Human-Computer Interaction in Engineering Simulation

Eleanor Clifford, Trinity College
Supervised by Professor Graham Pullan

Technical Abstract

In areas where highly skilled workers regularly deal with complex systems, efficient interaction with custom software is paramount, and usually involves custom hardware. Here we define ``efficiency'' as the relative speed at which tasks can be completed without a degradation in other desirable properties such as user comfort or error rate. Custom tactile interfaces are commonplace in areas such as sound engineering and aircraft piloting, but in engineering simulation we still rely on mouse or touchscreen interfaces that were designed for first-use intuitiveness rather than long-term efficiency.

This project bridges that gap by designing and testing an efficient interaction device for a set of subtasks of engineering simulation. Existing devices are suboptimal for these subtasks. This device is named SLIDER: the Software-Labelled Interaction Device for Engineering Research.

First, a new theory of Human-Computer Interaction (HCI) is developed to inform the choice of both interaction paradigm and interaction device. This is rooted in existing research but defines a new ``task space'', a geometric space with axes of task properties, such as task complexity, frequency, dimensionality, and so on. Subtasks of engineering simulation can be approximated as points in this task space, and interaction paradigms and devices that are optimal for nearby tasks inform the choice of paradigm and device for engineering simulation subtasks. Sound engineering is adjacent in this task space to engineering simulation, and thus, taking lessons from sound engineering, custom tactile interfacing is chosen as an interaction paradigm for the design of SLIDER.

Additional salient parameters of the task space are identified with reference to the dimensionality of common engineering simulation subtasks and the properties of those subtasks. It is determined that many subtasks of engineering simulation are ``2D joint asymmetric'', but these tasks are not optimal to control with existing devices. A 2D joint asymmetric task is defined as a task with two control variables which are connected but differ markedly in their meaning. An example is control of the width and position of a histogram window; these two parameters are asymmetric. This is in contrast, for example, to the task of controlling a position in 2D space, which would be a 2D joint symmetric task.

The SLIDER device is designed to be optimal for 2D joint asymmetric tasks and independent combinations of these tasks. Two specific 1D devices were chosen for maximal asymmetry while allowing for physical connection: a bounded motorised linear slider and an unbounded rotary encoder. The rotary encoder was attached on top of the linear slider such that it moves with the slider; the user can control both simultaneously with a single hand. A button was further included and physically connected in order to cycle the device through controlling independent subtasks. For software-controlled labelling of the device, an 8x8 dot-matrix display was chosen, as it is inexpensive and can sufficient to display a single character denoting a variable.

Specification is provided for five control modes designed for the motorised slider. These are: Passive, where it acts as an unmotorised slider; Go To, where it moves to a specified position

and then returns to Passive; Rumble, where it rumbles in place; Notch, where it acts as if there is a virtual ``notch'' in a specified position in the track (a location it wants to enter and remain in); and finally Bump, which is the inverse of Notch. Specification is also provided for an interface for connecting SLIDER to a computer. This uses a bidirectional Musical Instrument Digital Interface (MIDI) protocol, a common protocol for which there is a wide range of software support. A standard for MIDI communication with SLIDER is presented.

The design and implementation of all of the subsystems of SLIDER is detailed. This is split into electronic design, mechanical design, and firmware design. Electronically, SLIDER consists of a microcontroller, motor controller, battery, and the aforementioned human interface devices. Mechanically, the electronics are contained within a 3D-printed ABS box and an ABS bracket is used to mount the rotary encoder directly on top of the slider. These are designed to be replicable on an ordinary 3D printer. Firmware was developed for SLIDER to run on the microcontroller, using a protothreaded architecture to handle bidirectional MIDI messages while using PID control to handle the motorised controller's control modes.

Web software demos for SLIDER are introduced and explained in sufficient detail that the reader can write their own software for SLIDER. There are three demos. First, the simplest possible demo, where the slider and rotary encoder each move a HTML slider, and the button toggles a HTML checkbox. Then, two more complex demos which interact with an existing web-based engineering simulation visualisation framework in active use.

In the second demo, SLIDER is used to control a pair of histogram limit windows, with the slider controlling the window's position, and the rotary encoder controlling the window's width. When the button is pressed, the software reassigns the inputs it receives from SLIDER to control the alternate histogram. This demonstrates the ``Go To'' mode that was developed in firmware for the motorised slider: when the button is pressed, the motorised slider physically moves to the position where the new histogram was is centered.

In the third demo, SLIDER is used to control a spatial slice through a snapshot of an unsteady flow over a wing. This demonstrates the ``Notch'' mode developed for the motorised slider: a virtual ``notch'' is set up at the location of the trailing edge of the wing, so that the slider naturally falls into the notch location when it is brought nearby. This demo demonstrates the usefulness of the Notch mode to mark salient points the variable being controlled, and make them easier to reach.

A survey was conducted with five researchers from the Whittle lab, each of whom used SLIDER to control two demos, compared it to existing interfaces, and answered questions via an online form. Every responder remarked that they preferred SLIDER to existing interfaces, and almost all could see themselves using a device based on this prototype in their daily research.

To conclude, this project developed a new theory of HCI for engineering simulation, identified a type of control task which existing systems are suboptimal for, leveraged this research to design and build a new interaction device for use in engineering simulation: ``SLIDER'', and conducted a survey showing that real engineers preferred SLIDER over existing interaction devices for the given demos, and could see themselves using such a device in their daily work. This work has been consolidated into a webpage introducing the project: https://slider.cliffor d.lol, which includes a full build guide, introduction to writing software for SLIDER, and links to all source code and assets, released under free and open source licenses.

The findings of this project suggest a very promising future for custom interaction devices in engineering simulation.

Contents

1 Introduction

1.1 Context

Designing interfaces for use in engineering simulation and related technical fields is a distinctly different task than designing interfaces for the average user. There are two principal reasons for this. First, the design objectives are very different. Grudin [2005] distinguishes ``discretionary'' use from ``non-discretionary'' use, where the former is when a user uses a system because they want to, and the latter is where a user uses a system because they have to, for example because it is required for their job. As Grudin [2005] puts it, for non-discretionary use ``Research goals included reducing training time, but most important was eliminating errors and increasing the pace of skilled performance.'' This is exactly the goal of a designer for high-frequency engineering simulation tasks, and very different from the goals of a designer for discretionary use. Nevertheless the majority of recent Human-Computer Interaction (HCI) research has been into discretionary use. A complementary understanding of the difference between discretionary and non-discretionary tasks will be discussed in Section 1.3.

The second, and related difference between engineering simulation tasks and more common tasks is in the parameters of the tasks. This is discussed in detail in Section 1.3, but as a motivating example, consider that for a task that is performed at high frequency for a long period of time, learning time is relatively unimportant to the overall efficiency, but conversely for a task that is only completed a few times, learning time becomes paramount. One would expect, then, that the interaction systems which optimise overall efficiency should be very different for the two tasks.

1.2 Related Work

This project departs with intention from recent literature, and introduces a novel theoretical underpinning on which the choice of both broad interface paradigm and specific devices is based. Nonetheless, there is significant prior research which supports and informs the choices made in this work. Grudin [2005] discusses how HCI research has fallen into a number of separate research areas with different theories and methods. This is given in a historical context, but this disparity will be reinterpreted in terms of the new theoretical description introduced in Section 1.3.

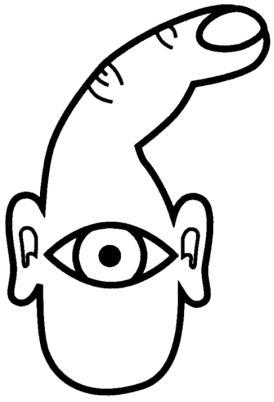


Figure 1: "The GUI's mental model of a user", originally by O'Sullivan and Igoe [2004], later reproduced by Klemmer et al. [2006]. This illustrates that existing interfaces such as touchscreen-based GUIs (Graphical User Interfaces) only leverage a limited subset of our human capabilities.

Draper [1993] discusses the instability of the notion of any given ``task'' in HCI, in particular that it varies between types of users and individuals. This informs an important complication in the task space description. Chalmers and Galani [2004] discuss heterogeneity in HCI design, and how people naturally take advantage of the so-called ``seams'' between different media. This is leveraged in Section 2.3.

Olivera et al. [2011] explore interfaces which take advantage of physicality in addition to the usual visual and auditory markers, to leverage the brain's capacity for parallel processing of interaction channels. They focus on fiducial markers, a different direction to our research.

Klemmer et al. [2006] criticise the homogenisation of modern computer systems, pointing out that common interface devices leverage our capabilities well for some tasks, but not for many others. Figure 1 is used to emphasise this point effectively. Likewise, Buxton [1986] suggests that the optimal device for a particular task depends on the specific details of the task.

So far there has been little research directed at designing specialist hardware interfaces for non-discretionary research tasks such as those in engineering simulation. Many of the devices which are commonly used for engineering simulation, such as mice, were designed for the common discretionary tasks of the general public. There are some interesting devices designed for high-dimenensional joint tasks such as Perelman et al. [2015]'s ``roly-poly mouse'' for 3D translation and rotation, but there is minimal research into efficient operation of lower dimensional tasks such as those that will be outlined in Section 2.2.

1.3 A new model of the HCI ``task space''

Human-Computer Interaction is the pursuit of optimal interaction between humans and computers. But what defines an optimum? The most important properties of an interaction paradigm depend on the properties of the task which the user is performing. For instance, when a user interacts with a website for the first time, the optimum interface is one which is clear, simple, and with complexities hidden in menus; it must be ``intuitive'', at the cost of the efficiency of people familiar with the website. This is in contrast to the optimum interface in an aeroplane cockpit, or a sound mixing desk, where the user is already familiar with the interface, and instead the focus is on other properties, such as efficiency, safety, or error rate.

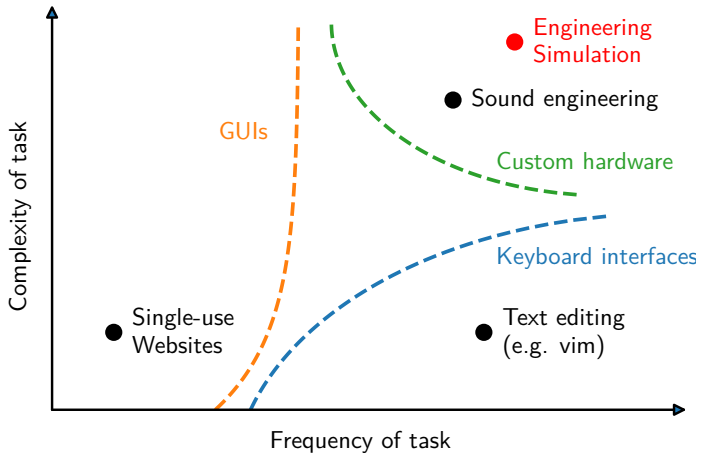


Figure 2: A two-dimensional slice through the task space, with example tasks plotted in black, our task in red, and approximate optimal interaction paradigms for efficiency suggested in dotted lines.

In order to better explain and reason about choices of interface and demonstrate the need for a new interaction paradigm in engineering simulation, a novel description of the HCI interface-choice problem was developed. The salient properties of the task are isolated, and the task is plotted on the ``task space'', a high-dimensional geometric space with task properties as axes. An example of a 2D slice through this space can be seen in Figure 2.

The true task space has many more than two dimensions -- other useful ones might be ``acceptable action latency'' and ``recoverability from error''. Thus, the examples given in Figure 2 do not necessarily all exist on a single plane, they are projections.

Tasks are not actually points either, but distributions in the space. Here, we say that tasks are ``distributions'' in the space in two senses. In the first sense, the exact location of particular subtasks of the parent task will vary in their position, but some positions will be more common than others. One could imagine histogramming the subtasks, weighted by their frequency, to produce the distribution of the parent task. This distribution could be bimodal if there are distinct categories of subtasks. In the second sense, this distribution also encodes the variability in task specification and interpretation (the ``notion'' of a task) as discussed by Draper [1993].

Further, the axes we find easy to reason about are not necessarily orthogonal to each other or easy to assign numerical values to, and the exact position of each task in the space is particular to a single user and may change over time. Nevertheless, the task space is a highly useful tool to reason qualitatively about similarities and differences between tasks, and thus decide on an interface paradigm to use. Engineering simulation includes a highly complex set of tasks, which many engineers work on full-time. Thus, it sits in the top right of Figure 2.

Grudin [2005]'s description of HCI would segregate engineering simulation from website usage by saying that engineering simulation is non-discretionary and most website usage is discretionary. The task space description illustrated in Figure 2 provides a complementary and more fundamental way to segregate them: engineering simulation is complex and high-frequency, and website usage (of specific websites) is the opposite. In fact, the task space can absorb Grudin [2005]'s description by simply considering discretionality as another axis of the task space. In many cases discretionality is an unimportant axis for interaction paradigm choice: for example, the optimal interaction paradigm for a low-frequency low-complexity non-discretionary task such as using a conference website to submit a research paper is nearly identical to an otherwise similar discretionary task such as using a shopping website.

So far we have used the task space to suggest a broad interaction paradigm, but it also be used to suggest specific interaction devices for subtasks. A number of axes of the task space that are salient for subtask device choice in engineering simulation will be discussed in Section 2.3. These will include the dimensionality of the subtask and the properties of those dimensions.

1.4 Project Aims

The aim of this project is to produce a prototype interaction device to be used in engineering simulation work, along with software demonstrations which closely align with real-world use cases. The success of the project can be evaluated based on the results of the comparative study performed in Section 7, in which professional engineers tried our prototype SLIDER device side by side with an equivalent existing interaction system. The metrics used in this study are necessarily qualitative, as effective quantitative research into the efficiency of a new interaction system would first require a number of months for users to become proficient with the new interaction paradigm. Nevertheless, in the study, every participant responded that they preferred SLIDER to existing devices, and almost all hypothesised that it would improve efficiency in their own work in the long term.

The SLIDER device is not designed to replace all tasks in engineering simulation, as some are well suited for existing devices. Rather, it is designed to be complementary, and fill a gap for

which existing devices are not optimal.

1.5 Contributions

This project makes the following specific contributions to the field of HCI and to developing workflow for researchers in engineering simulation:

- Development of a novel theoretical understanding of HCI, motivating heterogeneity in interface paradigm choice.
- Development of a prototype interaction device for use in engineering simulation tasks, including electronic design, mechanical design, and firmware development. This device is named SLIDER: the Software-Labelled Interaction Device for Engineering Research, and can be seen in Figure 3.
- Development of web demos for use with SLIDER, which are extensible for use in new projects, including those that use Pullan [2017]'s dbslice.
- Positive survey research showing that SLIDER is preferred to existing interaction systems for the given demos, suggesting that a device based on this prototype could improve the efficiency of working engineers.
- A website for the project: https://slider.clifford.lol. This includes an introduction to the project, a build guide, demos, specifications, and repositories for all source code and assets. All materials related to the project are contained here and released under free and open source licenses.

1.6 Structure of this Report

This project is primarily a design project, motivated by the discussion in this section. Thus, Sections 2 to 6 are devoted to the design and implementation of SLIDER. Firstly, Section 2 details the approach taken and the system design of SLIDER, i.e. the design of SLIDER from an external perspective: which specific interaction devices were chosen, how SLIDER interacts with a computer, and so on. This culminates in a full design specification for SLIDER as seen in Section 2.7. Then, Sections 3 to 6 detail the design of each subsystem of SLIDER from the ground up: electrical, mechanical, firmware, and finally the software of the demos. Section 7 presents and discusses the results of the survey, and lastly, Section 8 concludes the report and details how SLIDER can be integrated into new projects.

2 Approach and System Design

This section details the approach taken to design the SLIDER device, and the design decisions and rationale for the overarching system design of SLIDER, i.e. the design of all external-facing elements of SLIDER, including the choice of component control devices, display devices, control modes, and the interface with the computer. This culminates in Section 2.7 in a full external-facing specification for SLIDER.

2.1 Choice of interaction paradigm

From Figure 2 it is easily identified that engineering simulation is similar to sound engineering. A further dimension in the task space in which they are similar is the requirement of continuous control, i.e., there are parameters that are chosen in a continuous fashion, such as the level of an audio input, or a position along a wing to examine. This is distinct from the discrete nature of other tasks such as text editing.

Naturally, there are some dimensions in which engineering simulation and sound engineering differ. For example, in both acceptable action latency and acceptable easily-noticed error rate, engineering simulation is much higher than sound engineering. Nevertheless, the two tasks are very similar overall in their parameters.

Sound engineering, along with other tasks in the same region of the task space such as aeroplane piloting, has been shown to benefit from custom, tactile interfaces [Wanderley and Orio, 2002]. As such, this was investigated as a future interaction paradigm for engineering simulation.

2.2 Categorisation of common subtasks

Within the interaction paradigm of custom tactile interfaces, there are a number of options. The choice should be based on the specific requirements of simulation engineering tasks. The first requirement that should be noted is that variables controlled in simulation engineering tasks are very often continuous (e.g. setting the viscosity in a simulation) rather than discrete (e.g. typing characters in a text editor).

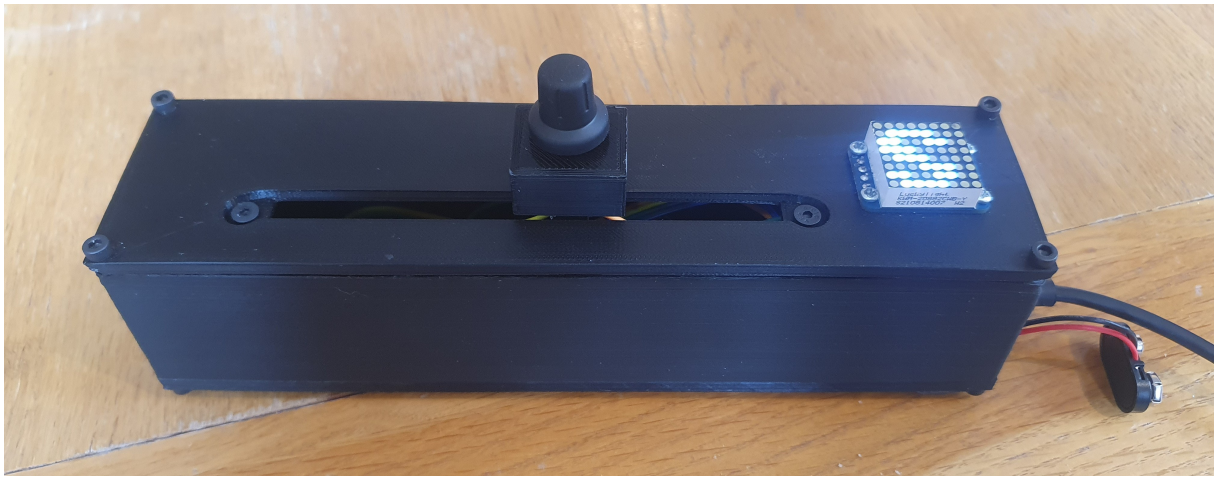
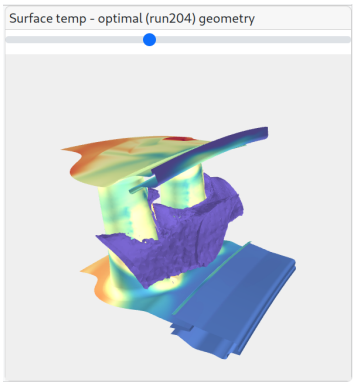
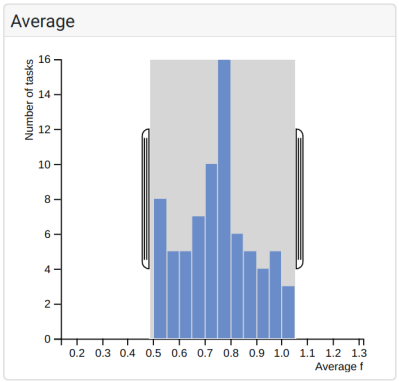


Figure 3: SLIDER: the Software-Labelled Interaction Device for Engineering Research



(a) An example of a 1D bounded control task. The slider shown is a single dimension (time), bounded to the limits of the data available.



(b) An example of a 2D joint asymmetric control task. There are two joint but asymmetric variables: width and position of the histogram window.

Figure 4: Examples of 1D and 2D control tasks from Pullan [2017]'s dbslice

It is useful to further break down tasks involving continuous variables by the dimensionality of the control required and the properties of those dimensions. These properties are in fact discrete-valued axes of Section 1.3's task space (e.g. the axis of dimensionality has discrete values 1D, 2D, 3D, etc.) These categories are as follows:

1. **1D control**. Examples are slices through data or control of a single variable such as the entry mach number of a simulation.
   a. **Bounded control**. This occurs when data is only available in a specific range, or the variable is fundamentally bounded, such as the mixing proportion of two gases.
   b. **Unbounded control**. The opposite of bounded control, for example when new data is generated on the fly based on a value with no fundamental bounds.
2. **2D control**. In each dimension, the 1D categories of bounded and unbounded apply, but there are further categories that depend on the relations between the dimensions.
   a. **Independent control**. This is simply two independent instances of 1D control.
   b. **Joint symmetric control**. This is 2D control where the variables are related and symmetric. A good example is control of a point in 2D space.
   c. **Joint asymmetric control**. This occurs when the variables are related, but not symmetric. A good example is control of the position and width of a 1D window.
3. **3D control**. This can be an independent combination of lower dimensional control tasks, or again joint symmetric or joint asymmetric. There are two interesting specific joint symmetric tasks:
   a. **Spatial control**
   b. **Rotational control**
4. **Higher dimensional control**. This is typically independent control tasks or independent combinations of joint lower dimensional control tasks. For example, rigid body motion is 6D, but this can be thought of as two independent 3D joint symmetric control tasks (rotation and translation).

Taking the demos of Pullan [2017]'s dbslice as examples of simulation engineering work, it is clear that many tasks are independent combinations of 2D joint asymmetric or 1D tasks. Some examples can be seen in Figure 4. Other tasks may be joint symmetric, but most 2D joint symmetric tasks are the poster-child for where a mouse or touchscreen is effective, as both are fundamentally 2D joint symmetric control devices. Some tasks can be higher dimensional,

typically rotation or translation, but a mouse can also be effective, though not optimal, for these tasks (using the scroll wheel for one axis). There are some 3D tasks for which a mouse is less well suited, such as 3D rotational control, but this has been a subject of prior research, such as by Perelman et al. [2015].

The biggest missing link in existing control devices for use in engineering simulation is therefore independent combinations of 1D and 2D joint asymmetric control tasks. SLIDER is a device designed to fill this interface gap.

2.3 Choice of control devices

A 2D device can be constructed by combining two 1D devices. The following devices are commonplace and easily added to SLIDER:

- Linear potentiometers (sliders)
- Rotary potentiometers (bounded knobs)
- Rotary encoders (unbounded knobs)

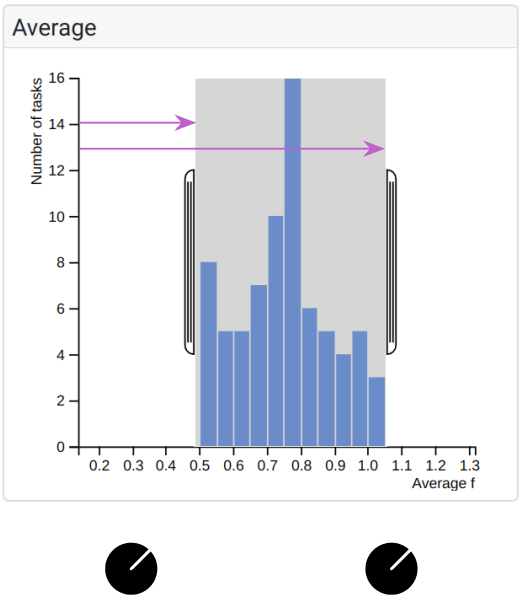
It is intuitive, although not necessarily trivial, to see that the two 1D devices to be used for a 2D joint asymmetric task should be both physically connected and asymmetric. We can examine this by considering a typical 2D joint asymmetric task: control of the width and position of a 1D histogram limits window. The use of symmetric or asymmetric devices is illustrated in Figure 5 for this task as well as a very similar symmetric task: control of the left and right edges of a histogram window.

In order to motivate the argument for using asymmetric devices, we first consider using two symmetric devices, for example two rotary knobs, as in Figure 5c. It would certainly be possible to control the histogram with these devices, but since there is little semantic separation between the two devices, it is not clear to the user which device is for the width, and which is for the position. More importantly, it is very easy to forget which controller is which when coming back to the device later. Contrast this to using a rotary device and a linear device, as in Figure 5d. The semantic separation of width and position is reflected in the asymmetry of the devices, and so it is much easier and faster for the user to commit to procedural memory the distinction between the two. Here ``procedural memory'' is the memory of routine procedures which the brain doesn't need to think actively about, such as walking. As Johnson [2003] puts it, when a task is committed to procedural memory it allows for a ``more efficient, autonomous mode of processing'' in the brain, as opposed to a ``deliberative, declarative mode of processing''.

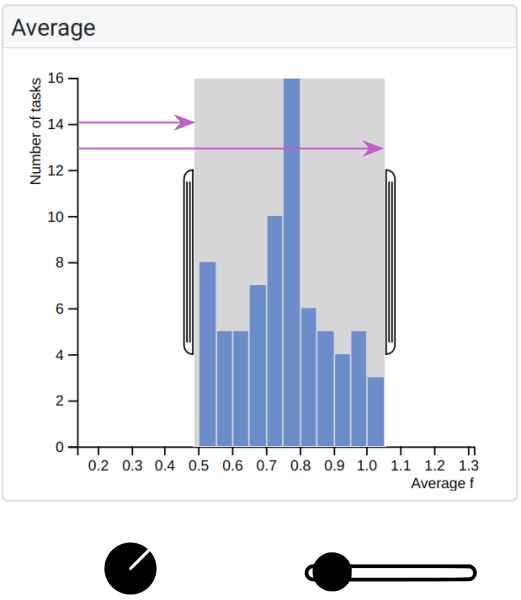
The arguments for using physically connected devices are twofold. The first is a practical one: if the two devices are physically connected in a reasonable way, they can both be operated with one hand rather than two. This is convenient for the user and allows their other hand to be free to operate a separate device, of a different type or of the same type.

The second argument is about semantic separation. A user may choose to use multiple of these devices, using physical layout to encode semantic information about which device is for which task. This is commonly done by sound engineers and pilots. Now imagine switching from one 2D joint asymmetric task to another by moving your hand(s). If the two devices are physically connected, then that conveys information to the user that operating the two devices together operates a singular 2D task, rather than operating two disconnected 1D tasks. It therefore requires less cognition to perform task switching.

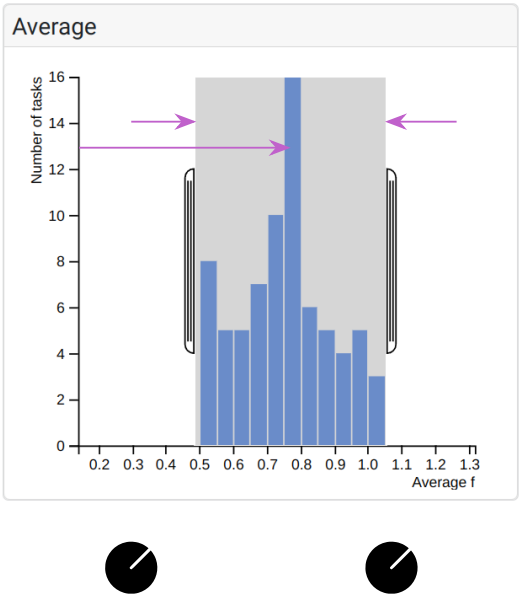
In the list of possible 1D devices given, in order to achieve maximum asymmetry, the natural



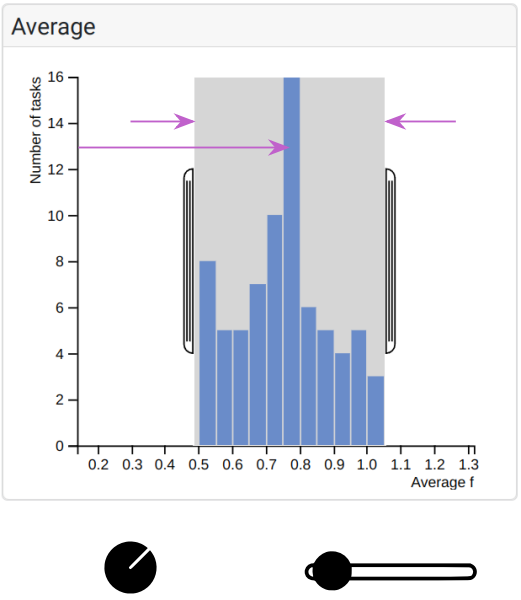
(a) Symmetric control, Symmetric devices. One knob controls the left edge of the window, the other controls the right edge. The variables are symmetric and so are the devices. Their difference can be naturally encoded spatially: the left knob controls the left edge and vice versa. This is easy to commit to procedural memory.



(b) Symmetric control, Asymmetric devices. The knob controls the left edge of the window, the slider controls the right edge. The variables are symmetric, but the devices aren't. This would require unnecessary cognition and/or reliance on visual cues to perform simple operations such as bringing the left edge close to the right edge.



(c) Asymmetric control, Symmetric devices. The left knob controls the width of the window, and the right knob controls the centre of the window. The variables are asymmetric, but the devices are symmetric. It requires unnecessary cognition to remember which device is which, and would be difficult to commit to procedural memory.



(d) Asymmetric control, Asymmetric devices. The knob controls the width of the window, and the slider controls the centre of the window. The variables are asymmetric, and so are the devices. This is natural to use, and easy to commit to procedural memory, as long as a consistent convention is kept across similar control tasks.

Figure 5: Comparison of the use of symmetric and asymmetric devices for 2D joint symmetric and asymmetric control tasks.

choice is one linear device and one rotary device. Within rotary devices, we have the choice of bounded potentiometers or unbounded encoders. Remembering that the 1D components of 2D tasks can be either bounded or unbounded, and noting that the linear device must be bounded, it is natural to choose the unbounded rotary device, i.e. a rotary encoder, such that one device is unbounded and one is bounded, to provide maximum flexibility. How these two devices can be physically connected will be presented in Section 4. A linear slider and rotary encoder is therefore likely a good pair of devices to choose for 2D joint asymmetric control.

Given that real-world tasks are often independent combinations of 1D tasks and 2D joint asymmetric tasks, it makes sense to also include a way to multiplex these devices between tasks, if too few physical instances of SLIDER are available. A simple button was chosen for this, which can be programmed in software to cycle between tasks. To see how these are combined, refer back to Figure 3. The button is activated by pressing down on the rotary encoder.

2.4 Motorised sliders and control modes

It is possible to acquire a linear potentiometer (slider) which can further be moved by a motor. Such a device is strictly superior to an ordinary slider, as it can simply become an ordinary slider by turning off the motor. In order to take further advantage of a motorised slider, the following control modes were designed and implemented:

1. Go To: Command the slider to go to a particular position. This is accurate to well within one MIDI value, and has a number of uses. In the demo that will be presented in Section 7.1, when SLIDER is switched from controlling one parameter to another, it moves back to the last set value of the new parameter.
2. Rumble: Rumble the slider. This could be useful to indicate that the simulation has become unstable, or that parameters are outside safety standards.
3. Notch/Bump: Command the slider to act as if there is a notch or bump in the track, i.e., ``fall into'' or ``fall out of'' a particular position. In the demo that will be presented in Section 7.2, the exact position of the wing trailing edge is a notch, since this is a particularly interesting position for the user to investigate.

2.5 Choice of display devices

In a situation where a user is using multiple SLIDER devices to control different subtasks at the same time (for example, two histogram windows), it is necessary to label these devices in order to inform the user which device is set up for which task. This should be software-controllable, as the necessary labels may change during a current task context or between task contexts. A good choice for this labelling is simple and cost-effective 8x8 dot-matrix displays; they provide the right amount of information without distracting.

2.6 Protocol design

This subsection details the design of the interfacing protocol between SLIDER and the computer it connects to.

2.6.1 The MIDI interface

MIDI, or the Musical Instrument Digital Interface, is a collection of standards originally designed for use in hardware musical devices such as synthesisers. It contains a standard for an interoperable protocol designed for a number of interface purposes such as transmitting the position of a slider. Each MIDI message contains up to 3 bytes, for example:

$\underbrace{0xC0}$	$\underbrace{0x03}$	$\underbrace{0x05}$
control change, channel 0	device number (0-127)	value (0-127)

This message indicates that a certain control device on channel 0, in this case a slider, has been set to a value of 5. Each time the slider is moved, a new message is sent. The MIDI protocol can operate over a USB connection, and its ubiquity and wide support makes it a natural choice to use as a protocol for sending the locations of the slider and rotary encoder from SLIDER to a computer.

2.6.2 MIDI control of SLIDER

Some control information must be sent in the opposite direction, from the computer to SLIDER, for example to set the control mode of the motorised slider in Section 2.4.

To achieve this control at minimal complexity, the MIDI can be used again, with messages now sent from the computer to SLIDER's microcontroller. SLIDER presents itself as an ordinary MIDI device. This can be seen in the block diagram in Figure 6.

Each connected SLIDER occupies one MIDI channel, of which there are 16. The second byte encodes a command, and the third byte contains data relevant to that command. For example, to set the content of the display to a character, the following MIDI message is used:

$\underbrace{0xC0}$	$\underbrace{0x30}$	$\underbrace{0x05}$
Control Change, channel 0	Set display to ASCII character	US-ASCII value (0-127)

Common variable symbols in engineering situations are Greek symbols; these are encoded using an alternate command:

$\underbrace{0xC0}$	$\underbrace{0x31}$	$\underbrace{0x05}$
Control Change, channel 0	Set display to Greek character	Greek charset value (0-127)

These character sets are determined ahead of time and placed into the microcontroller's firmware. The microcontroller translates MIDI control changes into a bitmap for the character, and sends this over a shared I^2C bus (a hardware level communication protocol) to the displays. This will be seen in the next section in the block diagram in Figure 6. The full specification of MIDI interface can be seen in Section 2.7.2

2.7 Specification

The following is the full external-facing specification of SLIDER. Based on this external-facing specification, Sections 3 to 5 will detail the choices and design of specific hardware and firmware to conform to this specification.

2.7.1 Mechatronic

SLIDER will consist of:

1. One motorised slider, with the following control modes:
    a. Passive: act as an unmotorised slider.
    b. Go To: move to the specified position, then enter Passive mode.
    c. Rumble: vibrate the slider in place.
    d. Notch: create a virtual notch at the specified position.
    e. Bump: create a virtual bump (inverse notch) at the specified position.
2. One rotary encoder, mounted on top of the motorised slider
3. One 8x8 dot matrix display, which can be set to ASCII or Greek characters.
4. One Micro-USB Type-B port for MIDI interfacing with a computer
5. One standard battery connector for a 9V battery.

2.7.2 MIDI Interface

Table 1: MIDI message specification from SLIDER to Computer, where SLIDER is configured to operate on channel Y. 0xWZ denotes a value in hexadecimal.

Purpose	Byte 1	Byte 2	Byte 3	Note
Slider position	0xBY	0x00	0x00 to 0x7F (slider position)	0xBY is a control change on channel Y
Encoder position	0xBY	0x01	0x00 to 0x7F (encoder position)	

Table 2: MIDI message specification from Computer to SLIDER, where SLIDER is configured to operate on channel Y. 0xWZ denotes a value in hexadecimal.

Purpose	Byte 1	Byte 2	Byte 3	Note
Set slider to passive mode	0xBY	0x00	unused	0xBY is a control change on channel Y
Set slider to Go To mode	0xBY	0x01	0x00 to 0x7F (position to go to)	Slider will return to passive mode after reaching new position or a timeout.
Set slider to rumble mode	0xBY	0x02	0x00 to 0x7F (rumble strength)	
Do not use	0xBY	0x03	unused	Unimplemented mode.
Set slider to notch mode	0xBY	0x04	0x00 to 0x7F (notch position)	
Set slider to bump mode	0xBY	0x05	0x00 to 0x7F (bump position)	
Set slider timeout	0xBY	0x05	0x00 to 0x7F (timeout)	Timeout is multiplied by 8 and then interpreted as milliseconds.
Set display to ASCII character	0xBY	0x30	0x00 to 0x7F (ASCII char)	
Set display to Greek character	0xBY	0x31	0x00 to 0x39 (Greek char)	Unicode points U+0390 onwards.
Set encoder position	0xBY	0x7F	0x00 to 0x7F (position)	Encoder does not move, but new events are relative to this new position

3 Electronic Design

In this section, specific electronic devices required to implement the specification in Section 2.7 will be identified. The block diagram for how these devices interact can be seen in Figure 6. This will lead to a full electronic schematic, which can be seen at the end of this section, in Figure 12.

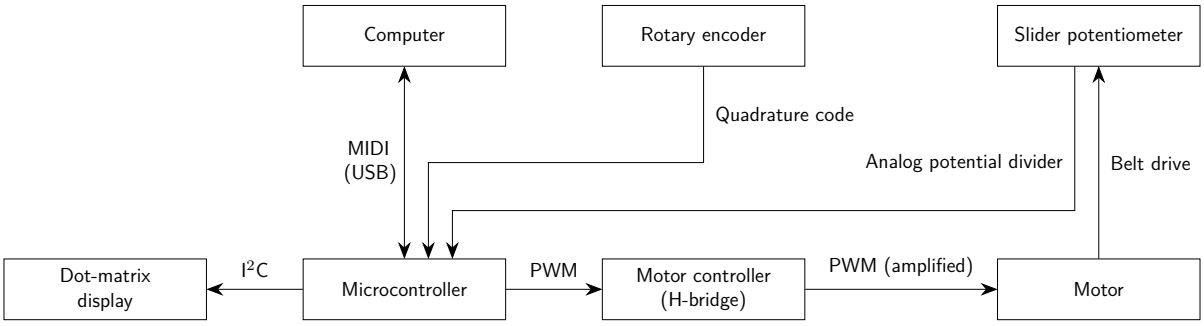


Figure 6: Block diagram of SLIDER

3.1 Microcontroller

Many simple microcontrollers would be sufficient to control SLIDER. An Arduino Micro (Figure 7) was chosen as an inexpensive and small microcontroller development board capable of acting as a USB slave device and with sufficient hardware pins for everything required.

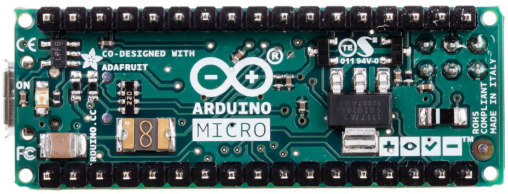


Figure 7: The Arduino Micro (image reproduced from promotional material)

3.2 Slider

For the motorised slider, a Bourns PSM01-082A-103B2 (Figure 8) was chosen as a simple motorised slider with appropriate length and connectivity with ordinary jumper wires. This contains a variable resistor track through which a simple analog potential divider can be constructed to provide position sensing.

Figure 8: The Bourns PSM01-082A-103B2 motorised slider (image reproduced from promotional material)

3.3 Motor controller

The motor of the motorised slider is a simple DC motor, so direction can be controlled by a H-bridge (which switches the polarity of the power connections), and its power can be controlled using Pulse-Width Modulation (PWM). In PWM, power to the motor is pulsed with a variable ``Duty Cycle'', where the Duty Cycle is the fraction of each cycle in which the motor is powered. This happens at high frequency in order to approximate a variable voltage to the motor. For this task, a simple motor controller development board was chosen, SKU 102514 from The Pi Hut (Figure 9).

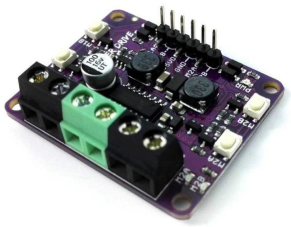


Figure 9: The Pi Hut 102514 motor controller (image reproduced from promotional material)

3.4 Power

The motor requires more power than a USB connection can provide, so an standard external connection for a battery is included. It can support 6V to 9.5V, but a 9V battery is recommended such that the motorised slider has the maximum power to overcome the friction from the wiring to the rotary encoder attached on top.

3.5 Rotary encoder and button

For the rotary encoder and button, a Bourns PEC11R-4220K-S0024 (Figure 10) was chosen as a simplest-possible solution with appropriate mounting and quadrature output.

3.6 Display

The 8x8 dot-matrix display is an Adafruit 1080 (Figure 11), which includes an I^2C (a low-level 4-wire connection protocol) LED driver circuit for interfacing with the microcontroller.

Figure 10: The Bourns PEC11R-4220K-S0024 rotary encoder (image reproduced from promotional material)

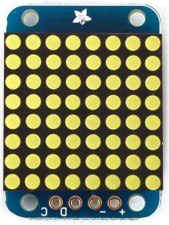


Figure 11: The Adafruit 1080 (image edited from promotional material)

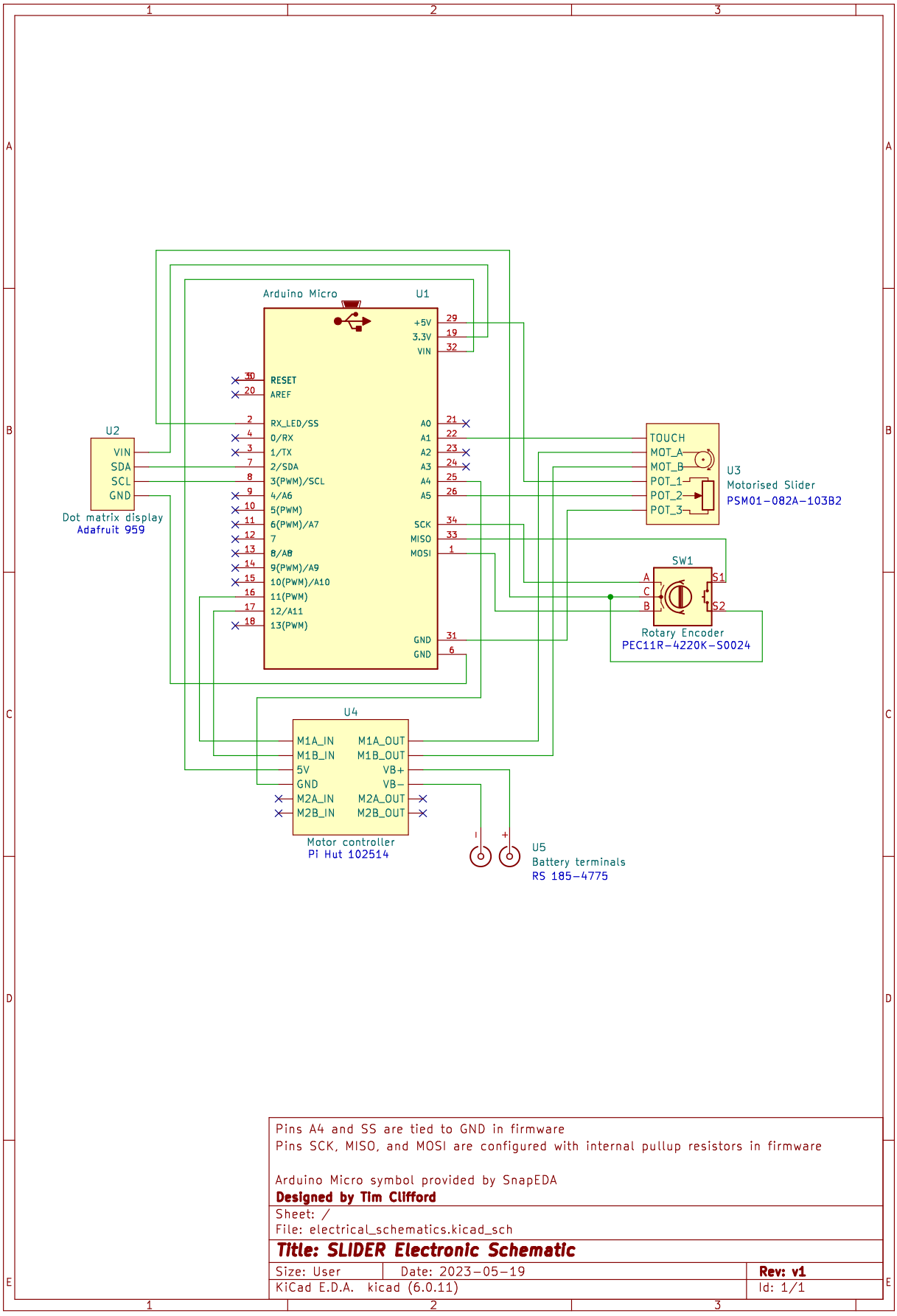


Figure 12: Electronic schematic of SLIDER, showing the wiring between the microcontroller, motorised slider, motor controller, rotary encoder, dot matrix display, and battery terminals.

4 Mechanical Design

In this section, the design of the physical housing of SLIDER is detailed, including the housing and the connection between the motorised slider and the combined rotary encoder and switch. The mechanical design is sufficient for the prototype, but could be improved. A potential improvement to improve robustness is detailed.

4.1 Housing

In order to contain the devices specified in Section 3, a simple and compact box was made for SLIDER. This box is designed to be printed on an ordinary 3D-printer in ABS, such that interested parties can easily replicate the prototype and improve on it. Full technical drawings of the parts of the box can be seen in Figures 14 to 16. The full assembly can be seen in Figure 13. Ordinary machine screws will self-tap into printed holes in ABS, making fastening very simple.

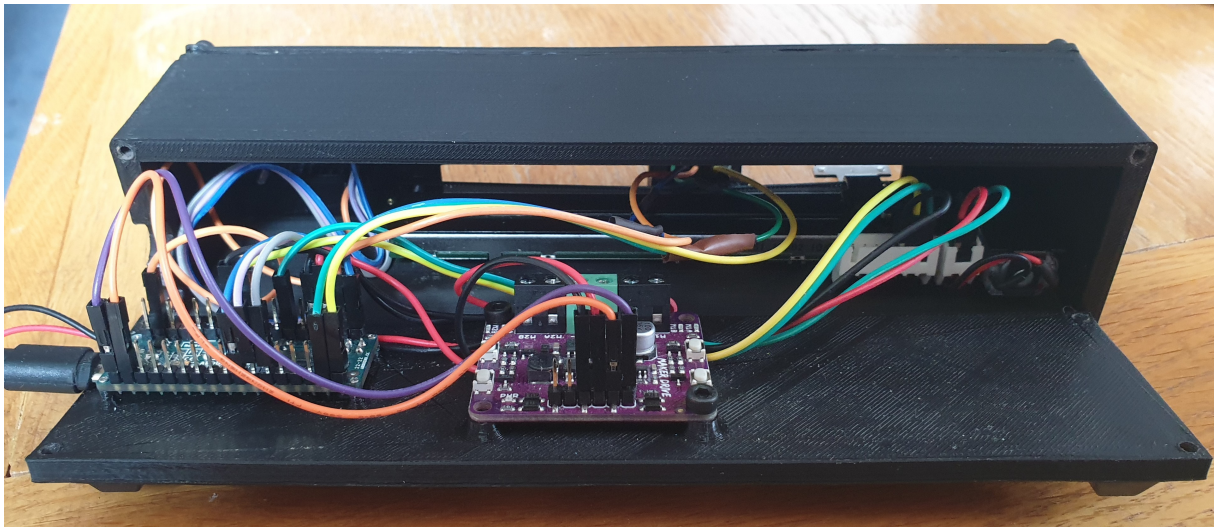
4.2 Bracket

To connect the rotary encoder to the motorised slider, a simple bracket was designed in the same way. A technical drawing can be seen in Figure 17. There is a hole for attachment to the rotary encoder's thread and nut, and a slot for a friction fit with the motorised slider. These can be seen in Figures 13b and 13c. A pure friction fit worked reasonably well, but to improve the mounting further a small quantity of removable adhesive putty was added.

4.3 Improvements

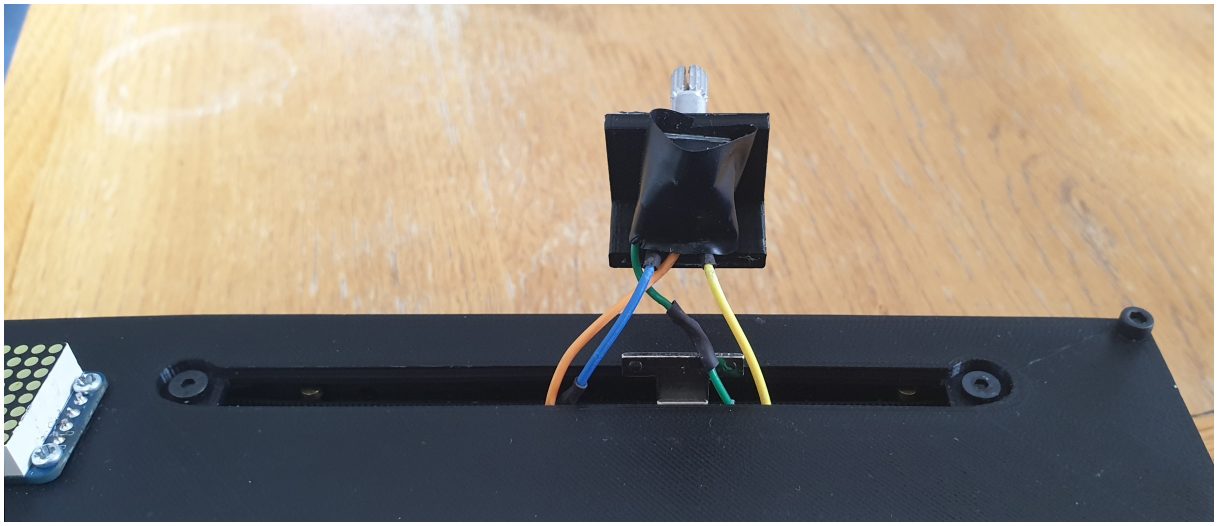
The wiring to the rotary encoder could be improved in a later version of SLIDER. For the prototype, as can be seen in Figures 13a and 13c, jumper wires attached to the microcontroller at one end were snipped at the other end and soldered to the rotary encoder, protected by heatshrink and electrical tape. Due to the compactness of the box's design, these wires rub against other wires inside the box when the slider moves, creating unnecessary friction both for the user and for the PID controller. More problematically, there are a number of relatively sharp surfaces inside the box, such as the edges of the motorised slider housing, which can occasionally cut into the wires if they rub against them. This necessitated repairs during testing.

An improvement to make SLIDER more robust would be to use a flexible ribbon cable to connect the rotary encoder to the microcontroller. This would require a custom (although very simple) PCB to mount to the rotary encoder and adapt it to the ribbon cable connector. Another PCB would be required to adapt the other end of the ribbon cable to ordinary 0.1in jumper headers for connection to the microcontroller. This second PCB would be mounted to the inside bottom of the box directly under the centre of the slider track, and the ribbon cable would be the right length such that it does not contact other surfaces during motion.



(a) Internal mounting and wiring

(b) Encounter mounting to bracket



(c) Encoder wiring and bracket mounting to motorised slider

Figure 13: Mechanical assembly of SLIDER

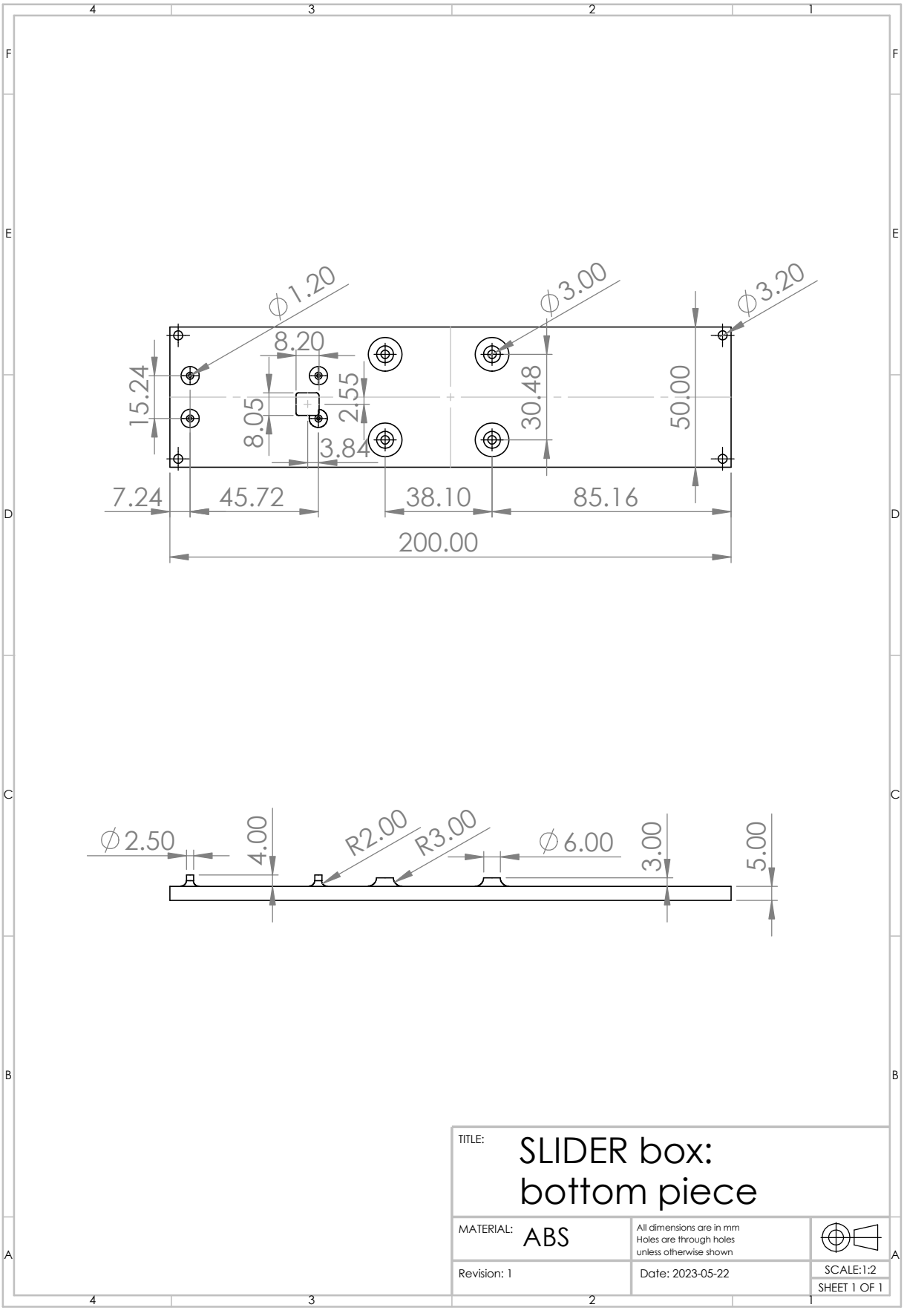


Figure 14: Technical drawing of the bottom piece of the SLIDER box.

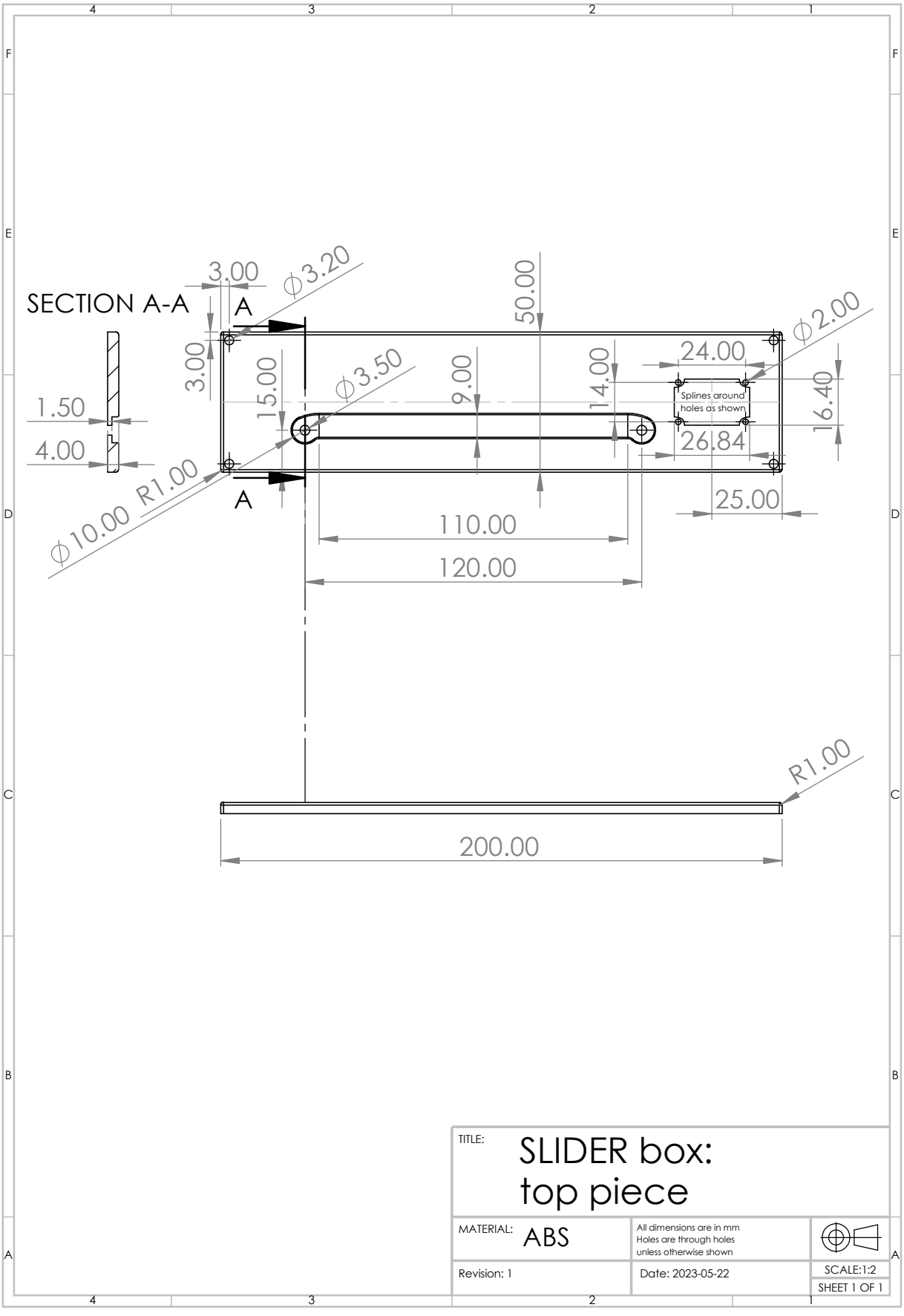


Figure 15: Technical drawing of the top piece of the SLIDER box.

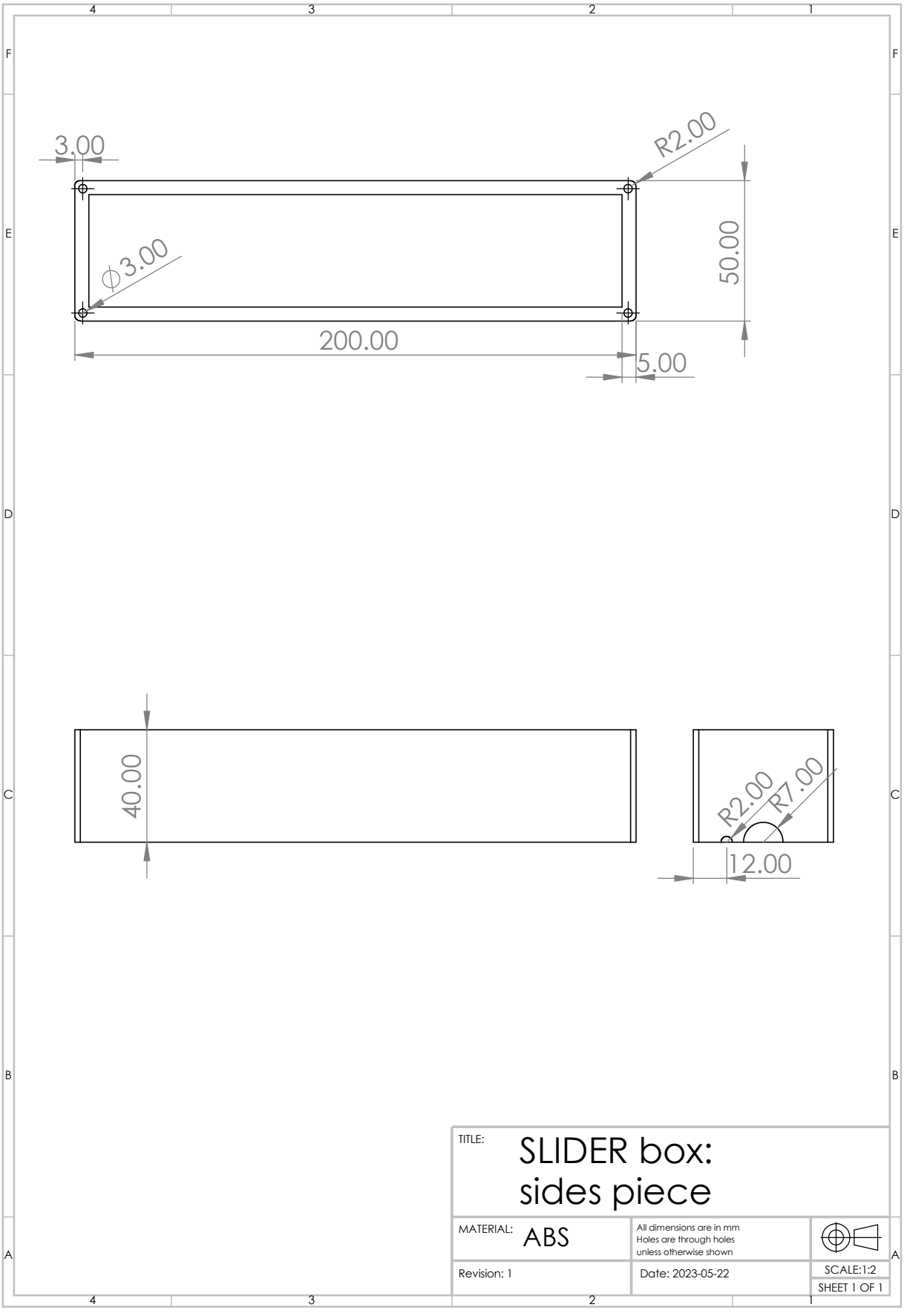


Figure 16: Technical drawing of the sides piece of the SLIDER box.

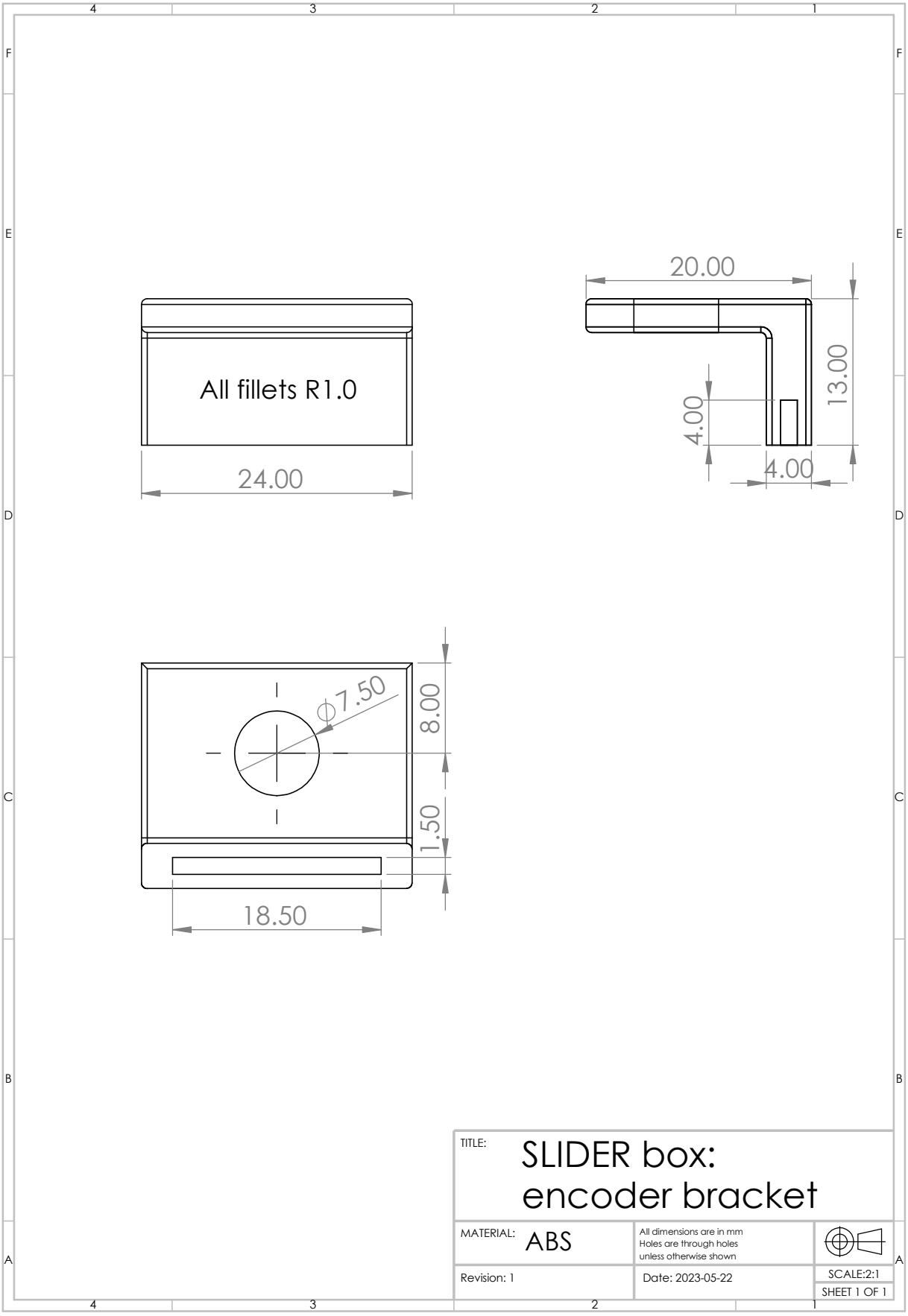


Figure 17: Technical drawing of the bracket used in SLIDER to attach the rotary encoder to the motorised slider.

5 Firmware Design

In this section, the design of SLIDER's firmware is detailed. The firmware is the code which runs on the microcontroller in order to control all of the electronic components and interface with the software via MIDI over USB. Two key elements of the firmware are explained in detail. Firstly, the multithreading architecture, and secondly the design and implementation of the control modes as specified in Section 2.7.

Each component controlled by the firmware -- the motorised slider, rotary encoder, and display -- is abstracted into a C++ class. Each can provide a thread which outputs MIDI events and a function which handles incoming MIDI events; each of the latter functions are combined into a single thread which handles all incoming MIDI events.

5.1 Multithreading with protothreads

These threads must be executed in parallel, otherwise it would not be possible to run each thread at an acceptable rate or give each an optimal share of CPU time for good performance.

In an ordinary computer, threads are scheduled and managed by the operating system kernel. Microcontrollers, such as the Arduino used in this project, are too resource-constrained to run anything but the most basic operating system, and typically run a single self-contained program. Thus, full threading support does not exist.

One solution is called ``protothreading'', which is a stripped down and less powerful type of threading. There exist libraries for the Arduino ecosystem which provide protothreading [Artin, 2020]. Protothreading must be understood in detail in order to understand when and how assumptions derived from ordinary threading will break down.

An example of a protothreaded program is as follows. In the main loop function, multiple threaded functions are run repeatedly, but act as if they are running almost simultaneously:

```
void loop() {
    midi_in_thread();
    other_thread(); // and so on
}
```

We can understand this by examining one of the protothreaded functions in the SLIDER firmware source code:

```
int midi_in_thread(void) {
    PT_BEGIN(&midi_in_pt);
    while (1) {
        midiEventPacket_t rx = MidiUSB.read();
        if (rx.header != 0) {
            motorised_slider.handle_midi_event(rx);
            rotary_encoder.handle_midi_event(rx);
            displays_handler.handle_midi_event(rx);
        }
        PT_YIELD(&midi_in_pt);
    }
    PT_END(&midi_in_pt);
}
```

PT_YIELD is a preprocessor macro which can be understood on a high-level as a function which temporarily hands over CPU time to other threads. In order to understand this in detail, we must examine this function with the preprocessor macros expanded (simplified for explanatory purposes, comments are my own):

```
    int midi_in_thread(void) {
        // if we have already started running this function,
        // jump to the address where we last exited.
        if ((&midi_in_pt)->lc != __null) {
            goto *(&midi_in_pt)->lc;
        }
        while (1) {
            midiEventPacket_t rx = MidiUSB.read();
            if (rx.header != 0) {
                motorised_slider.handle_midi_event(rx);
                rotary_encoder.handle_midi_event(rx);
                displays_handler.handle_midi_event(rx);
            }
            // set the address of the entry point, and exit with code
            // 1, which means that the thread is not finished yet.
            ((&midi_in_pt)->lc) = &&LC_LABEL86;
            return 1;
LC_LABEL86: // this is the entry point, immediately after we exited
        }
        // set an invalid entry point and exit with code 3,
        // meaning that the thread has finished.
        (&midi_in_pt)->lc = __null;
        return 3;
    }
```

We see that PT_YIELD, and protothreaded functions in general, operate by saving an address to jump to and then exiting. When the function is run again, PT_BEGIN jumps back to that address.

This can fail in a number of ways. The most important is that if the time cost of the instructions between each PT_YIELD (or other similar functions such as PT_SLEEP) is too high, then during this time every other thread will be starved. A specific example of this problem in the SLIDER firmware is in the MIDI output thread. In order to send the MIDI event, the firmware must call MidiUSB.sendMIDI. This in turn calls the inbuilt Arduino USB_Send. The problem is that USB_Send has a hardcoded timeout of up to 250ms, using the core Arduino delay function, in order to wait for a lock on the USB endpoint. This would starve the motor control thread (described in Section 5.2.1), causing a control stability failure.

This would be relatively easy to solve with true threading, but in a protothreaded architecture the only solution is to replace the timeout in USB_Send with a version that yields CPU time to the other threads. This was achieved by creating patched versions of the MIDIUSB and core Arduino libraries to introduce protothreaded versions of USB_Send and sendMIDI, named PT_USB_Send and ptSendMIDI respectively, where the protothread is passed in as an extra argument and delay is replaced by PT_SLEEP, which hands off control to the other threads.

5.2 Control Mode Design

The motorised slider class has an extra thread to control the motor, this depends on the active control mode state, which is set by the motorised slider's incoming MIDI event handling function. How the motor's power is set for each control mode is discussed below.

5.2.1 Go To via PID control

From Figure 6 and earlier discussion in Section 3 we know that we are effectively controlling the voltage of a DC motor connected via a belt to the slider, with feedback of position of the slider. It is usually sensible to apply Proportional-Integral-Derivative control, a generic and common control strategy where (in this instance) the output voltage is a linear combination of the difference in position, velocity, and integral of difference in position, as follows:

$$\begin{aligned} M_{\text{PID}}(t) &= P\left(x(t) - T\right) \\ &\quad + I \int_0^t \left(x(\tau) - T\right) d\tau \\ &\quad + D\frac{d}{dt}\left(x(t) - T\right) \end{aligned} \tag{1}$$

where	$M_{\text{PID}}(t)$	is the motor power set by the PID controller
	P	is the proportional constant
	I	is the integral constant
	D	is the derivative constant
	$x(t)$	is the slider position
	T	is the target position

Of course, the microcontroller can only operate a discrete quantised system, not a continuous system, so the motor power is instead set as follows, where backwards-Euler is used to estimate the derivative and integration is estimated using a sum of rectangles:

$$\begin{aligned} M_{\text{PID}}(k\Delta t) &= P\left(x(k\Delta t) - T\right) \\ &\quad + I \sum_{j=0}^{k} \left(x(j\Delta t) - T\right) \\ &\quad + D\left(x(k\Delta t) - x\left((k-1)\Delta t\right)\right) \end{aligned} \tag{2}$$

where	Δt	is the timestep between each iteration
	k	is an arbitrary natural number
	I and D	absorb any constant factor of Δt in integral/derivative estimation

It is perhaps tempting to make Δt as small as possible so as to approximate the continuous case, but there are two reasons not to. One is the that there may exist noise in $x(t)$, and by setting Δt larger, we allow the derivative estimate $x(k\Delta t) - x\left((k-1)\Delta t\right)$ to have larger absolute value and therefore be more robust to noise. The second, and related, reason is that

$x(t)$ is quantised; it can only take integer values in the range 0 -- 1023 due to hardware limits measuring the analog potential divider. Thus, if Δt is too small, $x(k\Delta t) - x\left((k-1)\Delta t\right)$ will typically be zero, and thus derivative action could not happen.

A complication with using a PID controller to control the motorised slider is that ordinarily they are designed for linear systems, i.e. systems which follow the linear property $f(ax+by) = af(x)+bf(y)$. The response function of the slider position to motor power is highly nonlinear, due to friction. For a motor power M less than some critical value M_c, the motor does not provide a high enough force to overcome the static friction on the slider and the resulting $x(t) = 0$. But above M_c, friction is overcome and $x(t) \neq 0$. Clearly this violates linearity.

This system can nonetheless be approximated to be linear (sufficiently for SLIDER's purposes) if M_c is taken as a virtual ``zero'' point, as follows:

$$M(t) = \begin{cases} M_{\mathrm{PID}}(t) + M_c & M_{\mathrm{PID}}(t) > 0 \\ M_{\mathrm{PID}}(t) - M_c & M_{\mathrm{PID}}(t) < 0 \\ 0 & \mathrm{otherwise} \end{cases} \tag{3}$$

Optimisation of the choice of the constants P, I, and D in PID controllers is a large topic, but for our purposes, hand-tuning simply by observing the slider is more than sufficient.

For SLIDER, the following values were chosen:

$$\begin{array}{lll} \Delta t & = 2 & \mathrm{ms} \\ P & = 1.5 & \Delta M/\Delta x \\ I & = 0 & \Delta M/\Delta x \\ D & = 3 & \Delta M/\Delta x \\ M_c & = 126 & \Delta M \end{array}$$

$$\begin{array}{rl} \text{where } \Delta M & \text{is a quantisation step of motor power (value 1 but dimensional)} \\ \Delta x & \text{is a quantisation step of slider position (value 1 but dimensional)} \end{array}$$

It may be initially surprising that I was taken to be zero, but this is intuitively understandable when considering that the high friction results in the slider stopping very quickly when power is removed, thus steady state error without integral control is minimal.

5.2.2 Rumble

Rumble control is much easier, the motor power can simply be set to vibrate as follows:

$$M(k\Delta t) = \begin{cases} -M((k-1)\Delta t) & k > 0 \\ M_r & \text{otherwise} \end{cases} \tag{4}$$

$$\begin{aligned} \text{where } M_r &\quad \text{is the software-defined rumble power} \\ \Delta t &\quad \text{sets the rumble frequency} \end{aligned}$$

For SLIDER, Δt for the rumble mode was set to 10ms, giving a comfortable rumble frequency.

5.2.3 Notch/Bump via windowed PID

In order to achieve a notch or bump, the PID controller from Section 5.2.1 can be used again, but with one major difference. The PID controller is commanded to bring the slider to the position of the notch, but the motor power it commands is multiplied by a windowing function centred on the position of the notch. This can be seen in Figure 18. A natural choice for this window would be a Gaussian curve, but in testing it was found that a squarer curve was more comfortable to use. A natural choice then would be to use a generalised Gaussian curve, as described by Nadarajah [2005], where another value β can be used to control the squareness of the curve, which higher values of β more square. The chosen curve can be seen in Figure 18.

A bump is defined to be an inverse notch; it can be achieved by inverting the direction of the motor output of a notch. Then the motor will act in the region of the bump to push the slider off of the bump.

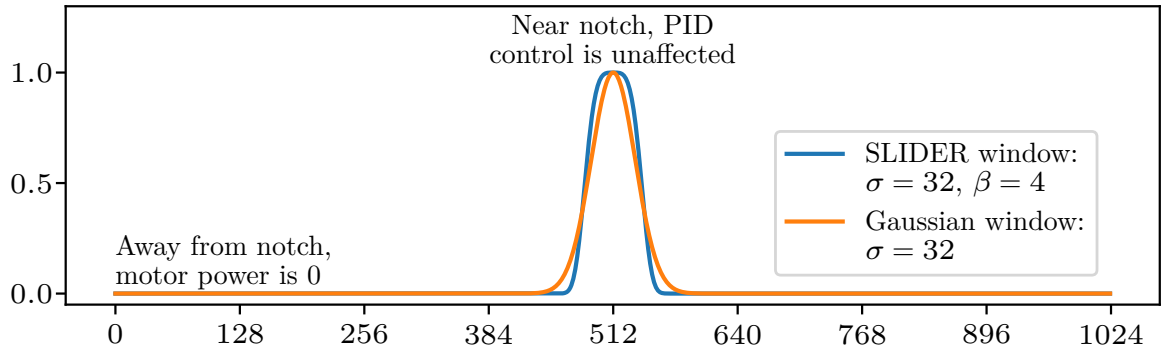


Figure 18: The generalised Gaussian window chosen for SLIDER compared to a Gaussian window with the same width parameter, both with notch centre at 512. 0 to 1024 is the full range of the slider. It can be seen from this graph how much of the slider track will fall into the notch.

6 Software Design

In this sections, the architecture is detailed of the software demos that have been written to demonstrate SLIDER's capabilities. Particular detail is given to ensure that the reader could use this section to integrate SLIDER into their own projects. SLIDER's MIDI interface, detailed in Section 2.7.2, is very simple, and the output messages it sends from the slider and rotary encoder will work with zero configuration in any MIDI-aware program. Taking full advantage of the MIDI messages that can be sent to SLIDER to configure it does require some special software.

6.1 Simple example

In order to best understand the software architecture of the demos, consider the following simple example, which will be explained in sufficient detail that the reader can use it develop their own MIDI-aware website, taking advantage of SLIDER. This example has the following directory structure:

```
simple-example
├── midi.js
├── midi.json
├── index.html.template
├── site_gen.py
└── index.html
```

`midi.js` is a file containing a javascript class called `midiHandler` and other helper functions. `midi.json` specifies the midi-aware inputs of the page, with the following example content:

```
[
    {
        "name": "Input 1",
        "shortName": "input1",
        "symbol": "1"
    },
    {
        "name": "Input 2",
        "shortName": "input2",
        "symbol": "2"
    },
    {
        "name": "Input 3",
        "shortName": "input3",
        "symbol": "3"
    }
]
```

This defines three inputs. For each, ``name'' is a human readable name, ``shortName'' is a machine-readable name, and ``symbol'' is the symbol associated with the input which will be displayed on SLIDER's 8x8 display.

`index.html.template` is a template for the main html index file of the page, and has the following in its body:

```
{MIDI_BOILERPLATE}

<input type="range" min="0" max="127" name="input1" id="input1" value="0">
<input type="range" min="0" max="127" name="input2" id="input2" value="0">
<input type="radio" name="input3" id="input3" value="0">
```

`{MIDI_BOILERPLATE}` is a location where a midi settings dialog will be added, this is why the file is called `index.html.template` -- running `site_gen.py` will insert this boilerplate and output the complete index file to `index.html`. The MIDI settings dialog for this example can be seen in Figure 19.

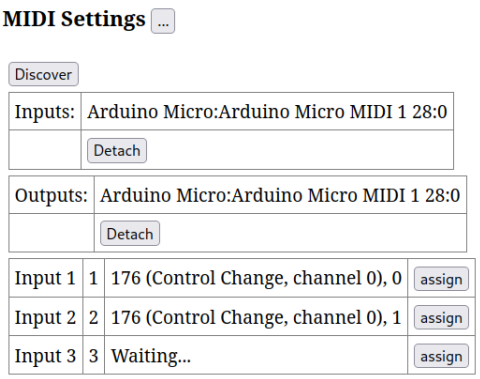


Figure 19: MIDI settings dialog for the simple example. The user attaches to the SLIDER inputs and outputs, then assigns by clicking "Assign" and moving the appropriate device.

`index.html.template` also contains the following javascript, which sets up MIDI handling:

```
// create the midiHandler object
let mHandler = new midiHandler(document.getElementById("midi"), "mHandler")

// populate it with event handlers for each input defined in midi.json
mHandler.midi_event_handlers["input1"] = function(value) {
    document.getElementById("input1").value = value;
}

mHandler.midi_event_handlers["input2"] = function(value) {
    document.getElementById("input2").value = value;
}

mHandler.midi_event_handlers["input3"] = function(value) {
    if (value === 127) { // button down
        var inp = document.getElementById("input3");
        inp.checked = !inp.checked;
    }
}

// start the event loop, which will run forever
mHandler.eventLoop();
```

The first line creates the `midiHandler` object, passing in the element of the MIDI settings dialog (part of `{MIDI_BOILERPLATE}`), and its own name, which it requires because it inserts HTML input buttons into the page which call functions of `midiHandler`. Next, `midiHandler.midi_event_handlers` is populated with an event handler function for each input which was defined in `midi.json`.

These event handler functions will be run as quickly as the webpage can handle, each time updating the page as desired by the website designer. In this simple example, the first two inputs, connected by the user to SLIDER's slider and rotary encoder, will simply move a HTML input slider. The third, connected to SLIDER's button, will toggle a checkbox.

The last line of the example javascript will start the event loop, which we will now discuss.

6.2 The event loop

Javascript has an event driven architecture, so when a page is attached to a MIDI device, a callback must be registered, which will be run every time a new MIDI event is received. The callback our `midiHandler` registers has the following pseudocode:

```
if (midi event is a control change)
    elementName <- element name as attached in MIDI settings dialog
    unhandled_events[elementName] <- value (byte 3) of MIDI event
end if
```

Then, the event loop has the following pseudocode:

```
loop forever
    for elementName in (elements with attached inputs)
        if (unhandled_events[elementName] != null)
            midi_event_handlers[elementName](unhandled_events[elementName])
            unhandled_events[elementName] <- null
        end if
    end for
    sleep(10ms)
end loop
```

It might seem strange to not simply run the event handlers in the callback directly. To understand why this would be a bad idea, consider that javascript has its own event loop, but it operates very differently to `midiHandler`'s event loop. In the javascript internal event loop, new events are added to a queue, and the queue is worked through one by one. If events arrive much faster than can be handled, they will still never be skipped, and the entire page will lag.

In `midiHandler`'s event loop, as seen above, new events might come in much faster than the event loop handles them, but the event loop will only ever run the event handlers on the latest values that were received. Some events may never be handled by `midiHandler`'s event loop, but this is hardly an issue -- in any reasonable MIDI controlled task, only the latest value matters. For example, if the MIDI events specify the position of a slider, only the final position matters, it is merely a nicety to have the page update with intermediate values.

Thus, `midiHandler`'s event loop prioritises updating in realtime rather than handling every value as it comes in. This prevents lag from being an issue in the more heavyweight demos that will be seen later.

6.3 A more complex example

`midiHandler` has a further features which are used in the demos in Section 7. This demo can be seen in Figure 21. Here is the event handler for SLIDER's button used in histograms demo, simplified for explanatory purposes:

```
mHandler.midi_event_handlers["switch"] = async function(value) {
    if (value != 127) return;
    if (Object.values(mHandler.control_change_dict).includes("r_center")) {
        // "r_center" is currently being controlled, switch to "e_center"
        // (SLIDER is on channel 0, see SLIDER MIDI message specification)
        mHandler.assignMIDI("e_center", 0, 0); // channel 0, device 0
        mHandler.assignMIDI("e_extent", 0, 1); // channel 0, device 1
    } else {
        mHandler.assignMIDI("r_center", 0, 0);
        mHandler.assignMIDI("r_extent", 0, 1);
}   }
```

This code checks if the `r_center` variable is in the dictionary describing active control assignments: `control_change_dict`. If so, the histogram with variables `r_center` and `r_extent` is being controlled, so the page reassigns SLIDER's inputs to the other histogram, with variables `e_center` and `e_extent`. If not, SLIDER's inputs are reassigned to the first histogram. Thus, pressing the button toggles SLIDER between the two histograms.

This is combined with another feature called assignment callbacks. Assignment callbacks are set each time an input is assigned to. Here is one example:

```
mHandler.assignment_callbacks["r_center"] = function(value) {
    // put SLIDER into Go To mode, and move it to the new position.
    // `midi_hist_shapes` contains the current shapes of the histograms
    mHandler.goTo(0, midi_hist_shapes["reaction_norm"].center);
    // `plot_indices` maps the names to indices in the html
    for (const [val, idx] of Object.entries(plot_indices)) {
        if (idx === plot_indices["r_center"]) {
            // set the border of the new plot to red
            document.getElementsByClassName("plot")[idx].style.border
                = "5px solid red";
        } else {
            // set the border of all other plots to white
            document.getElementsByClassName("plot")[idx].style.border
                = "5px solid white";
}   }   }
```

This assignment callback is set up such that when the SLIDER switches to controlling one histogram, it will physically move its motorised slider to the position of the centre of that histogram, and the page will update so that the newly controlled plot is highlighted in red. `goTo(channel, position)` is a convenience function to abstract away the interface specified in Section 2.7.2, there is also `setPassive(channel)`, `rumble(channel, value)`, `setNotch(channel, position)`, `setBump(channel, position)`, `setEncoder(channel, value)`, and `setSymbol(channel, value)`, which correspond to the other available MIDI messages detailed in Section 2.7.2.

7 Results and Discussion

In order to evaluate SLIDER against existing interaction devices such as touchscreens or mice, two demos were developed, and a number of researchers from the Whittle Lab were asked to use SLIDER for the demos, compare it to existing devices, and fill out an online survey. The setup of the demos can be seen in Figure 20. A total of five responses to the survey were obtained.

An important limitation of the demos and survey, with reference to the task space of Section 1.3, is that SLIDER targets high-frequency tasks performed by simulation engineers. The time available for a demo was not sufficient to gain quantitative data on whether SLIDER would provide an efficiency gain for these tasks in the long term, due to the learning curve overhead of an entirely new interaction paradigm.

With that in mind, the survey was designed to elicit mostly qualitative responses about how good SLIDER felt to use, and each person's estimate about its effectiveness in the long term and in their own projects.

The results were very positive, with every participant enjoying SLIDER more than a mouse or touchscreen. There were a number of interesting suggestions about how SLIDER could be improved, and interestingly also about how mouse interaction could be improved to give it a better shot. These are discussed in Section 7.3.

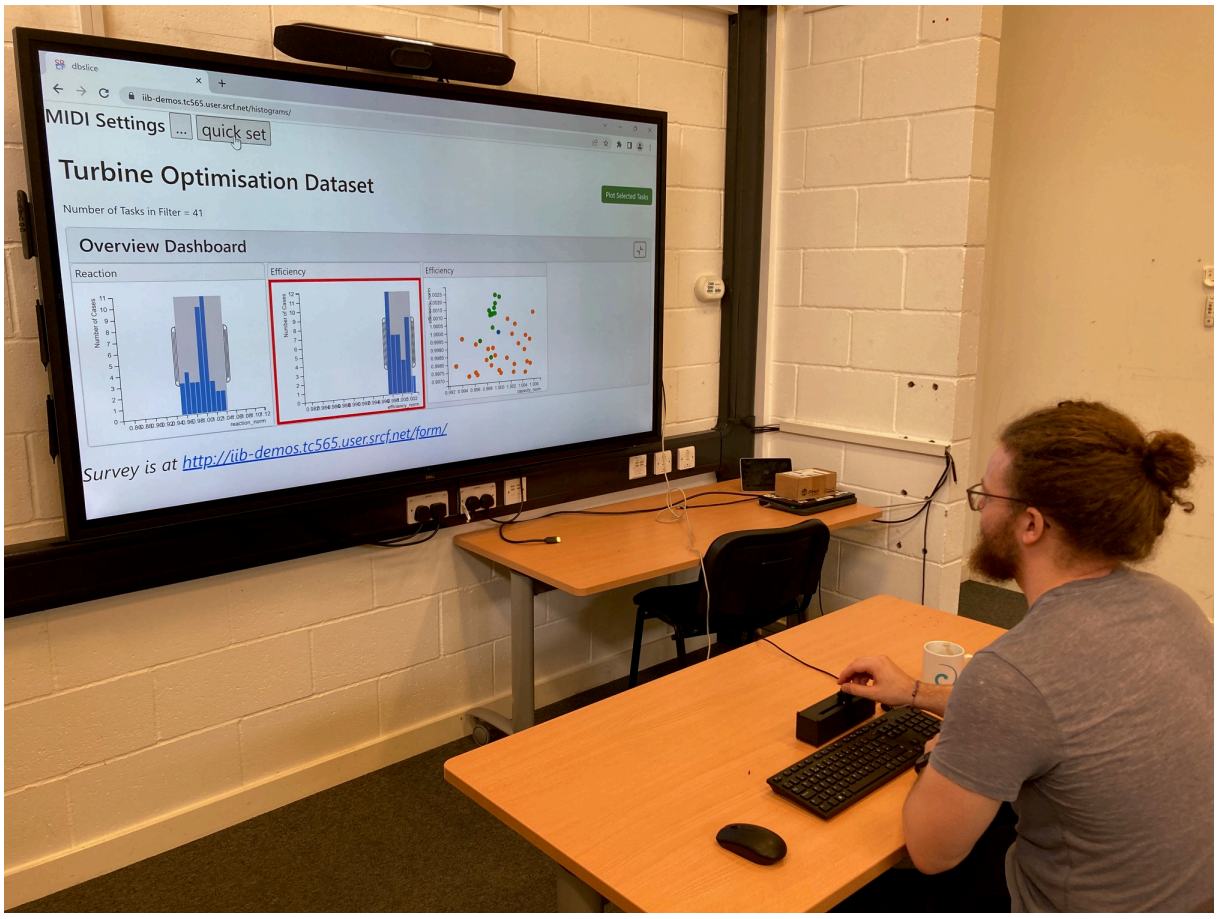


Figure 20: Setup for the demos

7.1 Demo 1: Dual histogram windowing

The first demo, which can be seen in Figure 21, is a good example of an independent combination of 2D joint asymmetric control tasks, which SLIDER is primarily designed for.

In the demo, the slider controls the histogram's position, and the rotary encoder controls its width. When the button is pressed, SLIDER switches to controlling the other histogram. This was also an opportunity to demonstrate the power of the motorised slider's ``Go To'' mode: when the button is pressed, the motorised slider physically moves to the position where the new control input was last left.

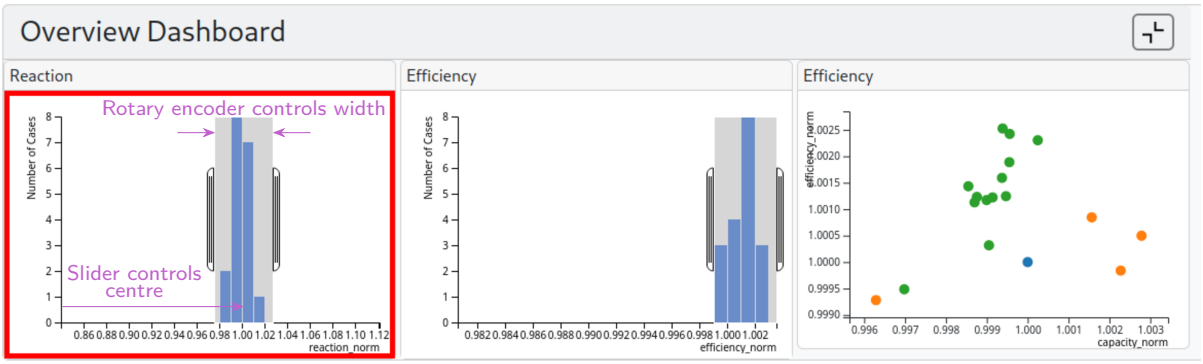


Figure 21: Histogram windowing demo. Annotations in purple are not part of the page.

7.2 Demo 2: Slice through an unsteady flow

The second demo is designed to demonstrate the power of the ``Notch'' mode from Section 2.4. The demo is a slice through a snapshot of an unsteady flow, as can be seen in Figure 22. The slider was set up to control the location of the slice through the flow, and the software would place a notch at the location of the trailing edge of the wing. The slider would then naturally fall into this location when it was brought nearby. This demonstrates the usefulness of the Notch mode to mark salient points in the variable being controlled, and make the salient points easier to reach. The button was set up to enable and disable the notch, so that other points in the vicinity could also be investigated.

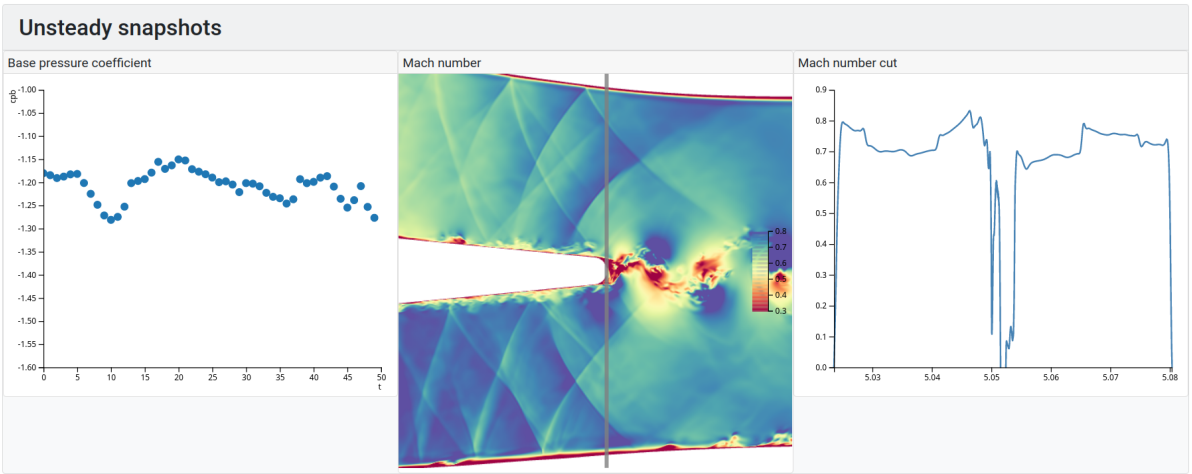
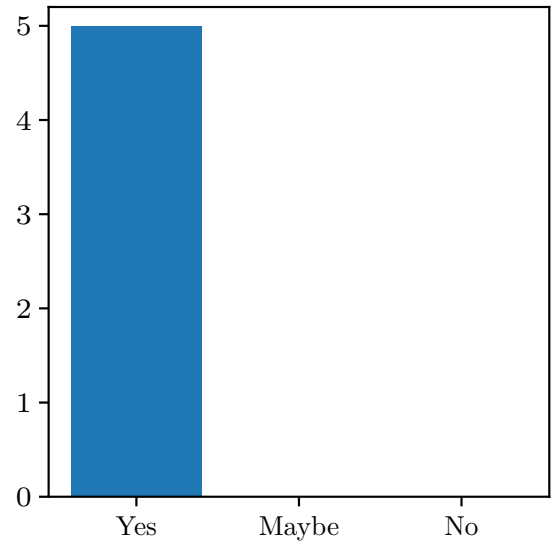


Figure 22: Unsteady flow slice demo

7.3 Survey and responses

In this section the survey and the responses to each question are summarised. Full responses from each participant can be seen in Appendix A.

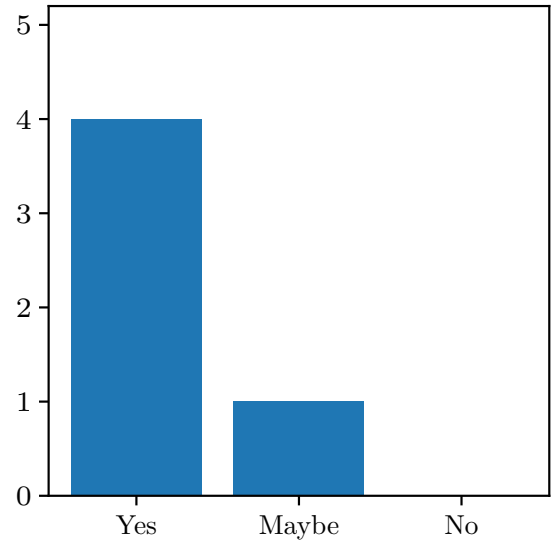
Q: Did you prefer using the prototype interaction device to using a touchscreen or mouse? Why?



Example response:

Yes, the interaction was much smoother, and more intuitive. Being able to change two parameters simultaneously made selecting the desired data much faster than having to click and drag using the mouse.

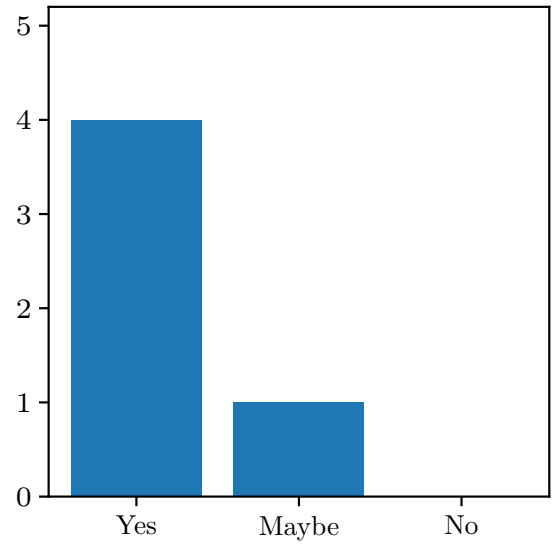
Q: Could you see yourself using an interaction device based on this prototype in your daily work? Please elaborate on what you envision, if applicable.



Example response:

I think the device would be a great way for guests/managers not well versed with the visualisation software to interact with the whole data set, as well as for the engineer to quickly process the data how they wish. Personally, I would probably use both this and a mouse, saving the mouse for more complex post-processing that cannot be done quickly with the device.

Q: Do you estimate that after becoming accustomed to such a device, you would be able to complete data analysis tasks faster, more comfortably, or otherwise better than with a touchscreen or mouse?



Example response (taken as Maybe):

Unsure. The overheads in setting up a tool in the specific way such as the demos for different applications may be larger than the gains in productivity. For a consistent application like DB slice then yes quite possibly.

Q: Do you have any suggestions as to what could improved in a future version of the device?

Some responses here related to adding functionality outside SLIDER's target tasks, such as for rotation or translation. The author believes this is a region of the task space where mice (or space mice, as one responder mentioned) are better suited.

There were a few comments that suggested that switches or other controls be added to navigate menus and/or otherwise multiplex the device. The author believes that this is a realm where keyboard control is more optimal, since regardless it requires the user to move their hand off of the main input of the device, keyboards are readily available, and it is a discrete task.

There was an interesting comment about the ergonomics of the device:

> The functionality of the device was very good, the ergonomics could be improved to make the device more comfortable to use, the controls could be lower to the table surface and the slider could have a little less resistance and reduced total range of movement

This would be a good topic to investigate in a second version. The height of the device was limited by the height of the motorised slider, but it would be possible to mount the motorised slider sideways, albeit with a more complicated bracket.

Q: Do you have any further comments?

An interesting comment here pointed out that it would perhaps have been more fair to set up mouse control to also control the histogram windows by center and width, using the scroll wheel to set the width. The demo instead had draggable edges for the histograms, as is the default in dbslice. This does point out an unforeseen limitation on the survey. Nonetheless, this is only a small part of what makes SLIDER much better than a mouse for 2D asymmetric control tasks, as is reflected in the other responses.

8 Conclusions

The key contributions and results of this project are as follows:

1. A novel theoretical description of the Human-Computer Interaction (HCI) interface choice problem was presented. This theory was used to identify that existing interaction systems in engineering simulation are not optimal.
2. A particular type of control task which is common in engineering simulation, but not well accommodated for in existing interaction systems, was identified and named: 2D joint asymmetric control.
3. A new interaction device, titled SLIDER: the Software-Labelled Interaction Device for Engineering Research, was designed and built to control 2D joint asymmetric tasks. Firmware was developed for the SLIDER device, including the development of five control modes for the motorised slider.
4. Web software demos were produced for the SLIDER device, including demos that integrate with an existing web-based visualisation framework.
5. A survey was conducted where five researchers in the Whittle lab compared using SLIDER to using existing interaction systems. Every participant preferred using SLIDER to existing systems, and most could see themselves using SLIDER in their daily work.
6. A website was produced for SLIDER, which can be seen at https://slider.clifford.lol. This includes an introduction to the project, a build guide, demos, specifications, and repositories for all source code and assets. All materials related to the project are contained here and released under free and open source licenses.

The findings of this project suggest a promising future for custom hardware interface devices in engineering simulation.

8.1 Future Work

The physical SLIDER device could be improved in a number of ways. The most important would be to fix a mechanical flaw in the prototype where the wires to the rotary encoder can occasionally be cut or frayed by repeated rubbing against sharp surfaces on the inside of the box. This is discussed in Section 4, along with a proposed modification to prevent it.

A future version might also include a second motorised slider, rotary encoder, and display. This is preferable to simply using a second SLIDER, because none of the other internal electronics would have to be duplicated, thus the overall cost would be much lower.

But certainly the most worthwhile future work would be to develop further software support to enable the use of SLIDER in real engineering simulation work. The author hopes that some of the readers will be motivated enough by this report to make it happen.

8.2 Using SLIDER in your projects

Everything required to build a SLIDER, including a parts list, build guide, and the full firmware, can be found at the project homepage: https://slider.clifford.lol . This website also contains links to code repositories for the full software demos, for you to use to integrate SLIDER in your own projects.

9 References

Iris Artin. Protothreads for arduino, 2020. URL https://gitlab.com/airbornemint/arduino-protothreads.

William Buxton. There's more to interaction than meets the eye: Some issues in manual input. *User centered system design: New perspectives on human-computer interaction*, 319:337, 1986.

Matthew Chalmers and Areti Galani. Seamful interweaving: heterogeneity in the theory and design of interactive systems. In *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques*, pages 243--252, 2004.

Stephen W Draper. The notion of task in HCI. In *INTERACT'93 and CHI'93 Conference Companion on Human Factors in Computing Systems*, pages 207--208, 1993.

Jonathan Grudin. Three faces of human-computer interaction. *IEEE Annals of the History of Computing*, 27(4):46--62, 2005.

Addie Johnson. Procedural memory and skill acquisition. *Handbook of psychology*, pages 499--523, 2003.

Scott R Klemmer, Björn Hartmann, and Leila Takayama. How bodies matter: five themes for interaction design. In *Proceedings of the 6th conference on Designing Interactive systems*, pages 140--149, 2006.

Saralees Nadarajah. A generalized normal distribution. *Journal of Applied statistics*, 32(7): 685--694, 2005.

Fernando Olivera, Manuel García-Herranz, Pablo A Haya, and Pablo Llinás. Do not disturb: Physical interfaces for parallel peripheral interactions. In *Human-Computer Interaction--INTERACT 2011: 13th IFIP TC 13 International Conference, Lisbon, Portugal, September 5-9, 2011, Proceedings, Part II 13*, pages 479--486. Springer, 2011.

Dan O'Sullivan and Tom Igoe. *Physical Computing: Sensing and Controlling the Physical World with Computers*. Course Technology Press, Boston, MA, USA, 2004. ISBN 159200346X.

Gary Perelman, Marcos Serrano, Mathieu Raynal, Celia Picard, Mustapha Derras, and Emmanuel Dubois. The roly-poly mouse: Designing a rolling input device unifying 2d and 3d interaction. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 327--336, 2015.

Graham Pullan. dbslice: Interactive, hierarchical, database-driven plotting, 2017. URL https://www.dbslice.org/.

Marcelo Mortensen Wanderley and Nicola Orio. Evaluation of input devices for musical expression: Borrowing tools from hci. *Computer Music Journal*, 26(3):62--76, 2002.

Appendices

A Full responses to survey

A.1 Responder 1

Did you prefer using the prototype interaction device to using a touchscreen or mouse? Why?

For the demos, the device was smoother and easier to select what you wanted to. I imagine with multiple/more software support, lots of/most tasks in visualisation could be improved with a similar device.

Could you see yourself using an interaction device based on this prototype in your daily work? Please elaborate on what you envision,

If programmable to adapt to other windows, it could be used with my interfaces/other programs.

Do you estimate that after becoming accustomed to such a device, you would be able to complete data analysis tasks faster, more comfortably, or otherwise better than with a touchscreen or mouse?

Yes, in many cases.

Do you have any suggestions as to what could improved in a future version of the device?

The 2D axes of motion work well, maybe a couple of extra switches to hand.

Do you have any further comments?

A.2 Responder 2

Did you prefer using the prototype interaction device to using a touchscreen or mouse? Why?

For the given applications it was very well designed and fit with the demos well. The latching and haptic feedback are an extra compared to the mouse.

Could you see yourself using an interaction device based on this prototype in your daily work? Please elaborate on what you envision,

I already use a 3D space mouse for CAD which provides better linkage between screen changes and input. Latching in the visualisation is useful. The device is likely too specific for most applications, in particular much of my work has more than two variables to control.

Do you estimate that after becoming accustomed to such a device, you would be able to complete data analysis tasks faster, more comfortably, or otherwise better than with a touchscreen or mouse?

Unsure. The overheads in setting up a tool in the specific way such as the demos for different applications may be larger than the gains in productivity. For a consistent application like DB slice then yes quite possibly.

Do you have any suggestions as to what could improved in a future version of the device?

Much like a space mouse (where 6dof are movable simultaneously) there could be applications to navigate space and time at the same time, this certainly speeds up searching for features.

Do you have any further comments?

Like the feedback, the latching and the snap back to position.

A.3 Responder 3

Did you prefer using the prototype interaction device to using a touchscreen or mouse? Why?

Yes, the interaction was much smoother, and more intuitive. Being able to change two parameters simultaneously made selecting the desired data much faster than having to click and drag using the mouse

Could you see yourself using an interaction device based on this prototype in your daily work? Please elaborate on what you envision,

Yes, possibly when looking at a large data set with many parameters and looking at the trends, although this may be improved with some additional controls on the device to allow simultaneous control of greater number of parameters and combinations thereof.

Do you estimate that after becoming accustomed to such a device, you would be able to complete data analysis tasks faster, more comfortably, or otherwise better than with a touchscreen or mouse?

Yes, assuming the current click and drag setup that is required for use with the mouse is unchanged

Do you have any suggestions as to what could improved in a future version of the device?

The functionality of the device was very good, the ergonomics could be improved to make the device more comfortable to use, the controls could be lower to the table surface and the slider could have a little less resistance and reduced total range of movement

Do you have any further comments?

Could this same functionality be achieved using just a mouse and some additional software rather than requiring a separate custom piece of hardware? For example when hovering over/clicking on a plot the scroll wheel does the same as the rotating knob and mouse horizontal movement the same as a the slider, some (like the Logitech master series) also have a secondary scroll wheel so wouldn't require the mouse movement input at all. It may or may not be as instantly intuitive but might be more convenient and become faster over time as you wouldn't have to switch to different control device

A.4 Responder 4

Did you prefer using the prototype interaction device to using a touchscreen or mouse? Why?

Yes, using the SLIDER felt intuitive.

Could you see yourself using an interaction device based on this prototype in your daily work? Please elaborate on what you envision,

If analysing large, visual datasets as shown in dbSlice.

Do you estimate that after becoming accustomed to such a device, you would be able to complete data analysis tasks faster, more comfortably, or otherwise better than with a touchscreen or mouse?

Yes. The movements were more intuitive than dragging the window with a mouse click.

Do you have any suggestions as to what could improved in a future version of the device?

The software side of the device could be implemented with dbSlice for everyone with a mouse to use. The mouse wheel could control the width of the window.

Do you have any further comments?

A.5 Responder 5

Did you prefer using the prototype interaction device to using a touchscreen or mouse? Why?

I preferred using the interaction device than a mouse when viewing the histograms. I found it very easy and intuative to use and interact with the dataset.

Could you see yourself using an interaction device based on this prototype in your daily work? Please elaborate on what you envision,

I think the device would be a great way for guests/managers not well versed with the visualisation software to interact with the whole data set, as well as for the engineer to quickly process the data how they wish. Personally, I would probably use both this and a mouse, saving the mouse for more complex post-processing that cannot be done quickly with the device.

Do you estimate that after becoming accustomed to such a device, you would be able to complete data analysis tasks faster, more comfortably, or otherwise better than with a touchscreen or mouse?

Probably, but it would take some time.

Do you have any suggestions as to what could improved in a future version of the device?

For visualising 3D surfaces/contours, I could imagine a 3D trackpad being integrated into the device to change the camera angle/position, with the current slider and dial providing the remaining camera controls. A way to navigate menus (say between different datasets to be visualised) could also be a useful addition.

Do you have any further comments?

The device seemed very polished and effective. Great project, well done! :)

B Risk Assessment Retrospective

The risk assessment submitted at the start of Michaelmas 2022 identified computer use as the sole risk of the project. One additional risk was encountered over the course of the project: soldering. A supplementary risk assessment was performed for this when it became apparent that it would be required. This could potentially have been identified at the start of the project, although at the time it was unclear whether it would be required.