12-2023

A modular framework for surface-embedded actuation and optical sensing in soft robots.

Paul Bupe Jr
University of Louisville

A MODULAR FRAMEWORK FOR SURFACE-EMBEDDED ACTUATION AND OPTICAL SENSING IN SOFT ROBOTS

By
Paul Bupe Jr
B.S., Georgia Southern University, 2013
M.S., Georgia Southern University, 2015
M.S., Georgia Southern University, 2020

A Dissertation
Submitted to the Faculty of the
J.B. Speed School of Engineering of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy
in Electrical Engineering

Department of Electrical and Computer Engineering
University of Louisville
Louisville, Kentucky

December 2023

A MODULAR FRAMEWORK FOR SURFACE-EMBEDDED ACTUATION AND OPTICAL SENSING IN SOFT ROBOTS

By

Paul Bupe Jr
B.S., Georgia Southern University, 2013 M.S.,
Georgia Southern University, 2015 M.S.,
Georgia Southern University, 2020

Dissertation approved on

October 30, 2023

by the following Dissertation Committee:

Dissertation Director
Cindy Harnett

John Naber

Tamer Inanc

Adrian Lauf

DEDICATION

This dissertation is dedicated to my wife, Dr. Gregorie Bupe. Her love, support, and belief in me have been, and continue to be, a constant source of strength and motivation. I could not have accomplished this without her.

Additionally, I dedicate this work to my parents and siblings, who have provided faithful and selfless support throughout my educational journey, from elementary school to the completion of my PhD.

ACKNOWLEDGMENTS

I am immensely grateful to my advisor, Dr. Cindy Harnett, for her invaluable guidance and expertise throughout this research journey. We had the unique experience of starting this research in the midst of the COVID-19 pandemic and her experience, innovative thinking, and flexibility were crucial in sustaining the progress of the research. I certainly do not take for granted, and I am very grateful for, the opportunity to have worked under her guidance, which has been invaluable to my learning and growth. My appreciation also goes to the members of my committee for their time, very helpful feedback, and support. Additionally, I would like to acknowledge the financial support from the National Science Foundation (Award #1935324) and the SCRAM consortium that allowed me to focus fully on my research.

Finally, my deep appreciation goes to all who contributed to making my time in Louisville so memorable, with special thanks to the Nau family.

ABSTRACT

A MODULAR FRAMEWORK FOR SURFACE-EMBEDDED ACTUATION AND OPTICAL SENSING IN SOFT ROBOTS

Paul Bupe Jr

October 30, 2023

This dissertation explores the development and integration of modular technologies in soft robotics, with a focus on the OptiGap sensor system. OptiGap serves as a simple, flexible, cost-effective solution for real-time sensing of bending and deformation, validated through simulation and experimentation. Working as part of an emerging category of soft robotics called Soft, Curved, Reconfigurable, Anisotropic Mechanisms, or SCRAMs, this research also introduces the Thermally-Activated SCRAM Limb (TASL) technology, which employs shape-memory alloy (SMA) wire embedded in curved sheets for surface actuation and served as the initial inspiration for OptiGap. In addition, the EneGate system is presented as a complementary technology that aims to provide modular actuation control and sensing in soft robotic applications. Designed to integrate seamlessly with thermal actuators and OptiGap sensors, EneGate utilizes a custom communication protocol to achieve a high degree of modularity. This dissertation demonstrates how these technologies collectively contribute to a more flexible, scalable, and adaptable future for soft robotics. It goes into the design specifics, communication protocols, and potential applications, offering a comprehensive modular solution for both sensing and control in soft robotics.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1 Problem Statement

The field of robotics has seen significant advancements in recent years, particularly in the development of soft robots. These robots offer a range of advantages over their rigid counterparts, including adaptability, safety, and the ability to navigate complex environments. However, they also present unique challenges in terms of sensing and actuation. Traditional sensors and actuators are often ill-suited for soft robotic applications, leading to a gap in the technology needed to fully realize the potential of soft robots. One of the key challenges lies in the real-time sensing of bending and deformation, which is crucial for control and navigation. Existing solutions often lack the lack modularity, flexibility, and cost-effectiveness required for practical applications in soft robotics. This problem is further exacerbated when the robots are required to operate in specialized conditions such as underwater. This work aims to address some of these challenges through the development of modular technologies, with a focus on the OptiGap Sensor System, a fully realized modular optical sensing system.

2 Research Scope and Objectives

This research contributes to an emerging category of soft robotics, specifically Soft, Curved, Reconfigurable, Anisotropic Mechanisms, or SCRAMs, which was Initiated in 2019 by a consortium of four universities and financially supported by the National Science Foundation. At the core of this technology lies the ability to alter

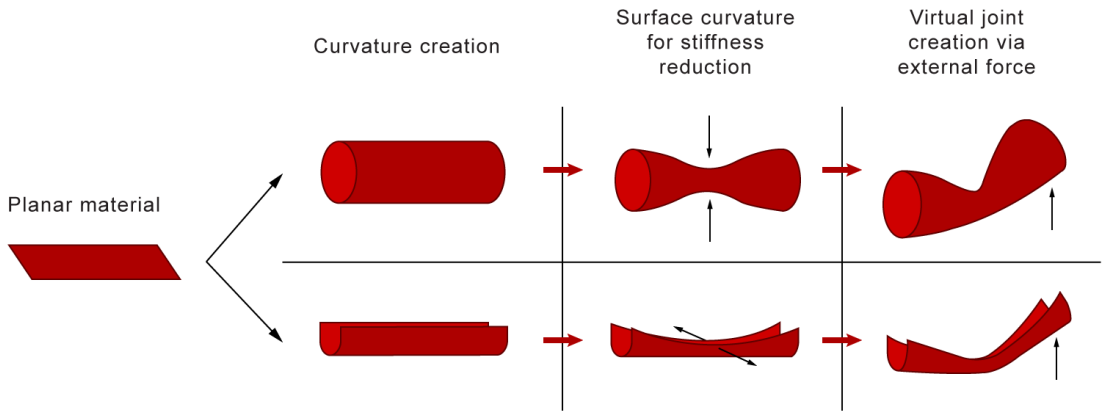


Figure 1. Overview of SCRAM Concept

the mechanical properties of thin-walled structures by modulating their local curvature, demonstrated in Figure 1. This feature enables SCRAM robots to function as continuum, compliant, and configurable soft robotic systems. One of the primary objectives of the SCRAM initiative is to broaden the conventional understanding of soft robots by incorporating flexible, non-stretching materials into their design. This expansion of the material selection allows for a more versatile range of applications and functionalities, thus pushing the boundaries of what is traditionally considered a soft robot.

The SCRAM project also focuses on introducing soft robotics technology that can adapt its shape and actuation in a reconfigurable manner. This is achieved by leveraging the intrinsic coupling between curvature and mechanical behavior in planar materials. By understanding and exploiting this relationship, the project aims to design, model, and control reconfigurable soft robots with novel capabilities. Another significant aspect of SCRAM technology is the development of localized sensing and actuation strategies, facilitated by techniques such as fabrics, sewing, and embroidery. These localized strategies enable more precise control and adaptability, making SCRAM robots highly suitable for complex tasks and environments. In addition, the SCRAM project is committed to advancing planar fabrication methods along

Table 1. SCRAM Research Publications

Title	Authors	Year
Embedded Optical Waveguide Sensors for Dynamic Behavior Monitoring in Twisted-Beam Structures	**Bupe et al.**	**In review**
Multimodal Locomotion in a Soft Robot Through Hierarchical Actuation	Yu, Qifan et al.	2023
OptiGap: A Modular Optical Sensor System for Bend Localization	**Bupe et al.**	**2023**
Strained Elastic Surfaces with Adjustable-Modulus Edges (SESAMEs) for Soft Robotic Actuation	Kimmer et al.	2023
Tunable Dynamic Walking via Soft Twisted Beam Vibration	Jiang, Yuhao et al.	2023
Development, Modeling, and Testing of a Passive Compliant Bistable Undulatory Robot	Kwan et al.	2023
Electronically Reconfigurable Virtual Joints by Shape Memory Alloy-Induced Buckling of Curved Sheets	**Bupe et al.**	**2022**
Characterizing the pressure response of microstructured materials for soft optical skins	Portaro et al.	2022
Multitouch Pressure Sensing With Soft Optical Time-of-Flight Sensors	Lin et al.	2022
Reconfigurable Curved Beams for Selectable Swimming Gaits in an Underwater Robot	Sharifzadeh et al.	2021
Curvature-Induced Buckling for Flapping-Wing Vehicles	Sharifzadeh et al.	2021
Collective Synchronization of Undulatory Movement through Contact	Zhou et al.	2021
Flexoskeleton Fingers: 3D Printed Reconfigurable Ridges Enabling Multi-functional and Low-cost Underactuated Grasping	Yu et al.	2021
Reconfigurable laminates enable multifunctional robotic building blocks	Jiang, Mingsong et al.	2021
Vacuum induced tube pinching enables reconfigurable flexure joints with controllable bend axis and stiffness	Jiang, Mingsong et al.	2021
Shape Change Propagation Through Soft Curved Materials for Dynamically-Tuned Paddling Robots	Jiang, Yuhao et al.	2021
Low attenuation soft and stretchable elastomeric optical waveguides	Uppal et al.	2021
Reconfigurable Soft Flexure Hinges via Pinched Tubes	Jiang, Yuhao et al.	2020
Flexoskeleton Printing Enables Versatile Fabrication of Hybrid Soft and Rigid Robots	Jiang, Mingsong et al.	2020
Absolute Length Sensor Based on Time of Flight in Stretchable Optical Fibers	Lin et al.	2020

with the integration of local, embedded actuators and sensors. These advancements are crucial for achieving the high degree of reconfigurability and adaptability that SCRAM robots are designed for.

3 Modularity in Robotics

Central to the research presented in this work is the focus on modularity and sensing in SCRAM robotics, with the OptiGap Sensor System serving as a key example of modular sensing technology. OptiGap was initially inspired by the development of the Thermally-Activated SCRAM Limb, or TASL. It has since evolved into a fully developed and tested system that addresses the need for real-time, flexible, and cost-effective sensing in SCRAMs and other robotic systems. EneGate, another technology under development, aims to further extend the modular capabilities of both the OptiGap and TASL technologies.

Modularity in soft robotics, as discussed by Paik [2], is a critical aspect of the OptiGap and EneGate projects. This approach involves the use of modular design principles for actuators, sensors, materials, and control systems, allowing for easy customization and adaptation to different tasks. The development of modular soft robotics enhances the adaptability and functionality of soft robots, making them suitable for a variety of applications [2]. For example, the concept of the 'soft LEGO', as proposed by Liao & Chen [3] decomposes complex systems into concise, controllable modular units, which is a principle central to OptiGap and EneGate. The modularization allows for reconfiguration and customization, which is essential for the dynamic environments these robots operate in [4].

Pigozzi & Medvet [4] explored the use of voxel-based soft robots (VSRs) for rugged terrain locomotion, demonstrating the versatility of modular design in adapting to challenging environments. This aligns with the aims of OptiGap and EneGate, where flexibility in design and function is crucial. Zhang et al. emphasized the motion

capability of soft robots and how their modular methodology aids in creating reconfigurable and flexible prototypes [5], a concept that is foundational to OptiGap's sensor system and EneGate's control mechanism. Zhang et al. [6] also presented a comprehensive review of modular soft robots, providing insights into the challenges and future directions of intelligent modular soft robots (MSRs). This review is particularly relevant to the development of OptiGap and EneGate, as it covers various modular units' materials, fabrication, actuation, sensors, and control aspects.

In the realm of actuation, Jin et al. presented vacuum-driven soft actuators with different origami skins, highlighting the diversity in modular actuation methods [7]. This diversity is crucial for the development of adaptable robotic systems like OptiGap and EneGate. Zhang et al. further demonstrated the practical application of modular design in rapidly assembling flexible robotic manipulators [8], aligning with the aims of the OptiGap and EneGate projects in creating adaptable and modular robotic systems.

4 Objectives

This research focuses on the following primary objectives aligned with the SCRAM project:

1. **Development of Planar Fabrication Methods for Embedded Actuators**
   The first objective aims to introduce the use of untrained and electronically controlled shape-memory alloy (SMA) wire as a surface actuator via tailored wire placement techniques to create compliant joints or induce buckling, leading to the Thermally-Activated SCRAM Limb, or TASL. Traditional actuators are often bulky, rigid, and not well-suited for the dynamic, flexible structures found in soft robotics. The use of untrained SMA wire as a surface actuator offers a lightweight, flexible and, when used as a momentary actuator, energy-efficient alternative. Tailored wire placement techniques are employed to create compli-

ant joints or induce buckling in the robot's structure, which involves the use of a ZSK tailored wire placement machine to sew SMA wire into a fabric material. The development of these new actuation techniques will be validated through experimentation, providing a comprehensive understanding of their capabilities and limitations.

2. **Localized Sensing and Actuation Strategies** The second objective focuses on addressing an important issue in soft robotics: real-time sensing. Central to this objective is the development of the OptiGap Sensor System. The OptiGap sensor system uses air gaps in flexible optical light pipes to create a low-cost and flexible sensor that can detect bending while being effectively mechanically transparent to the device being sensed. The system has been validated through simulation and experimentation, supporting the approach of using an air gap with a flexible sleeve to create bend-sensitive air gap patterns in a light pipe. The development of this sensor system also involves testing under various conditions, including underwater, to ensure its robustness and reliability.

3. **Integration and System Development** The third objective seeks to integrate the developed actuation and sensing technologies into a cohesive, modular system aligned with SCRAM applications. This includes circuit designs, algorithm development, and control firmware and software. Enegate, a technology still under development, aims to extend the modular capabilities in control and actuation, inspired by the level of modularity demonstrated by OptiGap.

5 Significance

The research presented in this work is significant for several reasons, primarily because it addresses key challenges in the field of soft robotics through the advancement of the new SCRAM class of soft robots. Central to this advancement is the OptiGap

sensor system, which offers a simple and modular approach to real-time sensing of bending and deformation. OptiGap's modular design allows for easy integration with existing systems while being materially transparent and scalable for various applications, making it a versatile solution. Another important aspect of this research is the development of new surface actuation techniques using SMA wire. The use of thermal actuators integrated into planar bistable structures offers an energy-efficient alternative to traditional SMA actuators, which are often energy-intensive. This energy efficiency is particularly important as it allows the robot to maintain its shape without the need for continuous power, thereby reducing both operational time and environmental impact. Finally, EneGate aims to provide a cohesive, modular system for both sensing and actuation, further enhancing the capabilities of soft robotic systems. The modular design of these technologies also enhances their scalability, enabling the creation of lightweight structures well-suited for distributed actuation and sensing, with the compatibility of OptiGap with modern microcontrollers expanding its range of applications and making it more accessible for individual researchers and even enterprises. Overall, the research contributes significantly to the field of soft robotics by offering a modular, scalable, and energy-efficient approach to sensing and actuation. Along with the SCRAM consortium, it sets a new direction for how soft robotic systems can be designed, integrated, and deployed, offering solutions that are both practical and innovative.

CHAPTER II

DEVELOPMENT OF SURFACE ACTUATION USING UNTRAINED SMA WIRE

1 Introduction

This chapter delves into the foundational technology that inspired the development of the OptiGap sensor system: the use of shape-memory alloy (SMA)-based surface actuation to modify the local curvature of curved, thin-walled, inextensible continuum sheets in order to create reconfigurable virtual joints. A thin flat material like paper can be stiffened by uniformly curving it along a single axis into a U-shape, as shown in Figure 2. This curved shape exhibits anisotropic properties in that it resists bending along the curve due to increased stiffness in the remaining axes. The stiffness of this structure can be altered by creating a small change in shape along the curve such as creating a flat spot which results in buckling, and thus reduced stiffness, under an external force [9, 10].

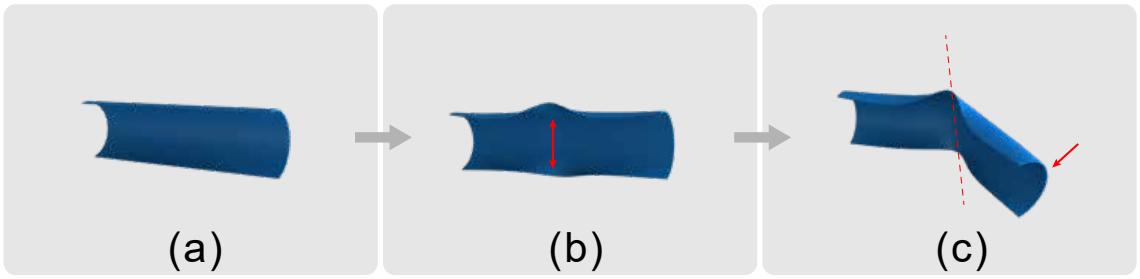


Figure 2. Overview and behavior of the Thermally-Actuated SCRAM Limb (TASL). (a) A thin planar material is curved into a U-shape to create stiffness. (b) Curvature is induced by SMA wire embedded in the surface of the material via tailored wire placement. (c) The structure buckles at the location of the surface curvature under external pressure, creating a joint.

The Thermally-Actuated SCRAM Limb (TASL) is a mechanism that aligns closely with the SCRAM category of soft robots, extending the concepts established in previous research [11, 12]. Utilizing aforementioned principles, joints can be formed at arbitrary locations along a continuum curved sheet. This is achieved by inducing surface weakness using SMA wire, followed by the application of external actuation to induce buckling at the desired position. The TASL design incorporates surface actuators directly into the material, deploying continuous SMA wire in a serpentine pattern, contrasting the approach of fixed-location, externally actuated wire tendons as found in [11]. SCRAMs are further distinct in their employment of planar materials and planar fabrication methods. The choice of a curved sheet over a tube enables truly planar and simplified fabrication, facilitated by established wire placement techniques such as couching [13]. The integration of a continuous SMA wire along the curved sheet enables joint location modification through the actuation of different wire segments, yielding a reconfigurable system.

1.1 Primary Contributions

First, the chapter demonstrates that untrained SMA wire can serve as a surface actuator to induce buckling in curved thin-walled structures, thereby forming a compliant joint. Notably, this joint can be generated at any point along the continuum structure by actuating distinct segments of a continuous SMA wire. Second, the chapter introduces a specific layout pattern for the untrained SMA wire when used as a surface actuator and characterizes its behavior. Third, a circuit is engineered and an associated control algorithm is developed for the purpose of selecting an arbitrary number of segments to activate. Experimental evidence substantiates that the proposed mechanism is capable of establishing virtual joints at various locations along the structure.

Curvature as a means for altering stiffness has gained considerable attention in literature [11,14–16] and exhibits natural analogs in fish fins, batoids, and insect wings [12]. Jiang et al. propose a methodology for generating directional, compliant virtual joints in thin-walled tubes by manipulating local surface curvature [11]. Actuation of opposing internal wire tendons exerts pinching forces on the tube's surface, consequently modifying local stiffness. This allows for the formation of virtual joints in any radial direction, with the original shape and stiffness recoverable upon release. The authors classify this mechanism as a SCRAM [11]. An extension of this work by Jiang et al. leverages internal negative pressure to achieve buckling/pinching in thin-walled tubes, controlled via a movable rigid confining sleeve [17].

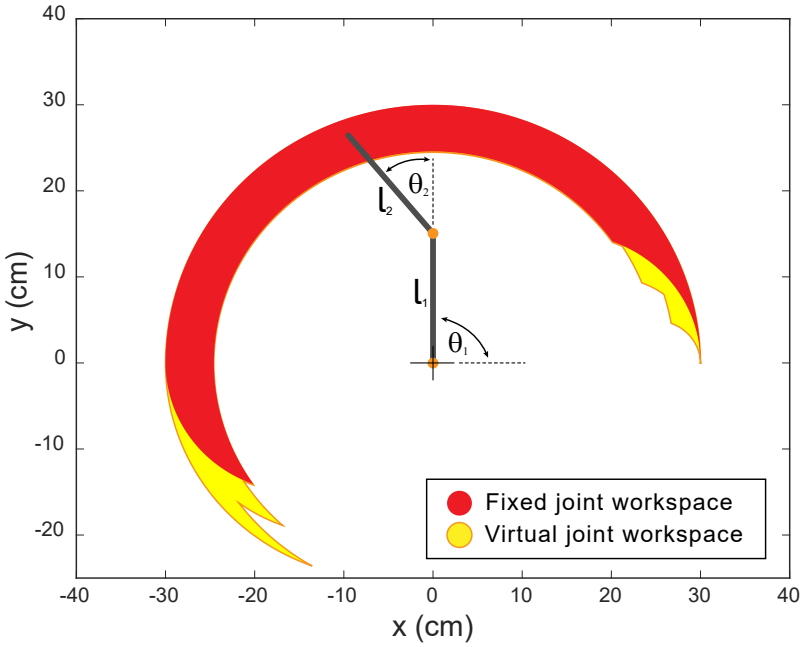


Figure 3. Workspace comparison between a fixed joint and reconfigurable virtual joint two-link planar arm where $0° < \theta_1 < 180°$ and $0° < \theta_2 < 70°$. The reconfigurable arm given five equally spaced joint locations has a wider reach than the fixed joint arm.

Sharifzadeh et al. introduce the use of buckling tape springs for locomotion via flapping fins and wings [12]. Employing a compliant, curved beam connected to an

electronic servo and a fin (or wing), oscillation of the servo applies a force at the beam's end, resulting in buckling. This action effectively forms a joint for part of the flapping cycle, leading to large deflections around the buckling point. Drag serves as the restorative force for the curved beam. System buckling behavior is tunable by adjusting design parameters such as beam length, curvature, and thickness [12].

2.1 Reconfiguration

The hysteresis and mechanical nonlinearities inherent in the reconfigurable system offer numerous advantages for applications in locomotion and object manipulation. The TASL with a singular virtual joint can be abstracted as a 2R planar robotic manipulator possessing two degrees-of-freedom (DoF), as depicted in Figure 3. If the limits on the joint angles of this example manipulator are $0° < \theta_1 < 180°$ and $0° < \theta_2 < 70°$ with the position of the tip given by

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \end{bmatrix} \tag{1}$$

the workspace of this manipulator is contained in the envelope of the plot of the tip positions for all θ_1 and θ_2 angles. The virtual joint position alters parameters l_1 and l_2, thereby modifying the workspace. Consequently, the TASL workspace represents the aggregation of workspaces for each feasible virtual joint location, resulting in a more extensive operational range compared to fixed-joint manipulators. In relation to the flapping mechanisms introduced by Sharifzadeh et al. [12], the capability to modulate the flexing point of the wing provides flexibility in altering the wing's motion patterns, thereby affecting the vehicle's overall dynamics for optimization in terms of either speed or energy efficiency.

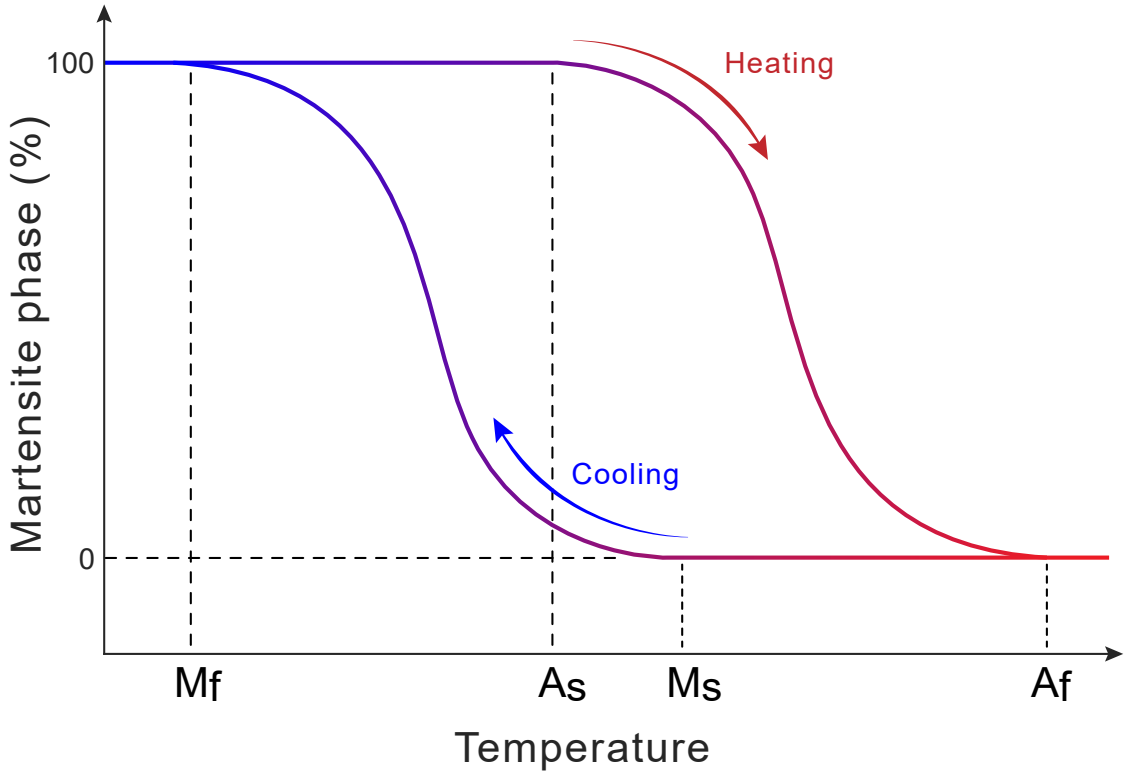


Figure 4. The heating and cooling cycle of SMAs. As the temperature increases, the atomic lattice structure of SMA wire transitions from the martensite (flexible) state to the austenite (stiff) state. As it cools, it transitions from the austenite state back to the martensite state.

2.2 Shape Memory Alloys

SMAs are useful in soft robotics for their ability to change shape to a known memorized shape when exposed to thermal energy. The training (memorizing) process for SMA wire involves bending it into the desired shape, heating it to a high temperature typically around 400− 500 °C, and then letting it cool down. Any subsequent heating near or past its rated activation temperature (which is significantly lower than the training temperature) will cause the SMA wire to attempt to resume its trained shape. This heating can be either external heating or Joule heating by running an electrical current through the wire. There are two general states or phases in which SMAs exist based on their internal crystal lattice structures. In the **cold** martensite phase, the SMA wire is flexible much like a regular solid wire. When heated, the

SMA wire's internal structure transitions to the **hot** austenite phase during which the wire stiffens into its trained shape and becomes effectively unbendable, with a spring constant that is 2-3 times higher than in the martensite phase [18]. As illustrated in Figure 4, heating a SMA causes it to transition from the martensite phase starting at the austenite start temperature (A_s) to the austenite phase ending at the austenite finish temperature (A_f). When cooling, the SMA wire starts transitioning to the martensite phase at the martensite start temperature (M_s) and finishes at the martensite finish temperature (M_f). SMA wire is useful as an actuator because as it heats up and changes its internal structure it contracts $4-8\,\%$, based on the alloy, which can generate significant forces albeit at a short actuation distance. This actuation distance can be increased up to $200-1000\,\%$ by training the wire into a coil spring shape, at the cost of a reduction in force [19]. Nickel-titanium (Nitinol) SMA is the most popular alloy and transitions in the internal crystalline structure under heating changes allow for a $4\,\%$ reduction in length [18].

SMA has been widely used in literature [13,20–39] for actuation [19,24,31,39–43], hinges for foldable systems [20, 21, 26, 44], and stiffness tuning layers for soft fluidic actuators [43]. Seok at al. [41] use SMA wire in the design of a soft mobile robotic platform that exhibits peristaltic locomotion – the wire is fabricated into a coil spring actuator whose spring constant is 2-3 times greater in the austenite phase than in the martensite phase. Koh et al. present a singe-body crawling robot that uses SMA spring actuators to drive two flat four-bar linkages and a folding six-bar linkage [42]. SMA wire is also commonly used to create finger-like actuators that exhibit planar motion. Kim et al. [24] present a finger-like actuator with SMA wire tendons embedded in PDMS that uses soft hinges to increase the bending deformation while Jin et el. [39] also use multiple SMA wires embedded in a rectangular PDMS structure that exhibits bidirectional planar motion. SMAs see wide use in textiles and are one of the most mature fields in active textiles [18, 34, 38, 45]. Buckner et al. [13] use

SMA wire as an actuating fiber trained to create in-plane bending motion in a fabric substrate that is able to present antagonistic motion if paired with a similar actuator on the other side of the fabric. The authors note a major challenge with integrating wire bending actuators into flexible fabric is the tendency for the wire to twist the fabric instead of bending in-plane, caused by any off-center forces, which they alleviate by flattening the round SMA wire into a rectangular profile via annealing and rolling.

There are a number of drawbacks to using SMA wire that need to be overcome in order for it to be a feasible option including the relatively low usable strain, controllability, accuracy, actuation frequency, and energy efficiency [40]. Similar to [13], the SMA wire in the TASL is used to create in-plane bending motion for surface actuation but without the need for rolling the SMA wire to flatten it and with no additional training needed other than the default factory straight-training the wire receives during the manufacturing process. Using it in this way allows for large deflections even without training as a spring. Unlike most SMA actuator application, the TASL uses SMA wire not as the primary actuator but only momentarily for the purpose of altering surface curvature so there is no need for great accuracy; this also results in greatly diminished power requirements. Finally, the use case of SMA wire in the TASL does not require precise closed-loop temperature control or fast actuation cycles so those two considerations are not of great concern in this application.

3 Design and Fabrication of the TASL

This section explores the physical specifications, material selection, and fabrication of the TASL. As earlier mentioned, one of the advantages of the TASL is that it uses planar fabrication techniques so fabrication is fairly simple, fast, and accessible. The TASL consists of a layer of fabric, a layer of thin plastic acting as a stiffener and finally SMA wire in a serpentine pattern (Figure 5), with planar dimensions of 250 mm x 100 mm. The SMA wire was laid out in a serpentine pattern with each

segment 80 mm long, 8 mm apart, and with 4 mm radius corners. The core of the design relies on the use of SMA wire as a surface actuator so the selection and layout of the SMA wire was an important design parameter.

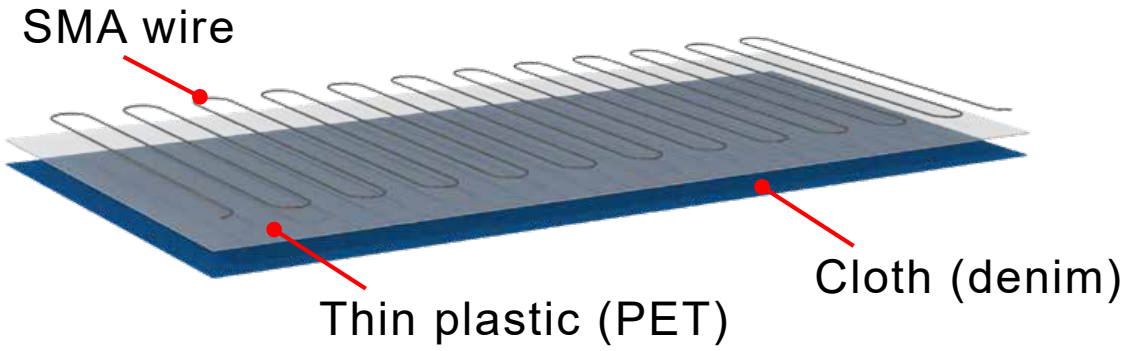


Figure 5. The TASL is constructed from two material layers, denim cloth and a stiffening PET plastic with SMA wire embedded into the materials using tailored wire placement.

3.1 Material Selection

3.1.1 SMA Wire

There are a number of important parameters to consider when selecting SMA wire such as the alloy, activation temperature, and wire diameter. Nitinol is the best choice of SMA alloy for most applications due to its great stability and thermomechanical properties as opposed to copper and iron-based alloys [40]. The selection of activation temperature depends on the environment and power requirements of the application. For instance, SMA wire that comes in contact with human skin needs to have an activation temperature low enough to not cause burns but high enough that body heat will not activate it. Another important consideration is hysteresis, which is the difference between the heating and cooling transition temperatures given by $\Delta T = A_f - M_s$. The selected SMA wire features an activation temperature of 40 °C and

a nominal 0.50 mm diameter. The selected activation temperature allowed for fast heating with an electrical current while the small diameter enabled for faster cooling.

3.1.2 Planar Material

In material selection the aim was to find a thin, inextensible, planar material that had the ability to bend without creasing while also providing enough rigidity to maintain a curved shape. This material also needed to be compliant enough to be actuated by the SMA wire. Initially, SMA wire was embroidered directly onto the denim fabric which yielded poor results when attempting to configure it into a curved shape. The SMA-embedded denim would simply fold over itself and wrinkle up without maintaining shape which meant that a stiffening layer was needed. TPU with a nominal thickness of 0.67 mm was then evaluated. When placing the SMA wire and attempting to curve the material, it was observed that the TPU lacked sufficient rigidity, even with the serpentine SMA wire pattern, causing the structure to crumple randomly. Attempting to bend it resulted in the structure crumpling in random locations. Next, a more rigid frosted Mylar (PET) film with a nominal 0.52 mm thickness was evaluated. With the SMA wire in place, this material was found to have the necessary rigidity to form a curved structure and also fold without crumpling. The PET film had a tendency to sometimes create sharp creases so we first stretched a layer of denim under the PET before placing the wire. This layer of denim stretched under the PET had the effect of keeping the bends smooth and constraining the crease to a small radius instead of a sharp end.

3.2 Characterization

3.2.1 SMA Wire Temperature vs. Current

Utilizing Joule heating for SMA wire activation necessitated initial characterization of current's effect on wire temperature. The subject of the test was an 8 cm SMA

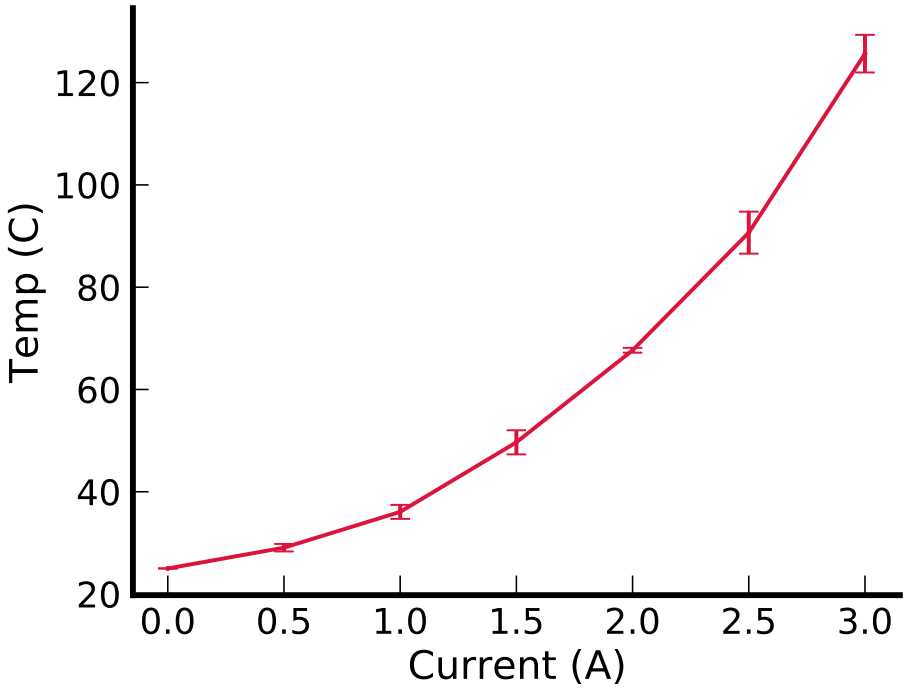


Figure 6. This plot correlates current to temperature since SMA is activated by heat and the TASL runs current through the wire to create heat.

wire piece with a cross-sectional diameter of 0.5 mm and a resistance of 300 mΩ. Current increments were fed into the SMA wire, and the steady-state temperature was measured using a FLIR E6 thermal camera with an emissivity of $\epsilon = 0.60$. Results appear in Figure 6 with $n = 3$. The SMA wire requires approximately 1 A to reach the rated 40 °C activation temperature, despite the A_s temperature being less than 40 °C. On average, less than 3 seconds were needed to reach the target temperature.

3.2.2 Blocked Bending Force

Two SMA patches were fabricated that contained one and two loops, respectively, of the serpentine pattern (with the two-loop patch shown in Figure 7a) in order to better characterize the forces generated by each segment of the TASL. The behavior of the patches is such that when the SMA wire is not activated, the patch can be moved freely and bent or curved into any position as shown in Figure 7b. When current is applied, the patch straightens out into a flat sheet whose stiffness depends

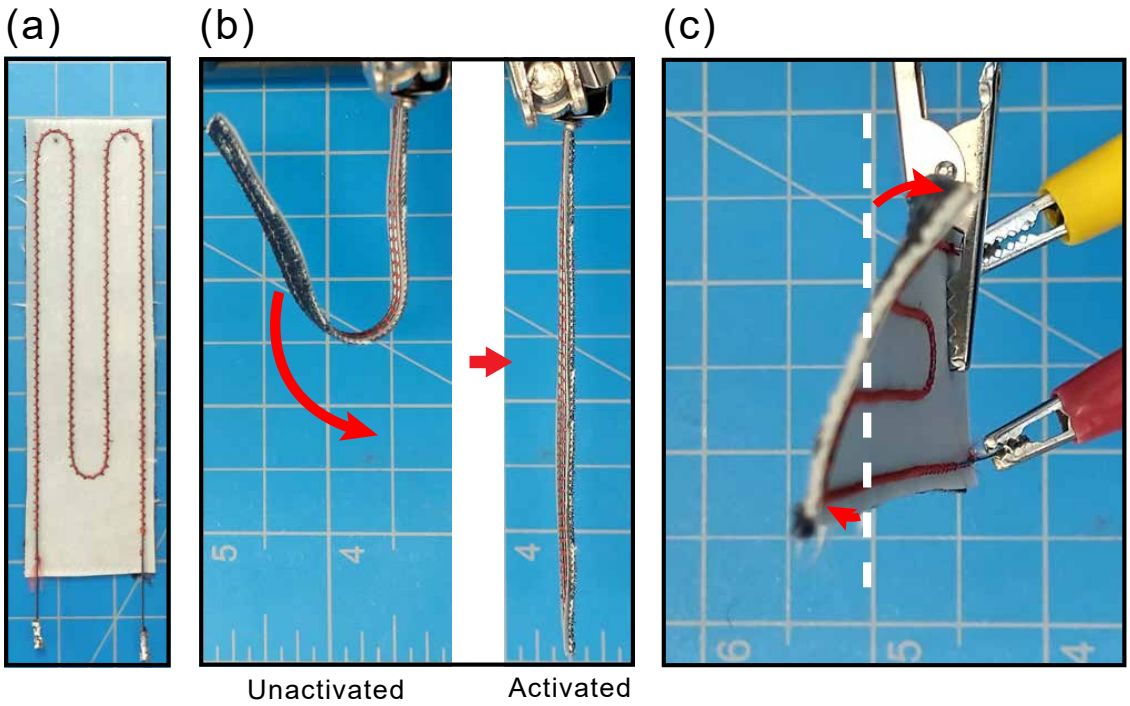


Figure 7. Characterization of the SMA wire patch which represents a single segment out of the continuous SMA wire serpentine pattern utilized in the TASL. (a) Example of the SMA wire patch with two loops. (b) The behavior of the SMA wire patch before and after activation. (c) The twisting behavior of the patch when fully activated.

on the rigidity of the SMA wire (EI) [13]. The speed at which the patch straightens out positively correlates with the applied current, which allows for the actuation force of the patch to be measured. This force was measured using the blocked bending tip force test. Each patch was firmly affixed on one end, leaving 10 cm of the patch for actuation. The free end was bent until the patch was curved into a U-shape with a 2.4 cm opening (diameter). An end cap that spanned the width of the patch was 3D-printed and attached to the tip of the patch (making contact with the SMA) and a piece of rigid wire was attached to the end cap, passed through an opening in the fixed end of the patch, and connected to a Nextech DFS20 force gauge. The steady-state force was recorded for increasing levels of current, shown in Figure 8. The single-loop patch converged to a maximum steady-state 0.8 N of force while the double-loop reached a steady-state 1.6 N by 3 A. These results indicate that the max

force increases linearly with the loop count.

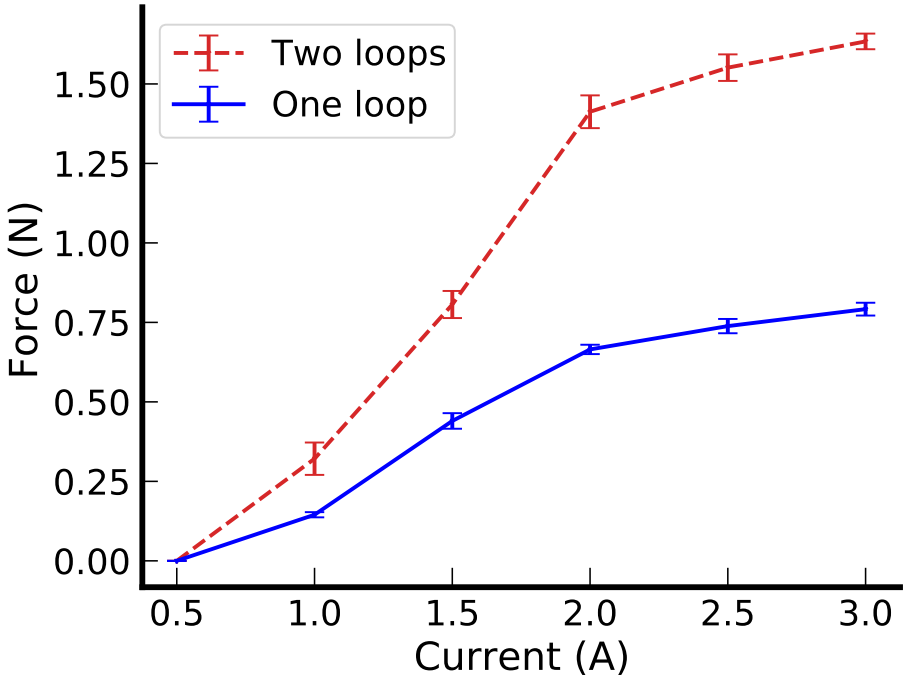


Figure 8. The maximum tip blocking force scales linearly with the number of loops.

3.2.3 Surface Actuation

As the forces increased, the patch would exhibit twisting at the free end as shown in Figure 7c. This behavior can be caused by any off-center forces causing the wire to twist rather than bend and is typically undesired behavior. In fact, Buckner et al. [13] went to significant effort to prevent twisting by physically flattening the SMA wire and training it to bend in-plane. For our application this bending is highly desired because it forces the material to temporarily crease which encourages buckling and thus the formation of a joint. To evaluate the patch's behavior as a segment within the full continuum structure, six evenly-spaced wires were attached to the SMA wire, creating two loops between each pair of wires, similar to the configuration in the patch. As shown in Figure 9, circular clamps were also attached at each end to induce camber and thus curving in the TASL. Images were captured using a thermal

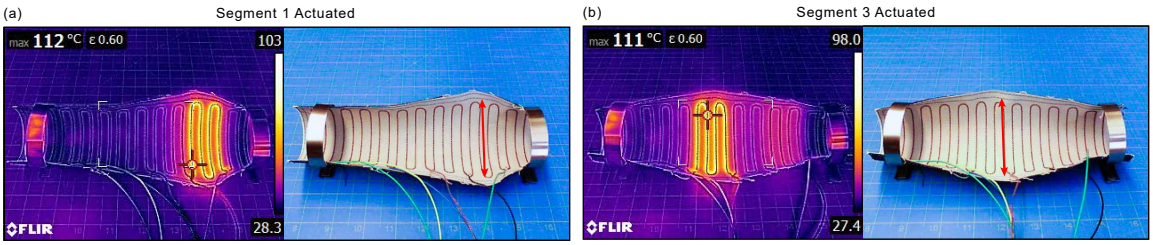


Figure 9. Fundamental behavior of the TASL. When an SMA wire segment is actuated, it stretches and creates a flat spot in the curved surface which allows for controlled buckling to create joints.

camera as well as a regular camera in order to better see the thermal behavior when activated. Figures 9a and 9b show that the SMA is able to significantly actuate the material in different areas based on which segment is active. There is also very little heating of the adjacent areas. This test was able to validate the behavior of the patch in its tendency to want to flatten when activated.

4 Control of the TASL

4.1 Hardware Control

The SMA wire segments are energized by the controlled routing of current through a segment using a MOSFET switch circuit. The fundamental operation of this circuit is to be able to switch a node (an SMA wire in this case) from ground potential to positive and vice versa, effectively creating a SPDT switch. As shown in Figure 10, this is achieved by using two matched MOSFETs in series, with the controllable node taken in between the MOSFETs. The gates of the two MOSFETs are wired to follow the exclusive-OR behavior such that the only two valid states are when one and only one of the two MOSFETs is on. When the top MOSFET is on and the bottom is off, the controllable node is at positive potential and inversely when the top MOSFET is off and the bottom is on, the controllable node is at ground potential. In this way the controllable node can act as a current source or sink for the SMA wire segment. This base switch circuit is repeated six times in order to control the

five SMA segments since n segments require $n + 1$ switch circuits. The full circuit was made into a PCB which was designed to be mounted to and controlled by an Arduino Mega microcontroller.

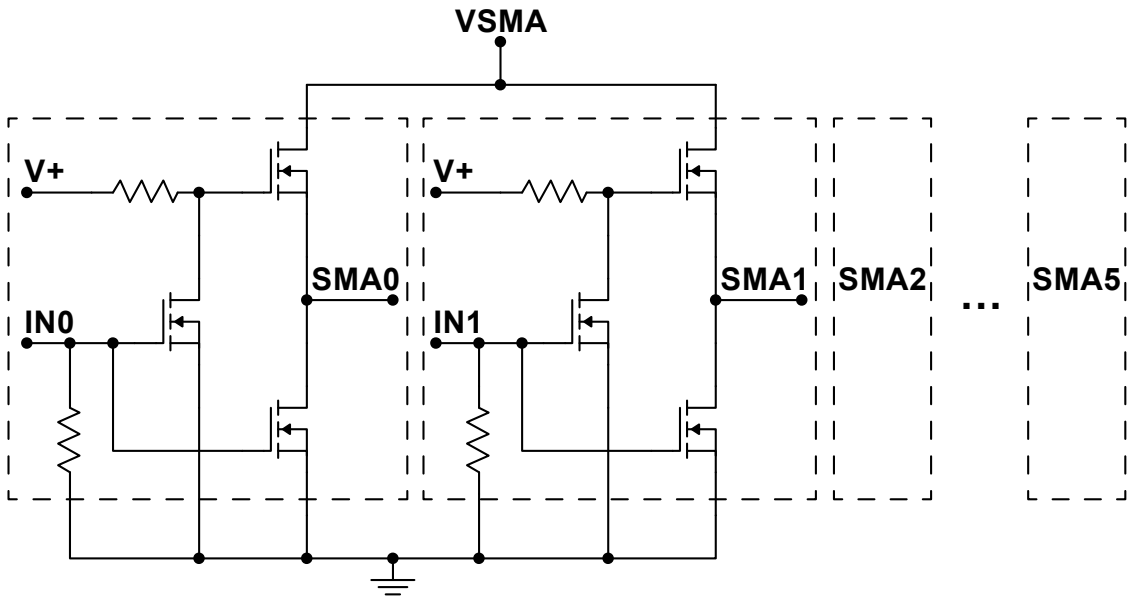


Figure 10. Simplified SMA wire segment driver circuit showing two of the six channels. For each channel, two MOSFETs are used to reference the channel to either ground or the positive VSMA.

4.2 Control Algorithm

Since TASL is a continuum mechanism capable of having an arbitrary number of segments, an algorithm was developed for determining which mode (source or sink) to set each node to in order to activate any desired segment(s). Each of the six nodes in the circuit can be represented as a single bit in a 6-bit binary number, with the rightmost node representing *bit* 0. A segment is therefore represented as a pair of adjacent bits, with *Segment 1* corresponding to bits 0 and 1. This definition also means a TASL with n nodes will have $n - 1$ controllable segments, as earlier mentioned. Using this scheme, a segment is energized when there is a transition from 0 to 1 or 1 to 0 between adjacent bits. Electrically, this represents a difference

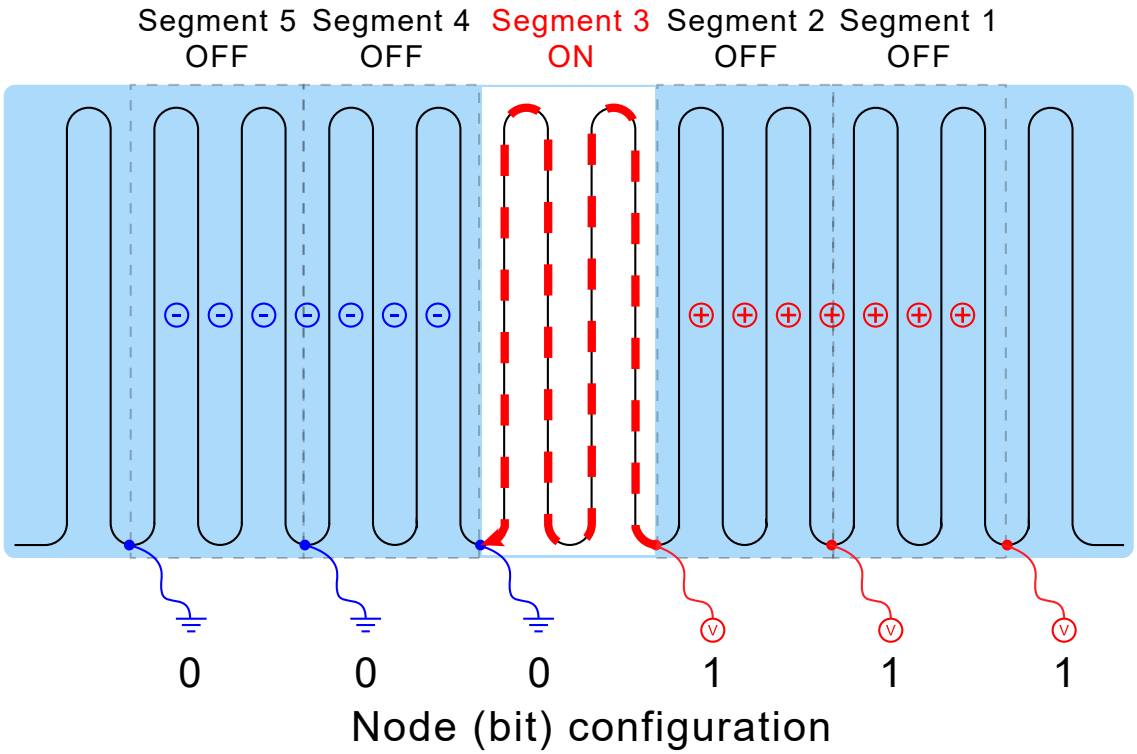


Figure 11. Visualization of the technique used by the control algorithm to activate a segment of SMA wire along the continuum by creating a potential difference between adjacent nodes.

in electrical potential and thus a flow of current through that segment, which is visualized in Figure 11.

The binary number $\mathcal{K}$ that represents the bit configuration needed to energize segment m is defined as

$$\mathcal{K} = 2^m - 1 \qquad m \geq 1 \tag{2}$$

As an example using this definition, $\mathcal{K}$ for energizing *Segment 3* is found to be

$$\mathcal{K}_3 = 2^3 - 1 = 7 = b000111 \tag{3}$$

The only transition between a 0 and 1 in this binary number occurs on the two bits representing segment 3, thus segment 3 is energized as visualized in Figure 11. Energizing multiple segments is then achieved by taking the Exclusive-OR of the $\mathcal{K}$-values of all the segments that need to be energized, given by

$$\mathcal{K}_1 \oplus \mathcal{K}_2 \oplus \cdots \mathcal{K}_n \qquad n \in S \tag{4}$$

where S contains the segment numbers that need to be energized.

5 Experimental Evaluation

In this section the TASL is evaluated for its ability to create reconfigurable virtual joints. The goal of the prototype was to demonstrate the successful creation of a virtual joint whose position could be reconfigured. Two 5 cm diameter end caps were laser cut out of 0.635 cm acrylic which where then used as end caps for the limb, fastened together using a stainless steel hose clamp. The assembled TASL was mounted horizontally on one end with a string attached to the opposite end to be used for external actuation of the limb. Control of the arm was achieved using the aforementioned hardware and a custom software solution. The SMA driver PCB communicated via USB serial to a local Linux control server. This control server also hosted the HTML-based graphical user interface (GUI) used to control the hardware and facilitated duplex communication between the UI and the hardware. Finally, a Dynamixel MX-28 actuator was used as the external actuating force, using a string and pulley system to force buckling in the TASL and create joints.

Results show that the TASL was able to achieve the stated goal of creating virtual joints at different locations. First to validate the reduction in stiffness after surface actuation, a constant pulling force was applied normal to the top plane of the TASL until it buckled. This was the equivalent of putting a weight on top of a half-cylinder. This experiment was performed first without any segments actuated and then with *Segment 3* actuated. Results in Figure 12 show that SMA surface actuation resulted in a 3.1 N reduction in the force required to cause buckling. The buckling for the actuated test was also much more smooth and controlled as opposed to the unactuated test.

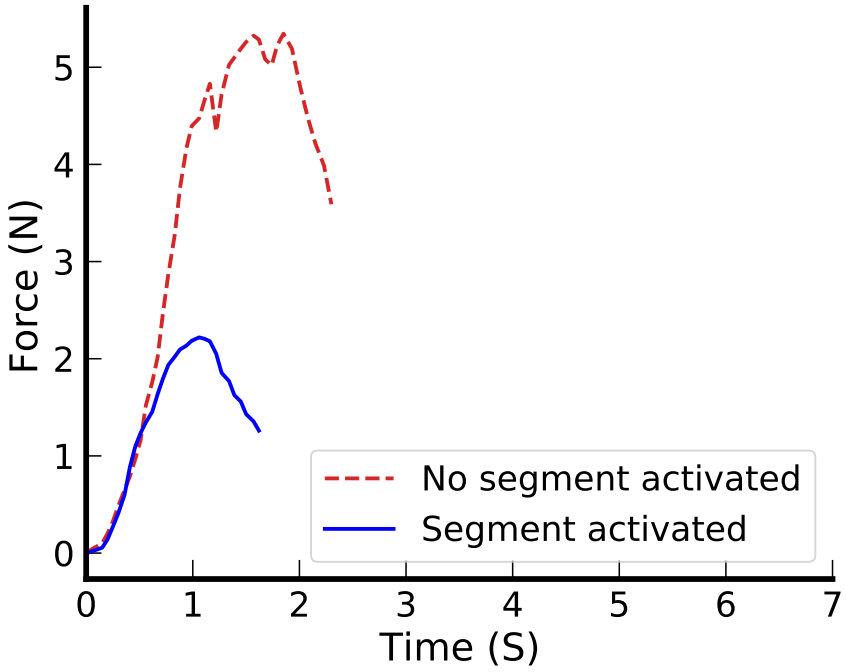


Figure 12. The force (normal to the tip) at which the TASL buckles is reduced from 5.35 N to 2.22 N after surface actuation. All tests were performed at 22.8 °C and standard environmental condition (static air, atmospheric pressure).

Next, the test setup was reconfigured such that the initial pulling force was at an approximately 45° angle as shown in Figure 13. When *Segment 1* was activated (Figure 13a), as hypothesized the TASL buckled at that point which allowed for a joint to form. Upon releasing the pulling force, the TASL reverted to it's flat position under its own power due to the elasticity of the combined materials creating a natural restoring force. Next, *Segment 3* was activated and repeated the same procedure. Once again the TASL buckled at the location of the activated segment and created a joint under an external force (Figure 13b).

6 Conclusions

This chapter introduces and validates the application of SMA wire for altering the local surface curvature of curved, thin-walled sheets, thereby enabling the creation of reconfigurable virtual joints. The mechanism falls under the category of Soft,

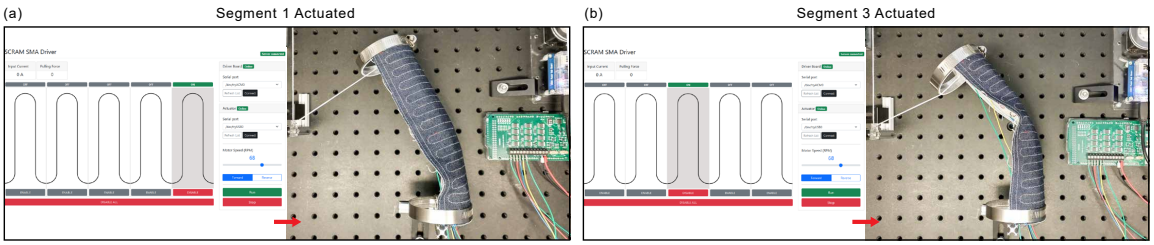


Figure 13. A demonstration of the TASL with different joint locations. (a) A joint formed with *Segment 1* actuated. (b) A joint formed with *Segment 3* actuated.

Curved, Reconfigurable, Anisotropic Mechanisms, or SCRAMs. In the development of the Thermally-Activated SCRAM Limb (TASL), initial demonstrations reveal the effective use of untrained SMA wire as a surface actuator, despite known drawbacks and limitations. Subsequent characterization of SMA wire tailored into a U-shaped patch indicates a tendency to straighten, irrespective of the initial shape. Extension of this U-shape into a continuous serpentine pattern spanning the material length occurs. Tests indicate that two loops of this serpentine pattern can exert up to 1.6 N of force, sufficient for bending most planar compliant materials. Validation of the concept through a reconfiguration demonstration confirms the successful creation of joints at various locations. TASL offers considerable promise due to its conceptual and fabrication simplicity. The system's nonlinearities and reconfiguration capabilities pave the way for the development of simple yet advanced variable-DoF robots. Utilization of planar material and fabrication techniques represents a novel direction in soft robotics, opening numerous avenues for advancements in both mechanisms and sensor technologies.

CHAPTER III

ADVANCED APPLICATIONS OF SOFT OPTICAL SENSING IN SOFT ROBOTICS

1 Introduction

This chapter presents the novel use of air gaps in flexible optical light pipes to create coded patterns for use in bend localization and other realtime soft sensing applications. Soft optical deformation sensors make a good partner with soft robotics because their mechanical properties match, their materials are compatible with rapid prototyping, and they are less susceptible than electronic sensors to electromagnetic noise and temperature drift. In the previous chapter discussion covered a new class of soft robots consisting of Soft, Curved, Reconfigurable, Anisotropic Mechanisms, or SCRAMs [9, 11, 12, 17, 46]. The Thermally-Activated SCRAM Limb (TASL), a SCRAM device that can create virtual joints along a continuum curved sheet made of denim and PET plastic by creating a surface weakness along the curve using SMA wire that is embroidered into the sheet, was introduced and demonstrated [9]. One shortcoming of the TASL was the lack of a bending sensor to verify that a joint formed at the desired location. This specifically required a bend sensor that was soft, small,

and flexible enough to embed into a surface. This chapter presents the optical gap (OptiGap) sensor system, a novel soft optical sensor system that is low cost, flexible, simple to fabricate, and can able to perform real-time bend sensing or localization on almost any modern microcontroller. The OptiGap sensor system enables creation of extrinsic intensity-modulated bend sensors functioning as flexible absolute linear encoders.

1.1 Primary Contributions

First, a novel approach is introduced that employs an air gap with a sleeve, as depicted in Figure 15, for generating coded, bend-sensitive patterns in optical light pipes. This is achieved through straightforward cutting techniques, obviating the need for complex equipment. Second, traditional robotics concepts of linear encoders and Gray codes are adapted to suit a flexible soft optics sensor. Lastly, a cost-effective, reconfigurable sensor system is presented. This system is not only rapid in its operation but also versatile in its material composition, allowing for swift customization tailored to specific applications.

2 Background and Literature Review

2.1 Fiber Optic Sensors

Fiber optic sensors (FOS) typically consist of three components: a light source, a fiber light pipe that carries light and can be modulated, and a photodetector. FOS can be classified as either intrinsic or extrinsic. Intrinsic sensors have the sensing components integrated into the fiber such that the light is always contained and modulated within the fiber while extrinsic sensors have external sensing components that modulate that light, with the fiber acting primarily as a light pipe [47, 48]. FOS can also be categorized by whether they modulate the wavelength, polarization,

phase, intensity, or a combination of those. Phase and intensity modulation sensors are by far the most common [48].

Recent advancements in soft robotics emphasize the integration of such soft sensors to enhance real-time feedback and adaptability [49–56]. Al Jaber et al. [54] introduce a method for registering the shape and orientation of soft robots using segmented optical fibers, a camera, and a calibration algorithm, demonstrating its potential for accurate shape reconstruction of continuum soft robots. Another work [49] presents stretchable optical waveguides as strain sensors for prosthetics, underscoring their potential in enhancing sensory capabilities in soft robotic systems. Next, Galloway et al. [56] integrates a fiber optic shape sensor into soft robotic systems, offering high-resolution shape information, while Every et al. [53] introduces a proprioceptive soft actuator using electrical impedance tomography for shape sensing. A roughness tuning strategy for fabricating multi-modal soft optical sensors is also presented [51], emphasizing their utility in enhancing soft robot controllability. Finally, [50, 52] further explore optical and electro-conductive yarn-based sensing mechanisms, respectively, highlighting their potential in applications ranging from wearable sensing technologies to minimally invasive surgery. These works underscore the the importance of real-time soft sensing for feedback and adaptability.

OptiGap sensors can be considered extrinsic intensity modulated FOS. As shown in Figure 14b the sensor system has three main components: IR LED emitters, flexible parallel light pipes, and a photodarlington detector. The light pipes all terminate into a single detector and the number of light pipes is variable, depending on the desired configuration of the sensor. Bend sensitivity is created at desired locations by cutting the light pipe and then re-attaching the pieces together using a sleeve to create a small air gap, shown in Figure 14a. This is done on each of the multiple light pipes in order to create the air gap patterns (or codes) used for bend localization.

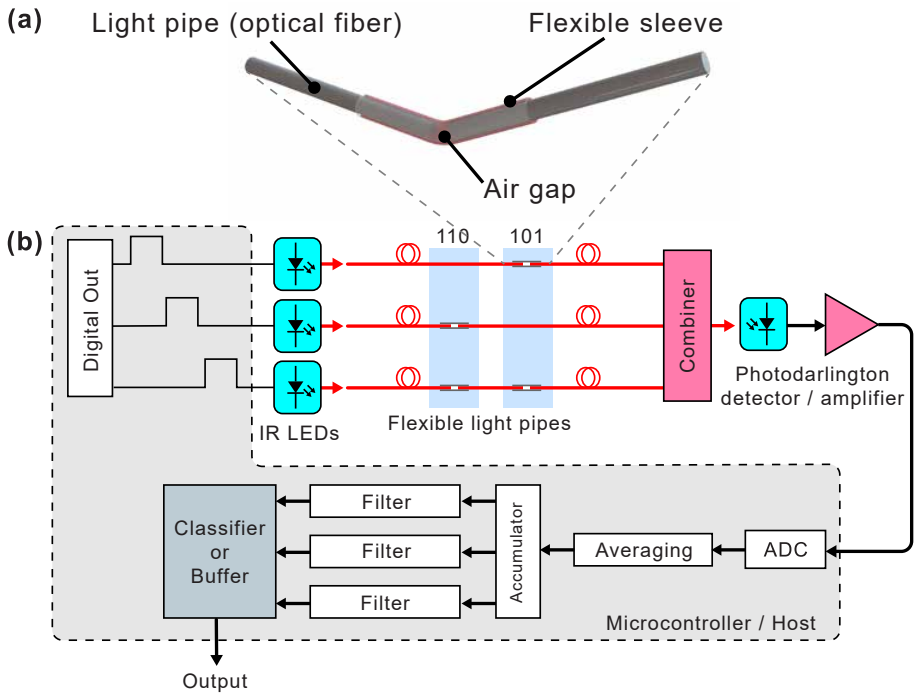


Figure 14. Overview of the OptiGap sensor system. (a) Construction of the air gap used to create bend-sensitive areas in the light pipe. (b) Block diagram of the OptiGap system showing the optical emitters and detector, the fibers with bend-sensitive gaps, and the internal block diagram of the STM32 microcontroller that handles classification of the detected air gap patterns.

2.2 OptiGap Bend-Sensitive Air Gaps

OptiGap creates bend-sensitive air gaps along a piece of fiber by concentrating the mechanical deformation to a predetermined location. This is achieved by cutting the fiber perpendicular to the fiber axis and then re-attaching it together using soft silicone tubing while leaving a small air gap. The simulation and model in Figure 15 shows the main working principle of the air gap: translation and/or rotation of one fiber face relative to the other changes the fraction of light transmitted across the gap. The greater the bend angle the more light escapes across the gap. The resulting change in intensity of the optical signal is then correlated with known deformation for use as a sensor.

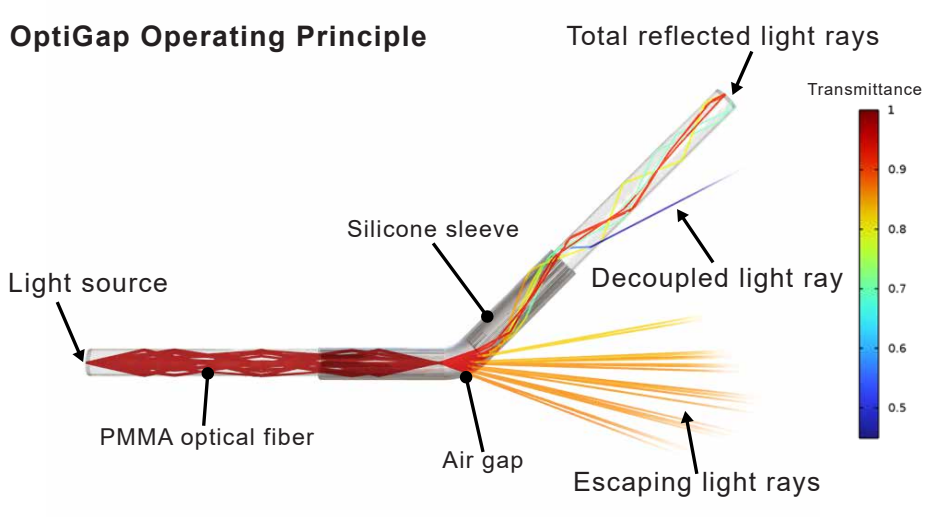


Figure 15. A ray optics model and simulation of the bend-sensitive air gap used in OptiGap sensors. This model is bent at a 45° angle with a 15° cone angle light source and shows a significant amount of light escaping at the gap.

2.3 Existing Approaches

The concept of using a gap in fiber for intensity modulation can be found in existing literature. For a gap created by making a slice perpendicular to the fiber axis, a simple 2-dimensional model can be used where a circular fiber face is translated across another face and the overlap area gives the amount of transmission. This type of model is used in [57–59] and is used to describe bending [57, 58] and pressure [59] induced changes in optical intensity. The sensor in [57] uses one input fiber and two output fibers to measure bending in flexion and extension with an LED source at one end and a photodetector at the other end. Similarly, [58] uses one input fiber and three output fibers separated by a gap. This allows for measuring flexion, extension, and lateral planes. Both of these sensors have a single sensitive area and provide no information on bending location. Lin et al. [59] measure pressure by correlating the increase in optical intensity as the faces of two fibers align due to an external force.

Other similar non-gap approaches exist to measure bending and pressure [60, 61]. Zhao et el. create a curve sensor by bending a 1 mm acrylic fiber into a U-shape and roughening one side with a laser cutter in order to increase optical bending losses [60].

However, this sensor cannot provide information on bend location, motivating the same group to investigate wavelength-based encoding of the deformation site, leading to the SLIMS sensor [62]. The SLIMS sensor relies on color dyes in polyurethane elastomers and color sensors to detect bend locations, which makes the fabrication process complex, material selection limited, and imposes a centimeter-length limitation [62], unlike OptiGap. The pressure sensor in [61] consists of a pressure sensing sheet that is made of intersecting rows and columns of fiber encased in PDMS, each with an emitter and detector. A map of displacement and/or force over an area is generated by the intensity modulation caused by the bending of the fibers [61].

3 OptiGap as a Bend Localization Sensor

3.1 Operating Principles

The OptiGap sensor can best be thought of as a flexible absolute linear encoder. A linear encoder measures the linear displacement of an object and typically consists of a slider rail with a coded scale (much like a measuring ruler) and a sensing head that slides over that scale and reads the scale. The reading of the scale can be done by magnetic, optical, capacitive, resistive, ultrasonic, inductive, or mechanical means [63]. Absolute encoders output a unique pulse code at each step so the displacement relative to some scale is always known. More intuitively, an absolute encoder can be thought of as a ruler with the numbers and tick marks present while an incremental encoder is the same ruler with tick marks but no numbers. Similar to an absolute encoder, the OptiGap system can encode absolute positions using bend-sensitive air gap patterns along parallel light pipes as a FOS. The pattern of these bend-sensitive air gaps used to encode the bend location follows an n-bit binary sequence, where n is the number of parallel paths.

Table 2. 3-bit Gray code sequences

Number	Gray code	Inverse Gray code
0	000	000
1	001	110
2	011	011
3	010	101
4	110	010
5	111	100
6	101	001
7	100	111

3.1.1 n-bit Inverse Gray Code

Gray code is a sequence of n-bit binary numbers where only a single bit is changed when transitioning from one number to the next, which can also be thought of as the Hamming distance between two adjacent numbers in the sequence is 1 [64]. As shown in the first column of Table 2, this can be a form of built-in error detection since a change of more than one bit in the sequence has to be an error, which is why Gray code is often used as the coding for encoders as well as many communications applications. Since the OptiGap system relies on a GNB classifier when performing bend localization to identify the active air gap pattern it's important to maximize the information gain from one pattern to the next, which is the opposite of what Gray code does. Inverse Gray code is where two adjacent n-bit numbers in the sequence differ by $n-1$ bits, which is the maximum number of bits that can change in a binary sequence. A 3-bit example of this inverse Gray code is shown in the second column of Table 2 and was generated using the technique proposed in [65].

3.1.2 Sensor Array Gap Pattern

Each air gap pattern on an OptiGap sensor array can be thought of as an n-bit binary word in an inverse Gray code sequence. The inverse Gray code of number 1 in Table

2 corresponds with the first vertical pattern of air gaps in Figure 16b. Similarly, number 7 in Table 2 corresponds with the last vertical pattern of air gaps. When a bend happens at an air gap pattern, the attenuation in optical intensity results in the bit pattern shown in Figure 16c and is similar to the pulse pattern generated by an absolute encoder. Unlike an encoder, the real intensity signals, shown in Figure 16d, are not consistent enough to directly convert to a binary signal, which is why the GNB classifier explained in Section 3.4 is used. Finally, each n-fiber sensor is limited to $2^n - 1$ sensitive patterns due to each pattern being equivalent to a binary word.

3.2 Fabrication

Table 3. OptiGap Configurations Tested

Length (m)	Material	Diameter (mm)	Air Gap Patterns
1.1 (Sensor A)	PMMA (fiber)	0.5	7
1.3 (Sensor B)	PMMA (fiber)	0.75	7
0.7 (Sensor C)	TPU (filament)	1.75	3

Because OptiGap is not a singular sensor but rather an adaptable *sensor system*, fabrication of any one sensor is entirely dependent on the application for that particular sensor. This is especially true of the placement of the bend-sensitive air gap patterns. While the system shown in Figure 16 has evenly spaced air gap patterns for the sake of comparison with the illustrations, the air gap patterns can be placed anywhere along the length of the light pipes, much like the gratings in fiber Bragg grating (FBG) sensors [66–72].

Table 3 shows the system of Figure 16 can be built from different diameter fibers and optical fiber materials. For PMMA fibers, the optical "combiner" in Figure 16 consists of the three fibers in a silicone tube, connected to the optical detector. Since the TPU fibers are bigger, the combiner is a commercial 3:1 optical fiber combiner (Industrial Fiberoptics part 97638-001, Industrial Fiberoptics Inc). With that in

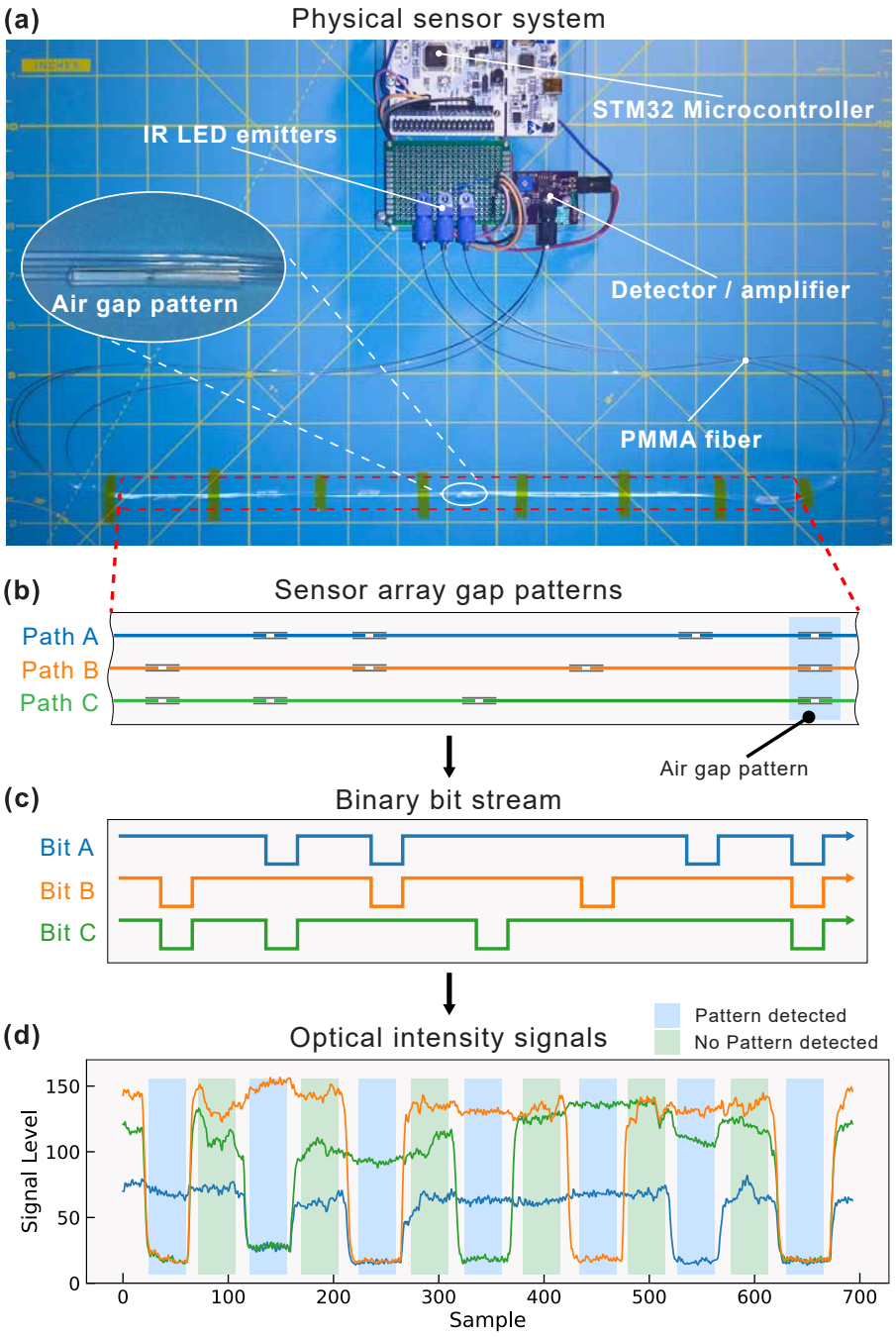


Figure 16. An OptiGap sensor with a visual explanation of the sensor operation. (a) All the physical components of an OptiGap sensor including the emitters and detector, the microcontroller, and the fibers with the sensor array gap pattern. (b) The air gap pattern form bit patterns equivalent to an inverse Gray code binary word sequence that translates to (c) a corresponding bit stream pattern similar to an encoder pattern. (d) Since the real-world bit stream is not a consistent absolute signal, a GNB classifier is used to identify the active patterns.

mind, a number of key properties that affect the behavior and usability of an OptiGap sensor for different applications are highlighted in the following sections. These properties are also summarized in Table 4.

3.2.1 Light Pipe Material

The light pipe material is the most important property because it greatly influences the total length of the sensor, the flexibility of the sensor, and the ability of the sensor to be embedded in various media. The chosen light pipe needs to have a very optically clear core with good flexibility. PMMA fiber was found to be the most versatile light pipe due to its great optical properties, availability in various diameters, and general flexibility without breaking. Specifically, part numbers CK-20 and CK-30 from Industrial Fiber Optics with nominal diameters of 0.5 mm and 0.75 mm, respectively, were used. Both fibers have a fluorinated polymer cladding, a core refractive index of 1.49, and a numerical aperture of 0.5.

3.2.2 Air Gap Sleeve Material

The material covering the air gap needs to be soft enough to allow bending to occur at the gap but not too soft that it crumples easily. This ideal softness was found to to equate to a Shore hardness of about 55A. The diameter of the sleeve material is not critical as long as it is smaller than the light pipe diameter so that it can firmly grip it. Testing found that high-temperature silicone tubing, such as McMaster-Carr part numbers 51845K66 and 51845K67, perfectly satisfies these requirements.

3.2.3 Optical Source, Detector, and Microcontroller

The choice of optical source and detector is not as critical as the previously mentioned design considerations, which is one of the strengths of this system. While any type of light source can be used in an OptiGap sensor, light in the IR spectrum is preferred

for most sensor applications. Because of the gaps created in the light pipes, it is possible for ambient light to enter the system and raise the overall optical noise floor. Using a light source and detector in the IR spectrum provides much-needed immunity to ambient light, allowing the sensor to work in more conditions. The primary requirement for the microcontroller is for it to have an analog-to-digital converter (ADC) and as many digital outputs as optical paths in the sensor.

3.3 Signal Processing

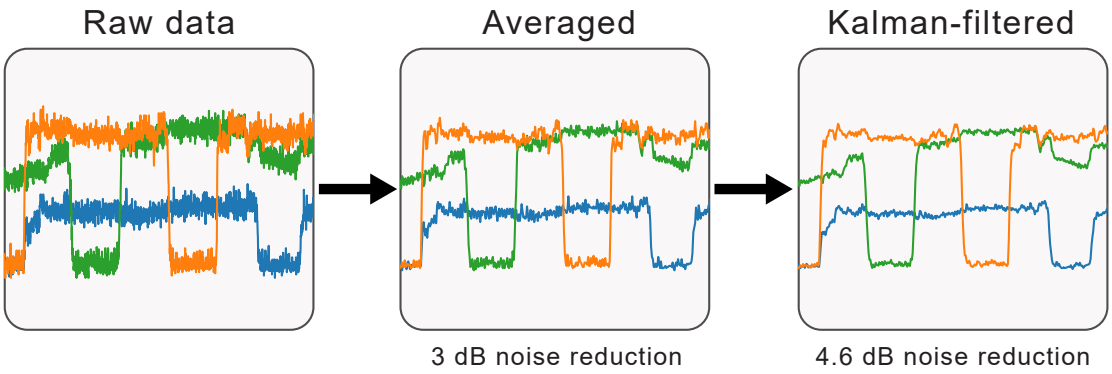


Figure 17. The signal processing stages for noise reduction of the input signal. First, the signal is averaged, which produces a 3 dB noise reduction, and then it is Kalman-filtered for an additional 4.6 dB noise reduction.

Before being sent to the GNB classifier, the signals go through a two-stage noise reduction signal processing chain shown in Figure 17. First, a simple averaging filter given by

$$\bar{A} = \frac{1}{n}\sum_{i=1}^{n} a_i$$

where n is the number of values to be averaged, is applied to the data. This filter serves both to smooth the raw data as well as stabilize the readings for the classifier by providing a slight delay. After the initial smoothing, a Kalman filter is used to further reduce the remaining signal noise without introducing any significant delay in the signal chain. A Kalman filter is an optimal and recursive algorithm that can estimate

a target value using both a current measurement as well as the *a priori* knowledge about the system [73]. These filters are especially suitable for this application due to both being computationally efficient and simple enough to run on a microcontroller.

3.4 Classification

Machine learning techniques have become important in the classification of real-time sensor data across many domains [74–79]. Tan et al. [74] introduce a modified Long Short-Term Memory (LSTM) network for detecting heel strikes and toe offs in gait cycles. When tested against the Movement Analysis in Real-world Environments using Accelerometers (MAREA) database, this method shows improved results compared to six other gait event detection algorithms. Other techniques include Vu et al.'s development of the Exponentially Delayed Fully Connected Neural Network (ED-FNN) for gait cycle percentage prediction [75], Khandelwal et al.'s DK-TiFA methodology for Initial Contact event estimation from accelerometers [76], and Su et al.'s use of the Deep Convolutional Neural Network (DCNN) for gait cycle segmentation with IMU data [78].

Algorithm 1 Logistic Regression Classification

Training Algorithm
Initialize weights $\mathbf{w}$ and bias b
repeat
    for each data point $(\mathbf{x}, y)$ **do**
        Calculate $z = \mathbf{w} \cdot \mathbf{x} + b$
        Calculate $\hat{y} = \frac{1}{1+\exp(-z)}$
        Update $\mathbf{w}$ and b using gradient descent
until convergence
Classification Algorithm
for each new data point $\mathbf{x}$ **do**
    Calculate $z = \mathbf{w} \cdot \mathbf{x} + b$
    Calculate $\hat{y} = \frac{1}{1+\exp(-z)}$
    Classify $\mathbf{x}$ as 1 if $\hat{y} \geq 0.5$, else 0

Logistic regression and random forest are both popular machine learning algo-

rithms used for binary classification tasks due to their simplicity and lack of complexity when compared to the aforementioned neural network approaches. Logistic regression is a linear model that estimates the probability of a binary outcome based on one or more predictor variables [80]. On the other hand, random forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) of the individual trees for prediction. It effectively addresses the overfitting problem seen in individual decision trees by averaging out biases and capturing the underlying patterns in the data [81]. While logistic regression assumes a linear relationship between predictors and the log odds of the outcome, random forest makes no such assumption, allowing it to capture complex, non-linear relationships in the data. Bahel et al. in [82] provide a comparison of binary classification algorithms, highlighting the extreme effectiveness of the random forest classifier.

Algorithm 2 Random Forest Classification

Training Algorithm
for $i = 1$ to N trees **do**
    Bootstrap sample D_i from the original dataset D
    Build a decision tree T_i on D_i
    For each split, randomly select m features without replacement
    Choose the best split based on information gain or Gini impurity
Classification Algorithm
for each new data point **x do**
    Initialize votes = {}
    for each tree T_i **do**
        Classify **x** using T_i to get c_i
        Add c_i to votes
    Classify **x** as the class with the majority votes

3.4.1 Naive Bayes Classifier

A GNB classifier is a supervised learning algorithm based on Bayes' theorem that determines how a measurement can be assigned to a particular class, C_i, assuming

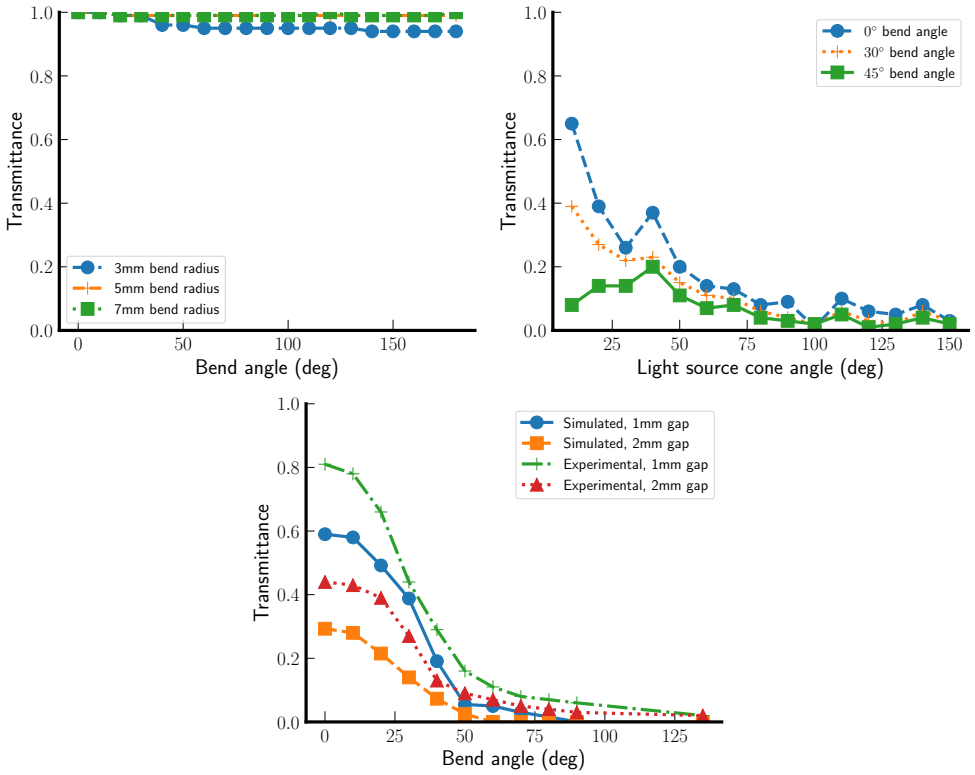


Figure 18. Simulation and experimental results. (a) The effects of bending a 1 mm piece of PMMA fiber on transmittance. A minimal drop-off starts at around 40°. (b) The cone angle of the light source has a noticeable impact on transmittance. The smaller the cone angle the higher the transmittance. There is extra sensitivity at 40° during the downward trend. (c) A comparison of simulated and experimental data of the bend angle and transmittance. The simulation and experimental data follow the same curves even though the experimental data has a higher offset.

each class follows a Gaussian (normal) distribution with a certain probability $P(C_i)$. The *naive* part of the name assumes independent random variables. The operation of the GNB classifier hinges on two key questions: (1) How can a measurement, x, be assigned to class C_i for a given distribution? (2) What is the probability of error in that assignment?

The answer to the first question has an intuitive start: given any number of classes, a measurement should *most likely* belong to the class that has the highest probability of occurring. This means that, assuming the class follows a Gaussian distribution, a

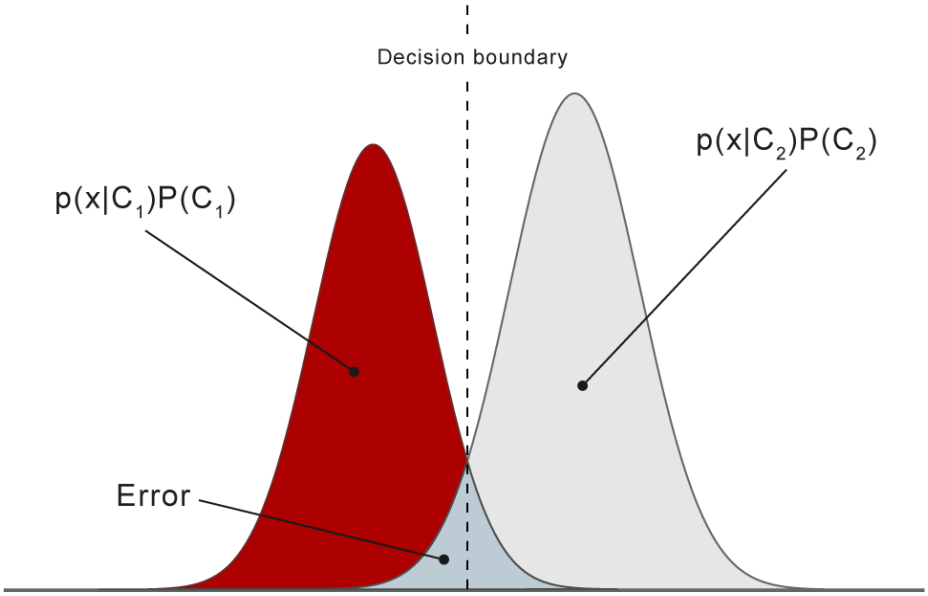


Figure 19. A visualization of a binary Bayes classifier shown the decision boundary and classification error.

measurement x belongs to class $C_i, \in [1, M]$ when

$$max\Big\{ f_x(x|C_i)P(C_i) \quad \forall \quad i \in [1, M] \Big\}$$

with the Gaussian probability density function given by [83]

$$P(x|C_i) = \frac{1}{\sqrt{2\pi\sigma_{c_i}}} \exp\left(-\frac{(x - \mu_{c_i})}{2\sigma_{c_i}} \right)^2 \tag{5}$$

Addressing the second question, the probability of error is the probability of a measurement, x, from one class being misclassified as belonging to a different class. Given a two-class problem, the total probability of error is defined as

$$\begin{aligned} P_{err} = P(x|C_2)P(C_2)\ x \in C_1 \\ +P(x|C_1)P(C_1)\ x \in C_2 \end{aligned} \tag{6}$$

A GNB was used in this OptiGap implementation because it is more efficient than if-statements or lookup tables, can handle new or previously unseen data, and can be more accurate by taking into account the relationships between multiple input variables.

Algorithm 3 Gaussian Naive Bayes Classifier

Training Algorithm
for each class c **do**
    for each feature j **do**
        Calculate $\mu_{c,j} = \frac{1}{N_c} \sum_{i=1}^{N_c} x_{i,j}$
        Calculate $\sigma_{c,j}^2 = \frac{1}{N_c} \sum_{i=1}^{N_c} (x_{i,j} - \mu_{c,j})^2$
    Calculate $P(c) = \frac{N_c}{N}$
Classification Algorithm
for each new data point x **do**
    for each class c **do**
        $P(c|x) \leftarrow P(c)$
        for each feature j **do**
            $P(x_j|c) = \frac{1}{\sqrt{2\pi\sigma_{c,j}^2}} \exp\left(-\frac{(x_j - \mu_{c,j})^2}{2\sigma_{c,j}^2}\right)$
            $P(c|x) \leftarrow P(c|x) \times P(x_j|c)$
    Classify x as $\hat{c} = \arg\max_c P(c|x)$

3.4.2 Fitting and Prediction

Fitting of the training data to a usable GNB classifier was performed using the GaussianNB module of the popular Scikit-learn Python library. In order to fit the model, data for each air gap pattern had to be captured from the sensor, labeled appropriately, then sent to the GaussianNB module. The output of the fitting process was a set of parameters representing the various Gaussian distributions and probabilities for each pattern as determined by the sensor data. The actual prediction was performed by a GNB classifier on the STM32 microcontroller, implemented using the Arm CMSIS-DSP C library which contains common signal processing functions. This allowed the microcontroller to perform real-time predictions while offloading the compute-intensive fitting process to a desktop computer.

3.5 Testing and Results

Testing methodology involved initial validation of the model and proposed air gap concept via simulation using COMSOL Multiphysics, followed by physical construction, characterization, and testing of system variations. Ray optics simulations were

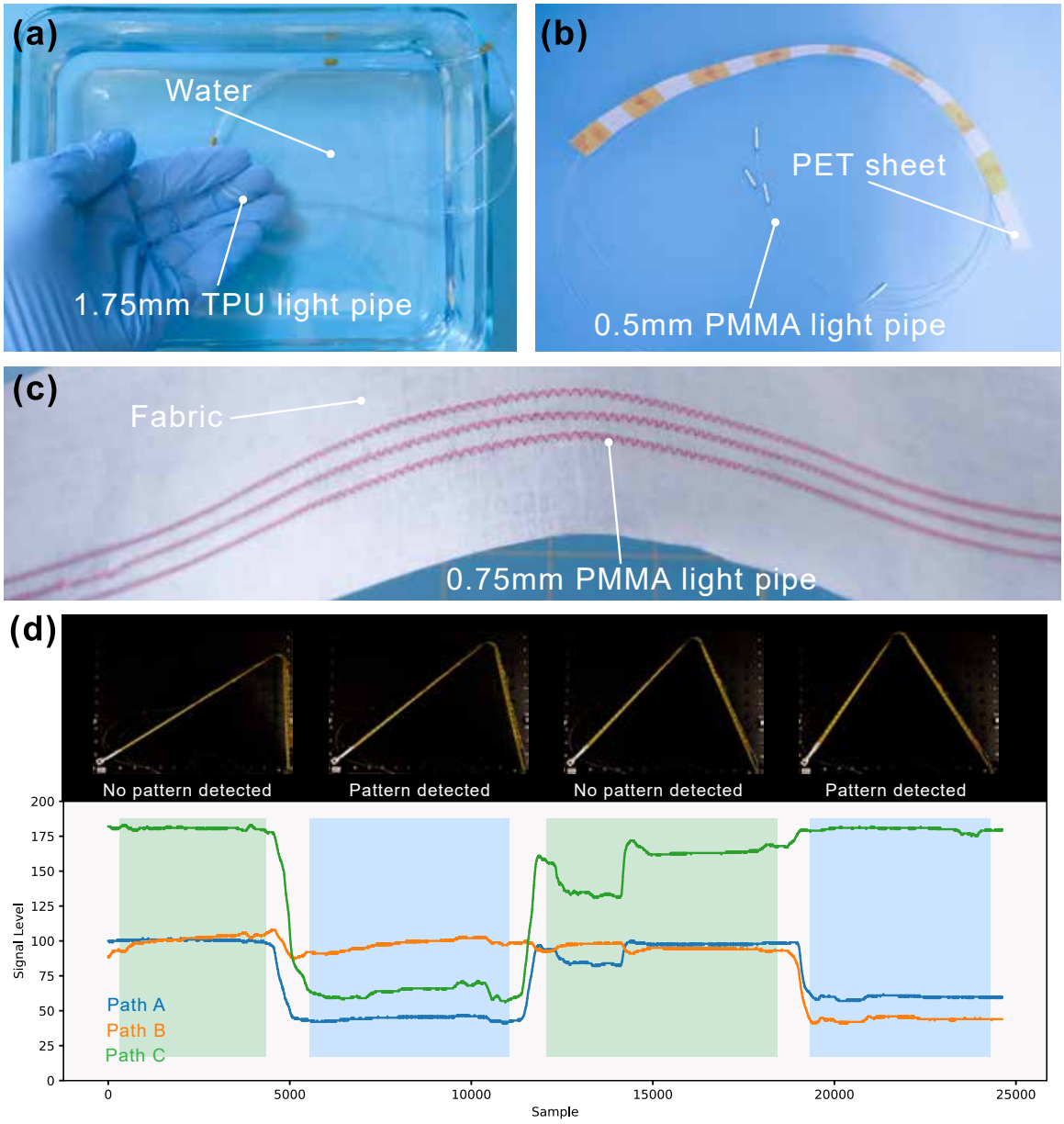


Figure 20. Experimental setup and results. (a) OptiGap sensor working underwater. (b) The sensor attached to PET. (c) OptiGap light pipes embroidered into fabric. (d) 0.5 mm PMMA sensor attached to a tape spring testing rig showing how the bend location corresponds to the air gap pattern.

chosen over wave optics due to the OptiGap system's design to operate with multiple materials; the fiber functions as a generic light pipe and is not material-dependent. Simulation included bending a PMMA fiber piece with air gaps of 1 mm and 2 mm, as well as a fiber piece without an air gap for reference. To accommodate various

light source lenses, simulation also explored the impact of altering the cone angle of the light source. Experimental testing of fiber bending involved creating and printing a CAD pattern on paper containing outlines of all test bend angles. Fiber alignment atop this paper for each bend angle ensured consistent and repeatable testing. Transmittance served as the primary metric for all tests and is given by

$$T = \frac{I}{I_0} \tag{7}$$

and is the ratio of transmitted light to incident light.

3.5.1 Bend Angle

The air gap significantly reduces transmittance as the bend angle increases. Bending a fiber with no air gap results in minimal loss of light as the simulation in Fig. 18a shows. For the smallest 3 mm bend angle, a 180° bend resulted in only a 6% drop in transmittance. Conversely, adding an air gap results in a 75% drop by 50° and effectively a total loss in transmittance by 125°, which is shown in Fig. 18c. The experimental results in the same figure match the simulation in terms of percentage drop. For both experimental and simulation, the drop in transmittance starts accelerating at 20° with most of the light attenuated by 50°. This establishes 20° as the working angle for a recognizable bend *for this particular sensor*, which is defined as the minimum bend angle required to reliably detect a bend. The offset between the experimental and simulation results is due to the simulation optical source not matching the intensity of the experimental source LED.

3.5.2 Gap Length

Fig. 18c also shows that the effective **bend sensitivity** of the sensor can be changed by changing the gap length, which in turn changes the min working angle. An increase in **gap length** increases the bend sensitivity at the cost of lower transmittance of the entire fiber. This can be alleviated by using a more powerful light source.

Table 4. OptiGap Sensor System Properties & Parameters

Name	Has An Effect On
Bend Sensitivity	Min bend angle for intensity drop
Diameter	Total transmittance based on light source
Gap Length	Bend sensitivity
Sensing Resolution	Minimum length between gaps

3.5.3 Diameter

Simulation results in Fig. 18b show that small cone angles produce the highest transmittance. This suggests that the **diameter** of the light pipe has a direct effect on the optical intensity, and thus maximum sensor distance. Testing indicates the transmittance can be maximized by using a light source with the smallest possible cone angle if multiple diameter fibers are going to be used, or by matching the light source to the fiber diameter if a single diameter is used for the application.

3.5.4 Sensing Resolution

The bend sensitivity also dictates the **sensing resolution** since only one air gap pattern should be bending at a time. The sensing resolution is defined simply as the minimum gap-to-gap spacing. Testing suggests that 5 cm is a reasonable resolution for most materials.

3.5.5 OptiGap GUI

To make the system easily usable and reconfigurable, a GUI (Fig. 21) was developed that enables quick data gathering from the sensors, interactive data labeling, and model fitting. Built in Python, it uses the PyQt6 framework for the graphical user interface and pyqtgraph for plotting capabilities. The software offers a host of functionalities, as outlined below:

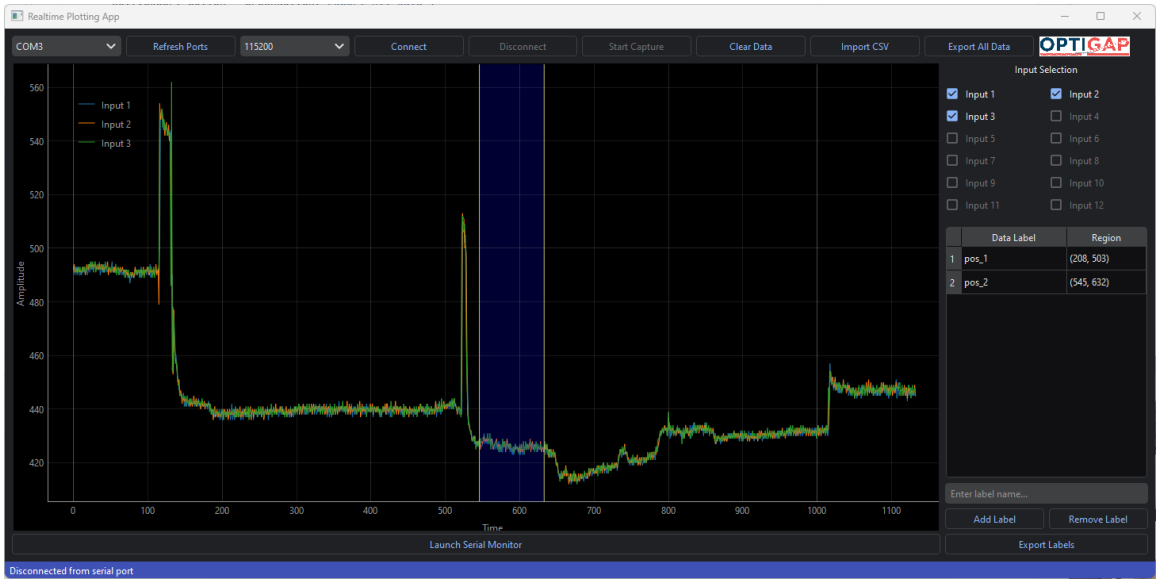


Figure 21. GUI created to facilitate fast testing, labeling, and fitting of data for the OptiGap system.

- Serial Port Configuration: Upon launching the program, users can navigate to the toolbar to configure the serial port. The `Serial Port Configuration` dialog allows setting parameters like Port and Baud Rate. Clicking `Connect` initiates the connection, and a successful link-up is indicated via a "Connected" status message.

- Data Capturing: The `Capture` button on the toolbar toggles data acquisition from the serial device. When activated, the software captures incoming data and updates the plot in real-time.

- Plot Customization: The right-hand panel of the GUI presents checkboxes for each input signal. Users can activate or deactivate these to customize the signals that appear on the plot.

- Data Labeling: A region selection tool, identifiable as two vertical lines on the plot, permits users to demarcate a range of data for labeling. This region can be adjusted manually by dragging the vertical lines or using the handles at the

extremities. After entering a label name, the `Add Label` button inserts this label into a table for future reference.

- Label Management: For existing labels, the `Remove Label` button allows users to select and delete them from the labels table.

- Data Import/Export: The software supports importing data from CSV files via the `Import CSV` button. Conversely, captured data and labels can be exported as CSV files using the `Export All Data` and `Export Labels` buttons, respectively.

3.5.6 Full System and Classifier Performance

After validating the model and gaining a better understanding of the effects of various properties on transmittance, the full OptiGap sensor system was validated. Initial tests of hand bending the sensor and verifying the classification were performed. For repeatability, an automated bending rig (Fig. 20d) was created, consisting of a tape spring arm with a servo on one end and a free-spinning shaft on the other end.

Results (Fig. 20d) show the output tracked with the location of the bend along the arm, showing the bend-sensitive areas and the non-sensitive areas. While there is no single accuracy metric for the system as a whole since that is entirely dependent on the fitting and fabrication of a particular configuration, this configuration tested had a 100% accuracy measure. However, one potential source of error is little separation between low and high signal levels, which can occur if the gap lengths are too large, resulting in transmittance below a usable threshold. Using the STM32 running at 100 Mhz, the sensor was outputting data at 175 Hz using a UART baud rate of 115200.

3.5.7 Sensor Attachment

Various techniques were explored for attaching the sensors to mechanisms, including tapes such as masking, painter's, fabric, and polyimide tape. Polyimide tape was the most versatile especially when sticking the sensor to our tape spring test station. Other techniques included using flexible but stabilizing material like a thin PET sheet shown in Fig. 20b as well as flexible silicone adhesives. Since the motivating factor around the development of the OptiGap system was to have a sensor that could sense the bending of the TASL in the previous chapter [9], various ways of embedding the sensor into the surface of materials are being explored. As Seen in Fig. 20c, a ZSK tailored wire placement machine was used to sew fibers for an OptiGap sensor into a fabric material much like the TASL.

3.5.8 Underwater Sensing

To demonstrate the versatility of the sensor system, a sensor was fabricated using off-the-shelf clear TPU 3D printer filament as the light pipe. This provided an opportunity to test the performance of the sensor system underwater. The sensor, shown in Fig. 20a, had no change in performance when submerged vs in free air, which raises the potential for use in underwater robotics.

3.5.9 Limitations

The primary limitation of the OptiGap system is that it currently can only detect bending at one location at a time, which stems from its basis as a linear encoder. Because OptiGap uses a GNB classifier, it is possible for the sensor to be fooled into thinking a bend is occurring at a particular location when multiple air gap patterns are bent at once. This limits applications to those of linear encoders but with the advantage of being customizable, compliant, and usable in wet conditions. The second

limitation, common to all soft optical sensors, is the inability of the sensor to work in very high-temperature environments due to the risk of melting the fibers.

4 OptiGap Case Study

Following the exploration of the design considerations, behavior, and inherent limitations of the OptiGap sensor system, it was tested in a practical applications in the field of soft robotics. The adaptability of the sensor system to function in varied conditions, such as underwater environments and complex material interfaces, highlights its broad utility. This adaptability becomes particularly significant in the context of soft robots, where mechanical transparency and real-time adaptability are essential. The ensuing application involving a twisted soft beam serves as a case study to demonstrate the integration of the OptiGap system into soft robotic structures, thereby offering a robust solution for capturing real-time, high-frequency dynamics essential for adaptive behavior.

Soft robots need onboard configuration sensing that doesn't affect their overall mechanics. Mechanical transparency is especially important in cases where the robot's function depends closely on its material properties. This case study explores a twisted soft beam (Fig. 22) developed by fellow collaborators [1] that converts simple, periodic input motion into complex cyclic movements upon ground contact. Utilizing a soft, twisted beam subjected to linear vibratory input, a repeating, semicircular trajectory at the beam's tip is generated. When this motion interacts with the terrain, it evolves into a more intricate movement pattern suitable for robotic walking. The experiment showcases that variables such as contact frequency, motion direction at the contact point, and the resulting motion path can be modulated by adjusting the input frequency [1]. For empirical validation, a test setup was constructed featuring a linear stage driven by the oscillating motion of a brushless motor, controlled by an ODrive motor control board. The beam was affixed to this linear stage, and optical

tracking markers were placed at its proximal and distal ends. An OptiTrack Prime 17W system was used to monitor the system's position at a 360 Hz rate. Additionally, a plate equipped with four load cells measured the contact forces between the leg and the ground along both Y and Z axes. The beam sample used had a twist angle of 90 degrees, and a 20g load represented the foot's mass. The rigid foot length was 66.5 mm, and the distance between the translational stage and the plate was set at 72 mm. This setup ensured a fixed contact distance of 5.5 mm between the foot in its unloaded, natural position and the plate. The experiment effectively demonstrated how walking direction and speed could be tuned by manipulating the frequency of the input actuator [1].

These gaits, as earlier mentioned, depend on beam loading, surface contact, and properties of the surrounding environment such as viscosity and density – all of which may change during run time. As a result, this case study focuses on the twisted soft beam's real-time dynamics. To capture these dynamics, three OptiGap sensors are strategically placed (Figure 23) on the beam: one along the top edge, another across the front, and the third across the back face at a 90-degree angle to the front sensor. These placements are designed to capture the beam's vibrational characteristics as it interacts with its environment. As the beam vibrates within the 1 Hz to 45 Hz range, the sensors record optical intensity data, which is then correlated with position data of the beam's tip captured by precise motion tracking system. This dataset, comprising x, y, z tip positions and optical intensity readings from the three sensors, serves as the foundation for the machine learning approach utilized in this case study. By training machine learning models on this data, the aim is to predict the direction of the beam's motion (forward or backward) at its point of contact, offering a unique method for real-time bend sensing in flexible structures.

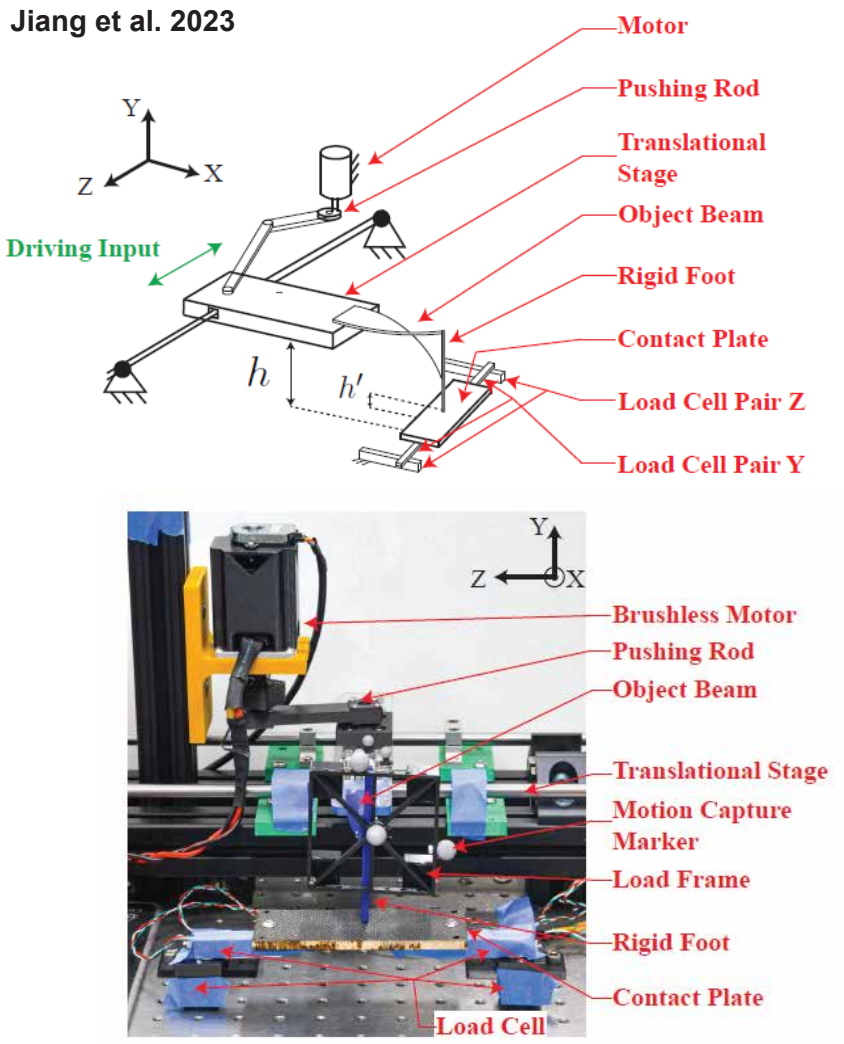


Figure 22. Lab test setup [1] used to evaluate an OptiGap sensor.

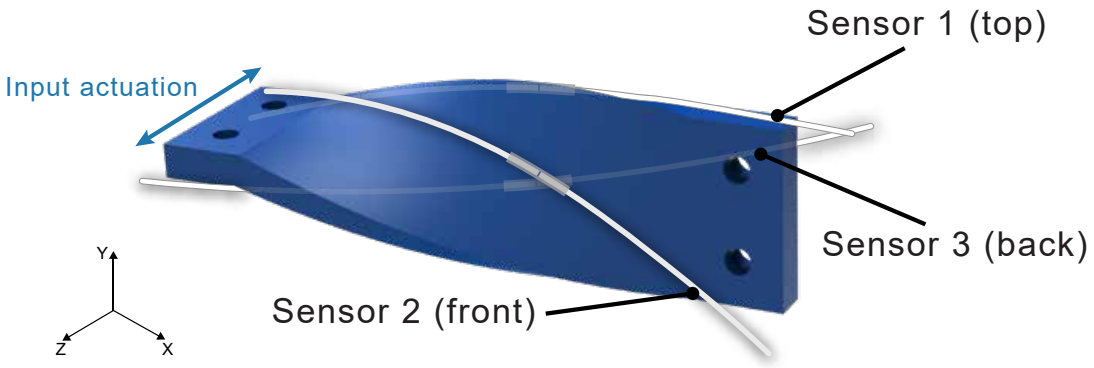


Figure 23. Positions of the 3 OptiGap sensors. Sensors 2 and 3 intersect in the middle on each side of the beam while sensor 1 is placed along the top edge.

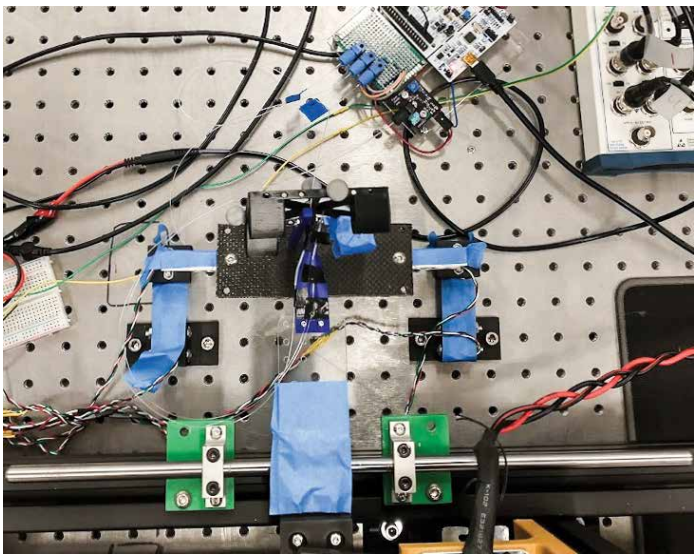


Figure 24. Photograph of vibrating beam with OptiGap sensors using the same test setup from [1] showing the translational stage, rigid foot, optical tracking markers, and 50 g mass.

4.1 Determining Contact Direction using Real-time Classification

This sensor application employs 500 um Polymethyl methacrylate (PMMA) optical fiber sensors integrated into the OptiGap system. Three fibers, featuring small air gaps enclosed by a flexible sleeve as core sensing elements, are strategically placed on the surface of the twisted soft beam. As first introduced, the OptiGap system utilizes these air gaps in flexible optical light pipes to create coded segments for bend localization but in this application, the OptiGap system is used to capture dynamic behavior and vibrations in the beam, particularly when oscillated horizontally at frequencies ranging from 1 Hz to 40 Hz.

The main objective is to construct a real-time binary classification model capable of determining the direction of motion at the point of contact (with the two classes being "forward" and "backward") for the beam, utilizing data from the three OptiGap sensors. Logistic regression (LR) serves as the baseline model, providing a straightforward yet effective starting point for classification. This is followed by the implementation of a random forest (RF) model, which offers a more effective ap-

proach to capturing complex data patterns. Given the sequential nature of the data, temporal features need to be extracted from the data which in this case is achieved by calculating a moving average (eq. 8) with a window of 5

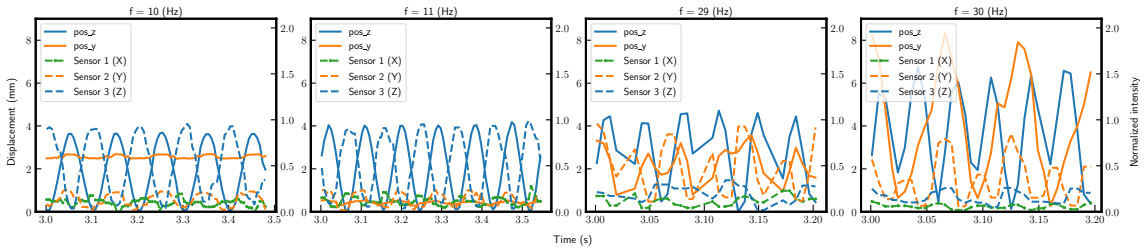


Figure 25. The correlation between the position displacement and sensor intensity over 6 cycles at selected frequencies.

$$\mathrm{MA}(s_i, t, w) = \frac{1}{w} \sum_{j=t-w+1}^{t} s_{i,j} \tag{8}$$

where s_i is a sensor reading and w is the window size, and calculating the gradient using the central difference at each time step (eq. 9)

$$\begin{aligned} g(i) &= \frac{x(i+1) - x(i-1)}{2} \\ g(0) &= x(1) - x(0) \end{aligned} \tag{9}$$

where $g(i)$ is the gradient index at i and x is the input data array. In general, a small window will better capture quick changes in the data, at the cost of noise. The optimal choice for the window size is a compromise between the sampling frequency and driving frequency, where a high driving frequency would benefit from a smaller window to better capture rapid changes. Data preprocessing steps also involve normalization and partitioning of the data into training, validation, and test sets. Finally, another random forest model is evaluated that utilizes a time-lagged input (TL RF) from the raw sensor data

$$\begin{aligned} X &= [x_1, x_2, \ldots, x_N], \quad \text{where each } x_i \in \mathbb{R}^{15} \\ x_i &= [s1(t_i), \ldots, s1(t_{i+4}), s2(t_i), \ldots, s3(t_{i+4})] \end{aligned}$$

where each input feature vector X is a concatenation of 5 sequential data points from the three sensors. x_i is the i^{th} feature vector, comprised of sequential data points from sensors $s1$, $s2$, and $s3$.

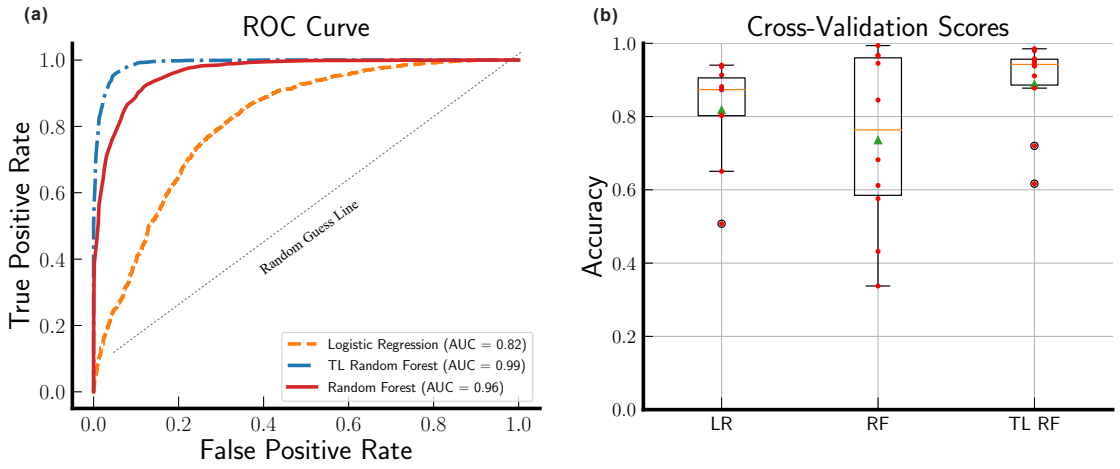


Figure 26. **(a)** The 10-fold cross validation results also show the TL random forest model as the best performer with a much tighter distribution than the rest. **(b)** The area under the curve (AUC) quantifies the models overall performance, with the TL random forest model showing the best performance of the three.

4.2 Experimental Setup

This experiment follows the methodology used in [1] to show how the output trajectory of a soft twisted beam can be influenced by the driving frequency. The same hardware and experimental setup is used, but with a 50 g weight. The beam is set to oscillate horizontally and the OptiTrak Prime optical motion tracking system captures the position data of the tip of the beam, which is then correlated with the OptiGap sensor readings. The main data outputs consist of x, y, z tip positions and three optical intensity readings at each sample point. These are recorded as the beam vibrates at frequencies ranging from 1 to 40 Hz.

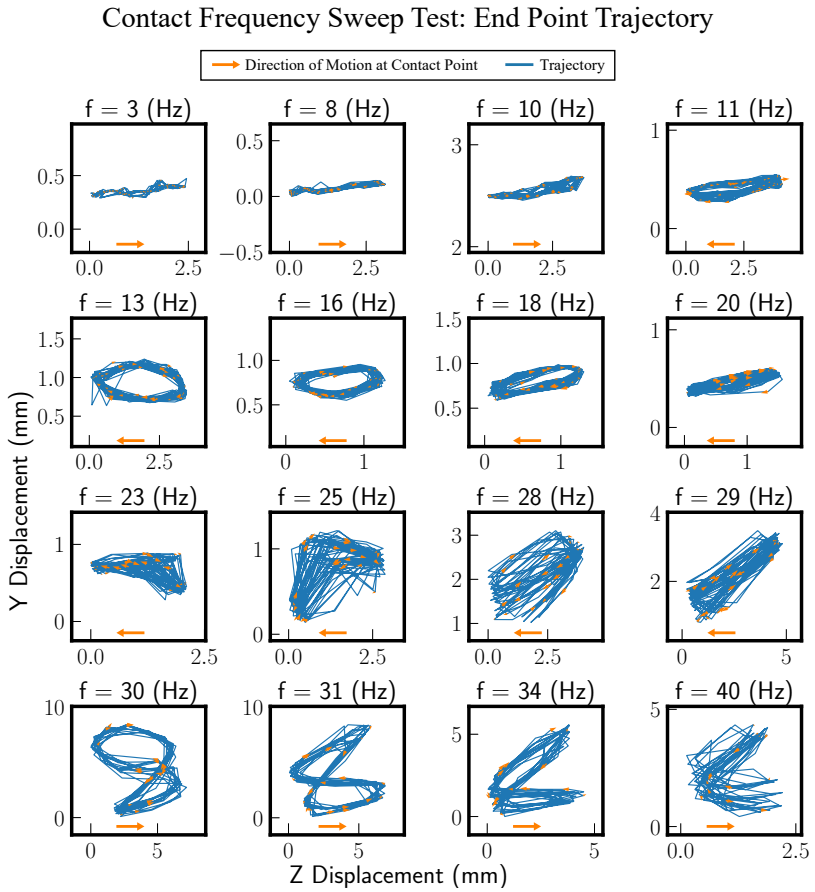


Figure 27. Single beam contact test results. End point trajectories for selected frequencies with arrows showing the direction of motion at the contact point.

4.3 Results and Discussion

The single beam contact test reveals distinct end-point trajectories at selected frequencies, as illustrated in Figure 27, which mimic the trajectories presented in [1]. Arrows indicate the direction of motion at the contact point, providing insights into how the twisted beam interacts with the ground. This data is further substantiated by Figure 28, which presents a Fourier transform analysis of the optical intensity readings from the OptiGap sensors. The dominant frequencies align well with the system input frequencies, validating the sensor's capability to accurately capture the vibrational characteristics of the twisted beams. This frequency analysis further supports the reliability of the OptiGap system in real-time sensing applications. Figure

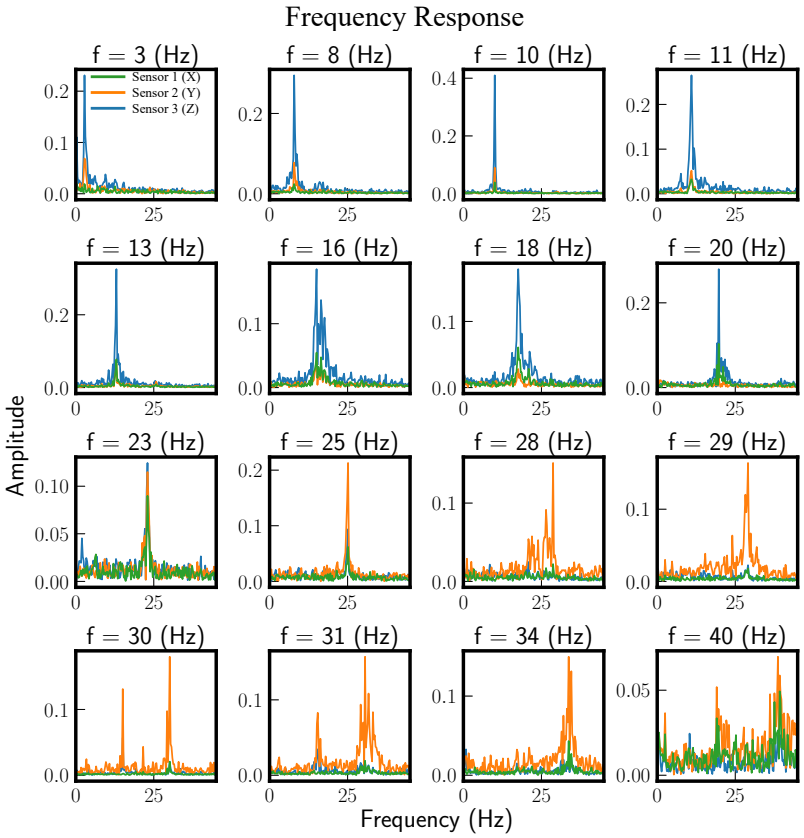


Figure 28. Frequency analysis. Fourier transform of the optical intensity readings from the OptiGap sensors. The dominant frequencies match the system input frequencies, validating the sensor data.

29 integrates the normalized optical intensities from the three OptiGap sensors with the endpoint trajectories at selected input frequencies.

Table 5. Performance Metrics

Metric	**Formula**
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
Area Under the Curve (AUC)	Integral area

The antagonistic placement of sensors 2 and 3 reveals interesting behavior of the beam's motion, offering a better understanding of how sensor placement can influence usefulness of the data. Figure 25 presents a cycle analysis, plotting six cycles for the

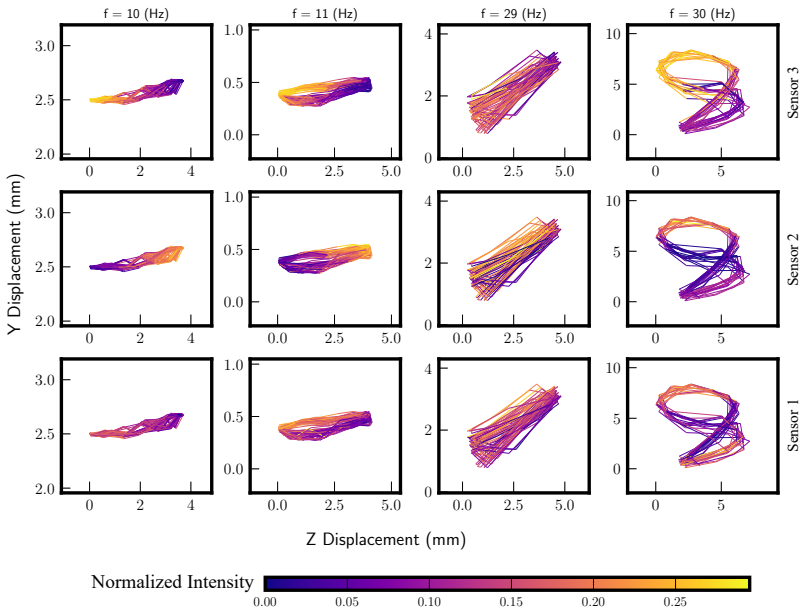


Figure 29. OptiGap sensor intensities. The normalized optical intensities of the three sensors overlaid over the endpoint trajectories of selected input frequencies, especially showing the effects of the antagonistic placement of sensors 2 and 3.

y and z positions, and normalized optical intensity. This figure reveals a strong correlation between each sensor's optical intensity and the beam's positional data, providing a temporal dimension to the sensor's capabilities. This cycle analysis not only validates the sensor's real-time performance but also opens the door for advanced control algorithms that can adapt to dynamic changes in the robot's environment or operational parameters.

Table 6. Classification Results

Metric	Logistic Regression	Random Forest	TL Random Forest
Accuracy	75%	90%	95%
Precision	75%	90%	96%
Recall	74%	90%	95%

4.3.1 Binary Classification

The evaluation of the three models reveals varying levels of performance. The classification report in Table 6 indicates that the logistic regression model achieves an

accuracy of 75%, precision of 75%, and recall of 74%. In contrast, the random forest model demonstrates an accuracy of 90%, precision of 90%, and recall of 90%. The TL RF model outperforms both, with an accuracy of 95%, precision of 96%, and recall of 95%. In a 10-fold cross-validation assessment shown in Figure 26(b), the logistic regression model yields a mean accuracy of 81.82% with a standard deviation of 13.16%. The random forest model's mean accuracy stands at 73.56% with a standard deviation of 22.88%, while the TL RF model achieves a mean accuracy of 88.86% with a standard deviation of 11.63%. The ROC curve analysis in Figure 26(a) further substantiates these findings, with the AUC values being 0.82 for logistic regression, 0.96 for random forest, and an impressive 0.99 for the TL RF model.

5 Conclusion

This chapter presented and demonstrated the OptiGap sensor system, a novel sensor system that is low cost, flexible, simple to fabricate, able to perform real-time sensing, and can work on any modern microcontroller. Validation through simulation and experimentation supports the approach of using an air gap with a flexible sleeve to create bend-sensitive air gap patterns in a light pipe. Potential applications span various robotics and automation systems requiring flexibility, reconfiguration, and noise immunity, including underwater conditions. The versatility in light pipe material selection based on application needs gives the OptiGap system a distinct advantage over other sensors. This chapter also demonstrated the use of OptiGap in a real sensing application in embedded monitoring of twisted beam dynamics and the application of machine learning for classifying forward vs reverse gaits. Results highlight the ability of OptiGap to be repurposed, including by applying different algorithms in the processing stage.

6 Design Manual

The ultimate aim for this sensor system involves integration of processing into a compact custom hardware package and development of a robust software package and toolchain for automated fitting of new sensor patterns. To that end, a design manual has been developed and included in Appendix .

CHAPTER IV

DISTRIBUTED ACTUATION AND SENSING

1 Introduction

This chapter focuses on the integration of EneGate, a system designed to further the modular capabilities initially realized in TASL and later in OptiGap. EneGate aims to create an integrated, modular framework where sensing and actuation components can be customized and scaled according to needs. As shown in Figure 30, it serves as a complementary technology to TASL by adding local control functionalities and to OptiGap by offering a modular platform for sensing that is adaptable for a wide range of applications.

The design philosophy behind EneGate is influenced by the foundational work on TASL, which served as the initial inspiration for reconfigurable actuation and later led to the development of OptiGap. Building on these foundations, EneGate aims to advance the concept of modular and decentralized actuation in robotics by essentially breaking up the continuous TASL into discrete TASL SMA "patches" that have the same behavior but can attached to any SCRAM-like device. These patches can then be energized by an EneGate node. This allows anyone without access to the specialized ZSK equipment to implement devices like the TASL by using these patches. EneGate also seeks to complement and enhance the modular features introduced by OptiGap by allowing OptiGap sensors to connect to an EneGate node or between two EneGate nodes. This effectively solves the scalability issues of OptiGap and gives it even more modularity.

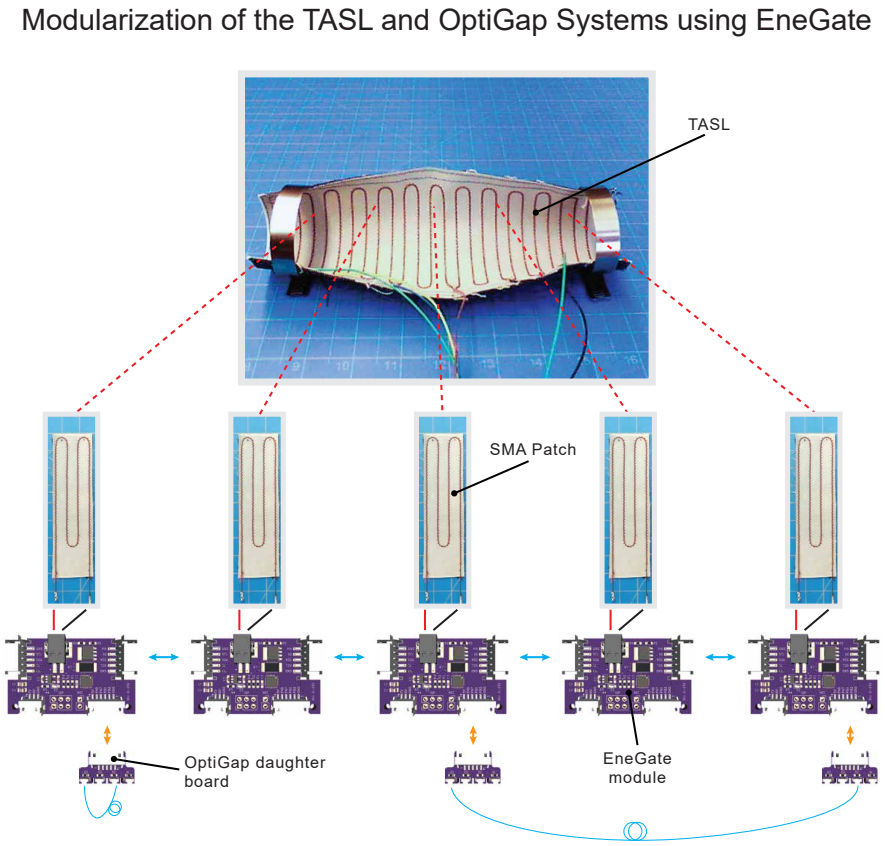


Figure 30. The full realization of a modular framework enabled by EneGate.

1.1 Design Requirements

Modularity is a central feature in this context, providing systems with flexibility, scalability, and adaptability. Such a modular design allows for the easy assembly, disassembly, and reconfiguration of components, making the system versatile for a variety of tasks and environments. EneGate is designed with a number of key requirements drawn from the experience of developing both TASL and OptiGap. These requirements are outlined in Table 7 and are divided into several categories including power management, control and monitoring, connectivity, and protection and feedback.

The following sections detail the design specifics of EneGate, including its communication protocols. The ability of EneGate to act as a controller for the surface embedded SMA actuators in TASL are also discussed. Finally, the compatibility between EneGate and OptiGap is examined to assess how these technologies can be

Table 7. Design Requirements

Category	Requirement	Reference
Power Management	Voltage Stability: Maintain a stable power source for all components.	Minimal functional requirement.
	MOSFET Switch: Control a high current load.	TASL needs MOSFET actuation (Chapter 2)
Control and Monitoring	Microcontroller: Serve as the central control unit.	Minimal functional requirement.
	Current Monitoring: Accurate monitoring of current through the MOSFET.	TASL is current-controlled (Chapter 2)
	Temperature Sensing: Monitor temperature levels of components.	TASL actuation force is temperature dependent (Chapter 2)
Connectivity	Daisy-Chaining: Ability to connect multiple boards in series.	TASL and OptiGap modularization (Chapters 2, 3)
	Communication Interface: Data extraction and real-time communication.	OptiGap sensor data (Chapter 3)
	Microcontroller Programming: Update microcontroller's firmware.	Minimal functional requirement.
Protection and Feedback	Operational Feedback: Visual indicator of board's status.	Minimal functional requirement.
	Back-powering Protection: Prevent inadvertent power flow between boards.	Minimal functional requirement.
	Over-current Protection: Limit current going through MOSFET.	TASL and minimal functional requirement.

integrated to provide a comprehensive modular solution for both sensing and control.

2 Communication Protocols

The choice of communication protocol can significantly impact the system's performance, reliability, and complexity. In distributed actuation systems, low-latency and high-reliability communication protocols are often preferred to ensure real-time control and coordination between the modules. One commonly used protocol in such applications is the Inter-Integrated Circuit (I2C) protocol, which allows for multiple slave devices to be controlled by a single master device. I2C is particularly well-suited for applications where the actuation units are closely spaced and require simple, two-

wire connections. For scenarios requiring longer-distance communication or higher data rates, the Universal Asynchronous Receiver-Transmitter (UART) protocol may be more appropriate. UART allows for full-duplex communication between devices and is generally easier to implement in hardware. In the context of the SCRAM project, I2C could be used for intra-node communication, while UART could be employed for interfacing with a central controller or external systems.

2.1 Manual Control Protocol

The EneGate board employs a simple yet effective serial communication protocol for manual control by an operator. Operating at a baud rate of 38400, the protocol employs a structured ASCII string format enclosed within start and end bytes, specifically curly braces `{ }`. This format is designed to facilitate more precise control by allowing commands to target specific nodes within a distributed system. This approach offers the dual advantages of simplicity and robustness, making it well-suited for a wide range of applications, from research and development to real-world deployments. The protocol's straightforward design also facilitates rapid integration into existing systems and software frameworks, thereby accelerating the development cycle.

The protocol supports a variety of commands, each serving a specific function in the control and monitoring of the EneGate board. Each command string follows the format `{P,1,20}`, where `P` represents the command, `1` specifies the target node ID, and `20` is the value to be set. This structured approach provides several advantages. First, it allows for the direct targeting of individual nodes, enabling more granular control. Second, the inclusion of start and end bytes adds an additional layer of error checking, making the protocol more resilient to noise and data corruption. For instance, the `P` command is used to set the Pulse Width Modulation (PWM) percentage, effectively disabling the Proportional-Integral-Derivative (PID) controller when

Table 8. Manual Control Protocol Commands

Command	Description	Value	Example
P	Set MOSFET PWM percentage. Disables PID controller.	0 - 100	{P,1}
C	Set global current limit. Default is 2 amps.	0.00 - 3.00	{C,2,1.23}
S	Set PID setpoint in amps. 0 disables PID.	0.00 - 3.00	{S,1,1.23}
M	Set output to machine parsable values	M	{M,1}
H	Set output to human-readable format	H	{H,2}
D	Set output to debug mode. Returns full state and raw ADC values.	D	{D,1,D}
T	Turn a GPIO pin on or off	0 (off) / 1 (on)	{T,1,1}
A	Read the analog value of a pin	A	{A,1}
W	Write a PWM value to a pin	0 - 100	{W,1,128}

invoked. This feature provides fine-grained control over the actuation mechanism, allowing for precise tuning and calibration. Similarly, the `C` command sets a global current limit, with the default value set at 2 amps. This safety feature is particularly crucial in applications where excessive current could result in hardware damage or operational failure.

The `S` command sets the PID setpoint in amps, offering another layer of control over the actuation mechanism. A value of zero disables the PID controller, providing a mechanism to quickly deactivate the control loop when necessary. The `M` and `H` commands are used to set the output format to machine-parsable comma-separated values and human-readable format, respectively. These commands are especially useful for data logging, enabling easy interpretation of the board's output. Finally, the `D` command activates the debug mode, which returns the full state of the device along with raw Analog-to-Digital Converter (ADC) values. This mode is particularly useful for troubleshooting and performance optimization, providing deep insights into

the internal workings of the EneGate board.

2.2 Binary Protocol (Node / Machine Communication)

The EneGate board uses a binary protocol to communicate with other EneGate boards and the central controller. This protocol serves as the backbone for the node-to-node interactions within the distributed system or with a host. This protocol is designed to be versatile, facilitating a wide range of commands for system control, data retrieval, and actuation. The protocol uses a structured message format, encapsulated within a data structure, to ensure data integrity and allow for extensible functionalities. A binary message using this protocol follows the form:

`<0x01><0x01><NODEID><COMMAND><TYPE><PAYLOADSIZE><PAYLOAD><CHECKSUM>`

where `<0x01><0x01> (<SOH><SOH>)` is the start frame. The rest of the message is encapsulated by the struct below.

```
typedef struct {
    uint8_t node_id;
    uint8_t command;
    uint8_t type;
    uint8_t payload_size;
    uint8_t payload[32];
} Message;
```

where:

- `node_id`: Specifies the target node for the command.
- `command`: Indicates the command to be executed.
- `type`: Distinguishes between GET and SET operations.
- `payload_size`: Indicates the size of the payload.
- `payload`: Contains additional data required for the command.

2.2.1 Command Set

The protocol supports a variety of commands, each with specific functionalities and payload requirements. The commands are categorized based on their operational

type: GET, SET, or both. The following table provides a comprehensive list of supported commands, their hexadecimal representation, operational type, expected payload, and description.

2.2.2 System Control Commands

- `HEARTBEAT`: This command serves as a keep-alive message, containing the node ID. It does not require a payload.
- `NODE_ID`: Allows for both setting and getting the node ID and corresponding I2C address. The payload for setting the node ID is the new ID value.
- `OPERATING_MODE and SWARM_MODE`: These commands are used to configure the system's operational behavior. They allow for setting and getting the operating mode (central/distributed) and the swarm behavior mode, respectively.
- `MASTER_ELECTION`: Initiates a master node election process and returns the ID of the newly elected master node. This is a GET-type command and does not require a payload.

2.2.3 Data Retrieval Commands

- `NEIGHBORS, ACTUATION_DATA, CURRENT_LIMIT, TEMPERATURE`: These commands are primarily GET-type commands used for data retrieval. They allow for fetching the node's list of neighbors, actuation data (current and temperature), actuation current limit, and actuation temperature, respectively.

2.2.4 Actuation Commands

- `ACTUATE`: This command is used to start or stop the actuation process with a specified intensity. The payload contains the intensity value for the actuation.

Table 9. Command Table for Communication Protocol

Command Name	Command	Type	Payload	Description
CMD_HEARTBEAT	0x41	–	–	Heartbeat message containing the node ID and its status (active/inactive)
CMD_NODE_ID	0x42	Get / Set	[NewNodeID]	Set/Get the node ID and corresponding I2C address
CMD_NEIGHBORS	0x43	Get / Set	[Neighbors...]	Update/Get the node's list of neighbors
CMD_ACTUATE	0x44	Get / Set	[Intensity]	Stop/Start the actuation with specified intensity
CMD_ACTUATION_DATA	0x45	Get	–	Get actuation data (current and temperature)
CMD_CURRENT_LIMIT	0x46	Get / Set	– / [Current]	Get/Set the actuation current
CMD_TEMPERATURE	0x47	Get	–	Get the actuation temperature
CMD_GPIO_MODE	0x48	Get / Set	[Pin] / [Pin] [Mode]	Get/Set the mode (input/output) of a GPIO pin
CMD_ANALOG	0x49	Get / Set	[Pin] / [Pin] [Value]	Get/Set an analog value for a GPIO pin
CMD_GPIO	0x4A	Get / Set	[Pin] / [Pin] [Value]	Read/Write a digital value to a GPIO pin
CMD_OPERATING_MODE	0x4B	Get / Set	– / [Mode]	Get/Set the operating mode (central/distributed)
CMD_SWARM_MODE	0x4C	Get / Set	– / [Mode]	Get/Set the swarm behavior mode
CMD_MASTER_ELECTION	0x4D	Get	–	Initiate a master node election and return the ID of the new master node
CMD_UPDATE_ROUTING_TABLE	0x4E	Get	–	Update the routing table based on the new system state
CMD_UPDATE_NEIGHBORS	0x4F	Get	–	Update the node's list of neighbors

2.2.5 GPIO Commands

- `GPIO_MODE, ANALOG, GPIO`: These commands are used for GPIO pin configuration and data manipulation. They allow for setting and getting the mode (input/output) of a GPIO pin, analog values, and digital values, respectively.

3 Operational Algorithm

3.1 Node Identification

Each node within the distributed system is uniquely identified by an ID, which ranges from 0 to 63. Correspondingly, the I2C addresses for these nodes range from `0x1E` to `0x5D` (30 to 93 in decimal notation). The node ID directly maps to its I2C address, given by `I2C Address = Node ID + 0x1E`, providing a one-to-one correspondence that simplifies the addressing scheme. This is particularly useful for targeted actuation and sensing operations, as well as for debugging purposes.

Table 10. Node IDs and Corresponding I2C Addresses

Node ID	I2C Address
0	0x1E
1	0x1F
⋮	⋮
63	0x5D

3.2 System State

The system state is represented as a 64-bit register, where each bit corresponds to a node on the I2C bus. A bit set to 1 indicates the presence of the node in the system, while a bit set to 0 indicates its absence. This register is broadcast to all nodes whenever a node joins or leaves the system, thereby keeping the network updated about its current state.

The bit position for each node is calculated using the formula:

```
bit_position = 93 - node_id;
```

The number 93 is used in the formula to align the bit position in the 64-bit system state register with the I2C address of the node. I2C addresses in this case range from 0x1E to 0x5D, which in decimal notation is 30 to 93. For instance, if node 30 joins the system, the system state would be updated as follows:

```
bit_position = 93 - 30;  // 63
system_state |= (1 << bit_position);
```

The 63rd bit in the 64-bit register will be set to 1, indicating the presence of this node with I2C address 0x1E (or 30 in decimal).

3.2.1 Heartbeat Messages

Each node periodically sends a heartbeat message to its immediate neighbors to confirm its active status. If a node fails to receive a heartbeat message from a neighbor within a predefined time window, it assumes the neighbor is disconnected, updates its neighbor list, and if it is the master, initiates a new master election.

3.3 Operational Modes

3.3.1 Central Mode

In Central Mode, the master node serves as the intermediary between the central controller and the other nodes. It receives commands via UART, relays them to the appropriate nodes using the I2C protocol, and forwards any responses back to the central controller.

3.3.2 Distributed Mode

In Distributed Mode, nodes operate autonomously based on local sensor data. They communicate with their neighbors to make collaborative decisions. Each node's mi-

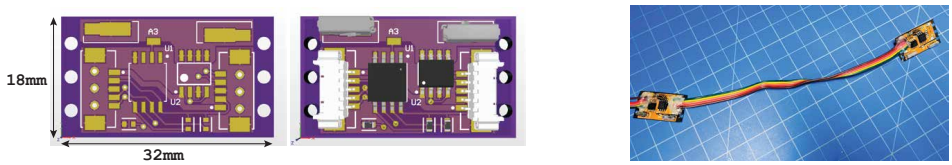


Figure 31. EneGate Rev. 1 PCB

crocontroller runs control algorithms that process sensor data to determine the appropriate actuation.

4 PCB Design

4.1 Initial Design

The main objective of the initial design was to create a compact and flexible circuit using the ATtiny85 microcontroller, which is simple yet versatile IC. The design aimed to provide the functionality of a Single Pole Double Throw (SPDT) switch using a pair of MOSFETs. Another objective was to facilitate connectivity between PCBs, achieved through two 5-pin connectors.

The main elements on the board consisted of:

1. ATtiny85 Microcontroller: An 8-pin IC powered by a 5V supply. Despite limited memory and peripheral options, it was considered adequate for simple switching tasks.

2. MOSFET Pair: This included one N-channel and one P-channel MOSFET and was used to emulate an SPDT switch.

3. Wago Surface Mount Terminal Blocks: These served as the external interface for making connections to SMA wire or other thermal actuators. Outputs could be switched either high or low.

The schematic was designed with a flex PCB, specifically to enable the board to flex along its central axis without compromising the integrity of its electrical traces.

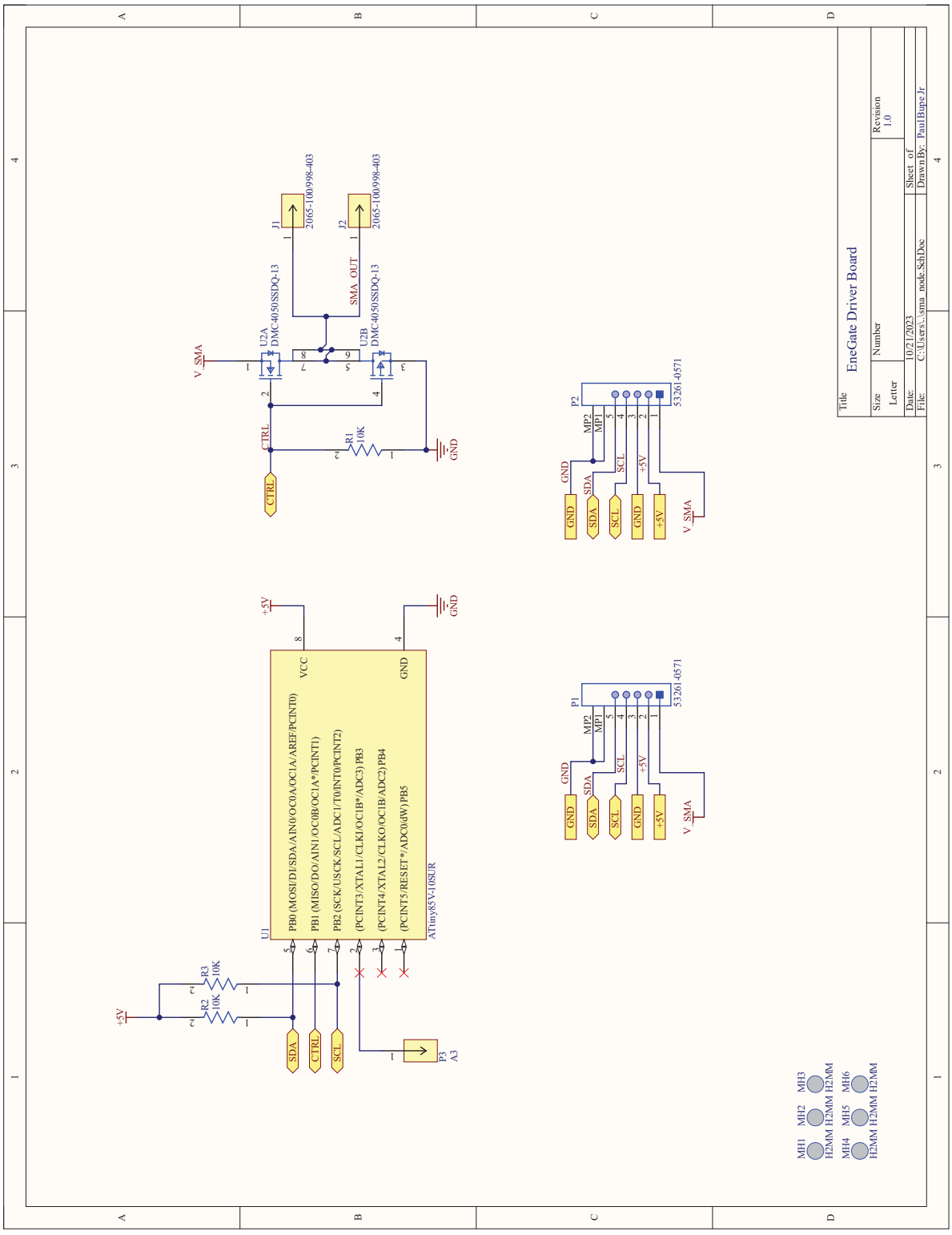


Figure 32. EneGate Rev. 1 Schematic

Components were placed strategically to prevent any overlap over the bending axis, thus ensuring that the board could maintain its flexible characteristics. The design also included the integration of pull-up resistors on each board; however, only one board in a connected chain needed the actual installation of these resistors. Additionally, it included a designated pin for MOSFET control and another pin set aside for general-purpose analog input. This minimalist approach was employed to focus solely on essential functions, reducing complexity.

Initial testing of the design affirmed the preliminary hypothesis, but also brought two major issues to light. First, a layout error resulted in the MOSFETs not behaving as an SPDT switch. Because they were contained in a single package, their symbol on the schematic had the N-channel MOSFET above the P-channel MOSFET. Typically N-channel MOSFETs are used as low-side drivers while P-channel MOSFETs are used as high-side drivers, which was not the case in this error. Second, the use of the ATtiny85 microcontroller posed its own set of limitations. While the microcontroller was selected for its compact size and cost-efficiency, it was not sufficiently powerful or had enough space for the tasks. Its restricted input/output (I/O) functionalities further limited the design's adaptability and scope for broader applications. Finally, the MOSFET package's inadequate thermal dissipation capabilities, particularly when handling high current levels — a critical requirement for thermal actuators, made the PCB not suitable for the intended work.

4.2 Second Revision

The primary intent behind the second revision was to make significant improvements over the initial design. Notably, the revision aimed at enhancing functionality by incorporating a microcontroller with a higher pin and peripheral count, thereby offering more configuration options. The most notable component switch was the upgrade from the ATtiny85 microcontroller to the ATtiny841. The latter was chosen for its

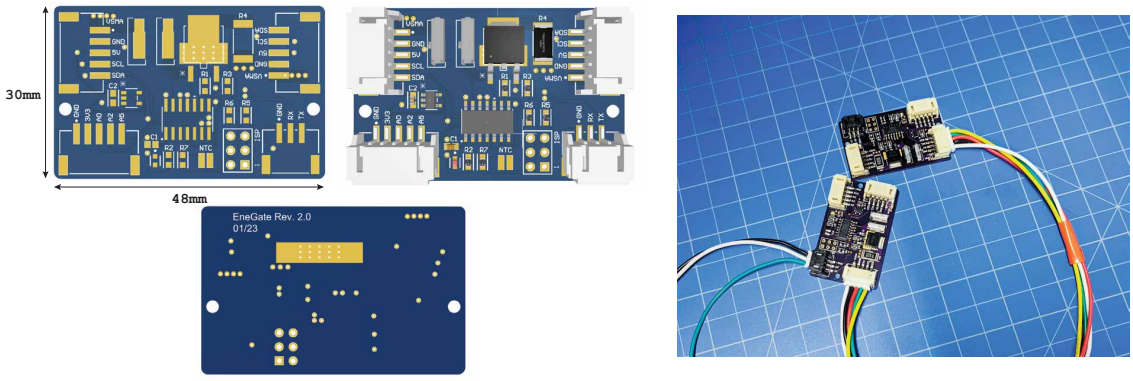


Figure 33. EneGate Rev. 2 PCB

higher pin and peripheral count, granting more flexibility in design options. Another significant addition was the inclusion of a sense resistor in the MOSFET current path to allow for accurate current measurement. The design also featured a dedicated 10 kΩ thermistor input and two LEDs: one controlled by the microcontroller and another connected to the output of the voltage regulator. This design made use of a 3.3 V voltage regulator, and due to the additional pins on the ATtiny841, a dedicated header with three general-purpose I/Os was added.

The schematic of this revision was segmented into three primary sections: the microcontroller, the voltage regulator, and the MOSFET driver section. With the addition of the sense resistor in the MOSFET's current path, the board could now make accurate current measurements. The daisy chain headers were also modified to carry VSMA, Ground,5 V, and the I2C lines. The MOSFET circuit underwent transformation from a SPDT configuration to a single-channel MOSFET. The board's layout presented a more streamlined design, primarily composed of four surface mount headers and a programming header. It also included two holes for a 10 kΩ thermistor and a 100 mΩ current sense resistor. This current sense resistor played a key role in closed-loop control when maintaining specific current levels, with similar control capabilities to the thermistor. The board design used a SOIC package for the microcontroller, which was not the most efficient use of the available board space.

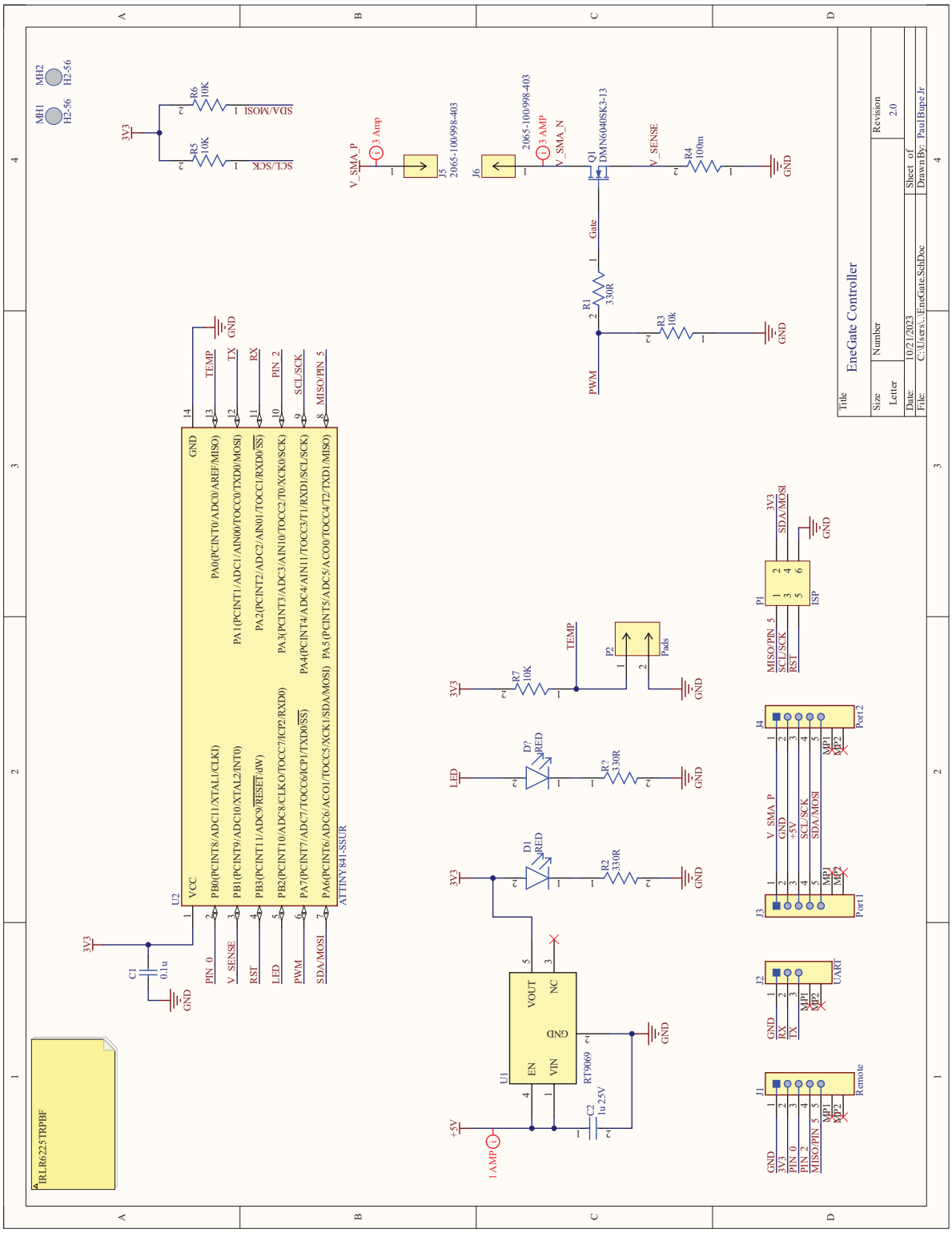


Figure 34. EneGate Rev. 2 Schematic

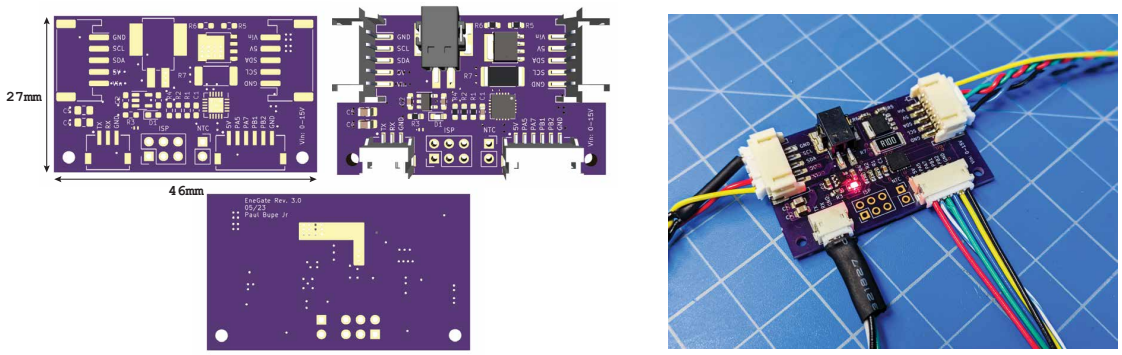


Figure 35. EneGate Rev. 3 PCB

While this revision saw various enhancements, it also had its challenges. The MOSFET's ability to reverse was removed, a capability present in the initial revision. However, this MOSFET was better equipped to handle higher thermal capacities and could manage significantly more current. Another potential limitation was the I2C pull-up resistor, which wasn't perpetually connected. The revision also introduced a dedicated ISP programming port and a serial UART for real-time data extraction from the board, an improvement over the previous board that required a programming clip.

4.3 Third Revision

In the third revision of the design, several key refinements were made to optimize the board's performance and functionality. One of the most significant changes was the transition from the SOIC package to the VQFN for the microcontroller, with a footprint of about 5 mm x 5 mm. This change not only reduced the footprint of the board but also allowed for a more compact design.

The actuator connector underwent a modification as well. Initially, an all-metal Wago connector was used, but this was replaced with a more traditional 2-pole insertion terminal, which is still surface-mounted. The change was necessitated by issues encountered with solder wicking into the insertion hole during the soldering process. Additionally, the metal WAGO connector proved to be tricky to desolder without

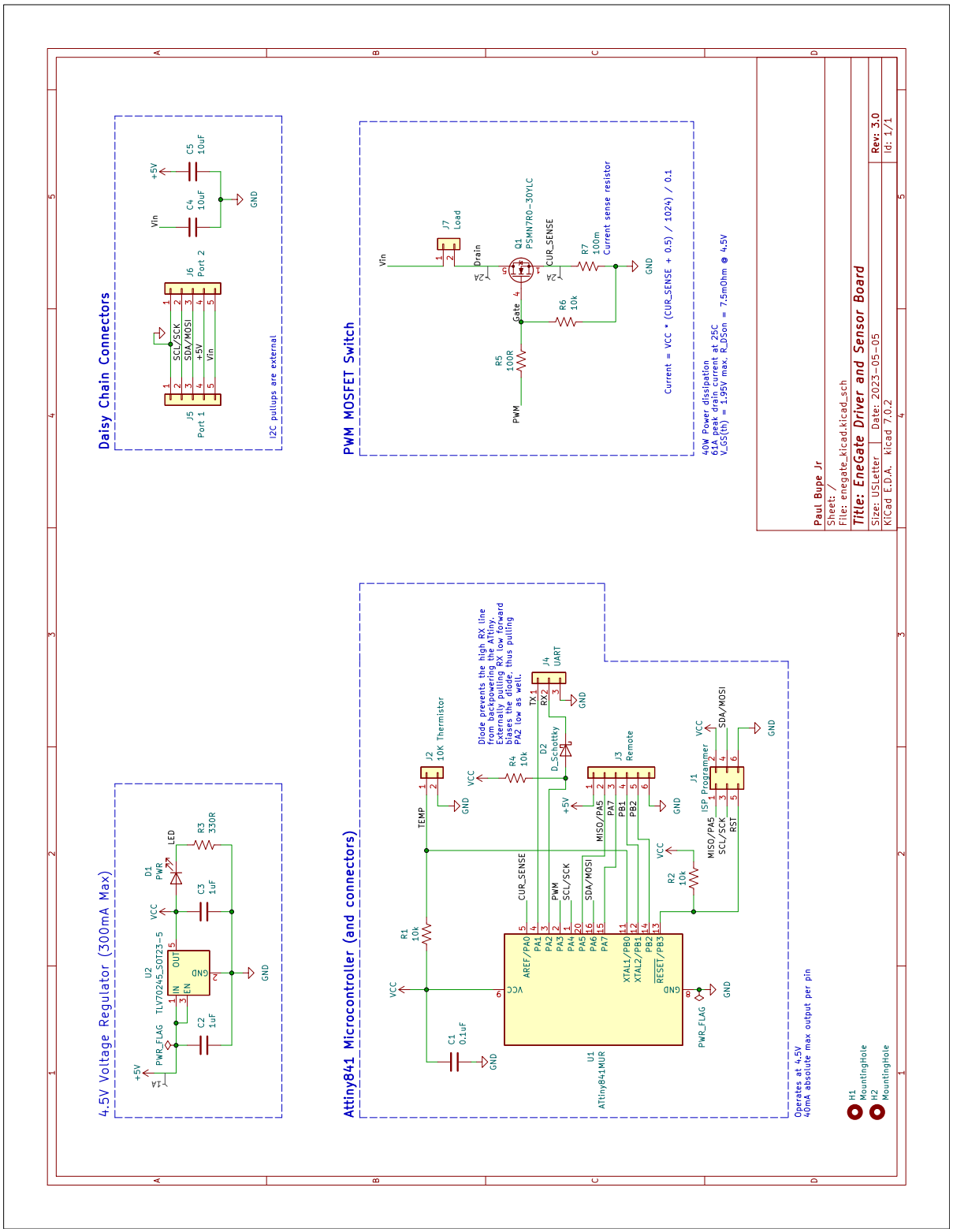


Figure 36. EneGate Rev. 3 schematic

damage to nearby components, making the 2-pole terminal a more practical choice.

Another noteworthy change was the adjustment of the system voltage from 3.3 V to 4.5 V. The higher voltage was chosen to enable the system to run on a standard 5V input, with a regulator stepping it down to 4.5 V. This higher voltage also allowed the microcontroller to fully turn on the MOSFET without the need for external circuits, such as a charge pump. The MOSFET used in this iteration was updated to one with 40 W of power dissipation, capable of handling 61 A peak drain current at 25 C. This MOSFET had a $V_{GS}(th)$ of 1.95 V max with an R_{DSon} of only 7.5 mΩ at 4.5 V, meaning the microcontroller is capable of fully turning it on directly. The same technique was employed where part of the bottom layer of the PCB serves as a heatsink for the MOSFET. This allows for high current throughput without the need for an external heatsink.

The connector for the daughter board was also revised. A smaller pitch connector was used, which allowed for the inclusion of an additional pin. This resulted in a total of four I/O pins along with power and ground, enhancing the board's expandability. To accommodate the additional pin for the daughter board, the pin LED was removed, although the ISP header was retained. Lastly, a simple voltage shifter circuit was added to the UART RX line to prevent back-powering the ATtiny microcontroller. These changes collectively contribute to a more efficient and functional board, addressing previous limitations while introducing new capabilities.

5 Firmware

The EneGate library is written in C++ and targets currently Arduino-based microcontrollers. It aims to provide a structured way to manage individual nodes in a network, each with its own set of commands and parameters. The library is object-oriented, encapsulating functionalities into a class named EneGate.

5.1 The EneGate Class

The EneGate class serves as the core of the firmware. Each object of this class represents a unique node in the network with its own ID, message buffer, and PID control parameters. It provides a well-structured and extensible framework for managing nodes in a network, complete with message processing. Its design allows for easy addition of new features and commands, making it a versatile choice for various applications.

```
class EneGate {
    public:
        EneGate(uint8_t node_id);
        ~EneGate();
        // ... (other public methods)
    private:
        uint8_t node_id;
        pid_data pid;
        // ... (other private members)
};
```

Here, `node_id` uniquely identifies each node, and `pid` is a struct that holds the PID controller's parameters and state variables. Upon instantiation, the constructor `EneGate::EneGate(uint8_t node_id)` initializes several key parameters. Notably, it sets the node ID and initializes the PID control variables, such as the proportional, integral, and derivative gains.

5.2 Data Structures

1. Message Structure

   ```
   typedef struct {
       uint8_t node_id;
       uint8_t command;
       uint8_t type;
       uint8_t payload_size;
       uint8_t payload[32];
   } Message;
   ```

   This struct encapsulates a single message, containing the message size, command, command type, node ID, and a payload.

2. Command Enumeration

```
enum class Command : uint8_t {
    CMD_NODE_ID = 0x42,
    // ... (other commands)
    Invalid = 0x99
};
```

This enum class defines various command types that a node can accept.

3. PID Data Structure

```
typedef struct pid_data {
    double kp;
    double ki;
    double kd;
    // ... (other PID variables)
} pid_data;
```

This struct holds the parameters and variables for the PID controller.

5.3 Communication Protocols

The message processing logic is encapsulated in the `process_byte` method. This method takes an incoming byte, appends it to a message buffer, and then decides which protocol the message adheres to—either binary or ASCII. If the message buffer reaches its maximum size, it gets reset to avoid overflow, ensuring robustness.

The binary protocol is designed for efficient communication. Messages in this format start with a frame of `"0x01 0x01"`.

```
bool EneGate::is_binary_protocol() {
    return (msg_buffer_index >= 2 &&
            msg_buffer[msg_buffer_index - 2] == 0x01 &&
            msg_buffer[msg_buffer_index - 1] == 0x01);
}
```

The `is_binary_protocol()` method checks if the incoming message conforms to the binary protocol.

The ASCII protocol is human-readable and starts with a "{" and ends with a "}".

```
bool EneGate::is_ascii_protocol(uint8_t byte) {
    return (byte == '}' || byte == '\n');
}
```

The `is_ascii_protocol()` method checks if the incoming message conforms to the ASCII protocol.

The `process_message` method serves as a dispatcher that routes incoming messages based on their command type. It employs a switch-case construct to handle various commands. For instance, in the case of the `CMD_CURRENT_LIMIT` command, it returns the current limit.

```
switch (command) {
    case Command::CMD_CURRENT_LIMIT:
        // Return the current limit
        break;
    // ... other cases
}
```

5.4 Real-Time Control

```
void EneGate::update_pid(double setpoint, double measured_value) {
    // ... (PID calculations)
    pid.output = pid.kp * pid.error + pid.ki * pid.integral + ...
}
```

The firmware incorporates a PID controller for real-time control tasks. The PID controller uses a separate data structure (`pid_data`) to hold its parameters and state variables and which updates the PID variables based on the current setpoint and the measured value. The method calculates the error, integral, and derivative terms and then computes the PID output.

6 Performance and Limitations

Testing the integrated system presented several challenges, particularly when implementing the protocol in the firmware. The microcontroller quickly reached its flash memory limit, a constraint exacerbated by the use of an open-source core. This

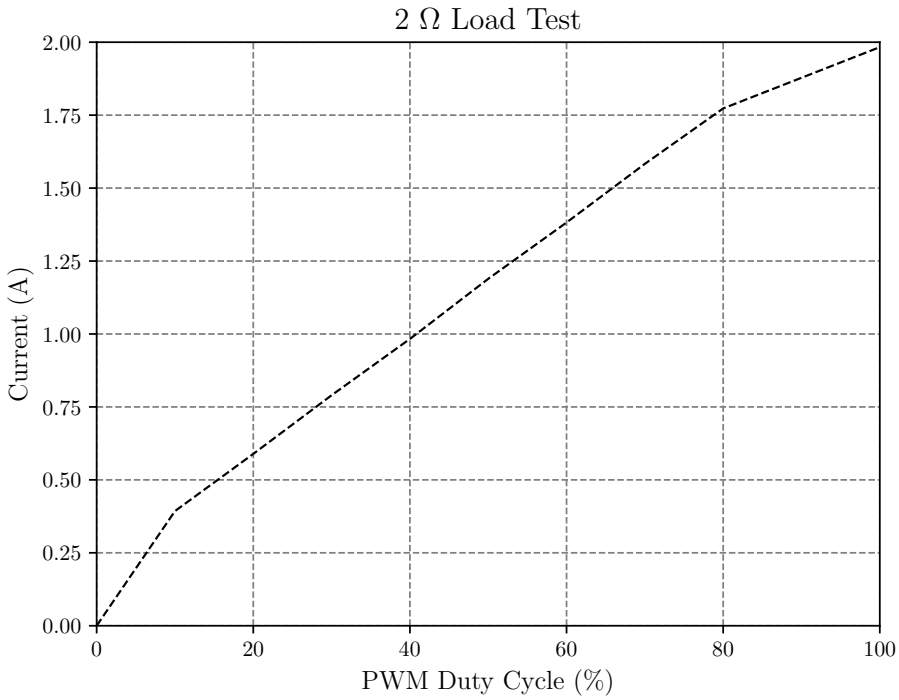


Figure 37. Graph showing the corresponding current to PWM with a 2 Ω load.

brought into question the feasibility of implementing the full protocol as initially planned. The original design intended to utilize an STM32 microcontroller, given that the first OptiGap prototype was based on this more robust platform. However, due to the chip shortage caused by the pandemic, the required parts were unavailable. The ATtiny microcontrollers were the available alternative, which made them perfect candidates at the time.

Implementing just the basic code for serial and I2C communications, along with the shell code for parsing the Enegate protocol, consumed close to 80% of the available flash memory. To mitigate this, a low-level rewrite was initiated, employing low-level C, register manipulation, and assembly language. This approach allowed for the implementation of serial, I2C, PWM, and ADC functionalities with significantly reduced flash usage. However, this low-level approach would likely deter user adoption due to its complexity, conflicting with the project's goal of creating a user-friendly, widely adopted system.

Given these constraints, the system components were tested individually. The first set of tests aimed to confirm the serial communication and making sure the serial connection was not back-powering the ATtiny841 microcontroller. The nodes were able to print to the serial port as well as receive commands from the host. Most importantly, there were no issues of back-powering the microcontroller through the serial port. The next set of tests focused on validating the PWM and actuator

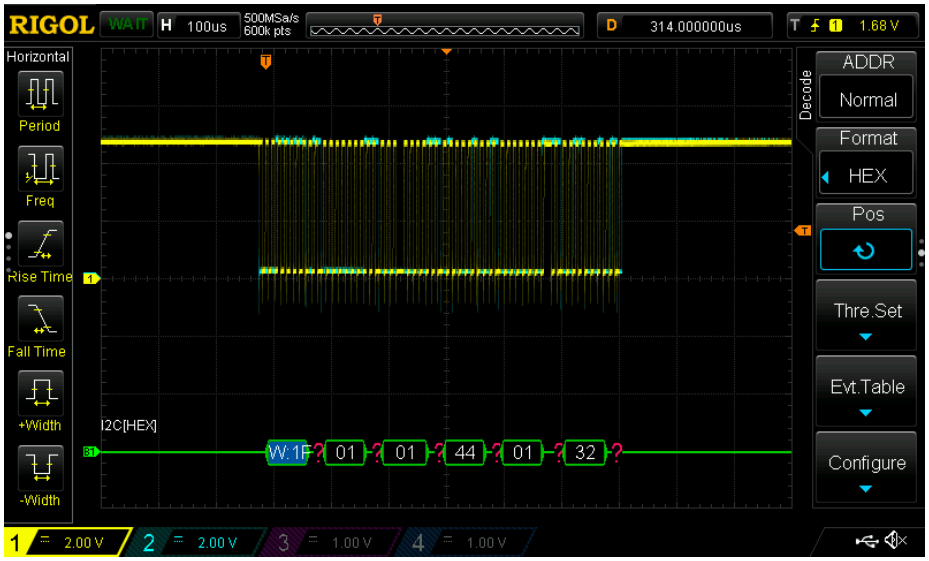


Figure 38. Oscilloscope capture of the EneGate protocol command `CMD_ACTUATE` with a value of 50 %.

functionalities. These tests were crucial for ensuring that the board could effectively control thermal actuators or any load. The results confirmed that both PWM and actuator functionalities were implemented successfully, showing an increase in current as the PWM duty cycle was increased.

Another critical aspect was the node-to-node I2C communications. Shown in Figure 38, a master node was configured to send a message conforming to the EneGate protocol to another node. Specifically, the `CMD_ACTUATE` command with a value of 50 % was sent. An oscilloscope with logic analyzer capabilities was used to verify the proper implementation of the I2C protocol, with results confirming that the I2C communication was robust and reliable. A connector was created to plug into the last

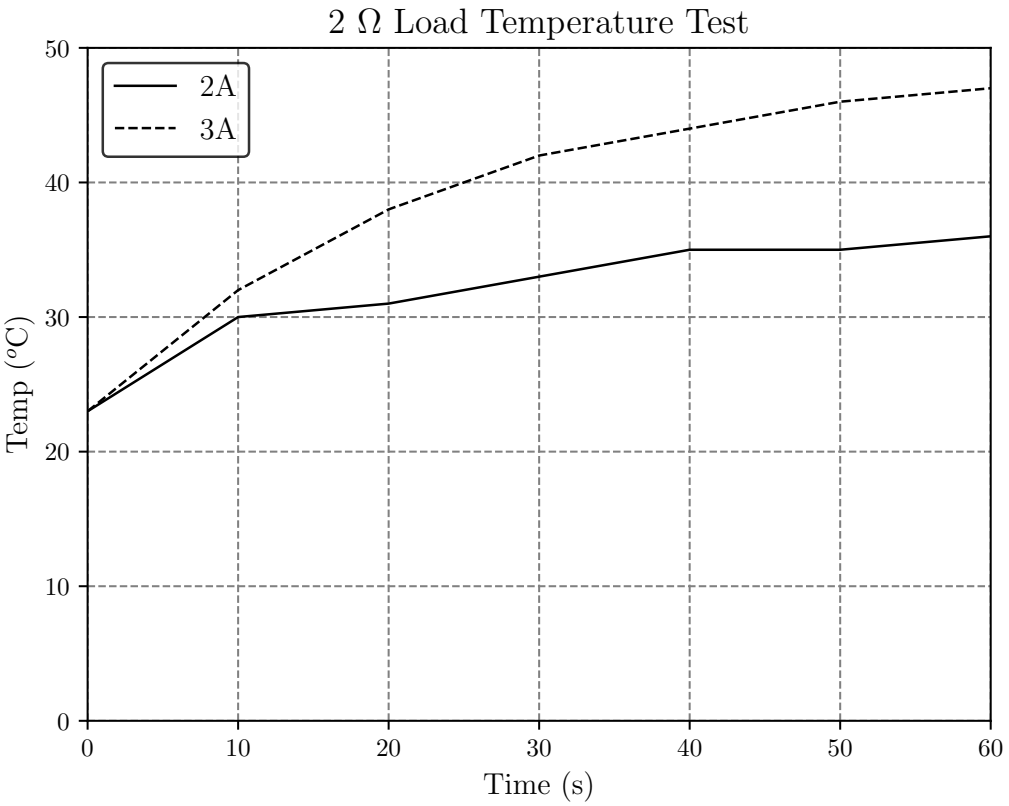


Figure 39. EneGate MOSFET package temperature readings

header, equipped with I2C terminating pull-up resistors. This modification aimed to stabilize the I2C communication lines and improve the reliability of the system.

During the testing phase, it was observed that at low voltages the ADC was not stable on some pins, including the one used for the current shunt. This instability resulted in inaccurate current measurements, particularly at low values. Subsequent tests were conducted to verify that the MOSFET and PCB could sustain high currents without an external heatsink. These tests were essential for confirming that the system could operate as a controller for thermal actuators, which can use large amounts of current in short bursts. Results shown in Figure 39 showed very stable thermal performance across the 3 A current range.

Next, the distributed actuation capabilities of the EneGate nodes were tested. In this test, two SMA SCRAM devices were connected at the extreme ends of a chain of

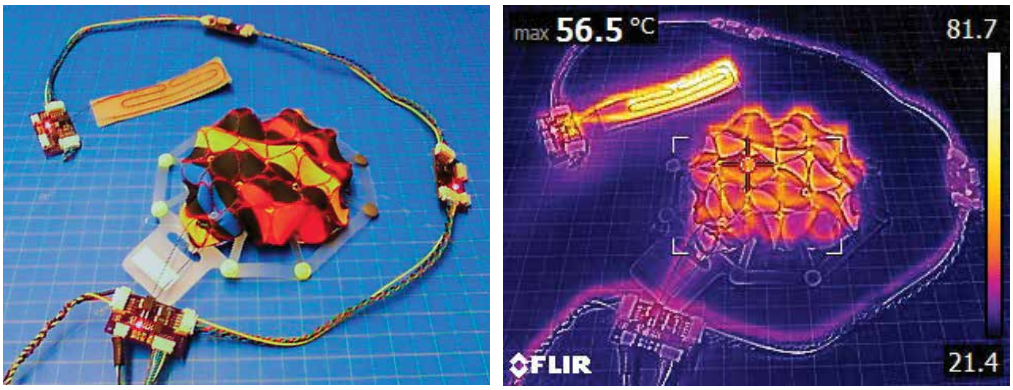


Figure 40. EneGate distributed actuation test.

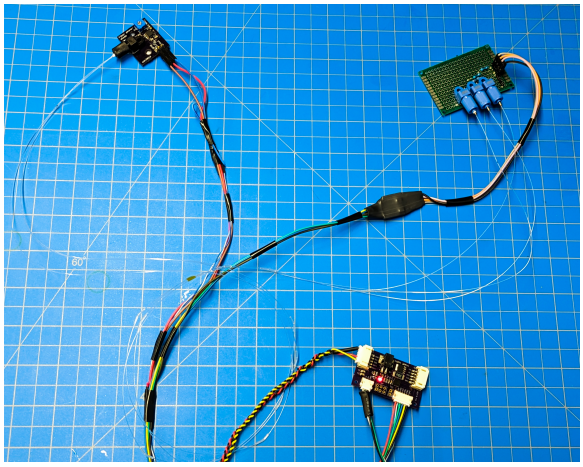


Figure 41. EneGate OptiGap test.

EneGates. The devices were then actuated to different intensities. Seen in Figure 40, a thermal camera was used to capture the results of the thermal actuation, showing that the successful use of EneGate for distributed actuation.

Finally, a wire harness was designed that connected to the GPIO header of an EneGate board to power three optical emitters and a receiver in order to test an OptiGap sensor with EneGate. Shown in Figure 41, this setup was used to successfully verify that the optical intensity signals from an OptiGap sensor could be captured and then relayed over the serial port.

6.1 Adherence to Design Requirements

The following table (11) outlines how the current revision of the EneGate board adheres to the initial design requirements set in Table 7. Overall it meets the stated requirements, with the exception of the inadequate microcontroller being used in the current version.

Table 11. Design Requirements

Category	Requirement	Implementation
Power Management	Voltage Stability: Maintain a stable power source for all components.	LTV70245 Voltage Regulator (4.5V, 300mA max)
	MOSFET Switch: Control a high current load.	PSMN7R0-30YLC MOSFET (61A peak, R_DS(on) = 7.5mOhm at 4.5V)
Control and Monitoring	Microcontroller: Serve as the central control unit.	ATtiny841 (Not fully met)
	Current Monitoring: Accurate monitoring of current through the MOSFET.	100m Ohm Sense Resistor
	Temperature Sensing: Monitor temperature levels of components.	10k Thermistor Interface
Connectivity	Daisy-Chaining: Ability to connect multiple boards in series.	Daisy Chain Connectors (I2C, VSMA, 5V, GND)
	Communication Interface: Data extraction and real-time communication.	I/O header, daughter board and UART Interfaces
	Microcontroller Programming: Update microcontroller's firmware.	ISP Programmer Port
Protection and Feedback	Operational Feedback: Visual indicator of board's status.	LED (connected to voltage regulator output)
	Back-powering Protection: Prevent inadvertent power flow between boards.	Schottky diode protection
	Over-current Protection: Limit current going through MOSFET.	100m Ohm Sense Resistor

CHAPTER V

DISCUSSION AND CONCLUSION

1 Research Objectives and Contributions

The primary objectives of this research centered around the development and validation of modular sensing and actuation systems for soft robotics, with a particular emphasis on OptiGap. These goals have been largely achieved. OptiGap has been subjected to real-world testing and validation through a robotics case study, affirming its role as a flexible and cost-effective solution for sensing. Concurrently, EneGate has been engineered with hardware and software frameworks that demonstrate significant potential for modular actuation and sensing in both lab-based and real-world robotic scenarios. A key strength of this research is its contribution to the field of modular technologies in robotics. OptiGap offers a unique and practical approach to sensing, adaptable to various requirements. EneGate augments this by introducing actuation functionalities that have the potential to enrich the modular landscape in robotics.

This research has also generated a substantial amount of code and software to support the modular technologies developed. Recognizing the value of community collaboration and the acceleration of technological advancement, most code and software are being made publicly available as open-source resources on GitHub. This initiative aims to foster further development and application of the technologies, allowing other researchers and practitioners to build upon, modify, or integrate the work into their own projects. By contributing these resources to the open-source community, the research not only advances the field of robotics but also promotes a collaborative ecosystem for ongoing innovation.

2 Limitations

However, the research is not without its limitations. One such limitation is OptiGap's sensitivity and resolution, which are contingent on the quality of the light pipe used and the algorithms deployed for data processing. While this presents a challenge, it is also a strength in that it also offers an opportunity for customization. OptiGap's compatibility with a wide range of light pipes and data processing algorithms means that it can be tailored to meet specific requirements. For example, OptiGap has been successfully tested with clear 3D printer filament, which is highly flexible and heat-resistant up to 190°C. The system's robustness is further enhanced by a design manual, which provides guidelines for effective utilization.

Another notable limitation is the lack of empirical testing to integrate OptiGap and TASL in different applications. The aforementioned work being done to put code into public repositories with documentation is a long term way of ensuring that these systems can be integrated into new testbeds. Although TASL initially inspired OptiGap and was intended to be integrated, OptiGap has evolved into a more versatile technology with broader applications, which is why this research has been conducted with an eye toward future adaptability and integration, ensuring that subsequent work can incorporate OptiGap sensors into the TASL. EneGate's PCB issues also posed a challenge, particularly the need to transition to an STM32 microcontroller. While this necessitates another significant board revision, it's worth noting that OptiGap was initially developed on an STM32 platform, ensuring compatibility. The focus on user-friendly code for both EneGate and OptiGap aims to facilitate broader adoption, justifying the decision to opt for a new board revision with increased flash memory.

3 Future Work

The ultimate goal for OptiGap and EneGate extends beyond academic research; the aim is to transition these technologies into real-world applications. The focus will be on integrating OptiGap and EneGate within the larger framework of addressable actuation and sensing in soft robotics. Addressable actuation, the ability to locally control and isolate actuation to specific regions of the actuator [84], will be key in achieving more precise and efficient control and can be significantly advanced by incorporating OptiGap's sensing technology and EneGate's actuation capabilities.

One key area of exploration will be the customization and scalability of these technologies for different soft robotic applications. This approach aligns with the research's focus on adaptability and integration, ensuring that subsequent work incorporates OptiGap sensors and EneGate actuators into various soft robotic platforms.

Addressing the limitations noted in the research, such as OptiGap's sensitivity and resolution, and EneGate's transition to a more powerful microcontroller, will also be an area of focus. Empirical testing in different scenarios will also help refine these technologies, validating their effectiveness in real-world applications. The open-source nature of the software developed in this research further supports this goal, as it invites collaboration and innovation from the broader robotics community.

The commercialization process, already initiated for OptiGap, will also be a significant focus of future work. This process will not only involve securing a patent and refining designs for mass production but also exploring partnerships for broader market application. The aim is to transition these technologies from academic research into practical, real-world tools.

REFERENCES

[1] Y. Jiang, F. Chen, and D. M. Aukes, "Tunable dynamic walking via soft twisted beam vibration," IEEE robotics and automation letters **8**, 1967–1974 (2023).

[2] J. Paik, "Soft components for soft robots," in "Soft Robotics," (Springer Berlin Heidelberg, Berlin, Heidelberg, 2015), pp. 272–281.

[3] J.-Y. Lee, J. Eom, W.-Y. Choi, and K.-J. Cho, "Soft LEGO: Bottom-Up design platform for soft robotics," in "2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)," (IEEE, 2018), pp. 7513–7520.

[4] F. Pigozzi and E. Medvet, "Evolving modularity in soft robots through an embodied and Self-Organizing neural controller," Artificial life **28**, 322–347 (2022).

[5] Y. Zhang, M. Su, Y. Guan, H. Zhu, and S. Mao, "An integrated framework for modular reconfigurable soft robots," in "2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)," (IEEE, 2018), pp. 606–611.

[6] C. Zhang, P. Zhu, Y. Lin, Z. Jiao, and J. Zou, "Modular soft robotics: Modular units, connection mechanisms, and applications," Advanced intelligent systems (Weinheim an der Bergstrasse, Germany) **2**, 1900166 (2020).

[7] T. Jin, T. Wang, Q. Xiong, Y. Tian, L. Li, Q. Zhang, and C.-H. Yeow, "Modular soft robot with origami skin for versatile applications," Soft robotics **10**, 785–796 (2023).

[8] B. Zhang, C. Hu, P. Yang, Z. Liao, and H. Liao, "Design and modularization of Multi-DoF soft robotic actuators," IEEE Robotics and Automation Letters **4**, 2645–2652 (2019).

[9] P. Bupe, D. J. Jackson, and C. K. Harnett, "Electronically reconfigurable virtual joints by shape memory Alloy-Induced buckling of curved sheets," in "SoutheastCon 2022," (2022), pp. 598–604.

[10] S. Timoshenko and J. Gere, *Theory of Elastic Stability* (Dover Publications, Inc, 2009).

[11] Y. Jiang, M. Sharifzadeh, and D. M. Aukes, "Reconfigurable soft flexure hinges via pinched tubes," in "2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)," (ieeexplore.ieee.org, 2020), pp. 8843–8850.

[12] M. Sharifzadeh and D. M. Aukes, "Curvature-Induced buckling for Flapping-Wing vehicles," IEEE/ASME Transactions on Mechatronics **26**, 503–514 (2021).

[13] T. L. Buckner, R. A. Bilodeau, S. Y. Kim, and R. Kramer-Bottiglio, "Roboticizing fabric by integrating functional fibers," Proceedings of the National Academy of Sciences **117**, 25360–25369 (2020).

[14] K. Nguyen, N. Yu, M. M. Bandi, M. Venkadesan, and S. Mandre, "Curvature-induced stiffening of a fish fin," Journal of the Royal Society, Interface / the Royal Society **14** (2017).

[15] V. Pini, J. J. Ruz, P. M. Kosaka, O. Malvar, M. Calleja, and J. Tamayo, "How two-dimensional bending can extraordinarily stiffen thin sheets," Scientific reports **6**, 29627 (2016).

[16] N. P. Bende, A. A. Evans, S. Innes-Gold, L. A. Marin, I. Cohen, R. C. Hayward, and C. D. Santangelo, "Geometrically controlled snapping transitions in shells with curved creases," Proceedings of the National Academy of Sciences of the United States of America **112**, 11175–11180 (2015).

[17] M. Jiang, Q. Yu, and N. Gravish, "Vacuum induced tube pinching enables reconfigurable flexure joints with controllable bend axis and stiffness," in "2021 IEEE 4th International Conference on Soft Robotics (RoboSoft)," (2021), pp. 315–320.

[18] T. L. Buckner and R. Kramer-Bottiglio, "Functional fibers for robotic fabrics," Multifunctional Materials **1**, 012001 (2018).

[19] RodrigueHugo, WangWei, HanMin-Woo, K. J. Y, and AhnSung-Hoon, "An overview of shape memory Alloy-Coupled actuators and robots," Soft Robotics (2017).

[20] J. K. Paik, E. Hawkes, and R. J. Wood, "A novel low-profile shape memory alloy torsional actuator," Smart Materials and Structures **19**, 125014 (2010).

[21] J. K. Paik and R. J. Wood, "A bidirectional shape memory alloy folding actuator," Smart materials & structures **21**, 065013 (2012).

[22] Y. Sugiyama and S. Hirai, "Crawling and jumping by a deformable robot," The International journal of robotics research **25**, 603–620 (2006).

[23] H.-B. Park, D.-R. Kim, H.-J. Kim, W. Wang, M.-W. Han, and S.-H. Ahn, "Design and analysis of artificial muscle robotic elbow joint using shape memory alloy actuator," International journal of precision engineering and manufacturing **21**, 249–256 (2020).

[24] H.-I. Kim, M.-W. Han, S.-H. Song, and S.-H. Ahn, "Soft morphing hand driven by SMA tendon wire," Composites Part B Engineering **105**, 138–148 (2016).

[25] J. K. Strelec, D. C. Lagoudas, M. A. Khan, and J. Yen, "Design and implementation of a shape memory alloy actuated reconfigurable airfoil," Journal of intelligent material systems and structures **14**, 257–273 (2003).

[26] E. A. Peraza-Hernandez, D. J. Hartl, and R. J. M. Jr, "Design and numerical analysis of an SMA mesh-based self-folding sheet," Smart Materials and Structures **22**, 094008 (2013).

[27] E. Peraza-Hernandez, D. Hartl, and D. Lagoudas, "SHAPE MEMORY ALLOY LAMINATE FOR DESIGN OF SELF-FOLDING RECONFIGURABLE STRUCTURES," (2013).

[28] *Computational Design of a Reconfigurable Origami Space Structure Incorporating Shape Memory Alloy Thin Films* (American Society of Mechanical Engineers Digital Collection, 2013).

[29] E. Yoshida, S. Murata, S. Kokaji, K. Tomita, and H. Kurokawa, "Micro self-reconfigurable robotic system using shape memory alloy," in "Distributed Autonomous Robotic Systems 4," , L. E. Parker, G. Bekey, and J. Barhen, eds. (Springer Japan, Tokyo, 2000), pp. 145–154.

[30] *Animating paper using shape memory alloys*, CHI '12 (Association for Computing Machinery, 2012).

[31] M. Sreekumar, T. Nagarajan, M. Singaperumal, M. Zoppi, and R. Molfino, "Critical review of current trends in shape memory alloy actuators for intelligent robots," Industrial Robot: An International Journal **34**, 285–294 (2007).

[32] G. Sun and C. T. Sun, "One-dimensional constitutive relation for shape-memory alloy-reinforced composite lamina," Journal of Materials Science **28**, 6323–6328 (1993).

[33] G. Sun and C. T. Sun, "Bending of shape-memory alloy-reinforced composite beam," Journal of Materials Science **30**, 5750–5754 (1995).

[34] C. Kim, G. Kim, Y. Lee, G. Lee, S. Han, D. Kang, S. H. Koo, and J.-S. Koh, "Shape memory alloy actuator-embedded smart clothes for ankle assistance," Smart Materials and Structures **29**, 055003 (2020).

[35] S. Seok, C. D. Onal, K.-J. Cho, R. Wood, D. Rus, and S. Kim, "Meshworm: A peristaltic soft robot with antagonistic nickel titanium coil actuators," Mechatronics, IEEE/ASME Transactions on **18**, 1485–1497 (2013).

[36] Z. J. Patterson, A. P. Sabelhaus, K. Chin, T. Hellebrekers, and C. Majidi, "An untethered brittle Star-Inspired soft robot for Closed-Loop underwater locomotion," IROS (2020).

[37] T. L. Buckner and R. Kramer-Bottiglio, "Functional fibers for robotic fabrics," Multifunctional Materials **1**, 012001 (2018).

[38] V. Sanchez, C. J. Walsh, and R. J. Wood, "Soft robotics: Textile technology for soft robotic and autonomous garments (adv. funct. mater. 6/2021)," Advanced functional materials **31**, 2170041 (2021).

[39] H. Jin, E. Dong, M. Xu, C. Liu, G. Alici, and Y. Jie, "Soft and smart modular structures actuated by shape memory alloy (SMA) wires as tentacles of soft robots," Smart Materials and Structures **25**, 085026 (2016).

[40] J. Mohd Jani, M. Leary, A. Subic, and M. A. Gibson, "A review of shape memory alloy research, applications and opportunities," Materials & design **56**, 1078–1113 (2014).

[41] S. Seok, C. D. Onal, K. Cho, R. J. Wood, D. Rus, and S. Kim, "Meshworm: A peristaltic soft robot with antagonistic nickel titanium coil actuators," IEEE/ASME Transactions on Mechatronics **18**, 1485–1497 (2013).

[42] J.-S. Koh and K.-J. Cho, "Omega-Shaped Inchworm-Inspired crawling robot with Large-Index-and-Pitch (LIP) SMA spring actuators," IEEE/ASME Transactions on Mechatronics **18**, 419–429 (2013).

[43] F. Schmitt, O. Piccin, L. Barbé, and B. Bayle, "Soft robots manufacturing: A review," Frontiers in robotics and AI **5**, 84 (2018).

[44] A. Firouzeh, Y. Sun, H. Lee, and J. Paik, "Sensor and actuator integrated low-profile robotic origami," in "2013 IEEE/RSJ International Conference on Intelligent Robots and Systems," (2013), pp. 4937–4944.

[45] V. Sanchez, C. J. Walsh, and R. J. Wood, "Textile technology for soft robotic and autonomous garments," Advanced functional materials **31**, 2008278 (2021).

[46] Y. Jiang, M. Sharifzadeh, and D. M. Aukes, "Shape change propagation through soft curved materials for Dynamically-Tuned paddling robots," in "2021 IEEE 4th International Conference on Soft Robotics (RoboSoft)," (2021), pp. 230–237.

[47] Eid, "Optical fiber sensors: review of technology and applications," Indonesian journal of electrical engineering and computer science (2022).

[48] B. P. Pal, "Optical fiber sensors: A versatile technology platform for sensing," Journal of the Indian Institute of Science **94**, 283–310 (2014).

[49] H. Zhao, K. O'Brien, S. Li, and R. F. Shepherd, "Optoelectronically innervated soft prosthetic hand via stretchable optical waveguides," Science Robotics **1** (2016).

[50] S. Sareh, Y. Noh, M. Li, T. Ranzani, H. Liu, and K. Althoefer, "Macrobend optical sensing for pose measurement in soft robot arms," Smart Materials and Structures **24**, 125024 (2015).

[51] M. McCandless, F. J. Wise, and S. Russo, "A Soft Robot with Three Dimensional Shape Sensing and Contact Recognition Multi-Modal Sensing via Tunable Soft Optical Sensors," in "2023 IEEE International Conference on Robotics and Automation (ICRA)," (2023).

[52] H. A. Wurdemann, S. Sareh, A. Shafti, Y. Noh, A. Faragasso, D. S. Chathuranga, H. Liu, S. Hirai, and K. Althoefer, "Embedded electro-conductive yarn for shape sensing of soft robotic manipulators," in "2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)," (2015).

[53] J. Avery, M. Runciman, A. Darzi, and G. P. Mylonas, "Shape Sensing of Variable Stiffness Soft Robots using Electrical Impedance Tomography," in "2019 International Conference on Robotics and Automation (ICRA)," (2019).

[54] F. Al Jaber and K. Althoefer, "Towards creating a flexible shape senor for soft robots," in "2018 IEEE International Conference on Soft Robotics (RoboSoft)," (2018).

[55] G. Rizzello, P. Serafino, D. Naso, and S. Seelecke, "Towards Sensorless Soft Robotics: Self-Sensing Stiffness Control of Dielectric Elastomer Actuators," IEEE Transactions on Robotics **36**, 174–188 (2020).

[56] K. C. Galloway, Y. Chen, E. Templeton, B. Rife, I. S. Godage, and E. J. Barth, "Fiber Optic Shape Sensing for Soft Robotics," Soft Robotics **6**, 671–684 (2019).

[57] M. A. Zawawi, S. O'Keeffe, and E. Lewis, "Compensated intensity-modulated optical fibre bending sensor based on tilt angle loss measurement," in "SENSORS, 2014 IEEE," (2014), pp. 370–373.

[58] M. A. Zawawi, S. O'Keeffe, and E. Lewis, "An extrinsic optical fiber bending sensor: A theoretical investigation and validation," IEEE sensors journal **15**, 5333–5339 (2015).

[59] J.-T. Lin, C. Newquist, and C. K. Harnett, "Multitouch pressure sensing with soft optical time-of-flight sensors," IEEE transactions on instrumentation and measurement **71**, 1–8 (2022).

[60] H. Zhao, R. Huang, and R. F. Shepherd, "Curvature control of soft orthotics via low cost solid-state optics," in "2016 IEEE International Conference on Robotics and Automation (ICRA)," (2016), pp. 4008–4013.

[61] C.-Y. Huang, W.-C. Wang, W.-J. Wu, and W. R. Ledoux, "Composite optical bend loss sensor for pressure and shear measurement," IEEE sensors journal **7**, 1554–1565 (2007).

[62] H. Bai, S. Li, J. Barreiros, Y. Tu, C. R. Pollock, and R. F. Shepherd, "Stretchable distributed fiber-optic sensors," Science **370**, 848–852 (2020).

[63] O. U. Lashmanov, A. S. Vasilev, A. V. Vasileva, A. G. Anisimov, and V. V. Korotaev, "High-precision absolute linear encoder based on a standard calibrated scale," Measurement **123**, 226–234 (2018).

[64] Usha and Sankar, "Binary orthogonal code generation for multi user communication using n-bit gray and inverse gray codes," Archiv fur Elektronik und Ubertragungstechnik [International journal of electronics and communications] (2012).

[65] D.-M. Pham, A. B. Premkumar, and A. S. Madhukumar, "Error detection and correction in communication channels using inverse gray RSNS codes," IEEE Transactions on Communications **59**, 975–986 (2011).

[66] G. H. Kim, S. M. Park, C. H. Park, H. Jang, C.-S. Kim, and H. D. Lee, "Real-time quasi-distributed fiber optic sensor based on resonance frequency mapping," Scientific reports **9**, 3921 (2019).

[67] L. Xu, N. Liu, J. Ge, X. Wang, and M. P. Fok, "Stretchable fiber-bragg-grating-based sensor," Optics letters **43**, 2503–2506 (2018).

[68] A. Fasano, G. Woyessa, P. Stajanca, C. Markos, A. Stefani, K. Nielsen, H. K. Rasmussen, K. Krebber, and O. Bang, "Fabrication and characterization of polycarbonate microstructured polymer optical fibers for high-temperature-resistant fiber bragg grating strain sensors," Optical Materials Express **6**, 649–659 (2016).

[69] S. T. Kreger, A. K. Sang, D. K. Gifford, and M. E. Froggatt, "Distributed strain and temperature sensing in plastic optical fiber using rayleigh scatter," in "Fiber Optic Sensors and Applications VI," , vol. 7316 (SPIE, 2009), vol. 7316, pp. 85–92.

[70] Y. Mizuno, A. Theodosiou, K. Kalli, S. Liehr, H. Lee, and K. Nakamura, "Distributed polymer optical fiber sensors: a review and outlook," Photonics Research **9**, 1719–1733 (2021).

[71] D. J. Webb, "Fibre bragg grating sensors in polymer optical fibres," Measurement science & technology **26**, 092004 (2015).

[72] A. Theodosiou and K. Kalli, "Recent trends and advances of fibre bragg grating sensors in CYTOP polymer optical fibres," Optical Fiber Technology **54**, 102079 (2020).

[73] E. Brookner, *Tracking and Kalman filtering made easy* (Wiley, New York, 1998).

[74] H. X. Tan, N. N. Aung, J. Tian, M. C. H. Chua, and Y. O. Yang, "Time series classification using a modified LSTM approach from accelerometer-based data: A comparative study for gait cycle detection," Gait & posture **74**, 128–134 (2019).

[75] H. T. T. Vu, F. Gomez, P. Cherelle, D. Lefeber, A. Nowé, and B. Vanderborght, "ED-FNN: A new deep learning algorithm to detect percentage of the gait cycle for powered prostheses," Sensors **18** (2018).

[76] S. Khandelwal and N. Wickström, "Novel methodology for estimating initial contact events from accelerometers positioned at different body locations," Gait & posture **59**, 278–285 (2018).

[77] D. Zeng, C. Qu, T. Ma, S. Qu, P. Yin, N. Zhao, and Y. Xia, "Research on a gait detection system and recognition algorithm for lower limb exoskeleton robot," Journal of the Brazilian Society of Mechanical Sciences and Engineering **43**, 298 (2021).

[78] B. Su, C. Smith, and E. Gutierrez Farewik, "Gait phase recognition using deep convolutional neural network with inertial measurement units," Biosensors **10** (2020).

[79] B. Chakraborty, "A proposal for classification of multisensor time series data based on time delay embedding," International Journal on Smart Sensing and Intelligent Systems **7**, 1–5 (2014).

[80] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning* (Springer New York, 2001).

[81] K. Kirasich, T. Smith, and B. Sadler, "Random forest vs logistic regression: Binary classification for heterogeneous datasets," SMU Data Science Review **1**, 9 (2018).

[82] V. Bahel, S. Pillai, and M. Malhotra, "A comparative study on various binary classification algorithms and their improved variant for optimal performance," in "2020 IEEE Region 10 Symposium (TENSYMP)," (2020), pp. 495–498.

[83] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification* (John Wiley & Sons, 2012).

[84] A. L. Evenchik, A. Q. Kane, E. Oh, and R. L. Truby, "Electrically controllable materials for soft, bioinspired machines," Annual review of materials research **53**, 225–251 (2023).

APPENDIX A: OPTIGAP DESIGN MANUAL

OptiGap Sensor System Design Manual

Objective of the OptiGap Sensor System
The OptiGap Sensor System is engineered to provide a robust, flexible solution for bend sensing in various applications ranging from robotics to wearables. It leverages optical transmittance in flexible light pipes interrupted by air gaps to detect bends at customizable sensitivity levels.

Target Applications
The system is highly adaptable and can be implemented in:

- Robotics: Specifically in limbs and joints
- Specialized cases like underwater robotics
- Wearable Technology: To monitor body movements
- Structural Monitoring: In buildings or machinery

Key Advantages

- High sensitivity in bend detection
- Operational in wet conditions
- Customizable properties such as bend sensitivity and sensing resolution
- Versatile attachment techniques

1. SYSTEM OVERVIEW

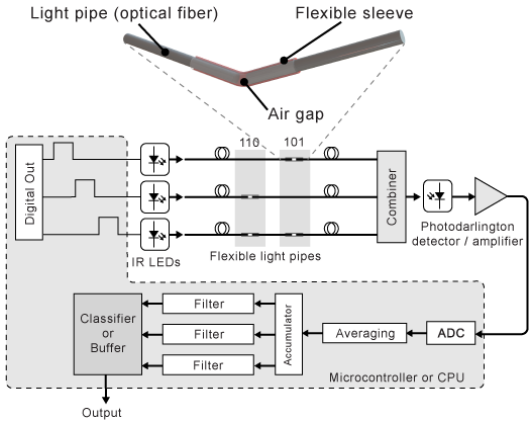
Working Mechanism

The core principle is the use of air gaps in parallel flexible light pipes to create bend sensitive segments which reduce the optical transmittance. The microcontroller or host interprets these variations in light transmittance to determine bending or bend location, depending on the application.

Components

The system consists mainly of three components:

- **Light Pipe**: Made from optical fibers of various materials, it is the core component for bend sensing.
- **Optical Source and detector**: LED or other light sources used to emit light into the light pipe. A phototransistor or similar device used to capture light from the light pipe.
- **Processor**: A microcontroller or computer used to emit and process the signals.



2. KEY PROPERTIES & PARAMETERS

PARAMETER	EFFECT ON SYSTEM	MIN	TYP	MAX	UNIT
Bend Sensitivity	Determines minimum bend angle for large intensity drop.		20		deg
Diameter	Affects overall light transmittance (tied to cone angle)		0.75		mm
Gap Length	Directly influences bend sensitivity at the cost of transmittance.		2		mm
Sensing Resolution	Specifies the minimum length between gaps	1	5		cm

Bend Sensitivity

- **General Applications**: If your application does not require high sensitivity to bends, a working angle of around 30 degrees is sufficient.
- **Precision Robotics**: For applications like surgical robotics where every degree matters, aim for a working angle of less than 10 degrees. This usually involves more comprehensive simulations and fine-tuning the air-gap patterns.
- **Wearable Devices**: If you're integrating the sensor into a wearable device, consider that most human joint angles don't exceed 120 degrees. A working angle of 20 to 40 degrees is typically suitable here.

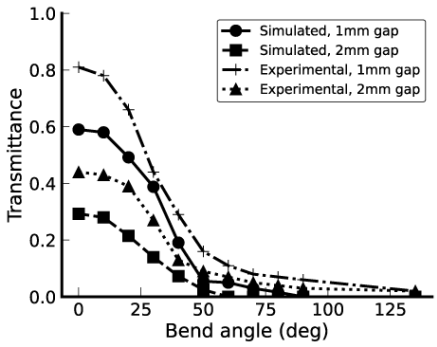


Figure 1

Diameter

- **Space-Constrained Applications**: In compact systems like wearables or miniaturized robotics, a smaller diameter would be more beneficial despite the lower transmittance.
- **Multi-Diameter Systems**: If the system uses light pipes of different diameters, it's preferable to use a light source with the smallest possible cone angle to maximize transmittance across all pipes.

Gap Length

- **High Sensitivity Systems**: For applications demanding high bend sensitivity, like biomechanical studies, longer gap lengths could be beneficial. If low transmittance is a concern, this can be offset with a more powerful light source.
- **Optimized Systems**: If you're looking for a balanced system, run simulations to find a gap length that offers a good trade-off between sensitivity and transmittance.
- **General Use**: For most applications, a gap length of 1 to 2mm will suffice.

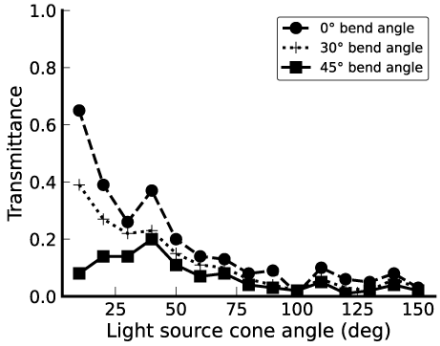


Figure 2

Sensing Resolution

- **Fine-Grain Sensing:** In applications like macro tactile sensing, a lower value closer to 1-2 cm may be preferable for higher resolution.

- **Robotic Arms or Industrial Systems:** These often don't require high resolution, and a 5-10 cm gap distance would be more than adequate.
- **Flexible or Soft Systems:** In applications involving softer materials or more complex bending patterns, finding an optimal sensing resolution may require iterative testing.

3. MATERIAL RECOMMENDATIONS

Light Pipe Material Options

MATERIAL	APPLICATIONS	DIAMETER (mm)	MAX OPERATING TEMP (°C)
Clear TPU 3D Printer Filament	Prototyping, non space-constrained applications, high temperature applications	1.75	190
PMMA	General, space-constrained	0.5 - 1	70

Air Gap Sleeve Material

MATERIAL	MAX OPERATING TEMP (°C)	Hardness (Durometer)	INNER DIAMETER (mm)
Silicone Rubber	215	55A	0.5 – 1.75

Emitter and Detector

TYPE	APPLICATION	SPECTRAL RANGE (NM)	EXAMPLE PART NUM
Phototransistor	For use in production	940	PT12-21C/TR8
IR Emitter	For use in production	940	IR12-21C/TR8
Photodarlington receiver	Prototyping, TPU light pipe	600 - 900	IF-D95OC
Emitter	Prototyping, TPU light pipe	640 - 660	IF-E97

APPENDIX B: MODEL CODE

TASL Code

```
"""
Helper library for the Dynamixel MX actuators using Protocol 2.0
Provides a 1-to-1 mapping of the entire control table as properties
    of an object.

* Author(s): Paul Bupe Jr
"""
from dynamixel_sdk.robotis_def import COMM_SUCCESS
from dynamixel_sdk import PacketHandler
from dynamixel_sdk import PortHandler
import logging


log = logging.getLogger(__name__)

# MX-28 Control Tables

# fmt: off
# EEPROM table
# Name                  (Address, Bytes)
MX_MODEL_NUMBER           = (0, 2)      # R
MX_MODEL_INFORMATION      = (2, 4)      # R
MX_FIRMWARE_VERSION       = (6, 1)     # R
MX_ID                     = (7, 1)     # RW
MX_BAUD_RATE              = (8, 1)     # RW
MX_RETURN_DELAY_TIME      = (9, 1)     # RW
MX_DRIVE_MODE             = (10, 1)     # RW
MX_OPERATING_MODE         = (11, 1)     # RW
MX_SECONDARY_ID           = (12, 1)     # RW
MX_PROTOCOL_TYPE          = (13, 1)     # RW
MX_HOMING_OFFSET          = (20, 4)     # RW
MX_MOVING_THRESHOLD       = (24, 4)     # RW
MX_TEMPERATURE_LIMIT      = (31, 1)     # RW
MX_MAX_VOLTAGE_LIMIT      = (32, 2)     # RW
MX_MIN_VOLTAGE_LIMIT      = (34, 2)     # RW
MX_PWM_LIMIT              = (36, 2)     # RW
MX_ACCELERATION_LIMIT     = (40, 4)     # RW
MX_VELOCITY_LIMIT         = (44, 4)     # RW
MX_MAX_POSITION_LIMIT     = (48, 4)     # RW
MX_MIN_POSITION_LIMIT     = (52, 4)     # RW
MX_SHUTDOWN               = (63, 1)     # RW

# RAM table
MX_TORQUE_ENABLE          = (64, 1)     # RW
MX_LED                    = (65, 1)     # RW
```

```
MX_STATUS_RETURN_LEVEL    = (68, 1)      # RW
MX_REGISTERED_INSTRUCTION = (69, 1)      # R
MX_HARDWARE_ERROR_STATUS  = (70, 1)      # R
MX_VELOCITY_I_GAIN        = (76, 2)      # RW
MX_VELOCITY_P_GAIN        = (78, 2)      # RW
MX_POSITION_D_GAIN        = (80, 2)      # RW
MX_POSITION_I_GAIN        = (82, 2)      # RW
MX_POSITION_P_GAIN        = (84, 2)      # RW
MX_FEEDFORWARD_2ND_GAIN   = (88, 2)      # RW
MX_FEEDFORWARD_1ST_GAIN   = (90, 2)      # RW
MX_BUS_WATCHDOG           = (98, 1)      # RW
MX_GOAL_PWM               = (100, 2)      # RW
MX_GOAL_VELOCITY          = (104, 4)      # RW
MX_PROFILE_ACCELERATION   = (108, 4)      # RW
MX_PROFILE_VELOCITY       = (112, 4)      # RW
MX_GOAL_POSITION          = (116, 4)      # RW
MX_REALTIME_TICK          = (120, 2)      # R
MX_MOVING                 = (122, 1)      # R
MX_MOVING_STATUS          = (123, 1)      # R
MX_PRESENT_PWM            = (124, 2)      # R
MX_PRESENT_LOAD           = (126, 2)      # R
MX_PRESENT_VELOCITY       = (128, 4)      # R
MX_PRESENT_POSITION       = (132, 4)      # R
MX_VELOCITY_TRAJECTORY    = (136, 4)      # R
MX_POSITION_TRAJECTORY    = (140, 4)      # R
MX_PRESENT_INPUT_VOLTAGE  = (144, 2)      # R
MX_PRESENT_TEMPERATURE    = (146, 1)      # R
# fmt: on


class Pymixel(object):
    """Helper class for the Dynamixel MX-Series actuators
    using Protocol 2.0
    """

    # SDK classes
    port_handler = None
    packet_handler = None
    port = None
    baudrate = None

    def __init__(self, port, mxl_id, baudrate=57600, protocol=2):
        """Create an instance of the Pymixel class"""
        self._id = mxl_id
        self.port = port
        self.baudrate = baudrate
        self.port_handler = PortHandler(port)
        self.packet_handler = PacketHandler(protocol)

        # Open port
        if not self.port_handler.openPort():
            raise Exception(f"ERROR: Could not open port {self.port}
    ")

```

```
98          if not self.port_handler.setBaudRate(self.baudrate):
99              raise Exception("ERROR: Failed to change the baudrate")
100
101     def _read(self, register):
102         """Read data from a register"""
103
104         addr, num_bytes = register
105         data = response = error = None
106         if num_bytes == 1:
107             data, response, error = self.packet_handler.
    read1ByteTxRx(
108                 self.port_handler, self._id, addr
109             )
110         elif num_bytes == 2:
111             data, response, error = self.packet_handler.
    read2ByteTxRx(
112                 self.port_handler, self._id, addr
113             )
114         else:
115             data, response, error = self.packet_handler.
    read4ByteTxRx(
116                 self.port_handler, self._id, addr
117             )
118
119         # Check response
120         self._error_handler(response, error)
121
122         return data
123
124     def _write(self, register, value):
125         """write to a register"""
126
127         addr, num_bytes = register
128         response = error = None
129         if num_bytes == 1:
130             response, error = self.packet_handler.write1ByteTxRx(
131                 self.port_handler, self._id, addr, value
132             )
133         elif num_bytes == 2:
134             response, error = self.packet_handler.write2ByteTxRx(
135                 self.port_handler, self._id, addr, value
136             )
137         else:
138             response, error = self.packet_handler.write4ByteTxRx(
139                 self.port_handler, self._id, addr, value
140             )
141
142         # Check response
143         self._error_handler(response, error)
144
145     @property
146     def model_number(self):
147         """Return the model number."""
148         return self._read(MX_MODEL_NUMBER)
```

```
149
150     @property
151     def model_information(self):
152         """Return the model information."""
153         return self._read(MX_MODEL_INFORMATION)
154
155     @property
156     def firmware_version(self):
157         """Return the firmware version."""
158         return self._read(MX_FIRMWARE_VERSION)
159
160     @property
161     def id(self): # pylint: disable=invalid-name
162         """Return the DYNAMIXEL ID."""
163         return self._read(MX_ID)
164
165     @property
166     def baud_rate(self):
167         """Return the serial baud rate."""
168         return self._read(MX_BAUD_RATE)
169
170     @property
171     def return_delay_time(self):
172         """Return the response delay."""
173         return self._read(MX_RETURN_DELAY_TIME)
174
175     @property
176     def drive_mode(self):
177         """Return the drive mode."""
178         return self._read(MX_DRIVE_MODE)
179
180     @property
181     def operating_mode(self):
182         """Return the operating mode."""
183         return self._read(MX_OPERATING_MODE)
184
185     @property
186     def secondary_id(self):
187         """Return the secondary ID."""
188         return self._read(MX_SECONDARY_ID)
189
190     @property
191     def protocol_type(self):
192         """Return the protocol type."""
193         return self._read(MX_PROTOCOL_TYPE)
194
195     @property
196     def homing_offset(self):
197         """Return the home position offset."""
198         return self._read(MX_HOMING_OFFSET)
199
200     @property
201     def moving_threshold(self):
202         """Return the velocity threshold for
```

```
203         movement detection."""
204         return self._read(MX_MOVING_THRESHOLD)
205 
206     @property
207     def temperature_limit(self):
208         """Return the maximum internal temperature limit."""
209         return self._read(MX_TEMPERATURE_LIMIT)
210 
211     @property
212     def max_voltage_limit(self):
213         """Return the maximum input voltage limit."""
214         return self._read(MX_MAX_VOLTAGE_LIMIT)
215 
216     @property
217     def min_voltage_limit(self):
218         """Return the minimum input voltage limit."""
219         return self._read(MX_MIN_VOLTAGE_LIMIT)
220 
221     @property
222     def pwm_limit(self):
223         """Return the maximum PWM limit."""
224         return self._read(MX_PWM_LIMIT)
225 
226     @property
227     def acceleration_limit(self):
228         """Return the maximum acceleration limit."""
229         return self._read(MX_ACCELERATION_LIMIT)
230 
231     @property
232     def velocity_limit(self):
233         """Return the maximum velocity limit."""
234         return self._read(MX_VELOCITY_LIMIT)
235 
236     @property
237     def max_position_limit(self):
238         """Return the maximum position limit."""
239         return self._read(MX_MAX_POSITION_LIMIT)
240 
241     @property
242     def min_position_limit(self):
243         """Return the minimum position limit."""
244         return self._read(MX_MIN_POSITION_LIMIT)
245 
246     @property
247     def shutdown(self):
248         """Return the shutdown error information."""
249         return self._read(MX_SHUTDOWN)
250 
251     @property
252     def torque_enable(self):
253         """Return the motor torque status."""
254         return self._read(MX_TORQUE_ENABLE)
255 
256     @property
```

```
257     def led(self):
258         """Return the status LED status."""
259         return self._read(MX_LED)
260 
261     @property
262     def status_return_level(self):
263         """Return the status return level."""
264         return self._read(MX_STATUS_RETURN_LEVEL)
265 
266     @property
267     def registered_instruction(self):
268         """Return the REG_WRITE instruction flag."""
269         return self._read(MX_REGISTERED_INSTRUCTION)
270 
271     @property
272     def hardware_error_status(self):
273         """Return the hardware error status."""
274         return self._read(MX_HARDWARE_ERROR_STATUS)
275 
276     @property
277     def velocity_i_gain(self):
278         """Return the I gain of velocity."""
279         return self._read(MX_VELOCITY_I_GAIN)
280 
281     @property
282     def velocity_p_gain(self):
283         """Return the P gain of velocity."""
284         return self._read(MX_VELOCITY_P_GAIN)
285 
286     @property
287     def position_d_gain(self):
288         """Return the D gain of position."""
289         return self._read(MX_POSITION_D_GAIN)
290 
291     @property
292     def position_i_gain(self):
293         """Return the I gain of position."""
294         return self._read(MX_POSITION_I_GAIN)
295 
296     @property
297     def position_p_gain(self):
298         """Return the P gain of position."""
299         return self._read(MX_POSITION_P_GAIN)
300 
301     @property
302     def feedforward_2nd_gain(self):
303         """Return the 2nd gain of feed-forward."""
304         return self._read(MX_FEEDFORWARD_2ND_GAIN)
305 
306     @property
307     def feedforward_1st_gain(self):
308         """Return the 1st gain of feed-forward."""
309         return self._read(MX_FEEDFORWARD_1ST_GAIN)
310 
```

```
311     @property
312     def bus_watchdog(self):
313         """Return the dynamixel bus watchdog."""
314         return self._read(MX_BUS_WATCHDOG)
315 
316     @property
317     def goal_pwm(self):
318         """Return the desired pwm value."""
319         return self._read(MX_GOAL_PWM)
320 
321     @property
322     def goal_velocity(self):
323         """Return the desired velocity value."""
324         return self._read(MX_GOAL_VELOCITY)
325 
326     @property
327     def profile_acceleration(self):
328         """Return the acceleration value of profile."""
329         return self._read(MX_PROFILE_ACCELERATION)
330 
331     @property
332     def profile_velocity(self):
333         """Return the velocity value of profile."""
334         return self._read(MX_PROFILE_VELOCITY)
335 
336     @property
337     def goal_position(self):
338         """Return the desired position."""
339         return self._read(MX_GOAL_POSITION)
340 
341     @property
342     def realtime_tick(self):
343         """Return the count time in millisecond."""
344         return self._read(MX_REALTIME_TICK)
345 
346     @property
347     def moving(self):
348         """Return the movement flag."""
349         return self._read(MX_MOVING)
350 
351     @property
352     def moving_status(self):
353         """Return detailed information of movement status."""
354         return self._read(MX_MOVING_STATUS)
355 
356     @property
357     def present_pwm(self):
358         """Return the present pwm value."""
359         return self._read(MX_PRESENT_PWM)
360 
361     @property
362     def present_load(self):
363         """Return the present load value."""
364         return self._read(MX_PRESENT_LOAD)
```

```
365
366     @property
367     def present_velocity(self):
368         """Return the present velocity value."""
369         return self._read(MX_PRESENT_VELOCITY)
370
371     @property
372     def present_position(self):
373         """Return the present position value."""
374         return self._read(MX_PRESENT_POSITION)
375
376     @property
377     def velocity_trajectory(self):
378         """Return the desired velocity trajectory from profile."""
379         return self._read(MX_VELOCITY_TRAJECTORY)
380
381     @property
382     def position_trajectory(self):
383         """Return the desired position trajectory from profile."""
384         return self._read(MX_POSITION_TRAJECTORY)
385
386     @property
387     def present_input_voltage(self):
388         """Return the present input voltage."""
389         return self._read(MX_PRESENT_INPUT_VOLTAGE)
390
391     @property
392     def present_temperature(self):
393         """Return the present internal temperature."""
394         return self._read(MX_PRESENT_TEMPERATURE)
395
396     @id.setter
397     def id(self, value): # pylint: disable=invalid-name
398         """Set the DYNAMIXEL ID."""
399         self._write(MX_ID, value)
400
401     @baud_rate.setter
402     def baud_rate(self, value):
403         """Set the serial baud rate."""
404         self._write(MX_BAUD_RATE, value)
405
406     @return_delay_time.setter
407     def return_delay_time(self, value):
408         """Set the response delay."""
409         self._write(MX_RETURN_DELAY_TIME, value)
410
411     @drive_mode.setter
412     def drive_mode(self, value):
413         """Set the drive mode."""
414         self._write(MX_DRIVE_MODE, value)
415
416     @operating_mode.setter
417     def operating_mode(self, value):
418         """Set the operating mode."""
```

```
419             self._write(MX_OPERATING_MODE, value)
420
421     @secondary_id.setter
422     def secondary_id(self, value):
423         """Set the secondary ID."""
424         self._write(MX_SECONDARY_ID, value)
425
426     @protocol_type.setter
427     def protocol_type(self, value):
428         """Set the protocol type."""
429         self._write(MX_PROTOCOL_TYPE, value)
430
431     @homing_offset.setter
432     def homing_offset(self, value):
433         """Set the home position offset."""
434         self._write(MX_HOMING_OFFSET, value)
435
436     @moving_threshold.setter
437     def moving_threshold(self, value):
438         """Set the velocity threshold for
439         movement detection."""
440         self._write(MX_MOVING_THRESHOLD, value)
441
442     @temperature_limit.setter
443     def temperature_limit(self, value):
444         """Set the maximum internal temperature limit."""
445         self._write(MX_TEMPERATURE_LIMIT, value)
446
447     @max_voltage_limit.setter
448     def max_voltage_limit(self, value):
449         """Set the maximum input voltage limit."""
450         self._write(MX_MAX_VOLTAGE_LIMIT, value)
451
452     @min_voltage_limit.setter
453     def min_voltage_limit(self, value):
454         """Set the minimum input voltage limit."""
455         self._write(MX_MIN_VOLTAGE_LIMIT, value)
456
457     @pwm_limit.setter
458     def pwm_limit(self, value):
459         """Set the maximum PWM limit."""
460         self._write(MX_PWM_LIMIT, value)
461
462     @acceleration_limit.setter
463     def acceleration_limit(self, value):
464         """Set the maximum acceleration limit."""
465         self._write(MX_ACCELERATION_LIMIT, value)
466
467     @velocity_limit.setter
468     def velocity_limit(self, value):
469         """Set the maximum velocity limit."""
470         self._write(MX_VELOCITY_LIMIT, value)
471
472     @max_position_limit.setter
```

```
473     def max_position_limit(self, value):
474         """Set the maximum position limit."""
475         self._write(MX_MAX_POSITION_LIMIT, value)
476 
477     @min_position_limit.setter
478     def min_position_limit(self, value):
479         """Set the minimum position limit."""
480         self._write(MX_MIN_POSITION_LIMIT, value)
481 
482     @shutdown.setter
483     def shutdown(self, value):
484         """Set the shutdown error information."""
485         self._write(MX_SHUTDOWN, value)
486 
487     @torque_enable.setter
488     def torque_enable(self, value):
489         """Set the motor torque status."""
490         self._write(MX_TORQUE_ENABLE, value)
491 
492     @led.setter
493     def led(self, value):
494         """Set the status LED status."""
495         self._write(MX_LED, value)
496 
497     @status_return_level.setter
498     def status_return_level(self, value):
499         """Set the status return level."""
500         self._write(MX_STATUS_RETURN_LEVEL, value)
501 
502     @velocity_i_gain.setter
503     def velocity_i_gain(self, value):
504         """Set the I gain of velocity."""
505         self._write(MX_VELOCITY_I_GAIN, value)
506 
507     @velocity_p_gain.setter
508     def velocity_p_gain(self, value):
509         """Set the P gain of velocity."""
510         self._write(MX_VELOCITY_P_GAIN, value)
511 
512     @position_d_gain.setter
513     def position_d_gain(self, value):
514         """Set the D gain of position."""
515         self._write(MX_POSITION_D_GAIN, value)
516 
517     @position_i_gain.setter
518     def position_i_gain(self, value):
519         """Set the I gain of position."""
520         self._write(MX_POSITION_I_GAIN, value)
521 
522     @position_p_gain.setter
523     def position_p_gain(self, value):
524         """Set the P gain of position."""
525         self._write(MX_POSITION_P_GAIN, value)
526 
```

```
527     @feedforward_2nd_gain.setter
528     def feedforward_2nd_gain(self, value):
529         """Set the 2nd gain of feed-forward."""
530         self._write(MX_FEEDFORWARD_2ND_GAIN, value)
531 
532     @feedforward_1st_gain.setter
533     def feedforward_1st_gain(self, value):
534         """Set the 1st gain of feed-forward."""
535         self._write(MX_FEEDFORWARD_1ST_GAIN, value)
536 
537     @bus_watchdog.setter
538     def bus_watchdog(self, value):
539         """Set the dynamixel bus watchdog."""
540         self._write(MX_BUS_WATCHDOG, value)
541 
542     @goal_pwm.setter
543     def goal_pwm(self, value):
544         """Set the desired pwm value."""
545         self._write(MX_GOAL_PWM, value)
546 
547     @goal_velocity.setter
548     def goal_velocity(self, value):
549         """Set the desired velocity value."""
550         self._write(MX_GOAL_VELOCITY, value)
551 
552     @profile_acceleration.setter
553     def profile_acceleration(self, value):
554         """Set the acceleration value of profile."""
555         self._write(MX_PROFILE_ACCELERATION, value)
556 
557     @profile_velocity.setter
558     def profile_velocity(self, value):
559         """Set the velocity value of profile."""
560         self._write(MX_PROFILE_VELOCITY, value)
561 
562     @goal_position.setter
563     def goal_position(self, value):
564         """Set the desired position."""
565         self._write(MX_GOAL_POSITION, value)
566 
567     def __del__(self):
568         self._close()
569 
570     def _close(self):
571         # Closes the port
572         self.port_handler.closePort()
573 
574     def _error_handler(self, res, err):
575         # Checks each request
576         if res != COMM_SUCCESS:
577             print(f"Abnormal response: {self.packet_handler.
    getTxRxResult(res)}")
578         elif err != 0:
579             print(f"Error: {self.packet_handler.getRxPacketError(err
```

```
    )}")


if __name__ == '__main__':
    print("Debug mode...")
    import time

    motor = Pymixel("/dev/ttyUSB0", 1)
    motor.torque_enable = 1

    while True:
        print(f'Voltage is: {motor.present_input_voltage / 10} V')
        print(f'Temperature is {motor.present_temperature} degrees C
    ')
        time.sleep(1)
```

Listing V.1. "Dynamixel driver.

```
"""
Simple extensible serial protocol for PC <-> Arduino communication

**<START><ACTION><DELIMITER><COMMAND><DELIMITER><DATA><END>**
* The START byte is ASCII "{" (123 decimal, 0x7B).
* The ACTION byte is an ASCII "?" for GET and ASCII "$" for SET.
* The COMMAND is up to three characters.
* The DATA bytes are variable length and are described in the Data
    section.
* The END byte is ASCII "}" (125 decimal, 0x7D).
* The CHECKSUM is calculated by subtracting 32 from all the
    characters in the packet(excluding the checksum) and summing them
    .The modulo 95 of this value is then calculated and 32 is added
    back to that value.

{?,S,234}@

* Author(s): Paul Bupe Jr
"""

import logging
import time
import math
import sys
import glob
import serial

log = logging.getLogger(__name__)

START                = '{'
END                  = '}'

class ScramIO(object):
    '''Python implementation of SCRAMio Protocol'''

    def __init__(self, serial_port, baudrate=9600, timeout=0.1):
```

```
33
34          if serial is None:
35              raise ImportError("Package pyserial is required for
    serial connection")
36
37          self.serial_port = serial.Serial(port=serial_port, baudrate=
    baudrate, timeout=timeout)
38          time.sleep(2)
39          log.info("Arduino Connected")
40
41      def _write(self, packet):
42          """Write to the serial port"""
43          # self.serial_port.write(bytes(packet, 'utf-8'))
44          self.serial_port.write(packet.encode('utf-8'))
45          # time.sleep(0.05)
46          # res = self.serial_port.readline().decode('utf-8').rstrip()
47          # print(res)
48
49      def command(self, command, data):
50          """
51          :param motor:
52          :param value:
53          :return:
54          """
55          packet = [START, '@',',',command,',', str(data),END]
56          # packet.append(self._calculate_checksum(packet))
57          packet = "".join(packet)
58          log.debug(packet)
59          self._write(packet)
60          # print(data)
61          # try:
62              # self._device.writeList(MOTOR, packet)
63          # except:
64              # e = sys.exc_info()[0]
65              # log.error("Send Error: %s" % e)
66
67      @staticmethod
68      def serial_ports():
69          """ Lists serial port names
70
71              :raises EnvironmentError:
72                  On unsupported or unknown platforms
73              :returns:
74                  A list of the serial ports available on the system
75          """
76          if sys.platform.startswith('win'):
77              ports = ['COM%s' % (i + 1) for i in range(256)]
78          elif sys.platform.startswith('linux') or sys.platform.
    startswith('cygwin'):
79              # this excludes your current terminal "/dev/tty"
80              ports = glob.glob('/dev/tty[A-Za-z]*')
81          elif sys.platform.startswith('darwin'):
82              ports = glob.glob('/dev/tty.*')
83          else:
```

```
            raise EnvironmentError('Unsupported platform')
        result = []
        for port in ports:
            try:
                s = serial.Serial(port)
                s.close()
                result.append(port)
            except (OSError, serial.SerialException) as e:
                # if e.errno == 13:
                    # raise e
                pass
        return result

    @staticmethod
    def _calculate_checksum(packet):
        """
        :param packet:
        :return:
        """
        checksum = 0
        for c in packet:
            if (c != START) and (c != END):
                try:
                    checksum += c - 32
                except TypeError:
                    checksum += ord(c) - 32
        return (checksum % 95) + 32

if __name__ == '__main__':
    # obj = ScramIO('/dev/ttyACM0')
    # obj.command("CM",1)
    # ser = serial.Serial('/dev/ttyACM0')
    print(ScramIO.serial_ports())
    # print(ser)
```

Listing V.2. "Serial bridge"

```
import os
# from smascram.scramio import ScramIO
import sys
from logging import debug, basicConfig
from flask import Flask, render_template
from flask_socketio import SocketIO, emit
import socketio
from pymixel2 import Pymixel
from scramio import ScramIO

# append the path of the parent directory
# sys.path.append(os.path.abspath('./pymixel'))
# print(sys.path)


basicConfig(level=os.environ.get("LOGLEVEL", "INFO"),
```

```
                    format="%(asctime)s [%(levelname)s] %(name)s: %(
    message)s")

app = Flask(__name__)
app.config['SECRET_KEY'] = 'vn4dn48hd%8$2#2'
socketio = SocketIO(app)

dynamixel = None
arduino = None

@app.route('/')
def sessions():
    return render_template('index.html')

@socketio.on('connect')
def init():
    port_list = ScramIO.serial_ports()
    emit('ports', port_list)

@socketio.on('actuator')
def update_actuator(actuator):
    print(actuator)

    dynamixel.torque_enable = 0
    dynamixel.operating_mode = 1
    dynamixel.torque_enable = int(actuator["torque"])
    dynamixel.goal_velocity = int(actuator["rpm"])

@socketio.on('segments')
def update_segments(segment):
    arduino.command("CM",segment)
    # print(segment)


if __name__ == '__main__':
    socketio.run(app, port=5000, debug=True)
```

Listing V.3. "Server (main.py)"

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1
    ">
  <meta name="description" content="HarnettLab">
  <meta name="author" content="Paul Bupe Jr">
  <title>SMA SCRAM Control</title>



  <!-- Bootstrap core CSS -->
  <link href="static/css/bootstrap.min.css" rel="stylesheet">
```

```
15
16    <!-- Favicons -->
17    <!-- <link rel="apple-touch-icon" href="/docs/5.1/assets/img/
       favicons/apple-touch-icon.png" sizes="180x180"> -->
18    <!-- <link rel="icon" href="/docs/5.1/assets/img/favicons/favicon
       -32x32.png" sizes="32x32" type="image/png"> -->
19    <!-- <link rel="icon" href="/docs/5.1/assets/img/favicons/favicon
       -16x16.png" sizes="16x16" type="image/png"> -->
20    <!-- <link rel="manifest" href="/docs/5.1/assets/img/favicons/
       manifest.json"> -->
21    <!-- <link rel="mask-icon" href="/docs/5.1/assets/img/favicons/
       safari-pinned-tab.svg" color="#7952b3"> -->
22    <!-- <link rel="icon" href="/docs/5.1/assets/img/favicons/favicon.
       ico"> -->
23    <!-- <meta name="theme-color" content="#7952b3"> -->
24
25    <!-- Custom styles for this template -->
26    <link href="static/css/scram.css" rel="stylesheet">
27  </head>
28
29  <body>
30
31    <div class="container">
32      <header class="py-1 mb-3 border-bottom">
33        <div class="position-relative">
34
35          <h3>SCRAM SMA Driver</h3>
36
37          <div id="serverBadge" class="badge bg-danger position-
       absolute top-50 end-0">Server Disconnected</div>
38        </div>
39      </header>
40
41      <main>
42
43        <div class="row">
44          <div class="col-md-9">
45
46
47            <div class="card mb-2 d-inline-block" style="width: 9rem;"
       >
48              <div class="card-header">
49                Input Current
50              </div>
51              <div class="card-body p-1">
52                <h5 class="text-center">0 A</h5>
53              </div>
54            </div>
55
56            <div class="card mb-2 d-inline-block" style="width: 9rem;"
       >
57              <div class="card-header">
58                Pulling Force
59              </div>
```

```
            <div class="card-body p-1">
              <h5 class="text-center">0</h5>
            </div>
          </div>


          <div class="row row-cols-5 g-0">
            <div class="col">
              <div class="d-grid gap-1 m-1">
                <span id="segBadge5" class="badge bg-secondary">OFF
</span>
              </div>
              <div class="card">
                <img src="static/img/seg_off.png" id="segImg5" class
="card-img-top" alt="...">
                <div class="card-body p-1">
                  <div class="d-grid gap-2">
                    <button type="button" data-segment="5" class="
segButton btn btn-secondary btn-sm">ENABLE</button>
                  </div>
                </div>
              </div>
            </div>
            <div class="col">
              <div class="d-grid gap-1 m-1">
                <span id="segBadge4" class="badge bg-secondary">OFF
</span>
              </div>
              <div class="card">
                <img src="static/img/seg_off.png" id="segImg4" class
="card-img-top" alt="...">
                <div class="card-body p-1">
                  <div class="d-grid gap-2">
                    <button type="button" data-segment="4" class="
segButton btn btn-secondary btn-sm">ENABLE</button>
                  </div>
                </div>
              </div>
            </div>
            <div class="col">
              <div class="d-grid gap-1 m-1">
                <span id="segBadge3" class="badge bg-secondary">OFF
</span>
              </div>
              <div class="card ">
                <img src="static/img/seg_off.png" id="segImg3" class
="card-img-top" alt="...">
                <div class="card-body p-1">
                  <div class="d-grid gap-2">
                    <button type="button" data-segment="3" class="
segButton btn btn-secondary btn-sm">Enable</button>
                  </div>
                </div>
              </div>
```

```
105             </div>
106
107             <div class="col">
108               <div class="d-grid gap-1 m-1">
109                 <span id="segBadge2" class="badge bg-secondary">OFF
    </span>
110               </div>
111               <div class="card">
112                 <img src="static/img/seg_off.png" id="segImg2" class
    ="card-img-top" alt="...">
113                 <div class="card-body p-1">
114                   <div class="d-grid gap-2">
115                     <button type="button" data-segment="2" class="
    segButton btn btn-secondary btn-sm">ENABLE</button>
116                   </div>
117                 </div>
118               </div>
119             </div>
120             <div class="col">
121               <div class="d-grid gap-1 m-1">
122                 <span id="segBadge1" class="badge bg-secondary">OFF
    </span>
123               </div>
124               <div class="card">
125                 <img src="static/img/seg_off.png" id="segImg1" class
    ="card-img-top" alt="...">
126                 <div class="card-body p-1">
127                   <div class="d-grid gap-2">
128                     <button type="button" data-segment="1" class="
    segButton btn btn-secondary btn-sm">ENABLE</button>
129                   </div>
130                 </div>
131               </div>
132             </div>
133           </div>
134           <div class="d-grid gap-2 my-2">
135             <button type="button" id="disableAll" class="btn btn-
    danger btn-sm">DISABLE ALL</button>
136           </div>
137
138
139         </div>
140         <div class="col">
141           <div class="card mb-2">
142             <div class="card-header">
143               Driver Board <span class="badge bg-secondary">Offline
    </span>
144             </div>
145             <div class="card-body">
146
147               <label for="driverPort" class="form-label">Serial port
    </label>
148               <select id="driverPort" class="form-select form-select
    -sm portSelect" data-device="driver" aria-label=".form-select-sm"
```

```
    >
149             <option value="--">--</option>
150             </select>
151 
152             <button type="button" class="btn btn-outline-secondary
     btn-sm my-2">Refresh List</button>
153             <button type="button" class="btn btn-dark btn-sm my-2"
    >Connect</button>
154           </div>
155         </div>
156 
157         <div class="card">
158           <div class="card-header">Actuator <span class="badge bg-
    secondary">Offline</span></div>
159           <div class="card-body">
160 
161             <label for="actuatorPort" class="form-label">Serial
    port</label>
162             <select id="actuatorPort" class="form-select form-
    select-sm portSelect" data-device="actuator"  aria-label=".form-
    select-sm">
163             <option value="--">--</option>
164             </select>
165 
166             <button type="button" class="btn btn-outline-secondary
     btn-sm my-2">Refresh List</button>
167             <button type="button" class="btn btn-dark btn-sm my-2"
    >Connect</button>
168 
169 
170             <hr>
171 
172             <div>
173               <label for="rpm" class="form-label">Motor Speed (RPM
    )</label>
174               <h4 id="rpmLabel" class="font-weight-bold text-
    primary text-center ml-2 valueSpan2">50</h4>
175               <input type="range" class="form-range" autocomplete=
    "off" min="1" max="100" step="1" id="rpm">
176             </div>
177             <br>
178             <div class="d-grid gap-2">
179               <div class="btn-group btn-group-sm" role="group"
    aria-label="Basic radio toggle button group">
180                 <input type="radio" class="btn-check" autocomplete
    ="off" name="btnradio" id="forward" autocomplete="off" checked>
181                 <label class="btn btn-outline-primary" for="
    forward">Forward</label>
182 
183                 <input type="radio" class="btn-check" autocomplete
    ="off" name="btnradio" id="reverse" autocomplete="off">
184                 <label class="btn btn-outline-primary" for="
    reverse">Reverse</label>
185               </div>
```

```
186                 </div>
187                 <hr>
188                 <div class="d-grid gap-2 ">
189                   <button id="run" class="btn btn-success" type="
    button">Run</button>
190                   <button id="stop" class="btn btn-danger" type="
    button">Stop</button>
191                 </div>
192               </div>
193             </div>
194
195           </div>
196         </div>
197
198       </main>
199       <footer class="pt-5 my-5 text-muted border-top">
200         HarnettLab &middot; &copy; 2021
201       </footer>
202     </div>
203
204
205     <script src="static/js/bootstrap.bundle.min.js"></script>
206     <script src="static/js/socketio.min.js"></script>
207     <script src="static/js/main.js"></script>
208
209
210 </body>
211
212 </html>
```

Listing V.4. "Frontend (index.html)"

OptiGap Code

```
1  import sys
2  import csv
3  import pyqtgraph as pg
4  from PyQt6.QtCore import Qt, QIODevice, QTimer, pyqtSignal, pyqtSlot
       , QObject, QThread
5  from PyQt6.QtSerialPort import QSerialPort, QSerialPortInfo
6  from PyQt6.QtWidgets import (QApplication, QMainWindow, QGridLayout,
        QVBoxLayout, QWidget, QComboBox,
7                               QPushButton, QHBoxLayout, QTextEdit,
       QLineEdit,
8                               QFileDialog, QCheckBox, QLabel,
       QStatusBar, QTableWidget,
9                               QTableWidgetItem, QHeaderView)
10
11 import numpy as np
12 import qdarktheme
13
14 pg.setConfigOptions(antialias=True)
15
```

```
16
17 class MainWindow(QMainWindow):
18     serial_data_received = pyqtSignal(str)
19
20     def __init__(self):
21         super().__init__()
22
23         self.setWindowTitle("OptiGap Realtime Plotting App")
24         self.resize(1300, 600)
25
26         main_widget = QWidget()
27         self.setCentralWidget(main_widget)
28
29         main_layout = QGridLayout()
30         main_widget.setLayout(main_layout)
31
32         # Top-level layouts
33         self.top_menu_layout = QHBoxLayout()
34         self.plot_layout = QVBoxLayout()
35         self.sidebar_layout = QVBoxLayout()
36
37         self.num_detected_signals = 0
38         self.region = (0, 1)
39         self.plot_lines = []
40         self.labels = []  # [(label, region), ...]
41         self.selected_row = -1
42         self.checkboxes = []
43         self.capture_enabled = False
44         self.plot_data = None
45         self.serial_connected = False
46
47         # Create a status bar and add it to the main window
48         self.status_bar = CustomStatusBar()
49         self.setStatusBar(self.status_bar)
50
51         # Top Menu
52         self.serial_port_dropdown = QComboBox()
53         self.top_menu_layout.addWidget(self.serial_port_dropdown)
54
55         self.refresh_button = QPushButton("Refresh Ports")
56         self.top_menu_layout.addWidget(self.refresh_button)
57
58         self.baud_rate_dropdown = QComboBox()
59         self.baud_rate_dropdown.addItems(["9600", "19200", "38400",
   "57600", "115200"])
60         self.top_menu_layout.addWidget(self.baud_rate_dropdown)
61
62         self.connect_button = QPushButton("Connect")
63         self.top_menu_layout.addWidget(self.connect_button)
64
65         self.disconnect_button = QPushButton("Disconnect")
66         self.disconnect_button.setEnabled(False)
67         self.top_menu_layout.addWidget(self.disconnect_button)
68
```

```
        self.capture_button = QPushButton("Start Capture")
        self.capture_button.setEnabled(False)
        self.top_menu_layout.addWidget(self.capture_button)

        self.clear_button = QPushButton("Clear Data")
        self.top_menu_layout.addWidget(self.clear_button)

        self.import_button = QPushButton("Import CSV")
        self.top_menu_layout.addWidget(self.import_button)

        self.export_button = QPushButton("Export All Data")
        self.top_menu_layout.addWidget(self.export_button)

        # Add a png logo to the top menu
        logo = pg.QtGui.QPixmap("optigap_logo.png")
        logo = logo.scaledToHeight(25)
        logo_label = QLabel()
        logo_label.setPixmap(logo)
        self.top_menu_layout.addWidget(logo_label)

        # Connect UI elements with their respective functionality
        self.capture_button.clicked.connect(self.toggle_capture)
        self.clear_button.clicked.connect(self.clear_plot_data)
        self.import_button.clicked.connect(self.import_csv)
        self.export_button.clicked.connect(self.export_data)
        self.refresh_button.clicked.connect(self.
    detect_serial_devices)
        self.connect_button.clicked.connect(self.connect_serial)
        self.disconnect_button.clicked.connect(self.
    disconnect_serial)
        self.baud_rate_dropdown.currentIndexChanged.connect(self.
    update_baud_rate)

        main_layout.addLayout(self.top_menu_layout, 0, 0, 1, 4)

        # Serial Monitor
        self.serial = QSerialPort()
        self.detect_serial_devices()

        self.serial_worker = SerialWorker(self.serial)

        self.serial_thread = QThread()
        self.serial_worker.moveToThread(self.serial_thread)
        self.serial_thread.start()

        self.serial_worker.data_received.connect(self.
    read_serial_data)

        # Plot layout
        self.plot_widget = pg.PlotWidget(name="Serial Data")
        self.plot_widget.showGrid(x=True, y=True)
        self.plot_widget.setLabel('left', "Amplitude")
        self.plot_widget.setLabel('bottom', "Time")
        self.plot_widget.setMouseEnabled(x=False, y=False)
```

```
        self.plot_widget.setMenuEnabled(False)
        self.plot_widget.addLegend()
        self.region_item = pg.LinearRegionItem()
        self.region_item.setRegion([0, 0])
        self.region_item.setZValue(10)
        self.region_item.sigRegionChangeFinished.connect(self.
    region_updated)

        self.plot_widget.addItem(self.region_item)
        self.plot_layout.addWidget(self.plot_widget)
        self.plot_layout.setSpacing(2)

        # Serial Monitor Button
        self.serial_monitor = SerialMonitorWindow(self)
        self.launch_monitor_button = QPushButton("Launch Serial
    Monitor")
        self.launch_monitor_button.clicked.connect(self.
    launch_serial_monitor)
        self.serial_data_received.connect(self.serial_monitor.
    read_serial_data)

        self.plot_layout.addWidget(self.launch_monitor_button)

        main_layout.addLayout(self.plot_layout, 1, 0, 1, 3)
        main_layout.setColumnStretch(0, 3)

        # Header for the labels table
        self.input_enable_header = QLabel("Input Selection")
        self.input_enable_header.setAlignment(Qt.AlignmentFlag.
    AlignCenter)
        self.sidebar_layout.addWidget(self.input_enable_header)

        # Add space between the header and the checkboxes
        self.sidebar_layout.addSpacing(5)

        self.input_checkboxes_layout = QGridLayout()
        num_checkboxes = 12
        num_columns = 2
        self.checkboxes = []
        for i in range(num_checkboxes):
            checkbox = QCheckBox(f"Input {i + 1}")
            checkbox.stateChanged.connect(self.
    checkbox_state_changed)
            checkbox.setEnabled(False)
            row = i // num_columns
            column = i % num_columns
            self.input_checkboxes_layout.addWidget(checkbox, row,
    column)
            self.checkboxes.append(checkbox)

        self.sidebar_layout.addLayout(self.input_checkboxes_layout)

        self.sidebar_layout.addSpacing(10)

```

```
        self.labels_table = QTableWidget()
        self.labels_table.setColumnCount(2)
        self.labels_table.setHorizontalHeaderLabels(["Data Label", "Region"])
        self.labels_table.horizontalHeader().setSectionResizeMode(0, QHeaderView.ResizeMode.Stretch)

        self.sidebar_layout.addWidget(self.labels_table)

        self.label_input = QLineEdit()
        self.label_input.setPlaceholderText("Enter label name...")
        self.sidebar_layout.addWidget(self.label_input)

        region_controls_layout = QHBoxLayout()

        self.add_label_button = QPushButton("Add Label")
        region_controls_layout.addWidget(self.add_label_button)

        self.remove_label_button = QPushButton("Remove Label")
        region_controls_layout.addWidget(self.remove_label_button)

        self.sidebar_layout.addLayout(region_controls_layout)

        self.export_labels_button = QPushButton("Export Labels")
        self.sidebar_layout.addWidget(self.export_labels_button)

        # Connect UI elements with their respective functionality
        self.labels_table.cellPressed.connect(self.table_cell_clicked)
        self.labels_table.cellChanged.connect(self.table_cell_changed)
        self.label_input.returnPressed.connect(self.add_label)
        self.add_label_button.clicked.connect(self.add_label)
        self.remove_label_button.clicked.connect(self.remove_label)
        self.export_labels_button.clicked.connect(self.export_labels)

        main_layout.addLayout(self.sidebar_layout, 1, 3, 1, 1)

        self.timer = QTimer()
        self.timer.timeout.connect(self.update_plot)
        self.timer.start(50)

    def detect_serial_devices(self):
        """
        Detects all available serial ports and populates the serial port dropdown with them.
        """
        available_ports = QSerialPortInfo.availablePorts()
        self.serial_port_dropdown.clear()
        for port in available_ports:
            self.serial_port_dropdown.addItem(port.portName())

    def update_serial_ports(self):
```

```
214         """
215         Updates the available serial ports in the dropdown menu.
216         """
217         self.serial_port_dropdown.clear()
218         ports = QSerialPortInfo.availablePorts()
219         for port in ports:
220             self.serial_port_dropdown.addItem(port.portName())
221 
222     def update_baud_rate(self):
223         """
224         Updates the baud rate of the connected serial port to the
    value selected in the baud rate dropdown menu
225         """
226         if self.serial.isOpen():
227             self.serial.setBaudRate(int(self.baud_rate_dropdown.
    currentText()))
228             self.status_bar.show_message("Baud rate changed to: " +
    self.baud_rate_dropdown.currentText())
229 
230     def connect_serial(self):
231         """
232         Attempts to connect to the selected serial port with the
    selected baud rate
233         """
234         port_name = self.serial_port_dropdown.currentText()
235         baud_rate = int(self.baud_rate_dropdown.currentText())
236 
237         self.serial.setPortName(port_name)
238         self.serial.setBaudRate(baud_rate)
239 
240         if self.serial.open(QIODevice.OpenModeFlag.ReadWrite):
241             self.status_bar.show_message("Connected to serial port:
    " + port_name)
242             self.serial_connected = True
243             # Disable the connect button
244             self.connect_button.setEnabled(False)
245             # Enable the disconnect button
246             self.disconnect_button.setEnabled(True)
247             # Enable the capture button
248             self.capture_button.setEnabled(True)
249         else:
250             self.status_bar.show_error("Failed to connect to serial
    port: " + port_name, 5000)
251 
252     def disconnect_serial(self):
253         """
254         Disconnects from the serial port and updates UI elements.
255         """
256 
257         # Stop the capture
258         if self.capture_enabled:
259             self.toggle_capture()
260 
261         if self.serial.isOpen():
```

```
262             self.serial.close()
263             self.status_bar.show_message("Disconnected from serial
    port")
264             self.serial_connected = False
265             # Enable the connect button
266             self.connect_button.setEnabled(True)
267             # Disable the disconnect button
268             self.disconnect_button.setEnabled(False)
269             # Disable the capture button
270             self.capture_button.setEnabled(False)
271
272     def launch_serial_monitor(self):
273         self.serial_monitor.show()
274         self.serial_monitor.raise_()
275         self.serial_monitor.activateWindow()
276
277     def toggle_capture(self):
278         """
279         Toggles the capture state and updates UI elements.
280         """
281
282         if not self.capture_enabled:
283             self.capture_button.setText("Stop Capture")
284             self.status_bar.show_message("Capture started")
285         else:
286             self.capture_button.setText("Start Capture")
287             self.status_bar.show_message("Capture stopped")
288         self.capture_enabled = not self.capture_enabled
289
290     def constrain_region_to_plot_bounds(self):
291         """
292         Constrain the region of interest to the bounds of the plot.
293         """
294
295         x_range = self.plot_widget.getAxis('bottom').range
296         min_x, max_x = min(x_range), max(x_range)
297         self.region = (max(min_x, self.region[0]), min(max_x, self.
    region[1]))
298         self.region_item.setRegion(self.region)
299
300     def region_updated(self):
301         """
302         Updates the region of interest.
303         """
304
305         self.region_item.setZValue(10)
306         x1, x2 = self.region_item.getRegion()
307         self.region = (int(x1), int(x2))
308
309     def table_cell_changed(self, row, column):
310         """
311         Updates the label and region information of a cell in the
    labels table when it is changed by the user.
312
```

```
313         Args:
314             row (int): The row of the cell that was changed.
315             column (int): The column of the cell that was changed.
316         """
317 
318         if row >= len(self.labels):
319             return
320 
321         if column == 0:
322             label = self.labels_table.item(row, 0).text()
323             if not label or label.isspace() or label == "":
324                 self.status_bar.show_error("Label cannot be empty",
    3000)
325                 self.labels_table.item(row, 0).setText(self.labels[
    row][0])
326             else:
327                 self.labels[row] = (label, self.labels[row][1])
328         elif column == 1:
329             region_str = self.labels_table.item(row, 1).text()
330             region = self.string_tuple_to_tuple(region_str)
331             if region is not None:
332                 self.labels[row] = (self.labels[row][0], region)
333             else:
334                 self.labels_table.item(row, 1).setText(str(self.
    labels[row][1]))
335 
336     def string_tuple_to_tuple(self, string_tuple):
337         """
338         Converts a string representation of a tuple to a tuple
    object.
339 
340         Args:
341             string_tuple (str): The string representation of a tuple
    .
342 
343         Returns:
344             tuple: The tuple object.
345         """
346 
347         try:
348             str_list = string_tuple.strip("()").split(",")
349             int_list = [int(x) for x in str_list]
350 
351             if len(int_list) != 2:
352                 self.status_bar.show_error("Invalid region input
    format")
353                 return None
354 
355             tuple_obj = tuple(int_list)
356             return tuple_obj
357 
358         except ValueError:
359             self.status_bar.show_error("Invalid region input format.
     Expected: (x1, x2)")
```

```
            return None

    def table_cell_clicked(self, row, column):
        """
        Updates the selected label's region when its cell is clicked.

        Args:
            row (int): The row of the cell that was clicked.
            column (int): The column of the cell that was clicked.
        """

        self.selected_row = row
        selected_region = self.labels_table.item(row, 1).text()

        if self.string_tuple_to_tuple(selected_region) is not None:
            self.region = self.string_tuple_to_tuple(selected_region)
            self.region_item.setRegion(self.region)
        else:
            self.status_bar.show_error("Invalid region input format. Expected: (x1, x2)")

    def add_label(self):
        """
        Adds a new label and region to the labels table and labels list.
        """

        label = self.label_input.text()
        # validate the label
        if not label or label.isspace() or label == "":
            self.status_bar.show_error("Label cannot be empty", 3000)
            return

        row = self.labels_table.rowCount()

        self.labels_table.insertRow(row)
        self.labels_table.setItem(row, 0, QTableWidgetItem(label))
        self.labels_table.setItem(row, 1, QTableWidgetItem(str(self.region)))

        # Clear the label input
        self.label_input.setText("")

        # Add to the labels list
        self.labels.append((label, self.region))

    def remove_label(self):
        """
        Removes the selected label and region from the labels table and labels list.
        """
```

```
407
408         if self.selected_row >= 0:
409             self.labels_table.removeRow(self.selected_row)
410             self.labels.pop(self.selected_row)
411             self.selected_row = -1
412         else:
413             self.status_bar.show_error("No label is selected", 3000)
414
415     def import_csv(self):
416         """
417         Allows the user to import a CSV file to plot data
418         """
419
420         file_name, _ = QFileDialog.getOpenFileName(self, "Import CSV
    ", "", "CSV Files (*.csv)")
421         if file_name:
422             with open(file_name, 'r') as file:
423                 reader = csv.reader(file)
424                 plot_data = [list(map(float, row)) for row in reader
    ]
425                 self.plot_data = np.array(plot_data)
426
427     def export_data(self):
428         """
429         Allows the user to export the plot data to a CSV file
430         """
431
432         if self.plot_data is None:
433             self.status_bar.show_error("No data to export", 3000)
434             return
435
436         file_name, _ = QFileDialog.getSaveFileName(self, "Export
    Data", "", "CSV Files (*.csv)")
437
438         if file_name:
439             self.save_to_file(file_name)
440
441     def export_labels(self):
442         """
443         Allows the user to export the labels to a CSV file
444         """
445
446         if self.plot_data is None:
447             self.status_bar.show_error("No data to export", 3000)
448             return
449
450         if len(self.labels) == 0:
451             self.status_bar.show_error("No labels to export", 3000)
452             return
453
454         file_name, _ = QFileDialog.getSaveFileName(self, "Export
    Labels", "", "CSV Files (*.csv)")
455
456         if not file_name:
```

```
457             return
458
459         data = []
460
461         for i in range(len(self.labels)):
462             label_range = self.labels[i][1]
463             label_string = self.labels[i][0]
464
465             if label_range[0] >= len(self.plot_data):
466                 self.status_bar.show_error("Label range is out of
    bounds", 3000)
467                 return
468
469             if label_range[1] >= len(self.plot_data):
470                 self.status_bar.show_error("Label range is out of
    bounds", 3000)
471                 return
472
473             # Get the data for the current label range and add the
    label string as a new column
474             label_data = self.plot_data[label_range[0]:label_range
    [1] + 1]
475             label_column = np.full((label_data.shape[0], 1),
    label_string, dtype=object)
476             label_data_with_label = np.hstack((label_data,
    label_column))
477
478             data.append(label_data_with_label)
479
480         # Combine all label data into a single array
481         data = np.vstack(data)
482
483         self.save_to_file(file_name, data)
484
485     def save_to_file(self, file_name, data=None):
486         """
487         Save plot data or specified data to a CSV file.
488
489         Args:
490             file_name (str): The name of the file to save.
491             data (Optional[List[List[float]]]): The data to save. If
     None, self.plot_data will be saved.
492
493         Returns:
494             None.
495         """
496
497         with open(file_name, 'w', newline="") as file:
498             writer = csv.writer(file)
499             if data is not None:
500                 writer.writerows(data)
501             else:
502                 writer.writerows(self.plot_data)
503
```

```
504         self.status_bar.show_message("Saved to file: " + file_name,
    3000)
505
506     def read_serial_data(self, data):
507         """
508         Process serial data received from the connected device.
509
510         Args:
511             data (str): The data received from the connected device.
512
513         Returns:
514             None.
515         """
516
517         if data:
518             self.serial_data_received.emit(data)
519             try:
520                 values = [float(value) for value in data.split(',')]
521                 if len(values) != self.num_detected_signals:
522                     self.num_detected_signals = len(values)
523                     self.update_inputs_checkboxes()
524                 # self.serial_data_received.emit(data)
525                 if self.capture_enabled:
526                     if self.plot_data is None:
527                         self.plot_data = np.array([values])
528                     else:
529                         self.plot_data = np.append(self.plot_data,
    np.array([values]), axis=0)
530
531             except Exception as e:
532                 print("read_serial_data(): " + str(e))
533
534     def update_inputs_checkboxes(self):
535         """
536         Update the input checkboxes based on the number of signals
    detected.
537
538         Returns:
539             None.
540         """
541
542         for i, checkbox in enumerate(self.checkboxes):
543             checkbox.setEnabled(i < self.num_detected_signals)
544             checkbox.setChecked(i < self.num_detected_signals)
545
546     def checkbox_state_changed(self, state):
547         """
548         Callback function when a checkbox state is changed.
549         Enables/disables the corresponding input and redraws the
    plot.
550
551         :param state: The new state of the checkbox.
552         :type state: int
553         """
```

```
554
555         sender = self.sender()
556         if state == Qt.CheckState.Checked.value:
557             msg = f"{sender.text()} is enabled."
558             self.status_bar.showMessage(msg)
559         else:
560             msg = f"{sender.text()} is disabled."
561             self.status_bar.showMessage(msg)
562
563         for plot_line in self.plot_lines:
564             plot_line.clear()
565
566         if self.capture_enabled:
567             self.draw_plot()
568
569     def clear_plot_data(self):
570         """
571         Clears the plot data and updates the plot.
572         """
573
574         self.plot_data = None
575         for plot_line in self.plot_lines:
576             plot_line.setData([0], [0])
577
578
579     def update_plot(self):
580         """
581         Updates the plot with the new data if capture is enabled.
582         """
583
584         if self.capture_enabled:
585             self.draw_plot()
586
587     def draw_plot(self):
588         """
589         Draws the plot with the filtered data from the checkboxes.
590         """
591
592         if self.plot_data is not None:
593             # Get the indices of the checked checkboxes
594             checked_indices = [i for i, checkbox in enumerate(self.
    checkboxes[0:self.num_detected_signals]) if
595                                checkbox.isChecked()]
596
597             # Filter the data based on the checked checkboxes
598             filtered_data = self.plot_data[:, checked_indices]
599
600             for i, data_line in enumerate(filtered_data.T):
601                 color = self.get_unique_color(i)
602                 label = f"Input {i + 1}"
603                 if i < len(self.plot_lines):
604                     # self.plot_widget.disableAutoRange()
605                     self.plot_lines[i].setData(data_line)
606                     # print("data_line: " + str(data_line))
```

```
                    # self.plot_widget.enableAutoRange()
                else:
                    plot_line = self.plot_widget.plot(data_line, pen=pg.mkPen(color, width=1), name=label)
                    self.plot_lines.append(plot_line)

    def get_unique_color(self, index):
        """
        Returns a unique color based on the index.

        :param index: The index to generate the color for.
        :type index: int
        :return: The unique color.
        :rtype: str
        """

        colors = [
            "#1f77b4",
            "#ff7f0e",
            "#2ca02c",
            "#d62728",
            "#9467bd",
            "#8c564b",
            "#e377c2",
            "#7f7f7f",
            "#bcbd22",
            "#17becf"
        ]
        return colors[index % len(colors)]

    def closeEvent(self, event):
        self.serial_thread.quit()
        self.serial_thread.wait()
        super().closeEvent(event)
        app.quit()   # Quit the application when the main window is closed


class SerialWorker(QObject):
    """Worker object that reads data from a serial port and emits it as a signal.

    Attributes:
        data_received (pyqtSignal): A signal that is emitted when data is received from the serial port.

    Args:
        serial (QSerialPort): The QSerialPort object to read data from.
    """

    data_received = pyqtSignal(str)

    def __init__(self, serial):
```

```
656         """Initializes a SerialWorker object.
657 
658         Args:
659             serial (QSerialPort): The QSerialPort object to read
    data from.
660         """
661         super().__init__()
662         self.serial = serial
663         self.serial.readyRead.connect(self.read_serial_data)
664 
665     @pyqtSlot()
666     def read_serial_data(self):
667         """Reads data from the serial port and emits it as a signal.
    """
668         try:
669             while self.serial.canReadLine():
670                 data = self.serial.readLine().data().decode(errors="
    ignore")
671                 self.data_received.emit(data)
672         except Exception as e:
673             print("read_serial_data(): ", str(e))
674 
675 
676 class SerialMonitorWindow(QMainWindow):
677     def __init__(self, parent_window):
678         super().__init__()
679 
680         # Save a reference to the serial port
681         self.serial = parent_window.serial
682 
683         # Set minimum size of the window
684         self.setMinimumSize(600, 340)
685 
686         # Create a main widget for the window
687         main_widget = QWidget()
688         self.setCentralWidget(main_widget)
689 
690         # Set the window title
691         self.setWindowTitle("Serial Monitor")
692 
693         # Create a vertical layout for the main widget
694         layout = QVBoxLayout()
695         main_widget.setLayout(layout)
696 
697         # Create a text edit widget for displaying serial data
698         self.serial_monitor_text = QTextEdit()
699         layout.addWidget(self.serial_monitor_text)
700 
701         # Create a horizontal layout for input controls
702         controls_layout = QHBoxLayout()
703 
704         # Create a line edit widget for entering serial data
705         self.serial_input = QLineEdit()
706         controls_layout.addWidget(self.serial_input)
```

```
707
708         # Create a combo box for selecting line endings
709         self.line_ending_dropdown = QComboBox()
710         self.line_ending_dropdown.addItems(["No Line Ending", "
    Newline", "Carriage Return", "Both NL & CR"])
711         controls_layout.addWidget(self.line_ending_dropdown)
712
713         # Add the controls layout to the main layout
714         layout.addLayout(controls_layout)
715
716         # Connect the returnPressed signal of the input line edit to
     the send_serial_data slot
717         self.serial_input.returnPressed.connect(self.
    send_serial_data)
718
719     def read_serial_data(self, data):
720         """
721         Slot for reading data from the serial port.
722
723         Parameters:
724             data (str): Data read from the serial port.
725         """
726         # Append the data to the text edit and scroll to the bottom
727         self.serial_monitor_text.insertPlainText(data)
728         self.serial_monitor_text.verticalScrollBar().setValue(self.
    serial_monitor_text.verticalScrollBar().maximum())
729
730     def send_serial_data(self):
731         """
732         Slot for sending data to the serial port.
733         """
734         # Get the data from the input line edit
735         data = self.serial_input.text()
736
737         # Get the selected line ending from the combo box
738         line_endings = ["", "\r", "\n", "\r\n"]
739         ending_index = self.line_ending_dropdown.currentIndex()
740         data += line_endings[ending_index]
741
742         # Write the data to the serial port and clear the input line
     edit
743         self.serial.write(data.encode())
744         self.serial_input.clear()
745
746
747 class CustomStatusBar(QStatusBar):
748     """
749     A custom status bar with methods for showing messages and errors
    .
750
751     Inherits from QStatusBar.
752     """
753
754     def __init__(self):
```

```
        """
        Constructor for the CustomStatusBar class.
        """
        super().__init__()

        # Keep track of the current message and background color
        self.current_message = None
        self.current_color = None

        # Create a timer for resetting the background color
        self.timer = QTimer()
        self.timer.timeout.connect(self.reset_background_color)

    def show_message(self, message, timeout=0):
        """
        Displays a message in the status bar with a background color of blue.

        Args:
            message (str): The message to display.
            timeout (int): The number of milliseconds to display the message. If zero, the message is displayed indefinitely.
        """
        self.current_message = message
        self.current_color = "#3f51b5"
        self.setStyleSheet(f"background-color: {self.current_color}; color: white;")
        self.showMessage(message, timeout)

        # Reset the background color after the timeout expires
        if timeout > 0:
            self.timer.start(timeout)

    def show_error(self, error, timeout=0):
        """
        Displays an error in the status bar with a background color of red.

        Args:
            error (str): The error message to display.
            timeout (int): The number of milliseconds to display the error message. If zero, the message is displayed indefinitely.
        """
        self.current_message = error
        self.current_color = "#f44336"
        self.setStyleSheet(f"background-color: {self.current_color}; color: white;")
        self.showMessage(error, timeout)

        # Reset the background color after the timeout expires
        if timeout > 0:
            self.timer.start(timeout)

    def reset_background_color(self):
```

```
803         """
804         Resets the background color of the status bar.
805         """
806         self.current_message = None
807         self.current_color = None
808         self.setStyleSheet("")
809         self.timer.stop()
810 
811     def clear_message(self):
812         """
813         Clears the current message and resets the background color
    of the status bar.
814         """
815         self.reset_background_color()
816         self.clearMessage()
817 
818 
819 if __name__ == "__main__":
820     qdarktheme.enable_hi_dpi()
821     app = QApplication(sys.argv)
822     qdarktheme.setup_theme("dark")
823     main_window = MainWindow()
824     main_window.show()
825     sys.exit(app.exec())
```

Listing V.5. "OptiGap Labeling GUI (main.py)"

CURRICULUM VITAE

Paul Bupe Jr

Education

Ph.D., University of Louisville, Louisville, KY, 2023

M.S., Georgia Southern University, Statesboro, GA, 2020

M.S., Georgia Southern University, Statesboro, GA, 2015

B.S., Georgia Southern University, Statesboro, GA, 2013

Publications

P. Bupe et al., "Embedded Optical Waveguide Sensors for Dynamic Behavior Monitoring in Twisted-Beam Structures" (Under Review, RoboSoft 2024)

P. Bupe and C. K. Harnett, "OptiGap: A Modular Optical Sensor System for Bend Localization," IEEE International Conference on Robotics and Automation (ICRA), London, United Kingdom, 2023, pp. 620-626. (NSF Award: 1935324)

P. Bupe, D. J. Jackson, and C. K. Harnett, "Electronically reconfigurable virtual joints by shape memory alloy-induced buckling of curved sheets," in SoutheastCon 2022, pp. 598-604, Mar. 2022. (NSF Award: 1935324)

P. Bupe, "Accurate Range-based Indoor Localization Using PSO-Kalman Filter Fusion" in Electronic Theses and Dissertations. 2048., 2020

P. Bupe, R. Haddad, and F. Rios-Gutierrez, "Relief and emergency communication network based on an autonomous decentralized UAV clustering network. In SoutheastCon 2015, pp. 1-8, April 2015