Flow: A Framework for Reality-Based Interfaces

by

Robert Michael Hemsley

BSc Computer Science with Distributed and Mobile Systems
Cardiff University, 2010

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfilment of the requirements of the degree of

MASTER OF SCIENCE IN MEDIA ARTS AND SCIENCES
at the **MASSACHUSETTS INSTITUTE OF TECHNOLOGY**

February 2014

Signature of Author Signature redacted

Robert Hemsley
Program in Media Arts and Sciences
September 20, 2013

Certified by Signature redacted

Henry Holtzman
Research Scientist, Chief Knowledge Officer

Accepted by Signature redacted

Pattie Maes
Associate Academic Head
Program in Media Arts and Sciences, MIT

Flow: A Framework for Reality-Based Interfaces

by

Robert Michael Hemsley

BSc Computer Science with Distributed and Mobile Systems
Cardiff University, 2010

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning on August 09, 2013,
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE IN MEDIA ARTS AND SCIENCES

ABSTRACT

This thesis outlines the design, development and evaluation of a computer vision framework known as Flow. Flow utilises a wearable head mounted camera to observe user interactions, providing recognition of objects, locations, people, and gestures via machine vision and understanding. This supports the creation of a hybrid interaction environment, allowing users to access digital functionality and actions based on their interactions within the physical environment. Flow highlights the potential of cross device, environment interactions by providing the architecture to connect between existing platforms and devices. Example applications scenarios are provided to evaluate the framework. Each demonstrates the versatility of the system in supporting human computer interaction and prototyping of ubiquitous systems. Insight through user feedback and a technical evaluation is also provided. The work outlined in this thesis demonstrates the potential of passive observational computer vision systems in the augmentation of user interactions and the potential for further dissolved computing environments.

Thesis supervisor: Henry Holtzman
Research Scientist, Chief Knowledge Offer

Flow: A Framework for Reality-Based Interfaces

by

Robert Michael Hemsley

The following people served as readers for this thesis:

Signature redacted

Thesis Reader

V. Michael Bove, Jr.
Principal Research Scientist
Program in Media Arts and Sciences, MIT

Signature redacted

Thesis Reader

Joseph A. Paradiso
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences, MIT

ACKNOWLEDGEMENTS

I would like to thank my advisor, Henry Holtzman for his continued encouragement and guidance throughout my time at the lab. I would also like to thank Mike Bove and Joe Paradiso for their continued patience and feedback as readers.

Thanks to Matthew Hirsch, Arlene Ducao, Dhairya Dand, Pip Mothersill and the rest of the Information Ecology research group for their input and inspiration throughout.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

"Humans speak, gesture, and use writing utensils to communicate with other humans and alter physical artifacts. These natural actions can and should be used as explicit or implicit input."

- GREGORY D. ABOWD [1]

Interactions with computers are no longer constrained to the traditional desktop environment of a monitor, keyboard, and mouse. The world is now saturated in computation, from touchscreen smart phones and tablets within users' pockets to embedded systems in homes and walls, each offering further touch points to access this digital world.

While the benefits of these individual devices are clear, their interfaces and collective interactions remain less well understood. In the digitisation of the world and the development of consumer electronics, there has been a focus on pure functionality, viewing each device as part of a standalone interaction. This focus has ignored the continuous, cross-device, and environmental nature of user interactions. Within this siloed world, users have grown accustomed to acting as an interpreter, bridging actions and information between devices. Rather than continuing this journey of isolated interactions, systems must be developed with the awareness of the user throughout their interactions, working across both digital devices and displays and the traditional non-digital environment. Through this process, natural interactions are enabled based on physical movements and allow common interfaces to seamlessly transition between our technology.

This environment of pervasive technology was predicted by Wesier[57] when he spoke of the pads, tabs, and boards that would provide computation across scales. In this vision of ubiquitous computing, the complexity that these ubiquitous devices have created was addressed through the dissolution of technology into the background and the proactive, knowledgeable nature of the connected devices and applications. Reflecting on this original vision, it is clear that the devices are available and the connectivity and infrastructure

present, but we still have yet to connect these and create the wrapper to provide dissolved interactions between device and environment.

Focusing on these devices, it is apparent that their approach to viewing and manipulating data has remained the same, taking a screen-centric form. This has created an environment where user interactions are channelled through these glass displays in a unnatural manner, through WIMP or touch-based interaction metaphors. This environment has visibly altered our behaviour, where we now graze between digital touch points, constantly viewing and manipulating this flat digitized world. This behaviour is acknowledged to be less than optimal, as it demands the user's full attention [62]. Much like eyeglasses, tools should be provided to focus the user on the task and not the tool itself.

So, how do we draw the user's attention away from this, to engage back into our reality while still remaining connected? Research efforts have attempted to address this by taking inspiration from the physical domain, using observations from our objects' affordances and the physical manipulation with which we have grown accustomed. This has produced new human computer interaction (HCI) fields such as Tangible User Interfaces (TUI) and Natural/Reality-Based Interfaces (RBI). With TUI, the focus is on the physical manipulation, creating new objects from which the digital environment can be affected and controlled through our inherent knowledge of physical manipulation. RBI research closely aligns with the vision of a dissolved computing environment where consideration for the physical user, existing interaction models, and their environment and position within it are used as guiding principles. This approach attempts to use the natural forms of interaction to which humans are accustomed, such as writing, speaking, and gesturing, as input. Through this, input interfaces are dissolved by matching them to our existing, expected models of interaction.

As we search for a more even balance between man and machine, we must return to the physical domain, bringing digital functionality and awareness with us. In this manner, we can reverse the current trend of skeuomorphism and the digital, visual representation of our world, instead augmenting the physical with digital actions cued by our natural behaviour. Similarly, in the

opposite direction, we must continue to improve the machine's understanding of our world, allowing our devices to react and provide interaction in a more subtle way. By attempting to bring together our digital devices and provide an awareness of our physical interactions, we can envision a future of seamless multi-device environment interaction.

This thesis therefore describes the development and testing of Flow, a computer vision system for wearable cameras (Figure 1.1) that attempts to address these ubiquitous computing challenges. Flow aims to provide a framework within which reality-based interfaces and applications can be developed, which would enable users to control and interact with their objects and devices across both digital and physical environments. This work considers the interaction possibilities enabled through the use of a wearable head-mounted camera and the passive observation of the user's interactions within the environment. In this work, a number of potential applications are considered, but the intention of Flow is to provide a research framework from which further dissolved computing environments may be explored.

Figure 1.1 Flow – A Framework for Reality-Based Interfaces

Flow approaches the goal of supporting reality-based interfaces in a number of ways. Firstly, Flow attempts to **move digital interactions away from pure display-centric forms, instead bringing digital information and functionality into the physical space around the user's existing objects and interactions.**

This is demonstrated through just-in-time information retrieval and the remote access of object functionality, as cued and executed by the user's interaction. In this, Flow generates contextual information, allowing devices to proactively gather and display data as cued by their real-world interactions. Secondly, Flow provides a **unified interface and control mechanism for multi-device interactions**. By remaining platform-independent, Flow operates between devices, enabling users to move freely between systems while maintaining a common interface and interaction technique throughout. The goal of this is to create an environment where users can reuse any device to control and access information relevant to their needs. This helps bridge the current void between our devices, viewing them as a single continuous system with which users can engage and utilise as needed. By basing the system on user video, **improved machine understanding of the observed interactions and environment** can be achieved. Through this contextual awareness and localisation, such understanding is possible for both the user and the objects with which they interact. This awareness enables user preferences and settings to follow them, proactively adjusting device behaviour at the point of interaction. Through this, we can **augment existing behaviour, creating user-defined digital rules** for our world. This addresses the current inflexibility of manufacturer-imposed interaction methods and allows users to assign new behaviours or actions to objects and remote services. These actions can be assigned to identified behaviour, such as virtual button presses, or be accessed through the connected multi-device environment Flow provides.

Within the following chapters, I present a selection of related work before completing 4 design explorations into the creation of reality-based interfaces. These explorations were undertaken during my studies and form a basis for the final development of Flow. The design and implementation of this framework is later described, demonstrated and tested through the creation of accompanying applications.

RELATED WORK

This thesis contributes to an array of projects that focus on cross-environment digital/physical interactions and the dissolving of technology and digital devices. Having this rich basis of prior work, many of the architectural approaches and design principle's for user interaction have been reimagined and utilised within this framework. As such, I shall review these prior contributions and further highlight how they have shaped this development.

With any system designed for ubiquitous digital/physical interaction, the fundamental question arises of how these physical environments should sense and become identifiable in a digital context. The most common approach, often utilised within the domain of the "Internet of things", is to augment our objects and environment with intelligence in the form of hardware that locally identifies and senses the environment. This hardware often senses, processes, and wirelessly communicates information about the environment and the object's interactions. In this manner, the intelligence is placed within the object it self, autonomously sensing and communicating data for analysis. This approach often creates an accurate model of the environment through this dedicated infrastructure, but in doing so creates an overhead, as objects must be physically augmented. This approach presents difficulties for scaling across our environment and requires further technical consideration for supporting the discovery and communication processes.

The alternative approach is to move the intelligence away from the object and environment and out onto the users themselves. In this manner, it is at the point of user interaction that the environment and objects are sensed. This person-centric approach to sensing is often explored through wearable technology, with devices such as RFID reading rings[26] or wearable cameras being used as input. By taking advantage of recent computer vision techniques, we can start to interpret the environment and identify user interactions without requiring the environment be augmented with additional hardware. It is this approach to sensing that shall be explored within this thesis. My contribution within this field of work is, therefore, to provide a framework that focuses on our everyday environment and to provide a richer

machine understanding of our interactions in terms of objects, people, and spaces. Through this, researchers can further explore how we can create cross-device and environment interactions utilising the metadata this system provides.

REALITY BASED INTERFACES

The field of reality-based interfaces (RBI), or natural user interfaces, approaches interaction design based on our observed interactions within the physical world, and attempts to transition current digital functionality into it rather than moving the physical into digital representations. Rather than taking cues from our lives and digitising them as icons, this field uses technology to build upon reality, ensuring that users remain grounded in the physical environment and not the displays or devices around them.

RBI systems are categorised through their use of naïve physics, body and environment awareness, and the user's skills. By taking a user-centred approach and inspiration from our existing interactions, RBI looks at dissolving technology and creating more adaptive, considerate systems.

An example of a reality based interface is the proverbial wallets[28] project, which attempts to bring remote financial information back into the user's wallet. In this project, the user's remote online bank balance is communicated through their physical wallet, using existing mental models of resistance, vibration, and physical size to communicate the relative account balance. This illustrates an RBI system due to it's use of naïve physics for feedback and its awareness of the user's body and environment within the interaction. As shown in figure 2.1, a resistive hinge is used to provide feedback on the user's balance, where increased resistance is mapped to a diminishing bank balance.

Figure 2.1 Proverbial Wallet – Tangible financial sense

Another example of a reality-based interface is the Digital Desk[58] project, which aims to bring digital functionality and interactions to the non-digital paper environment. Rather than bringing note-taking into a digital form where our existing skills of physical manipulation are ignored, the digital desk looks at augmenting existing reality with digital elements.

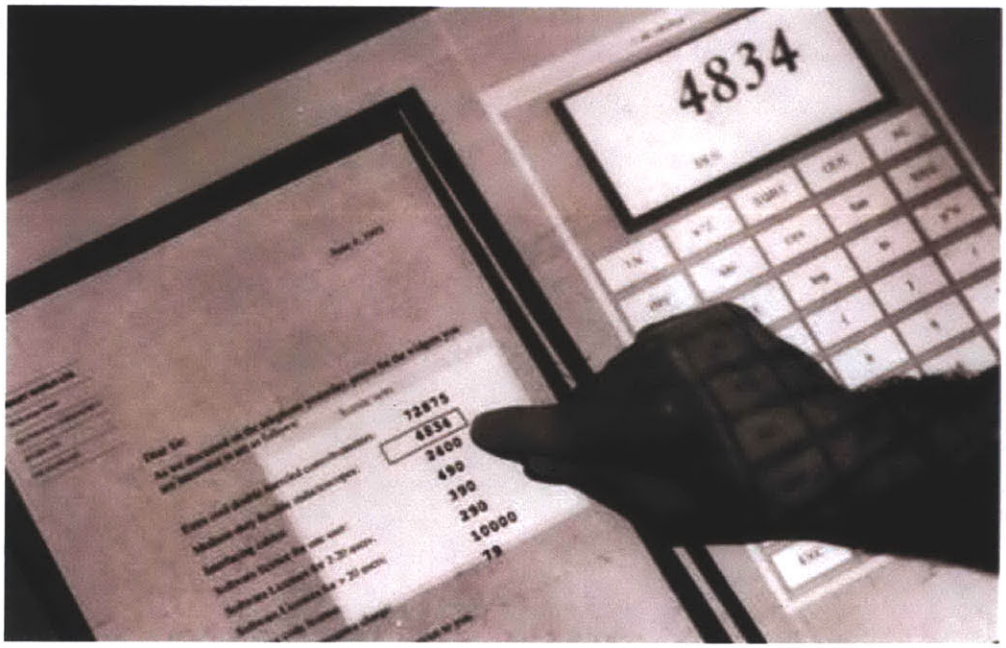

Figure 2.2 Digital Desk – Reality Based Interface

By using a camera and projector positioned above the work bench, the system tracks paper that is being interacted with and automatically projects digital functionality onto it, as illustrated in figure 2.2. Through this, an environment is developed where digital metaphors of copying and pasting can be directly used from the physical desks. Common tasks, such as undertaking calculations based on written notes, is also afforded in this system automatically, creating a spreadsheet environment based on the document.

These systems clearly demonstrate how natural interactions can be developed between our environments, with digital information being made available around existing objects and interactions. Systems such as the Digital Desk are confined to specific locations due to their projector and camera setup, and require the use of tractable, visually-tagged objects within the interaction. This approach to augmentation should be made available throughout our environment, and so it will be further explored in the framework developed.

AUGMENTED & MEDIATED REALITY

The field of augmented and mediated reality are highly relevant to this project, as they focus on the use of computer vision techniques to enhance our environment through rendered visual overlays.

The eye-tap[36] systems developed by Steve Mann provided sentinel work within the field of mixed/mediated reality, creating a basis for many of the emerging wearable devices currently in development, such as Google Glass[20]. These systems focused on bringing the computer as close to our environment as possible through a wearable head-mounted camera and display, as shown in figure 2.3.

Figure 2.3 Steve Mann wearing his EyeTap device, Augmented & Diminished reality

The system uses a head-mounted camera as input, viewing the environment from the user's perspective and processing it to make visual augmentation possible through the head-mounted display. Through this, various augmented and diminished reality applications are supported, such as personalised overlays for street signage, as shown in figure 2.3, or heads-up information and statistics based on the environment, such as during a sports event.

This system hints at the interaction possibilities enabled by exposing and analysing our environment from a head-mounted camera. In this example, the input is used to create a private environment of purely virtual elements, augmenting only the visual senses for a single point of view.

A further example of augmented reality used within real-world interactions is the NaviCam[43] system by Rekimoto et al. This project used a palmtop TV and CCD camera to view the user's environment and determine the location's context based on marked objects within the environment.

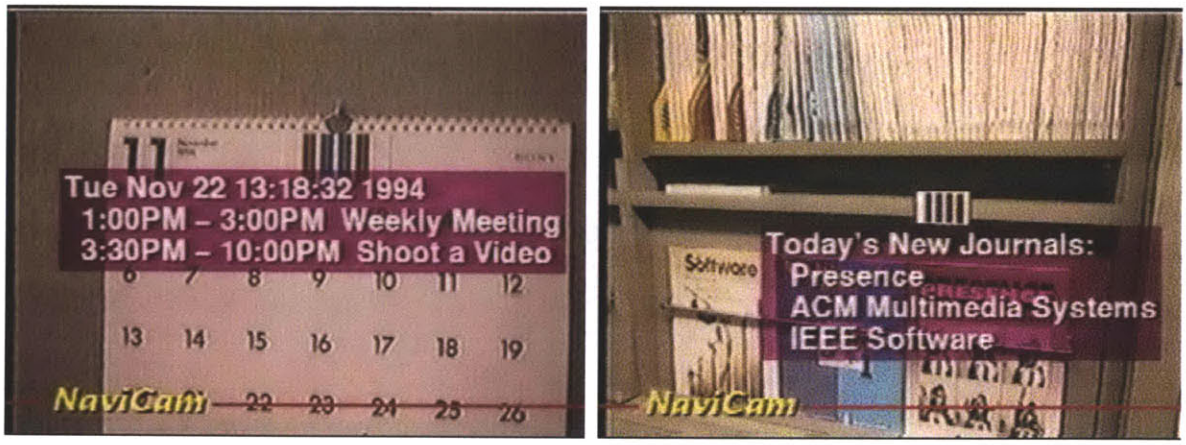

Figure 2.4 NaviCam Augmented Reality Objects

As shown in figure 2.4, visual colour codes are added to the objects, enabling detection within the video frames. Each identified object is then referred to further information such as the user's calendar, which is superimposed on the output video feed. This provides contextual information within the user's environment, which is activated by cues from the physical world. In the figure, we see the user's personal calendar appointments being overlaid onto the environment as cued by the presence of the physical object.

These systems focus on the augmentation of our visual senses by providing a rendered display through which the world is viewed. These augmentations are purely visual in nature, only effecting a single user's viewpoint through the head-mounted display. This approach is one way, with feedback only being exhibited visually over the physical environment with it. I wish to further explore this shortcoming within this thesis, and to demonstrate how our connected devices can be repurposed for these display purposes.

PHYSICAL MOBILE INTERACTION

The research field of Physical Mobile Interaction (PMI), as coined by Kindberg [29], focuses on the concept of using our mobile devices and their sensing and display capabilities to interact with and control the world around us. Kindberg highlighted how our mobile devices move with us between places, people, and objects, allowing them to encounter as much of the world as we do. PMI therefore looks at how this ever-present companion device can be used to create new forms of interaction between our environment and devices, providing control and insight into the user's lives.

An example of PMI is the Collect & Drop[8] project, which uses mobile devices to interact with nondigital objects such as printed posters. Through this, the user can interact with the object physically through a series of pointing gestures, with their mobile device interpreting and presenting a final action based on this.

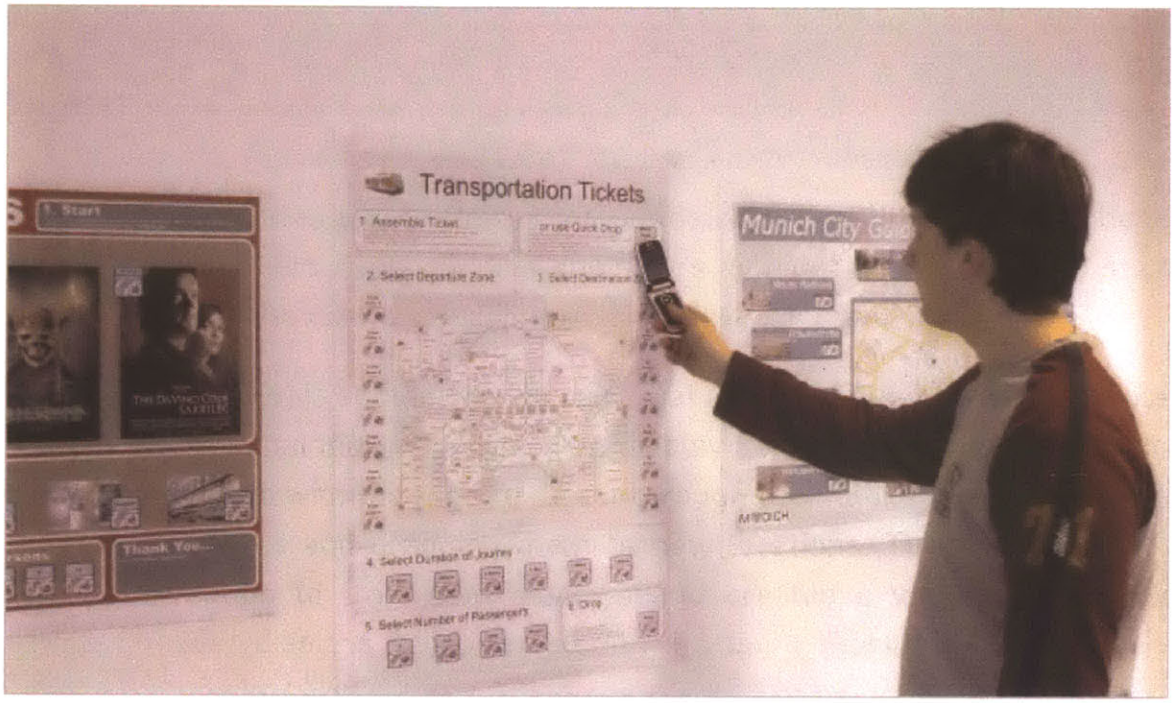

Figure 2.5 Collect&Drop – Physical Mobile Interaction

Figure 2.5 illustrates a user making an online booking by pointing at options on a physical printed display. In this, the user's mobile device forms a pointer, which identifies the user's selections and generates a travel itinerary.

Touch Projector [6] is another PMI project that uses computer vision to enable the remote manipulation of graphics on display devices. In this environment, the user interacts with the remote object through the mobile device's display, selecting graphics and dragging them to new locations. This brings touch-based interaction to traditional displays and creates a virtual continuous canvas between displays that is revealed by the user's mobile device.

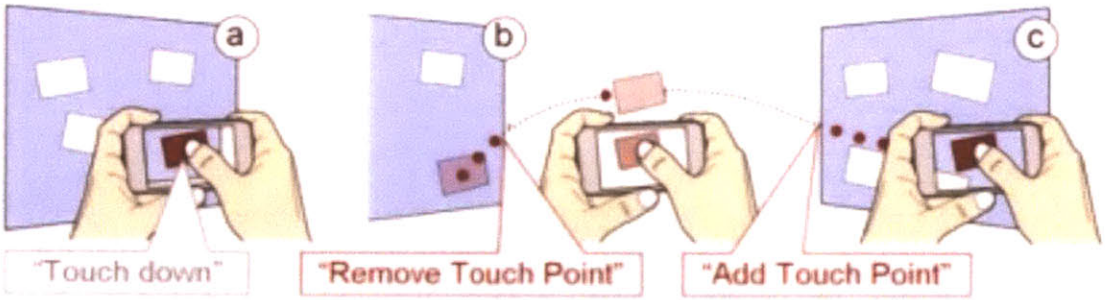

Figure 2.6 Touch Projector – Remote manipulation of projected graphics

Figure 2.6 shows this interaction, virtually dragging contents from a display across a physical wall and onto another output device.

Although interaction takes place through the mobile device's camera viewpoint, it does not provide an augmented reality view. Rather, the input is sent to the real world display, which updates the visual appearance. This interaction rationale of staying grounded in reality is highly desirable and will be pusued within Flow.

JUST-IN-TIME INFORMATION SYSTEMS

Another aspect of Flow is its ability to identify objects within our environment and access related information, just in the instant of interaction. This approach to information retrieval is commonly referred to as just-in-time information systems [45]. One example of prior work within this domain is the ReachMedia[14] project at MIT Media Lab. The system, as shown in figure 2.7, uses a wearable RFID reader to identify objects and gather associated object-specific information, presenting this back to the user via audio feedback.

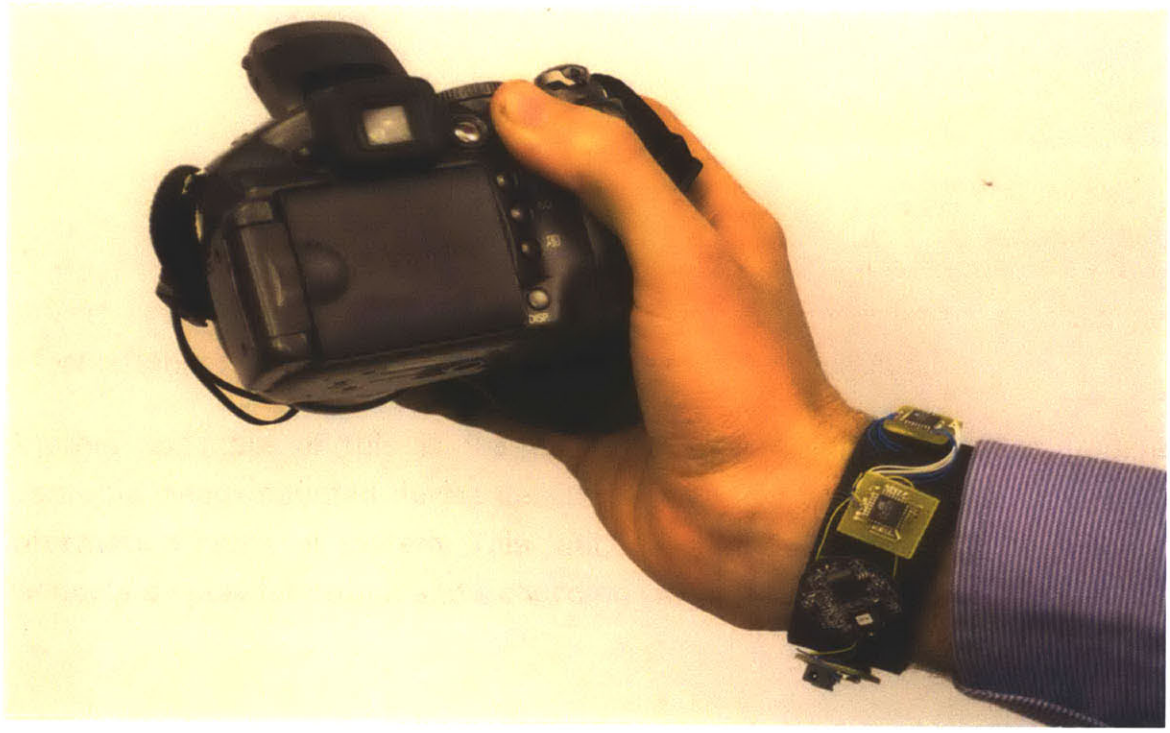

Figure 2.7 ReachMedia – RFID based just-in-time system

By augmenting our objects within RFID tags, the system demonstrates how objects can be sensed and connected to further, richer sources of information, presenting this in a hands-free manner via audio.

Another example of this is the Remembrance Agent by Rhodes [44], a wearable head-mounted device designed to be a proactive memory aid and information retrieval system. This unit, as shown in figure 2.8, provides a monocle display for output and a chording single-handed keyboard for input.

Figure 2.8 Bradley Rhodes head mounted Remembrance Agent

Using this, the user can input text into the system and automatically associate it to their current location, date, and time, making it indexed for future retrieval. As the user enters further notes, the system proactively retrieves related messages based on these vectors.

These systems demonstrate how contextual metadata can be associated with our actions and environment and be rapidly accessed in situ.

EXPLORATIONS

The following 4 explorations focus on the creation of hardware and software that enable multi-device interactions, as well as the mediation of information and functionality between the digital and physical environments. The first three projects explore this through the creation of Tangible User Interfaces [55] (TUI), and the later through Computer Vision, with mobile devices.

DROPLET

To explore the concepts of information visualisation, manipulation, and transportation between display devices, a project exploration called Droplet was undertaken.

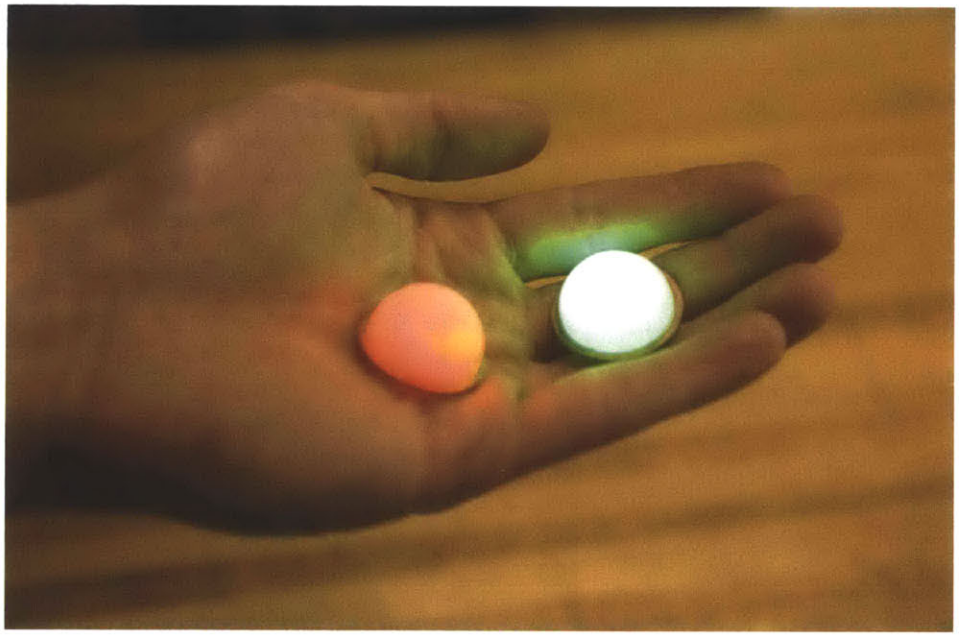

Figure 2.1 Droplet – Information Visualisation

Droplet, as shown in Figure 2.1, enables device-to-device communication through a physical hardware interface, utilising light and capacitive-based communication techniques for two-way data transfer. This approach gives any screen-based device with an appropriate refresh rate the ability to transmit data directly to a Droplet, allowing it to be extracted, stored, and visualised in a physical form. As a tangible object, Droplet enables the use of physical motor skills and familiar interaction techniques to pull information from any

display by simply placing the object over the required data. As a tangible object, this handheld device can represent the contained data as a standalone ambient display. When later placed on a capacitive touchscreen device, Droplet changes form, supplementing the ambient light output with the display capabilities of the device it is placed upon. Each Droplet can be tracked across the surface and transmit data to the device, allowing the display to visualise and provide insight on the information it contains. These tangible objects offer a low-cost physical token for information, which can be socially shared and distributed.

INTRODUCTION

Within our lives, we have become accustomed to the ubiquitous nature of our digital devices and their displays, adapting to accept the flat interface and its standard presentation of information. These interfaces and their approach to interaction ignore the finely-developed motor skills and our inherent understanding of our world around us with the intuition and naïve physics we have developed for it. This project explores how we can start to interact with our devices in a physical manner, and how mixed-reality interfaces can be developed for our physical objects without the use of AR overlays.

Rather than requiring the high cognitive loads observed with text or pictorial-based representations, ambient displays have been shown [22] to require less overhead due to their peripheral, glanceable form. By abstracting information and presenting it as light, we reduce its complexity and allow users' existing mental models to map and interpret the meaning of the data. Droplet achieves this by mapping colour and light intensity to the information the objects contained. It has been shown that the over-abstraction of information within these displays can lead to misinterpretation [35], and so the ability to appropriately abstract and transition between multi-level information must be addressed. To do this, Droplet explores the reuse of existing devices and their affordances for communicating multi-level information.

By combining this tangible object with existing display hardware, we are able to demonstrate how contextually aware interfaces can be developed around physical hardware.

SYSTEM OVERVIEW

The device takes the form of a small 30mm domed object that contains a low-cost ATtiny 45 microcontroller [40], RGB Led, Opto-relay, and phototransistor. These components are powered by a single coin cell battery, with the microcontroller operating in deep sleep mode, periodically checking for light level changes every 8 seconds based on the watchdog timer interval [40].

Encoding

To reduce the number of components, Droplet is self-clocking, encoding the signal within the transmitted data, as outlined in the Manchester encoding scheme [16]. This isochronous approach has a time period of 60ms, which provides a data rate of just over 2.0 bytes per second. This is relatively low, due to the refresh rate and rendering engines of the displays the Droplet device is placed upon. This approach to communication has similarly been explored by Vu [13] for wearable device authentication. To ensure the system is as cross-compatible as possible, the encoder takes the form of a web-based implementation written in HTML and JavaScript. Due to the range of HTML renders and the current lack of support for CSS 3 transforms, this time period has been selected as the lowest common denominator between the available hardware.

To extract information from a display to a Droplet, the user holds the object against the display and activates a bookmarklet, which sends a series of light pulses from the display to the device. These rapid light changes are detected by a Photodiode and decoded to provide raw binary data, which is interpreted and stored on the device. This collection process is illustrated in figure 2.2.

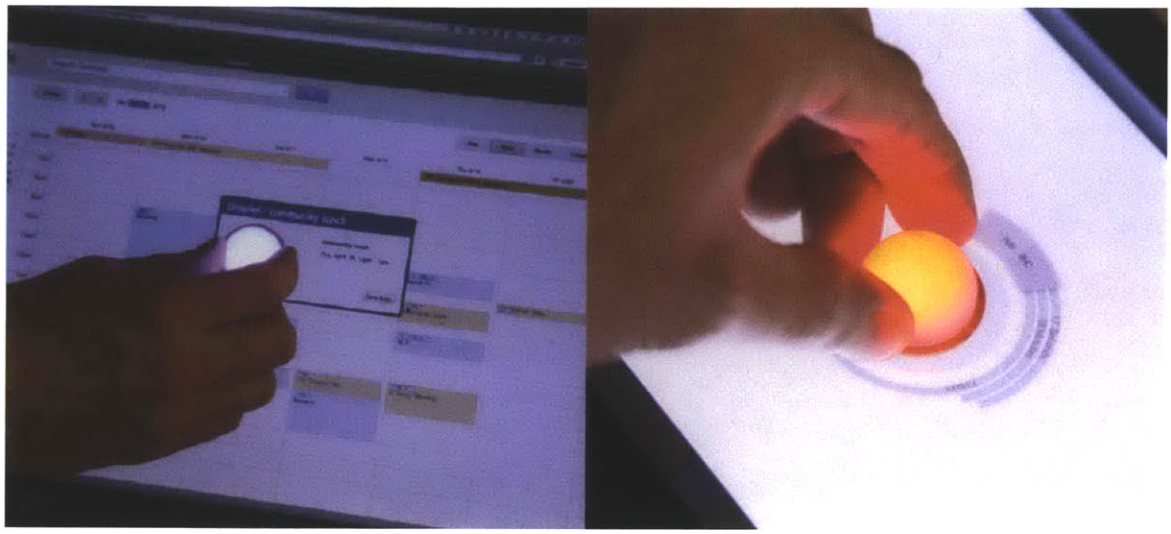

Figure 2.2 Extracting calendar information (Light based communication) & visualising on tablet device (Capacitive based communication)

The Droplet interprets both shortened URLs and Timestamps, which are visualised as a function of colour on the internal RGB LED. For timestamps, the LED changes colour from green to red, increasing the frequency of flashes as a function of the remaining time. For URLs, the colour represents changes to the stored website and is updated when the object is placed on the display, allowing it to be dereferenced, checked, and synchronised.

Through this, the Droplet provides an ambient display that engages the user's peripheral vision, enabling awareness of the information it contains. In figure 2.2, we see the Droplet being used as a countdown reminder to a specified event.

The Droplet's orientation and optional ID is detected based the spatial distribution of the touch points at its base. Figure 2.3 shows a Droplet orientation test using these 3 detected touch points to calculate the rotation and location of the object on the display. Without active capacitive communication, the object can be identified based on the relative length between these touch points.

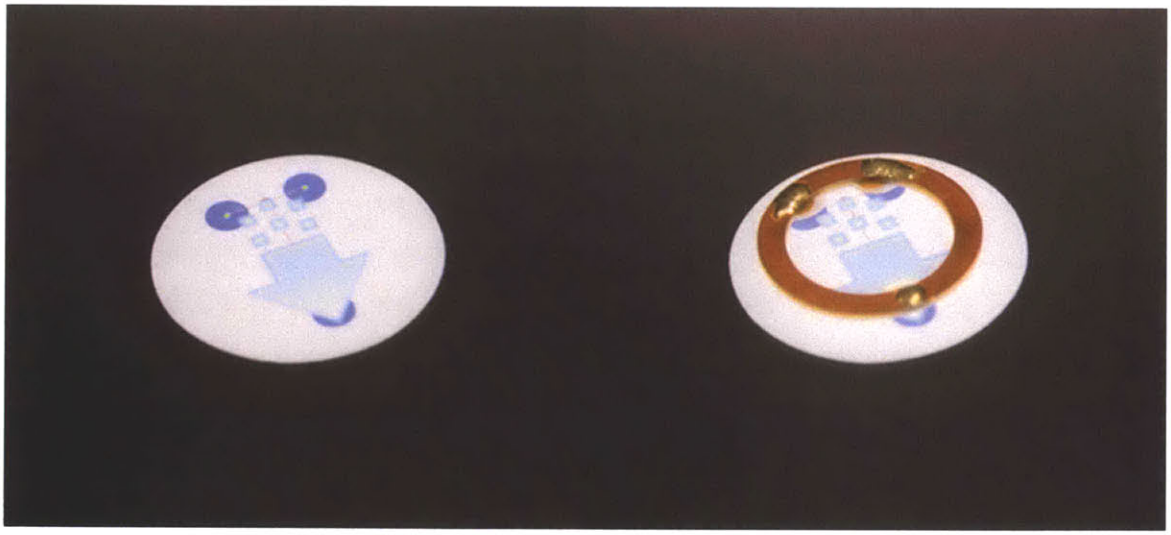

Figure 2.3 Testing object position and rotation based on touch points

VISUALISATION AND INTERACTION

When placed on the display, the object is augmented with user interfaces graphically drawn around the object. Figure 2.2 shows the Droplet providing a radial clock and textual description for the contained event. As a physical object, users can grasp the device, interacting with it as a widget to control the contained data. In the example of the timer, this provides a rotary dial to view and adjust the contained time. By rotating the object, the user scrubs through time, updating the events timestamp. In this manner, we are combining the physical affordances of the object with the software-defined display, thus creating virtual interfaces for the physical object.

This demonstrates the potential of bringing user-definable functionality to our hardware objects, allowing software controls and interfaces to be built around the device. Through this, we can reuse the capabilities of our existing hardware and start to design objects based on their physical affordances rather than the need to provide low-cost forms of input through LCD panels and soft buttons.

DISCUSSION

Droplet was an initial exploration into extracting information from the ubiquitous display in a cheap, direct, and cross-device manner, exploring how this data might be visualised and represented. The project has been

presented in the press and during open houses over the last year, and has been informally operated by around 50 users. The project was well-received, with the concept of device reuse and reprogrammable functionality being the most commented upon. Inter-device interaction still remains minimal, and so, in this application, the simplicity of physically grabbing and moving data between devices was well-received. I have continued to explore the concept of device reuse and extensible controls for physical objects, and have expanded it within later works.

STACKAR

To further explore the concept of software controls augmenting our physical objects, a design exploration was undertaken building a virtual breadboarding environment named StackAR.

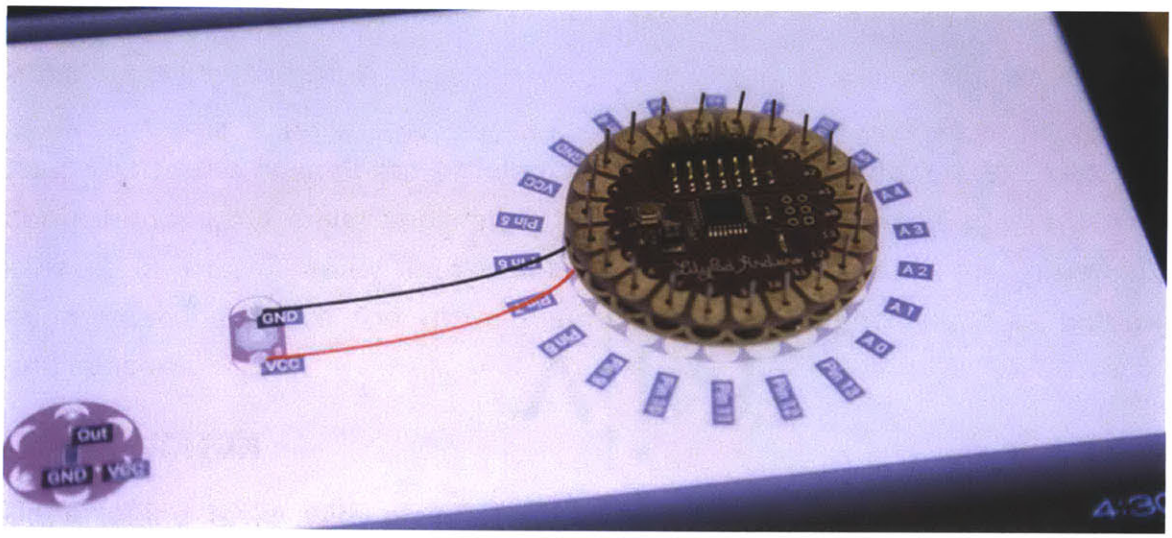

Figure 2.4 StackAR virtual breadboard with software defined components

StackAR takes the form of a shield for the Lilypad Arduino [9], an existing hardware prototyping platform for novice electronic users. The shield connects to the base of the Arduino, allowing it to be identified, tracked, and controlled from a tablet application through a visual software interface. This creates a mixed-reality interaction environment between the physically augmented hardware and virtually simulated components, such as buttons and sensors.

INTRODUCTION

Undertaking tasks with physical objects based on textual instructions can often be challenging, due to the possible misinterpretation created when visualising and mapping them to the object. In these environments, direct instruction and guidance around the object would be a more desirable approach to learning, as information can be provided in situ, just in time with the context of the user's interactions.

The StackAR project addresses this problem in the context of learning

electronics, by providing a visual digital interface for physical hardware. Through this, tutorial-type instructions such as PIN numbers and connection types can be displayed directly around the object. In this manner, users are shown in situ how to wire components, rather than risking potential misinterpretation through a written form.

SYSTEM OVERVIEW

The system itself takes the form of a hardware shield, which connects to the base of a Lilypad Arduino. This attachment mediates the communication between the Lilypad microcontroller and the tablet, allowing it to transmit and receive data using capacitive and light-based communication, as demonstrated by Droplet.

Rather than providing a StackAR bootloader, the Lilypad Arduino uses the firmata library [53], enabling it to be controlled and sensed via standard serial commands. The shield receives and sends information from the serial connection, bridging communication with the tablet via the light and capacitive-based techniques.

The StackAR breadboard application is run on an Android tablet as a HTML 5 web application. Through the HTML 5 multitouch libraries, the application senses the attached shield and its orientation, rendering a digital mirrored version of the hardware. Virtual components can be added to the workspace and wires can be connected between the physical and virtual hardware to create simple electronic circuits. When the virtual components are activated on the software interface, the application uses light-based communication to instruct the shield and Arduino of the state change. In figure 2.4, we can see a simple button wired to the hardware, allowing the user to interact with this simulated input to control the Arduino.

The tablet's built-in sensing capabilities can also be made available as simulated components, such as basic photodiodes and thermistors. By selecting these, their sensor values are locally read and communicated to the shield using light-based communication. Similarly, sensors attached to the hardware can use capacitive communication to relay data to the application, where, in turn, it can be visualised.

DISCUSSION

This exploration has been demonstrated alongside Droplet over the last year and has received informal feedback throughout the process. As an initial instructional tool for learning, its users felt that it provided further insight into component use, but was constrained by the physical dimensions of the tablet display. This makes scalability increasingly difficult, as the user's skills develop and they wish to interface with multiple components. Once again, the remote, visual controls for our objects were the most valued parts of this exploration.

Technically, this project highlighted the limitations of the capacitive communication, both in terms of data rate and reliability during user interaction.

AUGMENTING NON-DIGITAL OBJECTS

Much of our interactions still involve non-digital elements, and so the following work looks at how we can enhance these, augmenting their functionality. Within our environment, paper remains a common non-digital form of communication and expression, and so this work will initially focus on this domain.

INTRODUCTION

This application looks at how we can augment traditional paper-based business cards with our remote online identities to which they link. Despite the numerous ways in which we can share and distribute contact details, we still commonly use this paper approach. These disconnected objects are the lowest common form of communication between parties, and so are used to provide the references to further connect with the person's online identity. Having these details in printed form, we act as the interpreter, translating and interpreting the text to use with our devices.

To address this, I explored how these references can be directly accessed without our manual translation. By placing the card on the display, the owner's associated digital information is automatically collected and displayed around the object. Figure 2.5 shows the owner's current Skype status, LinkedIn profile, and music taste from last.fm. This demonstrates how an existing human-readable communication channel can be augmented, allowing the object to become a physical, tangible token to our online environment.

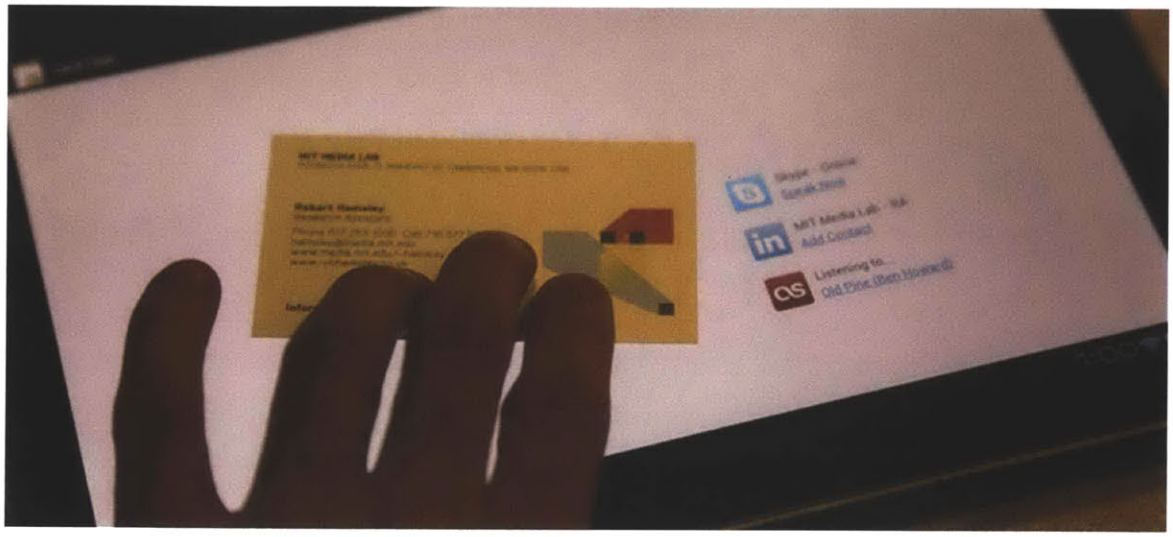

Figure 2.5 Augmented paper business card with online identity

Using the same approach, a second application was developed for storytelling and the creation of virtual paper puppets. In this application, either hand drawn or printed characters can be augmented, becoming interactive when placed on a capacitive touchscreen display. As shown in figure 2.6, this allows traditional creativity and storytelling to be merged with digital aspects. In this way, users can add sounds and animation to their paper characters, creating a digital association with the object. In both examples, our objects exhibit a digital shadow that provides alternative forms of interaction and information when viewed in a digital setting.

Figure 2.6 Augmented paper puppets

SYSTEM OVERVIEW

The paper is identified on the display by channelling the user's capacitance into 3 spatially distributed points. These are then used to identify when the object is being interacted with as well as its current location and orientation. By calculating the relative distance of the three touch points it's possible to generate a unique ID for each object. This approach is constrained to the number of touch points available on the surface and offers an address space relative to the size of the physical object. To channel the user's touch into these distinctive points, two techniques were explored.

The first uses conductive paint to create a touch surface on the underside of the object. The conductive path spreads the user's single touch across the surface, registering this as 3 independent touches. To reduce the visibility of the coating process on the material, a second technique has been explored using transparent vinyl masks. This approach uses a cut vinyl mask, which creates a thin conductive surface, similarly allowing the user's capacitance to be distributed into touch points.

To identify these touch points, a JavaScript library has been created which tracks and decodes the object's position and ID. This library can be easily injected into any existing websites, allowing them to react to the presence of the object. In the application scenario of the business card, this library is automatically loaded into the owner's website, enabling their online identities to be revealed when their physical business card is present.

DISCUSSION

These explorations demonstrated how our existing non-digital passive objects can become augmented without the overhead of additional electronics. In our search for ubiquitous computing environments, we often focus on bringing the digital to our objects and less on the reuse and connection with our existing devices. This exploration served as the basis for further discussion on how we can augment environments with additional functionality without using active hardware.

Within this work, the concept of "digital shadows", or connecting digital information to our objects, is demonstrated. This creates interesting

possibilities for just-in-time or physical mobile interactions by creating objects that can further describe their own functionality or properties.

MIRROR

As an exploration into ubiquitous computing systems, which can interpret interactions and mediate communication between our environments, I created a project called "Mirror".

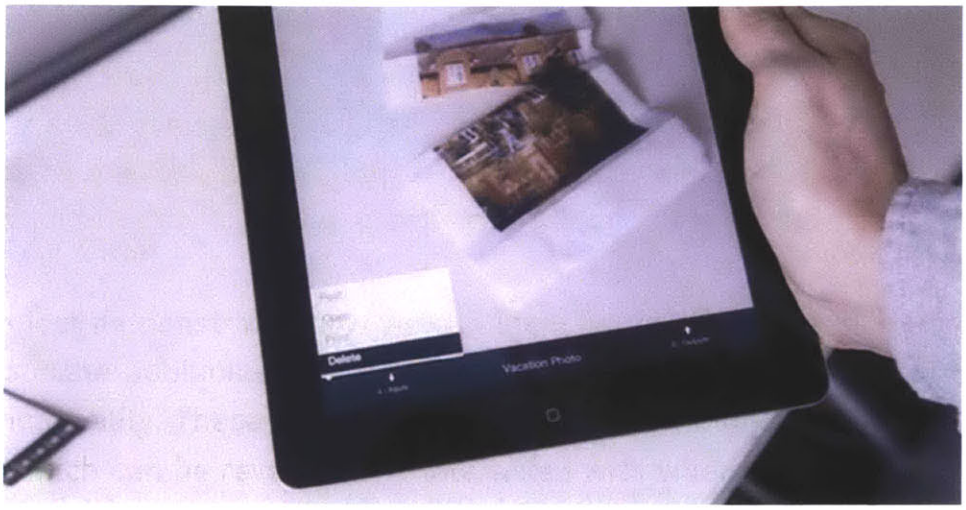

Figure 2.8 Mirror – Object & state identification

In Mirror, rather than using additional hardware for sensing, the project explores the potential of Computer Vision techniques for identifying objects and their state changes. This project forms the initial discussion and basis for the Flow framework, as developed within the remainder of this thesis.

INTRODUCTION

This project demonstrates how we can identify objects and user interactions, and associate additional digital information in the form of object properties and functionality. These stored properties create a "digital shadow" for each object, which can be revealed and interacted with when viewed from a digital device.

The manipulation of "digital shadows" is also explored in this work, as shown in figure 2.8. In this, Mirror can identify the object and its physical state, such as torn, automatically reflecting this observed change into the object's shadow. By tracking these changes, we can create metadata on the user's observed interactions and build systems with greater machine understanding of our environment and interactions.

SYSTEM OVERVIEW

Initially focusing on paper documents, Mirror explores how changes to a non-digital object can be seamlessly mirrored into a digital context without explicit direction from the user. To achieve this, the Mirror prototype takes the form of a tablet application that uses the attached camera as input into the computer vision system. With this, the user can use the tablet to expose our environment to the system and allow it to identify the objects around us and their properties, functions, and current visual state.

Accessing these associated functions, users can remotely interact with the object from the tablet device. To ensure that the system communicates between devices in near real time, the HTML 5 WebSocket [24] protocol has been used. This creates an open TCP socket between server and clients, allowing two-way real-time communication, as shown in Figure 2.9.

Figure 2.9 Mirror system communication

The tablet application locally processes each frame using a feature detection algorithm and identifies matching features with pre-processed object descriptors. Each identified object ID is then sent to the central server as a JSON message via the WebSocket protocol. Once received, the server uses this reference to retrieve further object details from the local database, such as input and output endpoints, and sends this back to the tablet for visualisation and user selection. As a user makes a selection through the tablet, this is communicated back to the server, where the action is processed and forwarded onto the appropriate connected client device. This architecture ensures that state is maintained centrally, and coordinates communication between the user's tablet and the device they wish to interact with.

STATE

Figure 2.10 shows the development of the fudicial markers from initial QR codes, AR Toolkit [27] markers, and to the final feature-based approach using the FAST [46], [47] algorithm. These markers serve to identify the object as well as its current state in terms of the 4 corners of the document. By tracking these individual elements, Mirror is able to detect when the object has been torn and destroyed or lightly deformed and crumpled. By tracking these elements, we are able to detect changes in this physical object and connect this to digital actions.

Figure 2.10 Fiducial marker development. QR Codes, AR Toolkit, FAST key points

In this prototype scenario, it was explored how the physical action of tearing a document in half can be mirrored directly to the digital action of deleting the file from which it was produced.

The initial QR code approach was ineffective in terms of user experience, as simultaneous tag recognition was unreliable. To overcome this, a second approach, focusing on the ARtoolkit fudicial markers, was explored. This provided sufficient target tracking, but requires the visual augmentation of the object, destroying its original visual appearance. During the research stage into fudicial markers, feature-based tracking was identified as a solution to dissolve the current markers.

As shown in Figure 2.10, the FAST feature detection algorithm generates key

points based on the Harris Corner Detector, producing a scale and rotation invariant descriptor for the object. By slicing the original document into 4 descriptors, we can simultaneously track how the corners relate to each other and, in turn, identify when its state changes.

OBJECT ACTIONS

Mirror also serves as an exploration into the field of Physical Mobile Interaction [49] (PMI) and the use of remote mobile devices as mediators and controllers for our physical hardware. In this exploration, Mirror uses the FAST [46], [47] feature detection algorithm to match key points from within each frame, with a library of pre-processed image targets. Through this, we are able to identify specific objects without having to adjust or augment their appearance. By associating each target with an object, we can link further properties and functionality describing a "digital shadow", which can be made available at the time of interaction.

In this exploration, the ability to access digital versions of objects and move these directly between our devices is demonstrated. As shown in figure 2.11, a PDF document is added to the system using each page as a tracked target image. This object has associated inputs defined through the Mirror web interface, defining actions as "View" and "Print", with each containing a URL to the associated PDF that produced the document.

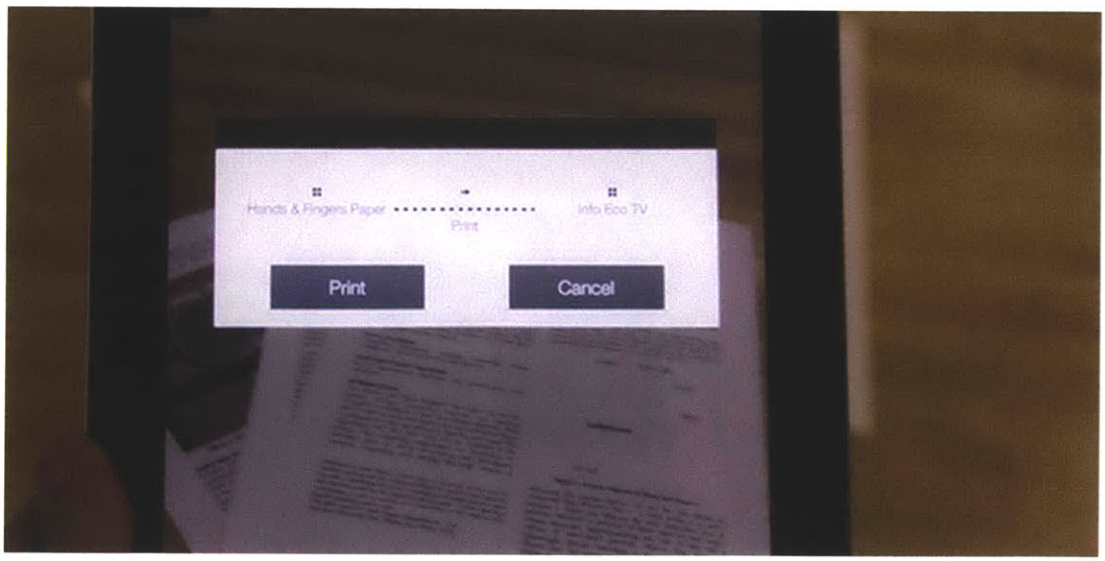

Figure 2.11 Physical Mobile Interaction – Printing

By viewing the document, either as a physical print or an on-screen PDF, the associated inputs are gathered from the object shadow and the options are displayed to the user. The user can then select input options, such as "View", before moving to another device to display its shadow properties and select an output. In this scenario, by moving to a laptop as output, the document will be seamlessly downloaded and displayed on the device that is being viewed. This interaction is made possible due to the stored URL endpoints and the local Mirror object handlers, which interpret the user's selected actions.

This physical mobile interaction can also be undertaken with devices without displays, such as printers. In this manner, the user can select "Print" and move directly to the physical printer they wish to output the document on. As this object is similarly tagged, it is identified by Mirror and an agent within the network gathers its network address and uses the lp [54] command line utility to silently send the document to the printer. Through this, a user is no longer required to maintain the overhead knowledge of the printer's address and how to connect to it in order to complete the task.

Communication between these devices is undertaken via a persistent connection from a desktop client to the central server. JSON-based messages are then passed, describing the selected functionality and the URL endpoint.

AUGMENTED REALITY INTERFACE

As highlighted in the user feedback in the previous explorations, having software defined elements to our objects is highly desirable, as it enables increased access to information and functionality without directly adding to the physical object. The flexibility of software with a hardware interfaces brings us the ability to reprogram the output actions of the physical interface.

In an AR environment, we are no longer constrained to placing objects on displays, and so larger immovable objects can be augmented and remotely controlled. Figure 2.12 shows an example of a door that has been augmented with the controls to open and close the lock from the mobile device.

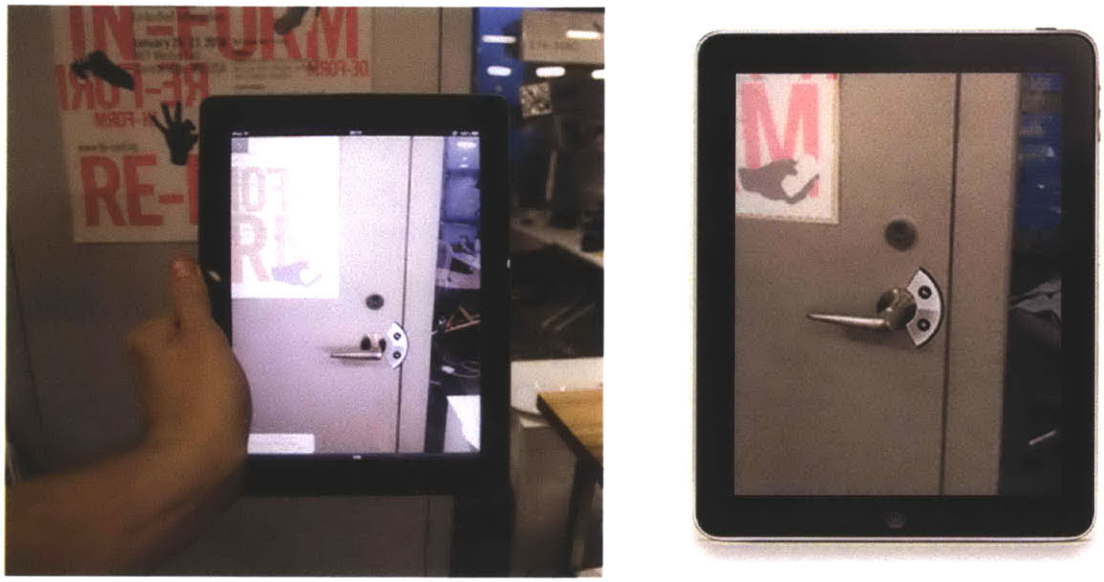

Figure 2.12 Mirror – Augmented Reality Lock

In this environment, when the user observes the door, its controls are rendered and positioned around the handle, enabling remote unlocking and locking functionality. Using this, an authenticated user can walk towards the door, with it automatically unlocking, based on their permissions, and it being viewed through their authenticated device. If an unauthorised user approaches, a "leave a note" drawing surface is displayed, as shown in figure 2.13. The user can then write virtual notes on the surface using the AR application, with this being emailed directly to the office owner as an attachment.

Figure 2.13 Mirror – Augmented reality door note

Focusing on objects which move between digital and physical forms, paper is, again, a natural choice for augmentation due its ability to be visually represented in both states. In this, we look at just-in-time interaction and the automatic association of remote digital information. Figure 2.14 shows a picture from Facebook, augmented with associated online information.

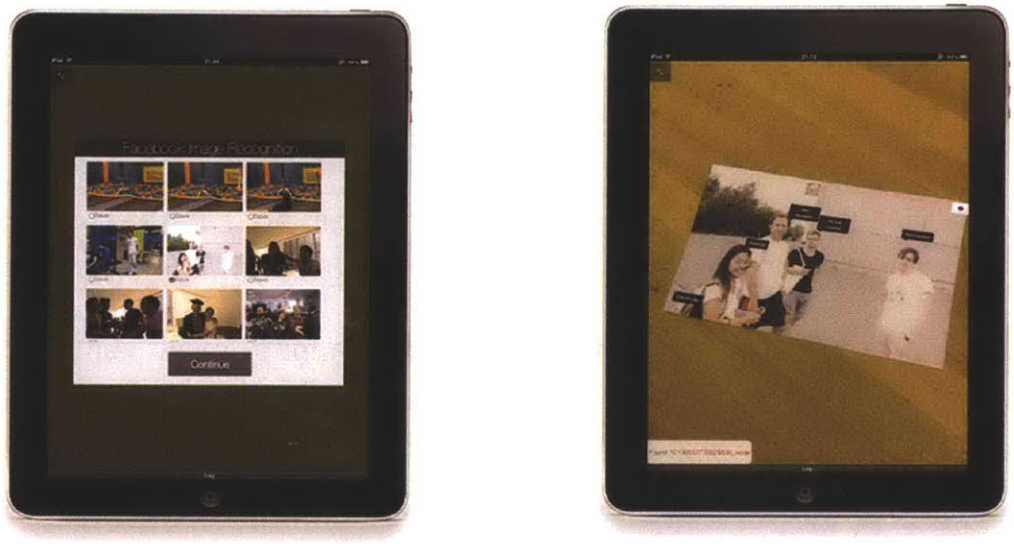

Figure 2.14 AR Facebook Image

In this application, all images available on the user's Facebook account are processed, bringing tags, comments, and location information to the object when viewed through this AR application. In this scenario, users can easily identify who is within the picture and view their associated information at the point of interaction without directly accessing Facebook.

To prototype these augmented reality interactions HTML and CSS 3, transforms are used with the feature detection algorithm provided by the PointCloud SDK[31]. This approach enables rapid development of augmented reality apps following common Agile web standards.

This form of augmentation once again demonstrates the goal of bringing digital functionality and information out into the physical space, rather than the opposite of digitally representing our world.

WEB CONTROL

In creating an environment where our objects can be visually identified and associated with actions and information, it is important to consider how we define and create these associations. In this exploration, a web interface was developed, as shown in figure 2.15. This provides high-level control over the target images being tracked by the application and defines the input and output interfaces for user actions, such as viewing and printing.

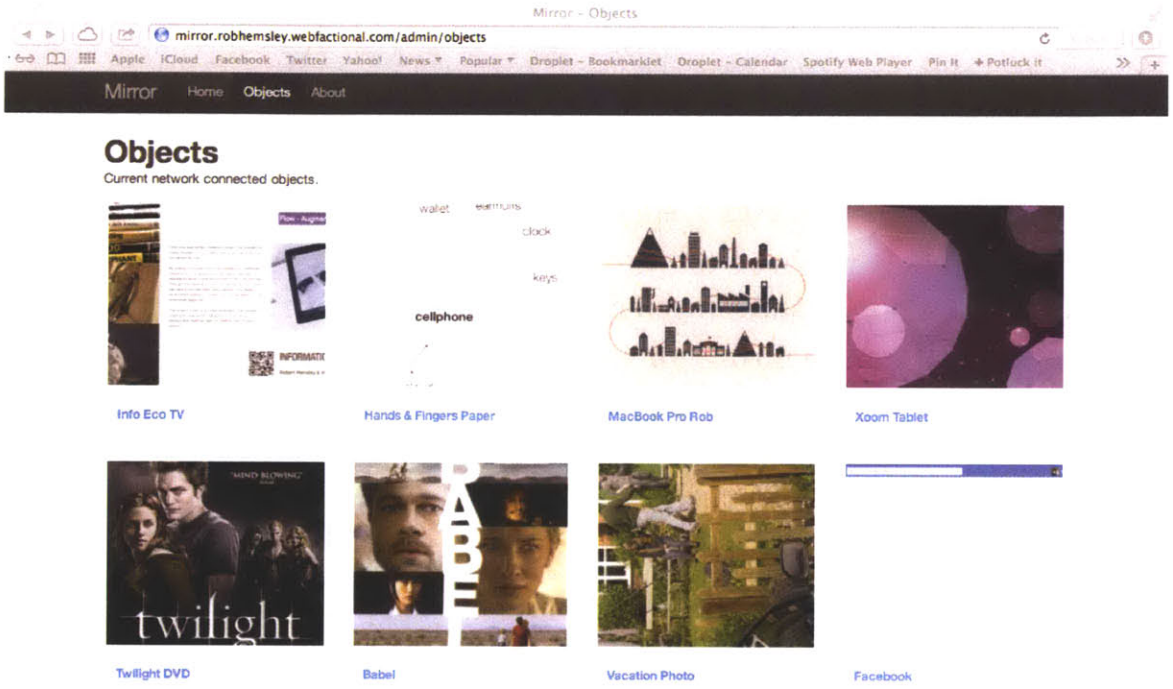

Figure 2.15 Mirror web interface

In this system, image descriptors must be generated through third party services, and therefore visual updates can exhibit a 10 minute lag. This makes the system ineffective for interaction with normal user devices, as the display must not visually change over time. In this prototype, screensavers and background images were used as tracking targets to provide consistent graphics between user interaction.

EVALUATION AND DISCUSSION

This exploration was similarly demonstrated during open house events, and received positive feedback from electronics manufacturers and those working across multiple consumer devices. During these demonstrations, the door lock and paper ripping examples were the most commented on, with the PMI

elements and digital shadow properties being perceived as the most valued.

To evaluate the user experience provided by Mirror, a pilot study was also undertaken where users completed 3 tasks with the system before engaging in an open discussion. The first task was to open a physical document on a laptop display, the second to tear a digital image in half, deleting the associated file, and the third to unlock an office door via the mobile device.

Within these interactions, it was highlighted that observing the world through the display was distracting and inhibited their interactions. User 2 went as far as to draw parallels with a war photographer being withdrawn from reality as the world is viewed through a camera lens. This issue was further highlighted during the paper ripping example, when users had to transition from using the mobile device, to physically undertaking an action, and then back to the mobile device. This created an unnatural experience in which the user had to explicitly reveal the state change to the system, forcing them to focus on the mobile device rather than their environment and normal interactions. This narrowing of attention was commented on by User 3 during the door lock AR example, where they felt the overlay graphics didn't provide improved interaction but instead found them too small and inaccessible. Observing this user, there were clear difficulties in scaling the AR graphics, with the tracking target occasionally being lost. This caused frustration and confusion with the user, forcing them to focus on the mobile device for feedback or cues regarding its operation.

During the testing of task 1, the output action selected between devices was chosen either automatically or manually by the user through the mobile device's display. The feedback from these interaction modes was mixed, with 2 users commenting that the automatic approach seemed more natural, as it required less conscious attention. The remainder of users felt more negatively toward this and were uncomfortable with the lack of feedback that the automated approach provided. Building on this feedback, the remainder of the thesis describes the implementation details of Flow, a framework designed to support reality-based interactions such as those initially explored within Mirror.

FLOW – FRAMEWORK

Constructing a framework to support reality-based interactions between our environment and digital devices has been a significant technical undertaking. I have personally designed each layer of the system, ensuring it remains as open as possible and free from commercial closed source libraries. Following this approach, it has been possible to ensure that the overall design rationale and system objectives have been maintained throughout.

In this chapter, I shall describe the system development and architectural rationale before outlining the individual system elements and their intercommunication.

INTERACTION RATIONALE

Based on previous explorations and related project work, the following design rationale has been identified, which forms the basis of the framework's core functionality.

GROUNDED IN REALITY

Within traditional AR environments, the user's viewpoint is augmented with digital content via a handheld or head-mounted display. This approach creates a private personal view of the world, which draws attention away from reality and excludes those with whom you interact from sharing this rendered view. This approach to technology prioritizes information access over real-world personal interactions, and so creates clear social implications for those interacting with the user.

To prevent the creation of a single viewer personalised environment, as systems like Eyetap [36] point towards, the Flow framework should be constructed around existing reality, avoiding augmented visual overlays. To achieve this, display reuse should be attempted, as demonstrated in Droplet and Mirror. This approach will focus the user's attention and interaction with the environment and their physical objects and devices, rather than on a private virtual overlay.

CROSS-ENVIRONMENT

Much of our physical environment has been flattened and iconized in order to allow us to continue interacting with it from a digital setting. This framework should be designed taking the opposite approach, constructed to bring digital functionality and flexibility to the existing physical environment rather than making digital representations of it. This approach is at the heart of RBI design and should be considered throughout.

As displays and mobile devices form an integral part of the user's interactions, the framework should integrate these, allowing users to select and undertake actions across devices.

TRANSPARENT

As the system observes the user's interactions, it is vital in terms of privacy and from an interaction standpoint that they are aware of which devices are recording activities and are part of the framework. The framework should provide a means to discover the objects and functionality, and to ensure they are not overly obfuscated from the system's operation. To reduce the gulf of execution between user interactions, the framework should also provide cues to which the user is accustomed, either from their digital or physical interactions.

TECHNICAL

In developing such a framework that adheres to this interaction rationale, there are many technical implementation details that must be considered.

OPEN SOURCE

To maintain control over all aspects of the framework, the solution should be developed using only freely available open source techniques. This ensures that unknown blackbox implementations are eliminated, helping enable further extensions and adjustments to be easily incorporated into the code base. Within the domain of computer vision feature detection, many rapid advances are currently being made, and so to enable further integration, the full implementation details must be made available.

PLATFORM-INDEPENDENT

To enable users to operate between their devices, a cross-platform solution must be found. The system will therefore take inspiration from current web-based technologies and their platform-independent nature in order to allow Flow to fluidly operate across devices.

REAL-TIME

As a reality-based interaction tool, the system must provide machine understanding and associated metadata in real time with their interactions. Protocols and algorithms should therefore be selected with the minimisation of computation time and network latency whereever possible.

SYSTEM OVERVIEW

Flow is comprised of 3 main elements, a wearable head mounted camera for input, a centralised image processing server for control, and platform-independent display clients for output, through which users access and interact with the system. In the following section, I shall outline the design and implementation of these elements.

CAMERA INPUT

The framework is designed to be flexible, taking camera input from a variety of sources through platform-specific video clients. The system follows a client/server model, so any device with a camera and networking capabilities can input video to the Flow server for processing. This allows the exposure of the environment and interactions within it to be made to the system using cell phones, desktops, laptops, and new wearable computing hardware.

Building on the initial Mirror project exploration, a cell phone application was developed for Android that provides low frame rate images to the central server. This application captures raw YUV frames at 640x480 pixels and converts them to JPEG, transmitting these via a WebSocket connection as base64 encoded data. This provides a low rate (2 fps), high latency stream of images designed for algorithm testing. As this operates from a handheld device, it is easily accessible to users but requires explicit activation and

directed interaction when revealing the environment and its objects.

To enable more natural interactions, the user's hands must remain free and unconstrained from the input device, and as such a wearable system is required. Systems such as Google Glass [20] are a natural choice, as they provide a head-mounted camera built around the Android operating system. Despite it's appropriateness, at the time of writing, the Glass system, like many other emerging wearable devices, is currently in development and not publically available. To overcome this barrier to development, the system continued using Android devices, further improving the video streaming algorithm while awaiting alternative commercial solutions.

The APX smart glasses [32] as shown in figure 3.1, are a recently-developed product that provides a transparent head-mounted display (HMD) and camera. This hardware, although still in development, was loaned to the project and used in the initial development and testing phases. As one of Flow's rationales is to bring digital functionality to our physical environment without visual overlays, the display portion of the glasses was not utilised. As this functionality is redundant, and due to their current wired form factor, a further, more subtle solution was explored.

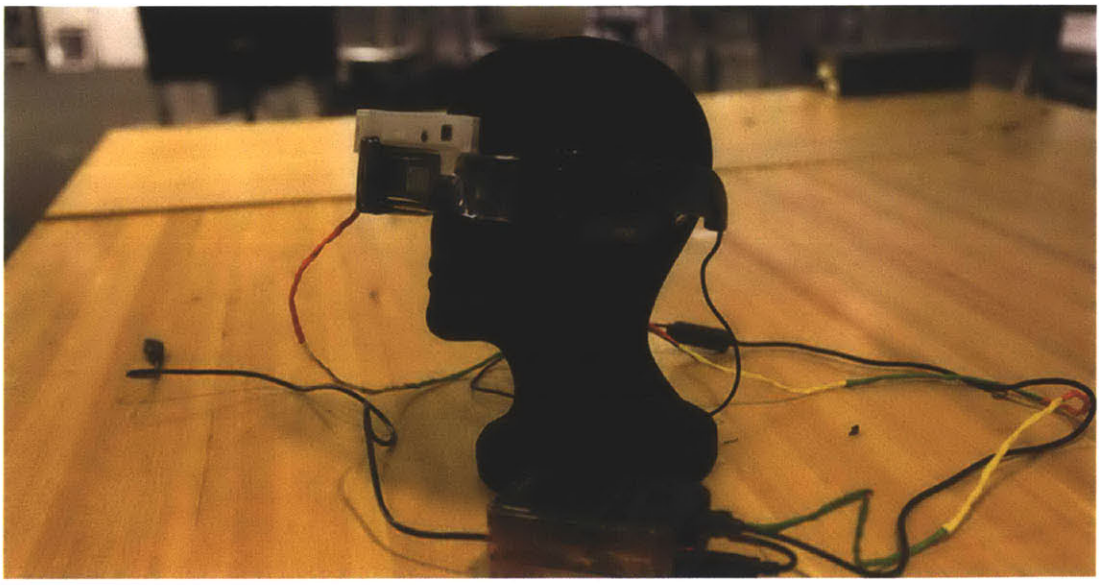

Figure 3.1 APX & EPSON Head Mounted Display

To reduce the visible appearance of the head-mounted camera, "spy" glasses were sought as an alternative. Within the final implementation, a pair of

MegaSight[61] video glasses were selected, providing a USB video device class interface (UVC), enabling their use as a webcam. Figure 3.2 below shows the glasses with their controller and display stand. Due to the UVC version implemented on the glasses, they were unable to operate directly with the Video4Linux drivers that are commonly found on the latest Android distributions. To overcome this, an external wireless controller was developed to stream video frames to the server.

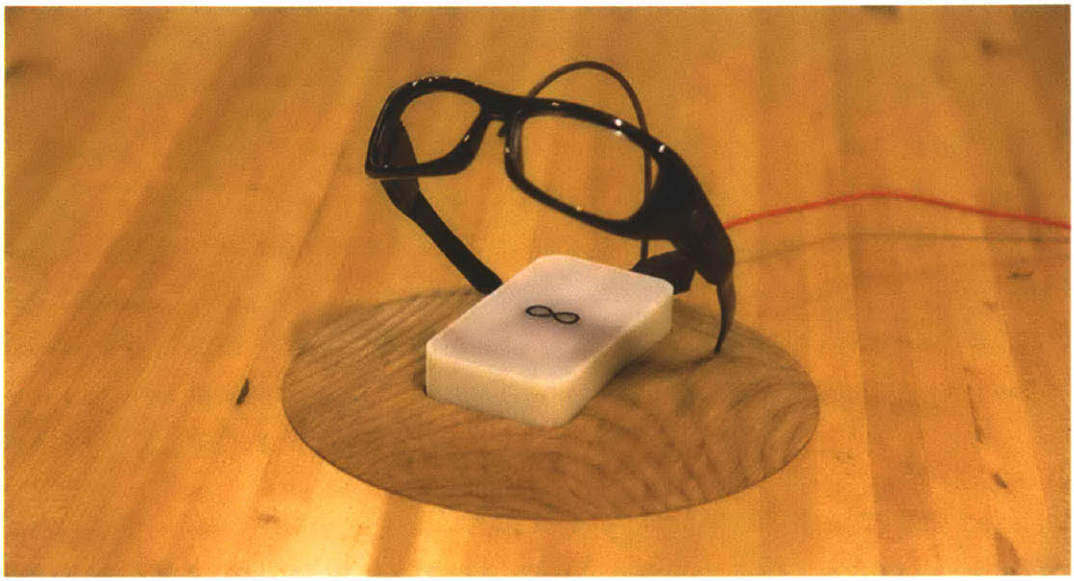

Figure 3.2 Flow glasses, controller & recharge stand

The controller to which the glasses are connected is formed from a Raspberry-Pi[41] computer with an 802.11n wireless adapter. This runs the Raspbian OS and a local Python OpenCV script that captures and chunks image frames into 8150 bytes for transmission over UDP. This unit, as shown in figure 3.2, also provides battery power to the Raspberry-Pi, allowing it to remain mobile and be worn by the user. When the unit is returned to the base station, power is wirelessly transmitted to the unit, recharging the contained Lithium ion batteries.

Through this controller, JPEG frames are captured at 640x480 pixels and streamed to the server at a capped 10 fps for processing. A web interface is also provided for the controller via a local web server that enables the remote shutdown, restart, and adjustment of the controller's frame rate.

CENTRAL SERVER

To process the video feed transmitted by the user, a centralised server is used that interprets frames and provides metadata on the observed environment. This creates a client/server architecture where system knowledge and intelligence are centrally maintained, with remote clients receiving specific commands based on these interpreted frames. This approach ensures that platform-specific code is minimised, with client implementations acting as a thin, lightweight application that simply displays and provides feedback to user interaction. The server is also responsible for maintaining the overall state and control between the user's objects and digital devices, acting as a router between communicating clients.

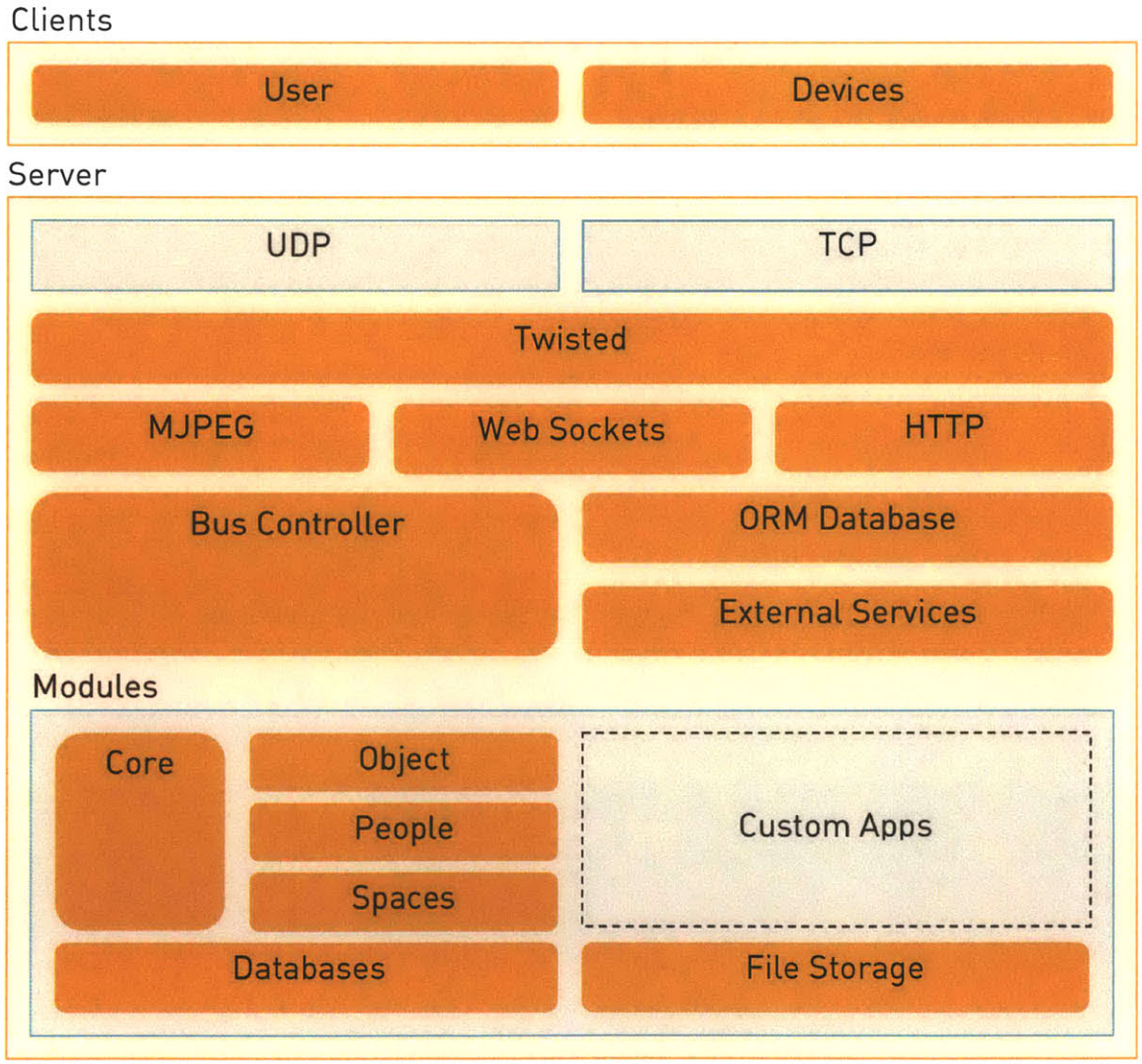

Figure 3.3 High Level System Block Diagram

The server implementation is developed using the Python programming language with the Twisted [19] networking library for low level TCP and UDP support. Further C++ based libraries, such as OpenCV, are also integrated into the system through associated Python bindings.

The server architecture is developed to be modular and flexible by following an enterprise service bus design. This ensures incoming video frames from a variety of users can be simultaneously processed and passed through a logic stack of independent application modules. Figure 3.3 illustrates the high level operation of the server, with TCP and UDP traffic being received from clients and processed through a series of independent modules that generate metadata.

For each Flow client, the server authenticates the object or user and handles incoming commands and image frames. Each connecting object or user must authenticate with the server, providing a persistent TCP WebSocket for real-time communication.

DATABASE

To abstract the system from the database implementation, the SQL Alchemy object relational mapper (ORM) is used. Through this native Python object, classes are defined which directly map to database entries. These objects are serialisable and passed throughout the system, where they are inflated as required on the users endpoint devices through Python, Java, or Javascript.

To provide spatial searches for the user's locations, the PostGIS library is used, which stores the user's location for each communicated message as EPSG:3857 [60] coordinates. This location is detected through the user's cell phone via a Wifi Positioning System (WPS) [5]. This approach ensures coarse location data is maintained with little effect on the user's battery.

Using this object orientated approach, a central key value store is provided, in which object shadow values may be stored.

MODULES

To remain extensible, the framework uses the concept of modules or

applications to create contained logic blocks that add and control the platform's functionality. The framework, by default, provides 5 modules within the domain of people, objects, and spaces, with a further two core modules providing helper functionality.

Each of these modules operates through clear defined interfaces via message events within a sandboxed environment to prevent cross-application interference. Each module maintains its own local file storage for processing and accessessing the global bus controller for persistent central database access, such as the key value pairs for the object's digital shadow.

The modules can be accessed either through the controller event bus via WebSocket messages or via a HTTP RESTful [15] interface. Each module inherits basic functionality that is overridden to provide customised behaviour. For example, the inherited method render_request can be overridden to provide external RESTful access to the module through top level URLs, such as:

http://flowserver.media.mit.edu/usr/<USR_ID>/apps/<APP_ID>/<REQUEST>/

Requests made in this form are tunnelled through the system to the render_request handler of the module, where low level response's for all HTTP traffic can be returned.

These modules are dynamically loaded at run time via the system control bus, which controls intercommunication between modules and their associated threads. Messages are handled via a subscription model where the name space is used to create unique event names in the form of "uk.co.robhemsley.flow.objrec.visible". As these events are fired, associated subscribed functions are called passing payloads, which further describe the observed event details. Using this, each module can generate and trigger events containing generic serialisable data that the system further handles and transmits between modules and clients.

The following section outlines the implementation of these 5 core modules, and the functionality and interfaces they provide.

LIVE

The Live module is responsible for aggregating input video streams and providing a single MJPEG video source for the user. Within initial prototypes, video frames were transmitted using the existing WebSocket command connection between the client and server. This required individual frames to be encoded in base64 [50] before transmission, adding a 33% overhead to the frame and further latency due to its TCP based transmission. Within later prototypes, raw frames were encoded as JPEG images and transmitted via UDP in chunks of 8150 bytes, before being reassembled at the server side. Building on the Twisted [19] Python module, a FlowMJPEG class was created that taps into the specified user's input stream and provides an open multipart HTTP connection for outputting JPEG image frames. These images are displayed to the user via the web interface, as shown in figure 3.4.

To enable cross-browser MJPEG support, an HTML 5 canvas element is utilised that requests and redraws the individual frames at a fixed rate of 10 frames per second. As the system is not intended for AR applications, the frame rate is intentionally capped to reduce the demand on the centralised server.

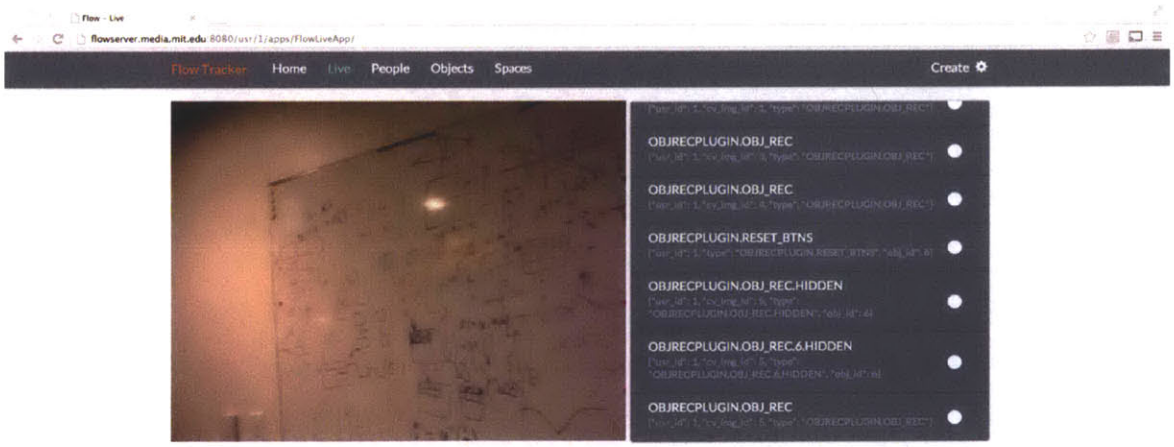

Figure 3.4 Flow Live Application

To provide feedback on the system's operations, the Live application connects

to the enterprise bus controller via an additional WebSocket and receives real-time serialised events. Shown on the right side of figure 3.4 is a list of system events generated while the white board object is viewed.

Through this module, endpoints are provided to which applications can view and extract live frames based on the framework's real-time metadata. As the feed is served over HTTP, application HTML renders can similarly access this data without requiring further interpretation or the use of video codecs.

OBJECTS

At the core of the framework is the object tracking module, which enables the detection and tracking of known planar surfaces from within the user's video feed. By preloading the tracker with images of known object surfaces from a variety of angles, the framework is able to naively identify objects during user interaction. As the framework aims to work across environments, this module maintains the visual state of the connected digital devices, updating the tracked surfaces as the device displays alter during user interaction.

As tested in the Mirror application, visual tracking can be achieved through augmenting objects with fudicial makers or through feature tracking algorithms which utilize the object's existing visual appearance as the marker. This "marker-less" tracking approach is used by Flow to reduce the visibility of the platform.

FEATURE DETECTION ALGORITHM

To achieve markerless tracking, the system must combine multiple algorithms, which cover the feature detection, description, indexing, and matching of target images. In recent years, many approaches have been proposed for these algorithms, and so the following section evaluates these, identifying those most appropriate for the framework.

For this evaluation, a 5 second video clip containing a single image target has been used as input and processed against the current 5 most common feature detection algorithms. The average number of detected features for these input frames and the source dictionary image are shown in figure 3.5. These tests were undertaken using OpenCV implementations and run on a

2.6Ghz Intel Core i7 Mac with 4 GB of RAM.

Algorithm	Features	Detection (ms)	Vocab Size (obj)	Feature Per ms
FAST[10]	457	1	606	457.00
ORB[11]	293	4	64	73.25
GFTT[12]	701	12	708	58.42
STAR[2]	89	10	87	8.90
SIFT[9]	218	28	230	7.79
SURF[3]	164	64	111	2.56
MSER[4]	48	38	46	1.26
BRISK[8]	84	243	106	0.35

Figure 3.5 Feature detection algorithms - test results

In selecting a feature detector, we wish to identify the largest number of visible, detectable key points in the least computationally expensive manner. The greater the number of key points produced for an input, the more potential matches can be made. From the tests, the two fastest detectors were identified as FAST [47] and ORB [48] with 457 and 73 features per millisecond of processing time.

Features from the accelerated segment test (FAST) algorithm are detected corners, which are identified through the use of a segmentation function that operates over a 7x7 image patch. For each corner candidate, a radial search is undertaken, examining pixels to find a continuous arc of values with higher or lower intensities relative to the nucleus pixel. As shown in figure 3.6, a Bresenham circle is generated of 12 points based on the nucleus pixel value p. To improve efficiency, not all points are tested, but rather only the 4 pixels (1, 5, 9, 13) on the compass points are used to reduce the search window size. For a candidate to be classified as a corner, at least 3 continuous points must be found, which implies an arc of pixels equal to or greater than 9.

Through FAST, a large number of key points are produced based on a single variable of the intensity threshold between the nucleus pixel and circular ring. Having such a large number of values to process through a descriptor and matching algorithm can be computationally expensive, and so the ORB algorithm has also been considered. This algorithms builds upon FAST,

adding a measure for "cornerness" which prioritises the detected points.

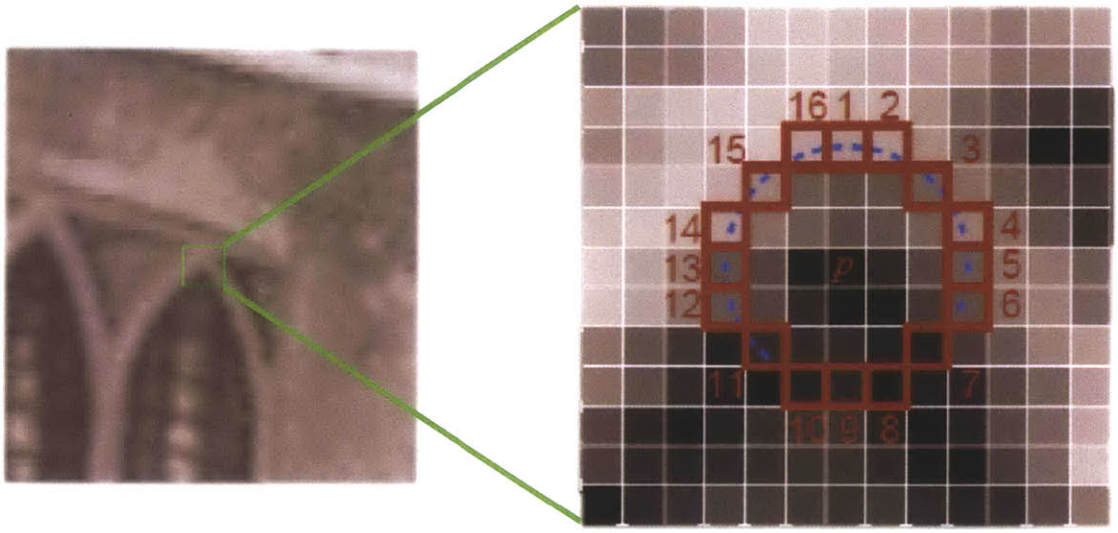

Figure 3.6 Fast Feature Detection[10]

ORB [48] takes the computed localizable corners produced by FAST and computes a measure for each corner based on the gradient changes observed within this window. By calculating the convulsion of these gradient changes on the x & y axis, we can determine where two edges intersect, creating a corner. Corners are clearly not always orientated along the x, y axis, and so this calculation isn't effective for orientated features. To account for this, eigenvalue decomposition of the matrix is undertaken, which produces a metric for the observed gradient change irrespective of the feature's orientation. If these eigenvalues are both relatively large, then a visible corner is assumed to have been located, and based on this relative value, the features are prioritised. Applying a threshold to this, we can reduce the number of key points and focus on those which are most pronounced and likely to be detectable between frames.

These two algorithms present an efficient approach to describing features and will be further evaluated with associated descriptor algorithms before a selection is made.

Once the features have been identified, a descriptor must be generated that represents the image in a scale-, rotation-, blur-, and lighting-invariant manner. Finding such an algorithm will help ensure a robust system can be

developed for use within real world environments.

The following two figures show the time required for the extraction, indexing, and matching of key points from both ORB and FAST detectors, using either FREAK [3], BRISK [33], and ORB [48] descriptors.

Descriptors – ORB Detector

Algorithm	Extraction (ms)	Indexing (ms)	Matching (ms)	Total	Matches
ORB	7	3	53	63	7
FREAK	34	2	29	65	7
BRISK	236	4	85	325	7

Figure 3.7 Descriptor tests - ORB

Descriptors – FAST Detector

Algorithm	Extraction (ms)	Indexing (ms)	Matching (ms)	Total	Matches
ORB	3	3	68	74	8
FREAK	33	3	217	253	6
BRISK	238	4	117	359	6

Figure 3.8 Descriptor tests – FAST

These tested algorithms share the common BRIEF [10] descriptor, but vary in terms of the sampling pattern used around each key point. Both FREAK and BRISK use a circular pattern with closely-packed subcircles, which increase in diameter with increased distance from the nucleus pixel. FREAK models this sampling pattern after the retinal ganglion cell distribution of the human eye, with pixel detail ranging from fine to coarse with increasing distance from the point of focus. Based on the above results, ORB consistently outperforms BRISK and FREAK, producing descriptors 4 times faster than the nearest competitor.

As the descriptors are binary and produce a cascade string for each image, the Fast Library for Approximate Nearest Neighbours [38] (FLANN) matcher with locality-sensitive hashing [52] (LSH) can be used to provide an efficient neighbourhood search. The FLANN library abstracts the randomized kd-tree, hierarchical k-means tree, and multi-probe locality-sensitive hashing algorithm, automatically selecting between these based on the input data set and available parameters. Due to the binary nature of the tested descriptor,

LSH is used, which stores points in several hash tables, hashing each point into separate buckets. During matching, descriptor buckets are retrieved and their elements compared using a brute force approach to identify nearest neighbours with the highest probability given the number of tables. These matches are efficient, as the hash function is a subset of the signature bits of the cascade string. Based on the tables, the descriptors have common sub-signatures, with the nearest neighbour distance becoming the hamming distance between string descriptors.

With the goal of reducing computation time, the ORB algorithm has been selected based on the above results. This descriptor with the ORB detector produces the same number of matches as the other algorithms with the least amount of computation time. Comparing this detector with FAST, which produces a greater number of detected key points, we can see that in the descriptor stage this negatively effects the matching process, and so ORB is favoured with its pre-processing prioritisation.

Using ORB, an initial prototype was developed utilising a cell phone for image processing and matching. Although current mobile devices provide sufficient processing power for real-time feature detection, matching still remains a bottleneck. Both the ORB and SURF algorithms were tested using an Android implementation that processed frames at a rate between 1 and 0.2 fps, which varied based on the number of detected features. This delay is unacceptable for the framework, and so the client-server model was adopted, using a dedicated server for processing.

PROCESSING

Using the previously described algorithms, each frame is passed via the enterprise bus controller to this module, where it is distributed across multiple threads, overcoming the Python Global Interpreter Lock (GLE).

For each frame in which an object is detected, its matched features and those that lie within the defined target's surface are serialised and passed to subscribing modules. The homography of the tracked object is also computed, and a perspective warp applied to provide a 2D planar image of the tracked area, as observed within the frame.

Figure 3.9 Computed Homography and perspective warp

Figure 3.9 shows a tracked page from a notepad undergoing a perspective warp based on the computed homography. This technique is later used to identify object changes based on the detected features within the extracted area. The relative size of the detected target is also used to predict the user's distance from the object. Through this, proxemics interactions are enabled, adjusting the environment based on the user's relative position.

VIRTUAL SURFACES

As a wearable system, the head-mounted camera can observe the user's interactions with both digital and non-digital objects and produce events based on observed changes. Through this, users can define virtual touch surfaces, which are activated by the observed presence of the user's hand. This allows non-touch screen display devices to have simulated touch input capabilities, and physical objects can be interacted with in a similar way.

To enable these virtual touch interactions, a module is created which enables the definition of areas of interest within the currently tracked planar surfaces. By using the previously described ORB [48] feature detector, key points within a specified area are tracked and as occlusion occurs this is detected. By default, if a more than 50% loss in key points is detected, as is the case when a hand is placed over the area of interest, a "uk.co.robhemsley.objrec.btn.down" message event is fired. Further events are also provided for button presses (down/up), which contain the button

object, dwell time, and detected key points. Using the object relational database, button objects can be associated with a target, which defines coordinates for a closed polygon region of interest. These coordinates are based on the source image and are compared with the descriptor using the MatPlotLib [25] containment function. Figure 3.10 below shows the virtual touchscreen application as described in section 4.3 with tracked interest areas.

Figure 3.10 Virtual surfaces - Key point Occlusion

During initial prototyping, this functionality was used to enable grabbing and dropping type gestures between objects. By detecting occlusion on the defined centre region of a display, users could indicate to Flow the intention to copy and paste information through these gestures. Between two devices, Flow copies data from the input device, sending it through the persistent WebSocket connection and displaying it on the output device.

SPACES

The spaces application manages the location services for the framework, providing events based on the user's physical movements. Figure 3.11 shows the interface provided to manage and view the user's location. By maintaining awareness of the user's position, we can create geographically aware services, as shown in section 4.12, and identify the location of objects viewed by the user. As the system identifies objects based on their visual appearance, it can lead to false positives when visually identical objects, such as consumer electronics, are placed within the same environment. Having this locational awareness can help reduce this error by weighting the likelihood of the object being in a specified location and viewed by a specific user.

In this interface, the user's current location is displayed as a pin on the interactive map, which updates periodically as the user moves. To define areas of interest, tools are provided for drawing polygon shapes. Each polygon is stored as an EPSG:3857 [60] projection coordinate and saved to the shared PostGIS database as a location object. This data enables geospatial queries on the user's location and the construction of definable areas of interest. As the user's primary device produces location updates, these coordinates are passed to the Spaces application and stored. As a user enters or exits a defined area, events are fired which can be subscribed to from within the IFTTT interface.

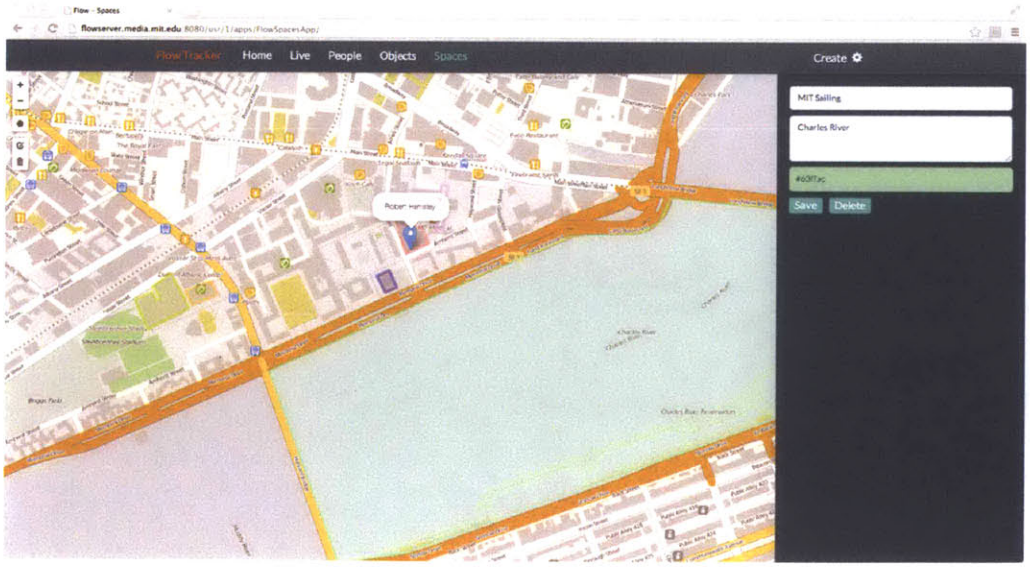

Figure 3.11 Spaces Application

As well as providing the ability to create user-defined locations, this module contains logic to predict areas of interest based on the user's historical movements. Building on previous work on the reality mining [13] of user movements for mobile opportunistic networks [23], this application automatically predicts the user's home and work locations. These are determined by gathering the coordinates of the user within specified time periods, such as the early hours or mid afternoon. This data is compared over a two week moving window, defining the location based on the frequency of appearances within a region. This naïve approach assumes a regular sleep and work pattern, but produces effective results for the framework's purpose.

PEOPLE

The People module provides basic face detection to enable the framework to have an awareness of whom the user is interacting with. At the time of writing, this module is still under development, with detection being undertaken using a hard-coded training set of user images.

To achieve this recognition, the OpenCV library is used, which detects and extracts head regions, comparing these to a training set of known faces using the Fisherface algorithm. To detect head regions, the Viola Jones [56] boosted cascade classifier is used with a frontal face Haar descriptor. These extracted regions are passed to the Fisherface algorithm, which uses linear discriminant analysis to determine and compare the characterizing features with the pre-computed classifier.

Informally using this module within the framework, many false positives were observed, and so this functionality is only currently utilised within the IFTTT module to provide head detection. Facial recognition events are produced but currently not further processed by the subscribed applications.

As a privacy measure, images are processed through this module with detected head regions undergoing a filter blur effect to obfuscate the user's visual appearance. This is done to prevent further modules from undertaking facial recognition and attempting to identify users who have not consented to being viewed by the system.

IF THIS THEN THAT

The If This Then That (IFTTT) module enables users to create their own rules for their environment, producing triggers and actions based around their physical interactions and movements. By exposing the events produced by each of the inbuilt and additional applications, a user can create their own logic wrapper for the environment, creating custom behaviour to suit their needs.

Figure 3.12 below shows the web interface provided by the application with the ability to add and edit rules. The current version supports "and"/"or" operators and provides outputs as onscreen messages, emails, or http get methods to third party services and endpoints. As illustrated, a rule has been created using the location (Spaces), object (Object Rec), and external api triggers, so that if the user attempts to leave their home location while it's raining, a message will be sent to their nearest display that they should bring a coat.

A second rule has also been created, associating a reminder with the action of viewing an object. Using this approach, it's possible to assign information to our objects, embedding additional detail into the world around us. Through this interface, it's possible to rapidly develop just-in-time systems by uploading object images and writing a custom rule.

This behaviour is supported through the enterprise service bus architecture and use of action events. For each application, IFTTT methods are provided, which define available actions and their mapping to event name spaces. For each rule, subscriptions are made to the appropriate action event and validated as the input conditions change. When the input conditions are evaluated as true, the output action is processed and appropriate messages are sent to the third party services and connected Flow devices.

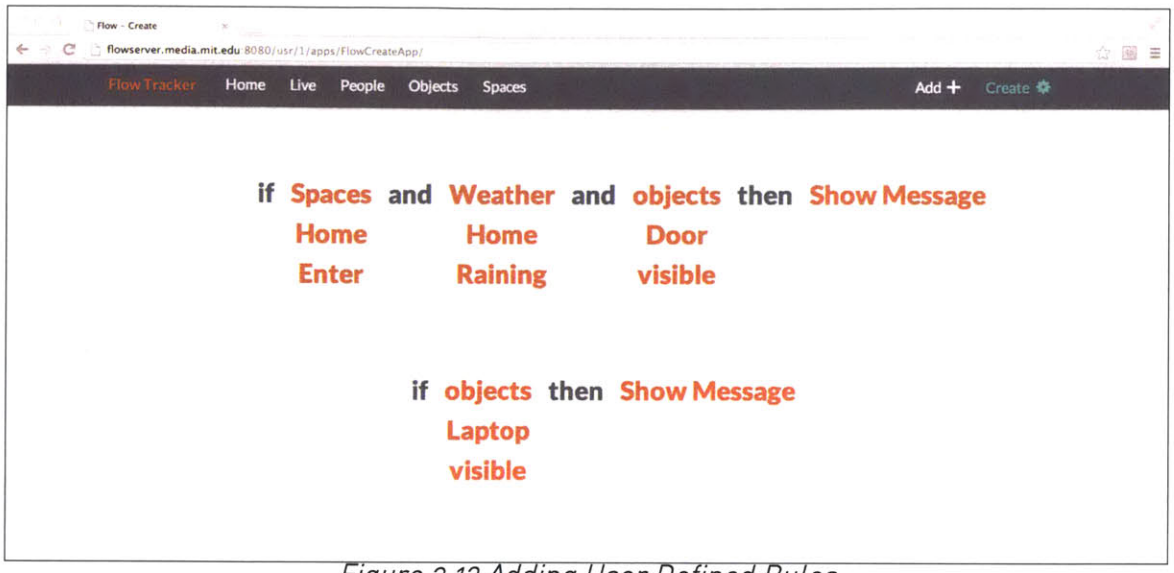

Figure 3.12 Adding User Defined Rules

OUTPUT CLIENTS

One of the primary goals of the framework is to enable users to freely move between their devices, and so a cross-compatible system that can integrate with existing hardware is required. To achieve this, the framework provides thin output clients, for a variety of platforms (Windows, Mac & Android), which are responsible for rendering common interfaces and relaying user interactions back to the central server. Figure 3.13 shows the client structure with common HTML web app elements and platform specific JavaScript/Native bridges.

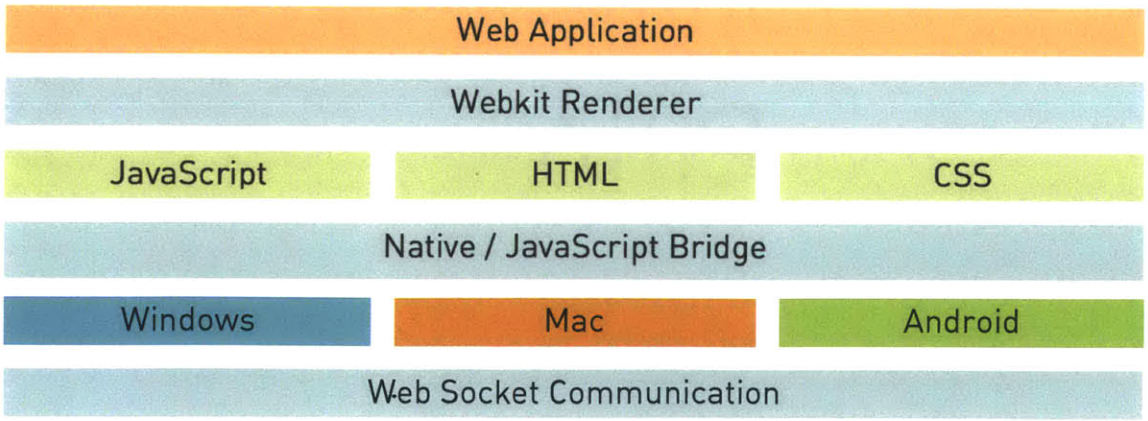

Figure 3.13 Client Architecture

These clients are installed on the user's devices, allowing them to become visually identifiable by the Flow framework and remotely controlled by the

server. To enable the device identification, the client ensures that each device remains synchronised with the server and that the current visual appearance is consistently maintained by the object tracker module. To achieve this, the client authenticates with the server and periodically transmits JPEG screenshot images via the persistent WebSocket connection. Updates occur automatically every 5 seconds, but are delayed if no visual change is detected between attempts.

The world wide web is a clear example of a successful cross-platform system that allows users to view common rendered HTML content between devices, independent of the specific hardware. This flexibility is desirable in Flow, and so each client is built around these internet technologies and wrapped with platform-specific functionality.

Figure 3.14 illustrates the layered nature of the display client. To create the impression of a native app, a "webview" webkit browser element is used, which fills the display area, rendering standard HTML content across the display. By defining the background colour of this object as transparent and simulating event click-through, a standard HTML webpage can be easily loaded by the client to create the effect of a native running application.

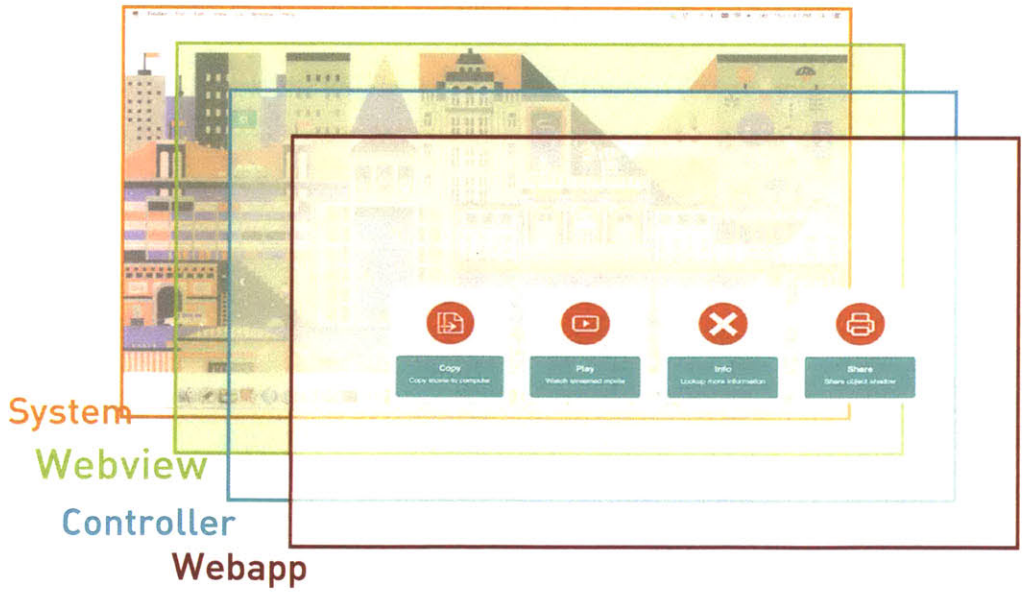

Figure 3.14 Flow Client Display Layout

To control the webview's current URL and communicate with the native

application, a Flow controller app and Javascript library is provided (Appendix 2). This controller app renders a settings overlay onto all devices and creates an embedded iframe element, which similarly expands to fill the display. Through this, Flow server messages can be received and processed, either being executed locally or forwarded to the embedded iframe for other web applications to interpret.

Communication

To communicate with a device client, JSON messages are used, which are transmitted through the persistent WebSocket connection. These messages, as illustrated below, contain 4 elements: a timestamp, message type, message id, and body.

```
{"timestamp": "1375228308",
"msg_id": "302732",
"type": 4,
"body": {
        "app_url": "http://www.robhemsley.co.uk/flow/Interact.html",
        "app_name": "uk.co.robhemsley.flow.virtualtouch"
        "body": {
                "btn_name": "play"}
        }
}
```

This generic message structure enables any serialisable object to be passed between client and server. Based on the message type, the body is processed and replaced with associated object classes. In the above example, message type 4 is used, which represents an APP_MSG for controlling the client's embedded browser. This message forwards the display to the contained app_url and, once loaded, passes this message object to the app's Javascript controller. In this example, an onscreen button marked "play" is created based on the message body contents. As the user presses this button, the web app calls back to the server via the client's Javascript libraries, wrapping a reply message object in the app_name namespace. In this manner, the Python module and embedded web app can directly communicate based on this shared name space and generic message event objects.

As well as rendering capabilities, each device client contains native functionality for common actions such as opening, saving, and deleting files.

Using the base action message type, these are intercepted by the app and executed using platform-specific code. Through this, Flow can remotely download and save data between devices, remotely executing it based on the user's interactions.

SYSTEM OVERVIEW

The Flow framework also provides a high-level system overview for the current user, as shown in Figure 3.15. This interface collates information from the individual system modules via the overridden "render_widget" method. Returned HTML from these calls are placed within display tiles, which populate the home screen.

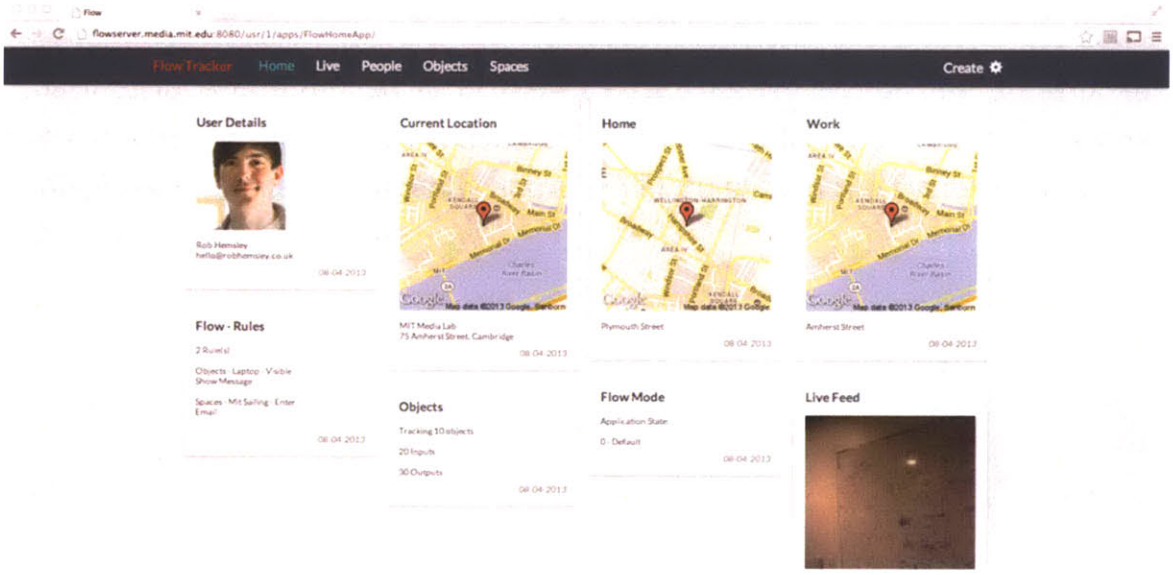

Figure 3.15 System Overview

This interface is intended to provide overall awareness on the state of the system and the user's interactions. As illustrated above, the user's live video is displayed alongside currently active IFTTT rules and the system-calculated location for home and work.

APPLICATION IMPLEMENTATION

To demonstrate the functionality of the framework and its ability to prototype reality-based interfaces, various application scenarios have been developed and tested. These explore the interactions and interfaces that can be created with the use of a wearable head-mounted camera and the observation of our everyday interactions. In the following chapter, I shall describe each scenario and their implementation details.

PHYSICAL MOBILE INTERACTION

As previously discussed, Physical Mobile Interaction focuses on the control of our environment through mobile devices, such as our cell phones. Within the framework, we demonstrate PMI by allowing users to access properties and functionality of the objects they interact with directly from their mobile devices.

Using the concept of digital shadows, each object within the framework contains associated key value pairs that define object properties and actions.

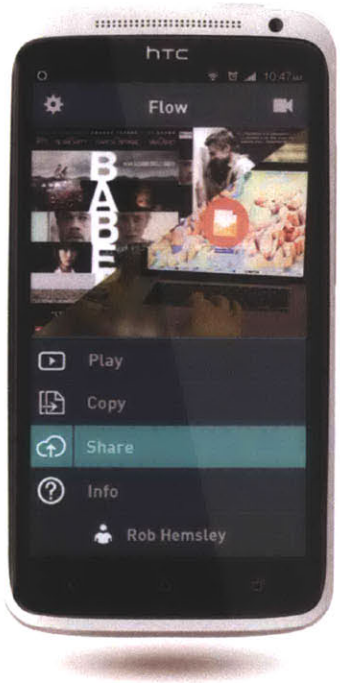

Figure 4.1 Physical Mobile Interaction – Event selection between DVD & Laptop

In this envisioned scenario, a user views an object, such as a paper document or onscreen PDF, and Flow observes this, automatically detecting the object and displaying associated shadow properties via their smart phone device. This approach ensures output and attention is maintained in the physical environment and not on a private, low resolution head mounted display. Through this display the user can make a selection and move to further devices or objects with which they wish to complete the interaction. Viewing this second object causes a suggested action to be displayed between objects, as shown in figure 4.1. In the scenario of viewing a document, the second object could be a display or printer, which would provide the associated shadow actions of "view" or "print". Based on the user's selection, the associated object metadata is sent to the output device. Within this interaction, the selection of "print" would lead to a document being produced on the printer, and the selection of "view" would cause the PDF to be downloaded and displayed on the device.

This application operates in two modes, either excepting manual action selection from the user via Tether (section 4.6) or autonomously based on the system's perceived likely intention.

IMPLEMENTATION

The application utilises the object recognition module and the associated digital shadow key value database to identify objects and retrieve associated actions. For each input action within the database, an endpoint URL is specified, which links to the object's digital representation, such as a movie file for a DVD or a PDF for a document.

Detected actions, based on the objects digital shadow are transmitted to the Tether app for visualisation, as explained in section 4.6. Users can explicitly make a selection from this application or use Flows to automatically provide an action based on the input and output object. In this scenario, Flow automatically selects a matching input and output action and presents this to the user for confirmation. When operating in a fully autonomous mode the system will silently select this action and complete this between the devices. All communication is undertaken between the client and server using WebSocket connections, with each payload containing a URL to the resource

which can be independently fetched and used in the action.

VIRTUAL TOUCHSCREEN INTERACTION

Through Flow, the user's interactions are observed from a head-mounted camera, providing a first-person perspective on their environment, objects, and actions. Through these observations, we have the opportunity to simulate interactions that the environment and objects cannot traditionally support.

In this scenario, a standard desktop display can be made to operate as a touchscreen device by observing the user's occlusion across the image, as caused by the presence of their hand. As shown in figure 4.2, this is used to allow digital shadow action selection when interacting between a physical and digital display object.

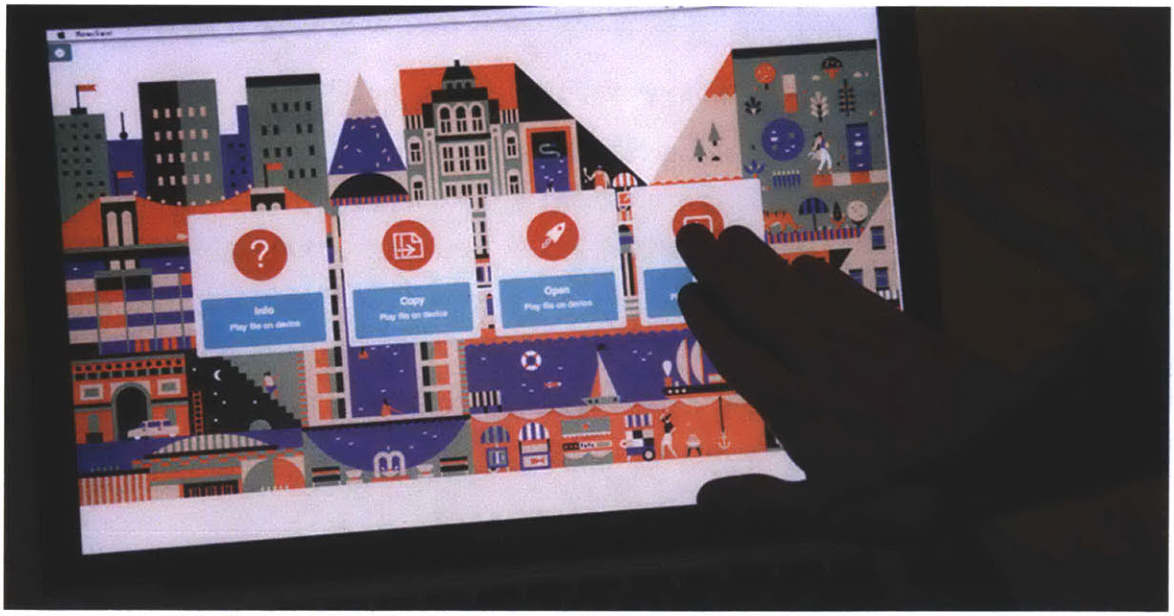

Figure 4.2 Virtual button action selection

In this scenario, the user firstly views a physical object, such as DVD, placing their hand across the object for a minimum of 3 seconds. This indicates to Flow that the object is required in an interaction, and so the digital shadow actions are gathered. If the user subsequently views a display object, the associated input/output between these objects is visually displayed to the user. To make a selection, the user can use the existing mouse pointer or can

place their hand over the desired action for a 3 second period. This approach to interaction allows users to select objects to interact with in a tangible manner by placing their hand over the desired object. This approach demonstrates how digital actions can be selected without the direct use of a GUI interface. In this example, to facilitate more complicated interactions, the display of the output device is reused to provide a common UI for user selection. In a similar manner to selecting a physical object for interaction, the user can touch the display, creating the impression of a touch screen.

IMPLEMENTATION

As outlined in section 3.10, virtual buttons can be created by tracking the occlusion of the user's hand over the tracked object's surface. By monitoring the relative number of detected features within the defined button region, it is possible to naively detect a button press and release.

This virtual interaction is provided as part of the object tracking module, and has been extended within this application to dynamically generate interest areas based on the display object's current user interface.

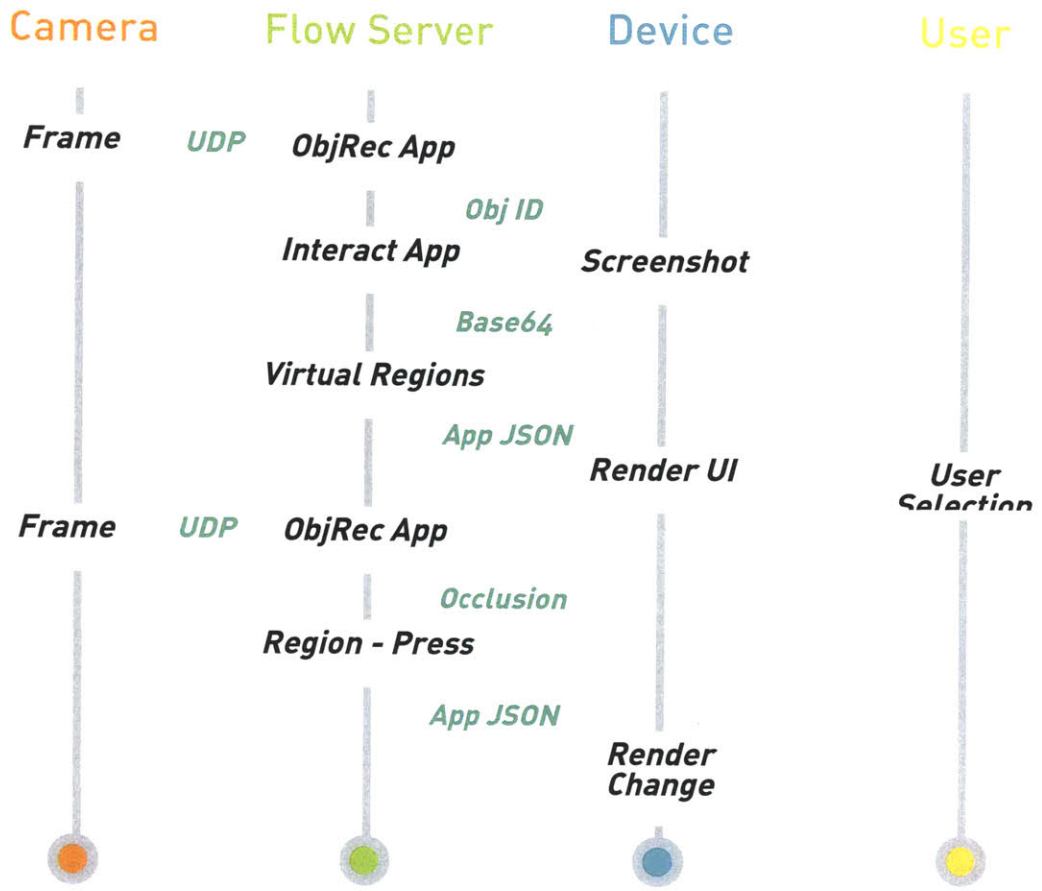

Figure 4.3 Interact app system diagram

Figure 4.3 shows the application's logic, with virtual touch regions being generated based on the object's shadow properties and the current device's display. As occlusion is detected over a region, the specified action is sent to the device client as JSON messages, where the local application updates the HTML display and executes the action function, giving the impression of a touchscreen event.

HARDWARE INTERACTION

One of the goals of the framework is to enable interaction between devices and hardware, irrespective of their platform or operating system. To demonstrate this, the following scenario integrates the platform with a low-powered, embedded system in the form of a digital lock, as shown in figure 4.4.

Figure 4.4 Digital Lock

The user can access the functionality of the lock (unlock/lock) either through the Tether application (section 4.6) or by viewing an associated action object such as a set of tagged keys. Once an action is selected, Flow sends the metadata to the object, making use of a bridging device to communicate the actions to the embedded hardware.

IMPLEMENTATION

Figure 4.5 shows the communication process that enables Flow to interact with embedded hardware devices. As the system uses persistent TCP-based WebSocket connections, lower-powered embedded hardware is unable to provide the necessary processing power and protocol stacks to enable direct communication. A bridging device is therefore used to marshal communication between the hardware and centralised server.

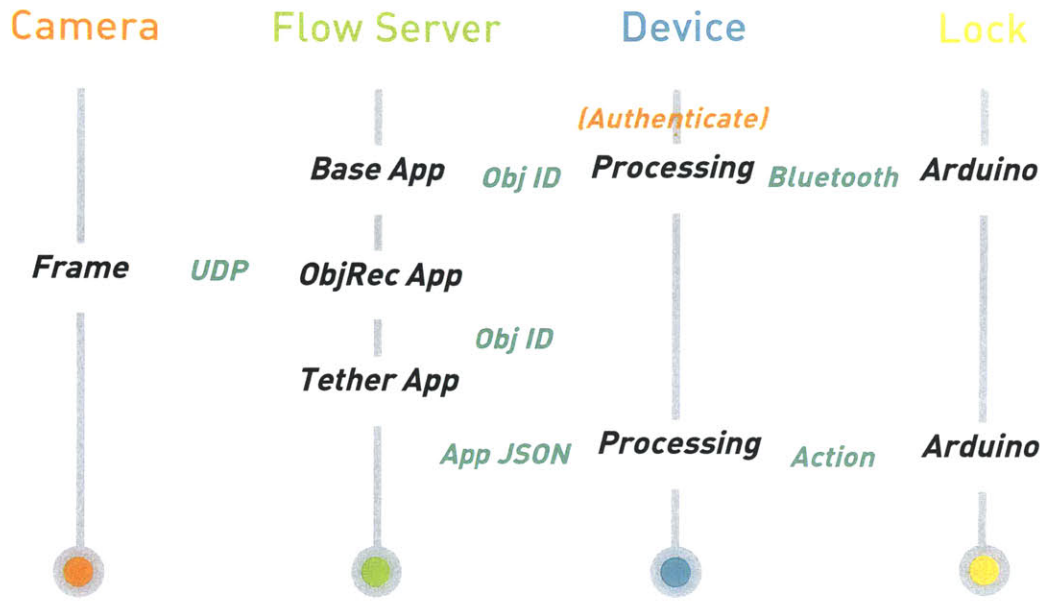

Figure 4.5 Flow hardware lock system diagram

In this scenario, the network connected lock is constructed from an Arduino [59] mini and a Bluetooth 2.1 module, which communicates with a bridging device to integrate with the framework.

A processing [17] sketch has been developed, which opens a WebSocket connection with the server and authenticates on behalf of the physical lock. As events are triggered on the framework, JSON messages are sent to the processing sketch endpoint, interpreted and forwarded to the lock via Bluetooth. To access the lock's functionality, a digital shadow is created that defines two outputs, lock and unlock, each containing an associated value for serial communication with the lock. These properties are displayed to the user via Tether, creating a remote interface for the object. A global event can also be created via the IFTTT interface, which automatically sends the unlock command as an object is viewed.

As the system uses feature detection for object recognition, a patterned key and key cover are used to provide sufficient surface detail for tracking.

TETHER

As discussed in chapter 3.5, the framework provides inbuilt object tracking that automatically identifies objects visible within the user's field of view.

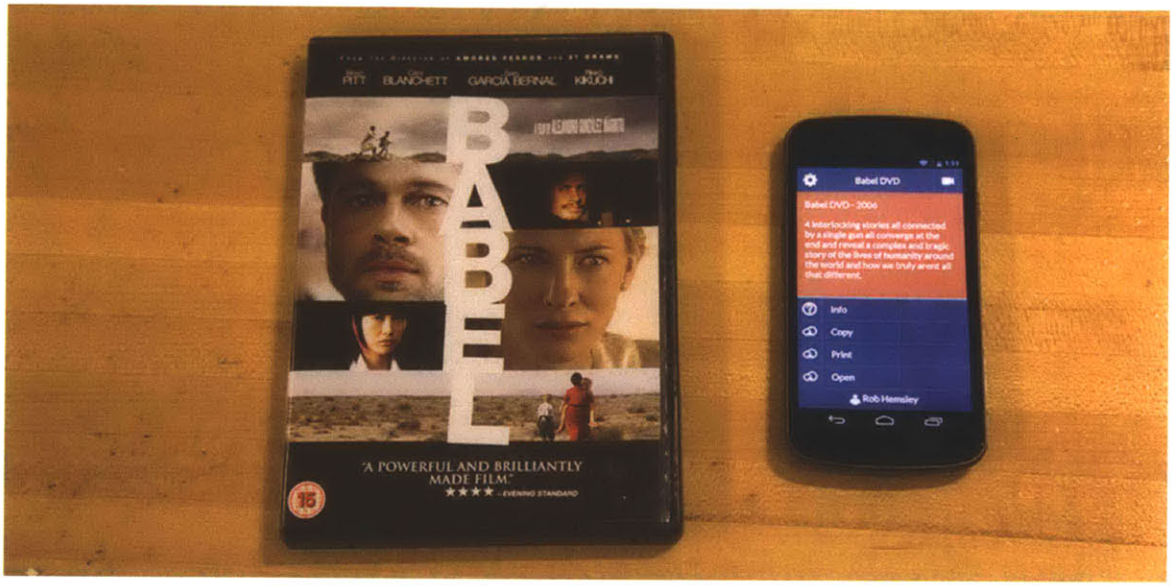

Figure 4.6 Tether Object Information

This functionality provides the opportunity for the creation of a just-in-time information system where knowledge can be accessed at the point of interaction. In our lives, we often turn to our devices to discover further relevant information based on our location, context, or objects. Using the Tether app, related information and functionality for the user's currently-observed objects can be proactively made available to the user as they explore their environment. The app operates on any of the user's local displays but defaults to the user's primary device, their cell phone. This allows users to interact normally within their environment while Flow identifies tracked objects and retrieves related information, forwarding this to their cell phone for inspection. If the user wishes to access further information or functionality, the FlowServices app, as shown in figure 4.6, can be used to access the object's digital shadow properties and functionality via this web app.

IMPLEMENTATION

To facilitate this interaction, the Tether app provides a Python Flow class and associated Javascript and HTML render. Once activated, the

"uk.co.robhemsley.flow.objrec.visible" event is subscribed to and called as objects are viewed. For each object, the digital shadow key value database is queried for input/output options and the object's description. These values are concatenated into a JSON dictionary and sent as an app message object from the server to the user's listed primary device, as shown in figure 4.7.

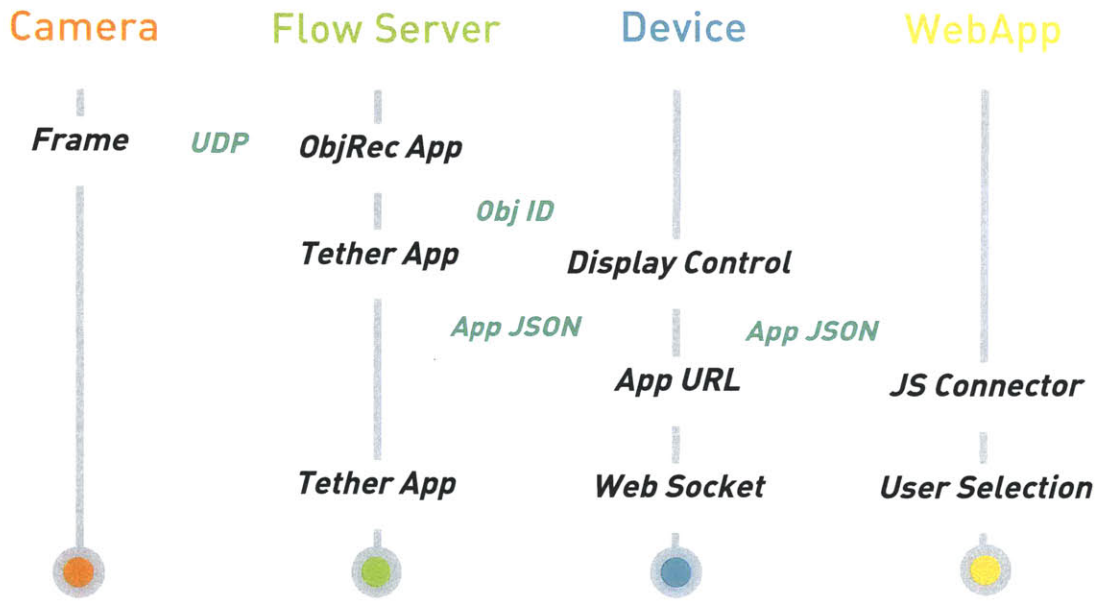

Figure 4.7 Tether App System Diagram

The FlowServices Android app receives messages via the WebSocket connection and navigates the embedded webview to the contained application renderer URL. As a webapp, the loaded HTML page communicates via the Flow Javascript libraries and receives the JSON payload once initialised. The local renderer dynamically generates the interface shown in figure 4.6, from which a user selection can be made. Each interaction on the webapp is sent back to the server, where it is interpreted before a response action is returned to the app. In this manner, the central server maintains the state of the remote app, populating and clearing the interface as actions are received. The tether app also provides access to the user's current video feed via the MJPEG stream supported by the Live module.

FOLLOW

Within our multi-device environment, we have become accustomed to moving between devices and operating systems, and have adapted to learning by rote

how to achieve the same task across many systems. In an environment supported by Flow, we can create interactions where the user can seamlessly move between their devices, with their services and notifications following with them.

The Follow app demonstrates the potential of this connected environment by allowing incoming email notifications and messages to follow the user between displays. As email messages are received, Follow extracts the message body and displays this to the user on the nearest device. Based on their distance from this display, the granularity of information also varies, changing from a high level email notification to the full message contents as they approach the device.

Figure 4.8 Follow App – Proximal aware email

In this scenario, a custom "Follow" application was created, which provides both a server-side Python module and supporting HTML and Javascript interface for display across scales on the user's devices.

IMPLEMENTATION

As shown in figure 4.8, once the module is loaded within the framework, it periodically connects to a user-defined POP3 mail box and checks for available messages. Once detected and received, the app extracts the from, subject, and body of the email message and saves this locally within the

user's temporary storage, ready for display.

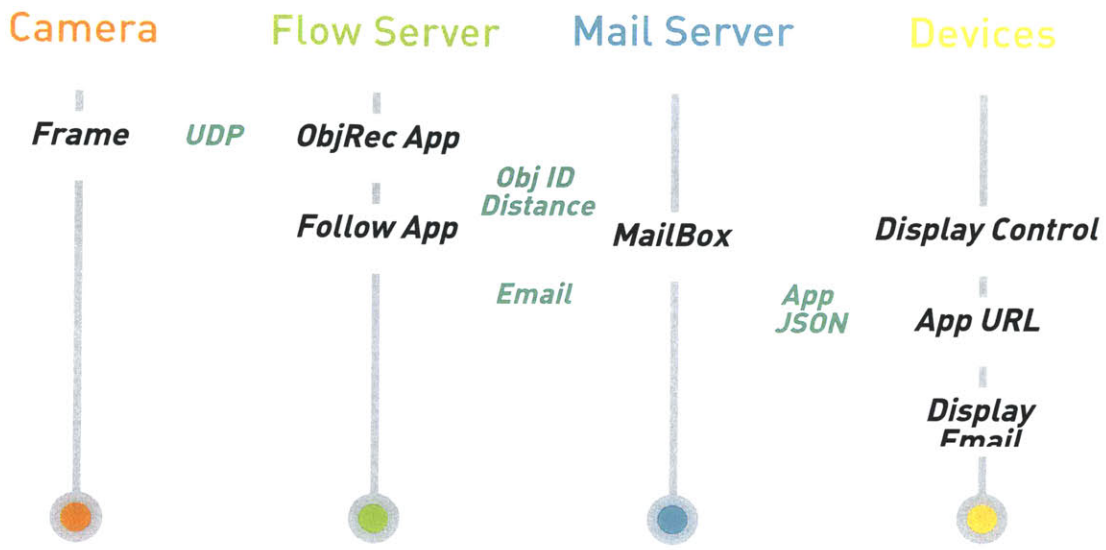

Figure 4.8 Follow app system diagram

As the app subscribes to the "uk.co.robhemsley.flow.objrec.visible" event, "Follow" is called as tracked objects become visible. If the observed object has display capabilities, then "Follow" sends the contents of the email and app, rendering the URL as a JSON message directly to the device. The end device controller then navigates to the app render URL and, once loaded, further processes the corresponding JSON message. In the Follow app, this causes the received email to be displayed on the user's nearest device, reusing its display capabilities and gaining the user's attention.

The follow app also varies the granularity of this displayed information based on the user's relative distance from the tracked surface. In this scenario, as the user steps closer to the display, the message changes from a high level summary to displaying the most recent message in its entirety. As previously described in 3.9, the distance measurement is calculated based on the known real-world dimensions of the tracked object plane and the computed homography for each frame. As the user's distance from an object changes between the defined states, JSON messages are transmitted to the end device, updating the display.

Through this application, we demonstrate the use of proxemics [6] interaction techniques, where content adjusts based on the user's relative distance.

UPDATE

The "Update" app looks at bringing digital functionality into our environment by associating digital actions to our existing behaviour. Using this app, when a user makes a visual update to a whiteboard or notepad, the action is detected and the changes are captured ready for use with other services such as Evernote or Email. In this scenario, an email with the detected changes is sent to a defined collaborator, ensuring they remain updated on the local updates made in the physical environment. Figure 4.9 shows the detection of an update made to a note pad.

Figure 4.9 Update App – Feature detection between updated writing

IMPLEMENTATION

Figure 4.10 shows a system diagram for the "Update" application and the detection process for identifying key point feature change. The app subscribes to the "uk.co.robhemsley.flow.objrec" event, which is called for each frame in which an object is visible. On detection, this event is passing a ThreadQueue object containing a JSON message that describes the detected object in terms of the matched key points.

Using the matplotlib Python module, a count of the total number of features present within the object is made, creating a baseline from which visual changes can be detected.

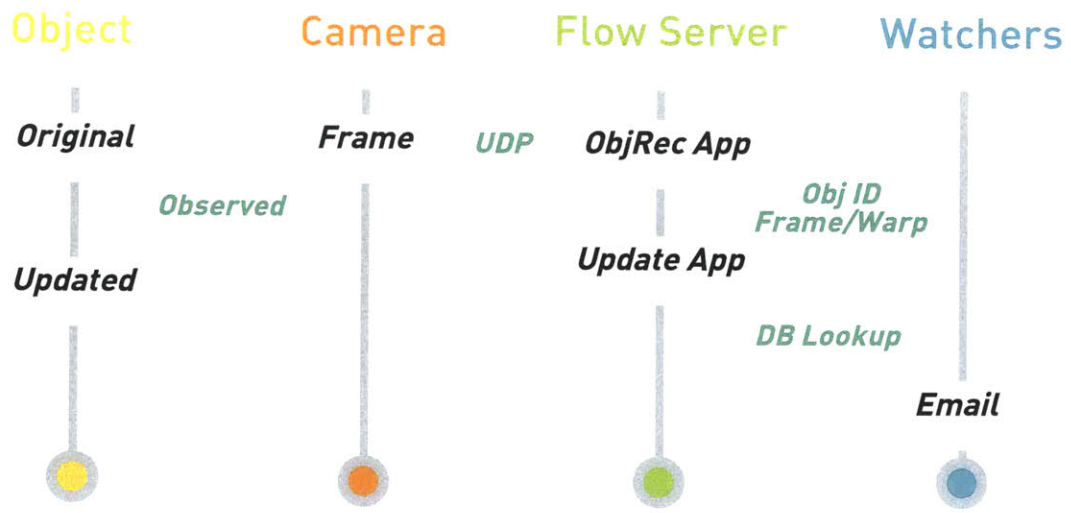

Figure 4.10 Update app system diagram

As updates are made, this process is repeated until there is at least a 25 percent increase in the number of visible features. At this threshold, the event "uk.co.robhemsley.flow.updateapp.update" is fired with a reference to the original and the perspective-warped frame in which the update was detected. In this scenario, the update causes a local key value lookup on the object's database for the "email_watch" field containing the contact details of those 'watching' the object. These addresses are then used with the email utility to send an update email containing the perspective-warped image as observed by Flow.

To reduce false positives created by incorrectly calculated homographies and occlusions, a moving average is calculated over 30 frames, creating a 3 second buffer.

Further work has also been conducted, making use of ICR and OCR software to directly convert detected image change into digital text. This processes was tested using the OCRopus [7] library but, due to the current image quality of the head-mounted camera, this has proved to be ineffective.

OBJECT PINNING

To demonstrate the location services made available by Flow, an object pinning application has been created. This app enables users to virtually tether an object to its current physical location, allowing actions to be

triggered when the object is observed to have moved.

IMPLEMENTATION

The pinning application, as outlined in figure 4.12, makes use of the user's cell phone location services in order to reference where an object currently being observed is located. This functionality is built into the Android FlowServices app, which periodically sends location updates to the server, ensuring that the user's location is centrally maintained. To reduce the effects of this on the device's battery, the location is determined based on the nearest WiFi router mac addresses using Google's reverse geolocation service [21]. This approach provides an accuracy of up to 20m which varies based on how tightly packed the available networks are.

The pin application subscribes to the "uk.co.robhemsley.flow.objrec.visible" event, allowing the object's location database field to be updated when an object becomes available. In this scenario, the pin app is integrated into the tether application, allowing a user to view an object and select "Pin" from the actions as displayed on the user's primary device. This button selection event is subscribed to by the pin app, and so the object's current location is saved as a key value to the object's shadow database entry.

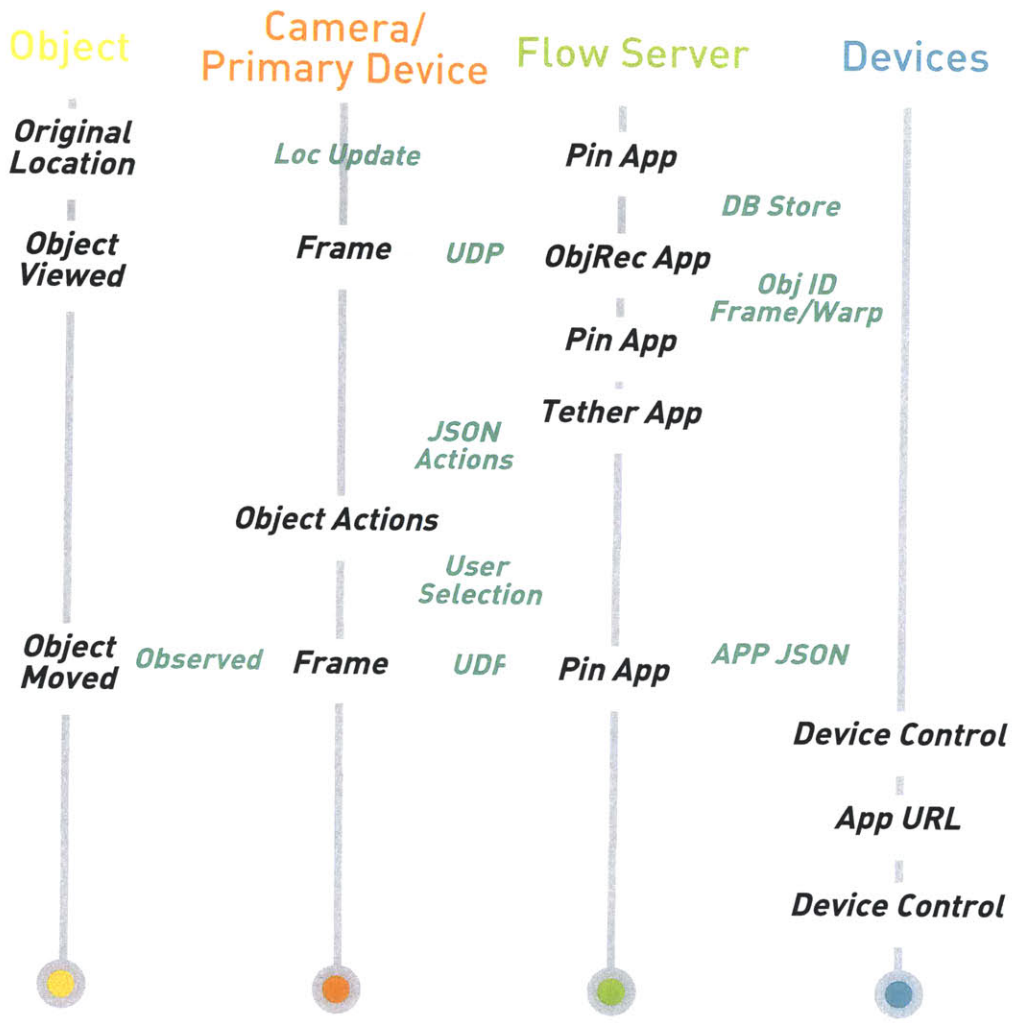

Figure 4.12 Pin app system design

As an object is viewed and detected by Flow, the app checks for a pin key value entry and compares the locations' distances using the PostGIS [42] extension. If the object has moved more then 100m, then the associated event is fired. In this scenario, an onscreen message displaying the object's current location is then displayed on the object owner's nearest display. As Flow can support multiple users, it is assumed that other users of the system update device locations as they interact with or pass tracked objects.

EVALUATION

To successfully evaluate the framework, technical and user-based interaction evaluations have been undertaken. Through this, insight has been gained regarding the interactions facilitated by Flow, as well as the technical challenges associated with developing such a computer vision system.

TECHNICAL

Throughout the development process, a modular approach has been followed, producing standalone applications that have been tested through a series of automated unit tests. With the completed framework, benchmark tests can be undertaken, validating the system's design and implementation.

Object tracking

For object identification, a target image library is used, to which incoming frames are compared and matched. With each additional target image, an average overhead of 34 milliseconds is observed to be added to the base matching time. This overhead varies based on the image size and the number of features. In this benchmark, images are constrained to 640x480 pixels and a maximum of 500 ORB features for each descriptor. To ensure the framework doesn't become unresponsive, less than 10 target images are used, ensuring the base computation time doesn't exceed the incoming frame rate of 10 fps. To account for this delay, the system concurrently executes a thread pool of 20 ORB detectors, to which each frame may be passed. The following figure shows the computation time for simultaneously tracking target images within the same frame.

	Number of visible targets					
	0	1	2	3	4	5
Time (ms)	90	102	121	138	153	169

Figure 4.13 Target Tracking Time

As shown, a small overhead of 15.8ms is added to the frame for each additional target image visible within the frame. Although further targets may be visible, the current cap of 1000 ORB features per frame prevents sufficient

matching across all targets, causing the system to drop all matches when more than 5 targets are simultaneously tracked.

Based on these results, the Flow framework is shown to support multiple object tracking, processing frames in a computationally acceptable time of 169 milliseconds. Although multiple object tracking isn't utilised within current applications, further scalability is demonstrated and the bottleneck in maximum ORB features identified.

Tracking Distance

To evaluate the system's robustness in detecting and tracking target images, a series of tests were undertaken, identifying targets from within various everyday environments. As shown in figure 4.14, tests were undertaken over various distances using a single target image, measuring 13x19cm, with 478 detected ORB features.

	Distance (cm)							
Environment	3	4	5	80	90	100	120	130
Library - Daytime								
Office - Daytime								
Office - Evening								
Outdoors - Daytime								

Figure 4.14 Distance tracking

The system successfully detected and matched the target within all environments with a maximum detection range between 4cm and 120cm, as observed within an indoor office environment with consistent lighting conditions. Due to the size of the target image and low feature density, the system cannot make accurate matches below 4cm. This clear cut-off is also a function of the fixed focal range of the head-mounted camera.

With distances greater than 120cm, accurate key points are no longer matched. This is partly due to the fixed image size of 640 x 480 pixels, which prevents detailed feature identification at this distance. The ORB detector has also been capped at 1000 features, in order to reduce the detection and matching times. This cap is responsible for the reduced distance observed in the library environment, as the ORB algorithm prioritises the corner features

of the bookshelves over the target image. This reduces the number of possible key points for matching, causing the inliers to drop below the required threshold.

These tests demonstrate Flow's appropriateness for close range interaction within our environment, where the user is explicitly engaging with tracked objects. Based on these results, distance is affected by lighting conditions, but not sufficiently to render the system unusable, as tracking is maintained to a distance of 90cm throughout these tests.

Communication

Flow utilises a variety of protocols, on both UDP and TCP transport layers, to communicate between the user's head-mounted camera, the server, and associated client devices. As a real-time system, the response time between the frame capture and output must be minimised, preventing lag and the impression of an unresponsive system.

The round trip time for capturing, processing, and communicating with the Tether app is shown below. This test was undertaken on a frame containing a single target image.

Frame Captured	Frame Sent	Frame Received	Frame Processed	Event Sent	Displayed by Tether
0.30	0.30	0.08	0.70	0.20	4.32

Figure 4.15 Round Trip Time (seconds) - Single Frame

As shown, the total response time is 5.9 seconds between frame capture and display. The majority of this time is observed in the communication and display portion of the cycle, consuming a total of 4.32 seconds. This server-to-device communication is undertaken through a TCP WebSocket connection, with data taking less than 30ms for transmission. Based on this, we can assume the delay is due to the HTML and JavaScript rendering and the loading of external resources. As the app contains an embedded browser, the display is dependant on the load time of the render app. Rerunning this test with the app preloaded, we observe this delay reducing to 1.43 seconds. This load delay can be reduced in the future iterations by caching or preloading apps prior to being activated.

From this test, we show that the user input frames and server processing operate with a 0.78 second delay, which is sufficient for non-AR based applications. This test highlights the tradeoff between providing flexible web-based rendered apps and a responsive user interface.

USER EVALUATION

As the framework focuses on human-computer interaction, an observational user study has been undertaken to evaluate the available interaction techniques. This study observes and times users completing tasks with and without the support of Flow.

To provide some background on the framework, users are first shown a demonstration of the DVD viewing example (section 4.1) and the interaction principles that Flow supports.

DIGITAL SHADOW

The first task explores the use of digital shadows and the ability to access associated information and functionality. Users are presented with a physical document and asked to make a copy, printing this on the nearest printer. This test was undertaken in an office environment with a laptop computer, paper documents, and a printer. This task was timed and undertaken twice, with and without the support of Flow.

The average completion time for this task 62.4 seconds without Flow and 23.7 seconds with its support. In this task, as the physical document has an associated digital shadow, users can look at the object, selecting the print action from their cell phone, and then physically move to the printer, where the document will be seamlessly sent and printed. In the non-Flow supported mode, users had to find the document on the laptop, using an internet search term to identify and download the document. Next, the users must open this on an appropriate application and then determine which printer, based on their name, was nearest. This process added a clear overhead in producing the document, which is reflected in the average times.

The same task was also repeated, with the system automatically selecting the print action without user interaction. As expected, the average time to complete the task fell to 17.7 seconds in this mode.

Based on the previous user feedback, a third mode was also provided, which automatically made recommendations for the user's action but allowed the user to manually activate it. In this mode, the user took an average of 22.4 seconds to complete the task, only a fraction shorter than manually selecting the action.

Based on these modes, users were asked to comment and rank the interaction styles they preferred. In this, users indicated a preference for auto-prediction with manual selection over the fully autonomous approach. User 3 indicated that this addressed their previous concerns on accidental activation during their everyday use and gave them the opportunity the check that the system's decision matched their intended action.

In this test, user 2 was familiar with the presented document and so was observed only briefly glancing at it before attempting to view the object's properties on the mobile device. This brief object interaction wasn't sufficient for the system to recognise the object, and so the user was left confused whether the system was responding. In interactions where the user already has existing knowledge, their object dwell time is observed to be reduced, causing the appearance to be missed by the system. This behaviour highlights the current limitations with detection speeds and system frame rates, which cause the system to miss these micro interactions. This is exacerbated by the web application's update times, which further adds to the impression of the system being unresponsive.

User 4 commented that within this interaction there were no details regarding the printing process, such as paper size, quantity, and colour, leaving them apprehensive to complete the action. In this, users are mapping their existing expectations of print dialogs onto the system, which at present are not supported, and so this conflict is observed.

VISUAL UPDATE

The second user test demonstrates Flow's ability to recognise objects and their visual state changes. In this, users were presented with a notepad or whiteboard and asked to draw or write a message before sending this to another user via email. This was conducted within an office environment with

and without the support of Flow. In the initial user tests the notebook proved to be a challenging target for the system to consistently detect and compute a homography. This lead to false positives being created as key features outside the objects area were included causing a visual update event to be fired. This behaviour demonstrates the need for tracked objects to contain a sufficiently large number of features and the viewing camera to support a higher image resolution. To overcome this the user study continued with a large white board surface containing partial illustrations for use as the tracking surface.

Users drew on the surface with the system detecting visual changes which were automatically emailed to a predefined user email address. To test the time required to undertake this action users were asked to verbally indicate when they finished drawing on the board at which point the timer was started. In this it took the system on average 13.2 seconds to detect, extract, email and receive the final update image on the users account. As messages were distributed through email, this time is heavily dependent on the image size and network traffic. Users were asked to complete the same task using the provided cell phone and the android OS, manually taking a photo and sending it via the local email application. In this depending on user familiarity with Android it took between 14.7 and 51.6 seconds to compete the task with an average of 24.7 seconds. This demonstrates Flows ability to effectively detect and inform users of visual changes to an objects surface without direct user interaction.

When observing this task it became clear that the required 25% increase in visual features is ineffective in detecting when the user has finished and wishes to send an update image. Most of the users' final update images failed to display all the changes the users had made to the board. This was most obvious with user three who used a large area of the board, meaning the camera was unable to capture in a single frame the full extent of the surface. When using the smaller note pad this was not observed as an issue.

Although Flow is quicker at updating users of visual changes, when further discussion was undertaken on the produced images it was felt the quality was too low to effectively interpret the data. When asked to indicate which images most clearly represented their updates they felt their manually captured

images were more informative, but it was acknowledged that this came at the expense of their active involvement.

GENERAL OBSERVATIONS

Partly due to the lack of feedback, users were observed to heavily focus on the mobile device, glancing at the environment and the mobile application to determine if an object had additional functionality. This level of active engagement in the mobile device goes against the goal of reality-based interaction, as users are focusing out of the reality around them. This may be due to the environment and the user's curiosity of the available functionality and lack of hands-free feedback.

FUTURE WORK

This work presents a framework to support the creation of reality-based interfaces that allow users to interact between their devices and physical environment. This has been achieved through the use of computer vision techniques to interpret the world from the user's first-person perspective and output metadata on the observed environment and interactions. This approach has provided increased machine understanding on the user's behaviour, based on the objects, spaces, and people with which they interact.

To achieve this, a considerable amount of time has been invested in the architectural development and overcoming current barriers to inter-device communication. With the completed framework, there is now the opportunity to develop new interaction applications, which go beyond those currently demonstrated. The following section outlines potential future work in terms of both the user interaction experience and technical implementation.

INTERACTION IMPROVEMENTS

The current applications are developed from a single user's perspective, with their sole interactions affecting the system's behaviour. In an multi-user environment, each user's perspective can act as input, enabling new proxemic interactions. For example, customised content & actions could be displayed based on the number of viewers observing an object. This could be used for security, failsafe applications where double authentication is required, or in the creation of adaptive displays that prevent shoulder surfing or multiple users viewing content.

In terms of user feedback, this is provided by the user's cell phone through the Tether app, or via reusing existing display devices. Alternative approaches should be explored that integrate techniques for haptic and audio feedback. This helps support the creation of hands-free interactions, where the user's cell phone is no longer the viewpoint into the augmented environment. As the user is assumed to carry a mobile device, a language of vibration patterns could be created which could be felt by the user, with the phone remaining in their pocket. This approach has been demonstrated by John Kestner within the Proverbial Wallets project [7].

As the system uses a wearable head-mounted camera, there are clear privacy concerns that are raised regarding recording third parties and how information is stored and used in the future. Through the goal of transparency and the "People" module, some privacy-preserving steps have been taken, but further work is required. Within information theory, the concept of forgetting machines [18] and the automatic degrading of data over time are useful in providing a bound lifetime to the collected data. This approach should be applied to the databases by altering stored point locations to broader ranges over time.

The system defaults to the user's primary device for direct interaction and notification via the tether application. This approach was designed to fit in with users' current interaction model of mobile devices, but alternative approaches should be developed and evaluated. For example, Google Glass [20] and APX Labs [32] both provide head-mounted displays, which could be evaluated for in-field interfaces. This would once again help create a hands-free interface, but would require users to focus on a peripheral display, removing them from reality. Another potential solution is the creation of a graphic language that communicates the existence of Flow-connected objects and devices without the need for a digital display. A similar approach has been successfully explored for touch devices by Arnall Timo [4], who outlines a series of pictorial representations for our touch-based interactions.

As a cross-platform framework, a unified interface and interaction design rationale has been followed throughout. The final design demonstrates a single approach, and so an exploration of user interface design techniques and the effects this has on creating a multi-model interaction environment should be evaluated.

TECHNICAL IMPROVEMENTS

The following outlines potential technical improvements for the framework.

ALGORITHM EFFICIENCY

Current implementations have been constructed around the Python programming language with bindings to C++ implementations of OpenCV. Future systems should avoid the challenges of Python and its associated threading locks by developing with a compiled language that takes advantage

of recent GPU-based feature tracking algorithms. This will bring noticeable speed improvements and reduce the current bottleneck exhibited by the Python object recognition module.

ENVIRONMENT MAPPING
Computer vision techniques used within autonomous vehicles navigation for environment mapping without prior knowledge could provide increased localisation and contextual awareness within Flow. Techniques such as the Simultaneous Localisation and Mapping [12] (SLAM) and Parallel Tracking and Mapping [30] (PTAM) algorithms utilise feature detection to build an interconnected feature map for an environment. Using this within the framework would provide increased localisation, as a continuous map can be developed, providing stabilised tracking if extended to an AR environment.

SENSOR FUSION
The current framework uses computer vision techniques as the only source of input, generating events based on each application's interpretation of the incoming frames. This method is naturally prone to error due to lighting changes, occlusion, and image quality, and so future applications should be developed utilising existing hardware sensors to improve accuracy. This creates a sensor fusion environment, where wearable devices such as the pebble watch [39] and WristQue [37] could provide sensor data to the framework, ensuring new and more accurate forms of interaction.

DISCOVERY
Integration of ad-hoc discovery protocols could provide an infrastructureless implementation, allowing digital objects to become self-describing to the users around them. In this decentralised approach, each object could provide the current visual appearance and functionality made available by the devices within its proximity. This would remove the requirement for continued network connectivity and reduce the scalability bottleneck currently present with a centralised system.

OBJECT TRACKING
As highlighted in the user feedback, the system relies on objects being visually unique or spatially separated for system identification. Further work on reducing this search space should therefore be explored, identifying objects through other means, such as RSSI, WPS, or through synchronised

visual change such as the rapid display of visual marks, like QR codes.

THIRD PARTY INTEGRATION

Events provided by the IFTTT module enable external communication with email and other services via HTTP GET requests. This support is currently limited and should be expanded to enable authentication with other APIs, such as Evernote and Dropbox. This would further create a unifying framework between our devices, physical environment, and existing services.

CONCLUSIONS

The thesis demonstrates a framework for the creation of reality-based interfaces, through the use of emerging computer vision techniques and a wearable head-mounted camera. The aim of this work is to demonstrate the future interaction potentials that such passive observational approaches provide. Through this, we can take a step closer to creating environments where users can freely move between their multi-platform devices and physical environment.

One of the goals of this work was to bring digital functionality to the user's environment and refocus their attention away from flat digital user interfaces and back into their physical reality. Through the supported object recognition and virtual surfaces provided by Flow, I have demonstrated that digital shadows can be created, bringing associated functionality and information to the physical objects and environment in which we interact. Taking this further, multiple interaction techniques have been presented, exploring the reuse of our existing mobile devices and displays as well as our physical interactions as cues for Flow. In this, virtual buttons have been demonstrated, allowing actions to be triggered from non-digital objects, as observed by the system.

It is my belief that, as we increase the number of digital devices and embedded systems, we must start to provide unified interfaces that prevent us from becoming overwhelmed by their diverging forms of interaction. Frameworks such as this, which focus on multi-device, cross-platform interactions, illustrate a potential future where users can create applications that freely move with them and enable users to customise how their actions affect their digital environment.

I hope that this framework can be utilised by future researchers in the exploration of dissolved, pervasive environments, and help further bring machine understanding and digital interactions to the physical environment in which we shall continue to live.

REFERENCES

[1] G. D. Abowd and E. D. Mynatt, "Charting past, present, and future research in ubiquitous computing," *ACM Transactions on Computer-Human Interaction*, vol. 7, no. 1, pp. 29–58, Mar. 2000.

[2] M. Agrawal, K. Konolige, and M. Blas, "Censure: Center surround extremas for realtime feature detection and matching," *Computer Vision–ECCV 2008*, pp. 102–115, 2008.

[3] a. Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: Fast Retina Keypoint," *Computer Vision and ...*, pp. 510–517, Jun. 2012.

[4] T. Arnall, "A graphic language for touch-based interactions," *Proceedings of Mobile Interaction with the Real World ...*, 2006.

[5] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *Computer Vision–ECCV 2006*, 2006.

[6] S. Boring, D. Baur, and A. Butz, "Touch projector: mobile interaction through video," *Proceedings of the 28th ...*, 2010.

[7] T. Breuel, "OCRopus," 2007. [Online]. Available: http://code.google.com/p/ocropus/. [Accessed: 06-Apr-2013].

[8] G. Broll, M. Haarländer, M. Paolucci, M. Wagner, E. Rukzio, and A. Schmidt, "Collect & Drop : A Technique for Physical Mobile Interaction," pp. 103–106.

[9] L. Buechley and M. Eisenberg, "The LilyPad Arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education," *... the SIGCHI conference on ...*, pp. 423–432, 2008.

[10] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," *Computer Vision–ECCV 2010*, 2010.

[11] M. Donoser and H. Bischof, "Efficient Maximally Stable Extremal Region (MSER) Tracking," *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1 (CVPR'06)*, vol. 1, pp. 553–560, 2006.

[12] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *Robotics & Automation Magazine, ...*, 2006.

[13] N. Eagle and A. (Sandy) Pentland, "Reality mining: sensing complex social systems," *Personal and Ubiquitous Computing*, vol. 10, no. 4, pp. 255–268, Nov. 2005.

[14] A. Feldman, E. Tapia, and S. Sadi, "ReachMedia: On-the-move interaction with everyday objects," *Wearable Computers, ...*, 2005.

[15] R. Fielding, "Architectural styles and the design of network-based software architectures," 2000.

[16] R. Foster, "Manchester Encoding." .

[17] B. Fry, "Processing." .

[18] A. Galloway, "Forgetting Machines," 2004. [Online]. Available: http://www.purselipsquarejaw.org/2003_12_01_blogger_archives.php. [Accessed: 08-Apr-2007].

[19] Glyph Lefkowitz, "Twisted," 2002. [Online]. Available: http://twistedmatrix.com/trac/. [Accessed: 05-Oct-2012].

[20] Google, "Google Glass," 2012. .

[21] Google, "Google Maps Geolocation API." [Online]. Available: https://developers.google.com/maps/documentation/business/geolocation/. [Accessed: 21-May-2012].

[22] D. Hausen, "Reducing Cognitive Load by Using the Periphery of our Attention."

[23] R. Hemsley, Y. Bando, and H. Holtzman, "Mobile Peer2Peer," 2012. [Online]. Available: http://shair.media.mit.edu. [Accessed: 22-May-2013].

[24] I. Hickson, "The Web Sockets API," *W3C Working Draft*, 2009. [Online]. Available: http://www.w3.org/TR/2009/WD-websockets-20091029/. [Accessed: 10-Aug-2012].

[25] J. Hunter, "Matplotlib: Python plotting," 2008. [Online]. Available: http://matplotlib.org. [Accessed: 24-Jan-2013].

[26] G. Interaction, "OnObject: Programming of Physical Objects for Gestural Interaction," 2010.

[27] H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto, and K. Tachibana, "Virtual object manipulation on a table-top AR environment," *Proceedings IEEE and ACM International Symposium on Augmented Reality (ISAR 2000)*, pp. 111–119.

[28] J. Kestner and D. Leithinger, "Proverbial wallet: tangible interface for financial awareness," ... *Conference on Tangible ...*, 2009.

[29] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic, "People , Places , Things : Web Presence for the Real World People , Places , Things : Web Presence for the Real World," 2001.

[30] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 1–10, Nov. 2007.

[31] 13th Lab, "PointCloud API," 2012. [Online]. Available: http://pointcloud.io. [Accessed: 03-May-2012].

[32] A. Labs, "APX Labs." [Online]. Available: http://www.apx-labs.com. [Accessed: 05-Dec-2012].

[33] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary Robust invariant scalable keypoints," *2011 International Conference on Computer Vision*, pp. 2548–2555, Nov. 2011.

[34] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[35] J. Mankoff, A. K. Dey, G. Hsieh, J. Kientz, S. Lederer, and M. Ames, "Heuristic evaluation of ambient displays," *Proceedings of the conference on Human factors in computing systems - CHI '03*, no. 5, p. 169, 2003.

[36] S. Mann and J. Fung, "Designing EyeTap digital eyeglasses for continuous lifelong capture and sharing of personal experiences," *Alt. Chi, Proc. CHI ...*, 2005.

[37] B. Mayton, "WristQue: a personal sensor wristband for smart infrastructure and control," no. 2008, 2012.

[38] M. Muja and D. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration.," *VISAPP (1)*, 2009.

[39] Pebble Technology, "Pebble Watch," 2012. [Online]. Available: http://www.getpebble.com.

[40] H. Performance, N. Program, D. Memories, P. Features, and S. M. Features, "Bytes In-System Programmable Flash ATtiny25 / V / ATtiny45 / V / ATtiny85 / V," 2013.

[41] R. Pi, "An ARM GNU/Linux box for $25," *Take a byte*, 2012.

[42] M. S. Refractions Research, Paul Ramsey, Dave Blasby, Kevin Neufeld, Mark Cave-Ayland, Regina Obe, Sandro Santilli, Olivier Courtin, Nicklas Avén, Bborie Park, Pierre Racine, Jeff Lounsbury, Chris Hodgson, Jorge Arévalo, Mateusz Loskot, Norman Vine, Carl Anders, "PostGIS Spatial Database," 2005. .

[43] J. Rekimoto and K. Nagao, "The world through the computer: Computer augmented interaction with real world environments," *Proceedings of the 8th annual ACM symposium ...*, pp. 29–36, 1995.

[44] B. Rhodes, "The wearable remembrance agent: A system for augmented memory," *Personal Technologies*, pp. 218–224, 1997.

[45] B. Rhodes and P. Maes, "Just-in-time information retrieval agents," *IBM Systems journal*, vol. 39, no. 3.4, pp. 685–704, 2000.

[46] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking," *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, pp. 1508–1515 Vol. 2, 2005.

[47] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," pp. 1–14.

[48] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *2011 International Conference on Computer Vision*, pp. 2564–2571, Nov. 2011.

[49] E. Rukzio, "Physical mobile interactions: Mobile devices as pervasive mediators for interactions with the real world," 2006.

[50] S. Josefsson, "RFC 3548-the base 16, base32, and base64 data encodings," *The Internet Society*, 2003. .

[51] J. Shi, "Good Features to Track," no. June, 1994.

[52] M. Slaney and M. Casey, "Locality-sensitive hashing for finding nearest neighbors," *Signal Processing Magazine, IEEE*, no. March, pp. 128–131, 2008.

[53] H. Steiner, "Firmata: Towards making microcontrollers act like extensions of the computer," *New Interfaces for Musical Expression*, pp. 125–130, 2009.

[54] The Open Group, "lp Utility," 2004. [Online]. Available: 07/10/2012.

[55] B. Ullmer and H. Ishii, "Emerging frameworks for tangible user interfaces," *IBM systems journal*, vol. 39, no. 3, pp. 1–15, 2000.

[56] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," ... *Vision and Pattern Recognition, 2001. CVPR ...*, vol. 1, pp. I–511–I–518, 2001.

[57] M. Weiser, "The world is not a desktop," *interactions*, 1994.

[58] P. Wellner, "Interacting with paper on the Digital Desk," *Communications of the ACM*, 1993.

[59] "Arduino," 2005. [Online]. Available: http://www.arduino.cc.

[60] "EPSG:3857." [Online]. Available: http://spatialreference.org/ref/sr-org/6864/. [Accessed: 24-Feb-2013].

[61] "MegaView." [Online]. Available: http://www.imegaview.com.

[62] "Tabs, Pads, and boards (and dodts)," *Dan Saffer*. [Online]. Available: http://www.kickerstudio.com/2008/11/tabs-pads-and-boards-and-dots/.

APPENDIX 1: SOFTWARE

Due to the framework's cross-platform goals, it has been developed in variety of languages, allowing it to operate between our devices. The following appendix documents a selection of the libraries created.

JAVASCRIPT HANDLER INTERFACE

getDeviceType();
Returns the type of client the webapp is being executed on allowing the functionality to alter etc

sendToBack();
Alters the focus of the app moving it to the back of the stack.

bringToFront
Alters the focus of the app moving it to the front of the stack.

forceTrackerUpdate
Signals to the Flow Server object tracker that the display has changed and the known client object image must be updated in the next available cycle.

showInspector
Launches the webkit inspector allowing developers to test and debug their web applications.

viewMjpegStream
Launches the Mjpeg stream currently

isServiceRunning
Returns a Boolean value indicating whether the Flow background service is currently operating and communicating with the server.

getScreenShot
Returns a Base64 encoded JPEG image for the devices current display. This image represents the tracked target the central server is processing against.

stopService
Stops the Flow background service from operating killing communication with the server.

startService
Starts the Flow background service if it is currently dead

startTracker
Starts the automatic update process that synchronises the devices current screenshot display with that of the server.

stopTracker
Stops the local automatic tracker from updating with the server.

open
Launches a local file using the default assigned application

download
Locally downloads the file

downloadOpen
Wrapper function for the download and open commands

downloadOpenFolder
Downloads file from passed URL and opens containing folder.

openUrl
Launches a web browser navigating to the passed URL.