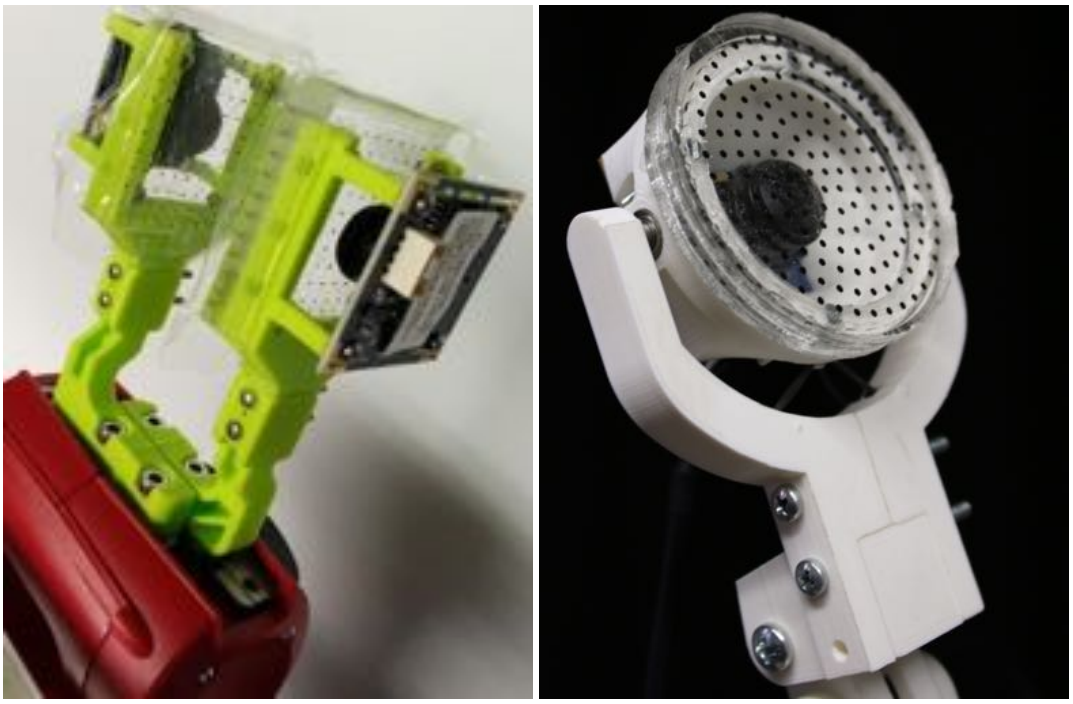
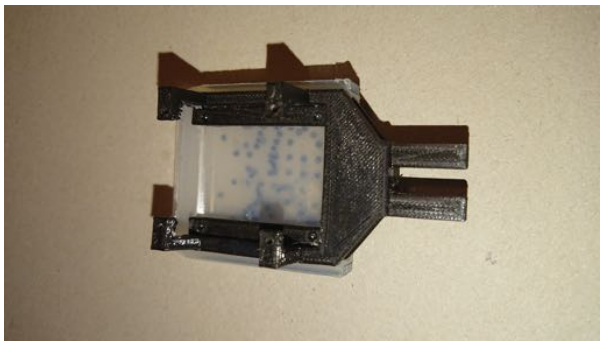


Figure 2.1: The original[1](left) and our (right) version of FingerVision. Our implementation of the sensor was especially designed such that it could be mounted not only on parallel grippers. This ability was achieved by making the rounded plastic carrier flexible relatively to the rest of the construction. A rubber band ties the carrier to prevent too loose movements.

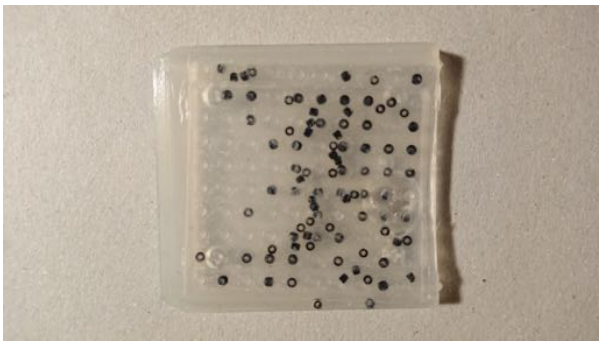
2.1.1 Improved Properties of Silicon Layer

Transparency and softness of the silicon greatly affect the quality of blob tracking and proximity vision algorithms. For instance, the blobs could hardly be determined as those in a translucent silicon. A too soft silicon broke during

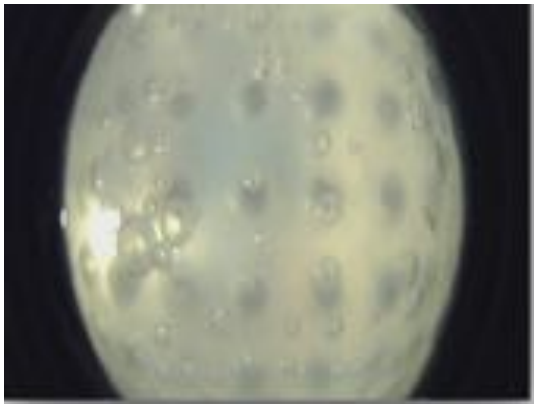
[1] Figure was obtained from `http://akihikoy.net/p/fv.html` (accessed on February 10, 2019)



(a) Assembled FingerVision constructed using the materials and sketches presented in [1].



(b) An illustration of the issue in which the micro breads were continuously losing their initial positions inside the silicon layer.



(c) A view from the camera showing the translucent silicon layer. Obviously, the blob tracking algorithm had difficulties in tracking the blobs.

Figure 2.2: The first prototype of FingerVision.

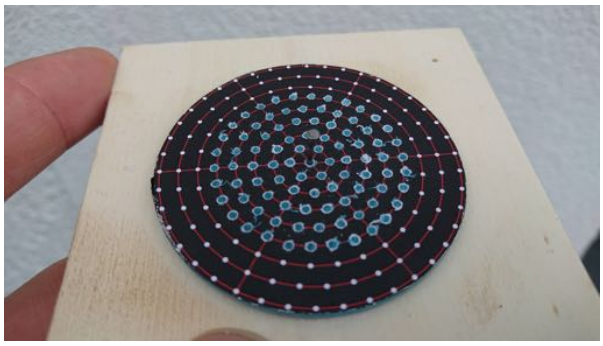
the molding process while a too hard silicon is not deformed enough during gentle in-hand manipulations. The first prototype of FingerVision is shown Figure 2.2(a). Therefore, several silicon types were tested until a certain type of silicon called “Replisil 19 N glasklar" was selected having the desired properties. In order to raise the softness of the silicon, the tactile mutator “Slacker" has been employed as proposed in [22] such that the following elasticity of the silicon was soft enough for gentle manipulations. Additionally, the additive “Slo-Jo" was employed to extend the dripping time up to 30 minutes. The increased dripping time granted sufficient time for infusing the blobs into the silicon. Another property of the resulting silicon was the its extended durability. After several months of intensive experiments, the silicon was still transparent and in a good condition.

2.1.2 Injecting Blobs into Silicon Layer

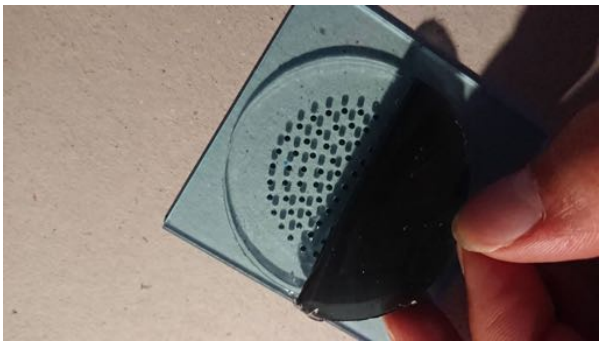
In the initial prototype of FingerVision, the micro breads were shaking and, therefore, losing their position inside the silicon layer as can be seen in Figure 2.2(b). The blob tracking algorithm identified these shakes as the deformation and produced wrong contact force estimations. To tackle this problem, an iron dioxide was injected into a clear silicon by employing a special form as shown in Figure 2.3(b). The blobs were created with a circle shape. The circle shape suited best for the blob tracking algorithm since there was no ambiguity in determining their shapes locations. Utilizing the form for injecting blobs made them equally distributed inside the silicon. The equal distribution was required for the blob tracking algorithm to prevent intersections of the blobs in the same region of interest.

2.1.3 Making Gripper Fingers and Mounting the Sensor

The fingers on which the original FingerVision was mounted were parallel and had only one degree of freedom, i.e. the actuated part of the gripper. While such fingers worked well for a parallel gripper, they were not optimal for our gripper as our gripper had already had two non-parallel and underactuated fingers with two phalanges. The original FingerVision



(a) The form used for injecting the iron dioxide into the silicon layer.



(b) Injecting iron dioxide into the silicon layer.

Figure 2.3: An illustration of the method for injecting blobs (iron dioxide) into the silicon layer. The blobs do not move inside the layer increasing the quality of the blob tracking algorithm.

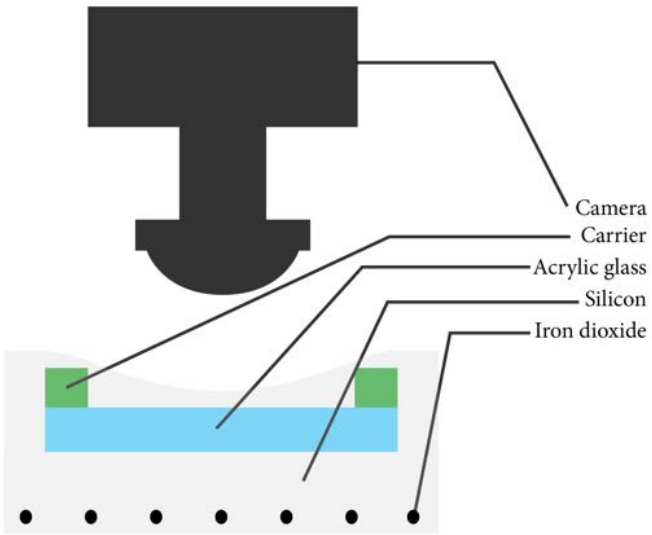


Figure 2.4: A schematic design of the modified FingerVision.

mounted on such a gripper had difficulties in grasping big objects. To tackle this problem, the link between the round carrier and the finger was done to be freely moving. This flexible link automatically adjusted the angle of the fingers that was needed to grasp items of different sizes. A fish-eye camera was mounted inside the round carrier and served as the sensing component. An acrylic glass was attached to the other side of the carrier and for a better adhesion with the silicon layer. Additionally, a rubber band tied the silicon carrier with the rest of construction in order to straighten the carrier while no object was grasped and, thus, prevented failed grasps. All white plastic parts of the sensor were printed with a 3D-printer. Figure 2.1 shows the assembled FingerVision which has been used in the experiments. The FingerVision was attached to the left finger while the right finger carried the same model but without the silicon layer and the camera. The modified FingerVision was manufactured in cooperation with DDU[2].

2.2 Estimation of Contact Forces

Contact forces are estimated using the black blobs inside the silicon layer. Since the silicon layer is soft, it is being deformed during the grasp and the dots are moving relatively to the camera as shown in Figure 2.6. The position shift of a blob on the image plane corresponds to the horizontal forces and the size change corresponds to a norm force. In this section, we first describe the blob detection method which is required to track the blobs . We will also show the calibration process which is needed to be performed once in the beginning of the actual blob tracking. Then, we will explain the concept of a Kalman filter and reveal how it is incorporated into the blob tracking method with the aim to reduce the oscillations of the detected blobs.

[2] Digital Design Unit group, Department of Architecture, TU Darmstadt

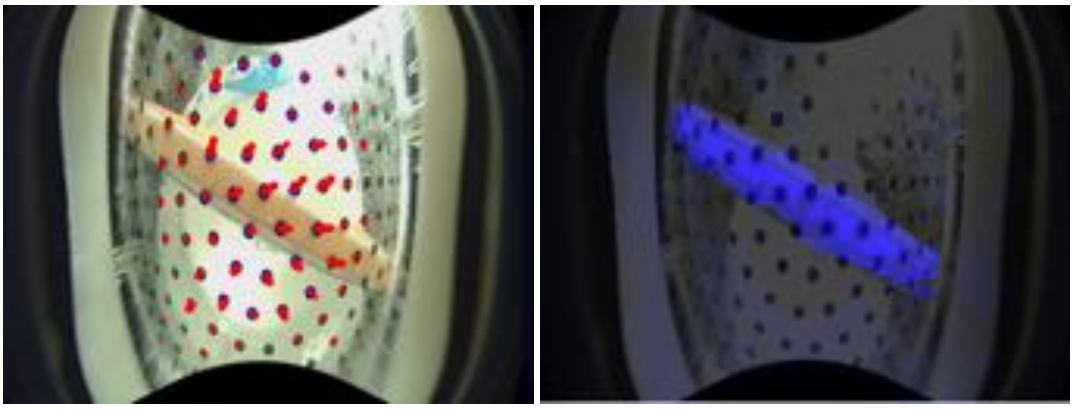


Figure 2.5: Force tracking (left) and proximity vision (right) views from FingerVision while grasping an orange pen.

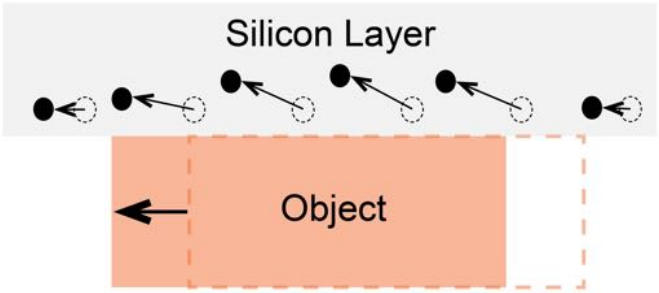


Figure 2.6: Blob movements during the contact with an object. The deformation of the silicon layer tends to be higher in the area of contact.

2.2.1 Blob Detection Algorithm

The blob detection algorithm of FingerVision relies on the `SimpleBlobDetector` class available in the OpenCV library [1]. `SimpleBlobDetector` consists of two main steps:

- Preprocess the image
- Detect blobs

The original detection (Algorithm 1) is done on the preprocessed image. First, all contours available in the image are identified by utilizing the OpenCV method `findContours()` and threaten as blob candidates. Next, each candidate is eliminated using the following filters. First, those candidates are excluded whose area are smaller or bigger than the actual blobs which all have the same area on the image plane. Next, the perimeter of a candidate is computed. The perimeter is used to calculate the circularity ratio by using the Polsby-Popper test which tells how circular the contour is. Thus, the candidates whose circularity ratio is not close enough to a perfect circle will be dismissed. Next, the convex hull area of a candidate is calculated and the candidate is discarded if the its ratio area/hull area is less than the minimum predefined ratio. This step is needed to remove the candidates with a concave shape since all real blobs have a convex or an almost convex shape. After filtering by convexity, those candidates are excluded whose color is not as dark as of the real blobs which all have a black color. Finally, only those candidates remain which correspond to the real blobs.

Algorithm 1: The blob detection algorithm.

```
contours ← findContours()
for contour ∈ contours do
    filterByArea(contour)
    filterByCircularity(contour)
    filterByConvexity(contour)
    filterByColor(contour)
end
```

2.2.2 Calibrating Blobs for Tracking

The calibration is needed to detect the initial locations and sizes of the blobs. This information will be then utilized for calculating the displacements of the blobs and, subsequently, to estimate the forces. The sensor is covered with a white paper to cut off the background. The locations and sizes of blobs are detected by running Algorithm 1 and saved for tracking the blobs.

2.2.3 Tracking Blobs for Estimating Forces

The calibration provides the initial locations and sizes of the blobs. This information is used to track the blobs by measuring the difference between the initial and current locations and sizes. We denote as x_o, y_o and s_o the initial x- and y- coordinates and size of a blob, respectively. Analogously, x_c, y_c and s_c are the current x- and y- coordinates and size of a blob, respectively. Thus, the differences of coordinates and size are

$$\begin{aligned} \{x_d, y_d\} &= \{x_c - x_o, y_c - y_o\} \\ s_d &= s_c - s_o. \end{aligned} \tag{2.1}$$

To retrieve the force values, the differences can be either used as is or using the following mapping function

$$\mathbf{f} = \frac{1}{N} \sum [f^x, f^y, f^z].$$

The torque is estimated as

$$\boldsymbol{\tau} = \frac{1}{N} \sum \mathbf{r} \times [f^x, f^y, f^z], \tag{2.2}$$

where $\mathbf{r}$ is a vector of blob positions. The method for estimating and tracking blobs was originally proposed in [1].

2.2.4 Improving Blob Tracking Using Kalman Filter

The original blob tracking algorithm identified some of the blobs as moving even if they did not actually move. The tracking algorithm failed because of frequent slight changes of the light condition in the environment. To eliminate wrongly detected movements, a real-time Kalman filter was integrated into the blob tracking algorithm to cut off sudden shifts of the blobs with a too high speed.
In the following subsections, we first describe the concept of the Kalman filter. Next, we show how the Kalman filter is incorporated into the blob tracking algorithm.

Kalman Filter

Kalman filter [23] works, under assumption that the transition model is linear, in two steps - prediction and update. The linear transition model is modeled as

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \epsilon_t, \tag{2.3}$$

where $\mathbf{x}_t$ is the state of the model at the current time step, $\mathbf{A}$ is the state transition model and ϵ_t is the zero mean Gaussian noise, i.e. $\epsilon_t \sim \mathcal{N}(0, \mathbf{T}_t)$. In the prediction state, the a priori state and error covariance are estimated as

$$\mathbf{x}_t^- = \mathbf{A}\mathbf{x}_{t-1}^+ \boldsymbol{\Sigma}_t^- = \mathbf{A}\boldsymbol{\Sigma}_{t-1}^+ \mathbf{A}^T + \mathbf{Q}_t.$$

In the update step, the a priori state is updated with

$$\begin{aligned} \mathbf{r}_t &= \mathbf{o}_t - \mathbf{H}\mathbf{x}_t^- \\ \mathbf{K}_t &= \boldsymbol{\Sigma}_t^- \mathbf{H}^T (\mathbf{H}\boldsymbol{\Sigma}_t^- \mathbf{H}^T + \mathbf{O}_t)^{-1} \\ \mathbf{x}_t^+ &= \mathbf{x}_t^- + \mathbf{K}_t \mathbf{r}_t \\ \boldsymbol{\Sigma}_t^+ &= (\mathbf{I} - \mathbf{Q}_t \mathbf{H})^{-1} \boldsymbol{\Sigma}_t^-, \end{aligned} \tag{2.4}$$

where $\mathbf{r}_t$ is the measurement innovation, or the residual, $\mathbf{o}_t$ is the measurement with the measurement matrix $\mathbf{H}$, $\mathbf{K}_t$ is the Kalman gain, $\mathbf{x}^+$ and $\boldsymbol{\Sigma}_t^+$ are the updated state mean and covariance, respectively.

Improved Blob Tracking Algorithm

Since we are interested in correcting the $x-$ and $y-$displacements of the blobs, we define as the state of the system $\mathbf{x}_t = \begin{bmatrix} x_0 & x_t & y_0 & y_t & s_t \end{bmatrix}^T$, where x_0 and y_0 stand for the original position of a particular blob and x_t and y_t stand for the position shift during the tracking. The transition matrix for **A** and the measurement matrix **H** from Equation (2.3) and Equation (2.4) have the following form:

$$\mathbf{A} = \begin{bmatrix} 1 & dt & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The diagonal elements of the measurement noise, priori and posteriori error estimate matrices are set to 0.1. The diagonal elements of the process noise covariance matrix are set to 0.01. The initial predicted state is initialized with $\mathbf{x}_0^- = \begin{bmatrix} x_0 & 0 & y_0 & 0 & s_0 \end{bmatrix}^T$. We set $dt = 1/15$ since 15 is the maximum FPS of the camera.Equation (2.1) is now modified as follows:

$$\{x_d, y_d\} = \{x_c^+ - x_o, y_c^+ - y_o\}$$
$$s_d = s_c^+ - s_o,$$

where x_c^+, y_c^+ and s_c^+ are the corrected $x-$ and $y-$locations as well as size of a blob, respectively.

Evaluation

We evaluated the improved tracking algorithm by comparing it with the original tracking algorithm. We first grasped a stick with the maximum effort and let both algorithms detect the blobs for 5 seconds. During this time frame, we collected the data from blob movements and calculated their average value. The average values were additionally filtered by employing a low-pass filter to emphasize the difference between the both signals. Figure 2.7 shows an example series of filtered average values produced by both methods. Although both series contain several hops, the series produced by the improved algorithm has fewer of them. Using force values with fewer hops is beneficial for force-feedback controllers as the resulting joint or Cartesian trajectories become smoother.

2.3 Proximity Vision

Proximity vision refers to image processing method with the aim to obtain the geometrical properties of grasped objects, such as position, orientation and slippage [1]. In this section, we first represent the concept of the histogram computation and how it is used in the proximity vision algorithm. Next, we specify the calibration step which is necessary to distinguish grasped objects from the background. Finally, we show how the tactile and geometrical properties mentioned above are extracted through the proximity vision.

2.3.1 Histogram of Colors

To track the object using FingerVision, a histogram of colors is used. A histogram is denoted as $H(h,s)$ [1], where h and s are hue and saturation components of the (Hue, Saturation, Value)-color model (HSV) respectively. The histogram is calculated using several frames from the camera of FingerVision. Next, the average of previously computed histograms is taken. To improve the detection of dark objects, we additionally use the value component of the HSV, i.e. the notation for the histogram becomes $H(h,s,v)$.

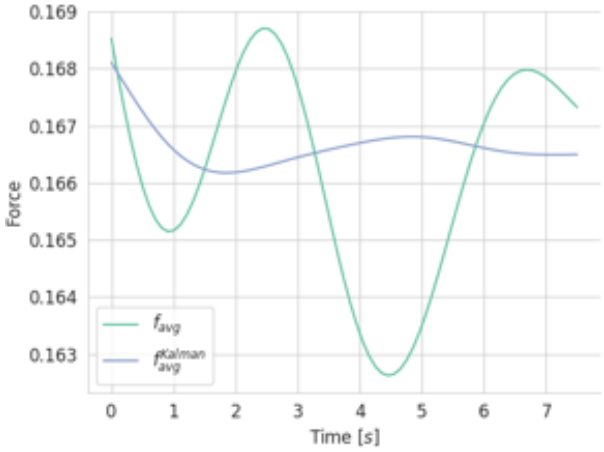


Figure 2.7: A comparison between the average blob movements tracked with and without the Kalman filter. The values were filtered using a low-pass filter to emphasize the difference between two methods.

2.3.2 Calibration

The calibration of proximity vision is needed to initialize the histograms of the background environment and the objects that will be grasped. In the first step of the calibration, the background histogram $H_{bg}(h,s,v)$ is computed, i.e. when no object is currently grasped (Figure 2.8(a)). After computing the background histogram, the object is placed between the fingers of the gripper and the region on which the object lies is manually defined on the image with a mouse using the HSV. For each region, a separate histogram $H_{def}(h,s,v)$ is computed. Simultaneously, the pixels with similar histogram values are identified as belonging to the same object indicating whether the computed histogram covers fully the object. Thus, the operation is performed multiple times for different light conditions and orientations of the object until such $H_{obj}(h,s,v)$ is found which covers the whole object in as much as possible settings of the environment. $H_{obj}(h,s,v)$ is updated after each manual definition [1]

$$H^{+}_{obj}(h,s,v) = min(255, H^{-}_{obj}(h,s,v) + f_{gain} max(0, H_{def}(h,s,v) - f_{bg} H_{bg}(h,s,v))),$$

where f_{bg} and f_{gain} stand for constant normalization factors.
To sum up, the calibration has the following steps:

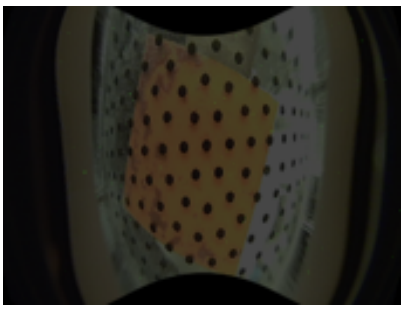
1. Compute histogram $H_{bg}(h,s,v)$
2. Grasp a test object, mark manually the relevant locations on the image using the UI and update $H_{obj}(h,s,v)$ after each marking operation.
3. Save $H_{obj}(h,s,v)$.

The advantage of computing the histograms of colors is that the background environment and the grasped objects can easily be differed by masking the current image with the previously saved background histogram. However, this approach still causes the issue in which objects from the background which have the same color as the grasped object will also be classified as grasped objects. To tackle this complication, we agreed to paint the test objects with the one color. Hereupon, a bright orange color was the most promising color for both force and slip detection algorithms.
To evaluate the calibration quality of proximity vision, the gripper grasped a stick with an orange and square tip. We manually rotated it between the fingers by approximately 90 degrees and collected the series of orientations, areas and locations inside the gripper. The desired calibration quality was achieved if there was a minimum oscillation of trajectories of collected series. Figure 2.8 shows an example of a desired calibration with series of stick's orientation, area and center. The minimal oscillation was not critical for future experiments.

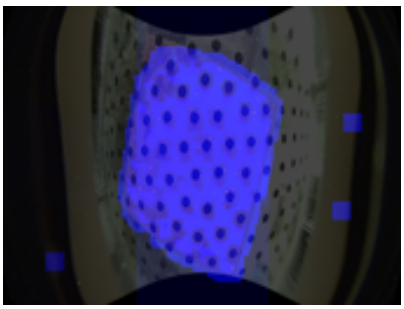
2.3.3 Tracking the Object

The calibration of proximity vision provides necessary data for tracking the object on the image plane of FingerVision. The object is tracked in the following way. The current image taken with the sensor's camera is back projected using the histogram of the object $H_{obj}(h,s,v)$ which has been gathered in the calibration step. The back projection cuts off the pixels whose histograms are not identical to $H_{obj}(h,s,v)$ and, therefore, retrieves the pixels of the object [1].

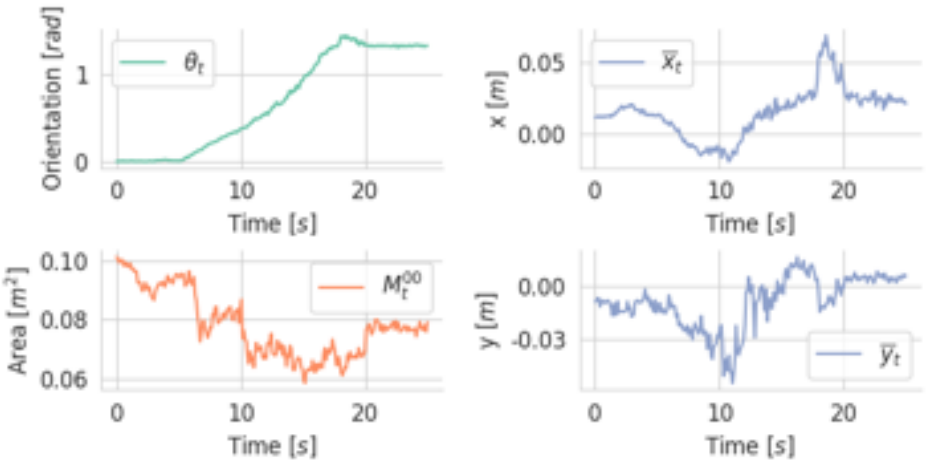
(a) The background histogram is computed while no object is grasped.



(b) Grasping the object. The object is not detected yet.



(c) Manually defining the object on the image with the mouse until the whole object is covered with blue pixels.



(d) Series of object's orientations, areas and centers during the rotation of the stick inside the gripper. The stick was manually rotated by 90 degrees.

Figure 2.8: 3 steps performed during the calibration of proximity vision (a)-(c) and calibration results (d).

2.3.4 Information Obtained Through Proximity Vision

Once the object's pixels are found, the tactile and geometrical information about the object can be extracted, i.e. whether the object slips and its orientation, position and area relatively to field of view of the camera. In this section, we describe how this information is obtained from the image by processing the relevant pixels that were gained in the tracking step.

Slippage Detection

A slip of an object is viewed as movement of object's pixels on the image plane [1]. During the slippage detection, a histogram for the current image from the camera is computed. The pixels are discarded if their histogram model is similar to histogram model of the background. The remaining pixels are masked using the saved histogram of the object. After the masking operation, only those pixels remain that correspond to the current object. Finally, the position of the object's pixels from the previous frame are compared to the newly detected pixels and the displacement of locations is considered as the slippage.

Exploiting Geometrical Information

The pixels that were extracted by tracking the object using proximity vision form an additional subimage. This subimage is used for deriving the geometrical features such as centroid, area and orientation using geometric moments of an image [1].
Geometric moments of order $(p+q)$ are the linear projections of the image function onto a polynomial basis [24], i.e

$$M^{pq} = \int\int x^p y^q f(x,y) dx dy.$$

For a discrete image, their approximation is defined as

$$M^{pq} = \sum_x \sum_y x^p y^q f(x,y). \tag{2.5}$$

The mass M^{00} refers to area of the image and the center of mass of the image refers to centroid and has the following form

$$\{\overline{x}, \overline{y}\} = \left\{ \frac{M^{10}}{M^{00}}, \frac{M^{01}}{M^{00}} \right\}. \tag{2.6}$$

The orientation of the image is the axis of least inertia and can be obtained from the central moments. The central moments of order $(p+q)$ are defined as

$$\mu_{pq} = \sum_x \sum_y (x-\overline{x})^p (y-\overline{y})^q f(x,y).$$

Next, covariance matrix is constructed using the second order central moments

$$\mu'_{20} = \frac{\mu_{20}}{\mu_{00}} = M^{20}/M^{00} - \overline{x}^2,$$
$$\mu'_{02} = \frac{\mu_{02}}{\mu_{00}} = M^{02}/M^{00} - \overline{y}^2,$$
$$\mu'_{11} = \frac{\mu_{11}}{\mu_{00}} = M^{11}/M^{00} - \overline{xy},$$

and the covariance matrix of the image is

$$\mathrm{cov}[I(x,y)] = \begin{bmatrix} \mu'_{20} & \mu'_{11} \\ \mu'_{11} & \mu'_{02}. \end{bmatrix}$$

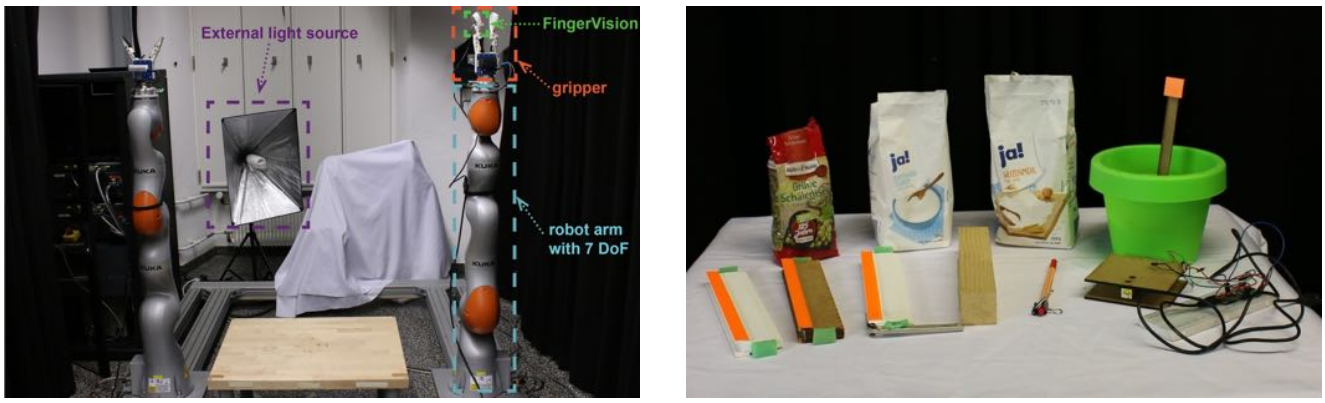
Finally, the orientation is the angle of the eigenvector with the largest eigenvalue

$$\theta = \frac{1}{2} \arctan \frac{2\mu'_{11}}{\mu'_{20} - \mu'_{02}}. \tag{2.7}$$

Now that the geometrical features of the subimage are known, they correspond to the geometrical features of the grasped object. Other image features such as an orientation of the axis of least inertia and symmetry properties can be obtained through exploiting the image moments of higher order. We, however, will use only area, centroid and orientation of the object throughout this thesis.

3 Incorporating Tactile Information Into Robot Control and Learning Tactile Skills

Once the tactile information is extracted by FingerVision, it can be utilized for understanding special tactile skills such as recognizing a collision with obstacles, following grasped and ungrasped objects as well as in-hand object manipulations based on the tactile- and torque-feedback. Moreover, the tactile information can be exploited for learning to predict real force values of pressing movements and to distinguish between specific crumbly substances.
In this chapter, we first show how the tactile information from the FingerVision is incorporated into robot and gripper controllers. Next, we evaluate the proposed tactile-feedback controllers in the experiments on a lightweight robotic arm. Moreover, we learn to predict real force values of pressing movements and types of specific substances through stirring. Finally, we discuss the limitations that we have encountered during the experiments as well as propose possible workarounds.



(a) Two light-weight KUKA arms with 7 degrees of freedom with grippers attached to each arm. FingerVision is mounted on the right arm. An external source of light in the background serves for compensating for changing light conditions during movements of the arm with FingerVision.

(b) Supporting materials: two sticks with orange stripes and different colors, a white stick with an orange stripe and metallic piece at the head, a red-orange pen with an additional attached weight, an Arduino weight scale, a bucket with a stick inside used for stirring, and three packages with flour, sugar and peas.

Figure 3.1: Experimental setup.

3.1 Hardware and Software Setup

Two lightweight KUKA arms each with 7 degrees of freedom were available, whereby the experiments in this chapter were conducted on the right arm (Figure 3.1(a)). We use MoveIt! package to generate trajectories which do not rely on the tactile data from FingerVision. We switch to using Cartesian control once we the tactile feedback controllers need to be executed. In both cases, the robot receives the commands via ROBot COMmunication interface (robcom) [25] that generates joint trajectories using B-Splines. Tactile-driven trajectories are computed and followed by calling the `go_cart(position, orientation, execution_time)`-method, where `position` is the 3D-position of the end effector and `orientation` is the pose of end effector given in quaternion. Both, `position` and `orientation` are defined relatively to the base frame. Analogously, we use robcom to control the joints directly by calling the command `go_joints(joints, execution_time)`, where `joints` is an array with desired joint positions. The control frequency of the arm is at 1000 Hz. We, however, control the robot at the frequency of 0.06 Hz since the camera of FingerVision worked with a maximal frequency of 15 frames per second. The controller gains were tuned to ensure the best stability during executions of trajectories.
A gripper is mounted on the last link of the arm. The gripper's actuator is the Robotis Dynamixel MX-64AR with a Proportional-Integral-Derivative controller (PID) built into its firmware. Since we did not have direct access to the PID, we were forced to use the provided Python interface to control the gripper, i.e. the gripper closes or opens to a desired position with a desired effort. The desired position corresponds to the first finger while the second finger is controlled

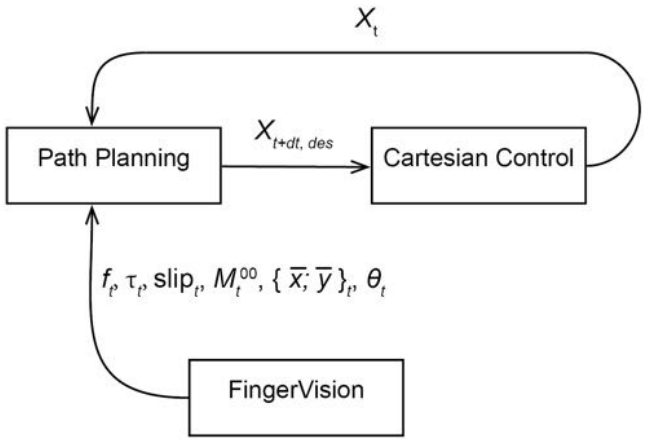


Figure 3.2: A schematic design of a general tactile-feedback Cartesian controller. The path planning unit plans the trajectory and sends the desired pose of the end effector in Cartesian space to the Cartesian control unit. The Cartesian control unit computes the trajectory from X_t to X_{t+1} using B-Splines, executes it and increments t. The path planning unit recomputes the trajectory based on the current tactile information provided by FingerVision.

indirectly by mirroring the first finger. For instance, calling `gripper_go(0, 100)` closes the gripper by applying the full effort while `gripper_go(100, 50)` opens the gripper with the half effort.
Finally, we implement each tactile-feedback controller in Python. The controllers process the sensory data from FingerVision and Realsense and send appropriate commands to the arm or the gripper.

3.2 Tactile-Feedback Control of the Arm

The tactile information from FingerVision is incorporated into closed-loop control systems in two ways: by modifying the Cartesian controller for controlling the whole arm in the task space and by utilizing a leaky-integrator controller for controlling the gripper. In this section, first describe different variations of tactile-feedback Cartesian controllers and, subsequently, evaluate each resulting controller in the experiments with the real robot arm.

3.2.1 Tactile-Feedback Control in Task Space

The pose of an end effector in Cartesian space is defined as $\mathbf{X} = \mathbf{x}\boldsymbol{\phi}^T$, where $\mathbf{x}$ is the position and $\boldsymbol{\phi}$ is the orientation of the end effector. Suppose that the end effector of the arm rests at some point X_0 and has to follow a trajectory $\mathbf{X}_{des}$ in the Cartesian space, whereby $\mathbf{X}_{des}$ contains the intermediate data points of the desired trajectory. The data points are sent to the robot with the frequency dt, i.e. $X_{t+dt} = X_{des,i}$, with $i \in \{0, \text{length}(\mathbf{X}_{des})\}$. The inverse kinematics solver of the robot calculates the path from X_t to X_{t+dt} using e.g. B-Splines and commands the robot to follow the computed trajectory within dt. This process repeats until the robot has passed all data points from $\mathbf{X}_{des}$. The advantage of the introduced Cartesian controller (illustrated in Figure 3.2) is that it allows to incorporate the tactile information from FingerVision in robot controllers with low effort. In the following subsections, we will introduce two tactile-feedback controllers based on the described method. The first controller recognizes collisions of the grasped object with obstacles and the second follows grasped and ungrasped objects. We evaluate each proposed controller in the experiments with the real robot.

Recognizing Collisions of the Grasped Object With Obstacles

Suppose that the gripper grasps an object with a constant force. In this case, the grasped object becomes new end effector of the arm. One important outcome of this assumption is that we can detect the collision of the grasped object with another object or a surface as well. In this case, FingerVision serves as the force sensor of the end effector. To detect the collision, we first save the initial average contact force before the movement. During the movement, we calculate the difference between the initial and current average contact force, and check if the difference has exceeded the maximum deviation value. Once a collision is detection, the arm goes back and can recompute its path based on the information about the obstacle. The collision detection controller is shown in Algorithm 2.
In order to determine the direction of the collision, the initial contact force vector is memorized. Once a collision is

detected, the direction of the collision is determined by subtracting the current contact force vector from the initial contact force vector. We evaluated the controller in two experiments - pushing the arm down and to the side. In both

Algorithm 2: The collision detection controller.

```
initialize threshold f_ε
while True do
    if f_t − f_0 > f_ε then
        x_{t+1} ← X_{des,0}
        break
    end
    else
        x_{t+1} ← X_{des,t+1}
    end
end
```

experiments, we grasped a stick with the gripper and placed the arm such that the stick was pointing towards the table. In the first experiment, we commanded the arm to go down towards a table. Once a collision was detected, we immediately stopped the movement and commanded the arm to go to the initial experiment. In the second experiment, we placed a wooden block on the table which served as an obstacle. The arm was moved against this obstacle and, again, once a collision was detected, the movement was immediately interrupted and the arm was commanded to go to the initial position.
The data collected during the execution of both experiments is shown in Figure 3.4. As can be seen, the arm managed to stop within a minor time frame after the collision. However, such controller had a limitation in which the movement speed had been too slow. FingerVision can process the images with a frequency of up to 16 Hz or 0.0625 seconds. If the robot covers a distance more than 0.001 meters in this time frame, it will not have time to react to the collision event. Thus, the execution speed of the controller has to be limited to 0.016 m/s in order to leave time for reacting to collision events.

Figure 3.3: Experimental setup in recognizing the collision of the stick with external obstacles by pushing the arm down (left) and to the side (right).

Following Grasped and Ungrasped Objects

The object following controller tries to follow an object in two different scenarios - by exploiting the average vector of the contact force and using the current geometrical information about the object.
In the first scenario, the object is grasped by the gripper with the full effort. A human pulls or pushes the object changing the current average vector of the contact force. Therefore, the force vector points exactly in the direction of the pushing or pulling movement. This information is utilized for moving the arm in the same direction. Let $\mathbf{f}_t = \begin{bmatrix} f_t^x & f_t^y & f_t^z \end{bmatrix}^T$ be

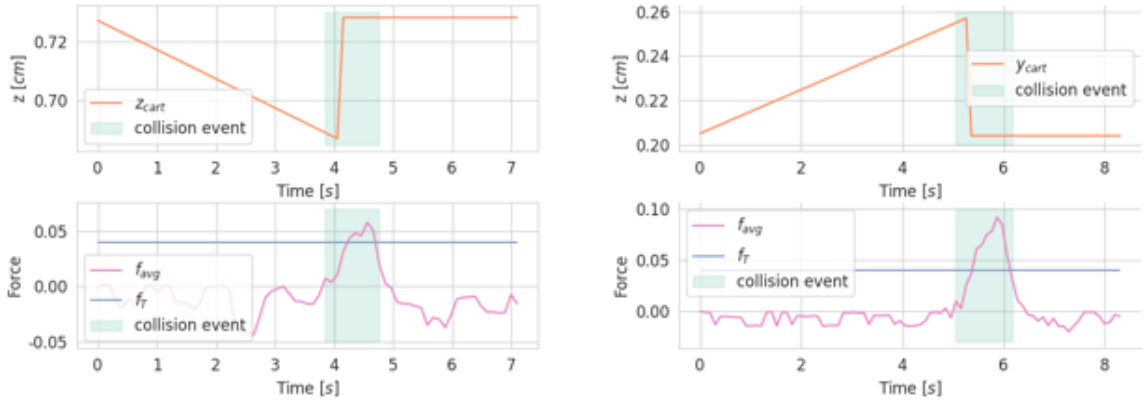


Figure 3.4: Force differences and robot commands during the collision detection experiments. The robot grasped a stick, pushed it vertically towards the table (left) and horizontally against a wooden block (right). The chosen threshold value 0.04 was high enough to tolerate small perturbations during movements of the arm and low enough to detect the collision with an external object. In both collision events, the robot immediately stopped the movement and returned to the initial position. Force values were not calibrated and, therefore, omitted in this Figure.

the current average vector of the contact. Then, in each time step, the Cartesian commands are sent to the arm according to the following rule:

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} x & y & z \end{bmatrix}^T, \\ \mathbf{x}_{t+1} &= \mathbf{x}_t + g(\mathbf{f}_t) \cdot \mathbb{1}(|\mathbf{f}_t| > f_\epsilon), \\ g(\mathbf{f}) &= \frac{(v_{max} - v_{min}) \cdot (\mathbf{f} - f_{min})}{f_{max} - f_{min}}, \end{aligned} \tag{3.1}$$

where f_ϵ denotes the force threshold and v_{max}, v_{min} denote the maximum and the minimum velocity, respectively. By adjusting f_ϵ, we make the controller more or less sensitive to changes of the contact force. v_{max} is used for adjusting the speed of the movement and v_{min} is normally set to 0.
In the second scenario, an object is hold by a human between opened fingers. The human moves the object between the fingers changing its location on the image plane of FingerVision. Hence, M^{00} as well as $\overline{x}$, $\overline{y}$ from Equation (2.5) and Equation (2.6) are exploited for controlling the arm in appropriate directions. The rule for controlling the arm in Cartesian space has therefore the following form:

$$\begin{aligned} x_{t+1} &= x_t + \min\{\overline{x}, x_{max}\} \cdot \mathbb{1}(|\overline{x}| > \overline{x}_\epsilon), \\ y_{t+1} &= y_t + v^y_{t+1}, \\ v^y_{t+1} &= \begin{cases} -0.001 \cdot \mathbb{1}(M^{00}_t > M^{00}_\epsilon) \\ 0.001 \cdot \mathbb{1}(M^{00}_t \leq M^{00}_\epsilon) \end{cases}, \\ z_{t+1} &= z_t + \min\{\overline{y}, z_{max}\} \cdot \mathbb{1}(|\overline{y}| > \overline{y}_\epsilon), \end{aligned} \tag{3.2}$$

where x_{max} and z_{max} stand for the maximum velocity in $x-$ and $y-$directions and $\overline{x}_\epsilon$, $\overline{x}_\epsilon$, M^{00}_ϵ denote the thresholds for centroid and area. Before applying the rules of both methods, the image frame requires to be rotated to the frame of the end effector so that the robot moves in the correct direction.
The main disadvantage of Equation (3.2) is the movement in $y-$direction. First, a good M^{00}_ϵ must be empirically found. Second, even if a good M^{00}_ϵ was determined, the method would not generalize to the object with different shapes. Finally, the whole object needs to be on the image plane. Hence, the first method is preferred over the second. In order to evaluate the object following controller using average force vector from Equation (3.1), we let the gripper grasp a stick as shown in Figure 3.5. We subsequently pulled and pushed the stick in y-direction with different forces exerted manually on the it and sent the commands generated by Equation (3.1). Figure 3.6 shows the resulting $y-$trajectories in Cartesian space which have been produced by Equation (3.1). As can be seen, the Cartesian y-value has been adjusted in conformity with the corresponding y-component of the average force vector at the same time step. The controller regularized the force signal by returning its value to the initial one. Figure 3.8 illustrates how the average force vector behaves during no movement as well as during pulling and pushing.

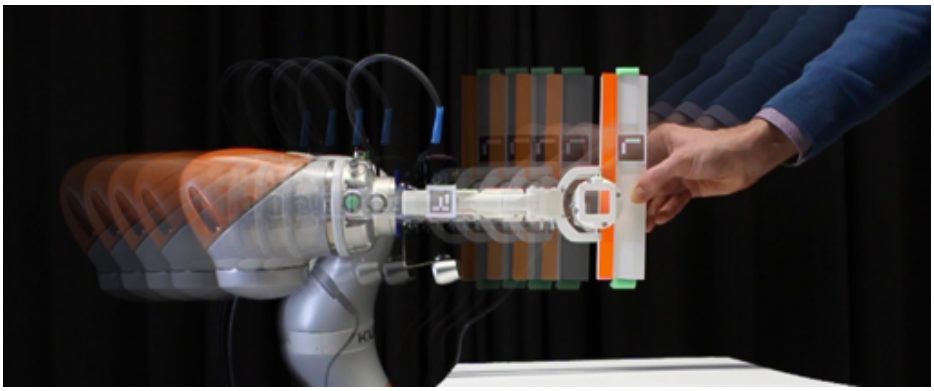


Figure 3.5: Experimental setup for evaluating the controller that commands the arm to follow grasped or ungrasped objects. If the object is grasped, the human varies the speed of the arm by applying different forces. If the object is ungrasped, the speed of the arm is regulated by the speed of the stick.

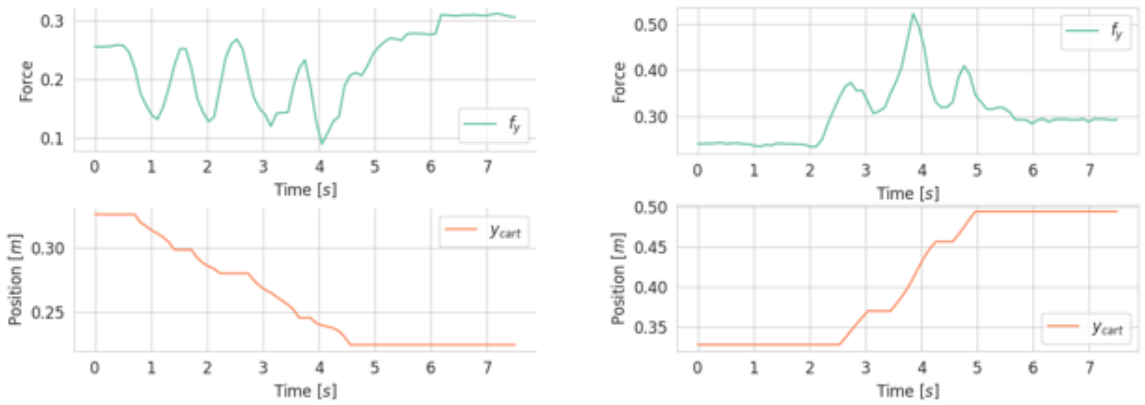


Figure 3.6: Following a stick in y-direction by pulling (left) and pushing (right) it based on the average contact force vector. The experimental setup is shown in Figure 3.5. We applied different forces in order to change the following speed. The controller regularized in some sense the force signal such that the force signal always tended to be near the desired force value, i.e. the force signal at the initial time step. Force values were not calibrated and, therefore, omitted in this Figure.

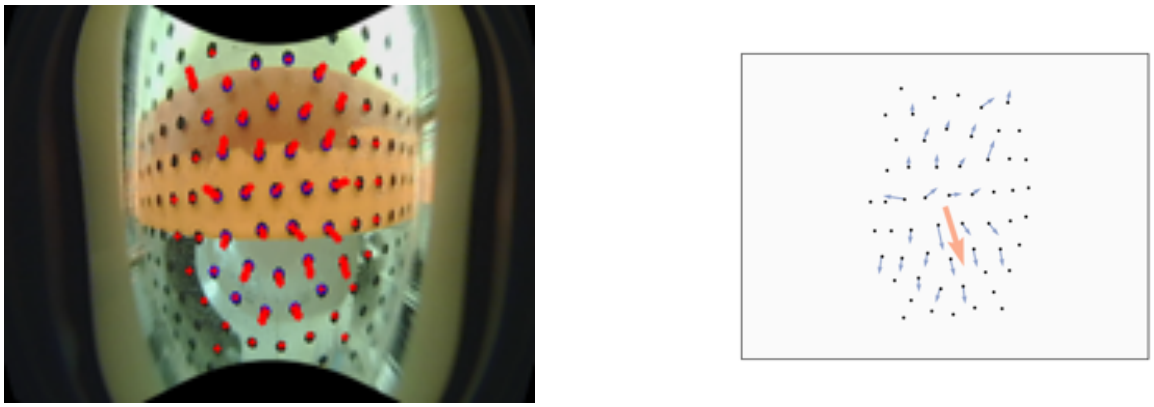
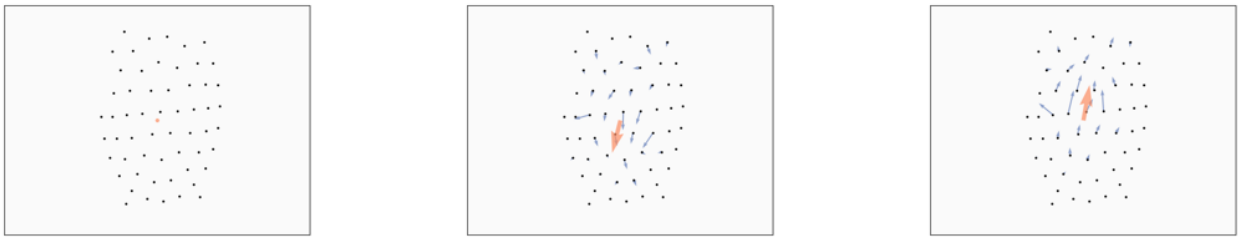


Figure 3.7: A screenshot from FingerVision (left) and the average force value painted with an orange color (right) before following a grasped stick. In both figures, the stick exerts only the contact force and no pulling or pushing force. Note that these figures show the state from different time frames but encode the same information.



(a) The force differences and the average value before executing the task.

(b) Force differences during the pulling task.

(c) Force differences during the pushing task.

Figure 3.8: An illustration of the differences between the force values between original and current states as well as the average force value computed from the differences. These figures correspond to the following a grasped object task.

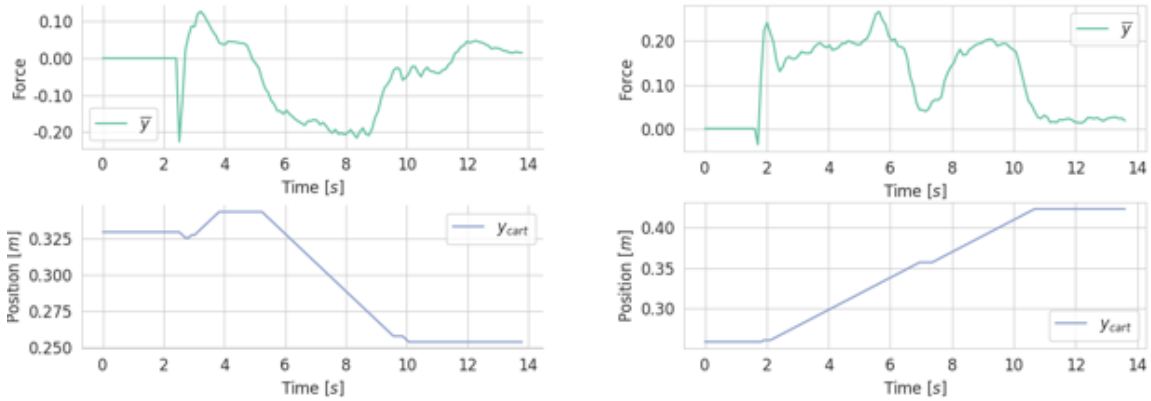


Figure 3.9: Following an ungrasped stick using proximity vision by pulling (left) and pushing (right) the stick in y-direction. In both cases, an $\overline{y}$-component of the centroid was used to compensate for position errors. The rest posture of the stick was defined at $\overline{y} = 0$. The ideal straight line at the beginning of $\overline{y}$-trajectories is caused by the stick being outside of the gripper. Force values were not calibrated and, therefore, omitted in this Figure.

To evaluate the object following controller using proximity vision, we performed a similar experiment as the previous one. Since the stick was ungrasped, we just moved it between the fingers forward and backwards in y-direction relatively to the robot's base frame and controlled the arm according to Equation (3.2). Figure 3.9 shows the trajectories generated by Equation (3.2) in both directions using the information from proximity vision. Although the generated trajectories complied with the location of the object on the image plane of FingerVision, complications arise if we want to control the arm in three dimensions simultaneously. While the center coordinates of the object on the image plane are useful for moving the arm in 2D-space, using the object's area for controlling in 3D-space brings the following complication with the area threshold. One possible way to find this threshold is to extract it empirically. However, if the threshold is found for one configuration of the gripper, it becomes useless for another configuration. Thus, object following using the average force vector is preferable over using the information from proximity vision.

Torque-Driven Rotation of the Arm

We evaluated the quality of the estimation of the average torque in the following experiment. The arm grasped a stick with a metallic piece at the head as shown in Figure 3.10. Next, the arm rotated the stick until such orientation of the stick in which it pointed towards the ground. We defined $\tau_t^y - \tau_{y,\epsilon} < 0$ as a stopping rule for the rotation movement, where τ_t^y stands for the y-component of the average torque at the time step t and $\tau_{y,\epsilon}$ stands for the threshold. This criterion ensures that the arm will not go farther passing the desired orientation of the stick. The threshold value $\tau_{y,min}$ has been empirically found during the experiments. Figure 3.11 shows the values of $\tau_{y,t}$ during the rotation of the arm. The corresponding end orientation of the arm with the stick is illustrated in Figure 3.10. As can be seen, using $\tau_{y,t} < \tau_{y,min}$

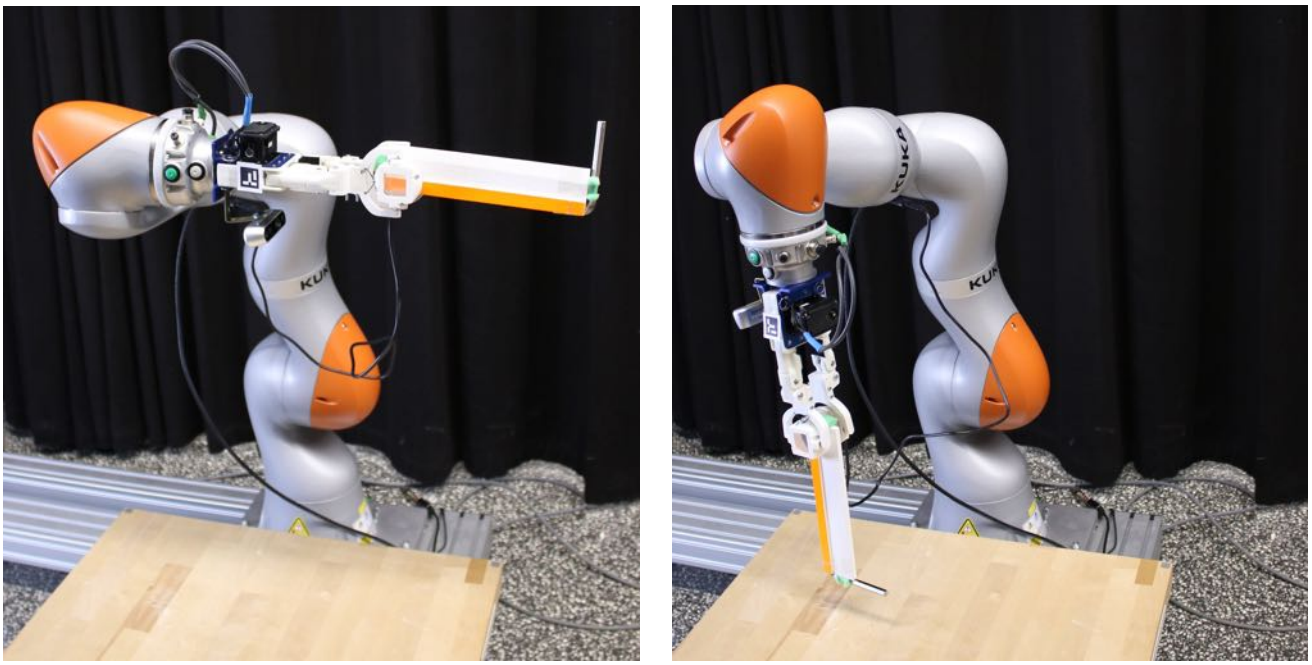


Figure 3.10: Torque-driven rotation of the arm. The arm grasps the stick with a metallic piece at the head (left) and starts rotating until the torque value becomes 0 (right), i.e. the stick exerts no rotational force.

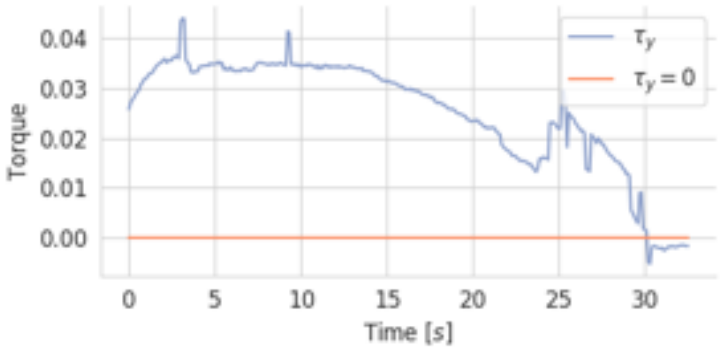


Figure 3.11: Series of $\tau_t^y - \tau_{y,\epsilon}$ during the torque-driven rotation of the arm. The arm stops the movement once $\tau_t^y - \tau_{y,min}$ becomes below 0. Torque values were not calibrated and, therefore, omitted.

3.3 Tactile-Feedback Gripper Control

In this section, we describe the leaky-integrator for gripper control and show how the tactile information can be incorporated into it. Different variations of the tactile-feedback leaky-integrator are introduced and evaluated in the experiments. Moreover, we combine a leaky-integrator with Cartesian controller for finding the minimum grasping effort needed for lifting up an object.

3.3.1 Tactile-Feedback Leaky-Integrator Controller

We employ leaky-integrator controller [11] for incorporating the tactile feedback from FingerVision into the gripper controller. Instead of computing the task space velocity, we compute the task space position directly in order to close the gripper, i.e.

$$x_t = \alpha x_{t-1} - (1-\alpha)L, \tag{3.3}$$

where x_t is the gripper position at time step t, α is the leakage factor and L is either 1 or 0 based depending on the tactile information. In the same way, Equation (3.3) can be used to open the gripper. In this event, one needs to change

the sign of L.
The controller smooths the generated gripper trajectory and manages the intensity of the tactile response through increasing or decreasing the speed of the gripper.

3.3.2 In-Hand Rotation

In this experiment, we aim for rotating a grasped object between the fingers of the gripper by 90 degrees. We describe two different approaches which employ the information about the slippage of the object as well as the average torque which is estimated by Equation (2.2).
In the first approach, the rotation is possible if we allow to some extent the slip of the object. The experimental setup is as follows. We first let the gripper grasp a pen with an additional weight at the head (Figure 3.1(b)) and place the gripper such that the pen is parallel to the ground (Figure 3.12(a)). Then we start to slowly open the gripper and observe the slippage signal. We note the slippage value once the gripper drops the pen. This value will be the slippage threshold for the given object. After finding out the threshold value, we did the following steps. We executed Algorithm 3 for opening the gripper using the following rule for the input signal:

$$L = \begin{cases} 1, \text{if object slips} \\ 0, \text{otherwise} \end{cases}$$

In practice, this rule means that we avoid the slippage which is causing the object's fall. Additionally, a stopping criterion needs to be defined. We stop opening the gripper once $|\theta_0 - \theta_t| > \pi$, where θ_t is the orientation of the object at time step t and is estimated by Equation (2.7). The proposed controller managed to hold the pen in all evaluation attempts.
In the second approach, the in-hand rotation is performed using the torque estimated in Equation (2.2). We evaluated the torque-driven in-hand rotation as follows. A stick with an orange stripe was grasped such that it stayed parallel to the ground as shown in Figure 3.13. The underlying idea behind the experiment is that the y-component of the torque decreases while the stick points more towards the ground and has to be zero if the stick is perpendicular to the ground. More precisely, we tried to rotate the stick by 90 degrees using the average torque by slowly releasing the gripper using the following rule:

$$L = \begin{cases} 1, \tau_t^y - \tau_{y,\epsilon} < 0 \\ 0, \text{otherwise.} \end{cases}$$

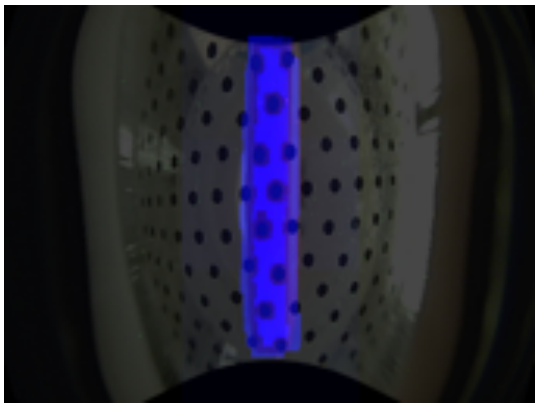
This rule was derived from the rule introduced in Section 3.2.1. In our implementation, we stop the controller once the slippage signal became 0 as we expected that the stick reaches the desired orientation between the fingers. Figure 3.14 shows the average torque values during the execution as well as the orientation of the object extracted using Equation (2.7). In all trials, the stick was not able to achieve the desired orientation even after adjusting $\tau_{y,\epsilon}$. The most successful trial is shown in Figure 3.13.
Based on the conducted experiments, we can say that the first approach is preferred over the second. Although the torque encodes in some sense the orientation of the object between the fingers, it diminishes for orientations that are close to be perpendicular to the ground. Therefore, if the object needs to be orientated exactly perpendicularly to the ground, θ estimated by Equation (2.7) can serve as a good reference.

3.3.3 Handover

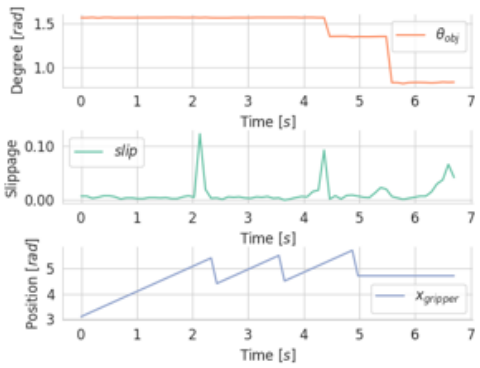
The input signal L from Equation (3.3) can be defined not only as slippage but likewise as a combination of tactile information. The leaky-integrator controller allows for grasping objects that are passed by a human co-worker, i.e. the handover skill. In the handover task, a human places an object between the fingers and the controller closes the fingers until the desired contact force has been reached. In order to learn this skill, the leaky-integrator controller requires the input signal of the following form:

$$L = \begin{cases} 1, \text{if the object slips and touches the sensor} \\ 0, \text{otherwise.} \end{cases} \tag{3.4}$$

By employing this rule, the gripper will be closed only if the stick is between the fingers and touches the sensor. The slip constraint ensures that the gripper starts closing when the object is located between the fingers. The touching constraint eliminates background objects with a similar color as the color of the stick. In this case, The controller for performing the handover task is described in Algorithm 3, where the thresholds are adapted for each object and obtained empirically. $y_m in$ is required for stabilizing the object once the human releases its fingers.

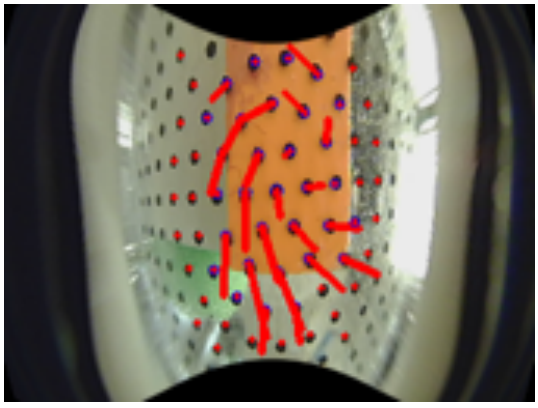


(a) A screenshot from proximity vision showing the location of the pen before rotation.



(b) Orientations and slippages of the pen inside the gripper during in-hand rotation. A leaky-integrator controller adjusts the position of the gripper preventing the pen from falling out of the gripper.

Figure 3.12: In-hand rotation of the pen using the tactile information from proximity vision.



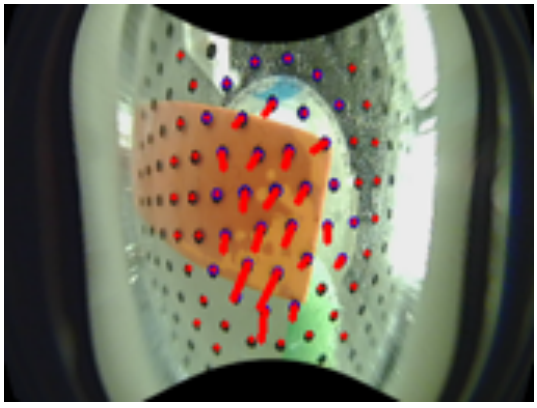


Figure 3.13: Forces before (left) and after (left) rotating the stick inside the gripper by adjusting the grasping effort based on the current torque. The grasping effort was being slowly decreased until the average torque became 0.

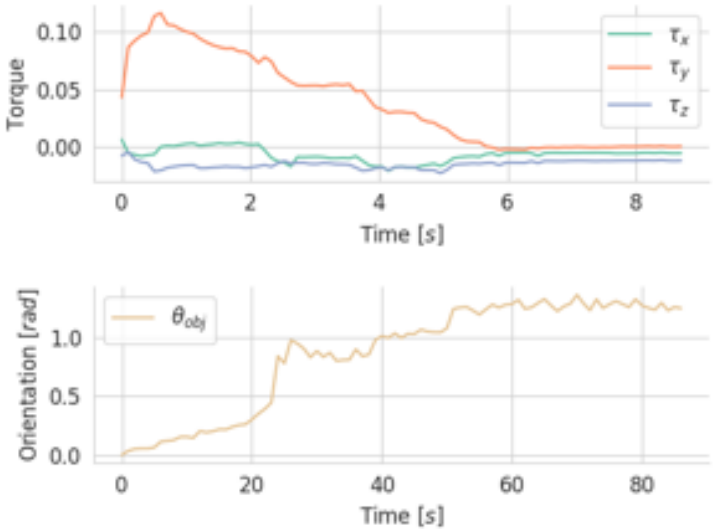


Figure 3.14: Average torque values and orientation obtained from FingerVision during the in-hand rotation task using torques. The torque values were nearly zero between 1.4 and 1.6 radians. Thus, the desired rotation by 1.5 radians could barely be achieved. Torque values were not calibrated and, therefore, omitted in this Figure.

We evaluated Equation (3.4) in the experiment illustrated in Figure 3.15. In all handover attempts, the gripper successfully started closing once the stick touched the sensor until there was no slippage detected. Additionally, the controller provided enough time for adjusting the orientation and the positioning of the stick. Figure 3.16 shows the data for one of trials. The L-signal was activated only when both force and slippage signals are activated which complies with Equation (3.4).
The handover skill can be effortlessly extended by employing the information about the orientation and the area of the object. For instance, the leaky-integrator controller can ensure that the object lies right in the middle of the fingers with some desired orientation.

Algorithm 3: Leaky-integrator controller for handover.

initialize thresholds $slip_{max}$, f_{max}, y_{min}
while $L \neq 0$ & $y_t > y_{min}$ **do**
  if $slip_t > slip_{max}$ & $f_t > f_{max}$ **then**
    $L \leftarrow 1$
  end
  $y_d \leftarrow \alpha y_d - (1-\alpha)L$
  $y_t \leftarrow y_t + y_d$
end

3.3.4 Finding the Minimal Effort Needed for Lifting up an Object

For some tasks, such as identifying the minimal needed for lifting up an object, one needs to control both the robot arm and the gripper in parallel. The controller for finding the minimal effort shown in Algorithm 4 grasps an object, tries to pull it up while ensuring that it will not fall. To prevent the object from falling down, the slippage signal from FingerVision is observed. A too high slippage will probably cause the object fall so that we need to ensure that the slippage does not go beyond the maximum slippage value. If the slippage exceeds the maximum value, the gripper readjusts the grasping force by closing the gripper with a slightly bigger effort. Such a control strategy is useful for example if a grasped object is fragile and can be damaged if the gripper will grasp it with too much force. We evaluated the controller from Algorithm 4 in the following setup. We placed the stick on the table, commanded the arm to go to the grasping position from the top and executed the controller. Figure 3.17 shows the series of object orientations, gripper efforts and the robot commands during execution. There is a clear correlation

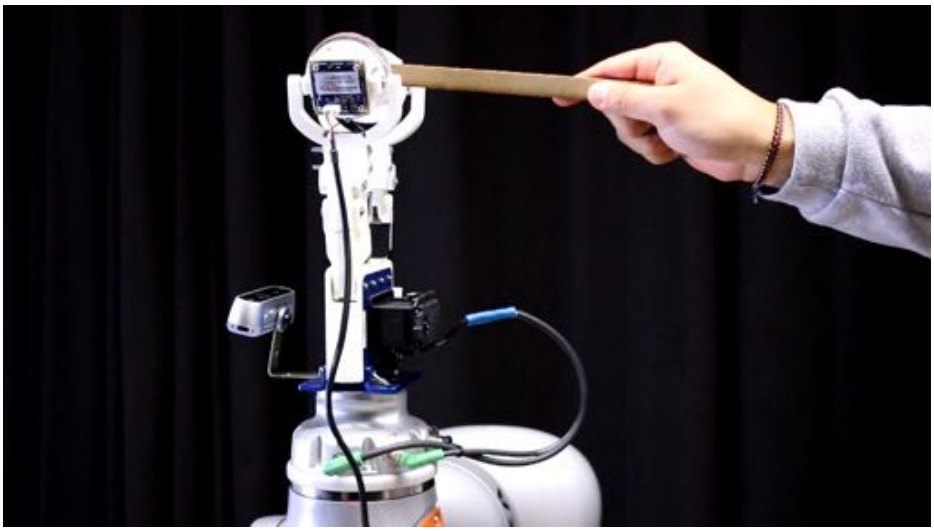


Figure 3.15: During the handover task, a human puts the stick between the fingers. A leaky-integrator controller closes the gripper once the object is between the fingers and touches the contact medium of FingerVision.

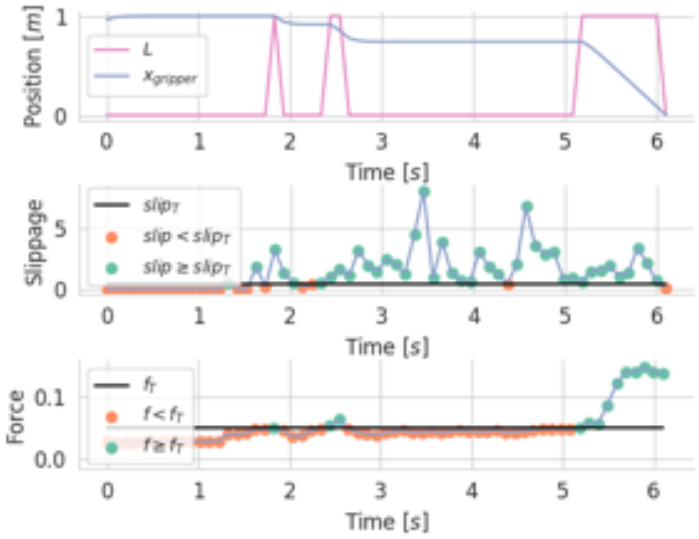


Figure 3.16: Grasping a stick with leaky-integrator controller in the handover task. The values of gripper commands were scaled using min-max normalization for viewing convenience. The L-signal was activated only if the object was slipping and touching the contact medium of the sensor at the same time. Force values were not calibrated and, therefore, omitted in this Figure.

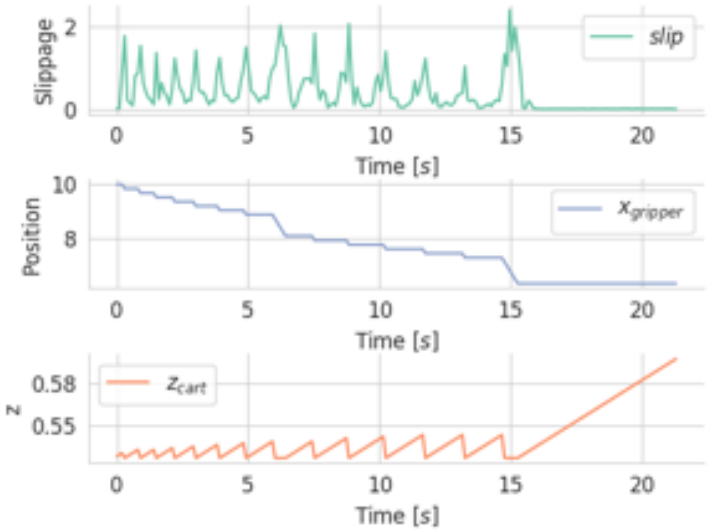


Figure 3.17: Lifting up the stick with minimum effort. From the second 16, no slip was detected anymore and, hence, the stick was pulled up without further gripper adjustments.

between the slippage signal and the gripper effort. The proposed controller proved to be working in this evaluation.

Algorithm 4: Controller for finding the minimal effort needed to pull up an object.

initialize y_{max}
$y_t \leftarrow y_{max}$
while $y_t \neq y_{des}$ **do**
  $z_{t+1}^{cart} \leftarrow z_t^{cart} + 0.001$
  if $slip_t > slip_{max}$ **then**
    $z_{t+1}^{cart} \leftarrow z_0^{cart}$
    $y_{t+1} \leftarrow y_t - 0.1$
  end
end

3.4 Learning Auxiliary Skills by Exploiting Tactile Sensing

The tactile data can be utilized not only in controllers, but also for learning auxiliary tactile skills using supervised learning methods. In this section, we define a regression model that maps the contact forces estimated by FingerVision onto real force values produced pressing movements. Furthermore, we define a classification model which distinguishes between various substances based on their physical properties. We evaluate the proposed models in two experiments - pressing a stick on the weight scale and stirring the substances.

3.4.1 Learning to Map Contact Forces to Real Force Values of Pressing Movements

We propose a ridge regression model which maps the abstract contact forces estimated by FingerVision to the real force values of a pressing movement. Before defining the regression problem, we first need to clarify the details of the experiment.
We assumed that the robot grasped the stick with the maximum effort. If the stick collides with some external object, we can measure the force value that the external object exerts. We used a weight scale as the external object as it provided the gravity force which we would be estimating. We collected force values provided by FingerVision as well as the real

weights measured with the scale during the collision as shown in Figure 3.18(a). Let the $\mathbf{f}_{1:t}$ denote a matrix with forces obtained from FingerVision with t seconds, i.e.

$$\mathbf{f}_{1:t} = \begin{bmatrix} f^x_{1,1:t} & f^y_{1,1:t} & f^z_{1,1:t} \\ f^x_{2,1:t} & f^y_{2,1:t} & f^z_{2,1:t} \\ \vdots & \vdots & \vdots \\ f^x_{n,1:t} & f^y_{n,1:t} & f^z_{n,1:t} \end{bmatrix}, \tag{3.5}$$

where n denotes the total number of the blobs. The feature vector is given by

$$X_i = \left[\text{vec}(\mathbf{f}_{1:t})\right]^T,$$

where vec(A) vec stands for the vectorization operator. Since the amount of features is fairly high, we used the ridge regression for prediction.
Now, the ridge regression problem is defined by the following minimization problem:

$$\min \frac{1}{2}\sum_i (f_{press,i} - X_i\beta)^2 + \frac{1}{2}\lambda \left\|\beta\right\|^2.$$

Here, β denotes the weights which will be learned during the training of the model and λ denotes the regularization parameter. $f_{press,i}$ is the pressing force to be predicted, i.e. $f_{press,i} = X_i\beta$.
For gathering the data, we used a self-made scale to measure the weight value and publish it to a Robot Operating System (ROS) topic. We then grasped the stick with the maximum effort. We did not modified the position of the stick inside the gripper while gathering data, i.e. we collected the data in a single run. In one trial, we performed the following steps:

- define the maximum pressing force
- execute the pressing movement with the defined maximum pressing force
- collect the data at each time step during the pressing movement
- go back to the initial position
- collect the data at each time step while going back to the initial position.

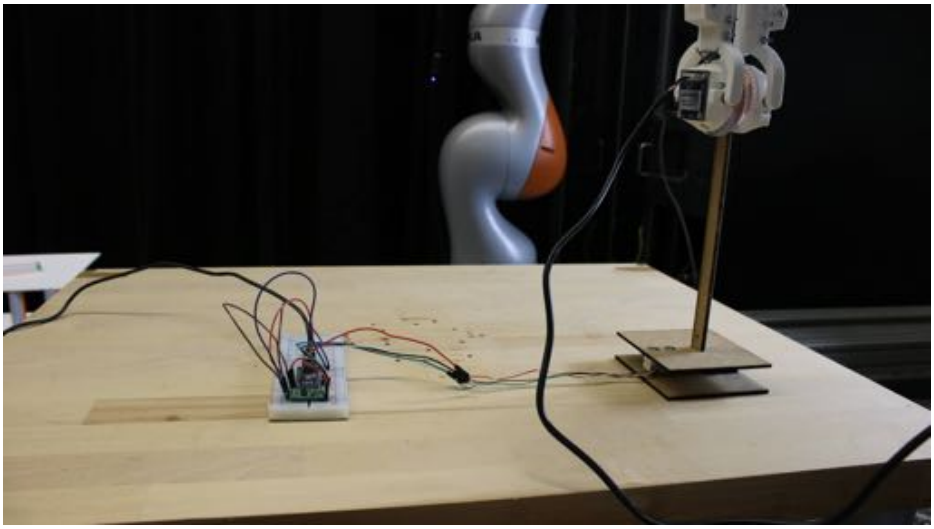
In each trial, we collected the immediate scale values as well as the corresponding movements of the blobs. We collected 20 trials with 20 different maximum pressing forces. After gathering the data, we removed the median and scaled the data according to the inter-quartile range. We scaled the data to make the outliers less influential to the sample mean and variance. Next, we trained a kernel ridge regression model with a radial-basis function kernel, i.e. the feature vector was replaced with radial-basis functions as follows

$$X_i \to \Phi(X_i),$$
$$\Phi(X_i) = \mathrm{e}^{-(rX_i)^2},$$

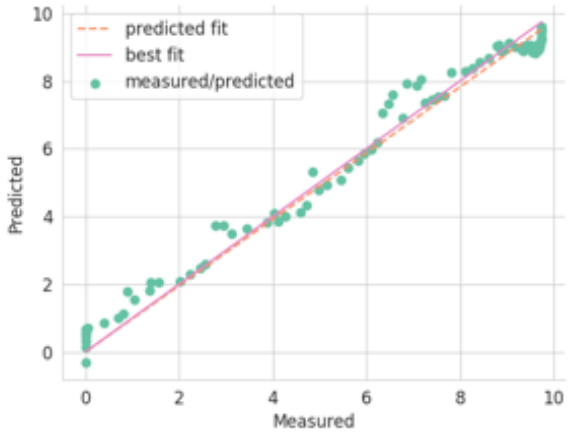
where r is the width of the radial-basis function. The main benefit of using the kernel ridge regression model is that it handles the data set with a big amount of features in a more efficient way. We set $\lambda = 0.02$ and $r = 0.002$.
Finally, we collected the data from one test trial and used the trained model to predict the pressing forces. We used the Mean-Squared Error (MSE) to evaluate the quality of the prediction. MSE measures the average squared differences between the true and predicted values, i.e.:

$$\text{MSE} = \frac{1}{n}\sum_i (X_{true,i} - X_{predicted,i})^2.$$

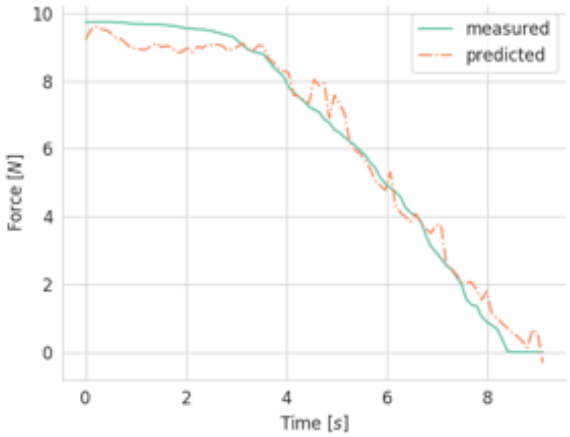
Figure 3.18(b) illustrates the predicted and best regression lines produced by the regression model. Both regression lines are almost aligned and the resulting MSE was 1.232. Figure 3.18(c) shows the measured and predicted pressing forces for the same test trial. As can be seen, the estimation of contact forces can be mapped to real force values of a pressing movement even if a small amount of trials was collected. However, the model was trained only for one particular grasping setting and without benefiting from the other available tactile information. The second experiment in the following subsection will use the additional tactile information for employing in a classification model.



(a) The experimental setup during learning pressing forces. A custom digital weight scale was used to measure the pressing force, and the measurements were sent directly to ROS. In each trial, the scale was pressed by the stick. At each time step of a trial, the force values from FingerVision and the weight measurement were collected and used later on for training a kernel ridge regression model. The regression model was trained to predict the real force values based on the force values provided by FingerVision.



(b) Predicted and best regression curves produced by the learned model. The resulting mean squared error was 1.232.



(c) Measured and predicted force values from the same data set as in Figure 3.18(b). The values were mapped from kilograms to newtons (N) using the gravitational acceleration $g \approx 9.8$.

Figure 3.18: Learning real pressing forces by pressing the weight scale with the stick.

3.4.2 Learning to Distinguish Between Crumbly Substances Through Stirring

[13] learned the viscosity of different liquids to discriminate between them using surface changes captured with a depth camera, e.g. in order to control the pouring velocity. We, instead, aim to develop a classification model that distinguishes between specific types of substances based on the tactile information from FingerVision.
To construct the feature vector, we utilized the same force matrix from Equation (3.5). Additionally, we employed the centroid $\{\overline{x}, \overline{y}\}$, the orientation θ and the area M^{00} of the stick recorded once at the beginning of the stirring movement. Thus, the feature vector for the ith trial is given by

$$X_i = \begin{bmatrix} \text{vec}(\mathbf{f}_{1:t}) & \overline{x}_1 & \overline{y}_1 & \theta_1 & M_1^{00} \end{bmatrix}^T$$

By incorporating $\overline{x}_1, \overline{y}_1$, θ_1 and M_1^{00} into the feature vector, only those blobs will be considered which roughly lie in the same location on the image as the stick.
We aimed to learn 3 types of substances - flour, sugar and peas. To collect the data, we stirred the substances with a stick using the stirring movement and grasping effort. In the initial experiment, we performed 60 trials with 4 different locations of the stick inside the gripper. However, this amount of trials was insufficient to distinguish between sugar and other two types. From Figure 3.20, it can be obtained that the real density value of sugar is relatively close to the real density value of flour. Thus, the classifier often misclassified sugar by recognizing it as flour and vice versa. To assess this issue, we employed another 60 trials with 4 additional grasping settings. Therefore, for each substance, the data set covered 8 different grasping settings.
The collected data was split into training and test parts with 72 and 48 trials, respectively. We then trained a MLP using the implementation provided by the package sklearn. Our neural network has 3 hidden layers with 10 neurons in each layer, and the logistic function as activation function. The output layer consists of 3 neurons which correspond to the types of substances. A schematic design of our neural network is illustrated in Figure 3.21. The weights were optimized using the Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (LBFGS) and the learning rate schedule for weight updates was set to be constant. The evaluation metrics of the trained MLP are provided in Table 3.1. The f_1-score is an accuracy measuring score which is computed using precision and recall as follows:

$$f_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$

where precision and recall are obtained by

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}},$$
$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}.$$

For almost all substances, the classification model was able to assign the correct class label indicating that the model might be overfitted. We, therefore, gathered additional test trials in order to evaluate the classifier on unobserved examples. We did not align the location of the stick in the test trials to those that have been collected for training the classifier. Nevertheless, the classifier was able to predict the type of the substance from each additional test trial disproving the assumption that the classification model was overfitted.
We have shown that the tactile information from FingerVision provides sufficient data which can be exploited to distinguish between different types of substances by stirring them. Even with a relatively small amount of training data, the model was generalized to predict correct types from unobserved trials.

	precision	recall	f_1- score	support
flour	1.00	0.94	0.97	16
sugar	1.00	1.00	1.00	16
peas	0.94	1.00	0.97	16
avg/total	0.98	0.98	0.98	48

Table 3.1: Evaluation of the MLP-classifier used for learning the densities of substances based on the tactile information from FingerVision. The initial data set was split into train (72 trials) and test (48 trials) sets. This table reports the results of employing the trained classification model on the test set.

[1] The densities were obtained from `https://www.aqua-calc.com` (accessed on February 10, 2019)

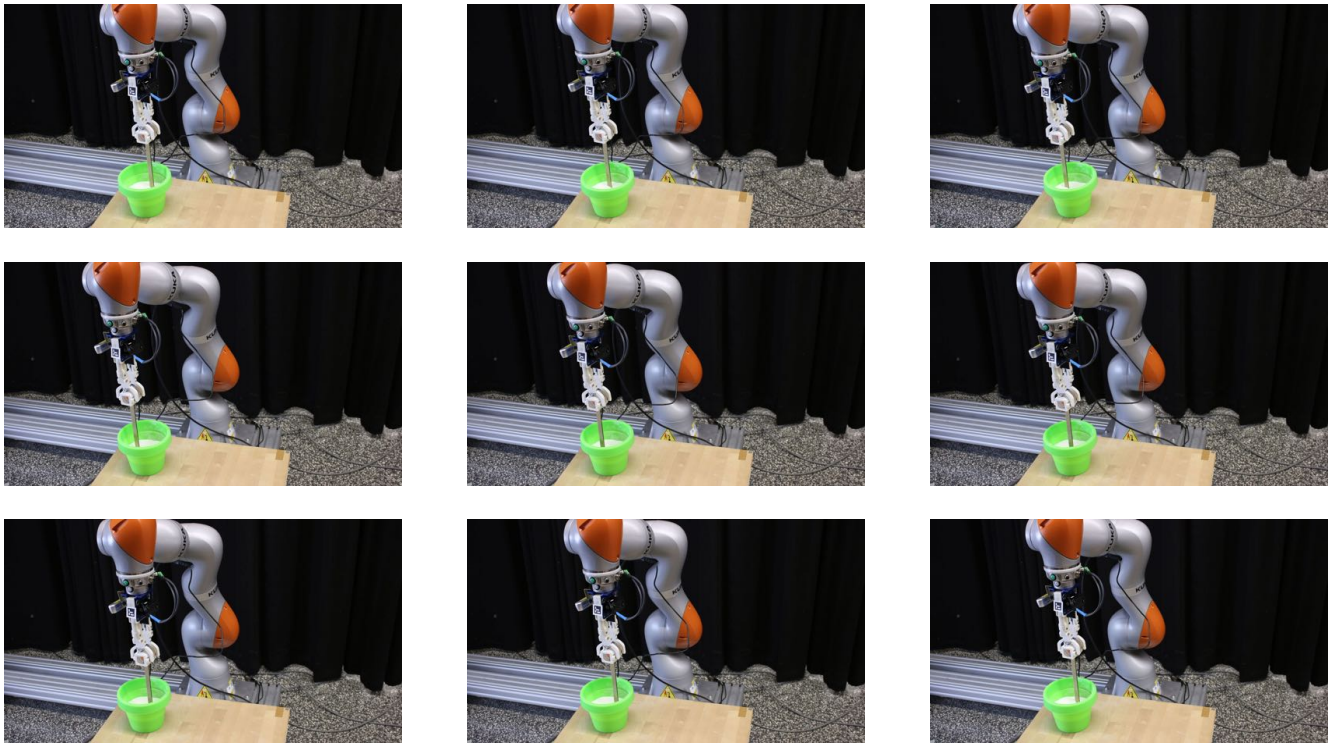


Figure 3.19: A sequence of images (from top left to bottom right) illustrating the stirring movement that was executed for learning to distinguish between substances. The arm describes a circle during one stirring movement. One data point corresponds to one stirring movement.

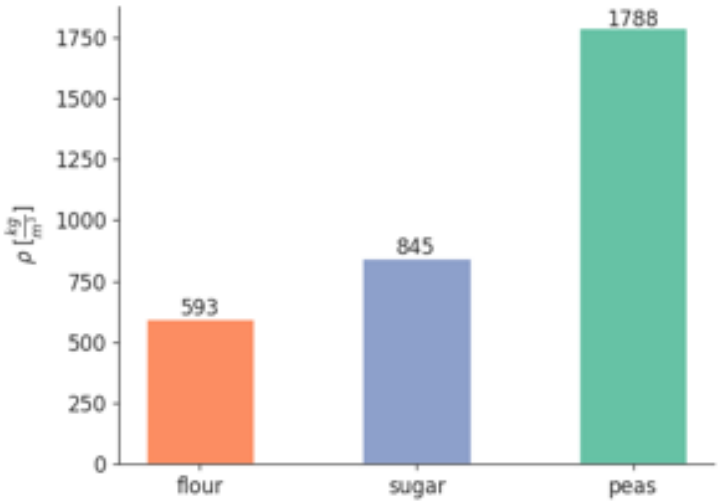


Figure 3.20: Real densities of three different crumbly substances[1] used in the stirring experiment.

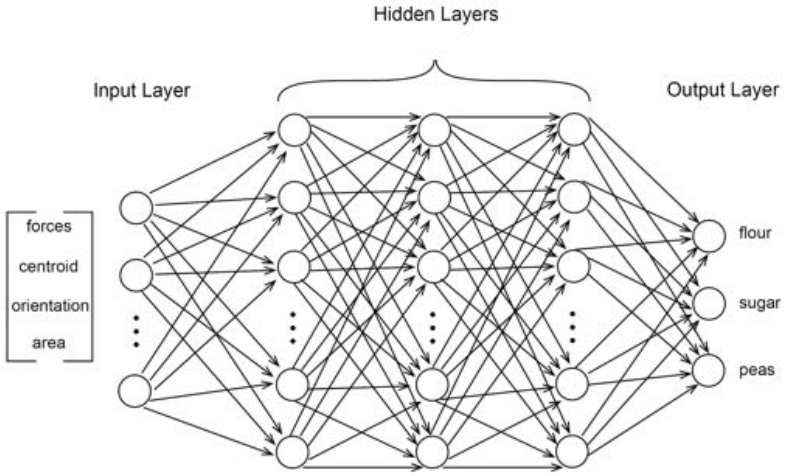


Figure 3.21: The design of MLP used for predicting the type of substance. The input and output layers correspond to the feature vector and class labels, respectively. The feature vector of one trial consists of 18004 features. Each hidden layer consists of 10 neurons.

3.5 Limitations

During the experiments, we encountered several issues with both FingerVision and robot control:

- Processing time of FingerVision - we achieved a maximum frequency of 20 frames per second. This limitation can be potentially solved by utilizing an additional PC only for dealing with the algorithms of FingerVision.
- Light intensity - this limitation was more challenging than the previous. Since we were moving the arm to which FingerVision was attached, the light conditions were changing as well. The issue was partly resolved by adapting the thresholds for different positions of the arm. Another quick fix was to install an external source of the light to compensate for changing light conditions. A more robust solution would be to put a small led light inside FingerVision as it was done for GelSight.
- Calibration of proximity vision - the original method for calibrating proximity vision introduced in [1] worked suboptimal in our environment. There were regularly pieces in the background environment with the same color as the grasped object such that they were also recognized as the grasped object. We solved this problem by introducing a different approach for calibrating proximity vision which was described in Section 2.3.2.
- Sending the commands to the robot via ROS - it turned out that employing the ROS interface of robcom caused delays in the real-time execution which resulted in non-smooth movements of the robot. Thus, we switched to using robcom directly while still benefiting from ROS which has been used for collecting and processing the data from FingerVision and RealSense camera.

While the issue with controlling the robot has been solved without much efforts, the issues with FingerVision are still present. However, the proposed workarounds for solving the problems with various light conditions and calibration of proximity vision significantly reduced their influence on the quality of tactile data. Nevertheless, we could not solve the problem with processing time of FingerVision's algorithms. Furthermore, the internal light source will potentially increase the quality of blob tracking and proximity vision algorithms dramatically.

3.6 Conclusion

We presented tactile-feedback controllers which were fed by the tactile information provided by a vision-based tactile sensor FingerVision. Although being straightforward, the proposed controllers competently coped with the tasks. Moreover, we demonstrated how to employ vision-based tactile sensor for learning auxiliary tactile skills. Our set of auxiliary tactile skills consisted of learning pressing forces based on estimation of contact forces and learning to predict the type of specific substances based the geometrical information about the grasped object along with the estimation of contact forces. Lastly, we discussed the issues arose during using FingerVision as well as provided conducted and possible solutions.

The objective of conducted experiments was to verify the potential of vision-based tactile sensors for using in architectural assembly tasks. FingerVision indeed demonstrated a great potential for various robotic tasks and provided the tactile data of a good quality. Therefore, FingerVision can be employed in the architectural assembly tasks by applying tactile-feedback control strategies similar to those that were proposed in this section.

4 Real-Time Human-Robot Collaboration in an Assembly Task

In common practice, architects precisely define the local and global geometry of a structure for a fixed state of an erected building. We, instead, aim to leverage the advantages of tactile sensing in a human-machine collaborative assembly task such that the positions of all elements do not need to be predefined in advance. Leveraging the advantages of tactile sensing, architects and robots can interactively collaborate by re-positioning vertical elements within a given structure. In this chapter, we first describe the architectural assembly task. Next, we introduce the robot control strategy which solves the assembly task. Finally, we establish a real-time monitoring of a scene on a global (position and orientation of elements) as well as local (applied force, object properties) levels. The scene is streamed into the digital architectural model allowing the architect to re-plan the assembly model based on the current situation.

4.1 Task Description

The environment of the assembly task is shown in Figure 4.1 and Figure 4.2 and consists of the following main elements: a box model, a building element (stick) which should be inserted into the box as well as the weight loads that are put on top of the box. An architect organizes the distribution of the sticks inside the box and forwards the instruction to the robot. The high-level control strategy of the robot performs the steps needed to grasp an available stick, rotate it and insert into the box. Finally, the robot outputs the location of the inserted stick and the architect re-arranges the design based on the received feedback. These steps repeat until the desired distribution of the sticks is achieved.

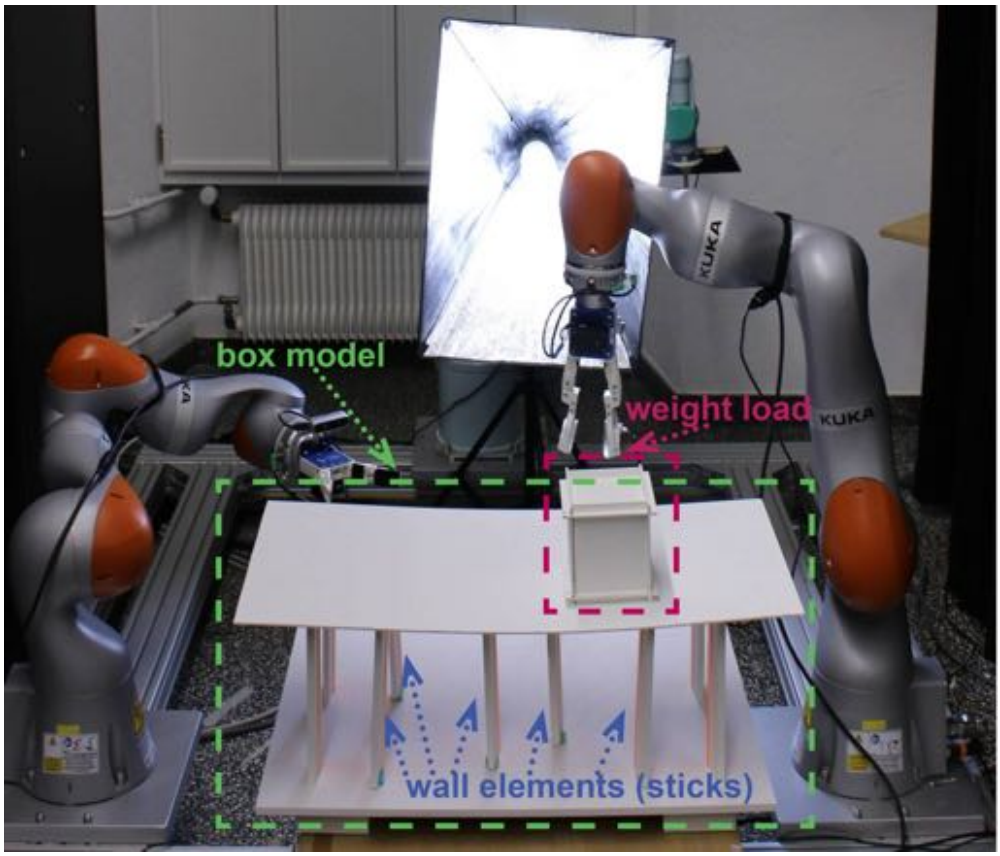


Figure 4.1: The experimental setup of the insertion task. The left arm locates the positions of AruCo markers. The right arm manipulates the building elements (sticks) with aiming at inserting them. A weight load on the roof of the box model serves as the stopping medium during the insertion of a stick. Several sticks have been already inserted into the box model.

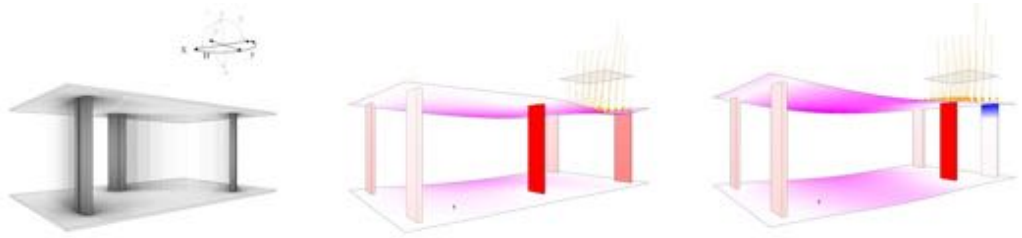


Figure 4.2: The deformed roof without any support (left) and a building element (stick) inserted below the point with the highest load (middle, right).

4.2 High-Level Control Strategy

In order to perform the insertion of building elements (sticks) into the box model, we developed a high-level robot control strategy consisting of the following main steps:

1. Approach the stick
2. Locate the stick between fingers
3. Scan the stick to find the best grasping location
4. Grasp the stick and rotate for insertion
5. Locate the table by detecting the collision
6. Insert the stick until the desired weight load is reached

In the following subsections, we describe each step of this control strategy in details.

4.2.1 Approach the Stick

The rest pose of iiwa is at 0 radians for each joint. Once we start to control the robot in Cartesian space, we encounter the singularity issue [26] when trying to move the arm to far points. To undertake this issue, we prepare the joint configuration of the arm such that no singularities can be met. For iiwa, we define such pose as $\theta = \begin{bmatrix} -0.2 & 0.190 & 0.031 & -1.12 & 0.004 & 1.785 & 0.0 \end{bmatrix}^T$.
To approach the stick with the arm, we simplify the environment by assuming that the stick always rests on the edge and the orange stripe is on the top. For locating the stick in the environment, an AruCo marker [27] is attached at the side of the stick. Approaching the AruCo marker is done as follows. We first capture current position and orientation of the marker using the Realsense camera which is attached to the observing arm. The camera frame is then transformed to the world frame using the tf-package.
After knowing the position and the orientation of the marker, we defined an abstract point p above the stick in the height of 45 cm. Using the MoveIt! package, we computed the desired trajectory path in Cartesian space to p. Since there might have been different resulting trajectories, we first let the MoveIt! package visualize the path in the Three-Dimensional (3D) visualizer. Once a satisfying trajectory is found, the robot executes it. After execution, the arm is placed right above the stick in the distance of about 40 cm.

4.2.2 Locate The Stick Between Fingers

Once we know the rough position of the stick, we need to move the arm such that the stick is perfectly located between the fingers. We defined as perfect position such a position in that the orange stripe is located right in the middle of the FingerVision's view. To solve this problem, we make use of FingerVision's capability of retrieving the object's centroid $\{\overline{x}, \overline{y}\}$ from Equation (2.6). We develop a simple controller (Algorithm 5) which moves the arm down in the direction of the stick until the stick is in the middle of the FingerVision's view. The stick lies in the middle of the view if its area M^{00} is greater than a minimum threshold area and its y-position is about 0. Checking for the area of the stick is needed to deal with the complication in which the y-position jumps to zero once the stick enters the FingerVision's view.

Algorithm 5: The Cartesian controller for locating the stick between the fingers.

init M_{min}^{00};
init $\overline{y}_{max}$;
while $M_t^{00} > M_{min}^{00}$ *and* $\overline{y}_t > y_{max}$ **do**
  $z_{t+1}^{cart} \leftarrow z_t^{cart} - 0.001$;
end

4.2.3 Scan the Stick to Find the Best Grasping Location

We are interested in grasping the stick at the middle to avoid the collision of the arm with the box while inserting. Since the AruCo marker is not perfectly attached to the middle of the stick, the current position of the arm needs to be fitted. We again make use of the information about the object location and area provided by FingerVision. We develop another simple controller (Algorithm 6) which scans the stick and retrieves the middle point of the stick in Cartesian space.

Algorithm 6: The Cartesian controller for finding the best grasping position. The start and end positions of the stick are detected through measuring the area of the stick. The middle point is computed and retrieved.

init M_{min}^{00};
while $M_t^{00} > M_{min}^{00}$ **do**
  $y_{t+1}^{cart} \leftarrow y_t^{cart} + 0.001$;
end
$y_{start} \leftarrow y_t^{cart}$;
while $M_t^{00} < M_{min}^{00}$ **do**
  $y_{t+1}^{cart} \leftarrow y_t^{cart} - 0.001$;
end
$y_{end}^{cart} \leftarrow y_t^{cart}$;
$y_{best}^{cart} \leftarrow y_{start}^{cart} + (y_{end}^{cart} - y_{start}^{cart})/2$;
return y_{best}^{cart}

4.2.4 Grasp the Stick and Rotate for Insertion

Grasping the stick is straightforward as we already performed all preliminary steps to achieve a perfect grasping position. Therefore, we simply close the fingers without employing any tactile information.
Next, the stick needs to be inserted into the box perpendicularly between its roof and floor. To make such insertion possible, the stick first needs to be rotated. The controller for performing the rotation is straightforward. First, the arm goes up until such position which enables the arm to be rotated without colliding the stick with the box. Once the arm reaches this position, we simply rotate it until the stick is ready to be inserted.

4.2.5 Locate the Table by Detecting the Collision

The previous step causes the arm to be in the position above the desired insertion position, i.e. the position where stick slightly touches the bottom surface of the box. To locate the table using the stick and FingerVision, we run the collision detection controller introduced in Section 3.2.1. The controller moves the arm down until a collision with the table is detected and immediately stop it once the collision is detected.

4.2.6 Insert the Stick Until the Desired Weight Load is Reached

We again use the collision detection controller to perform the last insertion step. The arm inserts the stick until the stick roughly reaches the location of the weight load. By adjusting the threshold, we can control how far the stick will be inserted as well as react to different weights of the load. Once the arm reaches the load, it releases the gripper, goes out from the box and returns back to the home position.
During the insertion, the stick can collide with the top surface of the box right before being actually inserted. This problem is assessed by rotating the stick by 45 degrees and inserting it until it is located inside the box. The stick is than rotated back and the arm continues inserting it until the weight load.

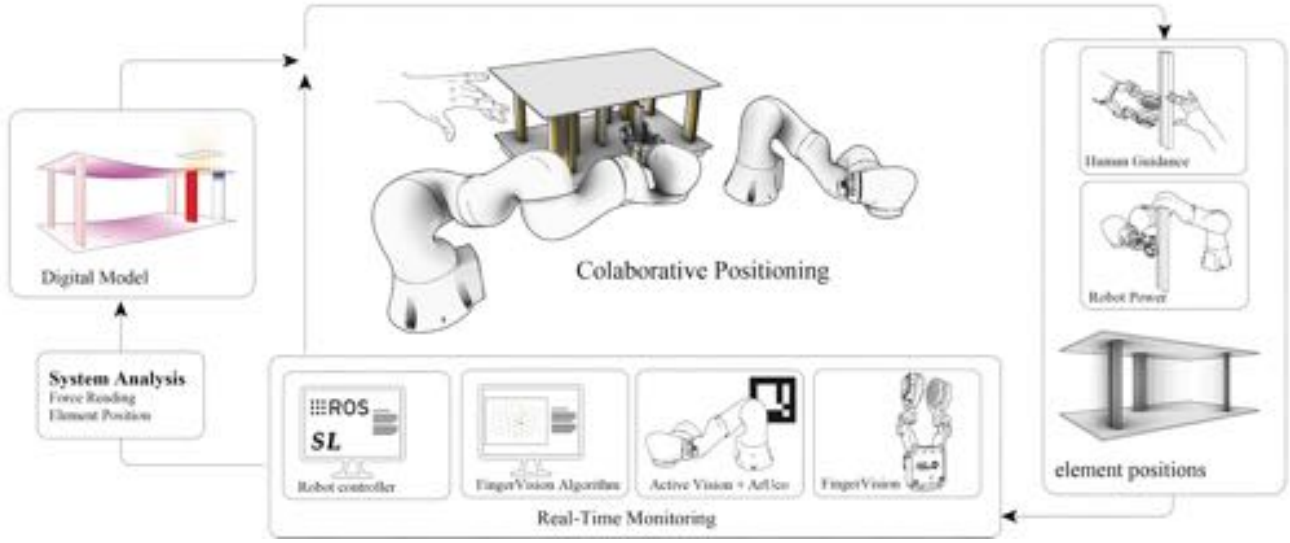


Figure 4.3: Interaction between different units involved in the insertion task.

4.3 Experimental Setup and Evaluation

We tested our high-level control strategy in the following experimental setup. We used sticks with orange stripes on both sides to emulate the building elements. The orange stripes were needed to make the force and proximity vision detection more reliable as was mentioned in Section 2.3.2. AruCo markers were attached to the sticks in order to detect and track their locations and poses using an Intel Realsense camera. The images from the Realsense camera are being continuously published to a Robot Operating System (ROS) topic and an open-source ROS package aruco_ros listens to this topic and performs the detection of markers. We utilized the tf-package to retrieve the current pose of the detected markers.
Assembly commands were planned in Grasshopper which orchestrated the planning; and published to a ROS-topic. We employed rosbridge for publishing and subscribing to the topics from Grasshopper since rosbridge provided a JSON API to ROS functionality for non-ROS programs. The robot controller listened to the topic and executed the insertion once the topic provided a new target location for the stick. A schematic diagram of the communication structure is drawn in Figure 4.3.
A model box was constructed by the DDU. The box was colored with the white color since in our preliminary experiments we found out that darker colors badly affected the force detection and proximity vision once FingerVision was inside the model box. Additionally, one or two loads with different weights were fixed on top of the box and served for preventing the sticks to be inserted into the box infinitely. The load weights could also be exploited for estimating the load distribution on the model's roof.
To send back the position of the inserted stick back to Grasshopper, the position of the stick needed to be scanned by the observing arm. The observing arm goes to the position from which the interior of the box becomes observable. From this position, the arm scanned the recently inserted stick and sent back the information to Grasshopper via rosbridge. The architect then re-designed the plan and sent the desired position of another stick according to the new plan. During all inserting steps, the architect could observe the scene in Grasshopper in the real time. If the architect feels unsatisfied with the position of the inserted stick, he can move it to a desired position inside the box model through executing one of the object following controllers introduced in Section 3.2.1.
Figure 4.4 shows a sequence of the steps during the insertion task. We were able to show that the tactile sensor allowed tight coupling between building components and the robot. Compared to common force-torque sensors, FingerVision generated sufficient tactile data for locating the building elements within the gripper and for inserting them into the box model. Locations of inserted elements were successfully fed back into the architectural digital model in the real-time. Our high-level control strategy enabled close collaboration between the architect and the robot.

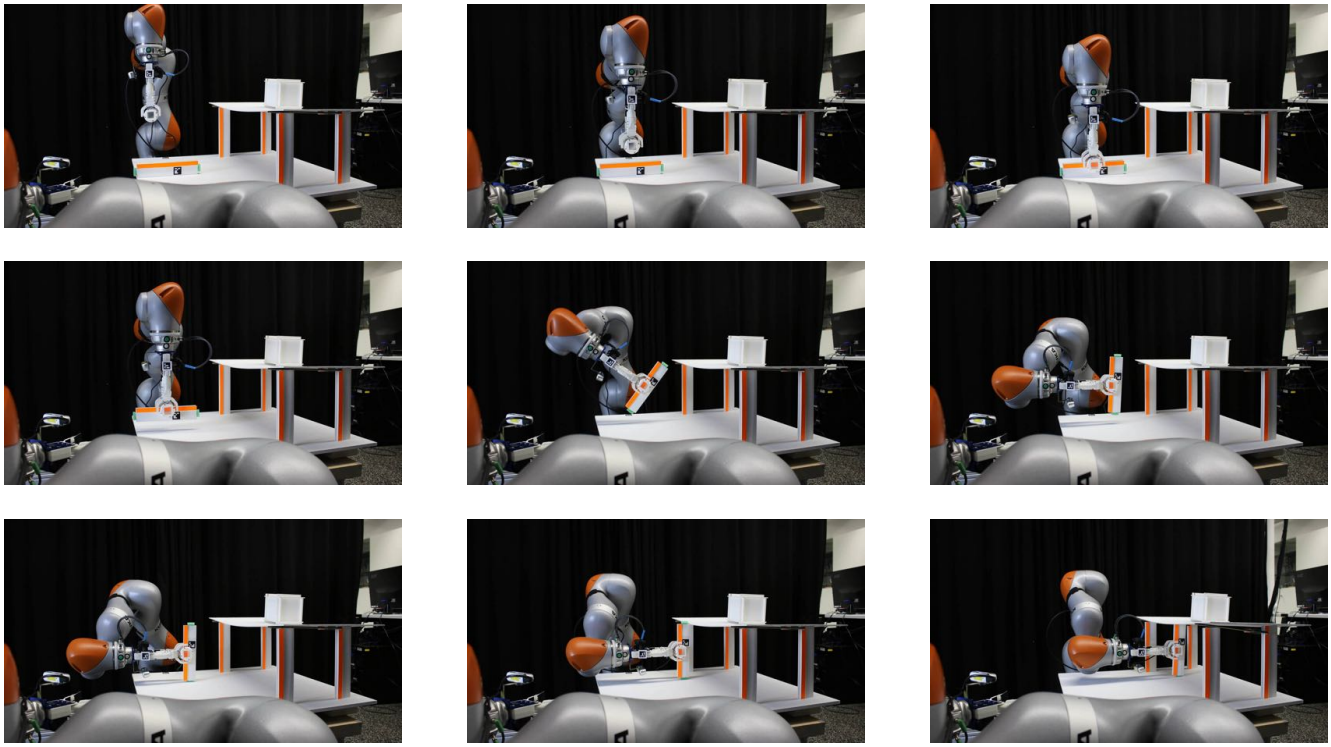


Figure 4.4: A sequence of the steps (from top left to bottom right) performed for inserting the stick inside the box model. The robot first locates the AruCo marker and approaches the arm to the marker. It then scans the stick to obtain the best grasping position, grasps it and rotates for inserting. After rotation, the arm locates the table by trying to collide the stick with it. Finally, the stick is inserted into the desired location but not farther than the load which lies on top of the surface of the model box.

5 Conclusion and Future Work

Robot-augmented editability of assembled structures is something yet to be achieved in architecture. For fine manipulation tasks, a good tactile sensor is required which tend to have high manufacturing costs. Therefore, we aimed at building a vision-based tactile sensor and employing it in tactile-feedback controllers due to low manufacturing costs. For this purpose, we improved the design of a vision-based tactile sensor FingerVision and implemented it using the available materials. FingerVision provided the controllers with the contact force estimation, slippages and geometrical information such as area and orientation of a grasped object. In order to make FingerVision less variant to slight changes of light conditions, we incorporated a Kalman filter into the blob tracking algorithm. We proposed a class of tactile-feedback controllers which employ the tactile data provided by the improved FingerVision. The proposed controllers were able to perform various tasks, such as

- recognizing collisions of grasped objects with external obstacles during arm movements,
- following grasped and ungrasped objects,
- torque-driven rotation of the arm,
- in-hand manipulations,
- handover,
- and finding the minimal effort needed for lifting up an object.

Moreover, we showed the ability of FingerVision to extract auxiliary skills such as determining the real force of a pressing movement and learning to distinguish between various types of substances. For this purpose, regression and classification models were trained that utilized contact forces and contact forces with tactile information, respectively.
Finally, we proposed the usage of a vision-based tactile sensor for offering the possibility for anyone for collaborating with the robot during building assembly structures. For this purpose, we implemented a test system that consisted of movable wall elements, a middle-level tactile-feedback control strategy, and a digital model which orchestrated the assembly process on the high-level. Instead of prescribing positions for the building elements, the positions were roughly defined by the digital model and the elements were inserted through the usage of FingerVision in the direction of the highest load. We proposed a novel method which determines the best grasping pose of the arm for grasping the building elements. The rough estimation of a grasping pose was done with AruCo markers and the refinement of the grasping pose was done by scanning the stick with FingerVision. Locations of placed elements were fed back into the architectural digital model. Moreover, real-time feedback allowed the human to guide the robot to a new position for an element by employing the object following controller.

We demonstrated various possibilities for employing FingerVision in robot controllers. However, several problems were encountered during this thesis. Future work might assess these problems and investigate further possibilities of using FingerVision as follows.
Incorporating a Kalman filter made the blob tracking algorithm more robust to slight changes of the light condition in the environment. However, once the light condition changes significantly, e.g. the arm is situated inside the box model as shown in Figure 4.4, the blob tracking algorithm will fail. In our workaround, we put an external source of light behind the robot which slightly increased the quality of detection algorithms. To assess this problem in a more robust fashion, the design of the sensor has to be potentially enhanced by placing a small led light inside it. This similar to GelSight design will make the blob tracking and proximity vision algorithms more robust, less variant to various light conditions and combine the advantages of both vision-based tactile sensors.
The proposed tactile-feedback controllers provide useful skills while still being simple and robust. Another possibility to employ the tactile feedback into robot control would be to train a VAE-GAN network using demonstrations similarly to [28]. Instead of feeding the images of the environment into the VAE-GAN network, one could generate the robot arm trajectories for in-hand manipulations using the tactile information provided by FingerVision. We verified the feasibility of performing ridge regression based on the contact forces estimated by FingerVision. This approach can be directly extended to ridge regression in the trajectory or task space and, therefore, be used for learning Probabilistic Movement Primitives [29, 30].
We exploited the rich tactile information provided by FingerVision for learning auxiliary tactile skills. This set of auxiliary

skills can be extended to learning more physical properties of substances and liquids or to learning to mimic liquid sensors.
Tactile-feedback leaky-integrator controller was sufficient for grasping lightweight objects. However, the controller often failed to react for slippages of heavy objects causing them to fall. In [31], the slip was predicted using a BioTac sensor. A similar prediction model could be used for preventing heavy objects from falling out of the gripper.
The main issue of the proposed method for using vision-based tactile sensors in the assembly task was the limited ability of the test design model to scale to various sizes. Future research in this area might explore the scaling and mobility of the robot, as well as articulating the elements in order to enable the robot-augmented editability. A judgment system needs to be defined which will be dealing with evaluating the deviation between planned and built models. Another interesting direction would be to examine the loads on deformed surfaces by gently moving a grasped stick on the surface.

Bibliography

[1] A. Yamaguchi and C. G. Atkeson, "Implementing tactile behaviors using fingervision," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 241–248, 2017.

[2] M. Lee and H. Nicholls, "Review article tactile sensing for mechatronics—a state of the art survey," *Mechatronics*, vol. 9, no. 1, pp. 1 – 31, 1999.

[3] L. Zou, C. Ge, Z. J. Wang, E. Cretu, and X. Li, "Novel tactile sensor technology and smart tactile sensing systems: A review," *Sensors*, vol. 17, no. 11, 2017.

[4] T. Zielinska, "Shear force sensor for robots," in *Proceedings of IEEE International Symposium on Industrial Electronics*, vol. 1, pp. 49–53 vol.1, 1996.

[5] K. Kamiyama, H. Kajimoto, N. Kawakami, and S. Tachi, "Evaluation of a vision-based tactile sensor," vol. 2, pp. 1542 – 1547 Vol.2, 2004.

[6] K. Sato, Y. Watanabe, Y. Makino, and T. Maeno, "Vision-based force sensor for touch panel devices using built-in camera," in *The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems*, pp. 852–855, 2012.

[7] W. Yuan, R. Li, M. A. Srinivasan, and E. H. Adelson, "Measurement of shear and slip with a gelsight tactile sensor," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 304–311, 2015.

[8] Y. Chebotar, O. Kroemer, and J. Peters, "Learning robot tactile sensing for object manipulation," in *International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[9] Z. Su, J. Fishel, T. Yamamoto, and G. Loeb, "Use of tactile feedback to control exploratory movements to characterize object compliance," *Frontiers in neurorobotics*, vol. 6, p. 7, 2012.

[10] A. Namiki and M. Ishikawa, "Optimal grasping using visual and tactile feedback," in *1996 IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems (Cat. No.96TH8242)*, pp. 589–596, 1996.

[11] F. Veiga, B. B. Edin, and J. Peters, "In-hand object stabilization by independent finger control," *CoRR*, vol. abs/1806.05031, 2018.

[12] A. Yamaguchi and C. G. Atkeson, "Grasp adaptation control with finger vision: Verification with deformable and fragile objects," 2017.

[13] C. Elbrechter, J. Maycock, R. Haschke, and H. Ritter, "Discriminating liquids using a robotic kitchen assistant," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 703–708, 2015.

[14] F. Gramazio, M. Kohler, and J. Willmann, "Authoring robotic processes," *Architectural Design*, vol. 84, 2014.

[15] A. Menges, "The new cyber-physical making in architecture: Computational construction," *Architectural Design*, vol. 85, 2015.

[16] V. Helm, *In-situ-Fabrikation*. PhD thesis, 2015.

[17] S. Stumm, J. Braumann, M. von Hilchen, and S. Brell-Cokcan, "On-site robotic construction assistance for assembly using a-priori knowledge and human-robot collaboration," vol. 540, pp. 583–592, 2017.

[18] J. Bard, M. Gannon, Z. Jacobson-Weaver, M. Jeffers, B. Smith, and M. Contreras, "Seeing is doing: synthetic tools for robotically augmented fabrication in high-skill domains," 2014.

[19] L. Vasey, E. Baharlou, M. Dörstelmann, V. Koslowski, M. Prado, G. Schieber, A. Menges, and J. Knippers, "Behavioral design and adaptive robotic fabrication of a fiber composite compression shell with pneumatic formwork.," 2015.

[20] G. Brugnaro, E. Baharlou, L. Vasey, and A. Menges, "Robotic softness: An adaptive robotic fabrication process for woven structures," 2016.

[21] G. Brugnaro and S. Hanna, "Adaptive robotic training methods for subtractive manufacturing," 2017.

[22] Y. Wenzhen, S.M. Massachusetts Institute of Technology, "Tactile measurement with a gelsight sensor," Master's thesis, 2015.

[23] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[24] M. Pawlak, *Image analysis by moments: reconstruction and computational aspects*. Oficyna Wydawn. Politechn., 2006.

[25] R. Lioutikov, O. Kroemer, G. Maeda, and J. Peters, "Learning manipulation by sequencing motor primitives with a two-armed robot," vol. 302, pp. 1601–1611, 2016.

[26] P. Donelan, *Kinematic Singularities of Robot Manipulators*. 2010.

[27] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280 – 2292, 2014.

[28] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine, "Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3758–3765, 2018.

[29] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic Movement Primitives," in *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS'13, (USA), pp. 2616–2624, Curran Associates Inc., 2013.

[30] A. Paraschos, E. Rueckert, J. Peters, and G. Neumann, "Probabilistic movement primitives under unknown system dynamics," *Advanced Robotics*, vol. 32, no. 6, pp. 297–310, 2018.

[31] F. Veiga, J. Peters, and T. Hermans, "Grip stabilization of novel objects using slip prediction," no. 4, pp. 531–542, 2018.