Design and Control of an Under-actuated Gripper Module for Interaction with a Remote Environment

L.W. (Wilbert) van de Ridder

BSc Report

Committee:
Dr. R. Carloni
Dr.ir. M. Fumagalli
Dr. H.K. Hemmes

March 2015

009RAM2015
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands
CONTENTS

I Acknowledgments 1

II Introduction 2

III Previous & Related work 3

IV Model 3
   IV-A Definitions & assumptions ........................................... 3
   IV-B Kinematic model ............................................................ 3
   IV-C Static model ................................................................. 4
   IV-D Simulations ................................................................. 5
   IV-E Stiffness DoF invariance .................................................. 6

V Mechanical design 7
   V-A mVSA .............................................................................. 7
   V-B Gripper ............................................................................. 7

VI Methods 8
   VI-A Experimental setup ............................................................ 8
   VI-B Control and calibration ......................................................... 8
   VI-C Force measurement experiments ........................................... 8

VII Discussion 10
   VII-A Difficulties and common errors ........................................... 10
   VII-B Force behavior ................................................................. 10

VIII Conclusions & Recommendations 11
   VIII-A Conclusions ..................................................................... 11
   VIII-B Recommendations ........................................................... 11

IX Future work 11

References 11

X Appendix 12
   X-A Technical drawings: Gripper .................................................... 12
      X-A1 Design iterations ............................................................... 13
   X-B Technical drawings: mVSA .................................................... 18
   X-C Electronics: Interface board ................................................... 20
   X-D Electronics: Force sensor interface ........................................... 21
   X-E Software: Overview ............................................................... 22
   X-F Software: Simulink interface .................................................... 23
   X-G Software: Motion profile generation ........................................... 24
   X-H Software: Magnetic switch center determination ....................... 29
   X-I Software: Simulink serial connector ........................................... 32
   X-J Software: Memory efficient logger ............................................ 33
   X-K Software: State manager ......................................................... 34
   X-L Software: PID Controller ......................................................... 35
   X-M mVSA calibration ................................................................... 36
Abstract

Service tasks on large buildings or industrial structures, e.g. inspection for structural integrity of bridges or industrial plants, window cleaning of skyscrapers, usually entail laborious and expensive support structures and hazardous working environments. Recent developments on UAVs interacting with a remote environment offer many new possibilities, leading to service robotics becoming airborne.

Within the AIRobots project, a European project in the field of innovative aerial service robots, a robotic manipulator has been developed to endow a UAV for interactive inspection tasks. A versatile end-effector has been designed to make full use of the broad range of possible applications the system can offer. Several task specific modules have been proposed and elaborated on.

In this work, a redesign of an under-actuated gripper module is carried out, whose preliminary conceptual design has been proposed in a previous project. Mechanical design of the gripper, modeling, control and a full construction of the system is realized. Moreover, in conjunction with the gripper module, a redesign of the mVSA-UT (miniaturized variable stiffness actuator) is conducted to adapt the actuator to add a second non-compliant output shaft for a complete integration of the whole system.
I. ACKNOWLEDGMENTS

First and foremost I would like to thank Raffaella Carloni and Matteo Fumagalli for their help and patience. Additionally, on the RaM group I would like to acknowledge Marcel Schwitz and Gerben te Riet o/g Scholten for technical advice and insights. Éamon Barret for fruitful discussions and his help with the mVSA. From the study Advanced Technology I would like to mention and thank Marijke Stehouwer for being my study adviser and her guidance, and Herman Hemmes for his willingness to be a part of my committee.

From the people close to me I would like to express my gratitude to Susan Borchers, my girlfriend, for her love and reinforcement. My parents, brother and sisters for believing in me. Close friends I would like to thank and mention is the Commodus group, in particular Tsjibbe Wiersma, my good friend and business partner. My house-mates from Huize Oktoknus, particularly Peter-Jan Vos, Bas Klok and Rolf de Jong.

At last but not least I want to acknowledge my boss and colleagues from Fortes Labs and Fortes Solutions for their support and flexibility.
II. Introduction

Recent developments on UAVs interacting with a remote environment offer many new possibilities, leading to service robotics becoming airborne. As part of the AIRobots [1] project, a robotic manipulator (Figure 3a) has been developed which extends the ability of a UAV (Figure 1) for interactive tasks using a versatile end-effector.

One of the proposed task-specific modules compatible with the end-effector is an under-actuated gripper. Under-actuation is of specific interest due to the numerous advantages in terms of size, mechanical / electrical complexity, weight, and cost while providing the ability to explore a large array of grasping techniques.

A novel under-actuated gripper design (see Figure 2) was proposed by T. Bartelds [4]. His study shows the possibility to influence the ratio of forces exerted by the phalanges on an object. Manipulation of the forces is done by using a spring whose stiffness can be changed; indicating the need for a variable stiffness mechanism as proposed by T. Bartelds; see Figure 4. For this purpose the mVSA-UT [5] (Figure 3b) will be used which is a miniature variable stiffness actuator developed at the Robotics and Mechatronics (RaM) group at the University of Twente.

The goal of this project is to integrate the gripper with the mVSA compatible with the robotic manipulator (Figure 3a). The gripper is mechanically redesigned to integrate the variable stiffness mechanism to allow for force tuning on the phalanges. Additionally, modeling, control, and a full construction of the system (Figure 5) is realized.

The report is organized as follows; an overview of the related and previous work is given in section III. Section IV discusses the model and simulations. Section V shows the design of the gripper and supporting components. Section VI describes the methods used. The following section, VII, discusses the results of the experiments. In the last section, conclusion and recommendations are given followed by suggested future work. Finally, an appendix is attached including schematics, CAD files and code examples that are not included in the report itself.
III. PREVIOUS & RELATED WORK

The four-bar linkage gripper design as proposed by T. Bartelds, based on designs discussed in [7] and [8], is the result of a comparison between various under-actuated gripper approaches and has led to the creation of an early proof of concept as shown in Figure 2.

R. Tummers conducted a study, as an extension to [3], in which the aforementioned gripper design has been extended to allow for opening and closing of the fingers using bowden cables [9]. This work adapted the gripper from T. Bartelds to be compatible with the versatile end-effector (Figure 3a).

Since compliant actuation is key to stable and save interaction research has led to the development of various compliant actuation systems. Most significant for this project is the work done on the miniature variable stiffness actuator (mVSA) [5] shown in Figure 3b. The mVSA is a miniaturized version of the vsaUT-II [10] [11].

IV. MODEL

![Upwards spring and Downwards spring](image)

Fig. 6: Studied spring configurations.

To aid in the gripper design and gain insight in its behavior a kinematic and static model has been developed. This is by no means a novel kinematic and static model but an extension on the work by T. Bartelds [4] adapted to design requirements.

In this extended model two spring configurations have been studied. The first configuration denotes the upward spring configuration, see Figure 6a. The other configuration considers a downward setup shown in Figure 6b which has already been investigated by T. Bartelds.

The dynamics of the system have not been studied. Additionally, only a static situation is considered assuming a planar grasp on a cylindrical object. Consequently, results are only valid as long as a full-contact situation ($F_1, F_2 > 0$) is depicted. The extension also encompasses additional design parameters, most notably $s$ (spring fixture distance) and $\theta_{off}$ which indicates the angle between the pulley mount and the compliant tendon fixture; these parameters allow defining a configuration in which the compliance is DoF invariant. This is further discussed in section IV-E. The resulting gripper schematic is shown in Figure 7.

The design process of the gripper was carried out for the upward spring configuration and as a result most attention is given to this setup. The whole model is shown here, for completeness and understanding, with its additions.

A. Definitions & assumptions

The definitions and assumptions have not changed with regards to the original model. The range of motion is constrained:

$$\theta_d \in [0, 0.5\pi] \text{ and } \theta_p \in [-\frac{1}{4}\pi, \frac{1}{4}\pi]$$  \hspace{1cm} (1)

B. Kinematic model

A contact model is considered where the proximal phalanx is always in contact with an object. The two degrees of freedom $\theta_p$ and $\theta_d$ correspond to the opening angle of the proximal and distal phalanx. A fixed circular object with radius $r_{obj}$ is placed with its center at the symmetry line of the gripper.

Both the opening angle $\theta_p$ of the proximal phalanx and the angle of the distal phalanx $\theta_d$ are fixed for a given object size $r_{obj}$ by Eq. (2) and Eq. (3) respectively when a full contact situation is considered.

$$\theta_p = 2 \cdot \arctan \left( \frac{r_{obj}}{L_0} \right) - \frac{\pi}{2}$$  \hspace{1cm} (2)

$$\theta_d = \pi - 2 \cdot \arctan \left( \frac{r_{obj}}{L_1 - L_0} \right)$$  \hspace{1cm} (3)

$$\angle 5 = \arccos \left( \frac{c^2 + L_2^2 - e^2}{2 \cdot c \cdot L_2} \right)$$  \hspace{1cm} (4)

$$\angle 4 = \pi - \angle 5 + \theta_d$$  \hspace{1cm} (5)

$$d = \sqrt{L_1^2 + c^2 - 2 \cdot L_1 \cdot c \cdot \cos(\angle 4)}$$  \hspace{1cm} (6)

$$\angle 2 = \arccos \left( \frac{a^2 + b^2 - d^2}{2 \cdot a \cdot b} \right)$$  \hspace{1cm} (7)

$$\angle 1_p = \arccos \left( \frac{L_2^2 + d^2 - c^2}{2 \cdot L_1 \cdot d} \right)$$  \hspace{1cm} (8)

$$\angle 1_a = \arccos \left( \frac{a^2 + d^2 - b^2}{2 \cdot a \cdot d} \right)$$  \hspace{1cm} (9)
\[ \angle 1 = \angle 1_a + \angle 1_p \]  

(10)

\[ \angle 3_b = \arccos \left( \frac{b^2 + d^2 - a^2}{2 \cdot b \cdot d} \right) \]  

(11)

\[ \angle 3_c = \arccos \left( \frac{c^2 + d^2 - L_1^2}{2 \cdot c \cdot d} \right) \]  

(12)

\[ \angle 3 = \angle 3_b + \angle 3_c \]  

(13)

\[ \angle 6 = \arccos \left( \frac{\mathbf{u} \cdot \mathbf{v}}{\| \mathbf{u} \| \| \mathbf{v} \|} \right) \]  

(14)

For the upward configuration, where the tendon is fixed to a point on the pulley, \( \mathbf{u} \) and \( \mathbf{v} \) are given by:

\[
\mathbf{u} = \left[ (a + f) \cdot \left\{ \frac{\cos(\beta_1)}{\sin(\beta_1)} \right\} - d_2 \cdot \left\{ \frac{\cos(\beta_2)}{\sin(\beta_2)} \right\} \right]
\]  

(15)

\[
\mathbf{v} = \left[ a \cdot \left\{ \frac{\cos(\beta_3)}{\sin(\beta_3)} \right\} - d \cdot \left\{ \frac{\cos(\beta_4)}{\sin(\beta_4)} \right\} \right] \]

(16)

\[
\beta_1 = \angle 1 + \theta_p + \theta_{off} \]

\[
\beta_2 = \theta_p + \angle 1_{cs} \]

\[
\beta_3 = \theta_p + \angle 1 \]

\[
\beta_4 = \theta_p + \angle 1_p \]

(17)

where:

\[
d_2 = \sqrt{L_1^2 + (c + s)^2 - 2 \cdot L_1 \cdot (c + s) \cdot \cos(\angle 4)} \]

(18)

\[
\angle 1_{cs} = \arccos \left( \frac{L_1^2 + d_2^2 - (c + s)^2}{2 \cdot L_1 \cdot d_2} \right) \]

(19)

Regarding the downward configuration no specific relation is considered since no design concept has been created to impose constraints, \( \angle 6 = -\angle 4 \alpha_{upward} \cdot 2 \) rad and \( s = 0 \) mm is used in the simulations.

C. Static model

A static model is derived from the free body diagram shown in figure 9. \( F_a \) is the result of torque \( \tau_a \) and is given by:

\[ F_a = \frac{\tau_a}{a} \cdot \sin (\theta_p + \angle 1) \]  

(20)

Summing the moments around the proximal joint \( pj \) results in a relation to determine \( F_b \):

\[ \sum M_{pj} = F_a \cdot a \cdot \sin (\theta_p + \angle 1) - F_b \cdot a \cdot \sin \angle 2 \]  

(21)

The spring force \( F_s \) is defined by the stiffness \( k \) (which can be dynamically set in a range \([0, \infty]\) using the mVSA) and the pre-tension angle \( \theta_i \):

\[ F_s = k \cdot \theta_i \]  

(22)

The moments around distal joint \( dj \) are summed to produce a relation for \( F_2 \):

\[ \sum M_{dj} = F_b \cdot c \cdot \sin(\angle 3) - F_s \cdot (c + s) \cdot \sin (\angle 3 + \angle 6) - F_2 \cdot (L_1 - L_0) \]  

(23)

Force equilibrium for \( F_{r,x} \) and \( F_{r,y} \) result in:

\[
\begin{align*}
\{ F_{r,x} \\ F_{r,y} \} + \sum_{i=b,s} F_i \cdot \left\{ \frac{\cos(\alpha_i)}{\sin(\alpha_i)} \right\} &= 0
\end{align*}
\]

(24)

Where the angles are defined as:

\[
\alpha_b = \theta_p + \angle 1 + \angle 2 + \pi
\]

\[
\alpha_s = \theta_p + \angle 1 + \angle 2 - \angle 6
\]

\[
\alpha_2 = \theta_p - \theta_d + \frac{\pi}{2}
\]

(25)

Finally \( F_1 \) can be determined by summing the moments around the proximal joint \( pj \):

\[ \sum M_{pj} = F_{r,y} \cdot L_1 \cdot \cos(\theta_p) - F_{r,x} \cdot L_1 \cdot \sin(\theta_p) - F_1 \cdot L_0 = 0 \]  

(26)
D. Simulations

The goal is to allow for force tuning in the phalanges of the gripper. Consequently, simulations are performed using Matlab (v2014a, Mathworks Inc., Natick, USA) to gain insight in the force (ratio) behavior. First the contact forces are calculated and shown in Figure 10 and 11. Subsequently, the force ratio $F_2/F_1$ is presented in Figure 12 and 13. Finally, a simulation is carried out to show the behavior of the forces in time when the stiffness is changed; the experiments are conducted using this scenario (see Figure 14 and 15).

The following parameters are used:

- $a = 10.7\, \text{mm}$
- $b = 27.0\, \text{mm}$
- $c = 5.0\, \text{mm}$
- $e = 24.4\, \text{mm}$
- $f = 3.0\, \text{mm}$
- $s = 1.5709799\, \text{mm} \lor 0$ (upward $\lor$ downward configuration)
- $L_0 = 15.0\, \text{mm}$
- $L_1 = 30.0\, \text{mm}$
- $L_2 = 20.8\, \text{mm}$
- $r_{\text{obj}} = 20.0\, \text{mm}$
- $\theta_t = \frac{1}{8}\pi\, \text{rad}$
- $\theta_{\text{off}} = 0.090403\, \text{rad}$

The downward configuration has already been discussed in a previous project but is added for comparison [4]. Comparing Figure 10 and 11 it is apparent that the spring configuration heavily influences the force behavior. In the upward form changing the stiffness at constant torque decreases both forces while the same action in the downward form cause the proximal phalanx to exert more force while the force on the distal phalanx diminishes. The behavior of the force on the proximal phalanx $F_2$ is identical in both setups while the behavior of $F_1$ is determined by the design. As a consequence the force ratio behavior is different to the same extend as depicted in Figure 12 and 13.

Additionally, Figure 14 and 15 show the behavior in time when the stiffness is changed. This particular situation is recreated in the experiments and used for model validation. Simulations and previous work show that it is possible to adjust the force ratio exerted by the phalanges. Additionally, it shows that the force behavior can be altered using different spring configurations.
E. Stiffness DoF invariance

In order for the spring force to be invariant of $\theta_d$ and $\theta_p$, it is required that for any combination of these two degrees of freedom the ratio of the tendon movements $s_{nonC}$ (non-compliant tendon length) and $s_c$ (compliant tendon length) is constant. The ratio between $F_1$ and $F_2$ can be adjusted as long as $\angle 6$ is non-zero. If $\angle 6 = 0$ the value of $F_b$ is merely diminished for $F_s > 0N$.

Two design parameters have been chosen for which values can be calculated such that the movement ratio, between the non-compliant and compliant tendon, is constant: $s$ and $\theta_{off}$. The method is to set up a system of equations for the tendon movements $s_{nonC}$ and $s_c$ as a function of $\theta_p$ and $\theta_d$ with adjustable parameters $s$ and $\theta_{off}$:

\[
\begin{bmatrix}
    s_c(\theta_1,\theta_2) \\
    s_{nonC}(\theta_2,\theta_d) \\
    s_c(\theta_2,\theta_d) \\
    s_{nonC}(\theta_2,\theta_d)
\end{bmatrix}
= \begin{bmatrix}
    a+f \\
    \frac{a}{a+f} \\
    \frac{a}{a+f} \\
    \frac{a}{a+f}
\end{bmatrix}
\]

(27)

$s_{nonC}$ and $s_c$ are defined as:

\[s_{nonC} = ((\pi - (\angle 1 + \theta_p)) \cdot a) - s_{nonC\_initial}\]

(28)

\[\angle total = \angle 1 + \theta_p + \theta_{off}\]

\[p_x = \cos(\angle total) \cdot (a + f)\]

\[p_y = \sin(\angle total) \cdot (a + f)\]

\[ph_x = d_2 \cdot \cos(\theta_p + \angle 1_{cs})\]

\[ph_y = d_2 \cdot \sin(\theta_p + \angle 1_{cs})\]

\[d_1 = \sqrt{(ph_x-p_x)^2 + (ph_y-p_y)^2}\]

\[s_c = ((\pi - \angle total) \cdot (a + f)) + d_1 - s_{c\_initial}\]

(29)

$p_x$ and $p_y$ are the coordinates for the tendon attachment to the pulley. $ph_x$ and $ph_y$ are the coordinates for the tendon attachment to the distal phalanx. $d_1$ is the tendon length between $p_{x,y}$ and $ph_{x,y}$. $s_{nonC\_initial}$ and $s_{c\_initial}$ are defined as the tendon lengths at $\theta_p = 0.25rad$ and $\theta_d = 0rad$.

The system from Eq. (27) is numerically solved using vpasolve method available in Matlab using $\theta_p1 = 0$, $\theta_d1 =$
Solutions obtained using the aforementioned parameters are $s = 1.570979 \text{mm}$ and $\theta_{off} = 0.090403 \text{ rad}$. To verify these values the tendon ratios are computed for most relevant configurations (shown in Figure 16) using the calculated values for $s$ and $\theta_{off}$. The results (Table I) show an error margin $< 1\%$ w.r.t. to the desired fraction $\frac{\pi + f}{a} = 1.280374$ and thus deemed valid.

### Table I: Tendon movement ratios for different angles of $\theta_p$ and $\theta_d$ for the upward spring configuration. All cases are depicted in Figure 16.

<table>
<thead>
<tr>
<th>Angle</th>
<th>$\theta_p = 0.1$</th>
<th>$\theta_p = -0.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_d = 0$</td>
<td>1.280374</td>
<td>1.280374</td>
</tr>
<tr>
<td>$\theta_d = 0.25$</td>
<td>1.280321</td>
<td>1.280345</td>
</tr>
<tr>
<td>$\theta_d = 0.5$</td>
<td>1.278679</td>
<td>1.279279</td>
</tr>
</tbody>
</table>

V. MECHANICAL DESIGN

All CAD design has been done using SolidWorks (v2014, SolidWorks Corp, Waltham, USA). In this section the (re)design of the mVSA, mVSA housing and gripper is discussed.

### A. mVSA

A minor redesign of the mVSA [5] has been carried out to add a non-compliant (stiff) output aside from the already existing compliant shaft (Figure 17). The stiff output is required to open and close the phalanges.

The mVSA housing is based on original design w.r.t. to tendon fixture by R. Tummers [9]. Additionally, a rotational encoder slot is created to be able to measure the output deflection on the compliant output. The tendons connecting the gripper to the mVSA are routed through bowden cables. The non-compliant and compliant pulleys are designed such that a radii ratio of $\frac{a}{f}$ is realized in accordance to the pulley mechanism in the gripper.

### B. Gripper

Using the work carried out by T. Bartelds [4] and R. Tummers [9] as a starting point, a gripper design has been realized (Figure 19) using the parameters calculated in the simulations. All iterations in gripper design and mechanical drawings are located in the appendix in Section X-A. All parts except for axes, bearings, nuts, and bolts have been printed in 3D. Nuts and bolts as well as part interlocking proved necessary to make the system strong enough due to the relatively weak nature of the 3D printed material. The tendons are made of Kevlar to prevent elongation and mitigate unwanted compliant behavior.

Special attention has been given to the routing of the non-compliant and compliant tendons. The non-compliant tendon routing is depicted in Figure 20. Due to the asymmetric distribution of the fingers, 1 versus 2, the compliant tendon is connected to all distal phalanges after being routed by a pulley system. This pulley system is schematically depicted in Figure 21. The implementation of this pulley system is shown in Figure 22 and 23 and is a result of careful spatial considerations. Most notably being the length of the entire gripper (from mount to fingers) keeping in mind it is meant to be attached to a UAV off-center (Figure 1). The tendons are attached to the distal phalanges using adjustable nails.
The upper phalanges are fitted with FSR-400 (Interlink Electronics, Canada) force sensitive resistors. These force sensors are connected by a voltage divider to the Arduino (schema available in Section X-D). The measuring resistor $R_m$ on the voltage divider is chosen to maximize the desired force sensitivity range and to limit current.

The tendons fitted on the gripper are connected to the mVSA using bowden cables which also allow for proper tensioning.

A hardware interface board has been created to allow for easy manual control of the system (see Section X-C).

### B. Control and calibration

For the mVSA setpoints are generated using a trapezoidal motion profile (discussed in X-G). From these angular setpoints the motor steps are calculated from the mVSA transfer function and controlled using a manually tuned PID controller.

A state manager is used to control switching between startup, calibration, manual control, and Simulink interface states.

Calibration of the open and closed position of the gripper is performed by slowly opening and closing until the endpoints are detected using software methods.

### C. Force measurement experiments

During the force experiments a ping-pong ball wound in isolation tape (Figure 28), to provide a more stable contact between the force sensors and the ball, is placed between the fingers. Force is applied by the mVSA to ensure a full-contact situation. Finally, the compliant output stiffness is changed to mimic the simulated cases for comparison and validation.

The measurements (notably the output voltage $V_{out}$) were captured in Matlab and processed to calculate the actual forces exerted on the sensors. The variable resistance of the FSR is related to the measured input voltage by:

$$V_{out} = \frac{V_{cc} \cdot R_m}{R_m + R_{fs}}$$

Where $R_m = 30\,\text{k}\Omega$ and $V_{cc} = 5V$. The FSR resistance in terms of these parameters is:

$$R_{fs} = \frac{(V_{cc} - V_{out}) \cdot R_m}{V_{out}}$$

From the calibration data provided by the manufacturer a very close approximation has been calculated using Microsoft Excel invoking a power fit resulting in the following relationship:

$$F = 296852 \cdot \frac{R_{fs}^{-0.825}}{98.1} \, (N)$$

Fig. 21: Schematic view of compliant tendon connection.

Fig. 22: Gripper compliant tendon routing (side view).

Fig. 23: Gripper compliant tendon routing (top view).

VI. METHODS

The next section describes the experimental setup, control and calibration, force experiments, and processing of data. First the experimental setup is described followed by the control and calibration. Finally, the experiments and data processing are discussed.

**A. Experimental setup**

A schematic overview of the experimental setup is shown in Figure 24). Matlab Simulink is installed on a computer and connected via serial to an Arduino Mega 2560 revision 3 (Arduino SA) using a newly developed serial connection library. For Simulink, an interface has been created which allows for real-time adjustments of parameters and data acquisition.

The Arduino prototyping board is fitted with a AIRobots interface shield and running a continuous calculation loop at 500Hz. A significant amount of software has been developed for the Arduino board during this project. Where possibly all reusable parts of the software have been re-factored into stand-alone libraries and have been published as open-source on Github. The software is discussed in detail in the appendix.

The mVSA is fitted with two instances of a Pololu 50:1 Micro Metal Gearmotor HP with Extended Motor Shaft (Pololu Corporation, USA) which actually have a transmission ratio of 51.45:1. The motors of the mVSA are connected to the AIRobots interface shield using Pololu High-Power Motor Drivers 18v15. The mVSA is configured with gears resulting in the following transfer function:

$$
\begin{bmatrix}
\phi_{\text{nonC}} \\
\phi_{C}
\end{bmatrix} =
\begin{bmatrix}
0.25 \cdot \phi_{m0} + 0.375 \cdot \phi_C \\
-\phi_{\text{nonC}} - 0.706 \cdot \phi_{m1}
\end{bmatrix}
$$

(30)
Fig. 24: Schematic overview of the experimental setup. Sensors are indicated by orange hexagons. (1) Simulink interface. (2) Serial connection. (3) Arduino Mega 2560 + AIRobots connectivity-shield connected to mVSA motors using Pololu high power motor drivers. (4) Motor position sensors. (5) mVSA motor housing. (6) mVSA Pivot magnetic switch. (7) mVSA. (8) mVSA housing. (9) Sensor holder. (10) mVSA Compliant output deflection sensor. (11) mVSA Frame position magnetic switch. (12) mVSA non-compliant output tendon pulley. (13) mVSA compliant output tendon pulley. (14) Gripper. (15) Proximal phalanx force sensor ($F_1$). (16) Distal phalanx force sensor ($F_2$). (17) Interface board.

Fig. 25: Experimental setup overview.

Fig. 26: mVSA + housing.

Fig. 27: Gripper.

Fig. 28: Gripper showing FSR attachment.
VII. Discussion

In this section the results from the experiments are reviewed. At first, encountered difficulties and errors are considered. Secondly and finally, the measurements are presented and discussed.

A. Difficulties and common errors

Friction is a significant factor in the system. This can be attributed to the non-compliant tendon thickness and stiffness, usage of bowden cables for both the non-compliant and compliant tendons. While friction in bowden cables is usually minor, it does have influence on forces on this scale. Additionally, the 3D printed material has a relatively high coefficient of friction. This has been minimized as much as possible by the usage of bearings, nylon spacers and specific tolerances to reduce self-contact.

Matlab Simulink proved to be fairly unstable in 64-bit mode on Microsoft Windows. The system crashed on seemingly random moments. Many times the experiments had to be rerun due to crashes. Issues were reported on-line with this type of setup using the 64-bit version of Matlab in combination with the Real-time target features and is reportedly subject to investigation likely to be fixed and/or mitigated in subsequent releases of Matlab.

The effects of symmetrical or asymmetrical amount of fingers configuration has not been investigated. The compliant tendon pulley system is designed to make it invariant of the amount of fingers but this has not been experimentally verified.

Every time the experiment was conducted there were minor differences in the initial compliant offset. This was because the rotational sensor to measure the compliant output deflection was not used due to technical difficulties and time constraints. The offset was manually applied and although closely approximated not guaranteed to be the same every time.

Force sensitive resistors (FSR) are known to be inaccurate w.r.t. repeatability, temperature influences, and inhomogeneous loading issues. It is a cheap method of force sensing but careful calibration is required to get somewhat accurate results. In these experiments the calibration values from the manufacturer is used, which are only a rough approximation.

B. Force behavior

Experimental results are shown in Figure 29, 30 and 31. Most importantly the behavior seen in the simulations regarding the effect of increasing stiffness causing a decrease in forces is equally apparent in the experiments as well. The repeatability within a single experiment is apparent from Figure 31.

A consistent lag can be observed in all experiments. The lag is mainly attributed to the use of a motion profile generator (which limits the acceleration and speed) and significant friction in the system.

The force ratio, as simulated, expects a larger range of the $F_2$ values compared to $F_1$. The experiments revealed identical behavior in the first two shown cases. The third case seems to indicate a relatively equal range of both forces. This may be caused by inaccurate pretensioning of the compliant output.
VIII. CONCLUSIONS & RECOMMENDATIONS

A. Conclusions

A mechanical redesign of the gripper with the addition of a variable spring has been realized. Additionally, an existing kinematic and static model have been extended and verified based on the new design. The mVSA has been redesigned adding a stiff output besides the already existing compliant output. Result is a complete integration between the gripper and mVSA, and control and construction of the whole system. The model, which behavior is verified by experiments, shows that it is possible to tune both the forces and force ratio exerted by the phalanges of an under-actuated gripper.

B. Recommendations

The current force sensing setup using a ball gives too many problems in the sense that it is hard to ensure a homogeneous force distribution on the sensor surfaces. Other problems were compliance, friction, and slip on the ball. The FSR (force sensitive resistor), while cheap and easy to use, requires careful placement. An improved setup is depicted in Figure 32.

![Fig. 32: Force sensing recommendation](image)

Additionally, it may be worth looking into alternative force sensing techniques which may involve more sophisticated sensors.

The current pulley system, especially for the compliant tendon, needs a lot of space. A setup which uses tendons/cable guides instead of the current pulley system may be a more stable solution and will probably make the gripper smaller and more reliable. Figure 33 shows a concept.

![Fig. 33: Gripper compliant tendon routing proposal](image)

The 3D printed material has noticeable limits on strength, weight, and tolerances which will make further miniaturization hard to carry out. Other materials may be required and can be investigated.

IX. FUTURE WORK

A different force control method is required in order to set a desired force (ratio) on the phalanges. The series elastic actuator approach may aid in developing alternative control methods [13].

Currently, only a symmetric non-compliant shape has been investigated. Other shapes and control methods are required to allow for adaptive gripping on uneven, rugged, compliant and undulating surfaces typically found in the natural world. Additionally, different grasping techniques (for example pinching) can be researched by extending the current setup. Implications of the geometric choices (e.g. the upward and downward spring configurations) should be more carefully examined. It may be interesting to find out which configuration is favorable over another considering different scenarios.

The development of a 20-sim model to study dynamics can be investigated and may provide more insight. There is already a 20-sim model available for the mVSA and for the gripper created by R.Tummers. These can be integrated which may be relatively easy since the current gripper 20-sim model contains a static spring which can be changed to a variable version. The model is currently setup for a spring configuration in which it is attached to two joints in a downward configuration.

REFERENCES

X. APPENDIX

A. Technical drawings: Gripper

This section contains an iteration overview and relevant technical drawings for the gripper. The final CAD model is shown in Figure 34. All design iterations leading up to the final model are shown in section X-A1 accompanied by the most important mechanical drawings of the final design.

All CAD files are available at the Robotics and Mechatronics group at the University of Twente.

Fig. 34: Gripper
1) Design iterations:

(1) Original design by Teun Bartelds [4].

(2) Improved design by Ramon Tummers [9].

(3) Iteration #1

(4) Iteration #2

(5) Iteration #3

(6) Iteration #4

(7) Iteration #5

(8) Iteration #6

(9) Iteration #7

(10) Iteration #8
Tendon pulley block

FILE / PART NAME: SidePulleys_Full
B. Technical drawings: mVSA

The original mVSA design has been adapted to support a non-compliant output shaft. Differences are shown in Figure 36. The mechanical drawing for the updated frame is added at the end of this section.

![Diagram of mVSA extended non-compliant output. Shaft highlighted blue in a and b](image)

**Fig. 36**: mVSA extended non-compliant output. Shaft highlighted blue in a and b
C. Electronics: Interface board

A simple interface has been designed which has the ability to indicate the current active state using LED’s, discrete input using two buttons and analogue input using two potentiometers.

Fig. 37: Realization of the interface board.

Fig. 38: Schema of the interface board.
D. Electronics: Force sensor interface

A simple interface, shown in Figure 39, between the Arduino board and force sensors.

Fig. 39: Schema of the force sensor interface
E. Software: Overview

A significant amount of software has been developed for this project to run on the Arduino Mega 2560 board. Worth noting is the usage of the AVR-STL\(^1\) library on the Arduino board allowing for easier memory management and data handling.

Where possible, the software was re-factored into standalone libraries and published as open-source on Github if they were considered useful to others. In the following sections it will be explicitly noted if the source code is publicly available.

\(^{1}\)http://andybrown.me.uk/wk/2011/01/15/the-standard-template-library-stl-for-avr-with-c-stream
F. Software: Simulink interface

A dedicated Matlab Simulink interface has been created to allow for easy parameter adjustments, logging, control and data acquisition. It is connected to the Arduino board using the Matlab real-time target functionality. Serial communication on the Arduino is handled by a newly created library discussed in Section X-I.

Fig. 40: Simulink interface
**G. Software: Motion profile generation**

Source code and documentation is available on Github.²

A motion-profile generator has been designed to calculate the position setpoints for the mVSA motors in a predictable less-jerky manner. This method creates a trapezoidal motion profile to reach a given setpoint while adhering to a given maximum velocity and acceleration. The generator is able to both calculate a complete path beforehand and generating it on the fly based on the current position and velocity.

The trapezoidal method has first been developed in Matlab (see Figure 41) and once it worked, implemented and experimentally verified on the system (mVSA) in C++ (see Figure 43-46). Additionally, a constant velocity profile method has been created and implemented as well (see Figure 42). The actual velocity measured in the experiments has been smoothed using a Gaussian kernel for easier comparison with the simulated velocity.

---

²[https://github.com/WRidder/MotionProfileGenerator](https://github.com/WRidder/MotionProfileGenerator)

---

Fig. 41: Simulation of trapezoidal motion for given setpoints. Maximum velocity is set at 0.15\(m/s\) and the maximum acceleration is set at 0.1\(m/s^2\).
Fig. 42: Simulation of a constant velocity profile for given setpoints. Maximum velocity is set at 0.15 m/s.
**Fig. 43:** Experiment and model verification using a constant setpoint.

**Fig. 44:** Experiment and model verification using a variable setpoint.
Fig. 45: Experiment and model verification using a variable setpoint.

Fig. 46: Experiment and model verification using a variable setpoint.
Listing 1: MotionProfile usage example

```c
#include "MotionProfile.h"

/**
 * Initialization
 *
 * @param int aVelocityMax maximum velocity (units/s)
 * @param int aAccelerationMax maximum acceleration (units/s^2)
 * @param short aMethod method of profile generation (1 = trapezoidal)
 * @param int aSampleTime sample time (ms)
 */
MotionProfile trapezoidalProfile = new MotionProfile(200, 100, 1, 10);

/**
 * Usage
 */
float finalSetpoint = 1000;
float setpoint = trapezoidalProfile->update(finalSetpoint);

// Check if profile is finished
if (trapezoidalProfile->getFinished()) {};

// Reset internal state
trapezoidalProfile->reset();
```
**H. Software: Magnetic switch center determination**

Magnetic switches are used to calibrate the mVSA frame and pivot position. These switches exhibited two types of behavior: single and double dip. A method was created which recognizes and determines the center point in both cases. The method was first developed and tested in Matlab and once verified implemented in C++ for usage on the Arduino microcontroller.

In the Matlab code, first the transition points are detected using `getTransitionPoints` and this is used as input for the `findCenter` function which determines the actual center. In the C++ code a single function handles all calculations.

![Fig. 47: Single dip simulation for the three possible cases. Green dots denote the transition points. Red dots denote the determined center.](image1)

![Fig. 48: Double dip simulation for the five possible cases. Green dots denote the transition points. Red dots denote the determined center.](image2)
Listing 2: getTransitionPoints.m

```matlab
% Transition point function
function R = getTransitionPoints(A, default)
    previous = default;
    R = [];
    % Loop values to find transition points
    for i = 1:size(A,1)
        if A(i, 2) ~= previous
            % Check if this is the first entry, if so, it's not a change
            if i == 1
                previous = A(1,2);
            else
                R(end + 1) = A(i,1);
                previous = A(i,2);
            end
        end
    end
end
```

Listing 3: findCenter.m

```matlab
% Center function
function center = findCenter(A, length)
    count = size(A,2)
    dist = zeros(1, count)
    % Calculate all distances
    for i = 1:count
        if i < count
            dist(1, i) = abs(A(i) - A(i + 1));
        else
            % Last value, wrap around
            dist(1, i) = abs(A(i) - length - A(1));
        end
    end
    % Get smallest distance
    [val, locn] = min(dist)
    % Calculate center
    center = mod(A(locn) + round(val / 2), length);
end
```

Listing 4: findCenter.cpp

```cpp
/*
 * FUNCTION: findCenter
 * Determines the amount of free ram available
 * @param v vector containing the transition points.
 * @param length total amount of points (e.g. 360 if using degrees)
 * E.g. [10, 20, 50, 100], 120 will return 15
 * @return center point
 */
int findCenter (std::vector<int>& b, int length, bool modulus = true) {
    std::vector<int> distancePoints;
    // Calculate all distances
    for(std::vector<int>::size_type i = 0; i != b.size(); i++) {
        if (i < b.size() - 1) {
            distancePoints.push_back(abs(b[i] - b[i + 1]));
        } else {
            // Last value, wrap around
            distancePoints.push_back(abs(b[i] - length - b[0]));
        }
    }
    // Get the smallest distance
    int smallestIndex = 0;
    int smallestValue = INT_MAX;
    for(std::vector<int>::size_type j = 0; j != distancePoints.size(); j++) {
        if (distancePoints[j] < smallestValue) {
            smallestValue = distancePoints[j];
            smallestIndex = j;
        }
    }
    // Calculate center
    int center = mod(A(locn) + round(val / 2), length);
```
smallestValue = distancePoints[j];
smallestIndex = j;
}

// Calculate the center
// If the modulus produces wrong/weird results, see: ...
int returnValue = (b[smallestIndex] + round(smallestValue / 2));

if (modulus) {
    returnValue %= length;
}
return returnValue;
I. Software: Simulink serial connector

Source code and documentation is available on Github.¹

The simulink serial connection library has been created due to both the lack of a standard approach and as a training exercise to learn about serial communication protocols.

Listing 5: SimulinkConnector usage example

```cpp
#include "SimulinkConnector.h"

/**
 * Initialization
 */

/* The output format accepts the following types:
   * l = long
   * ul = unsigned long
   * u = unsigned int
   * l or d = int
   * Multiple datatypes can be used at once, however,
   * the sendPacket() function only accepts long types.
   * You can use casting to transmit the other datatypes.
*/

/*
   * @param aOutputFormat char string containing the output format; e.g. "S %d %l %ul E".
   * @param aInputPacketVector vector to which serial input will be written
   * @param aOutputPacketVector vector from which output serial is composed
   * @param aOutputInterval sample time of serial output in milliseconds
*/

std::vector<long> outputPacketVector(5, 0);
// Configured an output of 5 long values from Matlab Simulink

std::vector<long> receivedPacketVector(5, 0);
SimulinkConnector simulinkConnection("S %l %l %l %l %l E", receivedPacketVector, ... outputPacketVector, 20);

/**
 * Usage
 */

// Arduino loop
simulinkConnection.update();

// Set output variables (at any point in the loop or supporting function):
outputPacketVector[0] = 99;

// Read input variables (this is always accessible after initialization):
int incomingValue = receivedPacketVector[2] // Read third value of incoming data vector
```

¹https://github.com/WRidder/Arduino-SimulinkConnector
This library is a fork from the Arduino logging library. Support has been added for the FlashStringHelper data type which enables the logging of strings (which can take up considerable space, especially when adding a lot of log statements) saved to the flash program space memory (PROGMEM) instead of the scarce SRAM. This enables us to add a significant amount of log statements while almost no scarce SRAM is needed.

Listing 6: Logging usage example

```c
#include <Logging.h>

// Initialization

const int LOG_LEVEL = LOG_LEVEL_INFOS;
const long BAUD_RATE = 115200; // Serial transmission rate in bits per second;
Log.Init(LOG_LEVEL, BAUD_RATE);

// Usage

// Debug logging, not shown since current log level is LOG_LEVEL_INFOS
Log.Debug(F("Pin %d selected as debug output pin\n"), 1);

// Log info, is shown since log level is LOG_LEVEL_INFOS
Log.Info(F("******************************************\n"));
Log.Info(F("Setup arduino complete\n"));
Log.Info(F("Initiating first stage...\n"));
Log.Info(F("******************************************\n"));
```

4https://github.com/WRidder/Arduino-FlashStringHelperLogger
5https://github.com/mrRobot62/Arduino-logging-library
K. Software: State manager

Source code and documentation is available on Github.6

Manager library for an existing state library.7 Enables easy usage with Arduino.

Listing 7: Statemanager usage example

```cpp
#include "Statemanager.h"

/**
 * Initialization
 *
 * @param byte [buttonStateSwitchPin] pin number of button that switches the states.
 * @param byte stateVector Vector containing all initial states.
 */
const State stateArr[] = {
    State(STATE_IDLE_ENTER, STATE_IDLE_UPDATE, STATE_IDLE_EXIT),
    State(STATE_MVSACALIBRATION_UPDATE),
    State(STATE_GRIPPERSIMULINK_MODE1_ENTER, STATE_GRIPPERSIMULINK_MODE1_UPDATE, ...
        STATE_GRIPPERSIMULINK_MODE1_EXIT),
    State(STATE_GRIPPERSIMULINK_MODE1_ENTER, STATE_GRIPPERSIMULINK_MODE1_UPDATE, ...
        STATE_GRIPPERSIMULINK_MODE1_EXIT)
};

std::vector<State> statesVector (stateArr, stateArr + sizeof(stateArr) / sizeof(stateArr[0]) );
StateManager stateMngr = StateManager(BUTTON_STATESWITCH_PIN, statesVector);

/**
 * Usage
 */
void loop()
{
    // Check states (check which one is active and run the update function)
    stateMngr.checkStates();
}

// Transition to next state
stateMngr.transitionToNextState();
```

6https://github.com/WRidder/Arduino-StateManager
7Finite state machine: http://playground.arduino.cc/code/FiniteStateMachine
L. Software: PID Controller

Source code and documentation is available on Github.\(^8\)

This PID library only supports the long data-type, preventing floating point operations, in order to speed up setpoint calculations. This is a fork from the original Arduino PID library. Additionally, some small features like error margins and obstruction detection have been added.

Listing 8: PID usage example

```c
#include "PID_Extended_Controller.h"

/**
 * Initialization
 * *
 * @param long P proportional gain
 * @param long I proportional gain
 * @param long D proportional gain
 * @param int Dir direction
 */

PIDController pid = PIDController(1, 0.1, 0.02, 0);

/**
 * Usage
 */

// Set control variables
long* InputAngle;
long* Output;
long* Setpoint;

InputAngle = new long(0.0);
Output = new long(0.0);
Setpoint = new long(0.0);

pid.SetControlVariables(long* InputAngle, long* Output, long* Setpoint)

// Update input variables
InputAngle = sensorReading();

// Compute PID output
pid.Compute();

// Use output
setMotorDutyCycle(*Output);
```

\(^8\)https://github.com/WRiddler/Arduino-PID-NoFloatingPoint
**M. mVSA calibration**

The mVSA is calibrated in two steps. First the frame position is determined by slowly opening and closing the fingers and using software methods for end-point detection. Secondly, the compliant output is calibrated using magnetic switches in the housing of the mVSA.

The full calibration routine is depicted in Figure 49.