A Novel Ecologically Valid Pain Behaviour Exploration and Potential Treatment Platform



Shuangyi Tong

St Catherine's College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Supervised by

Professor Timothy Denison and Professor Ben Seymour

Trinity 2024

Abstract

Chronic pain is a critical public health problem affecting a significant portion of the population. Pain, as an essential signal, possesses intrinsic properties that can only be effectively tested and demonstrated in realistic experiences, which are inadequately addressed in conventional pain research. This thesis aimed to resolve this issue by developing a novel pain research and treatment platform, focusing on improving ecological validity to more closely resemble real-life experiences. To demonstrate the potential of this platform, I designed a novel free-operant foraging task and conducted a series of experiments with various pain modalities. These experiments tested important hypotheses regarding pain modulation, including modulation by decision conflict and conditioned pain modulation, and demonstrated the selective modulation of motivational values by phasic pain. By the end of this thesis, I established a unified quantitative model that evaluates the effects of phasic and tonic pain within this generalizable foraging task. The platform has been further utilised by other researchers to study pain and tested for potential therapeutic applications. Overall, this novel platform, developed throughout my DPhil course, has demonstrated significant value for future pain research and treatment, with a strong focus on ecological validity.

Summary of Contributions

This thesis offers the following contributions:

- *Identifying factors contributing to low ecological validity in conventional pain research methods and illustrating the limitations caused by these factors.* Pain has intrinsic properties that can only be effectively tested and demonstrated in realistic experiences. Without simulating these experiences, it is difficult to apply the results to make accurate predictions in real life.
- *Proposing a novel pain research platform to enhance ecological validity.* I sought to address these limitations by identifying the key elements required to optimise ecological validity in pain research. Real-time motion tracking, simultaneous neurophysiological data monitoring, multimodal data integration, interactive model-based task control, portable experimental pain stimulation, and realistic visual graphics are all critical factors for an immersive experience that meets research demands.
- *Implementing the enhanced ecologically valid platform.* I utilised current Virtual Reality (VR) and sensor technology to develop a novel pain research platform.

With this new platform, I conducted a series of experiments and analysed the results to study human pain behaviour:

- *Testing pain modulation hypotheses.* I investigated whether pain is modulated by decision conflict in our novel free-operant foraging task. The positive results encouraged the development of more complex tasks based on this platform.
- *Identifying the phasic pain's effect that selectively modulates motivational values.* By introducing choices with variable phasic pain intensities, I demonstrated that motivational value and motivational vigour can be selectively modulated for pain and neutral choices in this free-operant decision-making foraging task.

- *Developing a novel computational model for an alternative objective pain measurement.* I created a computational model to measure phasic pain in the free-operant decision-making task. By analysing the physiological data, I showed that the phasic pain value estimated from the model predicts expected aversive value more accurately than subjective ratings. This suggests that the modelled phasic pain value could serve as an alternative tool for assessing phasic pain.

- *Comparing multiple aspects of pain modalities and their impact on human behaviour.* I conducted experiments with various pain modalities and demonstrated their impact on behaviour using our established behavioural metrics and model values.

- *Identifying the effect of tonic pain on behaviour.* By introducing new tonic pain stimulation into the task, I discovered that tonic pain has a significant effect on motivational vigour.

- *Identifying a potential behavioural marker for chronic pain progression.* I found that punishment sensitivity did not change in the tonic pain condition. However, previous studies have shown changes in punishment sensitivity in chronic pain patient groups. This finding supports that punishment sensitivity could serve as a behavioural marker for the progression of chronic pain.

- *Developing a novel algorithm and a computational model to describe free-operant behaviour accounting for both phasic and tonic pain's functions.*

In addition to the findings on pain behaviour mechanisms, the platform has further applications:

- *Creating a pain visualisation VR experience with physiological signal feedback, aiming to explore potential VR therapeutic applications.*

- *Allowing use by other researchers and clinicians to study different pain problems or to evaluate patients' conditions.*

Contents

1 Background and Motivation **1**

1.1 Complexity of pain 2

1.1.1 Basic pain neurophysiology 2

1.1.1.1 Ascending pathways 3

1.1.1.2 Descending control pathways 4

1.1.1.3 Neural representations of pain 4

1.1.2 Learning theories and pain 6

1.1.2.1 Reinforcement learning 6

1.1.2.2 A hierarchical pain system from the learning perspective 14

1.2 Limitations of research paradigm with low ecological validity 15

1.2.1 Lack of free-operant instrumental decision-making 16

1.2.2 Lack of motor actions 16

1.2.3 Lack of embodiment 17

1.2.4 Lack of realistic environment context 17

1.3 Proposed framework with enhanced ecological validity 17

2 Implementation of the platform **20**

2.1 An overview from the event-driven design principle 20

2.1.1 System Architecture 21

2.1.2 Control-side Algorithm 23

2.1.2.1 Event-driven task-specific control 23

2.1.2.2 Real-time data visualisation 23

2.1.2.3 Ad hoc control by researcher 24

2.1.3 Device-side Algorithm 24

2.1.3.1 High report data rate, no control data 25

2.1.3.2 Low report data rate, low control data rate 25

2.1.3.3 High report data rate, low control data rate 26

2.1.3.4 High report data rate, high control data rate 27

2.1.4 Event-driven control examples 27

2.1.4.1 Adaptive control by virtual events 27
2.1.4.2 Adaptive control by physiological response 28
2.1.4.3 Adaptive control by predictive models 28
2.2 Implementation details 29
2.2.1 The communication protocol 29
2.2.2 Control software implementation 32
2.2.2.1 Task-specific control script 32
2.2.2.2 Real-time visualisation 33
2.2.2.3 Researcher ad hoc control 33
2.2.3 Device software implementation 34
2.2.4 Integration with the hardware 34
2.2.4.1 Wireless physiological sensor system 35
2.2.4.2 Wireless thermal stimulator control 37
2.2.4.3 Electric pain stimulator control 37
2.2.4.4 EEG data collection and synchronisation 38
2.3 Data Analysis 40
2.3.1 Python library 40
2.3.2 Accelerated grid search model-fitting 40
2.4 The base task 41

3 Ecologically valid free-operant foraging study differentiates perceived pain in stationary and active contexts with unaffected vigour 43

3.1 Introduction 43
3.2 Methods 45
3.2.1 Study design 45
3.2.2 Experimental pain stimulation 46
3.2.3 Behavioural data processing and statistical tests 46
3.2.4 Physiological data 47
3.3 Results 47
3.3.1 Pain ratings 47
3.3.2 Electromyography 52

3.3.3 Dose-response effect on vigour . . . 53
3.4 Discussion . . . 55
3.4.1 Modulation by decision conflict . . . 55
3.4.2 EMG analysis challenges . . . 57
3.4.3 Unaffected vigour . . . 57
3.5 Conclusion . . . 58

4 Phasic pain modulates motivational values in a realistic free-operant foraging task quantified by a novel computational method 59

4.1 Introduction . . . 59
4.2 Methods . . . 60
4.2.1 Study design . . . 60
4.2.2 Experimental pain stimulation . . . 60
4.2.3 Skin conductance analysis . . . 61
4.2.3.1 Data collection processing . . . 61
4.2.3.2 Effect coefficient estimate . . . 61
4.2.4 Behavioural data and statistical analysis . . . 61
4.2.5 Computational model . . . 62
4.3 Results . . . 64
4.3.1 Phasic pain modulate motivational values for choices associated with different pain intensities . . . 64
4.3.2 Model fitting results and its correlation with physiological data . . . 66
4.3.3 Effect on vigour . . . 69
4.4 Discussion . . . 73
4.4.1 Phasic pain selectively modulates motivational values . . . 73
4.4.2 Interpretation of model parameters . . . 74
4.4.2.1 The motivational value estimated by the model . . . 74
4.4.2.2 Separated moving distance components for vigour cost estimation . . . 74
4.4.2.3 Lack of time factor in the free-operant computational model . . . 75

4.4.3 Non-zero bias . . . 75
4.5 Conclusion . . . 75

5 A comparison of different modalities of phasic pain's values and their impact on behaviour 76

5.1 Introduction . . . 76
5.2 Methods . . . 77
5.2.1 Study design . . . 77
5.2.2 Experimental pain stimulation . . . 78
5.2.2.1 Finger pain stimulation . . . 78
5.2.2.2 Back pain stimulation . . . 78
5.2.2.3 Wasp electrode stimulation . . . 78
5.2.2.4 Gel pad stimulation . . . 79
5.2.3 Behavioural data analysis and statistical tests . . . 80
5.3 Results . . . 80
5.3.1 Aversive choice probability . . . 80
5.3.2 Compare horizontal and vertical moving cost coefficients . . . 81
5.3.3 Compare pain utility function . . . 82
5.4 Discussion . . . 83
5.4.1 Demands for a large amount of data . . . 84
5.4.2 Ecological validity modulates decision values . . . 84

6 A unified model accounting for distinct behavioural functions of phasic and tonic pain in healthy individuals 85

6.1 Introduction . . . 85
6.1.1 Tonic pain's effect on motivational vigour . . . 85
6.1.2 Punishment sensitivity under the influence of tonic pain . . . 86
6.1.3 Conditioned pain modulation . . . 86
6.2 Methods . . . 87
6.2.1 Study design . . . 87
6.2.2 Experimental pain stimulation . . . 88

6.2.2.1 Electric phasic pain . . . 88
6.2.2.2 Tonic pain stimulation . . . 88
6.2.3 Statistical analysis . . . 88
6.2.4 Computational model . . . 89
6.2.5 EEG analysis . . . 90
6.2.5.1 Data collection and preprocessing . . . 90
6.2.5.2 ERP analysis . . . 90
6.2.5.3 Time frequency analysis . . . 90
6.2.5.4 Source analysis . . . 92
6.3 Results . . . 92
6.4 Discussion . . . 99
6.4.1 Distinct functions of tonic and phasic pain . . . 99
6.4.2 Punishment sensitivity changes . . . 101
6.4.3 Conditioned pain modulation and interaction between pain modalities . . . 103
6.4.4 Conclusion . . . 103

7 Applications and future directions 105

7.1 Applications . . . 105
7.1.1 Pain visualisation and neurofeedback therapy . . . 105
7.1.2 Collaborated projects based on this platform . . . 107
7.2 Future directions . . . 108
7.2.1 Caveats and Limitations . . . 108
7.2.1.1 Worse physiological signal qualities . . . 108
7.2.1.2 Complex environment . . . 109
7.2.1.3 Ergonomics and accessibility . . . 110
7.2.2 Future projects . . . 110
7.2.2.1 Phasic pain assessment . . . 110
7.2.2.2 Longitudinal chronic pain progression monitoring . . . 111
7.2.2.3 Therapeutic application . . . 111

8 References **113**

9 Appendix **129**

9.1 Specifications for the communication protocol 129

9.1.1 Register phase 129

9.1.2 Data report phase 131

9.1.3 Server control phase 131

9.2 Ethical approval 131

9.3 Justifications and safety considerations for tonic pain stimulation with pressure cuffs 132

9.4 Additional Figures 137

9.5 Core source code 137

9.5.1 Implementation core source 137

9.5.1.1 Wireless physiological sensor system 137

9.5.1.2 Wireless thermal stimulator control 140

9.5.1.3 Generic .NET implementation for device side application 144

9.5.1.4 Control software protocol implementation 150

9.5.2 Data Analysis 156

9.5.2.1 Data Analysis Framework 156

9.5.2.2 CUDA accelerated grid search 159

List of Figures

1 A simple diagram representing an agent interacts with the environment . . 7

2 A conceptual diagram of simple Markov decision process 7

3 Observed time preference. Reproduced from [1, Figure 2] 8

4 A schematic diagram for the proposed platform 19

5 Two data transmission diagram 22

6 Main views of the control-side program developed with Electron. **a**, Left panel: list of connected devices. Right panel: real-time visualisation of report data. **b**, Left panel: Task-specific control scripts available to choose from. Right panel: top, script management (activate and save), middle, script variable visualisation and control, bottom, script editor 24
7 Wireless physiological sensor system 36
8 Wireless thermal stimulator control 37
9 Stimulator setup 38
10 Data drop in EEG recording 39
12 Photo of the actual experiment 42
11 **a**, A illustrative figure depicting the foraging task. **b**, Participants' view (left eye) in the task. **c**, A perspective view of the task environment. **d**, An aggregated top-view heat map of head trajectories of all participants' data. Red dots denote baskets' preset locations. 42
13 Pre-block calibration pain ratings and pain ratings for the block at 100% stimulation intensity 48
14 Pre-block calibration pain ratings and pain ratings for the block at 75% stimulation intensity 49
15 Pre-block calibration pain ratings and pain ratings for the block at 50% stimulation intensity 50
16 Pre-block calibration pain ratings and pain ratings for the block at 25% stimulation intensity 51
17 Average pre-block calibration pain ratings and pain ratings 52
18 EMG activity when subject received a shock while picking up a fruit. Time 0 is the point when the fruit is picked up. 53
19 Pain intensity's dose-response effect on collection rate 54
20 Hand moving speed (Chapter 3) 55
21 Pain ratings and aversive choice probability 65
22 Pain ratings and distance bias 66
23 Model-fitting results (Chapter 4) 67

24 Fitted pain utility function compared with subjective ratings 68
25 Evoked SCR for fixation and fruit pick-up events 68
26 Skin conductance fitted coefficients' correlation with VAS ratings and modelled phasic pain value 69
27 Hand moving speed (Chapter 4) 70
28 Hand moving speed for non-painful choice 71
29 Hand moving speed for painful choice 71
30 Visualisation of instantaneous hand speeds at different normalised travelling distances towards non-painful choices 72
31 Visualisation of instantaneous hand speeds at different normalised travelling distances towards painful choices 72
32 Aversive choice probability for pain stimulations with different modalities 80
33 Horizontal and vertical moving cost coefficients comparison in finger pain condition 81
34 Horizontal and vertical moving cost coefficients comparison in back pain condition 82
35 Pain utility function for wasp electrode 82
36 Pain utility function for gel pad electrode 83
37 Motivational vigour comparison for different tonic pain conditions 93
38 Phasic pain ratings under the tonic pain's influence 94
39 Aversive choice probability under the tonic pain's influence 95
41 Tonic pain frequency band power 95
40 Phasic pain's ERP under tonic pain's influence 96
42 Model-based tonic pain frequency band power correlation (0.5 second before decision point) 97
43 Model-based tonic pain frequency band power correlation (0.5 second after decision point) 98
44 Motivational vigour fitting 98
45 Phasic pain fitting 99
46 Hand moving speed for reaching non-painful choices (Chapter 6) 100

47 Separate vigour constants fitting 101
48 Separate vigour constants fitting (based on Chapter 4's data) 102
49 The visual representation of body state 106
50 Reproduced from [2, Figure 4] (licensed under CC BY) 107
51 Reproduced from [3, Figure 1](licensed under CC-BY 4.0) 108
52 Single one-way transmission data layout for both device side and server side 129
53 Source analysis for tonic pain frequency band power 137

List of Tables

1 Comparison of two data transmission type supported by the protocol . . . 22
2 Currently supported devices running on our communication protocol . . . 31
3 The descriptor of a device used to register on the server 130
4 Visual report style suggestions 130
5 Visual control input style 130
6 Mutually supported data types 131
7 Comparison of studies using pressure cuffs for experimental pain stimulations 135

1 Background and Motivation

Chronic pain is a worldwide, prevalent health problem. Research shows that it affects approximately 35.0% to 51.3% of adult residents in the UK, and a considerable number of cases lead to disabilities [4]. This prevalent health condition generates substantial socio-economic costs. However, many forms of chronic pain lack effective treatment options. While pharmacologists continue to search for effective, non-addictive drugs for chronic pain, the opioid epidemic, caused by prescription overdoses in the United States and other countries, raises questions about whether pharmacological methods are a promising approach to treating chronic pain [5]. The reasons for the failure of traditional opioid treatment are complex, spanning different levels from molecular biochemistry at micro level to cognitive psychology at macro level. This complexity drives the need to explore non-pharmaceutical approaches.

Seymour [6] identifies at least four types of mechanisms that contribute to endogenous modulation: modulation by sensory inference, modulation by predictive value, modulation by decision conflict, and modulation by information value. These mechanisms exhibit a form of learning—an evolving process that updates itself to adapt to the environment based on new information. For example, the modulation by sensory inference process can be computationally modelled as a Bayesian inference [6, 7, 8]. To enhance the performance of these systems during learning, additional and comprehensive new information from the real environment is essential.

From this learning perspective, it can be hypothesised that certain types of pain disorders resulting from malfunctioned learning should be addressed through re-learning, incorporating a high volume of information from real-life experiences. However, traditional pharmaceutical approaches or basic rehabilitation exercises may encounter an information bottleneck by providing only limited pain-relevant information to the patients' pain processing systems. The complexity of pain processing, combined with the necessity of aversive learning to access rich information from real-life experiences, underscores the importance of studying or treating pain in a context that closely resembles real life—or, more broadly, in an ecologically valid context [9].

The complex mechanism of this endogenous control and the pain learning systems poses a challenging biomedical engineering problem for developing new technologies to treat chronic pain. Specifically, we aim to simulate a context with high ecological validity while ensuring high-resolution data monitoring and robust control of the environment and stimulation feedback. To investigate how various internal and external signals interact with the pain system, we need to leverage advances in biomedical technology, build a novel framework, and conduct experiments carefully. With a deeper understanding of the pain system, this could provide vital guidance for implementing an interactive, closed-loop treatment system, which is the direction of much current biomedical engineering research [10, 11]. In this chapter, we review the complexity of pain systems, list limitations in the conventional pain research paradigm, and propose contributions to a new, high-level, ecologically valid research system design.

1.1 Complexity of pain

To illustrate the complexity of pain, we begin with a review of pain neurophysiology and then discuss how pain is understood at the computational level, specifically pain as a learning signal guiding the control of behaviour.

1.1.1 Basic pain neurophysiology

Pain is an unpleasant sensation that is common in daily life. Its aversiveness naturally repels creatures from engaging in activities that cause pain. Although pain might simply be described as something people want to avoid, there is considerable complexity and high diversity in its perceived intensity and characteristics. The rich vocabulary used in the renowned McGill Pain Questionnaire [12] illustrates, at the very least, how diversely pain can be expressed verbally using a combination of adjectives. Modern physiological evidence also confirms and further recognises pain's complexity by exploring neural coding of pain [13]. Clinically, despite advanced medical imaging technology, the complicated nature of pain still poses great difficulties in correctly diagnosing and evaluating a chronic pain patient's condition, estimating the prognosis, and providing appropriate treatments. This inadequate understanding of pain mechanisms contributes to the opioid epidemic

caused by prescription-opioid overdoses in the United States and other countries [5], and it urges us to learn more about pain to reduce socio-economic costs, stop suffering, and save lives.

1.1.1.1 Ascending pathways

Impulses generated by various types of nociceptors for different stimuli and other primary afferents are passed into different laminae of the spinal cord. These cells innervate with primary afferents that vary in conducting velocities due to myelination and their diameter. This difference gives rise to an intriguing phenomenon known as the double pain sensation, where the first sensation is transmitted by the fast-conducting Aδ fibres, followed by a second sensation carried by the slower C fibres [14]. Besides nociceptors that specifically respond to pain (nociceptive-specific cells, NS cells), there is a type of intriguing nociceptors called wide dynamic range (WDR) cells. One can experience non-painful sensations like a 'slow brush' when these cells fire below the painful threshold, but it becomes painful when they fire more frequently [15]. Evidence shows that an NS cell can change to behave like a WDR cell during a peripheral nerve injury, and this could support the role of WDR cells in pain disorders including allodynia and hyperalgesia [16].

Two major ascending projections transmit these signals for further processing: the spinothalamic tract and the spinobulbar tract. Craig wrote that the origin cells of these two tracts overlap little [17]. The thalamic region is one of the major sites for integrating signals from the spinal cord and relaying processed signals to other brain sites. There is a high chance of developing thalamic pain after an injury to the thalamus [18]. Spinobulbar projections terminate at various sites in the brainstem, including homeostatic integration sites [19]. These tracts not only transmit pain but also carry temperature, itch, and other homeostatic sensations. Stimulating the spinothalamic tract commonly produces sensations of warmth, cold, pain, and paraesthesia [20]. Cortical regions closely related to emotions include the anterior cingulate cortex (ACC) and insular cortex. This common pathway provides the physiological evidence for viewing pain as a homeostatic emotion proposed by Craig [21].

1.1.1.2 Descending control pathways

Descending control pathways of pain have been known for a long time, and both suppression and potentiation of pain pathways exist [22]. The midbrain, pons, medulla, and hypothalamus are involved in the descending control of nociception [23, 24]. A well-studied mechanism is from the midbrain periaqueductal grey (PAG) to the rostral ventromedial medulla (RVM) and down to the dorsal horn laminae, which are the sites of origin cells for many nociceptive ascending pathways [25]. It is important to note that neurons in the dorsal horns exhibit considerable plasticity, and descending control can alter their activities. This results in a gradual behavioural change of existing reflexes to more desirable configurations [26].

Many cortical regions have connections to the PAG, including the anterior cingulate cortex (ACC), amygdala, and medial prefrontal cortex [25]. It is not fully understood how these regions collectively contribute to a modulation signal for pain, and both cognitive decision-making and emotion involve processing in these regions. This enigmatic integrating process becomes more intriguing given the evidence that descending control can be selective [24, 27]. Therefore, understanding precisely how these signals are computed and executed remains a vital question to understand pain [6].

1.1.1.3 Neural representations of pain

The thalamus was previously thought to be the sole site responsible for perceived pain. Today, however, it is acknowledged that many cortical regions are also involved in pain processing [28]. Wilder Penfield's renowned work illustrated a somatotopic map that projected the body surface onto the somatosensory cortex by directly stimulating somatosensory regions. However, locating a specific 'pain cortex' proved to be a challenging task. Despite over 20 years of searching, Penfield did not identify such a cortical region [29]. Consequently, researchers now believe that there is no monopolistic cortical region dedicated to pain. The cortical regions involved in pain, often collectively referred to as the 'pain matrix', are frequently discussed in the literature.

Reviews on pain representations often revisit the historical debate between specificity and

patterned activity. Following the introduction of the gate control theory of pain and the discovery of wide dynamic range cells—cells that respond not only to painful stimuli—the notion of central specificity coding for pain has waned [19]. The phenomenon where a group of neurons codes different types of signals by segregating its input space is not uncommon. Lankarany et al. [30] demonstrated that tactile information can be encoded differently for high-contrast and low-contrast features, using the term 'neural multiplexing' to describe this phenomenon, which closely parallels the multiplexing concept in telecommunications and computer networks. Nonetheless, nociception-specific neurons, which exhibit high activity under painful stimuli, are still identified in the insula [31], cingulate cortex [32], and somatosensory cortex [33]. These nociceptive neuron populations, scattered across the cortices, support the concept of the 'pain matrix'.

It has long been recognised that a complete pain experience is multidimensional. In 1968, Melzack and Casey [34] proposed a conceptual model for pain consisting of sensory-discriminative, motivational-affective, and cognitive dimensions. Lesion studies also support the existence of these components [35]. The term 'asymbolia' refers to patients who perceive pain but seem indifferent to it. Berthier et al. [36] observed that these patients demonstrated a lack of withdrawal and absent or inadequate emotional responses to painful stimuli, indicating the involvement of the insular cortex in motivational-affective processing. On the other hand, damage limited to the sensory-discriminative component was observed in a patient who, despite being unable to specify the exact location and quality of a painful cutaneous laser stimulation on his left hand, experienced unpleasant sensations [37].

All evidence suggests that a complete pain experience is sophisticated, requiring precise temporal activities at the appropriate locations within the central nervous system. Modern neuroimaging methods indicate that models for pain representation in the brain need account for both spatial and temporal resolutions [38]. Furthermore, individual differences contribute to the volume of data requiring analysis. The vast amount of physiological data correlated with pain challenges the belief that current statistical methods can fully unravel all mechanisms employed by the pain system. This reminded Kucyi and Davis [13] of Jonas and Kording's insightful experiment in reverse engineering a microprocessor's

architecture using popular statistical methods in neuroscience [39]. Jonas and Kording concluded that it was difficult to reverse engineer the microprocessor's architecture and retrieve the original blueprint featuring a hierarchical design using these methods alone. Clearly, neural systems are not microprocessors, and neuroscientists do not rely solely on empirical statistics. Enhanced neural recording technology will undoubtedly aid our understanding of pain coding, promising increasingly precise physiological models for pain.

1.1.2 Learning theories and pain

Besides directly studying neurophysiological signals correlated with pain, it is beneficial to extract additional insights into pain from a learning theory perspective. The concept of learning in relation to pain encompasses not only the use of pain as a signal to guide task control (i.e., avoiding danger) but also refers to an internal learning process within the pain system itself. In this section, we will first explore basic reinforcement learning theory, where pain is typically regarded as an external, negatively-valued reward. Subsequently, we will discuss the hierarchical structure of pain processing and consider how pain is learned at different levels of the behaviour control system.

1.1.2.1 Reinforcement learning

Markov decision processes

Reinforcement learning is a common modelling approach to describe animals' acquired behaviour. Despite it is popular to use reinforcement learning to solve difficult computer science problems, the early history of reinforcement learning has a branch focused on elucidating animal learning processes. In reinforcement learning, an agent interacts with its environment over time and receives feedback. The agent learns from this feedback and updates its policy to maximise a designated value [40]. Sutton and Barto [40] identified the two most important characteristics of reinforcement learning: 'trial-and-error search' and 'delayed reward'. Trial-and-error implies that there exists a direct value to be maximised or minimised. Thus, an algorithm that is suitable for solving reinforcement learning problems naturally aligns with certain optimal control problems. Delayed reward addresses a

more fundamental issue, namely the credit assignment problem in reinforcement learning: how to allocate credit to a sequence of actions taken before receiving a reward.

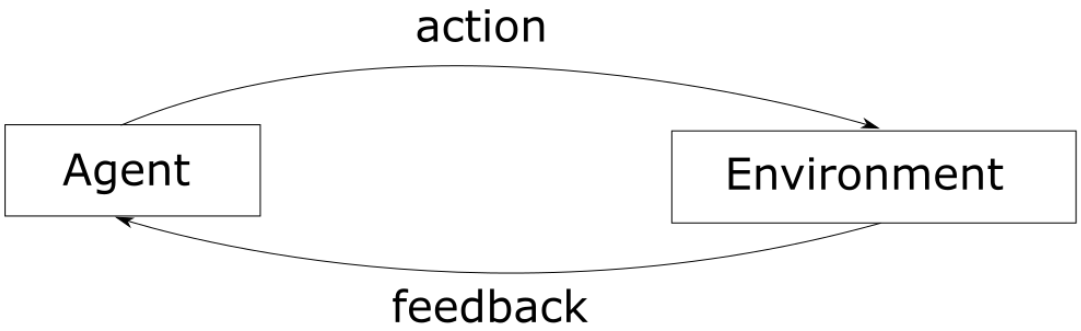


Figure 1: A simple diagram representing an agent interacts with the environment

Reinforcement learning problems are typically formalised by introducing Markov decision processes as the environmental model. The fundamental elements of a Markov decision process include states, actions, and transition probabilities. In a discrete-time Markov decision process, an agent takes an action and transitions to a subsequent state according to a defined transition probability, whilst receiving a reward.

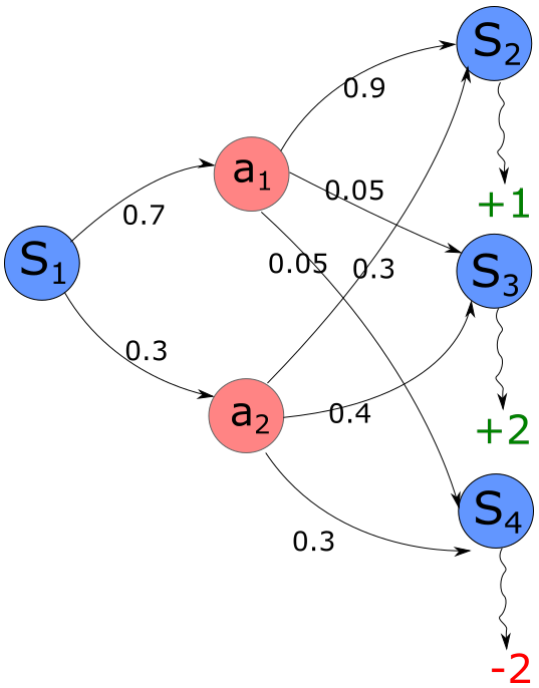


Figure 2: A conceptual diagram of simple Markov decision process

Reinforcement learning problems aim to maximize cumulative rewards. In a discrete-time Markov decision process with finitely many actions available at each state, and which is

guaranteed to terminate, the cumulative reward generated by following a policy must be bounded. However, it is common for tasks to lack a clear termination point, such as the classical cart-pole balancing problem [40]. In a non-terminating task, the reward may not be bounded, rendering the problem ill-defined. By introducing temporal discounting, which devalues future rewards in favour of immediate rewards, we can ensure the convergence of the cumulative reward value (provided all rewards are bounded by a maximum value and the discounting factor γ satisfies $0 \leq \gamma < 1$) [40]. Using discounting is not only a mathematical necessity but also reflects a commonly observed phenomenon in human decision-making, where immediate rewards are generally preferred over distant ones [41]. Similar effects occur with pain, but an additional factor known as 'dread' comes into play, which describes a preference for sooner rather than later pain. Ultimately, the temporal aspect of pain can be described by an 'N' shaped function, with maximum amplification occurring in the middle [1, 6].

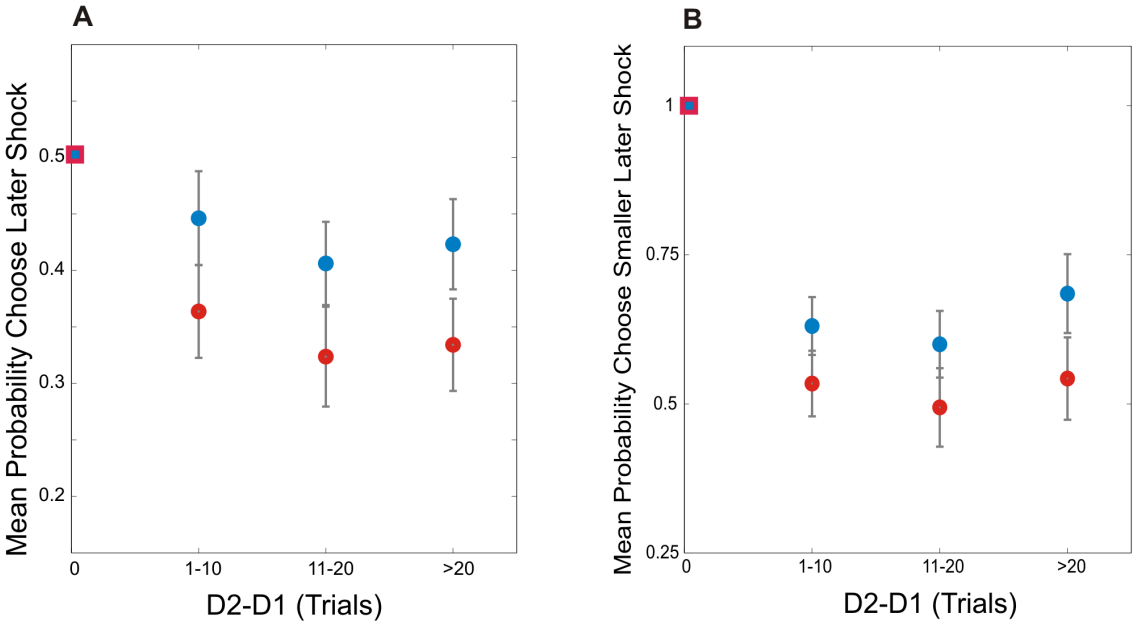


Figure 3: Observed time preference. Reproduced from [1, Figure 2] (licensed under CC BY 3.0) Compared to delayed shocks delivered in 1-10 or more than 20 trials, subjects tend to choose sooner pain if delayed shocks are delivered between 11-20 trials afterwards. This gives a "U" shape preference of sooner pain, which shows a "n" shape of delayed pain temporal factor.

The discrete-time Markov decision process with finitely many actions has addressed a

wide range of problems and is relatively simple to manipulate and prove the optimality of algorithms. However, when dealing with more realistic problems, algorithms must be adapted. To apply reinforcement learning algorithms to model the dynamics of pain, models that accommodate an environment with continuous time, continuous action space, and high-dimensional state space are likely required. To represent a vast state space, approximation methods are introduced [40], and state approximation using deep neural networks has facilitated breakthroughs in solving traditionally challenging problems [42]. By applying gradient descent directly to the policy function, it is possible to produce continuous actions, as opposed to the discrete actions generated from a value function [40]. Extending to the continuous-time case can be achieved through ad-hoc temporal discretisation. Although rigorously defined continuous-time Markov decision processes exist, they require careful analytical techniques [43]. Most applications of reinforcement learning opt for temporal discretisation with a reasonably high sampling rate.

Additionally, it is noteworthy that many human actions are executed in a hierarchical manner. For example, behavioural evidence suggests that there are at least two levels of action control in typing: the outer loop decides which word to type, while the inner loop manages the execution of specific motor actions fluently [44]. This analogy aligns with hierarchical reinforcement learning, where the environment model is a semi-Markov decision process. This model allows for two decisions separated by a wait time, which is a random variable. Actions then correspond to different wait time distributions depending on their complexity [45].

Objective functions

Objective functions or reward signals are crucial for reinforcement learning, guiding the agent to explore the environment and update policies to maximise the reward. This contrasts with supervised learning, which immediately informs the agent whether the action taken was correct [40]. Selecting an objective function in a specific task setting, such as achieving a high score in an Atari game, is straightforward. However, real-life environments and objectives are complex. It is implausible to assume that a single numerical value can guide all forms of human behaviour. Within certain contexts, pain may be considered

a punishment signal to minimize. Nonetheless, from an evolutionary perspective, the ultimate goal of human behaviour is not merely to minimize pain but to maximize survival rates. A corollary from this hypothesis is that identical pain stimuli may signify different levels of threats in cognitively perceived dangerous environments compared to safe ones, necessitating endogenous modulation of pain. Therefore, using cumulative pain values directly as an objective function for an individual's entire decision-making process may not adequately capture the nature of pain or serve as an effective objective function. This argument may even be initially ill-formed, as it is unclear whether a ground truth pain value ever exists. The same applies to reward, as both are subjective experiences. However, we are not suggesting that minimizing pain can never serve as an objective function. Evidence in avoidance conditioning supports the view that pain is what Pavlovian learning aims to minimize [2]. Pain is a significant learning signal for many decision processes in the brain, but it is not the objective function for the entire central nervous system. This subtlety could account for many perplexing chronic pain conditions.

Nevertheless, it is common to apply a negative reward value as punishment in training a reinforcement learning agent. In real life, an inevitable issue is an agent has to protect itself from harm or more generally stay within safety constraints. Navarro-Guerrero et al. [46] trained a robotic arm with negative reward signals when joint angles exceed a designated safe range. They showed that the robot arm trained with punishment successfully restricted the joint angles to the safe range while completing the task. It is also a major topic for safe reinforcement learning, and there are many approaches proposed to implement this safety mechanism for robots [47]. Many of those approaches are analogous to animals' pain systems. Although there are huge differences between negative signals used in those approaches and animals' pain, whether to refer to those signals as pain to left for a philosophiscal discussion.

Average reward reinforcement learning

The discussion thus far has primarily focused on reinforcement learning with a discount factor. Clearly, if there is no cycle in a finite Markov decision process, then the cumulative reward remains bounded. Consequently, a discount factor is not necessary to ensure a

bounded reward value. An alternative solution uses the average reward as the objective function. It has been demonstrated that in a cyclic task, reinforcement learning using the average reward converges to the optimal solution faster than the conventional discounted reinforcement learning [48].

While there is considerable theoretical discussion regarding whether using average reward reinforcement learning is preferable [40, 49, 50, 51], our application of average reward reinforcement learning in pain research focuses solely on the notion of average reward as an optimisation goal. There are two principal questions in modelling animal behaviour: what to do and when to do it. Decision-making neuroscience has devoted substantial attention to the first question but has been more limited in addressing the second [52]. This second question can be rephrased to inquire how to model animals' vigour. Niv et al. [53] proposed that, using average reward as the objective function, the resulting model can be used to describe changes in vigour, where, in this formulation, vigour is the time delay between two actions observed in animal studies. The average reward is used to calculate the opportunity cost, which increases with time, and it trades off against another vigour cost term, which decreases over time. This trade-off determines the optimal time delay between two consecutive actions and aligns with experimental data showing a relationship between the average reward animals receive and the time delay between their actions. Although no direct physiological evidence has been shown yet, it is believed that tonic dopamine encodes this average reward signal [54]. We will demonstrate in Chapter 6 that tonic pain significantly affects individuals' vigour.

Learning rates and one-shot learning

Reinforcement learning algorithms typically involve incremental learning, with a marginal learning rate taking small steps at each update. This approach enables the learning algorithm to fine-tune itself to an optimal set of parameters without overshooting. However, repeated trials are not always feasible in the real world. From a computational perspective, learning rates are crucial in addressing this issue. Lee et al. [55] investigated the mechanism of switching between incremental (low learning rate) and one-shot learning (high learning rate), finding that causal uncertainty mediates the learning rate.

Fast learning may be more critical in aversive learning, where repeated harm is particularly dangerous and often impractical. Seymour [6] suggested that phasic pain should be enhanced if the opportunity for learning is significant. Enhancing the pain signal is effectively the same as increasing the learning rate in many reinforcement learning models. Taking the temporal difference algorithm as an example, value function is updated as: $V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$ [40, equation 6.2]. Assume the expected pain is lower than the pain stimulus, $|R_{t+1} + \gamma V(S_{t+1})| > |V(S_t)|$. If we consider pain as negatively valued reward, then $R_{t+1} + \gamma V(S_{t+1}) - V(S_t) < 0$. α is the learning rate, then $\alpha > 0$. Since $R_{t+1} < 0$, then increasing the pain (decreasing the numerical value of R) is effectively the same as increasing the learning rate α.

Although boosting learning rates is one approach towards one-shot learning, more efficient yet restrictive fast learning in animals likely stems from hard-wired heuristics. Bolles [56] highlighted the significant discrepancy of difficulties in training rats to avoid shocks by pressing a lever versus escaping in a one-way shuttle box. It indicates there is an innate action hinted in this situation and a repertoire for possible innate actions available to be paired with [57]. Besides primitive heuristics, more complicated cognitive abilities can contribute to one-shot learning in a more complex environment by making predictions, which is a type of model-based learning in the reinforcement learning framework.

The credit assignment problem

The credit assignment problem can refer to different specific problems that all have to do with assigning "credits" to an entity involved in a dynamic process: a cue [58], an action, an agent, or an object within a decision process. Assigning credits to actions that receive delayed reward is the central problem in many reinforcement learning algorithms. In 1984, Sutton in his dissertation [59] refers to this specific assignment problem as temporal credit assignment and others as structural credit assignment. As pain is a learning signal, its temporal credit is likely resolved by learning systems in the brain [6].

Structural credit assignment can be of interest in multi-agent reinforcement learning and hierarchical reinforcement learning. In some multi-agent problems, agents are completely independent in taking actions. Credit assignment in this kind of multi-agent problem can

be reduced to temporal credit assignment by pairing each action taken by one of the agents with an individual state [60]. There are also examples with multiple separate learning algorithms in a single agent and an arbitration process to generate the final action decision. Lee et al. [61] proposed an arbitration mechanism between model-based and model-free reinforcement learning models in human decision making. Their arbitration model evaluates model-based and model-free models' performance separately with biophysical considerations to reach a unified decision.

A hierarchical decision model might be more relevant to pain research. Hierarchical reinforcement learning, although rigorous formulations may differ, usually introduces a notion of "macro decision" [45]. In order to grasp the power of hierarchical learning, one straightforward way is to learn an individual "macro" by another agent, the low-level one in the hierarchy. These small tasks are referred to as auxiliary tasks [40]. As we have seen the hierarchical control for typing in humans [44] from the previous section, it is common to have analogous auxiliary tasks in animals' motor control. Training with auxiliary tasks is a powerful tool in developing better robotics control algorithms, and from DeepMind's research trajectories, auxiliary tasks might be necessary if the algorithm is developed under the reinforcement learning framework [62, 63, 64, 65]. One of the problems of this is designing an auxiliary task's reward function is not trivial. Animals can rely on inductive bias shaped by long-time evolution [66], but at this point, reward functions for auxiliary tasks are handcrafted. It is hard to believe there exists a simple learning algorithm for all.

Structural credit assignment is also relevant for pain. Given a motor action is executed in a hierarchical order, picking up an object consists of a decision to pick up the object and another action control sequence instructing specific muscles to complete the action. Imagining an unexpected pain occurs during the course of action and given there is a chance of getting pain for choosing to pick up the object, how does the brain infer which decision in the hierarchy causes pain? It may not be a difficult problem for a healthy individual because of their intact visio-motor coordination, but it could be a different case for someone who has relatively weak muscle control. Pain signals could also interfere with the action execution. This problem will be developed further in the following sections, and in Chapter 3, we will show our results regarding whether pain is modulated when there is

a conflict between motor action decisions at different levels.

1.1.2.2 A hierarchical pain system from the learning perspective

In previous sections, we have seen the physiological hierarchical structure and some points related to hierarchical pain processing derived from reinforcement learning theories. In this section, we will review hypotheses for why pain is processed in a hierarchical manner.

Biological response time constraints

Innate responses are fast, sophisticated, and relatively hard-wired with limited learning capabilities. They are driven by spinal reflexes with precise connections from afferent inputs to specific motor neurons delivering timely motor responses that can save lives in an emergency. There are other types of learning in the action control hierarchy and the most complicated one is probably a model-based learner [6]. Model-based learner involves multiple regions in the brain and takes more time than a spinal reflex to decide an action. A model-based learner provides long-term predictions and makes better decisions, but it does not act as fast as simpler decision models. Therefore, it is necessary to have both high-latency yet more powerful model-based learning and low-latency primitive innate actions to account for certain time-sensitive tasks.

Models capture different aspects and objectives of a task

As an extension to the above argument, it is important to note not only action decisions are based on different models, but also sensory inputs are processed in a hierarchy and supplied to different levels of action models [67, 68]. The high-level model-based model usually uses convergent sensory inputs and makes decisions with more information including those that are less relevant. Extra complexity could be a problem in a specific context due to the bias-variance dilemma [69, 70]. Thus, it is helpful to keep both complex and simple learning models, and it is another problem to switch between them smartly [70].

Higher-level models might be optimised for different objectives. During the course of human evolution, a new subsystem within the hierarchy emerged, perhaps, for a distinct evolutionary reason. These subsystems were built on top of previous pain circuits and their

objectives vary. The diversity of such objectives can be seen from experiments showing that empathy for pain, a function for social collaboration, does not involve sensory components of pain [71].

Path dependence and irreversibility

Even though we have argued the advantages of keeping different learning systems organised in a hierarchy, it may not be the only way to achieve the same level of animals' motor skills. Evolution has a path-dependent property, and this property makes creature traits hard to reverse [72]. Hence, a hypothesis would be there are some mechanisms related to pain processing that are 'outdated', instead of deleting it, a modulation circuit could be introduced to coordinate and accommodate those 'outdated' mechanisms to coexist with newly evolved neural processes.

1.2 Limitations of research paradigm with low ecological validity

In this section, we will outline several limitations inherent in paradigms with low ecological validity. Developing tasks with high ecological validity requires state-of-the-art technology. Consequently, examples of low ecological validity are prevalent in earlier research paradigms. A typical example is quantitative sensory testing for pain. This testing generally involves administering quantifiable sensory stimuli to participants in stationary settings, where participants provide simple descriptive responses. Such testing procedures do not replicate the real-life context in which pain originates, nor do they allow participants to react to the pain and act optimally according to the pain signal. It is unquestionable that quantitative sensory testing is convenient and sometimes effective to show basic sensory characteristics. It is also used to predict outcome of chronic pain patients, although the results are inconclusive [73]. Despite its low ecological validity, quantitative sensory testing retains significant value in both research and clinical settings. Therefore, while we advocate for the use of paradigms with higher ecological validity in this section, it is important to recognise the importance of these conventional paradigms in various contexts.

1.2.1 Lack of free-operant instrumental decision-making

As discussed in the previous section, pain serves as a learning signal, indicating both the imminent danger (as a cue) and that damage has occurred (as an outcome). This value can be modulated to drive optimal control of behaviour. Therefore, it is crucial to consider pain processing in decision-making tasks. People routinely overcome pain to perform various actions, which are goal-directed and require strong motivational vigour driven by cognition [74]. While it is feasible to design a trial-based study where participants can choose to trade off between pain and some cognitive rewards, as they do in daily life, such a trial-based task may not accurately reflect real-world scenarios compared to a free-operant task where motivational vigour plays a key role [54]. We will show in Chapter 6 that free-operant task is necessary to observe a vigour effect of tonic pain.

1.2.2 Lack of motor actions

In previous discussions, we have illustrated that biological response time constraints could be one of the main reasons for having a hierarchical decision model. Fine motor actions often require precise control with high temporal resolution. At the spinal level, pain immediately alters motor execution via the nociceptive withdrawal reflex, which is modulated by analgesics affecting the threshold activating this reflex [75]. To observe these effects of pain, it is essential to allow participants to move freely. Additionally, pain naturally involves cognitive-motor interactions. A cognitive decision to execute a painful motor action might induce endogenous analgesia, referred to as pain modulation by decision conflict. We will explore this hypothesis in greater detail in Chapter 3.

In rehabilitation methods for chronic pain patients, physical exercises are often recommended. From a computational perspective, by physically moving a body part, the motor control system can infer the state of the injured site and gather more information about its condition. There are numerous factors that may make exercises beneficial for chronic pain patients [76]. Although many of these mechanisms are not well understood, further study of these effects requires a paradigm that incorporates a significant amount of physical motor activity.

1.2.3 Lack of embodiment

Embodiment in an experimental setting refers to the phenomenon where participants perceive objects presented in the experiment as extensions of their own body state. Pain is intricately linked to embodiment, as distinguishing the origin of the threat is critical. Research demonstrates that human subjects learn to differentiate whether pain originates from external or internal sources [77]. This distinction is also connected to the concept of peripersonal spaces [78], where embodiment plays a crucial role in defining the boundaries of these spaces.

Furthermore, the spatial and temporal properties of different types of pain significantly affect response actions. Changes in the location of pain stimulation induce sophisticated reflex trajectories and alterations in muscle activation [79, 80]. In Chapter 5, we will discuss how the location of pain stimulation also influences decision values.

1.2.4 Lack of realistic environment context

To demonstrate a clear and significant effect, conventional psychology experiments often control as many variables as possible. While controlling variables is crucial, an experimental environment with an overly simplistic setup can undermine the fidelity of simulating real-world experiences. Pain, as a signal indicating danger and prompting avoidance behaviours, is modulated depending on the context [81, 82]. In a well-controlled study, incorporating a realistic task enhances the resemblance to real-life scenarios, thereby generating results that are closer to actual settings.

1.3 Proposed framework with enhanced ecological validity

We have addressed the limitations inherent in setups with low ecological validity. Current research has begun exploring new methods that mitigate these limitations. For instance, gait analysis conducted while administering pain stimulation, which heavily involves motor actions, addresses the execution of tasks under pain conditions [80]. Studies using an embodied virtual hand have shown that virtual water can modulate heat pain [83]. However, to the best of our knowledge, no research has yet attempted to address all these

limitations simultaneously. In this section, we propose a framework for pain research and potential treatments with enhanced ecological validity, aiming to overcome all the limitations mentioned in the previous sections.

Virtual reality technology simulating realistic environment

Utilising virtual reality technology is crucial to create an immersive experience and generate realistic virtual environments. With advances in computer graphics and display technology, participants or patients can experience and seamlessly switch between realistic virtual environments. This capability enhances the fidelity of simulations used in experimental setups, making them more representative of real-world conditions.

High precision motion capture

Another key component in a virtual reality experimental suite is the motion capture system. To generate an embodied experience, the motion in the physical world should match the visual feedback in the virtual world. By capturing participants' physical movements and integrating them into the virtual environment with low latency, the matched visual input provides a strong sense of embodiment. It is then natural to include motor actions in the task design. Additionally, motion capture records critical motion data for various analyses relevant to pain, such as gait analysis.

Rich physiological sensors

Many of the limitations discussed above arise due to the requirements of certain physiological sensors. For instance, it is challenging to conduct tasks involving motor actions within an MRI scanner. Physiological data are crucial for pain research, ranging from simple heart rate sensors to more complex neuroimaging technologies. While enhancing the ecological validity of our tasks, it remains vital to develop new methods for collecting important physiological data.

Adaptive feedback control

The simulated virtual experience needs to change depend on the participant's action. In an ideal setup, a model is fitted to the participant's behaviour in real-time. The model predicts participants behaviour and provide statistics for their internal states, for example, their perceived pain and vigour. The task parameters are then changed depend on these estimated internal states. Task parameters are not limited to the virtual environment but can also include neural stimulation. Figure 4 illustrate the key components of the platform.

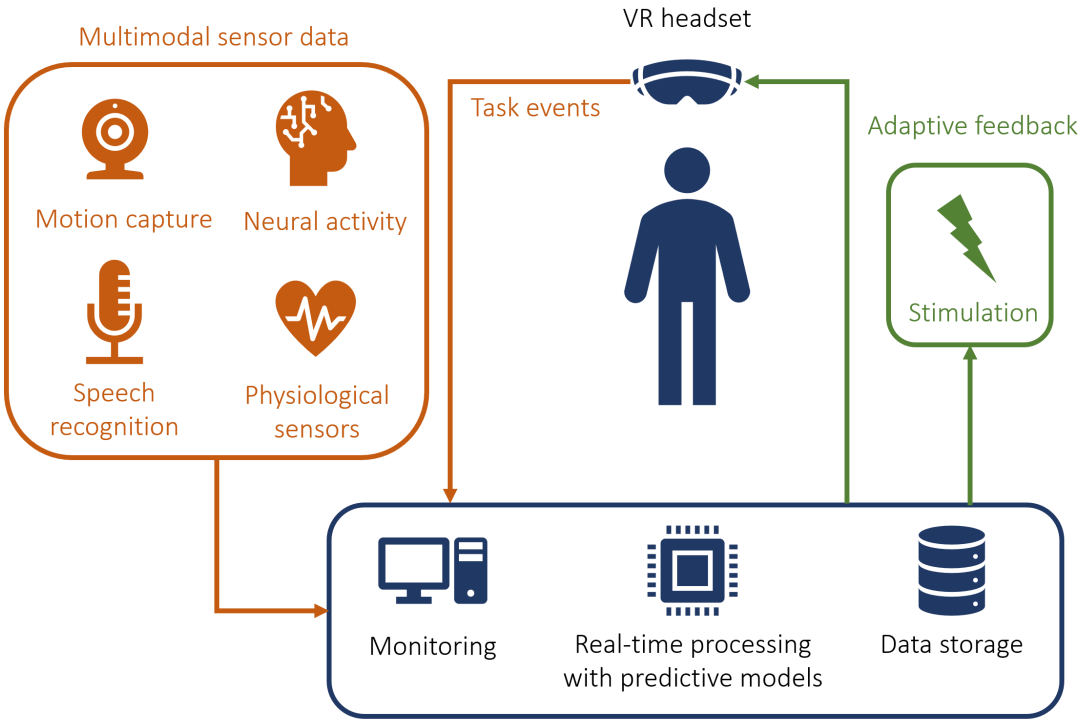


Figure 4: A schematic diagram for the proposed platform

2 Implementation of the platform

2.1 An overview from the event-driven design principle

Event-driven is a prevalent concept in computer system design. While event-driven design is widely accepted in today's data processing systems, neurophysiological control system in research and healthcare industry often employ simple procedural closed-loop or even open-loop design. As we are able to simulate experience and collect data at increasingly higher resolution and from a greater number of data sources, an event-driven design should serve as a basic design principle for systems that adaptively deliver content and feedback. We present a novel system implemented for pain research and therapy with ecologically valid virtual reality at its core. The event-driven design enables the system to achieve high modularity, data throughput, robustness, and versatility.

The concept of event-driven design is not new and is found at all levels of data processing systems, from the basic execution unit thread level [84, 85] to high-level information processing system architecture [86], and it even extends to complex applications for distributed business operations [87]. The event-driven concept is interpreted differently at various levels. In this paper, event-driven design is described as treating data of all modalities from different sources as equivalent events, with all data processing triggered by these events. Such a design makes asynchronous programming natural and improves the software system's robustness through modular design while maintaining high data throughput.

Our initial motivation is to study human pain mechanisms and develop pain therapeutics with adaptive feedback in ecologically valid contexts. Ecological validity is defined as designing an experimental or therapeutic task that resembles real-life scenarios. Ecological validity may not contribute many differences to some cognitive or physiological testing, but the context does modulate pain perception [6]. To obtain higher ecological validity, we integrated a virtual reality system with realistic task design. The richer interaction between participants and virtual reality, higher demands for physiological data collection, more flexible feedback for adaptive control, and low latency control pose new challenges for a task control system.

To meet these new demands, we designed and implemented a series of software components to form a novel event-driven system. In this section, we will first present an overview of the system architecture and then present the event-driven algorithms for handling various devices including virtual reality control, sensor data collection, and simulator control respectively. For the rest of this section, we then discuss a few examples featuring event-driven control.

2.1.1 System Architecture

Unlike a message-driven system where any devices can directly communicate with each other, we have chosen to adopt the client-server model where the device (client) can only communicate with the control server. This model allows for a central control role that monitors all the traffic and controls all devices in the system.

To establish a connection, a device first sends a device descriptor to the server. The descriptor includes a device identifier to identify itself and describes the data format it sends and receives. The data sent from the device to the server are termed ‘report data’, and the data sent from the server to the device are termed ‘control data’. Each side is required to send a 4-byte-long status code upon receiving data. Data transmission is bidirectional and concurrent. A separate channel for status code transmission would be useful, but for simplicity, our current implementation uses multiplexing and requires the data to be at least 5 bytes long. Figure 5 depicts the data transmission order for the two types of data. The controller does not initiate data transmission unless there is an incoming event (report data) from either the same device or another device.

Types of transmission	Specify report data order in descriptor	Supported data types	Requires additional library
High resources	No	JSON supported data types: object, array, number, string, boolean, null	JSON serialisation / desearilization
Low resources	A JSON array containing data field name and types	unsigned 32/16-bit integer little endianness (currently implemented)	No

Table 1: Comparison of two data transmission type supported by the protocol

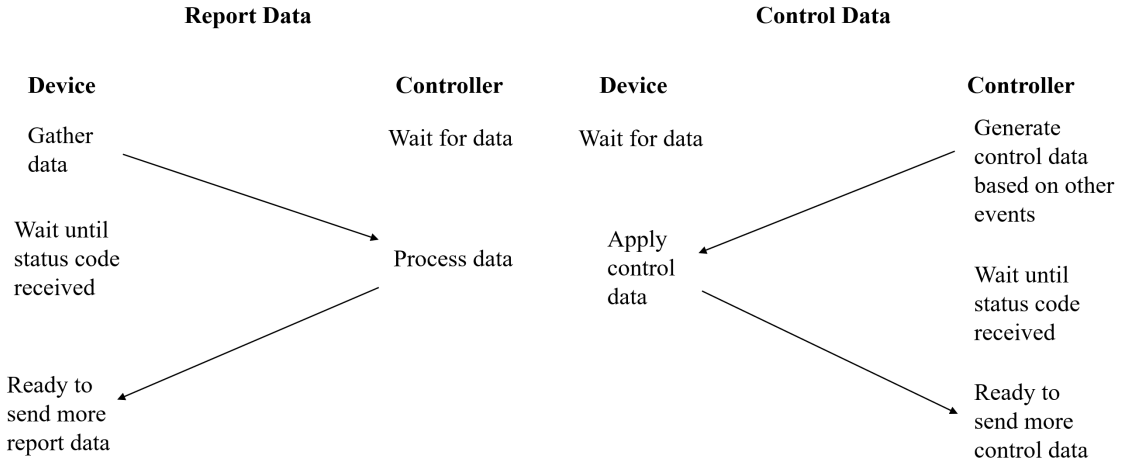


Figure 5: Two data transmission diagram. Both report data and control data can be transmitted concurrently, but for each type of data, it waits for previous transmission complete to proceed.

In a neurophysiological experiment, we have a wide range of devices transmitting data in different modalities. To balance development flexibility and performance, we have designed the communication protocol to accommodate both high-computing-resource devices and low-computing-resource devices. For a high-computing-resource device, the JavaScript Object Notation (JSON) is used as the data interchange format because it is lightweight, versatile, and well supported in many programming languages. For a low-computing-resource device, the device can transmit and receive compact binary data, thereby avoiding additional serialisation. It must specify the binary format in its descriptor so that the controller can decode the data correctly. Table 1 compares the two types of transmission.

2.1.2 Control-side Algorithm

2.1.2.1 Event-driven task-specific control

We have implemented the control-side program on Node.js, which provides comprehensive events APIs. We use the TCP/IP protocol for data transportation and addressing. Once an event is received, it is decoded into a JavaScript object represented as a dataframe. This object is then passed to a named function called `actionFunction`. Researchers are expected to override this function with their own control logic for specific tasks, providing a uniform control entry point encapsulated within a single function call. The `actionFunction` identifies the data source and processes the data accordingly. For any feedback control, the control data is written asynchronously by pushing the data into a control data queue.

Researchers override the `actionFunction` by supplying a task-specific JavaScript file where stateful variables can be defined and utilised within the `actionFunction`. Algorithm 1 illustrates an example of the typical structure of the task-specific control script.

Algorithm 1 Task-specific control

```
state ← InitState
TARGET_DEVICE_ID ← ID_STRING_CONSTANT
TARGET_DEVICE_ID2 ← ID_STRING_CONSTANT2
TARGET_DEVICE_ID3 ← ID_STRING_CONSTANT3
function actionFunction(deviceId, dataframe)
    if deviceID = TARGET_DEVICE_ID then
        controlCommand ← someAlgorithmicControl(state, dataframe) ▷ Process data
for this dataframe from this device and generate feedback command
        sendCommand(TARGET_DEVICE_ID2, controlCommand) ▷ send the control
command to another device asynchronously
    else if deviceID = TARGET_DEVICE_ID3 then
        ...
    end if
end function
```

2.1.2.2 Real-time data visualisation

Devices can specify how to visualise their report data in the descriptor. By properly specifying the file visualisation format, the control-side program can visualise the data in real-time (Figure 6a). Task-specific scripts may also contain important state information for

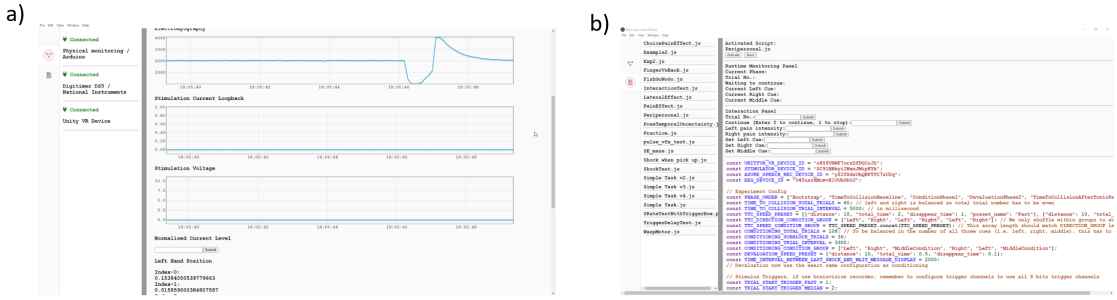


Figure 6: Main views of the control-side program developed with Electron. **a**, Left panel: list of connected devices. Right panel: real-time visualisation of report data. **b**, Left panel: Task-specific control scripts available to choose from. Right panel: top, script management (activate and save), middle, script variable visualisation and control, bottom, script editor

researchers to monitor in real-time. We have created a simple domain-specific language (DSL) for researchers to specify variables to be displayed in real-time. The DSL's syntax is a subset of JSON and is intended to be placed at the beginning of the script for the program to process the DSL correctly.

2.1.2.3 Ad hoc control by researcher

Besides algorithmic control commands generated by the script, in practice, it is often necessary for researchers to perform ad-hoc control of the task directly. We have facilitated this again through the DSL to preserve flexibility while enabling researchers to maintain full control during the algorithmic control runtime. In the DSL, researchers specify the variable to control and the function name to convert the input string into appropriate data types (Figure 6b). Ad hoc control modifies the script's state directly and does not go through the event system. That said, its effect can only be observed in the system after a new event is handled by the script.

2.1.3 Device-side Algorithm

The types of devices, their data modalities, and control feedback types all differ in some way. We categorise these devices by their data rate, which is quantified by the number of events rather than the number of bytes transferred. Given that the data protocol operates bidirectionally, in the following categorisation by data rate, report data and control data are interchangeable. We then present solutions to integrate different types of devices into

the event-driven loop.

2.1.3.1 High report data rate, no control data

The first type of device features a simple control flow yet is commonly found in neurophysiological research. A physiological sensor exemplifies this category. Our data protocol provides additional support for low-computing-resource devices, enabling the collection and wireless transmission of high-sampling-rate physiological data directly to the control-side program through commonly available, inexpensive IoT microcontrollers with a provided network stack. A practical example of this type of device is a physiological sensor collecting skin conductance response (SCR), photoplethysmography (PPG), and electromyography (EMG) data, implemented on an Arduino Nano 33 IoT. The simple flow of control is described in Algorithm 2.

Algorithm 2 High report data rate, no control data

```
allowSend ← true
while true do
    if not allowSend then
        allowSend ← CheckServerResponse()
    else
        dataToSend ← CollectData()
        SendReportData(dataToSend)
    end if
end while
```

2.1.3.2 Low report data rate, low control data rate

For simultaneous report and control data transmission, parallel execution is usually required to avoid data losses. However, if both report data and control data have low rate, and they are supposed to be separate in time domain, it is possible to transmit both types of data sequentially without encountering unexpected conflicts or noticeable delays. One example is our thermal stimulator controller running on a single-core processor. It waits for stimulation commands from the control-side program and reports temperature data after stimulation. Algorithm 3 summarises the flow of control

Algorithm 3 Low report data, low control data

```
while true do
    serverResponse ← ReadServerData()      ▷ Block until received full response from control-side
    if IsStatusCode(serverResponse) then
        CheckStatusCode(serverResponse)
    else                                                    ▷ Must be control data
        statesCode ← HandleControlData(serverResponse)
        sendStatusCode(statusCode)
        dataToSend ← CollectData()
        SendReportData(dataToSend)
    end if
end while
```

2.1.3.3 High report data rate, low control data rate

In cases where one of the data streams requires a high data rate while the other has a low data rate but demands low delay, multithreading to enable parallel execution may be necessary. A relevant example from our experience is controlling an electric stimulator. An electric stimulator continuously reports its DC voltage and current, while concurrently awaiting control commands to initiate stimulation. Algorithm 4 illustrates the control flow in the simplest case, avoiding data races.

Algorithm 4 High report data, low control data

```
semaphore ← Semaphore()
controlDataBuffer ← InitDataBuffer()
———— REPORT DATA THREAD ————
while true do
    serverResponse ← ReadServerData()
    if IsStatusCode(serverResponse) then
        ok ← CheckStatusCode(serverResponse)
        if ok then
            dataToSend ← CollectData()
            SendReportData(dataToSend)
        end if
    else                                                    ▷ Must be control data
        CopyMemory(controlDataBuffer, serverResponse)
        SemaphoreIncremnt(semaphore)
    end if
end while
———— CONTROL DATA THREAD ————
while true do
    SemaphoreDecrement(semaphore)
    HandleControlData(controlDataBuffer)
end while
```

2.1.3.4 High report data rate, high control data rate

In an ideal scenario where performance is sufficient and there are no data racing conditions between collecting data and applying control data, a multithreaded solution might be optimal. Nevertheless, if both report and control data have high data rates, a sequential flow of control may still be desirable to maintain a clean control flow without needing to address additional concerns for data races. An example is the virtual reality program developed in a game engine. Game engines typically adopt a loop-based control flow where the graphic frame is updated each loop. It is then natural to report all information as a single event and apply control data once per loop. The report and control data rate is then fixed at a 1:1 ratio. The control flow is illustrated in Algorithm 5. Whether handling control data or preparing report data first should match the control-side algorithm.

Algorithm 5 High report data, high control data

```
...                                                    ▷ Update loop call
statusCode ← HandleControlData(ReadServerData())  ▷ Read server data can be done
in a separate thread
sendStatusCode(statusCode)
dataToSend ← CollectData()
...                          ▷ Continue to other imperative statements in the update loop
```

2.1.4 Event-driven control examples

We discuss three typical examples using event-driven design.

2.1.4.1 Adaptive control by virtual events

A commonly used control flow in pain research involves delivering a wide range of pain stimulation based on the subject's interactions and actions within the virtual environment. As discussed in the previous section, the virtual reality device is considered to have both high report and control data rates. Typically, we report as detailed data as possible at each game engine refresh frame, including spatial locations of all tracked body parts of the subject, all interaction events subjects undertake, all environmental changes, and additional behavioural data collected by the game engine, such as eye-tracking data. A general rule of thumb to check if we have collected sufficient data is to attempt to replay the task based on the collected data to see whether everything in the virtual world can be replicated.

The control-side program now has sufficient information from each event to provide adaptive control of stimulation. For example, the control program can deliver conditional phasic pain stimulation if the subject makes a wrong choice, picks up a dangerous object, or moves in a punishable manner. More continuous control of tonic pain can also be applied. For instance, continuously increasing temperature via cutaneous heat stimulation is applied as the subject reaches their hand further into virtual hot water. This approach allows us to explore a wide range of pain stimulations under different conditions in an ecologically valid context [2, 77, 88].

While integrating cutaneous pain stimulation with virtual events is intriguing, incorporating brain stimulation is also feasible. Modern brain stimulation devices provide interfaces for remote programmable control [89]. With our event-driven platform, the integration's control flow and corresponding technical implementation are straightforward.

2.1.4.2 Adaptive control by physiological response

We implemented a neurofeedback virtual reality experience aimed at visualising subjects' physiological responses (Chapter 7). The concept behind this project involves using an immersive visual experience as additional feedback to help subjects learn techniques to control their physiological responses. In this example, the shapes and colours of abstract objects are dynamically controlled by real-time PPG, skin conductance, and chest respiratory transducer data. The PPG signal is converted to an estimated heart rate, which controls the sharpness of inner metallic cubes. Skin conductance influences the colour of particles, and the chest respiratory transducer's signal is converted to the radius of the outer particle sphere. All physiological signals to visualisation computations are implemented in the task-specific control script on the control-side.

2.1.4.3 Adaptive control by predictive models

A more compelling case involves implementing model-based control that integrates multimodal data. A straightforward example would be designing a task where the contingencies can be manipulated. By constructing an interpretable model of the brain, the objective for the control algorithm could be to promote or suppress target parameters in the brain model,

which is fitted to the real-time data of the subject [90].

The predictive model can also be applied to other devices, such as pain stimulators. We have proposed a decision model that combines both phasic and tonic pain with a trade-off in vigour cost. In many types of pain rehabilitation, exercises play a crucial role [76]. Researchers are also highly interested in utilising virtual reality experiences to promote exercises for pain patients [91]. With a model that predicts both vigour and pain severity, we can adaptively adjust exercise intensities to achieve optimal exercise levels.

The control algorithm can range from a simple linear PID control to a more complex non-linear neural network approximation [92]. The model's computation time may not be as rapid as the event rate feeding into the control algorithm. This is where the event-driven design elegantly handles differential event rates. The model can be computed on a separate device with a different event rate. Once the model completes its computation, it sends report data, and the control algorithm can then update the virtual reality experience or stimulation accordingly.

2.2 Implementation details

We have presented an overview of the system design adhering to the event-driven principle. In this section, implementation and technical details for various devices and control systems that I have developed are presented.

2.2.1 The communication protocol

As discussed in the overview, to establish a generalised framework that allows for rapid adaptation to new experimental or therapeutic plans and the integration of new sensors with various data modalities, a core component is the implementation of a new protocol. This protocol transmits data of different modalities, communicates special control and visualisation requirements between control-side and device-side software, and operates across as many types of devices as possible. Given our centralised control design, the classical client-server model is a suitable network topology for this purpose [93].

It is a routine task to create a specialised application-layer protocol, agreed upon by all

clients and the server, to transfer the desired structured information. In our case, the protocol must offer extra extensibility, low latency, light weight, simple implementation. Specifically, these requirements translate to:

- the protocol supports sending a wide range of data types;
- its computational complexity is reasonably low, which allows a fast and low-latency implementation to be possible; and
- the protocol requires minimum library support to run on various embedded devices and be portable; and
- it is as simple as possible for researchers to understand and implement on new type of devices without much additional work.

This application-layer protocol was named as ”Pain Lab Protocol”. By the point of writing up this thesis, we have implemented various devices communicating in this protocol. Table 2 shows a list of devices supported running on this protocol. The variety of devices implemented and tested confirms that our protocol meets the proposed requirements.

A complete protocol specification is presented in the Appendix. We highlight how the protocol addresses the proposed requirements.

Support a wide range of data types

A device descriptor must be sent by the device before transmitting any data. This descriptor informs the control software about the data decoding and modality. The data can be transmitted in either JSON format or a binary format, where the order of the binary frame is specified in the descriptor. In practice, we have encountered no issues transmitting all types of data over this protocol.

Adaptive encoding and decoding for devices with different performance

As illustrated in Table 1, the protocol allows two types of data format to maximise usability while keeping the performance reasonable for devices with different performance.

Device name	Platform	Function	Remarks
Control panel	Cross-platform PC	The central control server, collecting data, responding to data streams, sending adaptive control commands	Protocol implemented in JavaScript with Node.js backend
Virtual reality client	Unity game engine	The main program for virtual reality experience	Protocol implemented in C#. The protocol is cross-platform, but Unity game engine has further restriction on runable platforms
Physiological sensors	Arduino Nano IoT	Collect electromyography (EMG), Galvanic skin response (GSR), photoplethysmography (PPG) data wirelessly	Protocol implemented in Arduino's C++ and ARM assembly
Digitimer bipolar constant current stimulator	Cross-platform PC	Deliver electric stimulation with adjustable intensities	Protocol implemented in C#, requires National Instruments DAQ hardware
Medoc Pathway thermal stimulator	Windows PC	Deliver thermal stimuli with adjustable temperature	Protocol implemented in C#, requires Medoc's software
QST.Lab thermal stimulator	Linux (on Raspberry Pi Zero)	Deliver thermal stimuli wirelessly with adjustable temperature and report cutaneous temperature changes	Protocol implemented in C++ with Linux network stack
g.tec wireless EEG	Cross-platform PC	Record EEG wirelessly	Protocol implemented in C#, Labstreaming layer required
LiveAmp wireless EEG and physiological sensors	Cross-platform PC	Record EEG and other physiological data wirelessly, send triggers wirelessly	Protocol implemented in C#, Labstreaming layer required for EEG recording, Brain Products triggerbox required for wireless trigger
Speech recognition	Windows PC	Analyse subjects' speech to give pain ratings	Protocol implemented in C#, requires Azure subscription or Windows local speech engine

Table 2: Currently supported devices running on our communication protocol

Minimum library requried

In general, no additional library except a network stack required to run the protocol. This makes implementing a new device with this protocol simple and the implementation can be portable.

Comparison with other similar software

Numerous open-source and closed-source data collection software packages are available for neurophysiological research, most of which are focused on data collection. If one's sole intention is to collect data, our software may not have an advantage over other existing, popular software due to compatibility issues. For example, Lab Streaming Layer is a powerful open-source data collection tool with timestamp-based data synchronisation. While it is possible to develop an event-driven system using existing application layer protocols like Lab Streaming Layer protocol or more prevalent ones like protobuf, our implementation offers a more specific and flexible solution for the neurophysiological research field. For instance, we can easily implement this protocol on embedded devices with limited memory and flash, usually with just over a hundred lines of code. This also gives us greater control over the functionality of the protocol. For example, we can modify the protocol to transmit a minimal amount of data by removing the control data part, while continuing to use the same control software interface.

2.2.2 Control software implementation

We have discussed key algorithms on the control side. This section further discusses the implementation details of these algorithms.

2.2.2.1 Task-specific control script

As the control software is implemented in JavaScript on Node.js, for simplicity and compatibility, the script is required to be written in JavaScript. Users can load scripts from the graphical user interface. The control software provides a simple text editor for users to edit scripts. Once the user presses the `Activate` button, the script is evaluated directly using the JavaScript `eval` function. Although `eval` is not recommended in practice for security

reasons, we trust that users will take responsibility for ensuring the script is secure.

By directly evaluating the script, users can define stateful global variables. To adaptively control the devices, users must define (technically, override) an `actionFunction` in their script. The `actionFunction` is then called by the control software with a new dataframe from the device passed as parameters to the function. This provides an effective event-driven programming interface, as discussed in the previous section.

We additionally provide built-in functions to facilitate event-driven control. For example, `sendCommands` accepts a JavaScript object and pushes the data to be sent to the queue. The control software will convert the JavaScript object to the appropriate data format and send it once the device is ready to accept a new control command.

2.2.2.2 Real-time visualisation

Real-time visualisation is supported for different data modalities (Figure 6a). Devices specify these modalities in their descriptors to ensure proper visualisation. Common visualisations include line charts and static displays, where data is directly converted to a string and displayed alongside the corresponding field name. The control software constructs the visualisation interface by reading the specifications provided in the descriptor.

2.2.2.3 Researcher ad hoc control

Ad hoc control by the researcher was discussed in the previous section (Figure 6b). Here, we provide details of the Domain-Specific Language (DSL) that defines modifiable script variables.

The DSL script is intended to be included before the control script begins. A special string marks the end of the DSL script. The control software collects the DSL script and creates an `interactiveScope` object with field names specified in the DSL script. Since JavaScript's `eval` function wraps variables in a separate scope, a global `interactiveScope` variable is necessary to access these variables within the control script. The author of the control script stores the data they wish to enable for ad hoc control in `interactiveScope` and defines data conversion functions in the DSL so that the data can be properly displayed and accessed from the control software interface.

2.2.3 Device software implementation

We have demonstrated the main types of device-side algorithms, which cover most design considerations when implementing the software on the device. With the help of the effective communication protocol, it is straightforward to implement the protocol on the device and integrate it into the control network. Therefore, in the following paragraphs, we will discuss a few useful technical details for selected devices.

Proprietary hardware with lab streaming layer

It is worth mentioning that we have observed that for some proprietary hardware, manufacturers do not provide specifications for communicating with their devices. However, they sometimes provide an official Lab Streaming Layer (LSL) implementation without providing the source code. In such cases, we developed a compatible layer that forwards data streams from the Lab Streaming Layer to our control software.

Unity game engine

The Unity game, serving as the primary device in a virtual-reality-based study, not only renders realistic environments and reports participants' actions in the virtual world but also connects to various hardware to collect data. For example, eye-tracking data is collected and processed in Unity via the manufacturer-provided eye-tracking plugin for Unity. In Unity, we then write functions to compute a ray from the eye-tracking data and identify which object the subject is fixated on. In some cases, simple feedback control can also be transferred from the control software to the device-side Unity program. For example, we tested adaptively changing the position of virtual hands to assess whether this alters embodiment and uncertainty about body location. This demonstrates that device implementation does not fully exclude control elements; however, the control hierarchy is generally preserved (i.e., device-side control is still subject to the control software).

2.2.4 Integration with the hardware

In this section, we provide brief descriptions of the supported devices listed in Table 2 that require additional work to integrate with the hardware.

2.2.4.1 Wireless physiological sensor system

We developed a wireless physiological sensor system that collects electromyography (EMG), galvanic skin response (GSR), and photoplethysmography (PPG). The sensor system is built on an Arduino Nano 33 IoT microcontroller with a built-in Wi-Fi module, and the network stack sources are provided by the manufacturer.

The sensor parts are sourced from various manufacturers. All sensor parts output analogue signals, which are then converted to digital signals using the on-board analogue-to-digital converter (ADC). The Arduino Nano 33 IoT's on-board ADC exhibits high latency in signal conversion (over 1 ms). We configured the clock to reduce the conversion time of one sample to achieve a 1000 Hz sampling rate. This adjustment could potentially cause instability in the hardware, although we did not observe any issues with the ADC in practice. The converted digital data is then written to a data frame with a local timestamp (in microseconds). As transmitting the data to the control software takes a few milliseconds, we create a buffer in the memory to store data frames ready to be sent. Once the device is ready to send, the data frames in the buffer are sent to the control software in the specified protocol format. The complete device is shown in Figure 7.

There are a few limitations and precautions with the implementation.

Sampling rate control and delay correction

The first version of the program checked the current local time against the previously collected sample time. If more than 1 ms had passed, a new sample was collected and saved in the buffer. However, in the first version, the time of the previous sample was recorded at the end of the loop after sending the data. While the data sending is asynchronous, this introduced a gap that reduced the effective sampling rate from the designated 1000 Hz. We had a local timestamp in every data frame with microsecond precision, and we were able to use these timestamps to resample the data to the desired sampling rate. Another point to note with this correction method is that the Arduino's local time is in microseconds, and an unsigned 32-bit integer can hold up to 4294 seconds of time (approximately 1.19 hours) with microsecond precision. After this, the time resets, and the resampling algorithm needs to account for this. Fortunately, none of our experiments using this device

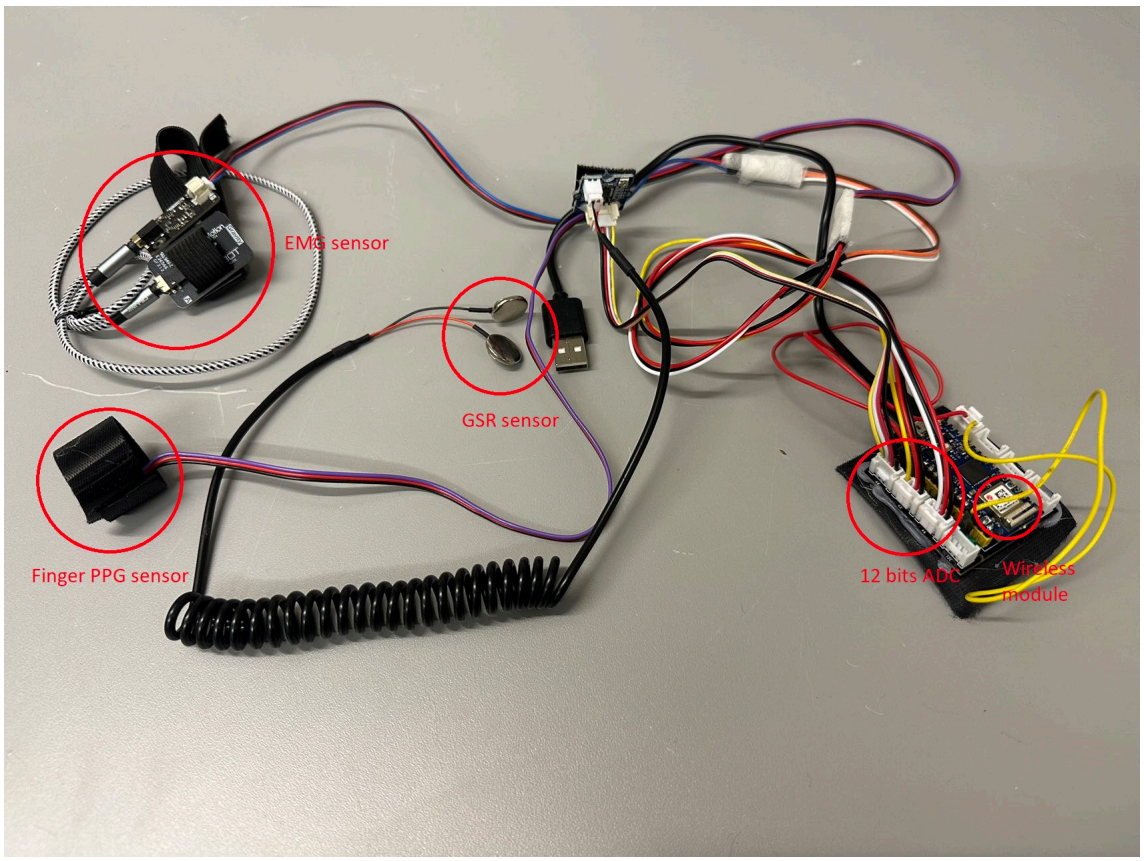


Figure 7: Wireless physiological sensor system

ran for over one hour.

Device stability

Initially, with the outdated Wi-Fi firmware, the device would hang while reading from the Wi-Fi module every few minutes. After updating the Wi-Fi firmware, the frequency of hangs significantly decreased to every few hours of use. We were unable to identify the issues in the firmware, and the ad hoc solution is to monitor the data on the control software and manually reset the device if a hang occurs.

Wireless interference can also cause data frame drops. New data is continuously pushed into the buffer. When the buffer is full, additional data frames will be dropped. If too many data frames are dropped, the device will automatically reset and establish a new connection to the control software after it resets.

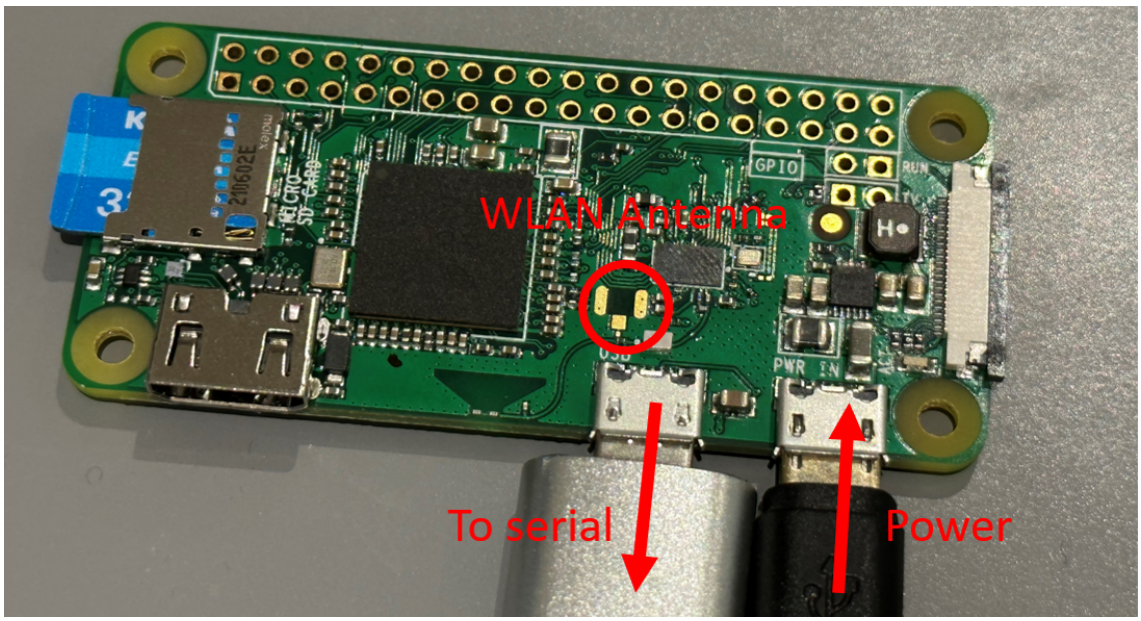


Figure 8: Wireless thermal stimulator control

2.2.4.2 Wireless thermal stimulator control

We acquired a portable-size thermal stimulator manufactured by QST.Lab. The stimulator can be controlled via serial input. To increase hardware compatibility, rather than developing on a microcontroller, we developed a Linux program that receives formatted commands from the control software wirelessly and converts these commands into serial signals. We run the program on a Raspberry Pi Zero W single-board computer (Figure 8).

The original motivation for developing on a Linux computer was to utilise multithreading to allow concurrent data reporting and control data application. However, we were unable to source a multi-core Raspberry model within a short time period. Therefore, we proceeded with a single-core model and wrote a single-threaded program. To prevent concurrent control and data transmission, we allow the report data to transmit briefly only after a control command has been executed. This approach works well for the function of the thermal stimulator.

2.2.4.3 Electric pain stimulator control

An electric pain stimulator is common in pain behavioural research. We used bipolar constant current stimulators manufactured by Digitimer to deliver painful electric shocks safely. There are two main types of stimulators. One is trigger-based, where square wave

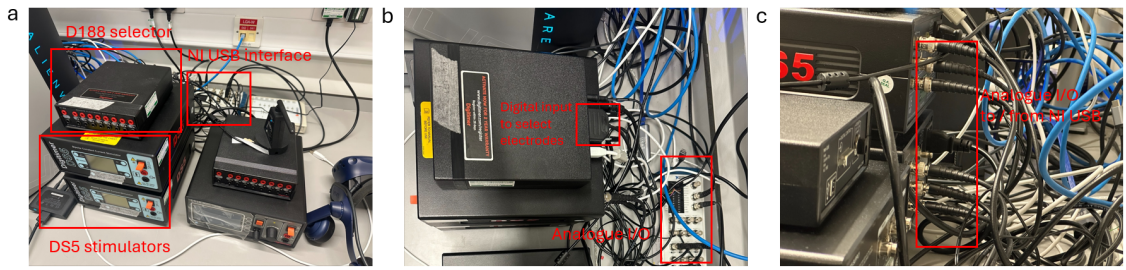


Figure 9: Stimulator setup

stimulations are delivered. Common models include Digitimer DS7A and DS8R. Another type of stimulator converts an input voltage signal to specify output voltage. This allows for highly flexible waveforms of stimulation by providing corresponding analogue voltage signals. However, due to noise present in the input signal, extra care must be taken when using high-impedance electrodes. Details and precautions are described in Chapter 5.

We generated the corresponding voltage signal using a USB interface manufactured by National Instruments (USB 6212 BNC). This interface can read and produce both digital and analogue outputs. We wrote a .NET program that acts as a device reporting real-time stimulation data and applying stimulation with low latency. The stimulation is parameterizable for pulse width, pulse frequency, and number of pulses. The setup also works with a switch box (Digitimer D188) that changes the output of a stimulator to different electrode pairs via digital signals. The setup configuration is shown in Figure 9.

2.2.4.4 EEG data collection and synchronisation

Data collection

Commercial wireless EEG devices commonly use proprietary software to collect data. Although they sometimes show an intention to embrace open source, the core implementation remains proprietary. LiveAmp, a wireless EEG system manufactured by Brain Products GmbH, has a data collection client that sends data in Lab Streaming Layer (LSL) format. While LSL format is open-source, the implementation of the client is proprietary. Therefore, we can only implement a compatible layer that forwards LSL data collected from the proprietary client to our own Pain Lab Protocol. However, we observed that recording over LSL can have an effective sampling rate less than the designated 500 Hz. We tested in our lab environment and recorded an average of 498.3 Hz ($\pm$1.8 Hz) sampling

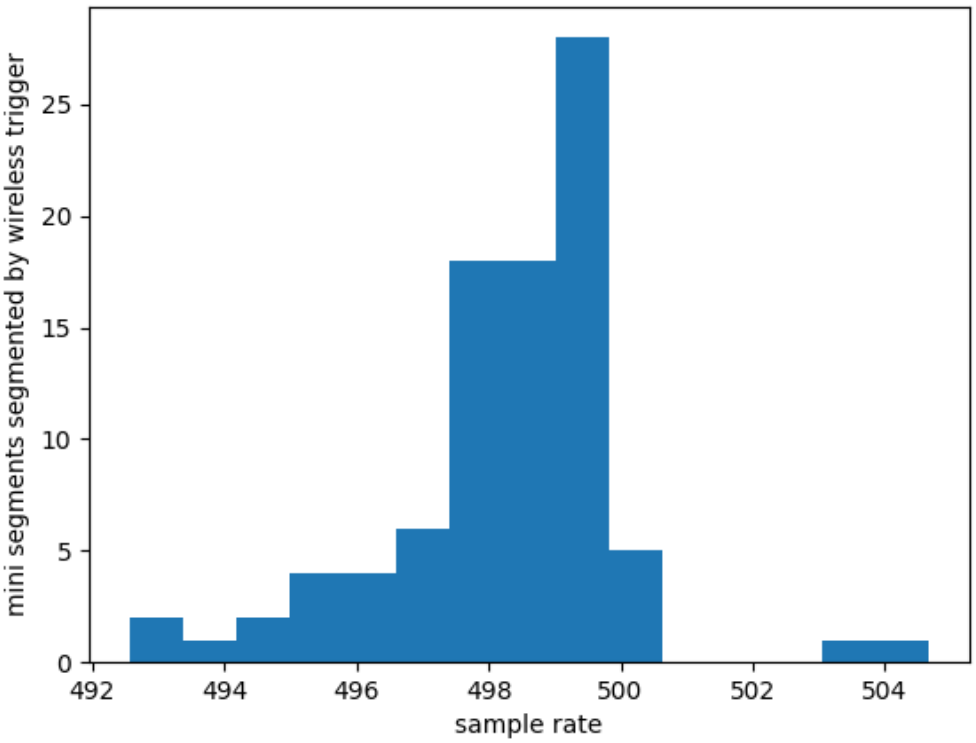


Figure 10: Data drop in EEG recording

rate over 91 segments (Figure 10).

We further investigated the causes of data losses. Testing in three different environments, we found that wireless interference at 2.4 GHz contributed most to the data losses. By turning off as many wireless devices as possible, we were able to achieve a 500 Hz sampling rate with the LSL recorder. Placing Apple AirPods nearby, which operate on the 2.4 GHz band, the effective sampling rate dropped to 482-485 Hz. We also tested different recording methods. Recording with the full proprietary BrainVision Recorder resulted in much less data loss: two samples lost out of one million in the worst case.

Given that data losses are still possible, for EEG analysis described in Chapter 6, we used local recordings saved to a flash memory card that is not affected by wireless interference.

Synchronisation

An essential component in EEG analysis is the use of event markers attached to the samples to identify meaningful EEG recording segments. These event markers should synchronise with the activities involving the subject. To minimise the latency, we utilise a

wireless trigger sender and receiver developed by Brain Products GmbH, which accepts serial signals and sends triggers wirelessly. We developed a client program that acts as a device receiving commands given by the control software and writes the corresponding command in serial to the wireless trigger sender.

We present essential source code in Appendix.

2.3 Data Analysis

2.3.1 Python library

We created a general library to process the data collected by the control software. After data is written to storage from the control software's memory, each device that has been connected to the control software has a separate JSON file that contains an array of data frames. Therefore, it is natural to organise each session's data in a folder. Recursive helper functions are created to process all sessions of data in a list structure. Lazy evaluation functionality and selective device data loading improves performance and avoids memory overflow in case the number of sessions is large. Key details are presented as source code in Appendix.

2.3.2 Accelerated grid search model-fitting

Computational models were originally implemented in Python. As the model-fitting problems we encountered were generally non-trivial with non-convex objective function, it is hard to find a robust optimization algorithm that is guarantee to converge to the global minimum. We then adopt grid search to solve the problem. However, grid search is computationally inefficient and quickly become intractable as the number of parameters to fit grows. On the other hand, grid search does not depend on previous steps and different sets of parameters can be comptued in parallel without any special treatment. We developed corresponding program on proprietary Compute Unified Device Architecture (CUDA) that runs on NVIDIA GPUs. Additional test scripts were written to ensure Python and CUDA version has the same result (up to some epsilon difference due to single precision used in the CUDA program). Key implementation details are included in the Appendix.

2.4 The base task

Although we have already conducted several different types of studies on this platform, the tasks discussed in the following four chapters are based on the same foundational task. This base task represents one of the most ecologically valid tasks implemented on this platform, incorporating the ecologically valid design elements proposed in Chapter 1.

The base task is a free-operant foraging task. Foraging theory is a broad and well-studied subject within animal behaviour research. Various foraging study paradigms have been established in both laboratory and natural environments [94, 95, 96]. A natural foraging task, while simple for a healthy human to perform, encapsulates key components relevant to pain research. Firstly, foraging is goal-directed; subjects actively search for food under varying levels of motivation, allowing for the examination of the interaction between motivation and pain. Secondly, a natural foraging task involves full-body movement. Given that many types of pain are closely linked to movement and exercise, motion capture technology—a fundamental component of immersive virtual reality—enables precise recording and analysis of movement under different pain conditions. Finally, foraging involves exploring the environment and learning an optimal strategy to maximise rewards, naturally linking pain to cognitive functions. Motivated by these considerations, we designed the free-operant foraging task described below, aimed at simulating realistic foraging behaviours.

At the beginning of each block, virtual pineapples of varying heights from 0.33 to 1 m were generated within the virtual space. The density of the pineapples was set to $5.0625\ /\mathrm{m}^3$. Five identical baskets were placed within the space. The spatial locations of trees and vegetation were generated using the game engine's default tree painting tool. Participants were asked to collect as many pineapples as possible. The experiment was divided into blocks, each with different conditions. Each block was set to one minute. Within each one-minute block, participants were allowed to freely perform any actions they wished within the physical boundary. Figure 11 illustrates the task from both a virtual view and an external view. Figure 12 shows the actual experiment.



Figure 12: Photo of the actual experiment

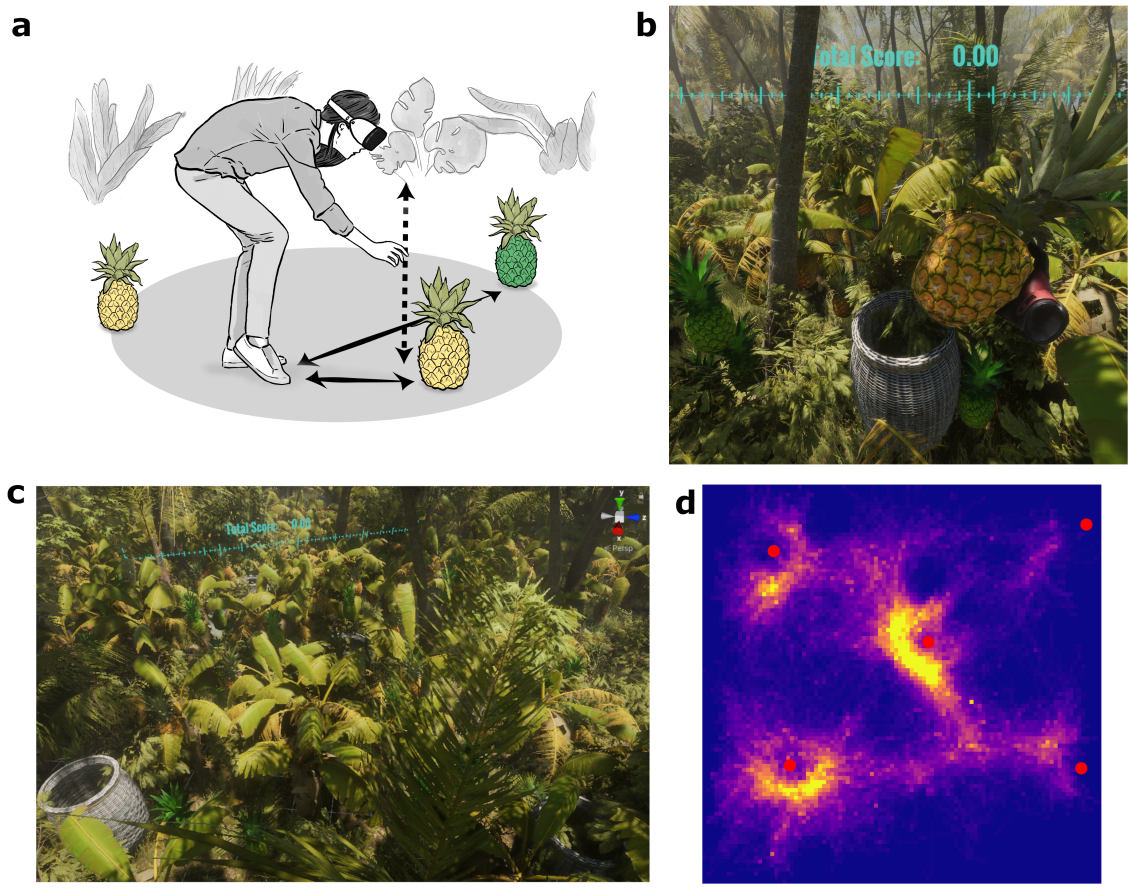


Figure 11: **a**, A illustrative figure depicting the foraging task. **b**, Participants' view (left eye) in the task. **c**, A perspective view of the task environment. **d**, An aggregated top-view heat map of head trajectories of all participants' data. Red dots denote baskets' preset locations.

3 Ecologically valid free-operant foraging study differentiates perceived pain in stationary and active contexts with unaffected vigour

3.1 Introduction

In Chapter 1, we discussed, from a computational perspective, that pain, as a control signal, can be modulated for different purposes. Pain is unlikely to be a single static value; rather, it is a complex dynamic signal generated by multiple internal modulation mechanisms. There are many different mechanisms that modulate pain. Seymour [6] summarised four major categories driving the endogenous control of pain: sensory inference, predictive value, decision conflict, and informational value

Modulation by sensory inference involves correcting pain signals using prior knowledge through inference processes such as Bayesian inference. Modulation by predictive value suggests that pain functions both as a cue and an outcome; as a cue, if it is conditioned with further pain, it is expected to be amplified. Modulation by informational value estimates uncertainty and modulates pain to promote or inhibit exploration. While there are more details to these types of modulation, in this chapter, we focus on examining modulation through the decision conflict hypothesis, as it fits well within our new platform with enhanced ecological validity.

As we have illustrated with hierarchical learning models for pain and action decisions, different models within the hierarchy can arrive at different decisions, resulting in conflict. An example is when a high-level model decides to undertake a painful action in exchange for a long-term reward, while the painful stimulus may activate low-level innate responses that could interfere with the action's execution.

We consider the case where innate responses interfere with actions instructed by the high-level decision-making system. If the action involves motor control and the innate response involves a reflex, the reflex initiated by the innate response can interfere with motor control. If the high-level action is motivated to proceed, eliminating the interference caused by

the innate response is preferable. In this case, we say the innate response is incompatible with a high-level action decision. Given that a descending control pathway affecting neurons in the dorsal horn of the spinal cord exists, it is reasonable to assert that endogenous analgesia is applied to suppress innate responses to pain [6]. Therefore, the conditions for actions instructed by high-level decision-making that activate endogenous analgesia in this context include:

- the action involves motor control;
- the action causes pain which elicit a reflex as the innate response; and
- the action is motivated to take.

Motor control plays a crucial role in the scenario described above. Extensive research has been conducted to study pain interference in motor learning among healthy individuals, focusing on task performance measures under different motor control paradigms and the correlated changes in neural observations [97]. However, to the best of our knowledge, little research has concentrated on the modulation effects and their impact on different levels within the hierarchical motor control system. It is worth noting that long-term studies on the prognoses of real chronic pain patients and the changes across multiple levels of their motor systems have been known and studied for a long time [98]. Finally, motivation (reward) and pain appear to compete at the high level of motor control, with the outcome having a differential effect on motor execution [99].

We use the free-operant foraging task described in Chapter 2 to examine this modulation effect. The primary outcome is the change in pain ratings before and during the task. A significant difference in pain ratings could indicate the activation of endogenous analgesia, as predicted by the decision conflict hypothesis. In addition to subjective reports, since previous studies have shown an increased nociceptive threshold with analgesics [75], we planned to test for changes in nociceptive thresholds using electromyography (EMG). Finally, increasing pain typically exhibits a dose-response effect [99]. As stimulation intensities increase, we expect participants to slow down their collection rate to experience fewer painful shocks within a fixed time period. If the aversive stimulation is viewed as a negative-valued reward, and given that pain ratings remain almost unchanged during the

task, this prediction aligns with previous free-operant learning literature in both animal and human studies [54, 100].

3.2 Methods

3.2.1 Study design

Twenty subjects were recruited to perform a free-operant instrumental foraging task to accrue reward points. A virtual boundary of 4×4 m was displayed when the subject approached the edge of the area. Within the VR context, this space resembled a flat forest (Figure 11), with the locations of trees and vegetation randomly generated. The task was structured into one-minute blocks. At the beginning of each block, subjects received a brief pain shock identical to the one they would experience during the task. They were then prompted to rate the shock's pain level. After submitting their pain ratings, the main task began. Virtual fruits of varying heights and locations were generated within the virtual space. The spatial coordinates of the subjects' heads and hands were tracked, requiring them to physically move towards a pineapple, pick it up by reaching with their dominant hand, clicking, and holding the button on the handset. Only the dominant hand was enabled in the virtual environment to control the pain stimulation is on the same side as the action side. Subsequently, they could drop the fruit into one of the baskets to earn points. Pineapples were the only available fruits in the experiment, as their spiky shape naturally matched the pricking pain induced by the electrical pain stimulation electrodes used. After the one-minute task ended, subjects were asked to rate the pain they experienced during the block.

Picking up a pineapple immediately elicited a brief, painful cutaneous electric stimulation to the proximal lateral side of the middle finger of the dominant hand. The intensity of the electric stimulation was held constant within each block. The experiment consisted of 20 blocks, with the initial 10 blocks designated for practice purposes with a fixed order of stimulation intensities. The first 10 blocks had deterministic stimulation intensities ranging from 0% to 100% and then back to 0% of the maximum calibrated intensity, determined by a pain calibration procedure, to control for the range effect [101, 102]. The

subsequent 10 blocks used randomised pain stimulation intensities across five different levels (0%, 25%, 50%, 75%, and 100%). After the 10 training blocks, subjects were aware that picking up a pineapple might result in a shock, but they did not know the exact pain level for each new block. Only the data from these latter 10 blocks were analysed.

3.2.2 Experimental pain stimulation

The electric stimulation pulse train consisted of 10 pulses at 200 Hz with a 1 ms pulse width. This stimulation was generated by a DS5 isolated bipolar constant current stimulator (Digitimer, Letchworth Garden City, UK) and delivered using a Wasp pain stimulation electrode. Electric current calibration was conducted based on subjects' ratings prior to performing the task. Subjects rated their pain on a 0-10 numerical rating scale while researchers adjusted the stimulation current using a staircase procedure typically employed to determine the nociceptive withdrawal reflex threshold [103]. The calibration process was considered complete when subjects provided consistent pain ratings within the desired range, as determined by the experimental design.

At the beginning and end of each block, subjects provided a pain rating for that block. Ratings during both the calibration process and the tasks were delivered vocally. During calibration, researchers recorded the subjects' vocal ratings. For post-block ratings, vocal ratings were analysed by speech recognition services (Microsoft, Redmond, US) and converted into a visual analogue scale bar displayed in virtual reality. In cases where the speech recognition services failed, the researcher recorded the subjects' pain ratings and overrode the value recorded by the automatic speech recognition system.

3.2.3 Behavioural data processing and statistical tests

Behavioural data were collected using the control software described in Chapter 2. The primary behavioural data were gathered by the Unity game device. This data was sent to the control software and saved in JavaScript Object Notation (JSON) format. Each subject's data was stored in an array of frames, where each frame contained the UNIX timestamp, the subject's head and hand locations, their current action, eye-tracking data, and environmental events. The UNIX timestamp was used to synchronise with physio-

logical data.

Data processing was conducted using custom scripts written in Python 3.9.13. Data frames were segmented into blocks and iteratively processed to extract relevant data. Statistical analysis was carried out using custom scripts written in Python 3.9.13 and R 4.2.2. For linear mixed models, the lme4 and lmerTest packages were used [104] [105]. For linear mixed model fitting, fixed effects, random effects for intercepts, and independent variables (IVs) were estimated. The model also estimated the correlation between intercept deviations and IV effect deviations across subjects.

3.2.4 Physiological data

Electromyography (EMG) was the primary physiological data collected. The data was gathered using a custom-made wireless sensor based on the Arduino Nano IoT platform. The EMG sensor component was manufactured by Myoware (Advancer Technologies, Raleigh, US). EMG electrodes were placed on the brachioradialis muscle, with the reference electrode positioned on the bony part of the elbow, aiming to detect the nociceptive withdrawal reflex. The raw signal was zeroed, rectified, and integrated [106]. The evoked response was extracted based on the time point when the subject picked up a pineapple, which induced a painful shock delivered to their finger.

3.3 Results

3.3.1 Pain ratings

The main expected result was a difference between the in-block pain ratings and the pre-task pain ratings reported by subjects. If the modulation by decision conflict hypothesis holds, then a decreased rating for the pain during the task compared to the pre-block pain ratings taken before starting the block is expected.

We first present the differences in pain ratings for each pain condition separately. A significant decrease in pain ratings for in-block pain was identified with 75% of the maximum stimulation intensity (Figure 14). The results for the 100% and 50% of maximum stimulation intensities conditions also show lower average ratings for in-block pain (Figures 13,

15). The 25% low pain condition shows a small difference in ratings between pre-block and in-block pain (Figure 16).

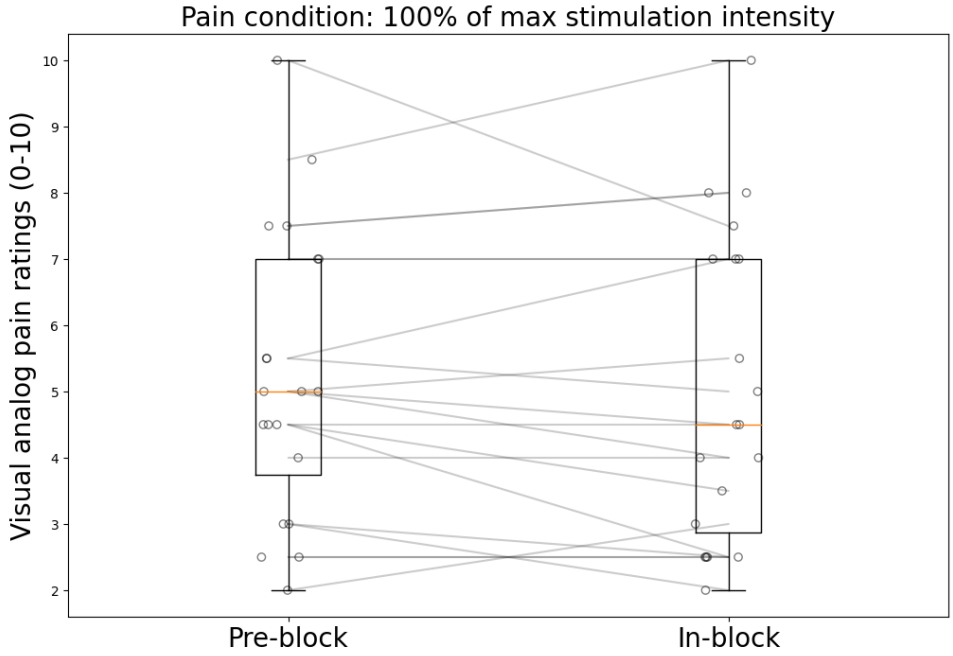


Figure 13: Pre-block calibration pain ratings and pain ratings for the block at 100% stimulation intensity. A two-tailed paired t-test was conducted to evaluate whether pre-block calibration pain ratings ($M = 5.2, SD = 0.50$) was significantly greater than the ratings for pain received during the block ($M = 5.03, SD = 0.38$): $t(19) = 0.770, p = .451$, Cohen's effect size $d = 0.172$, and Bayes factors $BF_{10} = 0.303$.

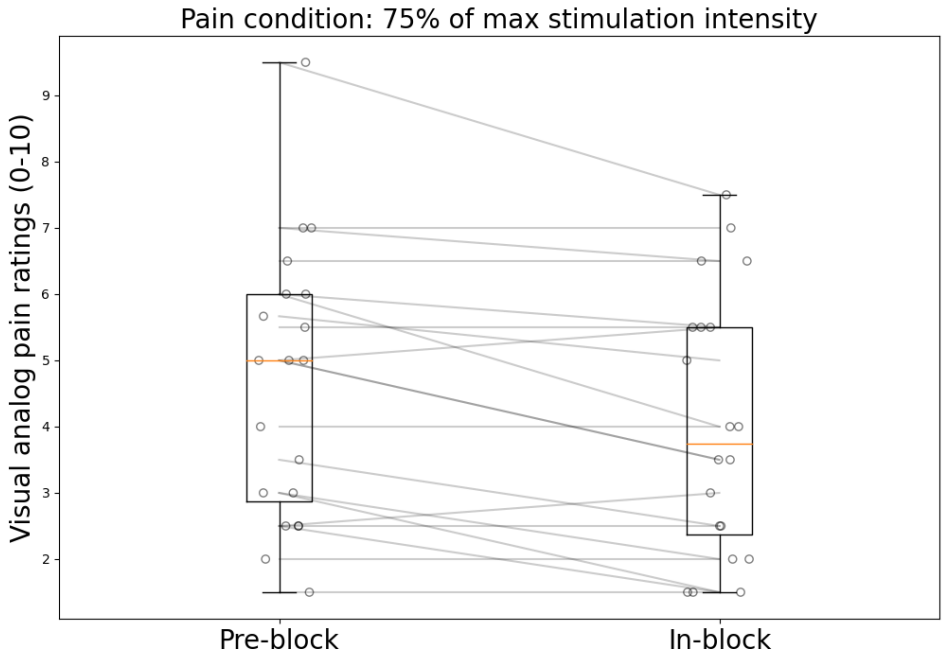


Figure 14: Pre-block calibration pain ratings and pain ratings for the block at 75% stimulation intensity. A two-tailed paired t-test was conducted to evaluate whether pre-block calibration pain ratings ($M = 4.63, SD = 0.47$) was significantly greater than the ratings for pain received during the block ($M = 4.03, SD = 0.48$): $t(19) = 3.453, p = .003$, Cohen's effect size $d = 0.772$, and Bayes factors $BF_{10} = 15.689$.

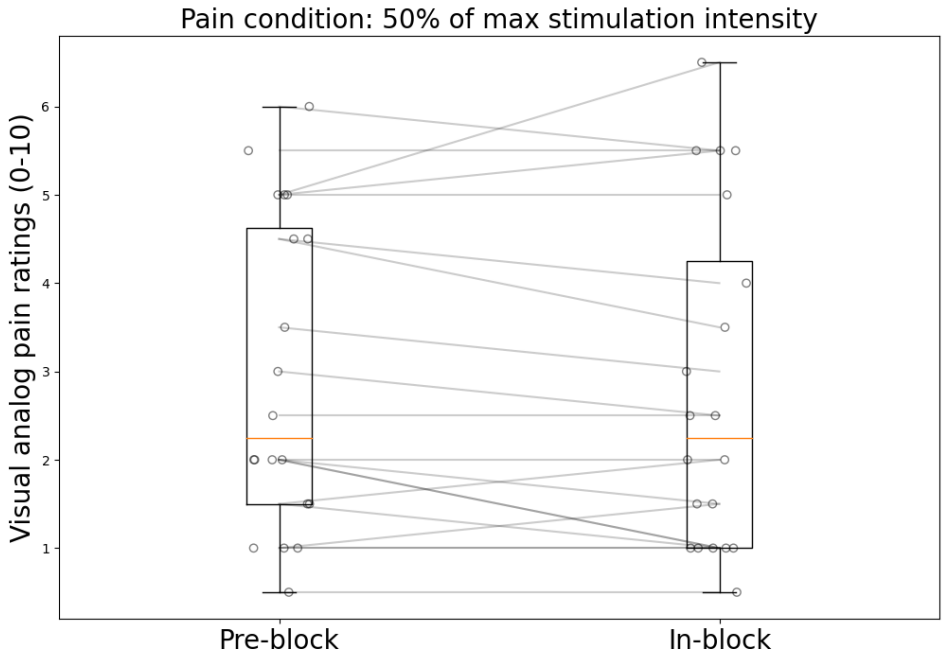


Figure 15: Pre-block calibration pain ratings and pain ratings for the block at 50% stimulation intensity. A two-tailed paired t-test was conducted to evaluate whether pre-block calibration pain ratings ($M = 2.95, SD = 0.5$) was significantly greater than the ratings for pain received during the block ($M = 2.8, SD = 0.45$): $t(19) = 1.101, p = .285$, Cohen's effect size $d = 0.246$, and Bayes factors $BF_{10} = 0.396$.

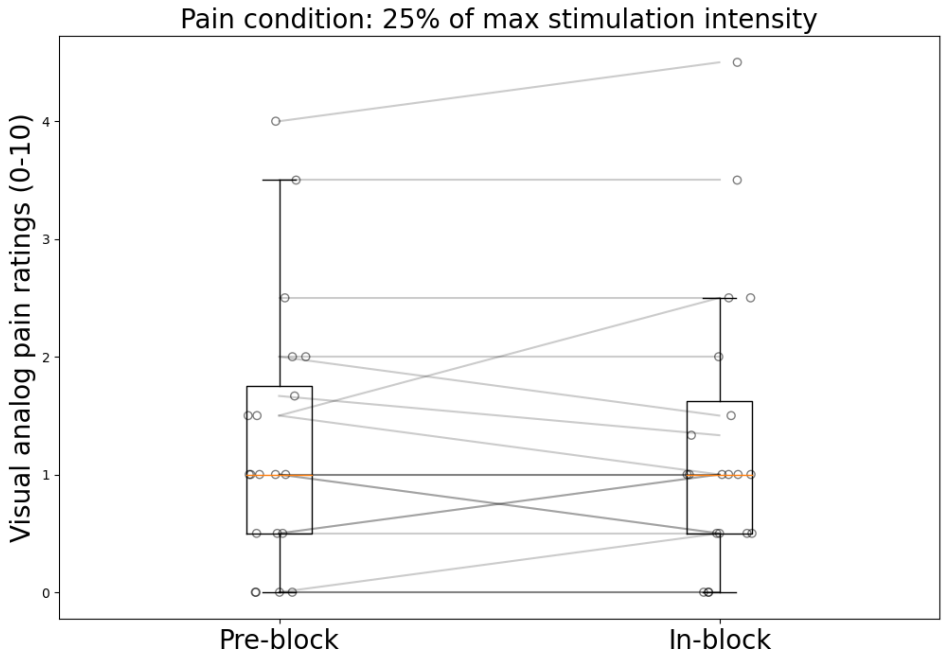


Figure 16: Pre-block calibration pain ratings and pain ratings for the block at 25% stimulation intensity. A two-tailed paired t-test was conducted to evaluate whether pre-block calibration pain ratings ($M = 1.26, SD = 0.32$) was significantly greater than the ratings for pain received during the block ($M = 1.29, SD = 0.25$): $t(19) = -0.370, p = .716$, Cohen's effect size $d = -0.083$, and Bayes factors $BF_{10} = 0.247$.

The following figure shows the average over all pain stimulation intensities. The significant test result is consistent with the modulation by decision conflict hypothesis.

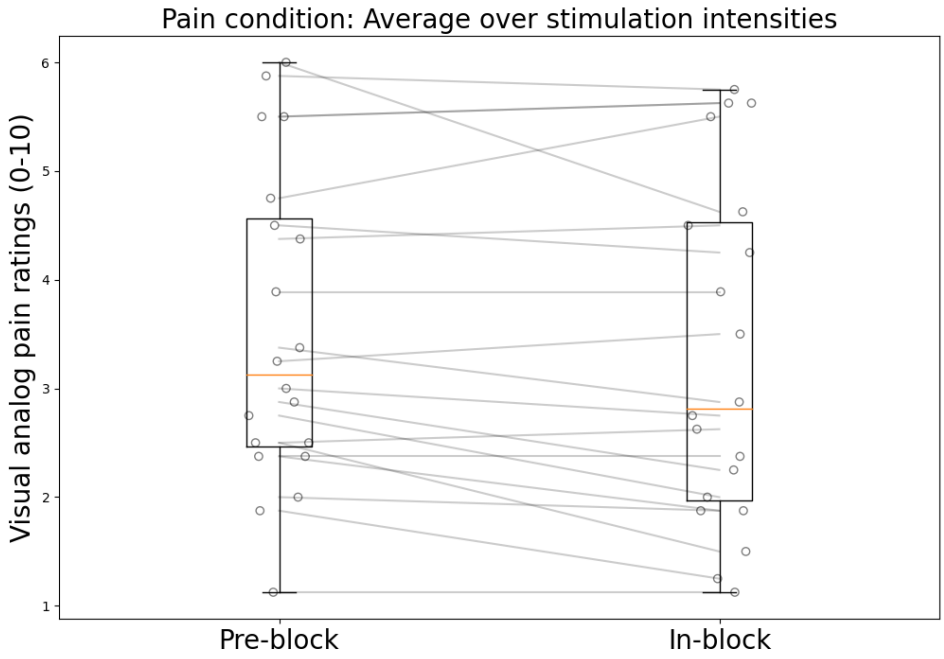


Figure 17: Pre-block calibration pain ratings and pain ratings for the block over all stimulation intensities. A one-tailed paired t-test was conducted to evaluate whether pre-block calibration pain ratings ($M = 3.52, SD = 1.85$) was significantly greater than the ratings for pain received during the block ($M = 3.28, SD = 1.64$): $t(19) = 2.123, p = .047$, Cohen's effect size $d = 0.475$, and Bayes factors $BF_{10} = 1.456$.

3.3.2 Electromyography

Based on the results that analgesics increase the threshold of the nociceptive withdrawal reflex [75], the hypothesis predicts that there should be an increased threshold of the nociceptive withdrawal reflex during the task compared to the pre-block calibration stimulation. However, as shown in Figure 18, we were unable to identify the nociceptive withdrawal reflex signal from the noisy EMG data.

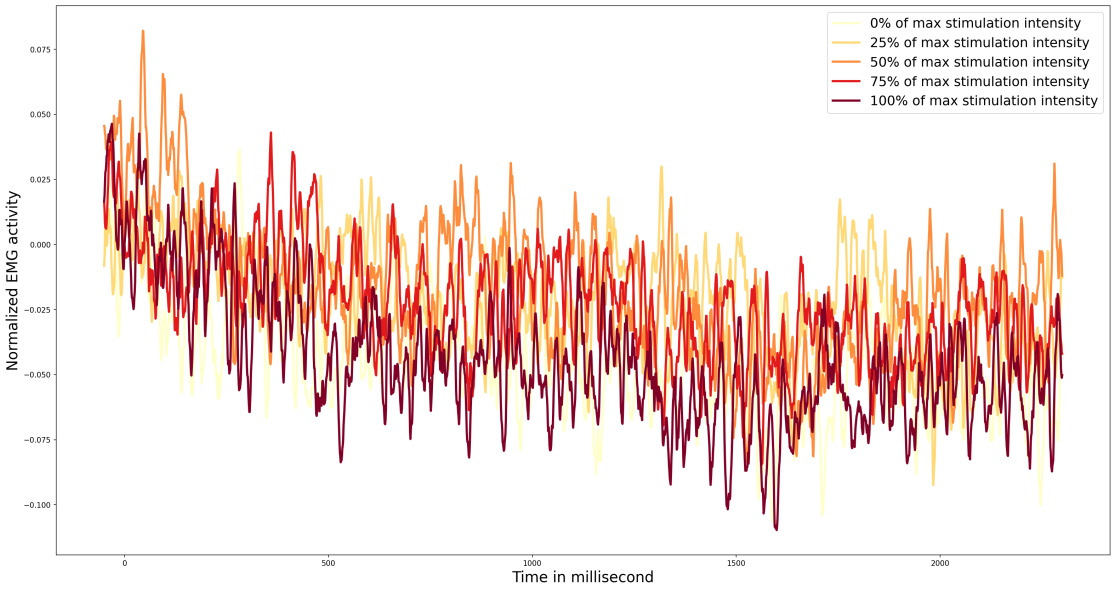


Figure 18: EMG activity when subject received a shock while picking up a fruit. Time 0 is the point when the fruit is picked up.

3.3.3 Dose-response effect on vigour

The dose-response effect is commonly observed in pain therapy and assessment, where the therapeutic method used is quantified against the pain intensity reported by subjects [107, 108]. In this study, the pain intensities varied across blocks. Since each fruit pick-up induced a painful shock, a corollary of the fear-avoidance model would suggest that subjects might avoid actions leading to pain, resulting in a reduction of collection rates as pain increases. However, as shown in Figure 19, the collection rate is not significantly correlated with perceived pain intensities.

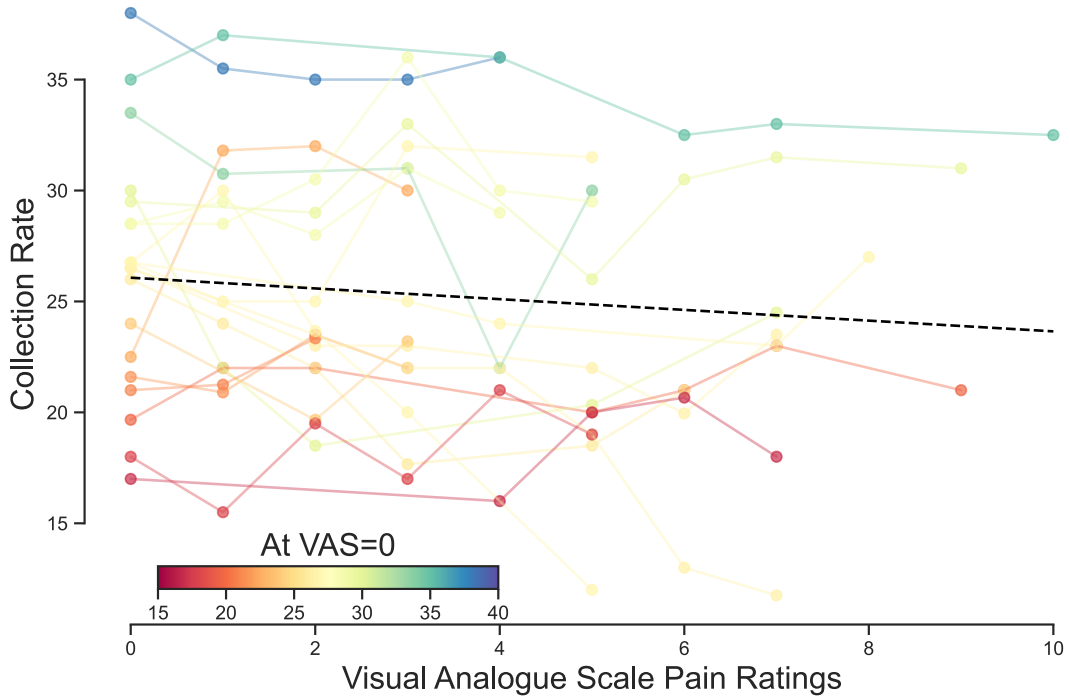


Figure 19: Pain intensity's dose-response effect on collection rate. Each coloured line represents one subject data with the colour being the value in pain-free condition (VAS=0). The dashed black line's slope and intercepts are fixed effect estimates from a linear mixed model. The slope's estimate shows an inverse relationship between pain choice probability and VAS rating, $\beta = -0.2419, 95\%CI[-0.699, 0.234], t = -1.049, df = 15.386, p = .311$

In addition to the collection rate, we also computed subjects' average hand speed as a measure of vigour. No significant correlations were found between stimulation intensities and hand movement speed.

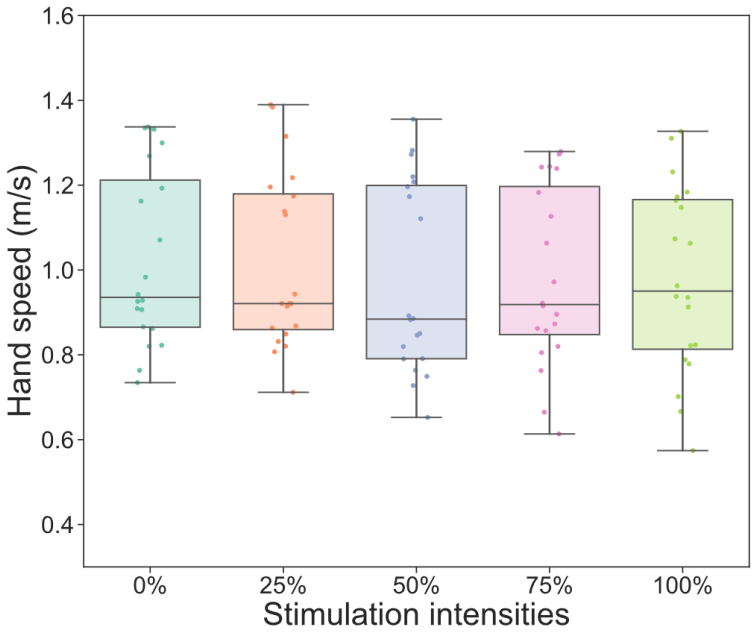


Figure 20: Average hand moving speed grouped by stimulation intensities. A linear mixed model shows the effect is also not significant enough to reject the null hypothesis, $\beta = -0.03935, t = -1.013, df = 19, p = .324$.

3.4 Discussion

In the following discussion, we highlight key findings and considerations regarding pain modulation, EMG analysis, and vigour response to phasic pain. A decrease in pain ratings during active tasks could support the modulation by decision conflict hypothesis, though alternative explanations like habituation or distraction effects cannot be ruled out. Challenges in EMG analysis, including device sensitivity and signal noise, limited the identification of reflex-related activity. Additionally, no significant changes in motivational vigour were observed across varying pain intensities, contrary to expected dose-response effects. These observations underscore the importance of ecologically valid environments and objective pain measurements, motivating further investigations detailed in Chapter 4.

3.4.1 Modulation by decision conflict

While it is encouraging to observe a decrease in pain ratings for in-block pain compared to pre-block calibration pain in the average over all stimulation intensities (Figure 17), as predicted by the modulation by decision conflict hypothesis, this finding does not rule out other possible explanations. For instance, one alternative explanation suggests that

attenuation to pain stimuli may be lower during active performance of actions compared to stationary settings [109]. We also note that, after applying false discovery rate correction for each individual stimulation condition, the only result that remained significant after multiple comparisons was the 75% intensity condition, which was also the only one significant without correction. While statistically significant results were obtained, the insignificant results suggest that the phenomenon could be more complex (e.g., not all variables were controlled) or that the data were particularly noisy. Nevertheless, this result provides potential evidence supporting the existence of a pain modulation mechanism driven by decision conflict.

Another critical point may argue that the multiple shocks experienced during the block, compared to only one in the pre-block calibration, could lead to habituation, resulting in decreased pain ratings. While we cannot completely rule out the possibility of habituation or sensitisation effects for subsequent pain stimulations, the block order was randomised, and the calibration stimulation was applied just before entering a new block. This should mitigate the bias introduced by general habituation and sensitisation effects. Furthermore, the first 10 training blocks had a fixed order of increasing and then decreasing stimulation intensities to account for the range effect, which should further reduce potential confounders.

One may also argue the rating for the stimulation was delayed for in-block stimulation compared to the calibration stimulation. For the initial shock in the block, it was a few tens seconds distant from the point inputting ratings, but there were other shocks that were just before the block ended. Post-experiment interview revealed some subjects reporting they were unable to recall the exact pain intensities for the in-block stimulations, it could also be used as an argument supporting the distraction (lowered attenuation) explanation. But it does not rule out modulation by decision conflict, as the putative target for endogenous analgesia is at the dorsal horn of the spinal cord [6, 17], the high level control in the cortical areas, may not aware that the pain signal is modulated.

Despite the questions raised for the methodology and various reasoning behind the discrepancy for ratings, it is clear that there is a difference in perceived pain intensities during actively performing goal-directed actions and in stationary settings. This proves the im-

portance of having a complicated ecologically valid environment to run experiments and treatment for pain. As there are question for ratings methodology for in-task pain, it motivates us to establish a new objective pain measurement independent of subjective ratings. This was one of the main motivations for Chapter 4's study.

3.4.2 EMG analysis challenges

As shown in Figure 18, the EMG data is noisy, and we were unable to identify the activity associated with the nociceptive withdrawal reflex. One possible reason could be that the EMG device we used was not sensitive enough to detect the EMG signal, as it was made in-house and no strict validation tests were performed to assess its sensitivity. A more likely reason is that the stimulation was expected, leading to a suppression of the reflex. Additionally, the in-block data was noisy due to the inclusion of all movement and motor actions unrelated to the reflex-induced activity. Consequently, we did not perform further analysis on the EMG data.

3.4.3 Unaffected vigour

The dose-response effect on vigour was another intriguing observation. It is commonly seen in pain pharmacology and rehabilitation, where researchers observe that a change in drug dosage or exercise intensity leads to a monotonic change in pain conditions [108, 107]. Phasic pain, as an aversive signal, would be expected to lead to a decrease in the number of shocks received as their intensity per shock increases. However, this effect was not observed in this experiment. This raises the question of whether phasic pain alters motivational vigour in healthy subjects. As shown in Figure 20, no significant changes in vigour were identified across different pain intensities. One possible explanation is that endogenous analgesia reduced the perceived pain, thereby undermining the dose-response effect. To further investigate this counterintuitive phenomenon, we designed the second experiment presented in Chapter 4, where we explore the dose-response effect in the context of aversive choices.

3.5 Conclusion

In this chapter, we designed and conducted a novel free-operant foraging instrumental task in which subjects were invited to collect as many fruits as possible, with each fruit pick-up inducing a painful shock. We found that the pain ratings for pre-block calibration stimulation were greater than the pain ratings for in-block stimulation at the same intensities. This outcome was predicted by the pain modulation by decision conflict hypothesis but could also be explained by mechanisms of attenuation to sensory stimuli. We did not observe a dose-response effect, which further motivates us to conduct a decision-making task aimed at constructing an objective model of pain measurement, as well as modulation of motvational values and vigour. These are described in detail in Chapters 4 and 6.

4 Phasic pain modulates motivational values in a realistic free-operant foraging task quantified by a novel computational method

4.1 Introduction

In the previous chapter, we demonstrated the intriguing discrepancy between ratings for pre-block and in-block pain using a novel paradigm. We also observed the vigour remains unchanged as the pain stimulation intensity goes up. We aimed to establish a more objective measure of pain within this complex paradigm to a) assist us in testing other hypotheses and analysing effects in the future, and b) provide an objective measurement of pain that could be clinically beneficial.

Drawing from the extensive decision-making literature that quantifies human rationale and predicts subsequent choices [110], a pain stimulus, serving as both a cue and an outcome, can be integrated into such a quantitative framework [67]. To quantify pain value, a common approach is to introduce choices within the task for subjects to select from. This approach connects our task to decision-making neuroscience research and enables us to utilise methods developed for value-based learning research [40, 110, 111]. Therefore, we introduced a new green pineapple as an additional choice that participants could opt for, where the original yellow pineapple no longer induced electric shocks.

In this way, we provided subjects with the option to endure pain while receiving a reward. By introducing a new options, we can differentiates the motivational value from vigour by measuring the collection bias for fruits with different pain stimulation intensities as motivational value and the usual moving speed as motivational vigour. We will present in the following sections that these two quantities are dissociable.

Fruits reset at the beginning of each block, and pain-free fruits were abundant for subjects to pick within the one-minute block. Taking pain now essentially became a trade-off between efficiently collecting as many fruits as possible (opportunity costs) and experiencing as little pain as possible. The opportunity cost is directly connected to vigour costs,

and it is hypothesised that the optimal balance between opportunity costs (which increase over time) and vigour costs (which decrease over time) determines the characteristics of free-operant behaviour [54]. This formulation effectively links the key concept of vigour cost with pain valuation, allowing us to establish a measure for phasic pain values.

4.2 Methods

4.2.1 Study design

Twenty-five subjects (17 female, mean age 25.71 years) recruited, and twenty-four subjects were included in the final analysis. We extended the study described in Chapter 3 by adding new fruit options: pineapples were categorized into two visually distinguishable types: 50% were green and 50% were yellow. The green pineapple was aversive, and picking one up immediately elicit a brief painful cutaneous electric stimulation to the proximal lateral side of the middle finger of the dominant hand. In contrast, picking up a yellow pineapple was always pain-free. The stimulation intensity when subjects pick a green pineapple follows the same 20-block design as in Chapter 3. The intensity of the electric stimulation was held constant within each block. The first 10 blocks were training blocks with deterministic stimulation intensity as described in Chapter 3 and the second 10 blocks were set with randomised stimulation intensities.

4.2.2 Experimental pain stimulation

Pain stimulation configurations were similar to stimulations in Chapter 3. For a few subjects we increased number of pulses to 100 pulses which gives 500 ms total time of stimulation, as we aim to induce pain for the max stimulation intensity to reach 7 on the VAS scale at the time of calibration. Due to the relatively low max voltage limit in DS5 (160V max with 100V max continuous stimulation) compared to other bipolar constant current stimulator (DS8R and DS7A has 400V max stimulation voltage), we can only increase the stimulation time for participants with high pain tolerance.

4.2.3 Skin conductance analysis

4.2.3.1 Data collection processing

Skin conductance was collected using a custom built wireless sensor. The microcontroller was an Arduino Nano 33 IoT. The skin conductance sensor was a Grove GSR sensor (Seeed Studio, Shenzhen, China). The firmware was written by the authors of this article with a designated sampling rate of 1000 Hz. Data was downsampled to 10 Hz and re-centred by the average baseline. Entire block of data was dropped out of analysis if the data loss is more than 20% of the total data due to temporary wireless transmission failure.

4.2.3.2 Effect coefficient estimate

As there are multiple consecutive pain stimuli delivered in a relatively close time period where skin conductance responses for different events can overlap, it is helpful to model the skin conductance response and construct a predicted time series of the skin conductance response for further analysis [112]. Bach et al. showed the skin conductance response can be modelled as a linearly time-invariant system with stimuli being the input to the system generating impulse responses [113, 112]. In this way, we constructed a time series data for the skin conductance response by by convolving the event trigger with the canonical response function published by Bach et al. [113].

To estimate the response magnitude for each event, generalized linear model (GLM) was used to fit the constructed skin conductance response time series. The GLM was fitted for each type of events separately. In this study, we fitted two main types of event: the first fixation time point of a painful fruit and the stimulation time point. The GLM used Gaussian distribution with an identity link function. We thresheld the fitted coefficients' p-values by 0.05. We further excluded coefficients with z score greater than 3.

4.2.4 Behavioural data and statistical analysis

Behavioural data was collected and analysed in the similar way as described in Chapter 3. In addition to the data types analysed in Chapter 3, in this study, the eye tracking data was the key data for our analysis. As we designed the study to encapsulate a decision

problem, unlike a traditional trial-based experiment, there was no clear point when did subject make a decision. Assuming fixation requires attention and drives decision-making process [114], we utilised the eye tracking data and projected the physical eye rotation to the virtual environment and identified the current fixated object. The time points of fixating on fruits were selected as key time points when the subject started evaluating the value of fruits in their mind.

Statistical analysis was carried out using custom scripts written in Python 3.9.13 and R 4.2.2. For linear mixed model, lme4 and lmerTest package was used [104] [105]. For all linear mixed models used in this article, fixed effects, random effects for intercepts and independent variables (IV) are estimated. The model also estimates the correlation between the intercept deviations and IV effect deviations across subjects. For example, aversive choice probability as the dependent variable and VAS ratings as the IV was fitted using the following R formula : `Aversive chocie probability ~VAS rating + (1 + VAS rating | Subject)`.

4.2.5 Computational model

A linear regression is performed at the aggregated block level. Within each one-minute block, each pickup, which could result in an immediate shock, is considered a trial. By analysing the data at the trial level, we can extract more detailed information and provide a parameterized behavioral description for each individual. This approach lays the groundwork for a robust, objective measure of phasic pain value within an ecologically valid context. In each trial, despite the presence of numerous lower-level controls required to complete the action, the primary decision involves choosing which fruit to pick up. We simplify the problem by assuming that there exists a set of decision points $\mathcal{S}$ as states, where the subject takes an action at a state $s \in \mathcal{S}$—choosing to pick up a fruit—and deterministically arrives at another state $s' \in \mathcal{S}$ if the block has not concluded. Based on the initial regression results, we assume that the reward is a linear combination of an internal reward r for each fruit, a negative utility function u representing the phasic pain value, and an effort cost function d. Therefore, the Bellman optimality equation for the action-value

function is

$$q_*(s,a) = r + C_p u(a) + C_m \cdot d(a) + \gamma \max_{a'} q_*(s',a')$$

As fruits are randomly distributed in our task, some foraging studies have suggested that animals may adopt a suboptimal greedy policy, opting for the closest fruit rather than solving the complex travelling salesman problem for a margianlly better optimal solution [115, 116]. To initially demonstrate the basic effects, we further eschew learning and future planning by setting $\gamma = 0$. One of the key simplifications is assuming the existence of a set of decision points in this free-operant task. To fit the model, we again utilised the eye tracking data, setting the decision point as the first time point when the subject saw a new fruit. Nevertheless, the task's available actions were partially observable, and subjects could explore for more actions if they were not satisfied with the currently available ones. This exploration action is not modelled in our formulation. To proceed with model fitting, we retrospectively identified the fruits that were picked up and selected only those time points when the fruits chosen afterwards were first seen as the decision points. We also kept track of the previous seen fruits, storing them in a memory queue, so the decision involved choosing the best option from these items in memory. Figure 23 shows the parameters fitted to the subjects' behavioural data.

As we disregarded planning, the model in Experiment 1 become comparing $C_p u(a) + C_m \cdot d(a)$ for each fruit at a particular time point. The first question is what are the fruits available for subjects to choose from. Fruits are abundant in the subjects' reachable area, however, many are not directly in subjects' vision field or hidden behind the trees. Subjects also need to identify the fruits' color to know whether the fruit is painful or not. Therefore, there is an additional search (or exploration) process that is not modelled here. We utilized eye tracking technology (HTC Vive Pro Eye), and assumes only fruits that were in the central vision field were considered by the subject. Fruits that were captured by eye tracking were saved into a first-in-first-out memory queue (the available choice set). We heuristically set the memory size to be 7, and the memory was cleared each time the subject chose a fruit. Next question is when does the subject makes a decision. The first time point that the subject looks at a pineapple not seen before could be used as the decision point in this free-operant task. However, we can not evaluate the model when the

actual fruit is not in the memory yet, as the model can never predict a correct choice out of a set of wrong choices. We can only retrospectively identify the fruit that the subject choose and only fit the model when the correct choice is in the choice set. This acausal problem is difficult to solve without taking time into consideration. We will show in the Experiment 2 model fitting, by quantifying the effort and opportunity costs as a function of time delay, the model-fitting process is causal.

We have converted the problem into finding a set of hyperparameters so that at each decision point, the maximum value of all choice differential values $C_p u(a) + C_m \cdot d(a)$ predicts the choice that the subject choose. Readers might find the problem similar to the multinomial logistic regression. However, as a decision model, we choose the objective function for the model-fitting to be the error rate (number of correct predictions divided by number of total predictions) rather than weighted least squares. The model is also flexible to take arbitrary parameters other than linear coefficients. As for this experiment, we choose u to be a sigmoid shape function with two additional x scale and translation parameters.

4.3 Results

4.3.1 Phasic pain modulate motivational values for choices associated with different pain intensities

We found that the probability of choosing a fruit decreased as the pain associated with it increased, participants avoided pain choices. Figure 21 shows an approximately linear decrease in aversive choice probability as a fucntion of Visual Analogue Scale (VAS) rating of pain stimuli where the aversive choice probability is defined as the number of painful fruits collected divided by the total number of fruits collected.

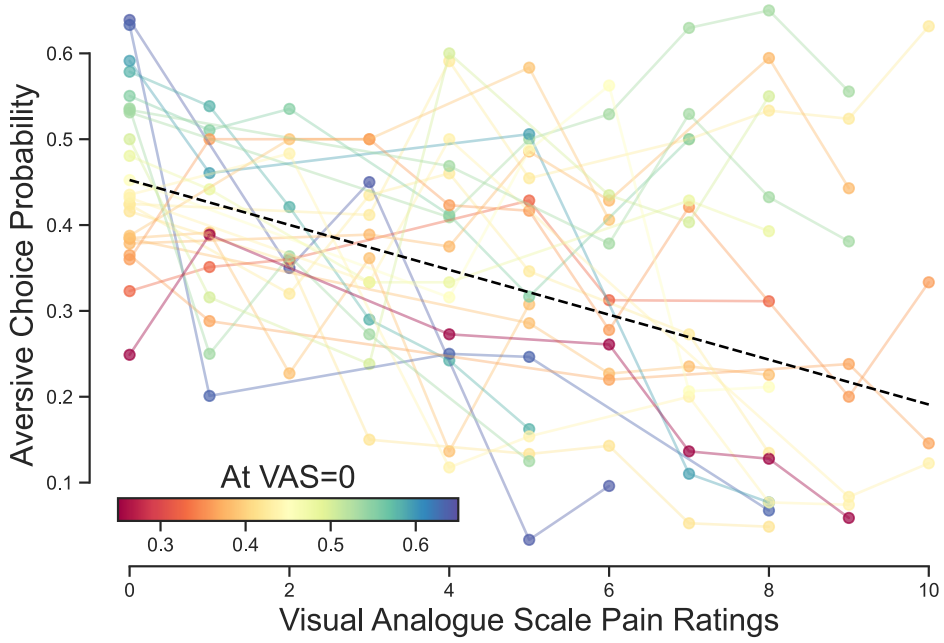


Figure 21: Each coloured line represents one subject data with the colour being the value in pain-free condition (VAS=0). The dashed black line's slope and intercepts are fixed effect estimates from a linear mixed model. The slope's estimate shows an inverse relationship between pain choice probability and VAS rating, $\beta = -0.0263, 95\% CI[-0.0383, -0.0144], t = -4.399, df = 21.269, p < .001$.

Next, since movement can be aversive due to its effortful and time-consuming nature (incurring opportunity costs), we investigated whether there is a relationship between moving distances and VAS ratings. We found that the egocentric distance differences between painful and non-painful fruits also depended on the VAS ratings. These distances were measured from the fruit to the subject at the moment the subject first looked at the fruit, as estimated from eye-tracking data, after collecting the previous one. As shown in Figure 22, the vertical axis represents the egocentric distances from non-painful fruits subtracted by those from painful fruits. As pain increased, participants traded off moving distance for pain.

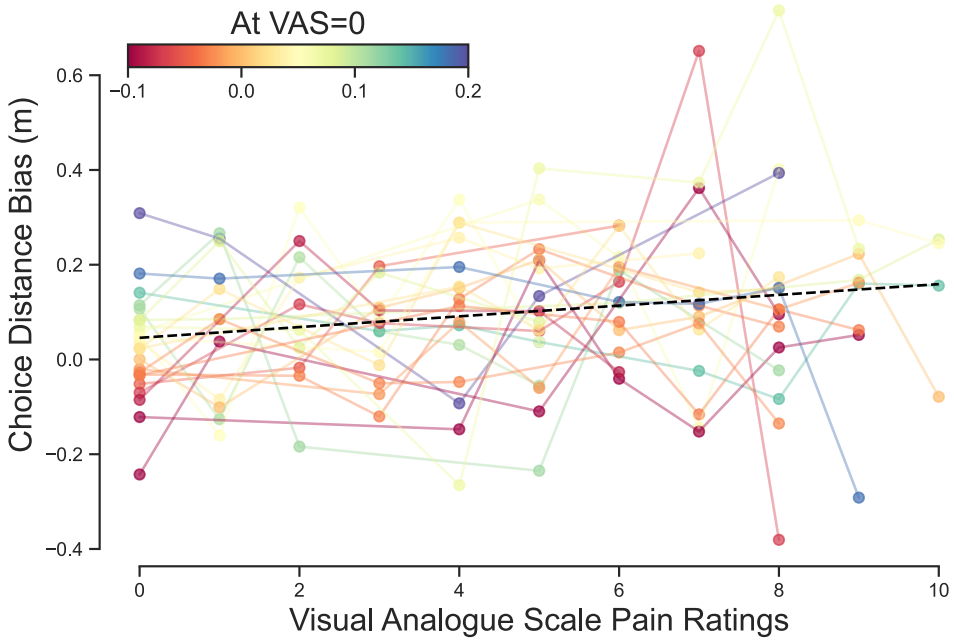


Figure 22: As described in Figure 21, the dashed line's slope is the fixed effect's estimate of choice distance bias, and it shows a positive relationship between the choice distance bias and VAS ratings, $\beta = 0.0113, 95\% CI[0.00294, 0.0197], t = 2.667, df = 29.359, p = .0123$.

4.3.2 Model fitting results and its correlation with physiological data

We have presented the regression results and identified two basic effect affecting choice values, we constructed a reinforcement learning model described in the methods and fit the model hyperparamters to the experiment data. Figure 23 shows the model fitting results.

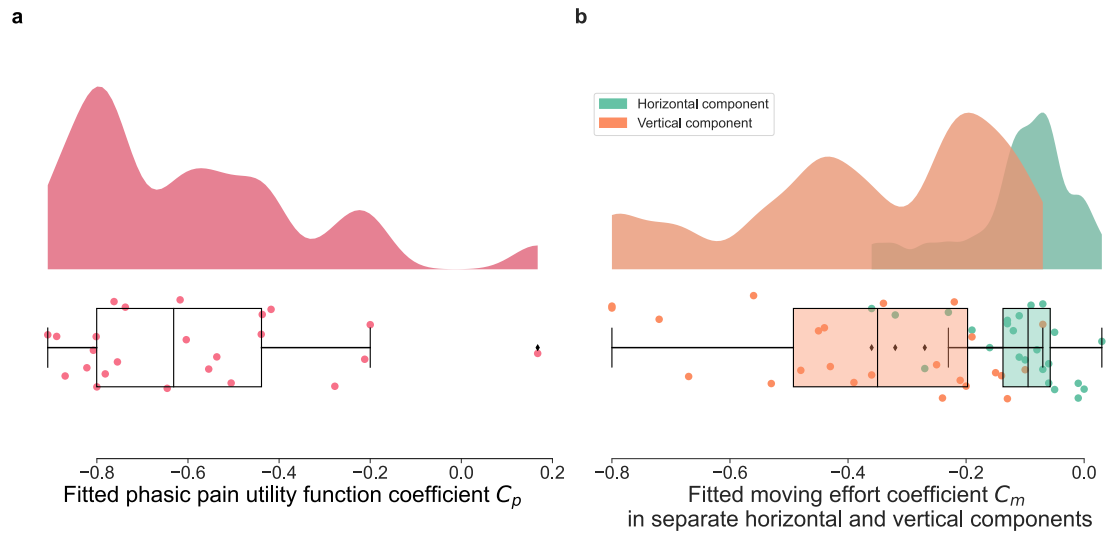


Figure 23: Model-fitting results. **a**, The negative phasic pain coefficients ($M = -0.59, SD = 0.26$) show the model captures the aversiveness of phasic pain stimuli in this free-operant decision-making task, $t(23) = -10.89, p < .001$, Cohen's effect size $d = -2.224$, and Bayes factors $BF_{10} = 66624133.080$. **b**, The moving effort coefficient C_m was separated into a horizontal component ($M = -0.11, SD = 0.10$) and a vertical component ($M = -0.37, SD = 0.22$) where the vertical component accounts for the distance moved vertically, and the horizontal component accounts for the distance moved horizontally. The fitted coefficients show lower effort cost to move horizontally than vertically, $t(23) = 6.72, p < .001$, Cohen's effect size $d = 1.372$, and Bayes factors $BF_{10} = 23463.388$

We further demonstrate that our model provides an additional measure of phasic pain values compared to the subjective pain ratings given by subjects. Figure 24 illustrates that the behavioral data fitted to our model exhibits a significant correlation with the subjective pain ratings. To elucidate the differences between subject-reported pain levels and the modelling values, we compared the evoked skin conductance responses (SCR) for shock events (at the time picking up a painful fruit) and fixation events (Figure 25). The fixation event is defined as the time point when the subject first saw a fruit, same as the decision point defined in the model. The change of skin conductance can be approximated by a linear time-invariant system. We can fit the real skin conductance data to a constructed time series by convoluting the event trigger with a known canonical response function (CRF) [113]. We then compared the fitted coefficient for each event against subjective pain ratings and the phasic pain value from the model. We found the model values explained more variance of fixation evoked SCR coefficient, while pain ratings explained

more variance of shock evoked SCR coefficient (Figure 26).

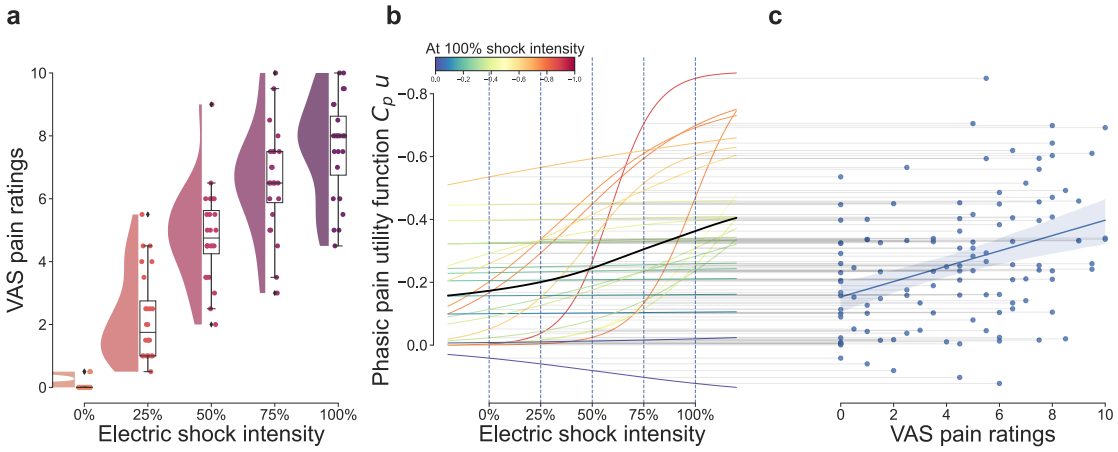


Figure 24: Fitted pain utility function compared with subjective ratings. **a**, VAS ratings at different electric shock intensities. **b**, Phasic pain utility values as a function of electric shock intensity. Each coloured line represents one subject's fitted curve. Black line is the average over all subjects. **c**, Linear regression shows significant correlation between VAS pain ratings and model estimated phasic pain values, $R^2 = 0.146, F(1, 118) = 20.13, p < .001$.

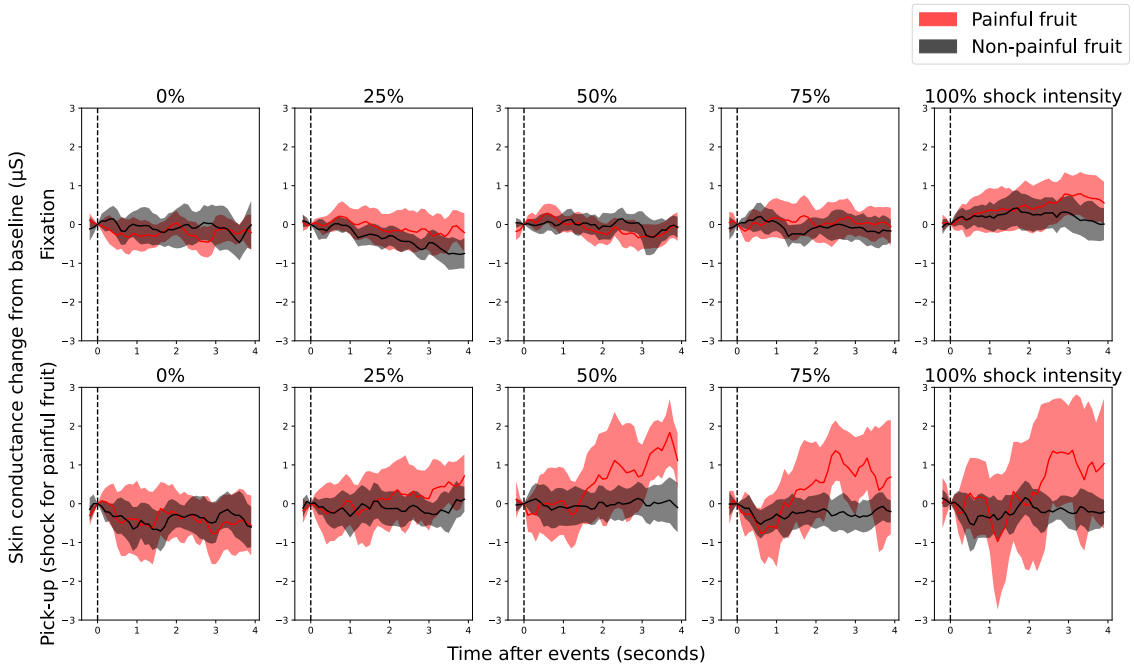


Figure 25: Evoked SCR for fixation and fruit pick-up events. Compared to seeing the visual cues, shock stimuli induced greater SCR when the shock intensities were high. Shaded area is the 95% confidence interval.

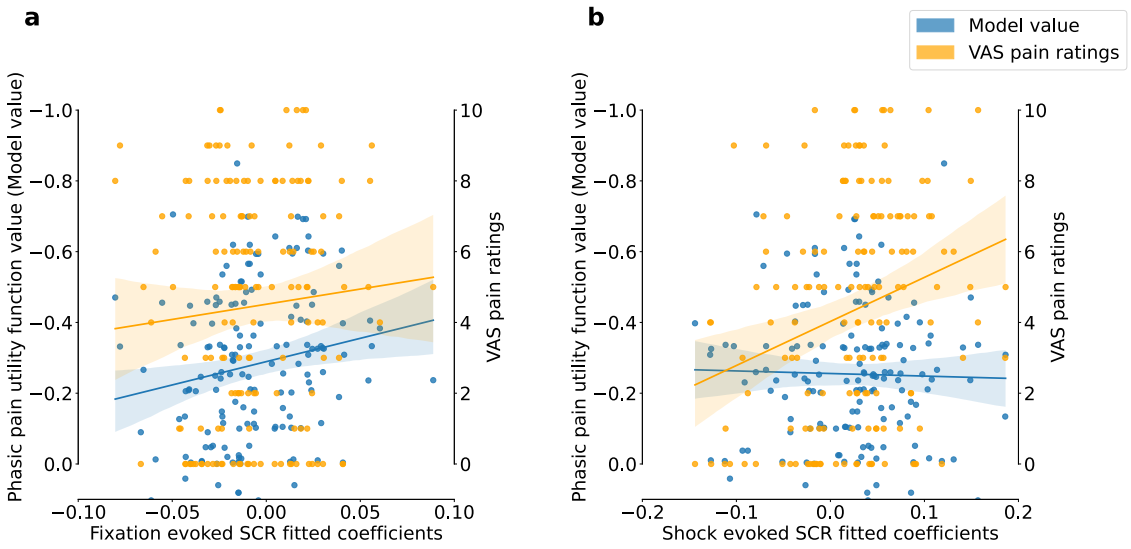


Figure 26: Skin conductance fitted coefficients' correlation with VAS ratings and modelled phasic pain value. **a**, ANOVA on linear regression results show modelled phasic pain values explained more variance ($F(1, 155) = 4.914, MSE = 0.00388, p = .0281$) of fixation evoked SCR coefficients than the variance ($F(1, 155) = 0.906, MSE = 0.0007343, p = .343$) explained by subjective pain ratings. **b**, In contrast, subjective pain ratings explained more variance ($F(1, 155) = 10.6, MSE = 0.04151, p = .00139$) of shock evoked SCR coefficients than the variance ($F(1, 155) = 0.092, MSE = 0.000383, p = .763$) explained by modelled phasic pain values.

4.3.3 Effect on vigour

The results above demonstrated the dose-response effect for phasic pain on the decision level. We did not identify a change in the moving speed in Chapter 3. However, in this experiment, we did found a significant effect for phasic pain on moving speed (Figure 27).

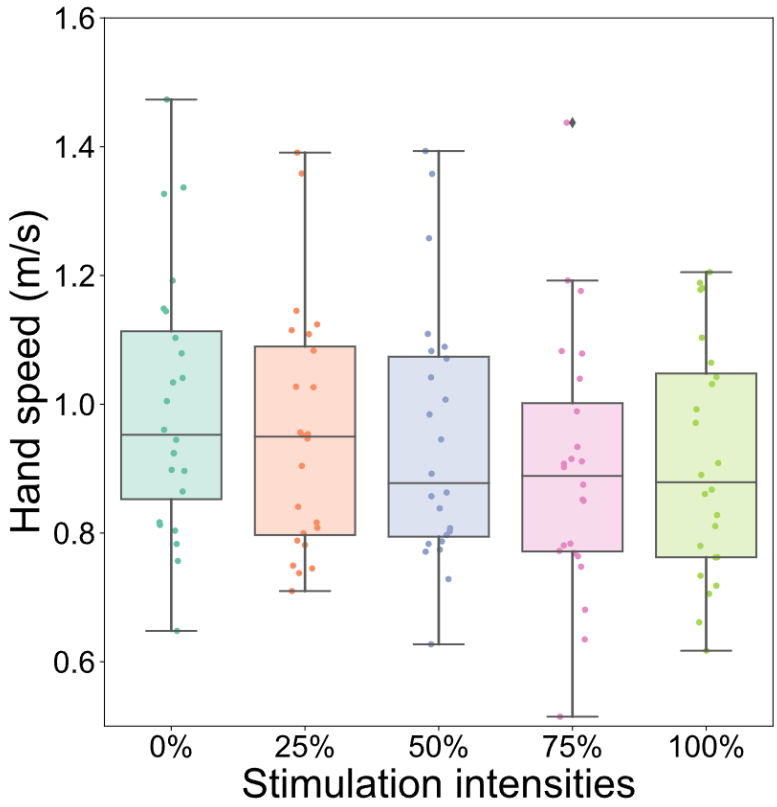


Figure 27: Average hand moving speed grouped by stimulation intensities. The linear mixed model shows a significant effect for phasic pain, $\beta = -0.0898, p < .001$

We tested whether the hand moving speeds were different for painful choices and non-painful choices, and the results showed that the hand speed reaching for a non-painful choice was not significantly affected across blocks (Figure 28), while the hand speed reaching for a painful choice was significantly affected by phasic pain intensities (Figure 29). Note that non-painful choices were always pain-free, and the phasic pain intensity setting per block only affected the shock intensity for painful choices.

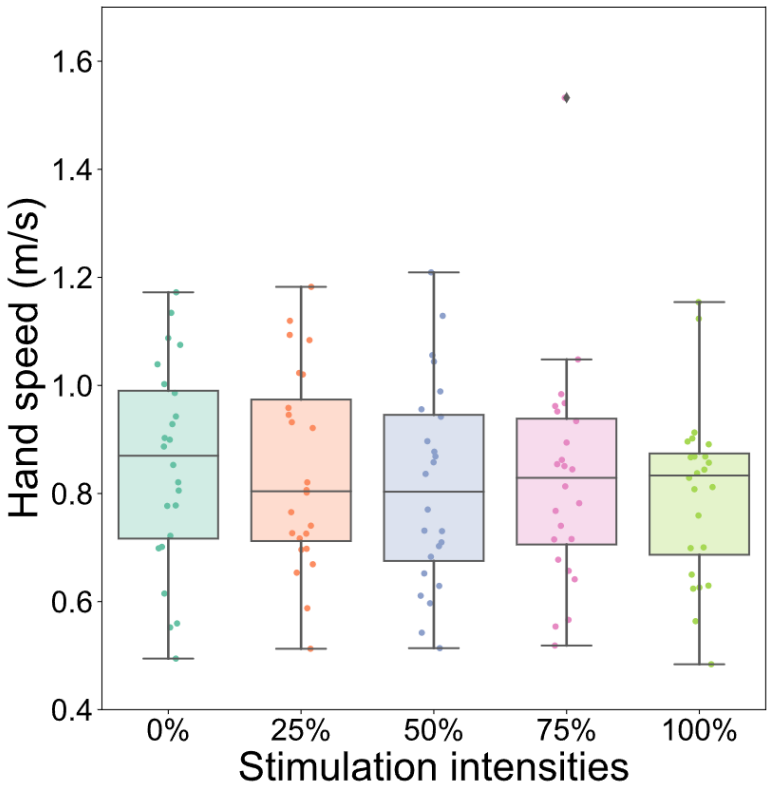


Figure 28: Hand moving speed for reaching non-painful choice. The linear mixed model shows the effect was not significant enough to reject the null hypothesis, $\beta = -0.04702, p = .133$.

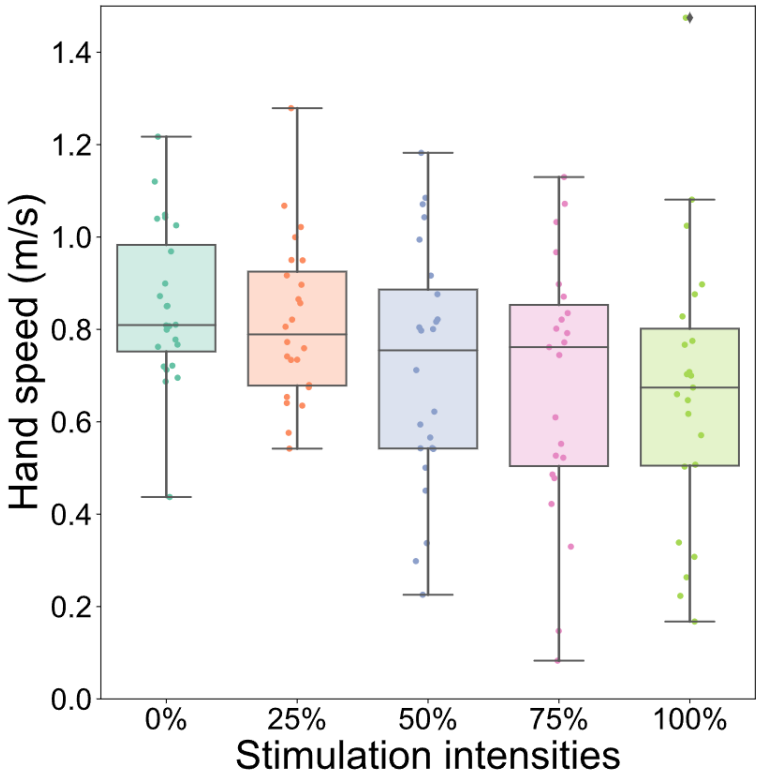


Figure 29: Hand moving speed for reaching painful choice. The linear mixed model shows a significant effect for phasic pain, $\beta = -0.224, p < .001$

We present an instantaneous speed visualisation for the above results. Visual inspection of the trajectories for reaching painful choices in different pain conditions reveals that they were more sparse than those for reaching non-painful choices, indicating a stronger effect of phasic pain and suggesting a selective modulation effect.

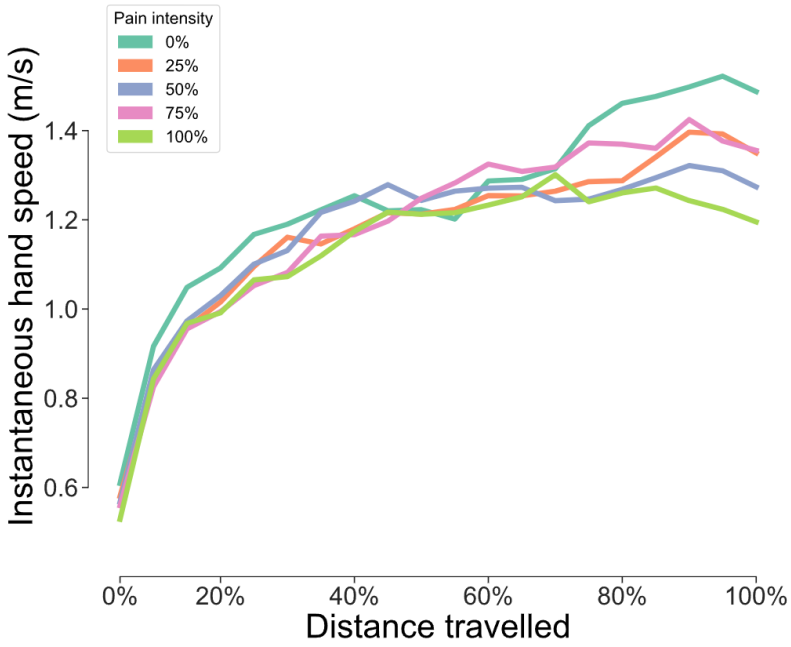


Figure 30: Visualisation of the instantaneous hand speed at different normalised travelling distances towards non-painful choices.

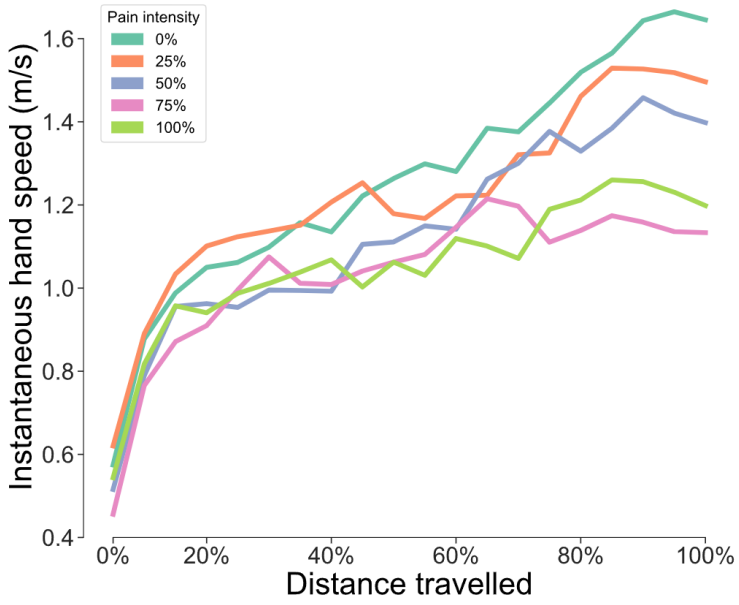


Figure 31: Visualisation of the instantaneous hand speed at different normalised travelling distances towards painful choices.

4.4 Discussion

This chapter presents the following key findings: phasic pain selectively alters motivational values for individual choices, and a computational model was established to capture this motivational value change. The effect of phasic pain on motivational value was measured through the trade-off between movement effort and pain. Additionally, movement distance was separated into vertical and horizontal components, revealing that vertical movement contributes more significantly to effort than horizontal movement. These findings provide a foundation for more advanced modelling of motor control in the context of aversive decision-making in future research.

4.4.1 Phasic pain selectively modulates motivational values

In Chapter 3, we observed that there was neither a significant dose-response effect for collection rates as pain intensity increased nor a significant dose-response effect for moving speed. However, by introducing a non-painful option, we observed a dose-response effect for the collection bias of painful choices. Additionally, a change in average moving speed across blocks as phasic pain intensity increased was observed, although it was not significant in Chapter 3. This effect appeared to be selective, showing a stronger effect for painful choices.

The motivational value generated by a value-based model guiding goal-directed actions has been studied extensively [117, 118, 119]. From the results of reward-based learning studies, considering pain as a negatively-valued reward signal, one would expect to observe a dose-response effect for aversive choices. Indeed, our results support the notion that phasic pain, as an aversive negatively-valued signal, modulates the motivational value for choices. Conversely, motivational vigour has attracted less computational research attention [100, 54], but it plays a key role in chronic pain and injury recovery. For instance, in the context of injury, it has been proposed that motivational vigour can aid recovery by reducing activity and hence suppressing unnecessary actions until healing occurs [120, 121]. The observation of a selective modulation for moving speed, as a metric for vigour, motivates us to conduct a new study incorporating tonic pain to determine

whether vigour can be modulated by tonic pain.

4.4.2 Interpretation of model parameters

4.4.2.1 The motivational value estimated by the model

The fitted model parameters provide an objective measurement based on the trade-off between pain and vigour costs. The fitted pain utility function offers a detailed quantitative measurement of subjects' pain (Figure 23a). As shown in Figure 24, the pain utility functions were correlated with subjective pain ratings. However, there remains a subtle difference between these two objective and subjective measurements, as revealed by skin conductance data. The skin conductance data indicate that the fitted pain utility function is significantly correlated with the skin conductance response at fixation, presumably the point at which subjects make decisions by calculating expected pain values, but not at the post-shock point. Pain ratings, by contrast, show the opposite correlation significance. As the fixation events coincide with the assumed decision time points in our model, this suggests that the motivational value from the decision model correlates more strongly with the fixation-evoked response—an expectancy effect occurring before the onset of phasic pain—rather than with the shock-evoked response following the phasic pain. This implies a distinction between the phasic pain perceived and reported by the subject and the motivational value that phasic pain carries as an aversive signal.

4.4.2.2 Separated moving distance components for vigour cost estimation

Furthermore, by analysing the data with the decision model at the trial level, we can separate the vigour cost into horizontal and vertical components. For the same movement distance, the vertical direction is more effortful than the horizontal direction (Figure 23b). While this result is not particularly surprising, it demonstrates the potential to analyse movement patterns in a value-based framework. This approach will be one of the key measurements used in Chapter 5 to assess movement pattern differences by delivering phasic pain stimulation to different sites of the body.

4.4.2.3 Lack of time factor in the free-operant computational model

Lastly, while the model successfully captures two main effects and demonstrates its strength in modelling decision values, it lacks a time factor. No time delay was estimated in the model for action execution or the gaps between actions. For the model fitting, we could only wait to compute the value functions for each fruits in the memory until the next fruit pick-up was seen by the subject to generate meaningful trial-based fitting data. This violation of causality should be addressed by introducing the time factor into both vigour cost and opportunity cost [54]. We will provide a solution to this problem in Chapter 6.

4.4.3 Non-zero bias

If subjects are unaware of the choices' associated aversive values (in our task, the green ones were always painful, and subjects understood this from initial training blocks), in an ideal condition, the aversive choice probability (Figure 21) should be 0.5 at VAS=0, and the choice distance bias (Figure 22) should be 0 at VAS=0. The non-zero bias shown in Figure 21 and Figrue 22 were likely caused by the bias for aversive choices carried from previous experience. A possible solution could be randomize the colour for aversive choices so that the bias can be maximally reduced.

4.5 Conclusion

In conclusion, we extended the study from Chapter 3 into a decision-making task by introducing new options that allowed subjects to choose between variable pain intensities. This approach enabled us to observe the selective modulation effects of phasic pain on motivational values and vigour—two key concepts in aversive decision-making and in chronic pain and recovery, respectively. Additionally, the novel computational model we developed offers a quantitative account of behaviour that could be clinically useful by providing an alternative evaluation of pain.

5 A comparison of different modalities of phasic pain's values and their impact on behaviour

5.1 Introduction

As discussed in Chapter 1, pain is topographic and embodied. Different locations of pain stimulation can activate various muscle groups, completing the nociceptive withdrawal reflex [79, 80]. Additionally, phasic pain, characterised by the activation of different fibres, can induce distinct sensory experiences, influencing the ecological validity when paired with the context. Building on the computational models validated in Chapter 4, we now possess the tools to quantify different types of phasic pain and their effects on behaviour.

The first property of pain we focused on was the location of the pain. Unlike conventional screen-based tasks, all sensory stimuli constructed in the VR paradigm possess a natural three-dimensional spatial component. Specifically, we aimed to study the behavioural and physiological differences between pain stimulation applied to the finger and the lower back. There are two hypotheses in this study. The first is from a physiological testing perspective. Pain stimulation at different body locations has been widely studied; for example, spatial acuity of pain is known to vary significantly across the body [122]. However, most studies have been conducted in stationary settings. By applying pain to different body locations during a real-world motor task, we can examine spatial pain perception differences in a realistic scenario. As we observed in Chapter 4, the vigour cost coefficients for vertical movement were lower than those for horizontal movement. We predicted that stimulation on the back would increase the aversiveness of vertical movement. This hypothesis assumes that pain stimulation on the back may indicate back injuries, leading the motor control system to limit back movement.

Secondly, from a learning theory perspective, pain perception is modulated by prior information from the context [67], and thus, the spatial information of the pain stimuli should influence the interpretation of the stimuli and the subsequent post-stimuli behaviour. Therefore, we were interested in determining whether there is a difference in the fitted pain utility function for back and finger pain. As a pineapple has a spiky shape, it is more

ecologically valid to experience prickling pain on the finger rather than on the back. We predicted that there would be an increase in the aversive value of pain on the finger compared to the back.

The second pain property that we are interested to test is the stimulation type. The type of fibres activated by the pain stimulation can form distinct experience. For example, wasp electrodes that has a small contact surface and often delivers sharp, immediate prickling pain, presumably by activating mainly Aδ fibre [17] while a gel pad electrode may activate c fibres as well, providing an additional vibrating sensation. Such difference may contribute to differentiable ecological validity. We hypothesized the prickling pain stimulated by Wasp electrodes provide higher ecological validity because it matches the pins and needles pain when stabbed by a pineapple's spiky surface. An increase of ecological validity should enhance the aversiveness of pain as its has a high chance of danger which is encoded in prior experience.

5.2 Methods

5.2.1 Study design

Twenty-three subjects were recruited, and all subjects were included in the final analysis. We extended the study described in Chapter 4 by adding back pain stimulation and an additional type of pain stimulation using gel pad electrodes. Optional non-painful fruit stimuli were preserved, as in the study from Chapter 4. We also reduced the number of pain stimulation levels to three: no pain, low pain, and high pain, to accommodate the additional location conditions. The pain location study followed a within-subject design, with experimental blocks of either finger pain or back pain randomised and tested within subjects. Consequently, each subject was tested under five conditions: no pain, low pain on the back, low pain on the finger, high pain on the back, and high pain on the finger. For stimulation types, we used a between-subject design: nine subjects used wasp electrodes, and the remaining fourteen subjects used gel pad electrodes.

Since the number of phasic pain conditions was reduced, we optimised the experiment by decreasing the number of training blocks from ten (two repetitions of all conditions)

to five (one repetition of all conditions). Ten randomised blocks following the training blocks were analysed.

5.2.2 Experimental pain stimulation

In this chapter, the phasic pain's modality is the central variable to test. We will describe each type of stimulation separately.

5.2.2.1 Finger pain stimulation

For finger pain stimulation using wasp electrodes, the stimulation was delivered to the same site as in Chapter 3 and 4, the proximal lateral side of the middle finger of the dominant hand. Because gel pad electrodes are bigger in size, they are attached to the intermediate and proximal dorsal side of the middle finger of the dominant hand.

5.2.2.2 Back pain stimulation

Back pain electrodes were attached at the point where the spine crossed the line joining the superior aspect of the iliac crests posteriorly (the intercristal line), commonly found to be the L4-5 spine level [123]. Tapes and bandages are used to ensure their contact with the skin.

5.2.2.3 Wasp electrode stimulation

Wasp electrodes has a circular shape and is small in size. It has an outer ring of 6.5mm diameter and an inner ring of 2.0mm diameter. Its small contact area causes high impedance and high energy density.

Because of its high energy density, we informed the subjects that it is possible that the electric pain stimulation can leave a temporary mark. Specifically, in this study, we noticed the following factors:

- In previous studies, we all used wasp electrodes and had no problems. In this study, to stimulate two body sites randomly across blocks, we had to use two stimulators. Two stimulators were identical and manufactured sequentially (SN: DS5-553-2019

and DS5-554-2019). However, DS5-553-2019 showed a faster DC drift when in idle.

- Digitimer DS5 works by delivering the current specified by the level of input voltage. This allows programmable stimulation wave form and stimulation intensities. However, as the input voltage line contains noises, the DS5 may attempt to deliver a small current.
- It is not a problem if the circuit has low impedance as the small current generated by the noisy voltage signal can be easily achieved by applying a small voltage. However, when the impedance is high, the stimulator can increase the voltage and maintains the DC voltage until it is manually zeroed.

After identifying these factors, it is recommended to

- Perform sufficient skin preparation with abrasive pads and isopropyl alcohol to reduce impedance.
- Keep monitoring the DC voltage at idle and frequently zero the DS5 stimulator manually.
- Use a trigger-based stimulator, for instance, DS8R, so that no current will be generated with small noises in the channel.
- Frequently ask subjects how do they feel on the stimulation site. For a small electrical burn, human subjects usually report an itching sensation.

These recommendations apply to other high impedance electrodes as well, for example, high frequency stimulation (HFS) electrodes.

5.2.2.4 Gel pad stimulation

20x25mm disposable solid gel electrodes were used.

To make within subject pain stimulation comparable, we calibrate pain ratings every five blocks in which all five conditions are randomised within each five blocks. We calibrate the stimulation intensities for finger and back pain respectively to match each other's subjective ratings.

5.2.3 Behavioural data analysis and statistical tests

Behaviour data analysis and statistical tests are carried out in the same way as in Chapter 4. For repeated measures ANOVA analysis, pain intensities and stimulation locations are within-subject, and stimulation electrode types are between-subject. The main difference is in model-fitting, we fit back and finger pain's parameters separately for each individual.

5.3 Results

5.3.1 Aversive choice probability

Before comparing the model-fitted parameters, we first performed a sanity check to ensure that the task was running as expected, as in Chapters 3 and 4. One of the main metrics for this free-operant decision-making task is to observe a dose-response effect by increasing pain stimulation intensities.

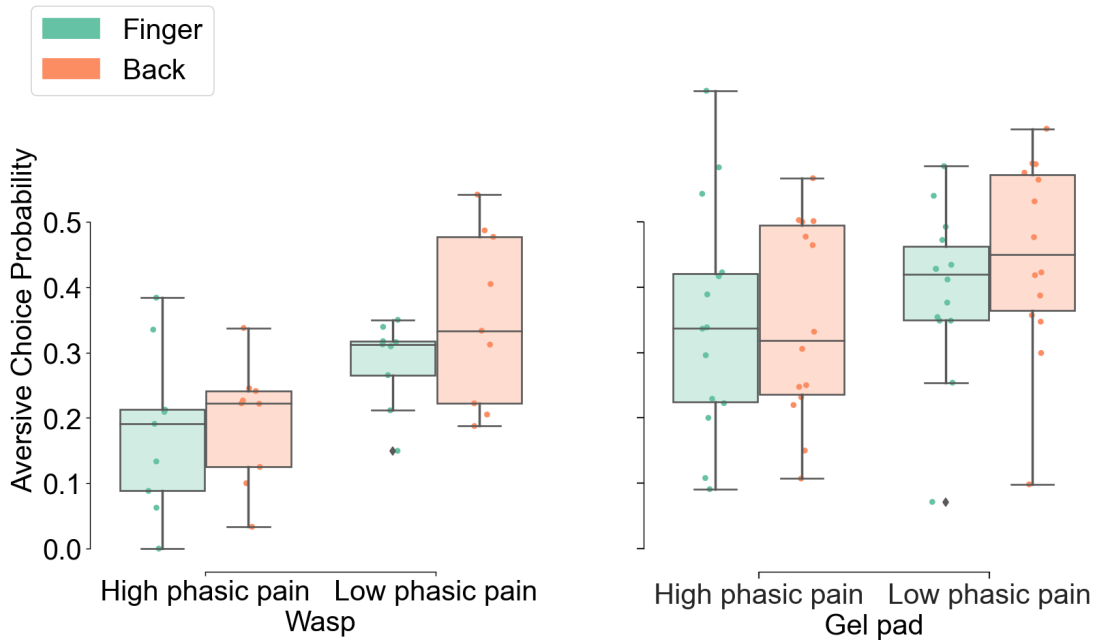


Figure 32: Aversive choice probability for different conditions. Three-way repeated measures ANOVA shows that the effect for phasic pain intensities was significant, $F(1, 21) = 11.507, p = .003$, and the effect for electrode types was significant, $F(1, 21) = 10.170, p = .004$. No significant effect for pain locations was identified and no significant interaction was identified.

We expected to observe a significant effect of pain stimulation intensities on choice probability. The noticeable difference caused by electrode type was also interesting. Although

electrode type was a between-subject condition, we controlled for pain ratings in the low and high pain conditions. This result suggests that the wasp electrodes were more aversive in decision values, specifically in this task, than the gel pad electrodes for the same pain ratings.

5.3.2 Compare horizontal and vertical moving cost coefficients

We compared the horizontal and vertical moving cost coefficients. In Chapter 4 Figure 23, we saw a significantly lower vertical moving cost coefficient than the horizontal one. As described in Chapter 4, the vertical component measures the distance in vertical direction required to reach a fruit, while the horizontal component measures the distance in horizontal direction required to reach a fruit. In this experiment, we saw the same trend but the results were not significant enough to reject the null hypothesis. We also performed two-tailed t-test for each electrode type separately, no significant results found.

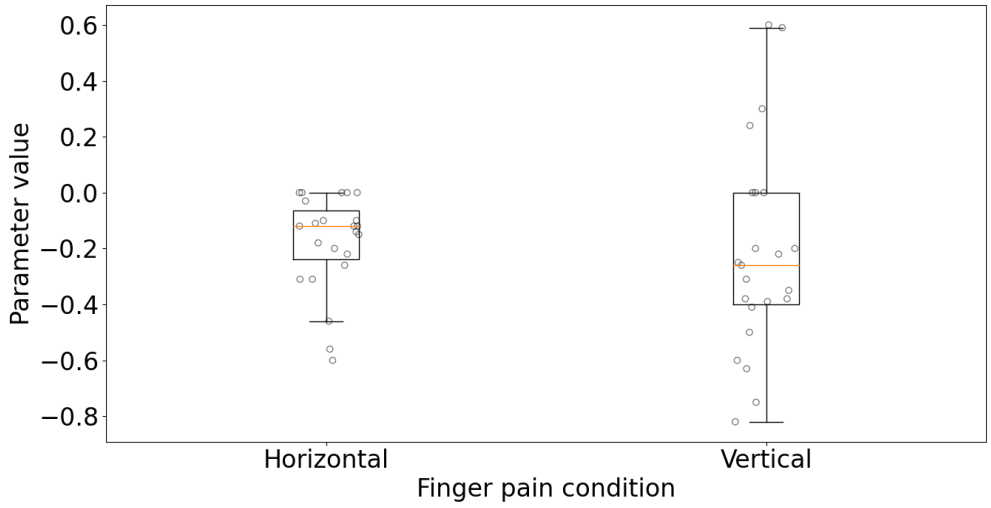


Figure 33: Horizontal and vertical moving cost coefficients comparison in finger pain condition including both wasp and gel pad electrodes conditions. The horizontal component showed higher coefficient values trend (less aversive) than the vertical component but not significantly enough to reject the no difference null hypothesis, two-tailed $t(22) = 0.381, p = .707$. Cohen's effect size $d = 0.079$, and Bayes factors $BF_{10} = 0.234$.

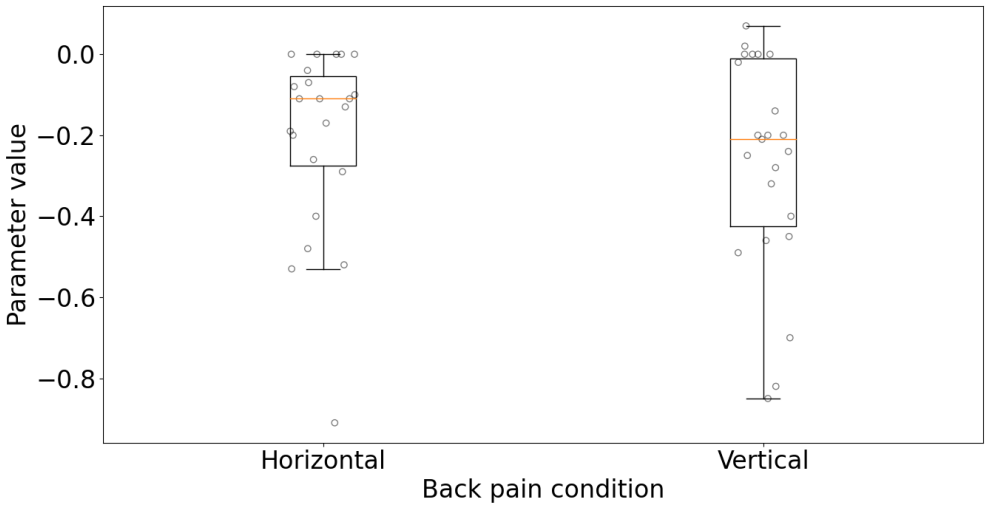


Figure 34: Horizontal and vertical moving cost coefficients comparison in back pain condition including both wasp and gel pad electrodes conditions. The horizontal component showed higher coefficient values trend (less aversive) than the vertical component but not significantly enough to reject the no difference null hypothesis, two-tailed $t(22) = 0.935, p = .360$. Cohen's effect size $d = 0.195$, and Bayes factors $BF_{10} = 0.324$.

5.3.3 Compare pain utility function

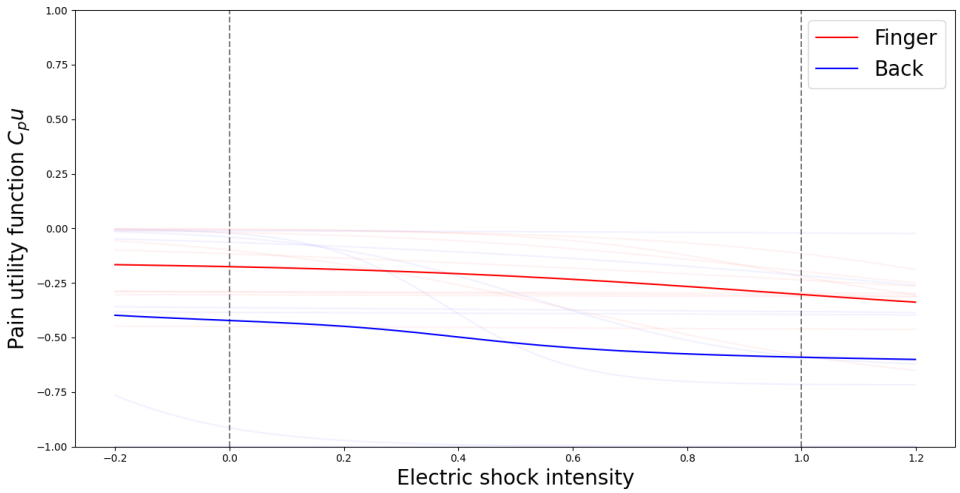


Figure 35: Pain utility function for wasp electrode. Two tailed t-test at zero stimulation intensity ($t(8) = 2.148, p = .0639, d = 0.716, BF_{10} = 1.490$) and at one stimulation intensity ($t(8) = 2.267, p = .0531, d = 0.756, BF_{10} = 1.814$) shows the aversive value for the back pain was more aversive than the aversive value for the finger pain given the same pain rating level

Initially, the result appeared counterintuitive. However, it is important to note that the electric shock intensity was normalised to the high phasic pain's electric current, where

ratings were calibrated to match each other (finger and back) within subjects. This suggests that, for the same level of pain ratings, stimulation on the finger had a less aversive decision value. A possible explanation is that the pain on the finger, when using Wasp electrodes, was more ecologically valid compared to the pain on the back. Although we have very few samples, Figure 36, which lacks this effect, seems to support this idea, as the gel pad electrodes were considered less ecologically valid in this context.

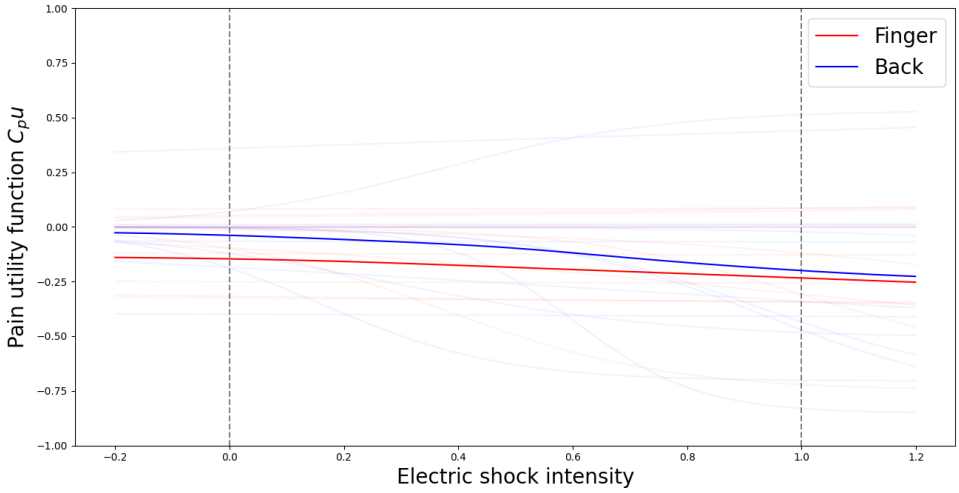


Figure 36: Pain utility function for gel pad electrode. Two tailed t-test at zero stimulation intensity ($t(13) = -1.088, p = .296, d = -0.291, BF_{10} = 0.370$) and at one stimulation intensity ($t(13) = -0.198, p = .846, d = -0.053, BF_{10} = 0.223$) did not show a significant effect for stimulation locations as shown in Figure 35

.

5.4 Discussion

This chapter presents our initial attempt to deliver stimulation targeting various pain modalities and examine their effects on healthy subjects. Although, due to technical and time constraints, the experiment did not collect a sufficient amount of data, it demonstrated methods for exploring the diversity of pain in a realistically simulated task. Moreover, it successfully identified significant factors contributing to the behavioural metrics established in Chapters 3 and 4.

5.4.1 Demands for a large amount of data

In this chapter, we gained an overview of the high diversity of data modalities. Pain locations [79, 80] and types of pain all played crucial roles in determining the effect of pain sensations on human behaviour. Each dimension of pain properties can be categorised into multiple distinct types or even parameterised by continuous real variables. The number of combinations of pain properties becomes vast and even intractable for conventional hypothesis-testing experiment designs. This phenomenon was described by Jonas and Kording in their comparative analysis of neuroscience research methods and microprocessor reverse engineering [39].

Enumerating all possible pain properties to conduct hypothesis testing is likely unnecessary to achieve a sufficient understanding of pain for treating diseases and developing safe robotics. Nevertheless, it is important to efficiently collect large amounts of data to apply a data-driven approach. Utilising modern, commercially available devices, such as a virtual reality suite accessible to general consumers, appears to be a promising avenue.

5.4.2 Ecological validity modulates decision values

Although we had a limited sample size, the results presented in Figures 32, 35, and 36 showed that the type of pain stimulation had a significant effect on the decision values, even when the pain ratings were the same. The pain ratings were recorded while subjects were stationary, whereas the decision values compared here were estimated from performing a free-operant decision-making task. By combining physical pain stimuli with a simulated virtual context, the properties of the pain stimuli could contribute to the overall ecological validity—how well the pain stimuli fit within the context. We then hypothesised that it was the differences in ecological validity, attributed to the properties of the pain, that modulated the decision values associated with the pain.

6 A unified model accounting for distinct behavioural functions of phasic and tonic pain in healthy individuals

6.1 Introduction

As a continuous form of pain, tonic pain is closely linked to chronic pain conditions. However, studying tonic pain in a well-controlled environment without causing harm to subjects has been challenging. Over the past few years, researchers have developed various methods to experimentally induce tonic pain [124, 125, 126]. In this chapter, we integrate tonic pain stimulation into our paradigm. Alongside the previously established free-operant instrumental decision-making task based on phasic pain, we have created a complex research platform incorporating multimodal pain stimulations with an ecologically valid task. The integration of tonic pain into this novel task also provides crucial insights into the nature of tonic pain.

6.1.1 Tonic pain's effect on motivational vigour

As discussed in Chapter 1 on pain functions, despite its unpleasant nature, there is wide recognition that pain serves a useful function. This is most clearly the case for short-lasting phasic pain, where incoming nociceptive signals warn of impending tissue damage, elicit rapid defensive responses, and drive learning that reduces future chances of encountering harm. This behaviour is objectively measurable through conditioning and avoidance behaviour, which quantifies pain in terms of its motivational value.

In Chapter 4, we demonstrated that it is possible to dissociate phasic pain's effect in modulation of motivational values in a free-operant instrumental decision-making task. Specifically, by introducing variable phasic pain to choices, we observed a dose-response effect on aversive choice probability, a monotonic change in decision values due to phasic pain, which validates phasic pain's effect on motivational value. We also observed a change in moving speed and showed the moving speed for reaching to fruits, as a potential metric of motivational vigour, is selectively modulated by phasic pain.

It is, therefore, intriguing to study whether tonic pain has an overall effect modulating the

motivational vigour. In ecological and ethological contexts, tonic pain is generally considered to serve a recuperative function. For instance, in the context of injury, it has been proposed that it can aid recovery by reducing motivational vigour and thereby suppressing unnecessary activity until healing occurs [120, 121]. In the past, with conventional study paradigms, the adaptive function of tonic pain has been much harder to quantify. However, as we have established a novel ecologically valid paradigm, studying the complex function and interaction between phasic pain and tonic pain has become promising. In this study, we extended the experiment in Chapter 4 by simultaneously delivering tonic pain (via an inflated pressure cuff) to the non-dominant arm, aiming to emulate a mild injury. We also extended the computational model to quantify both the aversive value of phasic pain and the reduced motivational vigour of foraging overall. In this way, we aim to computationally dissociate these two distinct components of adaptive motivated behaviour.

6.1.2 Punishment sensitivity under the influence of tonic pain

Punishment sensitivity is defined as the sensitivity to aversive signals as an outcome of actions in a decision-making task [127]. In our task, the aversive signal was the phasic pain incurred by making a painful choice. Therefore, the canonical measurement of its sensitivity was the aversive choice probability, which was used as a behavioural marker for changes in motivational value in Chapter 4. Mancini et al. [128] show that punishment sensitivity increases in chronic pain patients compared to healthy subjects. If tonic pain resembles chronic pain conditions, we would predict an increase in punishment sensitivity during experimental blocks with tonic pain compared to those without. However, it is also possible that no effect will be observed, as we are testing on healthy subjects with only one minute of continuous tonic pain stimulation, orders of magnitude shorter than the pain experienced by chronic pain patients.

6.1.3 Conditioned pain modulation

Most pain researchers would recognise that the experimental design in this chapter shares similarities with a conditioned pain modulation paradigm, where a phasic pain stimulus is

commonly used as a test stimulus and its perceived intensity is compared across conditions with or without a tonic conditioning stimulus [129]. In our case, the electric phasic pain serves as the test stimulus, while the tonic pain serves as the conditioning stimulus.

A positive conditioned pain modulation effect predicts that the perceived intensities of the phasic pain should be reduced when the tonic pain stimulus is present. However, aside from the fact that the conditioned pain modulation effect remains controversial, our experiment is only related to the methods of pain stimulation used in conditioned pain modulation, with a fundamental distinction: our task is a free-operant instrumental active task. In fact, our results showed an inverse trend compared to conditioned pain modulation, where the ratings for phasic pain increased when tonic pain was present. This result should serve as an important reference for extending the conditioned pain modulation paradigm to non-stationary active settings.

6.2 Methods

6.2.1 Study design

The study follows similar design as in Chapter 3, 4 and 5 and follows the base task design described in Chapter 2. 33 subjects (31 in final analysis, 15 female, mean age 24.81 years) performed a free-operant instrumental foraging task to accrue reward points and avoid pain. The study is divided into 24 one-minute blocks where randomised phasic pain and tonic pain conditions were assigned to the block. As in Chapter 5 that we reduced the number of phasic pain levels to three (no pain, low pain, and high pain), to accommodate the additional tonic pain conditions. These three phasic pain levels were then combined with two tonic pain conditions (with and without tonic pain) to form a factorial design experiment with a total of six different combinations. The pain conditions were fixed within each block. The six combinations were initially presented in a fixed order as training blocks, then the order was randomized and repeated three times, resulting in a total of 24 blocks.

6.2.2 Experimental pain stimulation

6.2.2.1 Electric phasic pain

Electric pain stimulation used the same method described in Chapter 4. Instead of calibrating a maximum pain stimulation current and equally dividing the current into five levels, we calibrate a high pain and a low pain stimulation current with a staircase procedure [103]. We aimed to target a rating of 7 on a 0-10 scale for high pain condition and a rating of 3 for low pain condition. The calibration were conducted every six blocks to provide consistent ratings for high and low pain conditions.

6.2.2.2 Tonic pain stimulation

We used pressure cuffs (VBM, Sulz, Germany) to safely induce tonic pain [130, 131]. The cuff was wrapped around the non-dominant arm's biceps. The cuffs were inflated prior to the start of each required block and immediately deflated after the block ended. Subjects were asked to report verbally when they felt discomfort and when they felt pain as the researcher manually inflated the cuff. Inflation stopped once the subject reported pain. To control for variables associated with wearing the cuff other than pain, pressure cuffs remained wrapped around the subjects' arms even in blocks without tonic pain.

Although the pressure cuffs has been used to safely induce tonic pain, it was not practiced in Oxford and not an approved procedure specified by the Central University Research Ethics Committee. In order to convince the ethics committee about its safety, we spent significant effort to prepare the documents justifying that the pressure cuffs can be applied safely within the limits based on previous research. Key details are attached in the Appendix.

6.2.3 Statistical analysis

Statistical analysis was carried out using custom scripts written in Python 3.9.13 and R 4.2.2. For linear mixed model, lme4 and lmerTest package was used [104] [105]. For all linear mixed models used in this chapter, fixed effects, random effects for intercepts and independent variables (IV) are estimated. The model also estimates the correlation

between the intercept deviations and IV effect deviations across subjects.

6.2.4 Computational model

Niv et al. (2007) proposed that the delay between consecutive actions performed by animals can be formulated as a trade-off between opportunity costs and vigour costs [54]. They developed this framework based on earlier works in average reinforcement learning, showing that this trade-off approach maximises average reward [53]. In their formulation, animals choose an action pair (a, τ). The first, a, is the common action in discrete decision-making models, and the separate τ represents the time delay between two actions. We built upon their proposed model to suit the requirements of our study. Similar to the model in Chapter 4, our action a represents the choice of which fruit to pick up. To account for the different effort costs incurred for each fruit, we again used a distance-based effort cost term $C_m \cdot d(a)$, as in Chapter 4. Additionally, a time delay between actions is required to model vigour. We assumed that subjects estimate the time delay by estimating their speed V_{speed} and calculate the delay based on the distance to the fruit and the estimated speed. Hence, for a decision time point state $s \in \mathcal{S}$, if the effort cost is inverse proportional to the time delay, the optimal differential value of an (a, τ) pair is

$$Q^*(s,a,\tau) = r + C_p u(a) - C_v C_m \cdot d(a) \left(\tau + \frac{d(a)}{V_{speed}(s)}\right)^{-1} - \overline{R}\left(\tau + \frac{d(a)}{V_{speed}(s)}\right) + \gamma V^*(s')$$

where τ is the additional delay accounting for the time waited before committing to execute the action. $s' \in \mathcal{S}$ is the deterministic subsequent state after choosing (a, τ). V^* is the optimal differential value function.

We fitted vigour constants for no tonic pain condition and with tonic pain condition separately. To fit the model, a key assumption in [54] is animals choose an action pair of an action and a delay. The delay includes both the time waiting and the time executing the action. It is an approximation of real animal behaviour, and in fitting to real human subjects data in this task, we do not have direct access to subjects' delay decision. Therefore, we keep track of the delay passed and compute the differential value at each time point. The central assumption in this model fitting solution is all differential values for the choices

that are not chosen should be less than the value for the final chosen choice. Algorithm 6 describes the model fitting algorithm.

6.2.5 EEG analysis

6.2.5.1 Data collection and preprocessing

Whole-scalp EEG data was continuously recorded with 32-channel LiveAmp system (Brain-Products GmbH, Munich, Germany) at 500Hz. We choose FCz as the reference. Recordings were first cleaned with EEGLAB (v2023.1) by removing channel drift (linear filter (FIR) transition band from 0.25 to 0.75 Hz). It also automatically removes bad data periods by thresholding max acceptable 0.5 second window standard deviation of 20, max acceptable channel RMS range of 7, and maximum out-of-bound channels of 25%. We then cleaned ocular artifacts with ICA in EEGLAB.

6.2.5.2 ERP analysis

ERP analysis was conducted in Python with MNE-Python 1.7.0 package. A bandpass filter from 1 to 30 Hz was applied, and the data was re-referenced to the average. Recordings were processed into 1s epoch with 0.3s baseline before the event trigger. Epochs with peak-to-peak amplitude exceeding 200 μV in channels surrounding the electrode of interest (Cz; F3, F4, Fz, Cz, P3, P4) were excluded. We search N1 amplitude peak from 100-170ms and P2 amplitude peak from 140-300ms at channel Cz [132, 133].

6.2.5.3 Time frequency analysis

Time frequency analysis was performed in Python with custom scripts relying on MNE-Python 1.7.0 package. A notch filter with 4Hz width centered at 50Hz was applied. In tonic pain activation analysis, we estimate the spectral power for each block of 60s length. Welch method was used to estimate the spectral power. A Hamming-tapered sliding window was used. The window contains 1000 samples (2 s) and was shifted with 500 samples overlap (50% overlap). The data for each subject was then divided into tonic pain and no tonic pain conditions. Average power over all blocks for each pain condition was computed for the following frequency bands: (4–8 Hz), alpha (8–12 Hz), beta (12–30 Hz),

Algorithm 6 Chapter 6 model fitting

```
function CombinedModel(memory, hyperParameters, estimatedSpeed, currentDelay)
    return optimal choice, optimal value
end function
function BlockFitting(hyperParameters, blockFrames)
    startTimestamp ← frames[0]['timestamp']
    needReevaluation ← True
    predictionIsCorrectForNow ← False
    lastIncorrectChoiceOptimalValue ← -INF
    correctCount ← 0
    totalCount ← 0
    for frame in blockFrames do
        memory ← EyetrackingUpdate
        estimatedSpeed ← distanceTravelled / (frame['timestamp'] - startTimestamp)
        currentDelay ← frame['timestamp'] - lastActionTimestamp
        choice, value ← CombinedModel(memory, hyperParameters, estimatedSpeed,
currentDelay)
        if choice = realChoice then
            if needReevaluation then
                if lastIncorrectChoiceOptimalValue > value then
                    predictionIsCorrectForNow ← False
                else
                    predictionIsCorrectForNow ← True
                end if
                needReevaluation ← False
            end if
        else
            if lastIncorrectChoiceOptimalValue < value then
                lastIncorrectChoiceOptimalValue ← value
            end if
            needReevaluation ← True
        end if
        if frame['picked_object'] != '' then
            if predictionIsCorrectForNow then
                correctCount ← correctCount + 1
            end if
            totalCount ← totalCount + 1
            needReevaluation ← True
            predictionIsCorrectForNow ← False
            lastIncorrectChoiceOptimalValue ← -INF
        end if
    end for
    return (1 - correctCount) / totalCount
end function
```

and gamma (30–100 Hz) [134, 135, 136]. Paired t-tests at electrode level were conducted to evaluate the significance of tonic pain effect. For the test results, false discovery rate (FDR) correction was applied across electrodes.

In model-based analysis of model parameter changes, 0.5 second windows (-0 s to 0.5s) of the decision points in the model-fitting (when subjects looked at new fruits) were taken and their spectral power were computed with Hamming-tapered Welch method. The spectral power was fit to a linear mixed model as the dependent variable and the independent variable was the model parameters or experiment conditions. To correct movement artifacts, head movement speed was calculated and used as an extra independent variable, assuming zero correlation with the other interested independent variable. For each frequency band, we further exclude epochs with power's z-value exceeding 3 in any channel to remove contaminated data.

6.2.5.4 Source analysis

Following the preprocessing, band-pass filters were applied to the preprocessed signals and epoched for the same 0–0.5 s after decision points as in the time-frequency analysis. Epochs rejected in time-frequency analysis were rejected for source analysis. The lead-field matrix was computed for a recursively subdivided icosahedron brain surface, consisting of a total of 1,284 vertices derived from MNE-provided templates. The sLORETA method was then applied to each band-passed signal to obtain a time-domain signal for each source location [137]. For each source location, the Welch method was used to estimate the total spectral power of each epoch. The same LMM employed in the time-frequency analysis was applied, except that the band power in the time-frequency analysis was replaced by the total spectral power estimated from inverse solutions of band-passed signals at source locations [138, 139, 140].

6.3 Results

Our first finding is that subjects showed a reduction in vigour: their moving speed slowed down (Figure 37a, b) and their fruit collection rates dropped (Figure 37c).

Next, we tested whether tonic pain affects subjects' perceptions and decisions regarding

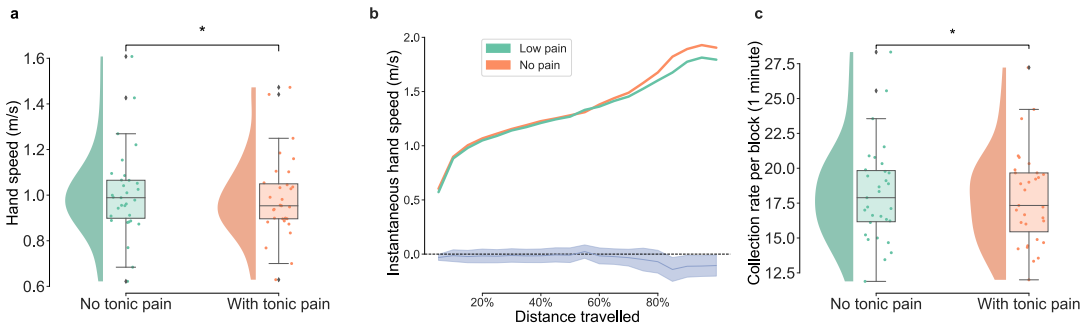


Figure 37: **a**, Average hand speed. A one-tailed paired t-test was conducted to evaluate whether the hand speed was faster in no tonic pain condition ($M = 1.006, SD = 0.190$) than in with tonic pain condition ($M = 0.990, SD = 0.178$), $t(30) = 1.972$, $p = .0290$. **b**, A visualisation demonstrating the instantaneous hand speed at different percentage of total distance travelled in reaching to the fruit. Shaded area is the 95% confidence interval of the pairwise difference mean. **c**, The average fruit collection rate over the one-minute block. A one-tailed paired t-test was conducted to evaluate whether the collection rate was higher in no tonic pain condition ($M = 18.222, SD = 3.452$) than in with tonic pain condition ($M = 17.907, SD = 3.352$): $t(30) = 1.933$, $p = .0314$.

phasic pain. Figure 38 compares the phasic pain ratings under different conditions. This experimental design also shares similarities with the conditioned pain modulation (CPM) paradigm [129], if the electric phasic pain is considered the test stimulus and the pressure tonic pain the conditioning stimulus. Contrary to the CPM effect, we found that the mean pain ratings were higher with tonic pain. However, this should not be viewed as evidence against other effective CPM results, as most CPM research has been conducted in a stationary setup, while our study featured an active task design. Furthermore, the results were not significant enough to reject the null hypothesis.

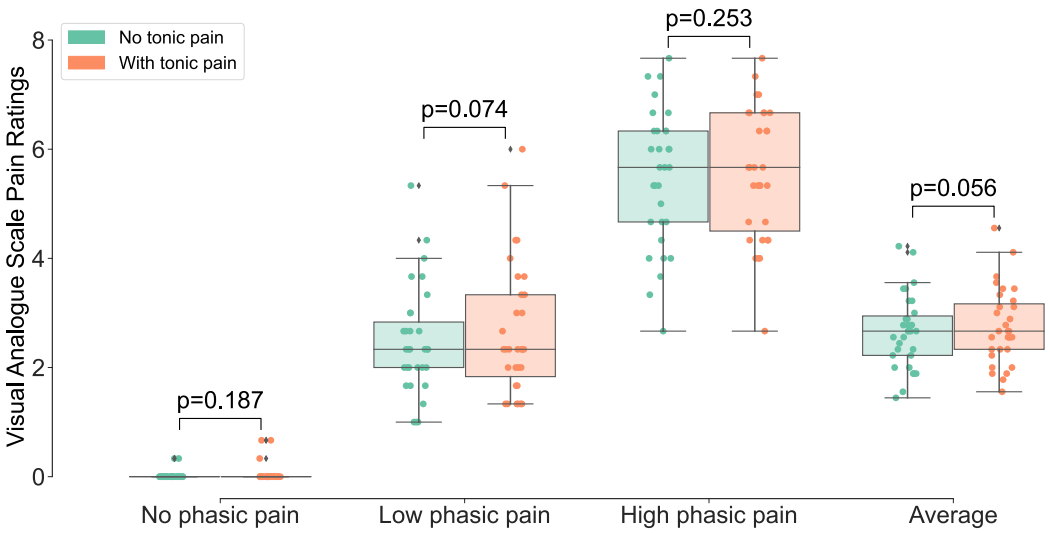


Figure 38: Phasic pain ratings in difference conditions. p-values are one-tailed paired t-test with the alternative hypothesis being higher ratings in with tonic pain condition. Two-way repeated measures ANOVA does not show a significant effect for tonic pain, $F(1,30)=2.678$, $p=0.122$. The interaction effect between tonic pain and phasic pain is also insignificant, $F(2, 60)=0.863$, $p=0.427$

If there is a trend of increasing phasic pain ratings with tonic pain, a natural question to ask is whether this is reflected in the motivational values. The reduction in motivational values of phasic pain—a decrease in aversive choice probability in our study—can be interpreted as an increase in punishment sensitivity. Research has shown that increased punishment sensitivity was observed in chronic pain patients [128]. Nevertheless, we did not observe a change in punishment sensitivity with or without tonic pain. The decrease in aversive choice probability was reproduced as in Chapter 4 (Figure 39 a), but the average aversive choice probabilities were similar in conditions with and without tonic pain (Figure 39b). This result should not be surprising when compared with significant findings in chronic pain patients, as temporary tonic pain stimulation in healthy subjects differs from chronic pain in patients. The differences in the effects of tonic pain between patients and healthy subjects support the theory of long-term neural adaptation in chronic pain patients, distinct from that in healthy subjects [141].

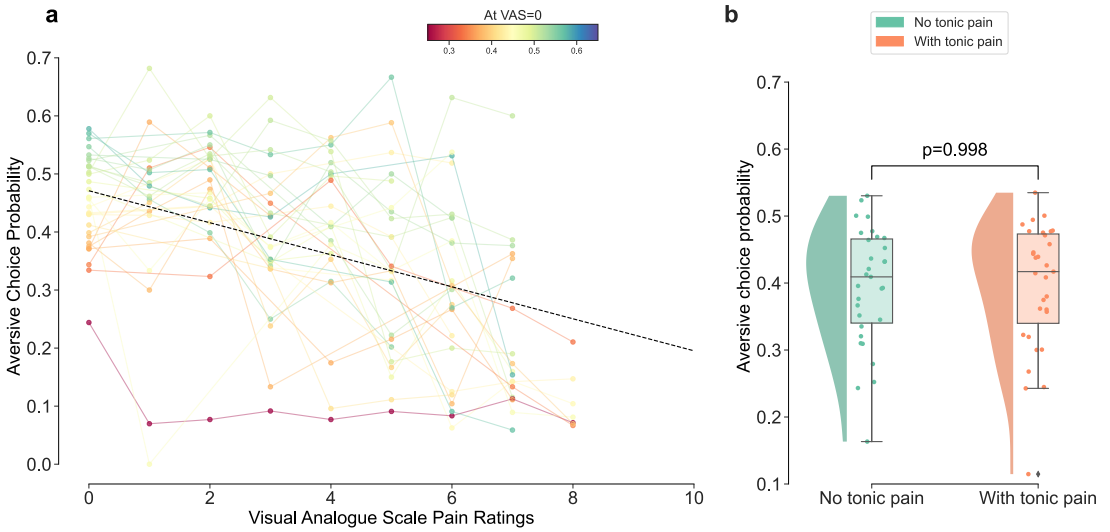


Figure 39: **a**, Chapter 6's version of Figure 21: Decreased aversive choice probability as subjective ratings increases. Estimated fixed effect for rating: $\beta = -0.0276, 95\% CI[-0.0349, -0.0201], t = -7.432, p < .001$. **b**, Similar average averisve choice probability with tonic pain ($M = 0.395, SD = 0.0934$) or without tonic pain ($M = 0.395, SD = 0.0869$). A two-sided paired t-test was conducted and found no significant differences between two groups, t(30)=0.00269, p=.998

EEG recordings show a strong effect for the phasic pain intensity on the amplitude of pain evoked response potentials (ERP) but not for tonic pain (Figure 40).

For tonic pain's neural signature, we performed time-frequency analysis and found significant reductions of band power were identified over sensorimotor, posterior parietal and midline electrodes were observed in the alpha band, and over widespread scalp regions in beta and gamma bands (Figure 41).

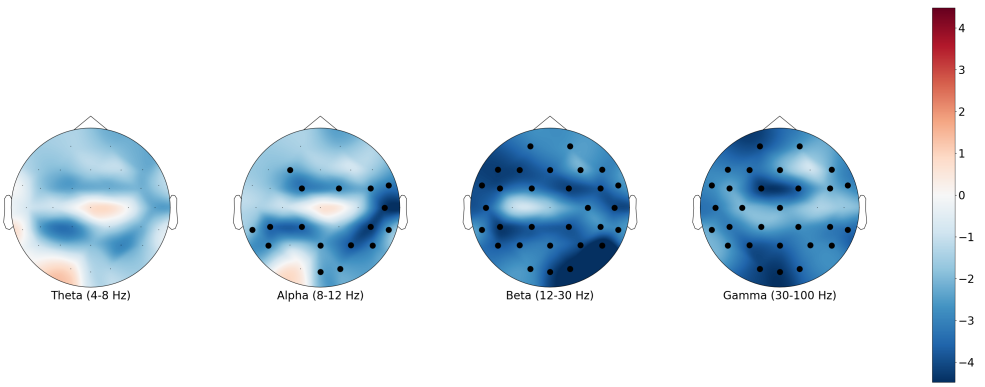


Figure 41: Frequency band power during tonic pain block contrasted with no tonic pain condition. t values are plotted to show the band power differences. Bold dots represent electrodes with significant effect after FDR correction.

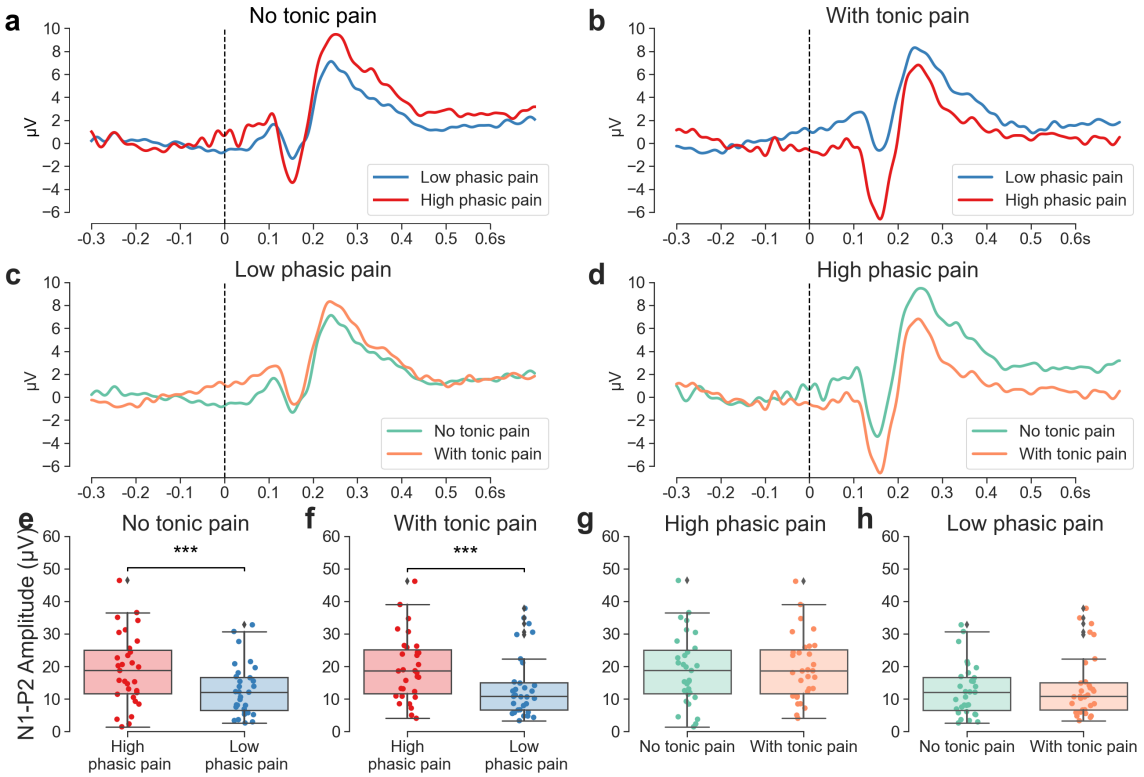


Figure 40: **a**, **b**, Phasic pain ERP comparison in each tonic pain condition. **c**, **d**, Phasic pain ERP comparison in each phasic pain condition. **e**, **f**, High phasic pain induced a significantly higher N1-P2 amplitude with or without tonic pain. **g**, **h**, Tonic pain stimulation does not show a significant effect on phasic pain N1-P2 amplitude. For high phasic pain, with tonic pain condition (M = 19.499, SD = 9.902) had a slightly higher mean amplitude, but this difference was not statistically significant (M = 18.978, SD = 10.720), one-tailed paired $t(30) = 0.494$, $p = .313$. For low phasic pain, with tonic pain condition (M = 13.918, SD = 9.609) also shows a slightly higher mean amplitude but not significant than in no tonic pain condition (M = 12.701, SD = 7.880), one-tailed paired $t(30) = 1.360$, $p = .101$. Two-way repeated measures ANOVA shows the interaction effect is not significant, $F(1, 30) = 0.212$, $p = 0.649$.

We further performed source analysis using LORETA method to identify the relevant brain regions for tonic pain. As we have limited number of EEG channels and the sensor environment is noisy, the source analysis results are considered as exploratory and are shown in Appendix Figure 53.

The widespread significant results shown in Figure 41, which were calculated from the entire block's absolute power, likely contained contaminated segments and included different events. To clean the data, we adopted a model-based approach and fitted a linear mixed model to each channel's data with head movement speed as a predictor. No significant correlation for the tonic pain condition was found before the key decision point (Figure 42), but significant negative correlations between Alpha and Beta activities around CP6, the right parietal region, were observed (Figure 43) after the key decision point. As the pressure cuff was worn on the non-dominant arm, 29 subjects had the pressure cuff on the left arm, and only 2 subjects had it on the right arm. The right parietal activation corresponds to the application of tonic pain on the left arm.

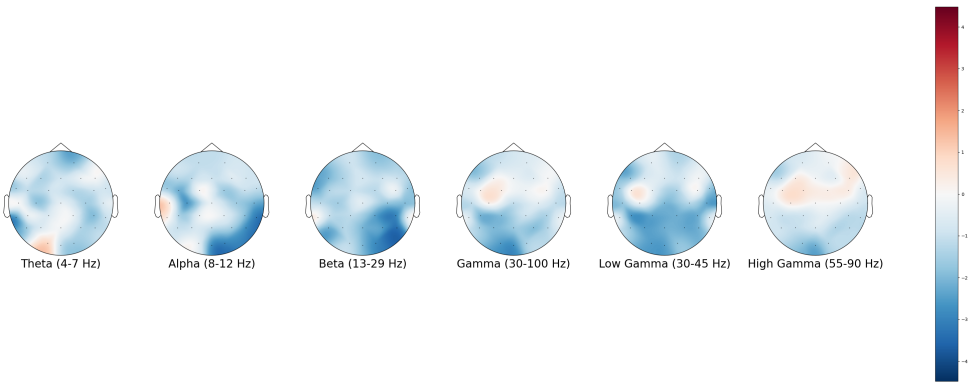


Figure 42: Model-based tonic pain frequency band power correlation (0.5 second before decision point)

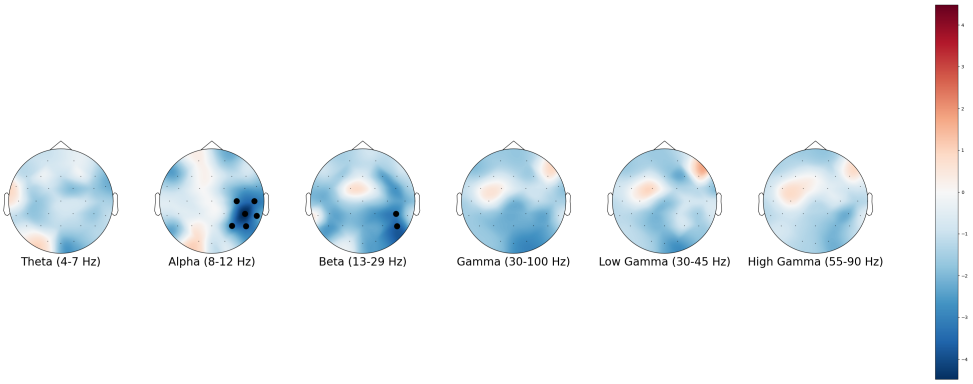


Figure 43: Model-based tonic pain frequency band power correlation (0.5 second after decision point)

As we have demonstrated the distinct phasic and tonic pain effects, we then integrate both phasic pain and tonic pain effects into a single combined model. We found the fitted vigour constants were significantly higher in with tonic pain condition (Figure 44a). By comparing the pairwise difference between two conditions of fitted vigour constants and hand speed, these two quantities are inversely correlated but not significantly enough to reject the null hypothesis (Figure 44b). This suggest there could be differences in estimated vigour for decision-making and the vigour shown in motor execution calculated from moving speed.

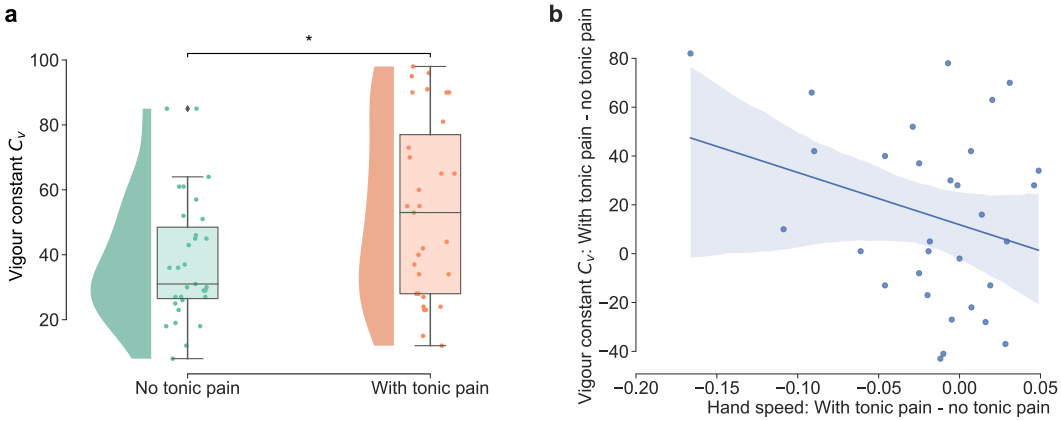


Figure 44: **a**, A one-tailed paired t-test was conducted to evaluate if fitted vigour constants were greater in with tonic pain condition ($M = 53.613, SD = 27.166$) than no tonic pain condition ($M = 38.161, SD = 18.880$), $t(30) = -2.372$, $p = .0121$. **b**, The pairwise difference of hand speed shows a negative correlated trend but not significant enough to reject the null hypothesis, $r(29) = -.280$, $R^2 = 0.0779$, $F(1, 29) = 2.448$, $p = .129$

The fitted phasic pain parameters also show no significant differences in punishment sen-

sitivity, in line with the behavioral results (Figure 45). The vigour constants and phasic pain function fitting results show we can combine two dissociable effects in a single model, and the effects are validated by the behavioural data.

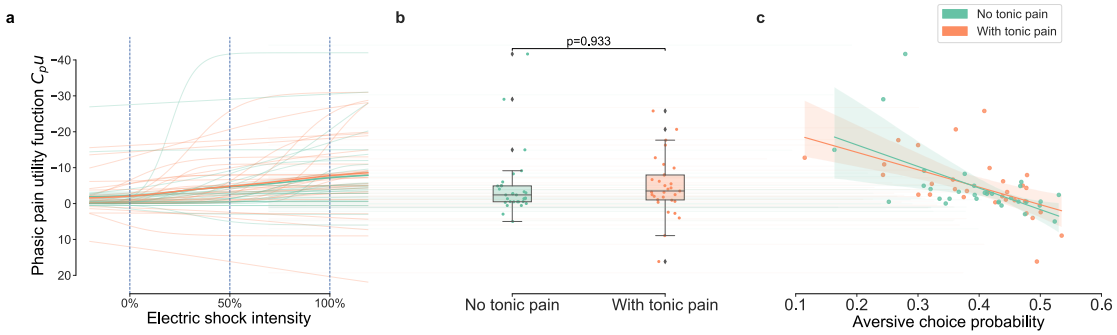


Figure 45: **a**, Phasic pain utility function curves. Solid lines are the average values in each condition. **b**, Phasic pain utility function values at 50% electric shock intensity. A two-tailed paired t-test was conducted and found no significant differences between no tonic condition ($M = -4.645, SD = 8.982$) and with tonic pain condition ($M = -4.807, SD = 8.146$), $t(30) = -0.0849$, $p = 0.933$. **c**, Phasic pain utility function values at 50% electric shock intensity plotted against aversive choice probability. Both conditions show a significant correlation between the fitted model values and behavioural choice probability. In no tonic pain condition, $R^2 = 0.337$, $F(1, 29) = 14.75$, $p < .001$. In with tonic condition, $R^2 = 0.311$, $F(1, 29) = 13.07$, $p = 0.00112$.

6.4 Discussion

6.4.1 Distinct functions of tonic and phasic pain

In Chapter 4, we showed phasic pain selectively modulates motivational values and vigour. As there is no change in the punishment sensitivity (measured by phasic pain's effect on choice probability) by tonic pain, we can conclude there is a distinct role of tonic and phasic pain in motivational value modulation, at least with healthy subjects.

Researchers have proposed that the fear of pain, derived from the fear-avoidance model, could contribute to the formation of chronic pain conditions [142]. A reduction in vigour is commonly observed in chronic pain patients [143]. In this chapter, we showed the tonic pain reduced moving speed, one of the behavioural metric for vigour. It is well known that exercise promotes recovery from chronic pain and helps break the vicious cycle [144]. Therefore, based on our results showing that tonic pain has an immediate effect on vigour, we recommend interventions aimed at elevating patients' motivational

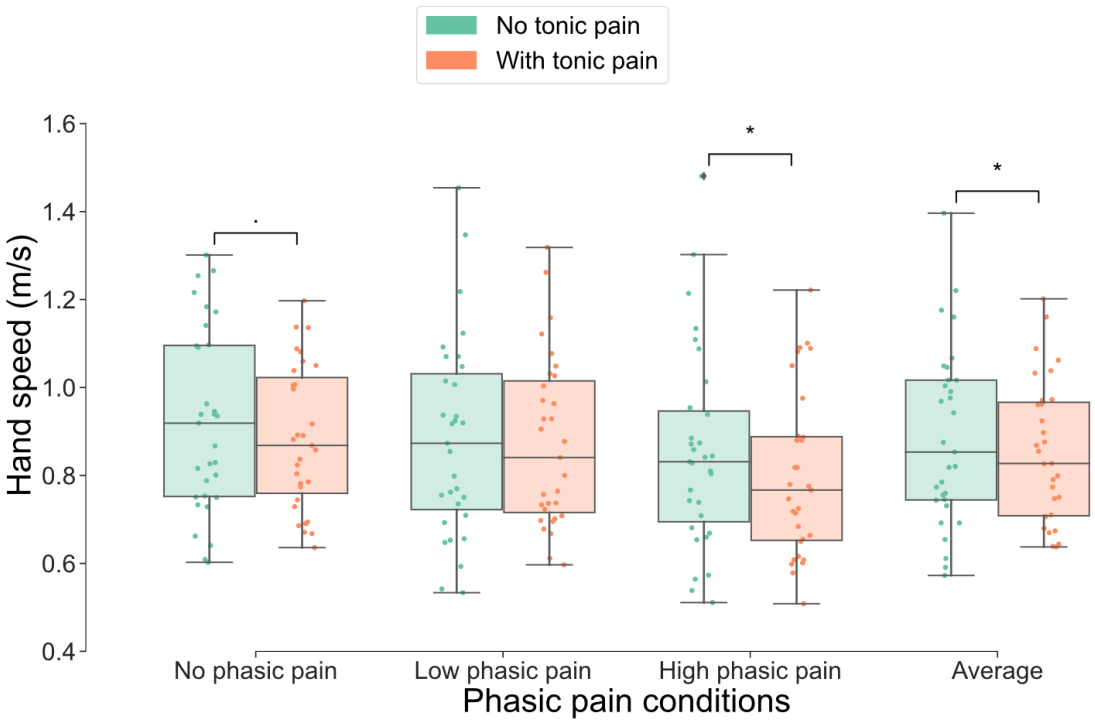


Figure 46: Hand moving speed for reaching non-painful choices. One-tailed paired t-test showed the average moving speed for tonic pain condition was significantly less than the moving speed for non-tonic pain condition, $t(30) = 1.975, p = .0287$. Repeated measures ANOVA shows phasic pain also shows a significant effect on moving speed, $F(2, 60) = 7.824, p < .001$.

vigour to aid chronic pain recovery.

Nevertheless, in Chapter 4, we also demonstrated a reduction in moving speed for increased phasic pain intensity. We further shown that the reduction of speed is selective: the reduction on reaching a non-painful choice was not significant while the moving speed for reaching a painful choice was significant. In this chapter, we found there is also a significant effect for tonic pain on reaching non-painful choices (Figure 46). Interestingly, the effect for phasic pain was also significant. Although in this chapter, phasic pain levels are not numeric, making it not directly comparable with Chapter 4's results, it suggests there could be an effect of tonic pain on phasic pain's vigour effect.

On the other hand, calculating the overall moving speed from empirical behaviour data can have additional noises apart from the true motivational vigour. We additionally fit separate vigour constants for each condition including both tonic and phasic pain conditions, and we found tonic pain conditions have significantly higher vigour constants than no tonic pain conditions. We did not find significant changes for vigour constants across phasic

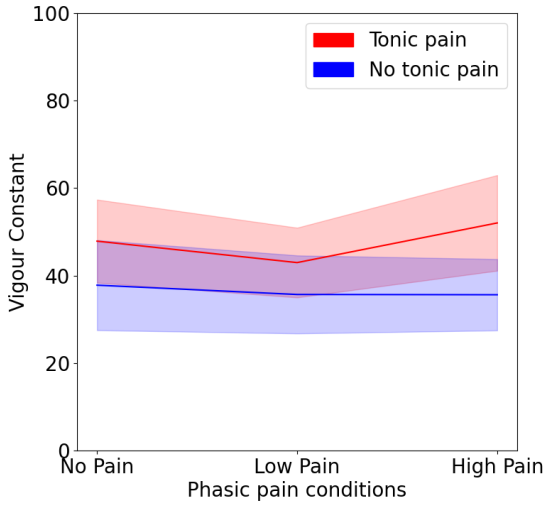


Figure 47: Separate vigour constants fitting. Average vigour constants for tonic pain condition was higher than no tonic pain condition, $t(30) = 2.516, p = .00872$. Repeated measures ANOVA showed the effect for tonic pain is significant, $F(1, 30) = 6.330, p = .017$, but the effect for phasic pain was not significant, $F(2, 60) = 0.398, p = 0.673$. The interaction effect was not significant $F(2, 60) = 0.670, p = .515$.

pain conditions (Figure 47).

We can apply the same model to Chapter 4's data. The fitted vigour constants also showed there was no significant effect for phasic pain (Figure 48). If the modelled vigour constants better represent motivational vigour than empirical moving speed, the results revealed an important correspondence between pain types and behavioural functions. It has been argued that tonic dopamine encodes long-term average reward, while phasic dopamine changes encode short-term reward prediction errors [145, 54]. We have shown that tonic pain causes an overall reduction in motivational vigour, whereas phasic pain leads to a selective reduction in motivational value. This is analogous to the overall changes in average reward putatively encoded by tonic dopamine and the selective changes in associative reward encoded by phasic dopamine.

6.4.2 Punishment sensitivity changes

Another key finding is that punishment sensitivity, measured by the aversive choice probability, does not show significant changes in the tonic pain condition. While there is evidence of increased punishment sensitivity in chronic pain patients [128], and given that the punishment sensitivity in their task is comparable to the one in our task, the likely

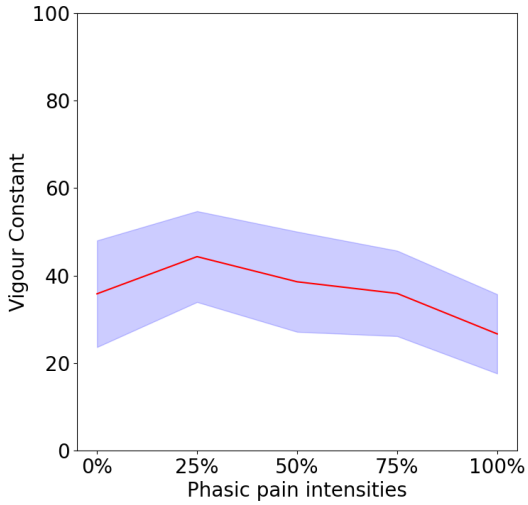


Figure 48: Linear mixed model does not showed a significant effect for phasic pain intensities, $\beta = -10.7, t = -1.363, p = .186$.

contributing factor is the duration of tonic pain. Chronic pain patients may experience continuous pain for months, whereas in our study, tonic pain stimulation lasted for merely one minute. Given that we observe little change in punishment sensitivity at the starting point and a significant increase in chronic pain patients, a corollary of this hypothesis, along with the mean value theorem from mathematics, suggests that we should observe an increase in punishment sensitivity if we monitor the progression of chronic pain over time.

Punishment sensitivity is a behavioural metric with underlying neural substrates, as evidenced by the extensive decision-making neuroscience literature [127]. If we can establish a progressive behavioural metric for chronic pain, it would serve as a critical neuromodulation target for new chronic pain treatments. The change in punishment sensitivity need not be solely the consequence of the continuous presence of pain; rather, it is possible that sensory sensitivity is also affected, contributing to a vicious cycle. Thus, by utilising this novel platform, it may be possible to employ adaptive control based on real-time estimation of critical cognitive markers, potentially halting the progression of these cognitive markers as observed in chronic pain patients. On the other hand, it is necessary to recognise that tonic pain stimulation on healthy participants does not replicate the exact same experience that chronic pain patients experience. It is important to conduct further research with chronic pain patients with this paradigm.

6.4.3 Conditioned pain modulation and interaction between pain modalities

Conditioned pain modulation has been controversial, yet it has received much attention in pain research and clinical practice. Strictly speaking, our study is not a conditioned pain modulation experiment, as we included an additional instrumental task requiring physical movement for subjects to perform while receiving pain stimuli. However, the core component in both our experiment and the conditioned pain modulation paradigm—the interaction between two types of pain, one serving as the conditioning stimulus and the other as the test stimulus—is comparable. Although the data does not show significant effects, a trend of increased ratings for phasic pain under tonic pain was observed. This could serve as preliminary evidence for future research on conditioned pain modulation within an active task.

Although our data, across a series of experiments, shows that tonic pain and phasic pain do not exhibit a strong interaction effect, this does not rule out the possibility of interactions between different pain modalities, especially in the long term. As discussed in relation to punishment sensitivity, the long-term effect of a specific type of pain is a fascinating research question that warrants further investigation.

6.4.4 Conclusion

In this chapter, we first integrated tonic pain into a free-operant task and observed a change in motivational vigour. The correspondence between pain modalities, in terms of their tonic-phasic aspects, and their relative short-term and long-term behavioural consequences, may resemble the correspondence of other critical neural encoding, such as the putative functions of tonic and phasic dopamine. This insight, supported by experimental data on the temporal properties of pain, deepens our understanding of pain's recuperative function in injury and its transient danger-avoidance function.

Another important finding is the absence of changes in punishment sensitivity in the tonic pain condition. Based on positive results from chronic pain patients, we hypothesise that punishment sensitivity could serve as a cognitive behavioural marker for the progression of chronic pain conditions. This makes it a natural target for neuromodulation and cognitive

behavioural therapy.

7 Applications and future directions

We conclude this thesis by presenting a series of applications including brief overviews of collaborated projects that utilised the platform described in this thesis. We further discuss the future directions of research that can be developed upon the results presented in this thesis.

7.1 Applications

7.1.1 Pain visualisation and neurofeedback therapy

This project was initially conceived as a public engagement initiative, but its design fully utilises the platform's capabilities and has the potential to be developed into a neurofeedback therapy system. A neurofeedback paradigm typically measures neural activities and presents this activity to the subject using visual, auditory stimuli, or other representations [146]. By measuring the neural activity and feeding back the representation of the activity to the subject, it aims to change their neural activity, forming a closed-loop control. Such paradigms have been shown to be effective in treating various neurophysiology-related diseases, including chronic pain [147, 148, 149]. In this project, we present physiological responses as abstract shape changes in VR. As VR is generally more immersive than screen-based experiences, we expect that VR-based neurofeedback could be more effective than conventional neurofeedback methods.

The system consists of the following components:

- Neurophysiological sensors
- Virtual reality presentation
- Control system

Neurophsyiological sensors

The neurophysiological data were collected using a wireless 32-channel EEG LiveAmp system (BrainProducts GmbH, Munich, Germany). In addition to the EEG data, the device also collects photoplethysmography (PPG) data, breathing rate (via an inflated chest belt

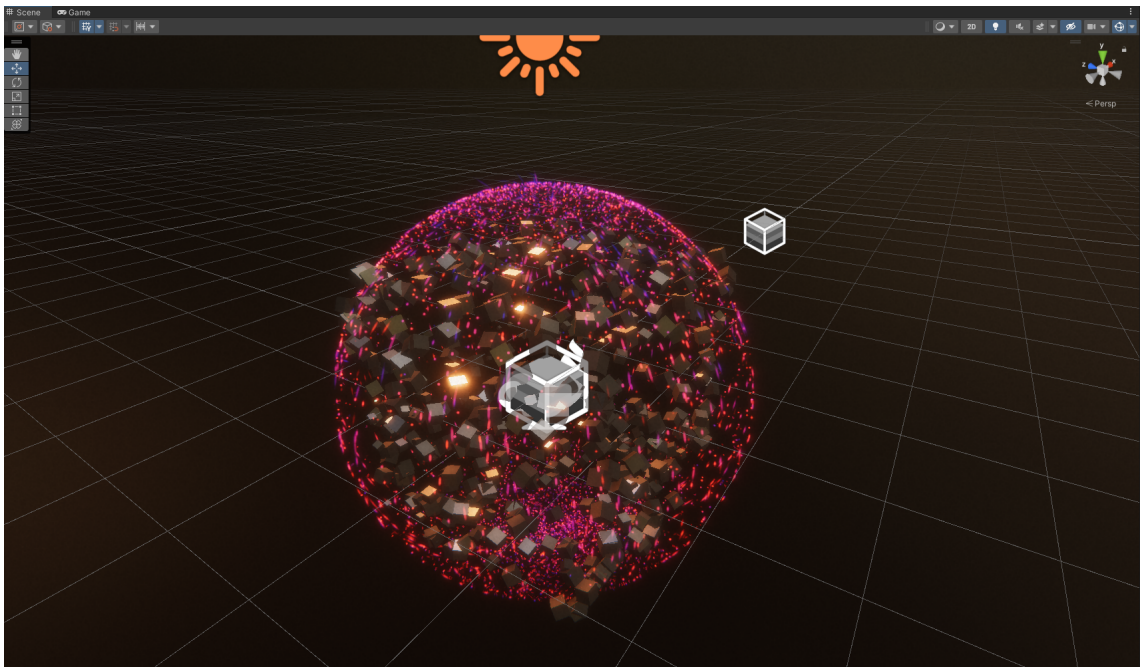


Figure 49: The visual representation of body state

connected to a pneumatic transducer), and skin conductance data.

Virtual reality presentation

A virtual reality experience was developed to visualise the processed sensory data. We designed an abstract, immersive three-dimensional emblem, featuring a cube comprised of small, shiny metal cubes inscribed within a bright particle sphere. The participant's camera is positioned inside the sphere. The shape of the objects can be changed in real-time to provide a visual representation of the participant's physiological state. Figure 49 shows the implemented visual presentation with three adjustable parameters: the outer sphere size, the colour of the spherical particles, and the sharpness of the metallic cube.

Control system

The control system processes the raw sensory data and adaptively changes the parameters of the visual presentation.

The core idea is that participants can learn to control their physiological state through feedback from the immersive visualisation. As part of the public engagement project, participants reported feeling that they could influence the visual representation by actively regulating their physiological responses, most notably their breathing rate. This experience paved the way for future development as a neurofeedback therapy.

7.1.2 Collaborated projects based on this platform

The platform has been successfully used as a general platform for multiple pain research projects in addition to my main research project described in Chapter 3, 4, 5, 6. We will briefly go through these collaborated projects.

Balancing safety and efficiency in human decision making

This project was a collaboration with Pranav Mahajan [2]. The paper proposed an optimal strategy to balance safety and efficiency and examined the proposed mechanism in human subjects. The enhanced ecological validity, brought about by the platform, improves the reliability of the results in reflecting real-life conditions [150, 151]. Figure 50 shows the design of the experiment and the devices used.

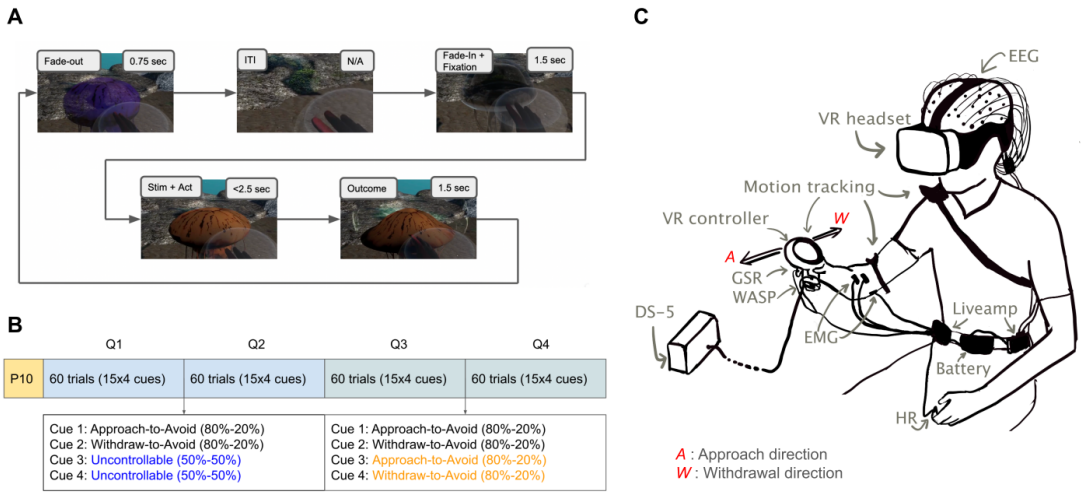


Figure 50: Reproduced from [2, Figure 4] (licensed under CC BY)

Neural mechanisms of multisensory cued pain anticipation during lateralised tonic pain

This project was a collaboration with Danielle Hewitt [3]. One of the critical elements of this experiment was establishing an embodiment and estimating the object's distance relative to the subject's body position. The immersive virtual reality setup was a perfect fit for the experimental design. Figure 51 shows the schematic diagram of the experiment design.

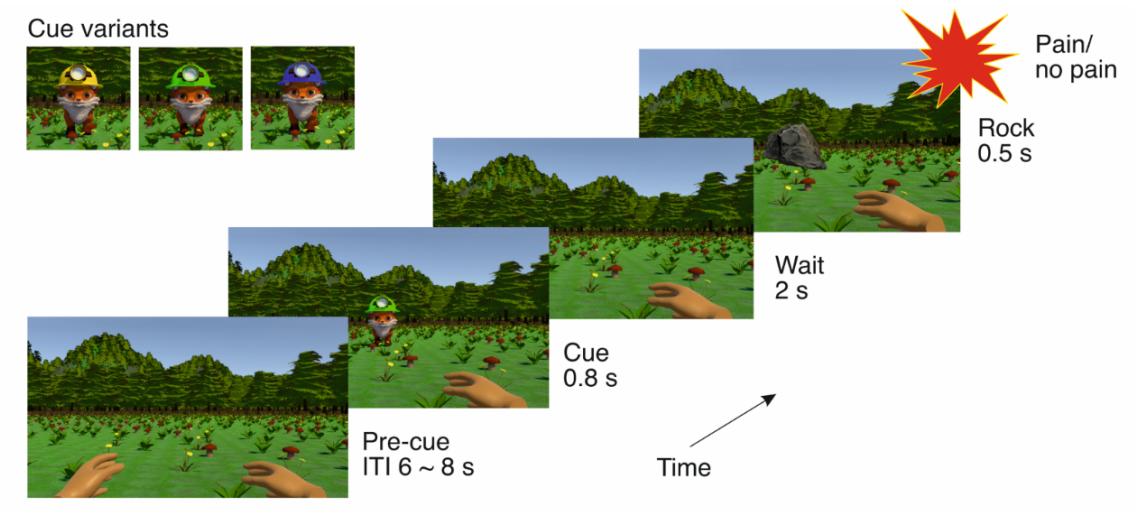


Figure 51: Reproduced from [3, Figure 1](licensed under CC-BY 4.0)

7.2 Future directions

Before we present the future directions based on this platform, we first discuss the limitations and caveats.

7.2.1 Caveats and Limitations

7.2.1.1 Worse physiological signal qualities

While a significant portion of this thesis describes the technical implementation of real-time wireless physiological data collection, signal quality has always been an issue when compared with stationary tasks without all the virtual reality equipment. For example, EEG signals are heavily influenced by motion artifacts. There are a few solutions to this, each with its own limitations.

Using wet electrodes

We have previously tested EEGs using dry electrodes. However, the motion artifacts were much greater than those with wet electrodes, which use conductive gel applied directly to the scalp. Therefore, we always recommend using wet electrodes instead of dry electrodes for virtual reality studies involving EEG. The drawback of using wet electrodes is that they take much longer to set up than dry electrodes and may cause more discomfort.

Using a tight EEG cap

We have tried applying a tight silicone cap to reduce the movement of electrodes during the pilot experiment. It might help in reducing motion artifacts; however, the subjects reported mild headaches following the experiment sessions. Therefore, we did not proceed with this method.

Using model-based method accounting for head movement

As we have presented in Chapter 6, it is effective to use a linear mixed model with head movement speed as a predictor to isolate signals caused by motion artifacts. The drawback of using a model-based approach is that we were then unable to directly test the raw data but could only interpret the statistics generated from the model. While it remains a useful technique, it is important to exercise extra care when interpreting the data.

7.2.1.2 Complex environment

Complex three-dimensional environment

The virtual reality system we discuss here is headset-based, where each eye receives a separately rendered image to construct stereoscopic vision. Therefore, the content is naturally three-dimensional. As subjects are usually allowed to freely move their heads, it is important to analyse their current fixation – in many conventional psychology experiments, this corresponds to the fixation cross prior to the trial. Eye tracking technology has worked well in our experiments and plays a key role in identifying the decision points in our free-operant task. Although the eye tracking method has been effective for our study purposes, the virtual reality environment is generally more complex, and simple eye tracking may not work for all experimental setups.

Complexity of analysis

The complexity of the three-dimensional virtual environment not only adds to the difficulty of task design but also significantly increases the complexity of analysis. In this thesis, we presented numerous behavioural metrics and real-time computational modelling to analyse a relatively simple foraging task, yet we are not confident that we have captured the full complexity of this task. The individual variability is large, which demands us to use mixed models accounting for random effects. We would like to identify other factors

that contribute to the great individual variability; however, it is difficult and challenging. While we can always fit a complex model to account for the complexity in data, such as fitting a neural network to the subject data, it is likely that we can quickly lose interpretability of the model and the model overfits. Developing new data-driven methods to overcome these challenges as we migrate from stationary fully-controlled experiments to more realistic experiments is imminent.

7.2.1.3 Ergonomics and accessibility

Perhaps one of the major reasons that hinder the popularity of virtual reality equipment among the general public is ergonomic design and accessibility. Although technology has advanced, current virtual reality headsets are still heavy and not suitable for extended periods of wear. They also require a greater open space than a normal computer with a monitor. When it comes to study design, we normally design experiments with less in-task time than conventional experiments. Based on our experience, subjects start to request breaks when the time wearing the headset exceeds half an hour if the task requires physical movement. The difficulty of staying in virtual reality for long periods restricts us from adding more repetitive trials to the experiment.

7.2.2 Future projects

There are a variety of possible pain research projects following the results presented in this thesis and based on the platform proposed in this thesis. We discuss three major possible projects in the following sections.

7.2.2.1 Phasic pain assessment

In Chapter 4, we observed a difference between subjective pain ratings and modelled pain aversiveness. While subjective pain ratings have been used as a major outcome for pain studies and in clinical practice, their subjective nature can make it difficult to compare across subjects. However, with our new virtual reality-based behavioural task, we can provide an objective measurement through the trade-off between the aversiveness of pain and vigour cost, where the vigour cost is measured from action speed and effort to move

across unit distance.

Although we have used experimental pain to conduct the study with healthy subjects, the experiment should be easily applicable to certain groups of pain patients. For example, for back pain patients with difficulty bending over, we remove the experimental phasic pain and use the variation of fruit height as the source inducing varying degrees of phasic pain, assuming it is more painful to bend further for this patient group. Then, the identical analysis procedure used for healthy subjects in our previous study can be applied.

Therefore, a potential project is to run the same study with pain patients without experimental pain. This method can be applied with additional treatment conditions, for example, the efficacy of a drug. These types of studies were usually evaluated based on subjective ratings, and our work can provide another additional objective measurement dimension to the results. It could even be developed to evaluate pediatric pain, as it does not require verbal reporting but an interesting task suffices.

7.2.2.2 Longitudinal chronic pain progression monitoring

In Chapter 6, we found that tonic pain does not change punishment sensitivity. Research has shown that punishment sensitivity is changed in the chronic pain patient group. Therefore, we can hypothesize a slow neural adaptation process exists in the progression of chronic pain. To our best knowledge, a quantitative longitudinal modelling of this neural adaptation process, while being very challenging, has not been done. A potential project aims to run the experiment in multiple sessions over a long time period, evaluating key cognitive metrics (e.g., punishment sensitivity) in a model-based way. Neural signatures correlated with modelled values can be analysed and compared longitudinally across sessions to establish a long-term neural adaptation model that deepens our understanding of the neural changes caused by chronic pain.

7.2.2.3 Therapeutic application

As the platform features closed-loop control, it is promising to implement adaptive control delivering customized therapy for patients. The neurofeedback project presented at the beginning of the chapter is one of the simplest paradigms utilising closed-loop control.

More complex adaptive control can be introduced based on our developed model. By analysing subjects' behavioural data, we can provide a real-time quantitative account of subjects' pain and vigour without subjective reporting. These metrics can be used to adjust the task difficulty in a rehabilitation application to promote physical movement while not exceeding the individual's limit. Such model-based individualised methods should be the future direction of rehabilitation.

8 References

[1] G. W. Story, I. Vlaev, B. Seymour, J. S. Winston, A. Darzi, and R. J. Dolan, "Dread and the disvalue of future pain," *PLoS Computational Biology*, vol. 9, no. 11, 2013.

[2] P. Mahajan, S. Tong, S. W. Lee, and B. Seymour, "Balancing safety and efficiency in human decision making," *bioRxiv*, 2024.

[3] D. Hewitt, S. Tong, S. Schreiber, and B. Seymour, "Tonic pain revalues associative memories of phasic pain," *bioRxiv*, 2025.

[4] A. Fayaz, P. Croft, R. M. Langford, L. J. Donaldson, and G. T. Jones, "Prevalence of chronic pain in the uk: a systematic review and meta-analysis of population studies," *BMJ Open*, vol. 6, no. 6, 2016.

[5] J. C. Ballantyne, "Opioids for the treatment of chronic pain: Mistakes made, lessons learned, and future directions," *Anesthesia & Analgesia*, vol. 125, no. 5, 2017.

[6] B. Seymour, "Pain: A precision signal for reinforcement learning and control," *Neuron*, vol. 101, no. 6, pp. 1029–1041, 2019.

[7] K. Wiech, J. Vandekerckhove, J. Zaman, F. Tuerlinckx, J. W. Vlaeyen, and I. Tracey, "Influence of prior information on pain involves biased perceptual decision-making," *Current Biology*, vol. 24, pp. R679–R681, Aug 2014.

[8] F. Mancini, S. Zhang, and B. Seymour, "Computational and neural mechanisms of statistical pain learning," *Nature Communications*, vol. 13, p. 6613, Nov 2022.

[9] J. F. Kihlstrom, "Ecological validity and "ecological validity"," *Perspectives on Psychological Science*, vol. 16, no. 2, pp. 466–471, 2021. PMID: 33593121.

[10] S. Stanslaski, P. Afshar, P. Cong, J. Giftakis, P. Stypulkowski, D. Carlson, D. Linde, D. Ullestad, A.-T. Avestruz, and T. Denison, "Design and validation of a fully implantable, chronic, closed-loop neuromodulation device with concurrent sensing and stimulation," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 20, no. 4, pp. 410–421, 2012.

[11] W. Bouthour, P. Mégevand, J. Donoghue, C. Lüscher, N. Birbaumer, and P. Krack, "Biomarkers for closed-loop deep brain stimulation in parkinson disease and beyond," *Nature Reviews Neurology*, vol. 15, pp. 343–352, Jun 2019.

[12] R. Melzack, "The mcgill pain questionnaire: Major properties and scoring methods," *PAIN*, vol. 1, no. 3, 1975.

[13] A. Kucyi and K. D. Davis, "The neural code for pain: From single-cell electrophysiology to the dynamic pain connectome," *The Neuroscientist*, vol. 23, no. 4, pp. 397–414, 2017. PMID: 27660241.

[14] N. Dafny, "Pain principles (section 2, chapter 6) neuroscience online: An electronic textbook for the neurosciences: Department of neurobiology and anatomy - the university of texas medical school at houston."

[15] A. D. B. Craig, "Why a soft touch can hurt," *The Journal of physiology*, vol. 588, pp. 13–13, Jan 2010. 20045900[pmid].

[16] A. F. Keller, S. Beggs, M. W. Salter, and Y. D. Koninck, "Transformation of the output of spinal lamina i neurons after nerve injury and microglia stimulation underlying neuropathic pain," *Molecular Pain*, vol. 3, pp. 1744–8069–3–27, 2007. PMID: 17900333.

[17] J. O. Dostrovsky and A. Craig, *Ascending Projection Systems*. Elsevier, 2013.

[18] N. Vartiainen, C. Perchet, M. Magnin, C. Creac'h, P. Convers, N. Nighoghossian, F. Mauguière, R. Peyron, and L. Garcia-Larrea, "Thalamic pain: anatomical and physiological indices of prediction," *Brain*, vol. 139, pp. 708–722, 02 2016.

[19] A. B. Craig, "Pain mechanisms: Labeled lines versus convergence in central processing," *Annual Review of Neuroscience*, vol. 26, no. 1, pp. 1–30, 2003. PMID: 12651967.

[20] K. D. Davis, A. M. Lozano, M. Manduch, R. R. Tasker, Z. H. T. Kiss, and J. O. Dostrovsky, "Thalamic relay site for cold perception in humans," *Journal of Neurophysiology*, vol. 81, no. 4, pp. 1970–1973, 1999. PMID: 10200232.

[21] A. (Bud) Craig, "A new view of pain as a homeostatic emotion," *Trends in Neurosciences*, vol. 26, no. 6, pp. 303–307, 2003.

[22] M. J. Millan, "Descending control of pain," *Progress in Neurobiology*, vol. 66, no. 6, pp. 355–474, 2002.

[23] W. D. Willis, "Anatomy and physiology of descending control of nociceptive responses of dorsal horn neurons: comprehensive review.," *Progress in brain research*, vol. 77, pp. 1–29, 1988.

[24] D. A. Simpson, M. P. Headley, and B. M. Lumb, "Selective inhibition from the anterior hypothalamus of c- versus a-fibre mediated spinal nociception," *PAIN*, vol. 136, no. 3, 2008.

[25] M. Heinricher, I. Tavares, J. Leith, and B. Lumb, "Descending control of nociception: Specificity, recruitment and plasticity," *Brain Research Reviews*, vol. 60, no. 1, pp. 214–225, 2009. A Decade of Pain Research: New Approaches, New Targets.

[26] J. R. Wolpaw and X. Y. Chen, "Operant Conditioning of Reflexes," *Encyclopedia of Neuroscience*, pp. 225–233, 2009.

[27] S. McMullan and B. M. Lumb, "Midbrain control of spinal nociception discriminates between responses evoked by myelinated and unmyelinated heat nociceptors in the rat," *PAIN*, vol. 124, no. 1, 2006.

[28] R.-D. Treede, D. R. Kenshalo, R. H. Gracely, and A. K. Jones, "The cortical representation of pain," *PAIN*, vol. 79, no. 2, 1999.

[29] L. Mazzola, J. Isnard, R. Peyron, and F. Mauguière, "Stimulation of the human cortex and the experience of pain: Wilder Penfield's observations revisited," *Brain*, vol. 135, pp. 631–640, 10 2011.

[30] M. Lankarany, D. Al-Basha, S. Ratté, and S. A. Prescott, "Differentially synchronized spiking enables multiplexed neural coding," *Proceedings of the National Academy of Sciences*, vol. 116, no. 20, pp. 10097–10102, 2019.

[31] Z.-H. Zhang, P. Dougherty, and S. Oppenheimer, "Monkey insular cortex neurons respond to baroreceptive and somatosensory convergent inputs," *Neuroscience*, vol. 94, no. 2, pp. 351–360, 1999.

[32] W. D. Hutchison, K. D. Davis, A. M. Lozano, R. R. Tasker, and J. O. Dostrovsky, "Pain-related neurons in the human cingulate cortex," *Nature Neuroscience*, vol. 2, pp. 403–405, May 1999.

[33] D. R. Kenshalo, K. Iwata, M. Sholas, and D. A. Thomas, "Response properties and organization of nociceptive neurons in area 1 of monkey primary somatosensory cortex," *Journal of Neurophysiology*, vol. 84, no. 2, pp. 719–729, 2000. PMID: 10938299.

[34] K. Casey, *Sensory, Motivational, and Central Control Determinants of Pain*, pp. 423–439. 01 1968.

[35] E. M. Nagasako, A. L. Oaklander, and R. H. Dworkin, "Congenital insensitivity to pain: an update," *PAIN*, vol. 101, no. 3, 2003.

[36] M. Berthier, S. Starkstein, and R. Leiguarda, "Asymbolia for pain: A sensory-limbic disconnection syndrome," *Annals of Neurology*, vol. 24, no. 1, pp. 41–49, 1988.

[37] M. Ploner, H.-J. Freund, and A. Schnitzler, "Pain affect without pain sensation in a patient with a postcentral lesion," *PAIN*, vol. 81, no. 1, 1999.

[38] A. Kucyi and K. D. Davis, "The dynamic pain connectome," *Trends in Neurosciences*, vol. 38, no. 2, pp. 86–95, 2015.

[39] E. Jonas and K. P. Kording, "Could a neuroscientist understand a microprocessor?," *PLOS Computational Biology*, vol. 13, no. 1, 2017.

[40] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. The MIT Press, 2020.

[41] G. J. Madden, A. M. Begotka, B. R. Raiff, and L. L. Kastern, "Delay discounting of real and hypothetical rewards.," *Experimental and Clinical Psychopharmacology*,

vol. 11, no. 2, p. 139–145, 2003.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[43] X. Guo and H.-L. Onesimo, *Continuous-time Markov decision processes: theory and applications*. Springer-Verlag, 2009.

[44] G. D. Logan and M. J. Crump, "Chapter one - hierarchical control of cognitive processes: The case for skilled typewriting," in *Advances in Research and Theory* (B. H. Ross, ed.), vol. 54 of *Psychology of Learning and Motivation*, pp. 1–27, Academic Press, 2011.

[45] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, pp. 341–379, Oct 2003.

[46] N. Navarro-Guerrero, R. J. Lowe, and S. Wermter, "Improving robot motor learning with negatively valenced reinforcement signals," *Frontiers in Neurorobotics*, vol. 11, p. 10, 2017.

[47] J. García, Fern, and o Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 42, pp. 1437–1480, 2015.

[48] S. Mahadevan, "Average reward reinforcement learning: Foundations, algorithms, and empirical results," *Machine Learning*, vol. 22, pp. 159–195, Mar 1996.

[49] Y. Wan, A. Naik, and R. S. Sutton, "Average-reward learning and planning with options," 2021.

[50] Y. Wan, A. Naik, and R. S. Sutton, "Learning and planning in average-reward markov decision processes," 2021.

[51] A. Naik, R. Shariff, N. Yasui, H. Yao, and R. S. Sutton, "Discounted reinforcement learning is not an optimization problem," 2019.

[52] M. Guitart-Masip, U. R. Beierholm, R. Dolan, E. Duzel, and P. Dayan, "Vigor in the Face of Fluctuating Rates of Reward: An Experimental Examination," *Journal*

of Cognitive Neuroscience, vol. 23, pp. 3933–3938, 12 2011.

[53] Y. Niv, N. Daw, and P. Dayan, "How fast to work: Response vigor, motivation and tonic dopamine," in *Advances in Neural Information Processing Systems* (Y. Weiss, B. Schölkopf, and J. Platt, eds.), vol. 18, MIT Press, 2005.

[54] Y. Niv, N. D. Daw, D. Joel, and P. Dayan, "Tonic dopamine: opportunity costs and the control of response vigor," *Psychopharmacology*, vol. 191, pp. 507–520, Apr 2007.

[55] S. W. Lee, J. P. O'Doherty, and S. Shimojo, "Neural computations mediating one-shot learning in the human brain," *PLOS Biology*, vol. 13, no. 4, 2015.

[56] R. C. Bolles, "The avoidance learning problem11supported by research grant gb-20801 from the national science foundation.," vol. 6 of *Psychology of Learning and Motivation*, pp. 97–145, Academic Press, 1972.

[57] N. T. Mackintosh, *CONDITIONING AND ASSOCIATIVE LEARNING*. CLARENDON PRESS, 1983.

[58] C. R. Gallistel, A. R. Craig, and T. A. Shahan, "Contingency, contiguity, and causality in conditioning: Applying information theory and weber's law to the assignment of credit problem.," *Psychological Review*, vol. 126, no. 5, pp. 761–773, 2019.

[59] R. S. Sutton, "Temporal credit assignment in reinforcement learning."

[60] A. K. Agogino and K. Tumer, "Unifying temporal and structural credit assignment problems," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '04, (USA), p. 980–987, IEEE Computer Society, 2004.

[61] S. W. Lee, S. Shimojo, and J. P. O'Doherty, "Neural computations underlying arbitration between model-based and model-free learning," *Neuron*, vol. 81, pp. 687–699, Feb 2014. 24507199[pmid].

[62] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," 2016.

[63] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," 2017.

[64] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. V. de Wiele, V. Mnih, N. Heess, and J. T. Springenberg, "Learning by playing - solving sparse reward tasks from scratch," 2018.

[65] R. Hafner, T. Hertweck, P. Klöppner, M. Bloesch, M. Neunert, M. Wulfmeier, S. Tunyasuvunakool, N. Heess, and M. Riedmiller, "Towards general and autonomous learning of core skills: A case study in locomotion," 2020.

[66] B. A. Richards, T. P. Lillicrap, P. Beaudoin, Y. Bengio, R. Bogacz, A. Christensen, C. Clopath, R. P. Costa, A. de Berker, S. Ganguli, C. J. Gillon, D. Hafner, A. Kepecs, N. Kriegeskorte, P. Latham, G. W. Lindsay, K. D. Miller, R. Naud, C. C. Pack, P. Poirazi, P. Roelfsema, J. Sacramento, A. Saxe, B. Scellier, A. C. Schapiro, W. Senn, G. Wayne, D. Yamins, F. Zenke, J. Zylberberg, D. Therien, and K. P. Kording, "A deep learning framework for neuroscience," *Nature Neuroscience*, vol. 22, pp. 1761–1770, Nov 2019.

[67] B. Seymour and F. Mancini, "Hierarchical models of pain: Inference, information-seeking, and adaptive control.," *NeuroImage*, vol. 222, p. 117212, 2020.

[68] B. Seymour and R. Dolan, *Emotion, Motivation, and Pain*. Elsevier, 2013.

[69] G. Gigerenzer and H. Brighton, "Homo heuristicus: Why biased minds make better inferences," *Topics in Cognitive Science*, vol. 1, no. 1, pp. 107–143, 2009.

[70] H. M. Dorfman and S. J. Gershman, "Controllability governs the balance between pavlovian and instrumental action selection," *Nature Communications*, vol. 10, p. 5826, Dec 2019.

[71] T. Singer, B. Seymour, J. O'Doherty, H. Kaube, R. Dolan, and C. Frith, "Empathy for pain involves the affective but not sensory components of pain," *Science (New*

York, N.Y.), vol. 303, pp. 1157–62, 03 2004.

[72] E. Desjardins, "Reflections on path dependence and irreversibility: Lessons from evolutionary biology," *Philosophy of Science*, vol. 78, no. 5, p. 724–738, 2011.

[73] K. K. Petersen, H. B. Vaegter, A. Stubhaug, A. Wolff, B. E. Scammell, L. Arendt-Nielsen, and D. B. Larsen, "The predictive value of quantitative sensory testing: a systematic review on chronic postoperative pain and the analgesic effect of pharmacological therapies in patients with chronic pain," *PAIN*, vol. 162, no. 1, 2021.

[74] B. W. Balleine and A. Dickinson, "Goal-directed instrumental action: contingency and incentive learning and their cortical substrates," *Neuropharmacology*, vol. 37, no. 4, pp. 407–419, 1998.

[75] F. von Dincklage, K. Send, M. Hackbarth, B. Rehberg, and J. Baars, "Comparison of the nociceptive flexion reflex threshold and the bispectral index as monitors of movement responses to noxious stimuli under propofol mono-anaesthesia," *British Journal of Anaesthesia*, vol. 102, no. 2, pp. 244–250, 2009.

[76] D. Rice, J. Nijs, E. Kosek, T. Wideman, M. I. Hasenbring, K. Koltyn, T. Graven-Nielsen, and A. Polli, "Exercise-induced hypoalgesia in pain-free and chronic pain populations: State of the art and future directions," *The Journal of Pain*, vol. 20, no. 11, pp. 1249–1266, 2019.

[77] Y. Yan, D. Hewitt, L. Hunt, and B. Seymour, "The cognitive basis of pain: arbitration between body-model and world-model pain avoidance learning," in *Conference on Cognitive Computational Neuroscience 2023*, vol. P-3A.54.

[78] R. J. Bufacchi and G. D. Iannetti, "An action field theory of peripersonal space," *Trends in Cognitive Sciences*, vol. 22, no. 12, pp. 1076–1090, 2018.

[79] C. L. Peterson, Z. A. Riley, E. T. Krepkovich, W. M. Murray, and E. J. Perreault, "Withdrawal reflexes in the upper limb adapt to arm posture and stimulus location," *Muscle & Nerve*, vol. 49, no. 5, pp. 716–723, 2014.

[80] E. G. Spaich, L. Arendt-Nielsen, and O. K. Andersen, "Modulation of lower limb withdrawal reflexes during gait: A topographical study," *Journal of Neurophysiol-*

ogy, vol. 91, no. 1, pp. 258–266, 2004. PMID: 12968008.

[81] A. Icenhour, L. Petrakova, N. Hazzan, N. Theysohn, C. J. Merz, and S. Elsenbruch, "When gut feelings teach the brain to fear pain: Context-dependent activation of the central fear network in a novel interoceptive conditioning paradigm," *NeuroImage*, vol. 238, p. 118229, 2021.

[82] A. Smith, K. Carlow, T. Biddulph, B. Murray, M. Paton, and D. S. Harvie, "Contextual modulation of pain sensitivity utilising virtual environments," *British Journal of Pain*, vol. 11, no. 2, pp. 71–80, 2017. PMID: 28491299.

[83] I. Käthner, T. Bader, and P. Pauli, "Heat pain modulation with virtual water during a virtual hand illusion," *Scientific Reports*, vol. 9, p. 19137, Dec 2019.

[84] F. Dabek, N. Zeldovich, F. Kaashoek, D. Mazières, and R. Morris, "Event-driven programming for robust software," in *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, EW 10, (New York, NY, USA), p. 186–189, Association for Computing Machinery, 2002.

[85] M. Karsten and S. Barghi, "User-level threading: Have your cake and eat it too," in *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '20, (New York, NY, USA), p. 89–90, Association for Computing Machinery, 2020.

[86] C. G. Cassandras, "The event-driven paradigm for control, communication and optimization," *Journal of Control and Decision*, vol. 1, no. 1, pp. 3–17, 2014.

[87] B. Stopford, *Designing event-driven systems*. O'Reilly Media, Incorporated, 2018.

[88] S. Tong, T. Denison, S. W. Lee, and B. Seymour, "A novel pain measurement tool by modelling free-operant foraging behaviour in immersive virtual reality," in *Computational and Systems Neuroscience (COSYNE) 2025*, vol. 1-137.

[89] M. N. Petrucci, K. B. Wilkins, G. C. Orthlieb, Y. M. Kehnemouyi, J. J. O'Day, J. A. Herron, and H. M. Bronte-Stewart, "Ramp rate evaluation and configuration for safe and tolerable closed-loop deep brain stimulation," in *2021 10th International IEEE/EMBS Conference on Neural Engineering (NER)*, pp. 959–962, 2021.

[90] J. H. Shin, J. H. Lee, S. Tong, S. W. Kim, and S. W. Lee, "Designing model-based and model-free reinforcement learning tasks without human guidance," in *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*.

[91] C. Eccleston, E. Fisher, S. Liikkanen, T. Sarapohja, C. Stenfors, S. K. Jääskeläinen, A. S. Rice, L. Mattila, T. Blom, and J. R. Bratty, "A prospective, double-blind, pilot, randomized, controlled trial of an "embodied" virtual reality intervention for adults with low back pain," *PAIN*, vol. 163, no. 9, 2022.

[92] J. H. Shin and S. W. Lee, "Simulation-based behavioral profiling by model-guided task optimization and task-guided data generation," in *Computational and Systems Neuroscience (COSYNE) 2025*, vol. 3-013.

[93] J. Melthis, S. Tang, P. Yang, M. Hanneghan, and C. Carter, "Topologies for combining the internet of things and serious games," *Journal of Intelligent and Fuzzy Systems*, vol. 31, pp. 2685–2696, October 2016. Copyright 2016, with permission from IOS Press. (www.iospress.nl) The publication is available at IOS Press through http://dx.doi.org/10.3233/JIFS-169108.

[94] E. L. Charnov, "Optimal foraging, the marginal value theorem," *Theoretical Population Biology*, vol. 9, no. 2, pp. 129–136, 1976.

[95] M. Lihoreau, L. Chittka, and N. Raine, "Travel optimization by foraging bumblebees through readjustments of traplines after discovery of new feeding locations.," *The American Naturalist*, vol. 176, no. 6, pp. 744–757, 2010. PMID: 20973670.

[96] S. M. Constantino and N. D. Daw, "Learning the opportunity cost of time in a patch-foraging task," *Cognitive, Affective, & Behavioral Neuroscience*, vol. 15, pp. 837–853, Dec 2015.

[97] D. Matthews, E. Elgueta Cancino, D. Falla, and A. Khatibi, "Exploring pain interference with motor skill learning in humans: a protocol for a systematic review," *BMJ Open*, vol. 11, no. 7, 2021.

[98] P. W. Hodges, "Pain and motor control: From the laboratory to rehabilitation," *Journal of Electromyography and Kinesiology*, vol. 21, no. 2, pp. 220–228, 2011.

[99] I. Kurniawan, B. Seymour, I. Vlaev, J. Trommershäuser, R. Dolan, and N. Chater, "Pain relativity in motor control," *Psychological Science*, vol. 21, no. 6, pp. 840–847, 2010. PMID: 20435952.

[100] A. Nair, R. K. Niyogi, F. Shang, S. J. Tabrizi, G. Rees, and R. B. Rutledge, "Opportunity cost determines free-operant action initiation latency and predicts apathy," *Psychological Medicine*, vol. 53, no. 5, p. 1850–1859, 2023.

[101] K. H. Kowal, "The range effect as a function of stimulus set, presence of a standard, and modulus," *Percept. Psychophys.*, vol. 54, pp. 555–561, Oct. 1993.

[102] S. S. Stevens, "On the psychophysical law.," *Psychological Review*, vol. 64, no. 3, pp. 153–181, 1957.

[103] M. B. Jensen, J. Biurrun Manresa, and O. K. Andersen, "Reliable estimation of nociceptive withdrawal reflex thresholds," *Journal of Neuroscience Methods*, vol. 253, pp. 110–115, 2015.

[104] D. Bates, M. Mächler, B. Bolker, and S. Walker, "Fitting linear mixed-effects models using lme4," *Journal of Statistical Software*, vol. 67, no. 1, pp. 1–48, 2015.

[105] A. Kuznetsova, P. B. Brockhoff, and R. H. B. Christensen, "lmerTest package: Tests in linear mixed effects models," *Journal of Statistical Software*, vol. 82, no. 13, pp. 1–26, 2017.

[106] E. Criswell, *Cram's introduction to surface electromyography*. Jones & Bartlett Publishers, 2010.

[107] H. J. McQUAY, D. CARROLL, and C. GLYNN, "Dose-response for analgesic effect of amitriptyline in chronic pain," *Anaesthesia*, vol. 48, no. 4, pp. 281–285, 1993.

[108] M. T. Pedersen, L. L. Andersen, M. B. Jørgensen, K. Søgaard, and G. Sjøgaard, "Effect of specific resistance training on musculoskeletal pain symptoms: Dose-response relationship," *The Journal of Strength & Conditioning Research*, vol. 27, no. 1, 2013.

[109] H. Brown, R. A. Adams, I. Parees, M. Edwards, and K. Friston, "Active inference, sensory attenuation and illusions," *Cognitive Processing*, vol. 14, pp. 411–427, Nov 2013.

[110] B. Seymour, J. P. O'Doherty, P. Dayan, M. Koltzenburg, A. K. Jones, R. J. Dolan, K. J. Friston, and R. S. Frackowiak, "Temporal difference models describe higher-order learning in humans," *Nature*, vol. 429, pp. 664–667, Jun 2004.

[111] J. Gläscher, N. Daw, P. Dayan, and J. P. O'Doherty, "States versus rewards: Dissociable neural prediction error signals underlying model-based and model-free reinforcement learning," *Neuron*, vol. 66, no. 4, pp. 585–595, 2010.

[112] D. R. Bach, G. Flandin, K. J. Friston, and R. J. Dolan, "Time-series analysis for rapid event-related skin conductance responses," *Journal of Neuroscience Methods*, vol. 184, no. 2, pp. 224–234, 2009.

[113] D. R. Bach, G. Flandin, K. J. Friston, and R. J. Dolan, "Modelling event-related skin conductance responses," *International Journal of Psychophysiology*, vol. 75, no. 3, pp. 349–356, 2010.

[114] J. L. Orquin and S. Mueller Loose, "Attention and choice: A review on eye movements in decision making," *Acta Psychologica*, vol. 144, no. 1, pp. 190–206, 2013.

[115] D. Anderson, "Optimal foraging and the traveling salesman," *Theoretical Population Biology*, vol. 24, no. 2, pp. 145–159, 1983.

[116] J. Y. Jeon, W. M. Shim, and S. B. M. Yoo, "Generative model for explaining spatial regularity detection behavior of humans while foraging," in *Conference on Cognitive Computational Neuroscience 2023*, vol. P-3A.12.

[117] M. Rothkirch, K. Schmack, F. Schlagenhauf, and P. Sterzer, "Implicit motivational value and salience are processed in distinct areas of orbitofrontal cortex," *NeuroImage*, vol. 62, no. 3, pp. 1717–1725, 2012.

[118] A. Rangel and T. Hare, "Neural computations associated with goal-directed choice," *Current Opinion in Neurobiology*, vol. 20, no. 2, pp. 262–270, 2010. Cognitive neuroscience.

[119] P. C. Holland and M. Gallagher, "Amygdala–frontal interactions and reward expectancy," *Current Opinion in Neurobiology*, vol. 14, no. 2, pp. 148–155, 2004.

[120] P. D. Wall, "On the relation of injury to pain the john j. bonica lecture," *PAIN*, vol. 6, no. 3, 1979.

[121] R. L. Waters, J. S. Yakura, R. H. Adkins, and I. Sie, "Recovery following complete paraplegia," *Archives of Physical Medicine and Rehabilitation*, vol. 73, pp. 784–789, Sep 1992.

[122] F. Mancini, A. Bauleo, J. Cole, F. Lui, C. A. Porro, P. Haggard, and G. D. Iannetti, "Whole-body mapping of spatial acuity for pain and touch," *Annals of Neurology*, vol. 75, no. 6, pp. 917–924, 2014.

[123] R. Chakraverty, P. Pynsent, and K. Isaacs, "Which spinal levels are identified by palpation of the iliac crests and the posterior superior iliac spines?," *Journal of Anatomy*, vol. 210, no. 2, pp. 232–236, 2007.

[124] T. M. Cummins, M. M. Kucharczyk, T. Graven-Nielsen, and K. Bannister, "Activation of the descending pain modulatory system using cuff pressure algometry: Back translation from man to rat," *European Journal of Pain*, vol. 24, no. 7, pp. 1330–1338, 2020.

[125] L. B. Sørensen, P. Gazerani, K. A. Sluka, and T. Graven-Nielsen, "Repeated Injections of Low-Dose Nerve Growth Factor (NGF) in Healthy Humans Maintain Muscle Pain and Facilitate Ischemic Contraction–Evoked Pain," *Pain Medicine*, vol. 21, pp. 3488–3498, 10 2020.

[126] N. Singla, P. J. Desjardins, E. B. Cosca, C. Parulan, A. Arriaga, K. C. Poole, D. M. Batz, and P. D. Chang, "Delayed-onset muscle soreness: a pilot study to assess analgesic study design features," *PAIN*, vol. 156, no. 6, 2015.

[127] B. Seymour, T. Singer, and R. Dolan, "The neurobiology of punishment," *Nature Reviews Neuroscience*, vol. 8, pp. 300–311, Apr 2007.

[128] F. Mancini, P. Mahajan, A. á. V. Guttesen, J. Onysk, I. Scholtes, N. Shenker, M. Lee, and B. Seymour, "Enhanced behavioural and neural sensitivity to punishments in

chronic pain and fatigue," *bioRxiv*, 2024.

[129] D. L. Kennedy, H. I. Kemp, D. Ridout, D. Yarnitsky, and A. S. Rice, "Reliability of conditioned pain modulation: a systematic review," *PAIN*, vol. 157, no. 11, 2016.

[130] T. Graven-Nielsen, H. B. Vaegter, S. Finocchietti, G. Handberg, and L. Arendt-Nielsen, "Assessment of musculoskeletal pain sensitivity and temporal summation by cuff pressure algometry: a reliability study," *PAIN*, vol. 156, no. 11, 2015.

[131] T. Joseph, M. Mansfield, T. Cummins, S. MacMahon, A. Fielding, T. Graven-Nielsen, and K. Bannister, "Does activation of a descending modulatory control impact the development of secondary hyperalgesia," Sept. 2022. The World Congress on Pain (IASP 2022), IASP 2022 ; Conference date: 19-09-2022 Through 23-09-2022.

[132] J. D. Favero, C. Luck, O. V. Lipp, A. T. Nguyen, and W. Marinovic, "N1-p2 event-related potentials and perceived intensity are associated: The effects of a weak pre-stimulus and attentional load on processing of a subsequent intense stimulus," *Biological Psychology*, vol. 184, p. 108711, 2023.

[133] E. N. van den Broeke, C. M. van Rijn, J. A. Biurrun Manresa, O. K. Andersen, L. Arendt-Nielsen, and O. H. G. Wilder-Smith, "Neurophysiological correlates of nociceptive heterosynaptic long-term potentiation in humans," *Journal of Neurophysiology*, vol. 103, no. 4, pp. 2107–2113, 2010. PMID: 20164395.

[134] A. Levy, M. Enisman, A. Perry, and T. Kleiman, "Midfrontal theta as an index of conflict strength in approach–approach vs avoidance–avoidance conflicts," *Social Cognitive and Affective Neuroscience*, vol. 18, p. nsad038, 07 2023.

[135] E. R. Chrastil, C. Rice, M. Goncalves, K. N. Moore, S. C. Wynn, C. E. Stern, and E. Nyhus, "Theta oscillations support active exploration in human spatial navigation," *NeuroImage*, vol. 262, p. 119581, 2022.

[136] E. Schulz, E. S. May, M. Postorino, L. Tiemann, M. M. Nickel, V. Witkovsky, P. Schmidt, J. Gross, and M. Ploner, "Prefrontal Gamma Oscillations Encode Tonic Pain in Humans," *Cerebral Cortex*, vol. 25, pp. 4407–4414, 03 2015.

[137] R. D. Pascual-Marqui, "Standardized low-resolution brain electromagnetic tomography (sloreta): technical details," *Methods and Findings in Experimental and Clinical Pharmacology*, vol. 24, no. Suppl D, pp. 5–12, 2002.

[138] E. Frei, A. Gamma, R. Pascual-Marqui, D. Lehmann, D. Hell, and F. X. Vollenweider, "Localization of mdma-induced brain activity in healthy volunteers using low resolution brain electromagnetic tomography (loreta)," *Human Brain Mapping*, vol. 14, no. 3, pp. 152–165, 2001.

[139] C. Gil Ávila, F. S. Bott, L. Tiemann, V. D. Hohn, E. S. May, M. M. Nickel, P. T. Zebhauser, J. Gross, and M. Ploner, "Discover-eeg: an open, fully automated eeg pipeline for biomarker discovery in clinical neuroscience," *Scientific Data*, vol. 10, p. 613, Sep 2023.

[140] R. D. Pascual-Marqui, "Notes on some physics, mathematics, and statistics for time and frequency domain loreta," December 8 2002. Available online or unpublished notes.

[141] B. Seymour, R. J. Crook, and Z. S. Chen, "Post-injury pain and behaviour: a control theory perspective," *Nature Reviews Neuroscience*, vol. 24, pp. 378–392, Jun 2023.

[142] R. J. Gatchel, R. Neblett, N. Kishino, and C. T. Ray, "Fear-avoidance beliefs and chronic pain," *Journal of Orthopaedic & Sports Physical Therapy*, vol. 46, no. 2, pp. 38–43, 2016. PMID: 26828236.

[143] H. Shuchang, H. Mingwei, J. Hongxiao, W. Si, Y. Xing, D. Antonius, and M. G. Opler, "Emotional and neurobehavioural status in chronic pain patients," *Pain Research and Management*, vol. 16, no. 1, p. 825636, 2011.

[144] A. Borisovskaya, E. Chmelik, and A. Karnik, *Exercise and Chronic Pain*, pp. 233–253. Singapore: Springer Nature Singapore, 2020.

[145] W. Schultz, P. Dayan, and P. R. Montague, "A neural substrate of prediction and reward," *Science*, vol. 275, no. 5306, pp. 1593–1599, 1997.

[146] R. Sitaram, T. Ros, L. Stoeckel, S. Haller, F. Scharnowski, J. Lewis-Peacock, N. Weiskopf, M. L. Blefari, M. Rana, E. Oblak, N. Birbaumer, and J. Sulzer,

"Closed-loop brain training: the science of neurofeedback," *Nature Reviews Neuroscience*, vol. 18, pp. 86–100, Feb 2017.

[147] J. Fernández-Alvarez, M. Grassi, D. Colombo, C. Botella, P. Cipresso, G. Perna, and G. Riva, "Efficacy of bio- and neurofeedback for depression: a meta-analysis," *Psychological Medicine*, vol. 52, no. 2, p. 201–216, 2022.

[148] N. Gninenko, S. Trznadel, D. Daskalou, L. Gramatica, J. Vanoy, F. Voruz, C. L. Robyn, A. Spadazzi, A. Yulzari, R. Sitaram, D. Van De Ville, P. Senn, and S. Haller, "Functional mri neurofeedback outperforms cognitive behavioral therapy for reducing tinnitus distress: A prospective randomized clinical trial," *Radiology*, vol. 310, no. 2, p. e231143, 2024. PMID: 38349241.

[149] R. Roy, R. de la Vega, M. P. Jensen, and J. Miró, "Neurofeedback for pain management: A systematic review," *Frontiers in Neuroscience*, vol. 14, 2020.

[150] T. Kucina, L. Wells, I. Lewis, K. de Salas, A. Kohl, M. A. Palmer, J. D. Sauer, D. Matzke, E. Aidman, and A. Heathcote, "Calibration of cognitive tests to address the reliability paradox for decision-conflict tasks," *Nature Communications*, vol. 14, p. 2234, Apr 2023.

[151] S. Zorowitz, G. Karni, N. Paredes, N. Daw, and Y. Niv, "Improving the reliability of the pavlovian go/no-go task for computational psychiatry research," 2023.

9 Appendix

9.1 Specifications for the communication protocol

The data layout for a single one-way transmission containing data is described in Figure 52. Replies are either “OK” or “FAIL”, so no need to include 4 bytes data size in a reply. The protocol is an application layer protocol built on TCP/IP. Strings are encoded in UTF-8; numerical values are in big endianness.

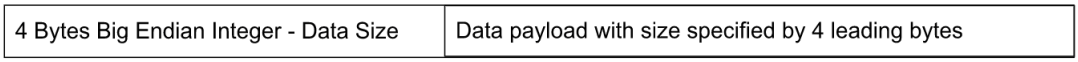


Figure 52: Single one-way transmission data layout for both device side and server side

The communication between the server and the client is divided into three phases:

9.1.1 Register phase

Once a device establishes connection to the server, it then needs to register itself on the server program before sending anything else. The format of data payload follows JavaScript Object Notation (JSON) standard and must specify the following fields to register successfully.

fields	description
name	Device name (human readable)
device_id	Unique, can be randomly generated or manually make a readable unique name
device_type	Type of the device (human readable)
data_to_report	A dictionary (object) of data fields and their types to be sent back to server
data_to_control	A dictionary (object) of data fields paired with their types that is controllable by the server
report_pack_order	If not null, the values specified in data_to_report are ordered in this given list and packed in a single data payload. Sometimes the device would like to selectively report data, then use null here and it must follow JSON format.

control_pack_order	If not null, the values specified in data_to_control are ordered in this given list and packed in a single data payload. Sometimes the server would like to selectively control some values, then use null here and it must follow JSON format.

Table 3: The descriptor of a device used to register on the server

The descriptor can optionally include a “visual_report” field specifying how the data can be visualised. Some suggested visualisation types are:

line_chart	A line chart
static	Just display the value
image/<format>	Decode the data as image data with given format

Table 4: Visual report style suggestions

Similarly, the descriptor can optionally include a “visual_control” field specifying how control data can be visually presented. Some suggested visualisation types are:

static	Plain input box
switch	Binary data controlled by switch buttons
slider	Numerical value adjusted by sliders

Table 5: Visual control input style

The data types to be supported by both the client and the server are listed below:

Type specifier	description
int	4 Bytes integer.
float	8 Bytes, double precision.
bool	4 Bytes, 0 represents false, others are true.
category	4 Bytes integer. Need to have another data field with name [name of the categorical variable]_category with value being a dictionary (object) of integer and its corresponding categorical name.

binary	4 Bytes leading integer specifying size. Binary data payload follows.
[tuple_length, data type]	A tuple type with fixed length. The type of elements in the tuple is one the types listed above.

Table 6: Mutually supported data types

The server should send an “OK” string in the data payload acknowledging the success of registration, otherwise the server sends a “FAIL”.

9.1.2 Data report phase

The device reports data to the server.

The server should send an “OK” string in data payload acknowledging the success of receiving the device’s data payload, otherwise the server sends a “FAIL” if it fails to understand the data format. In this case, the device can send data again. If the device does not receive a reply for a certain amount of time (device specific), then it can try to reconnect. It is not allowed to send anything else if not heard from the server once the data is all sent.

9.1.3 Server control phase

The server sends control data to the device.

The device should send an “OK” string in data payload acknowledging the success of receiving and executing server’s commands, otherwise the device sends a “FAIL” if it fails to execute given commands. The server should wait for the device to respond before sending more control data, otherwise it can only wait for the device’s response or reconnection.

9.2 Ethical approval

Data collection with real human participants described in Chapter 3, 4, 5, 6 has been approved by Medical Sciences Interdivisional Research Ethics Committee (Reference number R58778 with multiple amendments R58778/RE001, R58778/RE002, R58778/RE003, R58778/RE004, R58778/RE005, and R58778/RE006).

9.3 Justifications and safety considerations for tonic pain stimulation with pressure cuffs

Tonic pain stimuli will be delivered to participants using double or single-chamber pressure cuffs. Tonic pain stimulation may be delivered alone or with phasic pain stimulation at the same time. Cuffs are wrapped around arm or leg muscle groups. Cuffs are inflated using either manual inflator or automatic inflator. We have two types of cuffs, one for arm use and one for leg use. The cuffs are Reusable Tourniquet Cuffs manufactured by VBM Medizintechnik GmbH. Another research group in King's College London (KCL) is using the same cuff for pain stimulation on healthy participants. The arm cuff's size is 35cm long x 14cm wide, and the leg cuff's size is 61cm long x 24cm wide. At most one pressure cuff will have positive pressure at any given time point

The pressure will be applied for a max duration of 3 minutes continuously. In 24 hours or in one experiment visit, total pressure application is restricted to less than 20 minutes. We also set a minimum break time between two pressure applications, which is equal to the duration of the previous pressure application.

As the data of pressure pain threshold measured in previous studies, the pressure to induce pain is generally above systolic pressure. Therefore, the pressure cuffs will occlude blood flow. We do not explicitly monitor occlusion as the pressure application is brief (within 3 minutes) and the occlusion effect is temporary. Arterial occlusion is not a condition for the pressure to be used. The pain starts immediately after the pressure increases, so it is unlikely caused by ischemia, but rather caused by deep mechanical nociceptor activation.

Pain perception varies greatly for different individuals. From the data in the literature, the pressure pain detection threshold is generally below 750 mmHg. From the raw data collected in KCL, they actually saw several participants had pain tolerance thresholds (which participants would describe as intolerable if go higher) above 750 mmHg. Their machine can only deliver up to 750 mmHg, so they did not find out the true pain tolerance threshold of those people. The pressure inflator we currently have has a measurement range of up to 600 mmHg. Therefore, in practice, with current devices, we will are unlikely to exceed 600 mmHg. The pressure cuff is deformable, so when people move their arms

or legs, the contraction and relaxation of muscles can change the pressure. Generally, the pressure will show a gradually decreasing trend right after applying because the cuff slowly expands. 750 mmHg is an upper limit that is used in many pressure cuff pain stimulation studies, and we will keep the pressure below this value at all times.

In addition to that, we use a manual mechanical inflator with a safety valve that can release all pressure immediately. We will show participants how to operate the safety valve (it is simple) to give them the ability to release the pressure themselves should they wish. We will always tell the participants before the experiment starts that they can release the pressure at any point of time.

All the lab members (N=6) tried the pressure cuff, and we have tried the pressure up to 600 mmHg. Within the lab, people may find it painful somewhere around 200 mmHg to 350 mmHg (pressure pain detection threshold), so we expect thisn to be the typical range used in the experiment. The pain starts immediately after the pressure rises. So the pain is unlikely caused by ischemia but rather by activating deep nociceptors in the muscles.

We impose the following safety measures to ensure the cuff system is safe:

- Maximum pressure is limited to 100 kPa (750 mmHg)
- Maximum single pressure pain stimulus duration: 3 minutes
- Minimum break duration between two pressure pain stimuli: duration of previous pressure pain stimulus
- Total pressure pain stimuli time for one participant in one experiment visit and in 24 hours: 20 minutes
- Pressure cuff system is equipped with safety valves that release all air in the cuff chamber safely in an emergency
- Exclude participants with osteoporosis, hyptertension (over 140/90 mmHg), hypotension (below 90/60 mmHg), circulation problems including peripheral arterial disease or chronic venous insufficiency, diabetes, and existing painful conditions including neuropathic pain, Raynaud's and regional pain syndromes

- At most one pressure cuff will have positive pressure at any given time point. The pressure cuff is either applied on leg or arm.

The above safety limits are well within the limits in published studies that have been carried out. In a study (N=35) using the cuff system (Lemming et al., 2012) carried out in Linköping University Hospital in Sweden, they applied tonic stimulation with pressure cuff for 10 minutes at 25kPa, followed by 7 minutes zero pressure break, and another stimulation for 10 minutes with stimulation intensity being the mean value of pain detection threshold and pain tolerance threshold. The max pressure of their system is set to 100 kPa. To give a reference of how long the pressure can be applied, a study (N=12) applied cuff compression with the cuff being inflated to 40 mmHg above the systolic blood pressure and controlled regularly throughout the inflation period of 60 min (Laursen et al., 1999). Kristian Hennings, Adjunct Associate Professor at Aalborg University and Founder of Inventors' Way which manufactures cuff algometry for pain stimulation, replied to us in the email: "For persons without diseases such as osteoporosis, 100kPa is safe. Please don't use the device on osteoporosis patients; you should have that as an exclusion criterion in your experiments with cuff pressure algometry. However, 100kPa occludes the blood supply, so for safety reasons, continuous pressure stimulations have been limited to 10min. You need to occlude the blood supply for 30min-60min before there is a chance of nerve injury; 10 min has been selected to have a wide safety margin. One minute is not a problem, and to apply that in 20-30 blocks with an interval of 1 minute in between is acceptable with respect to safety."

The same system has been practiced in the UK. A research group in King's College London uses it to study conditioned pain modulation (Joseph et al., 2022). We visited the group and sourced the pressure cuff from the same supplier they are using.

Here is a table listing previous studies using pressure cuffs for experimental pain stimulation and their parameters:

References:

Joseph, T., Mansfield, M., Cummins, T., MacMahon, S., Fielding, A., Graven-Nielsen, T., & Bannister, K. (2022). Does activation of a descending modulatory control impact the

Studies	Pressure	Duration	Remarks
Joseph et al., 2022	Max 100 kPa (750 mmHg)	Applied concurrently with phasic pain stimulation	This study is carried out at King's College London. We visited their lab in October 2022 last year to learn how the pressure cuff system works.
Laursen et al., 1999	40 mmHg above the systolic blood pressure	60 minutes continuously	12 participants from the university (mean age: 26.1 years; range: 24 to 33 years)
Lemming et al., 2012	25 kPa (187 mmHg) constant pressure session + 36.4 kPa (± 3.3) (273 ± 24.8 mmHg) individual pressure session.	During testing, the max pressure limit is 100 kPa (750 mmHg)	Constant pressure session (10 minutes) + zero pressure (7 minutes) + individual pressure session (10 minutes) Participants include whiplash associated disorder (n = 25) and controls (n = 10). WAD participants have lower individual pressure (measured from pain thresholds)
Polianskis et al., 2001	Max 180 kPa safety limit preset in the cuff algometry	Stops when the participant presses the button at pain detection threshold and pain tolerance threshold	Although they specified 180 kPa (1350 mmHg) as the max pressure, the highest statistics they reported is on the left gastrocnemius-soleus (GS) muscle: 60.8 kPa ± 3.8 (456 mmHg ± 28.5)
Graven-Nielsen et al., 2016a	Max 80 kPa (600 mmHg)	Ten repeated cuff pressure stimulations (2-second duration and 1-second interval between stimuli)	
Graven-Nielsen et al., 2016b	Max 100 kPa (750 mmHg)	60 seconds continuously at pressure equals to visual analog scale rating 7	

Table 7: Comparison of studies using pressure cuffs for experimental pain stimulations

development of secondary hyperalgesia. Poster session presented at The World Congress on Pain (IASP 2022), Toronto.

Laursen, R. J., Graven-Nielsen, T., Jensen, T. S., & Arendt-Nielsen, L. (1999). The effect of compression and regional anaesthetic block on referred pain intensity in humans. Pain, 80(1), 257–263. https://doi.org/10.1016/s0304-3959(98)00214-0

Lemming, D., Graven-Nielsen, T., Sörensen, J., Arendt-Nielsen, L., & Gerdle, B. (2012). Widespread pain hypersensitivity and facilitated temporal summation of deep tissue pain in whiplash associated disorder: An explorative study of women. Journal of Rehabilitation Medicine, 44(8), 648–657. https://doi.org/10.2340/16501977-1006

Graven-Nielsen, T., Izumi, M., Petersen, K. K., & Arendt-Nielsen, L. (2016a). User-independent assessment of conditioning pain modulation by cuff pressure algometry. European Journal of Pain, 21(3), 552–561. https://doi.org/10.1002/ejp.958

Graven-Nielsen, T., Izumi, M., Petersen, K. K., & Arendt-Nielsen, L. (2016b). User-independent assessment of conditioning pain modulation by cuff pressure algometry. European Journal of Pain, 21(3), 552–561. https://doi.org/10.1002/ejp.958

Polianskis, R., Graven-Nielsen, T., & Arendt-Nielsen, L. (2001). Computer-controlled pneumatic pressure algometry-a new technique for quantitative sensory testing. European Journal of Pain, 5(3), 267–277. https://doi.org/10.1053/eujp.2001.0245

9.4 Additional Figures

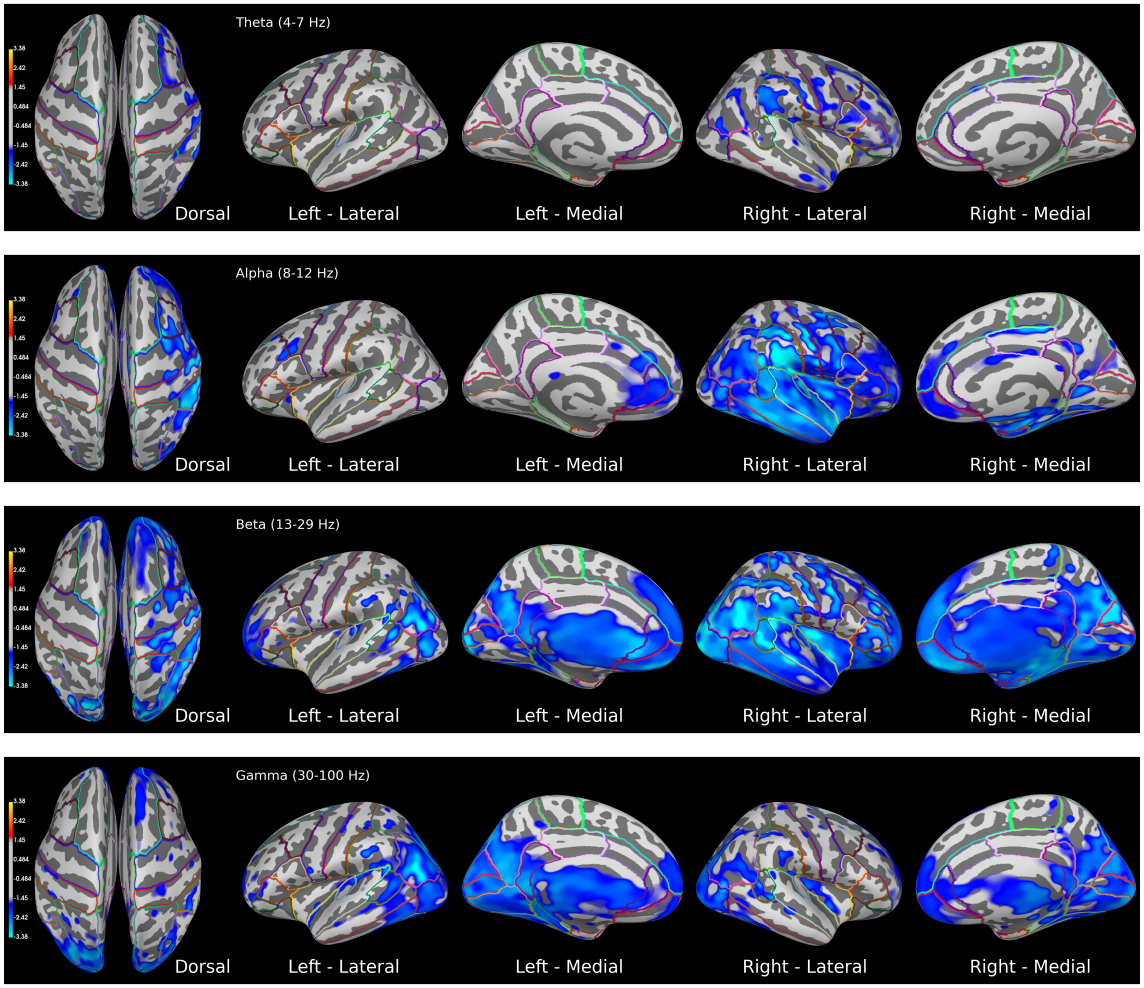


Figure 53: t-values for the effect of tonic pain in source space, generated from the LMM. Colours are thresholded at 1.310 for visibility with transparency, 1.697 for non-transparency, and 3.385 for the highest value on the color map. APARC parcellation was overlapped for visual inspection. Frequency band notations represented the pass band used for filtering the signal in each epoch before applying sLORETA in time-domain.

9.5 Core source code

We present selected core source code for software implementation and data analysis.

9.5.1 Implementation core source

9.5.1.1 Wireless physiological sensor system

```
#include <WiFiNINA.h>

#include "wpa_cred.h"
#include "device_descriptor.h"

int status = WL_IDLE_STATUS;              // the Wi-Fi radio's status
```

```
int ledState = LOW;                         //ledState used to set the
    ↪ LED
unsigned long previousMicrosInfo = 0;       //will store last time Wi-Fi
    ↪ information was updated
unsigned long previousMicrosReport = 0;
const unsigned long reportInterval = 30000;            // interval at
    ↪ which to update the board information
const unsigned long collectInterval = 1000;            // interval at
    ↪ which to collect analog signal
byte buffer[4000]; // micros (4 bytes) + uint16_t (2 bytes) * 3 = 10
    ↪ bytes per frame
uint16_t bufferPointer = 4; // four bytes are for total byte counts
uint16_t maxbuffer = 2004; // 4 + 10 * 200 = 1004, that's 450 frames
unsigned long consecutiveDroppedFrames = 0;
unsigned long MAXDROPPEDFRAMES = 33; // at least 33 * 450 = 14850
    ↪ frames dropped
bool allowSend = true;

int Reset = 2;

IPAddress server(192, 168, 0, 125); // painlab server
WiFiClient client;

uint32_t reverseWordEndianness(uint32_t src) { // use -O3 will get rid
    ↪ of the call stack
  uint32_t dest = 0;
  asm ("REV␣%0,␣%1"
    : "=r" (dest)
    : "r" (src));

  return dest;
}


void connectToWifi() {
  // attempt to connect to Wi-Fi network:
  while (status != WL_CONNECTED) {
    Serial.print("Attempting␣to␣connect␣to␣network:␣");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network:
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(5000);
  }

  // you're connected now, so print out the data:
  Serial.println("You're␣connected␣to␣the␣network");
  Serial.println("----------------------------------------");
}

void sendDeviceDescriptor() {
  uint32_t dataLength = strlen(descriptor);
  uint32_t dataLengthBE = reverseWordEndianness(dataLength);
  client.write(reinterpret_cast<byte*>(&dataLengthBE), 4);
  size_t writtenLen = client.write(descriptor, dataLength); // assume
      ↪ utf-8 works
  allowSend = false;
  Serial.print("descriptor␣bytes␣sent:␣");
  Serial.println(writtenLen);
}

void connectToPainlabControlPanel() {
  while (1) {
    if (client.connect(server, 8124)) {
      // connect to pain lab control panel
      Serial.println("Connected␣to␣control␣panel");
      break;
    }
    Serial.println("Can't␣connect␣to␣pain␣lab␣control␣panel");
```

```
    delay(5000);
  }

  Serial.println("Sending␣descriptor");
  sendDeviceDescriptor();
}

void setup() {
  analogReadResolution(12); // padded to fit uint16_t

  // pull high for reset pin
  digitalWrite(Reset, HIGH);
  pinMode(Reset, OUTPUT);

  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  // while (!Serial);

  // set the LED as output
  pinMode(LED_BUILTIN, OUTPUT);

  connectToWifi();
  connectToPainlabControlPanel();
  Serial.println("setup␣completed");
}

#define COPY_AND_ADVANCE(data) \
  {                                                                 \
    memcpy(buffer + bufferPointer, reinterpret_cast<byte*>(&data),
        ↪ sizeof(data));   \
    bufferPointer += sizeof(data);\
  }

void readAndPushFrameIntoBuffer() {
  if (bufferPointer > maxbuffer) {
    Serial.print("Frame␣dropped,␣allowSend:␣");
    Serial.println(allowSend);
    bufferPointer = 4;

    consecutiveDroppedFrames++;
    if (consecutiveDroppedFrames > MAXDROPPEDFRAMES) {
      Serial.print("Resetting");
      digitalWrite(Reset, LOW);
    }
  }
  unsigned long ts = micros();
  // Remember to change ADC clock to at least DIV64
  uint16_t a0_reading = analogRead(A0); // GSR
  uint16_t a2_reading = analogRead(A2); // Heart Rate
  uint16_t a7_reading = analogRead(A7); // Myoware muscle sensor

  COPY_AND_ADVANCE(ts);
  COPY_AND_ADVANCE(a0_reading);
  COPY_AND_ADVANCE(a2_reading);
  COPY_AND_ADVANCE(a7_reading);
}

void checkOK() {
  //Serial.println("reading reply");
  uint8_t simplifiedReply = client.read();
  //Serial.println(simplifiedReply);
  if (simplifiedReply == 1) {
    allowSend = true;
  }
  else if (simplifiedReply != 255) {
    Serial.println(simplifiedReply);
  }
}
```

```
void loop() {
 if (!allowSend) {
    checkOK();
 }

 unsigned long currentMicrosInfo = micros();

 // check if the time after the last update is bigger the interval
 if (currentMicrosInfo - previousMicrosInfo >= collectInterval) {
   previousMicrosInfo = currentMicrosInfo;
   readAndPushFrameIntoBuffer();
 }

 //Serial.println(currentMicrosInfo - previousMicrosReport);
 if (allowSend && bufferPointer > 4) {
    if (currentMicrosInfo - previousMicrosReport >= reportInterval) {
      previousMicrosReport = currentMicrosInfo;

      *(reinterpret_cast<uint32_t*>(buffer)) = reverseWordEndianness(
         ↪ bufferPointer - 4); // write four BE bytes to buffer
         ↪ header

      client.write(buffer, bufferPointer); // ~3.5ms
      consecutiveDroppedFrames = 0;
      bufferPointer = 4;
      allowSend = false;

      // if the LED is off turn it on and vice-versa:
      if (ledState == LOW) {
        ledState = HIGH;
      } else {
        ledState = LOW;
      }

      // set the LED with the ledState of the variable:
      digitalWrite(LED_BUILTIN, ledState);
    }
 }

}
```

9.5.1.2 Wireless thermal stimulator control

```
#include <stdio.h>
#include <string.h>
#include <string>
#include <iostream>
#include <chrono>
#include <queue>

// OS headers
#include <fcntl.h> // Contains file controls like O_RDWR
#include <termios.h> // Contains POSIX terminal control definitions
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>

#include "TcsII_device_descriptor.h"

class SimplePainlabProtocol
{
```

```
    int _sockfd = 0;
    char _recvBuff[4096];
    int _readOffset = 0;
    char _writeBuff[4096];
public:
    std::queue<std::string> _server_responses;
    uint32_t reverseWordEndianness(uint32_t src) { // use -O3 will get
        ↪ rid of the call stack
        uint32_t dest = 0;
        asm ("REV %0, %1"
            : "=r" (dest)
            : "r" (src));

        return dest;
    }

    int GetServerData()
    {
        int n = read(_sockfd, _recvBuff + _readOffset, 4096 -
            ↪ _readOffset);
        if (n == 0) {
            return 1;
        }
        int total_read = n + _readOffset;
        if (total_read >= 4)
        {
            uint32_t total_bytes = reverseWordEndianness(*(
                ↪ reinterpret_cast<uint32_t*>(_recvBuff)));
            if (total_read >= total_bytes + 4)
            {
                _server_responses.push(std::string(_recvBuff + 4,
                    ↪ _recvBuff + 4 + total_bytes));
                // handling sticky packets
                int bytes_to_move = total_read - (total_bytes + 4);
                memmove(_recvBuff, _recvBuff + total_bytes + 4,
                    ↪ bytes_to_move);
                _readOffset = 0;
                return 0;
            }
            else
            {
                _readOffset = total_read;
                return GetServerData();
            }
        }
        else
        {
            _readOffset = total_read;
            return GetServerData();
        }
    }

    int Connect(const char *server_ip, int server_port=8124)
    {
        struct sockaddr_in serv_addr;

        memset(_recvBuff, '0',sizeof(_recvBuff));
        if((_sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
            printf("\n Error : Could not create socket \n");
            return 1;
        }

        memset(&serv_addr, '0', sizeof(serv_addr));

        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(server_port);

        if(inet_pton(AF_INET, server_ip, &serv_addr.sin_addr)<=0)
        {
```

```
            printf("\n␣inet_pton␣error␣occured\n");
            return 1;
        }

        if(connect(_sockfd, (struct sockaddr *)&serv_addr, sizeof(
            ↪ serv_addr)) < 0)
        {
            printf("\n␣Error␣:␣Connect␣Failed␣\n");
            return 1;
        }
        else
        {
            std::cout << "Connected␣to␣server␣at␣" << server_ip << ":"
                ↪ << server_port << std::endl;
        }

        std::cout << "sending␣descriptor..." << std::endl;

        uint32_t data_length = reverseWordEndianness(strlen(descriptor)
            ↪ );
        write(_sockfd, reinterpret_cast<char*>(&data_length), 4);
        int n = write(_sockfd, descriptor, strlen(descriptor));

        std::cout << "total␣" << n << "␣bytes␣descriptor␣sent␣to␣the␣
            ↪ server" << std::endl;

        GetServerData();

        std::cout << "Server␣responded:␣" << _server_responses.front()
            ↪ << std::endl;

        _server_responses.pop();

        return 0;
    }

    int SendData(const char* buff, int send_size)
    {
        uint32_t data_length = reverseWordEndianness((uint32_t)(
            ↪ send_size));
        write(_sockfd, reinterpret_cast<char*>(&data_length), 4);
        write(_sockfd, buff, send_size);

        return 0;
    }
};

struct DataFrame
{
    uint32_t timestamp;
    uint32_t temperature;
};

class TSAII_Serial_Connection
{
    int serial_port;
    const std::string silent = "F";
    const std::string get_temp = "E";
    std::chrono::system_clock::time_point start;
public:
    TSAII_Serial_Connection()
    {
        start = std::chrono::system_clock::now();
    }

    int Connect()
    {
        serial_port = open("/dev/ttyUSB0", O_RDWR);
        ...

        std::cout << "TSAII␣connected" << std::endl;
```

```
        return 0;
    }

    int WriteSerial(std::string line)
    {
        int n = write(serial_port, line.c_str(), line.size());
        std::cout << line << std::endl;
        return n;
    }

    int PopulatingPostStimulationTemperatureData(DataFrame* dfs)
    {
        char read_buf [1024];
        int dfs_count = 0;
        std::chrono::system_clock::time_point stimulation_start = std::
            ↪ chrono::system_clock::now();
        while (std::chrono::duration_cast<std::chrono::milliseconds>(
            ↪ std::chrono::system_clock::now() - stimulation_start).
            ↪ count() < 2000
                && dfs_count < 1000)
        {
            int write_bytes = write(serial_port, get_temp.c_str(),
                ↪ get_temp.size());
            if (write_bytes < 1)
            {
                std::cout << "Tcs␣device␣not␣responding:␣check␣serial␣
                    ↪ connection" << std::endl;
                return dfs_count;
            }
            int retry_count = 0;
            int num_bytes = read(serial_port, &read_buf, sizeof(
                ↪ read_buf));
            if (num_bytes < 24)
            {
                continue;
            }
            dfs[dfs_count].timestamp = std::chrono::duration_cast<std::
                ↪ chrono::milliseconds>(std::chrono::system_clock::now
                ↪ () - start).count();
            dfs[dfs_count].temperature = std::stoi(std::string(read_buf
                ↪  + 12, 3));

            dfs_count++;
        }

        return dfs_count;
    }
};

int main(int argc, char *argv[])
{
    SimplePainlabProtocol* protocol = new SimplePainlabProtocol();
    TSAII_Serial_Connection* tsa_serial = new TSAII_Serial_Connection()
        ↪ ;

    protocol->Connect(argv[1]);
    tsa_serial->Connect();

    int in_post_stimulation_data_collection = 0;
    DataFrame dfs[1000];
    memset(dfs, 0, sizeof(DataFrame) * 1000);

    while (1)
    {
        int ret = protocol->GetServerData();

        while (ret)
        {
            std::cout << "try␣to␣reconnect..." << std::endl;
```

```
            protocol->Connect(argv[1]);
            ret = protocol->GetServerData();
        }

        if (protocol->_server_responses.front() == "OK" || protocol->
            ↪ _server_responses.front() == "FAIL")
        {
            // should check OK before sending next post stimulation
                ↪ data batches, but that's fine for now.
        }
        else
        {
            if (in_post_stimulation_data_collection)
            {
                // can't apply command during post stimulation data
                    ↪ collection phase
                protocol->SendData("FAIL", 4);
            }
            else
            {
                // block further command before post stimulation data
                    ↪ collection complete
                if (protocol->_server_responses.front() == "L")
                    ↪ in_post_stimulation_data_collection = 1;

                int n = tsa_serial->WriteSerial(protocol->
                    ↪ _server_responses.front());
                if (n == protocol->_server_responses.front().size())
                {
                    protocol->SendData("OK", 2);
                }
                else
                {
                    protocol->SendData("FAIL", 4);
                }
                int collected_frames = 0;
                if (protocol->_server_responses.front() == "L")
                {
                    collected_frames = tsa_serial->
                        ↪ PopulatingPostStimulationTemperatureData(dfs
                        ↪ );
                    protocol->SendData(reinterpret_cast<char*>(dfs),
                        ↪ sizeof(DataFrame) * collected_frames);
                    // unblock new commands from the host
                    in_post_stimulation_data_collection = 0;
                }
            }
        }
        protocol->_server_responses.pop();
    }

    return 0;
}
```

9.5.1.3 Generic .NET implementation for device side application

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using System.Text;

public class PainlabProtocolException : Exception
{
    public PainlabProtocolException()
```

```
    {
    }

    public PainlabProtocolException(string message)
        : base(message)
    {
    }

    public PainlabProtocolException(string message, Exception inner)
        : base(message, inner)
    {
    }
}

[Serializable]
public class NetworkConfig
{
    public string host;
    public int port;
    public UInt32 maxFrameBuffer;
    public UInt32 bufferQueueSize;
}

public class PainlabProtocol
{
    private static Semaphore _sendSync = null;
    private TcpClient _connection = null;
    private Thread _listenerThread = null;
    private Thread _senderThread = null;

    protected UInt32 _bufferSize = 0;
    protected Mutex _queueBufferLock = null;
    protected Semaphore _queueBufferSem = null;
    protected Queue<UInt32> _queueNumBytes = null;
    protected Queue<byte[]> _queueBuffer = null;
    private UInt32 _sendNumBytes = 0;
    private byte[] _sendBuffer = null;
    private Mutex _sendLock = null;
    protected byte[] _listenBuffer = null;
    protected UInt32 _numControlBytes = 0;
    protected byte[] _controlBuffer = null;

    protected bool _waitOnControl = false;
    protected Semaphore _waitOnControlSem = null;

    protected virtual void DebugOutput(string msg)
    {
        Console.WriteLine(msg);
    }

    public void Init(NetworkConfig netConf, bool waitOnControl=false)
    {
        _bufferSize = netConf.maxFrameBuffer;
        _queueNumBytes = new Queue<UInt32>();
        _queueBuffer = new Queue<byte[]>();

        _sendBuffer = new byte[_bufferSize];
        _listenBuffer = new byte[_bufferSize];
        _controlBuffer = new byte[_bufferSize];

        _sendSync = new Semaphore(0, 1);
        _queueBufferLock = new Mutex();
        _queueBufferSem = new Semaphore(0, (int)netConf.bufferQueueSize
            ↪ );
        _sendLock = new Mutex();

        if (waitOnControl)
        {
            _waitOnControl = true;
            _waitOnControlSem = new Semaphore(0, 1);
        }
```

```
    try
    {
        _connection = new TcpClient(((NetworkConfig)netConf).host,
                                    ((NetworkConfig)netConf).port);
    }
    catch (SocketException socketException)
    {
        DebugOutput("Socket␣exception:␣" + socketException.Message)
            ↪ ;
        return;
    }

    _listenerThread = new Thread(new ThreadStart(ListenThread));
    _listenerThread.Start();

    _senderThread = new Thread(new ThreadStart(SenderThread));
    _senderThread.Start();
}

void ListenThread()
{
    try
    {
        byte[] listenBuffer = new byte[_bufferSize];
        try
        {
            NetworkStream stream = _connection.GetStream();
            while (true)
            {
                UInt32 totalBytes = 0;
                int length = 0;
                int offset = 0;
                while ((length = stream.Read(_listenBuffer, offset,
                    ↪  _listenBuffer.Length - offset)) != 0 ||
                    ↪ length > 0)
                {
                    offset += length;

                    if (totalBytes == 0)
                    {
                        // new transmission
                        if (offset < 4)
                        {
                            length = 0;
                            continue;
                        }

                        Array.Reverse(_listenBuffer, 0, 4);
                        totalBytes = BitConverter.ToUInt32(
                            ↪ _listenBuffer, 0);
                    }

                    if (offset < (int)totalBytes + 4)
                    {
                        length = 0;
                        continue;
                    }
                    else
                    {
                        int extraBytes = offset - (int)totalBytes -
                            ↪  4;
                        bool handled = false;

                        if (totalBytes < 5)
                        {
                            // check if it is a reply
                            string reply = Encoding.UTF8.GetString(
                                ↪ _listenBuffer, 4, (int)
                                ↪ totalBytes);
```

```
                                if (reply == "OK")
                                {
                                    // signal sender thread to continue
                                    _sendSync.Release();
                                    handled = true;
                                }
                                else if (reply == "FAIL")
                                {
                                    DebugOutput("Server side failure");
                                    handled = true;
                                }
                            }

                            if (!handled)
                            {
                                // going into this section must be
                                    ↪ control data
                                Array.Copy(_listenBuffer, 4,
                                    ↪ _controlBuffer, 0, (int)
                                    ↪ totalBytes);
                                _numControlBytes = (UInt32)totalBytes;

                                if (_waitOnControl)
                                {
                                    _waitOnControlSem.Release();
                                }
                            }

                            // handle sticky packets
                            if (extraBytes > 0)
                            {
                                // Copy to the beginning of listen
                                    ↪ buffer
                                Array.Copy(_listenBuffer, (int)
                                    ↪ totalBytes + 4, _listenBuffer,
                                    ↪ 0, extraBytes);
                                offset = extraBytes;
                                // Reset totalBytes so that it knows to
                                    ↪  decode length first
                                totalBytes = 0;
                                // Set length so the next loop can
                                    ↪ proceed processing the remaining
                                    ↪  part if not getting new data
                                length = extraBytes;
                            }
                            else
                            {
                                length = 0;
                            }

                            break;
                        }
                    }
                }
            }
            catch (SocketException socketException)
            {
                DebugOutput("Socket exception: " + socketException);
            }
        }
        catch (PainlabProtocolException e)
        {
            DebugOutput("Painlab exception: " + e.Message);
            return;
        }
    }

    protected void SendData(byte[] data, UInt32 l=0)
    {
```

```
        if (_connection == null)
        {
            return;
        }

        try
        {
            NetworkStream stream = _connection.GetStream();
            if (stream.CanWrite)
            {
                UInt32 dataLength = (UInt32)data.Length;
                if (Convert.ToBoolean(1))
                {
                    dataLength = 1;
                }
                byte[] dataLengthBE = BitConverter.GetBytes(dataLength)
                    ↪ ;
                Array.Reverse(dataLengthBE);

                // Use lock because the main thread will invoke send
                    ↪ reply data
                _sendLock.WaitOne();
                stream.Write(dataLengthBE, 0, 4);
                stream.Write(data, 0, (int)dataLength);
                _sendLock.ReleaseMutex();
            }
        }
        catch (SocketException socketException)
        {
            DebugOutput("Socket␣exception:␣" + socketException.Message)
                ↪ ;
            return;
        }
    }

    protected void SendString(string stringData)
    {
        byte[] stringBytes = Encoding.UTF8.GetBytes(stringData);
        SendData(stringBytes);
    }

    protected byte[] StringToBytes(string stringData)
    {
        return Encoding.UTF8.GetBytes(stringData);
    }

    void SenderThread()
    {
        DebugOutput("sending␣descriptor");
        RegisterWithDescriptor();
        // need to wait for an OK for any send data to proceed
        _sendSync.WaitOne();

        while (true)
        {
            // Wait data ready
            _queueBufferSem.WaitOne();

            // copy to send buffer guarded with lock
            _queueBufferLock.WaitOne();
            _sendNumBytes = _queueNumBytes.Peek();
            Array.Copy(_queueBuffer.Peek(), _sendBuffer, _sendNumBytes)
                ↪ ;

            _queueNumBytes.Dequeue();
            _queueBuffer.Dequeue();

            _queueBufferLock.ReleaseMutex();
            SendData(_sendBuffer, _sendNumBytes);
```

```
            // wait for listener OK signal
            _sendSync.WaitOne();
        }

        return;
    }

    protected virtual void RegisterWithDescriptor()
    {
        return;
    }

    protected virtual byte[] GetFrameBytes()
    {
        return null;
    }

    protected virtual void ApplyControlData()
    {
        return;
    }

    public void UpdateFrameData(byte[] givenBytes = null)
    {
        byte[] frameBytes = null;
        if (givenBytes != null)
        {
            frameBytes = givenBytes;
        }
        else
        {
            frameBytes = GetFrameBytes();
        }

        // Copy to frame buffer
        _queueBufferLock.WaitOne();
        _queueNumBytes.Enqueue((UInt32)frameBytes.Length);
        _queueBuffer.Enqueue(frameBytes);
        _queueBufferLock.ReleaseMutex();

        try
        {
            // Signal sender thread new data is ready
            _queueBufferSem.Release();
        }
        catch (SemaphoreFullException semEx)
        {
            DebugOutput("Frame sent to host dropped");
        }
    }

    public void HandlingControlData()
    {
        if (_numControlBytes > 0)
        {
            string result = "";
            try
            {
                ApplyControlData();
                result = "OK";
            }
            catch (Exception e)
            {
                DebugOutput(e.Message);
                result = "FAIL";
            }

            _numControlBytes = 0;
```

```
            SendString(result);
        }
    }

    public void Close()
    {
        _connection.Close();
    }
}
```

9.5.1.4 Control software protocol implementation

Protocol implementation: protocol.js:

```
const { ipcMain, Debugger } = require('electron');
const { dataHub } = require('./data.js');
const { env } = require('./env.js');

class ProtocolError extends Error {
    constructor(message) {
        super(message);
    }
}

var socketCount = 0;
var sockets = {};
var socketIDDeviceIDMap = {};
var deviceIDSocketIDMap = {};
var waitingReply = {};
var errorState = {};
var controlDataQueue = {};
var simplifiedReplySockets = {};

const Buffer = require('buffer').Buffer;

function makeDataHandler(dataHubObj) {

    // Data from renderer
    function sendControlData(socketID) {
        if (controlDataQueue[socketID].length > 0) {
            if (!waitingReply[socketID]) {
                let sendBuf = dataHub.prepareControlData(
                    ↪ socketIDDeviceIDMap[socketID], controlDataQueue[
                    ↪ socketID][0]);
                sendBuf.writeInt32BE(sendBuf.length - 4);

                waitingReply[socketID] = 1;
                sockets[socketID].write(sendBuf, () => {});
                controlDataQueue[socketID].shift();

                sendControlData(socketID);
            }
            else {
                // console.log("reply not received");
            }
        }
    }

    ipcMain.on('control-data', (event, arg) => {
        if (!(arg[0] in deviceIDSocketIDMap)) {
            if (arg[0] == 'log') {
                let logFrame = arg[1];
                logFrame['timestamp'] = Date.now();
                dataHub.controlData["log"].push(logFrame);
            } else {
```

```
            console.log("Command␣send␣to␣device␣" + arg[0] + "␣does
                ↪ ␣not␣exist!␣Did␣you␣put␣the␣wrong␣device␣id␣code
                ↪ ?");
        }
    } else {
        let socketID = deviceIDSocketIDMap[arg[0]];
        // push data to the queue
        controlDataQueue[socketID].push(arg[1]);
        // try to send
        sendControlData(socketID);
    }
});

// Data from listener
return { sender: sendControlData,
    listener: (socketID, bufObj) => {
    // already registered
    // console.log('data coming in');
    if (socketID in socketIDDeviceIDMap) {
        let ret = dataHubObj.saveFrame(socketIDDeviceIDMap[socketID
            ↪ ], bufObj);
        if (ret != 0)
        {
            return 1;
        }
    } else { // new connection, must be a JSON descriptor
        let descriptor = JSON.parse(bufObj.toString());
        if (!'device_id' in descriptor) {
            console.log("descriptor␣sent␣by␣" + socketID + "␣doesn'
                ↪ t␣have␣a␣device_id␣field");
            return 1;
        }
        socketIDDeviceIDMap[socketID] = descriptor['device_id'];
        deviceIDSocketIDMap[descriptor['device_id']] = socketID;
        if (env.winRef) {
            env.winRef.webContents.send('connected',
                ↪ socketIDDeviceIDMap[socketID]);
        }

        let ret = dataHubObj.registerDescriptor(descriptor);
        if (ret != 0)
        {
            return 1;
        }

        if (descriptor.data_to_control) {
            simplifiedReplySockets[socketID] = false;
        } else {
            simplifiedReplySockets[socketID] = true;
        }

        sockets[socketID].on('end', () => {
            env.winRef.webContents.send('disconnected',
                ↪ socketIDDeviceIDMap[socketID]);
            console.log('device␣' + socketIDDeviceIDMap[socketID] +
                ↪  '␣sent␣FIN');
        });

        sockets[socketID].on('close', () => {
            env.winRef.webContents.send('disconnected',
                ↪ socketIDDeviceIDMap[socketID]);
            console.log('connection␣to␣' + socketIDDeviceIDMap[
                ↪ socketID] + '␣closed');
        });
    }

    return 0;
}}
```

```
}

function makeListener(dataHandler, socket) {
    var numBytes = "NA";
    var numBytesReceived = 0;
    var chunks   = [];
    var socketID = socketCount++;
    sockets[socketID] = socket;
    waitingReply[socketID] = 0;
    errorState[socketID] = 0;
    controlDataQueue[socketID] = [];

    function processChunk (chunk) {
        let startPos = 0;
        let stickyChunk = null;
        let handled = false;

        // check if the socket is in error
        if (errorState[socketID]) {
            console.log("Socket␣Error␣on␣" + socketID + "␣" +
                ↪ errorState[socketID]);
            return;
        }

        // if the received chunk belongs to a new transmission
        if (numBytes == "NA") {
            if (chunk.length < 4) {
                console.log("Buffer␣size␣less␣then␣4␣while␣numBytes␣not
                    ↪ ␣initialized");
                return;
            }
            numBytes = chunk.readUIntBE(0, 4);
            startPos = 4;

            // console.log('Frame at socket ' + socketID + ' expected
                ↪ size: ' + numBytes);
        }

        if (numBytesReceived + chunk.length - startPos > numBytes) {
            let cut = numBytes - numBytesReceived;
            chunks.push(chunk.slice(startPos, startPos + cut));
            stickyChunk = chunk.slice(startPos + cut, chunk.length);
            numBytesReceived += cut;
        } else {
            if (startPos > 0) {
                chunks.push(chunk.slice(startPos));
            } else {
                chunks.push(chunk);
            }
            numBytesReceived += chunk.length - startPos;
        }

        if (numBytesReceived < numBytes) {
            return;
        } else if (numBytesReceived == numBytes) {
            // console.log("size matched: " + numBytesReceived);
            let ret = 0;
            let payloadBuf = (Buffer.concat(chunks));

            // reset closure variables
            numBytes = "NA";
            numBytesReceived = 0;
            chunks = [];

            if (waitingReply[socketID]) {
                // a reply is either OK or FAIL, check the size just to
                    ↪  avoid translating too large string
                if (payloadBuf.length < 5) {
                    let replyString = payloadBuf.toString('utf-8');
                    if (replyString == 'OK') {
```

```
                        waitingReply[socketID] = 0;
                        dataHandler.sender(socketID);
                        handled = true;
                    } else if (replyString == 'FAIL') {
                        waitingReply[socketID] = 0;
                        dataHandler.sender(socketID);
                        handled = true;
                        console.log("Device sent FAIL");
                    } // else continue
                }
            }

            if (!handled) {
                ret = dataHandler.listener(socketID, payloadBuf);

                if (simplifiedReplySockets[socketID]) {
                    let simpRep = new Uint8Array(1);
                    if (ret != 0)
                    {
                        simpRep[0] = 255;
                    } else {
                        simpRep[0] = 1;
                    }
                    sockets[socketID].write(simpRep, () => {}); // one
                        ↪ byte reply
                } else {
                    let message = "0000OK";
                    if (ret != 0)
                    {
                        message = "0000FAIL";
                    }
                    let messageBuf = Buffer.from(message, 'utf8');
                    messageBuf.writeInt32BE(messageBuf.length - 4);
                    sockets[socketID].write(messageBuf, () => {});
                }


            }

        } else {
            throw new ProtocolError("Num bytes: " + numBytes + ",
                ↪ actual recceived: " + numBytesReceived);
        }

        if (stickyChunk) {
            processChunk(stickyChunk);
        }

        return;
    }

    return processChunk;
}

var dataHandler = makeDataHandler(dataHub);

module.exports = {
    executeProtocol: function (socket) {
        socket.on('data', makeListener(dataHandler, socket));
    }
}
```

Data decoding and encoding: data.js:

```
const { app, Menu, dialog } = require('electron');
const isMac = process.platform === 'darwin'
const fs = require('fs-extra');
var JSONStream = require( "JSONStream" );
```

```
const { env } = require('./env.js');
const Buffer = require('buffer').Buffer;

function dumpJSONArray(fileName, arr) {
    ...
}

class DataHub {
    constructor() {
        // device identifier : descriptor sent in register phase
        this.deviceDescriptors = {}
        // device identifier : dataframe
        this.deviceData = {};
        // device_identifier : control data object
        this.controlData = { "log": [] } // a default log object
        // auto save
        this.timestamp = Date.now().toString()
        fs.mkdir('instance/autosave/' + this.timestamp)
        this.autosave_control = fs.openSync('instance/autosave/' + this
            ↪ .timestamp + '/control_data.autosave', 'w');
        this.autosave_device_data = fs.openSync('instance/autosave/' +
            ↪ this.timestamp + '/device_data.autosave', 'w');

        const template = [
            ...
        ];

        const menu = Menu.buildFromTemplate(template)
        Menu.setApplicationMenu(menu)
    }

    registerDescriptor(descriptor) {
        let deviceID = descriptor['device_id'];

        if (deviceID in this.deviceDescriptors) {
            // reconnect, just log that
            console.log("deviceID␣" + deviceID + "␣reconnected");
            if (env.winRef) {
                env.winRef.webContents.send('connected', deviceID);
            }
        } else {
            // init device data array
            this.deviceData[deviceID] = [];
            this.controlData[deviceID] = [];
            console.log("deviceID␣" + deviceID + "␣connected");
            // update render process
            if (env.winRef) {
                env.winRef.webContents.send('new-descriptor', [deviceID
                    ↪ , descriptor]);
            }
        }
        this.deviceDescriptors[deviceID] = descriptor;

        return 0;
    }

    saveFrame(deviceID, bufObj) {
        let frameFormat = this.deviceDescriptors[deviceID]['
            ↪ report_pack_order'];
        let binaryDecode = false;
        if (frameFormat) {
            if (frameFormat.order) {
                binaryDecode = true;
            }
        }

        if (binaryDecode) {
          if (frameFormat.packed) {
            let frameBatch = { timestamp: Date.now(), frames: [] };
```

```
    let frameOffset = 0;
    let dataFormat = this.deviceDescriptors[deviceID]['
        ↪ data_to_report'];
    while (frameOffset < bufObj.length) {
      let frameObj = {};
      frameFormat.order.forEach(item => {
        var decodedData;
        if (dataFormat[item] == 'uint32_t_le') {
          decodedData = bufObj.readUInt32LE(frameOffset);
          frameOffset += 4;
        } else if (dataFormat[item] == 'uint16_t_le') {
          decodedData = bufObj.readUInt16LE(frameOffset);
          frameOffset += 2;
        }
        frameObj[item] = decodedData;
      });
      frameBatch.frames.push(frameObj);
    }
    this.deviceData[deviceID].push(frameBatch);
    fs.write(this.autosave_device_data, "\"" + deviceID + "\":␣
        ↪ " + JSON.stringify(frameBatch) + "\n");

    // send frame batch to front end
    if (env.winRef) {
      if (frameBatch.frames.length > 0) {
        if (this.deviceDescriptors[deviceID]['visual_report']['
            ↪ expand_packed']) {
          env.winRef.webContents.send('new-dataframe', [
              ↪ deviceID, frameBatch.frames]);
        } else {
          let frameToFront = frameBatch.frames[frameBatch.
              ↪ frames.length - 1];
          frameToFront["timestamp"] = frameBatch.timestamp;
          env.winRef.webContents.send('new-dataframe', [
              ↪ deviceID, frameToFront]);
        }
      }
    }
  } else {
    throw new Error("Not␣implemented␣yet");
  }
} else {
    try
    {
        var frameObj = JSON.parse(bufObj.toString());
    }
    catch (error)
    {
        console.log(error);
        console.log(bufObj.toString());
    }
    frameObj['timestamp'] = Date.now();
    this.deviceData[deviceID].push(frameObj);
    fs.write(this.autosave_device_data, "\"" + deviceID + "\":␣
        ↪ " + JSON.stringify(frameObj) + "\n");

    // send the frame to front end
    if (env.winRef) {
      if (frameFormat && frameFormat.packed) {
        let frameToFront = {timestamp: frameObj.timestamp};
        for (const prop in frameObj) {
          if (prop != 'timestamp' && Array.isArray(frameObj[
              ↪ prop])) {
            frameToFront[prop] = frameObj[prop][frameObj[prop].
                ↪ length - 1];
          }
        }
        env.winRef.webContents.send('new-dataframe', [deviceID,
```

```
                    ↪  frameToFront]);
                }
                else {
                  env.winRef.webContents.send('new-dataframe', [deviceID,
                      ↪  frameObj]);
                }

            }
        }

        return 0;
    }

    prepareControlData(deviceID, obj) {
        let frameFormat = this.deviceDescriptors[deviceID]['
            ↪ control_pack_order'];
        let controlFrame = null;

        if (frameFormat) {
            if (frameFormat == "serial") {
                controlFrame = Buffer.from("abcd" + obj['serial'], '
                    ↪ utf8'); // "abcd" is the placeholder for 4 byte
                    ↪ BE length
            } else {
                throw new Error("not␣implemented␣yet.");
            }
        } else {
            controlFrame = Buffer.from("abcd" + JSON.stringify(obj), '
                ↪ utf8'); // "abcd" is the placeholder for 4 byte BE
                ↪ length
        }

        obj['timestamp'] = Date.now();
        this.controlData[deviceID].push(obj);
        fs.write(this.autosave_control, "\"" + deviceID + "\":␣" + JSON
            ↪ .stringify(obj) + "\n");
        return controlFrame;
    }

    writeLog(obj) {
      fs.write(this.autosave_control, "\"log\":␣" + JSON.stringify(obj)
          ↪  + "\n");
      this.controlData["log"].push(obj);
    }
}

var dataHub = new DataHub();
exports.dataHub = dataHub;
```

9.5.2 Data Analysis

9.5.2.1 Data Analysis Framework

core.experiment_data:

```
from typing import List, Callable, Type
from core.utils import a, a1

from core.individual_subject import make_individual_data
from core.individual_subject import get_trial_end_timestamps,
    ↪ get_immediate_value_before_timestamp
from datasets.manage import get_subject_names
from datasets.manage import set_expt as manage_set_expt

def set_expt(expt_name: str) -> None:
    manage_set_expt(expt_name)
```

```
def make_experiment_data(exclusive_participants: List[str] = [],
                         exclude_participants: List[str] = [],
                         exclude_device_data: List[str] = [],
                         lazy_closure: bool = False) -> dict:
    exp_data = {}
    for subject_name in get_subject_names():
        if subject_name in exclude_participants:
            continue
        if exclusive_participants != []:
            if subject_name not in exclusive_participants:
                continue
        if lazy_closure:
            exp_data[subject_name] = lambda subject_name:
                ↪ make_individual_data(subject_name,
                ↪ exclude_device_data)
        else:
            exp_data[subject_name] = make_individual_data(subject_name,
                ↪  exclude_device_data)

    return exp_data

def get_multiple_series(experiment_data: dict,
                        series_maker: Callable[[dict], List[a]],
                        subjects: List[str]) -> List[List[a]]:
    if subjects == []:
        return []
    else:
        return [series_maker(experiment_data[subjects[0]])] +
            ↪ get_multiple_series(experiment_data, series_maker,
            ↪ subjects[1:])

def get_multiple_series_lazy(experiment_data: dict,
                             series_maker: Callable[[dict], List[a]],
                             subjects: List[str]) -> List[List[a]]:
    if subjects == []:
        return []
    else:
        return [series_maker(experiment_data[subjects[0]](subjects[0]))
            ↪ ] + get_multiple_series_lazy(experiment_data,
            ↪ series_maker, subjects[1:])

def get_multiple_series_lazy_subject_indexed(experiment_data: dict,
                             series_maker: Callable[[dict], List[a]],
                             subjects: List[str]) -> List[List[a]]:
    if subjects == []:
        return []
    else:
        return [series_maker(experiment_data[subjects[0]](subjects[0]),
            ↪  subjects[0])] +
            ↪ get_multiple_series_lazy_subject_indexed(experiment_data
            ↪ , series_maker, subjects[1:])

from multiprocessing import Process, Queue

def queue_wrapper_subject_indexed(q, series_maker, subject_name, args):
    ret = series_maker(args, subject_name)
    q.put((subject_name, ret))

def queue_wrapper(q, series_maker, subject_name, args):
    ret = series_maker(args)
    q.put((subject_name, ret))

def submit_to_subprocess(res_queue: Type[Queue],
                         series_maker: Callable[[dict], List[a]],
                         individual_data: dict,
                         subject_name: str,
                         subject_index: bool) -> None:
    if subject_index:
```

```
            p = Process(target=queue_wrapper_subject_indexed, args=(
                ↪ res_queue, series_maker, subject_name, individual_data))
            p.start()
        else:
            p = Process(target=queue_wrapper, args=(res_queue, series_maker
                ↪ , subject_name, individual_data))
            p.start()

    def get_multiple_series_subprocessed_lazy(experiment_data: dict,
                                              series_maker: Callable[[dict],
                                                  ↪ List[a]],
                                              subjects: List[str],
                                              procs: int=8,
                                              subject_index: bool=False) -> List
                                                  ↪ [List[a]]:
        results = {}
        current_pending = 0
        res_queue = Queue()
        for subject in subjects:
            if current_pending >= procs:
                res = res_queue.get()
                current_pending -= 1
                results[res[0]] = res[1]

            submit_to_subprocess(res_queue, series_maker, experiment_data[
                ↪ subject](subject), subject, subject_index)
            current_pending += 1

        while len(subjects) != len(results):
            res = res_queue.get()
            results[res[0]] = res[1]

        return [results[s] for s in subjects]
```

core.individual_subject:

```
import numpy as np
import math
import statistics
import time
import scipy.spatial.transform as transform

import mne

from typing import List, Tuple, Callable, Union, Any

from datasets.manage import get_datapack_names, load_datapack
from core.utils import CONTROL_DATA_KEYWORD, DEVICE_DATA_KEYWORD,
    ↪ DEVICE_DESCRIPTOR_KEYWORD, DATA_NOTES_KEYWORD
from core.utils import UNITY_DEVICE_ID, ARDUINO_DEVICE_NAME,
    ↪ UNITY_DEVICE_NAME, LIVEAMP_DEVICE_NAME,
    ↪ LIVEAMP_DEVICE_NAME_BRAINVISION, EEGLAB_NAME
from core.utils import verbose_out
from core.utils import butter_bandpass_filter, butter_lowpass_filter,
    ↪ butter_highpass_filter, median_filter, construct_crf_sequence

def brainvision_eeg_dataloader(datapack_name):
    raw = mne.io.read_raw_brainvision(datapack_name, preload=True)
    return raw

def make_individual_data(subject_name: str, exclude_device_data: List[
    ↪ str] = []) -> dict:
    individual_data = {}
    individual_data['name'] = subject_name
    for datapack_name in get_datapack_names(subject_name):
        if CONTROL_DATA_KEYWORD in datapack_name:
            individual_data['control_data'] = load_datapack(
                ↪ datapack_name)
        elif DEVICE_DESCRIPTOR_KEYWORD in datapack_name:
            individual_data['device_descriptor'] = load_datapack(
```

```
                ↪ datapack_name)
        elif DATA_NOTES_KEYWORD in datapack_name:
            individual_data['data_notes'] = load_datapack(datapack_name
                ↪ )
        elif DEVICE_DATA_KEYWORD in datapack_name:
            # remove last '__.json'
            device_name = datapack_name[datapack_name.find(
                ↪ DEVICE_DATA_KEYWORD) + len(DEVICE_DATA_KEYWORD):-7]
            if device_name not in exclude_device_data:
                verbose_out("Loading␣datapack:␣" + datapack_name + "␣
                    ↪ for␣device:␣" + device_name)
                individual_data[device_name] = load_datapack(
                    ↪ datapack_name)
        elif LIVEAMP_DEVICE_NAME_BRAINVISION in datapack_name and
            ↪ LIVEAMP_DEVICE_NAME_BRAINVISION not in
            ↪ exclude_device_data:
            individual_data['eeg'] = brainvision_eeg_dataloader(
                ↪ datapack_name)
            individual_data['eeg'].set_montage(mne.channels.
                ↪ make_standard_montage('brainproducts-RNP-BA-128'),
                ↪ on_missing='ignore')
        elif EEGLAB_NAME in datapack_name and EEGLAB_NAME not in
            ↪ exclude_device_data:
            individual_data['eeg_clean'] = mne.io.read_raw_eeglab(
                ↪ datapack_name, preload=True)
            individual_data['eeg_clean'].set_montage(mne.channels.
                ↪ make_standard_montage('brainproducts-RNP-BA-128'),
                ↪ on_missing='ignore')

    return individual_data
```

datasets.manage:

```
import os
import json

from typing import List

DATASETS_DIR = 'datasets'
EXPT_DIR = 'Expt1'
EXCLUDE_DIR = ['__pycache__']

def get_subject_names() -> List[str]:
    DATA_DIR = os.path.join(DATASETS_DIR, EXPT_DIR)
    print(DATA_DIR)
    return list(filter(lambda x: os.path.isdir(os.path.join(DATA_DIR, x
        ↪ )) and x not in EXCLUDE_DIR, os.listdir(DATA_DIR)))

def get_datapack_names(subject_name) -> List[str]:
    subject_dir = os.path.join(DATASETS_DIR, os.path.join(EXPT_DIR,
        ↪ subject_name))
    return list(map(lambda x: os.path.join(subject_dir, x), [x for x in
        ↪  os.listdir(subject_dir) if not x.startswith('.')]))

def load_datapack(datapack_name: str) -> dict:
    with open(datapack_name, 'r', encoding='utf-8') as f:
        return json.load(f)

def set_expt(expt_name: str) -> None:
    global EXPT_DIR
    EXPT_DIR = expt_name
```

9.5.2.2 CUDA accelerated grid search

```
typedef std::unordered_map<std::string,
                    std::vector<
                         std::tuple<float, // pain intensity
```

```
                                        std::string, // pain condition name
                                        uint32_t, // length of pickups
                                        uint32_t*, // pickup choices
                                        uint32_t, // total frame count
                                        uint32_t*>>*> FrameBlocksBySubjects
                                            ↪ ;

#ifdef CUDA_ENABLED
__host__ __device__
#endif
void getParameters(uint64_t i, float* free_parameter, uint32_t*
    ↪ num_ticks, float** parameter_ticks)
{
    uint64_t remainder = i;
    for (int k = 0; k < model::NUM_FREE_PARAMETER; k++)
    {
        uint64_t nx = num_ticks[model::NUM_FREE_PARAMETER - k - 1];
        free_parameter[model::NUM_FREE_PARAMETER - k - 1] =
            ↪ parameter_ticks[model::NUM_FREE_PARAMETER - k - 1][
            ↪ remainder % nx];
        remainder = (remainder - remainder % nx) / nx;
    }
    assert(remainder == 0);
    return;
}

#ifdef CUDA_ENABLED
__global__
#endif
void singleRun(uint64_t i, float& best_acc, float* best_parameters,
    ↪ uint64_t grid_size, uint32_t blocks,
                uint32_t* num_ticks, float** parameter_ticks, float*
                    ↪ pain_intensity,
                uint32_t* num_choices, uint32_t** pickup_choices,
                    ↪ uint32_t* num_frames, uint32_t** frames,
                float* result_grid)
{
    float acc_sum = 0;

#ifdef CUDA_ENABLED
    uint64_t parameter_index = blockDim.x * blockIdx.x + threadIdx.x +
        ↪ i;
    if (parameter_index >= grid_size)
    {
        return;
    }
#endif
    float free_parameter[model::NUM_FREE_PARAMETER];
#ifdef CUDA_ENABLED
    getParameters(parameter_index, free_parameter, num_ticks,
        ↪ parameter_ticks);
#else
    getParameters(i, free_parameter, num_ticks, parameter_ticks);
#endif
    for (int j = 0; static_cast<uint32_t>(j) < blocks; j++)
    {
        float block_acc = model::blockSimulation(free_parameter,
            ↪ num_frames[j], frames[j],
             pain_intensity[j], num_choices[j], pickup_choices[j]);
        acc_sum += block_acc;
    #ifdef DEBUG_SINGLE_TEST
        std::cout << pain_intensity[j] << "=======␣ACC:" << block_acc
            ↪ << std::endl;
    #endif
    }
    acc_sum /= blocks;

#ifdef CUDA_ENABLED
    result_grid[blockDim.x * blockIdx.x + threadIdx.x] = acc_sum;
#else
#ifndef DEBUG
```

```
    #pragma omp critical
#endif
    {
        if (acc_sum < best_acc)
        {
            best_acc = acc_sum;
            memcpy(best_parameters, free_parameter, model::
                ↪ NUM_FREE_PARAMETER * sizeof(float));

        #ifdef DEBUG
            for (int i = 0; i < model::NUM_FREE_PARAMETER; i++)
            {
                std::cout << best_parameters[i] << "␣␣␣␣";
            }
            std::cout << std::endl;
        #endif
        }

    #ifndef DEBUG_SINGLE_TEST
        if (i % (grid_size / 1000) == 0)
        {
            std::cout<<i / (float)grid_size * 100 << "\r";
        }
    #endif
    }
#endif
}

void gridSearch(uint64_t grid_size, uint32_t blocks, float*
    ↪ fitted_result, uint32_t* num_ticks, float** parameter_ticks,
    ↪ float* pain_intensity,
                uint32_t* num_choices, uint32_t** pickup_choices,
                    ↪ uint32_t* num_frames, uint32_t** frames)
{
    float best_acc = 1;
    float best_parameters[model::NUM_FREE_PARAMETER];

#if !(DEBUG_SINGLE_TEST || TEST_AGAINST_PYTHON)
    // too small grid size causes division by zero when not enable
        ↪ DEBUG_SINGLE_TEST
    assert(grid_size >= 1000);
#endif

#ifdef CUDA_ENABLED
    cudaError_t ret = cudaSuccess;

    uint32_t* devptr_num_ticks = nullptr;
    CUDA_DEVICE_ALLOC_AND_COPY(devptr_num_ticks, num_ticks, model::
        ↪ NUM_FREE_PARAMETER, sizeof(uint32_t), ret)
    float** devptr_parameter_ticks = nullptr;
    float** host_ptr_to_devptr_parameter_ticks = nullptr;
    CUDA_DEVICE_ALLOC_AND_COPY_NESTED_ARRAY(float,
        ↪ devptr_parameter_ticks, host_ptr_to_devptr_parameter_ticks,
        ↪ parameter_ticks, model::NUM_FREE_PARAMETER, sizeof(float*),
        ↪ num_ticks, sizeof(float), ret)

    float* devptr_pain_intensity = nullptr;
    CUDA_DEVICE_ALLOC_AND_COPY(devptr_pain_intensity, pain_intensity,
        ↪ blocks, sizeof(float), ret)

    uint32_t* devptr_num_choices = nullptr;
    CUDA_DEVICE_ALLOC_AND_COPY(devptr_num_choices, num_choices, blocks,
        ↪  sizeof(uint32_t), ret)
    uint32_t** devptr_pickup_choices = nullptr;
    uint32_t** host_ptr_to_devptr_pickup_choices = nullptr;
    CUDA_DEVICE_ALLOC_AND_COPY_NESTED_ARRAY(uint32_t,
        ↪ devptr_pickup_choices, host_ptr_to_devptr_pickup_choices,
        ↪ pickup_choices, blocks, sizeof(uint32_t*), num_choices,
        ↪ sizeof(uint32_t), ret)

    uint32_t* devptr_num_frames = nullptr;
```

```
    CUDA_DEVICE_ALLOC_AND_COPY(devptr_num_frames, num_frames, blocks,
        ↪ sizeof(uint32_t), ret)
    uint32_t** devptr_frames = nullptr;
    uint32_t frame_size = sizeof(uint32_t) * frame_defs::FRAME_SIZE;
    uint32_t** host_ptr_to_devptr_frames = nullptr;
    CUDA_DEVICE_ALLOC_AND_COPY_NESTED_ARRAY(uint32_t, devptr_frames,
        ↪ host_ptr_to_devptr_frames, frames, blocks, sizeof(uint32_t*)
        ↪ , num_frames, frame_size, ret)

    float* devptr_result_grid = nullptr;
    ret = cudaMalloc(&devptr_result_grid, MAX_LAUNCH_SIZE * sizeof(
        ↪ float));
    CUDA_CHECKS(ret)

    uint64_t current_count = 0;
    float* result_grid = new float[MAX_LAUNCH_SIZE];
    uint64_t argmin = 0;
    while (current_count < grid_size)
    {
        uint64_t launched_kernels = grid_size - current_count >
            ↪ MAX_LAUNCH_SIZE ? MAX_LAUNCH_SIZE : grid_size -
            ↪ current_count;
        singleRun<<<ceil((double)launched_kernels / (double)
            ↪ THREADSPERBLOCK), THREADSPERBLOCK, 0, 0>>>(current_count
            ↪ , best_acc, nullptr, grid_size, blocks,
        devptr_num_ticks, devptr_parameter_ticks, devptr_pain_intensity
            ↪ ,
        devptr_num_choices, devptr_pickup_choices,
        devptr_num_frames, devptr_frames, devptr_result_grid);
        ret = cudaDeviceSynchronize();
        CUDA_CHECKS(ret)

        ret = cudaMemcpy(result_grid, devptr_result_grid,
            ↪ launched_kernels * sizeof(float), cudaMemcpyDeviceToHost
            ↪ );
        CUDA_CHECKS(ret)

        for (int i = 0; i < launched_kernels; i++)
        {
            if (result_grid[i] < best_acc)
            {
                argmin = i + current_count;
                best_acc = result_grid[i];
            }
        }
        current_count += launched_kernels;
        std::cout<<current_count / (double)grid_size * 100 << "\r";
    }
    getParameters(argmin, best_parameters, num_ticks, parameter_ticks);

    CUDA_FREE_WITH_CHECK(devptr_result_grid, ret);
#else
#ifndef DEBUG
    #pragma omp parallel for schedule(dynamic)
#endif
    for (uint64_t i = 0; i < grid_size; i++)
    {
        singleRun(i, best_acc, best_parameters, grid_size, blocks,
            ↪ num_ticks, parameter_ticks, pain_intensity, num_choices,
            ↪  pickup_choices, num_frames, frames, nullptr);
    }
#endif

#ifdef CUDA_ENABLED

    CUDA_FREE_WITH_CHECK_NESTED_ARRAY(devptr_parameter_ticks,
        ↪ host_ptr_to_devptr_parameter_ticks, model::
        ↪ NUM_FREE_PARAMETER, ret);
    CUDA_FREE_WITH_CHECK(devptr_num_ticks, ret);
    CUDA_FREE_WITH_CHECK(devptr_pain_intensity, ret);
```

```
    CUDA_FREE_WITH_CHECK_NESTED_ARRAY(devptr_pickup_choices,
        ↪ host_ptr_to_devptr_pickup_choices, blocks, ret)
    CUDA_FREE_WITH_CHECK(devptr_num_choices, ret);
    CUDA_FREE_WITH_CHECK_NESTED_ARRAY(devptr_frames,
        ↪ host_ptr_to_devptr_frames, blocks, ret)
    CUDA_FREE_WITH_CHECK(devptr_num_frames, ret);
#endif
    std::cout<< best_acc << std::endl;
    fitted_result[model::NUM_FREE_PARAMETER] = best_acc;
    memcpy(fitted_result, best_parameters, model::NUM_FREE_PARAMETER *
        ↪ sizeof(float));
    for (int i = 0; i < model::NUM_FREE_PARAMETER; i++)
    {
        std::cout << fitted_result[i] << "␣␣␣␣";
    }
    std::cout << std::endl;
}

uint64_t generateGrid(uint32_t*& num_ticks, float**& parameter_ticks,
                  std::vector<int> slice_type, std::vector<std::vector<
                      ↪ float>> slice)
{
    assert(slice_type.size() == slice.size());
    assert(slice_type.size() == model::NUM_FREE_PARAMETER);

    uint64_t total_grid_size = 1;
    parameter_ticks = new float*[model::NUM_FREE_PARAMETER];
    num_ticks = new uint32_t[model::NUM_FREE_PARAMETER];
    for (int i = 0; i < model::NUM_FREE_PARAMETER; i++)
    {
        if (slice_type[i] == PLAIN_TICKS)
        {
            total_grid_size *= slice[i].size();
            num_ticks[i] = slice[i].size();
            parameter_ticks[i] = new float[slice[i].size()];
            std::copy(slice[i].begin(), slice[i].end(), parameter_ticks
                ↪ [i]);
        }
        else if (slice_type[i] == SLICED_TICKS)
        {
            int slice_size = round((slice[i][1] - slice[i][0]) / slice[
                ↪ i][2]) + 1;
            total_grid_size *= slice_size;
            num_ticks[i] = slice_size;
            parameter_ticks[i] = new float[slice_size];
            for (int j = 0; j < slice_size; j++)
            {
                parameter_ticks[i][j] = slice[i][0] + j * slice[i][2];
            }
        }
        else
        {
            throw std::invalid_argument("No␣such␣slice␣type");
        }
    }

    std::cout << "total␣grid␣size:␣" << total_grid_size << std::endl;

    return total_grid_size;
}
```