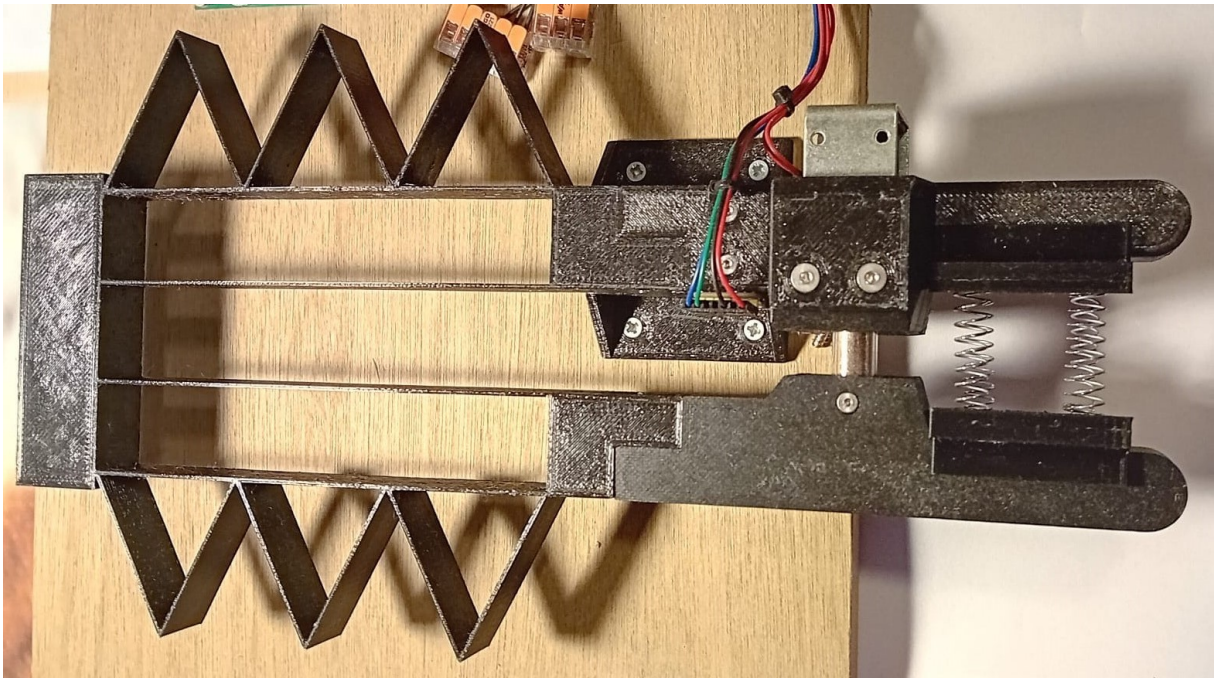
Design and quantitative assessment of a monolithic compliant gripper that can estimate grasped object stiffness without using tactile sensors



P. C. (Corné) Pruijssers
Delft University of Technology
Department of BioMechanical Engineering
Department of Precision and Microsystems Engineering
Delft, The Netherlands

Student information:

Student:	P. C. (Corné) Pruijssers
Student number:	4603893
Institution:	Delft University of Technology
Study:	MSc Mechanical Engineering
Track:	BioMechanical Design

Project information:

Thesis departments:	BioMechanical Engineering Precision and Microsystems Engineering
Project duration:	December, 2023 – Februari, 2025

Thesis information:

To obtain a Msc degree at:	Delft University of Technology
To be publicly defended on:	20-02-2025

Supervisors:

J. L. Herder	Department of Precision and Microsystems Engineering Mechatronic Systems Design
D. M. Syaifoel	Department of Precision and Microsystems Engineering Mechatronic Systems Design

Thesis committee:

J. L. Herder	Department of Precision and Microsystems Engineering Mechatronic Systems Design
D. M. Syaifoel	Department of Precision and Microsystems Engineering Mechatronic Systems Design
G. Smit	Department of BioMechanical Engineering Medical Instruments & Bio-Inspired Technology

Abstract— **Determining the ripeness of fruits is a major point of interest for the fruit industry, since this dictates harvest time, storage conditions and edibility. However, the current state of the art method, which is the penetrometer, performs destructive measurements on the fruits. Recently, ripeness-estimating grippers are being developed to overcome this destructiveness. However, the extra tactile sensors used to achieve this increase complexity, costs and susceptibility to wear. Therefore, in this paper, a gripper is designed that can determine the grasped object's stiffness without using additional force or pressure sensors, but instead estimates force from actuator calibration data, position sensor readings and the control signal. The gripper is designed to be affordable and simple to fabricate to facilitate potential commercial use in the end. To achieve this, a compliant linear guide is designed with on each of the two ends a gripper finger. The mechanism is printed as a single part on an FDM 3D-printer out of PETG, which facilitates the ease of fabrication. The chosen actuator is a solenoid actuator and a Time of Flight (ToF) distance sensor is chosen for position recording. To enable the device to measure object stiffness, calibration of the actuator and the mechanism's intrinsic stiffness is performed. To test the mechanism's performance, calibrated metal compressive springs with a known stiffness are used as test object for the gripper. The stiffness value that was estimated by the gripper is then compared to the known stiffness calibration value. The measured stiffness values differ by 1.0-8.0% from the calibrated spring stiffness values, which on the low error value side compared to general stiffness estimation methods. This implicates that the gripper works satisfactory in terms of stiffness estimation accuracy. Therefore, it can be stated that the main objective of this project is achieved, while also keeping the device low-cost and simple to manufacture.**

I. Introduction

A. Background

Determining the ripeness of fruits is a major point of interest for fruit farmers, distributors, selling markets and consumers worldwide [1]–[14]. This is relevant to determine when fruit needs to be harvested, how it needs to be stored and whether it is edible or not in its current state. Therefore, it is important to be able to determine the ripeness of fruit in a reliable and efficient manner.

The two most commonly measured mechanical fruit properties to determine the fruit's ripeness are stiffness and hardness. Those parameters correlate directly to ripeness [1]–[21]. The current state of the art of determining fruit ripeness is the penetrometer [10] [8]. This device applies pressure on the fruit with a probe until the probe pierces the fruit. The force applied at the moment of piercing is registered and from this, the hardness of the fruit is calculated. This then directly correlates to the ripeness of the fruit.

However, using a penetrometer has a significant drawback [10] [8]: the fruit that was tested gets punctured and damaged in the process, which is generally undesirable for fruit that needs to be sold. Another potential drawback is that the penetrometer measurement can only be applied in the fruit processing factory in a lab setup where a fruit sample needs to be taken out of the sorting and distribution chain. This is not very time-efficient compared to directly measuring the fruit's ripeness within the sorting and distribution chain. In most cases it would be more convenient to have a gripper that could directly grasp a fruit to determine its ripeness. In that case, while in a sorting factory, the gripper system can immediately discriminate between ripe and unripe and sort the incoming fruit supply accordingly. Furthermore, such a gripper could be used to determine the ripeness of a fruit while it is still connected to the tree or bush on which it grows. This could for example facilitate that underripe fruits can be left on the tree or bush for further ripening and only the ripe fruits can be harvested. This possibility would increase the yield and profit of ripe consumer-ready fruits and decrease food waste. In this case, instead of harvesting both ripe and unripe fruits, of which only the ripe ones are ready to be sold and eaten, only the fruit that is at that moment ready for picking will be picked and the rest will be left to ripen further until its ripeness is appropriate before harvesting.

B. Recent advancements

Recently, there have been attempts to solve the mentioned drawbacks of the current state of the art, which is the penetrometer [1]–[21]. To achieve this, grippers have been designed that can grab the fruit to determine its ripeness in-situ with on-board sensors instead of operating in an isolated lab setup. Furthermore, these systems determine the ripeness of the grasped fruit in non-destructive ways to ensure fruit integrity. Either fruit hardness or stiffness are measured to determine the fruit ripeness. Also, some of those gripper setups have potential to be used directly at the fruit harvesting stage to only harvest ripe fruits and leave the underripe fruits to ripen further. With all those recent developments, the most significant drawbacks of the traditional penetrometer tests are overcome.

C. Problem statement

The recent designs to overcome the penetrometer drawbacks can be divided into two categories, namely tactile and non-tactile systems. Regarding the non-tactile category, acoustic or accelerational sensors are used [5], [9], [10], [21]. However, acoustic sensors are considered quite impractical in real-life applications and accelerational sensors can potentially destroy the fruits since they rely on extremely high accelerational vibration forces. Therefore, these options are not considered to be practical solutions and we will consider the tactile category. All designs in the tactile category have two things in common [1]–[4], [6]–[8], [11]–[20]. Firstly, the grippers consist of an assembly of multiple components. Secondly, the grippers use sensors in the fingertips that are in direct contact with the grasped fruit to enable them to sense for example applied force or pressure. However, the multi-component design and assembly, together with the use of a sensor in the fingertip contact points, adds to increased manufacturing complexity, increased wear and more difficult serviceability. This also leads to higher manufacturing and maintenance costs and reduced reliability.

D. Objective

Considering the previously mentioned drawbacks of the tactile systems, it would be beneficial to design a gripper

that consists only of one mechanical part to reduce assembly time, cost and complexity. Also, it would be beneficial to design a gripper that is able to determine the hardness or stiffness of a grasped object without using any added force or pressure sensors at the fingertips or anywhere else in the mechanism.

Since stiffness is more straightforward to measure for a gripper than hardness, since it is merely the applied force on the object divided by its displacement, it was decided to focus on stiffness estimation rather than hardness estimation for a ripeness-estimating gripper.

So to conclude, the objective of this research is stated as follows:

The objective of this research is to design and experimentally evaluate a two-finger gripper of which the mechanism can be fabricated as a single mechanical part, which can estimate grasped object stiffness without the need of tactile sensors on the finger surfaces and to quantitatively assess the accuracy of the stiffness measurements.

E. Structure

In section II, the used methods will be elaborated upon. In section III, the results from the method will be stated. In section IV, the results will be discussed and coupled back to the research objective. In section V, conclusions are drawn. In appendix I, all measured calibration data is shown. In appendix II, all used MATLAB scripts are shown. The used Simulink model, SolidWorks model and SolidWorks simulation results are added as an attachment to this paper.

II. Methods

A. Concept

In Figure 1, the initial concept for the to-be-designed prototype is shown. This initial concept will be elaborated upon in the remainder of this subsection.

To be able to determine the stiffness of the grasped object, one could consider to select an actuator with a direct relation of its applied force as a function of its position and control signal. This relation could be established through proper actuator calibration. To achieve such a proper relation, the actuator must not have backlash or compressibility and have minimal friction. Then, the mechanism can be designed in such a way that it facilitates direct coupling between the actuator force and position on one side and the gripper force and displacement on the other side. In this way, only the actuator position or gripper displacement needs to be measured by an external sensor. This sensor can be placed on a practical location and is not prone to wear and tear since it does not get into contact with anything. This approach thus eliminates the use of an extra force sensor next to a position sensor for object stiffness estimation, which would make the system more complex, more expensive and more prone to wear.

To facilitate direct coupling between the actuator, the grasped object force and the position/displacement, a compliant mechanism is chosen. This eliminates friction and backlash, resulting in a direct coupling, in which only the mechanism's intrinsic stiffness is added to the actuator measurement. If this intrinsic stiffness is then properly determined first, the system can determine the object's stiffness. This is done by firstly extracting the overall stiffness from the overall force-displacement curve that is generated when compressing the object. Then, from that overall stiffness, the mechanism's determined intrinsic stiffness, which is also extracted from its measured force-displacement curve, is subtracted to get the estimated object stiffness. Also, the wish to make the mechanism as a single part would make a compliant mechanism a suitable option for the gripper design.

Another requirement of the design is that the compliant mechanism can be 3D-printed as a single mechanical part on a standard FDM 3D-printer out of plastic. This ensures that the mechanism is simple to fabricate and affordable. Also, the used electrical and other components of the total setup must be affordable and readily available, since this results in a total system that is affordable and of which the required parts are readily available for system assembly and part replacement. This is taken into consideration, since it would be beneficial if the mechanism would in the end be used on a commercial base after its development.

B. Electrical components and setup

1) Actuator selection: For the actuator selection, an actuator type is needed for which the force that it applies can accurately be estimated from knowing its position and the control signal. Therefore, actuators with for example high friction, backlash and compressibility can be eliminated. Considering this, one option draws the attention, namely a linear solenoid actuator. Solenoid actuators consist of a rod that sits in an electromagnetic coil. Applying a voltage on the coil creates a force that moves the rod. Due to the construction of a solenoid, the force as a function of the stroke length and control signal can accurately be determined. Also, solenoids are 1 order of magnitude less expensive than voice coil actuators, which would also satisfy the same criteria as solenoids. Due to this, a solenoid is a proper option for the mechanical gripper and will therefore be used. The specific solenoid that is selected for this project has a maximum stroke length of 18 mm this, together with the stated force-stroke characteristic in the datasheet, [22] is sufficient for this project, since deformations of several millimeters are enough to determine force-displacement characteristics. The solenoid works on a maximum voltage of 6V.

2) Distance sensor selection and control: To measure the gripper position, a distance sensor is needed that is affordable, compact, abundantly available and has a sufficient range and accuracy. For this, a Time-of-Flight (ToF) distance sensor is chosen. These sensor types are generally affordable, costing 2 orders of magnitude less than commercial laser

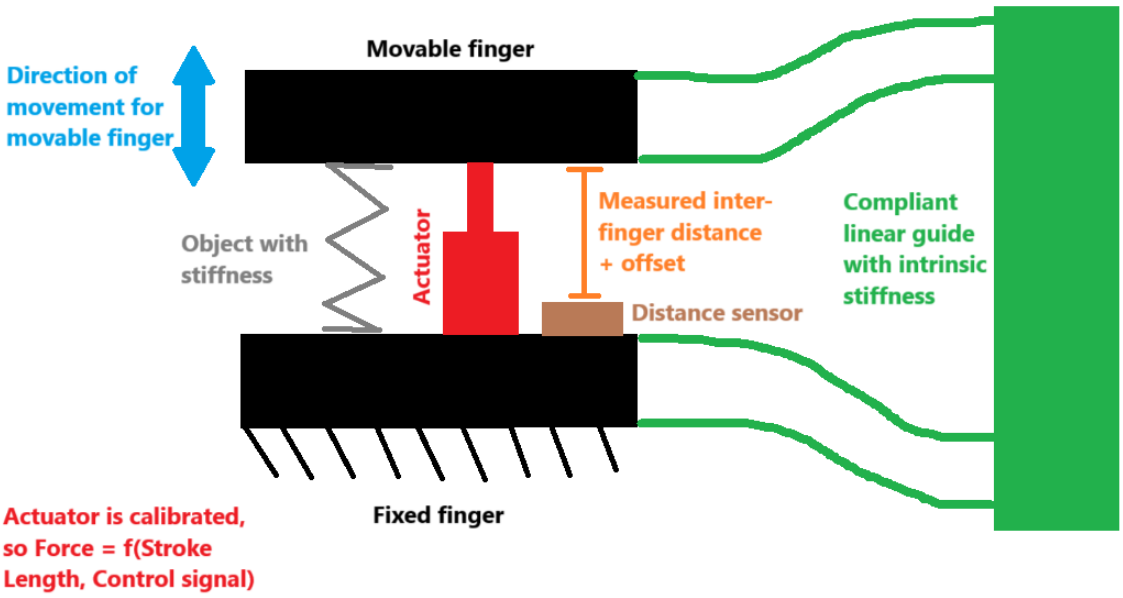


Fig. 1. Concept sketch of the to-be-designed gripper. As can be seen, the concept gripper consists of a fixed and movable finger. The movable finger can translate towards the fixed finger by means of an actuator. In this way, an object with a certain stiffness can be compressed. The movable finger is guided by a compliant linear guide. A distance sensor is added to measure the inter-finger distance. To perform stiffness measurements, the actuator is calibrated to give its applied force as a function of its stroke length and the control signal. In this way, only a position sensor is necessary to determine the object stiffness, eliminating the need for an additional tactile sensor for force estimation.

distance sensors. Also, they generally consist of a PCB that is 1.6mm high and 1-2 cm in dimensions, making them quite compact compared to alternative distance sensors, such a commercial laser distance sensors. Also, they are quite abundantly available and are available in the required range category. One drawback of these sensor types is that their resolution and noise uncertainty are generally quite low and high, respectively. The resolution is generally in whole mm with an noise uncertainty of $\pm$1-2 mm. Nevertheless, considering all the other advantages of such sensor types, a ToF sensor is chosen. The ToF sensor that is specifically chosen for this project is the VL6180 ToF distance sensor [23]. This sensor specifically is quite compact, being a flat rectangular PCB with dimensions of 20.3 x 17.7 mm. The range of the sensor is 5-100mm, which is also sufficient for this project, which requires a range of approximately 30-50mm. The only drawback is that this sensor has quite a low resolution of just 1 mm and quite a high noise uncertainty of $\pm$2 mm, making the sensor not very accurate. For comparison, the laser distance sensors mentioned have a resolution of $50 - 100\mu m$. However, considering the affordability, availability, compactness and sufficient range, this sensor is chosen. This sensor can work with both 3.3V and 5V microcontrollers. The sensor data is filtered with a moving average filter to reduce the noise.

For reading the sensor data, an Arduino script was made using the "Adafruit_VL6180X" library [24]. This library makes it possible to read out the sensor with only a single command instead of an entire I2C communication protocol. The maximum reading frequency of the sensor was determined to be 50Hz, giving a sample period of 0.02s or 20ms.

3) Motor driver selection and control: To control the solenoid, a motor driver was chosen. In this case, a simple DC motor driver that can be controlled by a pulse width modulation (PWM) signal was chosen to control the solenoid. Specifically, a 15A motor driver was chosen, since it was available to the author [25] [26]. One drawback is that this motor driver needs a minimum supply voltage of 6.5V to operate, while the solenoid works on 6V. To solve this, the voltage was set to 7V and by measurement with a multimeter, it was determined that at a PWM value of 225, the average output voltage was 6V, complying to the solenoid specifications. Therefore, PWM values between 0-225 were used instead of the full range of 0-255 (8-bits). This full range is generally supported by standard microcontrollers, like the Arduino and ESP32 series. Furthermore, the motor driver can work with both 3.3V and 5V microcontrollers.

Considering the above, a motor driver like the L298N would likely be an even better option since it only costs one third of the price, while being able to deliver enough current to the solenoid (around 2A) [27]. Also, this driver already works from 5V onwards, which ensures that the motor can be driven with a 6V source instead of a 7V source, enabling the use of the full PWM control range of 0-255, giving an even finer control signal resolution.

4) Microcontroller selection and communication: Since using the VL6180 sensor requires the controller to be programmed via Arduino IDE [28] due to the mentioned library, it was decided to run a dedicated Arduino Uno [29] that communicates over an UART connection to the main controller and on request from the main controller reads out the VL6180 data and sends it back to the main controller.

Also, since it is desired that the system is controlled through Simulink [30], an ESP32 development board [31] was chosen as the main microcontroller. This was firstly done since this board is supported by Simulink through the "Simulink Support Package for Arduino Hardware" addon [32], so it can be programmed and used by it. Secondly it was chosen since it has 3 UART ports, which ensures that one port can be used to communicate with the Arduino Uno to receive the distance sensor readings, while another port can be used to communicate with the connected computer over a serial connection to the running Simulink software for monitoring, tuning and data post-processing. The ESP32 board is also capable of PWM control for values between 0-255. Therefore, it is also directly used to control the motor driver.

Since the ESP32 and Arduino Uno work on 3.3V and 5V, respectively, a logic level converter was used between them for the UART communication to protect the ESP32 against overvoltage [33].

Also, a tactile push button was connected to the ESP32 so that the measurement cycle of the gripper can be triggered externally through a physical push button [34].

The entire electrical setup and its connections are schematically shown in Figure 2 and a picture of the wired electronics can be seen in Figure 3. All the selected components and their properties can be seen in Table I.

C. Compliant gripper design and fabrication

1) Chosen mechanism type, fabrication method and material: For the design of the gripper, a compliant design was chosen. The first reason for this is that it does not create friction and backlash. This is beneficial since the force must be estimated from the actuator by means of calibration and position measurement data instead of using an external force sensor. The second reason is that a compliant design can be 3D-printed as a single piece, which makes the fabrication process simple, reduces the assembly time and makes the system more affordable.

For the fabrication, PETG was chosen as the material, since it is easy to print, affordable and relatively ductile, making it a suitable choice for compliant parts [35]. Specifically, PETG from the 123-3D Jupiter series was used [36].

2) Configuration of the mechanism: Since a linear solenoid was chosen as the actuator, the most simple solution is to add one finger of the gripper to the solenoid perpendicular the solenoid's direction of motion and the other finger to the solenoid rod parallel to the other finger. In this way, there is a direct motion and force coupling between the solenoid and the fingers in which the object will be compressed. Then, a compliant linear guide is designed to support perfectly linear translational motion in the direction of motion of the solenoid so that both fingers can linearly translate towards each other. For the design of a compliant linear guide, it was chosen to use a double compliant parallelogram since this ensures perfectly linear motion between both linear guide ends. The described configuration is shown in Figure 4.

3) Chosen dimensions of the mechanism: The flexures were designed to be as flexible as possible in the direction of motion. Therefore, their thickness was chosen to be the minimum printable feature thickness. Since the available FDM 3D-printer had a 0.6mm nozzle installed and the minimum feature thickness that a FDM 3D-printer can print is 2 times the line width, or nozzle diameter, this ended up to be 1.2mm.

Also, it was chosen to make the flexures as long as possible to create maximum flexibility in the direction of motion. Since the maximum print size of the available printer was 220mm, it was therefore chosen to print the mechanism without the fingers attached to it to ensure that the compliant linear guide could be printed with the maximum possible flexure lengths on the print bed. In the end, while ensuring that the solid regions of the compliant stage remained solid enough to be considered rigid bodies, the remaining flexure length became 150mm. Another advantage of printing the fingers separately, is that during the design and experimentation phase, each finger can be reprinted separately with some modification if that was considered necessary, instead of needing to reprint the entire mechanism for a minor adjustment to one of the fingers.

The height of the mechanism was chosen to be 20mm. The first reason for this was to ensure that the mechanism has proper bending stiffness in the out-of-plane direction of the mechanism, while not becoming too stiff or material-consuming. This height was determined empirically. Also, this thickness ensures that the fingers are high enough to be able to grasp objects between them. Furthermore, this also ensures that the fingers are high enough to facilitate mounting of the ToF distance sensor on the side of one finger, pointing with its direction of sensing perpendicular to the finger on which it is mounted in the direction of motion, towards the side of the other finger. In this way, the compact ToF distance sensor can be integrated into the mechanism. All mentioned feature dimensions are shown in Table II.

4) Solenoid location and integration: Furthermore, it was chosen to place the solenoid in the middle of the height of the planar mechanism to ensure that its force is exactly aligned with the height middle of the the planar mechanism. In this way, the solenoid does not create an extra out-of-plane twisting torque on the mechanism. To facilitate this, a part of one of the fingers was made slightly higher in both directions with an appropriate rectangular hole in it to ensure that the solenoid could be shoved into the finger. This finger was chosen to be the static finger of the mechanism. Also, a hole was made in the other finger so that the solenoid rod could be pushed into it. This finger was chosen to be the moving finger. The reason for the selection of a moving and fixed finger is that this facilitates a perfect linear guide. The

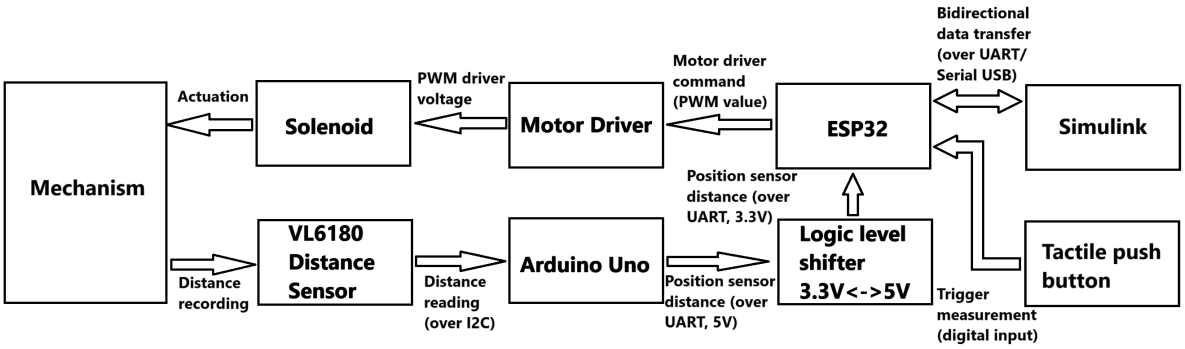


Fig. 2. Schematic of the entire electrical setup and connections. The ESP32 is the main microcontroller that runs the control loop. It communicates with Simulink for data transfer. The VL6180 distance sensor that is mounted on the mechanism gives the readings to the Arduino over I2C, which then sends it to the ESP32 over UART. The ESP32 sends PWM values as motor commands to the motor driver, which controls the solenoid. The solenoid actuates the mechanism.

TABLE I

USED COMPONENTS AND THEIR PROPERTIES.

Component	Voltage (V)	Range (mm)	Noise uncertainty (mm)	Max sample time	Max current (A)	PWM value	number of UART ports
Solenoid [22]	6	0-18	-	-	-	-	-
VL6180 distance sensor [23]	3.3-5	5-100	1-2	50Hz/20ms	-	-	-
15A motor driver [25] [26]	6.5-40	-	-	-	15	0-225@7V	-
L298N motor driver [27]	5-35	-	-	-	2	0-255@6V	-
ESP32 development board [31]	3.3	-	-	-	-	0-255	3
Arduino Uno [29]	5	-	-	-	-	0-255	1
Logic level converter [33]	3.3-5	-	-	-	-	-	-
Tactile push button [34]	-	-	-	-	-	-	-

solenoid finger was chosen to be the static one since this finger contains the most weight. Since the rod is lighter, this finger was chosen to be the movable finger.

5) Torsional stiffness reinforcement: One problem with using long and slender compliant flexures is that they have a low twisting torsional stiffness along the axis in the same direction as their length. therefore, this problem was addressed by adding triangles to the outside of the outside flexure of each parallelogram [37]. These triangles increase the torsional stiffness of each parallelogram without adding much to the bending stiffness. Also, in this location, the triangles are not in the way of other parts of the mechanism in both the resting state as well as in the deformed states along the entire range of motion of the mechanism. Since optimizing this triangle reinforcement is not part of the design problem, this is not performed. Instead, design parameters were chosen that result in sufficient torsional stiffness, without investigating whether there would be more optimal configurations, since this falls outside the scope of this research. In the end, the triangles were designed to be isosceles triangles with the base attached to the flexure 50mm in width and the height perpendicular to the base to also be 50mm in height. This resulted in sufficient torsional stiffness for the mechanism. The selected triangle dimensions can be seen in Table II, next to the other feature dimensions from subsubsection II-C.3.

6) Asymmetric loading problem: One drawback of the mechanism is that it can deform asymmetrically under certain asymmetric load conditions. This could for example happen in a real-life situation where one of the gripper fingers collides against an object that results in an in-plane force perpendicular to the direction of motion. This possible asymmetric deformation is not trivial to solve with the current mechanism. However, in the remainder of this paper, this problem is not further investigated. Furthermore, during the experiments performed in this paper, asymmetric loading has not been present, since the system is tested in a controlled lab setup, where the gripper is mounted onto a static plate and the only objects that touch the gripper are placed in between the gripper fingers. Due to this, collisions of the gripper fingers with objects that result in an in-plane force perpendicular to the direction of motion are not created. In this way, measurements can be performed without asymmetric loading disturbance to analyse the proof of concept.

TABLE II

CHOSEN FEATURE DIMENSIONS.

Feature	Size (mm)
Flexure thickness	1.2
Flexure length	150
Mechanism height	20
Triangle base	50
triangle height	50
Max print length	220

7) SolidWorks finite element method (FEM) analysis of the mechanism's stiffnesses: To estimate the bending stiffness and torsional stiffness of the designed mechanism, FEM analysis was performed in SolidWorks. For this, a new material definition was made for PETG, using material properties from [38]. These material properties are stated in Table III.

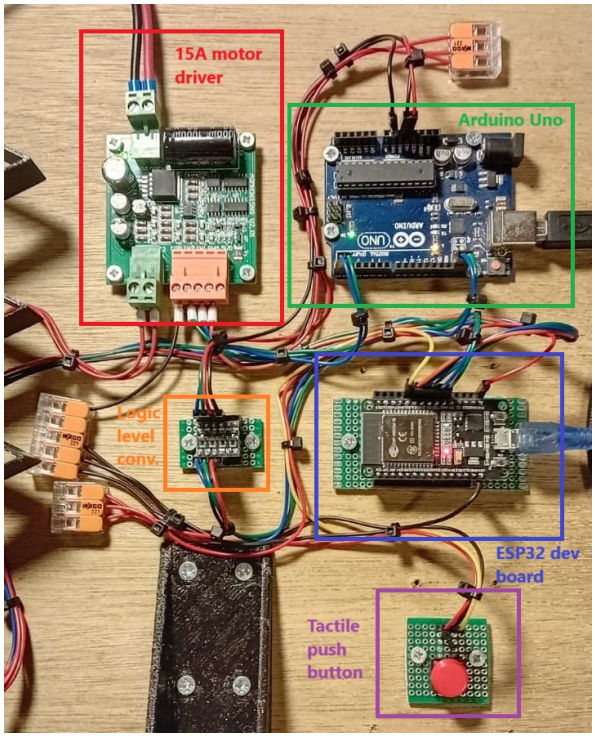


Fig. 3. The entire wired electrical setup. All components visible are labeled. The mechanism and the solenoid and distance sensor that are connected to it are on the left outside of the image. The laptop running Simulink is on the right outside of the image.

For the bending analysis, the large displacement button that pops up during the simulation was pressed, since there are indeed large displacements for the simulated compliant mechanism. In the end, the mechanism's intrinsic bending stiffness is experimentally determined through calibration and then compared to the measurement data obtained by the mechanism's own intrinsic stiffness measurements. Due to this, the SolidWorks FEM data is only used to give a first estimate of the performance and properties of the mechanism and is therefore not required to be extremely accurate. Considering this, the FEM results are not coupled back to the measurement and calibration data and the material properties used in the simulation are not investigated further on their exactness.

TABLE III
USED MATERIAL PROPERTIES FOR 3D-PRINTED OBJECTS FROM PETG FILAMENT [38].

Material property	Value
Modulus of elasticity	2.2 GPa
Poisson's ratio	0.33
Shear modulus	470 MPa
Density	1290 kg/m^3
Tensile strength	53 MPa
Compressive strength	55 MPa
Yield strength	47.9 MPa

8) Assembly: The mechanism's fingers were connected with bolts and nuts to the compliant linear guide and the solenoid and its rod were also connected to the fingers with bolts and nuts. Furthermore, a base was 3D-printed that could be screwed onto a wooden base plate, on which the mechanism and its electrical components could be mounted. Onto this 3D-printed base, the mechanism is also connected with bolts into threaded inserts placed into the 3D-printed base. The entire assembled mechanism can be seen in Figure 4. The entire assembly of the mechanism and its electronics, all mounted onto the wooden base plate, can be seen in Figure 5.

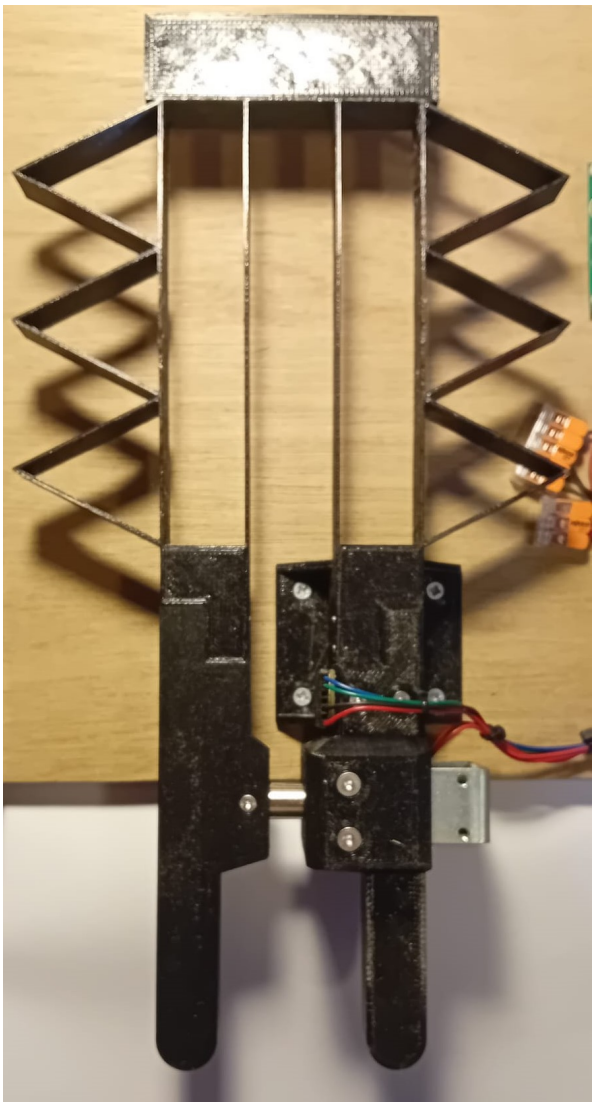


Fig. 4. Image of the gripper. Here, it can be seen that the mechanism consists of one fixed finger and one movable finger that can translate towards the fixed finger. The motion is facilitated by the compliant linear guide. The guide's outer beams are reinforced with triangles to increase the torsional stiffness. The solenoid facilitates the gripping motion and the inter-finger distance is measured by the VL6180 ToF sensor.

D. Simulink control model design

The Simulink control model that was developed can be seen in Figure 6 and Figure 7. The model consists of several

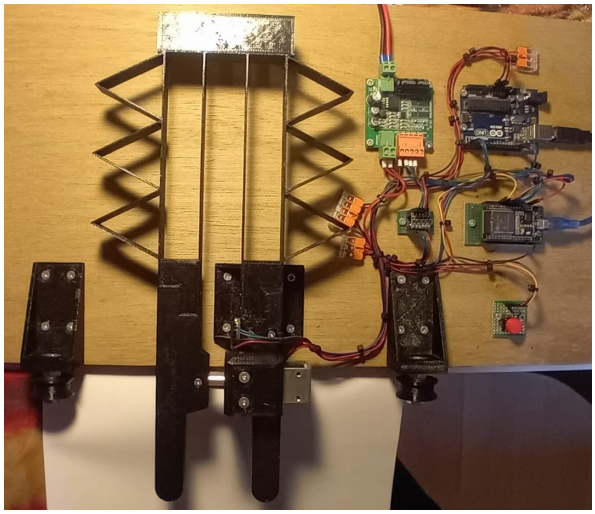


Fig. 5. Entire assembly of the mechanism and its electronics mounted to the wooden base plate.

parts. Part 1 is the main control loop, part 2 is the solenoid control, part 3 is the position sensor reading and processing, part 4 is the triangle wave reference force generator with trigger input, part 5 is the measurement logging and stiffness calculation and part 6 is the triangle wave trigger circuit. Also some "To Workspace" blocks and oscilloscope blocks are added to several parts of the total model. The first one to log the data to the MATLAB [39] workspace for further analysis and the second one to visualize the data live.

Looking at part 1, the control loop, it can be seen that there is a reference force given and from this, the estimated force is subtracted. This gives an error that goes through a PID control block. This block then gives a PWM value as a control signal. This PWM value subsequently goes through a saturation block to ensure that it stays between 0-225 and after that, the resulting value is rounded to an integer value. This value is then used to control the motor. Also, the current sensor distance is read from the sensor in part 3. This data is subsequently modified to state the displacement, which is used for the force-displacement analysis, and current stroke length, which is used for the force estimation. Then, in the control loop, both the current PWM value and the measured stroke length are put into a 2D lookup table block that uses linear 2D interpolation to estimate the exerted actuator force. This force is then again subtracted from the reference force and this error is fed back again to the PID controller block.

For the controller design, a PI controller was chosen. The I was added to the P to compensate for the steady-state error. The D was not used. The first reason for this is that the controller does not need to be very fast, so overshoot is not a problem. The second reason is that the position sensor data is very noisy, even with filtering. Therefore, a D term would only increase instability of the system due to noise amplification. The PI controller was tuned using the Ziegler-Nichols approach [40]. Firstly, K_P was increased until sustained oscillations occurred. From that signal, the oscillation period T_U was determined. Then, these two values were used to calculate initial K_P and K_I values for the controller, based on this formula [40]: $K_P = 0.45K_U$, $K_I = 0.54K_U/T_U$. After that, the K_P and K_I values were experimentally fine-tuned to improve the control loop behaviour. In Table IV, the Ziegler-Nichols PID tuning process is stated step by step.

The function generator of part 4 is a MATLAB function block that is used to create a triangle wave that starts at 0, linearly increases to the amplitude value and then linearly decreases to 0 again. The amplitude, period and the number of periods can all be set manually and the block triggers on an external trigger, originating from part 6. In part 5, a MATLAB function block is used to record the force and displacement values during the triangle wave. After the wave is over, it calculates the measured loading stiffness from that data.

To control and monitor the system, a control panel is designed in Simulink. With this panel, the signals and states of the mechanism can be monitored. Also, a measurement can be triggered from the control panel. Next to that, the parameters of the triangle wave can be set from here and the stiffness data for each measurement cycle is shown on the panel.

TABLE IV
ZIEGLER-NICHOLS PID-TUNING METHOD [40].

Step nr	Step
1	Increase K_P until sustained oscillations occur
2	Determine the oscillation period T_U
3	Calculate $K_P = 0.45K_U$
4	Calculate $K_I = 0.54K_U/T_U$
5	Experimentally fine-tune K_P and K_I

E. Calibration of the gripper system

1) Actuator calibration: To obtain the 2D actuator calibration curve of actuator force as a function of stroke length and PWM value, a calibration setup was created. This setup can be seen in Figure 8. Here, a sturdy wooden frame is created in which a kitchen scale is placed. On top of it, the actuator is mounted facing upwards with a 3D-printed bracket. On the solenoid rod, a downwards facing U-shaped 3D-printed profile is mounted. In this way, when the solenoid is turned on, it pulls the U-profile down against the scale and the scale can then read out the applied force. Business cards with a thickness of 0.75mm are used as spacers between the U-profile and the scale. Possible inaccuracies in this thickness value and compressibility of the cards are neglected in the experiments. In this way, the stroke length of the solenoid can be changed to enable measurement of the solenoid force at different stroke lengths. Before each measurement, it is ensured that the scale is reset to a reading of 0 when the net weight of the objects is modified, for example by adding or removing some business cards.

In this way, the stroke length is increased from 4 to 16mm in increments of 3mm (i.e. 4 business cards). At each stroke length, different PWM values ranging from 0 to 225 in increments of 45 are applied. For each combination of stroke length and PWM values, the displayed weight on the

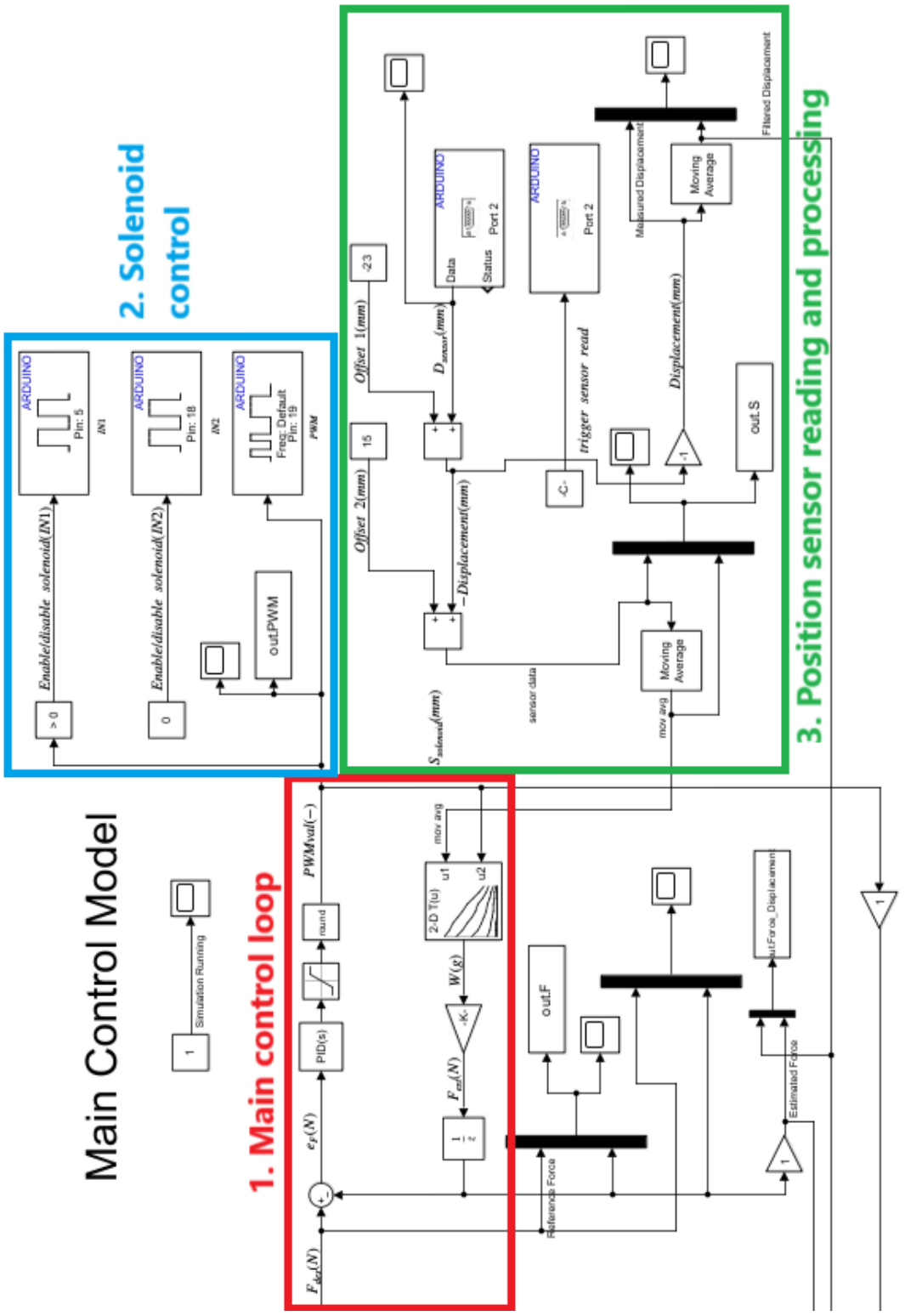


Fig. 6. Part 1 of the Simulink control model developed and used for controlling the gripper. The first 3 parts of the model are shown here. Part 1 is the main control loop, part 2 is the solenoid control, part 3 is the position sensor reading and processing. Also some ”To Workspace” blocks and oscilloscope blocks are added to several parts of the total model. The first one to log the data to the MATLAB workspace for further analysis and the second one to visualize the data live.

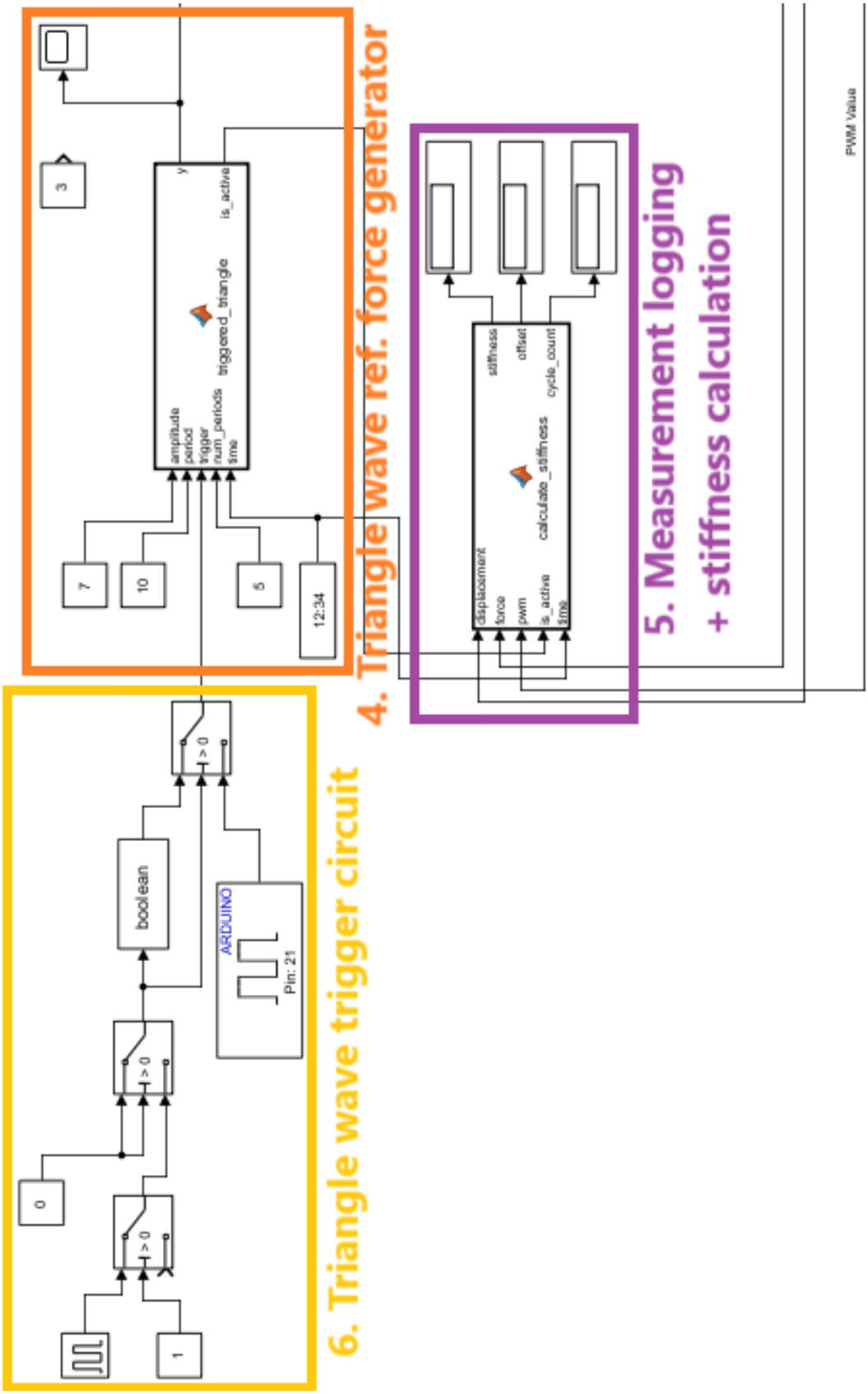


Fig. 7. Part 2 of the Simulink control model developed and used for controlling the gripper. The second 3 parts of the model are shown here. Part 4 is the triangle wave reference force generator with trigger input, part 5 is the measurement logging and stiffness calculation and part 6 is the triangle wave trigger circuit.

scale in grams is read and recorded. This entire sequence is repeated 5 times so that for each combination of stroke length and PWM value, 5 data points are recorded. Then for each combination, the mean and standard deviation of the 5 data points are calculated. With this information, conclusions about the accuracy of the calibration data can be made. In this way, a proper calibration curve of the solenoid is established. This calibration curve can then be used with 2D interpolation to obtain the estimated force for a given stroke length and PWM value.

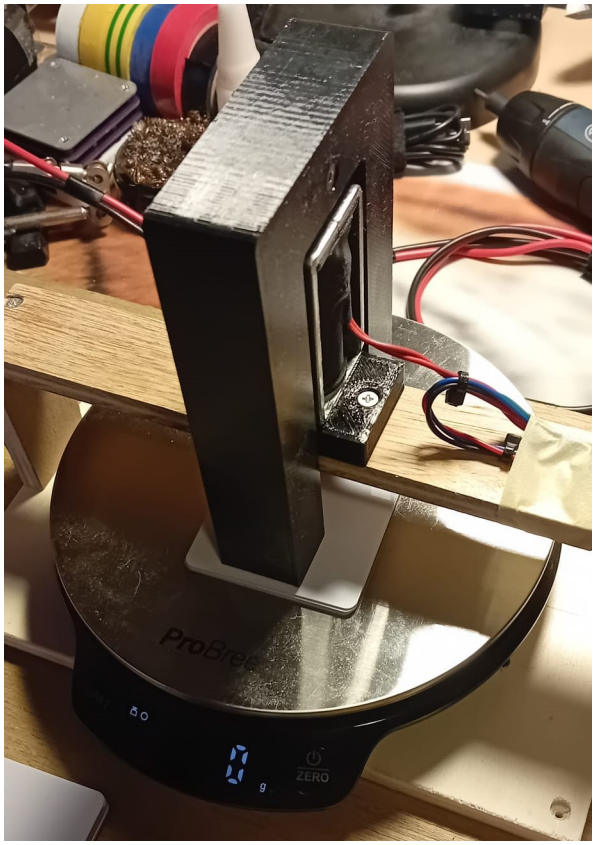


Fig. 8. Solenoid calibration setup. Here, the solenoid is mounted vertically onto a wooden base plate and a downward-facing U-beam is connected to the solenoid's rod. The solenoid can in this way apply force onto a kitchen scale. The stroke length of the solenoid is altered by putting 0.75mm business cards between the U-beam and the kitchen scale. In this way, for different control signal values and different stroke lengths, the created force by the solenoid can be determined by reading the scale output. This results in a 2D calibration curve that gives the solenoid force as a function of its current stroke length and the applied PWM value.

2) Mechanism calibration: To obtain the intrinsic stiffness of the mechanism itself, another calibration process was performed. The setup for this can be seen in Figure 9. Here a pulley is made out of 2 stacked ball bearings in a 3D-printed retainer ring. The bearings are connected to a 3D-printed bracket with a bolt and some nuts. This bracket is screwed to the wooden base at a position that allows the pulley to be properly aligned with the mechanism's fingers. Then, a rope is attached to the movable finger and the other end goes over the pulley. In this way the other end of the rope is horizontally oriented in the direction of motion. This way, the other end of the rope then drops down from the pulley in the direction of gravity. To this downward-facing end, a metal hook is added. On this hook, metal rings can be added as weights.

The weight of the hooks and rings are firstly measured with the same kitchen scale used for the solenoid calibration. Then, for the calibration process, the mechanism is firstly put into its neutral position without the hook pulling on it. This position is recorded. Then, the hook is left to dangle downwards along the rope, exerting its weight as a force on the movable finger along the direction of motion. The current position of the movable finger is then recorded. Next, 2 metal rings at a time are added to the hook and the finger position is recorded for each addition of 2 rings. This continues for up to 5 times 2 rings, at which point the gripper is fully closed. Then, the cycle is performed in the other direction. This means that firstly, the current position with all the weights on it is recorded again and then each time 2 rings are removed and the finger position is recorded again. When all rings are removed, then the hook is also removed and the finger position is recorded again. In the end, the position values are converted to displacement values where the mechanism's neutral position is set to be zero displacement. In this way, the entire trajectory of the finger is recorded in terms of force-displacement and due to the bidirectional approach of loading and unloading the finger, the hysteresis of the mechanism is captured. This entire approach is repeated 5 times. In this way, the mean and standard deviation of each force-displacement point can be calculated again. With this data, something about the accuracy of the calibration data can be stated. In this way, a proper force-displacement calibration curve is established. From this, the linear part for the loading and unloading curve are taken and linear regression is used to create a stiffness line through both linear regions. This results in a loading and unloading stiffness value. These stiffness values can then be compared to stiffness values obtained from measurements of the mechanism itself for its own force-displacement curve in order to state something about the performance of the mechanism.

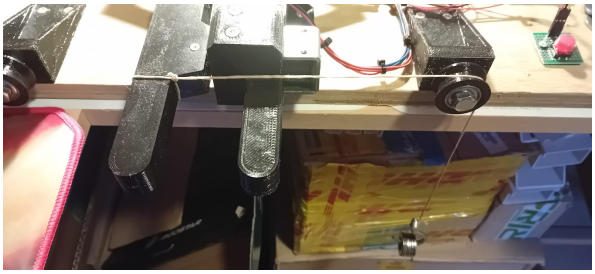


Fig. 9. Mechanism calibration setup. Here, rope is connected to the movable finger and it goes over a pulley so that the other end of the rope faces downward in the direction of gravity. Weights are added to the downward-facing rope end and the displacement of the movable finger is recorded as a function of the applied weight.

F. Synthesis, design and calibration of a compressive spring assembly

1) Spring selection: For the measurement of object stiffness by the mechanism, compressive metal springs were selected. Since metal springs have a completely linear force-displacement curve without hysteresis, this makes the analysis more straightforward. Also, since the gripper is made to compress objects, compressive springs make the most sense to select. Furthermore, these compressive metal springs can be added in parallel to each other to create different stiffness values, which are multiples of the stiffness value of a single spring. The springs that were selected were springs from a general spring assortment box. The springs need to be calibrated to obtain the exact spring stiffness.

2) Spring holder design: To facilitate the containment of the metal compressive springs in a way that they can easily be mounted in between the 2 gripper fingers and to ensure that multiple springs can be placed in parallel at once, a spring holder was designed and 3D-printed. The spring holder consists of 2 identical parts, which are essentially flat plates with a 3x3 grid of rings cut out into it, in which the ends of the selected compressive springs fit. Each plate then also has 2 perpendicularly oriented plates on it. This creates a U-shape that fits around a gripper finger to ensure that the assembly can be connected to the fingers. The 3x3 grid gives variability in the number of and the location of the compressive springs that can be added to it. The thickness of the spring-holding base plate was designed in such a way that the spring assembly is only slightly smaller than the distance between the fingers in the neutral position. This is done to ensure that it fits well in between and that the stiffness measurement of the springs can be performed along nearly the entire range of motion of the gripper. The entire spring assembly can be seen in Figure 10.

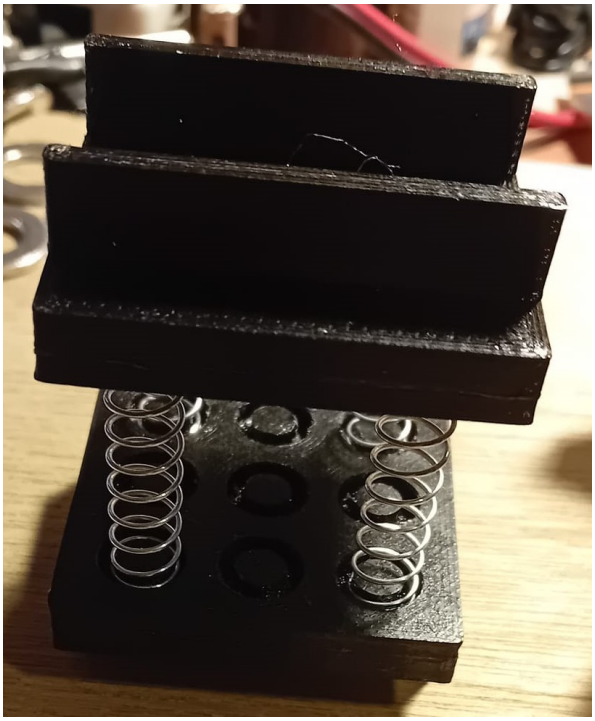


Fig. 10. Spring assembly. Here, 2 spring holders are 3D-printed out of PETG with a 3x3 grid in which the chosen metal springs fit. In this way, the stiffness of multiple parallel springs can be measured. Each holder has a U-shape at the side that does not contain the springs. This U-shape is meant to hook around the gripper fingers to ensure that the spring assembly remains connected to the gripper fingers.

3) Spring stiffness calibration: To calibrate the spring stiffness, the spring holder parts were modified. Each base plate with a 3x3 spring grid now does not have a U-profile on top of it, but instead has L-shaped pillars at each corner. One of the plates is made slightly wider than the other one, so that those L-shaped pillars can fall into each other with some space in between them. In this way, a linear motion guide is created to ensure that during compression, there is no sideways motion of the spring system due to asymmetric loading. The inner pillars are made of such a length that the springs can be compressed far enough before the pillars hit the other holder plate, which would act as a mechanical stop. In this way, the spring stiffness can be determined with large spring displacements to give more accurate results. The spring calibration assembly can be seen in Figure 11.

For the calibration process, 4 parallel springs were used in the holders, one at each corner, to make the system as stable as possible. To calibrate the spring stiffness, the height of the upper plate in the undeformed position was measured. Also, a gym weight, which was weighed to weigh 3001g, was added on top of the upper plate. This can be seen in Figure 12. When the weight is applied on the upper plate, the height of the upper plate was measured again. Since metal springs are perfectly linear, only these 2 force-displacement data points are required to determine the spring stiffness constant. This entire process is then repeated 5 times and for each of the 2 force-position points, the mean and standard deviation are calculated. In this way, something about the calibration accuracy can be stated. From these measurements, the displacement of the springs can be calculated by taking the absolute difference between both mean position points. From this, the spring stiffness constant can be calculated by dividing the force exerted by the weight, converted in N, by the absolute displacement value. This then gives the spring stiffness for 4 parallel springs. By dividing this stiffness value by a factor of 4, the spring stiffness of a single spring is determined.

G. Stiffness measurements with the gripper system

1) Intrinsic stiffness measurement with the gripper: To verify that the system can determine its own stiffness, experiments were performed. A signal generator was made in Simulink that outputs a triangle wave. This wave linearly increases from 0 to 0.75N (at which the gripper is almost entirely closed) in the first 5 seconds and then linearly decreases from 0.75N to 0 in the next 5 seconds. This wave is repeated 5 times. This data is subsequently filtered with a moving average filter with a window size of 45. From this filtered data, for each single wave, the values between

Fig. 11. Spring assembly calibration setup. Here two spring holder parts are 3D-printed with a 3x3 spring grid. Both spring holders have a linear guide at each corner to ensure that the spring assembly does not buckle when a weight is laid on top of the upper spring holder.

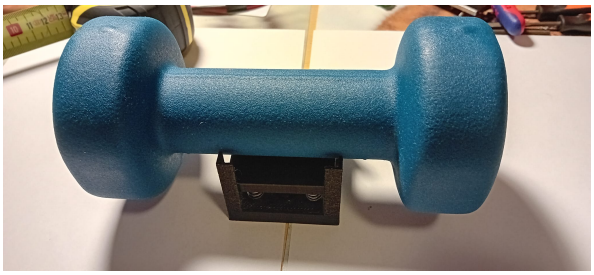


Fig. 12. Spring assembly calibration setup with a calibration weight on top. Here, a gym weight is laid on top of the spring assembly and the spring deflection is recorded. Dividing the weight by the deflection gives the spring stiffness.

1-4 seconds are chosen for the part of interest from the loading curve and between 6.5-10 seconds for the unloading curve from the total wave duration of 10 seconds for each wave. For this selected loading and unloading data, the linear part is selected by placing a displacement bound between 4-8mm. Here, on both the loading and unloading linear region, a linear regression is applied, which gives a loading and unloading stiffness. All these steps are performed 5 times to give 5 different force-displacement curves with 5 loading and unloading stiffnesses. From these 5 stiffnesses, the mean and standard deviation are calculated. In this way, a bar plot with 2σ error bars can be created for the loading and unloading stiffness. These stiffness bars are then compared to the stiffness value obtained from the mechanism calibration in subsubsection II-E.2.

2) Spring assembly stiffness measurement with the gripper: The stiffness of the metal compressive springs is measured with the calibrated gripper. To perform this, the spring holder assembly is placed between the gripper fingers. For the first analysis cycle, 2 springs are placed in the holders. Then 3 springs and lastly 4 springs. The experiments were not performed with 1 spring, since this resulted in an unstable control loop with large oscillations. This situation was not further investigated since it falls outside the scope of this project. To analyse the spring assembly stiffness for each mentioned number of springs, the following was performed for each number of springs: Another triangle wave with the same period as for the intrinsic stiffness analysis in subsubsection II-G.1 was created, but this time, the maximum value was set to 7N. This wave is also repeated 5 times. The data is then again smoothed with a moving-average filter with a window size of 45. Then differentiation of the force signal is applied to determine the loading and unloading region of the hysteresis curve. On the loading and unloaidng regions of the hysteresis curve, force bounds are set between 1-6N for the linear loading and unloading region. Linear regression is applied on the linear loading and unloading curve, resulting in a loading and unloading stiffness. All the steps were repeated 5 times so that in total, 5 loading and unloading stiffnesses were created. From these stiffnesses, the mean and standard deviation were calculated, resulting in a bar plot with 2σ error bars for the loading and unloading stiffness. These stiffness bars are compared to the calibration spring stiffness values obtained from subsubsection II-F.3 for the corresponding number of springs.

III. Results

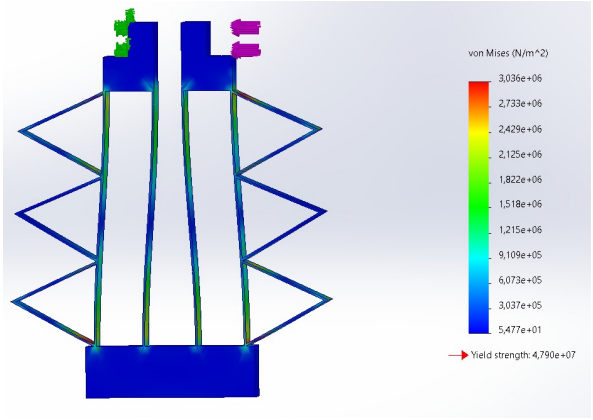
A. SolidWorks Mechanism Simulation Results

1) SolidWorks bending analysis: The SolidWorks mechanism bending analysis stress and deformation results are shown in Figure 13. Here, one finger is fixed and on the other finger, a force of 0.75N towards the fixed finger is applied. It can be seen that the bending displacement and maximum stress are 14mm and 3.0MPa, respectively. From this, it can be observed that the entire range of motion of the mechanism can be reached and that the maximum stress stays below the material's yield strength by over a factor of 10.

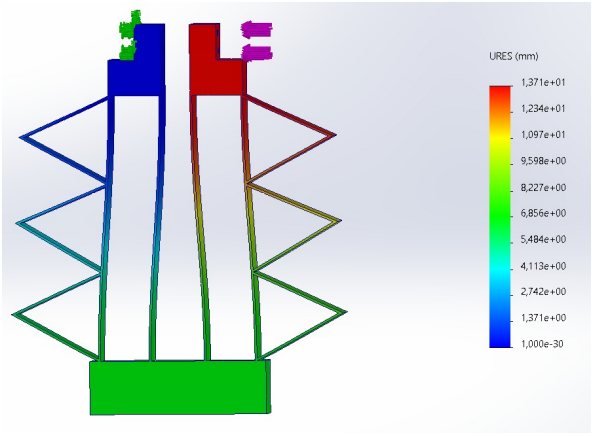
2) SolidWorks torsion analysis: The SolidWorks torsion analysis stress and deformation results are shown in Figure 14. Here, the intermediate body of the linear guide is fixed and on one finger a torsional force couple of 2x 10N is applied with a perpendicular distance between both forces of 15mm. It can be seen that the torsion displacement and maximum stress are 1.2mm and 2.3MPa, respectively. From this, it can be observed that the maximum stress stays below the material's yield strength by over a factor of 10.

B. Calibration of the gripper system - Results

1) Actuator calibration data: The actuator calibration graph is shown in Figure 15. Here, the actuator force in Newtons can be seen as a function of the stroke length and control signal PWM value. It can be seen that a decreasing stroke length and increasing PWM value both lead to a larger force value. Also, 95% confidence (2σ) intervals are shown

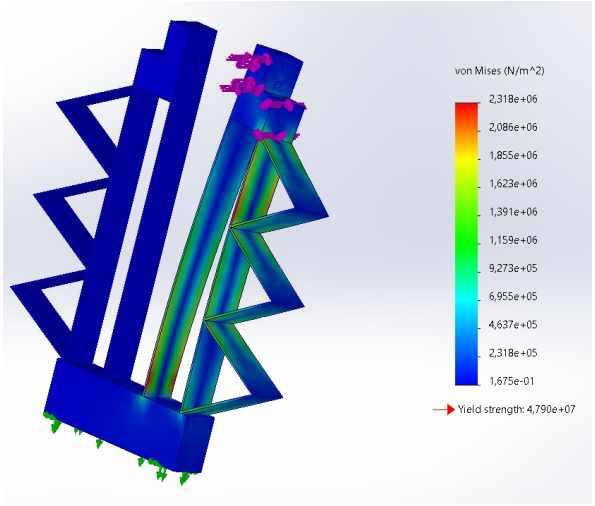


(a) SolidWorks mechanism bending analysis stress profile.

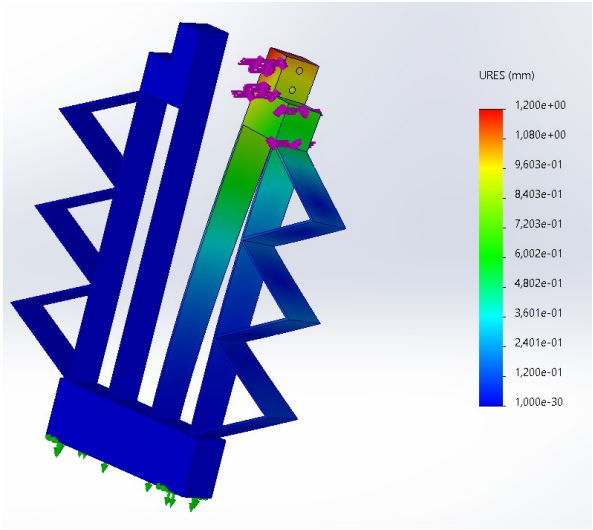


(b) SolidWorks mechanism bending analysis deformation profile.

Fig. 13. SolidWorks mechanism bending analysis stress and deformation profile.



(a) SolidWorks mechanism torsion analysis stress profile.



(b) SolidWorks mechanism torsion analysis deformation profile.

Fig. 14. SolidWorks mechanism torsion analysis stress and deformation profile.

for each data point. The entire original dataset can be seen in appendix I.

2) Mechanism intrinsic stiffness calibration: In Figure 16, the force-displacement curve for loading and unloading can be seen for the calibration process of the mechanism. The data points are given with a 95% confidence bound again. Through the linear parts of the loading and unloading curve, linear regression was used to determine the loading and unloading stiffness of the mechanism. The entire measurement dataset can be seen in appendix I.

3) Spring stiffness calibration: In Figure 17, the force-displacement curve for the calibration of the spring assembly is shown. The experiment was performed with 4 parallel springs, so that is the original measurement data. Since the metal spring has a perfectly linear stiffness, only 2 data points were measured. Each data point is shown with the 95% confidence bound. Through these 2 data points, a line is drawn of which the slope is the calibrated spring stiffness of 4 parallel springs. The values are then divided by 4 to give the spring stiffness of a single spring. The entire measurement dataset can be seen in appendix I.

C. Simulink model tuning - Results

1) Tuning of the PI controller: The PI-controller is tuned according to the Ziegler-Nichols method. Sustained oscillations were observed at $K_p = 25$. The resulting oscillation profile is shown in Figure 18. Here, it can also be seen that the oscillation period is 0.02s. Therefore, $K_u = 25$ and $T_u = 0.02s$. Then, according to [40], for a PI controller, the initial values for K_p and K_i are: $K_p = 0.45K_u = 11.25$ and $K_i = 0.54K_u/T_u = 675$. These values were manually fine-tuned and the resulting values were $K_p = 11$ and $K_i = 600$. The tracking results of this calibrated PI controller can be seen in Figure 19. Here, it can be seen that the reference force is tracked properly at externally enforced position/displacement changes and that for a fixed position, the reference force changes are also followed.

2) Sensor filtering: In Figure 20, the raw distance sensor data can be seen, together with the filtered sensor data. As

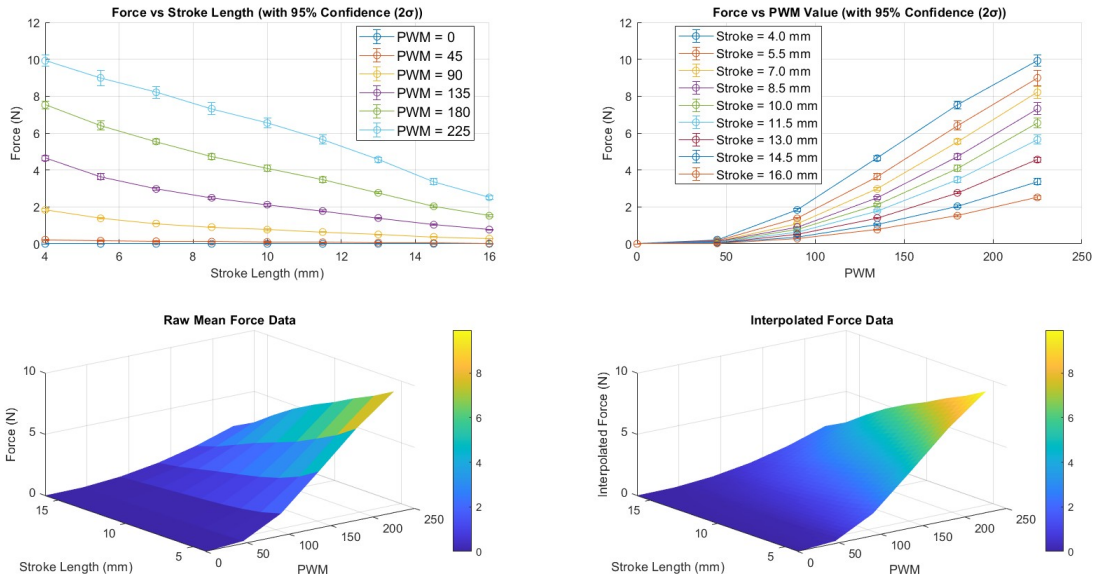


Fig. 15. Solenoid calibration curves. Here, the generated actuator force is shown as a function of its stroke length and the PWM control signal value.

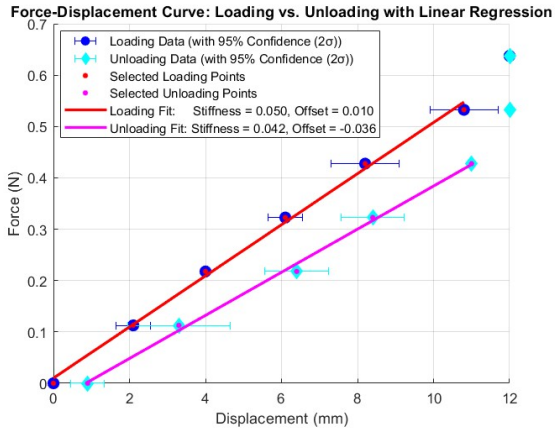


Fig. 16. Intrinsic stiffness calibration data of the mechanism. From this, the loading and unloading stiffness are determined.

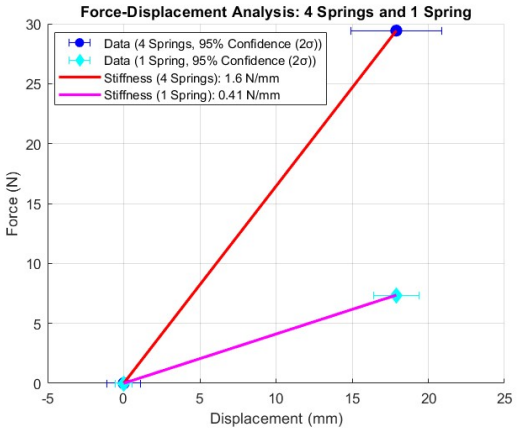


Fig. 17. Calibration data for the spring stiffness of 4 parallel springs. The data is divided by 4 to get the stiffness value for a single spring.

can be seen, the raw sensor data has a substantial amount of discrete noise. The filtered sensor data with a moving average filter with a window size of 10 has a substantially lower noise value than the unfiltered discrete data. This moving average filter with a window size of 10 gives a proper balance between filtering quality and introduced delay (0.2s). Therefore, this filter with this window size is chosen to filter the sensor input data. As can be seen, the force and control signal values also contain less noise due to the filtered position input.

3) Control panel: In Figure 21, the image of the used control panel is shown. At the top of the control panel, the time graphs of displacement, force and PWM value are shown from left to right, respectively. At the bottom left, there is a running button and an active button to show whether the Simulink model is running via ”monitor and tune” and whether a measurement is currently in progress, respectively. Also, there is a start measurement button there to start a measurement sequence. In the bottom center, the amplitude, period and number of periods of the used measurement triangle wave can be modified. In the bottom right, the calculated total loading stiffness, calculated in the same way as for the object stiffness analysis, is shown together with the offset value for the initial location of the linear regression line and the cycle number. This data is all given for the last performed measurement cycle.

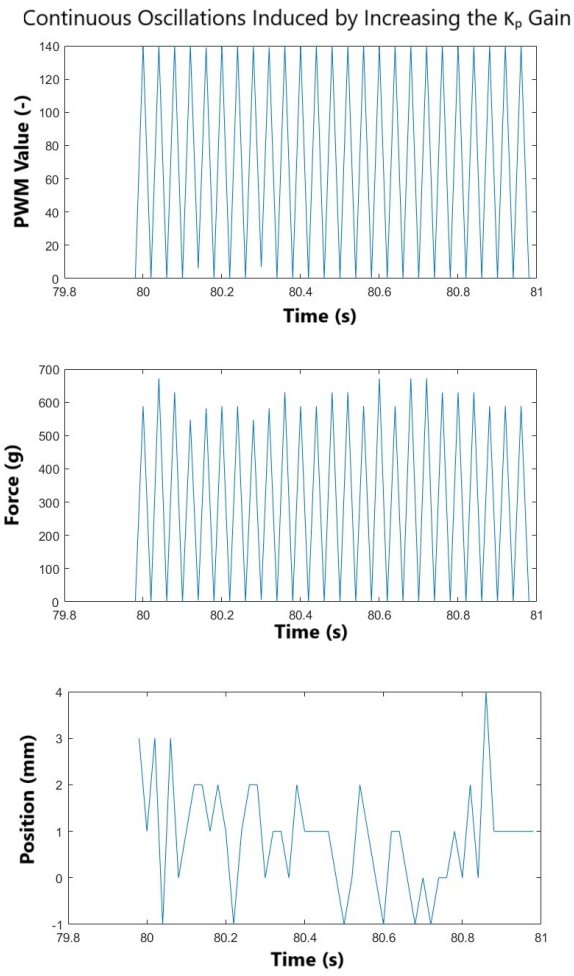


Fig. 18. Sustained oscillations, induced by increasing K_P. From this, the oscillation period can be determined to be 0.02s and together with the oscillation gain, the initial values for K_P and K_I can be determined for the PI controller.

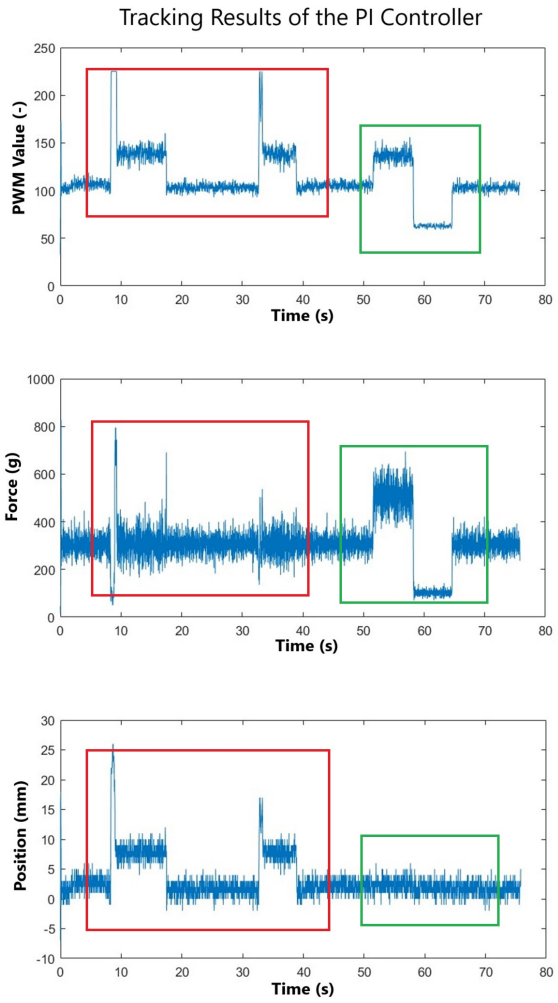


Fig. 19. Tracking results of the calibrated PI controller. Here, it can be seen that the reference force is tracked properly at externally enforced position/displacement changes (red part) and that for a fixed position, the reference force changes are also followed (green part).

D. Stiffness measurements with the gripper system - Results

1) Mechanism intrinsic stiffness measurement with the gripper: In Figure 22, the results of the mechanism's measurements of its own intrinsic stiffness are shown with an applied moving average filter over the data with a window size of 45. Here, 5 subfigures are shown that each contain 5 hysteresis loops. In these figures, a substantial part of the loading and unloading curve are coloured red and green, respectively. Within these region, the linear parts are coloured cyan and magenta, respectively. Through these linear parts, linear regression is performed, resulting in a loading and unloading stiffness value for each of the 5 subfigures. Then, for these 5 loading and unloading stiffnessess, the mean and standard deviation are calculated. The resulting mean loading and unloading stiffness together with their 95% confidence interval are shown in subfigure number 6.

2) Spring stiffness measurement with the gripper: In Figure 23, the measured spring stiffness data by the mechanism for 2, 3 and 4 parallel springs is shown with an applied moving average filter over the data with a window size of 45. Here, again 5 subfigures are shown that each contain 5 cycles. For each subfigure, the linear parts of the loading and unloading region are shown in red and green, respectively. For these linear regions, linear regression is used again to determine the loading and unloading stiffness. Then, the loading and unloading stiffnessess for all 5 subfigures are used to calculate the mean and standard deviation. In subfigure number 6, the mean loading and unloading stiffness are shown together with the 95% confidence bound.

E. Comparison of calibration and measurement data

1) Mechanism intrinsic stiffness data comparison: In Figure 24, the mechanism's intrinsic stiffness data from the calibration process and the mechanism's own measurements are compared. In Figure 25, the error percentages of the mechanism's measured intrinsic loading and unloading stiffness (absolute values), compared to the calibration data, are shown. These error values are 18% and 60% for loading and unloading, respectively.

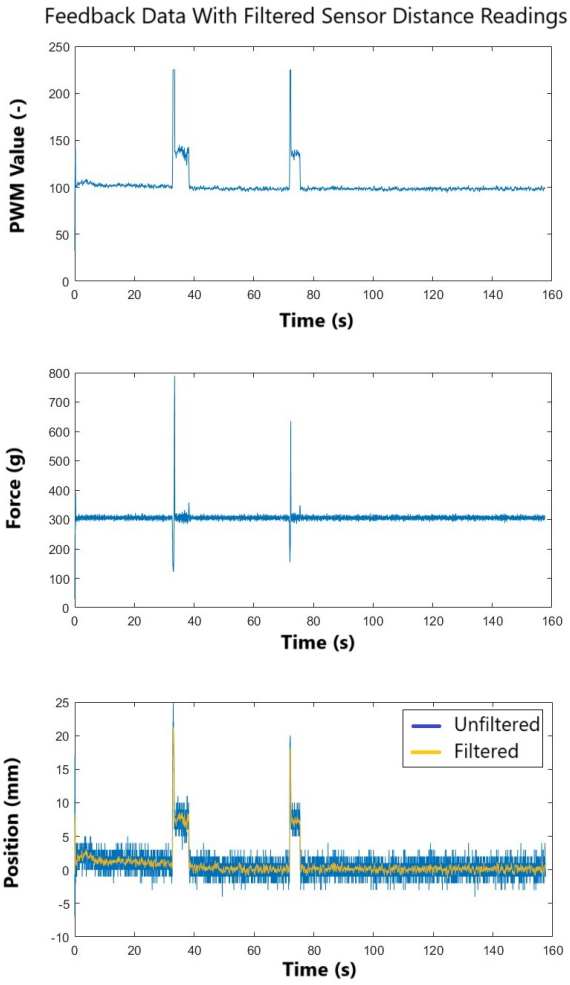


Fig. 20. Feedback with filtered position sensor data. The used filter is a moving average filter with a window size of 10, giving a balance between filtering performance and induced delay (0.2s).

2) Spring stiffness data comparison: In Figure 26, the spring stiffness values for 2, 3 and 4 springs are shown. This is done for the loading and unloading stiffness extracted from the mechanism's own measurements and their average single stiffness value and also for the calibration data value. In Figure 27, the error percentages of the loading and unloading stiffness values (absolute values) for 2, 3 and 4 parallel springs, compared to the calibration data, are shown. The average error values for 2, 3 and 4 parallel springs are 4.2%, 8.0% and 1.0%, respectively.

IV. Discussion

A. Mechanism performance

In Figure 24, it can be seen that the mechanism's estimation of its own intrinsic stiffness is not extremely accurate, varying 18 and 60% from the calibration data for loading and unloading, respectively. Nevertheless, since this intrinsic stiffness has such a low value that it falls within the 95% confidence interval of the spring stiffness measurements, the inaccuracy in the intrinsic stiffness measurement can be considered negligible when measuring the object spring stiffness.

Also, in Figure 26, it can be seen that for the stiffness measurement of 2, 3 and 4 parallel springs, the estimated average spring stiffness varies by 4.2%, 8.0% and 1.0%, respectively. Since generally, the error percentage for determining stiffness values lies between 3.5%-30.7% [41], the values obtained by the gripper designed in this paper are on the low error value side, stating that the gripper performs satisfactory in terms of stiffness estimation accuracy.

B. Research objective evaluation

As can be seen in section III, it is possible to create a mechanical gripper that can estimate grasped object stiffness by using inter-finger distance measurement, the control signal and calibration data, eliminating the need for a tactile sensor in the finger tips. This is achieved by choosing a solenoid as the actuator and calibrating it to get a 2D force curve as a function of its stroke length and the applied PWM value control signal. Here, the stroke length is measured by using a ToF distance sensor. To facilitate proper motion without additional friction, a compliant linear guide is used to guide the gripping fingers that are actuated by the solenoid. In the end, by means of calibration, proper calibration data is obtained for the actuator characteristics, the mechanism's intrinsic stiffness and the grasped springs' stiffnesses.

Furthermore, the demand to be able to fabricate the mechanism as a single mechanical part is satisfied, since the mechanism can be 3D-printed as a single piece on a standard plastic FDM 3D-printer out of PETG, which, as stated in subsubsection II-C.1, is also one of the most affordable and easy to print materials, while having enough ductility for making compliant parts.

Considering all of this, it can be concluded that the objective of this research paper, namely to design and experimentally evaluate a two-finger gripper of which the mechanism can be fabricated as a single mechanical part, which can estimate grasped object stiffness without the need of tactile sensors on the finger surfaces and to quantitatively assess the accuracy of the stiffness measurements, is achieved and that the quantitative results of the object stiffness measurements are in the upper spectrum of performance.

C. Recommendations for future research

A recommendation for future research would be to investigate whether the mechanism is also able to accurately determine the stiffness of real tomatoes or other fruits and vegetables, since this was the original perspective from which this thesis arose and this gives the device a practical use now its performance is validated for the ideal case of using perfectly linear metal springs. Also, since fruits and vegetables consist of organic matter, which is generally not linearly elastic, but contains viscoelastic damping effects and material hysteresis, it would be a larger challenge to determine the stiffness of such an object. Therefore, future research is needed and suggested to investigate the suitability

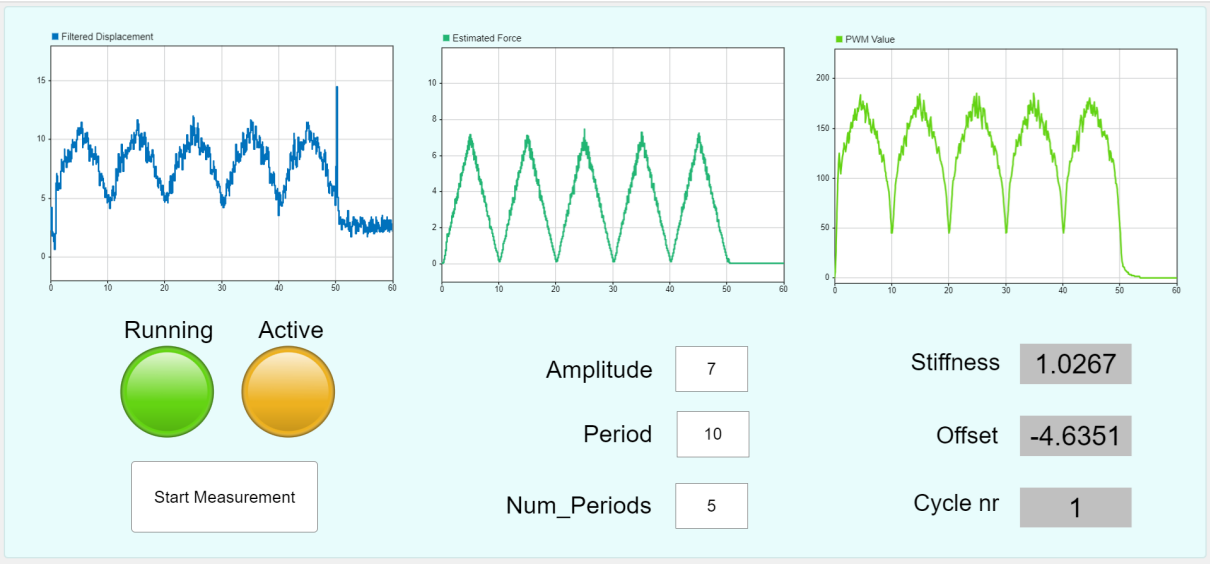


Fig. 21. Simulink control panel. Here, the position, force and control signal data is live plotted. Also, there are indicators showing whether the Simulink model is running and whether a measurement cycle is active. Furthermore, the parameters of the triangle wave can be tuned. Lastly, the calculated stiffness, the calculated offset and the corresponding cycle number are shown for the last completed cycle.

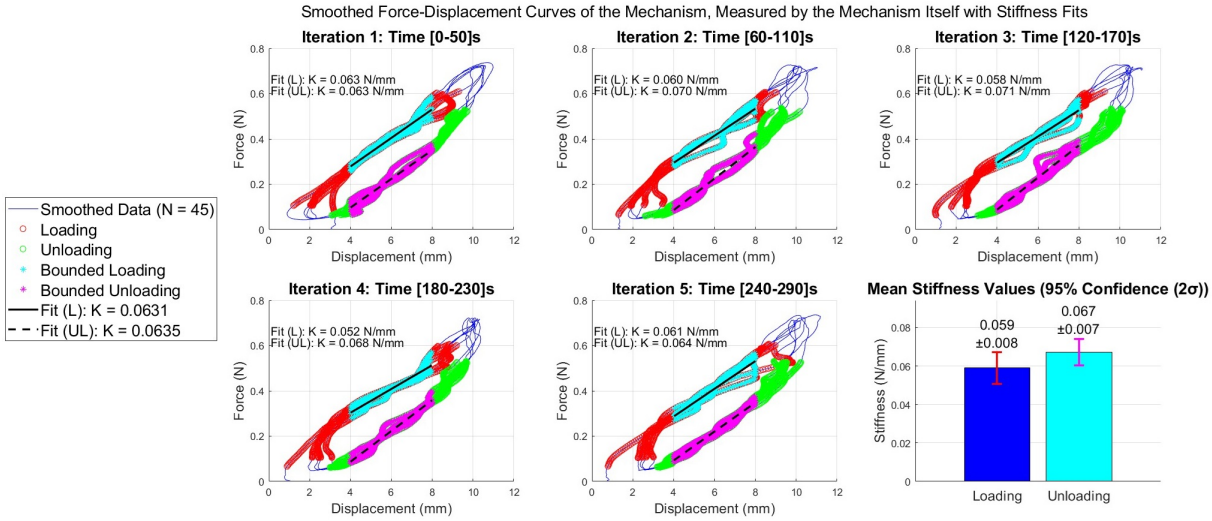


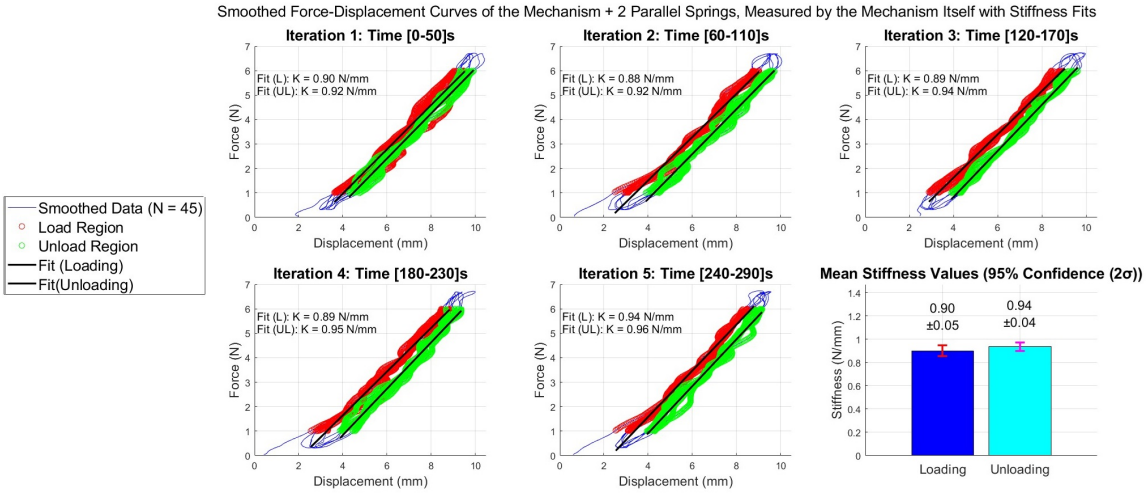
Fig. 22. Mechanism intrinsic stiffness measurement. Here, the mechanism repeats a contraction cycle 5 times per subfigure. This in is total done 5 times, giving 5 subfigures with hysteresis curves. In each hysteresis curve, the linear loading part (cyan) and the linear unloading part (magenta) are shown. From these linear loading and unloading parts, the linear loading and unloading stiffness values are calculated. In subfigure 6, the mean of the 5 calculated loading and unloading stiffnesses are shown, together with their 2σ (95%) confidence interval.

of the currently developed mechanism for determining the stiffness of such fruits and vegetables.

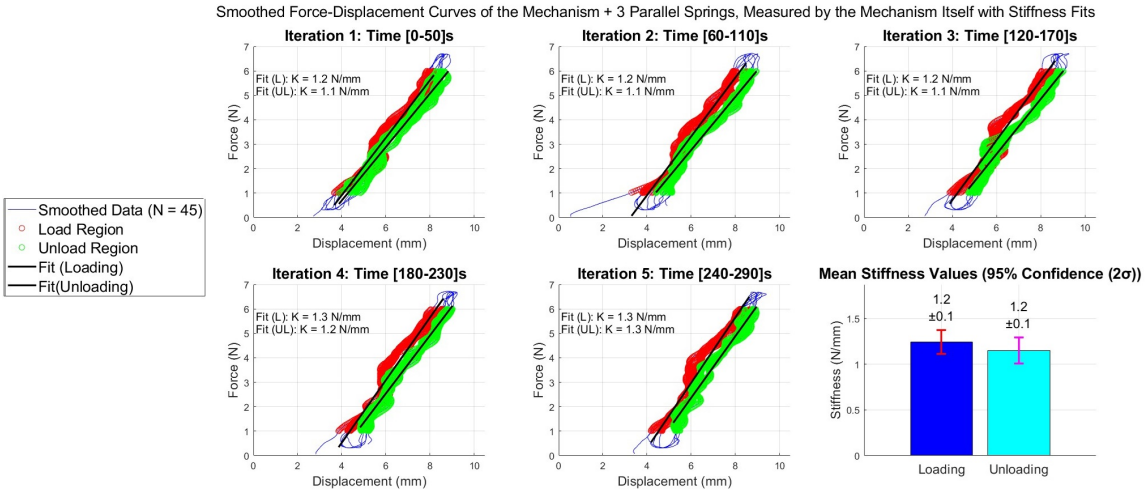
Another recommendation would be to try to integrate the distance sensor reading into the main microcontroller. In this way, no extra microcontroller is needed for that, which also saves on cost and complexity.

Also, it should be tested how well the system performs when mounted onto a robotic arm in a real use-case scenario. Furthermore, since in this case, the system gets exposed to accelerational forces due to the movement of the robotic arm, it should either be ensured that these accelerational forces remain negligibly low or they should be accounted for in the stiffness measurements. Both possibilities should also be further investigated.

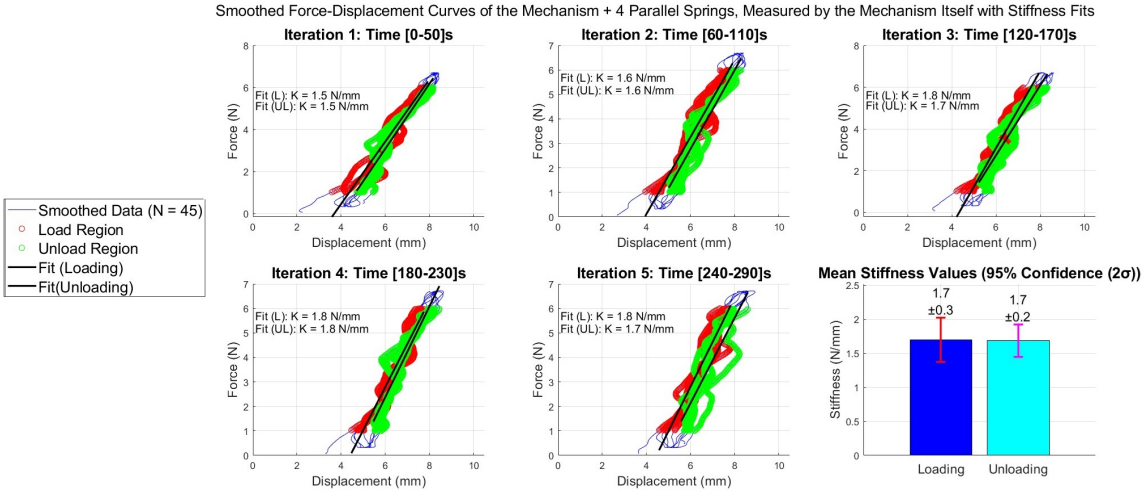
As mentioned in subsubsection II-C.6, one of the possible drawbacks of the current design is that it can deform asymmetrically under certain asymmetric loading conditions. This could for example happen in a real-life situation where one of the gripper fingers collides against an object that results in an in-plane force perpendicular to the direction of motion. Therefore, this effect needs to be studied further to determine whether this can be prevented or made insignificant by properly controlling the robotic arm on which it is mounted, or whether the design needs to be altered to remove this possible phenomenon to ensure that the mechanism can be used in real-life applications.



(a) Stiffness measurement of 2 parallel springs.



(b) Stiffness measurement of 3 parallel springs.



(c) Stiffness measurement of 4 parallel springs.

Fig. 23. Spring stiffness measurement by the mechanism. a) is for 2 springs in parallel, b) for 3 spring and c) for 4 springs. For each of a), b), c), the mechanism performs 5 compression cycles. This is done 5 times, resulting in 5 subfigures with each a hysteresis curve. From each hysteresis curve, the linear loading region (red) and linear unloading region (green) are marked. Then, for both regions, the linear stiffness is estimated. Then, if subfigure 6, the mean of the 5 loading and unloading stiffnesses is shown, together with a 2σ (95%) confidence interval.

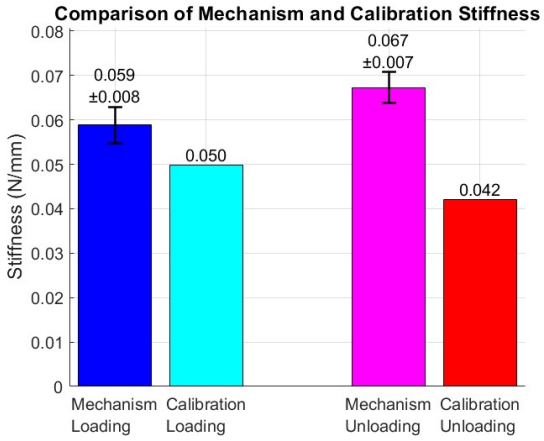


Fig. 24. Comparison of mechanism intrinsic stiffness calibration and measurement data.

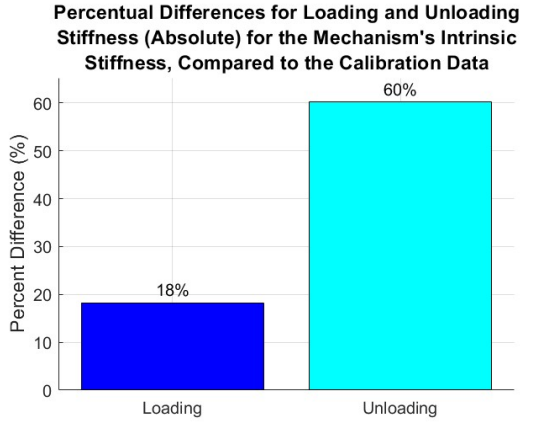


Fig. 25. Error percentages of the mechanism's measured intrinsic loading and unloading stiffness (absolute values), compared to the calibration data.

Furthermore, when using the system in the field, parts of the mechanism should be enclosed against dirt, rain and other elements to protect the mechanism. The mechanism's performance in such outdoor field conditions should then be evaluated.

Another recommendation would be to integrate the control electronics into a single control electronics box that is weather proof. In this way, the electronics are neatly worked away and protected from outside moist and dust.

A further recommendation would be to investigate the stability of the system with its control loop. This would give more insight into the functioning and limitations of the mechanism and also give more insights into the occurring oscillations that arose when using only a single metal spring, as mentioned in subsubsection II-G.2.

A last recommendation would be to increase the measurement speed of the mechanism by using only one triangle wave instead of five, as well as shortening the period of that wave. Using only one triangle can be implemented immediately. However, when shortening the wave period, the intrinsic stiffness calibration must also be redone, since the mechanism's intrinsic hysteresis is speed-dependent. In this way, a single stiffness estimation, which in the experiments performed in subsubsection III-D.2 took 50 seconds, can be reduced to only a few seconds. This is a more realistic speed for commercial implementation. When shortening the stiffness estimation time in this way, the measurement performance should be evaluated to determine whether it is considered sufficient or not. Also, the results could be compared to the results from this paper with 50 seconds measurement time to see how much the results from the increased measurement speed vary from it.

D. Broader use case perspective

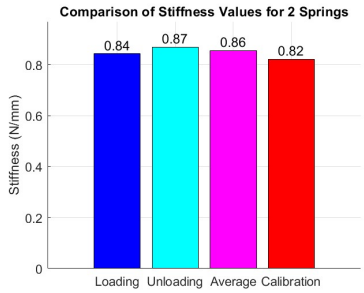
Originally, this project arose from the idea to design a gripper that is able to determine the ripeness of fruits and vegetables by feeling them when they are still connected to their trees, bushes and vines. The idea of this is that the gripper can determine whether the fruit is ripe or not and in this way only pick the ripe fruits, increasing yield and reducing waste. Nevertheless, there are also other opportunities where such a mechanism could be used.

One obvious application would be inside a fruit sorting factory, where the fruits must also be sorted based on their ripeness. Here, such a mechanism could perform that task. In this way, each piece of fruit can be picked up from the incoming supply with the gripper. Then, the gripper directly determines its ripeness and subsequently drops the fruit into the right end station. This can make the process quicker and also non-destructive, compared to using penetrometers for ripeness estimation. Also, this will help to reduce waste and increase edible yield again.

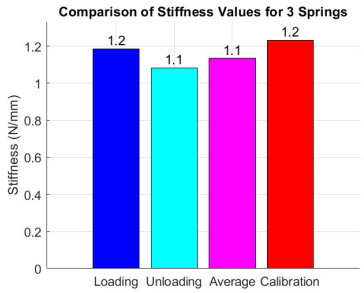
Another possible use case would be at home. Here, consumers could be able to put their fruits into such a gripper to determine whether it is still edible or not and potentially even to predict how long the fruit can still be stored before it will spoil. This can also reduce food waste and make the life of people easier. Since the gripper is designed to be as low-cost as possible, this is specifically beneficial for personal consumers, since personal consumers generally have a lower budget available than large companies for such devices. Therefore, this device could also be targeted at the personal consumer market.

A further possible use case is for example in labs and factories. Here, it could be used by the engineers and factory workers to determine the stiffness of certain springs or other components of which they want to know the stiffness. In this way, the person can just grab a spring or component with unknown stiffness and put it in between the gripper fingers. Then, the gripper can quickly give the estimated stiffness value of the object, which can then immediately be used by the person. In this way, this gripper could be a useful tool for engineers and factory workers that need to know a certain object's stiffness every once in a while.

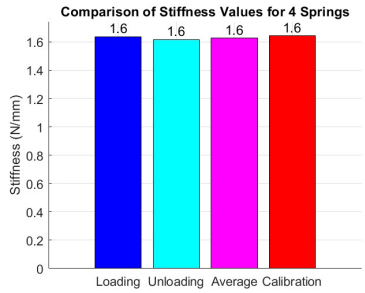
In a more general sense, the principle that was investigated in this paper, namely to estimate force by means of actuator calibration, position measurement and the control signal, may have more potential use, since it has been proven to work. This principle could be used in a multitude of other



(a) Stiffness data for 2 parallel springs.



(b) Stiffness data for 3 parallel springs.



(c) Stiffness data for 4 parallel springs.

Fig. 26. Comparison of stiffness calibration and measurement data for 2, 3 and 4 springs in parallel.

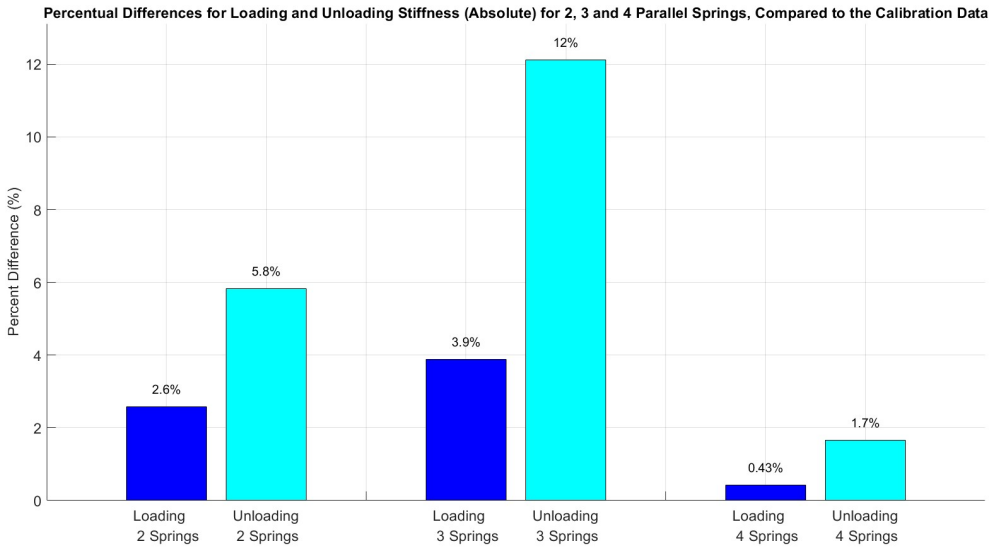


Fig. 27. Error percentages of the measured loading and unloading stiffness (absolute values) for 2, 3 and 4 parallel springs, compared to the calibration data.

scenarios where it can be beneficial to measure force, torque or weight without using force, pressure or similar sensors and an actuator is already required for actuation. One example could be a weight lifting crane. Here, if the lifting motor is calibrated to give a relation between the applied lifting force and the control signal, the weight of the lifted object can be determined.

V. Conclusion

The objective of this research paper was to design and experimentally evaluate a two-finger gripper of which the mechanism can be fabricated as a single mechanical part, which can estimate grasped object stiffness without the need of tactile sensors on the finger surfaces and to quantitatively assess the accuracy of the stiffness measurements.

To achieve this objective, a solenoid is chosen as the actuator and is calibrated to obtain a 2D curve that gives the actuator force as a function of its stroke length and the applied PWM control signal value. The stroke length is measured with a ToF distance sensor. To facilitate the required motion, a compliant linear guide is designed. This guide is 3D-printed as a single piece out of PETG, making the mechanism and its fabrication process straightforward and affordable. In the end, the mechanism's intrinsic stiffness is calibrated. In this way, the mechanism is used to determine the stiffness of 2, 3 and 4 parallel metal springs, which results in estimated stiffness values that vary by 1.0-8.0%, respectively, from the spring's calibrated stiffness values. This is in the upper end of stiffness estimation accuracy, which lies between 3.5%-30.7%.

In the end, it can be stated that the objective of this research paper is achieved, since a two-finger compliant gripper is designed that can be fabricated as a single mechanical part with 3D-printing and it can estimate grasped object stiffness without the need of tactile sensors on the finger surfaces and its measured stiffness values have a deviation from the objects' calibrated stiffness value of 1.0-8.0%, suggesting satisfactory performance.

Acknowledgements

I'd like to thank prof. dr. ir. Just Herder and ir. Domas Syaifoel for supervising this project and giving me valuable

feedback and directions. I'd also like to thank my family and friends for supporting me during this project, helping me with brainstorming and reviewing my paper.

REFERENCES

[1] W. Mandil, V. Rajendran, K. Nazari, and A. Ghalamzan-Esfahani, "Tactile-sensing technologies: Trends, challenges and outlook in agri-food manipulation," *Sensors*, vol. 23, 2023. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85170345300&doi=10.3390%2fs23177362&partnerID=40&md5=c9d89a63c65b2a7066ef7056b922c966

[2] I. Bandyopadhyaya, D. Babu, S. Bhattacharjee, and J. Roychowdhury, "Vegetable grading using tactile sensing and machine learning," *Smart Innovation, Systems and Technologies*, vol. 27, pp. 77 – 86–77 – 86, 2014. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84906719724&doi=10.1007%2f978-3-319-07353-8_10&partnerID=40&md5=357461b32b848017e509d97c9411b488

[3] V. Maharshi, S. Sharma, R. Prajesh, S. Das, A. Agarwal, and B. Mitra, "A novel sensor for fruit ripeness estimation using lithography free approach," *IEEE Sensors Journal*, vol. 22, pp. 22 192 – 22 199–22 192 – 22 199, 2022. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85139821047&doi=10.1109%2fJSEN.2022.3210439&partnerID=40&md5=3033154bcdd0ce53aab0e8073f293594

[4] L. Qin, J. Zhang, S. Stevan, S. Xing, and X. Zhang, "Intelligent flexible manipulator system based on flexible tactile sensing (ifmsfts) for kiwifruit ripeness classification," *Journal of the Science of Food and Agriculture*, vol. 104, pp. 273 – 285–273 – 285, 2024. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85168348130&doi=10.1002%2fjsfa.12916&partnerID=40&md5=1f1ceb109cf54fb4e56152a3e2097465

[5] Y. J. Chen, Y.-C. Liou, W.-H. Ho, J.-T. Tsai, C.-C. Liu, and K.-S. Hwang, "Non-destructive acoustic screening of pineapple ripeness by unsupervised machine learning and wavelet kernel methods," *Science Progress*, vol. 104, 2022. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85133789110&doi=10.1177%2f00368504221110856&partnerID=40&md5=bb2ec4c659376b086c14c7379d4536f6

[6] W. Huang, J. Xia, Y. Wang, X. Jin, H. Zhu, and X. Zhang, "Flexible multimode sensors based on hierarchical microstructures enable non-destructive grading of fruits in cold chain logistics," *Materials Today Sustainability*, vol. 25, 2024. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85183989532&doi=10.1016%2fj.mtsust.2024.100691&partnerID=40&md5=d2405dc92ac56ab3a5b9578395bdf31d

[7] F. E. Erukainure, V. Parque, M. A. Hassan, and A. M. R. FathEl-Bab, "Estimating the stiffness of kiwifruit based on the fusion of instantaneous tactile sensor data and machine learning schemes," *Computers and Electronics in Agriculture*, vol. 201, 2022. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85136294737&doi=10.1016%2fj.compag.2022.107289&partnerID=40&md5=76a303d37b55d5d192f14f9f52f0611d

[8] L. Scimeca, P. Maiolino, D. Cardin-Catalan, A. P. D. Pobil, A. Morales, and F. Iida, "Non-destructive robotic assessment of mango ripeness via multi-point soft haptics," vol. 2019-May, 2019, pp. 1821 – 1826–1821 – 1826. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85071423046&doi=10.1109%2fICRA.2019.8793956&partnerID=40&md5=f43d91d06667331f100ba595bc79933c

[9] V. Cortés, C. Blanes, J. Blasco, C. Ortíz, N. Aleixos, M. Mellado, S. Cubero, and P. Talens, "Integration of simultaneous tactile sensing and visible and near-infrared reflectance spectroscopy in a robot gripper for mango quality assessment," *Biosystems Engineering*, vol. 162, pp. 112 – 123–112 – 123, 2017. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85027870881&doi=10.1016%2fj.biosystemseng.2017.08.005&partnerID=40&md5=3f0162c24fbfdf54836461b2068b4ac0

[10] C. Blanes, V. Cortés, C. Ortiz, M. Mellado, and P. Talens, "Non-destructive assessment of mango firmness and ripeness using a robotic gripper," *Food and Bioprocess Technology*, vol. 8, pp. 1914 – 1924–1914 – 1924, 2015. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84947617343&doi=10.1007%2fs11947-015-1548-2&partnerID=40&md5=df65d6ccaed0f54746daee80a68923bc

[11] S. Li, W. Sun, Q. Liang, C. Liu, and J. Liu, "Assessing fruit hardness in robot hands using electric gripper actuators with tactile sensors," *Sensors and Actuators A: Physical*, vol. 365, 2024. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85177847366&doi=10.1016%2fj.sna.2023.114843&partnerID=40&md5=b59e737973359f481e82e063b59fdaf5

[12] J. Lin, Q. Hu, J. Xia, L. Zhao, X. Du, S. Li, Y. Chen, and X. Wang, "Non-destructive fruit firmness evaluation using a soft gripper and vision-based tactile sensing," *Computers and Electronics in Agriculture*, vol. 214, 2023. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85173566173&doi=10.1016%2fj.compag.2023.108256&partnerID=40&md5=f3af6fdf0cdcc5a1d4917e1ac1f6ea20

[13] L. Jin, Z. Wang, S. Tian, J. Feng, C. An, and H. Xu, "Grasping perception and prediction model of kiwifruit firmness based on flexible sensing claw," *Computers and Electronics in Agriculture*, vol. 215, 2023. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85177200081&doi=10.1016%2fj.compag.2023.108389&partnerID=40&md5=d50cf09f22fbb03c800fc014989cf93c

[14] C. Ma, Y. Ying, and L. Xie, "Visuo-tactile sensor development and its application for non-destructive measurement of peach firmness," *Computers and Electronics in Agriculture*, vol. 218, 2024. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85184655599&doi=10.1016%2fj.compag.2024.108709&partnerID=40&md5=813126815ba10fa075d65b102d73f687

[15] Z. Zhang, J. Zhou, Z. Yan, K. Wang, J. Mao, and Z. Jiang, "Hardness recognition of fruits and vegetables based on tactile array information of manipulator," *Computers and Electronics in Agriculture*, vol. 181, 2021. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85100154474&doi=10.1016%2fj.compag.2020.105959&partnerID=40&md5=d81b3dcfe063ef91a625044f09fc05df

[16] Y. Wei, L. Cai, H. Fang, and H. Chen, "Fruit recognition and classification based on tactile information of flexible hand," *Sensors and Actuators A: Physical*, vol. 370, 2024. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85186680285&doi=10.1016%2fj.sna.2024.115224&partnerID=40&md5=c0f8b9bcb229ebadbbe63eac775523a9

[17] L. Cai, H. Chen, X. Zuo, Y. Wei, and S. Dong, "Identification of fruit using a flexible tactile sensor array," *Instrumentation Science and Technology*, 2024. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85183103567&doi=10.1080%2f10739149.2024.2306462&partnerID=40&md5=2780e781c815d7437b9db073171da7e9

[18] W. Huang, J. Xia, N. Li, H. Zhu, and X. Zhang, "Improvement of a flexible multimode pressure-strain sensor (fmpss) for blueberry firmness tactile sensing and tamper-evident packaging," *Food Control*, vol. 156, 2024. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85173045764&doi=10.1016%2fj.foodcont.2023.110129&partnerID=40&md5=be6ecc1fe896b4da17fe46c9fcbd3ad9

[19] T. Hiruta, N. Hosoya, S. Maeda, and I. Kajiwara, "Experimental evaluation of frequency response and firmness of apples based on an excitation technique using a dielectric elastomer actuator," *Sensors and Actuators, A: Physical*, vol. 330, 2021. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85106927691&doi=10.1016%2fj.sna.2021.112830&partnerID=40&md5=bd5ba9388ff22f20505012d4d6c06501

[20] C. Ortiz, C. Blanes, P. Gonzalez-Planells, and F. Rovira-Más, "Non-destructive evaluation of white-flesh dragon fruit decay with a robot," *Horticulturae*, vol. 9, 2023. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85180427691&doi=10.3390%2fhorticulturae9121286&partnerID=40&md5=a9ee40b4c48032ef1a1befbce09492be

[21] C. Blanes, C. Ortiz, M. Mellado, and P. Beltrán, "Assessment of eggplant firmness with accelerometers on a pneumatic robot gripper," *Computers and Electronics in Agriculture*, vol. 113, pp. 44 – 50–44 – 50, 2015. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84922390693&doi=10.1016%2fj.compag.2015.01.013&partnerID=40&md5=c263ba2f4cc5c248335e0c652598c407

[22] "Solentec limited linear solenoid, 6 v dc, 57.7 x 25.4 x 31.8 mm." [Online]. Available: https://nl.rs-online.com/web/p/linear-solenoids/9059947

[23] "Vl6180 time-of-flight tof afstandssensor." [Online]. Available: https://www.tinytronics.nl/nl/sensoren/afstand/vl6180-time-of-flight-tof-afstandssensor

[24] "Github - adafruit/adafruit_vl6180x: Arduino library for adafruit vl6180x." [Online]. Available: https://github.com/adafruit/Adafruit_VL6180X
[25] "15a single dc motor driver sku dri0042." [Online]. Available: https://wiki.dfrobot.com/15A_Single_DC_Motor_Driver_SKU__DRI0042
[26] "Dc-motor driver - 1x15a." [Online]. Available: https://www.tinytronics.nl/nl/mechanica-en-actuatoren/motoraansturingen-en-drivers/dc-motoraansturingen-en-drivers/dc-motor-driver-1x15a
[27] "L298n dc-motoraansturing." [Online]. Available: https://www.tinytronics.nl/nl/mechanica-en-actuatoren/motoraansturingen-en-drivers/dc-motoraansturingen-en-drivers/l298n-bipolaire-stappenmotor-en-dc-motor-motoraansturing
[28] "Software." [Online]. Available: https://www.arduino.cc/en/software
[29] "Azdelivery 5 x microcontroller board atmega328 met usb-kabel inclusief e-book : Amazon.nl: Elektronica." [Online]. Available: https://www.amazon.nl/dp/B08KRYQNRL/ref=twister_B07Z6PN7V3?_encoding=UTF8&th=1
[30] "Simulink - simulation and model-based design." [Online]. Available: https://nl.mathworks.com/products/simulink.html
[31] "Esp32 wi-fi en bluetooth board - cp2102." [Online]. Available: https://www.tinytronics.nl/nl/development-boards/microcontroller-boards/met-wi-fi/esp32-wifi-en-bluetooth-board-cp2102
[32] "Simulink support package for arduino hardware." [Online]. Available: https://nl.mathworks.com/matlabcentral/fileexchange/40312-simulink-support-package-for-arduino-hardware
[33] "Runcci-yun 15 stks 4 kanalen iic i2c logic niveau converter bi-directionele module 3.3v naar 5v shifter voor arduino (pack van 15) : Amazon.nl: Elektronica." [Online]. Available: https://www.amazon.nl/gp/product/B082F6BSB5/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1
[34] "Tactiele button 12x12x7,3mm rood - 5 stuks - opencircuit." [Online]. Available: https://opencircuit.nl/product/tactiele-button-12x12x7.3mm-rood-5-stuks
[35] "Petg filament: All you need to know."
[36] "123-3d.nl - 3d-printers | kits | parts | filament." [Online]. Available: https://www.123-3d.nl/123-3D-Filament-Zwart-1-75-mm-PETG-1-kg-Jupiter-serie-i10227-t7396.html
[37] J. Rommers, "Design strategies for large range flexure mechanisms," Dissertation, Delft University of Technology, 2022. [Online]. Available: https://doi.org/10.4233/uuid:4eaf8bdb-53e2-49e5-9730-2550945b267f
[38] H. W. Huysamen, W. A. Kinnear, T. E. Fonternel, E. W. Turton, I. Yadroitsava, and I. Yadroitsev, "3d printed laryngoscope for endotracheal intubation," *South African Journal of Industrial Engineering*, vol. 31, pp. 209–217, 2020.
[39] "Matlab." [Online]. Available: https://nl.mathworks.com/products/matlab.html
[40] "Ziegler–nichols method - wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Ziegler%E2%80%93Nichols_method
[41] S. P. Patni, P. Stoudek, H. Chlup, and M. Hoffmann, "Online elasticity estimation and material sorting using standard robot grippers," *International Journal of Advanced Manufacturing Technology*, vol. 132, pp. 6033–6051, 6 2024.

APPENDIX I
ALL MEASURED CALIBRATION DATA

A. Solenoid calibration data

TABLE V
SOLENOID CALIBRATION DATA

1

s (mm)	PWM_val (-)					
	0	**45**	**90**	**135**	**180**	**225**
4	0	22	185	476	783	1035
5,5	0	17	145	384	675	953
7	0	15	114	308	575	860
8,5	0	11	93	260	493	775
10	0	11	80	222	431	690
11,5	0	8	64	179	355	577
13	0	8	53	145	284	470
14,5	0	4	38	105	208	344
16	0	2	29	79	156	259

2

s (mm)	PWM_val (-)					
	0	**45**	**90**	**135**	**180**	**225**
4	0	25	188	474	753	991
5,5	0	15	140	364	644	916
7	0	15	112	303	560	824
8,5	0	12	91	253	474	738
10	0	11	80	217	415	663
11,5	0	10	68	187	367	595
13	0	6	51	144	286	474
14,5	0	5	40	108	214	357
16	0	3	28	80	159	264

3

s (mm)	PWM_val (-)					
	0	**45**	**90**	**135**	**180**	**225**
4	0	25	193	483	767	1008
5,5	0	18	139	371	648	900
7	0	14	111	299	563	830
8,5	0	11	92	256	488	745
10	0	11	80	216	418	665
11,5	0	8	65	180	354	578
13	0	7	53	143	280	460
14,5	0	4	37	105	205	340
16	0	3	28	78	152	253

4

s (mm)	PWM_val (-)					
	0	**45**	**90**	**135**	**180**	**225**
4	0	22	184	465	763	1010
5,5	0	18	140	373	650	901
7	0	14	112	301	562	824
8,5	0	13	93	250	476	734
10	0	10	77	215	419	670
11,5	0	8	64	181	347	563
13	0	7	51	141	279	458
14,5	0	6	40	107	209	344
16	0	3	29	79	155	254

5

s (mm)	PWM_val (-)					
	0	**45**	**90**	**135**	**180**	**225**
4	0	21	187	467	771	1015
5,5	0	16	140	365	647	915
7	0	12	112	305	566	849
8,5	0	12	93	253	479	743
10	0	8	74	212	406	652
11,5	0	10	66	179	350	567
13	0	6	51	141	279	465
14,5	0	6	38	103	202	333
16	0	3	28	78	153	252

Mean

s (mm)	PWM_val (-)					
	0	**45**	**90**	**135**	**180**	**225**
4	0	23	187,4	473	767,4	1011,8
5,5	0	16,8	140,8	371,4	652,8	917
7	0	14	112,2	303,2	565,2	837,4
8,5	0	11,8	92,4	254,4	482	747
10	0	10,2	78,2	216,4	417,8	668
11,5	0	8,8	65,4	181,2	354,6	576
13	0	6,8	51,8	142,8	281,6	465,4
14,5	0	5	38,6	105,6	207,6	343,6
16	0	2,8	28,4	78,8	155	256,4

Std

s (mm)	PWM_val (-)					
	0	**45**	**90**	**135**	**180**	**225**
4	0	1,870829	3,507136	7,245688	10,99091	15,8019
5,5	0	1,30384	2,387467	8,018728	12,59762	21,48255
7	0	1,224745	1,095445	3,49285	5,890671	16,27268
8,5	0	0,83666	0,894427	3,781534	8,154753	16,23268
10	0	1,30384	2,683282	3,646917	8,983318	13,94633
11,5	0	1,095445	1,67332	3,34664	7,635444	12,40967
13	0	0,83666	1,095445	1,788854	3,209361	6,69328
14,5	0	1	1,341641	1,949359	4,505552	8,734987
16	0	0,447214	0,547723	0,83666	2,738613	5,029911

B. Mechanism calibration data

TABLE VI

MECHANISM CALIBRATION DATA

1

	Incr. Weight		Decr. Weight	
Weight (g)	**Distance (mm)**	**Displacement (mm)**	**Distance (mm)**	**Displacement (mm)**
0	12	0	11	1
11,5	10	2	9,5	2,5
22,2	8	4	6	6
32,9	6	6	4	8
43,6	4	8	1	11
54,3	2	10	0	12
65	0	12	0	12

2

	Incr. Weight		Decr. Weight	
Weight (g)	**Distance (mm)**	**Displacement (mm)**	**Distance (mm)**	**Displacement (mm)**
0	12	0	11	1
11,5	9,5	2,5	8	4
22,2	8	4	6	6
32,9	6	6	3	9
43,6	4	8	1	11
54,3	1	11	0	12
65	0	12	0	12

3

	Incr. Weight		Decr. Weight	
Weight (g)	**Distance (mm)**	**Displacement (mm)**	**Distance (mm)**	**Displacement (mm)**
0	12	0	11,5	0,5
11,5	10	2	8	4
22,2	8	4	5,5	6,5
32,9	6	6	4	8
43,6	3	9	1	11
54,3	1	11	0	12
65	0	12	0	12

4

	Incr. Weight		Decr. Weight	
Weight (g)	**Distance (mm)**	**Displacement (mm)**	**Distance (mm)**	**Displacement (mm)**
0	12	0	11	1
11,5	10	2	9	3
22,2	8	4	5	7
32,9	5,5	6,5	3,5	8,5
43,6	4	8	1	11
54,3	1	11	0	12
65	0	12	0	12

5

	Incr. Weight		Decr. Weight	
Weight (g)	**Distance (mm)**	**Displacement (mm)**	**Distance (mm)**	**Displacement (mm)**
0	12	0	11	1
11,5	10	2	9	3
22,2	8	4	5,5	6,5
32,9	6	6	3,5	8,5
43,6	4	8	1	11
54,3	1	11	0	12
65	0	12	0	12

Mean

	Incr. Weight		Decr. Weight	
Weight (g)	**Distance (mm)**	**Displacement (mm)**	**Distance (mm)**	**Displacement (mm)**
0	12	0	11,1	0,9
11,5	9,9	2,1	8,7	3,3
22,2	8	4	5,6	6,4
32,9	5,9	6,1	3,6	8,4
43,6	3,8	8,2	1	11
54,3	1,2	10,8	0	12
65	0	12	0	12

Std

	Incr. Weight		Decr. Weight	
Weight (g)	**Distance (mm)**	**Displacement (mm)**	**Distance (mm)**	**Displacement (mm)**
0	0	0	0,223606798	0,223606798
11,5	0,223606798	0,223606798	0,670820393	0,670820393
22,2	0	0	0,418330013	0,418330013
32,9	0,223606798	0,223606798	0,418330013	0,418330013
43,6	0,447213595	0,447213595	0	0
54,3	0,447213595	0,447213595	0	0
65	0	0	0	0

C. Spring stiffness calibration data

TABLE VII
SPRING STIFFNESS CALIBRATION DATA

Weight (g)	0	3001		
Measurement nr (-)	**D0 (mm)**	**Dend (mm)**	**K4**	167,6536
1	48	29	**K1**	41,91341
2	48	28		
3	47	31		
4	47	31,5		
5	48	29		
Mean	47,6	29,7		
Std	0,547723	1,4832397		

APPENDIX II
ALL MATLAB CODE

A. 01. Plot_solenoid_mean_and_std

```
% Load Force Data (Mean and Std) in grams
stroke_lengths = [4; 5.5; 7; 8.5; 10; 11.5; 13; 14.5; 16];  % Stroke lengths in mm
pwm_values = [0, 45, 90, 135, 180, 225];                    % PWM values

% Mean Force Data (in grams) -> Convert to Newtons
force_mean = [
    0, 23, 187.4, 473, 767.4, 1011.8;
    0, 16.8, 140.8, 371.4, 652.8, 917;
    0, 14, 112.2, 303.2, 565.2, 837.4;
    0, 11.8, 92.4, 254.4, 482, 747;
    0, 10.2, 78.2, 216.4, 417.8, 668;
    0, 8.8, 65.4, 181.2, 354.6, 576;
    0, 6.8, 51.8, 142.8, 281.6, 465.4;
    0, 5, 38.6, 105.6, 207.6, 343.6;
    0, 2.8, 28.4, 78.8, 155, 256.4
] * 9.80665e-3;  % Convert to Newtons

% Standard Deviation Data (in grams) -> Convert to Newtons
force_std = [
    0, 1.8708, 3.5071, 7.2457, 10.991, 15.8019;
    0, 1.3038, 2.3875, 8.0187, 12.5976, 21.4826;
    0, 1.2247, 1.0954, 3.4928, 5.8907, 16.2727;
    0, 0.8367, 0.8944, 3.7815, 8.1548, 16.2327;
    0, 1.3038, 2.6833, 3.6469, 8.9833, 13.9463;
    0, 1.0954, 1.6733, 3.3466, 7.6354, 12.4097;
    0, 0.8367, 1.0954, 1.7889, 3.2094, 6.6933;
    0, 1, 1.3416, 1.9494, 4.5056, 8.735;
    0, 0.4472, 0.5477, 0.8367, 2.7386, 5.0299
] * 9.80665e-3;  % Convert to Newtons

% Create Meshgrid for Interpolation
[StrokeGrid, PWMGrid] = ndgrid(stroke_lengths, pwm_values);

% Interpolation Function for Mean Force
forceInterpMean = griddedInterpolant(StrokeGrid, PWMGrid, force_mean, 'linear', '
    linear');

% Interpolated Grid for Plotting
[PWMInterp, StrokeInterp] = meshgrid(0:5:225, 4:0.5:16);
InterpolatedForceMean = forceInterpMean(StrokeInterp, PWMInterp);

% Plot 2x2 Subplots
figure;

% Subplot 1: Force vs Stroke Length for Each PWM (with Error Bars)
subplot(2, 2, 1);
hold on;
confidence_factor = 2;  % 95% confidence level
for i = 1:length(pwm_values)
    errorbar(stroke_lengths, force_mean(:, i), confidence_factor * force_std(:, i),
        'o-', ...
        'DisplayName', sprintf('PWM = %d', pwm_values(i)));
end
```

```
xlabel('Stroke Length (mm)');
ylabel('Force (N)');  % Update unit to Newtons
title('Force vs Stroke Length (with 95% Confidence (2  ))');
legend('Location', 'Best', 'FontSize', 12);  % Increase font size of legend
grid on;
hold off;

% Subplot 2: Force vs PWM Value for Each Stroke Length (with Error Bars)
subplot(2, 2, 2);
hold on;
for i = 1:length(stroke_lengths)
    errorbar(pwm_values, force_mean(i, :), confidence_factor * force_std(i, :), 'o-'
        , ...
        'DisplayName', sprintf('Stroke = %.1f mm', stroke_lengths(i)));
end
xlabel('PWM');
ylabel('Force (N)');  % Update unit to Newtons
title('Force vs PWM Value (with 95% Confidence (2  ))');
legend('Location', 'Best', 'FontSize', 11);  % Increase font size of legend
grid on;
hold off;

% Subplot 3: 2D Surface Plot of Raw Mean Force Data
subplot(2, 2, 3);
surf(PWMGrid, StrokeGrid, force_mean, 'EdgeColor', 'none');
xlabel('PWM');
ylabel('Stroke Length (mm)');
zlabel('Force (N)');  % Update unit to Newtons
title('Raw Mean Force Data');
colorbar;
grid on;

% Subplot 4: 2D Surface Plot of Interpolated Force Data
subplot(2, 2, 4);
surf(PWMInterp, StrokeInterp, InterpolatedForceMean, 'EdgeColor', 'none');
xlabel('PWM');
ylabel('Stroke Length (mm)');
zlabel('Interpolated Force (N)');  % Update unit to Newtons
title('Interpolated Force Data');
colorbar;
grid on;
```

B. 02. Plot_mechanism_F_D

```
% Input Force (grams) and Displacement Vectors for Loading and Unloading
force = [0 11.5 22.2 32.9 43.6 54.3 65]; % Force vector in grams

% Loading and Unloading Displacement Data (mean values) in mm
displacement_loading =   [0   2.1 4   6.1 8.2 10.8 12];  % Mean loading displacement
displacement_unloading = [0.9 3.3 6.4 8.4 11  12   12];  % Mean unloading
    displacement

% Standard Deviations for Loading and Unloading Displacement (example values)
std_loading =   [0           0.223606798 0           0.223606798 0.447213595
    0.447213595 0]; % Standard deviation for loading
std_unloading = [0.223606798 0.670820393 0.418330013 0.418330013 0           0 0
            ]; % Standard deviation for unloading

% Convert Force to Newtons (Optional)
force_N = force * 9.80665 / 1000; % Conversion: 1 gram = 9.80665e-3 Newtons

% Step 1: Specify Indices for Linear Regression
% Manually choose indices for regression
indices_loading = 1:6;    % Example: Use indices 2 to 6 for loading
indices_unloading = 1:5;  % Example: Use indices 2 to 6 for unloading

% Extract Data Based on Specified Indices
force_load_selected = force_N(indices_loading);
disp_load_selected = displacement_loading(indices_loading);

force_unload_selected = force_N(indices_unloading);
disp_unload_selected = displacement_unloading(indices_unloading);

% Step 2: Perform Linear Regression for Loading
coeffs_loading = polyfit(disp_load_selected, force_load_selected, 1); % Linear fit (
    degree 1)
slope_loading = coeffs_loading(1);   % Stiffness in N/mm
intercept_loading = coeffs_loading(2); % Offset

% Step 3: Perform Linear Regression for Unloading
coeffs_unloading = polyfit(disp_unload_selected, force_unload_selected, 1); % Linear
     fit (degree 1)
slope_unloading = coeffs_unloading(1);   % Stiffness in N/mm
intercept_unloading = coeffs_unloading(2); % Offset

% Generate Regression Line Data
displacement_fit_loading = linspace(min(disp_load_selected), max(disp_load_selected)
    , 100);
force_fit_loading = polyval(coeffs_loading, displacement_fit_loading);

displacement_fit_unloading = linspace(min(disp_unload_selected), max(
    disp_unload_selected), 100);
force_fit_unloading = polyval(coeffs_unloading, displacement_fit_unloading);

% Step 4: Plot the Data and Linear Regression
figure;
hold on;

% Plot Data Points with Horizontal Error Bars (2 Sigma = 95% Confidence)
```

```
errorbar(displacement_loading, force_N, 2*std_loading, 'horizontal', 'bo', ...
    'MarkerSize', 8, 'MarkerFaceColor', 'b', 'CapSize', 5, 'DisplayName', 'Loading Data (with 95% Confidence (2 ))');
errorbar(displacement_unloading, force_N, 2*std_unloading, 'horizontal', 'cdiamond', ...
    'MarkerSize', 8, 'MarkerFaceColor', 'c', 'CapSize', 5, 'DisplayName', 'Unloading Data (with 95% Confidence (2 ))');

% Highlight Selected Points for Regression
plot(disp_load_selected, force_load_selected, 'r.', 'MarkerSize', 12, ...
    'DisplayName', 'Selected Loading Points');
plot(disp_unload_selected, force_unload_selected, 'm.', 'MarkerSize', 12, ...
    'DisplayName', 'Selected Unloading Points');

% Plot Linear Regression Lines
plot(displacement_fit_loading, force_fit_loading, 'r-', 'LineWidth', 2, ...
    'DisplayName', sprintf('Loading Fit:     Stiffness = %.4f, Offset = %.4f', slope_loading, intercept_loading));
plot(displacement_fit_unloading, force_fit_unloading, 'm-', 'LineWidth', 2, ...
    'DisplayName', sprintf('Unloading Fit: Stiffness = %.4f, Offset = %.4f', slope_unloading, intercept_unloading));

% Labels, Title, and Legend
xlabel('Displacement (mm)');
ylabel('Force (N)');
title('Force-Displacement Curve: Loading vs. Unloading with Linear Regression');
legend('Location', 'best');
grid on;

% Step 5: Display Results
fprintf('Loading Linear Fit: F = %.4f * d + %.4f (N)\n', slope_loading, intercept_loading);
fprintf('Unloading Linear Fit: F = %.4f * d + %.4f (N)\n', slope_unloading, intercept_unloading);
fprintf('Loading Stiffness: %.4f N/mm\n', slope_loading);
fprintf('Unloading Stiffness: %.4f N/mm\n', slope_unloading);
```

C. 03. Show_spring_kalibration_stiffness

```
% Input Data
Force = [0 3001]; % Force in grams
Position_mean = [47.6 29.7]; % Mean position in mm
Position_std = [0.547722558 1.483239697]; % Standard deviation in mm

% Compute Displacement (Negative Difference from Initial Position)
Displacement_mean = -(Position_mean - Position_mean(1)); % Mean displacement in mm
Displacement_std = Position_std; % Standard deviation in displacement

% Analysis for 4 Springs in Parallel
% Compute Stiffness for 4 springs
Stiffness_4springs = (Force(2) - Force(1)) / (Displacement_mean(2) -
    Displacement_mean(1)); % Stiffness in g/mm
Stiffness_4springs_Nmm = Stiffness_4springs * (9.80665 / 1000); % Convert to N/mm

% Generate Linear Stiffness Line for 4 springs
Displacement_fit = linspace(Displacement_mean(1), Displacement_mean(2), 100); %
    Displacement values for fit
Force_fit_4springs = Force(1) + Stiffness_4springs * (Displacement_fit -
    Displacement_mean(1)); % Linear relationship

% Analysis for 1 Spring (1/4 Force and 1/2 Std)
Force_1spring = Force / 4; % Force vector for 1 spring in grams
Stiffness_1spring = Stiffness_4springs / 4; % Stiffness in g/mm for 1 spring
Stiffness_1spring_Nmm = Stiffness_1spring * (9.80665 / 1000); % Convert to N/mm

% Adjust Displacement Standard Deviation for 1 Spring
Displacement_std_1spring = Displacement_std / 2; % Standard deviation for 1 spring

% Generate Linear Stiffness Line for 1 spring
Force_fit_1spring = Force_1spring(1) + Stiffness_1spring * (Displacement_fit -
    Displacement_mean(1)); % Linear relationship

% Convert Forces to Newtons for Plotting
Force_N_4springs = Force * 9.80665 / 1000; % Convert grams to Newtons (4 springs)
Force_N_1spring = Force_1spring * 9.80665 / 1000; % Convert grams to Newtons (1
    spring)

% Plot Data Points with 2 Sigma Error Bars
figure;
hold on;

% Plot displacement points for 4 springs with 2 sigma horizontal error bars
errorbar(Displacement_mean, Force_N_4springs, 2 * Displacement_std, 'horizontal',
    ...
    'bo', 'MarkerSize', 8, 'MarkerFaceColor', 'b', 'CapSize', 5, ...
    'DisplayName', 'Data (4 Springs, 95% Confidence (2 ))');

% Plot displacement points for 1 spring with 2 sigma horizontal error bars
errorbar(Displacement_mean, Force_N_1spring, 2 * Displacement_std_1spring, '
    horizontal', ...
    'cdiamond', 'MarkerSize', 8, 'MarkerFaceColor', 'c', 'CapSize', 5, ...
    'DisplayName', 'Data (1 Spring, 95% Confidence (2 ))');

% Plot Linear Stiffness Line for 4 springs
```

```
plot(Displacement_fit, Force_fit_4springs * 9.80665 / 1000, 'r-', 'LineWidth', 2, ...
    'DisplayName', sprintf('Stiffness (4 Springs): %.4f N/mm', Stiffness_4springs_Nmm));

% Plot Linear Stiffness Line for 1 spring
plot(Displacement_fit, Force_fit_1spring * 9.80665 / 1000, 'm-', 'LineWidth', 2, ...
    'DisplayName', sprintf('Stiffness (1 Spring): %.4f N/mm', Stiffness_1spring_Nmm));

% Labels, Title, and Legend
xlabel('Displacement (mm)');
ylabel('Force (N)');
title('Force-Displacement Analysis: 4 Springs and 1 Spring');
legend('Location', 'best');
grid on;

% Display Stiffness in Command Window
fprintf('Stiffness (4 Springs): %.4f N/mm\n', Stiffness_4springs_Nmm);
fprintf('Stiffness (1 Spring): %.4f N/mm\n', Stiffness_1spring_Nmm);
```

D. 04. Mechanisme_eigen_stijfheid_bepaling_3

```
% Signal parameters
T = 10; % Period in seconds
N = 5;  % Number of periods
time_window = 60; % Total time window for each iteration in seconds
analysis_duration = 50; % Analysis duration in seconds within each window

% Time bounds for loading and unloading
t_loading_min = 1;
t_loading_max = 4;
t_unloading_min = 6.5;
t_unloading_max = 10;

% Smoothing parameters
window_size = 45; % Adjust based on noise level
smooth_disp = movmean(Displacement, window_size); % Smooth displacement
smooth_force = movmean(Force, window_size);       % Smooth force

% Bounds for displacement
Dminbound = 4;
Dmaxbound = 8;

% Initialize arrays to store stiffness values
loading_stiffnesses = zeros(1, 5);
unloading_stiffnesses = zeros(1, 5);

% Prepare figure for subplots
figure();
for iteration = 1:5
    % Time bounds for current iteration
    start_time = (iteration - 1) * time_window;
    end_time = start_time + analysis_duration; % Limit to first 50 seconds

    % Filter data for the current time window
    time_indices = t >= start_time & t < end_time;
    t_window = t(time_indices);
    disp_window = smooth_disp(time_indices);
    force_window = smooth_force(time_indices);

    % Identify loading and unloading regions using time
    loading_indices = [];
    unloading_indices = [];

    for i = 1:length(t_window)
        current_time = mod(t_window(i), T); % Map time to within a single period
        if current_time > t_loading_min && current_time < t_loading_max
            loading_indices = [loading_indices i];
        elseif current_time > t_unloading_min && current_time < t_unloading_max
            unloading_indices = [unloading_indices i];
        end
    end

    % Extract loading and unloading data
    loading_disp = disp_window(loading_indices);
    loading_force = force_window(loading_indices);
    unloading_disp = disp_window(unloading_indices);
```

```
56      unloading_force = force_window(unloading_indices);
57
58      % Apply bounds to loading and unloading data
59      bounded_loading_indices = (loading_disp > Dminbound & loading_disp < Dmaxbound);
60      bounded_unloading_indices = (unloading_disp > Dminbound & unloading_disp <
            Dmaxbound);
61
62      bounded_loading_disp = loading_disp(bounded_loading_indices);
63      bounded_loading_force = loading_force(bounded_loading_indices);
64      bounded_unloading_disp = unloading_disp(bounded_unloading_indices);
65      bounded_unloading_force = unloading_force(bounded_unloading_indices);
66
67      % Perform linear regression on bounded data
68      % Loading regression
69      coeffs_loading = polyfit(bounded_loading_disp, bounded_loading_force, 1);
70      stiffness_loading = coeffs_loading(1);
71      offset_loading = coeffs_loading(2);
72
73      % Store loading stiffness for current iteration
74      loading_stiffnesses(iteration) = stiffness_loading;
75
76      % Unloading regression
77      coeffs_unloading = polyfit(bounded_unloading_disp, bounded_unloading_force, 1);
78      stiffness_unloading = coeffs_unloading(1);
79      offset_unloading = coeffs_unloading(2);
80
81      % Store unloading stiffness for current iteration
82      unloading_stiffnesses(iteration) = stiffness_unloading;
83
84      % Generate loading and unloading fit lines
85      disp_fit_loading = linspace(min(bounded_loading_disp), max(bounded_loading_disp)
            , 100);
86      force_fit_loading = polyval(coeffs_loading, disp_fit_loading);
87
88      disp_fit_unloading = linspace(min(bounded_unloading_disp), max(
            bounded_unloading_disp), 100);
89      force_fit_unloading = polyval(coeffs_unloading, disp_fit_unloading);
90
91      % Plot results in subplot
92      subplot(2, 3, iteration);
93      hold on;
94      plot(disp_window, force_window, 'b-'); % Original data
95      plot(loading_disp, loading_force, 'ro'); % Loading points
96      plot(unloading_disp, unloading_force, 'go'); % Unloading points
97      plot(bounded_loading_disp, bounded_loading_force, 'c*'); % Bounded loading
98      plot(bounded_unloading_disp, bounded_unloading_force, 'm*'); % Bounded unloading
99      plot(disp_fit_loading, force_fit_loading, 'k-', 'LineWidth', 2); % Loading fit
100     plot(disp_fit_unloading, force_fit_unloading, 'k--', 'LineWidth', 2); %
            Unloading fit
101
102     % Add stiffness values as text annotations
103     text(0.1, 0.7, sprintf('Fit (L): K = %.4f N/mm', stiffness_loading), 'FontSize',
            12, 'Color', 'k', 'VerticalAlignment', 'top');
104     text(0.1, 0.65, sprintf('Fit (UL): K = %.4f N/mm', stiffness_unloading), '
            FontSize', 12, 'Color', 'k', 'VerticalAlignment', 'top');
105
106     % Add labels, title, and grid
```

```
107         xlabel('Displacement (mm)', 'FontSize', 14);  % Font size for labels
108         ylabel('Force (N)', 'FontSize', 14);  % Font size for labels
109         title(sprintf('Iteration %d: Time [%d-%d]s', iteration, start_time, end_time), '
                FontSize', 16);  % Title size
110         grid on;
111 
112         % Disable legend
113         legend('off');
114         hold off;
115 
116         % Print stiffness and offset values in the command window
117         fprintf('Iteration %d:\n', iteration);
118         fprintf('  Loading Stiffness: %.4f N/mm\n', stiffness_loading);
119         fprintf('  Loading Offset: %.4f N\n', offset_loading);
120         fprintf('  Unloading Stiffness: %.4f N/mm\n', stiffness_unloading);
121         fprintf('  Unloading Offset: %.4f N\n', offset_unloading);
122     end
123 
124     % Calculate the mean and std of the stiffnesses
125     mean_loading_stiffness = mean(loading_stiffnesses);
126     std_loading_stiffness = std(loading_stiffnesses);
127     mean_unloading_stiffness = mean(unloading_stiffnesses);
128     std_unloading_stiffness = std(unloading_stiffnesses);
129 
130     % Print the mean and standard deviation values
131     fprintf('\nMean and Standard Deviation of Stiffnesses:\n');
132     fprintf('  Mean Loading: %.4f N/mm\n', mean_loading_stiffness);
133     fprintf('      Loading: %.4f N/mm\n', std_loading_stiffness);
134     fprintf('  Mean Unloading: %.4f N/mm\n', mean_unloading_stiffness);
135     fprintf('      Unloading: %.4f N/mm\n', std_unloading_stiffness);
136 
137     % Plot the mean stiffness values with 2-sigma error bars as a bar chart in subplot 6
138     subplot(2, 3, 6);
139     hold on;
140 
141     % Plot bars
142     bar(1, mean_loading_stiffness, 'FaceColor', 'b');
143     bar(2, mean_unloading_stiffness, 'FaceColor', 'c');
144 
145     % Add error bars
146     errorbar(1, mean_loading_stiffness, 2 * std_loading_stiffness, 'r', 'LineWidth', 2,
            'CapSize', 10);
147     errorbar(2, mean_unloading_stiffness, 2 * std_unloading_stiffness, 'm', 'LineWidth',
            2, 'CapSize', 10);
148 
149     % Add text annotations above the bars with larger font size
150     text(1, mean_loading_stiffness + 2 * std_loading_stiffness + 0.01, ...
151         sprintf('%.4f\n %.4f', mean_loading_stiffness, 2 * std_loading_stiffness), ...
152         'HorizontalAlignment', 'center', 'FontSize', 14);  % Increased font size here
153     text(2, mean_unloading_stiffness + 2 * std_unloading_stiffness + 0.01, ...
154         sprintf('%.4f\n %.4f', mean_unloading_stiffness, 2 * std_unloading_stiffness),
                ...
155         'HorizontalAlignment', 'center', 'FontSize', 14);  % Increased font size here
156 
157     hold off;
158 
159     % Add title and labels
```

```
160 title('Mean Stiffness Values (95% Confidence (2  ))', 'FontSize', 16);
161
162 % Assign the x-ticks and labels
163 ax = gca();
164 ax.XTick = [1, 2];
165 ax.XLim = [0, 3];  % Adjust X-limits to fit the new tick labels properly
166 ax.XTickLabel = {'Loading', 'Unloading'};
167 ax.TickLabelInterpreter = 'tex'; % Interpret newline in tick labels
168
169 % Increase font size of the x-axis tick labels
170 ax.XAxis.FontSize = 14;  % Adjust this value to your desired font size
171
172 % Add y-axis label
173 ylabel('Stiffness (N/mm)', 'FontSize', 14);
174 grid on;
175
176 % Set y-axis to start at 0
177 ylim([0, max([mean_loading_stiffness + 2 * std_loading_stiffness,
        mean_unloading_stiffness + 2 * std_unloading_stiffness]) + 0.02]);
178
179 % Add a main title for the entire figure
180 sgtitle('Smoothed Force-Displacement Curves of the Mechanism, Measured by the
        Mechanism Itself with Stiffness Fits', 'FontSize', 16);
```

E. 04. Mechanisme_eigen_stijfheid_bepaling_legenda_plot

```
% Signal parameters
T = 10; % Period in seconds
N = 5;  % Number of periods
time_window = 60; % Total time window for each iteration in seconds
analysis_duration = 50; % Analysis duration in seconds within each window

% Time bounds for loading and unloading
t_loading_min = 1;
t_loading_max = 4;
t_unloading_min = 6.5;
t_unloading_max = 10;

% Smoothing parameters
window_size = 45; % Adjust based on noise level
smooth_disp = movmean(Displacement, window_size); % Smooth displacement
smooth_force = movmean(Force, window_size);       % Smooth force

% Bounds for displacement
Dminbound = 4;
Dmaxbound = 8;

% Initialize arrays to store stiffness values
loading_stiffnesses = zeros(1, 5);
unloading_stiffnesses = zeros(1, 5);

% Prepare figure for subplots
figure();
for iteration = 1:5
    % Time bounds for current iteration
    start_time = (iteration - 1) * time_window;
    end_time = start_time + analysis_duration; % Limit to first 50 seconds

    % Filter data for the current time window
    time_indices = t >= start_time & t < end_time;
    t_window = t(time_indices);
    disp_window = smooth_disp(time_indices);
    force_window = smooth_force(time_indices);

    % Identify loading and unloading regions using time
    loading_indices = [];
    unloading_indices = [];

    for i = 1:length(t_window)
        current_time = mod(t_window(i), T); % Map time to within a single period
        if current_time > t_loading_min && current_time < t_loading_max
            loading_indices = [loading_indices i];
        elseif current_time > t_unloading_min && current_time < t_unloading_max
            unloading_indices = [unloading_indices i];
        end
    end

    % Extract loading and unloading data
    loading_disp = disp_window(loading_indices);
    loading_force = force_window(loading_indices);
    unloading_disp = disp_window(unloading_indices);
```

```
56      unloading_force = force_window(unloading_indices);
57
58      % Apply bounds to loading and unloading data
59      bounded_loading_indices = (loading_disp > Dminbound & loading_disp < Dmaxbound);
60      bounded_unloading_indices = (unloading_disp > Dminbound & unloading_disp <
            Dmaxbound);
61
62      bounded_loading_disp = loading_disp(bounded_loading_indices);
63      bounded_loading_force = loading_force(bounded_loading_indices);
64      bounded_unloading_disp = unloading_disp(bounded_unloading_indices);
65      bounded_unloading_force = unloading_force(bounded_unloading_indices);
66
67      % Perform linear regression on bounded data
68      % Loading regression
69      coeffs_loading = polyfit(bounded_loading_disp, bounded_loading_force, 1);
70      stiffness_loading = coeffs_loading(1);
71      offset_loading = coeffs_loading(2);
72
73      % Store loading stiffness for current iteration
74      loading_stiffnesses(iteration) = stiffness_loading;
75
76      % Unloading regression
77      coeffs_unloading = polyfit(bounded_unloading_disp, bounded_unloading_force, 1);
78      stiffness_unloading = coeffs_unloading(1);
79      offset_unloading = coeffs_unloading(2);
80
81      % Store unloading stiffness for current iteration
82      unloading_stiffnesses(iteration) = stiffness_unloading;
83
84      % Generate loading and unloading fit lines
85      disp_fit_loading = linspace(min(bounded_loading_disp), max(bounded_loading_disp)
            , 100);
86      force_fit_loading = polyval(coeffs_loading, disp_fit_loading);
87
88      disp_fit_unloading = linspace(min(bounded_unloading_disp), max(
            bounded_unloading_disp), 100);
89      force_fit_unloading = polyval(coeffs_unloading, disp_fit_unloading);
90
91      % Plot results in subplot
92      subplot(2, 3, iteration);
93      hold on;
94      plot(disp_window, force_window, 'b-', 'DisplayName', 'Smoothed Data (N = 45)');
            % Original data
95      plot(loading_disp, loading_force, 'ro', 'DisplayName', 'Loading'); % Loading
            points
96      plot(unloading_disp, unloading_force, 'go', 'DisplayName', 'Unloading'); %
            Unloading points
97      plot(bounded_loading_disp, bounded_loading_force, 'c*', 'DisplayName', 'Bounded
            Loading'); % Bounded loading
98      plot(bounded_unloading_disp, bounded_unloading_force, 'm*', 'DisplayName', '
            Bounded Unloading'); % Bounded unloading
99      plot(disp_fit_loading, force_fit_loading, 'k-', 'LineWidth', 2, ...
100         'DisplayName', sprintf('Fit (L): K = %.4f', stiffness_loading)); % Loading
                fit
101     plot(disp_fit_unloading, force_fit_unloading, 'k--', 'LineWidth', 2, ...
102         'DisplayName', sprintf('Fit (UL): K = %.4f', stiffness_unloading)); %
                Unloading fit
```

```
103
104     % Add labels, title, and grid
105     xlabel('Displacement (mm)', 'FontSize', 14);  % Font size for labels
106     ylabel('Force (N)', 'FontSize', 14);  % Font size for labels
107     title(sprintf('Iteration %d: Time [%d-%d]s', iteration, start_time, end_time), '
            FontSize', 16);  % Title size
108     grid on;
109
110     % Scale legend size
111     legend('Location', 'northwest', 'FontSize', 16);  % Legend font size
112     hold off;
113
114     % Print stiffness and offset values in the command window
115     fprintf('Iteration %d:\n', iteration);
116     fprintf('  Loading Stiffness: %.4f N/mm\n', stiffness_loading);
117     fprintf('  Loading Offset: %.4f N\n', offset_loading);
118     fprintf('  Unloading Stiffness: %.4f N/mm\n', stiffness_unloading);
119     fprintf('  Unloading Offset: %.4f N\n', offset_unloading);
120 end
121
122 % Calculate the mean and std of the stiffnesses
123 mean_loading_stiffness = mean(loading_stiffnesses);
124 std_loading_stiffness = std(loading_stiffnesses);
125 mean_unloading_stiffness = mean(unloading_stiffnesses);
126 std_unloading_stiffness = std(unloading_stiffnesses);
127
128 % Print the mean and standard deviation values
129 fprintf('\nMean and Standard Deviation of Stiffnesses:\n');
130 fprintf('  Mean Loading: %.4f N/mm\n', mean_loading_stiffness);
131 fprintf('     Loading: %.4f N/mm\n', std_loading_stiffness);
132 fprintf('  Mean Unloading: %.4f N/mm\n', mean_unloading_stiffness);
133 fprintf('     Unloading: %.4f N/mm\n', std_unloading_stiffness);
134
135 % Plot the mean stiffness values with 2-sigma error bars as a bar chart in subplot 6
136 subplot(2, 3, 6);
137 hold on;
138
139 % Plot bars
140 bar(1, mean_loading_stiffness, 'FaceColor', 'b');
141 bar(2, mean_unloading_stiffness, 'FaceColor', 'c');
142
143 % Add error bars
144 errorbar(1, mean_loading_stiffness, 2 * std_loading_stiffness, 'r', 'LineWidth', 2,
        'CapSize', 10);
145 errorbar(2, mean_unloading_stiffness, 2 * std_unloading_stiffness, 'm', 'LineWidth',
        2, 'CapSize', 10);
146
147 % Add text annotations above the bars with larger font size
148 text(1, mean_loading_stiffness + 2 * std_loading_stiffness + 0.01, ...
149     sprintf('%.4f\n %.4f', mean_loading_stiffness, 2 * std_loading_stiffness), ...
150     'HorizontalAlignment', 'center', 'FontSize', 14);  % Increased font size here
151 text(2, mean_unloading_stiffness + 2 * std_unloading_stiffness + 0.01, ...
152     sprintf('%.4f\n %.4f', mean_unloading_stiffness, 2 * std_unloading_stiffness),
            ...
153     'HorizontalAlignment', 'center', 'FontSize', 14);  % Increased font size here
154
155 hold off;
```

```
156
157 % Add title and labels
158 title('Mean Stiffness Values (95% Confidence (2  ))', 'FontSize', 16);
159
160 % Assign the x-ticks and labels
161 ax = gca();
162 ax.XTick = [1, 2];
163 ax.XLim = [0, 3];  % Adjust X-limits to fit the new tick labels properly
164 ax.XTickLabel = {'Loading', 'Unloading'};
165 ax.TickLabelInterpreter = 'tex'; % Interpret newline in tick labels
166
167 % Increase font size of the x-axis tick labels
168 ax.XAxis.FontSize = 14;  % Adjust this value to your desired font size
169
170 % Add y-axis label
171 ylabel('Stiffness (N/mm)', 'FontSize', 14);
172 grid on;
173
174 % Set y-axis to start at 0
175 ylim([0, max([mean_loading_stiffness + 2 * std_loading_stiffness,
      mean_unloading_stiffness + 2 * std_unloading_stiffness]) + 0.02]);
176
177 % Add a main title for the entire figure
178 sgtitle('Smoothed Force-Displacement Curves of the Mechanism, Measured by the
      Mechanism Itself with Stiffness Fits', 'FontSize', 16);
```

F. 05. Object_stiffness_estimation_4

```
% Load your Force-Displacement data
t = Force_Displacement.time;
Displacement = double(squeeze(Force_Displacement.signals.values(:, 1))); % Displacement in mm
Force = double(squeeze(Force_Displacement.signals.values(:, 2)));        % Force in N

% Parameters for smoothing
window_size = 45; % Adjust based on noise level

% Initialize arrays to store stiffness values for loading and unloading
loading_stiffnesses = zeros(1, 5);
unloading_stiffnesses = zeros(1, 5);

% Prepare figure for subplots
figure;
for iteration = 1:5
    % Define the time bounds for each iteration
    start_time = (iteration - 1) * 60;
    end_time = start_time + 50; % Use a 50-second window for each iteration

    % Filter data for the current time window
    time_indices = t >= start_time & t < end_time;
    t_window = t(time_indices);
    disp_window = Displacement(time_indices);
    force_window = Force(time_indices);

    % Step 1: Smooth the data
    smooth_disp = movmean(disp_window, window_size); % Moving average smoothing
    smooth_force = movmean(force_window, window_size); % Smooth force signal

    % Step 2: Identify loading and unloading phases
    force_diff = diff(smooth_force); % Differentiate force to detect increasing or decreasing trends

    % Identify indices for loading (increasing force) and unloading (decreasing force)
    loading_indices = find(force_diff > 0);    % Increasing force
    unloading_indices = find(force_diff < 0);  % Decreasing force

    % Handle offset due to differentiation
    loading_indices = loading_indices(1:end-1);
    unloading_indices = unloading_indices(1:end-1);

    % Extract data for loading phase
    disp_load = smooth_disp(loading_indices);
    force_load = smooth_force(loading_indices);

    % Extract data for unloading phase
    disp_unload = smooth_disp(unloading_indices);
    force_unload = smooth_force(unloading_indices);

    % Step 3: Select Force Range [2N, 6N] for Loading and Unloading Curves
    min_force_threshold = 1.0; % Minimum force in N
    max_force_threshold = 6.0; % Maximum force in N
```

```
52
53      % Loading Curve: Find indices within the force range
54      force_in_range_load = (force_load >= min_force_threshold) & (force_load <=
            max_force_threshold);
55      disp_load_bounded = disp_load(force_in_range_load);
56      force_load_bounded = force_load(force_in_range_load);
57
58      % Unloading Curve: Find indices within the force range
59      force_in_range_unload = (force_unload >= min_force_threshold) & (force_unload <=
            max_force_threshold);
60      disp_unload_bounded = disp_unload(force_in_range_unload);
61      force_unload_bounded = force_unload(force_in_range_unload);
62
63      % Step 4: Perform Linear Regression on Bounded Regions
64      % Loading Curve
65      coeffs_bounded_load = polyfit(disp_load_bounded, force_load_bounded, 1); % [
            stiffness, offset]
66      stiffness_bounded_load = coeffs_bounded_load(1); % Slope = stiffness
67
68      % Unloading Curve
69      coeffs_bounded_unload = polyfit(disp_unload_bounded, force_unload_bounded, 1); %
            [stiffness, offset]
70      stiffness_bounded_unload = coeffs_bounded_unload(1); % Slope = stiffness
71
72      % Step 5: Generate Regression Lines for Bounded Regions
73      % Loading Curve Fit
74      disp_fit_bounded_load = linspace(min(disp_load_bounded), max(disp_load_bounded),
            100);
75      force_fit_bounded_load = polyval(coeffs_bounded_load, disp_fit_bounded_load);
76
77      % Unloading Curve Fit
78      disp_fit_bounded_unload = linspace(min(disp_unload_bounded), max(
            disp_unload_bounded), 100);
79      force_fit_bounded_unload = polyval(coeffs_bounded_unload,
            disp_fit_bounded_unload);
80
81      % Step 6: Plot Results in Subplots
82      subplot(2, 3, iteration);
83      hold on;
84      plot(smooth_disp, smooth_force, 'b');
85      plot(disp_load_bounded, force_load_bounded, 'ro');
86      plot(disp_unload_bounded, force_unload_bounded, 'go');
87      plot(disp_fit_bounded_load, force_fit_bounded_load, 'k-', 'LineWidth', 2);
88      plot(disp_fit_bounded_unload, force_fit_bounded_unload, 'k-', 'LineWidth', 2);
89
90      xlabel('Displacement (mm)', 'FontSize', 14);
91      ylabel('Force (N)', 'FontSize', 14);
92      title(sprintf('Iteration %d: Time [%d-%d]s', iteration, start_time, end_time), '
            FontSize', 16);
93      grid on;
94      xlim([0, 10.5]); % Set the x-axis limit from 0 to 10.5
95
96      % Add stiffness values as text annotations
97      text(0.1, 6, sprintf('Fit (L): K = %.4f N/mm', stiffness_bounded_load), '
            FontSize', 12, 'Color', 'k', 'VerticalAlignment', 'top');
98      text(0.1, 5.5, sprintf('Fit (UL): K = %.4f N/mm', stiffness_bounded_unload), '
            FontSize', 12, 'Color', 'k', 'VerticalAlignment', 'top');
```

```
99
100     hold off;
101
102     % Store the stiffness values for each iteration
103     loading_stiffnesses(iteration) = stiffness_bounded_load;
104     unloading_stiffnesses(iteration) = stiffness_bounded_unload;
105 end
106
107 % Calculate the mean and std of the stiffnesses
108 mean_loading_stiffness = mean(loading_stiffnesses);
109 std_loading_stiffness = std(loading_stiffnesses);
110 mean_unloading_stiffness = mean(unloading_stiffnesses);
111 std_unloading_stiffness = std(unloading_stiffnesses);
112
113 % Display the mean and standard deviation values in the command window
114 fprintf('Mean Loading: %.4f N/mm\n', mean_loading_stiffness);
115 fprintf('   Loading: %.4f N/mm\n', std_loading_stiffness);
116 fprintf('Mean Unloading: %.4f N/mm\n', mean_unloading_stiffness);
117 fprintf('   Unloading: %.4f N/mm\n', std_unloading_stiffness);
118
119 % Plot the mean stiffness values with 2-sigma error bars as a bar chart in the 6th
      plot
120 subplot(2, 3, 6);
121 hold on;
122
123 % Plot bars
124 bar(1, mean_loading_stiffness, 'FaceColor', 'b');
125 bar(2, mean_unloading_stiffness, 'FaceColor', 'c');
126
127 % Add error bars
128 errorbar(1, mean_loading_stiffness, 2 * std_loading_stiffness, 'r', 'LineWidth', 2,
      'CapSize', 10);
129 errorbar(2, mean_unloading_stiffness, 2 * std_unloading_stiffness, 'm', 'LineWidth',
      2, 'CapSize', 10);
130
131 % Add text annotations above the bars with larger font size
132 text(1, mean_loading_stiffness + 2 * std_loading_stiffness + 0.25, ...
133     sprintf('%.4f\n %.4f', mean_loading_stiffness, 2 * std_loading_stiffness), ...
134     'HorizontalAlignment', 'center', 'FontSize', 14); % Increased font size here
135 text(2, mean_unloading_stiffness + 2 * std_unloading_stiffness + 0.25, ...
136     sprintf('%.4f\n %.4f', mean_unloading_stiffness, 2 * std_unloading_stiffness),
        ...
137     'HorizontalAlignment', 'center', 'FontSize', 14); % Increased font size here
138
139 hold off;
140
141 % Add title and labels
142 title('Mean Stiffness Values (95% Confidence (2  ))', 'FontSize', 16);
143
144 % Assign the x-ticks and labels
145 ax = gca();
146 ax.XTick = [1, 2];
147 ax.XLim = [0, 3]; % Adjust X-limits to fit the new tick labels properly
148 ax.XTickLabel = {'Loading', 'Unloading'};
149 ax.TickLabelInterpreter = 'tex'; % Interpret newline in tick labels
150
151 % Increase font size of the x-axis tick labels
```

```
152 ax.XAxis.FontSize = 14;  % Adjust this value to your desired font size
153
154 % Add y-axis label
155 ylabel('Stiffness (N/mm)', 'FontSize', 14);
156 grid on;
157
158 % Set y-axis to start at 0
159 ylim([0, max([mean_loading_stiffness + 2 * std_loading_stiffness,
        mean_unloading_stiffness + 2 * std_unloading_stiffness]) + 0.5]);
160
161 % Add a main title for the entire figure
162 sgtitle('Smoothed Force-Displacement Curves of the Mechanism + 4 Parallel Springs,
        Measured by the Mechanism Itself with Stiffness Fits', 'FontSize', 16);
```

G. 05. Object_stiffness_estimation_legend_plot

```
% Load your Force-Displacement data
t = Force_Displacement.time;
Displacement = double(squeeze(Force_Displacement.signals.values(:, 1))); % Displacement in mm
Force = double(squeeze(Force_Displacement.signals.values(:, 2)));        % Force in N

% Parameters for smoothing
window_size = 45; % Adjust based on noise level

% Initialize arrays to store stiffness values for loading and unloading
loading_stiffnesses = zeros(1, 5);
unloading_stiffnesses = zeros(1, 5);

% Prepare figure for subplots
figure;
for iteration = 1:5
    % Define the time bounds for each iteration
    start_time = (iteration - 1) * 60;
    end_time = start_time + 50; % Use a 50-second window for each iteration

    % Filter data for the current time window
    time_indices = t >= start_time & t < end_time;
    t_window = t(time_indices);
    disp_window = Displacement(time_indices);
    force_window = Force(time_indices);

    % Step 1: Smooth the data
    smooth_disp = movmean(disp_window, window_size); % Moving average smoothing
    smooth_force = movmean(force_window, window_size); % Smooth force signal

    % Step 2: Identify loading and unloading phases
    force_diff = diff(smooth_force); % Differentiate force to detect increasing or decreasing trends

    % Identify indices for loading (increasing force) and unloading (decreasing force)
    loading_indices = find(force_diff > 0);    % Increasing force
    unloading_indices = find(force_diff < 0);  % Decreasing force

    % Handle offset due to differentiation
    loading_indices = loading_indices(1:end-1);
    unloading_indices = unloading_indices(1:end-1);

    % Extract data for loading phase
    disp_load = smooth_disp(loading_indices);
    force_load = smooth_force(loading_indices);

    % Extract data for unloading phase
    disp_unload = smooth_disp(unloading_indices);
    force_unload = smooth_force(unloading_indices);

    % Step 3: Select Force Range [2N, 6N] for Loading and Unloading Curves
    min_force_threshold = 1.0; % Minimum force in N
    max_force_threshold = 6.0; % Maximum force in N
```

```
52
53      % Loading Curve: Find indices within the force range
54      force_in_range_load = (force_load >= min_force_threshold) & (force_load <=
            max_force_threshold);
55      disp_load_bounded = disp_load(force_in_range_load);
56      force_load_bounded = force_load(force_in_range_load);
57
58      % Unloading Curve: Find indices within the force range
59      force_in_range_unload = (force_unload >= min_force_threshold) & (force_unload <=
            max_force_threshold);
60      disp_unload_bounded = disp_unload(force_in_range_unload);
61      force_unload_bounded = force_unload(force_in_range_unload);
62
63      % Step 4: Perform Linear Regression on Bounded Regions
64      % Loading Curve
65      coeffs_bounded_load = polyfit(disp_load_bounded, force_load_bounded, 1); % [
            stiffness, offset]
66      stiffness_bounded_load = coeffs_bounded_load(1); % Slope = stiffness
67
68      % Unloading Curve
69      coeffs_bounded_unload = polyfit(disp_unload_bounded, force_unload_bounded, 1); %
            [stiffness, offset]
70      stiffness_bounded_unload = coeffs_bounded_unload(1); % Slope = stiffness
71
72      % Step 5: Generate Regression Lines for Bounded Regions
73      % Loading Curve Fit
74      disp_fit_bounded_load = linspace(min(disp_load_bounded), max(disp_load_bounded),
            100);
75      force_fit_bounded_load = polyval(coeffs_bounded_load, disp_fit_bounded_load);
76
77      % Unloading Curve Fit
78      disp_fit_bounded_unload = linspace(min(disp_unload_bounded), max(
            disp_unload_bounded), 100);
79      force_fit_bounded_unload = polyval(coeffs_bounded_unload,
            disp_fit_bounded_unload);
80
81      % Step 6: Plot Results in Subplots
82      subplot(2, 3, iteration);
83      hold on;
84      plot(smooth_disp, smooth_force, 'b', 'DisplayName', ...
85          'Smoothed Data (N = 45)');
86      plot(disp_load_bounded, force_load_bounded, 'ro', 'DisplayName', 'Load Region');
87      plot(disp_unload_bounded, force_unload_bounded, 'go', 'DisplayName', 'Unload
            Region');
88      plot(disp_fit_bounded_load, force_fit_bounded_load, 'k-', 'LineWidth', 2, ...
89          'DisplayName', sprintf('Fit (Loading)'));
90      plot(disp_fit_bounded_unload, force_fit_bounded_unload, 'k-', 'LineWidth', 2,
            ...
91          'DisplayName', sprintf('Fit(Unloading)'));
92
93      xlabel('Displacement (mm)');
94      ylabel('Force (N)');
95      title(sprintf('Iteration %d: Time [%d-%d]s', iteration, start_time, end_time));
96      grid on;
97      legend('Location', 'Best', 'FontSize', 16); % Increased font size
98      xlim([0, 10.5]); % Set the x-axis limit from 0 to 10.5
99      hold off;
```

```
100
101     % Store the stiffness values for each iteration
102     loading_stiffnesses(iteration) = stiffness_bounded_load;
103     unloading_stiffnesses(iteration) = stiffness_bounded_unload;
104 end
105
106 % Calculate the mean and std of the stiffnesses
107 mean_loading_stiffness = mean(loading_stiffnesses);
108 std_loading_stiffness = std(loading_stiffnesses);
109 mean_unloading_stiffness = mean(unloading_stiffnesses);
110 std_unloading_stiffness = std(unloading_stiffnesses);
111
112 % Display the mean and standard deviation values in the command window
113 fprintf('Mean Loading: %.4f N/mm\n', mean_loading_stiffness);
114 fprintf('   Loading: %.4f N/mm\n', std_loading_stiffness);
115 fprintf('Mean Unloading: %.4f N/mm\n', mean_unloading_stiffness);
116 fprintf('   Unloading: %.4f N/mm\n', std_unloading_stiffness);
117
118 % Plot the mean stiffness values with 2-sigma error bars as a bar chart in the 6th
      plot
119 subplot(2, 3, 6);
120 hold on;
121
122 % Plot bars
123 bar(1, mean_loading_stiffness, 'FaceColor', 'b');
124 bar(2, mean_unloading_stiffness, 'FaceColor', 'c');
125
126 % Add error bars
127 errorbar(1, mean_loading_stiffness, 2 * std_loading_stiffness, 'r', 'LineWidth', 2,
      'CapSize', 10);
128 errorbar(2, mean_unloading_stiffness, 2 * std_unloading_stiffness, 'm', 'LineWidth',
      2, 'CapSize', 10);
129
130 % Add text annotations above the bars
131 text(1, mean_loading_stiffness + 2 * std_loading_stiffness + 0.1, ...
132     sprintf('%.4f\n %.4f', mean_loading_stiffness, 2 * std_loading_stiffness), ...
133     'HorizontalAlignment', 'center');
134 text(2, mean_unloading_stiffness + 2 * std_unloading_stiffness + 0.1, ...
135     sprintf('%.4f\n %.4f', mean_unloading_stiffness, 2 * std_unloading_stiffness),
          ...
136     'HorizontalAlignment', 'center');
137
138 hold off;
139
140 % Add title and labels
141 title('Mean Loading and Unloading Stiffness (with 95% Confidence (2  ))');
142
143 % Assign the x-ticks and labels
144 ax = gca();
145 ax.XTick = [1, 2];
146 ax.XLim = [0, 3];  % Adjust X-limits to fit the new tick labels properly
147 ax.XTickLabel = {'Loading', 'Unloading'};
148 ax.TickLabelInterpreter = 'tex'; % Interpret newline in tick labels
149
150 % Add y-axis label
151 ylabel('Stiffness (N/mm)');
152 grid on;
```

```
153
154 % Set y-axis to start at 0
155 ylim([0, max([mean_loading_stiffness + 2 * std_loading_stiffness,
        mean_unloading_stiffness + 2 * std_unloading_stiffness]) + 0.2]);
156
157 % Add a main title for the entire figure
158 sgtitle('Smoothed Force-Displacement Curves of the Mechanism + 2 Parallel Springs,
        Measured by the Mechanism Itself with Stiffness Fits', 'FontSize', 14);
```

H. 06. Plot_mechanism_stiffness_bars

```
% Define the stiffness values
calibration_loading = 0.0498;        % Mechanism Loading becomes Calibration Loading
mechanism_loading = 0.0589;          % Calibration Loading becomes Mechanism Loading
calibration_unloading = 0.042;       % Mechanism Unloading becomes Calibration
    Unloading
mechanism_unloading = 0.0673;       % Calibration Unloading becomes Mechanism
    Unloading

% Define the standard deviations for mechanism values (remains the same)
std_mechanism_loading = 0.004066285;  % Standard deviation for mechanism (Loading)
std_mechanism_unloading = 0.00344066; % Standard deviation for mechanism (Unloading)

% Calculate    2 sigma values
two_sigma_loading = 2 * std_mechanism_loading;
two_sigma_unloading = 2 * std_mechanism_unloading;

% Create figure and hold for multiple bars
figure;
hold on;

% Plot the bars with individual colors, swapping mechanism and calibration
bar(1, mechanism_loading, 'FaceColor', 'b');   % Mechanism (Loading) -> Bar 1
bar(2, calibration_loading, 'FaceColor', 'c'); % Calibration (Loading) -> Bar 2
bar(4, mechanism_unloading, 'FaceColor', 'm'); % Mechanism (Unloading) -> Bar 4
bar(5, calibration_unloading, 'FaceColor', 'r'); % Calibration (Unloading) -> Bar 5

% Add error bars for mechanism values
errorbar(1, mechanism_loading, std_mechanism_loading, 'k', 'LineWidth', 1.5, '
    CapSize', 10);
errorbar(4, mechanism_unloading, std_mechanism_unloading, 'k', 'LineWidth', 1.5, '
    CapSize', 10);

% Add labels above the error bars for mean and    2 sigma values
text(1, mechanism_loading + std_mechanism_loading, ...
     sprintf('%.4f\n  %.4f', mechanism_loading, two_sigma_loading), ...
     'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 12);
text(2, calibration_loading, sprintf('%.4f', calibration_loading), ...
     'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 12);
text(4, mechanism_unloading + std_mechanism_unloading, ...
     sprintf('%.4f\n  %.4f', mechanism_unloading, two_sigma_unloading), ...
     'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 12);
text(5, calibration_unloading, sprintf('%.4f', calibration_unloading), ...
     'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 12);

% Define the first row of x-axis labels
row1 = {'Mechanism', 'Calibration', 'Mechanism', 'Calibration'};

% Define the second row for "Loading" and "Unloading"
row2 = {'Loading', 'Loading', 'Unloading', 'Unloading'};

% Combine the rows of labels into a cell array
labelArray = [row1; row2];

% Combine the rows of labels into individual tick labels
tickLabels = strtrim(sprintf('%s\\newline%s\n', labelArray{:}));
```

```
52
53 % Set the x-axis tick positions and labels
54 ax = gca;
55 ax.XTick = [1, 2, 4, 5];
56 ax.XTickLabel = tickLabels;
57 ax.TickLabelInterpreter = 'tex'; % Needed to interpret newline
58 ax.FontSize = 12; % Increase font size for x-axis tick labels
59
60 % Add labels and title with increased font size
61 ylabel('Stiffness (N/mm)', 'FontSize', 14);
62 title('Comparison of Mechanism and Calibration Stiffness', 'FontSize', 14);
63
64 % Display grid
65 grid on;
66
67 % Adjust the x-axis limits to add space around bars
68 xlim([0.5, 5.5]);
69
70 % Adjust the y-axis limits
71 ylim([0, max([mechanism_loading + std_mechanism_loading, ...
72               calibration_loading, ...
73               mechanism_unloading + std_mechanism_unloading, ...
74               calibration_unloading]) + 0.01]);
75
76 hold off;
```

I. 07. Mechanisme_eigen_stijfheid_fout_percentage_bar_plots

```
% Define the stiffness values (Calibration and Mechanism)
calibration_loading = 0.0498;        % Calibration Loading
mechanism_loading = 0.0589;          % Mechanism Loading
calibration_unloading = 0.042;       % Calibration Unloading
mechanism_unloading = 0.0673;       % Mechanism Unloading

% Define the standard deviations for mechanism values
std_mechanism_loading = 0.004066285;  % Standard deviation for mechanism (Loading)
std_mechanism_unloading = 0.00344066; % Standard deviation for mechanism (Unloading)

% Calculate    2 sigma values for error bars
two_sigma_loading = 2 * std_mechanism_loading;
two_sigma_unloading = 2 * std_mechanism_unloading;

% Calculate the percentage errors (absolute values) for each
percent_error_loading = abs((mechanism_loading - calibration_loading) /
    calibration_loading) * 100;
percent_error_unloading = abs((mechanism_unloading - calibration_unloading) /
    calibration_unloading) * 100;

% Create figure and hold for multiple bars
figure;
hold on;

% Plot the bars with individual colors (for loading and unloading)
bar(1, percent_error_loading, 'FaceColor', 'b');   % Mechanism Loading -> Bar 1
bar(2, percent_error_unloading, 'FaceColor', 'c'); % Mechanism Unloading -> Bar 2

% Add labels above the error bars for the percentage error values (without standard
    deviation)
text(1, percent_error_loading + 0.5, sprintf('%.2f%%', percent_error_loading), ...
    'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 12);
text(2, percent_error_unloading + 0.5, sprintf('%.2f%%', percent_error_unloading),
    ...
    'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 12);

% Define the x-axis labels
x_labels = {'Loading', 'Unloading'};

% Set the x-axis tick positions and labels
ax = gca;
ax.XTick = [1, 2];
ax.XTickLabel = x_labels;
ax.FontSize = 12; % Increase font size for x-axis tick labels

% Add labels and title
ylabel('Percent Difference (%)', 'FontSize', 14);
title({'Percentual Differences for Loading and Unloading', ...
       'Stiffness (Absolute) for the Mechanism''s Intrinsic',...
       'Stiffness, Compared to the Calibration Data'}, 'FontSize', 14);

% Display grid
grid on;

% Adjust the x-axis limits to add space around bars
```

```
52 xlim([0.5, 2.5]);
53
54 % Adjust the y-axis limits based on the data
55 ylim([0, max([percent_error_loading + two_sigma_loading, ...
56                percent_error_unloading + two_sigma_unloading]) + 5]);
57
58 hold off;
```

J. 08. Create_spring_stiffness_barplots

```
% Define the stiffness values (2 springs)
loading_stiffness = 0.8432;
unloading_stiffness = 0.8699;
average_stiffness = 0.85655;
calibration_stiffness = 0.822;

%% Define the stiffness values (3 springs)
% loading_stiffness = 1.1852;
% unloading_stiffness = 1.0836;
% average_stiffness = 1.1344;
% calibration_stiffness = 1.233;

%% Define the stiffness values (4 springs)
% loading_stiffness = 1.637;
% unloading_stiffness = 1.6168;
% average_stiffness = 1.6269;
% calibration_stiffness = 1.644;

% Create the bar plot with individual bars
figure;
hold on;

% Plot each bar with its respective color
bar(1, loading_stiffness, 'FaceColor', 'b', 'DisplayName', 'Loading Stiffness'); % Blue
bar(2, unloading_stiffness, 'FaceColor', 'c', 'DisplayName', 'Unloading Stiffness'); % Cyan
bar(3, average_stiffness, 'FaceColor', 'm', 'DisplayName', 'Average Stiffness'); % Magenta
bar(4, calibration_stiffness, 'FaceColor', 'r', 'DisplayName', 'Calibration Stiffness'); % Red

% Add stiffness values below each bar
text(1, loading_stiffness, sprintf('%.4f', loading_stiffness), ...
    'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 14);
text(2, unloading_stiffness, sprintf('%.4f', unloading_stiffness), ...
    'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 14);
text(3, average_stiffness, sprintf('%.4f', average_stiffness), ...
    'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 14);
text(4, calibration_stiffness, sprintf('%.4f', calibration_stiffness), ...
    'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 14);

% Set the x-axis tick labels with font size 12
set(gca, 'XTick', 1:4, 'XTickLabel', {'Loading', 'Unloading', 'Average', 'Calibration'}, 'FontSize', 13);

% Add labels and title
ylabel('Stiffness (N/mm)', 'FontSize', 14);
title('Comparison of Stiffness Values for 2 Springs', 'FontSize', 14);

% Display grid and legend
grid on;
% legend('Location', 'Best');

% Adjust the y-axis limits to make room for the text labels
```

```
51 ylim([0, max([loading_stiffness, unloading_stiffness, average_stiffness,
       calibration_stiffness]) + 0.1]);
52
53 hold off;
```

K. 09. Create_spring_stiffness_barplots

```
% Define the stiffness values (2 springs)
loading_stiffness = 0.8432;
unloading_stiffness = 0.8699;
average_stiffness = 0.85655;
calibration_stiffness = 0.822;

%% Define the stiffness values (3 springs)
% loading_stiffness = 1.1852;
% unloading_stiffness = 1.0836;
% average_stiffness = 1.1344;
% calibration_stiffness = 1.233;

%% Define the stiffness values (4 springs)
% loading_stiffness = 1.637;
% unloading_stiffness = 1.6168;
% average_stiffness = 1.6269;
% calibration_stiffness = 1.644;

% Create the bar plot with individual bars
figure;
hold on;

% Plot each bar with its respective color
bar(1, loading_stiffness, 'FaceColor', 'b', 'DisplayName', 'Loading Stiffness'); % Blue
bar(2, unloading_stiffness, 'FaceColor', 'c', 'DisplayName', 'Unloading Stiffness'); % Cyan
bar(3, average_stiffness, 'FaceColor', 'm', 'DisplayName', 'Average Stiffness'); % Magenta
bar(4, calibration_stiffness, 'FaceColor', 'r', 'DisplayName', 'Calibration Stiffness'); % Red

% Add stiffness values below each bar
text(1, loading_stiffness, sprintf('%.4f', loading_stiffness), ...
    'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 14);
text(2, unloading_stiffness, sprintf('%.4f', unloading_stiffness), ...
    'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 14);
text(3, average_stiffness, sprintf('%.4f', average_stiffness), ...
    'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 14);
text(4, calibration_stiffness, sprintf('%.4f', calibration_stiffness), ...
    'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'center', 'FontSize', 14);

% Set the x-axis tick labels with font size 12
set(gca, 'XTick', 1:4, 'XTickLabel', {'Loading', 'Unloading', 'Average', 'Calibration'}, 'FontSize', 13);

% Add labels and title
ylabel('Stiffness (N/mm)', 'FontSize', 14);
title('Comparison of Stiffness Values for 2 Springs', 'FontSize', 14);

% Display grid and legend
grid on;
% legend('Location', 'Best');

% Adjust the y-axis limits to make room for the text labels
```

```
51 ylim([0, max([loading_stiffness, unloading_stiffness, average_stiffness,
       calibration_stiffness]) + 0.1]);
52
53 hold off;
```

L. 10. Signal_plotter

```
% close all;

t = PWM.time;
PWM_data = double(squeeze(PWM.signals.values));
S_data = double(squeeze(S.signals.values));
W_data = double(squeeze(W.signals.values));

% Plot
figure;

subplot(2,2,1)
plot(t, PWM_data);

subplot(2,2,2)
plot(t, S_data);

subplot(2,2,3)
plot(t, W_data);

%% Plot
% figure;
%
% subplot(2,2,1)
% plot(t(80*50:81*50), PWM_data(80*50:81*50));
%
% subplot(2,2,2)
% plot(t(80*50:81*50), S_data(80*50:81*50));
%
% subplot(2,2,3)
% plot(t(80*50:81*50), W_data(80*50:81*50));
```