Ambient Sensor Fusion for Virtual Reality Systems

Author:
Jake Alexander Fountain
B. Mathematics / B. Science (Physics) / B. Computer Science (Hons),
The University of Newcastle

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the
School of Electrical Engineering and Computing
The University of Newcastle, Australia

Submitted October 2018

This research was supported by an Australian Government Research Training Program (RTP) Scholarship
Declaration of Authorship

I hereby certify that the work embodied in the thesis is my own work, conducted under normal supervision. The thesis contains no material which has been accepted, or is being examined, for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made. I give consent to the final version of my thesis being made available worldwide when deposited in the University’s Digital Repository, subject to the provisions of the Copyright Act 1968 and any approved embargo.

Signed: Jake Fountain

Date: 31/10/2018
Contribution to joint publications

I hereby certify that the work embodied in this thesis contains published papers of which I am a joint author. I have included as part of the thesis a written declaration endorsed in writing by my supervisor, attesting to my contribution to the joint publication.

By signing below I confirm that Jake Fountain contributed the majority of the work for the following publications. This included motivating the work, developing the algorithms and performing the research and writing the papers. I, Shamus P. Smith, provided academic advice and writing assistance.


Signed: 

Date: 24/9/2018

Shamus P. Smith
Acknowledgements

Firstly, I want to acknowledge the support of some people in my personal life. To Melinda Duncan, thank you for being my best friend for nearly a decade. You have supported me enormously during the process of realising this work. To Anneliese Wild, thank you for making it easier to keep working even when things were at their hardest. And to my Father, Ian Fountain, thank you for always pushing me out of my comfort zone and teaching me the value of adventure.

Secondly, I would like to acknowledge the support in the form of equipment, office space and technical advice provided by the Newcastle Robotics Laboratory and the NUbots RoboCup team. In particular, I would like to acknowledge Trent Houliston for the numerous enjoyable and insightful conversations which informed my work.

I am also grateful to the University of Newcastle’s academic staff, particularly my supervisors Dr Shamus P. Smith and Associate Professor Stephan Chalup for their guidance and assistance over the course of my time at The University of Newcastle.

Lastly, this PhD research would not have been possible without the financial support I have received through the Australian Government Research Training Program Scholarship (RTP) and the University of Newcastle’s research allowances.

I am grateful for these opportunities.
This thesis is dedicated to my mother, Jennifer Fountain, who gave me the confidence and tools to carve my own path.
Modern Virtual Reality (VR) systems rely on precise measurements of the real world to realistically present a virtual environment. Range, capability and accuracy are key tracking properties which determine the realism and applicability of a VR system. Integration of two or more sensor systems can enhance these properties compared to each individual component system. To support a rich ecosystem of diverse tracking devices for all levels of user competency, algorithms for ambient sensor fusion are required - algorithms which do not require user intervention or knowledge. This thesis investigates each of the following three steps required for ambient sensor fusion. First, Correlation involves identification of sensor dependencies and temporal relationships. Second, Calibration aligns sensor measurement domains. Finally, Fusion combines sensor data to extract an underlying statistical model.

To address the correlation step, an algorithm for identifying rigid links between position and rotation sensors from different systems was developed and tested with several real world systems. Additionally, a novel model-less approach to determining latency between two sensor systems has shown promising results for use in comparison of arbitrary dependent signals.
To address the second sensor fusion step, an ambient calibration technique has been developed which determines the relationship between two systems solely from self-directed user movement. The algorithm was applied to the Microsoft Kinect v2 and popular VR systems to enable body tracking in VR with commodity devices and minimal user setup.

To address the final sensor fusion step, a modular multi-modal skeleton fusion algorithm was developed. The algorithm employs a novel constrained articulated Kalman filter to combine skeletal tracking results with high modularity in real time. To test the fusion system, the optical Leap Motion hand tracking system was fused with the inertial Perception Neuron hand tracking system. A user study was performed (n=18) and results suggested that the proposed system succeeds in generalising the component tracking systems to perform well in a wider variety of scenarios.

Finally, the research software has been made available in an open source C++ plugin called ‘Spooky’. Spooky currently supports Unreal Engine 4. Future work will be focused on improving the reliability and usability of the Spooky framework, while extending the techniques developed for each of the fusion steps to broader applications.
Contents

Declaration of Authorship i

Acknowledgements iii

Abstract v

Contents vii

List of Figures xi

List of Tables xix

Thesis Highlights xxi

1 Introduction and Motivation 1
  1.1 Virtual Reality and Tracking Systems 4
  1.2 Device Fragmentation and the Fusion Middleware 6
  1.3 Fusion Middleware Features 10
    1.3.1 Hardware-Software Abstraction 10
    1.3.2 Modularity and Sensor Fusion 12
    1.3.3 User Configuration and Representation 14
  1.4 Research Question and Contributions 16

2 Literature Review 21
  2.1 Sensor Calibration and the Need for Sensor Correlation 22
    2.1.1 Automatic and Manual Calibration 22
    2.1.2 Representing 3D Rigid Bodies 24
    2.1.3 6DoF Sensor Calibration 32
    2.1.4 Sensor Identification 34
  2.2 Sensor Fusion 37
    2.2.1 Formal Definition of Sensor Fusion 37
2.2.2 Bayesian Filtering ........................................ 39
2.2.3 Skeleton Fusion ........................................... 42
2.3 Existing VR Middleware Software ...................... 45

3 Sensor Identification ........................................ 50
  3.1 Matching of Sensors ........................................ 52
    3.1.1 Invariant Functional (IF) Method .................. 52
    3.1.2 Calibration Error (CE) Method .................... 55
    3.1.3 Match Selection and Score Filtering ............... 56
  3.2 Simulation Analysis ...................................... 57
    3.2.1 Notation for Randomly Generated Transforms .. 57
    3.2.2 Procedure ........................................... 58
    3.2.3 Results ............................................. 59
  3.3 Case Study: Offline Experiment With Real Sensors .... 62
  3.4 Case Study: Interactive Demonstration with Real Sensors .... 66
  3.5 Reflection and Future Work ............................. 70

4 Latency Calibration ........................................ 73
  4.1 Related Work ............................................ 74
  4.2 Model-less Latency Between Two Multi Dimensional Signals .... 75
  4.3 Application to Simulation ................................ 78
    4.3.1 Generating Simulation Data ....................... 78
    4.3.2 Simulation Results ................................ 80
  4.4 Application to Video Data ............................... 84
  4.5 Reflection and Future Work ............................. 89

5 Ambient Sensor Calibration ............................... 94
  5.1 Related Work ............................................ 95
  5.2 Implementation of Ambient Calibration ................ 98
    5.2.1 The Calibrator .................................... 99
    5.2.2 Calibration State Machine ....................... 101
    5.2.3 Fault Detection .................................... 104
  5.3 Evaluation ................................................ 105
    5.3.1 Method .............................................. 105
    5.3.2 Results ............................................. 108
    5.3.3 Discussion ........................................... 109
  5.4 Reflection and Future Work ............................. 113

6 Skeleton Sensor Fusion ................................ 115
  6.1 The Fusion Graph ....................................... 116
  6.2 Fusion Algorithm ....................................... 120
    6.2.1 Recursive Fusion .................................. 121
Contents

6.2.2 Prediction Update ................................................. 125
6.2.3 Constrained Measurement Update ......................... 127
6.2.4 Computing the Jacobian ........................................ 131
6.3 Evaluation .............................................................. 133
6.4 Reflection and Future Work ...................................... 134
   6.4.1 Confidence Modelling ....................................... 134
   6.4.2 Advanced Fusion Chain Selection ......................... 135
   6.4.3 Managing Off-Diagonal Covariance ...................... 136

7 Spooky: A Sensor Fusion Plugin .......................... 137
   7.1 Software Implementation .................................... 137
   7.2 Spooky UE4 Features ........................................... 139
      7.2.1 Spooky Fusion Plant .................................... 140
      7.2.2 Measurements ............................................ 141
      7.2.3 Spooky Skeletons ......................................... 141
      7.2.4 Spooky Spirits ........................................... 143
      7.2.5 Spooky Graveyards ...................................... 144
   7.3 Reflection and Future Work .................................. 145

8 User Study ................................................................. 146
   8.1 Method .............................................................. 147
      8.1.1 Tracking Setup ............................................ 148
      8.1.2 Grip Detection ............................................ 150
      8.1.3 Leap Motion Confidence Model ......................... 151
      8.1.4 Tasks ....................................................... 156
      8.1.5 Experiment ................................................. 159
   8.2 Results .............................................................. 164
      8.2.1 Objective Performance Metrics ......................... 166
      8.2.2 Throwing Task ............................................ 169
      8.2.3 Subjective Performance Metrics ....................... 172
      8.2.4 Verbal Responses ......................................... 174
      8.2.5 Summary of Results ..................................... 175
   8.3 Discussion and Conclusion .................................... 177
      8.3.1 Explanation of Participant Performance ............... 179
      8.3.2 Ground Truth Accuracy .................................. 181
      8.3.3 Reflection on Proposed Fusion Algorithm .............. 186
      8.3.4 Reflection and Future Work ................................ 187

9 Conclusion ................................................................. 189
   9.1 Thesis Summary ................................................... 189
   9.2 Future Work ....................................................... 191
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A  User Study Documents</td>
<td>194</td>
</tr>
<tr>
<td>Bibliography</td>
<td>206</td>
</tr>
</tbody>
</table>
List of Figures

1.1 An example software interface model for development of a virtual reality application. A developer must target each possible input and output device software development kit (SDK). Additional work is required to support alternative input systems; here Input Device 3 and Output Device 3 cannot be used because the developer did not include support. Input/Output Device 2 is compatible with the system, but not compatible with Input/Output Device 1 simultaneously. The majority of included devices are not used by a given user. 

1.2 The software structure of a standard middleware. Input and output devices are abstracted from the developer of the virtual environment. This allows support for a wide variety of input and output devices. However, many devices will not be compatible due to overlapping device domains, unless the developer explicitly handles each combination. For example, here the user can choose between Input 1 or Input 2, but not both.

1.3 Sensor matching and alignment concept.

1.4 The proposed sensor fusion middleware acts like a standard middleware, but also allows for the user to combine systems with overlapping domains. Overlapping sensors are fused, reducing noise and increasing tracking range.

1.5 The ambient sensor fusion pipeline visualised with a 2D toy example. Each pair of arrows represents a sensor coordinate frame. The grey trails represent recent data from the sensor, such as a position curve. The ellipses represent the rigid bodies which the sensors are attached to in the common coordinate frame.

1.6 Thesis map.

2.1 Spectrum of common mathematical representations of 3D rotation. Note that although so(3) is technically embedded in $\mathbb{R}^{3 \times 3}$ it can be parameterized by $\omega \in \mathbb{R}^3$ and is effectively dense.
2.2 The relationship between two 6DoF sensors has characteristic equation given by Equation 2.25. $S$ and $Q$ are the sensor reference frames. $S_t$ and $Q_t$ are the coordinate systems corresponding to the 6DoF pose of each sensor at time step $t$. $Y$ and $X$ are the rigid transforms linking the sensor reference frames and the sensor poses respectively. $A_t$ and $B_t$ are the 6DoF pose transforms of the two sensors at time step $t$.

3.1 Sensor matching and alignment concept. ........................................ 51
3.2 The relationship between two 6DoF sensors has characteristic equation given by Equation 3.1. $S$ and $Q$ are the sensor reference frames. $S_t$ and $Q_t$ are the coordinate systems corresponding to the 6DoF pose of each sensor at time step $t$. $Y$ and $X$ are the rigid transforms linking the sensor reference frames and the sensor poses respectively. $A_t$ and $B_t$ are the 6DoF pose transforms of the two sensors at time step $t$. 53
3.3 Invariant Functional (IF) Method - in this hypothetical example, $A$ and $B$ are correlated, while $C$ is not. ................................. 54
3.4 Calibration Error (CE) Method .................................................. 56
3.5 Simulation sample-noise and sample-timing analysis for the calibration error (CE) method. The heat map displays the fraction of correct identifications of the simulated sensors as a function of angular noise and number of samples averaged over the trial period (left). The sample-timing linear least squares relationship was $\tau_m(N_s) \approx 0.0018 \times N_s + 0.033\text{ms}$ (right). ................................. 60
3.6 Simulated noise performance analysis of the calibration error (CE) and invariant functional (IF) matching algorithms. The mean matching performance is equal to the fraction of correct identifications of sensor dependencies over a 52 second simulation. Note that since the IF method only compares rotation data, it is expected to perform well independent of positional noise. ................................. 61
3.7 Data visualisation for the offline real sensor case study. Left shows the OptiTrack marker coordinate frames, transformed into the Kinect reference frame using ground truth (see Figure 3.8), and the detected user. A rigid body skeleton is generated with the OpenNI software platform version 1.5.7 (centre). The candidate Kinect rigid bodies have their corresponding coordinate frames drawn at their origins. The two 6DoF measurements from the sensors are matched with the one of six Kinect rigid body streams in real time (right). At this instance, the left knee (red) has been correctly identified, while the right shoulder has been misidentified as the left shoulder (green). 63
3.8 The motion capture setup for measuring the Kinect reference frame for ground truth verification (left) and an example sensor placed on the shoulder and tracked by an OptiTrack system (right). 64
3.9 The OpenNI Kinect skeleton tracking software has several instabilities which make it unsuitable for rigid body identification. For example, the knee orientation is ambiguous when outstretched; here the rigid body for the knee flips 180° discontinuously. ................................. 66

3.10 Tracking results for an OptiTrack marker (left) and a PS Move controller (right) from the perspective of the PS Eye camera. Note that the OptiTrack marker has been transformed to the PS Move sensor space for visualisation only; all computations use raw data in the native space. ................................. 67

3.11 Two markers (top) were matched with a PS Move controller. Moment of attachment (bottom left) and moment of identification of the rigid link (bottom right; indicated by the white sphere) for the CE method. Typical time between the two events is 1-2 seconds, depending on the movement of the user (Figure 3.12). The amount of movement required to match here can be seen to be a 90° rotation and a translation of about 15cm. The IF method achieved similar results. ... 68

3.12 An example trace where two motion capture markers are attached sequentially, one at a time, to the PS Move sensor. Here it is assumed that when the ground truth distance between a marker and the PS Move (top) is less than 30cm, the two are rigidly attached. The matching result for the CE method (bottom) can be seen to match well with the ground truth estimator (centre). The broken and dotted lines indicate the times at which a sensor is attached and removed respectively. .................................................... 69

3.13 An example of real-time interactive sensor matching for the Microsoft Kinect. The user is tracked with OpenNI and NiTE software libraries and the sensor’s true location (left shoulder) is drawn as 3D coordinate axes (left). Hypotheses are shown as small white spheres placed at the origin of each articulated body part, with the single larger sphere representing the most likely hypothesis (centre). After approximately 10 seconds of tracking, the correct location is determined and most other hypotheses are eliminated (right). ............................................. 70

4.1 Visualisation of the proposed latency calibration method. The approach searches for values of latency \( l \) such that close points in \( x \) correspond to close points in \( y \). The algorithm uses the largest disagreement over all close points in \( x \) as the objective function. ............. 76

4.2 Example of latency calibration applied to one dimensional signals \( x, y \) procedurally generated as described in Section 4.3.1, with parameters \((n = 1, m = 1, \sigma_x = 0, \sigma_y = 0, K_x = 10, K_y = 2, s_x = 2, s_y = 5, T = 1 \text{ sec})\). The latency was successfully calibrated to within one frame of error. (top left: signals over time; top right: latency search; bottom left: signals before calibration of latency; bottom right: signals after calibration of latency) ................................. 81
4.3 Example of latency calibration applied to two 2 dimensional signals $x, y$ procedurally generated as described in Section 4.3.1, with parameters ($n = 2, m = 2, \sigma_x = 0, \sigma_y = 0, K_x = 10, K_y = 2, s_x = 2, s_y = 5, T = 1 \text{ sec}$). The latency was successfully calibrated to within one frame of error. .................................................... 82

4.4 Example of latency calibration applied to two 10 dimensional signals $x, y$ procedurally generated as described in Section 4.3.1, with parameters ($n = 10, m = 10, \sigma_x = 0, \sigma_y = 0, K_x = 10, K_y = 2, s_x = 2, s_y = 5, T = 1 \text{ sec}$). The latency was successfully calibrated to within one frame of error. .................................................... 83

4.5 Example of latency calibration applied to two 10 dimensional signals as in Figure 4.4, but here the latency was not correctly recovered. This is likely due to a lack of close points in $x$-space. .................. 84

4.6 Example of latency calibration applied to two 1 dimensional signals $x, y$ procedurally generated as described in Section 4.3.1, with parameters ($n = 1, m = 1, \sigma_x = 0.05, \sigma_y = 0.05, K_x = 10, K_y = 2, s_x = 2, s_y = 5, T = 1 \text{ sec}$). The latency was successfully calibrated close to the true value (black dashed line), though some error is present. .................. 85

4.7 Performance of latency calibration with varying dimension of the signals. For each value, 20 trials were performed each with random latency and procedural functions. Here, $T = 1 \text{ second}$, and the periods of the procedural functions were $s_x = 0.5, s_y = 5$. Strong performance is achieved since the independent signal $x(t)$ is periodic with period less than the sample time $T$, and so there are many high quality close points in $x(t)$. .................................................... 86

4.8 Performance of latency calibration with varying dimension of the signals. For each value, 20 trials were performed each with random latency and procedural functions. Here, $T = 1 \text{ second}$, and the periods of the procedural functions were $s_x = 2, s_y = 5$. Compared to Figure 4.7, performance is reduced by the fact that the function $x(t)$ is not periodic on the domain $[0, T]$, and hence the quality and number of close points is reduced. .................................................... 87

4.9 Performance of latency calibration with varying measurement noise of the signals for one dimensional (left) and two dimensional (right) signals. The red line indicates the boundary of 80% success. Here, $n$ and $m$ refer to the dimension of the independent signal $x(t)$ and dependent signal $y(t)$ respectively. .................................................... 88

4.10 Error in determined latency for the motion example using the proposed method for various image down-sample rates. The down-sampled image pixel count represents the dimension of the dependent signal $y(t)$. .................................................... 89
4.11 Error in determined latency for the long motion example using the proposed method for various image down-sample rates. The down-sampled image pixel count represents the dimension of the dependent signal $y(t)$. Compared to Figure 4.10, the data spanned 300 frames (10 sec), and the increased information and self-intersections enables the solution to be determined at resolutions greater than $7 \times 7$ pixels.

5.1 The relation between two 6DoF sensors is characterized by Equation 5.1. $S$ and $Q$ are the sensor reference frames. $S_t$ and $Q_t$ are the coordinate systems corresponding to the 6DoF pose of each sensor at time step $t$. $Y$ and $X$ are the rigid transforms linking the sensor reference frames and the sensor poses respectively. $A_t$ and $B_t$ are the 6DoF pose transforms of the two sensors at time step $t$.

5.2 Software architecture of the fusion plugin.

5.3 State machine for calibration between two systems. States: Uncalibrated (U), Refining (R), Calibrated (C).

5.4 The three tasks used to assess the performance of the ambient calibration. In the sorting task, the user must sort the cubes (a) into their respective colors (b) on the platforms a few steps away. In the pointing task, the user must point to a series of targets while standing in place (c). The walking task involves the user stepping around the tracking space while facing the Kinect and slowly raising and lowering their arms (d).

5.5 Distributions of errors for Rift (a) and Vive (b) for three different ambient calibration scenarios. Confidence ellipses (95%) are shown as visual aids.

5.6 To measure the accuracy of the ambient calibration $Y$ between the Kinect and the VR systems, an external gold-standard OptiTrack motion capture system was used to measure $K$.

5.7 An example of the typical results for calibration with walking activity. An outline of the user’s real body (white edge) from the Vive’s passthrough camera is shown on the right, overlayed with the virtual scene rendered from user perspective. The controllers are outlined with a green edge. This calibration took 16 seconds, and had an error of $22.2 \text{cm}$ and $5.25^\circ$.

5.8 An example calibration trace with calibration states (Figure 5.3) overlayed in color (Red = ‘Uncalibrated’, Yellow = ‘Refining’, White = ‘Calibrated’). The Kinect is moved at the 55 second mark, giving it a rotation 8 degrees from its original configuration. The error is corrected within 40 seconds.

6.1 Software architecture of the fusion plugin.
6.2 Example recursive fusion of a position measurement relative to the root node. The nodes which are included in the fusion chain have their coordinate systems shown in yellow. The fusion chain only consists of the first three nodes from the end of the chain to give the required 3 positional degrees of freedom. The green nodes represent a possible final pose of the fusion graph if the measurement were to be trusted completely. .......................................................... 122

7.1 Software architecture of the fusion plugin. .......................................................... 138
7.2 Data flow of the Spooky Skeleton Fusion Plugin within Unreal Engine 4. .......................................................... 140
7.3 An example of adding a sensor measurement to Spooky through Unreal Engine 4 blueprints. This method is called whenever new sensor information is available. The sensor is configured to have a known node location - the left hand. .......................................................... 142
7.4 Example Animation Blueprint calls to retrieve the fused Spooky result. 143
7.5 Example configuration of a Spooky Graveyard for fusion of the Leap Motion, Perception Neuron and Oculus Rift (as used in Chapter 8). 144
7.6 Spooky Graveyard class structure within a UE4 level. Items labelled in typewriter font are example objects owned by the surrounding box. .......................................................... 145

8.1 Oculus Rift with Leap Motion attached. .......................................................... 146
8.2 Default Perception Neuron glove (right) compared to the custom glove used in this study (left) as seen by the leap motion infrared cameras with tracking result overlayed. The default glove is not tracked at all. 148
8.3 The custom built glove for the right hand, allowing simultaneous Leap Motion and Perception Neuron hand tracking. Left image shows the sensor arrangement, and the right image shows the final glove appearance with outer-layer. .......................................................... 149
8.4 Example screenshot from Perception Neuron software, with the connected sensors shown on the left in green or yellow (depending on their connection strength). An example tracking result is shown on the right. .......................................................... 150
8.5 The avatar animated by the tracking data during the user study. 151
8.6 Example grip action. If the distance between the thumb and any of the fingers $d$ becomes less than 5cm, the closest object within 5cm of the midpoint of the engaged fingertips becomes gripped. 152
8.7 Software structure for the Leap Motion (LP) solution used for the user study, within Unreal Engine 4. .......................................................... 152
8.8 Software structure for the Perception Neuron solution for the user study, within Unreal Engine 4. The role of the Spooky Fusion Plant here is to retarget the animation from the PN skeleton to the UE4 Mannequin, and use the Oculus Rift data to position the avatar correctly. ................................................................. 152
8.9 Software structure for the Fused Tracking solution for the user study, within Unreal Engine 4. ................................................................. 153
8.10 Palm confidence as a function of angle $\theta_{\text{palm}}$ between the view direction and the palm normal direction. ................................................................. 155
8.11 Keyboard Task ................................................................. 156
8.12 Sorting Task ................................................................. 157
8.13 Throwing Task ................................................................. 159
8.14 Real-World training for the Sorting Task. ................................................................. 161
8.15 Real-World training for the Throwing Task. ................................................................. 162
8.16 Calibration poses performed before each task (marked with a check-mark). Since only the upper body of the participant was tracked, the participant was allowed to sit during calibration. ................................................................. 163
8.17 Example VR view and webcam footage recorded during the study for an example participant. ................................................................. 165
8.18 Participant scores (n=18). The plots display the inter-quartile range (box), median (red line), non-outlier range (whiskers), and outliers (circles). Outliers are points which deviate by more than $1.5 \times IQR$ from the 1st and 3rd quartiles. (LP = Leap Motion, PN = Perception Neuron, FT = Fused Tracking) ................................................................. 166
8.19 Sum hit density of participant throws for each tracking technology. The concentric rings represent the target, the white square and disk represent the ball table and spawn location, and the green cross represents the participant’s location (see Figure 8.13). ................................................................. 170
8.20 Ball hit density for the Throwing Task summed over all participants. The vertical lines indicate the location and size of the target and it’s rings. ................................................................. 171
8.21 Subjective response metrics for each task and each technology. Strongly statistically significant results are marked with *. Maximum mean rank is 3, indicating the technology was always prefered first of the 3 options. Minimum mean rank is 1, indicating the technology was always the last preference. ................................................................. 173
8.22 Tracking technology mean preference rank over all tasks. This indicates that Fused Tracking is the most generally preferred technology. ................................................................. 175
8.23 Response sentiment for each tracking technology. Percentages reported are fractions of response oppotunities; each participant had three opportunities to comment on each of the three technologies, once each task. However, the participant was blind to which technology they were talking about, and the prompt was open ended. ................................................................. 176
8.24 Equipment configuration for ground truth accuracy analysis of proposed fusion algorithm. OptiTrack motion capture markers were used to measure the ground truth position of the wrists relative to the head. 181

8.25 Test 1. Left (orange) and right (blue) wrist position measured relative to the head (black dot) viewed from the top. The x-axis points in the direction of the user’s view (red line), and the y-axis points left-wards (green line). Grid lines are 20cm apart. Notice that the Fused Tracking case reproduces many of the ground truth curves more faithfully than the Leap Motion or Perception Neuron alone. 183

8.26 Test 2. See Figure 8.25 for semantic key. Note that the two large identifiable curves in the right hand data (orange) of the Fused Tracking and Ground Truth cases are less identifiable or missing from the individual tracking cases. 184

8.27 Test 3. See Figure 8.25 for semantic key. In this test, a throwing motion was performed repeatedly with the right hand (orange). The ground truth trace is drawn beneath each trace for comparison (grey). Note how the Fused tracking combines the high accuracy of the Leap Motion in front of the headset with the tracking coverage provided by the Perception Neuron. 185
# List of Tables

1.1 Examples of common commercial and research tracking systems. See Table 1.2 for more information about tracker types. .......................... 3

1.2 Summary of tracking paradigms for determining real time position and orientation of objects. ................................................. 5

2.1 Manual vs Automatic Calibration .............................................. 24

3.1 Calibration error (CE) method - fraction of correct identifications for real sensor experiment .................................................. 64

3.2 Invariant functional (IF) method - fraction of correct identifications for real sensor experiment ................................................. 64

3.3 Confusion matrix of attachment errors for matching one marker to one of two possible locations A and B ............................................. 69

4.1 Snapshots for the videos captured in the two examples. .................. 88

5.1 System-node table example. ....................................................... 100

5.2 Transition conditions for the state machine shown in Figure 5.3. Each of the hard-coded values here are configurable in reality - these are the values used in this study, determined heuristically. ................. 103

5.3 Summary statistics for errors of ambient calibration procedure compared to ground truth. Values are reported in 'mean ± standard deviation' format. .................................................. 108

6.1 Structure of a node in the Fusion Graph .................................. 116

6.2 Types of Articulation. See Section 2.1.2 for definitions of the functions exp, T, S and more information on representing 3D transforms. .......... 117

6.3 Measurement structure ........................................................... 118

6.4 Measurement types and associated data structures ...................... 119

6.5 Measurement flags for controlling measurement fusion behaviour ... 119

6.6 Available Degrees of Freedom for Articulation Types ................. 124

6.7 Required Degrees of Freedom for Measurement Types .............. 125

6.8 Update functions for articulations with velocity modelled. .......... 126

8.1 Example Tracking Presentation Orders ...................................... 160
8.2 Paired change in performance metrics - reported numbers are the amount of change in the metric when a given participant was using the Fused Tracking (FT) solution compared to the baseline of the other two solutions (Leap Motion (LP) or Perception Neuron (PN)). 168
8.3 Wilcoxon signed-rank test for performance metrics when using Fused Tracking (FT) solution compared to either Leap Motion (LP) or Perception Neuron (PN). .......................... 169
8.4 Significance of participant preferences using Kendall’s $W$ test statistic. For significant results ($p < 0.05$), Kendall’s $W \in [0,1]$ indicates the amount of agreement among the participants. ...................... 174
8.5 Summary of the conclusions drawn from the results. ...................... 177
8.6 Error statistics for each tracking solution relative to the ground truth, over the three trials. ......................................................... 182

A.1 Table of Complaint Responses Toward Leap Motion Tracking .......... 203
A.2 Table of Complaint Responses Toward Perception Neuron Tracking . 204
A.3 Table of Complaint Responses Toward Fused Tracking ................. 205
• The problem of ambient sensor fusion is identified as a valuable research direction aimed at allowing researchers, professionals and hobbyists to create cheaper, high quality, modular sensor systems for VR (Chapter 1)

• A literature review is presented summarising the related work in the areas of sensor fusion and VR software middlewares (Chapter 2)

• The contributions of this thesis include four main theory areas:
  1. Ambient sensor identification (Chapter 3)
  2. Model-less calibration of latency between dependent sensor signals (Chapter 4)
  3. Ambient sensor calibration and alignment (Chapter 5)
  4. Modular sensor fusion for articulated bodies (Chapter 6)

• Additionally, two central practical contributions are discussed:
  1. An open-source plugin for Unreal Engine 4 / C++ implementing key sensor fusion algorithms (Chapter 7)
  2. A user study assessing the utility of the developed fusion algorithms (Chapter 8)
Chapter 1

Introduction and Motivation

The rapidly improving quality and availability of head mounted displays (HMDs) has seen a large increase in public interest in virtual reality (VR). For example, PlayStation VR exceeded 1 million units sold in less than a year by June 2017\(^1\) and reached 3 million sales by August of 2018\(^2\). HMDs allow for a wide variety of immersive experiences unavailable to other display platforms. However, HMDs typically block out the real world, leaving the user feeling disembodied in the virtual space. Without tracking of the user’s body and hands, interactions within the environment are limited. Furthermore, without visual body and hand representation within the virtual environment, states of presence and immersion are impeded (Cummings and Bailenson 2016; Skarbez et al. 2017). If the virtual environment is multi-user, communication can also be impeded by inaccurate or limited body or hand tracking (Greenwald et al. 2017).

Tracking systems for achieving high fidelity body and hand tracking are prohibitively expensive. For example, OptiTrack, Vicon, and similar gold-standard motion capture systems can have costs in the range of tens of thousands to hundreds of

\(^2\)blog.us.playstation.com/2018/08/16/celebrating-3-million-ps-vr-systems-sold/ (24/2/2019)
Chapter 1. Introduction and Motivation

thousands of US dollars. Many low cost devices exist as alternatives, but often do not provide the required tracking quality, range or capability for many purposes. Some examples include Leap Motion, Microsoft Kinect, Perception Neuron, Oculus Rift, HTC Vive and PlayStation VR (see Table 1.1 for more information). These devices cost from one hundred to a few thousand US dollars, but suffer from issues such as limited tracking volume, occlusion, drift and low accuracy (for example, see (Rietzler et al. 2016) for details on the Kinect v2).

This thesis presents work towards creating highly accessible fusion software which allows for the combination of low cost tracking systems. The objective of the research was to bridge the quality gap between commodity and gold-standard tracking systems. By minimizing the need for user configuration, the proposed software aims to decrease required technical expertise and increase access to high-quality tracking for businesses, research laboratories and hobbyists.

This research was focused on the domain of modern consumer grade VR platforms, described as follows. We assume that the head is always tracked with 6 degrees of freedom (6DoF). Optionally the system can include 6DoF tracking for each hand and other parts of the body, or independent objects. We will refer to this type of system as a N×6DoF VR system, where N is the number of tracked parts including the head. Examples include Oculus Rift with Touch controllers (3×6DoF), HTC Vive with tracked wand controllers and additional trackers (N×6DoF, with N depending on the number of devices), or PlayStation VR with PlayStation Move Controllers (3×6DoF). The algorithms proposed in this thesis were focussed on extending and improving the VR experience of any N×6DoF VR system by including additional 3rd party tracking systems which are automatically configured in real time to account for sensor placement and tracking quality.

Chapter 1 introduces and motivates the research topic of ambient sensor fusion for VR. Firstly, VR is defined, and examples of state-of-the-art systems are presented. Secondly, the problem of hardware and software fragmentation is identified. A set
Table 1.1: Examples of common commercial and research tracking systems. See Table 1.2 for more information about tracker types.

<table>
<thead>
<tr>
<th>System</th>
<th>Type (Table 1.2)</th>
<th>Tracking Target(s)</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oculus Constellation</td>
<td>Optical</td>
<td>Rigid bodies</td>
<td>IR camera tracks identifiable IR LEDs</td>
<td>oculus.com</td>
</tr>
<tr>
<td>Valve Lighthouse</td>
<td>Optical</td>
<td>Rigid bodies</td>
<td>Object mounted photosensors detect scanning laser</td>
<td>htcvive.com</td>
</tr>
<tr>
<td>PSVR / PS Move</td>
<td>Optical</td>
<td>Rigid bodies</td>
<td>Tracked objects include active colour coded beacons located by a stereo camera</td>
<td>playstation.com</td>
</tr>
<tr>
<td>Windows Mixed Reality Platform</td>
<td>Optical</td>
<td>Rigid bodies and hand skeleton</td>
<td>Cameras affixed to a HMD track both the position of the headset from ambient environment features and user motions (either controllers or free hands)</td>
<td>Hololens, VR</td>
</tr>
<tr>
<td>Mobile AR (AR Core/AR Kit)</td>
<td>Optical</td>
<td>Rigid Body</td>
<td>Mobile camera and inertial sensors fused into 6DoF pose of the camera localised from ambient environment features</td>
<td>AR Core, AR Kit</td>
</tr>
<tr>
<td>Optitrack / Vicon</td>
<td>Optical</td>
<td>Point cloud, rigid bodies, body/hand skeleton</td>
<td>Cameras track retro-reflective dots illuminated by IR LEDs</td>
<td>optitrack.com, vicon.com</td>
</tr>
<tr>
<td>Sixense (STEM / Razer Hydra)</td>
<td>Electromagnetic</td>
<td>Rigid Bodies</td>
<td>Wireless controllers and tracking packs</td>
<td>sixense.com</td>
</tr>
<tr>
<td>Polhemus</td>
<td>Electromagnetic</td>
<td>Rigid Bodies / Skeletons</td>
<td>Wireless tracking packs / small wired 'Micro Sensors'</td>
<td>polhemus.com</td>
</tr>
<tr>
<td>Perception Neuron</td>
<td>Inertial</td>
<td>Body and hand skeleton</td>
<td>Reconstructs skeleton from human kinematic model and measurement of orientation</td>
<td>neuronmocap.com</td>
</tr>
<tr>
<td>Kinect v1</td>
<td>Depth and Optical</td>
<td>Body skeleton</td>
<td>Uses structural light technique to determine depth image, extract human skeleton</td>
<td>(Khoshelham 2011)</td>
</tr>
<tr>
<td>Kinect v2</td>
<td>Depth and Optical</td>
<td>Body skeleton</td>
<td>Uses time of flight technique to determine depth image, extract human skeleton</td>
<td>(Yang et al. 2015)</td>
</tr>
<tr>
<td>Leap Motion</td>
<td>Optical and Depth</td>
<td>Hand skeletons</td>
<td>Uses dual IR cameras and IR illumination to track hands</td>
<td>leapmotion.com</td>
</tr>
<tr>
<td>Dexmo Exoskeleton</td>
<td>Mechanical</td>
<td>Hand skeleton</td>
<td>Hand exoskeleton with rotational joint sensors allows for kinematic reconstruction of fingertip poses</td>
<td>dextarobotics.com</td>
</tr>
</tbody>
</table>
of solutions to the fragmentation problem is then presented as a list of possible features for a software middleware. Some of the proposed features are fundamental to middlewares, while others are novel and extend the role of a middleware beyond simple device compatibility. The central contribution of this research has been the research, development and user testing of a selection of new sensor fusion algorithms which can be applied as a component of advanced VR middlewares.

1.1 Virtual Reality and Tracking Systems

For the purpose of this dissertation, Virtual Reality (VR) is defined as the paradigm of computing which aims to stimulate one or more of a user’s senses for the purpose of displaying a real time virtual environment which is difficult to distinguish from reality. This typically involves displays for the visual and auditory senses, but can also include displays for touch (haptic displays), smell or any other sense. The capabilities of a VR system are defined by the inputs and outputs available. Inputs are any signal which has an effect on the state of the virtual environment. Outputs are signals which depend on the state of the virtual environment. Low latency, high precision, high framerate coupling of inputs and outputs creates an effective VR system (Meehan et al. 2003). Here, the effectiveness relates to the realism of the experience, or the extent to which the system induces presence (Slater, Steed et al. 2013) while avoiding cybersickness (LaViola 2000). Mel Slater defines presence as the subjective sense of “being somewhere” (Slater, Steed et al. 2013), or somewhat more concretely as “how realistically participants respond to the environment as well as their subjective sense of being in the place depicted by the VE” (Slater, Khanna et al. 2009). The sense of presence is a unique and defining feature of VR, hence its use as a metric of VR system quality or effectiveness.

The quality of the VR system can be measured in many ways, with the most basic metrics being input-output latency, input-output precision and refresh rate. The
Table 1.2: Summary of tracking paradigms for determining real time position and orientation of objects.

<table>
<thead>
<tr>
<th>Tracking Technology</th>
<th>Description</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optical</td>
<td>Cameras / sensors measure object bearing with light near the visible spectrum</td>
<td>Cameras/sensors are ubiquitous and cheap</td>
<td>Suffers from occlusion; requires many cameras/sensors</td>
</tr>
<tr>
<td>Depth Sensing</td>
<td>The sensor uses some technique to generate a depth image of the environment from which object pose can be determined</td>
<td>Single sensor tracks large volume</td>
<td>Occlusion; narrow FOV; limited range; refresh rate and accuracy trade-off</td>
</tr>
<tr>
<td>Inertial</td>
<td>Use of self contained sensors which measure inertial properties of a rigid body, such as acceleration and rotational velocity</td>
<td>No occlusion; self contained</td>
<td>Suffers from drift over time</td>
</tr>
<tr>
<td>Mechanical</td>
<td>Mechanical arm is attached to tracked object; sensors in arm measure joint angles allowing for kinematic pose reconstruction of the object</td>
<td>Accurate and stable</td>
<td>Cumbersome and fragile</td>
</tr>
<tr>
<td>Electromagnetic</td>
<td>Measurement of electromagnetic fields (either natural or generated by a base station) allows absolute measurement of the sensor pose</td>
<td>No occlusion</td>
<td>Suffers from distortion caused by ferromagnetic materials</td>
</tr>
<tr>
<td>Global Positioning System (GPS)</td>
<td>A sensor which communicates to satellites which report the position of the sensor</td>
<td>Large scale tracking</td>
<td>Low precision; Legal restrictions; Limited accuracy indoors</td>
</tr>
<tr>
<td>Radio</td>
<td>Triangulation from radio signal hubs</td>
<td>No occlusion; suits large indoor complexes</td>
<td>Low precision</td>
</tr>
</tbody>
</table>

input-output latency, or simply latency, is the time between user action on an input and user perception of the response in the virtual environment, through one or more outputs. The input-output precision, or simply precision, is the difference between the action performed by a user and the perceived result through one or more outputs. The refresh rate of an output or input, measured in frames per second (FPS), measures the number of distinct percepts displayed by the output device or number of samples measured by an input device in one second. Each of these parameters are limited in the design of real VR systems, so it is necessary to apply scientific models of human perception to determine acceptable thresholds for such constrained parameters. For example, head movement to visual response latency is widely considered acceptable\(^3\) at or below 20ms. Above this value, people tend to experience cybersickness associated with dissonance between head movement and visual field.

\(^3\)For example, Oculus states the following in their design guidelines for developers “We believe the threshold for compelling VR to be at or below 20ms of latency. Above this range, users reported feeling less immersed and comfortable in the environment.” (https://developer.oculus.com/design/latest/concepts/bp-rendering/ (14/9/2018)).
change (LaViola 2000). However, this threshold is not discrete; particularly sensitive individuals may experience cybersickness below the 20ms threshold while other individuals can tolerate much more latency. Gains in quality of the environment are always made by increasing refresh rate and precision and decreasing latency.

The majority of VR input systems aim to measure real-world motions in order to replicate them precisely in the virtual environment. Tracking of head position and orientation is commonly used to drive visual outputs with correct view and projection properties, allowing for parallax depth cues and exploration of an environment by natural head movement (Bowman and McMahan 2007). Tracking of body pose allows natural input through movement and gestures. Additionally, body tracking allows for reconstruction of a user’s body in the virtual environment. This is particularly useful for visual displays which obscure the real world, such as most HMDs. Having a virtual body has been shown to increase the level of presence achievable in the VR system (Slater and Usoh 1993b). Classes of trackers are summarised in Table 1.2 and examples of tracking systems are given in Table 1.1.

The next section describes the central disadvantage of distributed development of hardware: fragmentation in device and software compatibility. It also describes how a software middleware can reduce this fragmentation immensely allowing widespread compatibility. Finally, a Fusion Middleware is proposed as a novel way to improve compatibility further.

### 1.2 Device Fragmentation and the Fusion Middleware

Typically, hardware developers will provide a software development kit (SDK) for interfacing with their hardware. It then becomes the job of the software developer to integrate each hardware SDK with their application, one for each device they wish
to support. However, the incentive to do this for a given hardware device is strongly related to the size of the user-base for the device. Paradoxically a device without any software cannot grow a user base. Figure 1.1 shows this fragmented software development model. A middleware solves this problem by handling the integration of software with hardware, allowing each application to simply link to a common interface independent of hardware device. This type of middleware will be referred to as a standard middleware. Figure 1.2 shows the software model for a standard middleware.

In many 3D systems it is desirable to extend tracking range or improve tracking accuracy by combining multiple sensor systems which ordinarily do not communicate. This approach provides a larger tracking space and typically reduces the cost of the system compared to single-unit large-volume tracking solutions. Combining sensor systems can also eliminate the disadvantages of the component sensor systems; for

![Figure 1.1](image.png)

**Figure 1.1:** An example software interface model for development of a virtual reality application. A developer must target each possible input and output device software development kit (SDK). Additional work is required to support alternative input systems; here Input Device 3 and Output Device 3 cannot be used because the developer did not include support. Input/Output Device 2 is compatible with the system, but not compatible with Input/Output Device 1 simultaneously. The majority of included devices are not used by a given user.
example, combining optical and inertial trackers maintains high frequency tracking while eliminating the drift inherent in inertial tracking. To combine sensor system information, the sensor spaces must first be aligned. Two systems are considered aligned when data from one system can be mapped to the other system. Manual alignment is time consuming and error prone: results are typically poor and depend heavily on user skill. Automatic alignment aims to minimise user engagement with calibration and set-up while improving the overall results (Gottschalk and Hughes 1993).

A standard middleware has no handling for device combination. For example, if you have two skeleton trackers, you can use one or the other, provided that a middleware or application has supported each. However, use of both trackers simultaneously, perhaps to increase the overall tracking area, is much more complicated due to overlapping sensor domains. More sophisticated mathematical and algorithmic machinery is needed for this case. Options for fusing sensors to one superior fused

![Figure 1.2: The software structure of a standard middleware.](image-url)

For example, here the user can choose between Input 1 or Input 2, but not both.
sensor should be available to a user in the middleware layer. Figure 1.3 outlines the concept behind fusing sensors to track the body. A middleware which handles simultaneous compatibility of overlapping sensors is a fusion middleware. Figure 1.4 shows the software structure of a fusion middleware.

Sensor fusion is desirable because it allows the extension of a system’s tracking range and improvement of its tracking accuracy. This approach is typically cheaper compared to single unit, high accuracy, large volume tracking solutions. Combining sensor systems can also eliminate the disadvantages of the component sensor systems; for example, combining optical and inertial trackers maintains high frequency tracking while eliminating the drift inherent in inertial tracking. Usability is a key factor; it is important to consider the role of an unskilled user in the use of sensor fusion systems for VR. Previous work suggests it should be possible to integrate additional sensors into an existing sensor network dynamically (Calatroni et al. 2010). Automatic fusion of tracking data aims to minimise user engagement with calibration and set-up. Without automation, set-up procedures are time consuming, error-prone and require that the user understand the underlying system (Gottschalk and Hughes 1993).
1.3 Fusion Middleware Features

This section outlines features were identified in early exploratory research as important for a fusion middleware to posses. A subset of these features was chosen as the novel feature set and central research component for the project. The remaining features are included here for completeness and are indicated thusly.

1.3.1 Hardware-Software Abstraction

This feature is discussed here for completeness, and should in fact be present in a standard VR middleware as well as a fusion middleware. Software developers need to know what input and output modalities the user has access to in order to design a virtual environment. Hardware manufactures should be able to target a flexible and robust software platform with well defined features. For example, a software developer may decide the user needs a HMD and both hands to have at least 6DoF tracking with button inputs. A user should then be able to combine the
minimum hardware together to satisfy the software’s specifications, perhaps using
different vendors for each of the HMD and two 6DoF trackers. It is common for
software to fail to be adopted because it requires niche hardware and conversely for
hardware systems to fail commercially if there is little supporting software. This
is especially problematic in the domain of VR due to the wide variety of input
and output hardware. For example, visuals can be displayed on a single screen, a
surrounding set of screens or a HMD. Each of these display systems requires different
configuration and rendering parameters within the software. On the software end, an
application would request abstract display and sensor resources from the middleware.
For displays, the application would receive information about what to render for the
particular display. For example, in the case of visual displays, a list of sets of view,
projection and viewport matrices would define the required images for display on
the device. Two sets of view parameters might define the views for each eye of a
stereo HMD, while four sets might represent a CAVE system\(^4\). Each view would be
rendered by the application and then passed to the middleware for post-processing
and display. This example device category would be compatible with stereo and
monoscopic screens, HMDs and CAVE systems. A similar problem exists for input
devices. For example, a highly enthusiastic user may purchase a full body tracking
system, with eye tracking and face tracking, while casual users might simply have a
3×6DoF system. Software developers have little incentive to program for hardware
with low adoption. A VR middleware system solves the above problems by allowing
developers to target a common interface, and users to upgrade hardware as they
desire.

\(^4\)CAVE: Cave Automated Virtual Environment - Consists of several screens surrounding the user
and often employs stereoscopy and head tracking to display perspective-correct images (Cruz-Neira
et al. 1993).
1.3.2 Modularity and Sensor Fusion

This thesis is focussed on the features described in this section. Hardware from any given vendor should be easily incorporated into an existing VR system to provide improved tracking. The system should allow user friendly configuration of the hierarchy of trackers and output devices. The simplicity of the configuration process relies heavily on the quality of the automatic calibration processes. For example, an application might request a 6DoF tracker and a number of traditional gamepad inputs such as a joystick or buttons. An end user would then be free to configure the middleware to use a Steam VR 6DoF tracked controller (see Table 1.1) while another user might use a motion capture marker from an OptiTrack system while holding an untracked gamepad. Sensors should also be combinable in the middleware layer, with automatic calibration and fusion of sensors performed by the middleware. Adding additional sensors to an already completely measured state model can statistically improve the tracking result by providing independently sampled data (Mitchell 2007). Difficulties with this include assessing the reliability of each sensor and weighting their influence on the state appropriately. The next section defines the features required to perform sensor fusion ambiently, with minimal involvement and configuration from the user.

Sensor Correlation

It is a non-trivial problem to identify relationships between sensors from different tracking systems. Typically, this task falls to the user. For example, if a user wishes to fuse a Microsoft Kinect and some wireless IMUs (WIMU), they would have to manually indicate to the software which body part each WIMU sensor is placed. Past research has indicated that it is possible to identify such placement automatically from spontaneous user movements (Bahle et al. 2013). Once sensor placement is known, the two sensor systems can be aligned.
While some algorithms for sensor identification are robust to latency, many are not, and any latency between the two sensor systems will cause problem for further steps of fusion. Since timestamps are often unreliable or unavailable, it is important to know the delay between two sensor signals. There exist methods to calculate latency between two related sensors (e.g. (Nikias and Pan 1988)), but they are often limited to sensors where the model is already known (i.e. after the calibration step). Thus, latency needs to be calibrated before the first step in the fusion process. This thesis groups latency calibration and sensor identification under the same category of problem, Sensor Correlation. It is important to know both sensor-sensor relationships and latencies between two systems before fusion can be performed.

Latency Compensation

Some sensors suffer from higher latency than other sensors, and timestamps are often unreliable or not available. This is particularly bad for sensors linked to visual displays as the human vision system is very sensitive to latency. When combining two independent sensors, a low latency sensor can be used to compensate for a higher latency sensor. Often this comes for free, but sometimes it must be handled explicitly. For example, optical sensors are typically slower than inertial sensors, so inertial sensors can be used to predict the motion of a body over the period of time between optical measurements. The time between optical measurements is typically small enough that the inertial sensor doesn’t drift significantly.

Sensor Calibration or Alignment

Two different sensor systems will measure the real world with reference to different coordinate systems. It is vital to obtain an accurate transformation mapping data to a common coordinate system for purposes of combined use. Performing this automatically for an unskilled user is a vital step in a fusion middleware, as manual alignment is time consuming and inaccurate (Gottschalk and Hughes 1993).
Data Fusion

Once two systems have been identified as correlated, and they have been successfully calibrated, the final step is to combine any overlapping measurements. Additionally, a hidden statistical model can be extracted from the measurements with approaches such as Bayesian filtering (Thrun et al. 2005). The final output should improve the stability, accuracy and tracking range of the sensor systems (Mitchell 2007).

Fault Tolerance

A sensor is considered faulty if it reports data which does not represent the state of the system within the sensor’s expected reliability. It is hence important to detect when this is the case and either re-weight the reliability of the sensor or ignore its data completely. It is also important to detect faults automatically for troubleshooting purposes. Consider the case of two sensor fused to track head position. If one sensor begins failing then the user may not notice the change due to the second sensor. However, the reduced tracking accuracy may lead to cybersickness (LaViola 2000). Hence it is important to notify the user of sensor failure or automatically reconfigure the system.

1.3.3 User Configuration and Representation

Applications in VR often require a representation of the user and their real environment. There are also user dependent calculations which developers tend to re-implement repeatedly for each application, such as inverse kinematics for user avatars, even though the results don’t change between applications for a given user. By handling these configurations and calculations in the middleware, this further reduces the load on the developer.
Avatar Kinematics and User Model Configuration

Tracking the entire human body is not necessary for many applications. For example, people will usually interact with the environment with their hands, ignoring the arms entirely. However, it is desirable to infer the plausible pose of the un-tracked body parts based on the tracked parts. This allows a realistic avatar to be animated, increasing levels of avatar body ownership and presence in VR (Slater and Usoh 1993a) while increasing the bandwidth of body language communication in collaborative environments (Garau et al. 2003). By handling the avatar pose in the middleware, simulated poses can fill in gaps between sensors using inverse kinematics and human behavioural models (Caillette et al. 2008). This feature is included in the presented work and described in Chapter 6.

The above model requires knowledge of several properties of a user’s physical body. By storing this information in the middleware layer, the work required by software developers is reduced and the user is saved from needing to enter their personal information into each application which they use. User configuration parameters would typically include body measurements such as inter-pupillary distance and body dimensions. Additional configuration would also be useful, such as the volume of space the player has to move around. Although such configuration functionality exists in this work, configuration is performed inconveniently within the Unreal Engine 4 Editor. More convenient configuration beyond the scope of this project.

Real World Reconstruction and Monitoring

VR systems typically replace the real world with a virtual environment entirely. However, it is necessary at times to inform the user about the real world. For example, if a user is wearing a HMD and a pet or young child enters the tracking volume, ideally there would be some system to warn you of this new obstacle and possibly even display it within the virtual environment. Such functions can
and should be handled by a VR middleware so that it is consistent across applications. VR middlewares like SteamVR and Oculus Home support user configuration of VR ‘play-space’ boundaries which are overlayed on the virtual environment when approached. Such functions could be extended to drawing of sensed obstacles, requiring less setup for the user, more accurate obstacle placement and even dynamic obstacle detection. A large tracking volume would require very high quality sensors to monitor it continuously, so it is more feasible for the middleware to fuse multiple sensors or fuse multiple measurements over time. For example, if a camera is mounted to the front of the HMD, the environment could be reconstructed over time as the user moves using structure from motion techniques (e.g. (Agudo et al. 2012)). This could also be combined with the real time depth video measured by devices such as the Kinect for detection of dynamic foreign objects such as pets or children. Real world monitoring features are not in the scope of this work, but are included here for completeness and to motivate future work.

1.4 Research Question and Contributions

Chapter 1 has established a set of possible features which could be added to VR middlewares to improve user experience. Of course, many of these features could also be applied to augmented reality (AR), mixed reality (MR) and other spatial computing domains. However, this research is focussed on the domain of VR systems since a wide variety of commodity and professional quality tracking devices are compatible with this platform already. Of the possible middleware features described in Section 1.3, this research focusses on sensor fusion (Section 1.3.2). The other features, such as ‘Real World Reconstruction and Monitoring’, are included in Section 1.3 to provide context and inform future work. The core ambient sensor fusion pipeline is visualized in Figure 1.5 and can be summarized by the three key steps which must be performed chronologically:
Chapter 1. Introduction and Motivation

Figure 1.5: The ambient sensor fusion pipeline visualised with a 2D toy example. Each pair of arrows represents a sensor coordinate frame. The grey trails represent recent data from the sensor, such as a position curve. The ellipses represent the rigid bodies which the sensors are attached to in the common coordinate frame.
1. **Correlation**: Determine existence of sensor-sensor dependencies and temporal offsets between systems

2. **Calibration**: Determine a mapping from one sensor system to another

3. **Fusion**: Extract an underlying model from the available data

Figure 1.5 illustrates these concepts with 2D sensors represented by their local coordinate frames. At first, connections between sensors is unknown. Recent pose data is visualised with gray trails. The correlation procedure analyses this data, identifying sensors which are connected. The calibration step then computes the relationships between the sensors which are connected, allowing them to be grouped and represented in the same space. Finally, the fusion step extracts an underlying model of the grouped sensors, represented here by the orange coordinate systems and ellipses. With these three steps identified, the research question of the thesis can now be formulated.

**Research Question**

How can existing sensor calibration and sensor fusion algorithms be extended for ambient fusion of modern tracking systems at the middleware level?

Chapter 1 has discussed the use of middleware for virtual reality, and the value of having accessible sensor fusion integrated with a middleware. A number of features have been identified as useful for extending the functionality of a basic middleware to reduce the load on software developers and improve deliverable functionality. The objective of this research was to investigate algorithms for extending existing middlewares with ambient sensor fusion. The key outcomes and contributions of this thesis include:

- Two algorithms for detecting correlations between 6DoF and 3DoF sensors from spontaneous user movements (Chapter 3).
• A highly novel, model-less approach to determining the latency between two dependent sensors from different systems. In particular, the sensors can be related non-linearly and be high dimensional (more than 100 dimensions) (Chapter 4).

• An algorithm for ambiently calibrating sensor systems from spontaneous user movement (Chapter 5).

• A modular algorithm for fusing data from articulated bodies, including constraint modelling and inverse kinematics (Chapter 6).

• An open source Unreal Engine 4 / C++ plugin with implementations of most of the above algorithms, called ‘Spooky’ (Chapter 7).

• A user study examining the effectiveness of the fusion algorithm proposed in Chapter 6, applied to fusing Perception Neuron and Leap Motion sensors in an immersive virtual environment (Chapter 8).

Figure 1.6 summarises the structure of the thesis. ‘Theory’ contributions marked in the thesis map are focussed on development and testing of the proposed algorithms. Meanwhile, ‘Practical’ contributions are those involving application to real software and user interface research. The thesis map demonstrates the broad and significant results achieved towards creating ambient sensor fusion algorithms.
Chapter 1. Introduction and Motivation

Ambient Sensor Fusion for Virtual Reality

Introduction and Motivation (Chapter 1)

Literature Review (Chapter 2)

Sensor Identification (Chapter 3)

Latency Calibration (Chapter 4)

Ambient Sensor Calibration (Chapter 5)

Skeleton Sensor Fusion (Chapter 6)

Spooky Ambient Sensor Fusion Plugin UE4/C++ (Chapter 7)

User Study (Chapter 8)

Conclusion (Chapter 9)

Figure 1.6: Thesis map.
Chapter 2

Literature Review

Chapter 2 reviews the relevant literature regarding sensor fusion and VR software middlewares. The steps for automatic sensor fusion are as follows:

1. **Correlation**: Identify sensor-sensor dependencies and temporal offsets between sensor systems
2. **Calibration**: Align sensor measurement domains
3. **Fusion**: Fuse sensor data to extract the most likely underlying model from sensor measurements

Firstly, this chapter discusses the state of the art for the first two steps in the fusion process in Section 2.1. In particular, it identifies the need for the first step of sensor correlation if the user is to be minimally involved with the calibration process. Secondly, Section 2.2 covers the third step, fusion, and discusses modern techniques for combining overlapping data from inhomogeneous sensors. Finally, Section 2.3 examines state of the art VR middleware systems and their capabilities.
2.1 Sensor Calibration and the Need for Sensor Correlation

When combining the output of multiple sensors it is necessary to know the transformation between the sensor reference frames. The process of determining this transformation is called alignment or calibration. For example, consider the scenario where two cameras view a common visual target. Suppose the target is also tracked with an inertial measurement unit (IMU). The cameras report relative bearings of the target, while the IMU reports orientation relative to the initial pose of the target. It is necessary to know the relative poses of the cameras in order to triangulate the target position. The cameras might also yield a measurement of the relative orientation of the target. It is necessary to know the frame of reference of the IMU relative to each of the cameras so that the orientation information can be fused with the camera measurements.

2.1.1 Automatic and Manual Calibration

The final accuracy of measured data depends on the manufacturing tolerances used during sensor construction. Minor damage to equipment can lead to additional degradation of accuracy. Hence, it is vital to perform frequent device and setup specific calibration. Accurate alignment by manipulation of the sensor itself (for example, aligning a camera with a predefined coordinate system so that it points along the x-axis) is time consuming, error prone and generally leads to poor results (Gottschalk and Hughes 1993). Using an auxiliary measurement device to measure each sensor pose is an expensive option. The best approach to alignment calibration is to solve for the model parameters based on data from correlated sensor streams. For example, the cameras in the scenario above can be calibrated relative to one another by presenting a known fiducial marker to both cameras simultaneously.
Calibration procedures can be complicated and time consuming, and must be re-performed if even small changes to the system configuration occur. Given minimum information about the desired role of each sensor it should be possible to unite the frames of reference of each tracker. Ideally this would be done without prompting the user. A common approach involves prompting the user to explore the state space of the model through movement and aligning the spaces by correlating simultaneous data. Examples of this type of calibration are common in commercially available devices. For example, the OptiTrack motion capture system (see Table 1.1) requires a calibration procedure where a rigidly connected set of markers must be moved around the tracking space. From the resulting data, the relative camera poses can be determined without the user needing to perform any programming.

Here, some key terms are defined relating to the nature of calibration algorithms. A *calibration algorithm* is an algorithm which determines the nature of statistical relationships between two or more sensors in a Bayesian network based on samples of correlated data. A *manual* calibration algorithm is one in which the user is engaged and aware of the procedure. An *automatic or ambient* calibration algorithm is one in which the user of the system is unaware, and the calibration is performed in the background as needed at start-up or during use. A *user dependent* calibration algorithm is one in which the sensor relationships are directly dependent on the user. For example, the calibration of a touch screen might involve the user pressing the screen at a set of displayed points. In this example, the user provides the information about where, in the touch measurement space, each screen point is, and so it is a manual user dependant calibration procedure. A *user sampled* calibration algorithm is one in which the user determines the state of the system for each calibration sample, but not necessarily the relationship between the sensors. An example of a manual user sampled algorithm would be the calibration of a collection of cameras by presenting a common fiducial marker. The user is responsible for exploring the state space of the system (position of the marker), but the relationship between measurements doesn’t depend on the user. It is possible for a calibration
procedure to be both user dependent and user sampled. For example, modifying the touch screen example, if the user is able to move the screen calibration point with a secondary input before touch calibration, then the procedure is user dependent and user sampled. The relative advantages and disadvantages of automatic and manual calibration procedures are summarised in Table 2.1.

<table>
<thead>
<tr>
<th>Manual</th>
<th>Automatic</th>
</tr>
</thead>
<tbody>
<tr>
<td>· Interrupts the user</td>
<td>· Does not interrupt user</td>
</tr>
<tr>
<td>· Adapts to individual user perception (if user dependent)</td>
<td>· Doesn’t depend on user ability or understanding</td>
</tr>
<tr>
<td>· Stable (user decides when to recalibrate)</td>
<td>· Instability can be frustrating for user</td>
</tr>
</tbody>
</table>

2.1.2 Representing 3D Rigid Bodies

This section reviews the most common mathematical representations of rotation and rigid body motions. A summary of each is given, followed by a comparison which highlights their importance for later use in the calibration and fusion sections of this thesis.

Special Euclidean Group SE(3)

The pose of a three dimensional rigid body can be modelled as a translation \( p \in \mathbb{R}^3 \) and a rotation \( R \in SO(3) \). Here, \( SO(n) \subset \mathbb{R}^{n\times n} \) is the Special Orthogonal Group of dimension \( n \in \mathbb{N} \), defined by

\[
SO(n) = \{ R \in \mathbb{R}^{n\times n} : R^T R = I_n \text{ and } \det(R) = 1 \} \tag{2.1}
\]

The set of vector-rotation pairs \( (p, R) \in \mathbb{R}^3 \times SO(3) \) form their own group under transform composition. Elements of this group can also be represented as a 4x4
matrix

\[ T = \begin{bmatrix} R & p \\ 0^T_3 & 1 \end{bmatrix} \]  \hspace{1cm} (2.2)

where \( 0^T_3 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \). If the spaces are labelled \( F_1, F_2 \) such that \( T : F_1 \to F_2 \). Then a point \((x, y, z) \in F_1\) can be transformed, giving \((x', y', z') \in F_2\) via

\[
\begin{bmatrix}
    x' \\
    y' \\
    z' \\
    1
\end{bmatrix} = T \begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix} \]  \hspace{1cm} (2.3)

If the ‘1’s in Equation 2.3 are replaced by zeros, then the result still holds but represents rotation of a vector instead of mapping of a point. The group of all such matrices \( T \) is called the **Special Euclidean Group**

\[
SE(3) = \left\{ \begin{bmatrix} R & p \\ 0^T_3 & 1 \end{bmatrix} : R \in SO(3) \text{ and } p \in \mathbb{R}^3 \right\} \]  \hspace{1cm} (2.4)

The pose of a rigid body can be represented uniquely by an element of \( SE(3) \), conceptually representing a mapping from local body space to a global reference frame.

**Matrix Functions**

This subsection defines the necessary notation for common matrix functions. The translation matrix corresponding to a vector \( v \in \mathbb{R}^3 \) is defined by

\[
T(v) = \begin{bmatrix} I & v \\ 0 & 1 \end{bmatrix} \]  \hspace{1cm} (2.5)
and operates on points by translating by \( \mathbf{v} \). The scale matrix corresponding to a vector \( \mathbf{s} \in \mathbb{R}^3 \) is defined by

\[
S(\mathbf{s}) = \begin{bmatrix}
    s_1 & 0 & 0 \\
    0 & s_2 & 0 \\
    0 & 0 & s_3 \\
    0 & 0 & 1
\end{bmatrix}
\]

and operates by scaling points in the cardinal directions. Finally, the matrix exponential function is defined by

\[
e^A = \exp(A) = \sum_{k=0}^{\infty} \frac{1}{k!} A^k
\]

where \( A \in \mathbb{R}^{n \times n}, n \in \mathbb{N} \) and \( A^k = A \cdot A \cdot \ldots \cdot A \) (k times). In particular, it can be shown that \( \phi(t) = \exp(At)\phi_0 \) is a solution to the differential equation

\[
\dot{\phi}(t) = A\phi(t)
\]

where the dot indicates differentiation with respect to \( t \). These identities will become useful when discussing rotations.

### Euler Angles

Perhaps the earliest representation of rotations was developed by Euler. Euler first identified that a 3D rotation can be represented by a set of three two dimensional rotations about orthogonal axes. A central difficulty in this approach is choosing the convention for the order of application of the rotations. XYZ Euler angles \( (\theta_1, \theta_2, \theta_3) \in \mathbb{R}^3 \) represent the rotation defined by the following relation to \( SO(3) \)

\[
R_{XYZ}(\theta_1, \theta_2, \theta_3) = R_z(\theta_3)R_y(\theta_2)R_x(\theta_1)
\]
where $R_\alpha$ is the rotation about the $\alpha$ axis, which is simple to compute in terms of standard trigonometric functions (Diebel 2006). This representation is dense in that each $(\theta_1, \theta_2, \theta_3) \in \mathbb{R}^3$ represents a rotation, but each rotation in $SO(3)$ has multiple Euler angle representations due to Gimbal Lock (Hanson 2005) and the periodicity of trigonometric functions:

$$R_{XYZ}(\theta_1, \theta_2, \theta_3) = R_{XYZ}(\theta_1 + 2\pi n_1, \theta_2 + 2\pi n_2, \theta_3 + 2\pi n_3)$$

(2.10)

for any $n_1, n_2, n_3 \in \mathbb{Z}$.

### Quaternions

Quaternions are a generalization of complex numbers to three dimensions. In the same way that multiplication in the complex plane represents rotation, multiplication by unit quaternions can be used to represent 3D rotations. The definition of the quaternion space is

$$\mathbb{H} = \{w + xi + yj + zk | (w, x, y, z) \in \mathbb{R}^4 \text{ and } i^2 = j^2 = k^2 = ijk = -1\}$$

(2.11)

Multiplication is defined on $\mathbb{H}$ based on application of standard field axioms and the assumed properties of $i, j, k$, with the exception that $i, j, k$ do not commute in multiplication. For convenience, in this work, $\mathbf{q} = (q_0, q_1) = (q_0, q_1, q_2, q_3) \in \mathbb{H}$ is used to denote the quaternion $q_0 + q_1i + q_2j + q_3k$. Often, $q_0$ is called the real part of the quaternion, and $q_1$ is called the imaginary part. The exponential function can be generalised from the complex plane to quaternions via $\exp : \mathbb{H} \to \mathbb{H}$ where

$$\exp_{\mathbb{H}}(q_0, q_1) = \exp_{\mathbb{R}}(q_0) \left( \cos(\|q_1\|), \frac{q_1}{\|q_1\|} \sin(\|q_1\|) \right)$$

(2.12)
It can be shown that if a 3D point \( \mathbf{x} \in \mathbb{R}^3 \) is encoded as the quaternion \( \mathbf{u} = (0, \mathbf{x}) \) then the quaternion product

\[
(0, \mathbf{x}') = \mathbf{u}' = \mathbf{quq}^*
\]

results in \( \mathbf{x}' \), the result of rotation about the unit axis \( \mathbf{a} = \frac{\mathbf{q}_1}{\|q_1\|} \) by the angle \( \theta = 2 \|q_1\| \) (Murray et al. 1994). Parametrizing Equation 2.12 gives the quaternion which rotates around the unit vector \( \mathbf{a} \) by the angle \( \theta \)

\[
\mathbf{q}_{\mathbf{a},\theta} = (\cos(\theta/2), \mathbf{a} \sin(\theta/2))
\]

Thus, elements of \( SO(3) \) can be encoded as unit length quaternions. Unit quaternions are a more compact representation of rotations, but they suffer from redundancy: \( q \) and \(-q\) both represent the same rotation.

**Lie Algebras** \( so(3), se(3) \)

The following section summarises Sections 2.2 and 3.2 of Murray et al. (1994). Lie Groups are an important alternative to all of the previously discussed representations because they have a dense representation in \( \mathbb{R}^3 \), but don’t suffer from Gimbal lock like Euler angles. The Lie Group representation \( so(3) \) is emergent from the differential equation for rotation of a point about an axis \( \omega \) through the origin. A point \( \mathbf{p}(t) \) rotating around \( \omega \) with angular velocity \( \|\omega\| \) at time \( t \in \mathbb{R} \) has velocity described by

\[
\dot{\mathbf{p}}(t) = \omega \times \mathbf{p}(t)
\]

Now, the cross product operator can be represented by the *skew symmetric* matrix corresponding to \( \omega \). The skew symmetric matrix of a vector \( \mathbf{a} = [a_1, a_2, a_3]^T \) is given
by the hat operator $\widehat{\cdot} : \mathbb{R}^3 \to \mathbb{R}^{3\times3}$ such that

$$\widehat{a} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_3 & a_1 & 0 \end{bmatrix}$$

(2.16)

Then, some algebra will show that the cross product with vector $a$ can be written in terms of matrix multiplication as

$$a \times x = \widehat{a} x$$

(2.17)

Writing Equation 2.15 with the skew symmetric matrix for $\omega$ gives

$$\dot{p}(t) = \widehat{\omega} p(t)$$

(2.18)

The solution to this differential equation is

$$p(t) = e^{\widehat{\omega}t} p(0)$$

(2.19)

where the matrix exponential is defined by Equation 2.7. The set of matrices $\widehat{\omega}$ is denoted $so(3) \subset \mathbb{R}^{3\times3}$ and is the Lie Algebra corresponding to the Lie Group $SO(3)$. For this thesis, elements $\widehat{\omega} \in so(3)$ are referred to as pivots. The Rodrigues formula provides a convenient method for evaluating the matrix exponential of $\widehat{\omega}$ for any $\omega \in \mathbb{R}^3$:

$$e^{\widehat{\omega}} = I + \frac{\widehat{\omega}}{||\omega||} \sin(||\omega||) + \left( \frac{\widehat{\omega}}{||\omega||} \right)^2 [1 - \cos(||\omega||)]$$

(2.20)

A similar process can be applied to a point rotating about an axis which does not pass through the origin to give $se(3)$, the Lie Algebra corresponding to the Lie Group of $4 \times 4$ matrices $SE(3)$. For every $\xi = [\omega^T v^T]^T \in \mathbb{R}^6$ there is a unique element $\widehat{\xi} \in se(3)$, given by

$$\widehat{\xi} = \begin{bmatrix} \widehat{\omega} & v \\ 0 & 0 \end{bmatrix}$$

(2.21)
Elements $\hat{\xi} \in se(3)$ are called twists, and each one corresponds to a rigid transform, given again by the matrix exponential:

$$e^{\hat{\xi}} \in SE(3)$$  \hspace{1cm} (2.22)

**Which Representation?**

The problem of representing 3D rigid body poses has no perfect solution. Choosing the best representation for a given task is particularly important when applying calibration and fusion algorithms. Some representations span the entirety of their embedded space, such as $so(3)$ and euler angles. For the purposes of this work, such representations will be referred to as dense representation. A key property of dense representations is that, for a rotation $x \in X$ where the rotation space $X$ is embedded in $\mathbb{R}^n$, $x + \delta x$ also represents a rotation for any $\delta x \in \mathbb{R}^n$. This is not true for representations such as quaternions, which must be unit length in order to represent a rotation. Dense representations suffer from two main problems:

- non-uniqueness - one rotation can be equivalently represented by many distinct $x \in X$.
- null-space jacobians - points $x \in X$ exist where some choices of $\delta x$ result in $x \equiv x + \delta x$. For example, Gimbal lock in Euler angle representations.

Dense representations necessarily are embedded in $\mathbb{R}^3$ because 3D rotations have three degrees of freedom. Conversely, some representations have unique representation and no singularities, but are not dense. For example, $SO(3)$. In this case, these representations are referred to as unique representations. These representations are a strict subset of $\mathbb{R}^n$, $n > 3$; for example, $SO(3) \subset \mathbb{R}^{3 \times 3} = \mathbb{R}^9$ with each matrix in $SO(3)$ orthonormal. The key concern when using these representations is that $x + \delta x$ is typically not another rotation, which creates difficulty when dealing with differentiation on these spaces.
Chapter 2. Literature Review

Table 2.1: Spectrum of common mathematical representations of 3D rotation.

<table>
<thead>
<tr>
<th>Representation</th>
<th>SO(3)</th>
<th>Quaternions</th>
<th>Pivots / so(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>Unique</td>
<td>Dense</td>
<td></td>
</tr>
<tr>
<td>Embedded Space</td>
<td>$\mathbb{R}^9$</td>
<td>$\mathbb{R}^4$</td>
<td>$\mathbb{R}^3$</td>
</tr>
</tbody>
</table>

Figure 2.1: Spectrum of common mathematical representations of 3D rotation. Note that although $so(3)$ is technically embedded in $\mathbb{R}^{3\times3}$ it can be parameterized by $\omega \in \mathbb{R}^3$ and is effectively dense.

Some representations are neither unique nor dense. For example, quaternions are embedded within $\mathbb{R}^4$ and must have unit length to represent a rotation. However, they also have degeneracy because $-q$ and $q$ represent the same rotation. Despite these properties, quaternions are often preferable to other representations because they have near-uniqueness and a simple restriction on their length, and thus $q + \delta q$ can be normalised to produce a quaternion representing a rotation which is quite stable for small perturbations $\delta q$. Figure 2.1 summarises the spectrum of rotation representations. Each use of a particular representation is typically chosen based on the desired properties for the application.

Hertzberg et al. proposed a technique for performing sensor fusion on manifold state domains (Hertzberg et al. 2013). Hertzberg introduces two binary operators to replace $+$ and $-$ on the manifold. If $S$ is a manifold of dimension $n \in \mathbb{N}$ and there exists operators\(^\text{1}\)

\[
\boxplus : S \times \mathbb{R}^n \rightarrow S, \tag{2.23}
\]

\[
\boxminus : S \times S \rightarrow \mathbb{R}^n \tag{2.24}
\]

\(^\text{1}\)Pronounced ‘boxplus’ and ‘boxminus’.
then most standard filters and algorithms can be adapted to work on the manifold while avoiding singularities and redundant representations. This is done by simply using $\Box$ wherever an element on the manifold is perturbed by a value in $\mathbb{R}^n$, and using $\sqsubset$ wherever two elements of the manifold are compared. Hertzberg et al. demonstrated the concept by applying it to least squares optimisation on a manifold and Kalman filtering on a manifold. Hertzberg et al. also applied the box operators to filtering of orientation of an IMU, avoiding both the problem of the redundancy involved with the Euler angle representation and the problem of manifold restrictions with representations such as quaternions and rotation matrices. This research is particularly relevant to skeletal tracking and fusion since articulated bodies suffer from similar singularity and redundancy problems in the manifold of possible joint angles. The fusion approach proposed in Chapter 6 takes a similar approach by transforming between representations depending on what properties are required for the operation.

### 2.1.3 6DoF Sensor Calibration

This section reviews the relationship between two rigidly linked 6DoF sensors. Each sensor is assumed to measure a 3D pose, including both rotation and position, relative to some fixed coordinate frame. The coordinate frames of each sensor are assumed to differ by a constant rigid transform. Given two sensors from separate sensor systems, with rigid linkage between the sensors, the two unknown variables which determine the dependence relationship are the fixed transformation between the sensor coordinate frames and the fixed transformation between the sensors themselves.

Let $S$ and $Q$ be two coordinate systems in $\mathbb{R}^3$ corresponding to two 6DoF sensor reference frames, each with a single sensor (Figure 2.2). Let a sample for each of the two sensors be observed for the time steps $t = 1, \ldots, N_s$ for some $N_s \in \mathbb{N}$. For each $t$, the sensors each define coordinate systems $S_t$ and $Q_t$ respectively. Sensor
Figure 2.2: The relationship between two 6DoF sensors has characteristic equation given by Equation 2.25. $S$ and $Q$ are the sensor reference frames. $S_t$ and $Q_t$ are the coordinate systems corresponding to the 6DoF pose of each sensor at time step $t$. $Y$ and $X$ are the rigid transforms linking the sensor reference frames and the sensor poses respectively. $A_t$ and $B_t$ are the 6DoF pose transforms of the two sensors at time step $t$.

measurements yield $A_t, B_t \in SE(3)$ such that $A_t : S_t \rightarrow S$ and $B_t : Q_t \rightarrow Q$ for each $t$. If the two sensors are rigidly linked, then there are two constant $X, Y \in SE(3)$ such that $Y : Q \rightarrow S$ and $X : Q_t \rightarrow S_t$. This gives the relationship

$$A_t X = Y B_t$$

(2.25)

for every $t = 1, \ldots, N_s$. Assuming homogeneous matrix representation, the robot-world and hand-eye calibration problem is defined as that of determining the transforms $X$ and $Y$ such that the sum squared error

$$\sum_{t=1}^{N_s} \|A_t X - Y B_t\|^2$$

(2.26)

is minimised, where $\| \cdot \|$ is the Frobenius Norm\(^2\).

\(^2\)For $M \in \mathbb{R}^{n \times m}$, the Frobenius norm is defined by $\|M\|^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} M_{ij}^2$, where $M_{ij}$ is the $ij$th component of $M$. 
Zhuang et al. (1994) was the first to describe a linear approach to approximating the solution to the robot-world and hand-eye calibration problem. They derived Equation 2.25 in the context of calibrating tool-flange and base-world transforms for industrial robots. Their technique computes the least squares solutions to two equations derived from the properties of the quaternion equivalent of Equation 2.25. First, the quaternions corresponding to $X$ and $Y$ are computed by solving a linear least squares equation. Second, the translation components of $X$ and $Y$ are computed using the result for their quaternion components in another linear least squares step. The technique generalises to 3DoF orientation sensors by only performing the first stage of the calculation.

Additional techniques have been developed by Dornaika and Horaud (1998): a closed form method and a non-linear optimisation approach. The non-linear method was demonstrated to be more robust to noise than both the linear method proposed by Zhuang et al. and the closed form method. The linear and closed form methods were shown to have similar stability. Li et al. (2010) proposed two additional techniques using dual quaternions and the Kronecker product. The latest approach was proposed by Shah (2013), also using the Kronecker product to determine the rotation components of $X$ and $Y$. Each algorithm requires $N_s \geq 3$ samples of corresponding matrix pairs $A_t$ and $B_t$. The algorithms also trivially cover the 3DoF case, usually by omitting the second part of the computation.

### 2.1.4 Sensor Identification

Automatic calibration is becoming increasingly important in many aspects of virtual reality because of the high precision required to deliver a realistic experience in tracked environments. Calibration is also important because of the natural variation between individual perceptions and physiology (Moser et al. 2015; Plopski et al. 2015). Plopski et al. presents a method for automatic calibration of see-through displays for aligning augmented reality content, identifying the need for continuous
automatically calibrated (Plopski et al. 2015). Destelle et al. demonstrates a procedure for fusing inertial data with optical data from the Microsoft Kinect to produce a hybrid skeletal tracking solution with accuracy rivaling that of expensive gold standard optical tracking solutions (Destelle et al. 2014). However, part of the fusion process involves meticulous alignment of each IMU reference frame with one another and alignment of the IMUs with respect to the Kinect reference frame. This step could be automated using the techniques discussed in Section 2.1.3, but only if the location on-body of each sensor is known. Thus, prior to performing automatic calibration, sensor correspondences must be identified - each IMU in this case matched to a bone on the skeleton tracked by the Kinect.

Previous work has been done on identification of placement of on body sensors from ambient data and models of human behaviour. It has been demonstrated that it is possible to identify the categorical on-body position (Kunze, Lukowicz, Junker et al. 2005) and yaw orientation (Kunze, Lukowicz, Partridge et al. 2009) of an IMU during walking. Detection of walk cycle characteristics in sensor measurements allow the desired information to be deduced based on models of typical walking patterns. Lester et al. (2004) demonstrated that it is possible to determine whether or not two sensors are attached to the same person based on frequency domain coherence function analysis of the two sensor data streams. By exploiting existing sensors and state space models (in this case, models of human behaviour), Calatroni et al. demonstrated improved integration of additional sensors into existing networks based on sporadic user actions (Calatroni et al. 2010). Bahle et al. (2013) used dynamic time warping and time series approaches to localise on body inertial sensors based on skeletal data extracted from an ambient depth camera. Bahle et al. aimed to use such methods to allow wearable devices to self correct for errors given anonymous public data from skeletal tracking cameras in public spaces. The angle which each limb made with the gravity vector was the key data used to compare the sensors with the skeletal data. Chapter 3 provides an alternative method which uses all six degrees of freedom, but assumes a higher level of skeletal tracking fidelity.
Banos et al. (2012) demonstrated a learning system for automatic modality translation from optical skeleton data to acceleration data for several upper torso IMUs. The approach involved transfer learning a linear multi-input-multi-output (MIMO) map. The resulting MIMO map translates a time sequence of skeletal data to the corresponding accelerometer data and vice versa. The method is a manual user sampled calibration algorithm and requires prior sensor location identification. By automatically detecting correspondences, Banos et al. notes the procedure could be automated. Chapter 3 extends the work by Banos et al. by investigating their suggestion of using calibration error to automatically localise on-body sensors.

Fault detection and correction of on-body sensors is also a popular research topic. Kunze and Lukowicz (2008) present heuristics for determining when an accelerometer is displaced by a small amount within its attached limb. Forster et al. (2009) demonstrates recalibration for detecting and compensating on-body inertial sensor rotations and displacements in a highly redundant sensor network. Chavarriaga et al. (2013) extends the method developed by Forster et al. with an unsupervised on-line expectation-maximisation algorithm. Chapter 5 implements an alternative approach by detecting faults by checking new calibration results compared to the accepted result.
2.2 Sensor Fusion

This section summarises the literature on the final step of automatic sensor fusion - the combination of measurements and extraction of an underlying model.

2.2.1 Formal Definition of Sensor Fusion

The problem of sensor fusion lies in the realm of dynamical systems analysis (Mitchell 2007). Typically, it is assumed that there is a hidden state of a system and this state is to be inferred using indirect measurements. Measurements may also be statistically independent, and this additional information can improve the accuracy of the result. Firstly, this notion is defined more precisely.

Definition 2.1 (Continuous Dynamical System). A continuous dynamical system \( S(S, D) \) of dimension \( n \in \mathbb{N} \) and order \( m \in \mathbb{N} \) is a subset \( S \subset \mathbb{R}^n \) and a function \( D : \mathbb{R} \times S \times \mathbb{R}^{mn} \rightarrow \mathbb{R}^k \) for some \( k \in \mathbb{N} \) such that for any \( x \in S \) the time evolution of the system satisfies

\[
D(t, x, \dot{x}, \ddot{x}, \ldots, x^{(m)}) = 0
\]  

(2.27)

Thus, a continuous dynamical system is simply one that can be predicted based on a differential equation. However, these predictions always contain some error when modelling the real world. To capture this notion, probability distributions over a state space are used. The space of probability distributions over a set is defined as follows.

Definition 2.2 (Probability Space). Given a set \( X \), the probability space of \( X \), denoted \( P(X) \), is the set of functions \( p : X \rightarrow \mathbb{R} \) such that \( p(x) \in [0, 1] \) and

\[
\sum_{x \in X} p(x) = 1
\]  

(2.28)
If $X$ is a continuous space such that $X \subseteq \mathbb{R}^n$, then $p(x)$ is a probability distribution function (PDF) and

$$\int_X p(x)dx = 1 \quad (2.29)$$

This concept captures the notion that a range of $x \in X$ are possible, each with relative likelihood represented by the probability distribution. A sensor is then defined as a function relating the state of a system to a probability distribution in a measurement space. This distribution describes the likelihood of a particular measurement outcome.

**Definition 2.3 (Sensors and Measurement).** A sensor of order $m \in \mathbb{N}$ of a system $S$ is a map $\phi : S \rightarrow P(M)$ for some $M \subset \mathbb{R}^m$ called the measurement space of the sensor. A measurement of a system in state $x$ by sensor $\phi$ is a random sample $z \in M$ of the PDF $\phi(x)(z) \in P(M)$. In this case, the sensor is said to measure the system $S$.

The problem of sensor fusion is then the process of inverting a set of sensor maps described above to infer the hidden state.

**Definition 2.4 (Sensor Fusion).** Let $\mathcal{S}(S, D)$ be a continuous dynamical system of order $n \in \mathbb{N}$. Let $\Psi = \{\phi_i : S \rightarrow P(M_i) : i = 1, \ldots, l\}$ be a set of sensors with measurement spaces $M_i \subset \mathbb{R}^{m_i}$ for some $m_i \in \mathbb{N}$ for each $i = 1, \ldots, l$. Suppose that measurement $z_{i,j} \in M_i$ is sampled from $\phi_i(x_j)(z)$ at time $t_j \in \mathbb{R}$ for $j \in \mathbb{N}$, $i = 1, \ldots, l$.

The problem of sensor fusion is to determine $p(x_k | z_{i=1,\ldots,j;j=1,\ldots,k})$ for each $k \in \mathbb{N}$ given the the system model $\mathcal{S}(S, D)$, the sensors $\phi_i$ and the measurements $z_{i,j}$ for $j \leq k$.

From this definition it can be seen that if any one of the sensors is noiseless and has a measurement function which is bijective, then each state can be determined by simply inverting the measurement function. However, in reality this is often a poor
approximation. All real sensors have noise, and typically the order of the sensor is less than the dimension of the system, excluding the possibility of simple inversion. Typically, sensor fusion problems require a real-time solution, where the a new state is estimated based on the last estimated state and the latest measurement. In the setting of VR, the underlying model is some representation of the user’s body, and sensors typically measure a set of rotations and positions of rigid body joints. The use of sensor fusion methodologies can extract a most likely model state given a set of measurements. The most popular fusion paradigm is Bayesian Filtering, and this thesis describes the use of this method to ambiently fuse sensors for VR.

2.2.2 Bayesian Filtering

Bayesian filtering is typically used to estimate the state of a system which is to be controlled by some form of actuator, such as a motor. For example, consider the problem of estimating the position of a wheeled robot while controlling it to perform some task. Bayesian filtering incorporates the rotational velocity of the tires to update the state estimation, while also incorporating computer vision data to correct for any errors accumulated over time. Bayesian filtering generalises to systems where control of the state is not the objective, such as in determining the pose of a VR user, by simply including no control inputs. Applying the properties of Total Probability and Bayes Rule to a state belief distribution allows a filter to be constructed which tracks the probability of all states over time without direct observation of the state (Thrun et al. 2005). The theorem of total probability states that the probability of state $x$ can be expressed in terms of a sum over all conditional probabilities, $p(x|y)$. That is

$$p(x) = \int p(x|y)p(y)dy$$  \hspace{1cm} (2.30)
Bayes rule relates complementary conditional probabilities:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$  (2.31)

These two equations form the basis of the Bayesian Filter. The Bayesian filter estimates the state of a system at time $t$, $x_t$, in the form of $\text{bel}(x_t) := p(x_t|u_{1:t}, z_{1:t})$, where $u_t$ is the control at time $t$, representing the change in the system state, and $z_t$ is the agglomerated measurement at time $t$. The Bayesian filter is updated in two steps. The first step is the prediction update, where the model is stepped forward in time according to the control $u_t$ and the internal dynamics model. The second step is the measurement update, which integrates the information inferred from a measurement.

1. Prediction Update:

$$\overline{\text{bel}}(x_t) = \int p(x_t|u_t, x_{t-1})\text{bel}(x_{t-1})dx_{t-1}$$  (2.32)

2. Measurement Update:

$$\text{bel}(x_t) = \eta p(z_t|x_t)\overline{\text{bel}}(x_t)$$  (2.33)

The intermediate belief is defined by $\overline{\text{bel}}(x_t) := p(x_t|u_{1:t}, z_{1:t})$ and $\eta$ is a normalisation constant such that $\int \text{bel}(x_t)dx_t = 1$. The prediction update is simply Theorem of Total Probability (Equation 2.30), while the measurement update is Bayes Rule (Equation 2.31). The values of $p(x_t|u_t, x_{t-1})$ and $p(z_t|x_t)$ are calculated with a state model in the form of a prediction equation giving a predicted next state $x_t = \psi(x_{t-1}, u_t)$, and a measurement equation giving a measurement from the current state $z_t = \phi(x_t)$.

An example of applying a Bayes filter would be localising a wheeled robot. The prediction update integrates information from the motion of the wheels, shifting the
entire probability distribution according to the movement. The measurement update would involve calculating the probability distribution for observing map features given the current state and comparing it to the observed location.

The difficulty of a Bayes filter is in storing the belief for a continuous state space, such as the pose of a robot. This problem is solved by making a some assumptions about the PDFs which are involved to increase the computational tractability. The next sections outline examples of particular filters which use different assumptions to produce filters of varying efficiency and effectiveness.

**Kalman Filters**

A *Kalman filter* is a Bayesian filter which assumes that

- $\text{bel}(x_t)$ is described by a multivariate Gaussian with mean $\mu_t$ and covariance matrix $\Sigma_t$
- The prediction equation is linear with some Gaussian noise $\epsilon_t$:
  $$x_t = \psi(x_{t-1}, u_t) = A_t x_{t-1} + B_t u_t + \epsilon_t$$
  \hspace{1cm} (2.34)
- The measurement equation is linear with some Gaussian noise $\delta_t$:
  $$z_t = \phi(x_t) = C_t x_t + \delta_t$$
  \hspace{1cm} (2.35)

These assumptions result in each probability distribution reducing to a multivariate Gaussian at each step of the algorithm. However, these assumptions are quite limiting. Consider the rather simple situation of including the heading of a robot in a localisation model. The assumptions break down due to non-linearity in the measurement and prediction equations caused by the presence of a rotation matrix.
dependent on heading. Very few systems can be effectively modelled using the basic linear Kalman filter.

Two alternatives are the Extended Kalman filter (EKF) and the Unscented Kalman Filter (UKF). Extended Kalman Filters approximate the measurement and prediction equations at each step with the first order Taylor series expansion centred at the current mean, $\mu_t$ (Reif et al. 1999). Unscented Kalman Filters approximate the action of the prediction and measurement equations by applying them to a set of weighted sample points from the belief distribution (Wan and Van Der Merwe 2000). After the transformation, the new mean and covariance matrix can be calculated from the result. Both methods approximate the best Gaussian distribution fitting the image of the belief under the non-linear prediction and measurement equations. Chapter 6 uses a modular EKF approach to model the entire skeleton of a user.

**Particle Filter**

A particle filter is a non-parametric Bayesian filter which uses a set of particles in the state space of the system to represent the PDF of the state estimator. This approach has the advantage of being relatively simple, non-linear and flexible in the types of distributions it can represent. However, the approach is infeasible for resolving high dimensional state spaces and complex distributions in real time due to the computational costs associated with the large required number of particles (Snyder et al. 2008).

**2.2.3 Skeleton Fusion**

The sensor fusion step involves combining data to extract an underlying model of the tracked system. In the setting of VR, the underlying model is an articulated body representing body pose. The objective of the final step of the ambient fusion process is to develop an algorithm for automatically fusing multiple skeletons
in a modular way. This section presents a summary of previous works involving multimodal skeleton tracking and fusion.

Approaches to tracking human body pose with one or more cameras or depth cameras can be broken down into three classes of techniques (Sigal et al. 2007). Discriminative methods use machine learning techniques to map image and depth feature spaces to pose spaces. Generative methods use models of the human body and optimisation techniques to minimise re-projection error. Discriminative methods are typically more effective and faster than generative approaches, but they do not include models of body shape and depend heavily on the amount and quality of training data. Hybrid methods use both discriminative and generative techniques to combine the relative advantages of each method.

Helten et al. (2013) describe an approach to track the human body in real time using a single depth camera and inertial sensors. Helten et al. modifies a hybrid tracking approach developed by Baak et al. (2013) by incorporating inertial data at each step of the process. Helten et al. uses a model of limb visibility to determine information in the measurements from the depth camera. The inertial sensors were used to:

- determine the initial forward direction of the body from the torso IMU
- lookup an initial pose whenever limbs were obscured in depth data

With these simple extensions of Baak et al.’s method, Helten et al. were able to generalise the algorithm to handle many obscured poses. However, this technique is not generalisable to modular fusion of skeletons; Helten et al. only uses the inertial sensors to generate plausible poses whenever the depth map is obscured.

Non-parametric belief propagation (NBP) (Sudderth et al. 2010) has been shown to be useful in determining human body pose due to its complete modelling of complicated Bayesian networks. These methods are computationally expensive and so are not feasible for real time tracking and fusion.
Pons-Moll et al. (2010) presents a technique for reconstruction of body pose from 2D cameras and inertial sensors using generative techniques. Their proposed technique uses numerical methods such as the Householder algorithm to minimise camera reprojection error of a laser scanned 3D model of the user. Inertial data is incorporated by simply adding a second term to the optimisation energy equation. This approach would be generalisable to fusing multiple skeletons without context of their origins, but it does not run in real time, which is a necessity for VR. Pons-Moll et al.’s approach assigns equal weight to the inertial and optical skeletons in the fusion step, which will produce poor results if one of the skeletons fails. However, it would be easy to modify the algorithm to assign skeleton weights based on some heuristic of information contained within the skeleton tracking result. The algorithm in Chapter 6 describes use of an analogous energy term approach to modify a Kalman filter to consider body constraints. Additionally, the inclusion of a per-articulation confidence term allows for modelling of occlusion.

Destelle et al. describes a simple approach to the fusion of skeleton tracking results from a Kinect depth camera and a collection of wireless IMUs (Destelle et al. 2014). The Microsoft Kinect (see Table 1.1) is used to measure the global position of the torso and initialise the pose of the user. The IMUs are then responsible for measuring the rotation of each of the 9 bones in the skeleton. This improves the stability of the result compared to just using the Kinect and adds positional tracking and simple startup calibration to the case of just using the IMUs.

Chapter 6 proposes a sensor fusion algorithm for combining data from multiple skeleton tracking solutions. The proposed solution involves use of an optimised constrained Kalman Filtering technique to resolve the pose of articulated bodies in real time. Chapter 7 details the implementation of this algorithm as an open source C++ plugin for the popular Unreal Engine 4 game engine. Chapter 8 evaluates the effectiveness of the proposed technique with a user study comparing Leap Motion and Perception Neuron data with and without fusion.
2.3 Existing VR Middleware Software

This section describes existing VR middlewares which perform a range of different functions, from basic windowing-level functions to game engine functions, through to sensor fusion.

Windowing-Level Libraries

Basic examples of VR middlewares include CAVElib, FreeVR\(^3\) and VR Juggler\(^4\). These libraries handle low level display and graphics abstraction for VR systems. All three of these libraries have a long history dating back to the 1990s. FreeVR and VR Juggler are open source libraries which have received minimal attention in the last few years. CAVElib was the first API developed for CAVE systems, but has since extended to numerous other immersive displays. CAVElib has been commercialised since 1996\(^5\). It was originally developed at the University of Illinois, Chicago\(^6\).

VRUI Toolkit

The Virtual Reality User Interface (VRUI) Toolkit is a graphics and interaction engine for 3D user interfaces across a range of display types and tracking devices\(^7\). VRUI also offers abstraction for hardware displays and input devices, and abstracts the distribution of computation across a network of computers. The distribution of a VR application across multiple computers is particularly important for larger VR systems such as CAVEs. A VRUI application can run on a variety of vastly different display and interaction systems, including screens, HMDs and CAVE systems.

---


VRUI is open source\(^8\) with the GNU General Public License and is aimed centrally at researchers who have a high level of skill. It intends to encourage consistent 3D user interfaces while reducing the work required for individual developers and researchers. For example, the toolkit has been used extensively in research on VR workflows (Kreylos et al. 2006).

**VRPN**

The Virtual Reality Peripheral Network (VRPN) is a library for the development of network distributed VR systems (Taylor et al. 2001). This allows different input and output devices to communicate independent of network topology, reducing the load on the user. VRPN does not handle windowing, graphics or application level details.

**SteamVR**

SteamVR and the underlying API OpenVR\(^9\) form a VR middleware developed by Valve alongside the well established Steam gaming platform for PCs\(^10\). SteamVR targets a wide variety of N×6DoF VR platforms, including HTC Vive, Oculus Rift and Windows Mixed Reality headsets. SteamVR handles user and room configuration, and allows for additional tracked rigid bodies from a few different devices such as the HTC Vive Tracker\(^11\). Additional 6DoF trackers can be emulated using the Microsoft Kinect v2 and Leap Motion through third party plugins\(^12\). However, calibration of such devices is performed manually by the user and sensor fusion is not supported.
OSVR

The Open Source Virtual Reality (OSVR) project is a movement toward establishing a VR software middleware\(^{13}\). OSVR focuses on reducing the fragmentation of the VR hardware and software market for gaming in particular. It aims to solve the paradoxical problem with development: software developers will not target a poorly established hardware platform, while hardware platforms cannot become established without compatible software. The project is open source, promoting development of support for a broad spectrum of VR input and output devices.

OSVR includes a module called *OSVR-Fusion*\(^{14}\) which allows for the combination of data from multiple devices, within the ‘Analysis’ group of modules). However, this module is relatively simple: alignment of sensor systems is performed manually by setting a zero pose of each device. Furthermore, fusion doesn’t support overlapping sensor domains nor hidden state modelling. For example, the orientation data of an IMU can be combined with the position data from an optical sensor, but two IMUs cannot be fused.

OpenXR

OpenXR is an open source middleware and standardisation initiative by the Khronos group\(^{15}\). The central objective of OpenXR is to enable widespread compatibility amongst VR and AR devices and applications. The project is supported by a majority of the large companies presently involved in VR and AR hardware, including Oculus, HTC, Microsoft, and Google, as well as companies building software for such devices, including Unity and Epic Games (Unreal Engine 4). Although still in early prototype stages, the OpenXR design is split into two main components: the application interface, responsible for abstracting connections between applications


\(^{15}\)[https://www.khronos.org/openxr](https://www.khronos.org/openxr) (12/10/2018)
and VR ecosystems such as SteamVR, and the device layer which allows agnostic compatibility between any VR ecosystem and and hardware device. The VR ecosystems themselves perform functions such as access to online store fronts, application management and social services.

**Ubitrack**

Ubitrack is an open source software system for modular fused real time 6DoF tracking for the purpose of augmented reality (AR) applications (Pustka, Huber, Waechter et al. 2011). Ubitrack features network infrastructure for modular sensor systems distributed across multiple computers. Ubitrack uses a Sensor Relation Graph (SRG) for data queries. The system also performs sensor correlation (the first step of ambient fusion) using coherence frequency analysis on rigid body properties such as angular and linear velocity (Strasser 2004). This enables ambient calibration. However, configuration of Ubitrack is difficult and aimed at researchers and experts\(^{16}\). The Ubitrack system is complex, containing code for numerous hardware drivers as well as the core fusion algorithms. This can be undesirable for rapid development due to increased complexity. Furthermore, Ubitrack has become outdated in the modern landscape of tracking devices, especially skeleton tracking devices. The proposed algorithms in this thesis extend the work done by UbiTrack, and are implemented in an open source code-base compatible with the popular game engine Unreal Engine 4 (Chapter 7).

**Fusion Kit**

Rietzler et al. presented a real time framework for combining skeleton tracking data extracted from multiple depth cameras (Rietzler et al. 2016). Their work

\(^{16}\)Setup requires significant expertise and configuration of code (http://campar.in.tum.de/UbiTrack/WebHome (14/9/2018).
is available as open source\textsuperscript{17} software called FusionKit. The software allows for multiple networked computers to send skeleton data from the Microsoft Kinect v2 to a central computer for registration and fusion. Identification of sensor dependencies is performed on a per-user basis, with skeletons matched before registration by joint configuration and length, or matched after registration by distance in the global coordinate frame. Registration is performed ambiently using Iterative Closest Point (ICP) algorithm (Besl and McKay 1992) performed on the skeleton joints generated by movement of a user within the mutual tracking space of two sensors. Chapters 5 and 6 extend this work to include inhomogeneous cases such as depth cameras with a VR system, and Leap Motion and Perception Neuron.

\textbf{Other Fusion Middlewares}

Society of Devices Toolkit (SoD-Toolkit) (Seyed et al. 2015) is another open source ubiquitous tracking platform supporting a wide array of devices such as Microsoft Kinect and Leap Motion. However, the system does not provide high quality articulated body tracking, but rather focuses on providing coarse scale multi-user information with multiple tablet devices.

Jester is an open source human skeletal sensor fusion layer for virtual environments (Schapansky 2014). Jester defines a middle-ware architecture for abstracting the hardware and software layers in a virtual environment, while also providing support for fusion of sensors with support for basic per-joint filters such as the Kalman filter. Jester also requires explicit calibration of systems. Chapter 6 describes our approach that fuses the skeleton as a whole in real time, including constraints, enabling emergent features such as inverse kinematics for position measurements.

\textsuperscript{17}Although presently only binaries are available due to reported licensing issues (\url{https://github.com/fg-ulm/fusionkit (14/9/2018)}).
Chapter 3

Sensor Identification

Chapter 2 has established that there are many effective methods for automatically aligning multiple 3D sensor systems, provided sensor dependencies are known. For example, recall that two rigidly linked 6DoF or 3DoF sensors from different sensor systems can be aligned using techniques including a variety of linear and non-linear optimisation approaches (Dornaika and Horaud 1998; Li et al. 2010; Shah 2013; Zhuang et al. 1994). However, if multiple unlabeled sensors are present, a user is required to identify sensor-sensor dependencies. For example, if a user is tracked by a Microsoft Kinect, and they wish to add an IMU to a body part, they must identify which body part the sensor is attached to, and this cannot be changed if the sensor is moved without the user reconfiguring. It would be significantly more convenient if the user could simply attach the sensor and let the system figure out where it is and how to fuse the data.

This chapter details a methodology for using sensor streams alone to automatically deduce the dependencies amongst a collection of unidentified 6DoF and 3DoF sensors. The aim is to allow for user disengaged automatic alignment of multiple sensor systems (Figure 3.1). The calibration techniques in (Dornaika and Horaud 1998; Li et al. 2010; Shah 2013; Zhuang et al. 1994) were originally developed for
industrial robots. In this chapter these algorithms are applied to sensor identification and calibration of 3D user interfaces to increase the usability of multiple sensor systems in 3D virtual environments. The techniques developed apply to numerous devices, including a wide variety of industrial and consumer grade tracking systems.

This chapter draws from the work published in Fountain and Smith (2016) and Fountain and Smith (2018) completed as part of this research project. The contributions of this work include two methods for identifying rigid links between sensors from separate systems (the invariant functional (IF) method and the calibration error (CE) method), a simulation analysis of performance, and two case studies applying the methods. The methods are focussed on identification of sensors for body-scale skeletal tracking and the algorithm represents the first key component in the ambient fusion process.

Previous work has been done on matching 6DoF time series assuming that $Y$, the transform between sensor systems reference frames (see Figure 3.2), is known (Shah 2011). No studies approaching the problem without knowledge of either $Y$ or $X$ (the

Figure 3.1: Sensor matching and alignment concept.
rigid transform between attached sensors) are known by the author. This chapter extends previous approaches by not assuming prior knowledge of $Y$ and $X$.

### 3.1 Matching of Sensors

Let two sensor systems $S$ and $Q$ each measure a collection of 6DoF sensors $\{S_k\}^{k=1,...,K}$ and $\{Q_l\}^{l=1,...,L}$. The sensors $S_k$ and $Q_l$ measure the $4 \times 4$ homogeneous matrices $A^k_t$ and $B^l_t$ for each time step $t$ such that $A^k_t : S_k \rightarrow S$ and $B^l_t : Q_l \rightarrow Q$. Whenever sensor $S_k$ is rigidly linked to sensor $Q_l$, then

$$A^k_t X_{k,l} = Y B^l_t$$  \hspace{1cm} (3.1)

where $X_{k,l} : Q_l \rightarrow S_k$ is the rigid link transform between the sensors and $Y : Q \rightarrow S$ is the transform between the sensor spaces (see Figure 3.2, reproduced from Section 2.1.3). Here, work is presented towards automatically identifying, or matching, such rigidly linked sensors in an automatic, user sampled calibration procedure.

#### 3.1.1 Invariant Functional (IF) Method

This section details a sensor matching method based on the intuition that some characteristics of the systems will remain invariant under the transformation from one sensor space to another. For example, two rigidly linked sensors will rotate through the same magnitude angle over a given time window.

Given two 6DoF sensor streams $A_i$ and $B_i$ for $i = 1, \ldots, t$, define the time series matrices $A_{1:t}$ and $B_{1:t}$ as the $4 \times 4t$ matrices given by

$$A_{1:t} = [A_1 \ldots A_t]$$ \hspace{1cm} (3.2)
and similar for $B_{1:t}$. Define an invariant functional under rigid linkage to be any $\phi : \mathbb{R}^{4 \times 4} \rightarrow \mathbb{R}$ such that, for any homogeneous $4 \times 4$ matrices $Y$ and $X$,

$$B_i = Y^{-1}A_i X, \forall i \in \{1, \ldots, t\} \Rightarrow \phi(A_{1:t}) = \phi(B_{1:t}) \quad (3.3)$$

If a $\phi$ can be found satisfying this property, then identifying a relation between the high dimensional time series $A_{1:t}$ and $B_{1:t}$ is reduced to the problem of discarding any hypotheses satisfying $\phi(A_{1:t}) \neq \phi(B_{1:t})$. This is simply a comparison of two one-dimensional time series, which can be performed with a sliding window.

One such $\phi$ for the simplified case of $A_t$ and $B_t$ representing $3 \times 3$ rotation matrices is that of the angle between the earliest and latest sample:

$$\phi_R(Z_{1:t}) := \angle(Z_t^{-1}Z_1) \quad (3.4)$$
Chapter 3. Sensor Identification

54

Figure 3.3: Invariant Functional (IF) Method - in this hypothetical example, A and B are correlated, while C is not.

where $\angle (Z)$ is the total angle through which $Z \in SO(3)$ rotates, and is given by

$$\angle (Z) := \max_{x \in S^2} \left\{ \cos^{-1} (\langle x, Zx \rangle) \right\}$$

(3.5)

Zhuang et al. provides a proof that this functional is invariant (Zhuang et al. 1994). Figure 3.3 visually outlines the algorithm operating with angular change on hypothetical example signals.

To measure the likeness of two candidate sensor streams with indices $k$ and $l$, the likelihood score was calculated for the invariant functional (IF) matching method according to

$$s_t(k, l) = \exp \left( -|\phi_R(A_{1,t}^k) - \phi_R(B_{1,t}^l)|^2 \right)$$

(3.6)

where

$$|\alpha|_{\theta} := \min\{ |\alpha|, 2\pi - |\alpha| \}$$

(3.7)

accounts for the periodicity of $\phi_R$. This score estimates the likelihood that the two streams are rigidly linked. The angle function $\angle(\cdot)$ was computed from the quaternion corresponding to the argument matrix.
3.1.2 Calibration Error (CE) Method

An alternative method of matching sensor streams is through computation of the transforms $Y$ and $X_{k,l}$ for each possible sensor pairing, followed by comparison of each pairing based on the error given by

$$E = \sum_{t=1}^{N_s} \left\| (A_t X)^{-1} Y B_t \right\|^2_{SO(3)}$$

(3.8)

where

$$\left\| \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \right\|_{SO(3)} := |\angle(R) + \|T\||$$

(3.9)

and $\angle(\cdot)$ is defined by Equation 3.5 and computed from the quaternion corresponding to $R$. Figure 3.4 visually summarizes the process of calibration and discarding any hypotheses with high error.

A collection of queues were used to collect the data for the calibration procedure; one for each sensor pair hypothesis $(k, l)$. The queues had maximum length $N_s \in \mathbb{N}$, with the oldest samples discarded whenever capacity was exceeded. The sample pair $(A^k_t, B^l_t)$ was inserted into queue $(k, l)$ only when at least one of the following was satisfied

$$\left\| (B^l_t)^{-1} B^l_{t-1} \right\|_{SO(3)} > \delta_s$$

$$\left\| (A^k_t)^{-1} A^k_{t-1} \right\|_{SO(3)} > \delta_s$$

where $\delta_s$ is a fixed sample threshold. When all of the queues reached maximum capacity, the robot-world and hand-eye calibration (Shah 2013) was performed $KL$ times to produce a $Y_{k,l}$ and $X_{k,l}$ for each $k$ and $l$. The likelihood score for the

1This is not actually a norm as $\angle(R)$ fails the triangle inequality and multiplicativity due to its periodicity. However, definiteness and positivity are the only properties needed for the purpose of measuring the size of the rotation of a matrix.
Chapter 3. Sensor Identification

Shah (2013) gives

Error:

Figure 3.4: Calibration Error (CE) Method

where the matrix norm $\| \cdot \|_{SO(3)}$ is defined in Equation 3.9. Pairings of sensors which are rigidly linked will have a likelihood score close to 1.

3.1.3 Match Selection and Score Filtering

Each of the above computations produce a set of likelihood scores \( \{ s_t(k, l) : k = 1, \ldots, K; l = 1, \ldots, L \} \) estimating the likelihoods that two given 6DoF sensor streams are rigidly linked. This can be done repeatedly over time, possibly at each frame of the process. Scores are used to rank the possibilities for each \( k \), and at time step \( t \) the highest scoring hypothesis was chosen as the estimated correct hypothesis for sensor \( S_k \) at time step \( t \). For this work, it is assumed that each sensor \( S_k \) is attached to one sensor \( Q_l \) from the other sensor system and no two sensors share the same parent sensor.

Additional filtering of the score was trialled in the form of score accumulation. The cumulative score \( \hat{s}_t(k, l) \) of each hypothesis was updated with each new score value
Chapter 3. Sensor Identification

\( s_t(k, l) \) such that
\[ \hat{s}_t(k, l) = \frac{s_t(k, l)\hat{s}_{t-1}(k, l)}{\sum_{l=1}^{L} s_t(k, l)\hat{s}_{t-1}(k, l)} \] (3.11)
and \( \hat{s}_0(k, l) = 1/L \). The summation term in the denominator ensures the likelihood scores are normalised across the possible matchings for sensor \( k \). That is,
\[ \sum_{l=1}^{L} \hat{s}_t(k, l) = 1 \] (3.12)

*Hypothesis elimination*, where hypothesis \((k, l)\) was discarded when \( \hat{s}_t(k, l) < \epsilon_s \), was also trialled. The likelihood score of any discarded hypothesis was not calculated to minimise computation time.

### 3.2 Simulation Analysis

Before discussing the simulation, some terminology for the random sampling of rigid body transformations must be defined.

#### 3.2.1 Notation for Randomly Generated Transforms

Define the 3D homogeneous normal distribution \( N_{3D}(\nu, \sigma) \) to be the probability distribution of the \( 4 \times 4 \) homogeneous matrices sampled as follows:

- The translation component is sampled from a multivariate normal distribution with covariance \( \Sigma = \text{diag}([\sigma^2, \sigma^2, \sigma^2]) \) and mean zero.
- The rotation magnitude is sampled from a univariate normal distribution with variance \( \nu^2 \) and mean zero.
- The axis of rotation is sampled uniformly from the unit sphere \( S^2 \subseteq \mathbb{R}^3 \).
Define the 3D homogeneous uniform distribution $U_{3D}(\alpha_{max}, x_{max})$ to be the probability distribution of the $4 \times 4$ homogeneous matrices sampled as follows:

- The translation component is uniformly sampled from the ball $B_{x_{max}}(0) \subset \mathbb{R}^3$ centered at zero with radius $x_{max}$.
- The rotation magnitude is sampled from the uniform distribution $U(0, \alpha_{max})$.
- The axis of rotation is sampled uniformly from the unit sphere $S^2 \subset \mathbb{R}^3$.

### 3.2.2 Procedure

A Microsoft Kinect v1 camera was used to record a depth map of a pilot user performing a series of poses. The OpenNI\(^2\) software library version 1.5.7 was used to fit a skeleton, consisting of a collection of six 6DoF rigid body poses, to the depth map. From these poses, two virtual sensors were simulated assuming rigid attachment to exactly one Kinect skeleton rigid body each. Kinect data was preferred over random rigid body motions because it provides measurements which conform to human-plausible trajectories. For each simulation trial, the depth data was replayed with online skeleton fitting and virtual sensors were generated for the duration of the recording with simulation characteristics fixed for each trial.

Simulated characteristics of the virtual sensors included latency, angular noise and translation noise. Each virtual sensor stream was formulated for $k = 1, 2$ as

$$B_{t}^{k} = Y^{-1}A_{t}^{k}G_{t}X_{k}L_{t}^{-1}$$

(3.13)

where $A_{t}^{k}$ is the $k$th Kinect rigid body pose at time step $t'$ and $Y$ and $X_{k}$ are the global and local transforms between the Kinect system and the virtual sensor system. $G_{t}$ and $L_{t}$ are randomly sampled noise matrices corresponding the the measurement

\(^{2}\text{OpenNI is an open source API for depth cameras and other natural input devices (www.github.com/OpenNI/OpenNI (28/4/2018)).}\)
noise of $A^t_k$ and measurement noise of $B^t_k$ respectively. To simulate latency, the indices $t$ and $t'$ were chosen to give a time difference between sample $t$ and $t'$ equal to the desired latency within error\(^3\) of $\pm 16.5\text{ms}$.

The matrix $Y$ was fixed for both virtual sensors. The value chosen was based on measurements from the offline real sensor experiment detailed in Section 3.3.

$$Y^{-1} = \begin{bmatrix}
0.1040 & -0.0023 & -0.9946 & -0.3540 \\
-0.1147 & 0.9933 & -0.0143 & -0.9437 \\
0.9879 & 0.1156 & 0.1030 & 1.2106 \\
0 & 0 & 0 & 1.0000
\end{bmatrix} \quad (3.14)$$

This value for $Y$ is therefore plausible for a pair of living-room scale sensor systems.

Each $X_k$ matrix was randomly generated $\sim U_{3D}(1, 0.1)$ at the start of the simulation. The random matrices were sampled independently such that

$$G_t, L_t \sim i.i.d \sim N_{3D}(\nu, \sigma) \quad (3.15)$$

where $\nu, \sigma \in \mathbb{R}$ were parameters fixed throughout a given simulation. The sensor streams $B^t_k, k = 1, 2$ were matched against all six of the Kinect rigid body sensor streams $A^t_l, l = 1, \ldots, 6$.

### 3.2.3 Results

The simulations were carried out on a 2.5GHz Intel Core i7 CPU with the implementation written in C++ using the Armadillo linear algebra library (Sanderson 2010). For each set of parameter values, a single simulation trial was performed. Each trial was based on 52 seconds of recorded Kinect depth map as described in Section 3.2.2. The main measurement of performance was the mean matching performance, equal

\(^3\)The latency can only be adjusted discretely and is thus limited by the time between frames of the captured depth images (33ms for 30Hz sample rate).
Figure 3.5: Simulation sample-noise and sample-timing analysis for the calibration error (CE) method. The heat map displays the fraction of correct identifications of the simulated sensors as a function of angular noise and number of samples averaged over the trial period (left). The sample-timing linear least squares relationship was $\tau_m(N_s) \approx (0.0018 \times N_s + 0.033)\text{ms}$ (right).

to the number of correct guesses divided by the total number of guesses over the trial. Because there were two virtual sensors, two guesses were made whenever a new sample was recorded and there were $N_s$ samples available, leading to around 2000 guesses over each 52 second trial.

Figure 3.5 profiles mean matching performance as a function of number of samples and noise level for the CE method. As expected, increasing the number of samples increases performance with noisy data. A graph of the computation time per hypothesis for the CE method as a function of number of samples $N_s$ is also included in Figure 3.5. Over the interval $[0, 200]$, the matching computation time per hypothesis $\tau_m$ is close to linear, with least squares fit yielding

$$\tau_m(N_s) \approx (0.0018 \times N_s + 0.033)\text{ms}$$ \hspace{1cm} (3.16)

The IF method always operates with just two samples, the latest sample and the earliest sample, and so runs in constant time after the initial samples have been gathered. Additional samples do not effect the IF method performance.

Figure 3.6 shows the noise and latency performance for the IF and CE methods.
Figure 3.6: Simulated noise performance analysis of the calibration error (CE) and invariant functional (IF) matching algorithms. The mean matching performance is equal to the fraction of correct identifications of sensor dependencies over a 52 second simulation. Note that since the IF method only compares rotation data, it is expected to perform well independent of positional noise.

in the form of heat-maps of mean matching performance. These experiments used \( N_s = 100 \) and \( \delta_s = 0.01 \) for the CE method, leading to an average total CE matching computation time of 2.57ms, varying with standard deviation 0.35ms. Neither accumulation filtering nor elimination was used for these experiments.

The IF method yielded a mean per-frame computation time of 0.14ms varying with standard deviation of 0.06ms. The simulation results suggest that sensors with sufficiently low angular measurement noise (\( \nu \leq 0.5^\circ \)) should be identifiable at a rate of over 60% by the CE method, even under conditions of high positional noise of \( \sigma = 10\text{cm} \). Note that the rate of correct matching given random selection is
\[ \frac{1}{6} \approx 0.167 \]. The simulation shows that the CE method has a small advantage over the IF method in terms of robustness to latency. The CE method also performs better than the IF method when there is no displacement noise present. However, that advantage diminishes when displacement noise is present. This observation is explained by the CE method discriminating between similar sensors due to its complete use of the available data. This characteristic also implies that the CE method is susceptible to noise in the translation components of the sensors. The IF method does not include position information and so is unaffected by the position error, but is also able to distinguish the sensors less effectively.

### 3.3 Case Study: Offline Experiment With Real Sensors

A Microsoft Kinect v1 depth camera was used to record depth data of a pilot user performing a series of poses including waving of each arm, lifting of each leg and walking. The poses included facing the Kinect and facing sideways to the Kinect while walking. Each limb was also moved independently of the others at least once during the recording. Meanwhile, two 6DoF rigid bodies were tracked with an OptiTrack Flex-13 tracking system with OptiTrack’s Motive rigid body tracking software\(^4\). The Kinect was also tracked with the OptiTrack system to provide the ground truth transform to the Kinect reference frame for qualitative verification of the collected data (Figure 3.8, left). Example depth maps measured by the Kinect are shown in Figure 3.7 with the sensor ground truth, OpenNI tracked skeleton and example matching results. The tracked rigid bodies were distinguishable but unlabeled and attached to the participants shoulder (Figure 3.8, right) and opposite knee, corresponding to the knee and shoulder bones of the skeleton identified by the Kinect.

\(^4\)Gold standard motion capture system based on retro-reflectance of infrared light from markers (see Table 1.1 and www.optitrack.com (28/4/2018)).
Figure 3.7: Data visualisation for the offline real sensor case study. Left shows the OptiTrack marker coordinate frames, transformed into the Kinect reference frame using ground truth (see Figure 3.8), and the detected user. A rigid body skeleton is generated with the OpenNI software platform version 1.5.7 (centre). The candidate Kinect rigid bodies have their corresponding coordinate frames drawn at their origins. The two 6DoF measurements from the sensors are matched with the one of six Kinect rigid body streams in real time (right). At this instance, the left knee (red) has been correctly identified, while the right shoulder has been misidentified as the left shoulder (green).

A skeleton pose was extracted from the recorded depth map using OpenNI software library version 1.5.7. The unlabeled sensors were matched to the Kinect skeleton rigid bodies using both the IF (Section 3.1.1) and CE matching (Section 3.1.2) techniques. The matching algorithms were tested with and without both score accumulation and hypothesis elimination, described in Section 3.1.3. All computations, including skeleton tracking and sensor matching, were performed in real time at ≥30Hz on a recorded depth map. For each frame, the two sensors were matched with one of six rigid bodies tracked by the Kinect: upper arms, lower arms and lower legs (below the knees). The resulting identifications were counted as correct or incorrect to calculate the mean matching performance for each sensor.

It was found that a latency of 76ms existed between the OptiTrack motion capture stream and the Kinect stream. Until this was accounted for, the algorithm
Figure 3.8: The motion capture setup for measuring the Kinect reference frame for ground truth verification (*left*) and an example sensor placed on the shoulder and tracked by an OptiTrack system (*right*).

Table 3.1: Calibration error (CE) method - fraction of correct identifications for real sensor experiment

<table>
<thead>
<tr>
<th>Rigid Body</th>
<th>No Score Accumulation</th>
<th>Score Accumulation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Knee</td>
<td>Shoulder</td>
</tr>
<tr>
<td>No Elimination</td>
<td>0.74</td>
<td>0.40</td>
</tr>
<tr>
<td>Elimination</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.2: Invariant functional (IF) method - fraction of correct identifications for real sensor experiment

<table>
<thead>
<tr>
<th>Rigid Body</th>
<th>No Score Accumulation</th>
<th>Score Accumulation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Knee</td>
<td>Shoulder</td>
</tr>
<tr>
<td>No Elimination</td>
<td>0.55</td>
<td>0.08</td>
</tr>
<tr>
<td>Elimination</td>
<td>0.03</td>
<td>0.95</td>
</tr>
</tbody>
</table>

performed no better than chance. The parameters used in this experiment were $N = 100$ samples gathered before calibration was performed with a difference of $\delta_s = 0.01$ required between successive samples. When elimination was included, the elimination threshold was $\epsilon_s = 0.1$.

Table 3.1 shows the results for the CE method. The accumulation filter increases performance by weighting previously incorrect hypotheses as less likely. However, this means it cannot recover as quickly from a change in sensor placement. Elimination with CE is very effective for this particular experiment. Table 3.2 shows results for the IF method. Elimination without accumulation for the IF method resulted in the knee sensor being identified with the wrong knee. This demonstrates how the IF method has more trouble distinguishing similar hypotheses compared to the CE method. The Kinect suffers from ambiguity in the 6DoF limb measurements and this instability has caused the IF method to perform poorly when elimination is used. The CE method is less susceptible to the instability because it uses more samples in this trial.

The best tracking result without elimination was the CE method with score accumulation. The worst matching score between the two sensors in this case was 76%. This is approximately 4.5 times better than random selection, which would score $\frac{1}{6} \approx 16.7\%$, as there are six choices for each sensor identity. Even the worst result for the CE method scored 40%, or 2.5 times better than random. Though direct comparison cannot be made, this result is similar to the results achieved by Bahle et al. (2013). Our results used 6DoF sensors rather than 3DoF sensors and our study does not have comparable statistical significance.

The observation that the knee sensor matched in general better than the shoulder is explained through inspection of the skeleton which had been fitted to the depth map. The shoulder had increased ambiguity compared to the knee in that particular trial, leading to strange behaviour of the orientation of the upper arm and shoulder. As an example of instability, Figure 3.9 shows the rigid body measured for the knee
in a separate trial swapping orientation by 180° discontinuously. It is also possible that the assumption of rigidity breaks down for the data obtained from the Kinect and OpenNI.

![Image](image.png)

**Figure 3.9:** The OpenNI Kinect skeleton tracking software has several instabilities which make it unsuitable for rigid body identification. For example, the knee orientation is ambiguous when outstretched; here the rigid body for the knee flips 180° discontinuously.

### 3.4 Case Study: Interactive Demonstration with Real Sensors

To verify the approach further, a third sensor system was created using Sony’s Playstation Move (PS Move) controller and a Playstation Eye (PS Eye) webcam. The open source PS Move API was used for connecting to, calibrating and tracking the PS Move controller with 6DoF (Pearl 2012). The PS Move controller includes an accelerometer and gyroscope for orientation tracking. The positional tracking is performed using a webcam by tracking the screen position and size of a spherical coloured marker attached to the controller. The PS Eye webcam was used for the positional tracking. The PS Move tracking system was used to test the proposed

---

5PS Move: UPC 711719805809. PS Eye: UPC 711719804703.
techniques by matching the PS Move pose with that of motion capture markers from the OptiTrack motion capture system discussed in Section 3.3.

A motion capture marker was used to track the position of the PS Eye camera so that the pose of the motion capture markers could be visualised on the PS Eye video feed alongside the PS Move tracking result. Figure 3.10 shows the tracking results for an OptiTrack marker and for the PS Move controller from the perspective of the PS Eye.

Figure 3.11 shows an example where the PS Move sensor is matched with one of two motion capture markers. The system was modified so that after all but one hypothesis was eliminated (according to Section 3.1.3), the final hypothesis could also be eliminated if its raw score dropped below the threshold $\varepsilon_s$. If this happened, all samples would be erased, and the matching procedure was reset. This allowed the system to recover when the sensor dependencies changed.

The system worked extremely well using either of the CE method or the IF method. Parameters which worked best were $N_s = 10$, $\delta_s = 0.1$ and $\varepsilon_s = 0.1$. It is possible to attach one sensor to the PS Move in an arbitrary rigid configuration and have it be correctly identified within 1-2 seconds, provided the controller is moved (see Figure 3.12). The amount of movement required to match here can be seen in

**Figure 3.10:** Tracking results for an OptiTrack marker (*left*) and a PS Move controller (*right*) from the perspective of the PS Eye camera. Note that the OptiTrack marker has been transformed to the PS Move sensor space for visualisation only; all computations use raw data in the native space.
Figure 3.11: Two markers (top) were matched with a PS Move controller. Moment of attachment (bottom left) and moment of identification of the rigid link (bottom right; indicated by the white sphere) for the CE method. Typical time between the two events is 1-2 seconds, depending on the movement of the user (Figure 3.12). The amount of movement required to match here can be seen to be a $90^\circ$ rotation and a translation of about 15cm. The IF method achieved similar results.

Figure 3.11 to be a $90^\circ$ rotation and a translation of about 15cm. Limitations include the assumption that exactly one sensor is attached to the controller. This results in false positives occasionally when neither of the sensors are attached and one false negative and instability when both are attached. Table 3.3 summarises the error types for the algorithm. This could easily be amended by using a slightly modified approach to elimination where sensors are eliminated based on their absolute score rather than their relative score.

Live Kinect Application

A live application of the system was created with the Microsoft Kinect, OpenNI and NiTE 2.2\textsuperscript{6}. The rigid bodies from the NiTE user tracker are matched against a

\textsuperscript{6}NiTE open source tracking middleware: \url{openni.ru/files/nite/} (28/4/2018)
Table 3.3: Confusion matrix of attachment errors for matching one marker to one of two possible locations A and B

<table>
<thead>
<tr>
<th>Truth</th>
<th>Predicted Attachment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>√</td>
</tr>
<tr>
<td>B</td>
<td>Unlikely</td>
</tr>
<tr>
<td>A∩B</td>
<td>50%</td>
</tr>
<tr>
<td>{}</td>
<td>50%</td>
</tr>
</tbody>
</table>

Figure 3.12: An example trace where two motion capture markers are attached sequentially, one at a time, to the PS Move sensor. Here it is assumed that when the ground truth distance between a marker and the PS Move (top) is less than 30cm, the two are rigidly attached. The matching result for the CE method (bottom) can be seen to match well with the ground truth estimator (centre). The broken and dotted lines indicate the times at which a sensor is attached and removed respectively.

The system demonstrates that the CE method is feasible for real time identification of sensor placement on a complex articulated body. In this case, the marker is assigned one of 10 locations on the body, including upper and lower arms and legs, body and head. Time taken to identify the location of the marker ranges from around 5 seconds on highly mobile
Chapter 3. Sensor Identification

body parts like the forearm, to around 15 seconds when placed on less mobile parts like the shoulders or thighs. Figure 3.13 shows some example screen captures of the program identifying a marker placed on the shoulder. The limitation of this system is centrally the quality of the tracked skeleton given by OpenNI: for ambiguous depth images it is impossible to correctly identify sensor placement.

![Figure 3.13: An example of real-time interactive sensor matching for the Microsoft Kinect. The user is tracked with OpenNI and NiTE software libraries and the sensor’s true location (left shoulder) is drawn as 3D coordinate axes (left). Hypotheses are shown as small white spheres placed at the origin of each articulated body part, with the single larger sphere representing the most likely hypothesis (centre). After approximately 10 seconds of tracking, the correct location is determined and most other hypotheses are eliminated (right).](image)

3.5 Reflection and Future Work

The methods examined in this work ignore some potentially useful information. The first is that the matrix $Y$ is actually constant for sensors belonging to the same pair of sensor systems, whereas matching assumed that $Y$ would be different for each
sensor pair. A significantly more sophisticated approach would be required to utilise this information because the possible match space increases from $KL$ hypotheses to $L^K$ hypotheses, where $K$ is the number of unidentified sensors and $L$ is the number of possible identities. This increase occurs because in the first case you must explore $L$ possibilities for each sensor independently, while in the second you must explore $L^K$ possibilities for the vector space of sensor identities $(I_1, \ldots, I_K)$.

A limitation of the CE method is that it cannot match sensors between left and right handed coordinate systems. This can easily be compensated for, but requires the user to understand and identify the handedness of each system. Further development to account for this case would require modification of the calibration methods to allow $\det(X), \det(Y) = \pm 1$, rather than restricting to $+1$, as is the case with each of the discussed methods (i.e. (Dornaika and Horaud 1998; Li et al. 2010; Shah 2013; Zhuang et al. 1994)). For example, within the method used in our algorithm, there is a step which restricts the determinants to be $+1$ (Shah 2013). It may be possible to simply compute the case of $\det = -1$, and choose whichever has lesser calibration error.

A separate static noise analysis placed the noise values for the Kinect rigid bodies at $\nu \approx 1-2^\circ$ and $\sigma \approx 10-20\text{cm}$. Comparing these values to Figure 3.6, mean matching performance near 0.5 would be expected. However, these heat maps assume that both sensors have the same values for $\nu$ and $\sigma$. A static noise analysis for the motion capture system yielded $\nu \approx 0.07^\circ$ and $\sigma \approx 0.1\text{mm}$. This suggests the low noise level of the motion capture system has improved the results to be better than expected for the offline case study (Section 3.3). Static noise analysis for the PS Move sensor, positioned 2m from the tracking camera, yielded $\nu \approx 0.5^\circ$ and $\sigma \approx 20-50\text{cm}$, with position noise increasing with distance from the camera. This low rotation noise coupled with the low noise of the motion capture system gives very good matching results for two sensors even with relatively few samples ($N_s = 10$).

In summary, two techniques were analysed for their ability to identify rigidly linked
6DoF and 3DoF sensors. The invariant functional (IF) matching method compares the angle which each rigid body has rotated through in during a sliding window. The calibration error (CE) technique uses the error from a calibration algorithm, originally developed for industrial robots, to assess the likelihood that two sensors are rigidly linked. The methods perform well in simulation and a noise and latency analysis reveals that although the CE method is more robust, it is also slower and significantly more difficult to implement. However, both methods are fast enough to perform real time matching, with the largest computation times on the order of a few milliseconds. It is recommended that IF is used for small systems, while CE is used when many sensors must be identified.

A case study was performed with offline data from a Microsoft Kinect v1 and an OptiTrack motion capture system. Considering the noise and instability inherent in extracting 6DoF rigid bodies from 3D depth data, excellent matching capability was achieved. The best result performed at least 4.5 times better than chance. A real-time interactive demonstration was also developed, demonstrating responsive and accurate sensor matching even when dealing with a large number of possible sensor locations. This allows the system to succeed in a very important case: locating sensors on a human skeleton, such as the skeleton tracked by the Microsoft Kinect and similar devices. This chapter thus provides a solution to the first problem identified in Chapter 2, that of sensor identification. Next, Chapter 4 details early work toward a solution to the problem of determining the latency between two signals. Latency calibration is an important step to perform before sensor identification can be performed, and Chapter 4 works toward automating this step.
Chapter 4

Latency Calibration

Chapter 3 identified latency as a significant obstacle impeding identification of dependence relations amongst sensor streams. Relative signal latencies must also be known for the calibration step of the ambient sensor fusion pipeline. Previous research on determining latency between two signals relies on knowledge of the function mapping one signal to the other. The approach proposed in this chapter does not require a model to be specified. Provided that the two sensor streams are related by a sufficiently continuous function, the latency can be determined. This continuity condition is expected to hold for most sensors measuring 3D pose reliably, but would likely fail in conditions where tracking exhibits blind spots or drifting. The key intuition behind the proposed approach is that when one signal changes, the other signal should change simultaneously and consistently. The approach generalises to high dimensional data, such as 6DoF data and even raw video time series, but the time complexity grows linearly with dimension. Simulation analysis has revealed that the algorithm is effective provided the sampled data is at least partly repetitious. An experiment was performed to successfully calibrate the latency between the pose of a virtual camera and the camera’s video stream generated by an OpenGL graphics engine. Only down-sampling operations were applied to the video data; no computer vision is necessary. The proposed approach is still exploratory, but the
results indicate this type of approach may have broad application in the future to computing latency and correlations between complexly related time-series.

4.1 Related Work

Identifying the latency between measurements from different sensor systems is an important step in sensor fusion (Kaempchen and Dietmayer 2003). Some sensor identification algorithms are robust to latency, such as frequency domain analysis. For example, Lester et al. 2004 found that up to 1 second of delay had minimal effect on their success rate provided enough data was sampled (around 13 seconds for 1 second of delay). Strasser 2004 used a similar technique to determine relationships between two positional sensors. However, frequency domain approaches apply only to signals which are invariant between sensor systems, such as angular or linear velocity magnitude. These models do not use all the data available and the methods don’t extend to sensors that share no invariants.

One approach to directly estimating latency is to include the delay as a variable in the state space of a Kalman filter (Fox 2006). Similarly, Nilsson and Händel 2010 treat the time-stamp of a sensor as a noisy observation and filter it with a Kalman filter to improve measurement ordering and synchronisation. However, these approaches require transformational models from one sensor stream to another. The proposed approach requires minimal information about the dependence relationship - simply that one signal is a (sufficiently) continuous function of the other.

A different approach involves the maximisation of the cross-correlation, defined as

\[ r_{xy}(\tau) = E\{X(n)Y(n + \tau)\} \]  \hspace{1cm} (4.1)

where \(X, Y\) are the one dimensional signals of interest such that \(Y(n) = X(n-l) + \delta_n\), \(\delta_n\) is a noise term, and \(E\) is the expectation operator (e.g. Nikias and Pan (1988)).
These types of algorithms have long been used in spatial audio analysis, where a similar audio signal arrives at multiple spatially separated microphones delayed by distance traveled (Knapp and Carter 1976; He et al. 2017). However, these approaches use direct comparison of the signals, which fails if one signal is some unknown, possibly non-linear, function of the other. The proposed approach avoids direct comparison of the signals, rather it compares times when each sensor changes in its own measurement space.

4.2 Model-less Latency Between Two Multi Dimensional Signals

Let $\mathbf{x}(t) \in \mathbb{R}^n$ for some $n \in \mathbb{N}$ and all $t \in [0, T] \subset \mathbb{R}$. Now let $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be a function for some $m \in \mathbb{N}$ such that the dependent curve $\mathbf{y}(t)$ is defined by

$$\mathbf{y}(t) = F(\mathbf{x}(t)) + \epsilon_t$$

(4.2)

where $\epsilon_t \overset{i.i.d.}{\sim} N(0, \Sigma_y)$, $\Sigma_y \in \mathbb{R}^{m \times m}$ represents measurement noise. Now, suppose that some positive or negative latency $l \in \mathbb{R}$ is introduced and only the offset signal $\tilde{\mathbf{y}}(t) = \mathbf{y}(t + l)$ and a noisy independent signal $\tilde{\mathbf{x}}(t) = \mathbf{x}(t) + \delta_t$ are accessible, where $\delta_t \overset{i.i.d.}{\sim} N(0, \Sigma_x)$. The objective of latency correction or calibration is to determine the parameter $\hat{l}$ so that the original signal can be reconstructed $\mathbf{y}(t) = \tilde{\mathbf{y}}(t - \hat{l})$.

The central difficulty comes from the separate domains for the independent and dependent variables $\mathbf{x}$ and $\mathbf{y}$. These spaces may even have different dimension. Prior approaches require direct comparison of the independent and dependent variables (see Section 4.1). But these approaches require knowledge of $F$, which is impossible to determine without already knowing the latency! The proposed approach determines $l$ without a model for $F$. After $l$ is determined, a model for $F$ can be created.
Figure 4.1: Visualisation of the proposed latency calibration method. The approach searches for values of latency $l$ such that close points in $x$ correspond to close points in $y$. The algorithm uses the largest disagreement over all close points in $x$ as the objective function.

The proposed solution rests on the modest assumption that $F$ is a Lipshitz continuous function on the relevant domain $x([0, T]) \subset \mathbb{R}^n$. For the purposes of this work, define $F$ to be Lipshitz continuous on $X \subset \mathbb{R}^n$ if there exists some fixed number $L > 0$ such that, for all $x_1, x_2 \in X$,

$$
\|F(x_1) - F(x_2)\| \leq L \|x_1 - x_2\| \quad (4.3)
$$

Therefore, if $F$ is Lipshitz continuous then for two time points $t_1, t_2 \in [0, T]$,

$$
\|y(t_1) - y(t_2)\| \leq L \|x(t_1) - x(t_2)\| \quad (4.4)
$$

From this inequality there must be a sub-linear relationship for the distances between points $x$ and simultaneous points in $y$. Intuitively, this means that points close together in $x$ space are proportionately close in $y$ space. However, the latent function $\tilde{y}(t)$ can and will violate this rule when naively compared to $\tilde{x}(t)$ without taking latency into account.
Chapter 4. Latency Calibration

If $F$ is Lipshitz continuous, then counting (in the discrete time case) the instances where

$$\|\hat{y}(t_1 + l) - \hat{y}(t_2 + l)\| > L \|\hat{x}(t_1) - \hat{x}(t_2)\|$$

(4.5)

will give a measure of the error for the estimated latency $l$, because when the latency estimate is correct, this condition will never be met. However, estimating $L$ is no simple task. $L$ can vary widely depending on the context and signals involved. So generalising this idea, a continuous error function is proposed. The overarching idea is to have high error whenever $\|\hat{x}(t_1) - \hat{x}(t_2)\|$ is small but $\|\hat{y}(t_1 + l) - \hat{y}(t_2 + l)\|$ is large.

Consider the function

$$E(l) = \max_{(t_1, t_2) \in S_{x,\delta}} \left\{ (\delta - \|\hat{x}(t_1) - \hat{x}(t_2)\|) \|\hat{y}(t_1 + l) - \hat{y}(t_2 + l)\| \right\}$$

(4.6)

where

$$S_{x,\delta} = \{(t_1, t_2) \in [0, T]^2 : \|x(t_1) - x(t_2)\| < \delta\}$$

(4.7)

is the set of close points in the independent signal which are closer than $\delta > 0$ in $X$. If $l$ is equal to the true latency, then when $\|x(t_1) - x(t_2)\| < \delta$, $\|\hat{y}(t_1 + l) - \hat{y}(t_2 + l)\| < L\delta$. Thus as $\delta \to 0$, $(\delta - \|x(t_1) - x(t_2)\|) \|\hat{y}(t_1 + l) - \hat{y}(t_2 + l)\| \to 0$. This property will not hold for at least some $t_1, t_2$ whenever $l$ is not chosen correctly. Consider the set $S_{x,\delta}$ of $(t_1, t_2)$ where $\|x(t_1) - x(t_2)\| < \delta$. If $l$ is chosen incorrectly, then it is highly likely that for at least one $(t_1, t_2) \in S_{x,\delta}$ that $\|\hat{y}(t_1 + l) - \hat{y}(t_2 + l)\| > L\delta$.

Figure 4.1 demonstrates this condition visually and demonstrates how adjusting for the latency reduces the error $E(l)$ by reducing $\|\hat{y}(t_1 + l) - \hat{y}(t_2 + l)\|$. $E(l)$ represents the worst example of this property over all the close points $S_{x,\delta}$, and thus $E(l)$ is a measure of error of the latency estimate $l$. The value of $l$ can then be found with a line search for minimum error $E(l)$ between two reasonable starting values. This analysis uses $[-T/4, T/4]$ sec for the latency search range. Algorithm 4.1 details the algorithm used to compute the latency between two data streams. The following sections explore the application of this approach to simulated data, including
determining the latency between a video stream and the pose of the camera which recorded the video.

4.3 Application to Simulation

The efficacy of the proposed method is not obvious simply from the mathematics due to the large amount of variation encoded in the path of the independent variable \( x(t) \) and the unknown function \( F \). Therefore, a set of simulations was performed to assess the utility of the proposed method. First, the methodology is described.

4.3.1 Generating Simulation Data

In order to simulate a family of functions for \( x \) and \( F \), a procedurally generated Fourier sum was used. The \textit{procedural function} \( \phi : \mathbb{R}^N \rightarrow \mathbb{R}^M \) of order \( K \in \mathbb{N} \), period \( s \in \mathbb{R} \), input dimension \( N \in \mathbb{N} \) and output dimension \( M \in \mathbb{N} \) is defined as a sum over a generalised Fourier-like series. Let \( a_{i,j,k}, b_{i,j,k} \overset{i.i.d.}{\sim} U[-1, 1] \) be independently uniformly sampled values for \( i = 1, \ldots, N \), \( j = 1, \ldots, M \), \( k = 1, \ldots, K \). Then \( \phi \) is defined component-wise by

\[
\phi_j(x) = \sum_{k=1}^{K} \sum_{i=1}^{N} \left( 1 - \frac{k}{K} \right) \left[ a_{i,j,k} \sin \left( \frac{2\pi x_i}{s} \right) + b_{i,j,k} \cos \left( \frac{2\pi x_i}{s} \right) \right] \tag{4.8}
\]

Note that this function is periodic with period \( s \).

A simulation trial involved generating \( x : \mathbb{R} \rightarrow \mathbb{R}^n \) and \( F : \mathbb{R}^n \rightarrow \mathbb{R}^m \) as described above with orders \( K_x, K_y \) and periods \( s_x, s_y \) respectively. For a given \( x \) and \( F \), the ‘observed’ functions were then sampled for \( t = 0, \ldots, N \) such that \( f = 100 \) frames
Algorithm 4.1: Latency Calibration

Data: Discrete time functions \( x(t) \in \mathbb{R}^n, y(t) \in \mathbb{R}^m \) for \( t = 1, \ldots, N \) and \( N \in \mathbb{N} \)

Result: Latency \( l \) between \( x(t), y(t) \)

1 \begin{align} &\text{// Initialise matrix of norms between pairs of points} \\
&\quad D_x, D_y \leftarrow \text{MatrixOfOnes}(N,N); \\
&\quad \text{for } i = 1, \ldots, T \text{ do} \\
&\quad \quad \text{// Pre-compute norms for all } x \\
&\quad \quad D_x[i,j] \leftarrow \|x(i) - x(j)\|; \\
&\quad \text{end} \\
&\text{// Pick } \delta \text{ such that } 1\% \text{ of closest points are in } S_{x,\delta} \\
&\quad \delta \leftarrow D_x.\text{getPercentile}(1\% + 100\%/N); \\
&\quad \text{// Record time pairs with close } x \text{ values} \\
&\quad S_{x,\delta} \leftarrow \{\}; \\
&\quad \text{for } i = 1, \ldots, T \text{ do} \\
&\quad \quad \text{for } j = i+1, \ldots, T \text{ do} \\
&\quad \quad \quad \text{if } D_x[i,j] < \delta \text{ then} \\
&\quad \quad \quad \quad S_{x,\delta} \leftarrow S_{x,\delta} \cup \{(i,j)\}; \\
&\quad \quad \text{end} \\
&\quad \text{end} \\
&\text{// Initialise set of latency values to try} \\
&\quad L \leftarrow \{-T/4, \ldots, -1, 0, 1, \ldots, T/4\} \subset \mathbb{N}; \\
&\quad l_{\text{best}} \leftarrow 0; \\
&\quad E_{\text{best}} \leftarrow \infty; \\
&\quad \text{for } l \in L \text{ do} \\
&\quad \quad E_l \leftarrow \{\}; \\
&\quad \quad \text{for } (i,j) \in S_{x,\delta} \text{ do} \\
&\quad \quad \quad \text{// Calculate } y \text{ distance and cache} \\
&\quad \quad \quad \quad \text{if } D_y[i+l, j+l] = -1 \text{ then} \\
&\quad \quad \quad \quad \quad D_y[i+l, j+l] \leftarrow \|y(i+l) - y(j+l)\|; \\
&\quad \quad \quad \text{end} \\
&\quad \quad \quad \text{// Add error (Equation 4.6) to list} \\
&\quad \quad \quad E_l \leftarrow E_l \cup \{\delta - D_x[i,j] \cdot D_y[i+l, j+l]\}; \\
&\quad \text{end} \\
&\quad \text{end} \\
&\quad \text{end} \\
&\quad \text{end} \\
&\text{end} \\
&\text{return } l_{\text{best}} \\
\end{align}
per second are sampled over the time interval \([0, T]\), according to 

\[
\tilde{x}(t) = x(t) + \delta_l \\
\tilde{y}(t) = F(x(t + l)) + \epsilon_l
\]  

\((4.9)\) \hspace{1cm} \((4.10)\)

where \(l\) is a latency value randomly selected \(l \in [-T/4, T/4]\). The noise values are sampled from gaussian distributions \(\delta_l \sim \mathcal{N}(0, \sigma_x^2 I_n)\) and \(\epsilon_l \sim \mathcal{N}(0, \sigma_y^2 I_m)\), \(\sigma_x, \sigma_y \in \mathbb{R}\). The performance of the proposed latency calibration algorithm was analysed over various values for the set of parameters

\[(n, m, l, \sigma_x, \sigma_y, K_x, K_y, s_x, s_y, T)\]

\((4.11)\)

### 4.3.2 Simulation Results

Figure 4.2 (top left) shows the form of a typical pair of dependent data streams generated by the procedure described in the previous section with \(n = m = 1\). Figure 4.2 (top right) shows the error values for latencies between \([-0.25, 0.25]\]. The minimum error (defined in Equation 4.6) is marked by a vertical red line, and for this data it coincides with the black dashed line indicating the true value of the latency. Figure 4.2 (bottom left) plots \((x(t), y(t))\) as a curve in \(\mathbb{R}^2\) without latency calibration. Figure 4.2 (bottom right) plots \((x(t), y(t - \hat{l}))\) as a curve with the estimate for latency \(\hat{l}\) included. It is clear that this graph represents a function, as there is one value of \(y\) for each \(x\). This function is of course \(F\). The algorithm can be conceptualised as an optimisation of \(\hat{l}\) to maximise the function-likeness of \(F\). The procedure generalises to higher dimensions as shown in Figures 4.3 and 4.4, where 2 and 10 dimensional examples are shown. Such high dimensional signals are common, for example, a 9DoF IMU (accelerometer, gyroscope and magnetometer) or even a collection of three 6DoF sensors tracked by a 3×6DoF VR system (total dimension 18). However, high dimensional signals fail more often, for example in
Figure 4.2: Example of latency calibration applied to one dimensional signals $x, y$ procedurally generated as described in Section 4.3.1, with parameters $(n = 1, m = 1, \sigma_x = 0, \sigma_y = 0, K_x = 10, K_y = 2, s_x = 2, s_y = 5, T = 1 \text{sec})$. The latency was successfully calibrated to within one frame of error. (top left: signals over time; top right: latency search; bottom left: signals before calibration of latency; bottom right: signals after calibration of latency)

Figure 4.5. The performance of the algorithm is also shown for a noisy 1D case in Figure 4.6.

Figure 4.7 plots latency calibration failure rate for the proposed algorithm as a function of $x$ and $y$ dimension. For each dimension pair $(n, m)$ for $n, m = 1, \ldots, 20$, 20 trials were performed. For each trial, procedural functions were generated for $x$ and $F$ with random latency $l \in \left[-\frac{T}{2}, \frac{T}{2}\right]$ and no noise $(\sigma_x = \sigma_y = 0)$. The latency algorithm was run for each trial and the fraction of successful calibrations was computed as the number of trials to successfully determine the true latency to within half a frame of error. The failure rate, the logical compliment of the success rate, was plotted as a heat map over dimension. It can be seen that performance
Figure 4.3: Example of latency calibration applied to two 2 dimensional signals $x, y$ procedurally generated as described in Section 4.3.1, with parameters $(n = 2, m = 2, \sigma_x = 0, \sigma_y = 0, K_x = 10, K_y = 2, s_x = 2, s_y = 5, T = 1 \text{ sec})$. The latency was successfully calibrated to within one frame of error.

is mostly independent of the embedded space $y$. This is because the topology of the curve is still homeomorphic to the $x$ space, and so the problem is topologically equivalent to identifying the latency in one dimension. Performance degrades as the dimension of $x$ increases, likely due to the increasing complexity of the independent curve. In particular, it is expected that performance would be significantly degraded when the dimension of $y$ is less than the dimension of $x$, as this results in higher likelihood of sampling non-injective region of $F$. That is, there is a higher likelihood that one value of $y$ corresponds to multiple values of $x$ for $t \in [0, T]$.

The performance of the algorithm as a function of the noise levels $\sigma_x, \sigma_y$ in the signals was also examined in simulation. For each pair of values for $\sigma_x, \sigma_y$, twenty random functions were generated for $x, F$ and $y$ as described previously, and the latency
Chapter 4. Latency Calibration

Figure 4.4: Example of latency calibration applied to two 10 dimensional signals $x, y$ procedurally generated as described in Section 4.3.1, with parameters $(n = 10, m = 10, \sigma_x = 0, \sigma_y = 0, K_x = 10, K_y = 2, s_x = 2, s_y = 5, T = 1 \text{ sec})$. The latency was successfully calibrated to within one frame of error.

The algorithm was run for each set of generated parameters. The fraction of correctly determined latencies was used to produce a heatmap indicating the robustness to noise of the algorithm (Figure 4.9). Note that in both the one and two dimensional cases, the algorithm is more sensitive to noise in the dependent signal $y$.

The simulations show that there are a large range of functions with which the proposed method is effective. However, this is by no means exhaustive, and accurate measures for the suitability of two data streams is not obvious. Future research should include work toward analysis of candidate sensor streams. The next section applies the proposed method to a key use case: calibrating the latency in a video signal compared to tracking data.
Chapter 4. *Latency Calibration*

4.4 Application to Video Data

An experiment was performed to assess the effectiveness of the latency calibration algorithm on more complex data. A common use-case for latency calibration was identified: determining the latency between a camera and a tracking device attached to the camera. A simulation was performed in which a scene was rendered with OpenGL. The camera was moved using standard mouse and keyboard controls, and the camera pose and corresponding rendered image was recorded at each frame. The scene was static, with basic shadows, specular/diffuse/ambient lighting and albedo texturing of two objects within a large room. Position and orientation of the virtual
Figure 4.6: Example of latency calibration applied to two 1 dimensional signals $x, y$ procedurally generated as described in Section 4.3.1, with parameters $(n = 1, m = 1, \sigma_x = 0.05, \sigma_y = 0.05, K_x = 10, K_y = 2, s_x = 2, s_y = 5, T = 1 \text{ sec})$. The latency was successfully calibrated close to the true value (black dashed line), though some error is present.

camera was recorded in Euclidean and quaternion format:

$$x(t) = [p_x, p_y, p_z, q_w, q_x, q_y, q_z]$$  \hspace{1cm} (4.12)  

The dependent signal $y(t)$ was simply the greyscale images rendered at each frame, downsampled from the original size of 500×500 pixels. The downsampling was performed by replacing blocks of pixels with a single pixel with brightness equal to the mean of the pixels within the block. In the case that the desired dimension $m \times n$ of the image did not divide the original resolution evenly, the remainder was simply discarded. The performance of the latency calibration was tested for several down-sampled image sizes. A latency of 10 frames was artificially added to the data
Figure 4.7: Performance of latency calibration with varying dimension of the signals. For each value, 20 trials were performed each with random latency and procedural functions. Here, $T = 1$ second, and the periods of the procedural functions were $s_x = 0.5$, $s_y = 5$. Strong performance is achieved since the independent signal $x(t)$ is periodic with period less than the sample time $T$, and so there are many high quality close points in $x(t)$.

Three key example scenarios were tested with the simulated camera setup. The rotation example involved only rotation of the camera in the pitch and yaw axes performed by standard mouse-look controls. The translation example involved translating the camera without rotation. The motion example involved unstructured translation and rotation of the virtual camera. In each example, the camera was controlled by the researcher and an attempt was made to create a self-intersecting curve in the camera pose space. Table 4.1 shows the first frames, example middle frames, and the end frames of the videos captured. Video was captured for 3 seconds after recording.
Figure 4.8: Performance of latency calibration with varying dimension of the signals. For each value, 20 trials were performed each with random latency and procedural functions. Here, $T = 1$ second, and the periods of the procedural functions were $s_x = 2$, $s_y = 5$. Compared to Figure 4.7, performance is reduced by the fact that the function $x(t)$ is not periodic on the domain $[0, T]$, and hence the quality and number of close points is reduced.

at 30 frames per second (90 frames per example). The rotation and translation examples were able to successfully determine the correct latency for any down-sampling rate of the dependent image signal. The motion example was unable to determine the latency correctly when the resolution was below $3 \times 3$ ($m = 9$ pixels). For higher resolutions, the algorithm was occasionally successful, as shown in Figure 4.10 where the error in latency for the motion example is given as a function of pixel count. However, this is not reliable enough for general use, so a follow up experiment was performed to examine the effect of sample count.

The follow up test was performed over a longer period of time for the motion case.
\[ n = 1, m = 1 \quad \text{and} \quad n = 2, m = 2 \]

**Figure 4.9:** Performance of latency calibration with varying measurement noise of the signals for one dimensional (left) and two dimensional (right) signals. The red line indicates the boundary of 80% success. Here, \( n \) and \( m \) refer to the dimension of the independent signal \( x(t) \) and dependent signal \( y(t) \) respectively.

**Table 4.1:** Snapshots for the videos captured in the two examples.
4.5 Reflection and Future Work

This section has described a novel method for determining the latency between two dependent signals without explicitly modelling the relationship between the two signals. Although this work is exploratory, with only simulated data tested, several
Long Motion Example

Figure 4.11: Error in determined latency for the long motion example using the proposed method for various image down-sample rates. The down-sampled image pixel count represents the dimension of the dependent signal $y(t)$. Compared to Figure 4.10, the data spanned 300 frames (10 sec), and the increased information and self-intersections enables the solution to be determined at resolutions greater than $7 \times 7$ pixels.

conclusions can be drawn. The conditions for this approach to be effective at the task of determining latency between independent signal $x(t)$ and dependent signal $y(t)$ can be qualified as follows:

- There must be a function $F : x \mapsto y$ which is Lipshitz continuous on the domain of interest.
- The continuity of $F$ must be sufficiently smooth on the domain of interest relative to the sample frequency of the signals (the precise quantitative conditions for this statement is left to future work).
• The dimension of the embedded space of \( y \) should not be less than \( x \). Performance is corroded as this condition is exceeded.

• The sampled curve \( x(t) \) should self intersect as many times as possible, with accuracy improving with additional intersections.

• Sample count \( N \) increases compute time with \( O(N^2) \), but also increases accuracy and reliability.

These conditions would be expected to hold for many common scenarios in VR tracking. For example, synchronisation of two 6DoF or 3DoF systems would be possible provided a pair of sensors are connected and the user was to perform some approximately periodic motion, like waving the sensors back and forth. The conditions are less likely to be satisfied if high dimensional signals such as video are used, but they can still be satisfied depending on the scenario as shown by the video simulation results. A case where the conditions are not satisfied is that of an accelerometer and position sensor, because the accelerometer is a derivative of the position signal, not simply a function (see below for further discussion of this case).

The algorithm is sensitive to Gaussian noise (Figures 4.6 and 4.9) but it generalises to higher dimensional signals (Figure 4.7) and complex functions \( F \) (Figure 4.1). The key strength of the algorithm lies in its loose requirements on the relationship between sensors, as demonstrated by the determination of the latency between high dimensional raw video data and camera motion. The function \( F \) in this case is very complicated and dependent on the environment around the camera, but it need not be modelled with the proposed approach.

The key limitations of the proposed algorithm are the time complexity and the curse of dimensionality. The time complexity increases with the number of time samples squared, and increases with the dimensionality of \( x \) and \( y \) in proportion to the time complexity of their norm operations. As the dimension of the embedded spaces of
the signals increases, the distance between points tends to increase due to the additional components of each vector. It is thus less likely for a curve to self-intersect, reducing the number of quality close points which can be used for reference. In one dimension, provided the function $x(t)$ is not monotonic, the curve will intersect itself. In two dimensions, the curve $x(t)$ is confined to a plane, and is thus likely to self intersect at some point (by analogy to random walks). As dimensionality increases, the curve becomes less likely to self-intersect and thus the proposed algorithm succeeds less often. Further research will investigate optimisation of this algorithm, as well as applications to real-data scenarios. The results also suggest that this algorithm might be useful in determining sensor-sensor dependencies, generalising the approaches proposed in Chapter 3 to the model-less case.

Another limitation is that often two sensors are not related by a simple Markov function $F(x(t))$, but by a function depending on previous time steps $F(x(t), x(t-1), x(t-2), \ldots)$. For example, if $y(t)$ is a video stream, and $x(t)$ is a gyroscope measuring the angular velocity of the camera, then $y(t)$ is related to an integral of $x$. In this case, the independent signal would need pre-processing in the form of numerical integration. Another failure case is the differential case, such as $x(t)$ representing 3D pose relative to a global frame, while $y(t)$ is the local acceleration and angular velocity measured by an IMU. In the differential case, it would be necessary to recognise this relation and pre-process $x$ by concatenating previous samples since

$$y(t) = F(x(t), x(t-1), x(t-2))$$

That is, replace the independent signal $x(t) \in \mathbb{R}^n$ with the signal $x'(t) = [x(t)^T, x(t-1)^T, x(t-2)^T]^T \in \mathbb{R}^{3n}$. This approach should capture the necessary information to determine latency between $x$ and $y$. Investigation of the effectiveness of these approaches is left to future work.

The proposed latency algorithm may also be useful for detecting sensor attachment and correlation. In the case that the latency between signals is known, if there is no
relationship between two signals, then the latency error (Equation 4.6) will be high even as \( \delta \to 0 \). If the latency is not known and two signals are not related, then the minimum error \( E(l) \) will remain large even as \( \delta \to 0 \). This observation could lead to a method for determining the existence of generic relationships between variables. However, the value at which Equation 4.6 indicates correlation is unclear and dependent on the particular signals. Significant additional research is required.
Chapter 5

Ambient Sensor Calibration

Previous chapters have investigated the first step of ambient sensor fusion - namely identification of sensor-sensor dependencies and calibration of latencies. This chapter discusses the second step of ambient sensor fusion: ambient calibration. The contributions of Chapter 5 revolve around the design of a state machine for ambient calibration, including:

1. A method for managing calibration data with minimal computational overhead.

2. A method for determining calibration tasks between tracking systems with dependent sensors.

3. Stable fault detection for when a calibration no longer describes the setup accurately.

An evaluation was performed on the task of aligning the skeleton tracked by a Microsoft Kinect v2 with a user’s true body pose for avatar representation and interaction within a VR system. The VR systems tested were the Oculus Rift with Touch controllers and the HTC Vive with wand controllers. The proposed method
Chapter 5. Ambient Sensor Calibration

requires only in-application self-directed movement from the user to achieve alignment. This is shown with an experiment simulating three common virtual reality scenarios. Section 5.1 outlines previous work in sensor fusion and alignment for virtual and augmented reality technologies. Section 5.2 describes the central contributions of this chapter. Section 5.3 details an experiment performed to evaluate accuracy, speed and computation requirements. The work in this chapter was also published as part of the completion of this research project (Fountain and Smith 2017).

5.1 Related Work

Recall from Chapter 2 that alignment of two 3D sensor systems $S$ and $Q$ involves determining the transform $\mathbf{Y} : Q \rightarrow S$. It is typically assumed that the two systems are affine representations of the real world and that $\mathbf{Y}$ only has rotational and

![Diagram](image)

**Figure 5.1:** The relation between two 6DoF sensors is characterized by Equation 5.1. $S$ and $Q$ are the sensor reference frames. $S_t$ and $Q_t$ are the coordinate systems corresponding to the 6DoF pose of each sensor at time step $t$. $\mathbf{Y}$ and $\mathbf{X}$ are the rigid transforms linking the sensor reference frames and the sensor poses respectively. $\mathbf{A}_t$ and $\mathbf{B}_t$ are the 6DoF pose transforms of the two sensors at time step $t$. 
translational components. Scale components are assumed to be known and corrected for before transformation by $Y$. In this case, $Y$ is in the 3 dimensional Special Euclidean Group, or $Y \in SE(3)$. The complete model for two 6DoF tracking systems is given by Figure 5.1 and has characteristic equation

$$A_t X = Y B_t$$  \hspace{1cm} (5.1)

Here, $S_t$ and $Q_t$ are the coordinate frames measured at time $t$ by the sensor systems $S$ and $Q$ respectively. Each measurement yields a result $A_t \in SE(3)$ for $S$ and $B_t \in SE(3)$ for $Q$. There are two unknown elements of $SE(3)$ to determine, $X : Q_t \rightarrow S_t, Y : Q \rightarrow S$ representing the sensor rigid connection and the reference frame relationship respectively. This type of system can be solved with the well established hand-eye calibration developed for robotics systems (Shah 2013). However, many devices, such as Kinect devices do not offer reliable orientation tracking, reducing the knowledge of the system to just the translation component of $A_t$. Pustka et al. summarise the literature on calibrating two sensor systems using a variety of methods including position-only point cloud alignment and 6DoF hand-eye calibration (Pustka, Huber, Bauer et al. 2006).

**Homogeneous Depth Camera Co-Registration**

Müller et al. (Müller et al. 2017) demonstrated a gait analysis system consisting of six fused Kinect v2 cameras networked across several computers. Registration of the Kinect v2 sensors was performed using mutually visible fiducial markers and Procrustes analysis. This process of explicit calibration achieves high accuracy such that the point clouds from each Kinect can also be unified. The result was a system which performed comparably to a gold standard motion capture system for

---

1Procrustes analysis is a shape analysis technique which operates on a point cloud. The analysis involves normalising the point cloud with respect to translation, scale and rotation aspects successively. This reduces the shape to a minimal representation where comparisons are more easily performed.
the purpose of analyzing walking gaits. However, this alignment procedure requires a custom designed visual marker and a manual procedure which requires time and expertise. Additionally, this method cannot be used with systems which cannot track the marker, such as the HTC Vive which uses laser-based tracking. Even for systems which could track the marker, such as the camera based tracking of the Oculus Rift or Playstation VR, custom software would need to be written for each system to account for lens distortion and other camera properties.

**Inhomogeneous Tracking System Alignment**

Czesak et al. created a system for full body tracking using three commodity tracking devices (Oculus Rift DK2, Leap Motion, Kinect v2) (Czesak et al. 2016). However, the system performed no closed-loop calibration and simply used each sensor system to track mutually exclusive body sections. There is ample room for improvement over this model. Destelle et al. demonstrates a procedure for fusing inertial measurement unit (IMU) data with optical data from the Kinect v1 to produce a hybrid skeletal tracking solution with accuracy rivaling that of expensive gold standard optical tracking solutions (Destelle et al. 2014). However, part of the fusion process involves meticulous alignment of the IMU reference frames with one another, and alignment of the IMUs with respect to the Kinect reference frame. The work here aims to automate such procedures.

The work in this chapter explores improved accessibility and ease of use compared to the discussed systems by ambiently aligning sensor systems during typical use. Changes in the state of a sensor system is common due to the trade off between stability and reconfigurability in sensor installation. Fault detection allows the system to function continuously without the need for manual re-calibration if a sensor drifts or is displaced. This is all done in real time.
5.2 Implementation of Ambient Calibration

To address the shortcomings of the systems discussed in Section 2.3 (Existing VR Middleware Software) and Section 5.1 a lightweight plugin for skeleton fusion was developed called *Spooky*, targeting modern game engines. By using a game engine as a middleware, Spooky does not handle device drivers and hardware configuration. By offloading driver and hardware abstraction, the result is a focused system which can be easily integrated into other C++ compatible software platforms. Presently, only Unreal Engine 4 is supported, with future support for other game engines planned.

This section describes the central contribution of this chapter: calibration procedures with fault detection to allow for ambient fusion of inhomogeneous tracking data. The proposed procedures calibrate sensor systems automatically based on ambiently acquired sensor data. The advantage of the proposed system over conventional methods is that no setup is required by the user. Also, if the system configuration is disturbed, recovery is automatic and doesn’t require effort from the user. This is all done in real time at VR compatible timescales (i.e. ≪ 11ms).

![Software architecture of the fusion plugin.](image)

**Figure 5.2:** Software architecture of the fusion plugin.
Chapter 5. *Ambient Sensor Calibration*

The structure of Spooky is described in detail in Chapter 7. Figure 5.2 is reproduced from Chapter 7 to summarise the structure of the system. There are three central software modules within the system - the Correlator, the Calibrator, and the Fusion Graph. These modules represent the three steps in ambient sensor fusion identified in Chapter 2. The Fusion Graph models an articulated skeleton with support for sensor fusion as detailed in Chapter 6. The Fusion Graph is the central module - its structure, a tree of nodes representing rigid bodies, defines how the Correlator and Calibrator work. The Correlator is responsible for determining correspondences between ambiguous sensors - sensors which could be attached to one of many nodes in the Fusion Graph. Details for the Correlator are given in Chapter 3. The Calibrator is responsible for determining the transforms between different sensor systems and detecting faults in calibration. The inner workings of the Calibrator are discussed and analysed in this chapter.

### 5.2.1 The Calibrator

For each iteration\(^2\) of the application, the Calibrator might receive a number of measurements. Each measurement corresponds to a sensor from a single sensor system. More than one measurement from a single sensor can be collected each frame. The Calibrator stores new measurements for a pair of connected sensors only if both of their measurements differ from their previous recorded measurements such that the sum of the position change (m) and the angular change (rad) is greater than \( r = 0.075 \). That is, if the position difference is more than 7.5cm or the angular difference is more than about 4.3° for both connected sensors, new measurements are stored\(^3\). This reduces redundant information and removes the dependency of the calibration on the timing of the user’s movements. Calibration between \( S \) and

---

\(^2\)Iterations may configured to be synchronous with the rendering loop, but need not be.

\(^3\)This value for \( r \) was chosen to suit living-room scale tracking devices based on a heuristic evaluation.
Q is then performed when $M$ measurement pairs $(S_t, Q_t)$ are available. The evaluation in this chapter uses $M = 100$. The values for $M$ and $r$ used were determined by trial and error to trade off calibration quality against the time taken to collect data. Larger $M$ values monotonically result in longer data gathering and calibration computation times, but give higher accuracy and reliability. Smaller $r$ values monotonically result in faster gathering times, but lower data variety and hence lower quality.

The Calibrator determines which systems can be aligned from the sensor measurements available to it. This is done continuously in real time using a data structure called the system-node table. A system is a label corresponding to a single reference frame (for example, “Vive”, “Rift”, “Kinect”, “OptiTrack”). A node is a label corresponding to a real world object, such as “Left Hand”, “Right Hand”, “Head” or even “Box 1”. A sensor is a label corresponding to a single sensor within a given system, usually simply an integer. Each sensor is tagged with exactly one node and one system, but each system and node can have many sensors and corresponding measurements. The system-node table maps one system and one node to their corresponding sensors, and the measurements corresponding to those sensors. Table 5.1 gives an example system-node table.

<table>
<thead>
<tr>
<th>Left Hand</th>
<th>Kinect</th>
<th>Vive</th>
<th>OptiTrack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>$k_1$</td>
<td>$v_1$</td>
<td>$o_1, o_2$</td>
</tr>
<tr>
<td>Right Hand</td>
<td>$k_2$</td>
<td>$v_2$</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$k_3$</td>
<td>$v_3$</td>
<td>-</td>
</tr>
</tbody>
</table>

Each frame, the Calibrator decides if calibration is viable between each unordered pair of systems $(S, Q)$. Calibration is performed as follows:

1. For each node (row in the system-node table):
   (a) Check if $S$ and $Q$ have corresponding measurements.
(b) If there are more than $m$ corresponding measurements for the node, store them for calibration later.

2. If more than $M$ measurements are stored in total, perform calibration as described in Arun et al. (1987), Shah (2013) and Pustka, Huber, Bauer et al. (2006).

3. Clear measurements which will not be used for further calibration.

In this way, the measurements are sorted and analyzed in real time with only a small overhead. Typical values for the parameters $M$ and $m$ were $M = 100$ and $m = 4$. The update frame-rate is set in configuration. The algorithm operates asynchronously relative to each sensor system. Interpolation is used to synchronize measurements sampled at times with small differences. However, two measurements received at the same time are naively considered synchronous. Ambient latency compensation techniques are discussed in Chapter 4, but the work in that chapter was performed after the work in this chapter, and so it was not used in the system for this chapter.

5.2.2 Calibration State Machine

The calibration of each pair of systems occupies one of three calibration states:

- **Uncalibrated (U)** - no calibration information available. The system is either still gathering data regarding the two tracking systems or the systems cannot be calibrated because they share no dependent data.

- **Refinement (R)** - partial data is available for the calibration. The system will continue to refine the result until convergence is achieved.

- **Calibrated (C)** - the calibration has converged and the system is no longer adjusting calibration. Fault detection is now running to detect a systematic error between the two systems.
A state machine is maintained for each pair of systems \((S, Q)\). Every time a calibration is performed between two systems, their calibration state is updated according to the state transition diagram in Figure 5.3.

![State machine for calibration between two systems. States: Uncalibrated (U), Refining (R), Calibrated (C).](image)

The transition conditions are computed from the mean validation error \(E\) of the latest calibration. Computation of \(E\) depends on the type of calibration. For example, \(E = \frac{1}{N} \sum_{t=1}^{N} ||A_t X - YB_t||\) with the Frobenius norm for a complete 6DoF hand-eye calibration (Shah 2013) or \(E = \frac{1}{N} \sum_{t=1}^{N} ||Yb_t - a_t||\) for position only point cloud alignment (Arun et al. 1987). Additionally, a quality measure \(q : \mathbb{R} \rightarrow [0, 1]\) is computed from the error value to create a bounded metric of calibration performance

\[
q(E) = \frac{1}{1 + (E/s)^2}
\]

where \(s\) is a tunable scale parameter, fixed at runtime. For 6DoF calibration error, \(s = 1\) was used, while \(s = 0.05\) was used for position only calibration error. This
accounts for the different magnitudes of the different norms. The quality measure \( q(E) \) is monotonically decreasing with \( E \), with \( q(E) = 1 \) indicating a perfect solution.

<table>
<thead>
<tr>
<th>Transition name</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial calibration</td>
<td>( q(E) &gt; 0.5 )</td>
</tr>
<tr>
<td>Calibration failed</td>
<td>( q(E) \leq 0.5 )</td>
</tr>
<tr>
<td>Error stabilized</td>
<td>((\Delta q(E) &lt; 0.01) \land (q(E) &gt; 0.90))</td>
</tr>
<tr>
<td>Improving error</td>
<td>( \Delta q(E) &gt; 0.01 )</td>
</tr>
<tr>
<td>Error diverges</td>
<td>( \neg(\text{Error stabilized}) \land \neg(\text{Improving error}))</td>
</tr>
<tr>
<td>Fault detected</td>
<td>See Section 5.2.3</td>
</tr>
<tr>
<td>Tracking data agrees</td>
<td>( \neg(\text{Fault detected}) )</td>
</tr>
</tbody>
</table>

The state of the calibration of \((S, Q)\) is defined by the tuple \((Y, E, N, q) \in SE(3) \times \mathbb{R} \times \mathbb{N} \times [0, 1]\). Here, \( Y \) is the \(4 \times 4\) homogeneous matrix \( Y : S \rightarrow Q \), \( E \) is the mean validation error, \( N \) is the number of samples which have been used to obtain \( Y \) and \( q \) is the quality of the calibration. After each calibration operation, the calibration state \((Y', E', N', q(E'))\) is updated using the new calibration result \((Y', E', N', q(E'))\) by interpolation weighted by \( N \) and \( N' \):

\[
Y \leftarrow \text{slerp}(Y, Y', N'/N + N') \tag{5.3}
\]
\[
E \leftarrow \frac{NE + N' E'}{N + N'} \tag{5.4}
\]
\[
q \leftarrow \frac{Nq + N' q(E')}{N + N'} \tag{5.5}
\]
\[
N \leftarrow N + N' \tag{5.6}
\]

Here, \( \text{slerp}(X, Y, \alpha) \in SE(3) \) is the spherical linear interpolation of \( RX \) to \( RY \) by \( \alpha \) and the linear interpolation from \( x \) to \( y \) by \( \alpha \in [0, 1] \). This type of interpolation update guarantees eventual convergence of the results, leading to a stable calibration result while accurately weighting any erroneous results. Table 5.2 describes the transition logic which is followed whenever a new calibration result is computed. Here, the signed change in quality \( \Delta q(E) \) is defined as the change in quality which
occurred due to the update in Equation 5.5. The state diagram in Figure 5.3 is used to update the state of the calibration.

### 5.2.3 Fault Detection

Fault detection is responsible for detecting a change in configuration in the sensor network, such as movement or drift of a reference frame. When the Calibrator is in the Calibrated state, new calibrations are performed as usual (Section 5.2.1). However, the update step given by Equations 5.3-5.6 is not performed. Instead a background calibration result $Y_B \in \text{SE}(3)$ is tracked. The background calibration result is updated in two steps based on each new calibration result $Y'$. First, a decay step:

$$Y_\beta = \text{slerp}(Y_B, I_{4,4}, \beta)$$  \hspace{1cm} (5.7)

where $I_{4,4}$ is the $4 \times 4$ identity matrix and $\beta \in [0, 1]$ is the decay rate. Secondly, an exponential filter step is used to compute the new value of $Y_B$

$$Y_B \leftarrow \text{slerp}(Y_\beta, Y', \gamma)$$ \hspace{1cm} (5.8)

where $\gamma \in [0, 1]$ was the exponential gain. After each calibration, $Y_B$ is compared to the calibration result $Y$ computed just before transitioning from the Refining state to the Calibrated state. If $Y_B^{-1}Y$ describes a rotation of angle greater than $\theta$ or displacement more than $d$, a fault is considered to be detected, and calibration is reset. In the evaluation, the choice of $\theta$ and $d$ was critical. The values were chosen to be robust to noise and false positives. The cost of this design is that fault detection can take longer than calibration from the Uncalibrated state. By using a mean filter for updating calibration state the result is bounded and convergent, whereas the exponential slerp filter for the fault state $Y_B$ is divergent and will eventually take the value of the new calibration result.
5.3 Evaluation

To evaluate the proposed solution, Spooky was used to calibrate a Kinect v2 skeleton tracking camera with a VR system. The VR system tracked the users hands and head with 6DoF while the Kinect measured body joint positions. Two VR systems were tested: the HTC Vive with Vive controllers and the Oculus Rift with Touch controllers. The Kinect v2 provides measurements of the user’s entire body, but at a coarse scale compared to the VR tracking systems. The mean error in Kinect v2 tracking compared to ground truth of gold standard tracking has been reported to be around 23cm for the hands and 5cm for the head (Rietzler et al. 2016). This error should be kept in mind when considering the results in the following sections.

5.3.1 Method

The Kinect was placed approximately 1.5 meters off the ground. A user performed tasks modeled around modern room scale VR applications. For example, walking, handling virtual objects and shooting (pointing) at distant virtual targets. The actions were performed while facing the body no further than 90° away from the Kinect. After around 10-50 seconds (depending on the task), the system computed the alignment as described by Arun et al. (1987). The system was configured with calibration sample thresholds of $\theta = 5^\circ$ and $d = 10\text{cm}$, and parameters $\beta = 0.1$ and $\gamma = 0.25$ (Equations 5.7-5.8). The resulting transform $Y$ was then compared to the actual location of the Kinect as measured by an OptiTrack motion capture system. The actions performed to collect the data included the following VR tasks:

- **Sorting** (Figure 5.4a,b) - the user is tasked with picking up virtual items from a shelf walking a short distance to place them on a virtual platform of the same color. This simulates applications such as *Job Simulator* by Owlchemy Labs.\(^4\)

---

\(^4\)http://jobsimulatorgame.com/ (1/9/2018)
• **Pointing** (Figure 5.4c) - the user is tasked with pointing at targets placed up to 90° either side of the user and up to 90° elevation. This task simulates applications where the user stands still but uses their hands, such as *Robo-Recall* by Epic Games\(^5\).

• **Walking** (Figure 5.4d) - the user walked around the tracking space while facing toward the Kinect while moving arms slowly up and down. This represents an ‘ideal’ calibration scenario with large amounts of movement around the tracking space.

**Figure 5.4:** The three tasks used to assess the performance of the ambient calibration. In the sorting task, the user must sort the cubes (a) into their respective colors (b) on the platforms a few steps away. In the pointing task, the user must point to a series of targets while standing in place (c). The walking task involves the user stepping around the tracking space while facing the Kinect and slowly raising and lowering their arms (d).
Figure 5.5: Distributions of errors for Rift (a) and Vive (b) for three different ambient calibration scenarios. Confidence ellipses (95%) are shown as visual aids.
5.3.2 Results

Table 5.3: Summary statistics for errors of ambient calibration procedure compared to ground truth. Values are reported in ‘mean ± standard deviation’ format.

<table>
<thead>
<tr>
<th></th>
<th>Position Error (cm)</th>
<th>Angle Error (°)</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walking</td>
<td>Rift</td>
<td>Vive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15.7 ± 4.7</td>
<td>19.9 ± 2.8</td>
<td>13 ± 2</td>
</tr>
<tr>
<td></td>
<td>3.65 ± 1.8</td>
<td>5.15 ± 0.9</td>
<td>11 ± 2</td>
</tr>
<tr>
<td>Sorting</td>
<td>Rift</td>
<td>Vive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>17.5 ± 2.7</td>
<td>24.3 ± 1.7</td>
<td>13 ± 3</td>
</tr>
<tr>
<td></td>
<td>2.93 ± 0.9</td>
<td>3.47 ± 0.7</td>
<td>15 ± 3</td>
</tr>
<tr>
<td>Pointing</td>
<td>Rift</td>
<td>Vive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>27.34 ± 6.4</td>
<td>35.8 ± 5.4</td>
<td>34 ± 4</td>
</tr>
<tr>
<td></td>
<td>4.07 ± 1.4</td>
<td>4.2 ± 0.9</td>
<td>40 ± 16</td>
</tr>
</tbody>
</table>

For each movement type, the task was performed until the Calibrated state was reached. The error was then recorded before resetting calibration and starting again. Each trial was repeated 10 times and the results are shown in Figure 5.5 and summarized in Table 5.3. The durations required to gather data for calibration are also noted in Table 5.3. Errors were computed based on ground truth Kinect pose measured using a gold-standard OptiTrack motion capture system (see Figure 5.6). The OptiTrack system was used to compute a mapping from the VR tracking space to the Kinect tracking space. First, the mapping \( V \in SE(3) \) was computed using a hand-eye calibration (Shah 2013) between an OptiTrack rigid body marker and a VR controller. Next, a marker was placed on the Kinect and calibrated such that it coincided in orientation and position to the true centre and orientation of the Kinect. The OptiTrack system then measures the mapping \( K \in SE(3) \) between the OptiTrack system and the marker on the Kinect. If the transform between the marker on the Kinect and the Kinect tracking space is \( I \in SE(3) \), the system equation is found by forming a loop in Figure 5.6 giving \( IKV = Y \). The approximation is made that \( I \) is equal to the identity transform, and thus the calibration error is

\(^5\text{https://www.epicgames.com/roborecall/} \ (1/9/2018)\)
given by the matrix $E(Y) := KVY^{-1}$. The error matrix $E(Y)$ will be equal to the identity when the calibration $Y$ is perfectly accurate. The final error values reported in Table 5.3 are decomposed into the magnitude of the translation of $E(Y)$ and the magnitude of the angle of $E(Y)$ when decomposed into angle-axis form.

5.3.3 Discussion

The mean error in Kinect v2 tracking compared to ground truth of gold standard tracking has been reported to be around 23cm for the hands and 5cm for the head (Rietzler et al. 2016). The calibration results (Figure 5.5 and Table 5.3) compare favorably with the expectation of the hand tracking error, but not the head tracking error. This is likely explained by the fact that less data from the user’s head is used in calibration than data from the arms simply because people tend to move their arms more than their head. This is a limitation with the problem of ambient calibration itself - you cannot control the user’s actions. Future work will improve

![Diagram](https://via.placeholder.com/150)

**Figure 5.6:** To measure the accuracy of the ambient calibration $Y$ between the Kinect and the VR systems, an external gold-standard OptiTrack motion capture system was used to measure $K$. 
Figure 5.7: An example of the typical results for calibration with walking activity. An outline of the user’s real body (white edge) from the Vive’s passthrough camera is shown on the right, overlayed with the virtual scene rendered from user perspective. The controllers are outlined with a green edge. This calibration took 16 seconds, and had an error of 22.2cm and 5.25°.

performance in this domain by incorporating prior information about relative reliability of different sensors, as is well established in the literature (e.g. Rietzler et al. (2016)). Subjectively, the skeleton tracked by the Kinect matches well with the real body position, even with 22.2cm and 5.25° error. Figure 5.7 shows such a typical calibration result visualized from first person and third person perspectives. The avatar aligns well with the real body seen through the Vive’s pass-through camera. It should be noted that the pass-through camera is offset slightly below and in front.
of the user’s eyes, and this introduces some error in where the body appears with the pass-through camera. More distant objects are less affected by this error, such as the feet. The method is agnostic to device and tracking method, though it does not achieve the same level of accuracy as Müller et al. (2017), who use time consuming and fault sensitive fiducial based calibration.

Worse results were observed in the pointing task. This is expected since a smaller variety of data is collected compared to the other tasks. In particular, the head remained mostly stationary and thus the hand trackers recorded the majority of data for calibration. The walking task and the sorting task performed similarly, suggesting that it is important to utilize the more accurate head tracking of the Kinect. However, it is necessary to use the hand data to sample outside the plane the head usually moves within (crouching of the user is rare). The walking task likely performed best due to the lower redundancy in data collected.

Figure 5.8 demonstrates an instance of ambient calibration, fault detection and recovery. The ground truth pose of the Kinect is shown as a function of time alongside the positional and rotational calibration error of the system compared to ground truth. The calibration states of the system are indicated by background color of the graph. A fault of about 8° in the yaw position of the Kinect is detected and corrected within 40 seconds. Calibration remains stable otherwise. The time taken to calibrate and detect faults is well within the desired range for ambient calibration (less than 1 minute). The actual mean computation time required for the calibration operations was measured to be at most 2.64 (± 0.30) ms. Analysis was performed on a Windows 10 PC with an Intel Xeon E5-1650 v3, 3.50GHz, 6 cores. However, this amount of compute time was only required on the frames where calibration was triggered, or once every 5 seconds or so. This amount is much less than the typical render budget of 11ms allocated for an application running at 90Hz. Also, multi-threading could be used to perform the calibration during the span several frames if necessary. During data collection, for 3 systems (Kinect, OptiTrack and a VR
system) computation requirements were measured to be about 0.3ms for processing measurements.

However, the Kinect cannot distinguish between the cases of the user facing the device and facing away from the device. The proposed technique does not take into account this shortcoming and so calibration can only be performed facing the Kinect. In future work, this could be overcome by breaking symmetry using the VR system. Additionally, the calibration model doesn’t account for the non-zero rigid transform which naturally must exist between two rigidly linked sensors. The Kinect tracks the wrist position, whereas the reported centre position of the controller is not on the wrist position, but rather on the controller itself. A similar model describes the head: the VR headset centre is not the same point as the head point.

**Figure 5.8:** An example calibration trace with calibration states (Figure 5.3) overlayed in color (Red = ‘Uncalibrated’, Yellow = ‘Refining’, White = ‘Calibrated’). The Kinect is moved at the 55 second mark, giving it a rotation 8 degrees from its original configuration. The error is corrected within 40 seconds.
measured by the Kinect. These offsets vary depending on the devices involved and the physiology and behaviour of the user, but could be considered constant over a session of usage. Therefore the approach of configuring offsets manually is highly inconvenient. Point cloud alignment used in this chapter doesn’t account for these differences and instead assumes the same point is measured from both reference frames (Arun et al. 1987). This explains the discrepancy in results between the Rift and the Vive seen in Figure 5.5. The Vive wand controllers are much larger than the Oculus Touch controllers, and so are more likely to have a larger rigid offset from the measured Kinect hand position. This problem of automatically determining individual offsets requires further research. The Rift results (Figure 5.5a) feature significant correlation between rotational error and positional error. This is due to the technique used for calibration; calculation of the rotation transform is performed first, and the positional error depends on the accuracy of the resulting transform. The Vive shows less correlation in angular and positional error, likely due to the previously described rigid link factor which masks the correlation.

5.4 Reflection and Future Work

A method for ambient calibration of sensor systems was presented. The central contribution is a state machine for gathering data, performing calibration and detecting faults in calibration. An evaluation was performed to calibrate two example VR systems with the Kinect v2. It was demonstrated that the resulting calibration has accuracy on the order of what is expected given the accuracy of the Kinect. Typical accuracy was demonstrated to be 20cm and 4° compared to the ground truth with only around 20 seconds of in-application user-directed movement. This work is an important step toward ambient calibration and fusion of real-time sensor systems. With the conclusion of this chapter, an algorithm has been developed, implemented and assessed for solving the second of the three stages of automatic
sensor fusion described in Chapter 2. Next, Chapter 6 discusses a new approach to fusing tracking data to a skeleton model for the final stage of ambient sensor fusion.
Chapter 6

Skeleton Sensor Fusion

Chapter 6 proposes a new approach to fusing the data from multiple skeletal sensors, after data has been aligned to the same coordinate system. The proposed technique solves the third and final step of the sensor fusion process described in Chapter 2. The purpose of fusion is to extract a model from partial or redundant data over time. In this chapter, the focus is on tracking articulated bodies in 3D. The model is thus a tree of joints with varying degrees of freedom. The proposed algorithm incorporates quadratic constraints into each measurement update to allow for estimation of pose from partial information. The algorithm also enables fusion of redundant data. Implementation is included in the Unreal Engine 4 plugin, Spooky (Chapter 7).

Contributions from this chapter include:

- A data structure called the Fusion Graph, which can flexibly describe an articulated model for body pose.
- An algorithm for determining a chain of key bones required for fusion of a given joint.
- An algorithm for performing non-linear pose extraction that scales to real time settings, with support for constraints.
6.1 The Fusion Graph

This section details the mathematical structure of the Fusion Graph, a tree of spatial nodes capable of representing the human body and other articulated objects. The Fusion Graph also generalises to rigid bodies trivially.

Articulated Model

In Spooky, the human body is modeled as a tree of nodes. The structure of a node is summarised in Table 6.1. The two central defining characteristics of a node $N$ are the node’s parent $P(N)$ and the set of ordered articulations $A(N) = (A_0, \ldots, A_d)$ within a node. Articulation types are detailed in Table 6.2. Each articulation $A_i$ has an internal state $(\mu_i \in \mathbb{R}^k, \Sigma_i \in \mathbb{R}^{k \times k})$ for some $k \in \mathbb{N}$, and a function $L_{A_i} : \mathbb{R}^k \to \mathbb{R}^{4 \times 4}$, called the pose function.

<table>
<thead>
<tr>
<th>Component</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>string</td>
<td>Unique name of the node for reference</td>
</tr>
<tr>
<td>Parent</td>
<td>pointer</td>
<td>Pointer to parent node in the fusion graph</td>
</tr>
<tr>
<td>Articulations</td>
<td>articulations</td>
<td>List of articulations that define the behaviour of the node</td>
</tr>
</tbody>
</table>

The local pose of node $N$ is then given by

$$L(N) = \prod_{i=d}^{1} L_{A_i}(\mu_i)$$  \hspace{1cm} (6.1)

Note that Equation 6.1 defines the multiplication in decreasing order, so that transformation of a point in $\mathbb{R}^4$ by $L(N)$ applies $L_{A_1}$ first, and $L_{A_d}$ last. The pose of the
Table 6.2: Types of Articulation. See Section 2.1.2 for definitions of the functions \( \exp, T, S \) and more information on representing 3D transforms.

<table>
<thead>
<tr>
<th>Articulation</th>
<th>Dimension ( l )</th>
<th>Pose Function ( L : \mathbb{R}^l \rightarrow \mathbb{R}^{4 \times 4} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bone</td>
<td>3</td>
<td>( \omega \mapsto T(v) e^{\omega} ), fixed ( v \in \mathbb{R}^3 )</td>
</tr>
<tr>
<td>Pose</td>
<td>6</td>
<td>( (\omega, v) \mapsto T(v) e^{\omega} )</td>
</tr>
<tr>
<td>Position</td>
<td>3</td>
<td>( v \mapsto T(v) )</td>
</tr>
<tr>
<td>Scale</td>
<td>3</td>
<td>( s \mapsto S(s) )</td>
</tr>
<tr>
<td>Pivot</td>
<td>1</td>
<td>( \theta \mapsto \exp(\hat{\omega}\theta) ), fixed ( \hat{\omega} \in so(3) )</td>
</tr>
<tr>
<td>Twist</td>
<td>1</td>
<td>( \theta \mapsto \exp(\hat{\xi}\theta) ), fixed ( \hat{\xi} \in se(3) )</td>
</tr>
</tbody>
</table>

Node \( N \) in global space is defined by the recursive function

\[
G(N) = G(P(N))L(N) \tag{6.2}
\]

The transform \( G(N) \) maps from the local node space to the global space, similar to a scene graph\(^1\). The next section describes the structure of measurements which provide information for determining the states \((\mu_i, \Sigma_i)\) of the articulations within the fusion graph.

**Measurements**

A *measurement* is defined as a piece of structured data that represents measured information about the pose of a node at a given instance. The structure of a measurement as implemented is summarised in Table 6.3. These parameters are set when measurements are created and input into Spooky. The name of the system which generated the data is set as a string and used to determine calibration procedures (Chapter 5). A measurement has one of several types, defined in Table 6.4, which define the method of fusion used when incorporating the measurement. The Node Set names a set of nodes, for example \( \{\text{hips, head, lower_arm_r}\} \), to which the measurement may correspond to. If the Node Set contains more than one node, then techniques from Chapter 3 can be used to determine which node it is attached

---

\(^1\)Common in computer graphics, a scene graph models a 3D scene as a tree of linear transforms where the pose of a node is inherited from its parents.
Table 6.3: Measurement structure

<table>
<thead>
<tr>
<th>Component</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor system</td>
<td>string</td>
<td>The name of the system which generated the measurement</td>
</tr>
<tr>
<td>Measurement Type</td>
<td>enum</td>
<td>The type of measurement from a prescribed list (Table 6.4)</td>
</tr>
<tr>
<td>Node Set</td>
<td>strings</td>
<td>The set of nodes which this sensor could be attached to</td>
</tr>
<tr>
<td>Root node</td>
<td>string</td>
<td>Node of the fusion graph to which measurements are relative</td>
</tr>
<tr>
<td>Dimension</td>
<td>( n \in \mathbb{N} )</td>
<td>The dimension of data stored, dependent on measurement type</td>
</tr>
<tr>
<td>Data</td>
<td>( z \in \mathbb{R}^n )</td>
<td>The data resulting from the measurement</td>
</tr>
<tr>
<td>Variance</td>
<td>( \Sigma_z \in \mathbb{R}^{n \times n} )</td>
<td>The covariance matrix associated with the data</td>
</tr>
<tr>
<td>Time-stamp</td>
<td>( t \in \mathbb{R} )</td>
<td>The time when the measurement was taken</td>
</tr>
<tr>
<td>Flags</td>
<td>boolean</td>
<td>A set of indicators for various measurement properties (Table 6.5)</td>
</tr>
<tr>
<td>Confidence</td>
<td>( C_z \in [0, 1] )</td>
<td>Likelihood that the measurement is reliable</td>
</tr>
</tbody>
</table>

to before fusion (provided other sensor data is available for the candidate nodes). The root node of a measurement refers to the node in the fusion graph to which measurements are relative. The default root node is the static global root node of the graph. However, some sensor systems are not statically referenced but move with some part of the fusion graph. For example, the Leap Motion can be attached to a VR headset to provide hand tracking. In this case the root node corresponds to the user’s head node.

Measurement data is stored as a multivariate Gaussian \( N(z, \Sigma_z) \) representing the measurement probability distribution (defined in Section 2.2.1). The timestamp records when the measurement was observed. A set of flags, described in Table 6.5, determine some of the settings for how the data is fused. Finally, a confidence value \( C_z \in [0, 1] \) defines the likelihood that the measurement is not anomalous. The precise ways in which these data are used is described in the following sections.
Table 6.4: Measurement types and associated data structures

<table>
<thead>
<tr>
<th>Measurement Type</th>
<th>Dimension</th>
<th>Structure (z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>3</td>
<td>(x, y, z)</td>
</tr>
<tr>
<td>Rotation</td>
<td>4</td>
<td>(q₀, q₁, q₂, q₃)</td>
</tr>
<tr>
<td>Pose</td>
<td>7</td>
<td>(q₀, q₁, q₂, q₃, x, y, z)</td>
</tr>
<tr>
<td>Scale</td>
<td>3</td>
<td>(sₓ, sᵧ, sₚ)</td>
</tr>
</tbody>
</table>

Measurements can be input into the Spooky system in two ways: individually or as a skeleton. If individual, the data is configured in either C++ or UE4 blueprints:\(^2\).

If skeleton measurements are used, then UE4 blueprints must be used. Skeleton measurements are convenient for sensors which already integrate with UE4 through Animation Blueprints. Skeleton measurements allow for a subset of bones from an animated skeleton to be configured for fusion quickly. See Section 7.2.3 for more information. Skeleton measurements appear in the fusion graph simply as a set of simultaneous measurements, so the analysis for both measurement types is the same.

Table 6.5: Measurement flags for controlling measurement fusion behaviour

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>Meaning when True</th>
<th>Meaning when False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Space</td>
<td>Measurement is relative to the sensor system root node</td>
<td>Measurement is relative to the parent node</td>
</tr>
<tr>
<td>Ignore Unchanged</td>
<td>Measurements are only applied if they do not match the hash of the previous measurement from the sensor</td>
<td>Measurements are always applied</td>
</tr>
<tr>
<td>Offset Accumulation</td>
<td>Accumulate transform offsets from other sensors</td>
<td>Assume no drift present</td>
</tr>
<tr>
<td>Velocity Measurement</td>
<td>The input measurement is differentiated and used to update the model’s velocity</td>
<td>Measurement updates pose of model directly</td>
</tr>
</tbody>
</table>

\(^2\)UE4 Blueprints are a graphical programming language native to UE4. See Chapter 7 for more information.
6.2 Fusion Algorithm

The fusion algorithm operates as follows for each frame of the program. Firstly, the collection of measurements received since the last fusion step are attached to their respective nodes. If more than one node could correspond to a given measurement, then methods from Chapter 3 are used to disambiguate. If more than one sensor system is present, then methods from Chapter 5 are used to find mappings between the sensor systems, provided there is at least one pair of dependent sensors between the systems (that is, there are two sensors attached to the same node). Once results from these two processes are complete, the fusion of data can begin. All measurements are transformed into the pre-configured fusion reference space. The fusion reference space is chosen to be one of the component system reference frames, selected at configuration by the user. Finally, the nodes are recursively fused, with fusion performed for each node’s parent first, and the result used to drive an avatar. Figure 6.1, reproduced from Chapter 7, graphically summarises these 3 steps, performed by the Correlator, Calibrator and Fusion Graph respectively.

![Software architecture of the fusion plugin.](image)
6.2.1 Recursive Fusion

Algorithm 6.1: Recursive Fusion Algorithm

**Data:** Node N, Timestamp t, Reference Sensor System S, confidenceThreshold (= 0.01)

**Result:** Updated state of N and possibly parents of N

```plaintext
begin
    // Fuse parents
    if Parent(N) exists then
        Fuse(Parent(N), t, S);
    end
    // Fuse this node
    for m in N.measurements do
        // Ignore low confidence measurements
        if m.confidence < confidenceThreshold then continue;
        // Transform to reference space
        m' ← m.transform(N.getCalibrationResult(S));
        // Traverse parents
        fusionChain ← DetermineFusionChain(N, m'.type);
        // Adjust parents and self to fuse
        ConstrainedEKFUpdate(fusionChain, m, t);
    end
    // Don’t fuse measurements twice
    Clear(N.measurements);
end
```

The pseudocode for the proposed recursive fusion algorithm is given in Algorithm 6.1. The first step is to check if the node has a parent, and if so, call the fuse function recursively on the parent. Thus, a node only incorporates its own measurements into the graph after the best pose of its parent is found. The next step iterates over the list of unused measurements and fuses them into the fusion graph independently. Instead of updating the entire fusion graph as one large filter, which is computationally expensive\(^3\), the system traverses the node’s parents until sufficient degrees of freedom are found to satisfy the measurement type. The set of nodes included in a particular fusion operation is called the *fusion chain*.

\(^3\)The EKF update algorithm is \(O(n^2)\) where \(n\) is the filter state dimension (Merwe and Wan 2001).
The fusion chain starts from the measured node and spans the parent nodes required for fusion. At its longest, the fusion chain ends at the root node for the measurement. However, typically no more than three nodes are included in the fusion process. The fusion chain selection algorithm operates by counting degrees of freedom (DoF) in rotation, position and scale. For example, consider a fusion graph made of only bone type nodes (Table 6.4). If fusing a position measurement, then three nodes will be included in the fusion chain. A position measurement requires 3 DoF of position to be satisfied. On its own, a bone type node can only pivot about a point and so has 0 DoF in position. However, the fixed translation component of a bone means that
the parent bone of a bone can move its child to any point on a sphere, and thus it provides 2 positional DoF. In this case, it is said that the node has a *lever child*. The same applies for the parent of the parent, totaling 4 DoF in position for a fusion chain three nodes long, satisfying the required 3 DoF. Of course, this ignores the size of the levers involved, and a measurement could lie outside the reachable distance. However, it was found that this algorithm was sufficient for most use cases. Possible extensions are discussed in Section 6.4. Putting aside this limitation, 3 nodes is the minimum fusion chain required to place the child in the correct position with at least 3 degrees of positional freedom. Figure 6.2 provides an illustration of this scenario for a hypothetical fusion graph of bone type nodes. Algorithm 6.2 details the procedure for determining the fusion chain.

Algorithm 6.2: Fusion Chain Selection

Data: Start Node N, Measurement Type T

Result: List of nodes for fusion

1 begin
2 nodes ← EmptyList();
3 currentNode ← N;
4 pDoF ← 0; // Positional
5 rDoF ← 0; // Rotational
6 sDoF ← 0; // Scale
7 requiredDoF ← getRequiredDoF(T) ; // (p,r,s)
8 while any of (pDoF,rDoF,sDoF) < requiredDoF do
9 pDoF ← pDoF + currentNode.getPDoF();
10 rDoF ← rDoF + currentNode.getRDoF();
11 sDoF ← sDoF + currentNode.getSDoF();
12 nodes.insert(currentNode);
13 currentNode ← Parent(currentNode);
14 end
15 return nodes;
16 end

The available degrees of freedom for a node are given by the sum of available degrees of freedom for each articulation. The DoF for a given articulation depends on whether not the articulation has a lever child, and is given in Table 6.6. The required DoF for a given measurement type is given in Table 6.7. The required DoF and
available DoF are a function of measurement type and articulation type only, leading to a stable fusion chain.

Assumptions of this model include:

- Measurements of a node are not dependent on measurements of the children, and thus the parent can be fused first. (However, the state and measurements of a node is influenced by its parent’s state, and so modifying the parent is justified)

- Order of fusion of measurements has minor impact on the end result

Once the fusion chain has been determined, the states of the articulations in the fusion chain are concatenated into one conglomerate state vector $\mu \in \mathbb{R}^n$:

$$\mu = [\mu_1^T \ldots \mu_k^T]^T$$

(6.3)
Table 6.7: Required Degrees of Freedom for Measurement Types

<table>
<thead>
<tr>
<th>Measurement Type</th>
<th>Position DoF</th>
<th>Rotation DoF</th>
<th>Scale DoF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rotation</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Pose</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Scale</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

The covariance matrix is also concatenated, with non-diagonal entries set to zero:

\[
\Sigma = \begin{bmatrix}
\Sigma_1 & 0 & 0 & \cdots & 0 \\
0 & \Sigma_2 & 0 & \vdots & \\
0 & 0 & \Sigma_3 & 0 & \\
\vdots & \ddots & 0 & \\
0 & \cdots & 0 & \Sigma_n
\end{bmatrix}
\] (6.4)

Prediction and measurement updates are then performed on this combined state.

### 6.2.2 Prediction Update

Each articulation includes an option for modelling the velocity of the state vector. For example, if the pivot \( \omega \in \mathbb{R}^3 \) is the default state and the velocity is optionally stored as an additional \( \dot{\omega} \in \mathbb{R}^3 \), then the complete state vector is

\[
\mu = \begin{bmatrix}
3 \\
\dot{3}
\end{bmatrix}
\] (6.5)

For position, scale, pivot and twist articulations, the update function is simply a linear function of the current state and the time elapsed since the last prediction step. For bone and pose articulations, the prediction function is given by the composition of rotations in \( SO(3) \), mapped back into \( so(3) \). Table 6.8 defines the prediction functions \( U : (\mu_i, \delta t) \mapsto \mu_i \) given the change in time since last update \( \delta t \). Note that \( \delta t \) is calculated per node, and is zero if the node has been updated this frame already,
Table 6.8: Update functions for articulations with velocity modelled.

<table>
<thead>
<tr>
<th>Articulation Type</th>
<th>Prediction Update Function $U : \mathbb{R}^l \times \mathbb{R} \rightarrow \mathbb{R}^l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bone</td>
<td>$\begin{pmatrix} \mathbf{\omega}, \mathbf{\dot{\omega}}, \delta t \end{pmatrix} \mapsto \log \left( e^{\delta t \mathbf{\dot{\omega}} e^{\mathbf{\omega}}} \right), \mathbf{\dot{\omega}}$</td>
</tr>
<tr>
<td>Pose</td>
<td>$\begin{pmatrix} \mathbf{\omega}, \mathbf{v}, \mathbf{\dot{\omega}}, \mathbf{\dot{v}}, \delta t \end{pmatrix} \mapsto \log \left( e^{\delta t \mathbf{\dot{\omega}} e^{\mathbf{\omega}}} \right), \mathbf{v} + \mathbf{\dot{v}} \delta t, \mathbf{\dot{\omega}}, \mathbf{\dot{v}}$</td>
</tr>
<tr>
<td>Position</td>
<td>$\begin{pmatrix} \mathbf{v}, \mathbf{\dot{v}}, \delta t \end{pmatrix} \mapsto \left( \mathbf{v} + \mathbf{\dot{v}} \delta t, \mathbf{\dot{v}} \right)$</td>
</tr>
<tr>
<td>Scale</td>
<td>$\begin{pmatrix} \mathbf{s}, \mathbf{\dot{s}}, \delta t \end{pmatrix} \mapsto \left( \mathbf{s} + \mathbf{\dot{s}} \delta t, \mathbf{\dot{s}} \right)$</td>
</tr>
<tr>
<td>Pivot</td>
<td>$\begin{pmatrix} \theta, \mathbf{\dot{\theta}}, \delta t \end{pmatrix} \mapsto \left( \theta + \mathbf{\dot{\theta}} \delta t, \mathbf{\dot{\theta}} \right)$</td>
</tr>
<tr>
<td>Twist</td>
<td>$\begin{pmatrix} \theta, \mathbf{\dot{\theta}}, \delta t \end{pmatrix} \mapsto \left( \theta + \mathbf{\dot{\theta}} \delta t, \mathbf{\dot{\theta}} \right)$</td>
</tr>
</tbody>
</table>

so multiple calls to prediction update for one node are equivalent to one prediction update if made in the same frame. The prediction update is then approximated by

$$
\bar{\mathbf{u}}_i = U_i(\mathbf{\mu}_i, \delta t_i) \tag{6.6}
$$

$$
\bar{\Sigma}_i = J(U_i)(\mathbf{\mu}_i)\Sigma_iJ(U_i)(\mathbf{\mu}_i)^T + \mathbf{P}_i \delta t_i \tag{6.7}
$$

where $J(U_i)(\mathbf{\mu}_i)$ is the Jacobian \(^4\) for $U_i$ with respect to $\mathbf{\mu}_i$, and $\mathbf{P}$ is the update process noise. The Jacobian here is calculated numerically with a finite differences method (see Section 6.2.4 for more information). Note that the inclusion of the $\delta t$ term in the process noise term prevents multiple prediction updates for the same timestamp (when $\delta t = 0$) from changing the state more than a single prediction update.

\(^4\)Note that in the current implementation of Spooky (Chapter 7), the Jacobian is not computed for the prediction update in FusionProcedures.cpp: Node::getChainPredictedState. Rather, the Jacobian is simply approximated by the identity matrix. This will be fixed in future work.
6.2.3 Constrained Measurement Update

Figure 6.2 shows an example of optimizing the skeleton for minimal measurement error. Even in this simple case, there is more than one solution. Thus, a method for distinguishing under-constrained problems is required for stable inverse kinematic solutions. Furthermore, the human body has limited joint flexibility. To solve for under-constrained problems and model joint restrictions, constraint terms are included with each articulation. The constraint model is quadratic in the articulation state space. The constraint model has two parameters: centre and variance \((c, \Sigma_c)\) of the same dimension of the state of the articulation. These parameters are configured through UE4 blueprint functions. Spooky includes a good set of constraints for the human body defined in the default Spooky Skeleton (see Section 7.2.3).

A quadratic energy function is used to model the likelihood of the posterior state. Given the combined predicted state of a fusion chain \((\mu, \Sigma)\), the combined constraints for the articulations involved \((c, \Sigma_c)\), the measurement function \(M\) which maps the fusion chain state \(\mu\) to the measurement space, and the measurement itself \((z, \Sigma_z)\), define the pose energy \(E(\mu') \in [0, \infty)\) of a new state \(\mu' \in \mathbb{R}^n\) to be

\[
E(\mu') = \frac{1}{2} (\mu' - \mu)^T \Sigma^{-1} (\mu' - \mu) + \frac{1}{2} \lambda (\mu' - c)^T \Sigma_c^{-1} (\mu' - c) + \frac{1}{2} (z' - M(\mu'))^T \Sigma_z^{-1} (z' - M(\mu'))
\]  

(6.8)

(6.9)

(6.10)

where the parameter \(\lambda \in [0, 1]\) is called the stiffness, an intrinsic property of the skeleton configured by the user. The probability density of the state \(\mu'\) is then proportional to the negative exponential of the pose energy

\[
p(\mu') = \eta e^{-E(\mu')}
\]

(6.11)

where \(\eta \in \mathbb{R}\) is such that \(\int p(\mu')d\mu' = 1\). This construction is similar to the
measurement update for an Extended Kalman Filter (EKF), except that a typical EKF has no constraint term (Thrun et al. 2005).

The function $M$ is in general non-linear and thus the probability distribution $p(\mu')$ is non-gaussian. However, writing $\mu' = \overline{\mu} + \Delta \mu$, and linearising $M$ at $\overline{\mu}$ such that

$$
M(\mu') = M(\overline{\mu} + \Delta \mu) \approx M(\overline{\mu}) + J(M)(\overline{\mu})\Delta \mu
$$

the energy can be approximated by

$$
E(\mu') \approx \frac{1}{2} \Delta \mu^T \Sigma^{-1} \Delta \mu
$$

$$
+ \frac{1}{2} \lambda (c_0 + \Delta \mu)^T \Sigma_c^{-1} (c_0 + \Delta \mu)
$$

$$
+ \frac{1}{2} (z_0 - J_m \Delta \mu)^T \Sigma_z^{-1} (z_0 - J_m \Delta \mu)
$$

where

$$
c_0 = \overline{\mu} - c \quad (6.14)
$$

$$
z_0 = z - M(\overline{\mu}) \quad (6.15)
$$

$$
J_m = J(M)(\overline{\mu}) \quad (6.16)
$$

Equation 6.13 is quadratic in $\Delta \mu$, and so the associated probability distribution is Gaussian. The objective is then to find the mean and variance of this distribution and hence find an approximation for the posterior state $\mu', \Sigma'$.

To find the solution to this problem, a key property of the vector derivative will be needed.

**Property 1.** For $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times n}$ such that $A^T = A$, then

$$
\frac{d}{dx}(x^T A x) = 2x^T A \quad (6.17)
$$
Proof. Using tensor notation, where \( A^i_j \) is the \( ij \)th entry of \( A \) and the Einstein summation convention is followed, gives

\[
\frac{d}{dx} x^T A x = \frac{d}{dx^k} (x_i A^i_j x^j) = x_i \frac{d}{dx^k} A^i_j x^j = \delta_{ki} A^i_j x^j + x_i A^i_k \delta^j_k = A_{kj} x^j + x_i A^i_k \\
= x^T (A^T + A) = 2x^T A
\]

\[
\mu_0 = \Sigma_0 h \Sigma_1 \mu + \Sigma_1 c + J_m^T \Sigma_0^{-1} (z_0 - M(\mu) + J_m \mu)
\]

Property 1 allows the energy function in Equation 6.8 to be differentiated with respect to \( \mu' \). Under the linearisation of \( M \) the first derivative of the energy function has a single root which is also an approximation of the posterior mean \( \mu' \) (global minimum of a multivariate quadratic). Also, the second derivative of the energy will be constant and approximately equal to the inverse of the posterior covariance \( \Sigma' \).

**Result 1.** Constrained Measurement Update

\[
\Sigma' = \left[ \Sigma^{-1} + \lambda \Sigma_c^{-1} + J_m^T \Sigma_m^{-1} J_m \right]^{-1}
\]

\[
\mu' = \Sigma' \left[ \Sigma^{-1} \mu + \lambda \Sigma_c^{-1} c + J_m^T \Sigma_m^{-1} (z' - M(\mu) + J_m \mu) \right]
\]

These equations give the (approximate) maximum likelihood posterior state estimate defined by Equations 6.8 and 6.11, assuming that the measurement function \( M \) is (approximately) linear.
Proof. First apply Property 1 to Equation 6.8 to obtain

\[
\frac{dE}{d\mu'} = (\mu' - \bar{\mu})^T \Sigma^{-1} \tag{6.20}
\]

\[
+ \lambda (\mu' - c)^T \Sigma_c^{-1}
\]

\[
+ (z' - M(\mu'))^T \Sigma_z^{-1} \frac{d}{d\mu'}(-M(\mu')) \tag{6.21}
\]

where the measurement term in Line 6.21 follows from the differentiation chain rule.

Then, applying the linearisation of \(M\) from Equation 6.12

\[
\frac{d}{d\mu'}(-M(\mu')) \approx -\frac{d}{d\mu'}(M(\bar{\mu}) + J(M)(\bar{\mu}) \Delta \mu) \tag{6.22}
\]

\[
= -\frac{d}{d\mu'}(M(\bar{\mu}) + J(M)(\mu' - \bar{\mu}))
\]

\[
= -J(M)(\bar{\mu})
\]

Hence,

\[
\frac{dE}{d\mu'} = (\mu' - \bar{\mu})^T \Sigma^{-1} \tag{6.23}
\]

\[
+ \lambda (\mu' - c)^T \Sigma_c^{-1}
\]

\[
- (z' - M(\mu'))^T \Sigma_z^{-1} J_m
\]

Differentiating a second time gives

\[
(\Sigma')^{-1} \approx \frac{d^2E}{(d\mu')^2} = \Sigma^{-1} + \lambda\Sigma_c^{-1} + J_m^T \Sigma_z^{-1} J_m \tag{6.24}
\]

which is equivalent to Equation 6.18, as desired. The mean of \(p(\mu')\) under the linear approximation is where \(p(\mu')\) is maximum and \(E(\mu')\) is minimum. Thus, the mean is given by \(\frac{dE}{d\mu} = 0\). So, collecting like terms in Equation 6.23 gives

\[
(\mu')^T \left[\Sigma^{-1} + \lambda\Sigma_c^{-1} + J_m^T \Sigma_z^{-1} J_m\right] = \bar{\mu}^T \Sigma^{-1} + \lambda c^T \Sigma_c^{-1}
\]

\[
+ (z' - M(\bar{\mu}) + J_m\bar{\mu})^T \Sigma_z^{-1} J_m \tag{6.25}
\]
Transposing this result, remembering that all of the covariances are symmetric, and recalling the result of Equation 6.24,

\[(\Sigma')^{-1}\mu' = \Sigma^{-1}\mu + \Lambda\Sigma^{-1}_c c + J_m^T\Sigma^{-1}_m(z' - M(\mu) + J_m\mu)\]  \hspace{1cm} (6.26)

which is equivalent to Equation 6.19, as desired. \(\square\)

Result 1 is only an approximation unless \(M\) is linear, so for cases where linearity is a poor approximation, \(\mu'\) is optimised iteratively according to Algorithm 6.3. This algorithm iteratively computes the posterior according to Result 1, and then recomputes the linearisation of \(M\) centered at the new estimate, effectively searching the measurement function domain to find the minimum energy. The search ends prematurely if the energy increases (Line 10). The max search iterations was configured based on the expected complexity of the given optimization and is constant per measurement type. If the measurement type is positional or rigid body, then the max iterations is 2. Otherwise, a single step is used.

### 6.2.4 Computing the Jacobian

Computing the Jacobian for the chain measurement and update functions was performed numerically. The definition of the Jacobian for \(F: \mathbb{R}^n \to \mathbb{R}^m\) at the point \(x \in \mathbb{R}^n\) is given component-wise by

\[J(F)(x)_j^i = \frac{\partial F^i}{\partial x^j} \bigg|_x\]  \hspace{1cm} (6.27)

Thus, \(J(F)(x)\) is a matrix of dimension \(m \times n\) and a small change \(\delta x\) gives the approximate change in \(F\)

\[\delta F \approx J(F)(x)\delta x\]  \hspace{1cm} (6.28)
Algorithm 6.3: Iterative Measurement Update

Data: Max iterations \( N \in \mathbb{N} \), prior state \( (\mu, \Sigma) \), constraints \((c, \Sigma_c)\), measurement \((z, \Sigma_z)\), measurement confidence \( C_z \in (0.01, 1] \), stiffness \( \lambda \in [0, 1] \)

Result: List of nodes for fusion

\[
\begin{align*}
\text{begin} & & \\
& & \mu' \leftarrow \mu; \\
& & E_{\text{last}} \leftarrow \infty; \\
& & \text{lastState} \leftarrow (\mu, \Sigma); \\
& & \Sigma_z \leftarrow \Sigma_z / C_z; \\
& & \text{for } i = 1 \ldots N \text{ do} \\
& & J_m \leftarrow J(M)(\mu'); \\
& & \Sigma' \leftarrow \left[ \Sigma^{-1} + \lambda \Sigma_c^{-1} + J_m^T \Sigma_z^{-1} J_m \right]^{-1}; \\
& & \mu' \leftarrow \Sigma' \left[ \Sigma^{-1} \mu + \lambda \Sigma_c^{-1} c + J_m^T \Sigma_m^{-1} [z' - M(\mu') + J_m \mu'] \right]; \\
& & \text{if } E(\mu') > E_{\text{last}} \text{ then} \\
& & \quad // \text{Last estimate was better, return that:} \\
& & \quad (\mu', \Sigma') \leftarrow \text{lastState}; \\
& & \quad \text{break; } \\
& & \text{else} \\
& & \quad E_{\text{last}} \leftarrow E(\mu') ; \\
& & \quad \text{lastState} \leftarrow (\mu', \Sigma'); \\
& & \text{end} \\
& & \text{return } (\mu', \Sigma'); \\
\text{end} \\
\end{align*}
\]

Now, a fusion chain of length \( k + 1 \) has global space transform defined as a function of the combined chain state \( \mu \) by

\[
G(N)(\mu) = G(P^{k+1}(N)) \cdot L(P^k(N))(\mu_k) \ldots \cdot L(P(N))(\mu_1) \cdot L(N)(\mu_0) \quad (6.29)
\]

This equation defines the measurement function for most of the measurement functions for the fusion model, and computing the Jacobian often is necessary. For example, a position measurement is given by

\[
M_{\text{pos}}(\mu) = G(N)(\mu)x_0 \quad (6.30)
\]
where \( x_0 = [0, 0, 0, 1]^T \). To perform the Jacobian computation, the following optimisation was implemented. Since each transform \( L(P_i^i(N)) (\mu_i) \in SE(3) \) depends only on a small range of values within the combined state \( \mu = [\mu_0^T, \ldots, \mu_k^T]^T \), the Jacobian can be computed locally and combined to give the full Jacobian. Considering the \( q \)th component of the \( \mu_i \) derivative, we have

\[
\frac{\partial}{\partial \mu_i^q} G(N)(\mu) = G(P^{k+1}(N)) \cdot L(P^k(N))(\mu_k) \cdot \ldots \cdot L(P^{i+1}(N))(\mu_{i+1}) \cdot \left[ \frac{\partial}{\partial \mu_i^q} L(P^i(N))(\mu_i) \right] \cdot \ldots \cdot L(P^1(N))(\mu_1) \cdot L(N)(\mu_0)
\]

(6.31)

(6.32)

Then choosing\(^5 h << 1\) the local derivative can be approximated by

\[
\frac{\partial}{\partial \mu_i^q} L(P^i(N))(\mu_i) \approx \frac{1}{h} \left[ L(P^i(N))(\mu_i + he_q) - L(P^i(N))(\mu_i) \right]
\]

(6.33)

where \( e_q \) is the \( q \)th unit vector in the local state space. Furthermore, by incrementally calculating the undifferentiated left and right wings of Equations 6.31 with increasing \( i \), data re-use reduces the time complexity of the computation significantly even for large chains.

### 6.3 Evaluation

Evaluation of the proposed algorithm is given in Chapter 8, where a user study is performed to evaluate the fusion of the Leap Motion and Perception Neuron hand tracking systems. Results suggest that the algorithm is effective at generalising the component tracking systems to a wider variety of tasks compared to the component systems.

\(^5h = 0.0001\) was chosen in the implementation of Spooky.
6.4 Reflection and Future Work

This section highlights some of the shortcomings of the proposed iterative constrained EKF algorithm. Future work is discussed to address these issues.

6.4.1 Confidence Modelling

The confidence value $C_z \in [0, 1]$ of each measurement represents the trustworthiness of each measurement. If the value falls below a given threshold (0.01 by default) then the measurement is not fused at all. Otherwise, the confidence varies the magnitude of the measurement covariance (Algorithm 6.3, Line 5). Modeling the covariance and confidence values for a given sensor is a key limitation of the system. Often the software interface of a sensor system gives only the measurement data, without adequate modelling of the covariance nor confidence. This manifests as instability and reduced tracking quality even when a reliable system is available. It is a challenge for a sensor to know when its own information is unreliable without external reference.

Particular attention to detail was given to the confidence and noise model for the Leap Motion used in the user study in Chapter 8. Without this careful programming, the Leap Motion would cause large errors in tracking when it was tracking ambiguous states. This is undesirable for a modular system which is aiming to be usable by non-programmers or simply those without time to configure each new sensor they get. Future work will investigate machine learning methods to determine confidence models automatically. In particular, new methods for reasoning about knowledge confidence, like the newly proposed Logical Induction, indicate a promising direction of research Garrabrant et al. 2016.
6.4.2 Advanced Fusion Chain Selection

The proposed method for chain selection is simple and constant for a given node type and measurement type. However, the procedure is a coarse approximation. For example, if position measurements are fused to a chain of bone type articulations, three bones will be selected for the fusion chain. However, if the measurement position is further away than the sum of the lengths of the two parent bones, the point cannot be reached. In this case, the result is unstable over successive frames as the algorithm attempts to minimise the error each frame. A more intelligent algorithm would modify more parent nodes in this case, taking into account both prior state probability and constraint energy.

Here, analysis of this problem is formalised and future research directions proposed. Let \( n \in \mathbb{N} \) be the dimension of a complete fusion chain, from node to root, so that \((\mu, \Sigma) \in \mathbb{R}^n\) describes the current pose of the fusion chain. Note that the states are encoded within \( \mu \) such that the node being fused has its state first, followed by its parent, and so on, finishing with the root node state. Define \( P_k : \mathbb{R}^n \to \mathbb{R}^k \) be the projection map of dimension \( k < n \) which operates by discarding the tail of the vector. That is,

\[
P_k ([x_1, x_2, \ldots, x_n]^T) = [x_1, x_2, \ldots, x_k]^T \tag{6.34}
\]

and thus, the Jacobian of \( P_k \) is the \( k \times n \) identity matrix \( I_{kn} \). The energy function (Equation 6.8) of \( P_k(\mu) \) is then given by

\[
E_p(P_k(\mu')) = \frac{1}{2} P_k(\mu' - \bar{\mu})^T [I_{kn} \Sigma I_{nk}]^{-1} P_k(\mu' - \bar{\mu}) \tag{6.35}
\]

\[
+ \frac{1}{2} \lambda P_k(\mu' - c)^T [I_{kn} \Sigma_c I_{nk}]^{-1} P_k(\mu' - c) \tag{6.36}
\]

\[
+ \frac{1}{2} (z' - M(P_k(\mu')))^T \Sigma_z^{-1} (z' - M(P_k(\mu'))) \tag{6.37}
\]
The problem is to find the minimum $k > 0$ such that

$$E_p(P_k(\mu')) \leq E_0$$ (6.38)

for some ideal energy $E_0$. If a probability density of $p(\mu') \geq p_0$ is desired, then the energy threshold $E_0$ can be chosen to be

$$E_0 = -\log(p_0)$$ (6.39)

In this case, simply repeating the fusion computation with increasing values of $k$ would be too expensive. However, there may be some way to estimate when the condition in Equation 6.38 is satisfied without performing the full optimisation procedure. Further investigation is left for future work.

### 6.4.3 Managing Off-Diagonal Covariance

Currently the off-diagonal covariances values from the fusion are discarded when the state of each articulation is updated with the fusion result. This assumes that the pose of each articulation is independent of each other articulation. Better accuracy could be achieved by storing these articulation correlations in a table, and retrieving them when both articulations appear in the fusion chain again. Computationally this would not cost significant time. However, it is difficult to know how much effect this would have, hence this feature is left for future work since the programming work involved would not be insignificant.
Chapter 7

Spooky: A Sensor Fusion Plugin

Previous chapters have described the design of several ambient fusion algorithms for automating and enhancing sensor fusion in VR settings. This chapter details the implementation of many of the proposed algorithms within an open source C++ plugin, Spooky. The plugin is designed to provide sophisticated sensor fusion functionality to game engines and existing middlewares. Currently, Spooky supports Unreal Engine 4 with future plans for expansion to other popular game engines and middlewares like SteamVR, and in the future OpenXR. Spooky will require significant further work before it is truly ‘plug-and-play’. This project has prioritised research over deployment. This chapter and the open-source repository¹ are included for reference and convenience to those who may benefit from them.

7.1 Software Implementation

Spooky is written in C++ using only the standard C++11 libraries and the Eigen mathematics library (Guennebaud and Jacob 2010). Figure 7.1 summarizes the structure of the system. There are three central software modules within the system.

¹http://github.com/JakeFountain/Spooky
- the Correlator, the Calibrator, and the Fusion Graph. The Fusion Graph models an articulated skeleton with support for sensor fusion as detailed in Chapter 6. The Fusion Graph is the central module - its structure, a tree of nodes representing rigid bodies, defines how the other two modules work. Each node inherits the transform of its parent and is updated based on the fusion of the latest measurements assigned to the node. The Correlator is responsible for determining correspondences between ambiguous sensors - sensors which could be attached to one of many nodes in the Fusion Graph. Details for the Correlator were given in Chapter 3. The Calibrator is responsible for determining the transforms between different sensor systems and detecting faults in calibration. The inner workings of the Calibrator were discussed in Chapter 5. These modules represent the three steps in ambient sensor fusion identified in Chapter 2 and the implementation of these three features are included in Spooky. Chapter 4 discussed a new, model-less method for analysing arbitrary signals. This work is in earlier stages and thus it is not yet included in the Spooky system.

The Interface component shown in Figure 7.1 serves the function of passing data

![Figure 7.1: Software architecture of the fusion plugin.](image-url)
to the submodules. Each frame, through the interface, data is added through calls to `addMeasurement` functions. Sensor measurements are stored within the system as C++ shared pointers. These are pointers to memory which are automatically cleaned up whenever all associated references are deleted, dramatically simplifying data caching code. Additionally, nodes for the Fusion Graph are configured at program start through the Interface with calls to `addNode` functions. The parameters provided through `addNode` and `addMeasurement` calls define how the data is fused.

Adding measurements and bones one at a time through code is not particularly user friendly, and so UE4 features were used to supplement the C++ code of Spooky.

### 7.2 Spooky UE4 Features

The Spooky UE4 plugin was built to provide a test-bed for the proposed algorithms and allow for other developers to more easily integrate Spooky into their projects. This section describes some innovative features of Spooky UE4, and how these features have been designed for future scaling up to ubiquitous automatic sensor fusion.

The structure of the UE4 plugin is centered around the Spooky Fusion Plant - a UE4 Blueprint Component. This component can be added to any object to provide in-engine access to Spooky sensor fusion functionality. Another key blueprint class in Spooky UE4 is the `Spooky Skeletal Mesh Component`. This component inherits from the UE4 Skeletal Mesh Component, the default option for animating articulated bodies in UE4. The Spooky Skeletal Mesh Component includes all of the usual features of a Skeletal Mesh Component, but also has dedicated meta-data for each bone to allow for easy configuration of sensor input and fusion. For example, confidence values are stored for each bone. If a device which is to be fused has an animation blueprint and skeletal mesh associated with it, the skeletal mesh can be replaced by a Spooky skeletal mesh with minimal effort. After a little configuration, this skeleton can be essentially dragged and dropped into the Spooky Fusion Plant.
Figure 7.2: Data flow of the Spooky Skeleton Fusion Plugin within Unreal Engine 4.

as an input, and it will be fused with the other measurements or skeleton inputs (Figure 7.2). Retargeting between skeletons, transferring an animation from one skeleton to a different target skeleton, is also supported. To drive an avatar with fused Spooky data, you must simply re-target the default Spooky animation blueprint to your own skeletal mesh or to another Spooky Skeletal Mesh Component. In this way, a Spooky Skeletal Mesh Component also doubles as an input for fused Spooky data. Therefore, Spooky Skeletal Mesh Components can form a graph of skeletons, with data flowing from one to the next, being fused by Spooky along the way.

7.2.1 Spooky Fusion Plant

The Spooky Fusion Plant (SFP) class (USpookyFusionPlant) is a wrapper for the spooky::Core class, written in C++ and defined in SpookyFusionPlant.h. The class consists of an instance of the spooky::Core class, plus input and output
methods which translate between generic C++ data structures and UE4 specific data structures. A single SFP is intended to manage the data of a single local user. At startup a model for the user skeleton must be configured by listing a hierarchy of rigid body nodes with associated articulations as described in Chapter 6. This is performed by providing one or more USpookySkeletalMeshComponents to the USpookyFusionPlant::AddOutputTarget member function. At runtime, each skeleton is traversed and configured within spooky::Core based on the configuration of the USpookySkeletalMeshComponent (see Section 7.2.3).

### 7.2.2 Measurements

Adding measurements to a SFP can be done using two methods. The first is adding singular measurements through calls to USpookyFusionPlant::AddMeasurement functions. Different types of measurements each have their own AddMeasurement function (e.g. AddRotationMeasurement). In the case of Unreal Engine 4, this is done through calls to blueprint functions. Blueprints are a graph based graphical programming language used within Unreal Engine 4. Nodes represent functions and data is passed via the graph edges. Therefore, no C++ code needs to be written. Figure 7.3 shows a call to AddPoseMeasurement which adds a pose measurement. This function is called every time new tracking data is available.

### 7.2.3 Spooky Skeletons

A Spooky Skeleton is a more convenient method for sending measurements to Spooky. It relies on the standard Animation Blueprint feature of UE4. Many tracking device plugins, including Leap Motion and Perception Neuron, animate a skeleton through an animation blueprint. Spooky provides a custom class for configuring meta-data about the pose of the animated skeleton: SpookySkeletalMeshComponent. This class allows for setting of sensor confidences and variances, bone
constraints and retargeting transforms for mapping to other different skeletal meshes. Provided with the Spooky repository is a set of useful pre-made blueprint classes for typical use cases. The blueprint class `Content/Spooky/Skeletons/SpookySkeleton` implements a useful default model for fusion targetting. It includes the set of constraints implemented and used in the User Study (Chapter 8).

The process of adding target nodes for Spooky to fuse can be done through blueprints, one bone at a time. More conveniently, an entire skeletal mesh can be targeted. This adds a bone node (see Chapter 6) to Spooky for every bone in the skeleton. Configuration information must be set with blueprint calls. The fusion result can then be applied to an instance of the skeletal mesh by calling the Animation Blueprint node `GetSpookyResult` in the Skeletal Mesh’s Animation Blueprint (Figure 7.4). This updates the skeleton pose with the latest Spooky result every

![Figure 7.3: An example of adding a sensor measurement to Spooky through Unreal Engine 4 blueprints. This method is called whenever new sensor information is available. The sensor is configured to have a known node location - the left hand.](image)
Figure 7.4: Example Animation Blueprint calls to retrieve the fused Spooky result.

frame.

Constraints are configured relative to the default pose of the Skeletal Mesh. Constraints consist of a zero-strain point and a flexibility matrix representing the amount which the bone pose can vary from the zero-strain point. The constraints allow for otherwise under-constrained inverse kinematic problems to be solved emergently. For example, knowing just a wrist pose relative to the shoulder leaves the elbow position unconstrained. If no constraints are configured, then multiple solutions exist to an underconstrained fusion problem, and the algorithm becomes unstable. See Section 6.2.3 for more information about constraints.

7.2.4 Spooky Spirits

A Spooky Spirit is a blueprint class which can be fed into the Spooky Fusion Plant to provide sensor data. Spooky Spirits inherit the SpookySpiritInterface (Content/Spooky/Interfaces/SpookySpiritInterface), which provides functions which will be called when connecting to a Spooky Fusion Plant. A Spooky Spirit stores a reference to its assigned Spooky Fusion Plant. A Spooky Spirit can add measurements through either blueprint calls to the Spooky Fusion Plant, or through a Spooky Skeleton. The role of the Spooky Spirit is to convert sensor data into Spooky form, including providing variances, data types and confidences. Also, a Spooky Spirit may perform additional computations to determine these parameters. For example,
in the user study, a model for the Leap Motion measurement confidence is implemented in its Spooky Spirit (Section 8.1.3).

### 7.2.5 Spooky Graveyards

A Spooky Graveyard combines multiple Spooky Spirits, plus one or more Spooky Skeletons, into a complete avatar system which can be dragged and dropped into an existing UE4 level. A Spooky Graveyard inherits from the blueprint class `BaseGraveyard` (Contents/Spooky/Graveyards/BaseGraveyard). A Spooky Graveyard has a `SpookySpirits` list member variable and a member variable called `GraveyardSpookySkeleton`. A Spooky Spirit added to the `SpookySpirits` list will be automatically fused into the target `GraveyardSpookySkeleton` (see Figure 7.5). Interaction logic can be added on top of the `GraveyardSpookySkeleton` to allow for actions such as picking up virtual objects (for example, see Section 8.1.2). Figure 7.6 shows the class structure of a Spooky Graveyard.

![Figure 7.5: Example configuration of a Spooky Graveyard for fusion of the Leap Motion, Perception Neuron and Oculus Rift (as used in Chapter 8).](image)
Chapter 7. Spooky

Figure 7.6: Spooky Graveyard class structure within a UE4 level. Items labelled in typewriter font are example objects owned by the surrounding box.

7.3 Reflection and Future Work

Spooky is an Unreal Engine 4 plugin written in C++ with minimal dependencies. Spooky has a convenient interface for creating fused sensor systems using the Unreal Engine Editor. The plugin targets developers, researchers, businesses and enthusiasts who wish to develop software which uses a variety of tracking systems in unison. Spooky functions through the algorithms proposed in Chapters 3, 5 and 6. Future work will include extending the capabilities of Spooky to other game engines and VR software middlewares. Spooky is open-source and can be found at http://github.com/JakeFountain/Spooky.
Chapter 8

User Study

Chapter 6 proposed a method for modular skeletal fusion with constraints to address the third and final step of the ambient sensor fusion problem. Chapter 7 described implementation of the proposed fusion algorithm in the form of an Unreal Engine 4 plugin, Spooky. This chapter evaluates the effectiveness of the proposed fusion algorithm when applied to the context of hand tracking. A user study was performed to evaluate objective and subjective quality of the fusion of two commercial grade tracking devices: Leap Motion and Perception Neuron. The Leap Motion tracks the user’s hands from a pair of infra-red cameras attached to the front of a HMD (Figure 8.1). The Perception Neuron tracks a user’s hands using a network of IMUs.

![Figure 8.1: Oculus Rift with Leap Motion attached.](image)
attached to the body and hands. The Leap Motion is accurate, but suffers from occlusion and limited tracking volume - the user’s hands must be in front of the HMD. Conversely, the Perception Neuron doesn’t require line of sight and tracks without interruption, but the IMUs gather significant drift under fast motion, and difficulties exist in calibrating the home position of the sensors. The complementary nature of these devices was identified as an ideal test scenario for the fusion algorithm. The experimental hypothesis is that the fused tracking solution will achieve higher accuracy and range compared to the component tracking solutions for a range of tasks.

Study participants \((n = 18)\) completed a series of tasks in VR using the Leap Motion\(^1\) and Perception Neuron\(^2\) tracking systems. The Oculus Rift HMD was used to track the participants head and display the virtual environment during each experiment. During each task, the tracking solution was varied and data about both task performance and participant preference was collected.

### 8.1 Method

This section describes the process of evaluating the proposed fusion algorithm. First, the experiment was planned and submitted for evaluation to the University of Newcastle Human Research Ethics council in late 2017. After minor revisions to the experimental plan, ethics approval was granted in early 2018\(^3\). See Appendix A for the ethics documents associated with the experiment. Four students were recruited in March of 2018 to participate in the study. However, a software problem was identified in the fusion algorithms used for these participants. Once this problem was corrected, the algorithm was finalised for the remainder of the participants.

---

\(^1\)Leap Motion Orion software version 3.2.1

\(^2\)Axis Neuron software version 3.6.32

\(^3\)HREC Approval Reference Number: H-2017-0391 (approved 25-Jan-2018)
Thereupon, 18 students were recruited during the period of April-June 2018 for the final user study. The results of the user study are presented in Section 8.2.

8.1.1 Tracking Setup

![Image showing default and custom gloves compared to Leap Motion tracking]

**Figure 8.2:** Default Perception Neuron glove (*right*) compared to the custom glove used in this study (*left*) as seen by the leap motion infrared cameras with tracking result overlayed. The default glove is not tracked at all.

During the study, the participants wore a HMD (Oculus Rift) with Leap Motion (Figure 8.1) and the arm and hand sections of the Perception Neuron inertial tracking system. The default glove which the Perception Neuron suit is packaged with is not visible to the Leap Motion infra-red camera because it absorbs too much infra-red light, so a custom glove was built to house the sensors. Figure 8.2 shows how the Leap Motion infra-red camera cannot track the glove from the original Perception Neuron compared to the tracking of the custom glove. The Perception Neuron sensors themselves also absorbed infra-red light and interfered with the Leap Motion tracking, so the custom glove was designed to cover most of the sensors (Figure 8.3). The custom glove was built from two white cotton gloves - one to act as the anchor point for the Velcro-backed sensors, and one glove to cover the sensors. The fingers of the covering glove were separated from the palm to increase flexibility. The arm
Figure 8.3: The custom built glove for the right hand, allowing simultaneous Leap Motion and Perception Neuron hand tracking. Left image shows the sensor arrangement, and the right image shows the final glove appearance with outer-layer.

Sections of the modular Perception Neuron motion capture suit were used, including all finger tracking modules (Figure 8.4 shows exactly which sensors were worn). This allows for the arms and fingers to be tracked with 20×3DoF IMUs.

Three tracking solutions were tested:

- *Leap Motion* alone (LP)
- *Perception Neuron* alone (PN)
- *Fused Tracking* of both systems together (FT)

In all tracking cases, the UE4 Mannequin default skeletal mesh was animated by the tracking data (Figure 8.5). This skeleton is then able to interact with the virtual environment rendered by Unreal Engine 4 (UE4), within which the tasks were completed. The position of the torso was determined by the position of the head as tracked by the Oculus Rift. The PN and FT cases included tracking of the shoulders to fingertips, while the LP case only tracked the elbows to fingertips.
8.1.2 Grip Detection

Grasping of objects by a given hand was computed based on the distance between the thumb and the fingers of the animated skeleton. This functionality was implemented in the UE4 blueprints programming language. When any fingers were closed such that the distance between the thumb and the closest finger was less than 5cm, a grasp was initiated, attaching the nearest graspable object to the hand bone and disabling physics simulation for that object (Figure 8.6). The object must also be no more than 5cm from the midpoint of the engaged fingertips. If this distance was exceeded by all fingers, the object was detached and physics simulation re-activated. The velocity of a grasped object before and after release are conserved, allowing for actions like throwing. If one hand was used to grasp an object already being grasped by the other, it would be released by the current hand and grasped by the new hand.

Figures 8.7 to 8.9 outline the different software configurations for each tracking case, inside of UE4. The Leap Motion configuration simply animates the hands of the UE4 Mannequin in the head tracked reference frame directly, and then grip detection is performed on the resulting skeleton. A Spooky Fusion Plant was used for the Fused Tracking solution to combine the two systems (see Chapter 7). A Spooky Fusion

![Image of hands and sensors](image)

**Figure 8.4:** Example screenshot from Perception Neuron software, with the connected sensors shown on the left in green or yellow (depending on their connection strength). An example tracking result is shown on the right.
Plant was also used to combine the Perception Neuron and Oculus Rift data for the Perception Neuron tracking solution. The function performed by the Fusion Plant in the latter case is simply to retarget the default Perception Neuron skeleton animations and Oculus Rift position data to the standard UE4 Mannequin. No Fusion is performed in this case because the data does not overlap. The Fused Tracking case includes a “Leap Motion Noise Model” component, which is discussed in Section 8.1.3.

8.1.3 Leap Motion Confidence Model

The Leap Motion relies on camera sensors positioned on the front of a HMD to track hands and fingers. This approach suffers from occlusion artifacts such as ambiguous poses and obscured information. Simply taking the mean of two such erroneous tracking results leads to poor tracking. Thus a model of uncertainty is required to effectively fuse the data. The likelihood model used for this user study was defined as a function of reported hand pose from the Leap Motion. The architectural position of this module is shown at the top of Figure 8.9. Two key confidence values $C_{pos}, C_{rot} \in [0, 1]$ modeled the bulk position and rotation confidences of the hand.
Chapter 8. User Study

Figure 8.6: Example grip action. If the distance between the thumb and any of the fingers $d$ becomes less than 5cm, the closest object within 5cm of the midpoint of the engaged fingertips becomes gripped.

Figure 8.7: Software structure for the Leap Motion (LP) solution used for the user study, within Unreal Engine 4.

Figure 8.8: Software structure for the Perception Neuron solution for the user study, within Unreal Engine 4. The role of the Spooky Fusion Plant here is to retarget the animation from the PN skeleton to the UE4 Mannequin, and use the Oculus Rift data to position the avatar correctly.
respectively. See Algorithms 6.1 and 6.3, and Section 6.4.1 for information on how the confidence values are used within the proposed fusion system.

The position confidence was defined by

\[ C_{\text{pos}} = C_{\text{Leap}} \exp \left( - \frac{||\vec{x}_{\text{wrist}} - [300, 0, 0]||^2}{(300)^2} \right) \]

(8.1)

where \( \vec{x}_{\text{wrist}} \in \mathbb{R}^3 \) was the reported position of the wrist relative to the Leap Motion coordinate system, \( C_{\text{Leap}} \) is the confidence reported by the Leap Motion software, and the highest confidence point \([300, 0, 0]\) is 30cm in front of the Leap Motion sensor. This position uncertainty falls to zero as the wrist moves to the peripheral of the Leap sensor field of view. It should be noted the self reported confidence \( C_{\text{Leap}} \) of the Leap Motion was not sufficient to correctly model the reliability of the sensor - it was merely a monotonically increasing function of duration that the hand
had been continuously seen.

The rotation confidence value was partially based on the extension of the fingers. The Leap Motion UE4 interface allows for polling of the state of extension of all of the fingers. When the fingers are extended, the orientation of the hand becomes more stable as the fingers provide stable reference points. Thus the confidence value for finger extension for a given hand was computed linearly as

\[
C_{\text{grasp}} = 0.8 \frac{n_{\text{ext}}}{5} + 0.2
\]

where \(n_{\text{ext}} \in \{0, 1, \ldots, 5\}\) is the number of fingers extended for the hand.

The orientation confidence was then defined by

\[
C_{\text{rot}} = C_{\text{pos}} C_{\text{grasp}} C_{\text{palm}}
\]

where the palm direction confidence \(C_{\text{palm}}\) was computed from the normal direction of the palm \(n_{\text{palm}}\) reported by the Leap Motion software such that

\[
\theta_{\text{palm}} = \cos^{-1}(n_{\text{palm}} \cdot v_{\text{look}})
\]

\[
C_{\text{palm}} = \max \left\{ \left[ 1 - \exp \left( -\left( \frac{\theta_{\text{palm}} - \pi}{2}\right)^2/(\pi/3)^2 \right) \right] , 0.1 \right\}
\]

Here, \(v_{\text{look}}\) is the unit vector pointing along the view direction of the participant as tracked by the HMD. A plot of this function is given in Figure 8.10. The minimum confidence is obtained when the hand normal and view direction are orthogonal. This models the case when the Leap Motion camera is viewing the hand from within the plane of the palm, and the fingers are easily obscured or ambiguous.

In addition to the above confidence signals, a confidence signal was computed for individual finger bones. The line of sight confidence, computed for each finger bone
b_f, was given by
\[
C_{\text{los}}(b_f) = \begin{cases} 
1 & \text{if } b_f \text{ visible} \\
0.25 & \text{if } b_f \text{ obscured}
\end{cases}
\]

A finger bone was considered visible if a line trace from the head to the finger bone, performed with UE4 functionality, struck the bone first. If it struck another bone first, it was considered obscured.

The final confidence for finger bone b_f belonging to hand h was given by
\[
C(b_f) = \begin{cases} 
C_{\text{pos}}(h)C_{\text{los}}(b_f) & \text{if } b_f \text{ measured this frame} \\
0 & \text{if } b_f \text{ not measured this frame}
\end{cases}
\]

where determination of whether or not the bone was measured was based on whether or not new data was received for it from the Leap Motion plugin. The term C_{\text{los}} was necessary because even when the fingers were not visible the Leap Motion system still would send data as though they were.
8.1.4 Tasks

Three tasks in VR were used to assess the quality of the hand tracking. The tasks were chosen to span a range of possible interactions which are easy to perform in the real world. High quality hand tracking should allow for analogous tasks to be performed in VR, even by novice users. The tasks also represent typical tasks present in VR user interfaces, including reaching, selection, grasping, manipulation, and dynamic actions. The tasks were implemented in UE4, utilising the built in graphics, collision and physics simulation systems. During the experiment, participants were seated in a stable chair, and each task appeared in front of them one at a time. For each task, two key data were recorded: score and slip count. The score is an integer which represents how well the participant performed. The slip count represents the number of times the participant was unable to correctly operate the tracking system to complete the task. Since the tasks are simple and participants received real-world training beforehand, it is assumed that the majority of misses represent the slip count, not user mistakes.\footnote{A mistake is when a user misunderstands the correct action to take, whereas a slip is when a user understands the correct action but does not perform it correctly.}

The first task was called the Keyboard Task. Participants were presented with a virtual keyboard approximately 1 metre wide and 40cm deep, angled at 45° toward the participant (Figure 8.11). The keyboard had 96 identical hexagonal keys, each
Figure 8.12: Sorting Task

5cm wide (measured side-side), spaced evenly. During the task, a random key glows white, indicating the target to the participant. If the participant touches the target key successfully, the key turns green for 1 second and then a new target key is indicated. If the participant touches an incorrect key, the touched key turns red to provide negative feedback for 1 second, and then a new target key is indicated. This process repeats for the time limit of the task (90 seconds). The task was explained to the participant as ‘Press as many correct keys as possible within the time limit’. The data recorded for this task was the number of correct key presses, representing the score for this task, and the number of incorrect key presses, representing the slips for this task.

The second task was called the Sorting Task. This task involved picking up and inspecting a randomly generated virtual object, followed by sorting of the object into its corresponding category (Figure 8.12). The properties of the object were chosen such that all sides of the object need to be inspected. Cubes were chosen for the objects, with sorting based on the following properties:
• Colour

• Number of dots (odd or even)

Each cube was 10cm wide, with a diffuse material of uniform colour. The dots were placed on the faces of the cube randomly, with a uniform probability distribution across each face. The colour of a cube could be red, green or blue. The number of dots on a cube was either two or three. During the task, a cube would spawn on a virtual table in front of the participant. When the cube was sorted (correctly or incorrectly), or the cube was dropped on the ground, a new cube would spawn. When a cube was placed into a container, a glowing light was used to indicate correct (green) or incorrect (red) responses. Cubes were sorted into one of four containers: [Red,Odd], [Green,Odd], [Blue,Odd], or [Any Colour,Even]. The three odd containers were smaller and directly in front of the participant, behind the spawn point of the cubes. The even container was approximately twice as large and placed to the left of the cube spawn point. The labels for the containers were displayed in large text above each container. The objective of the task was described to participants as ‘Sort as many cubes correctly as possible within the time limit’. The data recorded for this task included the number of cubes correctly sorted, representing the score for this task, and the number of cubes dropped on the ground, representing the slips for this task. While it is possible that a slip can result in a dropped cube landing in a box, this case is unlikely and is thus ignored.

The final task was called the *Throwing Task*. This task involved throwing virtual balls at a large virtual target (Figure 8.13). The target was placed approximately two meters from the participant, flat on the ground. The target was 130cm in radius, with concentric disks of varying colour indicating the centre. A white ball of 10cm diameter spawned on a small table in front and to the right of the participant. When the ball hit the ground, a new ball was spawned, and the ball landing location recorded. The objective of the task was described to the participant as ‘Throw as many balls as close as possible to the centre of the target as possible.’ The data
recorded for this task included the number of balls which landed anywhere on the target, representing the score for this task, and the number of balls which did not land on the target, representing the slips for this task. Additionally, the precise location of ball landings was recorded for supporting analysis (Section 8.2.2).

8.1.5 Experiment

For each participant, each task was completed three times, with a different tracking solution each attempt. Participants were assigned an integer identification number (ID) starting with participant 1 and increasing chronologically with order of participation. The presentation of tracking technologies was counterbalanced according to Table 8.1, with the table repeated for participants with ID greater than 12. This counterbalancing ensures that every 12 participants see each of the 6 permutations.

\textsuperscript{5}It is assumed that with perfect tracking a participant would achieve practically no misses of the large 2.6m diameter target. This is supported by the fact that during training all participants demonstrated the ability to land a real ball within 30cm of a similarly positioned target on their first throw.
Table 8.1: Example Tracking Presentation Orders

<table>
<thead>
<tr>
<th>Participant ID</th>
<th>Keyboard Order</th>
<th>Sorting Order</th>
<th>Throwing Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L P F</td>
<td>F L P</td>
<td>P F L</td>
</tr>
<tr>
<td>2</td>
<td>P F L</td>
<td>L P F</td>
<td>F L P</td>
</tr>
<tr>
<td>3</td>
<td>F L P</td>
<td>P F L</td>
<td>L P F</td>
</tr>
<tr>
<td>4</td>
<td>F L P</td>
<td>L P F</td>
<td>P F L</td>
</tr>
<tr>
<td>5</td>
<td>P F L</td>
<td>F L P</td>
<td>L P F</td>
</tr>
<tr>
<td>6</td>
<td>L P F</td>
<td>P F L</td>
<td>F L P</td>
</tr>
<tr>
<td>7</td>
<td>P L F</td>
<td>F P L</td>
<td>L F P</td>
</tr>
<tr>
<td>8</td>
<td>L F P</td>
<td>P L F</td>
<td>F P L</td>
</tr>
<tr>
<td>9</td>
<td>F P L</td>
<td>L F P</td>
<td>P L F</td>
</tr>
<tr>
<td>10</td>
<td>F P L</td>
<td>P L F</td>
<td>L F P</td>
</tr>
<tr>
<td>11</td>
<td>L F P</td>
<td>F P L</td>
<td>P L F</td>
</tr>
<tr>
<td>12</td>
<td>P L F</td>
<td>L F P</td>
<td>F P L</td>
</tr>
</tbody>
</table>

Key: L = Leap Motion, P = Perception Neuron, F = Fused Tracking

on the 3 elements \{L, P, F\} twice for each task. For each of the two instances which one technology sees a particular permutation $\sigma$, the permutations for the other technologies are the two 3-cycles (aka shifts) of $\sigma$ in their two possible configurations. This gives the following symmetries:

- Each participant sees a latin square of technology orderings.
- For every 3 participants, each task sees a latin square of technology orders.
- For every 6 participants, the tasks see all permutations of the 3 orderings of three tracking technologies.
- For every 12 participants, the tasks see all permutations of the 2 sets of 3 orderings of three tracking technologies.

Before each experiment, participants were trained for up to 5 minutes on each of the tasks in the real world. They were first shown an image of the keyboard from the Keyboard Task with a target key lit up (as shown in Figure 8.11), and the objective was explained. Next, the Sorting Task was explained to them, and they practised on a real-world version of the task which used coloured paper squares with dots
on them (Figure 8.14). The squares had dots on both sides, demonstrating to the participant that they needed to check all sides of the cubes in the VR Sorting Task. If the participant failed to sort the six paper squares successfully, the task was re-explained and they could re-attempt until they succeeded. Finally, the participants were shown a printed version of the Throwing target placed approximately 2m from them. They were told the target would be much bigger in VR, and then participants were allowed to have a practise throw of a real ball at the real target (Figure 8.15). In the cases of the Sorting and Throwing task, it was explained to the participants that grasping an object was based on the distance between their thumb and fingers, rather than grasping the surface of the object.

Before beginning the experiment, participants were told to pay attention to the following two factors:

- **Perceived Accuracy**: ‘Pay attention to how closely each tracking system matches your real hand.’
After each task, the participant was stopped and asked to rank the tasks in order of perceived accuracy and utility to provide subjective measures of tracking quality. Attempts were labeled anonymously in chronological order A, B and C. An open ended question was also asked at the end of each task to allow the participants to describe any problems they noticed in detail:

‘Did you find anything frustrating during any of attempts A, B or C?’

Before each task, the Perception Neuron system was re-calibrated to account for drift. This involved asking the participant to assume an A-pose (arms by sides) and then a T-pose (arms parralel to the horizon), while seated (Figure 8.16). The first calibration was done without the HMD to allow for guidance from the researcher.
Figure 8.16: Calibration poses performed before each task (marked with a checkmark). Since only the upper body of the participant was tracked, the participant was allowed to sit during calibration.

However, once the calibration procedure was understood by the participant, successive calibrations were performed while the participant wore the HMD. Calibration was performed this often to give a fair result for the PN tracking case - otherwise the drift of the system can render it unusable during the later tasks. The PN space was hand aligned with the HMD space. This alignment simply involved a yaw offset of the forward direction for the Perception Neuron to match that of the VR system.

Prior to recruitment of participants, the experimental procedure was approved by the Human Research Ethics Committee of The University of Newcastle, Australia. Participants were recruited from the School of Computing’s student population at The University of Newcastle, with partial course credits given to eligible students as incentive. A total of 23 students participated, with the first 4 participants discarded after a mistake was identified in the algorithm. After the first four participants, the fusion algorithm and experimental procedure was finalised and did not change for the remainder of the experiment. During the study, it was noted that one participant
was not engaged with the trials and gave seemingly random responses. The lack of engagement was confirmed post session after review of the session video. Thus all of this participant’s data was excluded from the analysis.

Of the 18 participants which contributed to the reported results, 17 were male and 1 was female, with ages ranging from 20 to 38, median age 22. Only 4 contributing participants had never used a HMD before. Of the remaining 14, half of the participants (7) had less than 1 hour of experience using a HMD prior to the study, within the last year. Some participants (3) indicated they had 1-5 hours of HMD experience, and 4 participants indicated that they had more than 5 hours of HMD experience in the last year. Two participants indicated that they had used the Leap Motion before the experiment. No participants had used the Perception Neuron before the experiment. Participants were tested for normal stereoscopic vision with a Frisby test (Bohr and Read 2013) before their experiment, and they were asked to read some virtual text while wearing the HMD to confirm basic visual acuity. The ability to distinguish the colour of the cubes for the Sorting Task was confirmed during the training exercise, where the square paper was coloured the same as the virtual cubes (Figure 8.14). An example participant is shown in Figure 8.17 completing each of the 3 tasks.

### 8.2 Results

This section presents the quantitative and statistical analysis of the results. Analysis is broken down into objective analysis, discussing task performance metrics (Sections 8.2.1 and 8.2.2), and subjective analysis, discussing participant responses to each tracking system (Sections 8.2.3 and 8.2.4).
Figure 8.17: Example VR view and webcam footage recorded during the study for an example participant.
8.2.1 Objective Performance Metrics

For each task, a score was computed based on the participant’s ability to complete the task over the 90 second trial period. Figure 8.18 shows the distribution of scores for each task, separated by the tracking technology used to achieve the score. The slip count was also recorded for each task to measure relative tracking error between the tracking solution conditions. Due to the fact that one participant completed each condition once, analysis of the performance metrics was done with dependent statistical testing. This approach has the added benefit of accounting for the distribution of participant skill level. Therefore, statistical analysis was performed on the differences in each metric between the Fused Tracking case and the individual tracking technology cases for the same participant. That is, the change in metric is given by

$$\text{change}_{p,t}(X) = M_{p,t}(\text{FT}) - M_{p,t}(X)$$  \hspace{1cm} (8.6)

where $M_{p,t}(X)$ is the metric for participant $p$, task $t$ and technology $X \in \{\text{LP, PN}\}$. Table 8.2 shows the distribution of score and slip count changes when Fused Tracking

![Figure 8.18: Participant scores (n=18). The plots display the inter-quartile range (box), median (red line), non-outlier range (whiskers), and outliers (circles). Outliers are points which deviate by more than $1.5 \times IQR$ from the 1st and 3rd quartiles. (LP = Leap Motion, PN = Perception Neuron, FT = Fused Tracking)](image)
is used compared to the metric achieved in the Leap Motion and Perception Neuron cases.

The data was observed to be heavily skewed in many cases and thus was not normally distributed (see Table 8.2). Therefore, the Wilcoxon signed-rank test was used to analyse the significance of the data. Note that in each case, the Wilcoxon test determines if the distribution of change is significantly non-zero centered. This test does not assume normality and has the advantage of being robust to outliers. Table 8.3 presents the results of the Wilcoxon test, including the median difference, the test statistic $T = \text{minimum sum rank}$, the 2-tailed significance $p$, and the $r$ effect magnitude:

$$r = \frac{z}{\sqrt{2n}}$$

where $z$ is the $z$ score corresponding to $p$, and $n = 18$ is the number of participants. The values which are strongly statistically significant ($p < 0.05$) are indicated by $\ast$. The results which possibly represent an effect ($0.05 \leq p \leq 0.10$) are marked by $\dagger$. Define such cases weakly significant results. These cases are worth discussing, but should be treated with a larger sense of uncertainty.

Table 8.3 shows that for the Sorting Task, in the Fused Tracking case, participants achieved weakly significantly higher scores (median 1.5 points more) and made significantly fewer slips (median 3 slips fewer) than in the Leap Motion case. However, the Perception Neuron case performed significantly better than Fused Tracking in the Sorting Task (median 2.5 points more, 1 slip less). Additionally, in the Keyboard Task, participants made significantly more slips with the Fused Tracking (median 5 and 4 slips for the Leap Motion and Perception Neuron respectively). In the Throwing Task, the Leap Motion made more slips (median 2.5 more slips) than Fused Tracking, with weak significance. In the remaining cases, Fused Tracking performed with no significant difference from the individual tracking solutions. Thus, in terms of score, Fused Tracking performed better than at least one of the individual tracking systems or statistically similar. In terms of errors, significantly more slips were
Table 8.2: Paired change in performance metrics - reported numbers are the amount of change in the metric when a given participant was using the Fused Tracking (FT) solution compared to the baseline of the other two solutions (Leap Motion (LP) or Perception Neuron (PN)).

<table>
<thead>
<tr>
<th>Task</th>
<th>Change in Score (Higher is better)</th>
<th>Change in Slip Count (Lower is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FT−LP</td>
<td>FT−PN</td>
</tr>
<tr>
<td>Keyboard</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorting</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throwing</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

†: (0.05 ≤ p ≤ 0.1) weakly significant result
* : (0 ≤ p < 0.05) strongly significant result
Table 8.3: Wilcoxon signed-rank test for performance metrics when using Fused Tracking (FT) solution compared to either Leap Motion (LP) or Perception Neuron (PN).

<table>
<thead>
<tr>
<th>Task</th>
<th>Keyboard</th>
<th>Sorting</th>
<th>Throwing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>FT–LP</td>
<td>FT–PN</td>
<td>FT–LP</td>
</tr>
<tr>
<td>Change in Score</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>-3.5</td>
<td>-2.5</td>
<td>1.5</td>
</tr>
<tr>
<td>Min Sum Rank $T$</td>
<td>48.5</td>
<td>51.5</td>
<td>27.5</td>
</tr>
<tr>
<td>Sig. $p$ (2-tailed)</td>
<td>0.184</td>
<td>0.236</td>
<td>0.063†</td>
</tr>
<tr>
<td>Effect $r$</td>
<td>-0.221</td>
<td>-0.197</td>
<td>-0.31</td>
</tr>
</tbody>
</table>

| Change in Slip Count | | | | | | |
| Median | 5.0 | 4.0 | -3.0 | 1.0 | -2.5 | -1.0 |
| Min Sum Rank $T$ | 22.5 | 31.0 | 9.0 | 0.0 | 35.0 | 83.5 |
| Sig. $p$ (2-tailed) | 0.01* | 0.056† | 0.002* | 0.002* | 0.088† | 0.93 |
| Effect $r$ | -0.427 | -0.319 | -0.513 | -0.515 | -0.285 | -0.015 |

†: (0.05 ≤ $p$ ≤ 0.1) weakly significant result
*: (0 ≤ $p$ < 0.05) strongly significant result

made by the Fused Tracking in the keyboard task.

8.2.2 Throwing Task

The data for the Throwing Task is more continuous in nature compared to the other two tasks. This section discusses further analysis of this data. Figure 8.19 shows heat-maps for the 2D histograms of ball landing position for each tracking technology. A diagram of the setup is also included for reference. The green cross indicates the participant’s seated location in the virtual world. The white square indicates the position of the table with ball spawn point in the centre. The target location is indicated by concentric rings, with colors corresponding the the area on the target bounded by that ring. Participants were instructed to throw as many balls as possible, as close as possible to the centre of the target, within the time limit (90 sec). The histograms are normalised over the same range [0, 16] throws per
Figure 8.19: Sum hit density of participant throws for each tracking technology. The concentric rings represent the target, the white square and disk represent the ball table and spawn location, and the green cross represents the participant’s location (see Figure 8.13).

22.5 $\times$ 15cm cell. The maximum value of hit density occurs in the Fused Tracking case. The plots also include the total number of throws which were attempted, as well as the valid throws - those which landed inside the plot area.

Statistical analysis shows that the means of the distributions are not significantly different. Note however, that the Leap Motion has many more throws distributed near the ball spawn point, and in fact only $347/464 \approx 75\%$ of attempts landed within the plot area. Compared to the valid rates of the Perception Neuron ($328/370 \approx 88.6\%$) and of the Fused Tracking ($323/373 \approx 86.6\%$), the Leap Motion has a high rate of noise in control of the ball. Many of these invalid throws were due to the
Figure 8.20: Ball hit density for the Throwing Task summed over all participants. The vertical lines indicate the location and size of the target and its rings.

A participant moving their hand outside of the valid tracking zone for the Leap Motion, causing the tracking to become unstable and unintentionally throw the ball outside the valid region. This problem is not present in the Fused Tracking system.

Figure 8.20 shows the distribution of throws summed over the axis orthogonal to the throw direction (summed over the vertical axis on the page for Figure 8.19). The target is now represented by a set of vertical lines while the X axis has the same range as in the 2D plots in Figure 8.19. The broken lines represent the outer edge of the target, the dotted lines represent the boundaries of the coloured regions of
the target, and the solid vertical line represents the centre of the target. The plot visually reinforces the noise inherent in the Leap Motion, with the green distribution spread broadly from $X = -250cm$ to $X = 0cm$. The Levene test\(^6\) reveals that the variance of the 1D Leap Motion histogram (stddev $\approx 91$) is significantly larger than that of the Perception Neuron (stddev $\approx 69$, $F \approx 48$, $p < 0.05$) and the Fused Tracking (stddev $\approx 74$, $F \approx 36$, $p < 0.05$). The Perception Neuron and Fused Tracking variances are not significantly different. Thus the Leap Motion was very unreliable in the Throwing Task.

### 8.2.3 Subjective Performance Metrics

Participants were asked the following question after completing each task:

- **Quality question**: ‘In which attempt did the virtual hands match your real hands the closest?’ followed by ‘Which matched the second closest?’.

- **Utility question**: ‘In which attempt was the tracking easiest to use?’ followed by ‘Which attempt was the second easiest to use?’.

The attempts were labeled anonymously and counterbalanced across participants so that the responses were blind. Both of these questions produce a ranking of the attempts in order of increasing preference. To analyse these responses, a rank value is associated with the technology which the participant used in the preferred attempt, with 3 points awarded if it was the first preference, 2 points for second preference, and 1 point for third preference. This approach is a standard approach consistent with Kendall’s $W$ test statistic for judging (Cafiso et al. 2013).

For both the Quality and Utility responses, the mean rank for each technology and each task is summarized in Figure 8.21. Statistical significance test results are given

\(^6\)The Levene test is a statistical test which measures differences in the variance of two sample distributions.
in Table 8.4. There is no significant consensus in terms of either Quality or Utility for the keyboard task. For Quality, both the Sorting and Throwing Tasks reached a weak consensus ($W \approx 0.2, p < 0.05$). In these two tasks, different individual tracking technologies were preferred, but the Fused Tracking solution was consistently mean-ranked at around 2 points. Thus, while Perception Neuron fails at the Throwing Task and Leap Motion fails at the Sorting Task, the Fused Tracking achieves moderate success with both. For Utility, only the Sorting Task demonstrated any consensus, with a weak consensus ($W \approx 0.2, p < 0.05$) that the Perception Neuron again outperformed the other two solutions. Despite the remaining individual task mean ranks being individually explainable by random noise, the trend is still evident that the Fused Tracking performs somewhere between the two individual tracking solutions.

Combining the the mean rank over all tasks yields the results in Figure 8.22. Although no significant consensus exists for the combined data in either Quality or Utility, the Quality trend is that Fused Tracking is the most preferred system. Also, Fused Tracking was not the least preferred option in terms of general Utility. The
Table 8.4: Significance of participant preferences using Kendall’s \( W \) test statistic. For significant results \( (p < 0.05) \), Kendall’s \( W \in [0,1] \) indicates the amount of agreement among the participants.

<table>
<thead>
<tr>
<th>Task</th>
<th>Quality Kendall’s ( W )</th>
<th>Quality Significance ( p )</th>
<th>Utility Kendall’s ( W )</th>
<th>Utility Significance ( p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>0.06</td>
<td>0.348</td>
<td>0.05</td>
<td>0.411</td>
</tr>
<tr>
<td>Sorting</td>
<td>0.21</td>
<td>0.024*</td>
<td>0.30</td>
<td>0.005*</td>
</tr>
<tr>
<td>Throwing</td>
<td>0.23</td>
<td>0.017*</td>
<td>0.08</td>
<td>0.249</td>
</tr>
</tbody>
</table>

\( \dagger \): \( (0.05 \leq p \leq 0.1) \) weakly significant result
\( \star \): \( (0 \leq p < 0.05) \) strongly significant result

following section reinforces this conclusion by analysing the open ended verbal responses of the participants.

8.2.4 Verbal Responses

In between tasks, after being asked to rank attempts A, B and C as above, the participants were asked the following open ended question:

“Did you find anything particularly frustrating with any of the attempts A, B or C?”

The responses of participants were summarized and recorded by the researcher, and any specific associations with an attempt was noted. Additionally, although participants were not prompted to speak during their experiments, participants occasionally voiced difficulties. The researcher also recorded summaries of these comments with the associated attempt identifier. The participants were blind to the tracking type at any given time, but the researcher was not.

These comments were grouped by participant, technology and task, giving each participant 3 chances to comment on each given technology (LP, PN, FT). Thus for each technology the number of opportunities for comment was (18 participants, 3 tasks each) 54 opportunities. Each comment was analysed for sentiment. If the comments were clearly majority negative or positive, the comment was assigned a ‘Negative’ or ‘Positive’ classification. Otherwise the comment was labeled ‘Mixed’. Any time a technology was not mentioned by the participant, a ‘No Mention’ classification
was assigned. Assignments were performed by hand, and the comment summaries and associated assignments are included in Appendix A, Tables A.1, A.2 and A.3. The classifications are summarized in Figure 8.23. The Fused Tracking solution received the fewest negative comments. Given that the prompt was for participants to identify negative aspects of each attempt, the fact that the Fused Tracking was mentioned the least further indicates user preference toward Fused Tracking.

8.2.5 Summary of Results

The following results summarise this section:

1. Section 8.2.1 concludes that for each task, Fused Tracking performed better than at least one of the individual tracking systems, or otherwise statistically similar. However, the Fused Solution exhibited significantly higher slip count for the Keyboard Task.

2. Section 8.2.2 further discusses the performance in the case of the Throwing Task, showing that the fused solution significantly reduces the noise associated with the Leap Motion.
3. Section 8.2.3 concludes that participants preferred the fused solution as a general solution compared to the individual tracking solutions which failed in particular scenarios.

4. Section 8.2.4 demonstrates that participants reported problems with the Fused Tracking solution less often than the individual tracking solutions.

Table 8.5 outlines and compares the qualities of each of the investigated tracking solutions. If a narrow set of tasks will be performed, then there will be an individual solution which performs better than the Fused Tracking. However, Fused Tracking

Figure 8.23: Response sentiment for each tracking technology. Percentages reported are fractions of response opportunities; each participant had three opportunities to comment on each of the three technologies, once each task. However, the participant was blind to which technology they were talking about, and the prompt was open ended.
Table 8.5: Summary of the conclusions drawn from the results.

<table>
<thead>
<tr>
<th></th>
<th>Leap Motion</th>
<th>Perception Neuron</th>
<th>Fused Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good Performance</td>
<td>Keyboard, Throwing</td>
<td>Keyboard, Sorting</td>
<td>All ✓</td>
</tr>
<tr>
<td>Bad Performance</td>
<td>Sorting</td>
<td>Throwing</td>
<td>None ✓</td>
</tr>
<tr>
<td>High Slip Count</td>
<td>Sorting</td>
<td>Throwing</td>
<td>Keyboard ✗</td>
</tr>
<tr>
<td>Throwing Distribution</td>
<td>Noisy</td>
<td>Low Noise</td>
<td>Low noise ✓</td>
</tr>
<tr>
<td>Quality Preferences</td>
<td>Throwing</td>
<td>Keyboard, Sorting</td>
<td>Most Consistent ✓</td>
</tr>
<tr>
<td>Utility Preferences</td>
<td>Throwing</td>
<td>Keyboard, Sorting</td>
<td>Most Consistent ✓</td>
</tr>
<tr>
<td>Complaints</td>
<td>Most</td>
<td></td>
<td>Fewest ✓</td>
</tr>
</tbody>
</table>

achieves the best generalization to a broad set of tasks, and participants notice fewer flaws with the tracking.

8.3 Discussion and Conclusion

The performance of the Leap Motion tracking was degraded by the presence of the bulky glove and sensors. This is because the the Leap Motion can only track a limited set of hand shapes and proportions. Thus, the results should not be interpreted as a direct comparison of the Leap Motion to the Perception Neuron. Rather, the results indicate that the Fused Tracking is able to take two imperfect results and generalise the tracking when one system fails. Improving the result of the Leap Motion tracking, perhaps by construction of a less bulky glove with similar capabilities to the Perception Neuron, would also see a proportional increase in the performance of the Fused Tracking. Thus, any results indicating objective performance of the Leap Motion must be observed skeptically. For example, many participants found the Leap Motion incapable of turning over the cubes in the Sorting Task. Based on researcher experience, the Leap Motion can perform this task significantly more
reliably than without the glove, but it is still noisier than the Perception Neuron and Fused Tracking. The presence of the glove simply accentuates the limitations of the Leap Motion, such as ambiguity of hand pose and tracking range. The conclusion that the Fused Tracking is able to overcome the limitations of individual systems remains unaffected by this issue, as an increase in Leap Motion capability would likely lead to a proportional increase in fused tracking quality.

The Perception Neuron had significant latency compared to the Leap Motion. This was even observed by some of the participants. This is perhaps because of filtering performed by the Perception Neuron system. The fusion process itself also introduced some additional latency for the Fused Tracking system. However, the responsiveness of the Leap Motion created a faster fused result than Perception Neuron alone, as evidenced by fewer complaints about slowness and low sensitivity. The total compute time for the Spooky Fusion system depends on the number and type of articulations active and the number of measurements received for a given frame. Typically, per measurement per bone fusion was approximately 0.3ms for rigid body measurements and 0.05-0.1ms for rotational measurements. Rigid body measurements require more bones to be adjusted in the inverse kinematics fusion process (Section 6.2.1) and hence are computationally more expensive to fuse. On average, in this study, 5-6ms of CPU time was consumed by the fusion algorithm every frame. This is real-time compatible for 90Hz (duration < 11ms). Further work will need to be done in the future to reduce this computation time and enable widespread use with more complex skeletons, more expensive applications and slower machines.

For each task, the slip count was computed as the sum of instances where the user failed to successfully execute the appointed task. In the case of the sorting task, user mistakes (where a cube was placed in a box, but sorted incorrectly), were excluded from the slip count. The slip count was then used as a metric of accuracy for comparing the tracking systems. This assumes that increasing tracking quality will decrease slip count. This is clearly the case for the Sorting and Keyboard Tasks
because the tasks are very simple to complete in reality. For the Throwing Task, during training, the participants were able to throw the real ball within 130cm of the real target from a distance of 2m. Often, participants were able to hit the target A4 page directly (see Figure 8.15). However, in the virtual task, the act of throwing the virtual ball is quite different, and depends on the velocity of the hand and the time at which a release is registered. In particular, the virtual ball has no mass and so a different technique must be employed. Regardless, more accurate and responsive tracking should enable the participant to quickly learn the new technique compared to poor tracking. Thus, the assumption still holds that increased tracking quality should decrease slip count in the Throwing Task.

8.3.1 Explanation of Participant Performance

The Keyboard Task was designed to be the simplest task, and this showed in the statistically similar performances across all technologies. However, the Fused Tracking solution saw a significant increase in slips compared to the individual systems. Participants complained that the Leap Motion was noisy and struggled to track their fingers when they tried to press the buttons. Meanwhile, participants noted that the Perception Neuron was slow or required large unnatural movements. However, participants preferred the Perception Neuron for its usefulness, thanks to its stability, despite its low accuracy. Thus it is likely that the fused system, while complained about the least, was not as useful at this task due to noise from the Leap Motion reducing the stability compared to the Perception Neuron. In particular, the Fused Tracking was noted to struggle by the researcher when the Leap Motion would gain or lose tracking, or struggle to identify an ambiguous pose, leading to transient highly inaccurate poses.

The Sorting Task was designed to require the participants to grasp an object, turn the object over in their hand for inspection, all while performing a simple cognitive task, followed by a reaching task to sort the cube. The Perception Neuron performed
best in this scenario thanks to its stability. This was especially evident for the grasping and turning actions which have high ambiguity for the Leap Motion. With this action, fingers are occluded and blend into the bulk of the hand. Participants noted that the Perception Neuron was rigid or had low sensitivity, but still preferred it overall. Participants complained a large amount about the Leap Motion, especially that the virtual hand wouldn’t rotate when they rotated their hand, and that grasping the cube was difficult. The Fused Tracking solution once again didn’t perform as well as the Perception Neuron due to occasional failures to discard inaccurate measurements from the Leap Motion. Participants also complained more about the Fused Tracking compared to the Perception Neuron, but not nearly as much as the Leap Motion. However, the success of the Perception Neuron didn’t generalise to the Throwing Task.

The Throwing Task was designed to test fast, large, dynamic motions. The responsiveness of the Leap Motion was able dominate this task, despite the limitation that participants could not have their hand tracked behind their head. Participants would typically attempt to throw overarm at first, but quickly switched to a shot-put or underarm style throw after discovering the ball would drop when it left their field of view. After trying a few different strategies and making a large number of slips, participants were often able to land a large fractions of balls close to the target centre with the Leap Motion. This explains the large variation in the throwing distribution for the Leap Motion. The Perception Neuron was more consistent with the success of the throws, but participants struggled to gain distance and complained that releasing the ball at the right time was difficult. Participants also complained that they had to compensate an offset in throwing direction for the Perception Neuron, as would be expected if the calibration pose was slightly off centre. The Fused Tracking solution was able to combine the advantages of the two to achieve a slightly better velocity than the Perception Neuron, while being significantly more reliable than the Leap Motion. Participants also complained the least about Fused Tracking in this task.
8.3.2 Ground Truth Accuracy

To support the subjective user evaluation, an objective accuracy analysis was performed, piloted by the researcher. However, access to a gold standard hand tracking solution was not available. Instead, an OptiTrack motion capture system was used to track the position of the user’s wrists to provide a partial ground truth evaluation (Figure 8.24). The fingers were not measured by the OptiTrack motion capture systems due to system quality limitations. Each tracking condition was evaluated simultaneously during a set of three unstructured trial motions. The resulting wrist position was recorded for each of the tracking conditions Leap Motion (LP), Perception Neuron (PN) and Fused Tracking (FT) This data was compared to the simultaneous recording of the ground truth (GT) head and wrist positions measured by the OptiTrack system. The distance between each of the three tracking results and the ground truth position was used as the key metric for comparison (Table 8.6). The positions were all transformed to be head-relative, and any time at which optical tracking was lost for the Leap Motion was excluded. The results suggest that the Leap Motion is more accurate, while the Perception Neuron is least accurate. Fused Tracking is typically slightly more accurate than the Perception Neuron. The true advantage of the Fused Tracking can be observed with closer...
Table 8.6: Error statistics for each tracking solution relative to the ground truth, over the three trials.

<table>
<thead>
<tr>
<th>Trial</th>
<th>Tracking</th>
<th>$\mu_E$ (cm)</th>
<th>$\sigma_E$ (cm)</th>
<th>$\mu_E$ (cm)</th>
<th>$\sigma_E$ (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>LP</td>
<td>12.1</td>
<td>3.8</td>
<td>13.0</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td>PN</td>
<td>24.6</td>
<td>13.8</td>
<td>20.3</td>
<td>9.4</td>
</tr>
<tr>
<td></td>
<td>FT</td>
<td>26.8</td>
<td>12.4</td>
<td>23.8</td>
<td>8.9</td>
</tr>
<tr>
<td>Test 2</td>
<td>LP</td>
<td>14.1</td>
<td>4.6</td>
<td>11.9</td>
<td>2.6</td>
</tr>
<tr>
<td></td>
<td>PN</td>
<td>23.8</td>
<td>14.3</td>
<td>26.9</td>
<td>19.0</td>
</tr>
<tr>
<td></td>
<td>FT</td>
<td>21.3</td>
<td>8.2</td>
<td>23.4</td>
<td>9.9</td>
</tr>
<tr>
<td>Test 3</td>
<td>LP</td>
<td>11.9</td>
<td>2.0</td>
<td>14.1</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td>PN</td>
<td>32.1</td>
<td>7.8</td>
<td>38.0</td>
<td>11.6</td>
</tr>
<tr>
<td></td>
<td>FT</td>
<td>22.6</td>
<td>5.1</td>
<td>22.9</td>
<td>14.3</td>
</tr>
</tbody>
</table>

scrutiny of the measured trajectories. Note that the errors are intended to be interpreted comparatively between each of the LP, PN and FT cases since there is a quantity of alignment error in the GT signal between the marker positions and the corresponding points tracked by the systems under examination.

The wrist trajectories as measured by each tracking solution are visualised in Figures 8.25-8.27. The view is top-down, with the forward looking x-axis direction marked by a red line. Note how the Leap Motion can only track the hand when it is in a narrow cone projected from the front of the headset, and even within this region, tracking is inconsistent. Note that this is mostly due to the bulky glove - thus again only comparative conclusions can be reached between the different tracking systems, rather than absolute. Meanwhile, the Perception Neuron tracks consistently, but fails to reproduce many of the subtleties of the true path. For example, Figure 8.27 demonstrates the significant positional offset of the Perception Neuron in the left hand (blue). Alternatively, the proposed Fused Tracking solution demonstrates higher accuracy in front of the headset, while also reproducing the motions of hand behind the headset. Focussing on loops formed by the trajectory in the ground truth plot, corresponding loops are identifiable in the Fused Tracking solution (for example, the two large loops for the bottom left quadrant of Figure 8.26). There is
Figure 8.25: Test 1. Left (orange) and right (blue) wrist position measured relative to the head (black dot) viewed from the top. The $x$-axis points in the direction of the user’s view (red line), and the $y$-axis points left-wards (green line). Grid lines are 20cm apart. Notice that the Fused Tracking case reproduces many of the ground truth curves more faithfully than the Leap Motion or Perception Neuron alone.
Figure 8.26: Test 2. See Figure 8.25 for semantic key. Note that the two large identifiable curves in the right hand data (orange) of the Fused Tracking and Ground Truth cases are less identifiable or missing from the individual tracking cases.
Figure 8.27: Test 3. See Figure 8.25 for semantic key. In this test, a throwing motion was performed repeatedly with the right hand (orange). The ground truth trace is drawn beneath each trace for comparison (grey). Note how the Fused tracking combines the high accuracy of the Leap Motion in front of the headset with the tracking coverage provided by the Perception Neuron.
considerable error still present in these curves, but the Fused Tracking has produced a more faithful result.

### 8.3.3 Reflection on Proposed Fusion Algorithm

The most significant limitation of the proposed fusion system is the reliance on a meticulously programmed noise model for the Leap Motion (see Section 8.1.3). The Perception Neuron measures orientation with essentially constant noise levels. The Leap Motion has a highly discontinuous accuracy due to occlusion, ambiguous hand poses and low field of view. The Leap Motion software interface for Unreal Engine does provide a measurement of tracking certainty for each hand. However, this function increases with the amount of time the hand has been seen continuously, reaching its maximum value of 1 within a second or two. This is not useful for determining when to trust the reported finger positions, and furthermore, doesn’t model the changes in certainty of hand orientation when the hand is closed. To achieve a useful fused tracking result, a meticulous model was constructed based on the pose of the hand reported by the Leap itself (see Section 8.1.1). For example, the certainty of the finger bones was based on a ray-trace check from the head pose to the reported finger positions to test for visibility. If not visible, the measurement was rated low in confidence. This is far from ideal because the noise is then a function of hand pose, which may be completely incorrect. In particular, this caused issues when the gloved hand was closed, as the Leap Motion would give highly varying values for the wrist roll angle, sometimes flipping the hand upside-down entirely.

Two improvements over this approach are clear. The first is to base the noise model for very noisy sensors on the measurements of a more reliable sensor. For example, in this case, the Perception Neuron would have provided good information about whether or not the Leap Motion could be relied upon. The key drawback to this approach is that this breaks the modularity of the system, because the Leap Motion now depends on the Perception Neuron. If the Perception Neuron were swapped
out for another technology, compatibility could not be guaranteed. Alternatively, any new data from an unreliable sensor could be compared to the expected value based on the fused prior. This creates a closed loop system which presents several difficulties such as instability. The best approach might be to provide a secondary skeleton, consisting of only data from user-marked ‘reliable’ sensors (sensors which have approximately constant noise levels), such as the Perception Neuron. A noisy sensor could then query this skeleton and use the result to build a noise model.

The second enhancement would be to use machine learning approaches to map the pose of the hand to a certainty value for each bone. It is unlikely that mapping the noisy sensor state to a confidence value would yield truly excellent results, but some improvement should be possible by capturing nuanced state variability and time-dependent aspects of the tracking signal such as jitter. Techniques such as recursive neural networks are capable of learning very complex time-dependent models of arbitrary functions with offline training (for example, Du et al. (2015)). A machine learning approach could operate on the raw image from the Leap Motion, but this approach would not generalise to other sensor types and would require large amounts of training data. Alternatively, combining this approach with a secondary sensor would likely produce good results. Training with ground truth tracking data from a gold-standard tracking system would be ideal, but it might be possible to learn a model based solely on the paired, more reliable sensors in the fusion system, such as the Perception Neuron in this case. Implementing this in a modular way presents a considerable challenge for future work. The advantage of such a solution is that the manufacturer wouldn’t need to program a complex noise model, or use expensive equipment to train a model.

8.3.4 Reflection and Future Work

The following points summarise the outcomes of this chapter:
The fused tracking solution was found to be a more general solution than either of the individual solutions, although fused tracking did not perform as well as the best solution for a given individual task.

The fused tracking solution was preferred by participants for tracking quality and was also complained about the least.

The fused tracking in particular is able to resolve ambiguities in hand orientation of the Leap Motion, allowing for increased performance in the Sorting Task.

Further work needs to be done on the noise model for the Leap Motion - fusing the Leap Motion with the Perception Neuron increased the noise in the solution compared to the Perception Neuron alone. This leads to unstable results in cases where the Leap is tracking poorly.

In conclusion, the proposed Fused Tracking solution was able to generalise best over all tasks tested. When creating a tracking solution for a VR user interface, if only a narrow set of interactions is needed, and data is available on which solution is appropriate, then choosing a lone tracking solution suited to the task is the best option. However, if a broad set of interactions are required, then the proposed fusion algorithm is recommended. Also, if choosing a tracking technology for a task for which little comparative data exists between tracking technologies, the proposed fusion algorithm has a higher likelihood of providing a good solution. Future work will first involve submission of the work in Chapters 6 and 8 to a peer reviewed journal or conference. Following submission, the work will be extended with a focus on improving the fusion algorithm, particularly in the domain of modelling occlusion and reliability of sensors, toward the goal of creating a fusion algorithm which is always the best choice.
Chapter 9

Conclusion

Fusion of sensors can convey significant benefit to interactive immersive display systems, including AR and VR systems. However, existing algorithms for fusion require significant expertise. Expert design or intervention is often required for calibration and setup. Existing automated fusion systems also require expert installation and setup - only after these steps are performed can casual users make use of them. Furthermore, these systems, such as Ubitrack (Pustka, Huber, Waechter et al. 2011), have become outdated in the modern ecosystem of VR and AR tracking devices.

9.1 Thesis Summary

This thesis has presented research into every aspect of the ambient sensor fusion pipeline:

- **Sensor Correlation** - Determination of broad statistical characteristics of sensor systems relative to one-another. This includes identifying statistical dependencies and relative temporal relationships (Chapters 3 and 4).
• **Sensor Calibration** - Automatically monitor data to compute precise geometric relationships between sensor systems with overlapping domains. This includes detecting when an established model has a fault or a physical reconfiguration (Chapter 5).

• **Sensor Fusion** - Combining sensor systems with overlapping domains by extracting the state of an underlying model (Chapters 6 and 8).

The methods used to investigate these problems include simulation, prototyping, pilot testing and ethics reviewed human user testing. The code produced during the research is provided as part of an open source Unreal Engine 4 plugin called ‘Spooky’ (see Chapter 7 for more details). This plugin allows for modular ambient fusion for any sensor which has Unreal Engine 4 support.

Outcomes for sensor correlation include two central contributions. The first contribution is a pair of algorithms for detecting rigid links between 3DoF and 6DoF sensors from different systems, before the relationship between the systems is calibrated (Chapter 3 and publications Fountain and Smith (2016) and Fountain and Smith (2018)). This functionality allows for the detection of statistical relationships between a range of sensor types before calibration of the precise parameters of the relationship. After sensor-sensor relationships are identified, data collected from the sensors can be used build models of the relationship in the sensor calibration step. The second contribution toward sensor correlation is a model-less algorithm for determining the latency between two related signals (Chapter 4). During the investigation of the sensor identification and sensor calibration problems, the issue of latency between the signals in question arose. The key challenge is performing latency calibration before sensor calibration, because desynchronisation is detrimental to calibration accuracy. The proposed method has been shown to be capable of computing the latency between two sensors with a highly non-linear relationship and over 100 dimensions. This approach is a significant step toward methods for
comparison of signals with arbitrary statistical relationships. The approach may also lead to new approaches for model-less sensor identification.

Outcomes for sensor calibration centre around a state machine developed to automatically perform calibrations between various sensor systems and detect when the arrangement changes (Chapter 5 and publication Fountain and Smith (2017)). The proposed method is able to determine the relationship between the Microsoft Kinect v2 and a VR system within around 15 seconds of spontaneous user movement. Furthermore, the calibration system is able to detect and correct a fault, even as minor as an 8° rotation of a sensor, within around 35 seconds.

Contributions to the final stage of ambient sensor fusion, the data fusion step, include the theory and implementation of a modular skeleton fusion algorithm (Chapter 6). The algorithm extracts a constrained skeletal model from sensor data. Constraints allow for emergent solution to under and over constrained kinematics problems during fusion of sensors. The algorithm was applied to fusing two hand tracking solutions: Leap Motion and Perception Neuron. The fused tracking result was tested with a user study ($n = 18$ participants; Chapter 8). Across a variety of tasks, the fused tracking solution was able to perform more consistently than the component tracking systems. Furthermore, participants had fewer complaints about the fused solution compared to the component systems.

9.2 Future Work

Although the work completed for this thesis is considerable, there is much potential for expansion. Particularly, the latency calibration topic shows promise in the field of comparing signals with arbitrary relationships, with several significant research avenues apparent. The work presented in Chapter 4 is exploratory and indicates that the method could be generalised and optimised to compute latency between complex signals with no user input. The latency calibration method also suggests a more
general approach to identifying sensor-sensor relationships than those presented in Chapter 3, which only apply to rotational and positional sensors. Ultimately, the approach may also generalise to pattern matching in other domains, such as machine learning in temporal scenarios.

Sensor calibration presented in Chapter 5 would benefit from further research into fault detection. The system favours false negatives for faults in order to minimise interruption of the user. Further work is need to determine a method for calibrating two positional sensors connected by a non-zero length rigid link. Presently, the system simply assumes that the positions of two connected sensors is small relative to their gross motions.

Many directions for further research are present for the skeleton sensor fusion algorithm presented in Chapter 6. Opportunities have been identified for improvement of the selection of relevant bones to be involved in a given fusion step. A significant domain for improvement lies in the modelling of noise profiles of individual sensor systems. Machine learning techniques would be the most appropriate research direction for improving the quality of information/uncertainty models for sensors. Such models would relieve hardware developers of meticulously hand-coding a noise model while also improving quality. Finally, submission of the work in Chapters 6 and 8 for publication is planned.

The software plugin ‘Spooky’, will require additional work in order for it to be used as effortlessly as originally envisioned, but this is the nature of programming for research outcomes. Portability of the system was reduced by the use of the UE4 skeletal mesh component class for configuration of the calibration and fusion. While it is still possible to use the C++ interface, significant conveniences are only implemented using UE4 features. Also, the compilation system is not configured for pure native C++. There are a handful of missing algorithms at the lowest level of the system for calibration and fusion which would expand the capabilities of the system,
such as unpaired point cloud alignment. These algorithms are not yet implemented because they were not needed for the core components of the research.

Ultimately, the significant body of research communicated in this dissertation has produced a set of useful algorithms for ambient sensor fusion, an open source implementation of these algorithms, and a user study testing them. It is hoped that this work improves the process of building complex VR systems consisting of multiple sensor systems. In particular, this work expands access to such systems for researchers, businesses and enthusiasts who might otherwise lack the skill and time to design systems themselves, or the capital to afford gold standard single unit systems. This work also represents a significant component of future VR software middlewares, such as OpenXR, which promote a diverse ecosystem of immersive hardware systems and compatible software for a wide range of applications.
Appendix A

User Study Documents

The following pages contain the key documents involved with the ethics approval of the user study performed in Chapter 8. The first document is the approval notice for the study. The next is the information sheet distributed to the participants prior to signing up to the study. The consent sheet follows, which was signed by participants on the day of the experiment. The demographics questionnaire was completed by all participants after the experiment during their precautionary recovery time. Finally, the researcher data sheet provides a scaffold for the experimenter to record the responses of the participant during the experiment.

After the ethics information, Tables A.1 to A.3 list the recorded participant responses with their categorisations. The statistics for these responses are discussed in the results section of Chapter 8.
Thank you for your Response to Conditional Approval (minor amendments) submission to the Human Research Ethics Committee (HREC) seeking approval in relation to the above protocol.

Your submission was considered under L2 Low Risk Research Expedited review by the Ethics Administrator.

I am pleased to advise that the decision on your submission is Approved effective 25-Jan-2018.

In approving this protocol, the Human Research Ethics Committee (HREC) is of the opinion that the project complies with the provisions contained in the National Statement on Ethical Conduct in Human Research, 2007, and the requirements within this University relating to human research.

Approval will remain valid subject to the submission, and satisfactory assessment, of annual progress reports. If the approval of an External HREC has been “noted” the approval period is as determined by that HREC.

The full Committee will be asked to ratify this decision at its next scheduled meeting. A formal Certificate of Approval will be available upon request. Your approval number is H-2017-0391.

If the research requires the use of an Information Statement, ensure this number is inserted at the relevant point in the Complaints paragraph prior to distribution to potential participants You may then proceed with the research.

For Noting

Please provide a separate introductory paragraph (above 'Why is the research being done?') to introduce the research team e.g. "You are invited to participate in the research project identified above which is being conducted by Mr Jake Fountain as part of his PhD studies at the University of Newcastle, under the supervision of Dr Shamus Smith and Associate Professor Stephan Chalup, from the School of Electrical Engineering and Computing, University of Newcastle.

Please ensure the Research Team to include A/Prof Stephan Chalup has been updated across all participant documents.

Conditions of Approval

This approval has been granted subject to you complying with the requirements for Monitoring of Progress, Reporting of Adverse Events, and Variations to the Approved Protocol as detailed below.

PLEASE NOTE:
In the case where the HREC has "noted" the approval of an External HREC, progress reports and reports of adverse events are to be submitted to the External HREC only. In the case of Variations to the approved protocol, or a Renewal of
approval, you will apply to the External HREC for approval in the first instance and then Register that approval with the University’s HREC.

- **Monitoring of Progress**

Other than above, the University is obliged to monitor the progress of research projects involving human participants to ensure that they are conducted according to the protocol as approved by the HREC. A progress report is required on an annual basis. Continuation of your HREC approval for this project is conditional upon receipt, and satisfactory assessment, of annual progress reports. You will be advised when a report is due.

- **Reporting of Adverse Events**

1. It is the responsibility of the person **first named on this Approval Advice** to report adverse events.
2. Adverse events, however minor, must be recorded by the investigator as observed by the investigator or as volunteered by a participant in the research. Full details are to be documented, whether or not the investigator, or his/her deputies, consider the event to be related to the research substance or procedure.
3. Serious or unforeseen adverse events that occur during the research or within six (6) months of completion of the research, must be reported by the person first named on the Approval Advice to the (HREC) by way of the Adverse Event Report form (via RIMS at [https://rims.newcastle.edu.au/login.asp](https://rims.newcastle.edu.au/login.asp)) within 72 hours of the occurrence of the event or the investigator receiving advice of the event.
4. Serious adverse events are defined as:
   - Causing death, life threatening or serious disability.
   - Causing or prolonging hospitalisation.
   - Overdoses, cancers, congenital abnormalities, tissue damage, whether or not they are judged to be caused by the investigational agent or procedure.
   - Causing psycho-social and/or financial harm. This covers everything from perceived invasion of privacy, breach of confidentiality, or the diminution of social reputation, to the creation of psychological fears and trauma.
   - Any other event which might affect the continued ethical acceptability of the project.
5. Reports of adverse events must include:
   - Participant's study identification number;
   - date of birth;
   - date of entry into the study;
   - treatment arm (if applicable);
   - date of event;
   - details of event;
   - the investigator's opinion as to whether the event is related to the research procedures; and
   - action taken in response to the event.
6. Adverse events which do not fall within the definition of serious or unexpected, including those reported from other sites involved in the research, are to be reported in detail at the time of the annual progress report to the HREC.

- **Variations to approved protocol**

If you wish to change, or deviate from, the approved protocol, you will need to submit an **Application for Variation to Approved Human Research** (via RIMS at [https://rims.newcastle.edu.au/login.asp](https://rims.newcastle.edu.au/login.asp)). Variations may include, but are not limited to, changes or additions to investigators, study design, study population, number of participants, methods of recruitment, or participant information/consent documentation. **Variations must be approved by the (HREC) before they are implemented** except when Registering an approval of a variation from an external HREC which has been designated the lead HREC, in which case you may proceed as soon as you receive an acknowledgement of your Registration.

---

**Linkage of ethics approval to a new Grant**
HREC approvals cannot be assigned to a new grant or award (ie those that were not identified on the application for ethics approval) without confirmation of the approval from the Human Research Ethics Officer on behalf of the HREC.

Best wishes for a successful project.

Associate Professor Helen Warren-Forward
Chair, Human Research Ethics Committee

For communications and enquiries:
Human Research Ethics Administration
Research & Innovation Services
Research Integrity Unit
The University of Newcastle
Callaghan NSW 2308
T +61 2 492 17894
Human-Ethics@newcastle.edu.au


Linked University of Newcastle administered funding:
Why is the research being done?
Virtual reality tracking technologies, and the associated sensor systems, are a core component for the realisation of interactive virtual environments. 3D user interfaces often require information about a user’s physical position, orientation or motion in 3D space. To improve interaction in virtual environments it is often desirable to have extended tracking range or improved tracking accuracy by combining multiple sensor systems which ordinarily do not communicate. The purpose of this research is to verify the usefulness of new sensor algorithms designed to combine multiple virtual reality (VR) tracking systems. The aim of this study to ensure that the algorithms perform well for a variety of users. Given the increasingly widespread use of virtual reality technologies it is important that any supporting algorithms work well with a variety different users, given the subjective nature of virtual reality experiences. This research is being conducted in the School of Electrical Engineering and Computing by Jake Foundation as part of his PhD studies at UON, under the supervision of Dr Shamus Smith and AP Stephan Chalup.

Who can participate in the research?
Student enrolled in INF3950 and SENG2260 courses at the University of Newcastle can participate. However, only people with normal or corrected (with contact lenses) vision can participate as the use of glasses in head-mounted displays is difficult. Additionally, you must be able to pass a simple stereoscopy test. Other exclusions are people with cold or flu symptoms, people with limited hand dexterity or people known to suffer from conditions that might be aggravated by wearing a HMD, such as vertigo, claustrophobia or epilepsy.

What choice do you have?
Participation in this research is entirely your choice. Only those people who give their informed consent will be included in the research. Whether or not you decide to participate, your decision will not disadvantage you. If you do decide to participate, you may withdraw from the research at any time without giving a reason. If you wish to withdraw, any questions you answer or data collected will not be used, and your responses will be shredded and discarded.

What would you be asked to do?
Participants will need to sign a consent form and complete a brief stereo vision test. It should be noted that the experiment session will be video recorded. Footage will be kept private by default and used only for research analysis. You may opt in to reuse of your footage in research dissemination in an anonymous way. After signing the consent form, participants will then have a short training session. The training session will involve performing real-world versions of virtual tasks to later be performed in VR. Next, the participant will be required to put on an upper body motion capture system (see Figure 1), over their clothes. This system tracks arm and finger positions. Once ready, the participant will put on a VR headset and the experiment will run for about 15 minutes, with instructions displayed in VR and verbal guidance from the researchers.

Figure 1 - Perception Neuron Motion Capture System - The experiment involves wearing this suit over your clothes. The suit fits the majority of body types with adjustable bands and multiple glove sizes available.
The participant will also need to answer some questions as they proceed through the experiment. The experiment involves seated actions such as grasping virtual objects, throwing virtual objects and pressing virtual buttons. After the experiment, a 15 minute waiting period is mandatory to allow recovery from possible side effects of VR. During the waiting period, participants will complete a simple demographics survey.

**How much time will it take?**
The session will take approximately 60 minutes to complete. The session will consist of the following:

- Welcome, opportunity to ask questions and stereoscopic eye test (2 minutes)
- Consent form completion (3 minutes)
- Video recording starts
- Training session with real world versions of virtual tasks (5 minutes)
- Putting on the motion capture equipment (10 minutes)
- VR tasks (button pressing, throwing objects, sorting objects) (15 minutes)
- Take off equipment (5 minutes)
- Video recording ends
- Fill out demographics survey and wait for possible effects of VR to wear off (15 minutes)

**What are the risks and benefits of participating?**
There are no permanent risks in participating in the session. Virtual reality technology can involve very immersive experiences. Some people also experience a motion sickness like effect. Thus we have limited exposure to the VR experience to 15 minutes (as per industry advice – also see [http://www.oculus.com/warnings](http://www.oculus.com/warnings)). Thus you will spend 15 minutes after your VR experience on non-VR activities before leaving the session. However, you are free to stop participating at any time during the session. Students in courses with approved Research Awareness Assessment will receive 2 credit points via the SONA system for completing this evaluation.

**How will your privacy be protected?**
All paper copies of the consent form and anonymous demographic forms will be stored in a locked room, and will be accessible only to the research team. Task performance data, video data and question responses from the tests will be encrypted and kept on a secured computer drive. This data will be retained for at least 5 years at the University of Newcastle. You may optionally consent to reuse of recorded video footage as part of the research dissemination – in this case, you will most likely be wearing a VR headset which covers your face. Otherwise, we will blur your face to protect your privacy.

**How will the information collected be used?**
The research team will write reports, journal articles and conference papers. The results will also be reported in Jake Fountain’s PhD thesis. No individual will be identified in these publications. Non-identifiable data may also be shared with other parties to encourage scientific scrutiny, and to contribute to further research and public knowledge, or as required by law. Research publications from the project will be available via NOVA: The University of Newcastle Research Online ([http://nova.newcastle.edu.au](http://nova.newcastle.edu.au)). You can request a summary of the results by providing your email on the demographic questionnaire. We expect the summary of results to be available in December 2018.

**What do you need to do to participate?**
Please read this Information Statement and be sure you understand its contents before you consent to participate. To ask any questions about the information sheet please contact: Jake Fountain (ph: 4921 6361, email: jake.fountain@uon.edu.au) or Shamus Smith (email: shamus.smith@newcastle.edu.au). You can enrol for an evaluation session via the SONA system.

Kind Regards,

Jake Fountain (PhD candidate)       Dr Shamus P. Smith

**Complaints about this research**
This project has been approved by the University’s Human Research Ethics Committee, Approval No. H-2017-0391. Should you have concerns about your rights as a participant in this research, or you have a complaint about the manner in which the research is conducted, it may be given to the researcher, or, if an independent person is preferred, to the Human Research Ethics Officer, Research Services, NIER Precinct, The University of Newcastle, University Drive, Callaghan NSW 2308, Australia, telephone (02) 4921 6333, email Human-Ethics@newcastle.edu.au.
Consent Sheet for the Research Project:
Virtual Reality Interaction Enhancement with Sensor Fusion (VRIESF)

If you would like to participate, please read the following and, if appropriate, tick the corresponding box to indicate your agreement:

☐ I agree to participate in the VRIESF session and give my consent freely.
☐ I understand that the project will be conducted as described in the Information Statement, a copy of which I have retained.
☐ I understand I can withdraw from the project at any time and do not have to give any reason for withdrawing.
☐ I understand that my personal information will remain confidential to the researchers except as required by law.
☐ I have had the opportunity to have questions answered to my satisfaction.
☐ I understand that the experiment session will be video recorded and the footage only used for research reference. By default, no images or audio of you will be reproduced in any publications or releases. It might be the case that some still or video images of your session would be useful to include as anonymous supporting data in publications or releases. If you would allow this, please indicate below. If not, leave it blank.
☐ YES ☐ NO  Optional: I consent to release of session footage (video and/or still images) as a component of research dissemination with the understanding that my full face will not be shown (either partially covered by a VR headset, or blurred/obscured in post-production when I am not wearing a VR headset that covers my eyes).

NOTE: The following people are excluded from this study and should not complete this form:
- People with cold or flu symptoms
- People without normal stereoscopic vision. Corrected with contact lenses is fine, but not glasses
- People with limited hand dexterity, for example repetitive strain injury (RSI)
- People who suffer from conditions that might be aggravated by wearing a head mounted display (HMD), for example vertigo, claustrophobia or epilepsy

Participant ID: ____________________________
Print Name: ______________________________ Date: __________________

Signature: ________________________________

Researcher use only: ☐ Frisby Stereotest passed

This project has been approved by the University’s Human Research Ethics Committee, Approval No. H-2017-0391. Should you have concerns about your rights as a participant in this research, or you have a complaint about the manner in which the research is conducted, it may be given to the researcher, or, if an independent person is preferred, to the Human Research Ethics Officer, Research Services, NIER Precinct, The University of Newcastle, University Drive, Callaghan NSW 2308, Australia, telephone (02) 4921 6333, email Human-Ethics@newcastle.edu.au.

Document ID RD2
Demographics Questionnaire
Virtual Reality Interaction Enhancement with Sensor Fusion (VRIESF)

Date: ________________________  Participant ID: __________________________________________

1. Gender (M/F/Other): ____________
2. Age (years): _________________
3. Before the experiment today, have you used a head-mounted virtual reality display before? (For example: Oculus Rift, HTC Vive, Samsung Gear VR, PlayStation VR, Google Cardboard, etc.)
   - ☐ No (go to question 5)
   - ☐ Yes

4. How much time have you spent immersed within virtual reality head mounted displays in the last year? Estimate the total over all sessions in the last year:
   - ☐ Less than an hour
   - ☐ 1-5 hours
   - ☐ 5-20 hours
   - ☐ More than 20 hours

5. Have you ever used any of the tracking devices from the experiment before? (e.g. Leap Motion, Perception Neuron)
   - ☐ No
   - ☐ Yes, I have used:
     __________________________________________________________
     __________________________________________________________

If you would like a summary of the final results, please add your email address below:
____________________________________________________________________
____________________________________________________________________

This project has been approved by the University’s Human Research Ethics Committee, Approval No. H-2017-0391. Should you have concerns about your rights as a participant in this research, or you have a complaint about the manner in which the research is conducted, it may be given to the researcher, or, if an independent person is preferred, to the Human Research Ethics Officer, Research Services, NIER Precinct, The University of Newcastle, University Drive, Callaghan NSW 2308, Australia, telephone (02) 4921 6333, email Human-Ethics@newcastle.edu.au.
**Experiment Verbal Responses**

(Ask the participant the following questions after each task (3 trials) and record their responses in the table below)

**Q1** “A sense of ownership over the virtual arm is the degree to which you believed the arm to be your own. Based on your perceptions, rank trials A, B and C in terms of sense of ownership over the virtual arm. Start from the most convincing end with the least convincing.” (response e.g. A,C,B)

**Q2** “Rank trials A, B and C in ease of task completion, starting with the easiest and ending with the hardest.” (response e.g. A,C,B)

**Q3** “Did anything frustrate you during the trials? What frustrated you about the trials?” (open ended response, researcher summarise noteworthy comments)

<table>
<thead>
<tr>
<th></th>
<th>Task 1 (Buttons)</th>
<th>Task 2 (Throwing)</th>
<th>Task 3 (Sorting)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Q1 Ownership</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Q2 Ease of Use</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Q3 Frustration</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Notes:

________________________________________________________________________
________________________________________________________________________

This project has been approved by the University’s Human Research Ethics Committee, Approval No. H-2017-0391. Should you have concerns about your rights as a participant in this research, or you have a complaint about the manner in which the research is conducted, it may be given to the researcher, or, if an independent person is preferred, to the Human Research Ethics Officer, Research Services, NIER Precinct, The University of Newcastle, University Drive, Callaghan NSW 2308, Australia, telephone (02) 4921 6333, email Human-Ethics@newcastle.edu.au.
Table A.1: Table of Complaint Responses Toward Leap Motion Tracking

<table>
<thead>
<tr>
<th>Task</th>
<th>Participant</th>
<th>Comment</th>
<th>Assigned Sentiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>5</td>
<td>Sorry my hand is not working; tracking [frustrating]</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>at long reach didn't pick up tracking well</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Hands all over the place</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>It was too wacky but good depth</td>
<td>mixed</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>hand was wacky</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>fingers spazzed out</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>fingers moved a lot - unstable</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>Hand wouldn't skew up; Palm would hit [wrong] key</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>I seem to have broken my left arm</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>Fingers seem a tad wonky; Unstable fingers at distance</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>my hand was getting out and got my other finger, fingers doing their own thing. Tried spreading fingers</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>Hand got stuck and moved jarringly</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>Calibration out of sync</td>
<td>-ve</td>
</tr>
<tr>
<td>Sorting</td>
<td>5</td>
<td>cube was falling on; hand was rotating without movement</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>lost sight of hand</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>cube was flipped over</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Struggled; fingers curling and hand spinning</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>really buggy</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>responsiveness off - glitched a bit</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>pick up and turning cube unstable</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>dropped (joke) a lot</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>kept flying out of control; elbows were scary</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>lots of dropping</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>A mess, glitching out; Rotating the wrong way</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>hands are gone again</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>tracking couldn't figure out what I was doing</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>Quick tracking speed but arms were not good</td>
<td>mixed</td>
</tr>
<tr>
<td>Throwing</td>
<td>5</td>
<td>this is frustrating</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>test sitting often</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>A lot already close from right hand</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Arm and hands and digits flipped out</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Overall didn't work</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>hard to throw; easy to pick up; Arm is bugged out - is it working?</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>23</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table A.2: Table of Complaint Responses Toward Perception Neuron Tracking

<table>
<thead>
<tr>
<th>Task</th>
<th>Participant</th>
<th>Comment</th>
<th>Assigned Sentiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>5</td>
<td>My palm touched the keyboard</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Thinks correspond accurately; required larger movements</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Very rigid and slow, left pinky randomly moving</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Elbow moved meant that hand moved angled backwards</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Hand was tilted</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Perfect, hand a little above where it was</td>
<td>+ve</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>Felt fast</td>
<td>+ve</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Left index finger wouldn’t go down</td>
<td>-ve</td>
</tr>
<tr>
<td>Sorting</td>
<td>5</td>
<td>Grip wouldn’t trigger</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Left right not best overall</td>
<td>+ve</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Uncontrollable</td>
<td>+ve</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Sensitivity low but I got used to it</td>
<td>mixed</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Thumb unresponsive, pinky wiggling out</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>Stable turning</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>Right hand didn’t work; sometimes cute would be flicked away</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>Thumb unresponsive</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>Started grasping differently</td>
<td>mixed</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>Didn’t work for picking up at first but I learned a better approach</td>
<td>mixed</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throwing</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Almost at right side to hit centre, hard to throw far</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Right hand felt off vs left pinky stretched</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>Fingers cut off, I found a way, hand was inconsistent</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>Underarm worked, thumb was stuck</td>
<td>mixed</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Pinky finger backwards</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>Picking up the ball was frustrating</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>Had to throw to right to hit target</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>Hands very wonky, fingers bad, couldn’t let go of ball</td>
<td>-ve</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>Why is...my pinky up?, spawned hand in weird place; tracking more consistent than A</td>
<td>mixed</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Fingers bent backwards, grabbing ball difficult, release was slow</td>
<td>-ve</td>
</tr>
</tbody>
</table>
Table A.3: Table of Complaint Responses Toward Fused Tracking

<table>
<thead>
<tr>
<th>Task</th>
<th>Participant</th>
<th>Comment</th>
<th>Assigned Sentiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>5</td>
<td>Finger not moving</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Didn’t correspond accurately, require larger movements</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Much better</td>
<td>+ve</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Fingers spazzed out</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>Similar to B but a little better</td>
<td>+ve</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>Skeleton got tangled</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>Model seemed unrealistic</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>Seemed to match cause</td>
<td>+ve</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>Left hand floated away when using right and accidentally hit key</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Best camera angle but fingers worse than B (Leap)</td>
<td>mixed</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td></td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Calibration out of sync</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Cubes also thrown away, close to arm movements and more responsive</td>
<td>mixed</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Hand may not fingers, grab too easy to let go</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Tracking didn’t pick up, arms didn’t load, cube was flicked away</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Thumb unresponsive</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>Troubles grasping at first</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>Picking up better, could move around a bit more</td>
<td>+ve</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>Sometimes lost grip</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Lots faster even though a bit glitchy</td>
<td>+ve</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorting</td>
<td>5</td>
<td>Right arm didn’t match, letting go before pushing worked a bit better</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Right hand was doing something strange</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Overarm didn’t work, underarm worked</td>
<td>mixed</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>Right hand jumping around</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Difficult at first</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>Picking up the ball was frustrating, didn’t pick up my hand</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>Ball dropped from hand at peripheral</td>
<td>−ve</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Release was slow. Left arm was tracking at position of right but it started out the best</td>
<td>mixed</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
References


References


References


References


Slater, M., A. Steed and M. Usoh (2013). Being There Together: Experiments on
Presence in Virtual Environments (1990s). Technical Report, Department of Com-
puter Science, University College London, UK.

and Presence in Immersive Virtual Environments’. In: Presence: Teleoperators &
Virtual Environments 2.3, pp. 221–233.

(1993b). ‘The Influence of a Virtual Body on Presence in Immersive Virtual En-
vironments’. In: VR 93, Virtual Reality International, Proceedings of the Third
Annual Conference on Virtual Reality, pp. 34–42.

Snyder, C., T. Bengtsson, P. Bickel and J. Anderson (Dec. 2008). ‘Obstacles to High-
Dimensional Particle Filtering’. In: Monthly Weather Review 136.12, pp. 4629–

Strasser, F. (2004). ‘Bootstrapping of Sensor Networks in Ubiquitous Tracking En-

‘Nonparametric Belief Propagation’. In: Commun. ACM 53.10, pp. 95–103. doi:
10.1145/1831407.1831431.

(2001). ‘VRPN: A Device-Independent, Network-Transparent VR Peripheral Sys-
tem’. In: Proceedings of the ACM Symposium on Virtual Reality Software and
505008.505019.

sachusetts: The MIT Press.

Wan, E. A. and R. Van Der Merwe (2000). ‘The Unscented Kalman Filter for Non-
linear Estimation’. In: Adaptive Systems for Signal Processing, Communications,

and Improving the Depth Accuracy of Kinect for Windows V2’. In: IEEE Sensors