# 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 12, 2020

# Acknowledgements

In no particular order thanks to Florent, Liz, Andrew, as well all my fellow living companions in Ladywell who've come together in this unusual time. Special thanks Jamie Forth for humoring the exprimental direction of this project and offering diverse references and support.

# 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 specification (Application Programming Interface) this model and similar models 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 tracked by colour and located on a two dimensional surface.

# Contents

# List of Listings

# List of Figures

Chapter 1

# Introduction

This is a project for the discussion and development of a computer vision based interaction system. Originally it (maybe ambitiously) aimed to create a fully fledged software solution, to explore the psychological aspects of interacting with computers by moving objects around in a space. Motivated by years spent in some space or another, hunched in front of a screen in all manner of contortions, it is the aim of this work to challenge this composition and contribute an alternative. It is an attempt to further expand computer interaction and assimilate computing with space.

Presented here is an open-source software prototype, a spatial interaction system where coloured objects (in this case felt shapes), are detected and located by a camera and projected back onto. This model of spatial objects on an interaction surface is the proposed departure from the keyboard-mouse-monitor paradigm. The floor (or table) becomes the monitor and objects in the space the mouse. The potential for the use of a keyboard for text input is also envisaged, and beyond that, any kind of controller that could be used in space. The software is named

This project presents an open-source application using interaction model described above, named **Colour Locator** (abbreviated with **CL** throughout). The software uses OpenFrameworks (a C++ toolkit for creative coding)(OF). This library allowed for speedy assembly with out-of-the-box consolidation of various graphics, computer-vision and GUI libraries. In this case ofxCv, ofxXmlSettings and ofxGui, alongside the built-in graphics functionality of OF. Due to the large number of additional libraries and active community it seemed a good environment for future expansion and collaboration.

Alongside the software, an abstract specification is described. This considers this interaction model in a conceptual manner, so that it might be analysed from a multi-disciplinary scientific perspective. Human-Computer Interaction, as a discipline, lives at the cross roads of diverse fields such as computing, design and behavioural science. Although the scope and context has not allowed for extra-disciplinary collaboration, through the presentation of this secondary component it aims to keep itself open to this future possibility.

In its current form the potential user base would be the author and other software developers, interested in pursuing and developing the software in an collaborative way. The far reaching

end goal is one of prototyping software for diverse and non-technical users, essentially to make this model available to any computer user. A critical reference that I would like to put special note to here is the work of Bret Victor and Alan Kay, particulary in the rather elusive project Dynamicland (based in the US) and Victor's other writing and design. This was a cornerstone of the contextual research, a vital point of reference and inspiration for the project.

A contextual and historical N.B.

The original aims of the project were partly affected by the restrictions brought about on daily life due to the Novel coronavirus pandemic.

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 Acronyms / definitions

- CL - **Colour Locator**, the software prototype detailed in this report.
- CV - Computer vision
- HCI - Human Computer Interaction
- KMM - keyboard-mouse-monitor
- Exocortex - An externalised software extension for a one or more cognitive tasks.
- GUI - Graphical User Interface

## 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].

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

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

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.

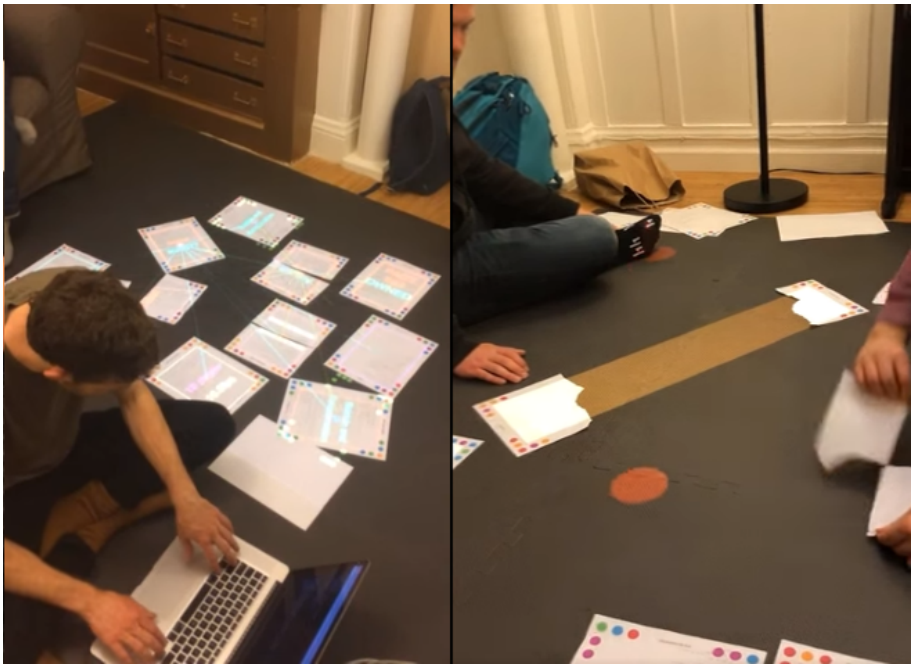

Figure 2.2: *Paperprograms* in action

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.

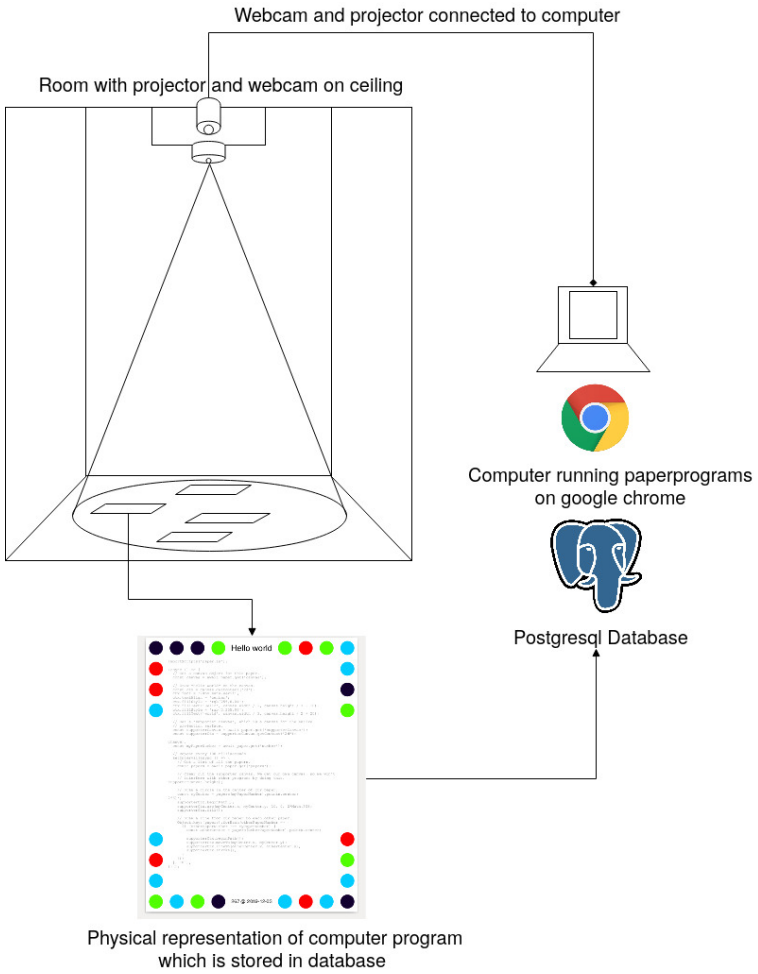


Figure 2.3: The initial physical schema: *Paperprograms*

## 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

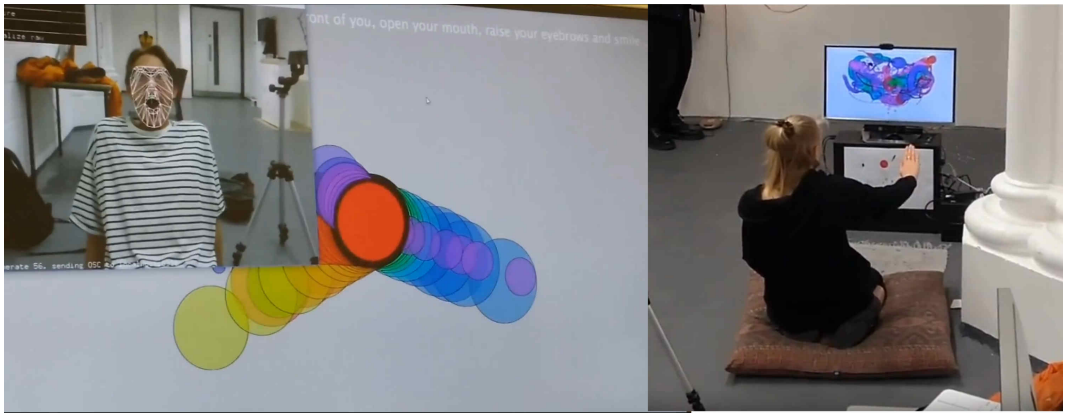

Figure 2.4: Multi-modal painting

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.
[[]

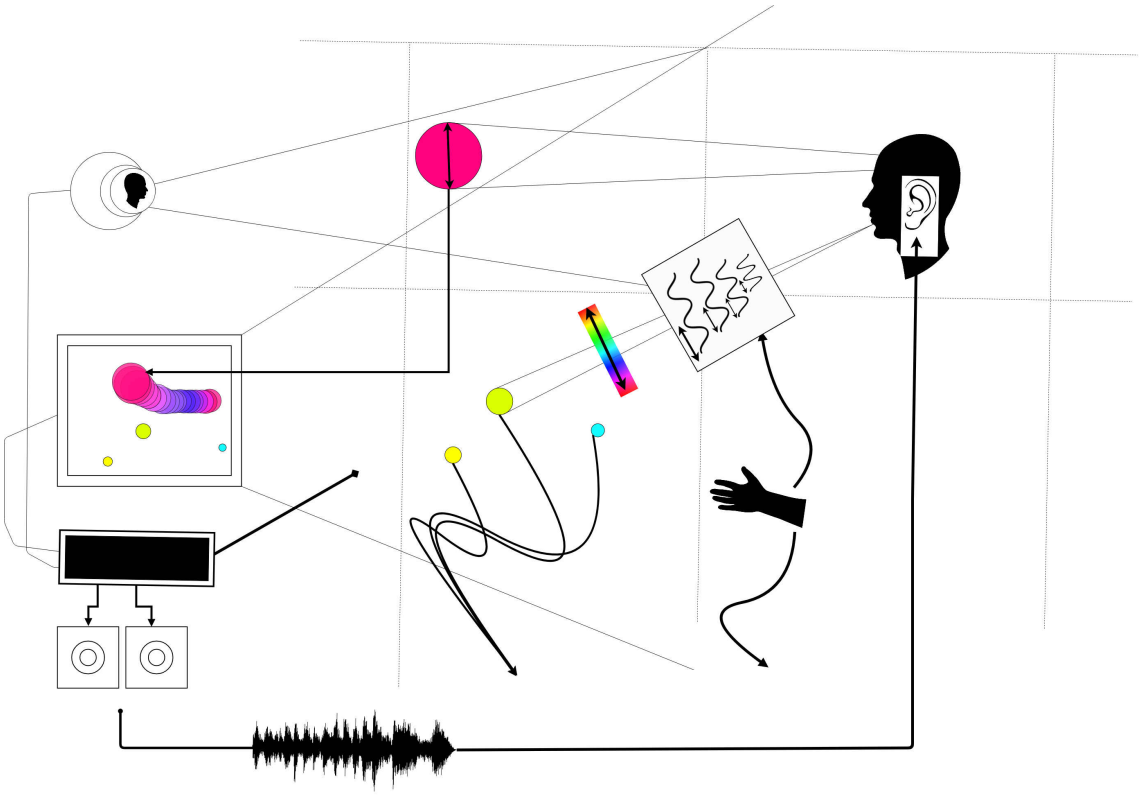

Figure 2.5: Multi modal schematic

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 tracked by colour and located on a two dimensional surface.

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.

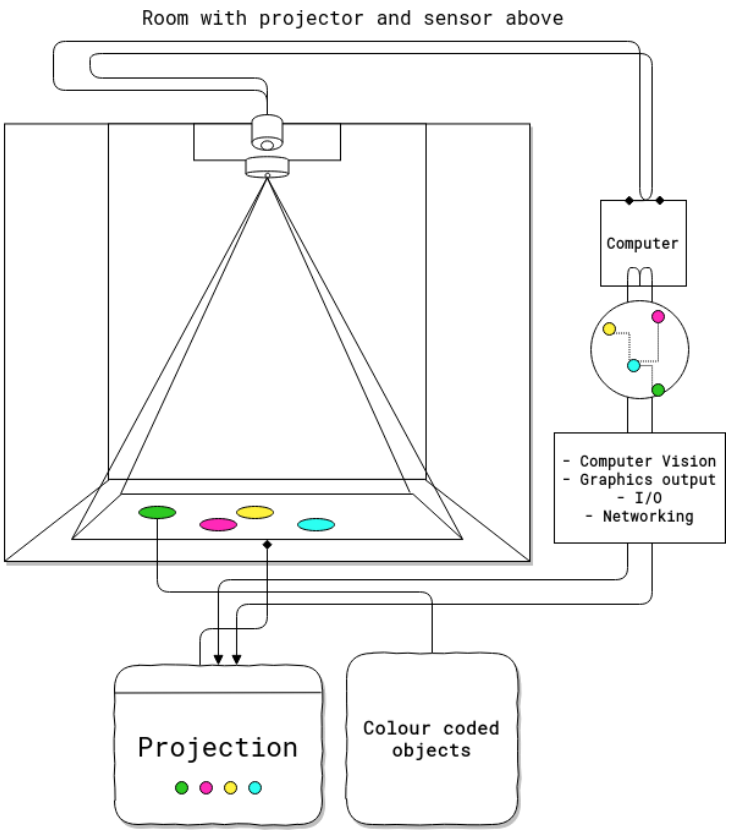


Figure 3.1: Abstract system schema

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 *Cognitive 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 the testing of different software and approaches (detailed in Chapter 5) the setup for the software outcome was chosen. This is illustrated in Fig. 4.1. The hardware used was an **Epson EH-TW650 3LCD**, a **Logitech C920** HD Webcam and a laptop running Ubuntu Linux (18.04 LATS). 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 by following links in Appendix II (8.2). See Video link 8.3.1 for video documentation of all the different elements of the system.

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 blob tracking for each colour in the `targetColours` and calling `findContours`, 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. See video link 8.3.2 for demonstration of this.
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:

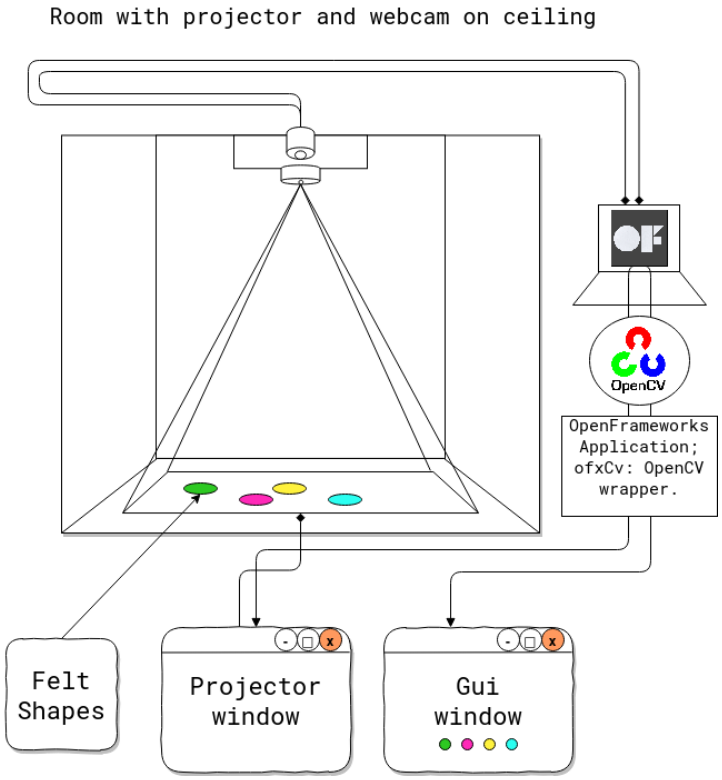


Figure 4.1: Finalised system schema

```
// 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

- `TargetColor`
- `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 for the computer vision (CV) tracking. Controls for the CV parameters are available in the **GUI** window, as handles to crop the active region of the camera frame were the CV 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 centre the rectangle with the pink circles on upper left and bottom right corners is the active detection region.

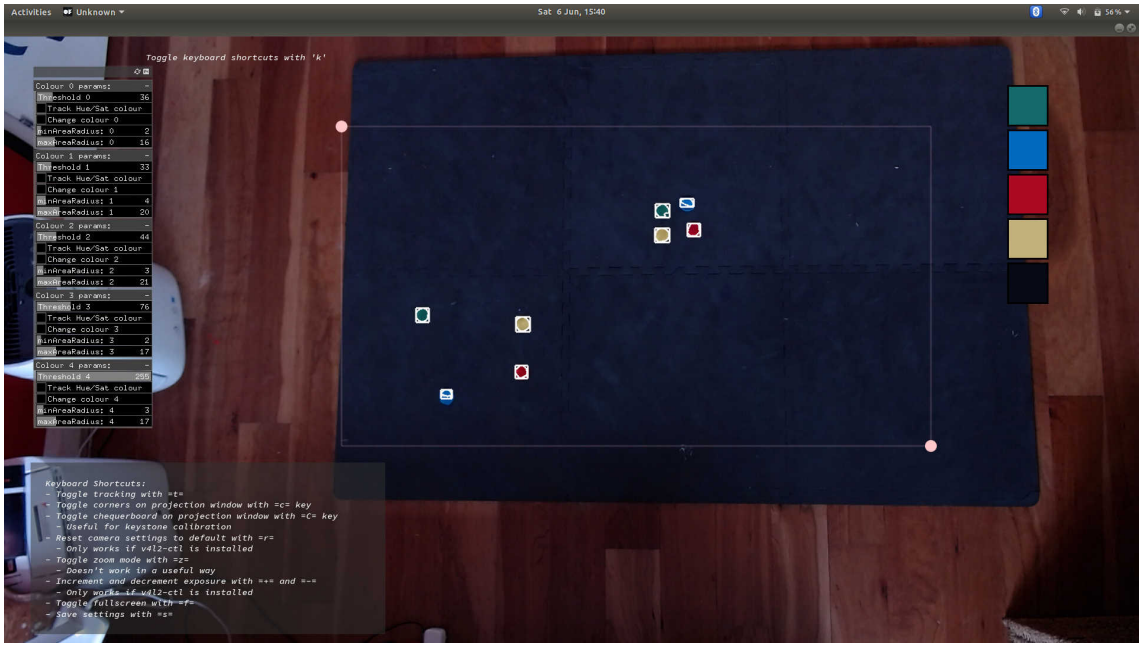


Figure 4.2: GUI window.

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 the blobs/points, of the different colours, are looped over and the location of their centres accessed. The locations are mapped to the projector window size and aspect ratio and the colour index is stored.[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.

[2]The words *blob* and *point* are used interchangeably to refer to the colour felt shapes that are used detected and located.

```
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.

An example of detection and a corresponding projection can be seen in Appendix I (Figure 8.2).

### 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. The settings are saved in a file named `settings.xml`, stored in the data folder.

### 4.2.3 GUI keyboard shortcuts

The GUI interface some other functionality that it is relevant to briefly 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`
- Toggle chequerboard on projection window with `C`
  - 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. It is split 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 can be cross over between these categories, such as with pattern and shape, where patterns can be combinations of shapes and shapes which make up patterns. 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 focus around 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 level 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 felt circles in 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 at a specific distance from the sensor.

## 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 $O(k * n^2)$, where $n$ is the number of points (blobs) and $k$ is the number of pairs [3]. See Listing. 3 for the code. [13]

## 4.5 Memory mapping prototype

A prototypical experiment using the **CL** system has also been added. This is a rudimentary interface for inputting text and assigning it to blobs in space. When the `t` key is pressed two dialogue boxes are triggered. The first takes the text you would like to assign and the second the id number of the blob you would like to attach it to. The id number is sanitised, to remove all but numeric characters and checked so that there is a blob with that id. See listing 4 to see the code. A `std::map`, from the **C++ Standard Library**, is used to associate a particular number with a string of text. This acts as a key value pair to retrieve the text for each associated number, stored in `mapi`. The maps are stored in the vector `maps`. This rudimentary prototype is a beginning for further developments of the system. See Commit 12beeff8 and video link 8.3.6 for video documentation.

[3] This may not be precise but the main takeaway is that is not scalable. It runs well 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.

```
// testing text dialog
if(key=='t'){
    // Text dialog for input text
    string out = ofSystemTextBoxDialog("Enter some text:");
    // Text dialog for blob id number
    string idstring = ofSystemTextBoxDialog("Enter blob number:");
    // Sanitize by removing anything non alphanumeric from the idstring
    idstring = std::regex_replace(idstring, std::regex(R"([\D])"), "");
    // convert to int
    int blobid=-1;
    if(!idstring.empty()) blobid = stoi(idstring);
    // If blob exists store map and id
    if(blobid<ss->blobs.size() && blobid >= 0){
      map<int, string> tmpmap;
      tmpmap[blobid] = out;
      // cout << tmpmap[blobid] << endl;
      maps.push_back(tmpmap);
      mapi.push_back(blobid);
    } else {
        ofSystemAlertDialog("Blob does not exist");
        cout << "Blob does not exist\n";
    }
}
```

Listing 4: Text input using the OF ofSystemTextBoxDialog function.

## 4.6 API

In the software outcome there is only a rudimentary "API" which is to access the colour points. It can only be accessed inside the program itself at the current time; there is no external API. 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. 5).

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

Listing 5: Accessing the the parameters for point 'n'

A simple proposed class for each blob seen in Fig. 6. Having this as a class would be useful for extensibility. It may remain a relatively simple class as other processing could be done on top of the colour point detection.

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

Listing 6: Proposed point class.

Chapter 5

# Creative process and software testing

## 5.1 Inspiration

The project has been heavily inspired by other software and research as previously acknowledged. The basic idea behind this project is to describe and implement an open-source version in *openFrameworks* (OF). The projects that inspired this one were physically unavailable; being in the US. *Paperprograms* (PP) was available to download but as described below it was not suitable for this idea. The objective was to aim for lower level architecture, in both language and theory. Create a ground system with which to build many different types of software on top of, all utilising the spatial model of interaction.

## 5.2 Paperprograms testing

Paperprograms (PP) was a starting point for testing but it was stable enough to develop on. It also suffers from being quite slow, due to the Computer Vision and graphics being done in the browser (using a version of OpenCv compiled to WebAssembly) [14]. 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 bl ob 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. It is intended, like Dynamicland as a tool for computing, but the goal of this project is to abstract the model and open it to use beyond doing computing itself. Again DL and PP also have this in spirit too but this aimed to be lower level.

## 5.3 OpenCv and Cinder

Some early testing in vanilla c++ and the OpenCv library was also done. See this link for these files. This involved using OpenCv without another framework but found OF had more available in close reach. Cinder (a similar c++ framework) was also considered but certain libraries for graphics didnt seem to be working so stuff with OF.

## 5.4 ofxPiMapper, projection mapping issue.

There is an open branch for called pimapper which is where it is intended to remerge some earlier commits. This early work was changing around the projector setup to include ofxPiMapper for doing some projection mapping. For the final outcome no projection mapping is implemented as such, other that the controls for the detection/projectoin area (See the GUI window in Fig. 4.2). This only has controls for the controlling the size and position of the active area, not the orientation or exact corners. Using the homography avaiable in ofxPiMapper would mean for more control when changing this active region as well as precise and simple mapping to it. In the current setup keystone calibration on the projector is required which works fine but can be awkward to achieve (see **??**) for this.

## 5.5 Design and development

Creative processes of this project has been goverened by building on the principles originating in the background research and the specification. It was also influenced as the project developed by technical compelxity in the given timeframe and the outcome was refactored to include this.

### 5.5.1 Prototyping

The project is itself in prototypical form. More prototyping of actuation reponses would have been useful ideation, as this has happened as the project has developed rather than in a more structured manner. Before further development further prototyping would be done, particularly of projection code.

## 5.6 Other testing

### 5.6.1 Natural light versus synthetic.

As seen in Fig. 8.1. The camea and projector were setup next to a hanging light. This was an important component for stability in tracking. At night the light is obviously needed for lighting the space, but in the daytime it is also necessary for creating a stability of light. If the natural light was used only the colour tracking would be much less stable. If a body disrupted the natural light source for a moment the tracking would struggle to pick up the

same colours after the disruption. With the hanging light turned on this was not a problem.

A future design consideration relating to tracking and stability would be to consider a sensor capable of tracking depth, such as a Kinect. This would allow for detection objects at a certain range and would mean there would be less disruption to the tracking. This was used in the MultiModal project (2.7). In this project a higher resolution camera was chosen to do the tracking without any depth sensing cabailities. This trade off could be explored in further development.

### 5.6.2 Slow algorithms

As discussed in briefly in 4.4 there is issues with the complexity of algorithm for finding pairs of points of a certain distance from eachother. This could be improved fairly quickly with further development and insight. More on algorithmic debugging and improvements are detailed in section 6.1.2.

## 5.7 Raspberry pi testing

Some testing on the raspberry pi has also been carrried out (see Fig. 8.3). This was on a Raspberry pi 4 running Raspbian and openFrameworks (armv6). There are points for accessibility here as it ran out of the box without configuration. Speed was an issue though. There was a big delay in frame-rate on the camera and the response on the projector window was lagging. An interesting experiment this shows there is a good deal of effciency improvements that could be made. It could be that it would always struggle on the relatively slow processing capability of the pi but currently, it is unusable on that platform.

Chapter 6

# Debugging and problem solving

## 6.1 Main technical issues

This chapter will deal with debugging and problem solving processes and how they were used for various technical problems faced in this project.

### 6.1.1 Projection mapping

Originally, implementing a form of projection mapping was proposed. A simplified calibration tool is seen in the **Colour Locator**. This doesn't include any advanced perspective point mapping but simply mapping the location and scale of the located points. The perspective mapping is needed if the projector is not exactly perpendicular to the ceiling. Currently **CL** relies on the projectors built in keystone mapping. The calibration process is highlighted in the video linked in Appendix III (8.3.3).[1]

The issue early on was that attempting to draw to the `ofFbo`, OF's frame buffer object would warp the contents of the frame buffer in the centre. This is because no perspective mapping is implemented in the `ofFbo` object. See image 8.4, linked in Appendix III, to see the issue. This was an interesting and confusing bug, but the combination of internal location mapping and the projector's mapping was the solution.

### 6.1.2 Algorithms and optimisation

The prototype is currently slower than it could be. Some optimisation in the algorithms will be necessary for it to be more useful and extensible. As discussed in section 4.4, it's complexity means while it may run for a few colour points it is not scalable when there are many points. "Depth first search" looks like a good candidate for use in further development.

Optimisation could also be achieved by better utilisation of C++'s low level memory management capabilities. This has been used already, for example the State class is instantiated

[1] ofxPiMapper was one solution with projective transformation algorithms. Development with this as previously mentioned could be useful as it removes the need for keystone calibration which is limited on some projectors.

as a `shared_ptr` which is passed to both the `Projector` and `ofApp` classes. This "smart pointer" means variables are not being moved around by value, but each class references the same shared instance. Further efficiencies in memory could be achieved through changing how the region of interest is cropped (this involves passing large arrays of pixel data around) and with the detection algorithms.

### 6.1.3 Lighting issues

The ambient light in the space has been a source for a few bugs. As light changes throughout the day this causes hues of the colours to change. If there is change in light, target colours that are set are no longer calibrated to the colours coming in through the camera. These issues motivated the controls on the **GUI** window. This meant that CV parameters can be adjusted and re-calibrated quickly and different combinations could be experimented with. The keyboard shortcuts that controlled the exposure and reset the camera settings were another useful part of this to quickly get the system working. An algorithm that adjusts colours every so often could be useful to find the optimum values and calibrate over time would be a useful solution. This is in the scope of future development.

### 6.1.4 Memory prototype and high level reflection

The prototype add-on (4.5) which uses text dialogues to input and assign text to a particular blob is itself a working prototype. Development of the overall system is needed for greater stability. Particularly in regards to encoding collections of *blobs* with particular identifiers. This will increase stability as currently the *blobs* can move around spontaneously as the system updates. More individualised identification and encoding is needed so that particular patterns of *blobs* are identifiable. Currently the blob which is assigned an index of `0` may be reallocated a new identifier which then disrupts the text assignment. Illustration of this bug is illustrated in video links 8.3.6 and 8.3.7.

Chapter 7

# Evaluation and conclusions

In conclusion a software prototype for spatial interaction has been presented alongside arguments in favour of this model. The main capability that it allows for, is the location of coloured objects on a two dimensional surface. With this basic tool there is extensive scope for future development. Reflections and strategies more progression are detailed in the sections below.

## 7.1 Design iteration

The design of this project has iterated as it has progressed. Chapter 5 discusses this in detail. From an evaluative perspective the creative process has lacked in certain areas and achieved in others. Apart from not quite achieving the more advanced system that was originally proposed, it has stayed true to the original aims to design and formulate a spatial computing environment. What it has lacked is a more formal process for this iteration. Thorough prototyping would have greatly benefited the project. This would have sped up the greater development process by identifying pitfalls in design that have taken time to develop.

## 7.2 Practicality of current setup

### 7.2.1 Algorithms

Efficient algorithms for identifying clusters of points are an essential for more advanced combinations of the sensory parameters. These will be needed for a more stable system. As mentioned previously (6.1) currently single blobs are can easily be disrupted by noise and interaction from the user. Algorithmically locating higher level patterns amongst the coloured blobs will solve this issue. In Paperprograms and Dynamicland sheets of paper are lined with dots and the patterns of these colours encode an indentity for each program. Similarly another algorithm could use some sort of pattern recognition to identify such encodings. When tested on the Raspberry pi there it was unusably slow (5.7) so there is plenty of room for improvement.

Envisaged in the Colour Locator different patterns of *colour points* were to represent corresponding TUI (tangible user interface)-like elements [8]. For example a combination of three red points and two green in a row instantiate a **handle class** which could be used to display a block of text. Another pattern might represent an **image class** which displays an image. This could lead to one development of the software, as a tool to create user interfaces, on top of this spatial model.

### 7.2.2 Depth camera

A depth sensor could be useful as discussed (6.1.3) for a more stable system. However, there would be a trade off between potential use and the accessibility of the system, as this would add the need for more equipment and cost. Some testing could be done with different and easily sourced sensors to final a practical solution.

## 7.3 Psychological aspects

The psychological aspects that were originally discussed and speculated on in Chapter 2, have not been explored in detail. Apart from the overall design being aimed at the psychological modes of interaction, this has been left out of scope due to a focus on software engineering. Memory is one area that this software could work well with. Due to our 3 dimensional lives, memory is something that could benefit from interacting with a computer around a room as opposed to a 2 dimensional monitor. While obviously not beneficial for all tasks there is room for enquiry and application in this regard.

In the original aims there was some hope to build some simple tools with CL. One idea was for a tool for improving memory, taking inspiration from Anki, a program for spaced-repetition of flash cards. A couple of sources purport good results for increasing memory capacity [15, 5] and it seemed a good candidate translating into CLs model.

## 7.4 Secondary outcome

The theoretical outcome (4.3) was a brief addition to the software **Colour Locator** (CL). Intended as a formal conceptual summary of the design and developmental aims of CL it aims for to better describe the model, for the discussion of it's potential use cases. It also offers a description of the problems this project aims to address; attempting to add a more robust specification for development of the prototype.

## 7.5 Social aspects. Proposed social evaluation

Originally, user surveys were to be used in evaluation. This would involve survey analysis and conversation analysis [12] to assess various aspects of the system. This would be in regard to usability, and any advantages and difficulties with use. This was not carried out as CL will only be of interest to developers in its current state. No user testing in person has

been tried out due to not being able to get appropriate users to physically use the work and also it's incomplete nature. A survey of proposed survey questions are listed below. These aim to gauge the potential users interest in the use of such software and thereby areas for system improvement.

### 7.5.1 Proposed survey questions for potential developers

- Could you envisage integrating this interaction model into your workflow?
- Are you interested in open-source software development?

### 7.5.2 Proposed survey questions for general users

- Could you envisage integrating this interaction model into your current usage of computers?
- Do you feel like you could have a healthier or more productive relationship with your computer (Desktop or laptop)?
- Do you feel that this would help you have a healthier relationships when using your computer? {Rate on a scale from 1(not at all useful) to 5(very useful)}

## 7.6 Durability

The implementation has been tested, identifying different points for improvement and progression of the software. As the system progresses beyond the scope of this report, more bugs will probably be found. Overall the prototypical nature of the software means that improvement is inevitable. The aim for creating a system to improve on the stability and speed of *Paperprograms* was not achieved. This is one the major failings of the project in that it is not offering, in it's current state, anything directly usable to in this area of interaction-design.

## 7.7 Final

Here is a concluding statement regarding the project overall. Areas for improvement and design progression have been specified alongside assessment of the achievements and bugs. I've learnt some useful lessons on design and how to enhance the creative processas well as gained technical insight into software engineering, the C++ language.

Chapter 8

# Appendices

Here are links to additional images, screenshots, code repository links and video documentation.

## 8.1 Appendix I: Additional images


Figure 8.1: Camera and projector secured on ceiling.

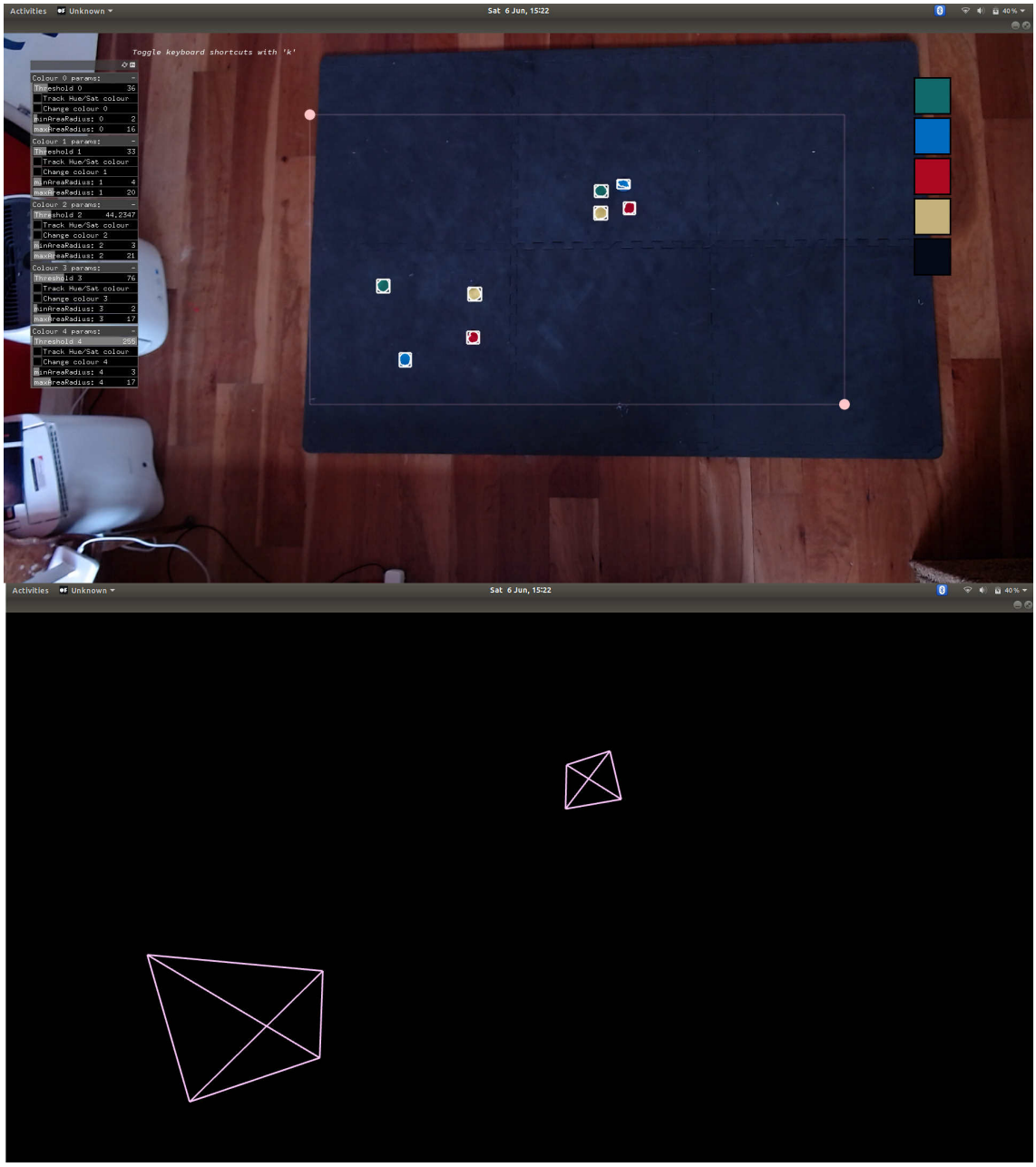


Figure 8.2: Detection and Corresponding projection.

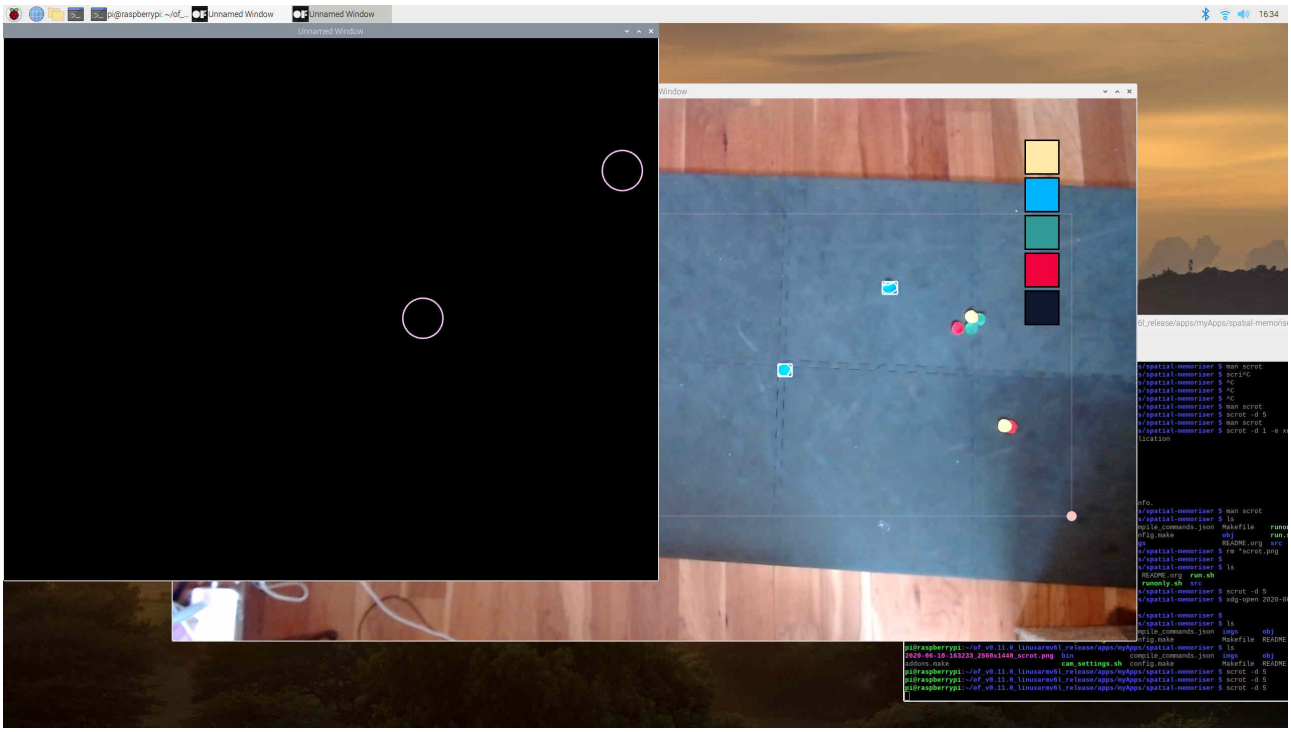

Figure 8.3: Testing on Raspberry Pi 4.

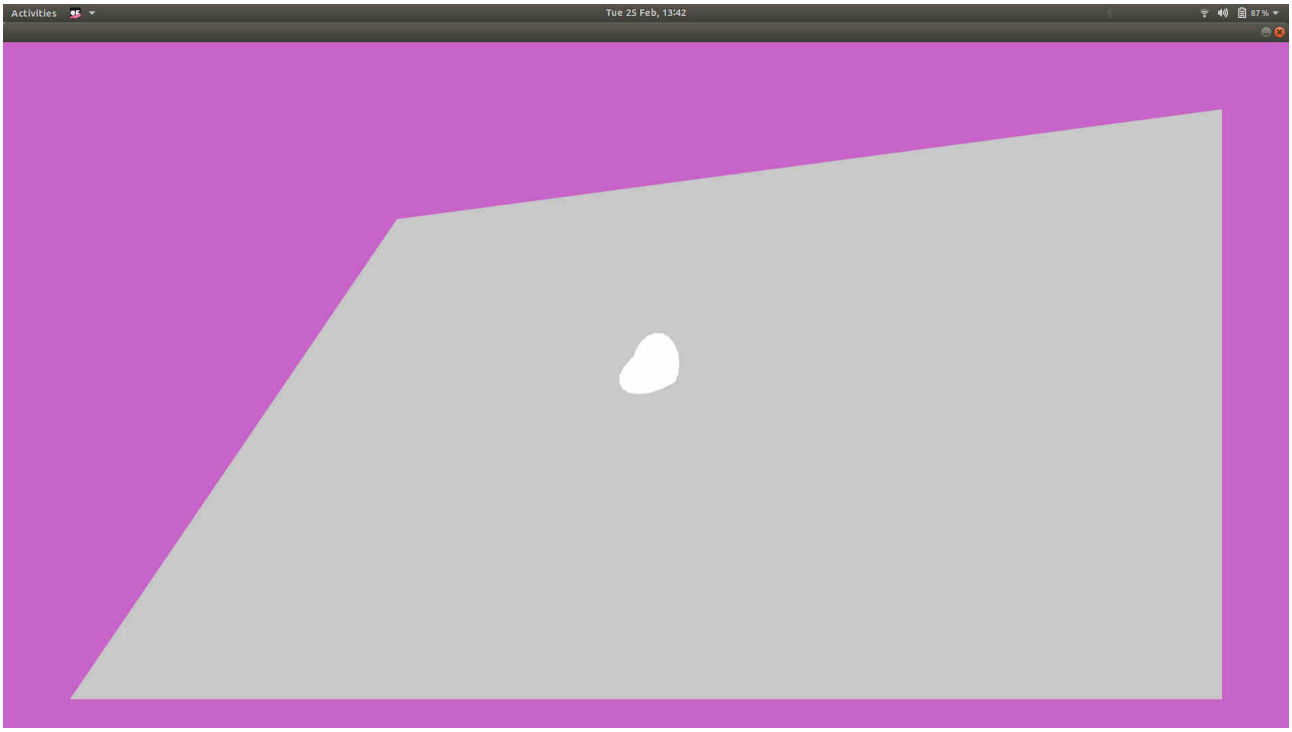


Figure 8.4: Projection mapping bug.

## 8.2 Appendix II: Code repository links

- Code repository on Gitlab.
- Code repository on Github.

## 8.3 Appendix III: Links to video documentation

### 8.3.1 Full video link

The link to the full video containing all the sections below.

### 8.3.2 Basic Colour blob tracking

Demonstration of the basic colour tracking on early prototype version.

### 8.3.3 Keystone calibration

Demonstrates keystone calibration and aspect and location mapping in **CL**.

### 8.3.4 Target colour adjustment

Adjustment of the target colours so tracking works for each

### 8.3.5 User testing

Some rudimentary user testing

### 8.3.6 Text input prototype

Demonstration of the text input prototype

### 8.3.7 General use

Some major bugs visible

# 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] T. Cormen, *Introduction to algorithms*. 2009.

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

[15] A. E. S. Hanson and C. M. Brown, "Enhancing l2 learning through a mobile assisted spaced-repetition tool: an effective but bitter pill?," *Computer Assisted Language Learning*, vol. 33, no. 1-2, pp. 133–155, 2020.