Describing systems for the exploration of tangible and spatial computer interaction

Final year project in Creative Computing BSc (Hons)

Louis James

Department of Computing
Goldsmiths, University of London

June 9, 2020

Acknowledgements

Thanks to my family, Florent, Chudleigh dwellers, Jamie...

Abstract

Presented here is a specification for experimental approaches to computer interaction based on spatial and tangible methods. The report describes a prototypical implementation of a base system for computer interaction using physical objects on a surface. A theoretical API (Application Programming Interface) for such a system and similar systems is proposed. This utilises computer vision techniques to analyse a surface to track objects and the subsequent projection of graphics back onto the space creating a feedback system for interaction. An attempt to gather together literature and examples of such existing systems is made. The fundamental principle of the interaction model described is summarised in the sentence below:

> Physical objects on an *action surface* have interactive properties; each object is both a sensor and actuator.

Contents

List of Listings

List of Figures

Chapter 1

Introduction

1.1 Project aims

- open source project for tangible interaction
- Prototypical
- ethnomethodological frameworks for evaluation

Chapter 2

Background

The motivation for this project stems in part from a feeling of frustration in how working on computers can often be a constricted and static affair. From this came a pondering of how one might expand, and move away from, the *keyboard-mouse-monitor* model to improve the utility of computers regarding our physical health, wellbeing and perceptive abilities. How might a spatial, haptic and tangible environment for interaction create an improved space for working and thinking with computers as well with our physical health [1]? How might such an environment fundamentally augment our cognitive capabilities; memory and learning as well as creativity itself?

2.1 Definitions

2.1.1 HCI - Human Computer Interaction

2.1.2 KMM - keyboard-mouse-monitor model

2.1.3 Exocortex -

2.2 *Exocortices* and augmented cognition

I started off looking at *Exocortices* and other personal archiving systems. Systems that allow the user to externalise thought and memory. This could be via simply storing and organising work and ideas efficiently and methodically or unifying many tasks or different workflows into a singular interface.
Org mode is a good example of such a system. Org mode is a "computing environment for authoring mixed natural and computer language documents" [2]. Designed for taking notes, producing documents and organising, it runs inside of the text editor, Emacs. It has the ability to export to different formats such as HTML, LaTeX and supports "outlining, note-taking, hyperlinks, spreadsheets, to-do lists, project planning, GTD" as well as literate programming, all in plain-text [2] [1].

[1] This document is produced with org-mode.

Another point of reference when I was looking at externalised 'artificial information-processing systems' was Devine Lu Linvega's Exocortex XXIIV – nataniev. *XXIIV* is a personal archive and log with documentation of Linvega's personal tools and artworks. Originally a static, javascript and lisp based website with diaries, blog type posts and categorised personal logs, it is now somewhat stripped back in style and has been rewritten in C (C99) [3].

Both these two systems have their own specific use-cases; *Org-mode*– in academia and *XXIIV*– an experimental personal archive. They both utilise the contemporary and prevailing *keyboard-mouse-monitor* paradigm of computer interaction to push the boundaries of cognition in this medium, particularly regarding memory and productivity. These two projects were a birth point in thinking about how software systems can augment thought and improve learning ability and productivity.

Information visualisation is another tool for the amplifying cognition that most take for granted. The externalisation and translation of data into shape and colour allows us to see patterns not easily seen in listed data. Furthermore utilising visualisation for memory tasks by organising attention and concept mapping are useful ways to increase our abilities [4]. Scientist Michael Nielsen also offers some approaches to increasing long term memory through the use of simple flash card software that orders things as you review them by how well you know them. He suggests this and the process itself of creating question/answer flashcards improves memory capacity, understanding and our ability to do deep readings of a subject [5, 6].

2.3 A virtual exploration of a 'dynamic land'

Another principal point of reference was *Dynamicland*, a research project in Oakland, USA. The aim of the project is to implement and research a new more powerful and accessible model of computing.

> In Oakland, we built the first full-scale realization of the vision, inviting thousands of people into our space to collaborate. Together, these artists, scientists, teachers, students, programmers, and non-programmers created hundreds of projects that would have been impossible anywhere else. – Dynamicland.org

Dynamicland is a communal computer where the building is the computer (ENIAC). Programs are embodied in the room on pieces of colour-coded paper. The programs are recognised via the codes and their code, stored in a database is then run, it can also *read* code using OCR but generally the code is there symbolically. Projectors on the ceiling transform the paper and workbenches into whatever the programmer decides. This relatively simple model makes for an exciting new ecosystem for collaborative computing and expressive programming. Victor highlights his ideas for the progression of computing and interaction in a series of talks (available online) and on his website. In his talk "Seeing Spaces" he describes a new kind of maker-space which allow makers to see across time and possibilities. *Dynamicland* seeks to offer a computational medium which allows for full use of the human senses;

a more humane representation of thought [7].

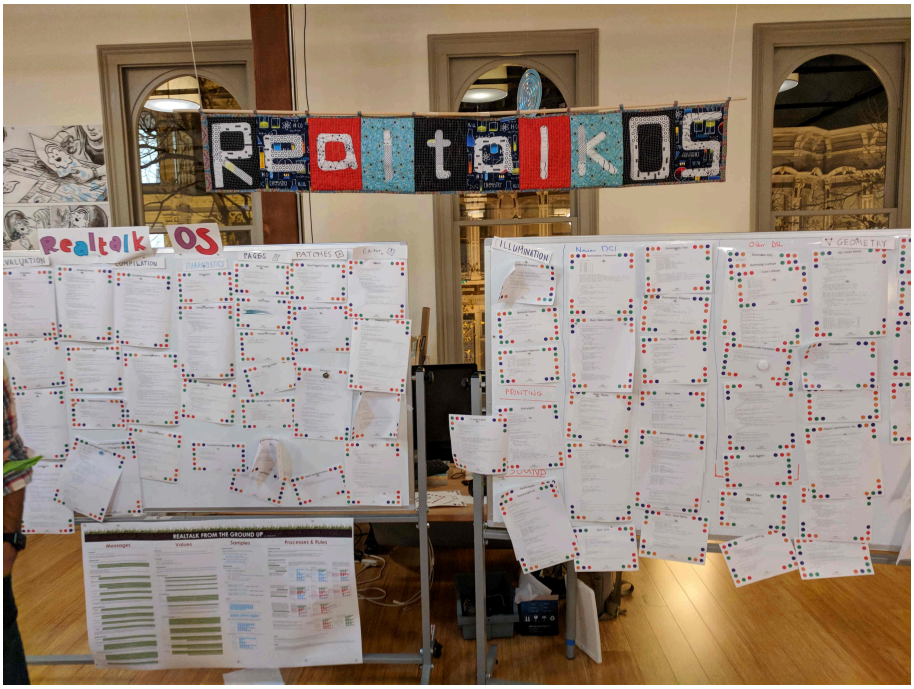

Figure 2.1: RealtalkOS, the operating system of *Dynamicland*

DL was a major inspiration for the main technical model for this project, an *augmented* workspace either on the floor or a table which is projected onto. A camera/s pointing down onto the projection space is the sensor for detecting interaction, with the projector as the actuator. This base model can be seen in Figures 2.3 and 4.1.

2.4 Paper programs

Looking to find some of the code for *Dynamicland* (DL) and a more detailed specification of **DL** I stumbled across *Paper Programs* (PP) (*Dynamicland* has an 'open-source model', but it is only open if you can visit it physically as the source code is physically in the space). *Paper Programs* (PP) is a browser-based partial clone of *Dynamicland*. PP takes one element of dynamicland, i.e. the representation of computer programs in a spatial environment, on pieces of paper. Programs are written in Javascript and stored in a Postgresql database. This idea of 'physicalizing' some method or element of the computer and allowing the direct haptic manipulation of it has further inspired this project.

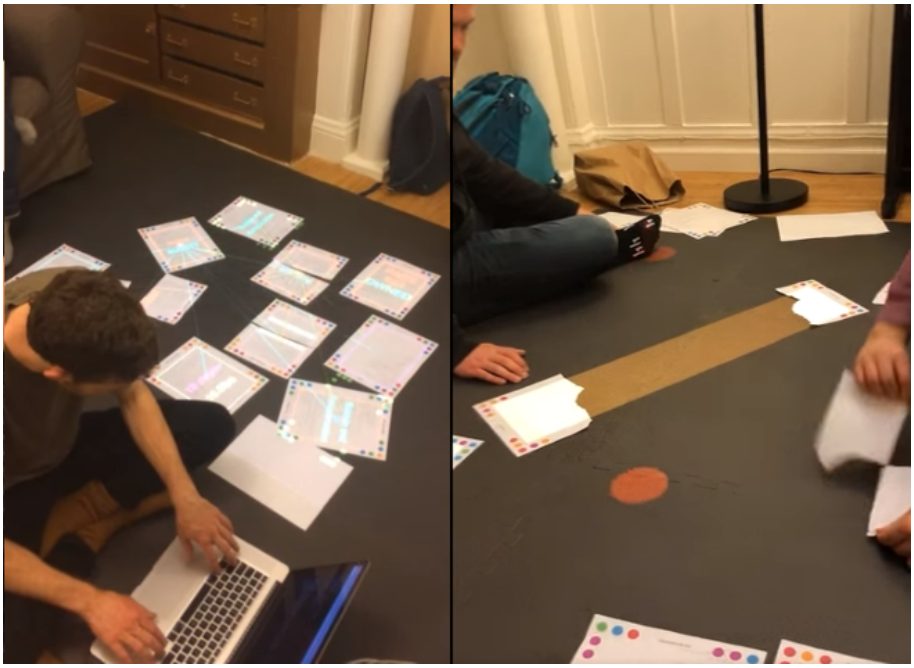

Figure 2.2: *Paperprograms* in action

PP aims, like Dynamicland, to create a collaborative programming environment where anyone in the space can write Javascript programs and interact with others. As in DL each program has a unique code and a colour encoding. It follows the same basic hardware model. That being a projector and camera on the ceiling and the paper "programs" (See Fig. 2.3.). This new vision of collaborative computing and somewhat "multi-modal" interaction is one of the initial inspirations and an important reference for this project.

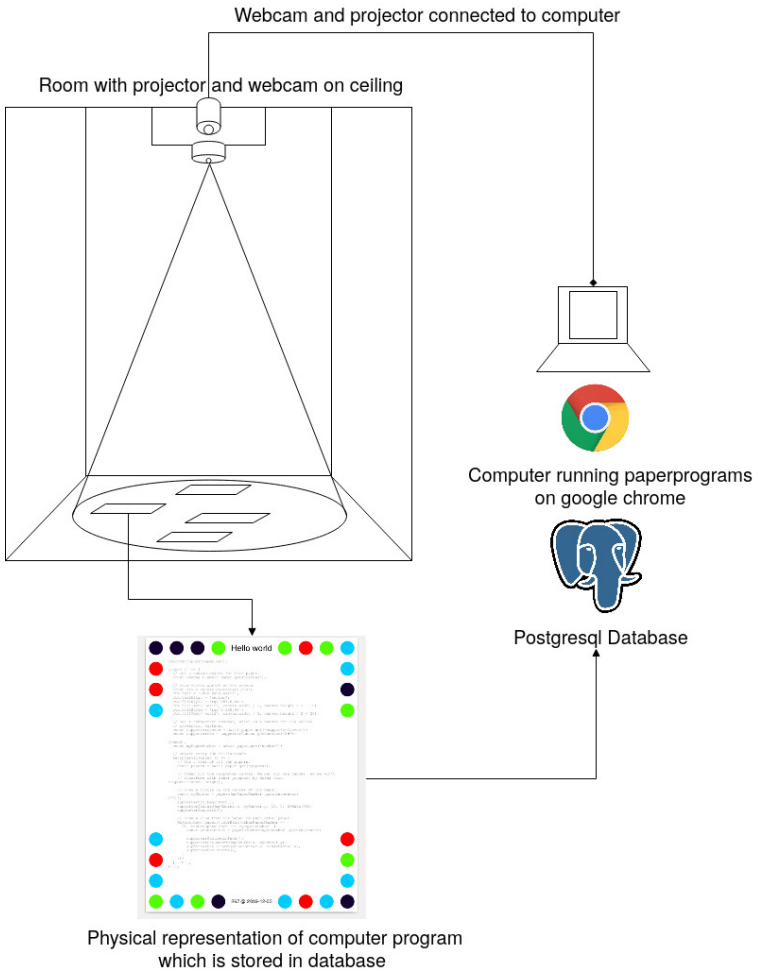

Figure 2.3: The initial physical schema: *Paperprograms*

2.5 Tangible bits - Ishii and Ullmer

Another significant reference exploring novel approaches to interaction involving physical objects was the paper: *Tangible bits: towards seamless interfaces between people, bits and atoms* (1997). It describes the motivation for users to be able to "grasp and manipulate" bits, making them "tangible". The paper also presents three prototypes, – the *metaDESK*, *transBOARD* and *ambientROOM* and establish a new HCI approach "Tangible user interface[s]" (TUI) with equivalence to Graphical user interfaces (GUI's) [8]. It is an academic precursor to Dynamicland and is a starting point for tangible interaction, merging ubiquitous-computing, augmented reality and psychological approaches to HCI.

2.6 Implementation and abstraction

In the SAGE Handbook of Digital Technology Research's chapter on Haptic interfaces design parameters are listed:

- Cutaneous Perception
- Frequency
- Duration
- Rhythm
- Location
- Intensity
- Texture
- Kinesthetic Perception
- …

These parameters present considerations for the design of such interfaces but also a formalisation of haptic interaction in the abstract [9]. It takes the possible elements of 'hapticity' and lays them out. This motivated a second outcome to the implementation itself, to construct a *formal* specification for spatial and tangible interaction so as to describe the elements conceptually. This could then be used for further development of similar systems and allow for multi-disciplinary scientific experimentation. The benefits of having such a blueprint would be to present spatiality and tangibility (in relation to HCI) formally so as to allow for identification of elements for use.

Future researchability potential. [10]

2.7 Multi-modal interaction

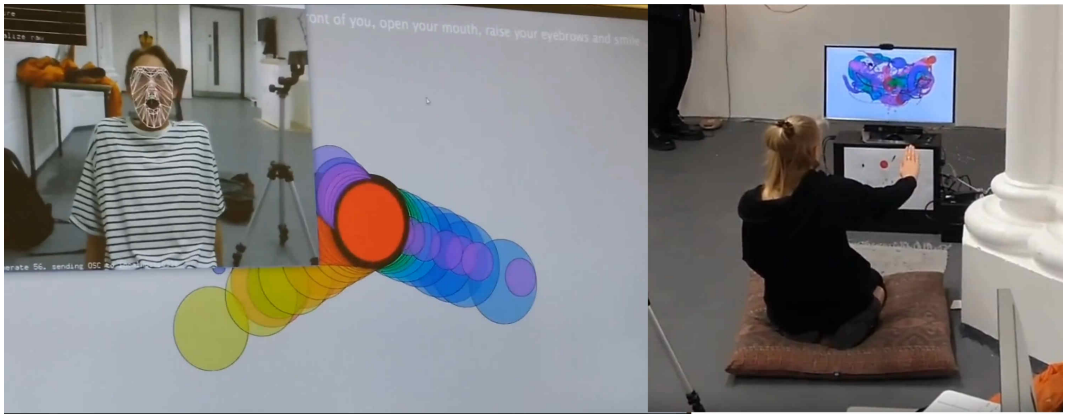

Figure 2.4: Multi-modal painting

An experimental project I produced in 2017 has also informed the direction of this project. This work was a multi-modal paint program; where hand movements and facial expressions controlled different parameters of a paint program. This included colour, size and position of the stroke. Additionally the different modes of input were also controlling parameters on a looping synthesizer. The installation was multi-modal in input and output. It was an artwork in outlook and formed an initial experiment in designing interaction. The work was particularly successful with children, who seemed to quickly get the hang of the controls. It

also included the combination of a variety of inputs to interaction with a variety of outputs. Thought not necessarily the most effective or widely applicable it explored the capabilities of some more unusual interactive modes.

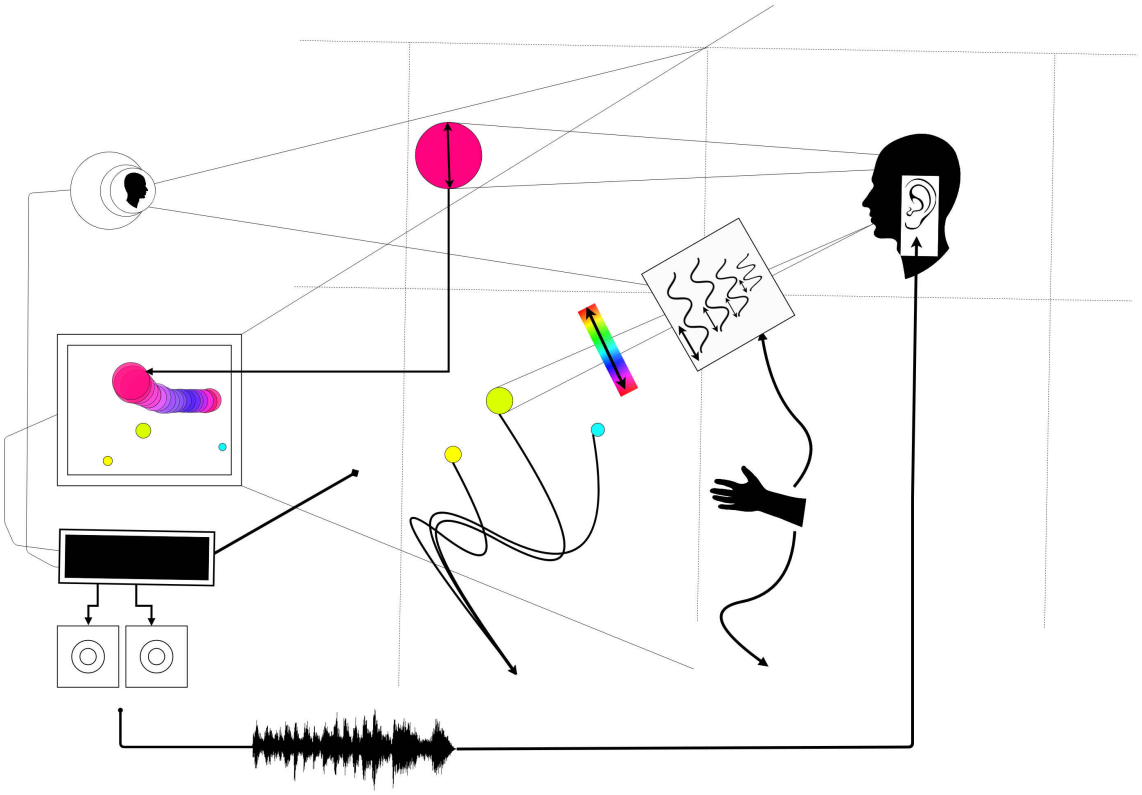

Figure 2.5: Multi modal schematic

Chapter 3

Specification and context

3.1 Brief

To sum up the fundamental principle of the style of interaction that this document aims to describe is summarised in the sentence below.

> Physical objects on an *action surface* have interactive properties; each object is both a sensor and actuator.

I provide this foundation so as to differentiate it from commonly used contemporary systems. It highlights that a 'live' surface will act as a space where objects are augmented with additional properties i.e. input and output to a computer system.

3.2 Technical

As in the original specification the aim was to create a system for spatial interaction. Initially I imagined it to work on a table top surface (in the end it was developed on a floor mat due to considerations in my development environment; see Chapter 4). The other principle component was that interaction would be based on the placement and movement of objects around the work-surface. The position and movements of these objects would be picked up by a camera and actuated by a projector; both situated above the surface looking down onto it. A horizontal setup would also be possible, with for example, magnetised components keeping the objects to a board. Alongside the spatial objects a computer keyboard may be used for additional input such as inputting text or formatting.

The original specification involved using *Paper Programs* and build on top of this. With the *PP* system, I planned to write a program/s to explore the psychology of interaction with such a system. This could take the form of a game-like psychology experiment. Rather than risk attempting a psychology thesis, within a computing project focus has been put on creating and exploring the implementation and formalisation of the interaction model itself. Due to technical issues with *PP* and the motivation to explore an alternative interaction model, I decided to implement the system using **openFrameworks**, a C++ toolkit for experimental application development. I chose this framework as it has straightforward 'out

of the box' graphics capabilities as well as numerous add-ons. These include *OpenCV* [11] wrappers and GUI libraries as well as an active community of users. This combination in one framework seemed suitable for quick experimentation and prototyping for this project. Other C++ libraries were to be considered; Cinder and OpenCv as well as just OpenCv. The physical setup would include a Projector and HD webcam and computer for running the application. See Fig. 4.1 for the software and hardware schematic for this technical conception.

3.3 Design considerations

An important design consideration that has driven this project is accessibility. From my research into similar projects an aim was to create a platform, that would be open source and easily setup, so that others could easily run and further develop it. This was another reason for using openFrameworks which is cross platform (Windows, OSx, IOS and Linux). This would mean with minor or no modification of the code, it could be run on all the major desktop platforms. The hardware requirements are also the kind which are either cheaply (relatively) sourced or commonly available in educational institutions (one of the target areas for which further development was envisioned).

Due to the limited scope of this project in both time and academic context a secondary theoretical component is conceived[1]. This is in the form of a theoretical specification and API for this project and similar systems. As discussed previously (2.6) a set of parameters and variables can form a useful part of a conceptual illustration and formalisation. This would include diagrammatic illustrations of different classes representing elements of the system, such as I/O and transformable objects.

[1] Due in part to the ongoing Coronavirus pandemic.

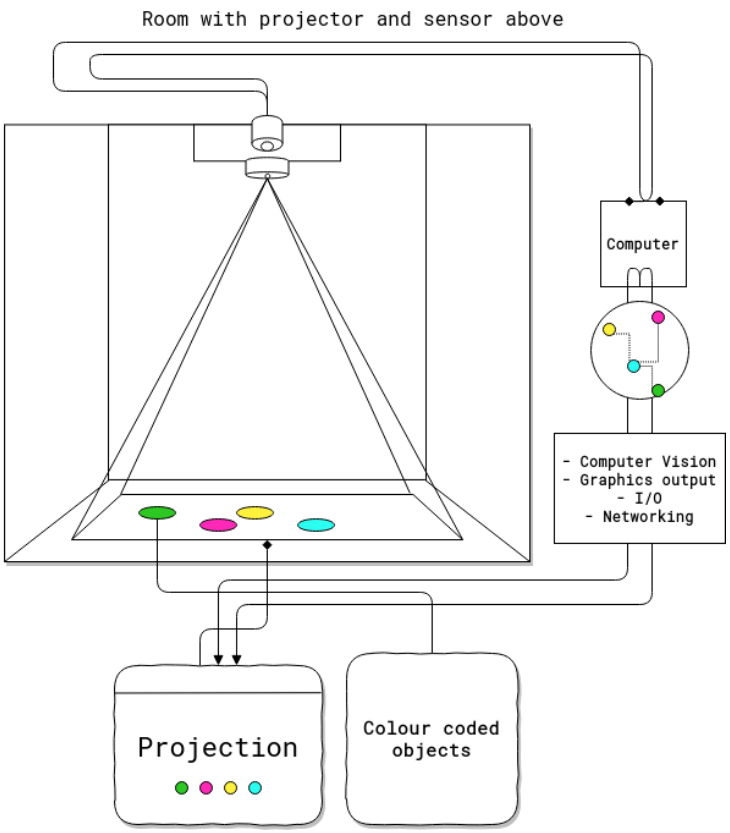

Figure 3.1: Abstract system schema

The formalisation will address how various aspects of this interaction model can distribute and externalise cognitive work. *Annotating* (such as crossing out or underlining) and *Cogntive tracing* (manipulating items into different orders or structures) are two methods for externalising cognition. These two methods and others methods will be connected to elements of the interaction model. [12]

3.4 Users

As an academic and open-source software design project the intended audience for the work can be split into two categories. This would be open-source developers and technologists and academics working in the fields of HCI and other related disciplines such as Cognitive Science and Psychology.

As an open-source project this project aims to attract programmers interested in exploring new models for interaction. How can a desk or room be transformed into a new interactive medium. Those with specialisations in different areas of computing and beyond could contribute to different branches of advancement. To present outcome as an open project gives scope for further development which the scope and context of this thesis has not allowed for. With the theoretical outcome an academic audience is intended. Scientific exploration of the ideas in this report could allow for optimisation of the purported benefits and modelling of

interaction. Cross over between these two above distinctions is also likely and this project hopes to sit at the intersection of the two.

Chapter 4

Project in depth

4.1 Finalised Design

After testing of different software and approaches (detailed in Chapter 5) the setup for the software outcome was decided. This is illustrated in Fig. 4.1. The hardware used was an Epson EH-TW650 3LCD, Logitech C920 HD Webcam and a laptop running Ubuntu Linux (18.04 LTS). The projector was secured to the ceiling with a mount and all cables were extended to the floor. The projector setup can be seen in Appendix I, Fig 8.1. All the source code can be found at the following: Gitlab link.

The software architecture consists of three classes:

- `ofApp`, creates the GUI interface window with controls for tweaking CV settings and input parameters
- `Projector`, this class creates the projector window.
- `State`, this class stores variables that can be shared between the `Projector` and `ofApp` classes.

4.2 Implementation details

4.2.1 Computer vision and fundamentals

The first essential component to get working was the computer vision. The core of this involves doing blob tracking for each colour in the `targetColours` times calling `findContours` and passing in (by reference) the cropped pixel array using the corresponding `contourFinder` object. Therefore, we loop five array, an `ofPixels` object containing the camera pixel data for the active detection region.
Five different colours were chosen as it is the same as in *PP*. Given its identical hardware setup it seemed a good number. Having more colours means thresholds will be lower so as to distinguish between less distinct colours; for example pink and red. The contour finder has a number of parameters which allow for fine grained control over the tracking. They are listed below:

- `TargetColor`

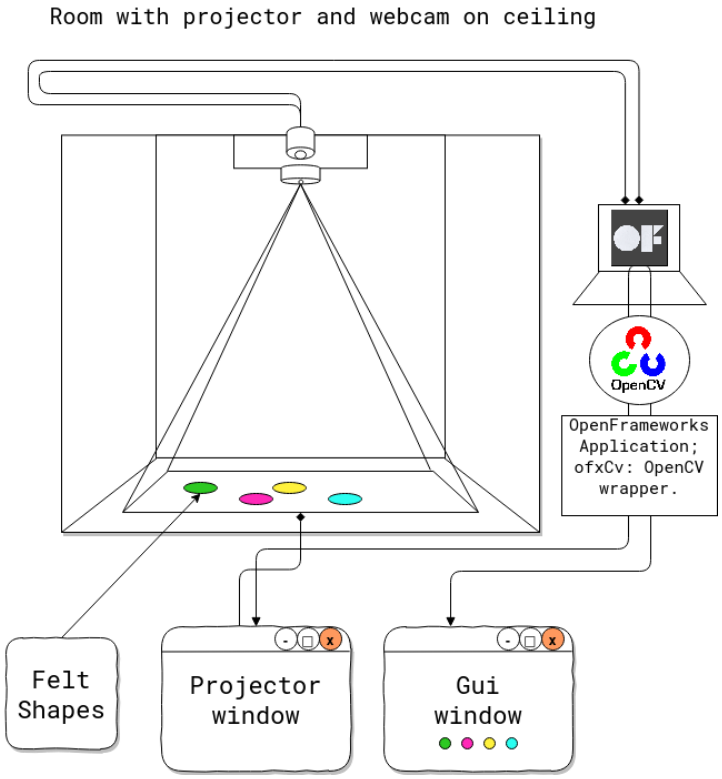

Figure 4.1: Finalised system schema

```
// Check new frame
if(cam.isFrameNew()) {
    // Loop for number of colours and track target colours
    for(int i = 0; i < num_colours; i++){
        // if finding: find // cv on / off
        if(ss->find) ss->contourFinders[i].findContours(camPix);
    }
}
```

Listing 1: Computer Vision with ofxCv

- `Threshold`
- `MinArea`
- `MaxArea`
- `MinAreaRadius`
- `MaxAreaRadius`

Architecturally the application is comprised of two windows the **GUI** and **Projector**, represented in two classes `ofApp` and `Projector` respectively. The **GUI** window is a control panel or the computer vision tracking. Controls for the parameters are available in the **GUI** window as well to crop the active region part of the camera frame were the computer vision happens. In the screenshot (Fig. 4.2) the tracking parameters are seen on the left and the target colours are on the right. In the center the rectangle with the pink circles on upper left and bottom right corners is the active detection region.

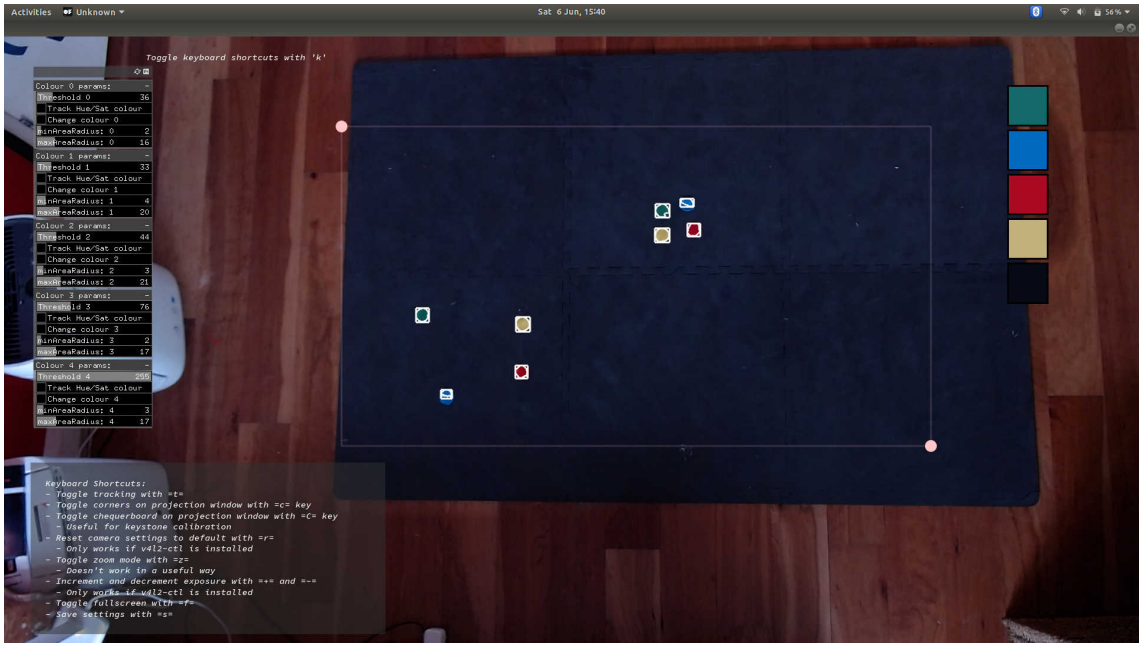

Figure 4.2: GUI window.

The other main window used is in the **Projector** class. This deals with the display of the reaction surface. The crux of what this class achieves is in the mapping and locating of the various colour blobs detected by the `ContourFinder`'s. This is shown in the code block 2. The `contourFinders` are accessed via the `State` class [1]. All of the areas of interest are looped over and their *centroids* accessed. The locations are mapped to the projector window size and this and the colour index is stored.
An example of detection and a corresponding projection can be seen in Appendix I (Figure 8.2).

[1]This is the third class which allows for the sharing of variables and objects between the `GUI` and `Projector` classes. It is consists of a Shared Pointer to the State class, `shared_ptr<State>`, which is passed as an argument to the `GUI` and `Projector` classes.

```
for (auto j = 0; j < ss->contourFinders[i].getBoundingRects().size(); j++) {
  cv::Point2f p_;
  cv::Point3f p__;
  // Get centre of blob
  p_ = ss->contourFinders[i].getCenter(j);
  // map cropped camera to window
  p__.x = ofMap(p_.x, 0, ss->width_height.x, 0, mw);
  p__.y = ofMap(p_.y, 0, ss->width_height.y, 0, mh);
  // Store location and colour index
  p__.z=i;
  blobs.push_back(p__);
```

Listing 2: Crucial projector code.

4.2.2 Settings

To allow for tweaking and debugging during further development there is the ability to save the settings of the computer vision parameters. This uses the ofxXmlsettings addon. In the `setup()` method of the `ofApp` class we load and loop over the settings. There is also a function, `saveSettings()`, which allows one to save settings at any time. This is assigned to the `s` key.

4.2.3 GUI keyboard shortcuts

The GUI interface has the a bit more that it is relevant to quickly describe. The keyboard shortcuts allow for various controls of the interface. A chequerboard and corner markers can be toggled on the projector window. A simple zoom mode can be enabled but is not very functional. There is also some interfacing for v4l2-ctl, a CLI application for controlling the settings on the camera. This allows for quick and dynamic controlling exposure and other settings, which can be useful when getting an optimal image for colour and blob detection. The full list of shortcuts is listed below.

- Toggle keyboard shortcuts with `k`
- Toggle tracking with `t`
- Toggle corners on projection window with `c` key
- Toggle chequerboard on projection window with `C` key
  - Useful for keystone calibration
- Reset camera settings to default with `r`
- Toggle zoom mode with `z`
  - Doesn't work in a useful way
- Increment and decrement exposure with `+` and `-`
  - Only works if v4l2-ctl is installed
- Toggle fullscreen with `f`
- Save settings with `s`

4.3 Abstract Specification

Here I will discuss the theoretical segment. This is brief speculative look at how we can and might further model the elements of interaction in a formal way. I have split it into three parts: data structures, physical elements and Sensory devices. This offers three different perspectives on the abstraction and formalisation process.

4.3.1 Sensory devices

Identified here are four main parameters that one can think of as input or sensor categories to the camera and processing algorithms. They are listed below. These parameters can be variously tweaked and manipulated to interact with a program. There is cross over between, such as with pattern and shape where patterns can be combinations of shapes which forms shapes in themselves. They can also be combined in various ways so as to produce interaction. In fact they will likely be most useful when combined as it stretches the possible arrrangements and states that can be created.

- Colour
- Shape
- Location
- Relative position and arrangement
- Pattern

For example, as in the Colour Locator prototype different arrangements of coloured shapes can act as marker points for location in the space. Different combinations of these shapes can become symbolic for objects or images that the program associates with them.

4.3.2 Data structures

Here are the theoretical data structures. These can focus arround the sensory parameters described above.

```
Template Colour {
  vector<int> HSB_VALUE || RGB_VALUE;
  int ALPHA_VALUE;
}
Template Shape {
  int SIDES;
  vector<int> ANGLES;
}
Template Location {
  int X;
  int Y;
}
Template Pattern {
  vector<int> VALUES;
}
```

It can also be useful to think about what the data structures or higher order combinations of the data structures might represent. What analogues of Gui elements or other digital structures could they correspond to.

4.3.3 Physical elements

When building the Colour Locator system was using felt circles of five different colours were used.This model could also be expanded beyond the scope of the setup in the Colour Locator. Here we use a camera for detection but other kinds of sensors would be equally useful. A depth sensor would be great for stability only tracking colour that is beyond a certain depth.

4.4 Relative point mapping

Another element of the software outcome is this elementary algorithm for finding pairs of points. It looks for pairs of points that are less than some distance away from each other and then collects them and stores them in an array. This algorithm is currently very slow, with a worst case algorithmic complexity of roughly $O(k * n^3)$, where n is the number of points (blobs) and k is the number of pairs [2].

4.5 API

In the software outcome there is only a rudimentary "API" which is to access the colour points. If can currently only be accessed inside the program at the current time. There is no networking or connectivity. For each detected blob you have its colour id (a number from 1 to 5 corresponding to each of the tracked colours), location (x and y coordinates). These active points form the basis with which to build other augmentation on top of. In the current version of the software these values are stored in a simple 3 dimensional vector from the **openCv** library (`cv::Point3f`) (see Fig. 4).

A simple proposed class for each blob seen in Fig. 5. Having this as a class would important for extensibility. It may remain a relatively simple class as other more of the processing could be done on top of the colour point detection.

[2]This may not be precise but the main takeaway is it is not scalable. It runs ok with a few points and tight thresholds but it becomes very slow if there is many points of interest.

```
vector<vector<int>> Projector::findPairs(vector<cv::Point3f> &blobs) {
  vector<vector<int>> pairs;
  for (int i = 0; i < blobs.size(); i++) {
    for (int j = 0; j < blobs.size(); j++) {
      if (i != j) {
        float dist = ofDist(blobs[i].x, blobs[i].y, blobs[j].x, blobs[j].y);
        if (dist < 400) {
          // Loop over pairs
          bool _found = false;
          for (int k = 0; k < pairs.size(); k++) {
            vector<int>::iterator iti, itj;
            iti = find(pairs[k].begin(), pairs[k].end(), i);
            itj = find(pairs[k].begin(), pairs[k].end(), j);
            // Check pair has already been found
            if (iti != pairs[k].end() && itj != pairs[k].end()) {
              // Push pair to pairs
              // pairs.push_back({i, j});
              _found = true;
            }
          }
          if (!_found)
            pairs.push_back({i, j});
        }
      }
    }
  }
  return pairs;
}
```

Listing 3: Algorithm for mapping and connecting points.

```
ss->blobs[n].x // X position
ss->blobs[n].y // Y position
ss->blobs[n].z // Colour id
```

Listing 4: Accessing the the parameters for point 'n'

```
class colourPoint {
    public:
        colourPoint(loc, col_id){
             location=loc;
             colourId=col_id;
        }
    Point2f location;
    int colourId;
}
```

Listing 5: Proposed point class.

Chapter 5

Creative process and software testing.

As mentioned *Paperprogams* was a starting point for playing around with but I found that I couldn't set it up and have it stable enough to develop on. It also suffers from being quite slow, due to the Computer Vision and graphics being done in the browser (it uses a version of OpenCv compiled to WebAssembly) [13]. While WebAssembly has the scope for doing high-performance computation in the browser but I found there was still a significant lag from detecting papers to projecting back down on to them. Another branch which had implemented blob detection on the GPU I also found to be slow and unstable (Link to pull request), this may have been due to my lighting and camera setup.
After testing with *PP* and finding it to be unstable and difficult to develop on Cinder and OpenCV were considered. Another reason for moving away from *PP* was it already being a fully fledged system in itself. It has potential for developing some interesting tools collaboratively but for this solo project working alone the social aspect would not be utilised.

5.1 pimapper, projection mapping issue.

5.2 Goverened by

5.2.1 ...technical implementability

5.2.2 ...research and experience

5.3 Raspberry pi testing

5.4 Practicality of current setup

5.5

Chapter 6

Debugging and problem solving

6.1 Main technical issues

6.2 Problem solving

Chapter 7

Evaluation and Conclusions

7.1 Scope

7.2 Theoretical

7.3 Social aspects. Proposed social evaluation

7.4 Future scope for software development

7.4.1 Improvements

7.5 Survey?

Chapter 8

Appendices

8.1 Appendix I: Additional images

Figure 8.1: Camera and projector secured on ceiling.

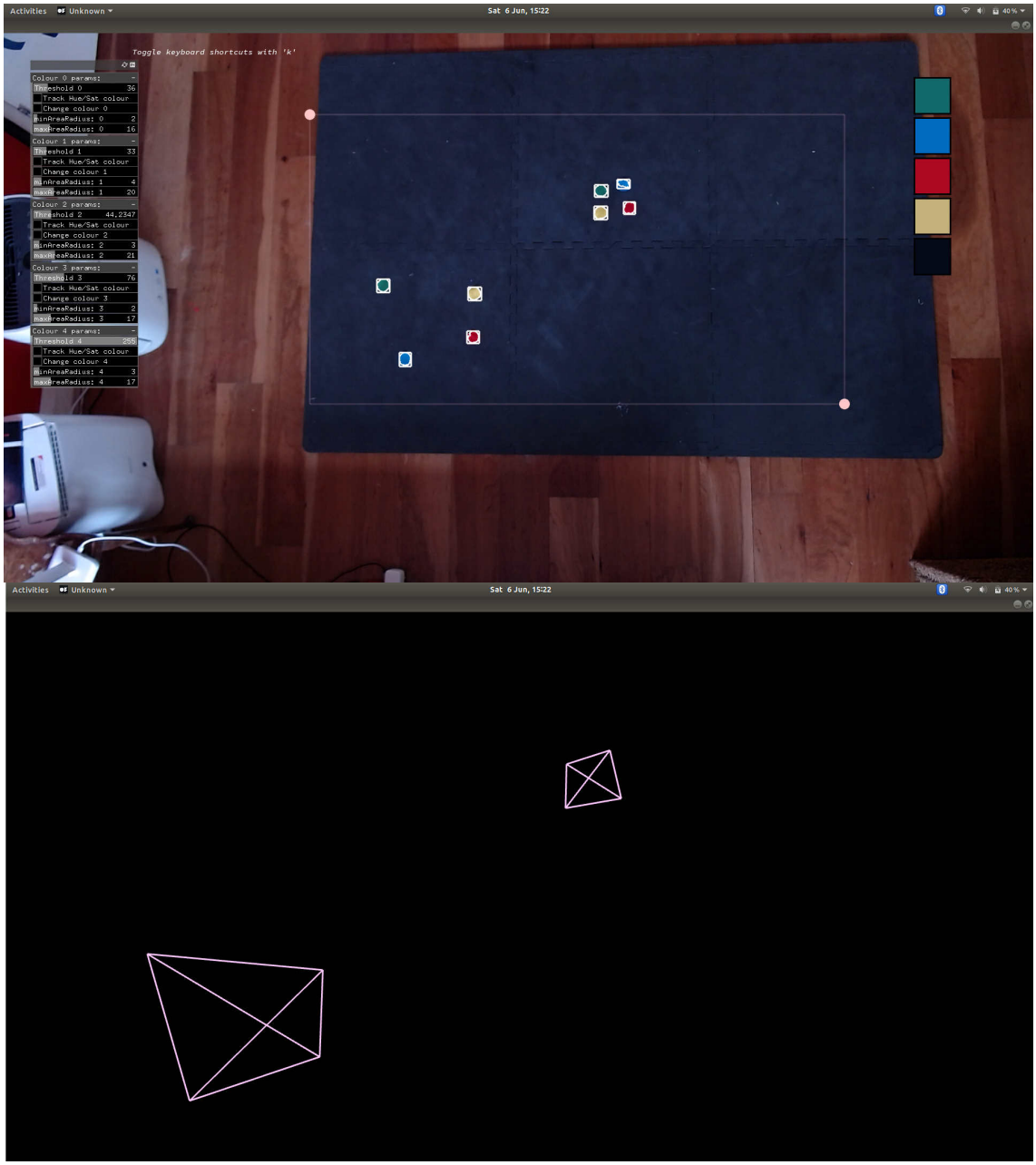

Figure 8.2: Detection and Corresponding projection.

Bibliography

[1] J. M. Twenge, "Why increases in adolescent depression may be linked to the technological environment," *Current Opinion in Psychology*, vol. 32, pp. 89–94, 2020.

[2] E. Schulte, D. Davison, T. Dye, and C. Dominik, "A multi-language computing environment for literate programming and reproducible research," *Journal of Statistical Software*, vol. 46, pp. 1–24, 1 2012.

[3] D. L. Linvega, "The nataniev ecosystem is a collection of exocortex tools," 2020, 18W04, https://wiki.xxiivv.com/site/about.html.

[4] C. Ware, *Information visualization perception for design*. The Morgan Kaufmann series in interactive technologies, Waltham, MA: Morgan Kaufmann, 3rd ed.. ed., 2013.

[5] M. A. Nielsen, "Augmenting long-term memory," 2018, http://augmentingcognition.com/ltm.html.

[6] S. Carter and M. Nielsen, "Using artificial intelligence to augment human intelligence," *Distill*, 2017. https://distill.pub/2017/aia.

[7] B. Victor and A. Kay, "Dynamicland," 2014, https://dynamicland.org.

[8] H. Ishii, "Tangible bits: designing the seamless interface between people, bits, and atoms," in *Proceedings. International Symposium on Mixed and Augmented Reality*, pp. 199–199, IEEE, 2002.

[9] S. Higgins, "The sage handbook of digital technology research," *International Journal of Research Method in Education: E-research in Educational Contexts: The Roles of Technologies, Ethics and Social Media*, vol. 38, no. 3, pp. a. pp 336–337 b. pp 144–158 c. pp 217–235, 2015.

[10] J. Lazar, *Research Methods in Human-Computer Interaction*. 2nd edition.. ed., 2017.

[11] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[12] H. Sharp, *Interaction Design: Beyond Human-Computer Interaction*. 5th edition edition.. ed., 2019.

[13] JP, "Paper programs is a browser-based system for running javascript programs on pieces of paper," 2018, https://paperprograms.org/.