SCUOLA DI DOTTORATO
UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Department of Informatics, Systems and Communication (DISCo)
PhD program in *Computer Science* Cycle: *XXIX*

Protobject: a Rapid Prototyping Platform for Internet of Things and Smart Home

Surname: *Bellino*
Name: *Alessio*
Registration number: *760394*

Supervisors:
Giorgio De Michelis
Flavio De Paoli

PhD Coordinator:
Stefania Bandini

ACADEMIC YEAR *2017*

Abstract

Nowadays, Arduino and 3D printers are frequently used for the rapid prototyping of interactive systems related to the internet of things and smart home. 3D printers are used for building pieces, which are then assembled and eventually made interactive exploiting components connected to Arduino (e.g. sensors, servos, lights). However, although Arduino and 3D printers are remarkable for making prototypes, they are not so suitable for designing them. The design is a preceding stage: without (at least a bit of) design, there can be no innovative making. Moreover, design is usually carried out doing trials and errors. Therefore, design tools should be flexible and quickly adaptable to any unexpected change of environment. With the aim to make the design stages of prototyping easier and more flexible, I designed Protobject, a rapid prototyping tool able to add interactivity to existing stationary objects letting makers and designers reinvent their usage. In fact, Protobject, which works as a multipurpose camera-based sensor, is able to recognize visually observable properties of any stationary object around us (e.g. white goods, lights, windows, doors, curtains). In particular, Protobject can sense their state (e.g., lights on/off) and users' interactions with them (e.g. when turning on a light). Protobject allows users to save money and time during the design stages of interactive prototypes in the fields of internet of things, home automation and tangible interfaces.

Acknowledgements

I would like to acknowledge *Giorgio De Michelis* and *Flavio De Paoli* for their support during my PhD. Many thanks also to *Federico Cabitza*, with whom I discussed many things during my PhD, and Hans Gellersen, from who I learned lots of things.

Thanks to the reviewers of my thesis, Maristella Matera and Giulio Jacucci, who gave me some suggestions for improving the final version of the work.

Thanks to friends and colleagues, who in a way or in another gave me something: Chris, Giuseppe, Giusi, Lorenzo, Massimo, Micol, Norino, Piero, Pierre, Santo, Simeon, Veronica, and many others.

Finally, special thanks to *Daniela Bascuñán*, my life partner, who supported me to review my dissertation.

Dedication

Dedicated to my dad, who helped me to disassemble the world...

Contents

Chapter 1

Introduction

1.1 Background

Internet of things is becoming reality[1]. Connected objects equipped with sensors and actuators (motors, servos, solenoids, etc.) are increasing in our everyday environment. The open issues regarding IoT are not only about available technologies, but also about the design of appealing applications able to open new possibilities to users. Good design should be aimed to set users free from repetitive and boring duties. This requires designers to investigate and observe human behaviour, understand technologies deeply and be creative.

Broadly speaking, to design connected objects and the spaces in which they are embedded, designers should think about them as opportunities for interactions, between objects and people, between spaces and who live in them, and among people. In fact, interaction with the everyday environment is a great source of context that can be leveraged to support users with context-aware systems and, in general, interactive products. This was demonstrated by a large body of work (e.g. [Wei91] [Dou04a] [SAW94] [ADB$^+$99] [Sat01]), of which Mark Weiser was the pioneer.

A telling example of interaction could be the following: when the telephone rings, the user picks up the phone. This means that he/she is busy answering the call. This information can be used to change the Skype state from "available" to "do not disturb" automatically. Consequently, when the user hangs up the phone, Skype can automatically return to "available". In some cases, also the absence of interaction can be leveraged since it can also create a useful context. For example, if it is raining when the user leaves home and his/her umbrella is in its stand (i.e., he/she *did not* interact

[1]According to "The Internet of Things Report: Examining How The IoT Will Affect The World" (Business Insider, November 2015), there will be 34 billions of connected things in 2020.

with it), this may mean that he/she forgot to take it. The information about umbrella presence in its stand, in combination with the one provided by external services, such as weather.com, could be used to send a notification to the user's smartphone: "Remember your umbrella. It is raining heavily!". These are examples of interactions that can be detected according to changes of the object states: for example, the interaction of taking the umbrella changes the state of the stand from "umbrella present" to "umbrella absent".

A wide range of works has explored how the design and development of context-aware behaviour and interactive products in home environments can be supported (e.g. [DAS01] [GPZ05] [ROPT05]). Moreover, the emergence of rapid prototyping tools, either research (e.g. [GF01] [VSH^{+}12]) or commercial ones (e.g. Arduino and 3D printers), made these products cheaper, easier and faster to develop.

In general, context-aware applications and the consequent interactive products[2] are usually developed through an iterative design process that encompasses the understanding of users' needs and then the envisioning, prototyping and evaluation of a product. In this regard, Protobject [Bel16] is a platform designed to facilitate and accelerate the aforementioned design process.

1.2 The Protobject platform

Protobject is a rapid prototyping platform composed of a sensing tool and two frameworks for developers. The scope of Protobject is to enable observation of the states of any stationary object (e.g. white goods, doors, lights, windows, curtains, umbrella stands) as input for prototyping and testing smart systems. In many cases, Protobject can be used to sense also users' interactions with objects. In fact, some interactions can be detected by recognizing changes of object states: for example, the action of closing a door (interaction) changes its state from "open" to "close"; the action of taking an umbrella (interaction) from a stand changes its state from "umbrella present" to "umbrella absent". Protobject is designed to exploit state changes for triggering events and, therefore, prototyping actions. For example, when somebody picks up the phone receiver answering a call (state change), Skype status changes automatically to "busy" (triggered event).

The main features of Protobject are summarized as follows:

- Protobject uses cameras as generic sensors. A smartphone is enough since it can be easily turned into a Wi-Fi camera by installing specific apps: Protobject in-

[2]When referring to context-aware applications, I prefer to use 'interactive product' because I think it better represents the final outcome of design, i.e., a product, which is really situated in the world.

cludes an app especially developed for Android smartphones. Due to the frequent replacement of smartphones ([Wal])], the use of old smartphones is encouraged since they are usually equipped with a camera and Wi-Fi connections, and they can be reused at no cost.

- Protobject allows users to define regions on the captured video in order to concurrently sense different observable states on a single image. Protobject is able to sense objects without altering them or their surrounding environments; hence it is flexible and easily adaptable to changes.
- Protobject supports the detection of binary states (e.g., open/close, present/absent), and discrete states (e.g., door closed/door slightly open/door half open/-door open). By adding physical markers on the observed objects, Protobject supports the tracking of states that, otherwise, cannot be normally observable through images (e.g. the rotation of a symmetrical knob). Finally, Protobject supports the detection of generic presences (e.g. someone/something on a sofa).
- Protobject supports visual training sessions to learn the states associated with an object as a set of images. At design time, the designer captures the different images of interest. At run time, Protobject matches the camera live stream against the stored images of states to detect the current state.
- Two frameworks for Protobject were designed to facilitate the development of interactive products: a JavaScript framework to let developers work at code level, and a visual framework to let users visually connect modules to design specific behaviours. Both frameworks support the definition of actions triggered by state changes.
- Protobject can be used with Arduino for extending sensing/actuation capabilities. Protobject has been designed to detect visual states, and perform software actions (e.g. send a message, call a function). By integrating Arduino, it is possible to control motors/servos/actuators and detect states that are not inferable by visual properties of objects (e.g. internal states of appliances).

Protobject uses deliberately a ‘low tech’ approach: smartphones, low skill requirements, existing objects to be reinvented and reused, and coloured markers that can be easily printed off: all of them are widely available resources.

Protobject is designed for developers, researchers, designers and makers who aim at designing and prototyping novel products in the field of home automation, internet of things and tangible interfaces (e.g. Figure 1.1). In these fields, being aware of the context sensing interactions with (and state of) the surrounding objects is frequently required. Sensing interactions allows the design of dynamic spaces, which can be considered as state machines. The latter react to user’s interactions, which are detected according to change of states. Protobject allows designers to detect such changes of states,

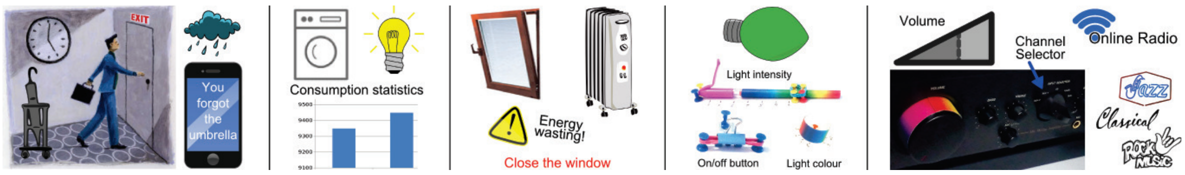


Figure 1.1: Prototyping usage examples of Protobject: (1) smart objects (e.g. smart umbrella stand), (2) energy awareness tools, (3) energy saving tools and (4) tangible interfaces also (5) reinventing and reusing existing interfaces (e.g. Hi-Fi amplifier).

facilitating and accelerating the prototyping of dynamic spaces and, consequently, of novel interactive products, scenarios, and interactions.

1.3 Protobject and the design process

Behind the development of interactive products, there is a design process. The latter has to be put forward in order to understand how Protobject can affect it.

First, I introduce what interaction design is, in accordance with largely shared definitions, discussing the actors who are involved in the design activities and the importance of users' involvement in the design (user-centred design). Finally, I show four basic stages of the interaction design process proposed by Preece et. al. (2015) [RSP15] discussing how Protobject improves them.

1.3.1 What interaction design is

Many authors defined the concept of interaction design[3]. Most of these definitions include three different concepts: (1) people/humans, (2) products/spaces/computers and (3) interactions, which somehow merge the first two concepts. For carrying out interaction design, different competences should be put together since people's needs, desires and behaviour have to be understood (human side competence) to design effective machines (technical side competences).

[3]Preece et. al. describe interaction design as "designing interactive products to support the way people communicate and interact in their everyday and working lives" [RSP15]. Winograd, instead, as "designing spaces for human communication and interaction" [Win97]. Thackara, finally, as "the why as well as the how of our daily interactions using computers" [Tha01].

1.3.2 People involved in interaction design

The emergence of a new effective user experience depends on the knowledge designers have about previous designs[4], users, technologies, and interaction ways. Particularly, designers' concerns should be focused on people's actions and reactions to certain events as well as the way in which people communicate and interact among each other and with surrounding objects. However, this is not enough to create appealing user experiences. Designers must also pay attention to the role that other elements play: what people say about the human experience, what they feel about it, desirability, aesthetics, the business aspect, the technical aspects, manufacturing issues and the marketing side – to name a few.

Undoubtedly, mastering all these fields in interaction design processes is not easy. Therefore, good design can be achieved only through multidisciplinary teams – composed of designers, engineers, programmers, sociologists, toy makers, etc. –, who are able to apply their knowledge to face the different aspects of design problems.

The work carried out by these multidisciplinary teams has significant advantages, such as the emergence of different kinds of ideas, the development of new methods and the originality of the final product. Nevertheless, the divergence of knowledge that exists among all the possible backgrounds involved in the field of interaction design may foster communication issues among the experts of different areas. In other words, people from one discipline could regard as relevant something that is not very important for somebody from another field of study [Kim90]. Thus, to avoid misunderstandings, people being part of multidisciplinary teams should learn to speak a common language. In this regard, the concept of state change – which is the base of Protobject operating way – can be considered as a basis for a common language able to facilitate the mutual understanding between product designers and developers. In fact, developers usually have a clear understanding of the concept of state machine whereas designers are aware of the possible configurations (of state) of their design (e.g. light on/of, table closed/opened).

1.3.3 User-Centred Design Approach

The involvement of users throughout development is the most effective way to take into consideration their activities when it comes to developing new things. By doing this, more suitable and usable products can be created since they are in tune with users' goals, requirements and needs.

[4]Novel designs cannot exclude the knowledge of past design. In this regard, Walter Benjamin, critic and philosopher, noticed that the Paul Klee's "Angelus Novis" (New Angel) has his face toward the past.

Gould and Lewis (1985) proposed three principles that lead the user-centred design. These principles would foster a good design of computer systems [GL85]:

1. *Early focus on users and their activities*[5]: this principle is aimed to study users' cognitive, anthropomorphic, behavioural and attitudinal features. To do so, users must be observed and studied during their normal activities. Then, users are involved in the design process.
2. *Empirical measurement*: this principle points out that users' performances, reactions and interactions with the system are observed, measured and analysed This process has to be carried out in all the stages of design, from early developments (e.g., sketch) to late development (e.g., prototypes near to the final product). Choosing what to measure is a problem of design as well.
3. *Iterative design*: the design of interactive products is carried out through several cycles of iterative development. If there are problems in a user testing stage, these are corrected in a new cycle of design to analyse the effects of what has been corrected.

Nowadays, these three principles are widely accepted when designing under the user-centred approach (e.g. [MVSC05] [RSP15]). The first principle is one of the most relevant since a good analysis of the users' activity allows designers to avoid wasting time while looking for the right direction. Any decision related to design should be made in the light of users, their activities and environment. Moreover, the users' behaviour with interactive systems can be effectively understood on real prototypes embedded in real contexts [Dou04b]. Therefore, designing prototypes is crucial and the use of Protobject can be advantageous to do it.

1.3.4 Interaction design process

According to Preece et. al., four main activities are carried out in the design process of interactive products [RSP15]. They are:

1. establishing requirements for the user experience;
2. idea generation and designing alternatives that meet those requirements;

[5]Gould and Lewis (1985) use tasks instead of activities. In this regard, I prefer to use activities because I think they better represent human beings' way of living. In my opinion, the concept of tasks is too strict.

3. prototyping the selected alternative designs so that they can be deeply observed, communicated and assessed;

4. evaluating what is being built throughout the process and the user experience it offers.

To conduct the design process following users' feedback, the aforementioned activities have to be part of an iterative cycle whose objective is to refine and improve the target product. Both users and designers participate in the discussion on how to open new – also unexpected – possibilities to users according to their wishes, needs, wills and expectations. Then, different perspectives will arise – taking into consideration factors like feasibility, usefulness and necessity. In this case, iteration and the constant revision of ideas are crucial as designers must consider successes and failures throughout the design process, without seeing people interacting with the designed system. Moreover, this is particularly relevant because designers never get the final solution right at the first attempt [GL85]. In brief, after establishing some requirements, *"the design emerges iteratively, through repeated design-evaluation-redesign cycles involving users."* [RSP15].

First activity: Establishing Requirements

Requirements are qualities and performances demanded of a system in accordance with user's needs, desires and expectations. Requirements are often not clear at the beginning. They usually emerge after the interpretation of such needs, desires and expectations envisioning new behaviour with the aim to change the previous one. Therefore, product requirements emerge after deep understanding and creative analysing of users, focusing on how they can be supported in their normal activities. Defining product requirements is the starting point of any design cycle and a fundamental part of the user-centred approach.

Second activity: Designing Alternatives

This activity focuses on generating alternative ideas for meeting product requirements and designing it. Idea generation and designing alternatives is the most creative stage of the design process. Two kinds of design are carried out at this stage: conceptual design and concrete design. The former consists in the creation of a conceptual model, which depicts what users will be able to do with the newly envisioned product and how they will learn to interact with the product. The latter deals with concrete aspects of the product, i.e., colours, sounds, images, menus and/or other interaction mechanisms, icons. In both conceptual and concrete design, designers always keep alternatives in mind.

Third activity: Prototyping

Prototyping is fundamental in interaction design as it allows designers to understand how users interact with the product before creating its final version. Prototypes are useful for designers to explore ideas and test requirements as well as first design choices. At initial stages, designers may create prototypes with simple and easy to get materials (e.g. paper, cardboard, stationary items, recycled objects, etc.) to test the design and evaluate its strengths and weaknesses. At further stages, prototypes are refined and improved so that they look always more like the final creation.

Fourth activity: Evaluating

Designers need to know how users will behave and interact with their designed object and evaluate it to understand if it really meets users' needs and desires and/or if it generates unexpected problems. Evaluations also allow designers to choose among different alternatives and decide the ways in which they will develop those alternatives in order to improve performances or fix problems. In the framework of the user-centred approach, evaluation is an important aspect to involve users in design.

1.3.5 Protobject impact in the design process

The four activities previously surveyed are closely connected: "*alternatives are evaluated through the prototypes and the results are fed back into the design or might identify missing requirements*" [RSP15]. This iteration cycle is one of the key characteristics of a user-centred approach. Iteration is a process to acquire new knowledge: through iteration, design teams learn new things that can be used for improving the next cycle.

Protobject can accelerate the design process improving it mainly in (a) the design of different alternatives and idea generation (second activity) and (b) prototyping (third activity).

Regarding the design of different alternatives and idea generation (a), Protobject allows designers to reinvent the use of existing objects remixing them for different scopes. Seeing objects from different perspectives, observing alternative designs (i.e., different existing options) and remixing existing objects – which can be seen as a repertoire of things to merge together – is a common activity for idea generation. Protobject lets designers reinvent, reuse and remix different existing things tinkering (meant as the act of doing bricolage) with them.

Regarding prototyping (b), it must be also pointed out that Protobject transforms prototyping in a vehicle to encourage designers to reflect while designing [Sch83]. In fact, Protobject allows designers to embed [Dou04b] the “object of design” in the real world making it work in a real context. This allows designers to have real experiences with the designed objects letting designers “reflect in practice” [Sch83] on them.

Of course, idea generation and prototyping are highly related from the Protobject point of view. On the one hand, “reflection in practice” with prototypes “embedded in the real world” allows designers to generate new ideas. Inspiration, in fact, is an experience emerging from practice [BDME$^+$11]. On the other hand, generating new ideas and alternatives “embedded in the real world” through real prototypes allows designers to “reflect in practice” on the designed objects.

The next sections extend the discussion on the aforementioned concepts regarding (1) idea generation and (2) prototyping.

Idea generation, alternative design and Protobject

One of the aspects that is crucial in design is generating alternatives ideas. In this regard, Linus Pauling[6] once said: “*The best way to get a good idea is to get lots of ideas*”. Unlike engineering, the design is not aimed to create the best solution to a pre-existent problem. Nevertheless, design is aimed to propose transformations of interactions and current practices in order to improve them. Consequently, there is no optimal design at all, but different alternative ideas have to be tested and investigated together with stakeholders. Therefore, to think such novel ideas, we need a creative mind, i.e., to be able to observe what already exists and transform it creating something different offering new possibilities of action and interaction to stakeholders. For example, when designers observe objects in the environment, they get the inspiration to think of new perspectives and create new possibilities. In this sense, designers create novel interactive objects or give interactivity at existing ones by drawing on knowledge gained from things they have seen or experienced [MP14].

This approach has been assumed as basic in a discipline like architecture and is part of all its educational programs. For instance, students of master’s programs are required to observe their environment and experiment with the objects that are a potential source of inspiration. These students are encouraged to see things differently observing them from different perspectives [BDME$^+$11].

Moreover, A. Telier states that “*Envisioning and realizing concepts is carried out through the manipulation of a variety of representations and viewpoints. Design can thus be viewed as a kind of bricolage, where different materials are brought together, mixed, and configured in various iterations. Resources that provide an element of surprise and discovery may help designers to see things in a new way. Inspiration has to do with*

[6]American chemist, biochemist, peace activist, author, and educator, Twice a Nobel Price winner.

particular qualities of objects, people, ambience, and places. It always emerges in a context" [BDME$^+$11].

In addition, when designers discover what can be newly conceived in what already exists, also relating objects that are apparently unrelated, they need to transform and reprogram those objects and their context. This process involves exploration of the context, experimentation and widening perspectives. Shifting perspectives, carrying out experiments and taking time to discover new possibilities in what already exists is particularly important to view things differently and, consequently, to create a new perspective [LW98].

In this regard, A. Telier states that "*different materials, media, and representational forms are necessary for conveying and exploring different (conceptual, technical, aesthetic) aspects of a design. Design work is looked on as an act of 'metamorphing', where design concepts are envisioned and realized through objectifying and manipulating a variety of representations. Experiences gained in one context could be applied in another*" [BDME$^+$11].

Following this argument, it can be asserted that novel ideas and products usually come from cross-fertilization of different perspectives, individuals, and contexts. This means that innovation depends on how designers observe existing objects and use them as a starting point for their creation. In that sense, Protobject encourages designers to tinker with existing objects and their representations seeing them from many different viewpoints (i.e., reinventing them) and putting them in real contexts. Tinkering with Protobject also helps designers to relate "unrelatable" objects and materials to convey and explore different aspects of design. In brief, Protobject fosters the cross-fertilization of ideas letting designers combine different objects for generating novel ideas and design alternatives.

Moreover, to communicate a new idea more effectively, it is essential to show what we mean through prototypes in which experiencing real interactions is possible. In other words, what we say is not enough to convey the experiential dimension of what we design [JW07]. Protobject helps communicate ideas because it is able to add an "experiential dimension" (interactivity) to the designed objects fast and easily.

To conclude, I could state that novel and alternative ideas come from seeking different perspectives and looking at other designs. In this sense, Protobject fosters tinkering with objects and its approach could be also seen under a perspective closer to the makers' one. According to Banzi, "*Reusing existing technology is one of the best ways of tinkering. Getting cheap toys or old discarded equipment and hacking them to make them do something new is one of the best ways to get great results*" [Ban09]. Protobject, supporting tinkering, allowing designers to reuse and reinvent everyday objects, even if they are old and/or damaged.

Rapid prototyping of different designs with Protobject

Prototypes can be defined as materializations of a design whose aim is to explore to what extent such a design is suitable for a certain scope. In that sense, prototypes are communication artifacts among stakeholders who can discuss and explore design ideas.

Developing prototypes is a relevant stage of design since it fosters reflection in design as discussed by Schon [Sch83]. He claims that "*designers learn and conduct professional artistry through processes of reflection-in-action, in which knowing and doing are inseparable*" [BDME+11]. He outlines how these processes are carried out "*in on-the-spot experiments where the materials of the situation (models, sketches, drawings) at hand 'talk back', often in surprising ways, and where the naming and framing of the specific problematic or puzzling design situation are important activities*" [BDME+11].

This way of conceiving design has been clearly shaped by John Dewey. Under his perspective, real-life experience fosters the comprehension of facts, situations and things [Dew13]. According to Dewey, "*creative processes include everyday practical reflections as well as artistic production and scientific research*" [BDME+11]. Along the same lines, Merleau-Ponty states that human perception plays an important role in understanding the world, generating sense in human experience [MP62]. Dewey and Merleau-Ponty mean that knowledge is inseparable from perception and interaction, which are the results of experience.

This idea supports the notion of embodied interaction proposed by Paul Dourish [Dou04b], which conveys the idea of perceiving situations as a whole. He states that embodiment is about 'things' that are embedded in the world and their reality depends on the 'degree of embodiment' with the real world. Actions and meaning are carried out in a specific context (i.e., they are embodied in the world) and can be understood only through their experiences, accomplishments and executions.

As explained previously, Protobject allows designers to reflect-in-action using a repertoire of exemplars (objects to be reinvented). Thanks to Protobject, such a repertoire can be easily leveraged making the development of prototypes flexible and adaptable to any change of setting and environment. Moreover, prototypes designed with Protobject are embodied in a context. Therefore, designers can have real experiences with the designed prototypes. According to Dourish, having real experiences with "things" (prototypes) embedded in the world helps designers to explore and better understand their design and, as discussed in the previous section, to better communicate design ideas.

Prototypes have several purposes. To begin with, they facilitate the decision-making process when designers have different alternatives that emerge from the iterative design process. Secondly, prototypes (help to) answer the questions that may arise in the design process. Additionally, prototypes are useful to determine whether a design is feasible or not and to test and evaluate the user experience.

Protobject accelerates these choices and, therefore, the design process of interactive products. In fact, when prototyping, developers can start from two different situations: from scratch or modifying what already exists [RSP15] – and Protobject supports the latter situation. Modifying what already exists is often convenient to speed up the design process. I could say that Protobject allows designers to produce high-fidelity prototypes using a kind of low-fidelity approach. The distinction between high-fidelity and low-fidelity prototypes was discussed by Preece et. al. [RSP15].

Low-fidelity prototypes must be simple, cheap, and quick to produce. These characteristics make these prototypes quite attractive for developers since they are easily modifiable and, consequently, facilitate the exploration of alternatives in the design process. At initial stages, this feature plays an important role in development, considering that prototypes should be flexible and allow exploration in order to create suitable final products [RSP15].

Low-fidelity prototypes are usually not interactive since they are produced through storyboards and sketches. I could state that Protobject reduces the gap between low-fidelity and high-fidelity prototypes. In fact, Protobject allows designers to see objects and spaces as state machine from the initial stages of design. Thanks to Protobject designers can somehow achieve high-fidelity prototypes straightforward having a clear idea of the states of objects in the environment. To some extent, I could state that Protobject allows designers to create high-fidelity prototypes, which look like the final product, exhibiting a real interactive behaviour. Nevertheless, the approach to produce prototypes with Protobject is close to low-fidelity, i.e., simple, cheap and fast to produce encouraging explorations and modifications as well.

1.4 Dissertation chapters

This dissertation is organized as follows. Chapter 2 presents the literature review discussing Protobject with respect to bricolage-oriented prototyping tools. Since Protobject allows designers to tinker with existing objects reusing and reinventing them to develop prototypes, I discuss other tools in relation with this capability. Chapter 3 presents details on how Protobject works, i.e., the recognition algorithm, the use of markers to facilitate the recognition and so on. Chapter 4 presents the architecture of Protobject discussing its components: a smartphone app, a desktop application and two frameworks for facilitating the development of smart apps through Protobject. Chapter 5 discusses the details on the two aforementioned frameworks that support the development of applications through Protobject. The first framework lets users develop with Meemoo [O$^+$12], a data-flow (visual) programming language I enriched with modules specifically designed for Protobject. The second one uses JavaScript and the browser. Protobject offers many prototyping possibilities (e.g. Figure 1.1) and chapter 6 presents six usage scenarios where Protobject is used, also coupled with an Intel (Arduino compatible) Edison board. Four usage scenarios have been developed using Protobject only whereas the other two have been developed using Protobject coupled with Edison. Chapter 7 presents the system evaluation, i.e., the detectable events, the robustness of the recognition of states, generic presences (e.g., of someone/something on a sofa) and the precision that can be achieved with Protobject when detecting continuous movements (e.g. the opening angle of a door, or the position of a continuous knob). Chapter 8 presents the user evaluation, which was carried out through two workshop sessions conducted with 22 participants. Finally, the conclusions are presented.

Chapter 2

Literature Review

2.1 "Savage" and scientific mind dichotomy

In order to position the literature review, I borrow and use a dichotomy coming from anthropology. The dichotomy in question, developed by the French anthropologist Lévi-Strauss, discusses the difference between the Bricoleur and the Engineer [LS66]. In his work, *The Savage Mind*, he describes the "Bricoleur" as someone who "*works with his hands and uses devious means compared to those of a craftsman*". The word "Bricoleur" actually originates from the old French verb *bricoler*, which in English means do-it-yourself, puttering or tinkering with tools and materials on hand for creating or fixing things. According to Lévi-Strauss, the Bricoleur "*is adept at performing a large number of diverse tasks; but unlike the engineer, he does not subordinate each of them to the availability of raw materials and tools conceived and procured for the purposes of the project*". Therefore, the Bricoleur adapts his/her projects to a limited set of materials and tools that he/her has on hands. Lévi-Strauss affirms that the bricoleur approximates the "savage" mind whereas the engineer approximates the scientific mind.

This dichotomy between bricoleurs and engineers can be extended to tools as well: some of them are more suitable for engineers whereas others for bricoleurs. Regarding prototyping tools, many of them support the engineer's approach, but there exist also prototyping tools that are more suitable for supporting the bricoleur's approach. As discussed in the introduction, Protobject supports the bricoleur's approach since it allows designers to reuse, combine and reinvent existing objects (materials on hands), puttering or tinkering with them.

In the following subsection, I discuss this dichotomy focusing on commercial and research rapid prototyping tools.

2.2 Prototyping tools

2.2.1 Virtual prototyping tools

Virtual prototyping is the analysis and simulation of physical products using computer models. Virtual models allow engineers to conduct some of the same tests that they could conduct on physical prototypes [Wan02]. Many tools and approaches have been developed for creating virtual prototypes [DAM$^+$11], [CC04], [ZWPG03], interactive virtual prototypes [Che99], [DAK03], [BCF06] (where haptic feedback is added) and mixed prototypes (combinations of virtual and physical prototypes, e.g., 3D printed components) [BCCP09], [AK09].

Since Protobject is strongly focused on physical and real prototyping, similar tools and approaches are out of the scope of this dissertation. Moreover, unlike Protobject, virtual prototyping and its related branches require high initial investment costs in terms of hardware (e.g. CAVE for stereoscopic vision, haptic devices) and high competences of the operators [CTG99].

Undoubtedly, virtual prototyping – which is also known as computer-aided engineering (CAE) or engineering analysis simulation [CTG99] – approximates the scientific mind and, therefore, the engineer's approach. This is because engineers are allowed to define any detail and behaviour of the simulated products exactly as they want.

2.2.2 Rapid prototyping tools

The term "rapid prototyping" is usually referred to techniques that quickly allow the fabrication of physical products (e.g. through digital fabrication, sensor-based devices, etc.).

Digital fabrication tools

With respect to digital fabrication devices, I could state that, in the past, they have approximated the scientific mind and, therefore, the engineer's approach. In fact, 3D printers or additive layer manufacturing tools could be used only after appropriate 3D CAD/CAM modelling, involving ways of doing typical of the engineer's approach. In fact, with 3D CAD/CAM modelling tools, users design exactly the pieces they want, as engineers are used to do. However, after the emergence of commercial 3D printers, the positioning of digital fabrication has changed over time, moving toward the bricoleur side, also thanks to services like Thingiverse[1] and Autodesk Tinkercad[2]. Thingiverse is a repository of shareable 3D-printable things that can be easily edited, combined and remixed thanks to Tinkercad. Somehow, the emergence of 3D printers – in addition to software/services like Thingiverse and Tinkercad – has brought CAD/CAM modelling closer to bricoleurs.

Concerning research digital fabrication tools, some of them are more oriented towards bricoleurs (i.e., they offer components that can be readapted and reused for many general purposes) whereas others support mostly the engineer's approach (i.e. they do specific prototyping). For example, engineer-oriented tools are Printed Optics [WBHP12] and Instant Inkjet Circuits [KHC$^+$13]. Printed Optics uses 3D printed optical elements able to make a device interactive. Instant Inkjet Circuits allows engineers to fabricate any electronic circuit by printing it [SPD$^+$10]. Lamello [SHH$^+$15], Acoustruments [LBHH15] and Sauron [SCH13] are examples of bricoleur-oriented prototyping tools. Lamello focuses on acoustic sensing for creating tangible input components that recognize user's interaction. Acoustruments is designed to give tangible functionalities (several interaction primitives are investigated) to smartphones devices exploiting their audio system. Sauron modifies hollow 3D models to enable an embedded camera to sense interactions on physical controls like buttons, sliders and so on.

Sensor-based tools

As mentioned before, a relevant branch of rapid prototyping tools is based on devices and sensors. The most popular is undoubtedly Arduino. As described in the Arduino introduction guide[3], *"Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs – light on a sensor, a finger on a button, or a Twitter message – and turn it into an output – activating a motor, turning on an LED, publishing something online, [etc.]"*. Arduino is frequently used together with 3D-printers. After fabricating 3D pieces, in fact, makers are used to assemble and make them interactive developing specialized components connected to Arduino (e.g. light sensors, knob potentiometers, servos, and so on). Arduino off-the-shield components are specifically designed to be easily used in order to face many different predefined design needs.

There exist also many research approaches that have been developed for creating/designing interactive objects using devices and sensors. For example, Phidgets [GF01], d.tools [HKB$^+$06] and .Net Gadgeteer [VSH$^+$12] are approaches that provide hardware/software packaged modules, which can be used for the rapid prototyping through their well-defined APIs.

[1]www.thingiverse.com
[2]https://www.tinkercad.com/
[3]https://www.arduino.cc/en/Guide/Introduction

We could state that all the aforementioned sensor-based prototyping tools (both commercial and research ones) approximate the bricoleur's approach. This is because they offer a *finite stock of materials and tools* [LS66] (i.e., predefined sensors, modules and behaviours) that can be used for facing different and undefined prototyping needs.

Wizard of Oz approaches

The Wizard of Oz [SC93] method is "*based on the observation of the user when operating an apparently fully functional system whose missing features are supplemented by a hidden human wizard*". Therefore, hidden humans substitute systems functionalities, which are simulated through Wizard-of-Oz tools. This approach can be clearly used for rapid prototyping of interactive systems replacing the use of sensors, which can be simulated by humans.

Actually, Protobject does not work as a Wizard-of-Oz tool since humans are not needed for the detection process. Nevertheless, from the HCI perspective, the potential contribution of Protobject to the IoT and smart home fields may be comparable to the one of the Wizard-of-Oz technique. In fact, Protobject works somehow like its hidden humans, namely observing and sensing the environment.

According to a recent survey [GPT15], most of the Wizard-of-Oz tools have been developed for prototyping and testing specific scenarios or systems. On the other hand, just few tools (e.g. [ABC$^+$09]) are general-purpose and customizable, although customization could be time-consuming. Protobject, instead, is designed to offer a simple general mechanism for defining and recognizing (without humans) object states leveraging them for prototyping scopes.

I put the Wizard-of-Oz techniques at the bricoleur's side since they let designers use, reuse and reinvent existing objects without any modification. Anyhow, Wizard-of-Oz tools have to be suitably developed for specific scenarios or systems and, therefore, the engineer's side is somehow present.

Paper prototyping

In paper prototyping [BLM03] [STG03], components of systems are represented using paper whereas their behaviour is simulated by people. To some extent, I believe that paper prototyping is (philosophically) quite close to Protobject. Instead of using paper and people to simulate technological behaviour, Protobject lets designers reinvent the use of existing objects, giving them new interactivity to better simulate interactive behaviours for prototyping scopes. Probably, paper prototyping approximates in the best way the world of bricoleurs.

2.2.3 Extending the bricoleur's world discussing Protobject

Prototyping with sensor-based rapid prototyping tools usually involves the use of off-the-shelf sensors and requires the fabrication and assembly of 3D-printed components. Of course, 3D-printed components and dedicated sensors have many benefits in end-user interactive smart systems, and their use is deep-rooted in the community [HVSS12] [AM04] [SDA99] [ZB06] [OWM15]. The usage of sensors and 3D-printed components is advantageous also in the last stages of prototyping (e.g. when products are given to intended users for testing and collecting feedback) so that the prototype can work stably. Generally, I believe that sensor-based approaches like Arduino and 3D printers are remarkable to make prototypes rapidly and at low cost. Nevertheless, making prototypes does not actually mean designing them: without (at least a bit of) design, there can be no innovative making. Therefore, the design is a preceding stage, which is usually carried out in the early stages of prototyping.

In these stages, sensor-based approaches (i.e., Arduino and similar boards) and 3D printed components may have some limitations. In design stages, in fact, designers usually follow an approach based on trials and errors [TVH02] [CS15]. This means that changes of environmental settings, design or scenario can occur frequently. This may make the sensor-based approaches cumbersome. In fact, it could be necessary to move, add or replace sensors, set up again the system, move wires reinstalling sensors in different positions, contexts or environments wasting time.

Installation of sensors in existing hardware, instead, is usually error prone. Moreover, eventual hardware modifications are often irreversible. For example, consider the installation of a sensor to detect the opening angle of a door. The irreversibility of mechanical modification is clear (e.g. users may need to drill the doorframe to install the potentiometer), but also measurement errors could be done, e.g. drilling the doorframe wrongly.

In the early stages of prototyping, also digital fabrication may be cumbersome. It is expensive (especially when several different versions of components have to be fabricated and tested), frequently slow (in particular 3D printers), and error prone (due to design errors during CAD modelling, compelling users at printing again components after the needed revisions).

In general, while designing, makers and designers may wish to focus just on scenarios, interactions, and the operating way of interactive products rather than on electronics, sensors, or fabrication processes.

In order to overcome these limitations and make design stages of prototyping simpler and more flexible, other approaches could be helpful. In this regard, Protobject, which strongly supports the bricoleur's approach, lets users rapidly change the environment settings without being concerned about sensors, wires and the general settings of the environment. Protobject also lets users reinvent and reuse existing objects without any (irreversible) modification.

Moreover, unlike sensors, which are not general-purpose (i.e., a sensor is required for each property that has to be sensed), Protobject lets users segment its video stream in different regions and sense the state of many objects at the same time with a single camera. Protobject can be adapted for fulfilling many and unexpected usages and eventual changes of environment. In fact, Protobject lets users exploit objects available on hands (which are mainly "poor" or recycled) reinventing their usage in order to fulfil specific prototyping needs. This aspect makes Protobject very close to the bricoleur's approach extending it, opening new possibilities to bricoleur-oriented designers. Finally yet importantly, makers and designers can use Protobject also without having hardware/software skills, which are instead required to implement sensors or design 3D CAD models to be printed.

2.2.4 Other tools and systems similar to Protobject

Protobject is not the only tool that extends the bricoleur's approach through the capability to add interactivity to existent objects. For example, Touch and Activate [OST13] adds touch-sensing capability to any rigid object exploiting its resonant properties, which change according to the user's interactions (e.g. different ways to grasp). Resonances are detected using a sensor, i.e., a pair consisting of a vibration speaker and a piezo-electric microphone attached to the target object. Therefore, Touch and Activate also allows rapid prototyping reinventing the usage of existing objects, but it has some drawbacks. The speaker and the microphone have to be physically attached to objects, and this could be not ever possible. Moreover, objects have to offer good resonant properties and to be not too big [OST13]. Finally, unlike Protobject, Touch and Activate is not able to sense any visually observable property of objects (e.g. light on/off) but just properties that can be sensed exploiting different resonances (e.g. see their prototyping examples with Lego).

Eyepatch [MAWI07] is another rapid prototyping tool, which is designed for sensing interactions and objects by using cameras. It offers many ways to be trained and, therefore, its usage possibilities are wide even if it is more difficult to manage than Protobject. At any rate, its usage possibilities are different from the Protobject ones. For example, Eyepatch allows the detection of people walking around, tracking objects using classifiers, or recognize objects (although training could be time-consuming). Nevertheless, unlike Protobject, Eyepath is not designed explicitly for the detection of state of objects. Although Eyepath allows the recognition of objects, it is not able to recognize their state in many cases. In fact, using SIFT classifier [Low99], Eyepath recognizes objects with invariance to scale, pose and illumination. Nevertheless, the state of objects may change according to different scales and poses (e.g. the scale and pose of a

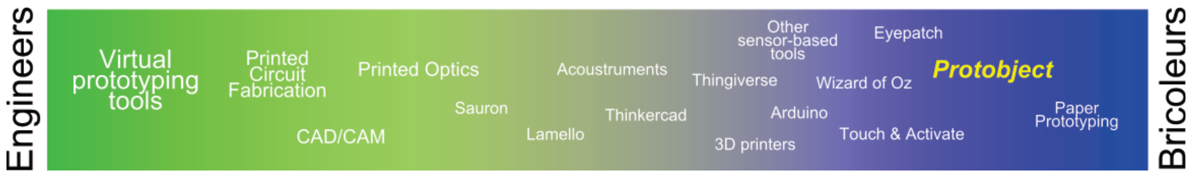


Figure 2.1: The continuum between "engineers" and "bricoleurs" and the discussed prototyping tools.

door changes according to its opening angle; or the pose of a knob changes according to its position). It means that Eyepath could be trained, for instance, to detect where a knob of an oven is (detecting its position on the captured image), but associating a certain rotation of the knob directly with the function of the oven (e.g., Defrost, Grill etc.) is not possible.

Protobject, instead, is designed for explicitly associating a certain state of an object (e.g. door opening angle, or the position of a knob) directly with a return value (e.g. the angle value, or the function selected by the knob). This simplicity makes it straightforward and suitable for many different cases (see usage scenarios later in chapter 6). Finally, Eyepatch cannot sense different regions of the video stream simultaneously as Protobject does.

2.3 Discussion on the dichotomy

It would be wrong to categorize the aforementioned prototyping tools in tools for engineers and tools for bricoleurs in a mutually exclusive way. In fact, as in many dichotomies, "engineers" and "bricoleurs" should be considered as end-points of a continuum [AR06] and not like distinct categories. The prototyping tools discussed in the previous section, therefore, should be positioned within the continuum between "engineer" and "bricoleur".

Figure 2.1 shows my vision on how some of the prototyping tools discussed up to now could be positioned in the continuum between "engineers" and "bricoleurs".

Finally, note that approaches having different positioning can be normally used together by users without any problem (e.g., mixed prototyping). In fact, engineers and bricoleurs, despite different approaches to solve problems (and eventual different preferences when choosing prototyping tools) live within a restrictive reality. The latter, according to Lévi-Strauss, can be explained through the fact that engineers and bricoleurs are compelled to consider the pre-existing set of theoretical, practical and technical means in a similar way [LS66]. Therefore, I can state that users can choose to behave as engineers or bricoleurs according to the use of different prototyping tools (i.e., the known restrictive reality), which are usually chosen according to convenience, practical constraints and prototyping scopes.

2.4 Other camera-based tools and systems

Since Protobject uses cameras for sensing objects in the environment, in this section, I discuss other related camera-based tools and systems even if not explicitly designed for prototyping. Some of them are designed to get information about physical objects attaching visual markers on them (e.g. 1D-barcodes [ALF06] or 2D codes [RG04]), which can be then read using cameras. In [QBVG08], the authors do not use any markers, but combine different factors (i.e., local feature correspondences, verification with projective geometry, and search space restriction by multimodal constraints, such as GPS location) to recognize objects. Cameras are also exploited to detect objects on tables suitably designed for the development of tangible user interfaces. Examples are reacTIVision [KB07], which uses fiducial markers (i.e., a sort of QR code) or ColourTable [MPW08] and MouseHaus Table [HYLDG03], which use colour and shape recognition respectively.

Other video-based systems are designed as end-user sensing systems. Crayons [FO03] lets users make their classification algorithm to recognize objects and interactions. Light Widgets [FO02] lets users create virtual interactive elements on any surface. Slit-Tear Visualizations [TGF09], instead, aims at recognizing environmental events (e.g. cars going across a street). Besides the fact that these approaches ([FO03], [FO02] and [TGF09]) are not designed for rapid prototyping, they often rely on users, which annotate images to provide training data and this process is time-consuming. On the contrary, the training required by Protobject is simpler since users are required to give just a name to any recorded image in the database of states.

Another relevant end-user sensing system tool is Zensors [LLW+15]. I put forward Zensors since it is an ambitious tool able to detect the state of existing objects. Moreover, using cameras, Zensors can be used as a generic sensor for any observable property (e.g. door open/closed) like Protobject. At any rate, Zensors exploits machine learning and crowd-powered answers (real workers are recruited from Mechanical Turk) for training sensors, i.e., for classifying images. This approach is not applicable for the rapid prototyping of interactive systems since training could require several time and has a cost (i.e., Mechanical Turk workers). Moreover, the recognition frequency of Zensors can range from near real-time (one second) to longer period (almost a day) [LLW+15]. Therefore, Zensors is not suitable for actual real-time prototyping. Instead, Protobject frequency recognition ranges from 30 milliseconds to one second, being more adapt for real-time usages. Finally, like Eyepath, Zensors does not allow makers and designers to associate directly a certain state of an object (e.g. light turned off) with a return value (e.g. “Off”).

Associating a certain state of an object directly with a return value is the most remarkable feature that makes Protobject different from other approaches. Its relevance for rapid prototyping in fields like IoT, Smart Home and tangible interfaces can be easily understood through the usage scenarios displayed later (chapter 6).

Chapter 3

Protobject operating way

3.1 Region definition and state change

Protobject allows the observation of objects in the environment through regions that define the areas of interest (i.e., the object - or part of it - to be observed) in the captured images. Moreover, Protobject detects change of states in such regions; the latter are leveraged to trigger interactive behaviour.

3.1.1 Basic principles

I introduce Protobject operating way through an example of region and state definition of a home phone. On the video stream captured by Protobject (see Figure 3.1), two regions are defined: the first one (green region) contains the receiver whereas the second one (yellow region) contains the light (i.e., a LED) of the phone indicating if there is a call or not. Protobject needs to record the entire set of possible states of each region in order to detect eventual changes of states. So, a database of images, one for each known state of each region, is created. For example, the possible states of the receiver (green region) are two: “Down” and “Up”. The possible states of the light (yellow region) are also two: “Normal” and “Calling”.

The recognition occurs comparing the images captured by Protobject in each region with the related images of state previously recorded in the database. Thus, an algorithm identifies the most similar image for each region calculating a similarity index (SI) [Gos12]: the highest one identifies the current state. Although SI usually ranges from 0 to 1, Protobject re-computes it so that SI ranges from 0 to 1000 (it is basically multiplied by 1000). In this way, SI is easier to read by users.

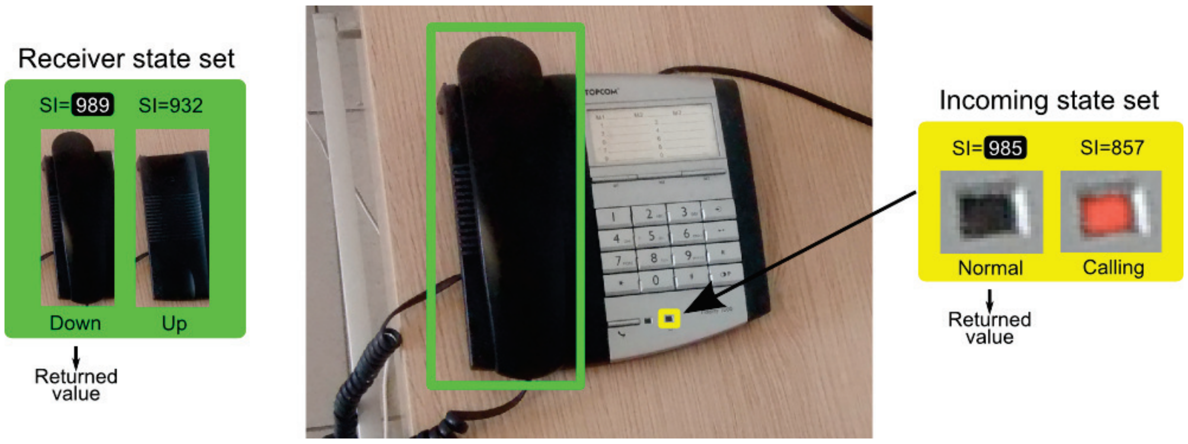


Figure 3.1: The recognition algorithm

For example, in Figure 3.1, the highest SI for the phone receiver is 989 and the "Down" state is returned whereas the highest SI for the light of the incoming call is 985 and the "Normal" state is returned. The recognition process is carried out on the video stream to detect any change of state over time (e.g. when the phone rings, the light of the incoming call turns on and Protobject returns "Calling" instead of "Normal"). Of course, as in Zensors [LLW$^+$15], the segmentation of the video stream in regions can be exploited also for sensing many objects simultaneously using a single camera (e.g. see the case study of the umbrella stand shown later).

Note that Protobject does not follow the position of the defined regions in objects. This means that if the phone in Figure 3.1 were moved, Protobject would detect an error since the defined regions would go off-centre. I put forward this drawback just to point out that Protobject should be used only with stationary objects. The phone in Figure 3.1 can be used anyhow with Protobject as long as it is properly fixed on the table (e.g. with Blu Tack) to avoid unexpected movements.

3.1.2 Avoiding detection interferences

The regions to take into account during the detection should be chosen so that the user interaction interferes as little as possible with the detection.

For example, according to the green region in Figure 3.2, when users pick up the receiver, their hand can influence the detection. According to the cyan region, instead, the possibilities of interference with the hand of users are lower since users are not used to pick up the receiver from that region. Of course, the cyan region is better than the green region. The aspects that regard possible interferences with users' interaction should always be taken into account when defining regions.

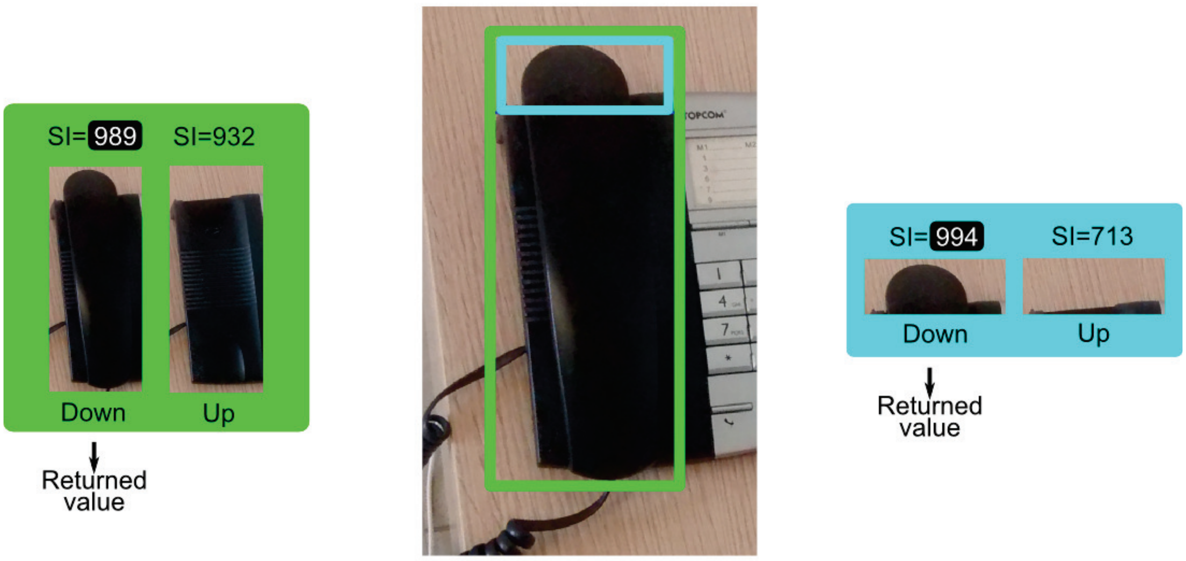


Figure 3.2: The cyan region is better than the green region.

3.2 The similarity algorithms

Protobject combines two different algorithms to compute the similarity index between images: (1) Pearson correlation between RGB intensities [Gos12] and (2) Cosine similarity between HoG descriptors [DT05].

3.2.1 Pearson correlation between RGB intensities

Pearson correlation algorithm computes the linear relationship between the RGB intensities of two images. If two images of the same scene are obtained under different lighting conditions and corresponding RGB intensities are linearly related, a high similarity will be obtained between the images. Therefore, Pearson correlation coefficient is suitable for determining the similarity between images where intensities are linearly related [Gos12].

3.2.2 Cosine similarity between HoG descriptors

The HoG descriptor is a vector-space model and *"appears to be a reasonably good model of perceptual similarity"* [DE], which is approximated by Euclidean or cosine distance between two HoG vectors. Moreover, it *"uses intensity gradients rather than intensity* *directly, which means that the responses of edges are localized; it is sensitive to local but not global contrast due to its normalization scheme; it can handle minor misalignment due to the bilinear interpolation between HoG cells"* [DE].

Protobject extracts HoG descriptors from the two images to compare. Therefore, it computes the similarity between those descriptors through the cosine similarity. For extracting HoG descriptors, many parameters have to be set [DT05]. They are cell size, block size, block overlap, orientation bins and block normalisation scheme. The cell size is the only parameter that changes according to the size of the regions. I empirically found a good setting for Protobject scopes according to the following formula (region width and height are defined in pixels):

$$cellSize = \lfloor \frac{20 \cdot regionWidth \cdot regionHeight}{90000} + 9 \rceil$$

The other parameters, instead, are the same for any size of the regions. The block size is 2, the block stride is 1, the orientation bins are 6 and the block normalisation scheme is the L2 norm.

3.2.3 Combination of the algorithm and discussion

Protobject computes the similarity index between images through the linear combination of the aforementioned algorithms as follows:

$$\begin{aligned} SI = \{&PearsonCorr([R1, G1, B1, R2, G2, B2, \ldots], [R1, G1, B1, R2, G2, B2, \ldots]) \cdot 0.5 \\ &+CosineSimilarity(HoGDescImage1, HoGDescImage2) \cdot 0.5)\} \cdot 1000 \end{aligned}$$

Since both algorithms give values that range from 0 (lower similarity) to 1 (higher similarity), their weight was set to 0.5 (balancing their effect). Moreover, the resulting value is multiplied by 1000. In this way, the combined SI given by the above formula ranges from 0 (lower similarity) to 1000 (higher similarity).

The combination of these two algorithms ensures a good reliability against the lighting variances that could occur during the use of the systems designed and prototyped through Protobject. Evaluations displayed in section 7.3 show the robustness of the combined algorithm.

3.3 Detection of continuous changes

When detecting continuous movements such as opening a door, opening of a curtain or the position of a continuous knob (e.g. the volume knob of a Hi-Fi amplifier), Protobject can detect the actual positions only according to the positions (states) previously

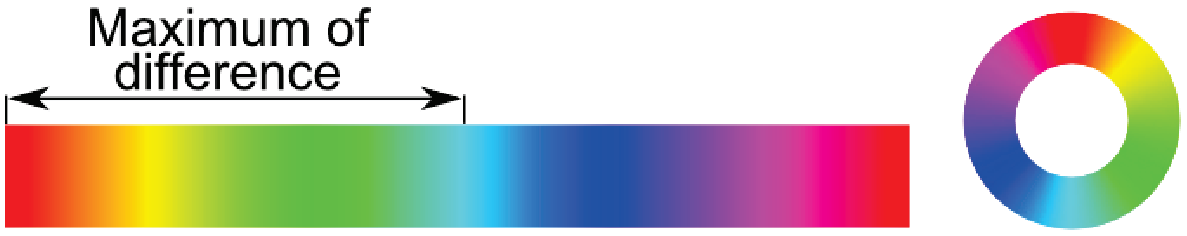


Figure 3.3: The coloured marker and the colour wheel.

learned. If the actual position were not equal to a position previously learned, Protobject would just recognize the nearest position among the learned ones. For example, if Protobject learned ten equidistant positions regarding the opening angle of a door (i.e., from 0 to 90 degrees at an interval of 10 degrees) and the actual angle of the door were 8 degrees, the state actually detected by Protobject would be 10 degrees since it is the nearest to 8 degrees. Therefore, although Protobject does not allow an actual regression, it works satisfactorily in any case by detecting the nearest state among the trained ones.

Thanks to the fact that Protobject detects the nearest state among the learned ones, the precision of Protobject when detecting continuous movements especially depends on the number of learned states. In this regard, section 7.5 discusses some evaluations on the precision of Protobject when detecting continuous movements.

3.4 Detection using markers

3.4.1 Coloured markers

Protobject lets users add coloured markers to objects to enable or improve the tracking of states that, otherwise, would not be observable (e.g. rotation of a symmetrical objects like continuous knobs). When these objects have properly attached coloured markers, different colours correspond to different positions (states). In this way, Protobject can perceive clear differences between the images recorded in the database of states and, therefore, it recognizes the state of the object.

The coloured markers I designed are based on the *colour wheel*. The colour wheel (Figure 3.3-right) displays *with continuity* all the visible colours (i.e., the visible spectrum). Moreover, complementary colours (which are somehow the “maximum of difference” between two colours) are placed on the opposite side of any diameter. A strip with the colour wheel (starting from red, conventionally, see Figure 3.3-left) displays all the visible colours (from red to red) continuously. Moreover, the “maximum of difference”

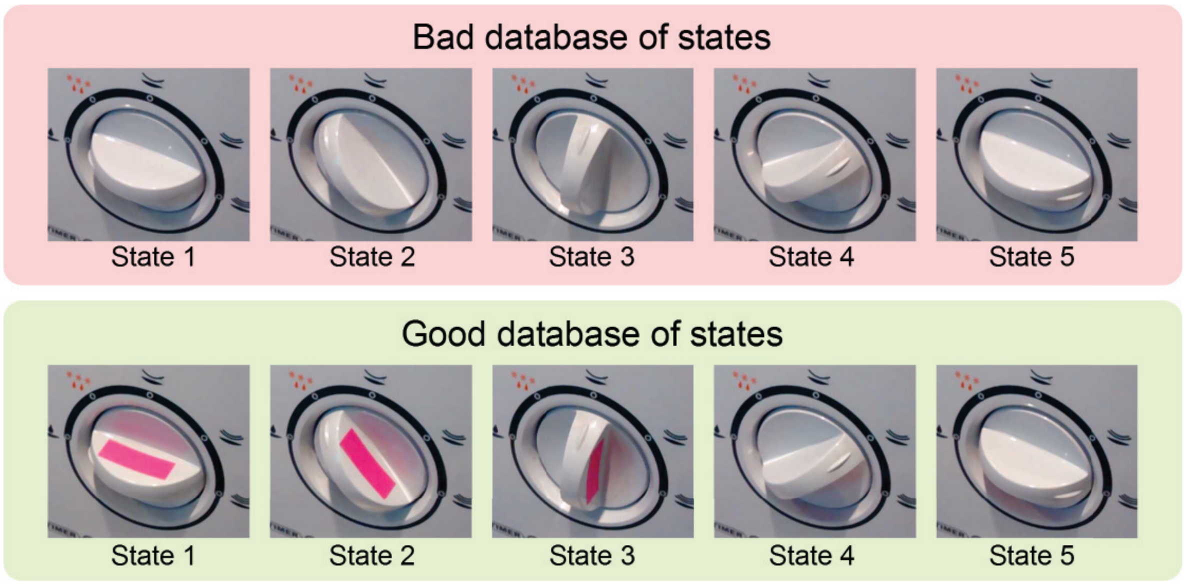


Figure 3.4: In the bad database of states above, "state 1" and "state 5" look very similar and therefore, Protobject makes detection errors. In the good database of states below, in which a simple coloured marker was added to the knob, instead, state 1 and state 5 look very different letting Protobject avoid any detection error.

from the red colour (at the extremes of the strip) can be found in the middle of the strip (cyan colour in Figure 3.3-left).

As shown later in two usage scenarios (sections 6.2.3 and 6.2.4), the colour wheel features (i.e. *continuity* and maximum of *differences* between opposite colours) can be leveraged to easily enable the state recognition of different kinds of objects that move continuously (e.g. knobs, linear potentiometers). Continuous movements are recognized leveraging the vicinity among similar colours. This means that little movements correspond to little variations of colour (i.e., colours that are near each other, e.g. different red tonalities). On the contrary, ample movements correspond to ample variations of colours (i.e., colours that are somehow distant to each other, e.g. red and cyan).

Finally, different kinds of colour-wheel-based markers can be leveraged allowing movement recognition with different precisions (I discuss and evaluate the precision given by two different colour markers in section 7.5.1).

3.4.2 Other simple markers

Sometimes, there are objects whose change of state does not produce enough changes of its visually observable features. Also in these cases, simple markers can help Protobject to recognize the states.

For example, Figure 3.4-Above shows the database of states of a discrete knob (five steps), in which “state 1” and “state 5” look very similar. Then, Protobject could make errors when detecting those states. In this and other similar cases, adding a simple marker (e.g. a scotch tape), can be enough to let Protobject distinguish easily those states avoiding detection errors. Figure 3.4-Below shows the database of states in which “state 1” and “state 5” look very different thanks to a simple violet marker attached to a side of the knob.

3.4.3 Discussion on features of the different markers

Protobject usually does not need any marker for detecting discrete states. As I discuss in section 7.3, I tested Protobject for detecting the state of many objects, also changing lighting conditions, and I never used any simple marker to facilitate the recognition. Simple markers are just a possible option to be used in borderline cases (i.e., when the recognition results are inadequate like in the case of Figure 3.4).

Markers based on the colour wheel, instead, can be used for detecting continuous movements. In some cases, they must be used to make the recognition possible, i.e., when continuous movements do not produce any change of the visible properties of the object (e.g. rotation of continuous knobs as the ones displayed in the usage scenarios in sections 6.2.3 and 6.2.4). In other cases, the detection can work also without those markers, e.g. when detecting the opening angle of a door (e.g. see section 7.5.2).

3.5 Avoiding false detection

As discussed in section 3.1.2, defining regions trying to avoid interferences with the user’s interactions is a good practice, but there are cases where it could be not enough. For example, interacting with a button, the hand of the user is necessarily prone to interfere with the detection (e.g. the case in section 6.2.4). In those cases, there could be false detections during the user’s interaction. Nevertheless, Protobject can be configured to avoid false detection identifying and ignoring the cases in which the interaction of the user interferes with the detection. This filtering occurs exploiting a threshold value that can be optionally set by the designer for each region. If SI values were lower than the threshold, none of the possible state would be detected.

For example, Figure 3.5 shows the detection of a button (its possible states are “ON” and “OFF”) considering three possible cases:

1. The SI computed with the “ON” image (i.e., 992) is higher than the one computed with the “OFF” image (i.e., 964). Therefore, since the SI computed with the “ON”

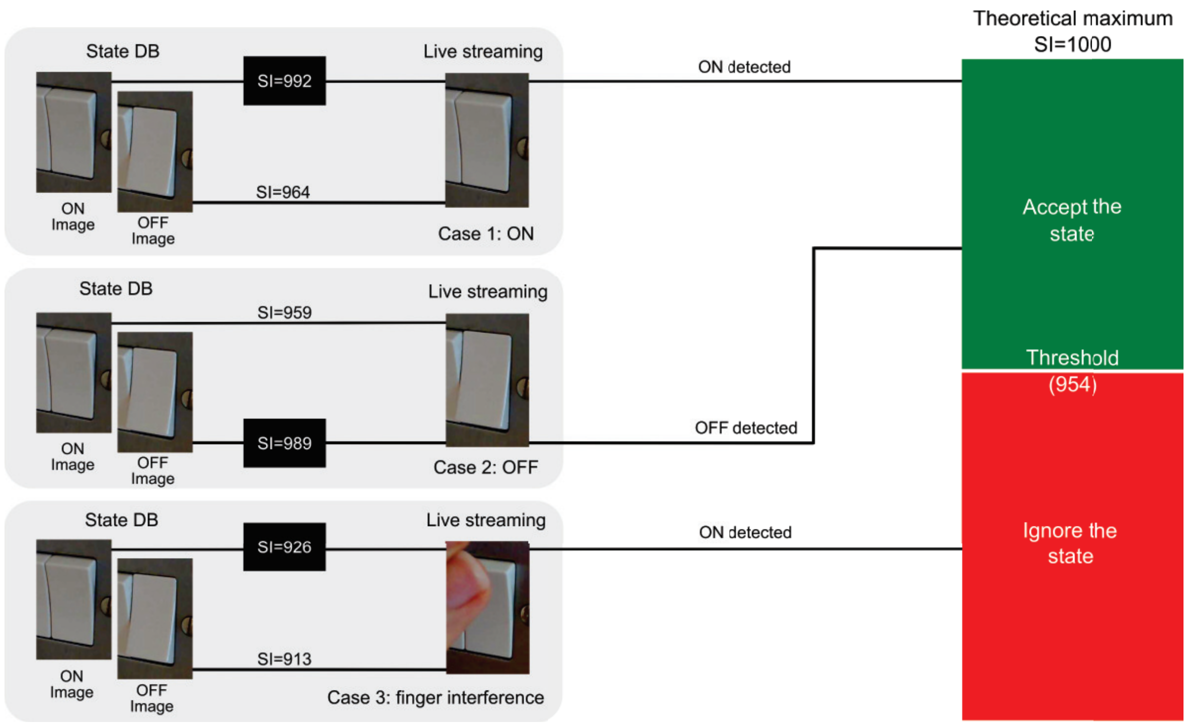


Figure 3.5: Threshold usage to avoid false detections. Case 3 shows the interference of the user's finger during the interaction. At any rate, Protobject ignores this case since the higher SI (which is 926) is anyway lower than the threshold (which is 954).

image is also higher than the threshold (i.e., 954), Protobject detects the “ON” state.

2. The SI computed with the “OFF” image (i.e., 989) is higher than the one computed with the “ON” image (i.e., 959). Therefore, since the SI computed with the “OFF” image is also higher than the threshold (i.e., 954), Protobject detects the “OFF” state.
3. The SI computed with the “ON” image (i.e., 926) is higher than the one computed with the “OFF” image (i.e., 913). Nevertheless, the SI computed with the “ON” image is lower than the threshold (i.e., 954) and Protobject does not detect anything.

Designers can identify the right threshold values through empirical tests when modelling the experiment, prototype or system.

3.6 Detecting presences

Protobject can be used also for detecting generic presences (e.g. people sitting on a sofa, a dog lying in its doghouse, or smartphones leant on a table). In order to detect presences, Protobject needs to record just the image of the absence (e.g. Figure 3.6-left). Therefore, if an absence occurred, the SI value would be near 1000 (e.g. 997 for the case 1 in Figure 3.6) since the captured image would be very similar to the image of the absence. Otherwise, if a presence occurred, the SI value would be far from 1000 (e.g. 839 and 806 for the case 2 and 3 in Figure 3.6) since the captured image would be quite different from the image of the absence. Therefore, users are allowed to detect generic presences leveraging threshold values suitably set at a certain distance from 1000. For example, a good threshold value for the case in Figure 3.6 would be 928.

Protobject works detecting *presences* only when the database of states is composed by *one image* (i.e., the image of the absence, as displayed by Figure 3.6-left). When the database of state is composed by *two or more images*, Protobject works detecting *states*. When detecting presences, Protobject uses the cosine similarity between HoG descriptors in any case.

In the case study of the umbrella stand displayed later (section 6.2.1), I show how to exploit threshold values for detecting people in front of a door. The recognition of presences is evaluated in section 7.4.

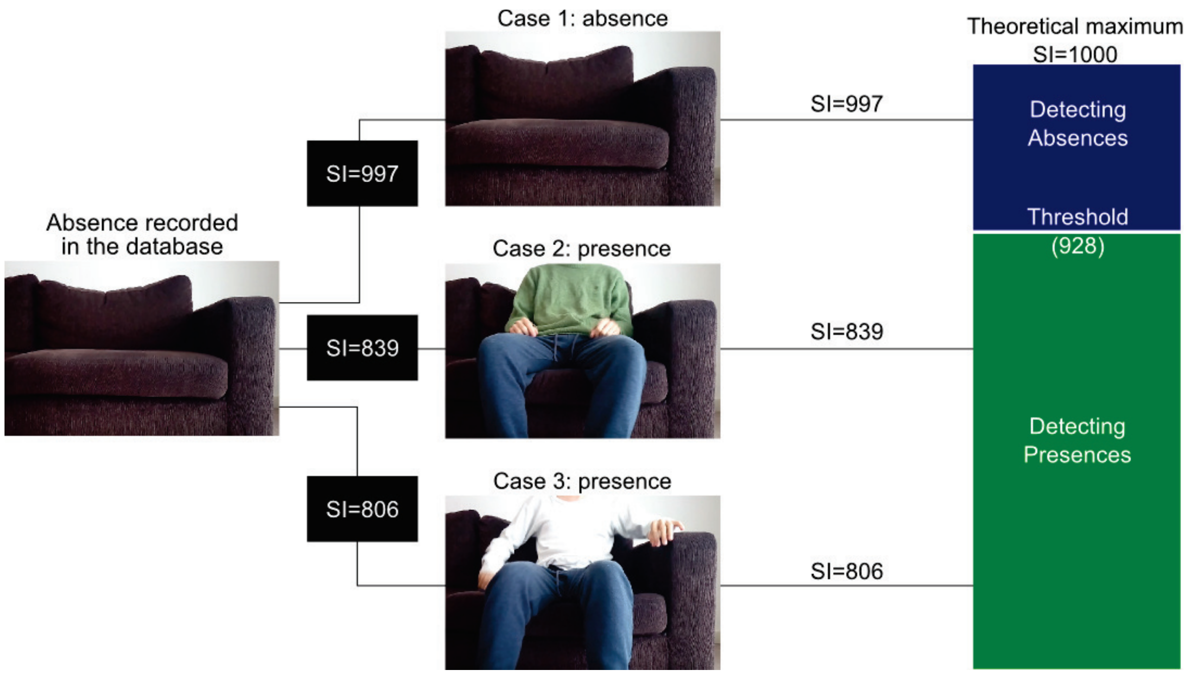


Figure 3.6: Detection of a presence on a seat. In the case 1, the SI (997) is higher than the threshold (928), and the absence is detected. In the case 2 and 3, the SIs (839 and 806) are lower than the threshold (928), and presences are detected.

Chapter 4

Protobject platform

Protobject platform is composed of (a) two main applications that are mutually dependent and (b) two external frameworks for facilitating the development of smart applications.

a. The two main applications are:

    i. a smartphone app (described in section 4.1) that turns a smartphone into a Wi-Fi camera;

    ii. a desktop application (described in section 4.2) that gets images from the smartphone camera and allows the definition of object states and the recognition of such states dispatching them to external components through an integrated WebSocket server.

b. The two external frameworks (described in section 4.3) that facilitate the development of smart applications are:

    i. a JavaScript framework to be used for defining interactive behaviour of smart apps at code level;

    ii. a visual framework based on Meemoo [O$^+$12] that allows non-tech-savvy users to visually connect modules to easily define simple interactive behaviour of smart apps.

Both frameworks work as external components being independent from the two main applications (this means that other external frameworks or components could be added at any time regardless of the two main applications). Even if the two frameworks are different, their operation way is quite similar: they get states from the WebSocket server

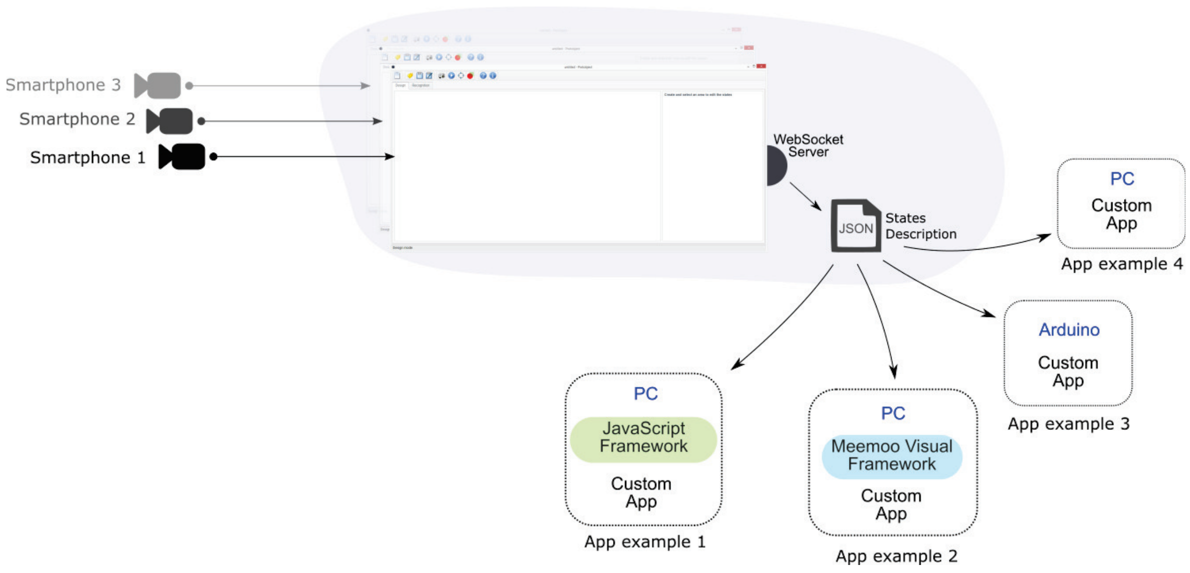


Figure 4.1: Protobject platform architecture.

to generate events according to state changes. Such events are then available and can be used easily for programming smart applications.

Figure 4.1 illustrates the architecture of Protobject with examples of possible applications:

- App example 1 is executed on a PC and programmed using the JavaScript framework for Protobject.
- App example 2 is executed on a PC and (visually) programmed using Meemoo framework for Protobject.
- App example 3 is executed on Arduino and programmed using Arduino programming language (i.e., a set of C/C++ functions[1]) managing the states coming from the WebSocket manually at code level.
- App example 4 is executed on a PC using any programming language managing the states coming from the WebSocket manually at code level.

4.1 Smartphone app

The smartphone app turns a smartphone into a Wi-Fi camera and sends the video stream in M-JPEG format to the desktop application (see next section). Protobject is

[1]https://www.arduino.cc/en/Main/FAQ

resolution independent, i.e., it works with cameras of any resolution. The smartphone app uses the best resolution provided by the smartphone automatically. Resolution is anyhow limited to 1280x720 in order to avoid latency in the image; in fact, higher resolution would require more bandwidth and images could be not fluent.

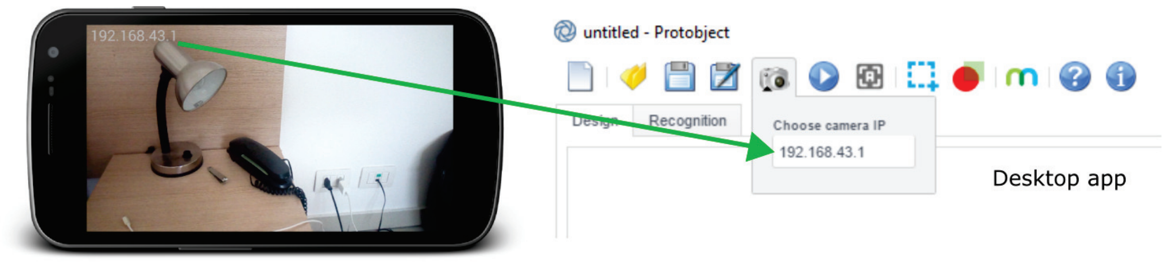


Figure 4.2: The smartphone displays the local IP address of the camera to be connected with Protobject desktop application.

Once the smartphone app is executed, it shows the local IP address to be connected with Protobject desktop app (Figure 4.2). Currently, only a version for Andriod is available, but I plan to devolop an iOS version in the near future.

4.2 Desktop application

Protobject desktop application (Figure 4.3) is a multi-platform application (developed using NW.js) that can be executed on any PC Desktops with Windows, Linux or Mac OS.

4.2.1 The user interface

Protobject desktop application works in two modalities: design and detection mode. The design mode lets Protobject record the possible states of the objects (i.e., building databases of states). The detection mode lets Protobject detect states of objects according to the states previously recorded in the design mode.

Design Mode. After connecting and positioning the smartphone camera towards the target objects, users can create regions (e.g. the semi-transparent regions on the plug and the lamp in Figure 4.3-1 and 2) drawing them with the mouse. Next, they can

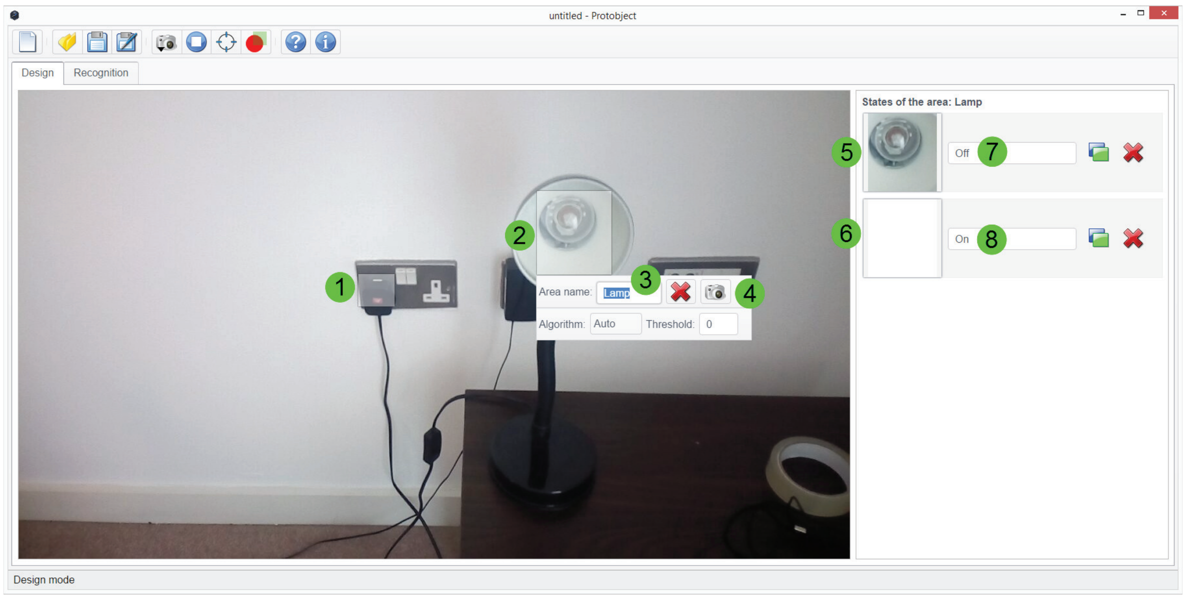


Figure 4.3: Screenshot of the Protobject in design mode. Two regions are used for detecting the plug and the lamp. The states of the selected region (lamp, and therefore "Off" and "On") are displayed at the right.

assign a name to each region (e.g. Lamp, Figure 4.3-3). For each region, users must record the possible states they want to detect (by clicking on the button in Figure 4.3-4). Each state (e.g. "Off" in Figure 4.3-5 and "On" in Figure 4.3-6) is associated with a return value that can be written in the fields next to it (Figure 4.3-7 and 8).

Of course, interacting with the objects is needed to record these 'pictures of states'. For example, to build the states of the lamp in Figure 4.3, it is needed to turn it on to record the "On" state and turn it off to record the "Off" state.

Once a new object is created and defined, it can be saved (Figure 4.3, see Protobject toolbar). Object specifications are saved into a specific file format (*.ptj). It is a JSON containing all the configurations of the object to be detected by Protobject. Protobject saves the regions (name, position and dimension) with the recorded images of the states (database of states). Of course, objects previously saved can be loaded, or new (blank) objects can be created.

Detection Mode. Enabling the detection mode, each region displays the current state (e.g. the state names previously inserted in Figure 4.3-7 and 8) in real time (e.g. Figure 4.4-1 and 2). The number below the state name (displayed with green background) is the similarity index that indicates the similarity between the detected

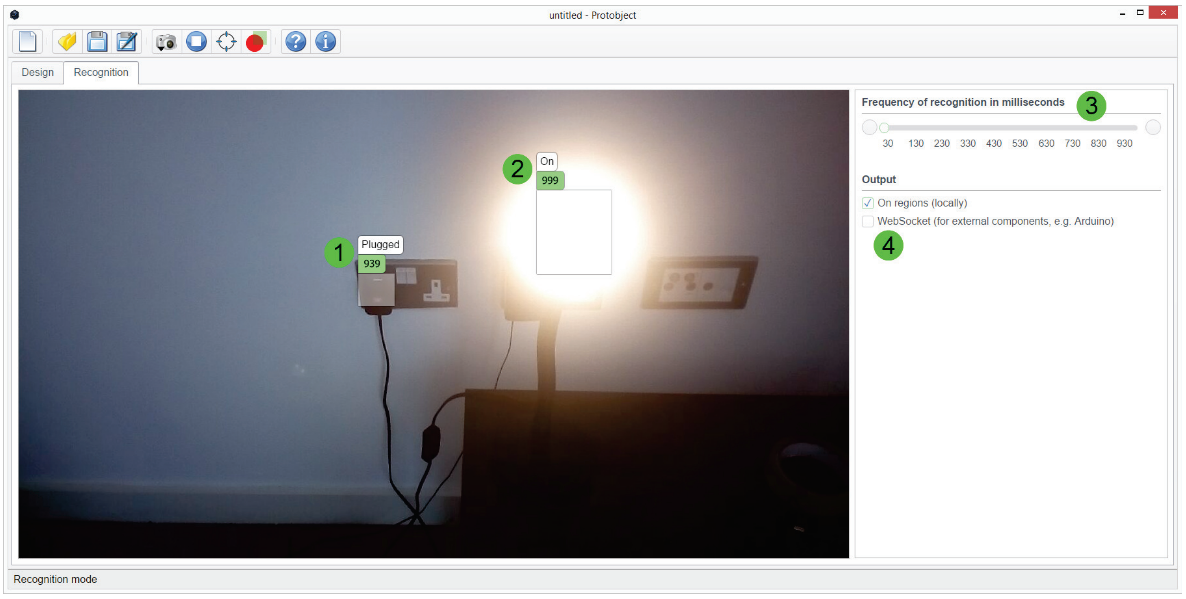


Figure 4.4: The region of the lamp and plug display their current state ("Plugged" and "On" respectively) in recognition mode.

state and the recorded state[2].

States are refreshed at a certain frequency (Figure 4.4-3). Enabling the WebSocket [Fet11] output (Figure 4.4-4), Protobject can send the detected states to other components (e.g. frameworks and/or external applications that behave according to the received states).

4.2.2 Technical details

State recognition frequency. States are detected over time at a frequency that can range from 30 milliseconds to one second, but it can slightly vary depending on Wi-Fi speed, network congestion and number and size of the regions to detect.

Simultaneous use of multiple cameras. Protobject desktop application can use different Wi-Fi cameras simultaneously. In fact, Protobject can be executed in different instances at the same time (on a single computer) and each of them supports a Wi-Fi cam. States detected on the different instances of Protobject are merged and managed by the WebSocket server described in the next subsection.

[2]Protobject similarity indexes range from 0 (lower similarity) to 1000 (higher similarity). Further details are showed in section 3.2

WebSocket server for sending states to external components. Protobject desktop application embeds a WebSocket [Fet11] server that is used as a connection bridge between the different instances of Protobject and external components (e.g. Arduino, frameworks, external apps and so on). Protobject sends the states of objects (i.e., JSON data) to external components that process them. A unique WebSocket server is shared among all the instances of Protobject executed on a single computer. For these aims, Protobject desktop application generates a JSON containing the states of objects. For example, the JSON object generated according to the scenario of Figure 4.4 (where the regions "Plug" and "Lamp" were defined) would be the one displayed in Code 4.1.

Code 4.1: JSON containing the states of the scenario in Figure 4.4

```
[{
  "region": "Plug",
  "state": "Plugged"
}, {
  "region": "Lamp",
  "state": "On"
}]
```

4.3 Frameworks

To facilitate the development of smart applications, I designed two frameworks specifically designed for Protobject. They behave as external components and get states from the desktop application through the WebSocket server.

The first uses Meemoo [O$^+$12], which is a data-flow programming language [JHM04] (like Max Cycling 74 [Puc91] and Pure Data [P$^+$96]) that lets non-tech-savvy users to define their interactive behaviour in a visual way, connecting and combining components specifically designed for Protobject.

The second framework uses JavaScript, a high-level, dynamic, untyped, and interpreted programming language [BTL$^+$13]. Moreover, JavaScript is also highly accessible since it is directly supported by any browser without requiring any additional software (compiler, IDE, etc.).

Further details on those frameworks are discussed in chapter 5.

Chapter 5

Protobject design frameworks

Software frameworks are commonly used to facilitate the development of software applications. In this regard, I designed two frameworks for Protobject following two directions: the first one uses Meemoo [O^+12], a visual programming language for which many modules specifically designed for Protobject were designed. The second one uses JavaScript, a popular programming language that is commonly used also by designers, especially for Web development.

5.1 Meemoo: a visual programming language

5.1.1 Background

Meemoo (Figure 5.1) is a modular framework that lets users create simple applications using predefined components without writing any code. Like Max Cycling 74 and Pure Data, Meemoo is a programming environment designed especially for audio, video, and graphical processing, but it can be easily used for defining any interactive behaviours. In this regard, I designed specific modules[1] for Meemoo to make it work with Protobject.

[1]Modules could be also called "modelling constructs" since they allow the definition of application behaviour according to the "composition model" of Meemoo. Anyhow, I prefer to call them "modules" as in the original definition of Meemoo. Finally, note that in Max Cycling 74 and Pure Data such modules are called "objects".

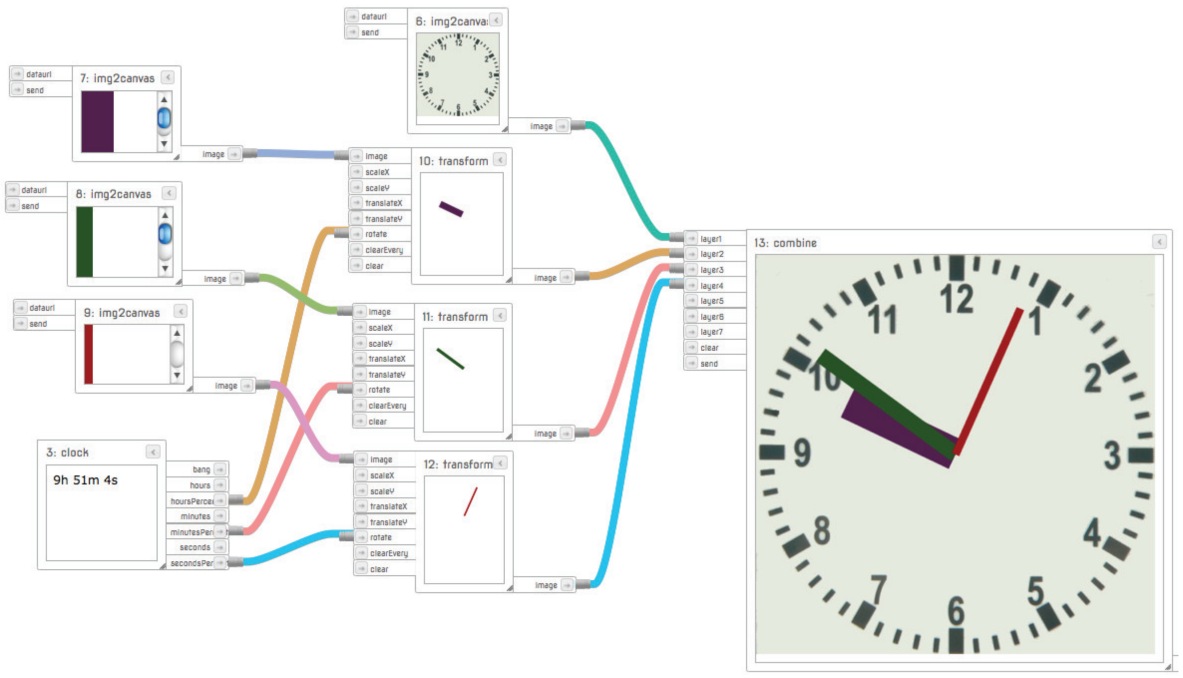


Figure 5.1: Meemoo application example: modules can have inputs and/or outputs and are connected by tubes.

5.1.2 Protobject modules for Meemoo

Several modules were designed to allow non-tech-savvy users to develop interactive behaviour using Protobject (Figure 5.2).

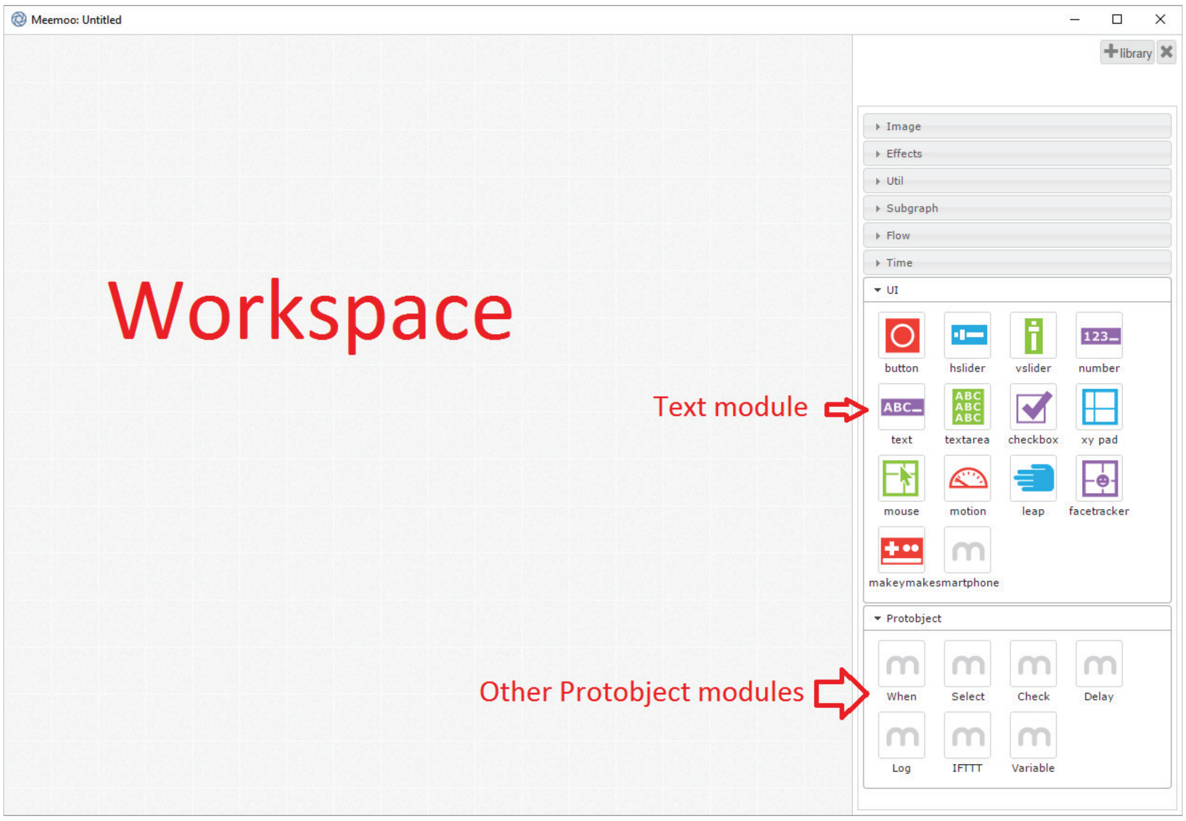


Figure 5.2: Meemoo framework for Protobject: the workspace (gray area at the center) and the modules designed for Protobject (at the right).

For describing how these modules work, I use an example: a smart umbrella stand. It simply works alerting the user with a smartphone notification if, contemporarily, (1) it is raining, (2) he/she leaves home (3) leaving the umbrella in the umbrella stand.

The smart umbrella stand uses three regions (Figure 5.3):

1. door: to understand if the main door is open or closed;
2. presence: to understand if there is someone in front of the door (presence recognition, as discussed in section 3.6);
3. umbrella: to understand if the umbrella is in the umbrella stand or not.

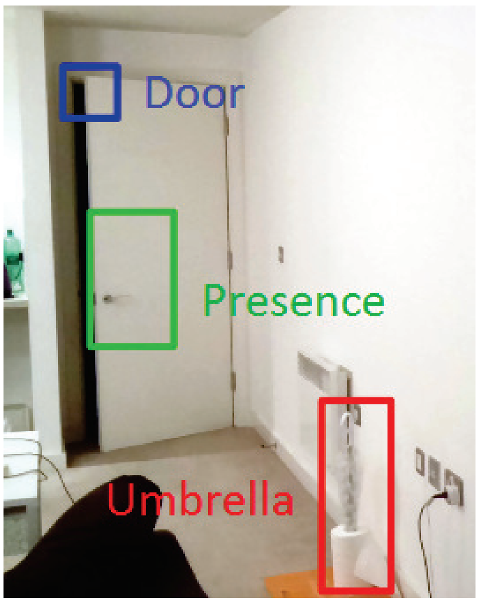


Figure 5.3: Necessary regions for the smart umbrella stand.

A detailed description of the smart umbrella stand prototype is presented in section 6.2.1.

When module

The "when" module is connected with Protobject desktop application through its WebSocket. It gets states from Protobject and sends a signal (i.e., bump, as described in [O+12]) when one of the selected (ticked) states changes. For example, when the door state changes from open to close and vice versa (Figure 5.4).

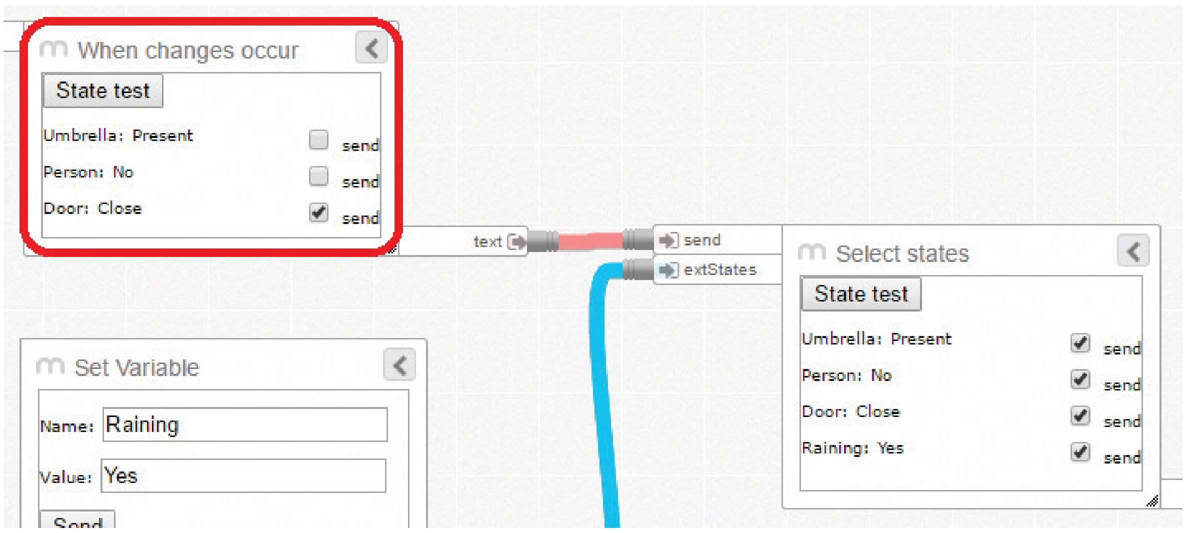


Figure 5.4: The when module defined with Meemoo.

Variable module

The variable module simulates external variables in the system that are not possible to get or define using the graphical programming languages. I used this component in the smart umbrella stand to simulate that it is raining outside (raining: yes) as displayed in Figure 5.5.

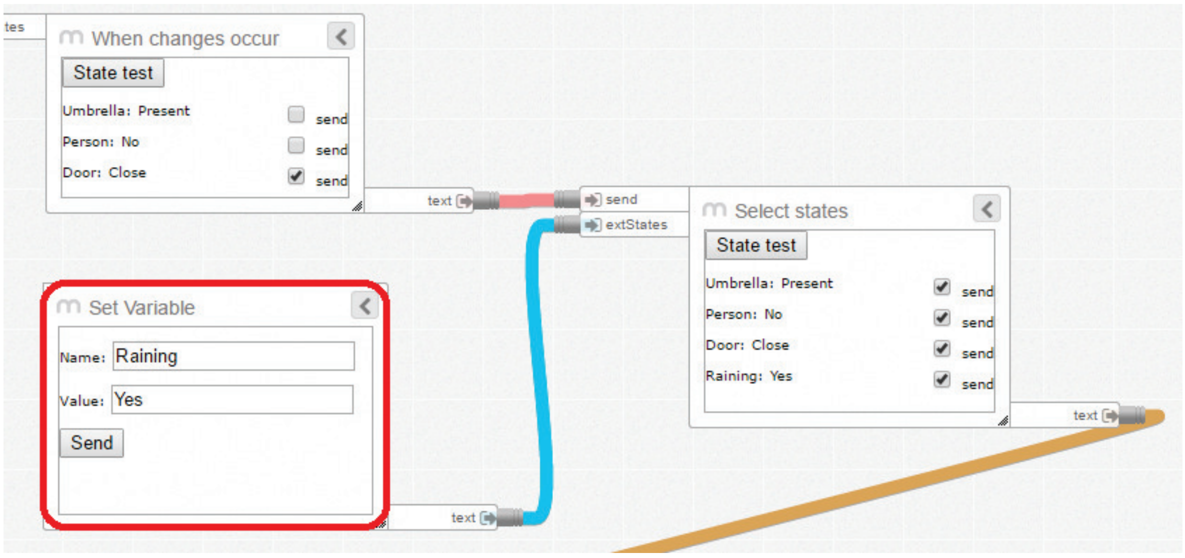
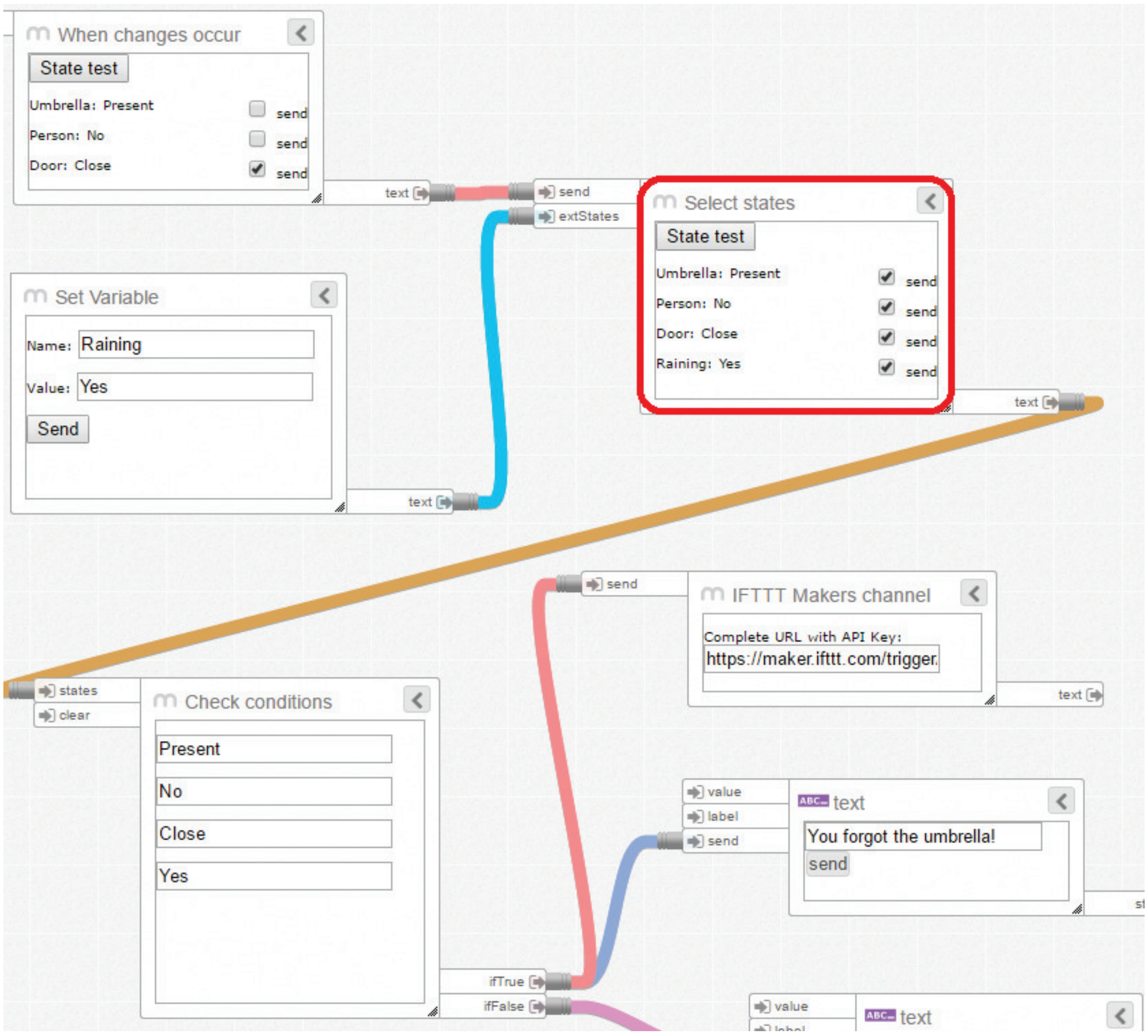


Figure 5.5: The variable module defined with Meemoo.

Select module

The select module is connected with Protobject desktop application through its WebSocket. It lets users select (tick) the states of interest and sends them to the check module for verifying a certain condition. In the example in Figure 5.6, the select module sends four states to the check module.



Umbrella: Present ☐
Person: No ☐
Door: Close ✓

Select states: Umbrella: Present ✓ Person: No ✓ Door: Close ✓ Raining: Yes ✓

Name: Raining
Value: Yes

Complete URL with API Key: https://maker.ifttt.com/trigger.

Present
No
Close
Yes

You forgot the umbrella!

Figure 5.6: The select module defined with Meemoo.

Check module

The check module lets users define the conditions (AND only) needed to trigger specific actions. For example, Figure 5.7 displays the conditions needed to send the notification "you forgot the umbrella" to the user's smartphone.

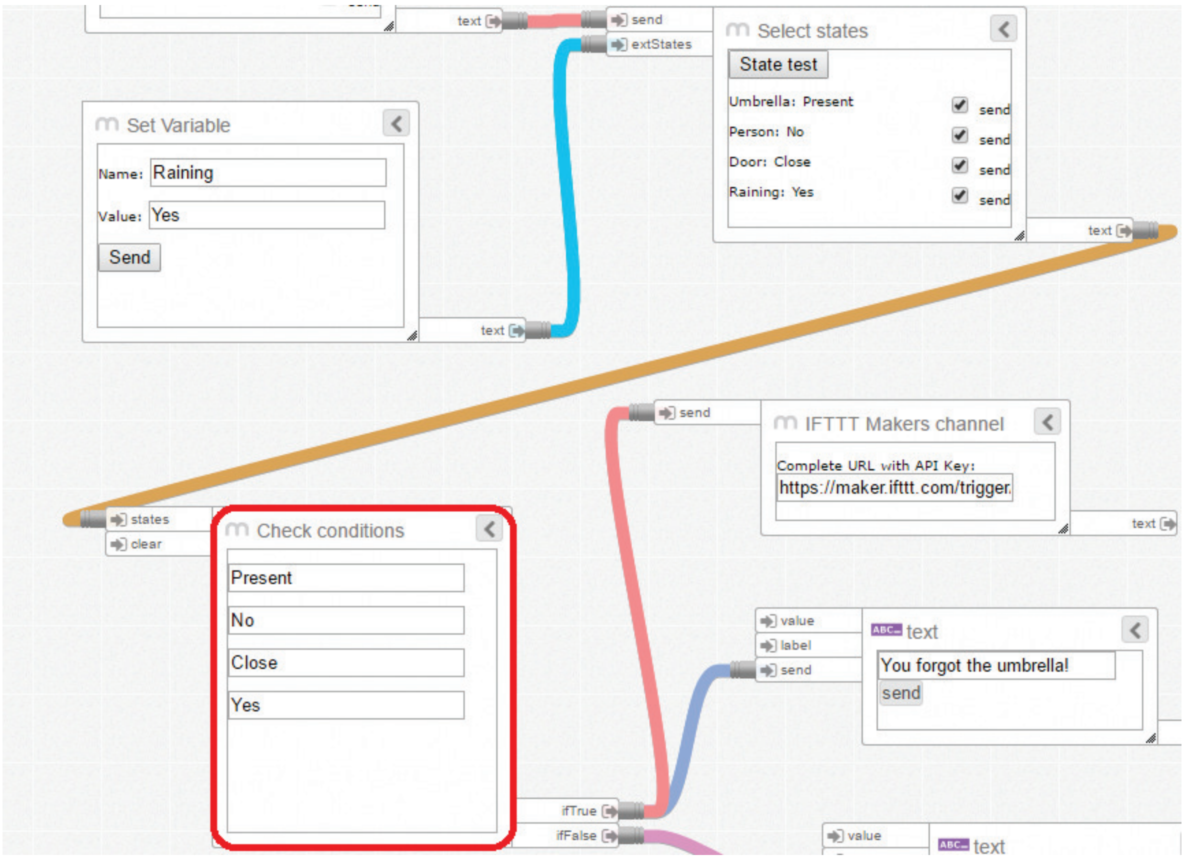


Figure 5.7: The check module defined with Meemoo.

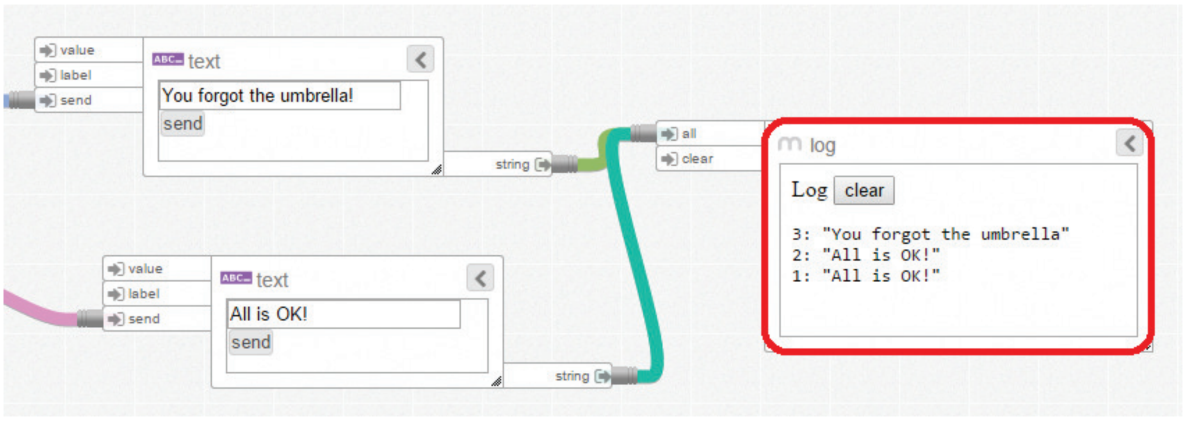
The conditions are:

1. Umbrella: Present (i.e., the user forgot the umbrella in the umbrella stand).
2. Presence: No (i.e., there is nobody in front of the door, which means that the user left home);
3. Door: Close (i.e., the door was just closed);
4. Raining: Yes (i.e., the external variable to simulate the raining condition).

Note that 'raining' is just simulated using an internal variable. In order to have a real raining condition, external APIs should be used and a specific modules should be designed accordingly. In fact, it is hard to design a general module (with a user friendly interface) able to communicate with any existing APIs in the world. Therefore, it is probably easier to use such APIs programming at code level (e.g. using JavaScript). In section 6.2.1, I show how to get the raining condition through external APIs using JavaScript and the weather.com API.

Log module

The log module (Figure 5.8) shows a log of the interactive behaviour. It is used for testing if the prototype works properly.



You forgot the umbrella!

All is OK!

```
3: "You forgot the umbrella"
2: "All is OK!"
1: "All is OK!"
```

Figure 5.8: The log module defined with Meemoo.

IFTTT module

The IFTTT module makes requests through the maker channel of IFTTT [UPYHB$^+$16], a web-based service that allows users to create chains of simple conditional statements. The module is used to send commands to any application whose behaviour is controllable through IFTTT. In the smart umbrella stand, I use this module to send a notification to the smartphone via mail, interfacing the maker channel with the Gmail channel (if Maker Channel is triggered, send a mail notifying that the user forgot the umbrella, Figure 5.9).

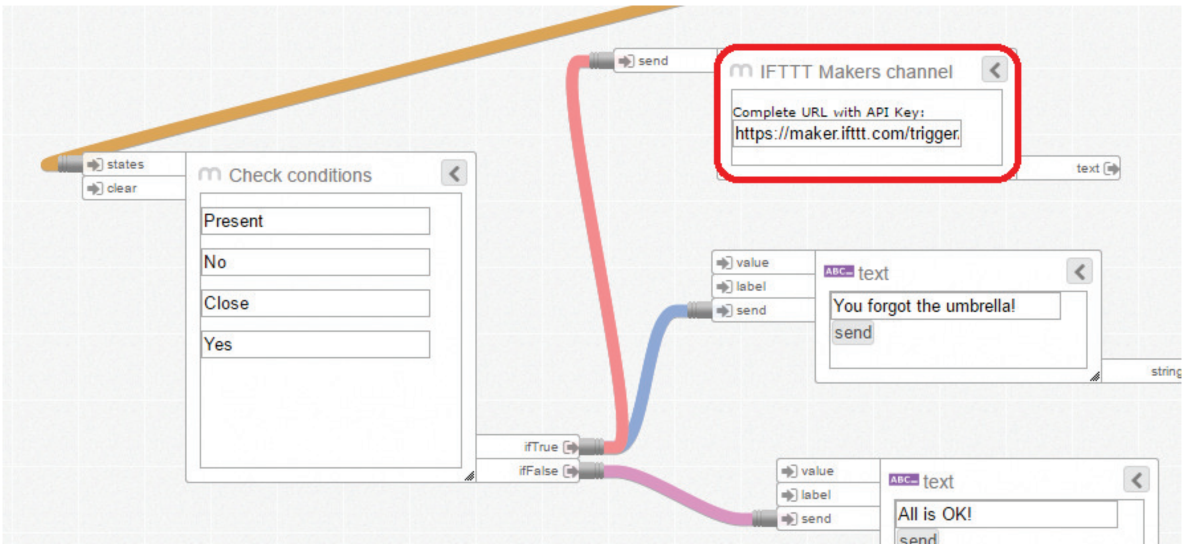
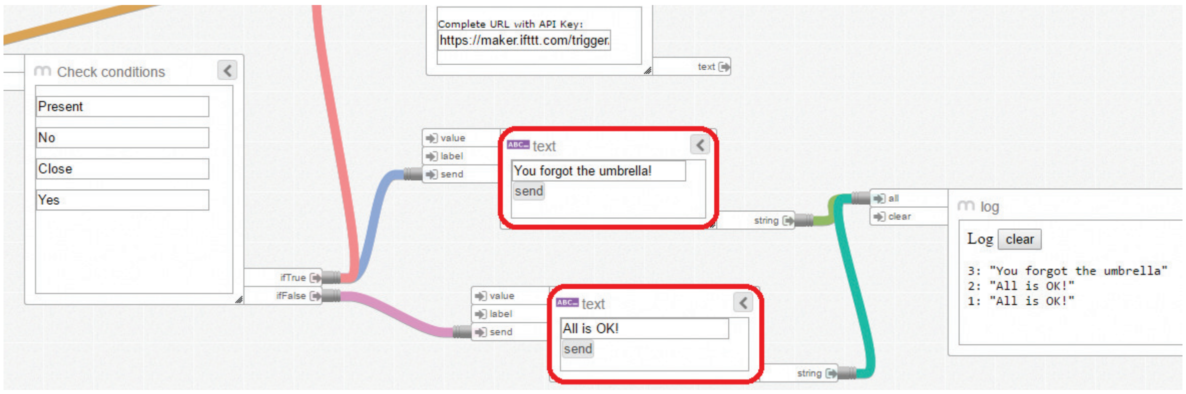


Figure 5.9: The IFTTT module defined with Meemoo.

Text module

The text module (Figure 5.10), already present in the original implementation of Meemoo, is used for sending simple messages according to different behaviour. Those messages are displayed over time using the log module.

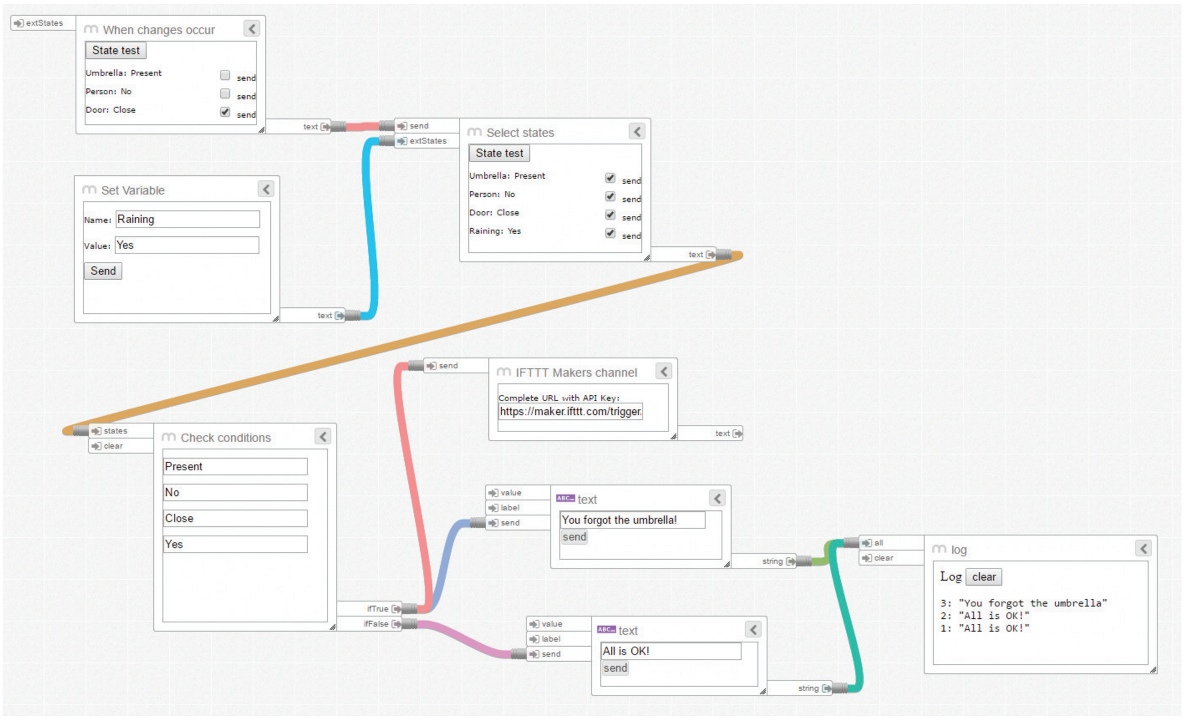


```
3: "You forgot the umbrella"
2: "All is OK!"
1: "All is OK!"
```

Figure 5.10: The text module defined with Meemoo.

Complete smart umbrella stand description

Figure 5.11 describes the entire workflow of the smart umbrella. When the door state changes, the application checks the states of the following variables: door, presence, umbrella and raining (external simulated variable). If the conditions are *Door=Close,* *Presence=No, Umbrella=Present and Raining=Yes*, the application writes on the log "You forgot the umbrella!" and sends a notification (using IFTTT module), otherwise, it writes on the log "Everything is OK!" and does nothing.



Umbrella: Present
Person: No
Door: Close

Umbrella: Present
Person: No
Door: Close
Raining: Yes

Name: Raining
Value: Yes

Present
No
Close
Yes

Complete URL with API Key:
https://maker.ifttt.com/trigger

You forgot the umbrella!

All is OK!

```
3: "You forgot the umbrella"
2: "All is OK!"
1: "All is OK!"
```

Figure 5.11: Smart umbrella prototype defined with Meemoo.

5.2 JavaScript

5.2.1 Background

JavaScript became one of the most popular programming languages thanks to the Web. Besides, I developed a framework for JavaScript because no additional software is needed for making it work. In fact, just a browser and a text editor are enough to prototype smart applications with Protobject.

5.2.2 Application design

Applications are actually HTML5 pages where Protobject framework is included (Code 5.1 line 5). Therefore, the core of the application is written in the body of the page

(Code 5.1 line 14). Finally, opening the HTML page using any browser, the application is executed.

Code 5.1: HTML5 and JavaScript basic application example

```
1  <!DOCTYPE html>
2    <html>
3    <head>
4    <title>Protobject Base App</title>
5      <script src="js/protobject.js"></script>
6    </head>
7
8    <body>
9      Application example
10   </body>
11
12   <script>
13    //Protobject initialization
14     protobject.connect("ws://localhost:8085/");
15     //Application behaviour starts here...
16   </script>
17
18 </html>
```

5.2.3 API

The framework provides several functions for managing the interaction with Protobject. These functions are presented according to the states defined in Figure 4.4, where a Lamp (On/Off) and a Plug (Plugged/Unplugged) were defined.

Connect

Protobject *connect* creates the connection with the Protobject WebSocket server.

Loop

Protobject *loop* is called every time a state is recognized by Protobject. Loop frequency depends on the recognition frequency set on the Protobject desktop application.

Debug

Protobject *debug* shows on the browser console all the states detected by Protobject.

Code 5.2 shows an example of connect, loop and debug used together.

Code 5.2: Loop and debug used together

```
//Protobject initialization
protobject.connect("ws://localhost:8085/");

//loop, managed by Protobject according to the recognition frequency
protobject.loop(function() {
  //Show the states of all the detected regions (plug and lamp) on the browser console
  console.log(protobject.debug());
});
```

Get a state

Protobject *getState* get the state of the region passed as parameter. This is used for programming an action when a state changes.

For example, in Code 5.3, the browser console displays a message when the area "Lamp" is turned on or off.

Code 5.3: Messages when turning on/off the lamp

```
//Protobject initialization
protobject.connect("ws://localhost:8085/");

//loop, managed by Protobject according to the recognition frequency
protobject.loop(function() {
  if (protobject.getState("Lamp")=="On") {
    console.log("The lamp was turned on.")
  } else if (protobject.getState("Lamp")=="Off") {
    console.log("The lamp was turned off.")
  }
});
```

When an event occurs

Protobject *onEvent* triggers an anonymous function when a change of state occurs. In particular, it can trigger a function when (1) the state of a region changes generically, or (2) when the state of a region becomes one specific state.

Code 5.4 shows an example where a function is triggered when the state of the region "Lamp" changes generically. Note that Code 5.4 produces somehow the same result of Code 5.3.

Code 5.4: Messages when turning on/off the lamp (second version)

```
//Protobject initialization
protobject.connect("ws://localhost:8085/");

//when the state of the area "Lamp" changes, the function is triggered
protobject.onEvent("Lamp", function(e) {
  if (e.detail.state=="On") {
    console.log("The lamp was turned on.")
  } else if (e.detail.state=="Off") {
    console.log("The lamp was turned off.")
  }
});
```

Code 5.5 shows another example where a function is triggered when the state of the region "Lamp" changes becoming On and Off. Note that also Code 5.5 produces somehow the same result of Code 5.3 and Code 5.4.

Code 5.5: Messages when turning on/off the lamp (third version)

```javascript
//Protobject initialization
protobject.connect("ws://localhost:8085/");

protobject.onEvent("Lamp.On", function(e) {
  //when the state of the region "Lamp" becomes "On", the function is triggered
  console.log("The lamp was turned on.")
});

protobject.onEvent("Lamp.Off", function(e) {
  //when the state of the region "Lamp" becomes "Off", the function is triggered
  console.log("The lamp was turned off.")
});
```

Different events (depending on changes of states of different regions) can trigger different functions in the same code. For example, Code 5.6 shows messages in the browser console when the lamp is turned off and the plug is disconnected.

Code 5.6: Messages when turning off the lamp and disconnecting the plug

```javascript
//Protobject initialization
protobject.connect("ws://localhost:8085/");

protobject.onEvent("Lamp.Off", function(e) {
  //when the state of the region "Lamp" becomes "Off", the function is triggered
  console.log("The lamp was turned off.")
});

protobject.onEvent("Plug.Unplugged", function(e) {
  //when the state of the region "Plug" becomes "Unplugged", the function is triggered
  console.log("The plug was disconnected.")
});
```

Connect with IFTTT

Protobject *iftttMaker* triggers the maker channel of IFTTT. In the example of Code 5.7, line 8 sends an e-mail to the user smartphone when the plug is disconnected. Note that IFTTT has to be suitably configured beforehand.

Code 5.7: Triggering IFTTT maker channel when disconecting the plug

```javascript
1 //Protobject initialization
2 protobject.connect("ws://localhost:8085/");
3 
4 protobject.onEvent("Plug.Unplugged", function(e) {
5   //when the state of the region "Plug" becomes "unplugged", the function is triggered
6   console.log("The plug was disconnected.")
7   //send the notification through IFTTT, which has to be suitably configured
8   protobject.iftttMaker("https://maker.ifttt.com/trigger/Unplugged/with/key/");
9 });
```

External send

Protobject WebSocket can be used for letting application components exchange generic messages among each other. For this aim, programmers can use the *externalSend* function, which is able to send custom messages through the Protobject WebSocket. This function is used in the Smart Boiler usage scenario (section 6.2.5 and Code 6.5, line 77)

5.3 Discussion

Meemoo, being a graphical programming language, has been used in Protobject, mainly, for unskilled users. On the contrary, JavaScript is more suitable for programmers letting them define also the behaviour of complex applications. In fact, unlike Meemoo, JavaScript lets programmers define real behaviour enabling any kind of connection with external APIs. Meemoo, instead, is often limited allowing programmers just to simulate application behaviour. The next chapter displays how to program several prototypes using both Meemoo and JavaScript putting forward Meemoo limitations.

Chapter 6

Usage Scenarios

6.1 Exploration of possibilities

Many use scenarios emerge when exploring the range of possibilities enabled by Protobject. The exploration occurred through several informal sessions where I presented Protobject to makers, designers, researchers, developers, architects and entrepreneurs both inside our university and in other contexts such as Fablabs and co-working spaces. Their aim was to explore possibly usage scenarios, but I also collected other types of feedback.

6.1.1 Sessions with users

Selected users

I interviewed 1 entrepreneur, 3 designers (one of them has a background as interior designer), 14 researchers and 5 developers. I also arranged three focus groups: the first with 3 developers, the second with an entrepreneur, a maker and an architect and the third with 5 makers. Therefore, the total number of people to whom I presented Protobject was 34. All the participants were selected through personal contacts both inside and outside the university.

Procedure

Both for interviews and focus groups, I introduced Protobject displaying its user interface and its operation way. Moreover, I made on-the-fly demos on how Protobject could be configured and used to detect interactions and states. The objects I used for these demos were a door (detecting its positions) and a lamp (detecting its on/off states). Next, I made one example of possible usages of Protobject for prototyping (the Skype scenario I explained in the introduction). Finally, I asked to think of possible novel usages and discuss the potentialities of the prototype. From that moment, both interviews and focus groups were completely unstructured to encourage people to bring forth ideas, thoughts and opinions. During both interviews and focus groups, I put much effort into discovering possible novel usages of Protobject. In fact, in the last interviews, when the discussions were about to finish, I introduced other ideas – also coming from the interviews and focus groups conducted previously – to revive the discussion and encourage the emergence of other new ideas. The last interviews were particularly enriching.

6.1.2 Exploration of results

As I expected, users gave me many ideas that I developed later. They are described in this section as usage scenarios. I could not develop all the ideas that emerged because of their variety. Anyway, to show the range of possibilities opened by Protobject, I discuss briefly the most interesting ideas that emerged.

Some of the ideas regarded security. For example, Protobject could be used to check the closing of doors, taps, or check if the cookers knobs are closed rightly. In general, many interactive systems concerning security could be easily prototyped thanks to Protobject. Other ideas regarded objects whose states, due to their nature, are difficult to detect. Examples are liquid soap or toilet paper. Protobject could be easily used for prototyping scenarios in which detecting the state of these objects is needed, e.g., to check if liquid soap or toilet paper are running out. As it is possible to imagine, the majority of the ideas regarded systems that encompass connected home, communicating devices and their remix. Our houses are full of electronic devices equipped with LEDs and LCD displays. Protobject can read them and, therefore, get the state of these devices. These pieces of information could trigger different actions. For example, when playing a DVD, the room lights dim automatically; or when the washing machine stops, a notification is sent to the user's smartphone, and so on.

Summarizing, I could state that the main topics discussed by users were (1) IoT and smart home prototyping, (2) energy awareness or saving tools, (3) tangible interfaces and the (4) reusing, remixing, recycling of existing objects. For each of these topics, I developed at least one idea as usage scenario.

6.2 Introducing the usage scenarios

According to the aforementioned exploration, I designed and developed six usage scenarios to test the prototyping potentialities of Protobject. Four usage scenarios are carried out using just Protobject whereas the remaining two usage scenarios are carried out using Protobject in connection with an Intel Edison board (Arduino compatible). There are cases, in fact, as I mentioned previously, in which Protobject cannot be used alone, needing to be extended by Arduino, especially when active components such as LCD displays, LEDs, relays or servos are requested to be controlled.

I used Protobject alone to design (1) a smart umbrella stand; (2) an energy saving tool; (3) a tangible interface to control a light; (4) a radio controller reinventing the use of a Hi-Fi amplifier. Instead, I used Protobject in addition to an Intel Edison board to design (5) a smart boiler and (6) an energy awareness tool.

Presenting each of these cases, I put forward and discuss different Protobject features. The usage scenario 1 shows how to recognize people in front of a door. Usage scenarios 2 and 6 show how to use multiple instances of Protobject for managing two different cameras. In usage scenarios 3 and 4 coloured markers are used. In addition, usage scenario 3 shows a strongly bricoleur-oriented example in which I designed some interaction primitives, i.e., buttons and two kinds of potentiometers (using paperclips, pens, bottle caps, drawing pins and rubber bands put in a corkboard) to be observed by Protobject. Finally, usage scenarios 5 and 6 show how to manage an Intel Edison board together with Protobject using WebSocket connections.

6.2.1 Usage scenario 1: a smart umbrella stand

Description

This usage scenario shows the prototyping of a smart umbrella stand able to send a notification (i.e., an email) to the user's smartphone if (1) he/she leaves home and (2) forgets the umbrella when (3) it is raining (an external API is used for this scope). It happens that when users leave home, they realize that it is raining and have to go back home. In some cases, this is annoying since users have to take the elevator, lift many floors, open the door, take the umbrella and go out again. The smart umbrella stand, is designed to send a notification, if needed, just after leaving home avoiding the aforementioned (annoying) procedure. Figure 6.1 displays the prototyping scenario: a Motorola Moto G, which works as Wi-Fi camera, was placed on a selfie stick that was putted on top of a tripod (see the right part of Figure 6.1). Three regions (see Figure 6.1) were defined on the video stream. The blue region, which selects the upper part of the door to avoid the interference of people walking around, is used to sense if the

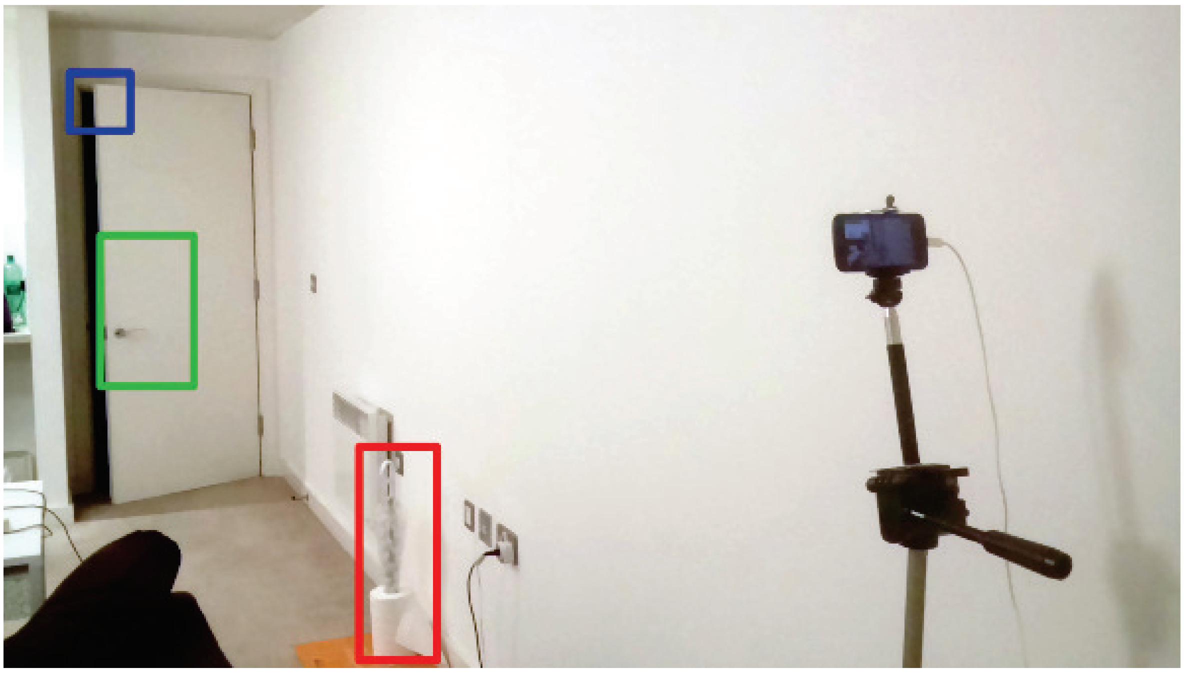


Figure 6.1: Smart umbrella stand prototyping scenario

door is open or closed. The green region is used to sense the presence of someone in front of the door. The red region is used to understand if the umbrella is in its stand or not. Making reference to the bricoleur approach, note that the umbrella stand has been built reusing a kitchen roll (the umbrella is placed in its hole).

With the states detected in the three regions, we can have two general cases creating an ambiguity: (1) the user opens and closes the door remaining at home; (2) the user opens the door, leaves home, and closes the door. Of course, the operating algorithm should react only to the second case. For this reason, the green region is used to detect if someone is occluding the door. I captured just the state of the closed door since I exploit the similarity index between the closed door and the live video stream. If this value is lower than a certain threshold (i.e., the detected state *is not* similar to the state of the closed door), it means that someone is in front of the door (presence recognition, as discussed in section 3.6). In this way, after the closing of the door, if the user is in front of it, this means that he/she remained at home. On the contrary, after the door has been closed, nobody is in front of it, this means that the user has just left home. Only in this case, if it is raining and the umbrella is still in the umbrella stand, the notification (e.g. a mail) is sent immediately to the user's smartphone. This prototype does not require a high frequency of recognition. Therefore, I set Protobject so that states are refreshed every 200 milliseconds.

Discussion

The entire prototyping process of the smart umbrella stand system takes around 25 minutes using the JavaScript framework: 10 minutes are used to set the environment (i.e., positioning the smartphone and capture the possible states of red, green and blue region) and 15 to develop the software, consisting of 24 lines of code (HTML + JavaScript, see Code 6.1).

The application uses the *weather.com* API to understand if it is raining or not (lines 12-17, Code 6.1) and the IFTTT maker channel to send the notification through the Gmail channel (line 20, Code 6.1).

Code 6.1: The smart umbrella stand prototype

```
 1  <!DOCTYPE html>
 2  <html>
 3    <head>
 4      <title>Protobject Smart Umbrella Stand</title>
 5      <script src="framework/protobject.js"></script>
 6    </head>
 7    <body>
 8    <p>Smart umbrella stand</p>
 9    </body>
10    <script>
11      protobject.connect("ws://localhost:8085/");
12      raining=false;
13      $.getJSON('http://api.openweathermap.org/.../weather?q=Milan,IT', function( data ) {
14        if (data.weather[0].main=="Rain") {
15          raining=true;
16        }
17      });
18      protobject.onEvent("Door.Close", function(e) {
19        if (protobject.getState("Umbrella")=="Present" && protobject.getState("Person")=="No"
        && raining==true) {
20          protobject.iftttMaker("https://maker.ifttt.com/trigger/Umbrella/key")
21        }
22      });
23    </script>
24  </html>
```

The prototyping process takes around 20 minutes using Meemoo: 10 minutes for setting the environment and 10 to arrange, set and connect components (Figure 5.11). Unlike the version designed with the Protobject JavaScript framework, the Meemoo version cannot get the rain state since no module was developed with this aim. The raining state, therefore, has to be simulated using a variable (see Figure 5.5). In any case, mail notifications can be sent using the IFTTT module for Meemoo.

Once the software is developed, this prototype is portable in any similar scenario (e.g. in Figure 6.2) in which a door and something able to host an umbrella is available (anything, containing an umbrella so that it is visible, can be used) without being concerned of reinstalling devices, wires, sensors, or of modifying hardware. In fact, changing the environment, users would need just 10 minutes to make the prototype work again. They just should set the position of the smartphone and capture again the states of the new objects in the new environment.

Portability can be useful in many cases. Besides the ones in which users want to try different design alternatives, there could be cases in which designers wish to show their

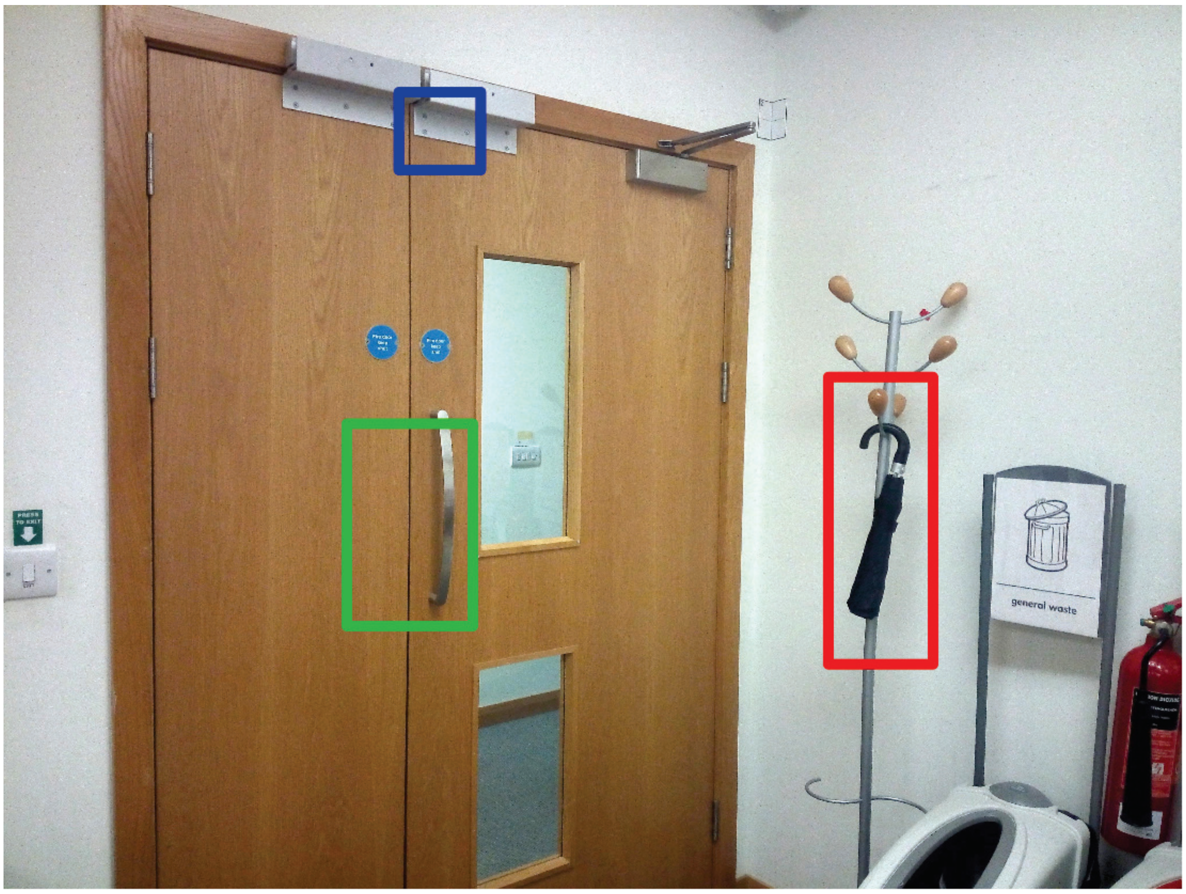


Figure 6.2: The smart umbrella stand, previously designed for the scenario in Figure 6.1, was readapted to the scenario displayed in this figure. The blue region was used to sense the opening/closing of the door; the green region was used to sense the presence of someone in front of the door; the red region was used to sense the presence of the umbrella (note that a coat hanger was reinvented as umbrella stand).

ideas to possible investors directly in their office/house (this would be not possible using traditional approaches unless designers were allowed to install sensors anywhere). Moreover, it is noteworthy that just one smartphone is needed to sense three different things at the same time: (1) the presence of the umbrella, (2) the opening/closing of the door and (3) the presence of someone who is occluding the door. Using traditional prototyping approaches, instead, a switch should be installed on the door to detect its opening/closing and a proximity sensor should be installed on, for instance, a traditional umbrella stand (to detect the presence/absence of the umbrella). Moreover, the position of people should be detected using dedicated geolocation systems for internal spaces, e.g. Cisco CMX[1] (traditional GPS or Wi-Fi localization do not work well in internal spaces).

Finally, a limitation of the Protobject approach in this usage scenario is that it is possible to detect just if a generic person is going out without being able to identify specific people. Using dedicated geolocation systems, instead, people could be identified by their smartphone. Anyway, I believe that this limitation is negligible considering early stages of prototyping.

6.2.2 Usage scenario 2: energy saving tools

Description

Energy saving has become crucial recently with the emergence of green movements. In this regard, Protobject can be used for prototyping tools able to signal wasting behaviours. In particular, Protobject can be used to make crosschecks in the environment to understand eventual conditions in which energy is wasted.

This usage scenario shows an energy saving tool able to make a crosscheck to understand if a window is open while the heater is working. In such a case, a notification is sent to the user's smartphone signalling the wastefulness. The crosscheck is carried out using two smartphone cameras. The first camera looks at the windows (green arrow in Figure 6.3) whereas the second one looks at the controls of the heater (blue arrow in Figure 6.3). The operating algorithm is quite simple: if the heater is working and the window is open, the system signals the wastefulness sending a notification, e.g. to the user's smartphone, indicating also the current state of the heater (e.g. Comfort, Eco, etc.).

As in the smart umbrella stand case, the usage conditions do not require a high frequency of recognition and I set Protobject so that states are refreshed every 600 milliseconds.

[1]The Cisco Connected Mobile Experiences (CMX) is a suite of mobile software solutions to detect, connect, and engage the mobile devices within a venue. It provides, among others, an indoor location service.

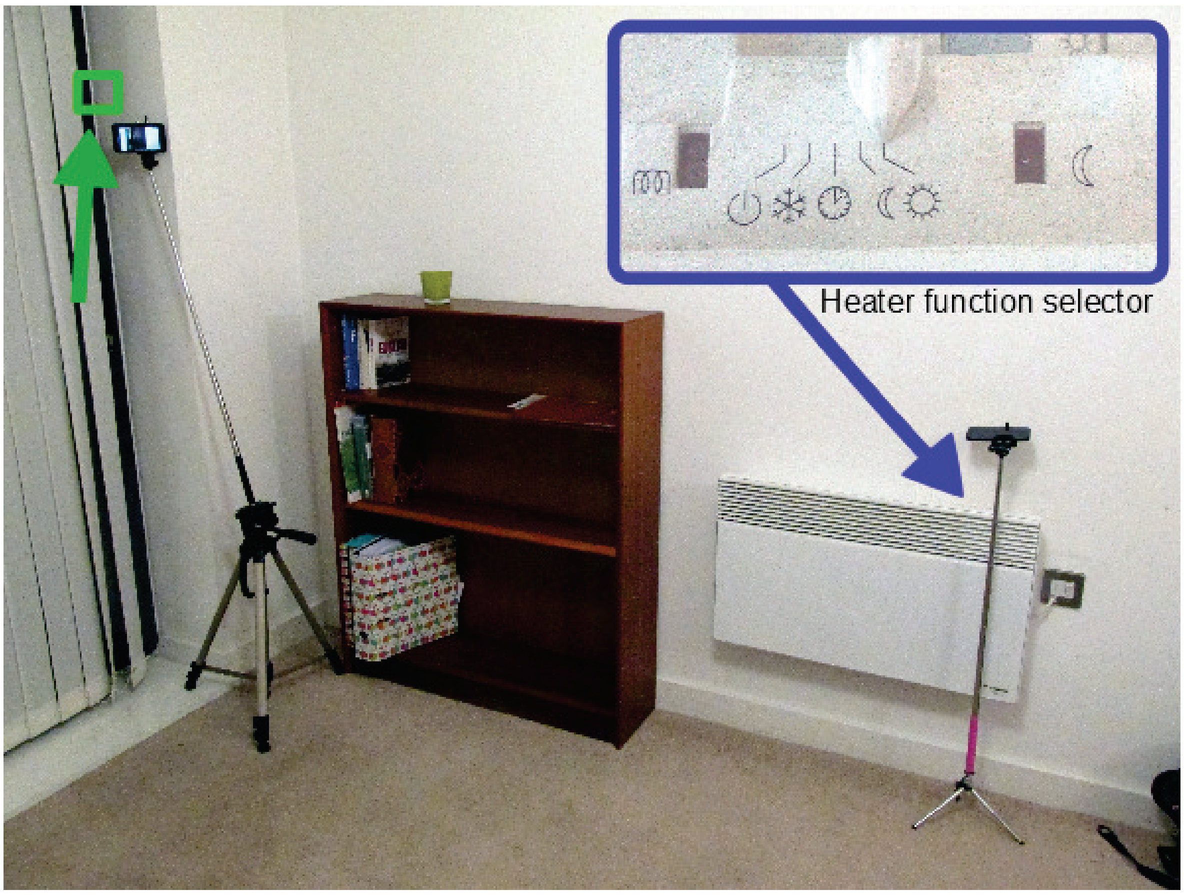


Figure 6.3: Energy saving scenario.

Discussion

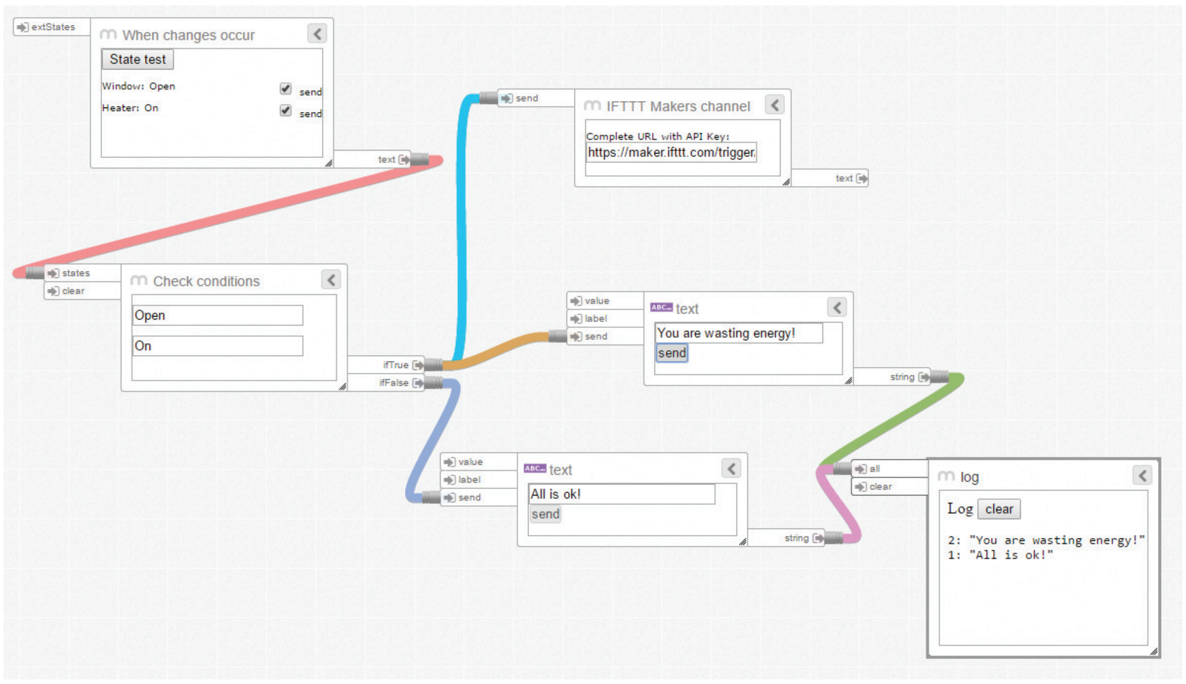
The entire prototyping process of this energy saving tool takes around 25 minutes, 15 of which were used to set the environment and 10 to develop the software. Moreover, the software consists of 23 lines of code (HTML + JavaScript see Code 6.2).

The application uses the IFTTT maker channel to send the notification through the Gmail channel (line 14 and 19, Code 6.2).

Code 6.2: The energy saving tool prototype designed with the JavaScript framework

```
 1  <!DOCTYPE html>
 2  <html>
 3    <head>
 4      <title>Protobject Energy Saving Tool</title>
 5      <script src="framework/protobject.js"></script>
 6    </head>
 7    <body>
 8      <p>Energy saving tool</p>
 9    </body>
10    <script>
11      protobject.connect("ws://localhost:8085/");
12      protobject.onEvent("Window.Open", function(e) {
13        if (protobject.getState("Heater")=="On") {
14          protobject.iftttMaker("https://maker.ifttt.com/trigger/EnergyWaste/key")
15        }
16      });
17      protobject.onEvent("Heater.On", function(e) {
18        if (protobject.getState("Window")=="Open") {
19          protobject.iftttMaker("https://maker.ifttt.com/trigger/EnergyWaste/key")
20        }
21      });
22    </script>
23  </html>
```

The prototyping process takes around 25 minutes using Meemoo: 15 minutes for setting the environment and 10 to arrange, set and connect components (Figure 6.4).



Window: Open
Heater: On

Complete URL with API Key:
https://maker.ifttt.com/trigger

Open
On

You are wasting energy!
send

All is ok!
send

```
2: "You are wasting energy!"
1: "All is ok!"
```

Figure 6.4: Energy saving tool designed with Meemoo

Also in this case, the prototype is lightweight, flexible and portable in a very little time. Just 15 minutes would be needed to re-position smartphones and build the new databases of states according to new hardware (however, note that the alternative hardware should be able to offer the same functionalities as the previous one). Moreover, there is no need of any sensor. Finally, note that in both prototypes I ran two instances of Protobject in order to use two cameras simultaneously.

The presented prototype would be unfeasible using traditional prototyping approaches (Arduino and sensors). Arduino is able to measure the instant consumption of any appliance using specific CT sensors. Anyhow, detecting the current state of the heater (i.e., Comfort, Eco, etc.) is still hard with Arduino or other similar approaches. Therefore, to achieve the same prototyping scope using traditional ways, users should use specific hardware (e.g. Samsung Smart Home products). On the contrary, Protobject allows this prototyping also using old and cheap electronic devices, which do not need any modification (if just detection is required, as in this case).

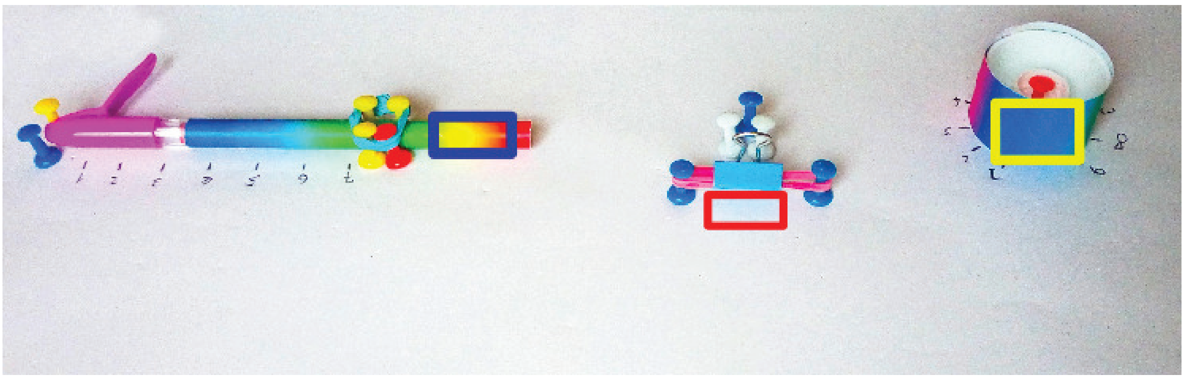


Figure 6.5: Example of an easy-to-build tangible interface.

6.2.3 Usage scenario 3: tangible interfaces

Description

Protobject can be used to capture interactions with tangible interfaces that do not need any electronic component. I designed three kinds of components: buttons and two kinds of potentiometers (i.e., knob and linear). These components were designed so that they can be built using materials easy to find – paper-clips, pens, bottle caps, drawing pins and rubber bands putting them in a corkboard (see Figure 6.5).

The knob potentiometers are designed using a bottle cap fixed to the corkboard using a drawing pin. The latter works also as a rotation pivot. Buttons are designed using paperclips fixed with drawing pins and a rubber band as depicted in Figure 6.5 (near the red region). The linear potentiometers are designed using the body of a pen that moves through four drawing pins inserted in the corkboard and kept together with a rubber band.

I leveraged the colour wheel features discussed in the section 3.4.1 (i.e., *continuity* and maximum of *differences* between opposite colours) in order to let Protobject recognize the state of our potentiometers (both knobs and linear) easily. A colour strip that goes from red to cyan (i.e., the maximum of difference) was put inside the body of the pen (see Figure 6.5). Note that the pen must be transparent so that Protobject can see the strip inside. Another strip that goes from red to red was placed to wrap the bottle cap (see Figure 6.5). In this case, the strip is a continuous band so that the red at the first extreme joins with the red at the second extreme.

The interface in Figure 6.5 was used for controlling a virtual lamp displayed on a computer screen (Figure 6.6). The button turns on or off the lamp, the linear potentiometer regulates its intensity whereas the knob potentiometer is used to change its colour. Protobject senses the interactions with the linear potentiometer through the blue region

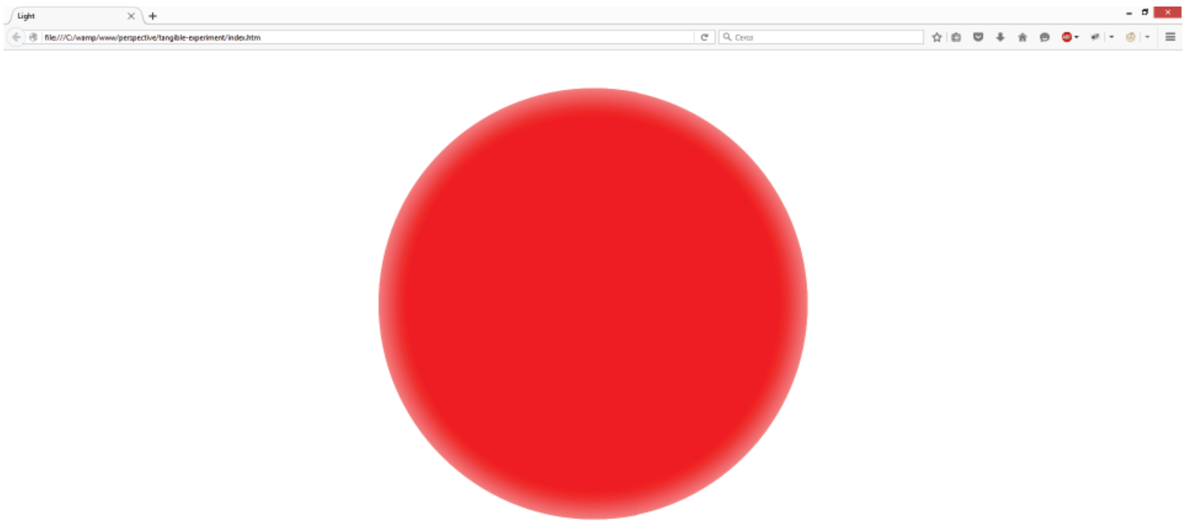

Light
file:///C:/wamp/www/perspective/tangible-experiment/index.htm
Cerca

Figure 6.6: The virtual lamp displayed on the browser.

in Figure 6.5. The knob potentiometer, instead, is sensed through the yellow region in the same figure, and the button through the red region.

The usage conditions of this prototype require a high frequency of recognition (to make interactions reactive). Therefore, I set Protobject so that states are refreshed every 30 milliseconds. Note that the real frequency of recognition actually calculated was around 80 milliseconds.

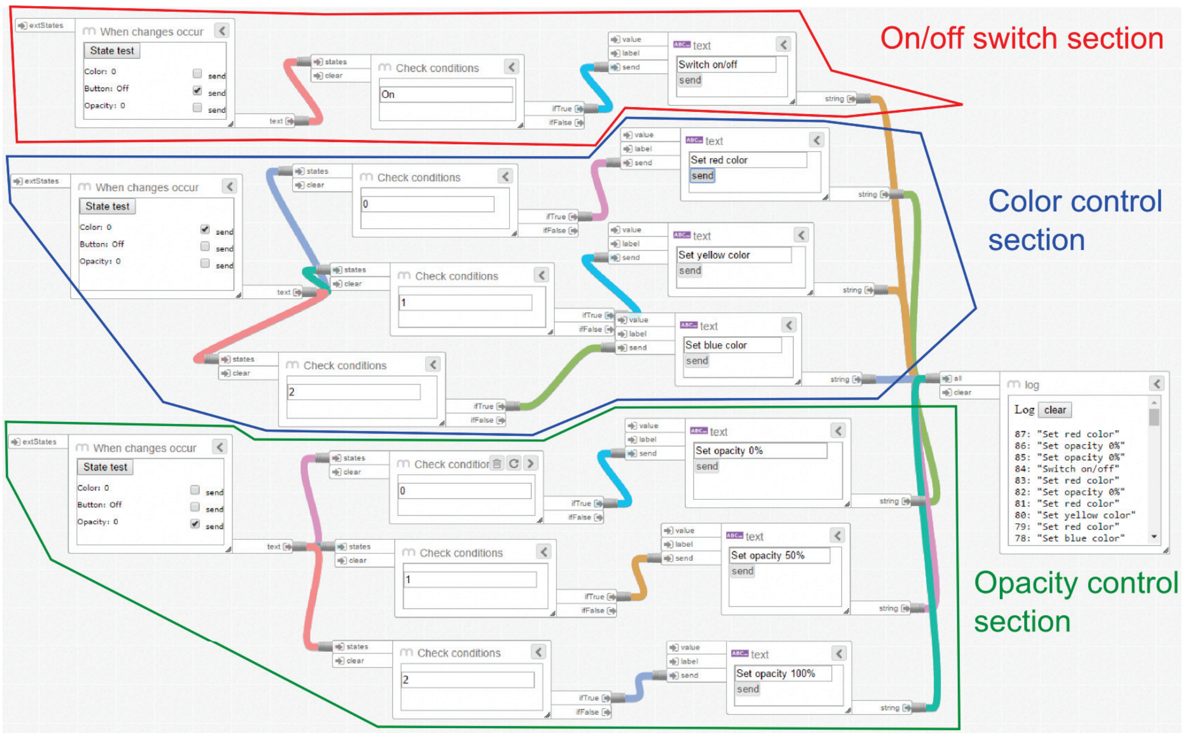
Discussion

The entire prototyping process of this tangible interface takes around 40 minutes of which 15 were used to set the environment and 25 to develop the software. In any case, note that tangible components were designed beforehand. Therefore, during the 15 minute required for setting the environment, I only installed the components in the corkboard, positioned the smartphone and built the database of states. Finally, this application has 48 lines of code (HTML + JavaScript + CSS, see Code 6.3).

Code 6.3: The tangible interface light control designed with the JavaScript framework

```
<!DOCTYPE html>
<html>
  <head>
    <title>Protobject Light Control</title>
    <script src="framework/protobject.js"></script>
    <script src="jquery.js"></script> <!-- This prototype uses jQuery -->
    <style>
    #light {
      background: red none repeat scroll 0 0;
      border-radius: 400px;
      box-shadow: 0 0 86px #fff inset;
      height: 700px;
      left: 50%;
      margin-left: -350px;
      margin-top: -350px;
      position: absolute;
      top: 50%;
      width: 700px;
      transition: 100ms ease all;
      display:block;
     }
    </style>
    </head>
    <body>
      <div id="light"></div> <!-- This is the virtual lamp displayed on the screen -->
    </body>
    <script>
      RGBColorArray=["rgb(255, 0, 0)", "rgb(255, 204, 0)", "rgb(102, 255, 0)",
      "rgb(1, 255, 0)", "rgb(1, 255, 230)", "rgb(0, 77, 255)",
      "rgb(230, 0, 255)", "rgb(255, 0, 179)", "rgb(255, 0, 26)"];
      protobject.connect("ws://localhost:8085/");
      protobject.onEvent("Button.On", function(e) {
        if ($('#light').css("display") == "block") {
            $('#light').css("display","none"); //turn off the light
          } else {
            $('#light').css("display","block"); //turn on the light
          }
      });
      protobject.onEvent("Opacity", function(e) {
        //this is the pen potentiometer with states from 0 to 7
        $('#light').css("opacity",parseInt(protobject.getState("Opacity"))/7);
      });
      protobject.onEvent("Color", function(e) {
        //this is the knob potentiometer with states from 0 to 9
        $('#light').css("background",RGBColorArray[parseInt(protobject.getState("Color"))]);
      });
    </script>
</html>
```

The design of this prototype is not possible using Meemoo since a specific component should be designed to represent the light. Anyhow, it is still possible to simulate how the lamp works using Meemoo exploiting the log module to show the lamp behaviour (Figure 6.7).



```
Log clear

87: "Set red color"
86: "Set opacity 0%"
85: "Set opacity 0%"
84: "Switch on/off"
83: "Set red color"
82: "Set opacity 0%"
81: "Set red color"
80: "Set yellow color"
79: "Set red color"
78: "Set blue color"
```

Figure 6.7: The virtual lamp simulated with Meemoo. The log module shows the lamp behaviour over time. To make this example more readable, only three different colours (red, yellow and blue) and opacities (0%, 50% and 100%) where simulated.

The use of easy-to-build components can increase the speed in prototyping different tangible interfaces. I designed just three basic components, but new ones could be envisioned exploring different interaction primitives (e.g. [LBHH15]). I aim to spread the simplicity of this design approach so that designers and makers can envision new, alternative and more sophisticated components similar to the ones I designed.

Note that using Arduino, a switch and two potentiometers (knob and linear) would be needed. Following this approach, instead, these three components, redesigned in imaginative ways, were sensed using only one smartphone.

A limitation of the Protobject approach in this usage scenario regards the reactivity of the system. Although around 80 milliseconds could seem a good refresh rate, approaches that use sensors are much faster and the difference is somehow perceptible.

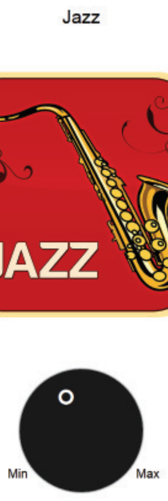


Figure 6.8: The web radio displayed on a browser.

6.2.4 Usage scenario 4: remixing culture

Description

Why not reinvent the way to use objects in order to satisfy new requirements? Makers who participate in the remixing culture [CY09] are used to get inspiration from existing objects, combining them, or adapting their designs [OWM15]. For example, in this usage scenario, I designed a prototype for controlling a web radio (see Figure 6.8) using a Hi-Fi amplifier (Figure 6.9).

The volume knob, which is sensed through the blue region in Figure 6.9, is used to regulate the radio volume. The input selector knob (i.e., Cd, Tuner, etc.), which is sensed through the yellow region in the same figure, is used as channel selector (Jazz, Classical, Rock, etc.). Note that a coloured strip that goes from red to red was placed to wrap the volume knob to allow its recognition. Otherwise, it would be impossible to detect the state of the volume knob since its values are not distinguishable (it would always display the same visible property regardless of its position).

To make the use of the prototype reactive, I set Protobject to sense states every 30 milliseconds (I actually calculated around 80 milliseconds as in the previous case).

Discussion

This is just a usage scenario in which I show how Protobject lets users reinvent, reuse and remix existing objects. Many objects with physical interfaces surround us. They

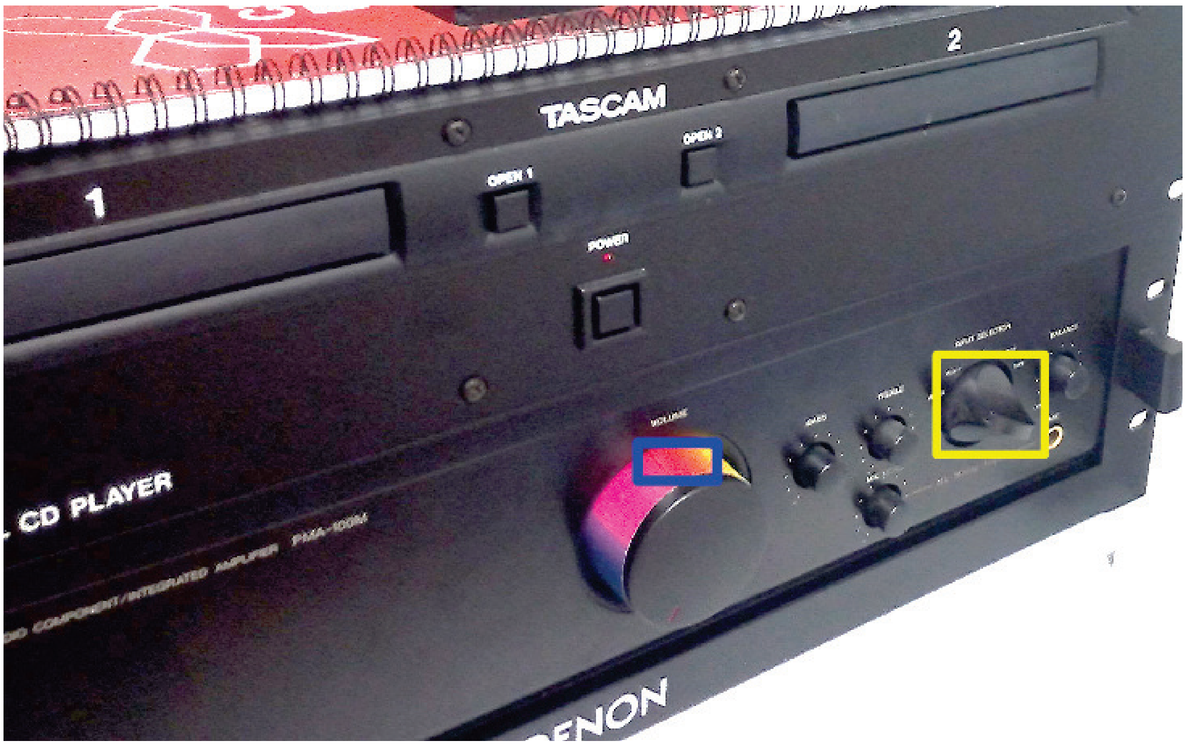


Figure 6.9: A Hi-Fi amplifier turned into a web radio controller.

can be easily reused for the rapid prototyping of novel interactive products and meet new requirements.

The entire prototyping process of this usage scenario takes around 45 minutes of which 10 were used to set the environment and 35 to develop the software. Moreover, this application has 146 lines of code (HTML + JavaScript + CSS).

Code 6.4: The webradio designed with the JavaScript framework

```
<!DOCTYPE html>
<html>
  <head>
    <title>Protobject Web Radio</title>
    <script src="framework/protobject.js"></script>
    <script src="jquery.js"></script> <!-- This prototype uses jQuery -->
    <style>
    #channelContainer {
      color: #000;
      font-family: sans-serif;
      font-size: 30px;
      height: 440px;
      left: 50%;
      margin-left: -225px;
      margin-top: -440px;
      padding-top: 10px;
      position: absolute;
      text-align: center;
      top: 50%;
      transition: all 150ms ease 0s;
      width: 450px;
    }
    #volumeKnob {
      background: #333 none repeat scroll 0 0;
      border-radius: 400px;
      box-shadow: 0 0 2px #000;
      color: #fff;
      font-family: sans-serif;
      font-size: 90px;
      height: 190px;
      left: 50%;
      margin-left: -100px;
      margin-top: 160px;
      padding-top: 10px;
      position: absolute;
      text-align: center;
      top: 50%;
      transform: rotate(-128deg);
      transition: all 150ms ease 0s;
      width: 200px;
    }
    #channelImg {
      display: block;
      margin-left: 20px;
      margin-top: 71px;
      width: 399px;
    }
    #min {
      display: block;
      font-family: sans-serif;
      font-size: 20px;
      left: 50%;
      margin-left: -119px;
      margin-top: 794px;
      position: absolute;
      width: 100px;
    }
    #max {
      display: block;
      font-family: sans-serif;
      font-size: 20px;
      left: 50%;
      margin-left: 89px;
      margin-top: 794px;
      position: absolute;
      width: 100px;
    }
    audio{
      display:none;
    }
    </style>
  </head>
  <body>
    <div id="channelContainer">Off <img id="channelImg" width="250px" src="Off.png" /></div>
    <div id="volumeKnob">&deg;</div>
    <span id="min">Min</span> <span id="max">Max</span>
    <audio id="RadioChannel" controls loop><source src="Radio.mp3"
    type="audio/mpeg"></audio>
    <audio id="JazzChannel" controls loop><source src="Jazz.mp3" type="audio/mpeg"></audio>
    <audio id="ClassicalChannel" controls loop><source src="Classical.mp3"
    type="audio/mpeg"></audio>
    <audio id="PopChannel" controls loop><source src="Pop.mp3" type="audio/mpeg"></audio>
  </body>
  <script>
    protobject.connect("ws://localhost:8085/");  //Protobject initialization
    protobject.onEvent("Channel", function(e) {
      canale=protobject.getState("Channel");
      $("#RadioChannel")[0].pause();
      $("#JazzChannel")[0].pause();
      $("#ClassicalChannel")[0].pause();
      $("#PopChannel")[0].pause();
      $("#channelContainer").html(canale+' <img id="channelImg" width="250px"
    src="'+canale+'.png" />');
      if (canale != "Off")
        $("#"+canale+"Channel")[0].play();
    });
    protobject.onEvent("Volume", function(e) {
      volume=protobject.getState("Volume");
      if (volume=="0") {
        $("#volumeKnob").css("transform","rotate(-132deg)");
        setVolume(0);
      } else if (volume=="1") {
        $("#volumeKnob").css("transform","rotate(-110deg)");
        setVolume(0.083);
      } else if (volume=="2") {
        $("#volumeKnob").css("transform","rotate(-88deg)");
        setVolume(0.166);
      } else if (volume=="3") {
        $("#volumeKnob").css("transform","rotate(-66deg)");
        setVolume(0.249);
      } else if (volume=="4") {
        $("#volumeKnob").css("transform","rotate(-44deg)");
        setVolume(0.332);
      } else if (volume=="5") {
        $("#volumeKnob").css("transform","rotate(-22deg)");
        setVolume(0.415);
      } else if (volume=="6") {
        $("#volumeKnob").css("transform","rotate(0deg)");
        setVolume(0.497);
      } else if (volume=="7") {
        $("#volumeKnob").css("transform","rotate(22deg)");
        setVolume(0.58);
      } else if (volume=="8") {
        $("#volumeKnob").css("transform","rotate(44deg)");
        setVolume(0.663);
      } else if (volume=="9") {
        $("#volumeKnob").css("transform","rotate(66deg)");
        setVolume(0.746);
      } else if (volume=="10") {
        $("#volumeKnob").css("transform","rotate(88deg)");
        setVolume(0.829);
      } else if (volume=="11") {
        $("#volumeKnob").css("transform","rotate(110deg)");
        setVolume(0.912);
      } else if (volume=="12") {
        $("#volumeKnob").css("transform","rotate(132deg)");
        setVolume(1);
      }
    });

    function setVolume(val){
      $("#RadioChannel")[0].volume=val;
      $("#JazzChannel")[0].volume=val;
      $("#ClassicalChannel")[0].volume=val;
      $("#PopChannel")[0].volume=val;
    }
    setVolume(0);
  </script>
</html>
```

The design of this prototype would not be possible using Meemoo since a specific component should be designed to make the radio work. Anyhow, it is still possible to simulate the behaviour of the radio using Meemoo exploiting the log module (Figure 6.10).

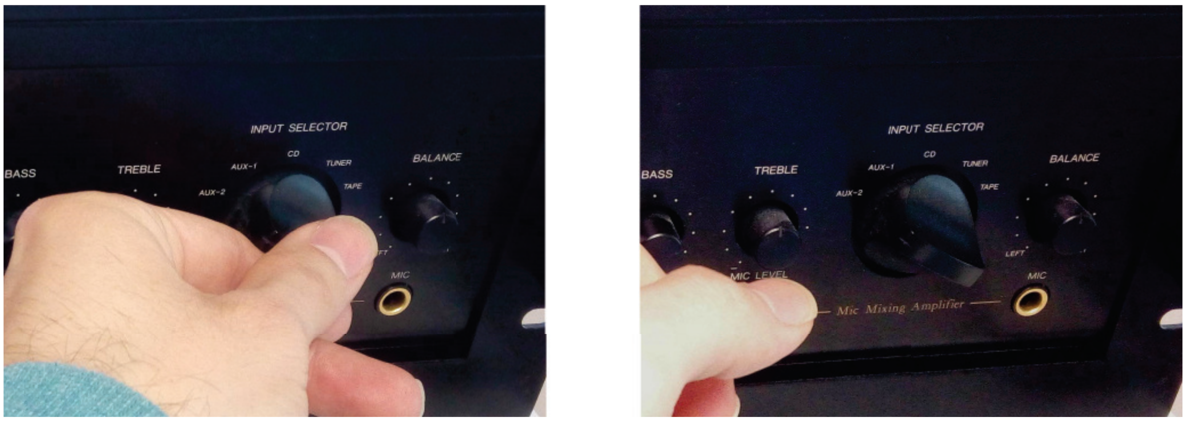
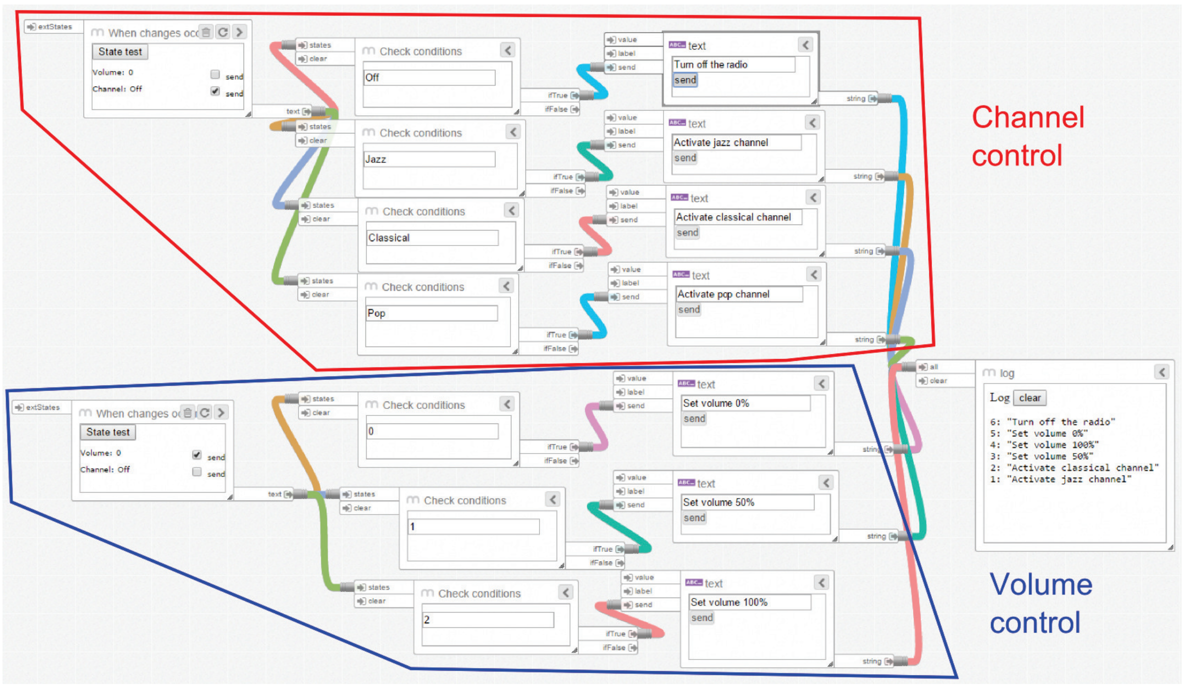


Figure 6.11: The knob selector state can be detected only when users move away their hand from the knob.



```
6: "Turn off the radio"
5: "Set volume 0%"
4: "Set volume 100%"
3: "Set volume 50%"
2: "Activate classical channel"
1: "Activate jazz channel"
```

Figure 6.10: The webradio simulated with Meemoo. The log module shows the webradio behaviour overtime. To make this example more readable, only three different channels (Jazz, Classical and Pop) in addition to Off are simulated. Moreover, only three different volume intensities (0%, 50% and 100%) are simulated.

The use of traditional prototyping approaches (i.e., Arduino and 3D printers) for similar cases (i.e., reusing, reinventing and remixing existing objects) is unfeasible.

A limitation of the Protobject approach in this usage scenario regards, as in the previ-

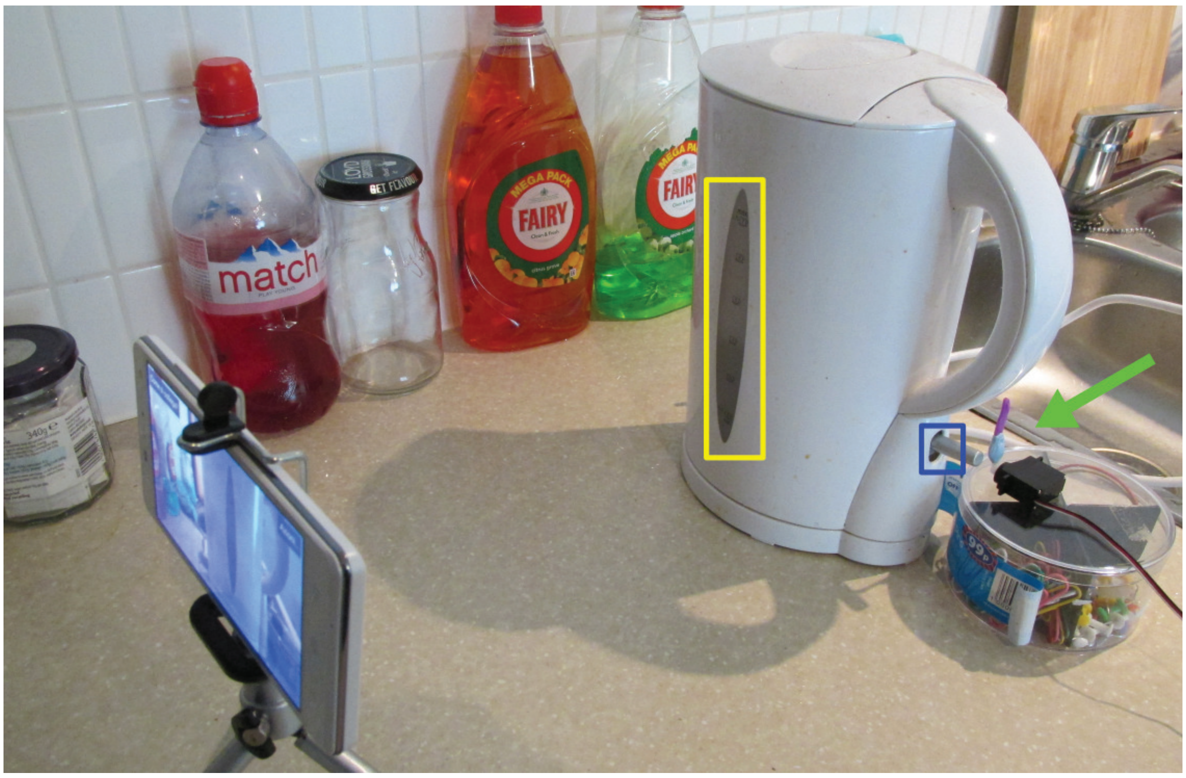


Figure 6.12: The smart boiler in which Protobject senses the liquid level and the state of the switch.

ous case, the reactivity of the system: sensors would offer a better reactivity. Another limitation regards the interactions with the function knob selector. In fact, these interactions cannot be detected immediately when they occur (Figure 6.11-Left), but only after user's hand has been moved away (Figure 6.11-Right), i.e., when Protobject is able to observe the change of state. As described in section 3.5, I exploit a specific threshold value to avoid false detection when the user's interaction interferes with the knob.

6.2.5 Usage scenario 5: smart boiler

Description

This scenario displays a smart boiler designed using Protobject in addition to an Intel Edison board (which is Arduino compatible). For designing it, I leveraged a traditional boiler, which can be mechanically turned on (pushing down a button). Moreover, the boiler is designed to show the water level through a suitable transparent container.

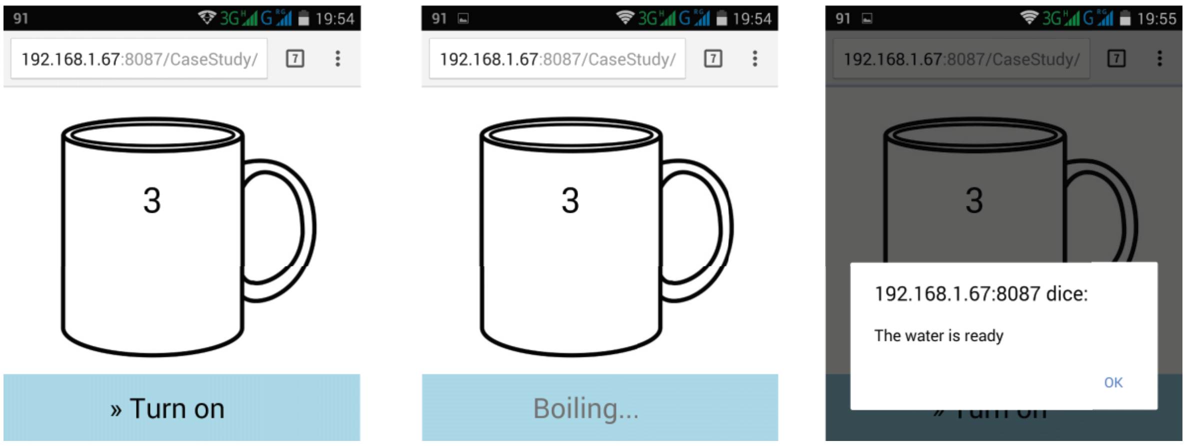


Figure 6.13: The smartphone app of the boiler.

The water level is detected by Protobject using the yellow region in Figure 6.12. In addition, the Intel Edison board manages the turning on of the boiler. To avoid any hardware modification on the boiler, I just used a servo suitably placed near the boiler connected with the Intel Edison board. The servo can mechanically push down the on button of the boiler (Figure 6.12, green arrow) turning on it. The servo and its support (I reused a common plastic container) were fixed with Blu Tack. A simple smartphone app (Figure 6.13) allows the control of the smart boiler estimating also the number of cups that can be filled according to the current water level. Users can activate the smart boiler remotely pushing a button on the app. The smart boiler cannot be turned on if the water is under a certain level. In that case, the app displays a notification indicating that water is not enough. The state of the boiler, which can be "heating" (when the mechanical button is pushed down) or "off" (when the mechanical button returns up), is also sensed by Protobject using the blue region in Figure 6.12. In this way, the smartphone can give feedback to the user when the water is boiling (i.e., when the state of the boiler is "heating" – Figure 6.13-centre) and when it is ready (i.e., when the state of the boiler pass from "heating" to "off" – Figure 6.13-right).

Discussion

Since this prototype uses the Intel Edison board, the use of Meemoo does not make sense. Therefore, I show the JavaScript version only. The entire prototyping process of this smart boiler takes around 65 minutes of which 30 were used to set the environment 20 to develop the smartphone app (i.e., the HTML5 app) and 15 to develop the software to manage the Intel Edison board. Moreover, the entire application has 90 lines of code, i.e., 74 of HTML + JavaScript + CSS (see Code 6.5) and 16 of Node.js (see Code 6.6) for managing the Intel Edison board.

Code 6.5: The smart boiler smartphone app designed with the JavaScript framework

```
<!DOCTYPE html>
<html>
  <head>
    <title>Protobject Smart Boiler App</title>
    <script src="framework/protobject.js"></script>
    <script src="jquery.js"></script> <!-- This prototype uses jQuery -->
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
    #imgCup {
      left: 12%;
      position: absolute;
      width: 80%;
    }
    #numberCup {
      font-family: sans-serif;
      font-size: 35px;
      position: absolute;
      text-align: center;
      top: 95px;
      width: 79%;
      z-index: 40;
    }
    #button {
      background: lightblue none repeat scroll 0 0;
      font-family: sans-serif;
      font-size: 28px;
      left: -8%;
      padding: 5%;
      position: absolute;
      text-align: center;
      top: 290px;
      width: 98%;
      z-index: 99;
    }
    </style>
  </head>
  <body>
    <img id="imgCup" src="cup.jpg" /></div>
    <div id="numberCup" >3</div>
    <div id="button">&raquo; Turn on</div>
  </body>
  <script>
    protobject.connect("ws://localhost:8085/");
    protobject.onEvent("Level", function(e) {
      if (parseInt(protobject.getState("Level")) < 2) {
        $( "#numberCup" ).html("Too low");
        $( "#button" ).css("color","#777");
      } else {
        $( "#numberCup" ).html(protobject.getState("Level"));
        $( "#button" ).css("color","#000");
      }
    });
    protobject.onEvent("Button.Boiling", function(e) {
      $( "#button" ).html("Boiling...");
      $( "#button" ).css("color","#777");
    });
    protobject.onEvent("Button.Off", function(e) {
      $( "#button" ).html("&raquo; Turn on");
      $( "#button" ).css("color","#000");
      alert("The water is ready!");
    });
    $("#button").click(function() {
      if (parseInt(protobject.getState("Level")) < 2) {
        alert("There is no enough water in the boiler!");
      } else {
        if (button == "Boiling") {
          alert("Water is heating...");
        } else {
          protobject.externalSend("TurnOn");
        }
      }
    });
  </script>
</html>
```

Code 6.6: The smart boiler node.js Intel Edison code

```

var Cylon = require('cylon'); <!-- This prototype uses Cylon (Cylonjs.com) -->
var WebSocket = require('ws'); <!-- Library for WebSocket connection -->
var ws = new WebSocket('ws://192.168.1.84:8085'); //Connect to Protobject WebSocket
```

```
Cylon
  .robot()
  .connection('edison', { adaptor: 'intel-iot' })
  .device('servo', { driver: 'servo', pin: 3, connection: 'edison' })
  .on('ready', function(my) {
    ws.on('message', function(data) {
      if (data == "TurnOn"){
        my.servo.angle(60);
        setTimeout(function(){  my.servo.angle(140); }, 1000);
      }
    });
  })
  .start();
```

As before, in this usage scenario, I used Protobject to reinvent and reuse an existing object. Moreover, the Intel Edison board was leveraged for extending Protobject. For achieving the same aims by using traditional prototyping approaches (i.e., Arduino and 3D printers), a weight sensor should be installed under the boiler in order to estimate the water level. This would required the design of a suitable boiler support able to host the weight sensor, whose fabrication would be time-consuming.

The WebSocket server of Protobject manages the communication among Protobject, the Intel Edison board and the smartphone App. The Intel Edison board (as well as Arduino) easily supports WebSocket connections over Wi-Fi.

The Protobject approach has a little drawback in this usage scenario. In fact, while using the boiler, it is frequently removed from the base and the latter could move a little bit during the usage. These movements could cause errors while Protobject detects the water level. Therefore, to avoid unexpected movements of the base, it should be suitably fixed to the hosting table (I used Blu Tack, but more stable alternatives may be used).

6.2.6 Usage scenario 6: energy awareness tool

Description

Raising awareness of energy consumption has become crucial over time [SCG$^+$11] and Protobject can be exploited for these purposes since it can be used to check the state of lights, white goods, and any electronic device whose behavioural properties are, or can be made, visible. Evaluating the states of some devices, it is possible to estimate their energy consumption.

This usage scenario shows an energy awareness tool able to estimate the consumption of some lights and an oven observing their states. Regarding the oven, Protobject can estimate the consumption taking into account how it is currently working (e.g. Defrost, Grill, etc.). Objects are observed exploiting two smartphones (a Motorola Moto G and a Sony Experia E4). The camera of the first smartphone looks at five lights (Figure

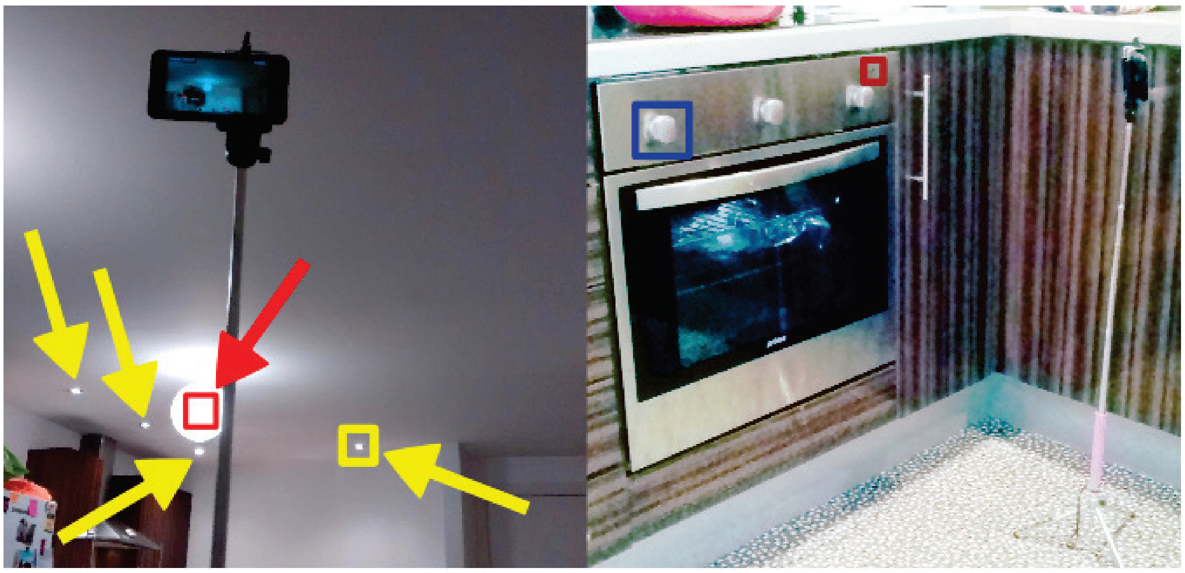


Figure 6.14: Energy awareness scenario.

6.14-left) whereas the camera of the second smartphone looks at the oven (Figure 6.14-right). Two switches control the five lights: the first switch controls the four lights indicated by the yellow arrows in Figure 6.14-left whereas the second switch controls the biggest light indicated with the red arrow in the same figure. Since the switches are two, I exploit just two regions (yellow and red in Figure 6.14) to let Protobject sense which lights are turned on (I assume that if one of the four lights indicated by the yellow arrows is turned on, also the remaining three lights are turned on). Regarding the oven, it is sensed exploiting two further regions (see Figure 6.14-right): the blue one senses the function selector knob (i.e. to detect which function is set) whereas the red one senses the resistance indicator light (i.e., to detect if the oven is working).

The detailed instant consumption of the lights and the oven is displayed through a desktop web app. Moreover, the total consumption is displayed through an LCD display (Figure 6.15) managed by the Intel Edison board. As before, Protobject exchanges messages with the Intel Edison board through its web socket server.

Discussion

Since this prototype uses the Intel Edison board, the use of Meemoo does not make sense. Therefore, I show the JavaScript version only. The entire prototyping process of this smart boiler takes around 50 minutes of which 15 were used to set the environment 20 to develop the smartphone app (i.e., the HTML5 app) and 15 to develop the software to manage the Intel Edison board. Moreover, the entire application has 100 lines of code, i.e., 77 of HTML + JavaScript + CSS (see Code 6.7) and 23 of Node.js (see Code

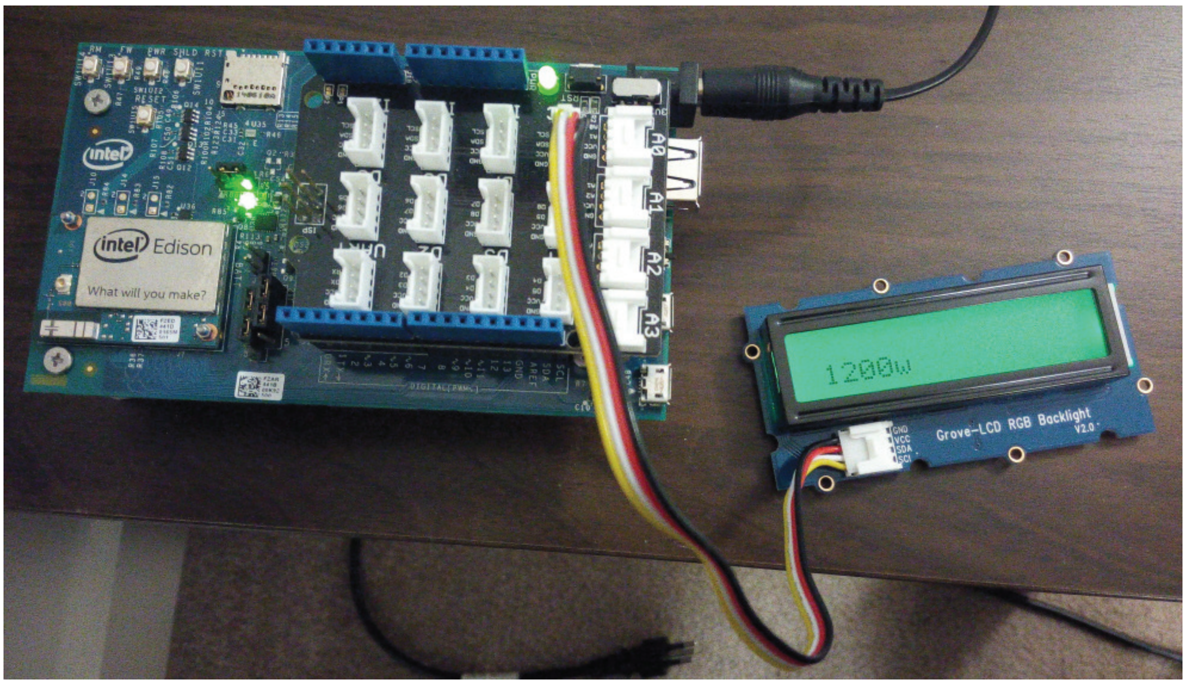


Figure 6.15: The LCD display shows the instant consumption.

6.8) for managing the Intel Edison board.

Code 6.7: The energy awareness app designed with the JavaScript framework

```
<!DOCTYPE html>
<html>
  <head>
    <title>Protobject Energy Awareness</title>
    <script src="../../js/protobject.js"></script>
    <script src="../../js/jquery.js"></script>
    <style>
    html, body {
      font-size:170%
    }
    </style>
  </head>
  <body>
    <table style="width:100%">
      <tr>
        <td><h3>Object</h3></td>
        <td><h3>Status</h3></td>
        <td><h3>Consumption</h3></td>
      </tr>
      <tr>
        <td>Light 1</td>
        <td id="l1"></td>
        <td id="l1c"></td>
      </tr>
      <tr>
        <td>Light 2</td>
        <td id="l2"></td>
        <td id="l2c"></td>
      </tr>
      <tr>
        <td>Oven</td>
        <td id="o1"></td>
        <td id="o1c"></td>
      </tr>
    </table>
    <h3 id="tot">Total: </h3>
  </body>
  <script>
    protobject.connect("ws://localhost:8085/");  //Protobject initialization
    protobject.loop(function() {
      if ((protobject.getState("OvenFunction")) == "Off") {
        oven=0;
      } else if (protobject.getState("OvenFunction") == "Defrost"){
        oven=920;
      } else if (protobject.getState("OvenFunction") == "Light"){
        oven=20;
      } else if (protobject.getState("OvenFunction") == "Grill" &&
    protobject.getState("Heating") == "Yes"){
        oven=1000;
      } else if (protobject.getState("OvenFunction") == "Fan" &&
    protobject.getState("Heating") == "Yes"){
        oven=1800;
      } else if (protobject.getState("OvenFunction") == "Grill" &&
    protobject.getState("Heating") == "No"){
        oven=20;
      } else if (protobject.getState("OvenFunction") == "Fan" &&
    protobject.getState("Heating") == "No"){
        oven=20;
      }
      $("#o1").html(protobject.getState("OvenFunction"));
      $("#o1c").html(oven+"w");
      if ((protobject.getState("Light1")) == "On") {
        light1=70;
      } else {
        light1=0;
      }
      $("#l1").html(protobject.getState("Light1"));
      $("#l1c").html(light1+"w");
      if ((protobject.getState("Light2")) == "On") {
        light2=70;
      } else {
        light2=0;
      }
      $("#l2").html(protobject.getState("Light2"));
      $("#l2c").html(light2+"w");
      totalConsumption=parseInt(light1+light2+oven);
      $("#tot").html("Total consumption: " + totalConsumption + "w");
      protobject.externalSend("Consumption: " + totalConsumption + "w");
    });
  </script>
</html>
```

Code 6.8: The energy awareness app node.js Intel Edison code

```
var Cylon = require('cylon');
var WebSocket = require('ws');
var ws = new WebSocket('ws://192.168.0.101:8085');
function writeToScreen(screen, message) {
  screen.setCursor(0,0);
  screen.write(message);
}
Cylon
  .robot({ name: 'LCD'})
  .connection('edison', { adaptor: 'intel-iot' })
  .device('screen', { driver: 'upm-jhd1313m1', connection: 'edison' })
  .on('ready', function(my) {
    my.screen.setColor(0, 10, 0);
    writeToScreen(my.screen, "Ready! Waiting for data...");
    ws.on('message', function(data, flags) {
      if (data.indexOf("Consumption:") > -1){
        console.log(data);
        writeToScreen(my.screen, "                              "); //clear the screen
        writeToScreen(my.screen, data);
      }
    });
  })
  .start();
```

Note that although the installation of any sensor was not required, I used the Intel Edison board to control the LCD display. Moreover, this usage scenario has two analogies with the usage scenario of the energy saving tool. As in that study, I ran two instances of Protobject in order to use two cameras simultaneously. Moreover, as in that scenario, the prototype was made of two old and cheap electronic devices without any modification and avoiding the use of specific (and expensive) hardware like Samsung Smart Home products.

Finally, a limitation of Protobject in this usage scenario is that it can just estimate the instant consumption according to the state of the objects. Therefore, the consumption of electronic devices has to be known a priori in each of their functions. In this regard, detailed hardware datasheets to know consumptions, which are often difficult to find, could be essential.

6.3 The emerged approach

After discussing the limitations of the approach emerged thanks to Protobject, I present them together with its advantages discussing the way in which Protobject may support and change the design process.

6.3.1 Protobject limitations

As shown in the above usage scenarios, Protobject can be effective in many situations. Nevertheless, as any rapid prototyping tool, Protobject has limitations that need to be taken into consideration.

During prototyping, users have to consider that Protobject is not actually designed to detect interactions, but just their effects (the knob selector discussed in the usage scenario 4 – Figure 6.11 – is a clear example of this limitation). In other words, Protobject exploits changes of state to detect interactions. For example, Protobject cannot sense interactions with touch buttons since visible results of this interaction cannot be observed.

Moreover, the position of the camera(s) is crucial. As discussed before in section 3.1.2, avoiding interferences with people who are interacting is important. In fact, Protobject has to be able to observe any property adequately. Therefore, cameras have to be positioned cleverly in order to avoid interferences with people but also allowing a clear view of the objects adequately (according to my experience, in many cases, modifying the position of the camera avoided recognition errors). Moreover, cameras must be stable and fixed (e.g. using a tripod) since any eventual unexpected movement would cause detection errors. Finally, tripods could disturb during normal testing activities. Although in the early stages of prototyping some annoyances are negligible, there could be cases in which tripods may impede users' interactions. In that case, an alternative camera positioning may be explored (e.g. installing cameras on the ceiling).

Regarding the class of objects that can be sensed, Protobject can be used only with objects that have a stationary nature. So, Protobject cannot be used to sense “mobile” objects such as smartphones, laptops or cordless phones.

6.3.2 The novelty of the approach

Protobject can accelerate the design process especially in idea generation and prototyping.

Regarding idea generation, Protobject allows designers to reinvent the use of existing objects remixing them for different scopes by tinkering (meant as the act of doing bricolage) with them. Seeing objects from a different perspective, looking at existing things to discover their potential for new functionalities and remixing them are common activities in idea generation. All the use cases presented in this section were designed according to this philosophy: reuse, remix and reinvent.

Regarding prototyping, Protobject allows designers to embed [Dou04b] the “object of design”[BDME+11] in the real world making it work in a real context. This is aimed to have real experiences with the object of design encouraging designers to “reflect in practice” [Sch83] on it. All the use cases were refined and improved reflecting in practice thanks to Protobject.

Of course, idea generation and prototyping are highly related from the Protobject point of view. On the one hand, “reflection in practice” with prototypes ‘embedded in the real world’ allows designers to generate new ideas. Inspiration, in fact, is an experience derived through practice [BDME+11]. On the other hand, generating new ideas and alternatives “embedded in the real world” through real prototypes allows designers to “reflect in practice” on their objects of design.

Besides focusing on the aforementioned aspects of the design process, I now discuss and put forward the novelty of Protobject in comparison with other similar tools. As discussed in the literature review, Zensors and Eyepatch are somehow quite related to Protobject. Nevertheless, Zensors is not applicable in rapid prototyping (due to the crowd-powered training, which makes no sense when time and money are limited) whereas Eyepatch does not allow the segmentation of the video stream in different regions. Moreover, both Zensors and Eyepatch do not allow makers and designers to associate a specific state of an object with a certain return value.

Associating a certain state of an object directly with a return value is the most remarkable feature of Protobject. The selection of states is the step through which the designer decides the way in which he wants to enrich things with interaction. Somehow, with this step, designers transform the space of their designs into a state machine. The states recognized in specific objects, in fact, allow the definition of the global states taken into account in the interaction. This feature may make a significant difference concerning the way of supporting the design process. In fact, this lets designers make prototypes through performances [Dan05], which are highly related to the bricoleurs' ways of doing [CS15]. A performance involves a transformation of existing entities, which, in our case, can be directly correlated with the change of state of existing objects. In other words, it can be said that a change of state is the consequence of a performance. In this regard, Protobject lets designers give a name to the result of a performance (i.e. the name of the resulting state). For example, performing the turning on of a lamp, the consequence is that the lamp is on, and this result can be easily learned by Protobject and identified through a name ("On").

Finally, the strength of Protobject is that it is conceptually simple: users do something (performance) and assign a name (i.e. the resulting state) to the effect of that performance. As displayed by the preceding usage scenarios, this approach resulted to be bricolage-oriented – all the usage cases show design through bricolage –, flexible and fruitful for prototyping IoT and Smart Home systems.

Chapter 7

System evaluation

7.1 Overview

I evaluated many aspects of Protobject and in particular:

1. section 7.2 discusses the capability of Protobject to detect different kinds of events and interactions;
2. section 7.3 discusses the robustness of the recognition of object states;
3. section 7.4 discusses the robustness of the recognition of generic presences when changing light conditions;
4. section 7.5 discusses the precision of recognition when detecting continuous movements, also evaluating two different colour-wheel-based markers.

7.2 Detectable events

Protobject can be used to sense different events and interactions replacing many sensors in a simple and effective way. I tested Protobject to detect the (1) presence/absence of objects in a fixed position (e.g. an umbrella in the umbrella stand, or towels in a towel rack); (2) opening angle of doors, or of similar rotating objects; (3) level of a coloured liquid inside a transparent container; (4) position of a curtain; (5) interactions with mechanical buttons or switches; (6) changes of state of LCD displays, LEDs or lights; (7) interaction with potentiometers; (8) amount of the toilet paper in the bathroom; (9) turning on or off of electronic devices; (10) flap orientation of the air conditioner; (11) running water running from a tap (also estimating its amount); (12) presence or absence of a person (e.g. on a sofa, in front of a door); (13) colour of an area or of a light. Note that these events were the ones I tested; the list is not exhaustive and many other events could be added.

There are also states of objects that Protobject cannot detect. For example, Protobject struggles with the detection of the (1) state of a glass door (since it is transparent); (2) state of a window, if it is covered, e.g., by a curtain.

The robustness of the detection concerning the majority of the objects listed in this section is evaluated in the next section.

7.3 Robustness evaluation of state recognition

When prototyping, environmental light conditions are normally stable and Protobject rarely makes recognition errors. At any rate, there could be situations in which light conditions change, also when prototyping. This is why I evaluated Protobject recognition simulating different lighting variances to measure its robustness. In particular, this section discusses the evaluations when recognising states considering light changes.

7.3.1 Procedure

In order to have real prototyping conditions, I performed these evaluations in a domestic environment. We tested the recognition of the states of several objects (Figure 7.1) and in particular, I detected:

1. the amount of water from a tap;
2. the position of a curtain;
3. the interaction with a mechanical button;
4. the state of a LCD display;
5. the function knob position of an oven;
6. the colour of a surface;
7. the knob position using a colour-wheel-based marker;
8. the level of a coloured liquid;
9. the opening angle of a door;
10. the presence and colour of towels in a towel rack.

Tests were carried out using a THL 5000 smartphone as Wi-Fi camera.

For each target object in Figure 7.1, I recorded the database of states at the best possible light conditions in the environment (as they are supposed to be during normal prototyping activities). Then, a dataset with 600 images for each state of each object

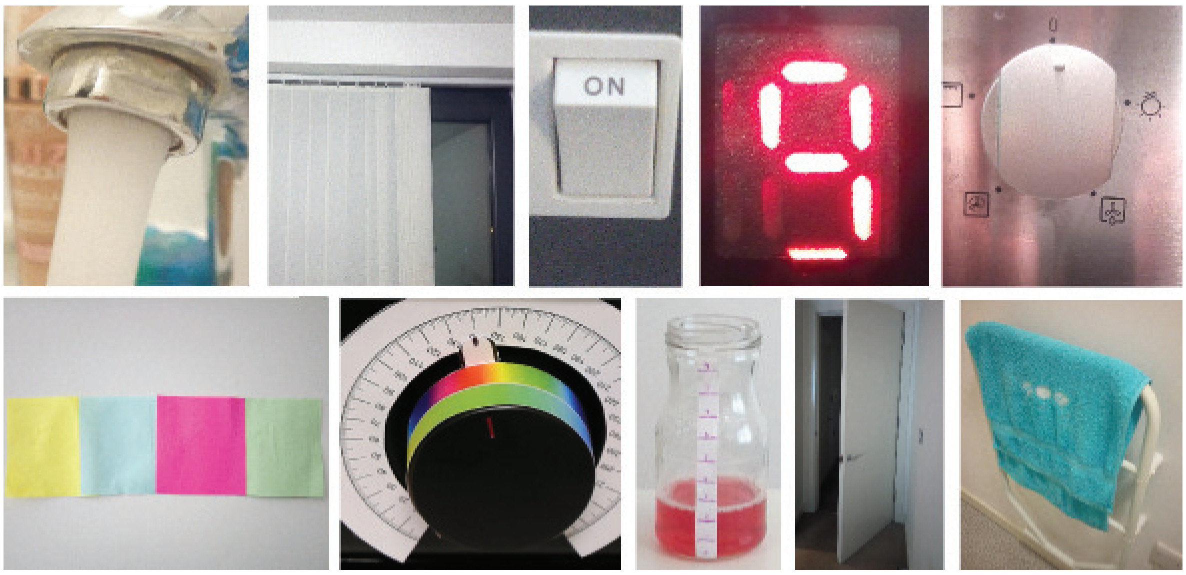


Figure 7.1: Evaluated objects.

was recorded while generating shadows moving randomly an opaque A3 cardboard (Figure 7.2). Shadows were generated to simulate different lighting conditions. After collecting the dataset for each object simulating shadows, robustness tests were executed on the each dataset.

In particular, for each state of each object, I performed a robustness test over 600 detections getting (1) the state of the object that would be actually recognized by Protobject to check for errors and (2) the general luminance value of the image (mean of pixel RGB intensities) to keep trace of the light variance that caused eventual errors.

7.3.2 Details and results

Detailed results of each state for each object are displayed in Appendix A.

Case 1: amount of water from a tap

Five different amounts of water (states) were recorded in the database: No-Water, Amount 1, 2 and 3, and Max-Amount. Regarding errors, the state "Amount 1" was detected with 1 error. In particular, "Amount 2" was detected instead of "Amount 1" in 1 out of 600 detections. Considering the state "Amount 1", the water looked like unstable due to its low pressure and this probably caused the error. Regarding the other states, at any rate, no error occurred.

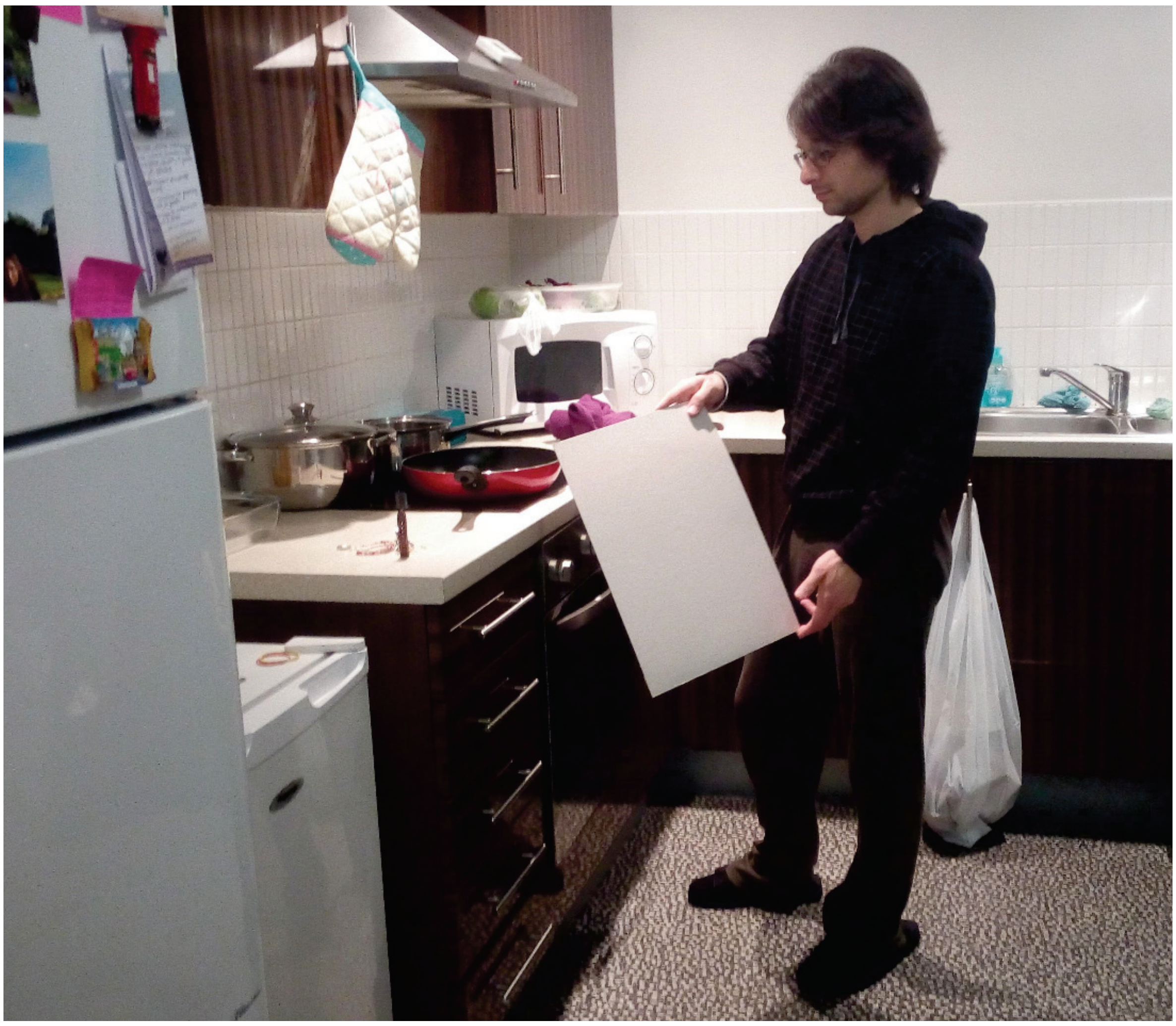


Figure 7.2: The cardboard used for generating shadows on the objects.

Appendix A.1 summarizes these results for each detected state through a time series of 600 detection highlighting errors.

Case 2: position of a curtain

11 different curtain positions were recorded in the database, from open to closed. Since the region to detect the state of the curtain was defined in its upper part, shadows on the curtain cannot be generated without covering the region observed by Protobject. Therefore, this evaluation was carried out turning on and off three different lights in the room. No error occurred for any of the recorded positions.

Appendix A.2 summarizes these results for each detected state through a time series of 600 detection highlighting errors.

Case 3: interaction with a mechanical button

On and off states were recorded in the database. No error occurred for any of the recorded positions.

Appendix A.3 summarizes these results for each detected state through a time series of 600 detection highlighting errors.

Case 4: state of an active LCD display

Ten states were recorded in the database, i.e., digits from 0 to 9. No error occurred for any of the recorded positions.

Appendix A.4 summarizes these results for each detected state through a time series of 600 detection highlighting errors.

Case 5: function knob position of an oven

Five states were recorded, i.e., Off, Defrost, Light, Grill and Fan. No error occurred for any of the recorded positions.

Appendix A.5 summarizes these results for each detected state through a time series of 600 detection highlighting errors.

Case 6: colour of a surface

Four states were recorded, i.e., brown, green, pink and yellow. No error occurred for any of the recorded positions.

Appendix A.6 summarizes these results for each detected state through a time series of 600 detection highlighting errors.

Case 7: knob position (using a colour-wheel-based marker)

16 states were recorded, i.e., 16 different positions. No error occurred for any of the recorded positions.

Appendix A.7 summarizes these results for each detected state through a time series of 600 detection highlighting errors.

Case 8: level of a coloured liquid in a transparent container

10 states were recorded, i.e., 10 liquid levels. No error occurred for any of the recorded positions.

Appendix A.8 summarizes these results for each detected state through a time series of 600 detection highlighting errors.

Case 9: opening angle of a door

Ten states were recorded, i.e., ten opening angles from 0 degrees to 90 degrees.

Since the region to detect the state of the door was defined in its upper part, shadows on the door cannot be generated without covering the region observed by Protobject. Therefore, this evaluation was carried out turning on and off three different lights in the room.

No error occurred for any of the recorded positions.

Appendix A.9 summarizes these results for each detected state through a time series of 600 detection highlighting errors.

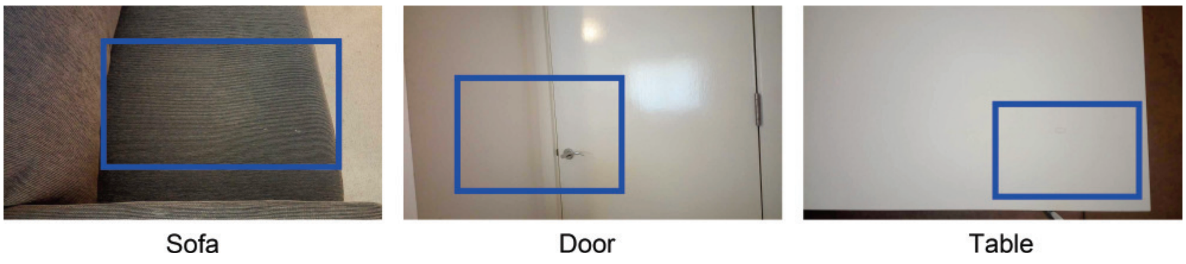


Figure 7.3: Presence recognition cases

Case 10: presence and colour of towels in the towel rack

Four states were recorded, i.e., three different colour towels and the absence of towels. No error occurred for any of the recorded positions.

Appendix A.10 summarizes these results for each detected state through a time series of 600 detection highlighting errors.

7.4 Robustness evaluation of presence recognition

This section discusses the evaluations when detecting presences/absence of someone/-something on a certain region considering light changes. As before, I performed these evaluations in a domestic environment.

I tested the detection of three kinds of presences (Figure 7.3) and in particular:

1. the presence/absence of a person on a sofa;
2. the presence/absence of a person in front of a door;
3. the presence/absence of a smartphone on a region of a table.

7.4.1 Procedure

In this evaluation, I tested different light conditions moving randomly an opaque brown cardboard (A3 size) between light sources in the environment and the target area in which the presence has to be detected (e.g. blue regions in Figure 7.3).

For each kind of detection in Figure 7.3, I recorded the state of absence at the best possible light conditions in the environment (as it is supposed to be done during normal prototyping activities). Therefore, I discovered empirically threshold values for detecting of the presence of someone/something as discussed in section 3.6. After discovering and setting the right threshold values for each case, I carried out the evaluations testing different ways of dressing (cases 12 and 13) and each kind of smartphone (case 14).

In particular, the evaluation consisted in 600 detections for any presence (i.e., way of dressing and model of smartphone) and the absence. The 600 detections were tested for errors simulating shadows with the cardboard. For evaluating presences, unlike the previous evaluation on state recognition, I performed a live evaluation without recording any dataset.

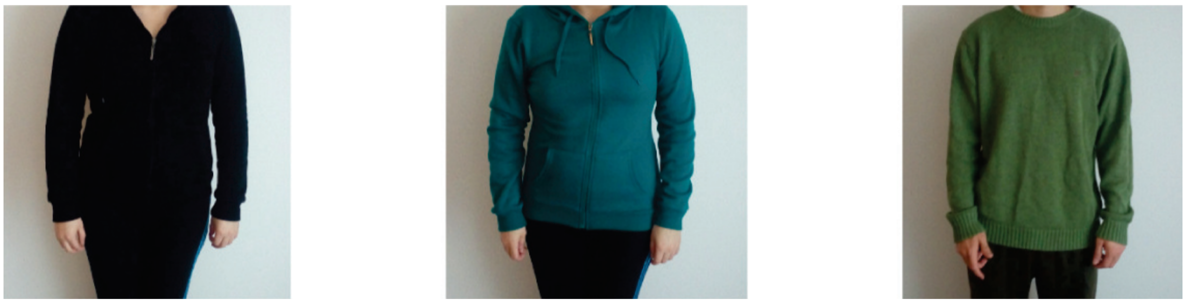


Figure 7.4: Three ways of dressing.

7.4.2 Details and results

Case 11: Presence and absence of a person on a sofa

The state of the empty sofa was recorded (Figure 7.5-Left). Therefore, I tested the presence of a person dressed in three different ways (see Figure 7.4 and 7.5) after discovering the optimal threshold value, which is 954 (see Figure 7.5).

According to that threshold, no errors occurred while simulating different shadow conditions for any of the four conditions (3 different presences – i.e. 3 ways of dressing – and absence).

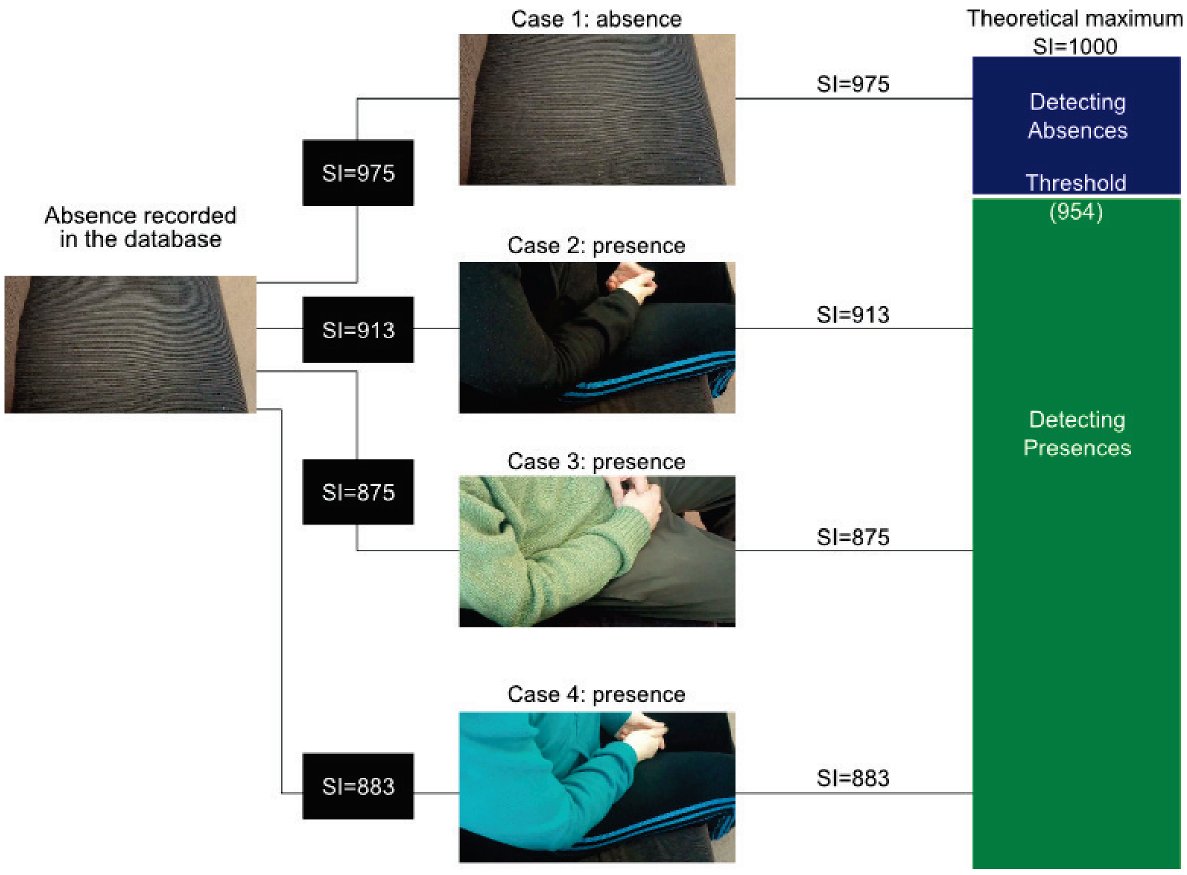


Figure 7.5: Leveraging threshold for presence detection on a sofa.

Case 12: Presence and absence of a person in front of a door

The state of the closed door was recorded (Figure 7.6-Left). Therefore, as before, I tested the presence of a person dressed in three different ways (see Figures 7.4 and 7.6) after discovering the optimal threshold value, which is 915 (see Figure 7.6). According to that threshold, no errors occurred while simulating different shadow conditions for any of the four conditions (3 different presences – i.e. 3 ways of dressing – and absence).

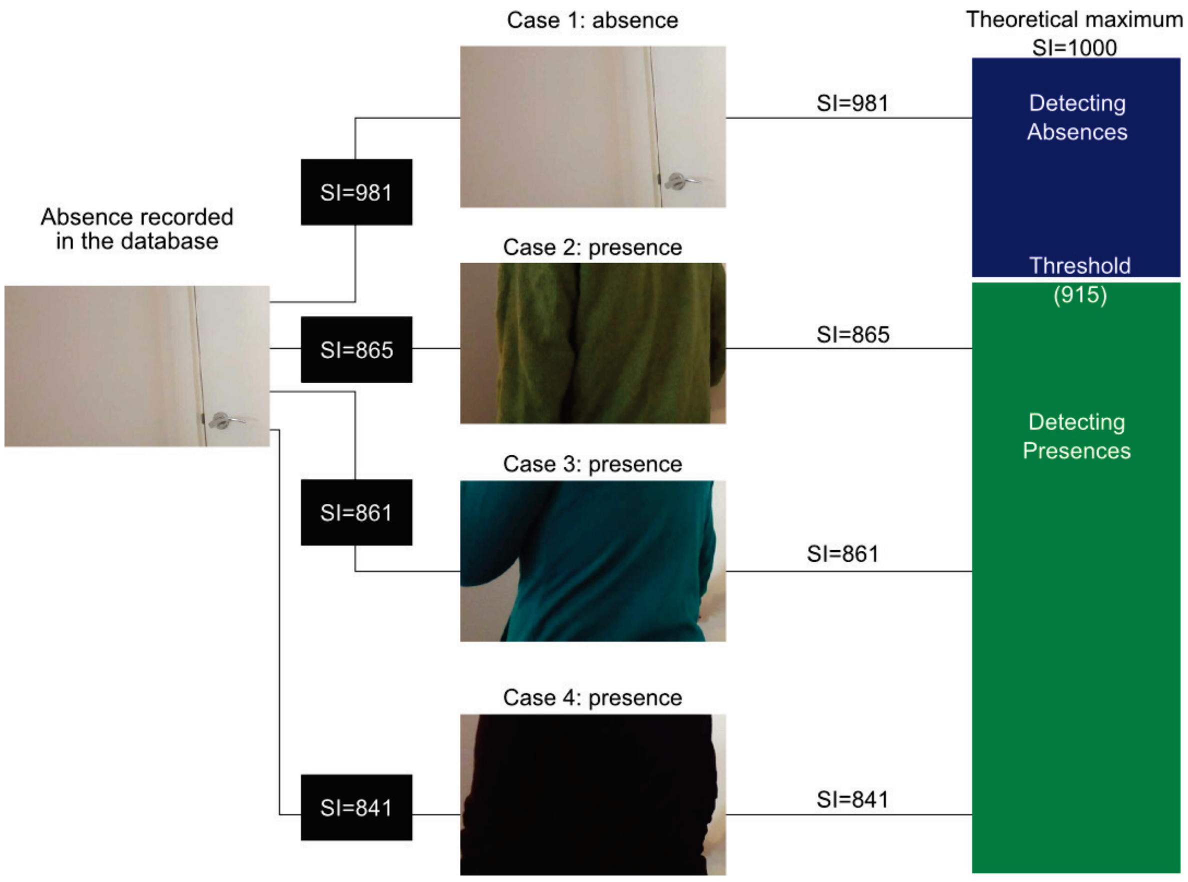


Figure 7.6: Leveraging threshold values for presence detection in front of a door.

Case 13: Presence and absence of a smartphone on a specific region of a table

The state of the empty table was recorded (Figure 7.7-Left). Therefore, I tested the presence of three smartphones (see Figures 7.7 and 7.8) after discovering the optimal threshold value, which is 718 (see Figure 7.8).

According to that threshold, no errors occurred while simulating different shadow conditions for any of the four conditions (absence and presence of three different smartphones).

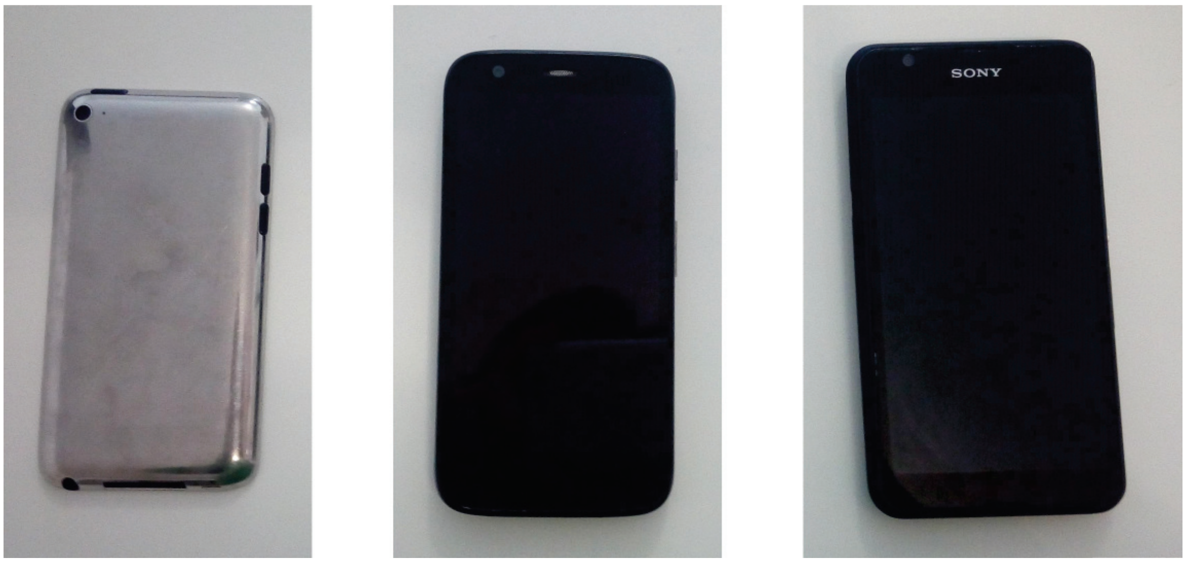


Figure 7.7: Three different tested smartphones. An Apple iPod (back side), a Motorola Moto G (front side) and a Sony Experia G4 (front side). We purposely put forward the back side of the iPod since the front side looks black i.e., in a very similar way to the other two smartphones making the case not so significant.

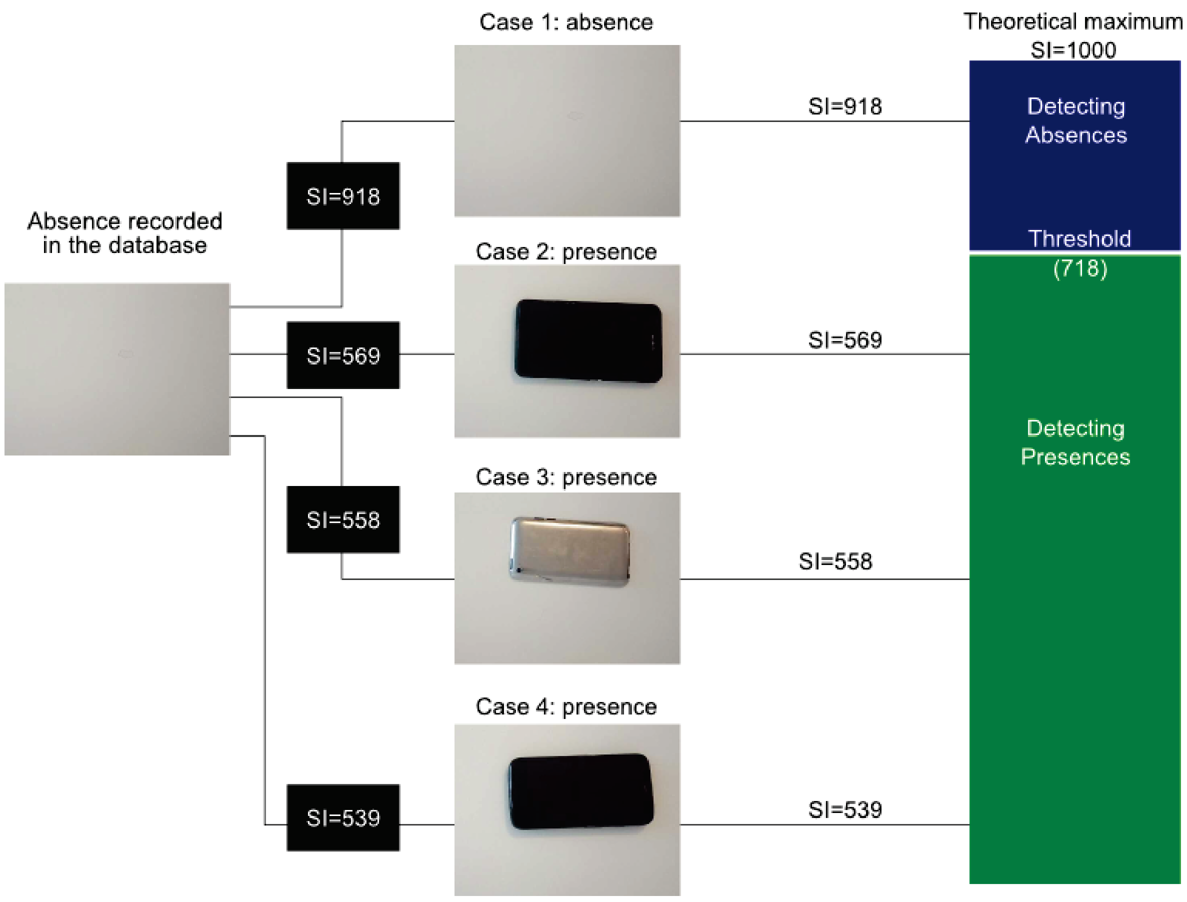


Figure 7.8: Leveraging threshold values for presence recognition of smartphones on a table.

7.5 Precision evaluation when recognizing continuous movements

I evaluated the precision of Protobject when detecting discretized continuous movements in two different cases. In the first case, I evaluated Protobject precision when recognizing the movement of a continuous knob using two different colour-wheel based markers. In the second case, I evaluated Protobject precision when detecting the opening angle of a door.

The aim of this experiment was just to evaluate the precision of Protobject when detecting continuous movements. Therefore, changes of lighting condition were not taken into account (recognitions were tested at the normal lighting condition of learning without generating intentionally any shadow).

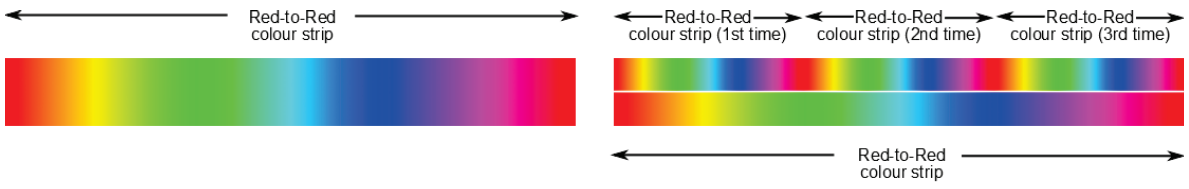

Red-to-Red colour strip
Red-to-Red colour strip (1st time)
Red-to-Red colour strip (2nd time)
Red-to-Red colour strip (3rd time)
Red-to-Red colour strip

Figure 7.9: Different kinds of coloured strips can be used to allow rotation recognition.

7.5.1 Continuous knob position

I evaluated the precision of Protobject when detecting changes of continuous variations using colour-wheel-based markers. In particular, I used those markers to evaluate Protobject precision when detecting the movement of a continuous knob. This test aims at discovering how many different knob positions can be distinguished by Protobject stably. I consider a recognition as "stable" when it is performed correctly for 200 consecutive detections.

I evaluated two different coloured markers. The first marker (Figure 7.9-left) is a normal linearized colour wheel, which goes from red-to-red. The second marker (Figure 7.9-right) is composed of two parallel strips. The first strip is a red-to-red linearized colour wheel (Figure 7.9-right bottom strip). The other strip is composed of three red-to-red linearized colour wheels, which are repeated three times and stretched so that the total length is equal to the length of the first strip (Figure 7.9-right upper strip).

The markers displayed in Figure 7.9 allow Protobject to recognize knob positions with different precisions.

In order to discover the precision given by the two kinds of coloured markers, I performed many trials. In particular, the recognition was tested recording databases of states at different precisions (every 40 degrees, 35 degrees, 30 degrees and so on) finding the best precision that each marker could achieve, i.e., until Protobject made stable recognitions.

Procedure

I used a volume knob (47mm of diameter) of a Hi-Fi amplifier. The knob rotation ranges from 0 degrees (minimum volume) to 300 degrees (maximum volume).

To measure different positions easily, a goniometer was printed and placed around the knob as depicted in Figure 7.10-right. The goniometer covers all the volume range from 0 degrees (the minimum volume) to 300 degrees (the maximum volume). Protobject was connected to a THL-5000 smartphone used as Wi-Fi camera, which was placed at the right of the knob supported by a selfie stick. The region used to detect the knob

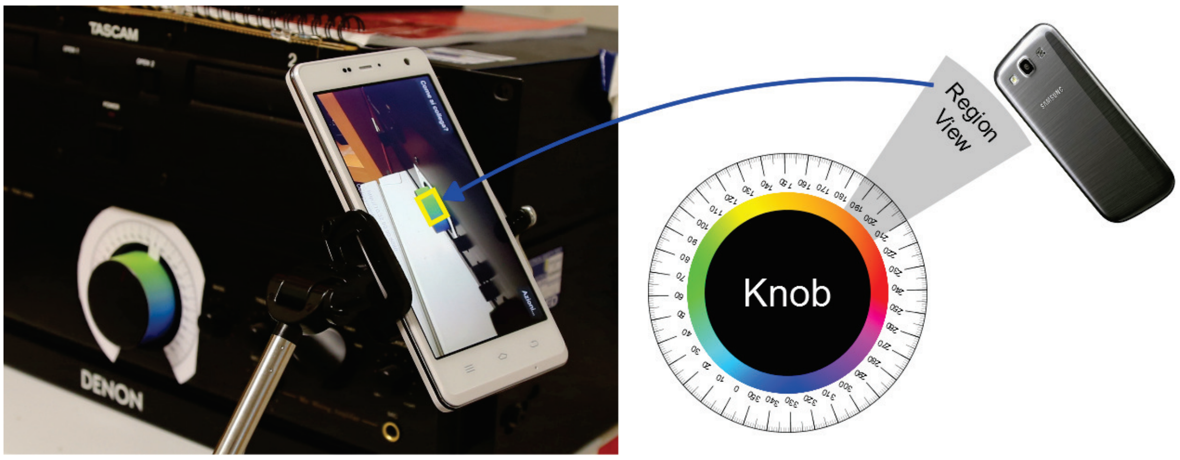


Figure 7.10: Experiment settings

(the yellow one in Figure 7.10-left) covers an area of 25 degrees (i.e., from 185 degrees to 210 degrees in the goniometer, as depicted in Figure 7.10-right).

Results

The best precision that can be achieved by the strip in Figure 7.9-left is equal to *25 degrees*. It means that Protobject was able to distinguish 13 states (i.e., movements every 25 degrees) in the range of 300 degrees stably (200 consecutive correct detections for each state).

The best precision that can be achieved by the strip in Figure 7.9-right, instead, is equal to *5 degrees*. It means that Protobject was able to distinguish 61 states (i.e., movements every 5 degrees) in the range of 300 degree stably (200 consecutive correct detections for each state). This result was beyond my expectations, even if I were aware of the fact that, using the strip in Figure 7.9-right, Protobject had more information to discriminate the different positions of the knob in comparison with the strip in Figure 7.9-left. Figure 7.11 summarizes the experimental results discussed in this section.

7.5.2 Opening angle of a door

I evaluated Protobject when detecting the opening angle of a door. In particular, this test aims at discovering how many different position of a door can be recognized by Protobject stably. A recognition is considered "stable" when it is performed correctly for 200 consecutive detections.

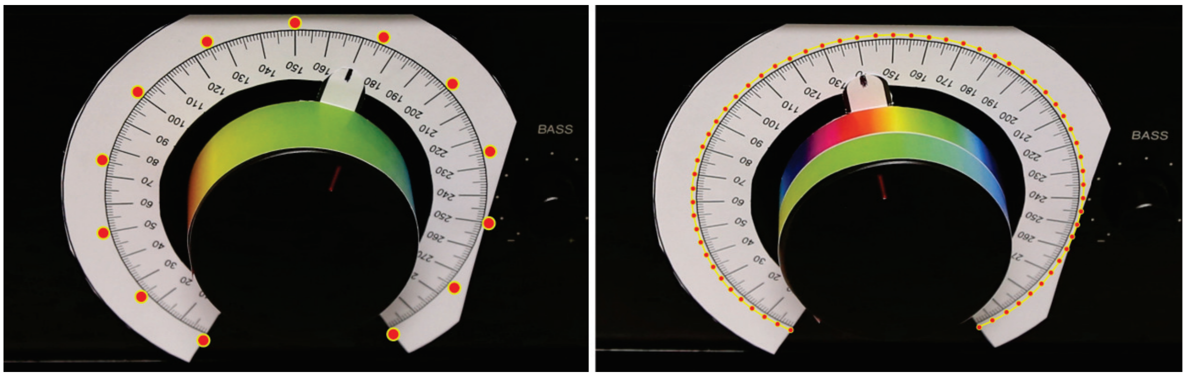


Figure 7.11: Summary of the precision results when detecting knob positions using two different coloured strips. Using the strip at the left, Protobject can distinguish 13 different positions whereas, using the strip at the right, Protobject can distinguish 61 different positions.

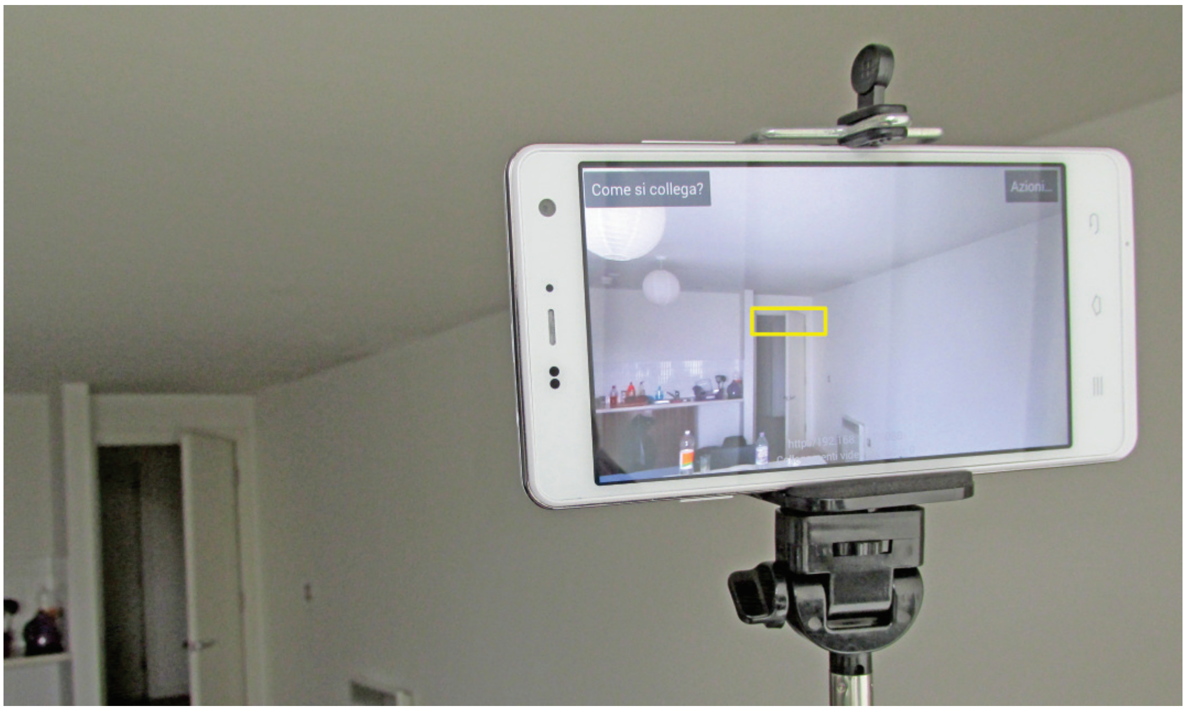


Figure 7.12: The yellow region was used for detecting the opening angle of a door.

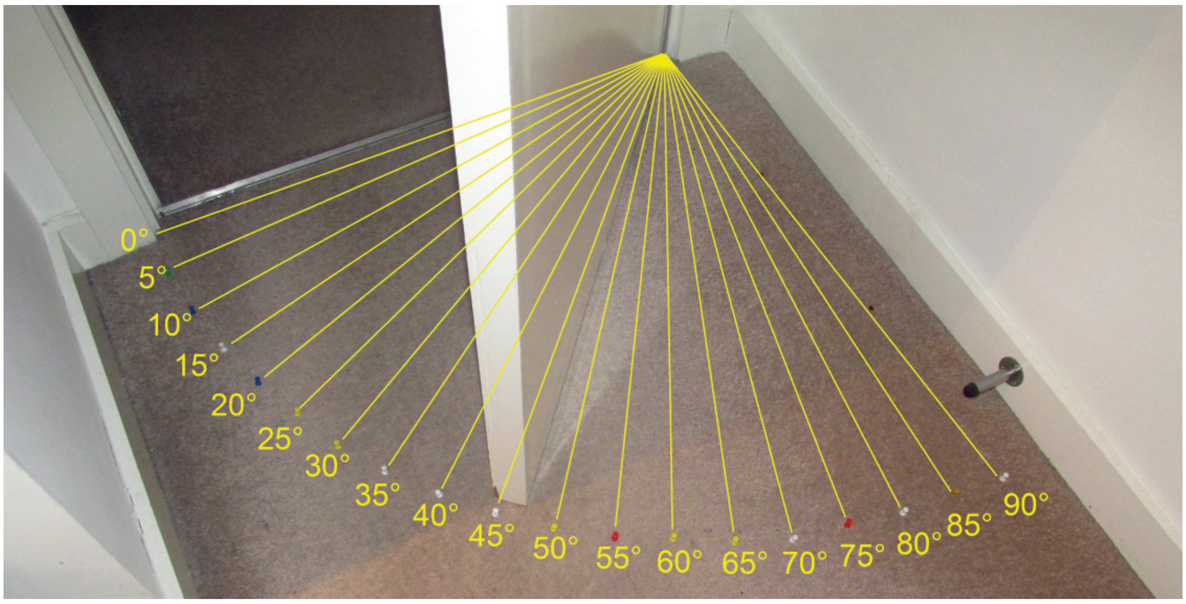


Figure 7.13: Precision achieved by Protobject (every 5 degrees) when detecting the opening angle of a door. Note the drawing pins inserted in the carpet to mark the different opening angles.

Procedure

I used the door in Figure 7.12, whose position can range from 0 degrees (closed) to 90 degrees (open). Protobject was connected to a THL-5000 smartphone used as Wi-Fi camera, which was placed at around five meters from the door supported by a selfie stick. The region used to detect the knob is the yellow one in Figure 7.12. In order to discover the recognition precision, I performed many trials recording databases of states at different precisions (every 7 degrees, 6 degrees and so on) until Protobject made stable recognitions. I used drawing pins inserted in the carpet around the door to mark the different opening angles tested during the experiment.

Results

The best precision that can be achieved when recognizing the opening angle of the door in Figure 7.12 is *5 degrees*. It means that Protobject was able to distinguish 19 different states (i.e., movements every 5 degrees) in the range of 90 degrees stably (600 consecutive correct detections for each state). Figure 7.13 summarizes the experimental results discussed in this section.

7.6 Summary of the results

7.6.1 States and presence recognition

Table 7.1 shows all the case studies with the algorithm used for the detection and the number of errors. I consider errors strictly. This means that I consider the detection of a state as completely failed, even if just one error (over 600 detection) occurred for that state.

Cases of state detection	**Recorded states**	**Errors**	**Successful rate**
1 (water from tap)	5	1 out of 5	80%
2 (curtain position)	11	0 out of 11	100%
3 (mechanical button)	2	0 out of 2	100%
4 (LCD display)	10	0 out of 10	100%
5 (hoven knob)	5	0 out of 5	100%
6 (surface colour)	4	0 out of 4	100%
7 (knob, double strip)	16	0 out of 16	100%
8 (door opening angle)	10	0 out of 10	100%
9 (coloured liquid level)	10	0 out of 10	100%
10 (towel rack)	4	0 out of 4	100%
Cases of presence detection	**Kinds of presences**		
11 (person / sofa)	3 dressing ways	0 out of 4	100%
12 (person / door)	3 dressing ways	0 out of 4	100%
13 (smartphone / table)	3 smartphones	0 out of 4	100%

Table 7.1: Summary of the evaluations regarding lighting variances. Total successful rate (percentage average) 96.5%. Regarding presences, shadows were simulated on four situations i.e., three presences (dressing or smartphones) and the absence.

7.6.2 Recognizing continuous movements

Regarding the detection of knob positions, different colour markers offer different precision. The simple colour marker offers a precision of 25 degrees, whereas the double-side colour marker offers a precision of 5 degrees. The same precision of 5 degrees was achieved also regarding the recognition of the opening angle of a door.

7.6.3 Discussion

These evaluations were carried out trying to cover the multiplicity of the objects and presences that can be sensed by Protobject. Actually, objects can have very different observable properties (depending on materials, reflections properties, etc.) and so, it could be hasty to generalize these evaluations to the variety of objects around us and to their possible states.

At any rate, according to these evaluations (96.5% average successful rate), I can state that Protobject is able to ensure an adequate reliability for early rapid prototyping of interactive products, also in the case of lighting variances like the ones I tested.

Regarding the detection of states (cases 1-10), an error occurred just on the detection of the amount of water from a tube, probably due to water instability.

Regarding the detection of presences (cases 11-13), no error occurred in any case. At any rate, designers, makers and researchers have to discover empirically the best threshold values that fit adequately the possible variances of environmental lighting conditions.

Moreover, I remind that Protobject detects any kind of presence that produces an excess of the threshold and there is no possibility of distinction among different objects. For instance, a cup of tea would be detected as a smartphone and vice versa. Therefore, the detection of presences should be used carefully and cleverly.

Protobject was also quite precise when recognizing discretized continuous movements. According to the evaluation displayed in this section, I can state that Protobject can also be used in most of the cases in which the detection of continuous movements is required.

Chapter 8

User evaluation

8.1 Overview

In order to evaluate Protobject potentialities and easiness of use, I designed a workshop on Protobject, which was carried out in two repeated sessions. The first session, attended by 14 people, took place on 1st of October. The second session, attended by 8 people, took place on 2nd of October. Therefore, 22 people (12 women and 10 men) tried and tested Protobject during the two sessions. The workshop sessions were organized in Milan during the Brera Design Days (Figure 8.1), an event attended mainly by designers.

8.2 Hosting environment

Workshop sessions were carried out in a room of the “Mediateca Santa Teresa”, placed in Milan on 25, Moscova street. The room was arranged with four tables and the materials required for designing different prototypes (Figure 8.2).

8.3 Participant recruitment

Participants were recruited by the organizers of Brera Design Days through their networks (i.e., Facebook, Twitter and the website of the event). Workshop was also advertised on Fablabs, co-working spaces and several local Facebook groups focused on design and technology.

DESIGN DAYS

PROGETTO E INNOVAZIONE TRA DIGITALE E TERRITORIO

I WORKSHOP DI MEDIATECA

PROTOBJECT

1 - 2 OTTOBRE

—

workshop

Protobject è un sistema che consente la prototipazione rapida di sistemi interattivi in modo semplice ed economico senza l'uso di sensori o schede elettroniche. Il suo punto di forza è la semplicità: la disponibilità di un PC e uno smartphone - e un supporto per lo stesso, come ad esempio un treppiedi - è l'unico prerequisito.

Protobject è in grado di trasformare la telecamera di uno smartphone in un sensore multiuso in grado di rilevare l'ambiente circostante e le interazioni delle persone con lo stesso. Protobject può essere utilizzato per rilevare numerosi eventi come ad esempio l'apertura di una porta, la presenza di un ombrello in un portaombrelli, il sollevamento della cornetta da un telefono fisso o la modalità di funzionamento di un forno. Questi eventi possono quindi essere sfruttati per creare sistemi intelligenti: ad esempio, sollevando la cornetta rispondendo a una chiamata, Skype passa automaticamente da disponibile ad occupato. Al contrario, quando si conclude la chiamata e si ripone la cornetta, Skype torna allo stato disponibile automaticamente.

Nella prima parte del workshop saranno illustrati numerosi esempi pratici d'uso di Protobject. Nella seconda parte del workshop, i partecipanti saranno guidati nella realizzazione di alcuni di semplici sistemi intelligenti.

Il workshop Protobject è in tre edizioni a numero chiuso (16 partecipanti ad edizione).
Per ciascuna edizione del workshop, verranno ammesse 16 persone sulla base dell'ordine di registrazione.

1 ottobre
14. 30 - 17.30 - Ia edizione workshop Protobject

2 ottobre
9.00 - 12.30 - IIa edizione workshop Protobject
14. 30 - 17.30 - IIIa edizione workshop Protobject

A cura di:
Alessio Bellino e Giorgio De Michelis

Brera DesignDays
TALKS & WORKSHOPS

1 / 2 OTTOBRE 2016
MEDIATECA SANTA TERESA MILANO

www.breradesigndays.it

Figure 8.1: Brera Design Days workshop.

Figure 8.2: Room of the workshop.

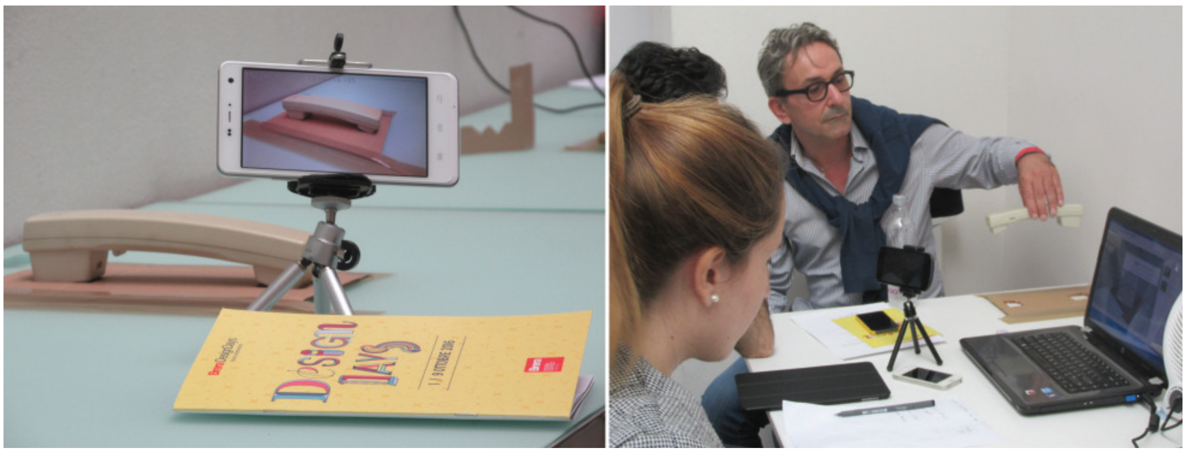


Figure 8.3: Home phone connected with Skype.

8.4 Work teams

The 14 people of the first workshop session were split into 4 work teams. Two of them had 4 people each whereas the remaining two had 3 people each. Teams were composed mainly of designers, but in each one of them there was a person who declared to have at least soft programming competences.

The 8 people of the second workshop session were split into 3 work teams. Two of them had 3 people each whereas the third group had 2 people. Also in this session, teams were composed mainly by designers. In the groups composed of 3 people, one of them declared to have at least soft programming competences. In the last group composed of 2 people, none of them had programming competences.

I expected the person who declared to have at least soft programming competences to be able to guide the other members of the group who did not have such competences. Finally, I personally guided the members of the last group of the second session since there was nobody who declared to have at least soft programming competences.

8.5 External people

Some people gave their help during the workshop.

In the first session of the workshop, Simeon Petrov (programmer, graduated at the University of Milan-Bicocca in Computer Science), Micol Riva (designer, graduated at Politecnico of Milan in Interior Design), Daniela Bascuñán (photographer) and Flavio De Paoli (one of my thesis supervisors) were present.

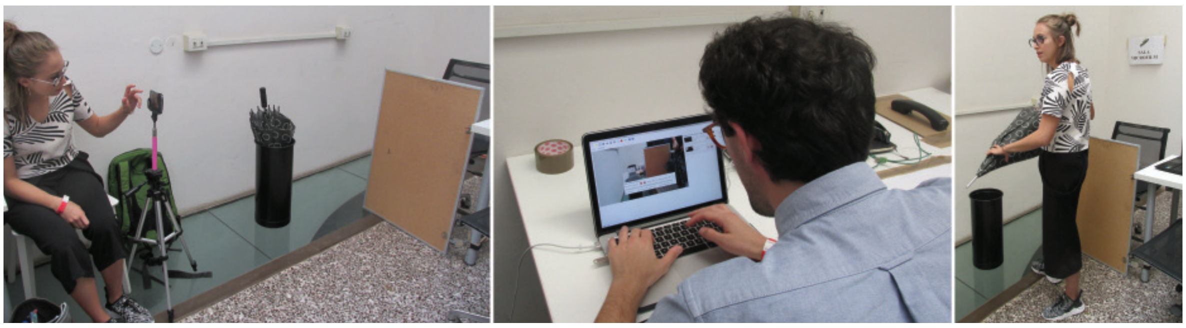


Figure 8.4: Smart umbrella scenario: the umbrella stand and the fake door.

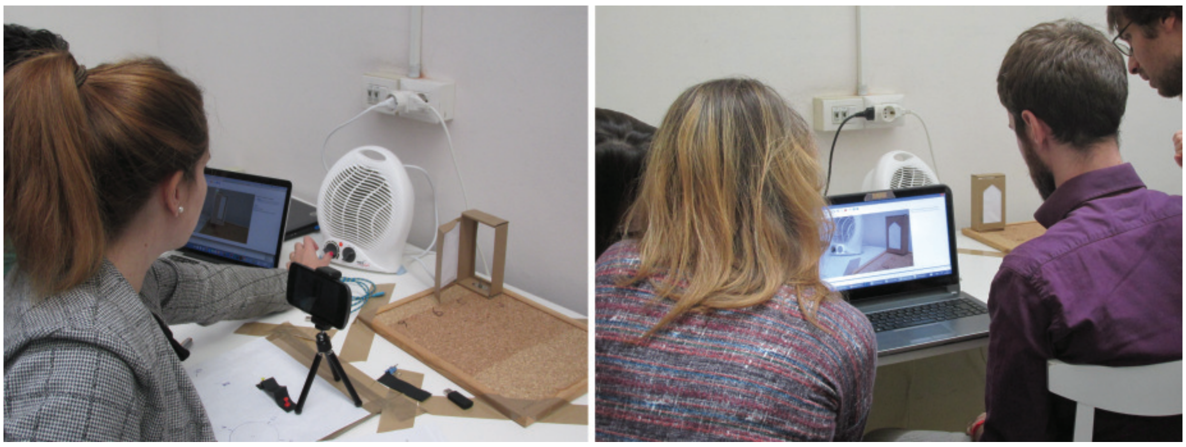


Figure 8.5: Energy saving tool: the heater and the fake window of cardboard.

In the second session of the workshop only Micol Riva and Daniela Bascuñán attended to help.

In particular, Simeon Petrov and Flavio De Paoli helped explaining programming logic when needed. None of them was previously trained to use Protobject.

Micol Riva collected some feedback during the workshop.

8.6 Materials

During the workshop, participants were guided to design several prototypes, which were: (1) a smart phone interacting with Skype (answering a call, Skype goes to busy automatically and vice versa, finishing the call, Skype returns active), an energy saving tool (displayed in section 6.2.2), a smart umbrella stand (displayed in section 6.2.1) and a tangible interface for controlling a light (displayed in section 6.2.3). For building these prototypes, I pre-arranged different materials and components. One phone receiver was

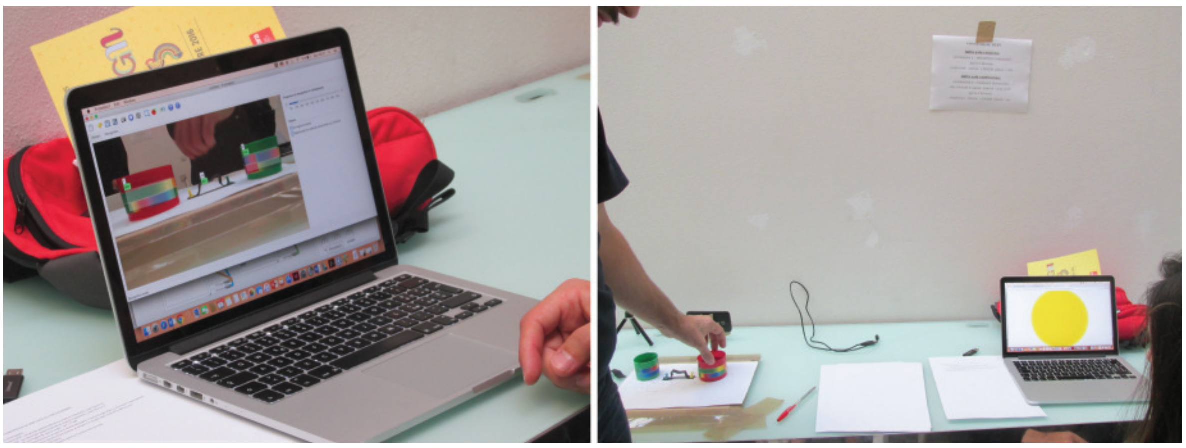


Figure 8.6: Tangible interface for controlling a light: two caps used as potentiometer and a strip of paper used as button.

placed in any of the four tables for designing the smart phone connected with skype (Figure 8.3). A panel simulating a door was installed (near a table) and was used together with the umbrella and the related umbrella stand for designing the smart umbrella stand prototype (Figure 8.4). A heater was used together with a simulated window (Figure 8.5), which was used for designing the energy saving tool. Corkboards with caps (used for simulating potentiometers) and a paper strip (simulating a button) were used for designing the tangible interface for controlling a light (Figure 8.6).

Each participant was asked to bring his/her computer. We provided smartphones and tripods for holding them up but participants could use their smartphones if they wanted.

8.7 Procedure

At the beginning of the workshop, I gave a short introductory talk explaining Protobject aims and operating ways. Then, I explained the work to be done during the workshop. The work consisted in partially guided design of several prototypes. In particular, the design of the first prototype (phone connected with Skype) was entirely guided: it was explained how to capture the states of the receiver (picked and hanged-up) and how to design the interactive behaviour, initially using Meemoo (the visual programming language) and next using JavaScript code using the browser. After the first prototype, each group worked on different prototypes in sequence. Each group completed at least three prototypes (including the Skype one).

The duration of the workshop sessions was around 3 hours. Some participants went out before that time whereas others stayed at the workshop more than three hours. We did not invite any participant to leave the workshop after the three expected hours of the workshop.

After the workshop, participants were invited - via mail - to fill a questionnaire.

8.8 The questionnaire

The anonymous questionnaire (see appendix B.1) had three sections:

1. personal information
   - age (numeric field)
   - academic background (free text area)
   - current job (free text area)
   - familiarity with technology (6-point scale from 1 to 6)
   - coding abilities (multiple selection)
   - smartphone renew frequency (multiple selection)
   - smartphone ownership (yes/no)
2. system evaluation using 6-point scale from 1 (lowest quality) to 6 (highest quality)
   - use of states for prototyping
   - easiness of use of the Protobject UI
   - easiness of use of Meemoo for defining interactive behaviour
   - easiness of use of Javascript + browser for defining interactive behaviour
3. two free text areas where participants could write
   - what they learned from the workshop experience
   - how Protobject could be improved

8.9 Experimental results

This section presents the results of the questionnaire, together with the observations and feedback collected during the workshop.

8.9.1 Questionnaire

18 out of 22 participants filled the questionnaire and the following sections display the results.

Participant profiles

Participants ages were quite different (M: 33, SD:10.2). The oldest participant was 57 years old whereas the youngest was 21 years old.

With respect to the academic background (see Figure 8.7), 9 out of 22 participants studied design (from architecture to product and industrial design), 4 studied computer science (or one of its sub-disciplines), the rest had different backgrounds (e.g., sociology and electro-technology or design and engineering).

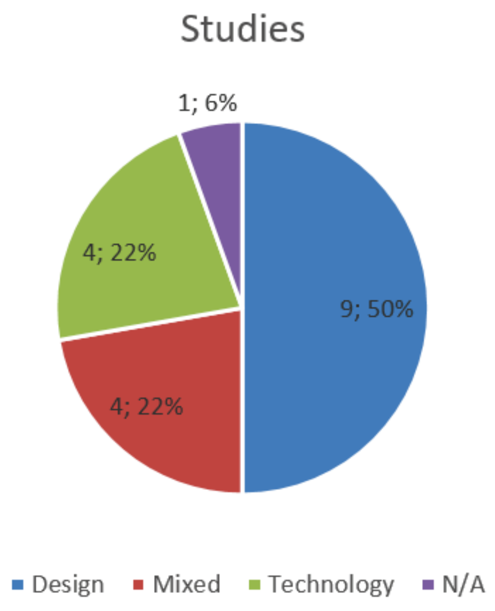


Figure 8.7: Academic background of workshop participants.

Regarding the job (see Figure 8.8), 7 out of 18 work as designers, 4 work in the technological field, two of them are still students, the rest work in other fields.

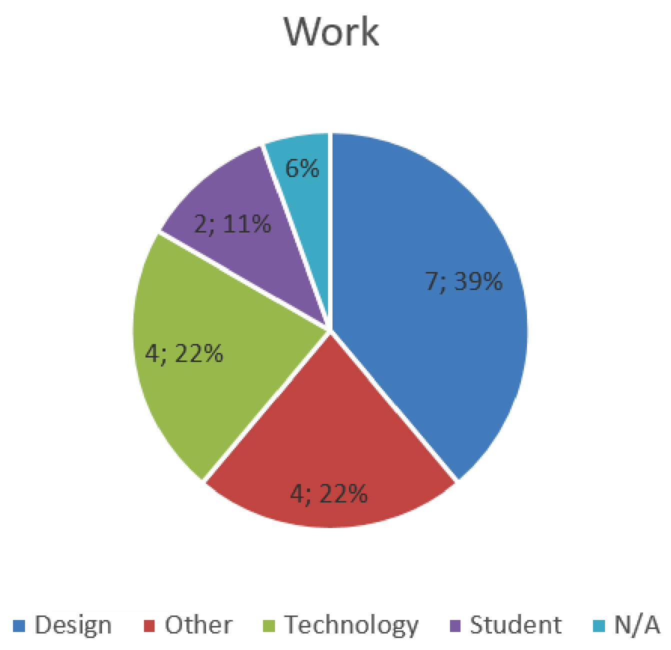


Figure 8.8: Jobs of workshop participants.

Regarding the proneness to renew smartphones (see Figure 8.10), 6 out of 18 declared to change their smartphones only when it is broken, 1 declared to renew his/her smartphone once a year, 4 renew their smartphones every two years and the rest of the participants every three years.

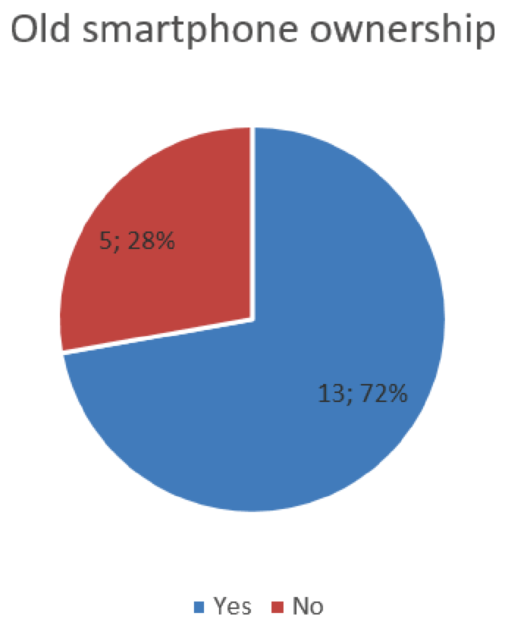


Figure 8.9: Old smartphone ownership of workshop participants.

Moreover, 13 out of 18 participants declared to have old smartphones at home whereas the rest did not (see Figure 8.9). These figures indicate that most of participants could use Protobject at home exploiting old smartphones (continuing to use their personal smartphone for the normal activities).

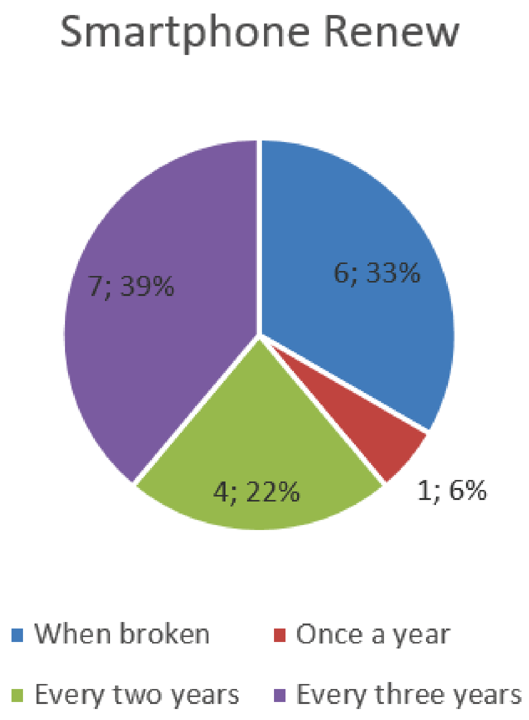


Figure 8.10: Smartphone renew period of workshop participants.

All participants self-evaluated their relationship with technology at least 3 out of 6 (the highest score, indicating "the ability to understand all technological gadgets"). Figure 8.11 shows the response distribution of this item with mean in red and mode in blue.

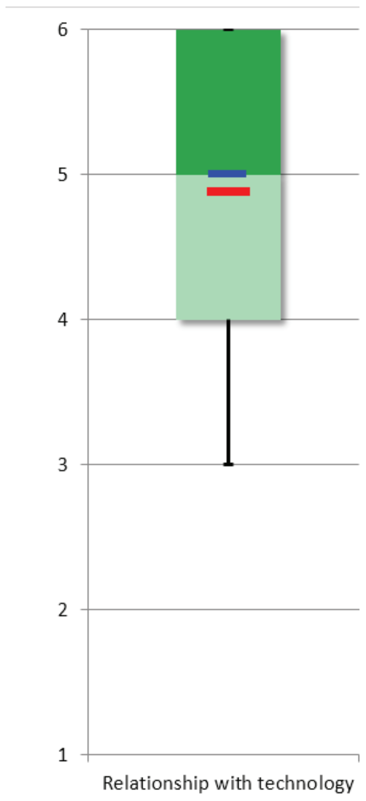


Figure 8.11: Response distribution of the relationship with technology of workshop participants with mean in red and mode in blue.

Regarding their programming abilities (see Figure 8.12), 3 out of 18 participants have no idea of programming, 2 are able to understand logic and processes without being able to code, 9 are able to develop simple script and programs, 2 are able to develop average-complexity software, 1 is able to develop high complexity software. Finally, 1 participant did not answer the question.

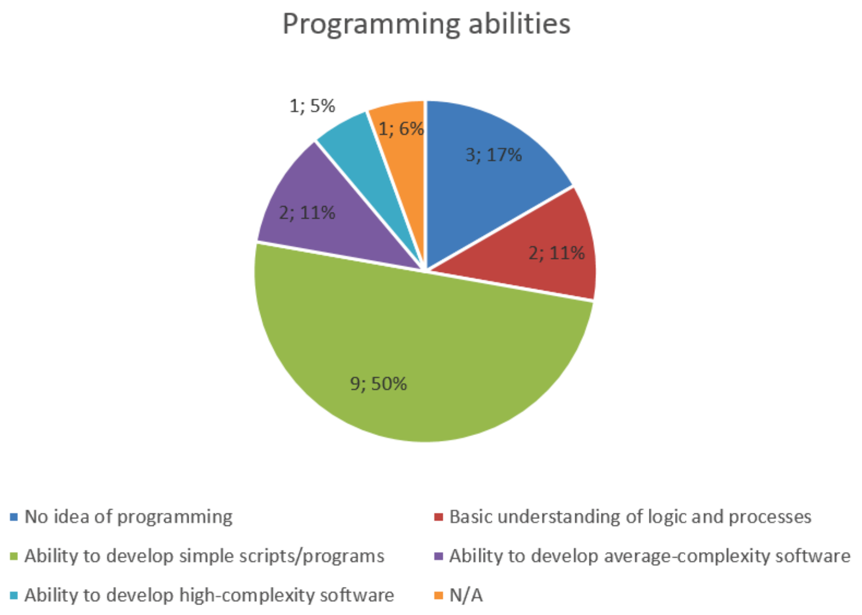


Figure 8.12: Programming abilities of workshop participants.

Quantitative analysis

Binomial test. A binomial test was carried out to detect positive or negative tendencies on the responses of the questionnaire. The 6-point ordinal scale was chosen because of its balanced condition between negative values (1, 2 and 3) and positive values (4, 5 and 6). Significant positive tendencies (p<.05) were detected for 3 out of 4 items taken into account: use of states for prototyping (proportion 0.00 vs. 1.00, p<.001), easiness of use of Protobject UI (.22 vs. .78, p<.031) and easiness of use of Meemoo (.22 vs. .78, p<.031). Regarding the use of Javascript + browser for defining interactive behaviours, no significance tendency was found (.39 vs. .61, p<.481). The results are summarized in Figure 8.13 through Boxplots with mean in red and mode in blue.

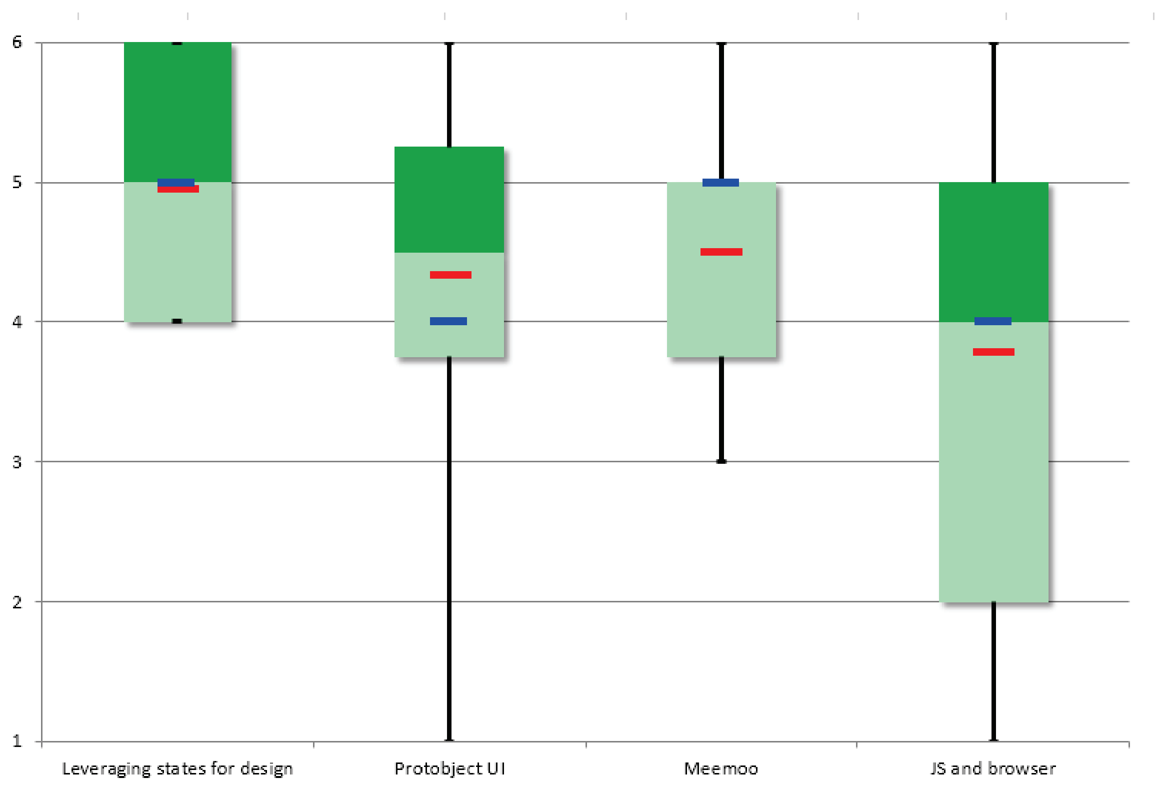


Figure 8.13: Response distrubution of workshop participants with respect to different evaluation items with mean in red and mode in blue.

Comparison between groups. Further analysis on the questionnaire items were carried out also with respect to 'different programming abilities', 'relationships with technology', 'academic backgrounds' and 'different jobs' performing a Kruskal-Wallis test.

No difference of responses were found for the easiness of use of Protobject. Therefore, I could say that the easiness of use of Protobject does not seem to depend on users' skills or background.

No difference of responses were found for the use of states for prototyping. Therefore, I could say that users may be generally able to understand how to use states regardless of their skills or background.

No difference of responses were found for the use of Meemoo. Therefore, I could state that users may be generally able to understand Meemoo way of use regardless of their skills or background.

Differences of responses were found only for the use of JavaScript + browser according to different 'relationships with technology' (χ^2=10.325, df=3, p=.016), 'academic backgrounds' (χ^2=8.476, df=2, p=.014) and jobs (χ^2=10.705, df=3, p=.013).

In particular, participants with a better relationship with technology evaluated the use of Javascript + browser as simpler than other participants (mean rank: 5.50 for 3 value, 4.25 for 4 value, 14.33 for 5 value and 9.50 for 6 value).

Moreover, also participants with better technical background evaluated the use of Javascript + browser as simpler than other participants (mean rank: 6.83 for design background, 7.63 for mixed background and 15.26 for technical background).

Finally, also participants with more technical jobs evaluated the use of Javascript + browser as simpler than other participants (mean rank: 5.36 for designers, 8.00 for others, 11.25 for students and 15.25 for technical jobs).

Discussion. It is noteworthy that all participant evaluated Protobject UI and the use of states for prototyping as simple to use. Also Meemoo was evaluated as simple by most of the participants regardless of their different academic backgrounds, jobs or relationships with technology. Differences between groups regarding the use of Javascript, instead, was expected since traditional designers could be worried by traditional programming languages.

Qualitative analysis

In order to analyse the open section of the questionnaire, I identified the most salient ideas and concepts that emerged from the participants' answers. I focused on the most relevant and enlightening feedback that allowed me to understand the participants' perceptions and impressions of Protobject.

First complete question. How would you describe the experience of today? What did you learn? Was it useful or useless? Do you think that Protobject could change the way to design interactive systems? Please, try to give as many details as possible.

Second complete question. What would you improve about Protobject? The interface? The way of use? If you have any ideas, try to explain them here trying to give as many details as possible.

User answers. In this section, the significant answers of the participants are displayed. The complete answers can be read in Appendix B.2.

Participant 1, architect
First answer: "It was very useful. I will try to use it for lighting projects."
Second answer: not relevant.

Participant 2, programmer and web designer
First answer: "I think that with the right application [...] it would be possible to develop something that can really change users' habits [...]".
Second answer: "Meemoo editor, sometimes, is not really simple and there is no affordance[1] in relevant steps [of the programming process]. Some additional help for not skilled users would be nice".

Participant 3, programmer and web developer
First answer: "I learnt that it is possible to make ideas really work leveraging everything [...] reinventing any [object] using old smartphones".
Second answer: not answered.

Participant 4, designer
First answer: "It was an interesting experience and I am still figuring out situations in which I can make experiments using Protobject. Surely, I believe that I would be not so scared if I needed to make prototypes of this type".
Second answer: "I would improve the interface because with a good user experience [Protobject] could be still simpler and intuitive. Probably the initial presentation was not able to make clear the value/meaning of Protobject".

Participant 5, architect and teacher
First answer: "[Thanks to this workshop] I have a clearer idea regarding programming possibilities".
Second answer: "I would make [Protobject] clearer".

Participant 6, student of design and engineering
First answer: "The concept and idea are charming and useful. It is challenging for innovation [and] I am thinking already how and where [I] can apply it".
Second answer: not relevant.

Participant 7, student and industrial designer
First answer: "I had an interesting experience. It could be interesting to use [Protobject] in the meta-design[2] process to make a concept understandable or for a first prototyping idea of an interactive system".
Second answer: "The [Protobject] UI could be improved to make it more understandable for different users with different backgrounds".

[1]Affordance is defined as *"The qualities or properties of an object that define its possible uses or make clear how it can or should be used". Merriam-Webster Dictionary.*

[2]Meta-design is a preliminary process aimed to define the initial stages of a design project.

Participant 8, software developer
First answer: “It is hard to answer because I have never designed an interactive system. The experience was interesting, I liked the idea. The interfacing with JavaScript is good letting programmers focus only on designing without wasting time on programming processes. For me, it is hard to think of scenarios where Protobject can be used for final systems and not for design aims [...]”.
Second answer: “The [Protobject] UI could be improved but a user guide could be anyhow enough”.

Participant 9, manager
First answer: “[The workshop] was interesting and [the use of Protobject] intuitive”.
Second answer: not relevant.

Participant 10, product/UX designer
First answer: “[The workshop was] really interesting, but the initial presentation was not so clear. For the next times, I would advice to bring more specific examples of use inside the design process of an interactive system. For example, if I wanted to make a sensor for monitoring a baby, first I would try the interaction with Protobject and next, I would make the system with Arduino. Protobject is not a substitute for any other technology but it helps in the research process and this should be made clearer”.
Second answer: “[Protobject] icons were not so clear and there were many windows”.

Participant 11, product designer
First answer: “The workshop was really interesting. I think that I could use Protobject for my interior design projects”.
Second answer: “[Protobject] UI is not so intuitive for people who are not familiar with programming principles”.

Participant 12, project manager fablab
First answer: “[Protobject] is interesting and easy to use, and therefore suitable also for unskilled users. We will try to use it for interaction design projects and test them with final users”.
Second answer: “[It would be good] adding confirmation messages when states are recorded and labeled. The programming tool [that is Meemoo] should be supported by tool-tips for helping users who are not programmers”.

Participant 13, web developer
First answer: n/a
Second answer: n/a

Participant 14, designer
First answer: “Surely it would be possible to use Protobject for prototyping a basic interactive system. Maybe, understanding better [its] functionalities and possibilities,

it will become possible to design more complex systems [... not relevant]".
Second answer: "For people who do not (or slightly) know programming, the systems could appear not straightforward [...]. Moreover, a new training process is needed if the camera is moved (voluntarily or involuntary) and this makes the training inconvenient [... not relevant]".

Participant 15, architect, product designer
First answer: "[The experience] was really useful and [Protobject] is interesting".
Second answer: "Protobject [smartphone app] is not available for iPhone".

Participant 16[3]*, designer consultant*
First answer: "The experience was really interesting for people who want to understand and make real their ideas without knowing technology. I understood the potentialities of Protobject and that designers' way to do could change thanks to it. In the future, I believe that Protobject approach can bring a real innovation creating a bridge between designers and programmers, which could be a really important paradigm change when designing. I would like to follow the development of Protobject, also to better understand its potentialities in the context of experimental design".
Second answer: "I would like to use Protobject without being able to program, facilitating the transition between real interactions and computer making it more straightforward. In the future, I think that Protobject could help the exchange of knowledge between programmers and designers, letting them to share their practice with the purpose of co-design".

Participant 17, product designer
First answer: "[The experience] was new, really useful and I learnt how to develop simple code with a software and a smartphone".
Second answer: n/a

Participant 18, consultant
First answer: "[The experience] was really useful".
Second answer: n/a

Extracted themes and discussion. I grouped the answers of the participants around their main themes, as follows:

Usefulness/interestingness of the experience. Many participants perceived the experience as useful and/or interesting (P1, P4, P6, P7, P8, P9, P10, P11, P12, P15, P16 P17 and P18).

[3]The participant does not seems to be a native speaker of Italian. Due to some unclear sentences in Italian, his/her answers were translated making them understandable.

Clarity of the initial presentation. P4 stated that the initial presentation did not make clear Protobject value and meaning. Also P10 stated that the presentation of Protobject was not clear, thinking (together with P8) that Protobject could be used for final systems instead of exclusively for prototyping. Anyhow, even if thinking about final systems, P8 understood that Protobject is designed for prototyping. P10 would have preferred clearer examples of Protobject usage inside the design process of an interactive system (i.e., making it clearer that initial prototypes could be made with Protobject, but prototypes closer to the final product should be built with Arduino).

User interface of Protobject. According to P4, the user experience could be improved. Also P8 expressed the same idea saying moreover that a user guide could be enough. According to P7, the user interface could be improved to let people with different backgrounds understand it easily. P10 stated that the icons of Protobject were not so clear. Also P11 stated that the UI is not so intuitive especially for people who are not familiar with programming. Anyhow, P12 stated that Protobject is easy to use and suitable for unskilled users and P9 also claimed that Protobject is intuitive.

Meemoo editor. P2 stated that Meemoo has no affordance in many important points and additional help would be needed. Also P12 expressed somehow the same ideas claiming that Meemoo should be supported by tool-tips for helping users.

Towards real usage. P1 expressed the intention to use it for lighting projects whereas P11 for interior design projects. Also P12 declared the intention to use Protobject for designing interaction design projects.

Thinking of applications. P2, P4, P6 were thinking of applications where Protobject could be used. Those comments suggest that, even if Protobject scopes are clear, for many participants it is not easy to imagine applications that can be prototyped with Protobject.

Make ideas that really work. P3 learned that it is possible to reinvent the use of any object just using old smartphones.

Simplifying the understanding when programming interactive systems. P4 would be not so scared when prototyping interactive systems. Moreover, P5 has a clearer idea regarding programming and P17 learned (and therefore understood) how to develop simple interactive behaviours. Somehow, P16 also expressed a similar evaluation claiming that Protobject could help designers since it simplifies their understanding of interactive systems because knowing technology is not required.

Meta-design helping collaboration between designers and programmers. P7 stated that Protobject could be used for meta-design process for making prototypes understandable. This suggests that Protobject could be used for letting designers and programmers work together. This was made clearer by P16, who said that Protobject can create a bridge between designers and programmers letting them share common practices for co-design.

Interfacing with JavaScript. P8 stated that the good interfacing with Javascript lets programmers focus straightforwardly on design rather than wasting time on programming.

App for iPhone. P16 stated that the app for iPhone is missing. This suggests that he/she probably has an iPhone and would like to try Protobject in the future.

Training way. P14 stated that the training could be inconvenient. In fact, there could be wanted or unwanted movements of the camera that compel users to train again the system.

8.9.2 Observation

During the workshop, I observed different behaviour of the participants. The most relevant ones are discussed in this section.

Use of states for prototyping

Listening to some of the participants while they were working, I could observe that designers are frequently unable to understand, without the help of a programmer, which are the relevant states for defining the interactive behaviour of a system. This is probably because of their ways of thinking are different from the programmers'. Anyhow, after explaining the rationale to them, they seem to become aware of the reasons why those states were chosen. Therefore, the approach used by Protobject, can help to support communication between designers and developers facilitating their mutual understanding. Thanks to Protobject, the state of objects (i.e., the environment seen as a state machine) become their common language and designer learn to see objects and/or environment as state machine.

Definition of regions

Participants generally were not able to define autonomously regions on objects in the best possible way. In fact, they tended to define regions on the entire objects rather than only where relevant changes of states were observable. This means that the process described in section 3.1.2 has to be carefully explained to Protobject users in order to make it clear.

Engagement with Protobject

The most tech-savvy participants and some designers seemed to be really engaged when designing with Protobject. In particular, at the first session of the workshop, I observed that the three programmers used Protobject for more than the three expected hours of the workshop exploring its functionalities. Also two designers stayed overtime at the workshop together with two programmers. At the second session of the workshop, the team composed of two designers remained more than three hours asking explanations on many of the different functionalities of Protobject. In particular, a participant took notes of any explained step asking also how to use Protobject with IFTTT. In this regard, a live demonstration was provided.

8.9.3 Feedback received during the workshop

Most of the feedback received during the workshop was also reported by the participants in the questionnaire. Therefore, I do not add comments regarding that.

The only additional feedback I report was given by a product designer. She suggested that Protobject can be developed in two directions, the first maker-oriented and the second designer-oriented. For her, Protobject can be really attractive for makers since it lets them be fast when designing interactive behaviours. Regarding designers, instead, Protobject lets them design interactive systems faster reducing errors. In this regard, she told us one experience she made while designing the case of a router. In that case, the customer gave a short time (one week) to the designers for designing the router and its interactive behaviour. This time was not enough to develop a real interactive product and interaction could be just imagined. When interactivity was added to the product (in a next stage) it appeared to be not usable. On the contrary, using Protobject the product could be interactive at early stages and designers (and the customer) can experiment with it earlier.

8.10 Evaluation discussion

All in all, Protobject was well received by workshop participants. Most of them consider it useful, interesting and learned a lot. In particular, tech-savvy participants appreciated its simplicity when developing interactive systems. Some designers, instead, appreciated the meta-design process enabled by Protobject. Most of them, in fact, increased their awareness of the operating way of interactive systems.

Chapter 9

Conclusions

9.1 Creativity support

Protobject seems to be a successful platform able to support the creativity [BLM03] of designers, makers, researchers and developers. In fact, according to the informal sessions with users described in chapter 6, many different ideas came out thanks to Protobject. Those that well represent the range of the possibilities opened by Protobject were easily developed in the usage scenarios described before. Moreover, a new design idea came out from a workshop session, even if idea generation was not a workshop aim. The idea consisted in a smart key hook that, if the user forgets the keys, activates an alert system near the door, when she opens it before leaving home. This was an interesting result considering the limited time and limited sets of objects that there were at the workshop. This indicates that Protobject may help users to be more creative.

9.2 Different approaches: sensors and Protobject

In contrast with the Protobject approach, I could state that using sensors has the advantage to give a sort of "mental model" to solve problems to makers and designers. This means that users know exactly the scope for which a sensor was designed, i.e., sensing rotations, light intensity, noise, weight, presence, and so on. Therefore, having something to detect (e.g. presence/absence), people often choose the right sensor (e.g. a proximity sensor) without thinking much about it.

With Protobject, instead, users are compelled to think "out of the scheme". They have to find a creative solution to solve a problem, reinventing the way they use objects and cameras to understand what they need. This is not always easy: for example, I am not able to describe exactly the creative process required for designing the tangible UI components described in the usage scenario 3. Although its final design was considered 'exciting' by some people, the creative process required several efforts since it consisted of a cycle where different solutions were created and tested.

I think that when users are able to find a solution for their scope in a short time thanks to Protobject, it is advantageous, otherwise, they could go back to a more traditional way of prototyping, i.e., Arduino, standard sensors and 3D printers. In fact, when using sensors, the capability to detect something is intrinsic in the same sensors. On the contrary, when using Protobject, the capability to sense something depends especially on the user's ability to think "out of the scheme" in a creative way, as bricoleurs are used to do.

9.3 Further research

To better support non-tech-savvy users, Meemoo could be extended through further modelling constructs (i.e., modules) to enrich it. This would facilitate non-tech-savvy users to develop more complex applications using the visual notation already offered by Meemoo. To define these new modelling constructs, further research could be carried out for identifying recurrent requisites of the class of applications that can be developed with Protobject. Therefore, designers without any programming competences could "compose" - through a well-defined composition model - almost any kind of application using Meemoo. Moreover, also other visual programming languages different from Meemoo could be evaluated. Actually, different approaches were developed in this direction, i.e., to allow non-tech-savvy users to design their software using visual notations. Those approaches were used also for the composition of mashups. On the dark side, some of these tools had little success (e.g., Yahoo Pipe, which disappeared few years ago) because of their difficulty of use [MO14], and/or are prevalently used for teaching scopes (e.g., Google Blockly). Moreover, several studies demonstrated that the overall score of students who use a conventional syntax-based language are significantly higher than those who learned by using blocks-based visual programming languages [CMP+11]. So, I wonder whether it is worth using visual programming languages even if just for learning. To conclude, the overall effectiveness (easiness of use, user acceptance, user learning benefits) of visual composition tools for Protobject deserve to be investigated, also trying to use different open-source visual programming languages like Google Blocky.

9.4 Social impact of Protobject

We have moved from a “read-only” culture to a “read/write” culture [Les08]: instead of passively consuming information or products, users now actively participate in creating their own, personalized objects. Moreover, since not only makers remix, everyday users can consistently reuse and re-appropriate mass-produced designs [WM07]. In this regard, I think that Protobject may also support this cultural change enabling users to design, create, reinvent and remix physical objects that previously required mass manufacture. In fact, it can help any users (also those who are not very skilled in hardware, software and programming) to get closer to the maker culture for prototyping personalized objects.

Finally, I chose to release Protobject as free software (https://protobject.com) to speed up its development. My future aim is to support a community of designers, makers and developers who design, share and remix prototypes and ideas. Like Arduino, I imagine Protobject as an open project able to make the early stages of prototyping easier and faster.

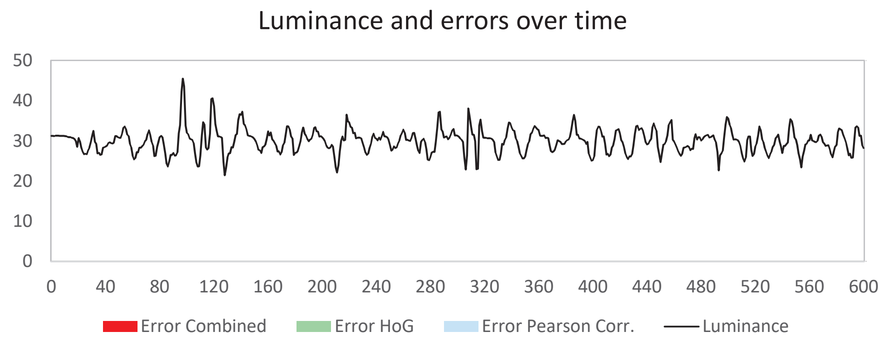
Appendix A

Robustness of the detection

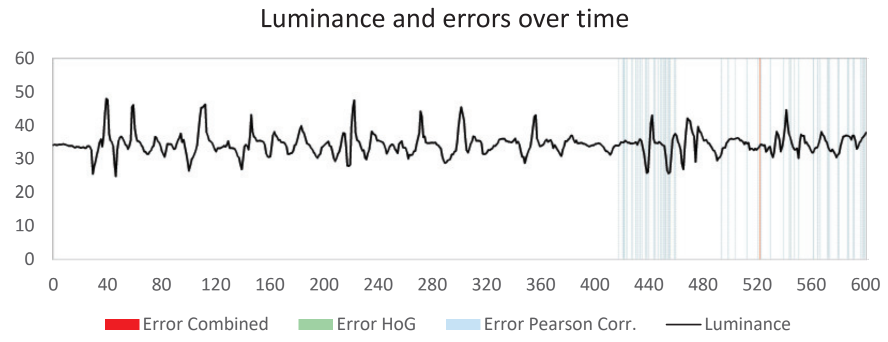
These graphics display detection errors carried out by different algorithms (Pearson, HoG and combined HoG+Pearson) for each state of objects over 600 detection (x axis) displaying also luminance variations (y axix).

A.1 Amount of water from a tube

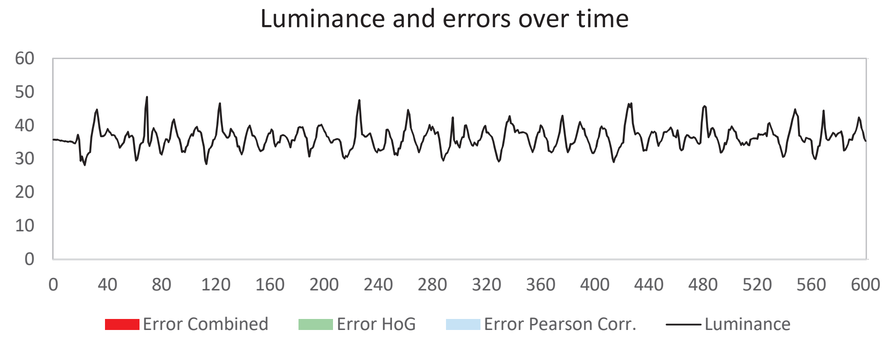
		Errors		*Luminance*	
Object:	Water-tube	**HoG+Pearson:**	0	**Max:**	45.47
State:	NoWater	**HoG:**	0	**Min:**	21.42
		Pearson:	0	**Average:**	29.94
				SD:	2.91



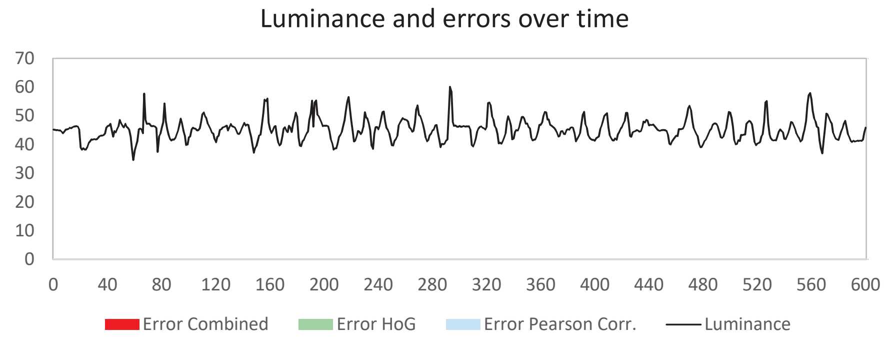
			Errors		*Luminance*
Object:	Water-tube	**HoG+Pearson:**	1	**Max:**	47.96
State:	Am1	**HoG:**	0	**Min:**	24.87
		Pearson:	53	**Average:**	34.40
				SD:	3.18



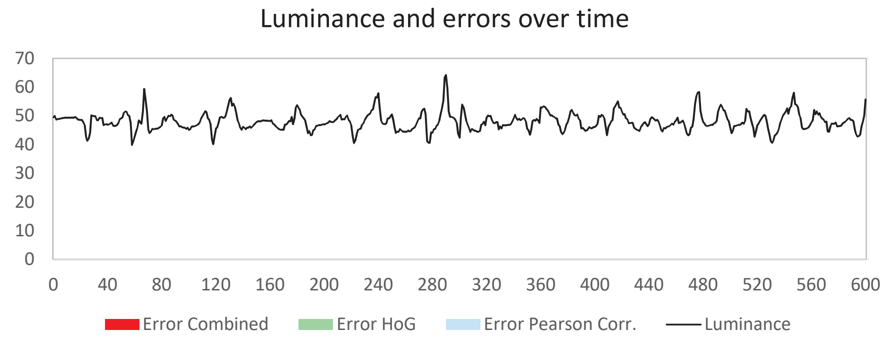
			Errors		*Luminance*
Object:	Water-tube	**HoG+Pearson:**	0	**Max:**	48.51
State:	Am2	**HoG:**	0	**Min:**	28.12
		Pearson:	0	**Average:**	36.24
				SD:	3.22



			Errors		*Luminance*
Object:	Water-tube	**HoG+Pearson:**	0	**Max:**	60.10
State:	Am3	**HoG:**	0	**Min:**	34.57
		Pearson:	0	**Average:**	45.18
				SD:	3.68

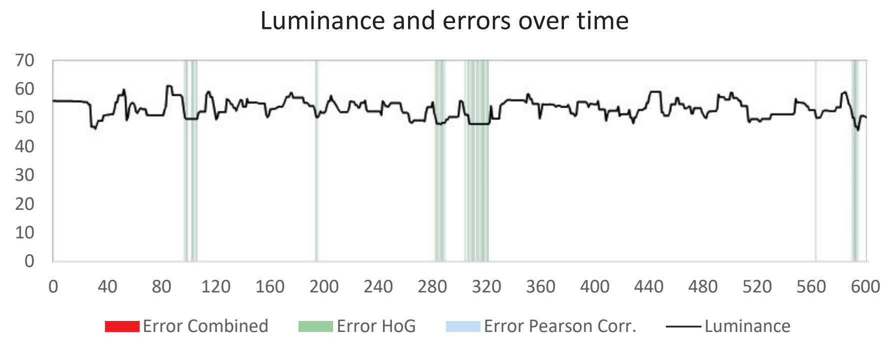


			Errors		*Luminance*
Object:	Water-tube	**HoG+Pearson:**	0	**Max:**	64.21
State:	MaxWater	**HoG:**	0	**Min:**	39.83
		Pearson:	0	**Average:**	48.01
				SD:	3.14

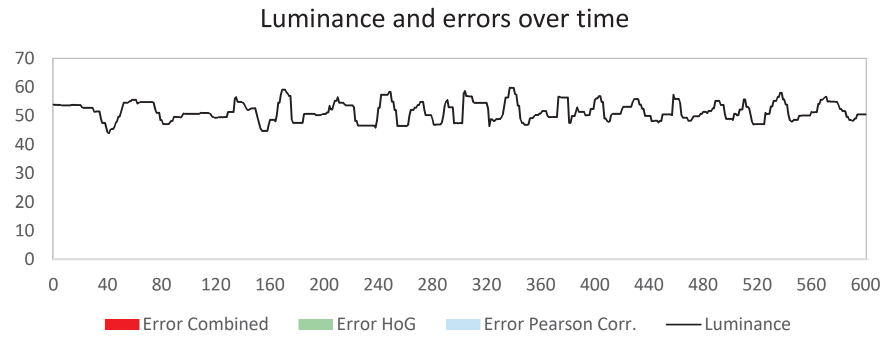


A.2 Curtain position

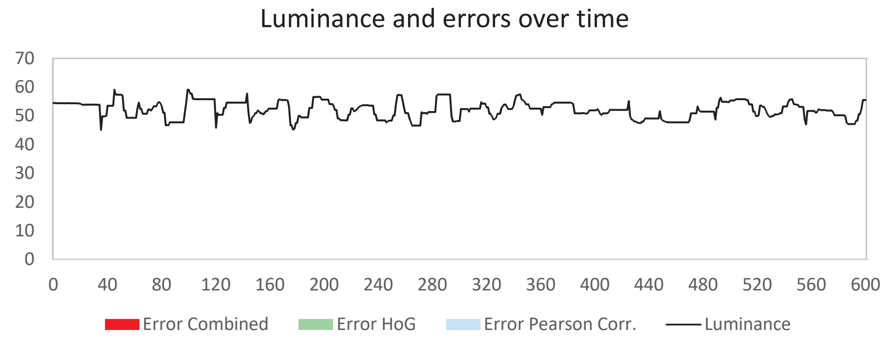
			Errors		***Luminance***
Object:	Curtain	**HoG+Pearson:**	0	**Max:**	61.06
State:	Pos00	**HoG:**	41	**Min:**	45.67
		Pearson:	0	**Average:**	53.20
				SD:	2.91



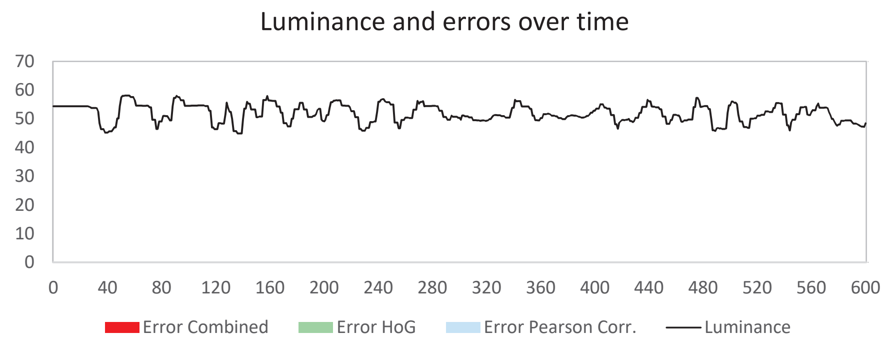
			Errors		*Luminance*
Object:	Curtain	**HoG+Pearson:**	0	**Max:**	59.71
State:	Pos01	**HoG:**	0	**Min:**	43.94
		Pearson:	0	**Average:**	51.48
				SD:	3.23



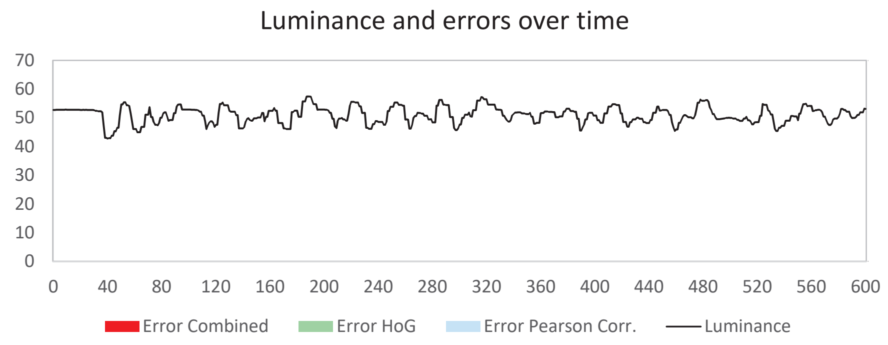
			Errors		*Luminance*
Object:	Curtain	**HoG+Pearson:**	0	**Max:**	59.12
State:	Pos02	**HoG:**	0	**Min:**	45.09
		Pearson:	0	**Average:**	52.08
				SD:	2.86



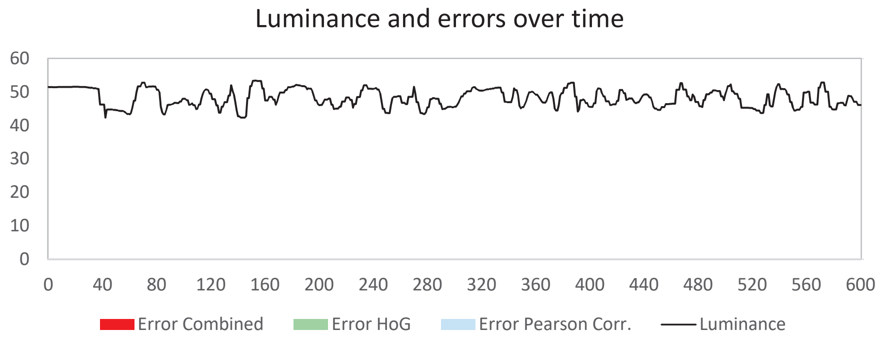
		Errors		*Luminance*	
Object:	Curtain	**HoG+Pearson:**	0	**Max:**	58.13
State:	Pos03	**HoG:**	0	**Min:**	44.88
		Pearson:	0	**Average:**	51.83
				SD:	3.06



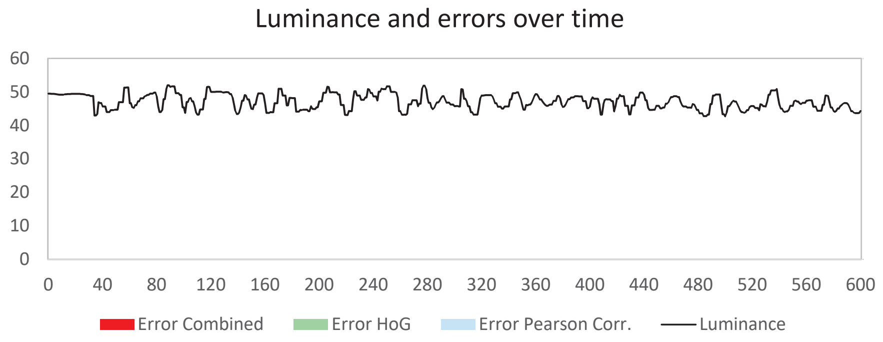
		Errors		*Luminance*	
Object:	Curtain	**HoG+Pearson:**	0	**Max:**	57.41
State:	Pos04	**HoG:**	0	**Min:**	42.76
		Pearson:	0	**Average:**	50.97
				SD:	2.80



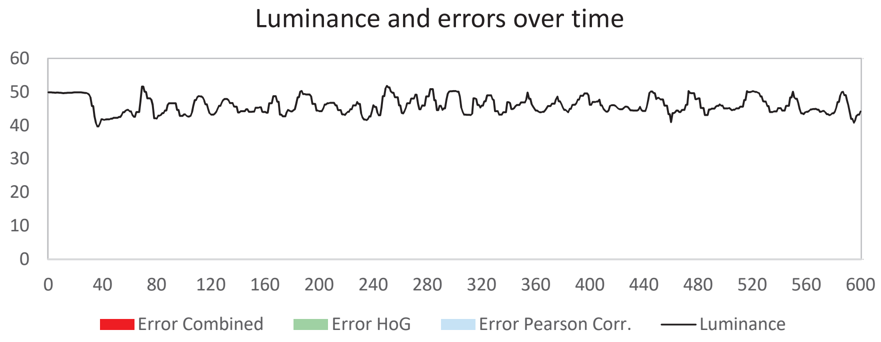
			Errors		*Luminance*
Object:	Curtain	**HoG+Pearson:**	0	**Max:**	53.35
State:	Pos05	**HoG:**	0	**Min:**	42.28
		Pearson:	0	**Average:**	48.17
				SD:	2.62



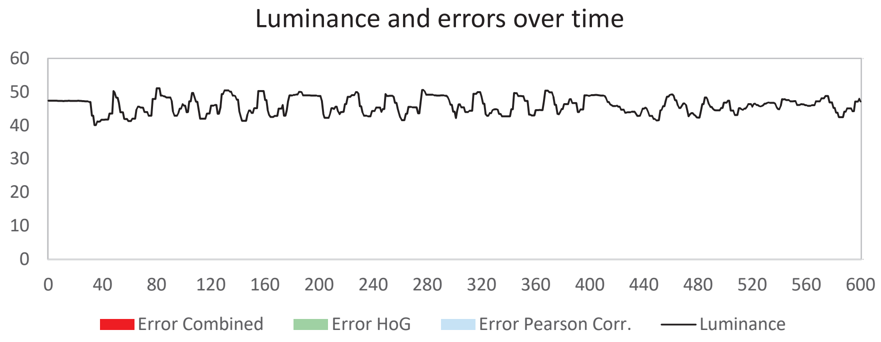
			Errors		*Luminance*
Object:	Curtain	**HoG+Pearson:**	0	**Max:**	51.97
State:	Pos06	**HoG:**	0	**Min:**	42.68
		Pearson:	0	**Average:**	47.08
				SD:	2.23



			Errors		*Luminance*
Object:	Curtain	**HoG+Pearson:**	0	**Max:**	51.82
State:	Pos07	**HoG:**	0	**Min:**	39.70
		Pearson:	0	**Average:**	45.94
				SD:	2.42

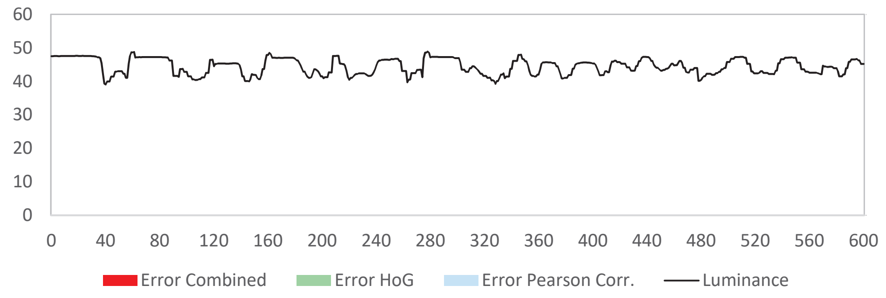


			Errors		*Luminance*
Object:	Curtain	**HoG+Pearson:**	0	**Max:**	51.07
State:	Pos08	**HoG:**	0	**Min:**	40.06
		Pearson:	0	**Average:**	45.93
				SD:	2.46



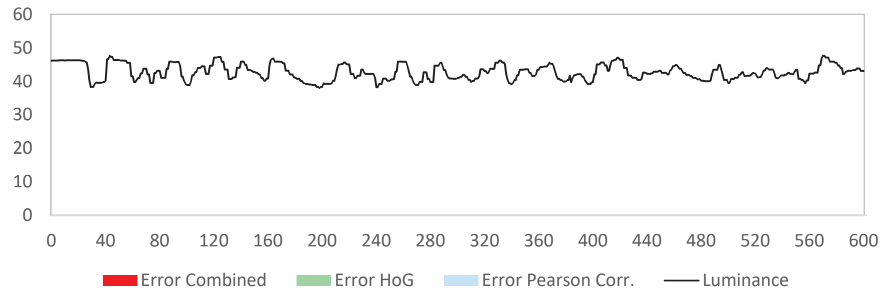
			Errors		*Luminance*
Object:	Curtain	**HoG+Pearson:**	0	**Max:**	48.88
State:	Pos09	**HoG:**	0	**Min:**	39.01
		Pearson:	0	**Average:**	44.43
				SD:	2.36

Luminance and errors over time



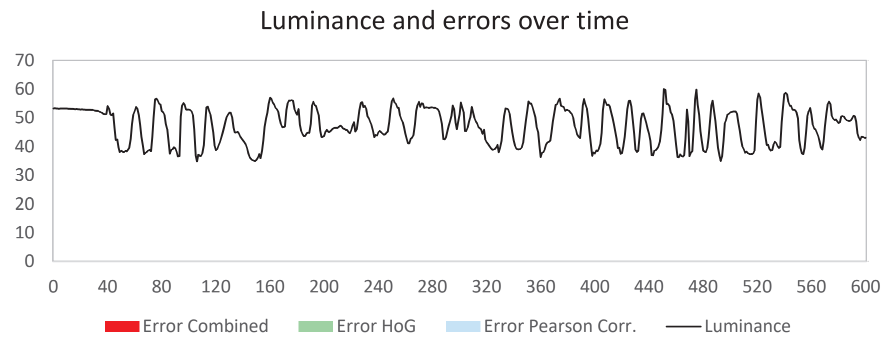
			Errors		*Luminance*
Object:	Curtain	**HoG+Pearson:**	0	**Max:**	47.63
State:	Pos10	**HoG:**	0	**Min:**	37.96
		Pearson:	0	**Average:**	42.71
				SD:	2.33

Luminance and errors over time

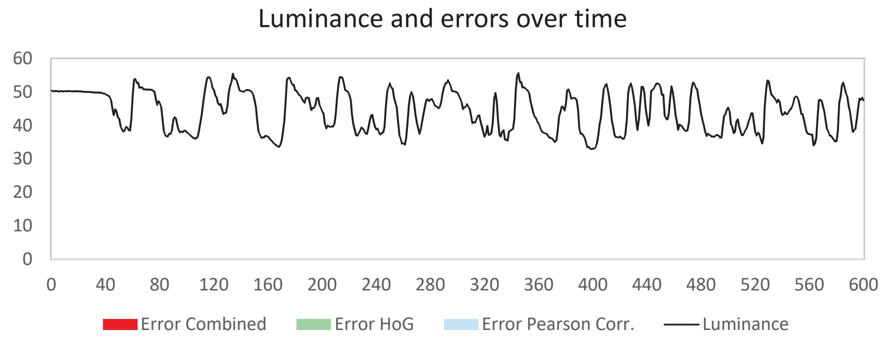


A.3 Mecchanical button

			Errors		*Luminance*
Object:	Button	**HoG+Pearson:**	0	**Max:**	60.00
State:	Off	**HoG:**	0	**Min:**	34.76
		Pearson:	0	**Average:**	47.14
				SD:	6.16



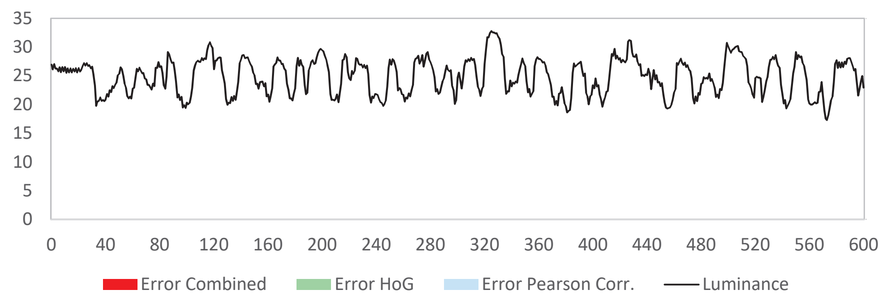
			Errors		***Luminance***
Object:	Button	**HoG+Pearson:**	0	**Max:**	55.61
State:	On	**HoG:**	0	**Min:**	32.90
		Pearson:	0	**Average:**	44.09
				SD:	5.86



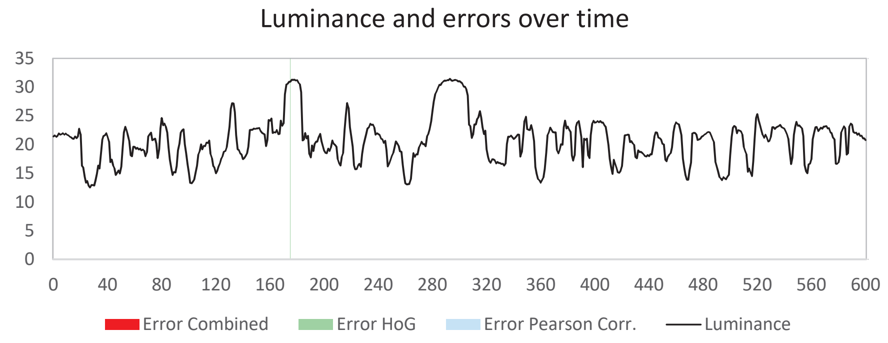
A.4 LCD display

			Errors		***Luminance***
Object:	LCD-display	**HoG+Pearson:**	0	**Max:**	32.79
State:	Digit0	**HoG:**	0	**Min:**	17.27
		Pearson:	0	**Average:**	24.87
				SD:	3.09

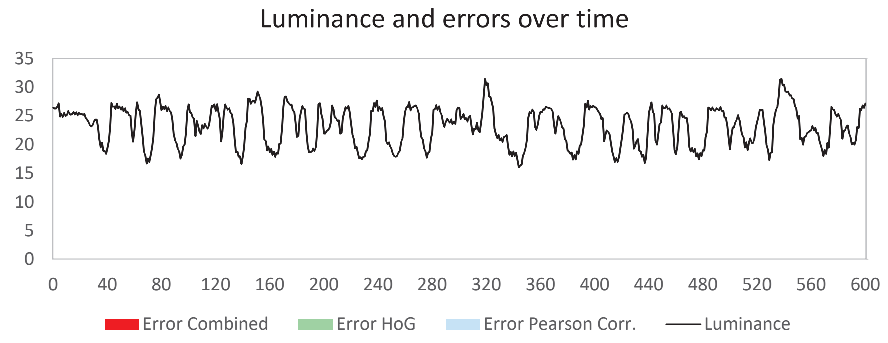
Luminance and errors over time



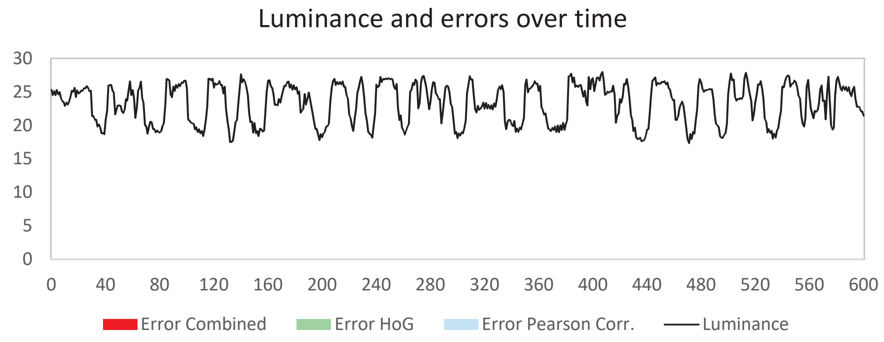
		Errors		*Luminance*	
Object:	LCD-display	**HoG+Pearson:**	0	**Max:**	31.44
State:	Digit1	**HoG:**	1	**Min:**	12.50
		Pearson:	0	**Average:**	20.62
				SD:	3.93



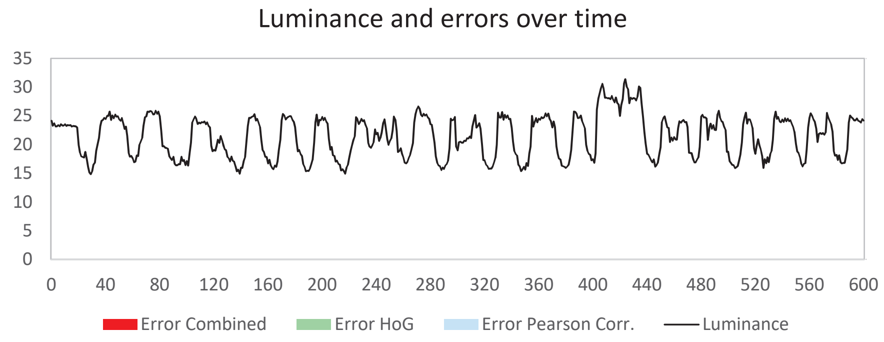
		Errors		*Luminance*	
Object:	LCD-display	**HoG+Pearson:**	0	**Max:**	31.46
State:	Digit2	**HoG:**	0	**Min:**	16.04
		Pearson:	0	**Average:**	23.24
				SD:	3.28



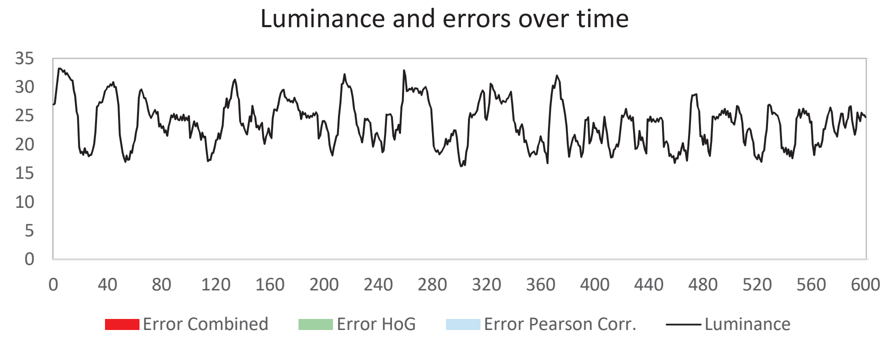
			Errors		*Luminance*
Object:	LCD-display	**HoG+Pearson:**	0	**Max:**	27.97
State:	Digit3	**HoG:**	0	**Min:**	17.33
		Pearson:	0	**Average:**	23.28
				SD:	2.89



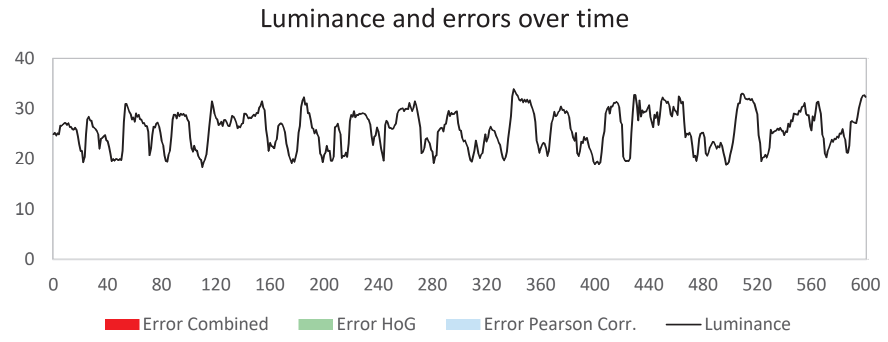
			Errors		*Luminance*
Object:	LCD-display	**HoG+Pearson:**	0	**Max:**	31.39
State:	Digit4	**HoG:**	0	**Min:**	14.82
		Pearson:	0	**Average:**	21.38
				SD:	3.66



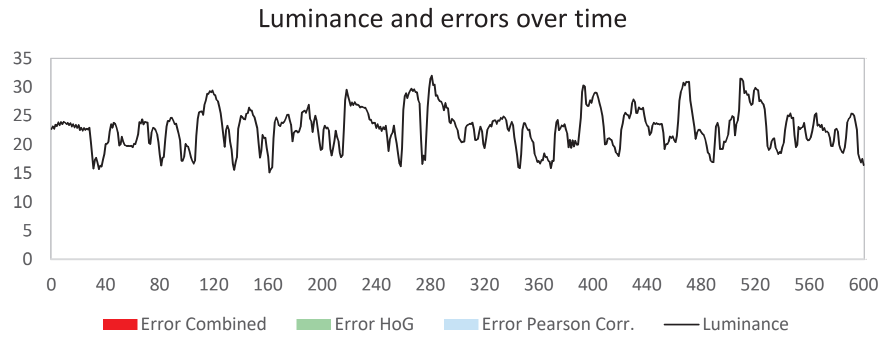
			Errors		*Luminance*
Object:	LCD-display	**HoG+Pearson:**	0	**Max:**	33.24
State:	Digit5	**HoG:**	0	**Min:**	16.20
		Pearson:	0	**Average:**	23.74
				SD:	3.86



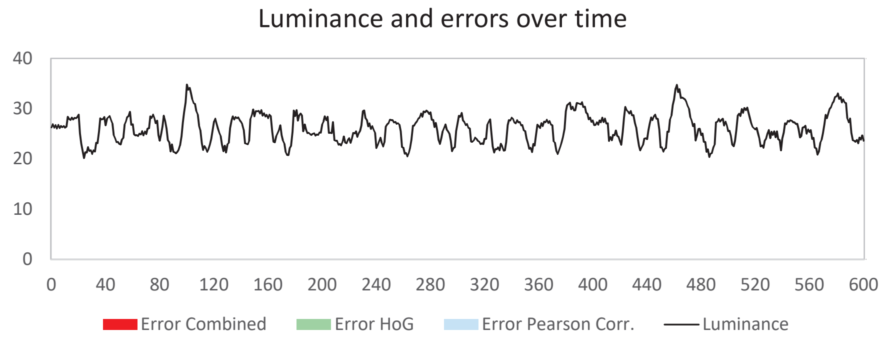
			Errors		*Luminance*
Object:	LCD-display	**HoG+Pearson:**	0	**Max:**	33.87
State:	Digit6	**HoG:**	0	**Min:**	18.36
		Pearson:	0	**Average:**	25.87
				SD:	3.74



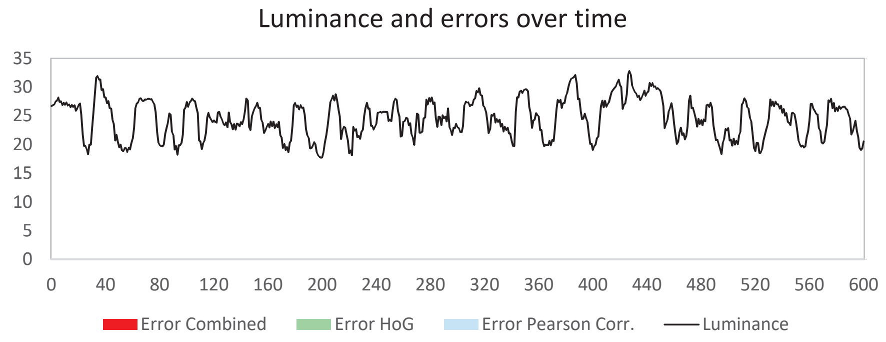
			Errors		*Luminance*
Object:	LCD-display	**HoG+Pearson:**	0	**Max:**	31.99
State:	Digit7	**HoG:**	0	**Min:**	15.11
		Pearson:	0	**Average:**	22.89
				SD:	3.46



			Errors		*Luminance*
Object:	LCD-display	**HoG+Pearson:**	0	**Max:**	34.78
State:	Digit8	**HoG:**	0	**Min:**	20.15
		Pearson:	0	**Average:**	26.03
				SD:	2.87

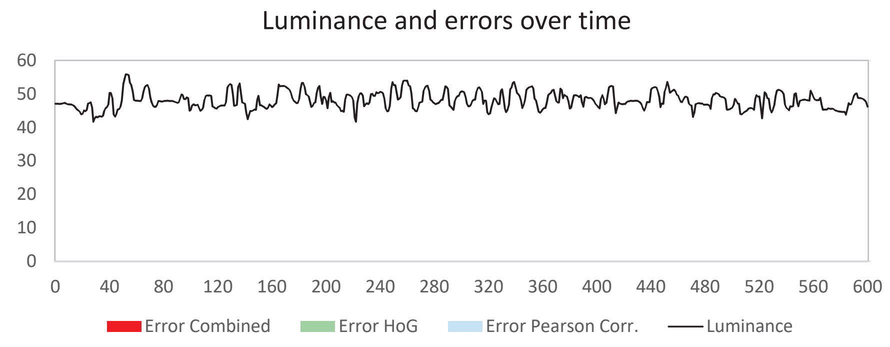


			Errors		*Luminance*
Object:	LCD-display	**HoG+Pearson:**	0	**Max:**	32.81
State:	Digit9	**HoG:**	0	**Min:**	17.71
		Pearson:	0	**Average:**	24.51
				SD:	3.27



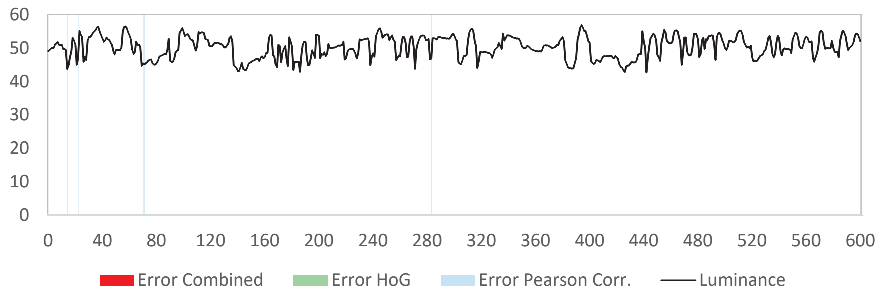
A.5 Oven knob position

			Errors		***Luminance***
Object:	Oven-knob	**HoG+Pearson:**	0	**Max:**	55.86
State:	Defrost	**HoG:**	0	**Min:**	41.68
		Pearson:	0	**Average:**	48.05
				SD:	2.49



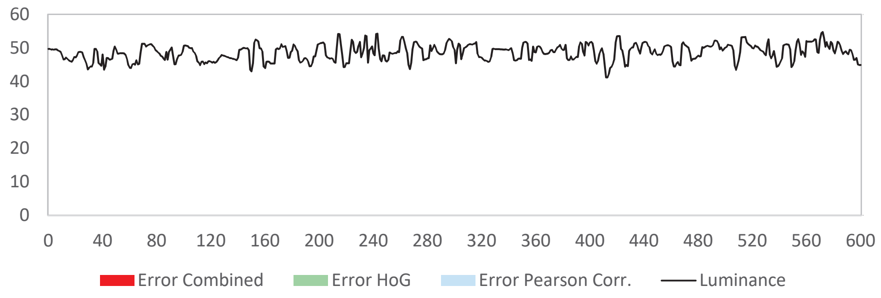
			Errors		*Luminance*
Object:	Oven-knob	**HoG+Pearson:**	0	**Max:**	56.82
State:	Fan	**HoG:**	0	**Min:**	42.71
		Pearson:	7	**Average:**	50.20
				SD:	3.20

Luminance and errors over time

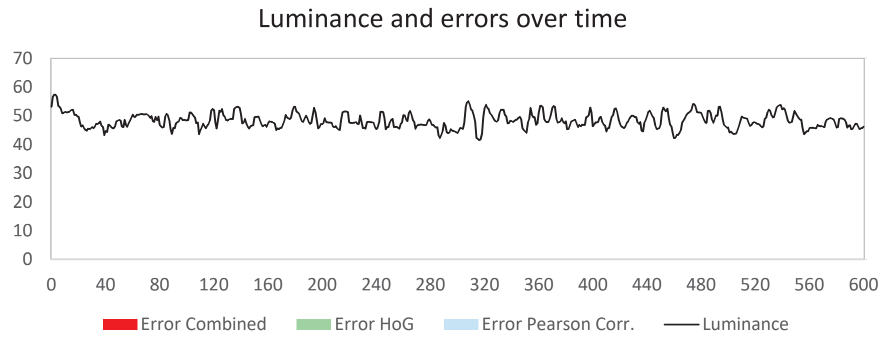


			Errors		*Luminance*
Object:	Oven-knob	**HoG+Pearson:**	0	**Max:**	54.75
State:	Grill	**HoG:**	0	**Min:**	41.14
		Pearson:	0	**Average:**	48.59
				SD:	2.42

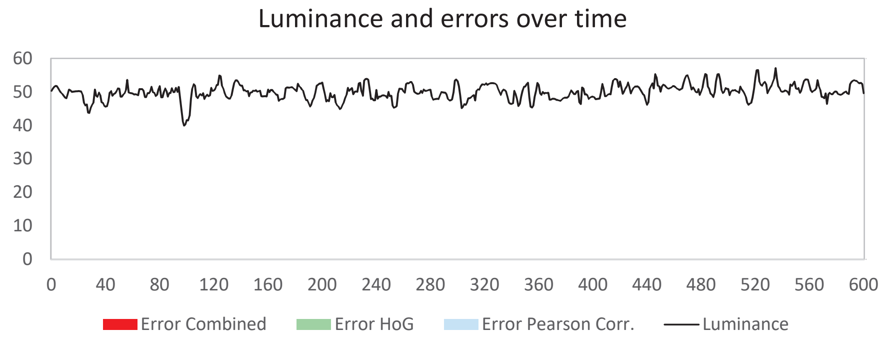
Luminance and errors over time



			Errors		*Luminance*
Object:	Oven-knob	**HoG+Pearson:**	0	**Max:**	57.41
State:	Light	**HoG:**	0	**Min:**	41.51
		Pearson:	0	**Average:**	48.30
				SD:	2.62

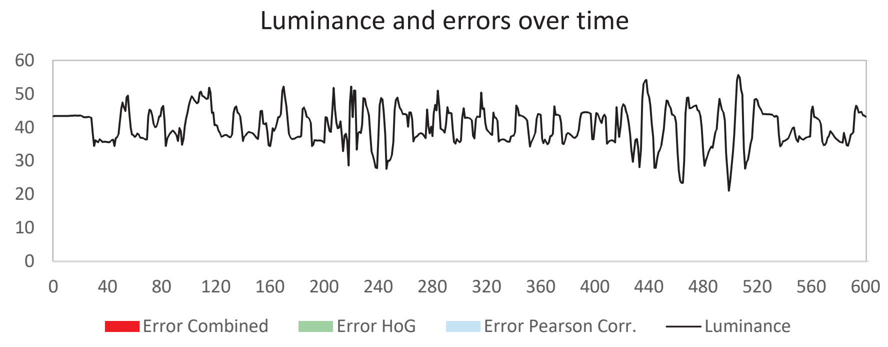


			Errors		*Luminance*
Object:	Oven-knob	**HoG+Pearson:**	0	**Max:**	57.11
State:	Off	**HoG:**	0	**Min:**	39.89
		Pearson:	0	**Average:**	49.94
				SD:	2.28

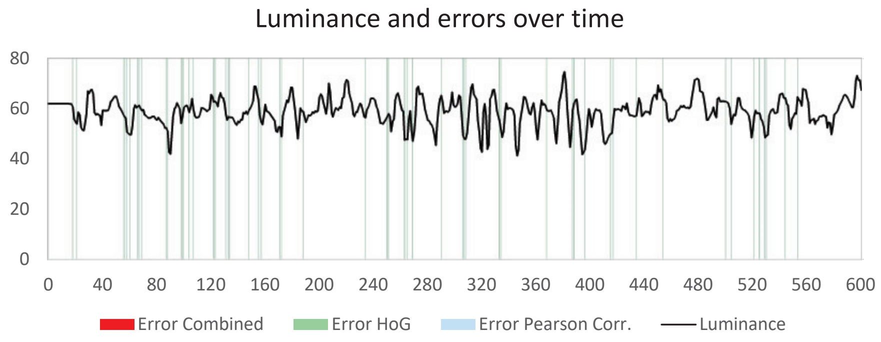


A.6 Colour of a surface

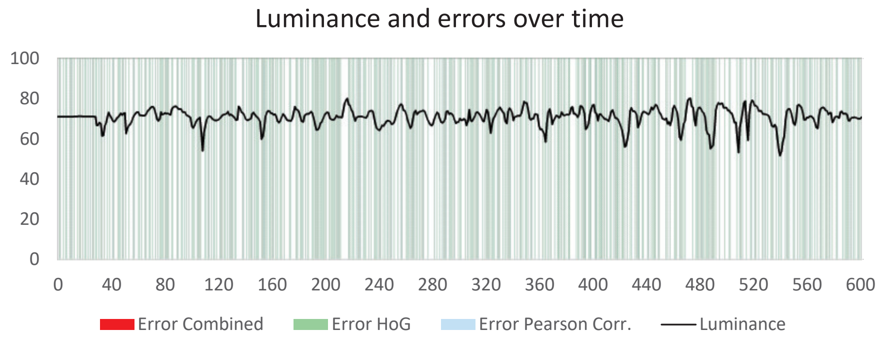
			Errors		*Luminance*
Object:	Post-it-color	**HoG+Pearson:**	0	**Max:**	55.66
State:	Brown	**HoG:**	0	**Min:**	21.07
		Pearson:	0	**Average:**	40.31
				SD:	5.39



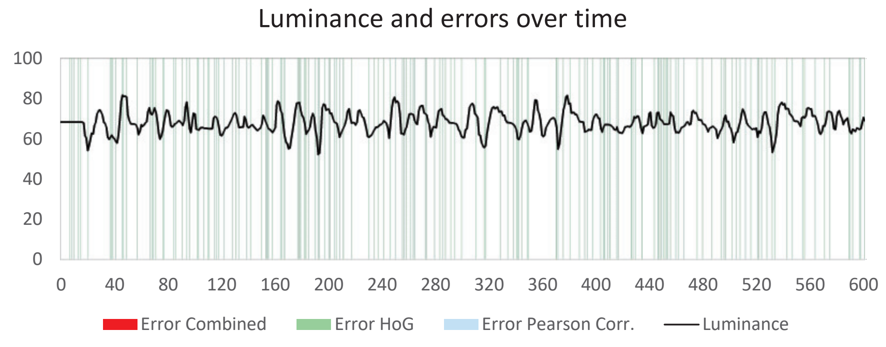
		Errors		*Luminance*	
Object:	Post-it-color	**HoG+Pearson:**	0	**Max:**	74.62
State:	Green	**HoG:**	54	**Min:**	41.34
		Pearson:	0	**Average:**	58.92
				SD:	5.48



		Errors		*Luminance*	
Object:	Post-it-color	**HoG+Pearson:**	0	**Max:**	80.11
State:	Pink	**HoG:**	398	**Min:**	51.66
		Pearson:	0	**Average:**	70.79
				SD:	4.11

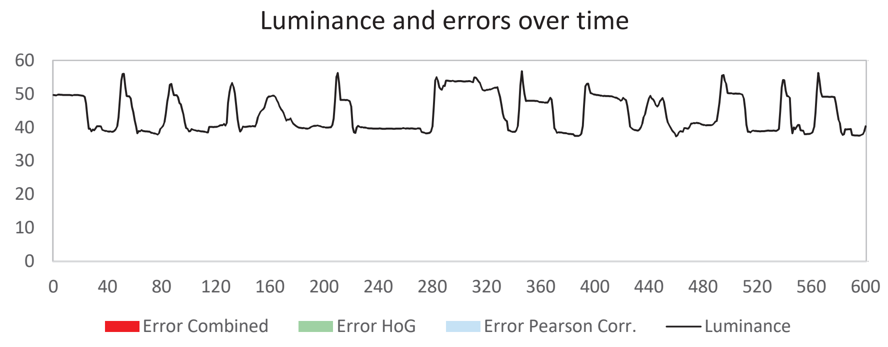


			Errors		*Luminance*	
Object:	Post-it-color	**HoG+Pearson:**	0	**Max:**	81.63
State:	Yellow	**HoG:**	158	**Min:**	52.21
		Pearson:	0	**Average:**	68.02
				SD:	4.98

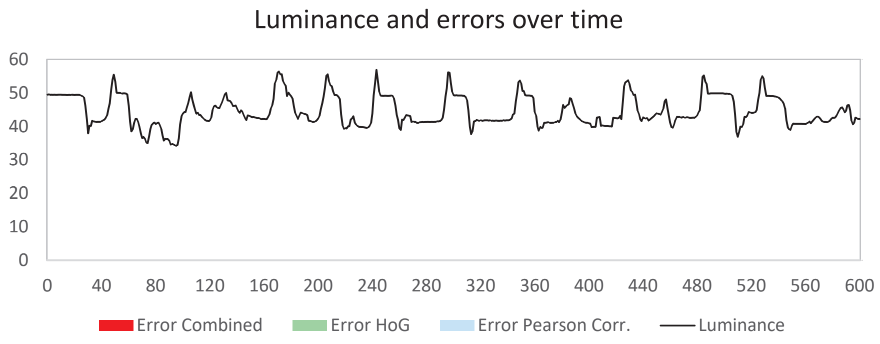


A.7 Position of a knob (colour-wheel-based marker)

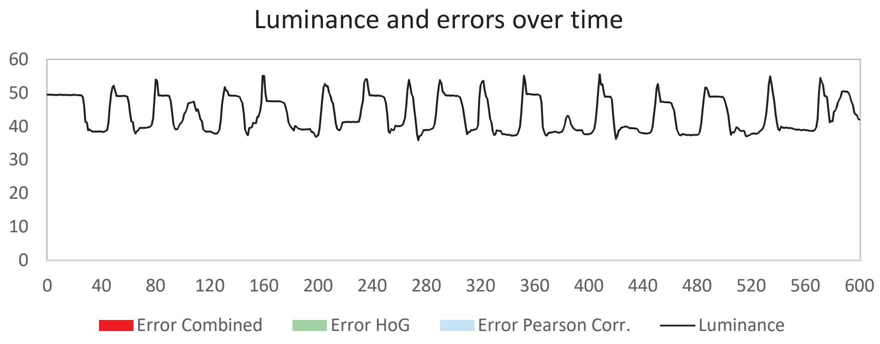
			Errors		***Luminance***
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	56.79
State:	0	**HoG:**	0	**Min:**	37.30
		Pearson:	0	**Average:**	44.03
				SD:	5.43



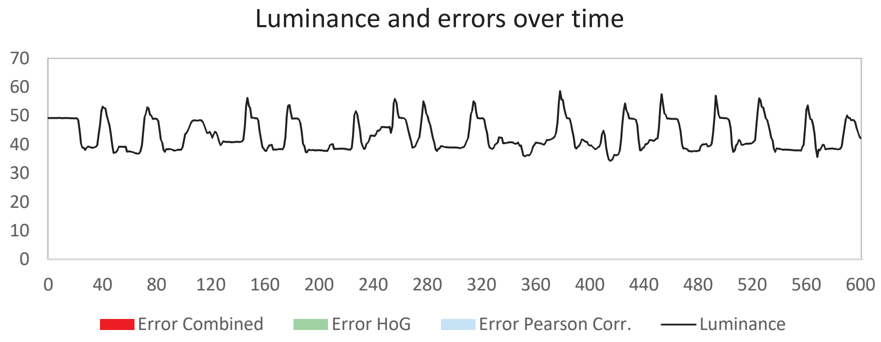
			Errors		***Luminance***
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	56.86
State:	25	**HoG:**	0	**Min:**	34.19
		Pearson:	0	**Average:**	44.50
				SD:	4.44



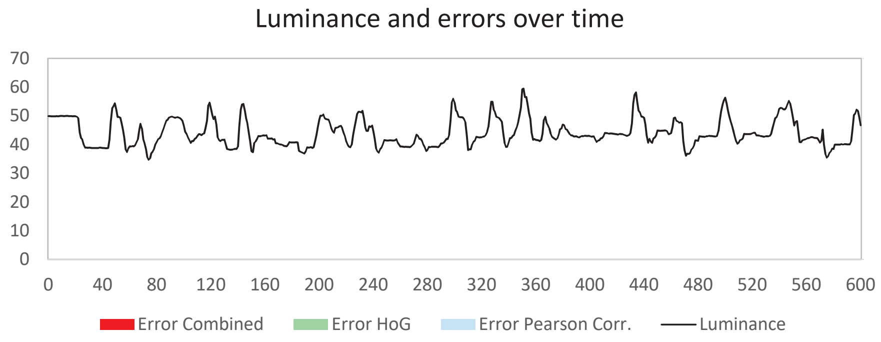
			Errors		***Luminance***
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	55.53
State:	50	**HoG:**	0	**Min:**	35.82
		Pearson:	0	**Average:**	43.51
				SD:	5.17



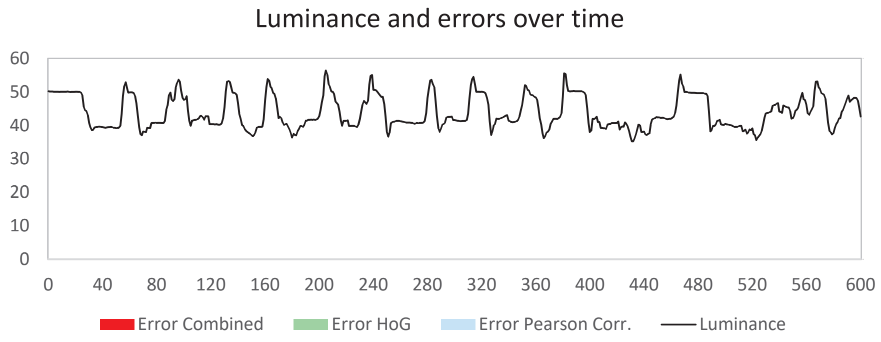
			Errors		*Luminance*
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	58.61
State:	75	**HoG:**	0	**Min:**	34.34
		Pearson:	0	**Average:**	43.06
				SD:	5.22



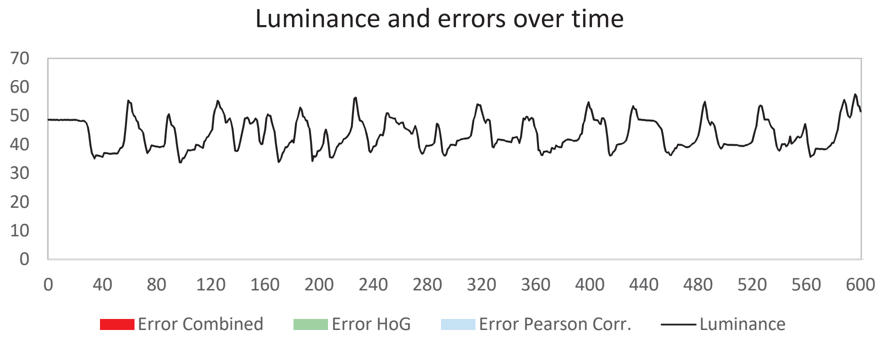
			Errors		*Luminance*
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	59.51
State:	100	**HoG:**	0	**Min:**	34.68
		Pearson:	0	**Average:**	44.06
				SD:	4.68



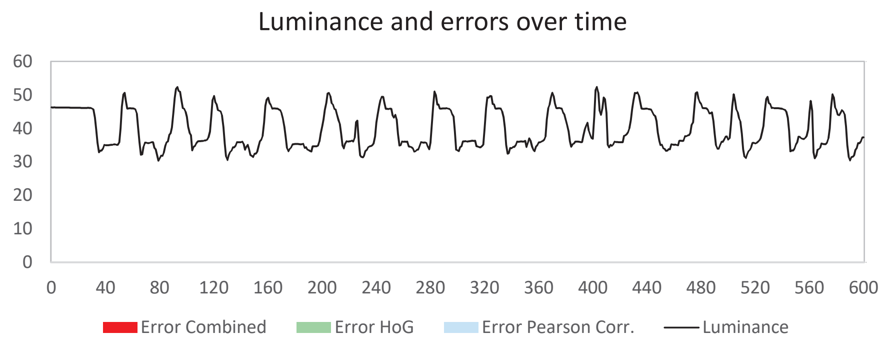
			Errors		*Luminance*
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	56.39
State:	125	**HoG:**	0	**Min:**	35.19
		Pearson:	0	**Average:**	43.86
				SD:	4.81



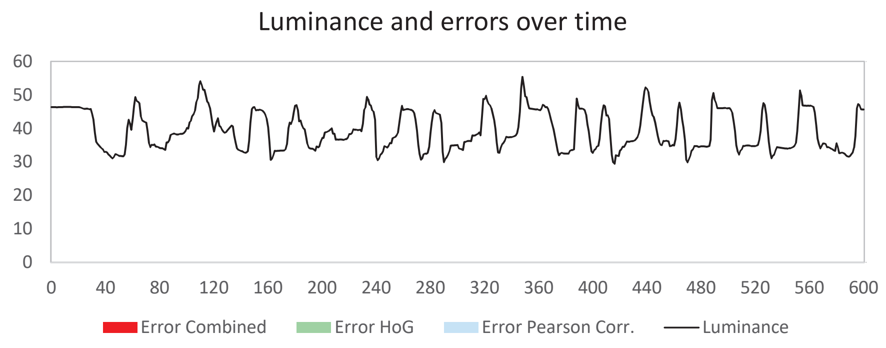
			Errors		*Luminance*
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	57.51
State:	150	**HoG:**	0	**Min:**	33.74
		Pearson:	0	**Average:**	43.55
				SD:	5.24



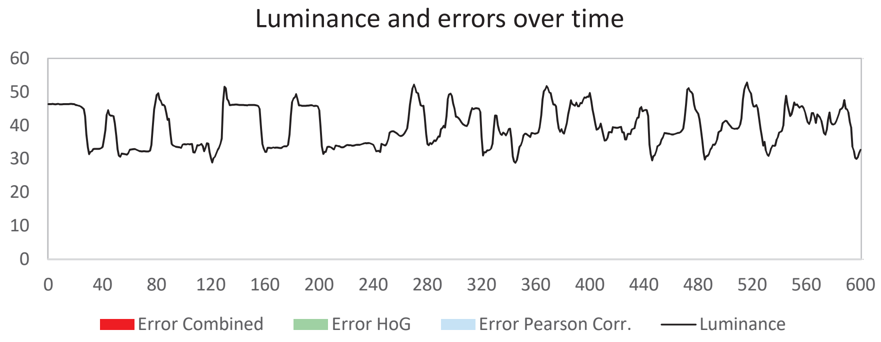
			Errors		***Luminance***
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	52.39
State:	175	**HoG:**	0	**Min:**	30.33
		Pearson:	0	**Average:**	39.83
				SD:	5.72



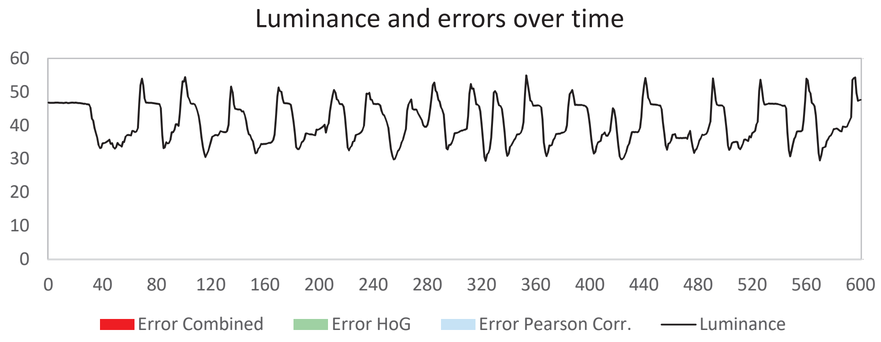
			Errors		***Luminance***
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	55.40
State:	200	**HoG:**	0	**Min:**	29.40
		Pearson:	0	**Average:**	39.26
				SD:	5.74



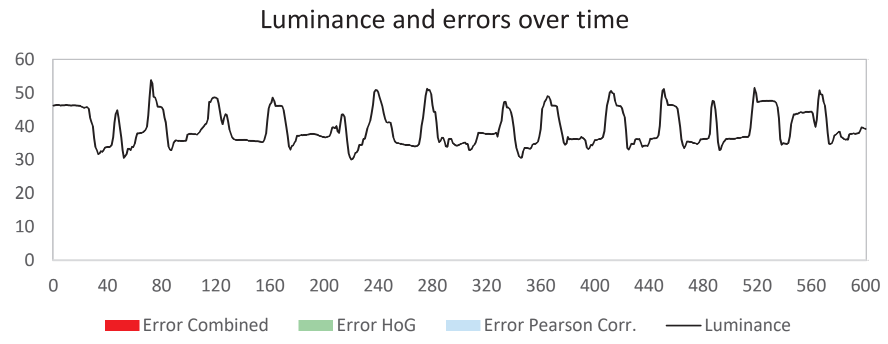
			Errors		***Luminance***
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	52.80
State:	225	**HoG:**	0	**Min:**	28.78
		Pearson:	0	**Average:**	39.52
				SD:	5.84



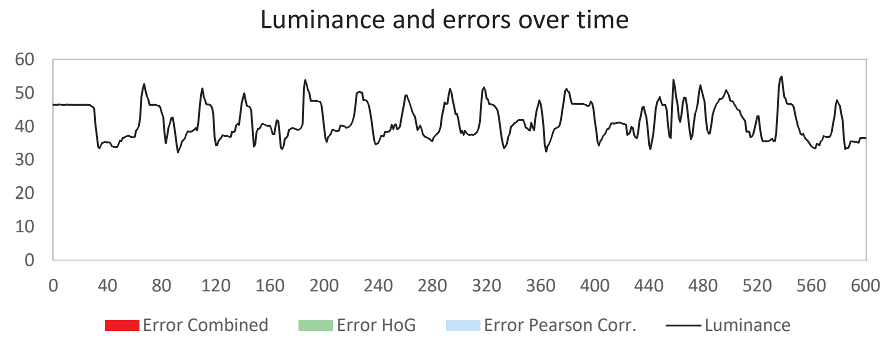
			Errors		***Luminance***
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	54.92
State:	250	**HoG:**	0	**Min:**	29.41
		Pearson:	0	**Average:**	40.78
				SD:	6.04



			Errors		*Luminance*
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	53.80
State:	275	**HoG:**	0	**Min:**	30.06
		Pearson:	0	**Average:**	39.45
				SD:	5.26

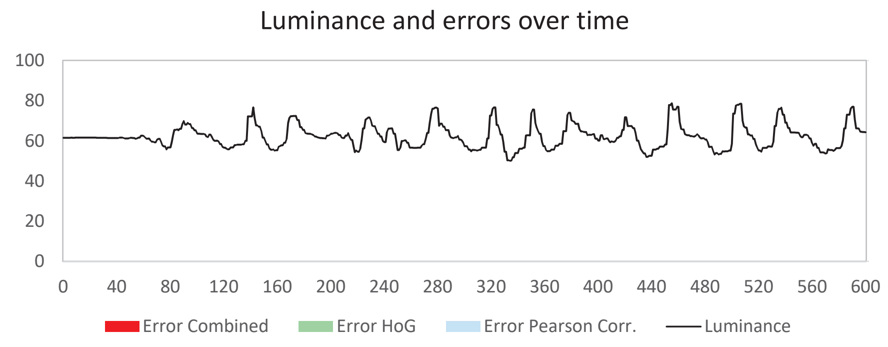


			Errors		*Luminance*
Object:	Color-strip-knob	**HoG+Pearson:**	0	**Max:**	54.90
State:	300	**HoG:**	0	**Min:**	32.22
		Pearson:	0	**Average:**	41.53
				SD:	5.06

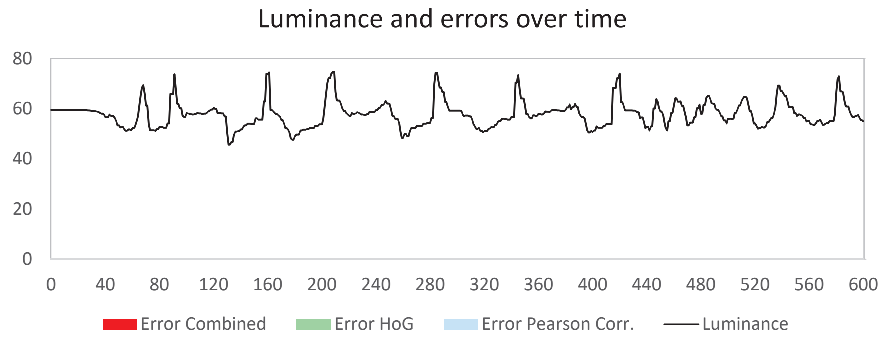


A.8 Liquid level

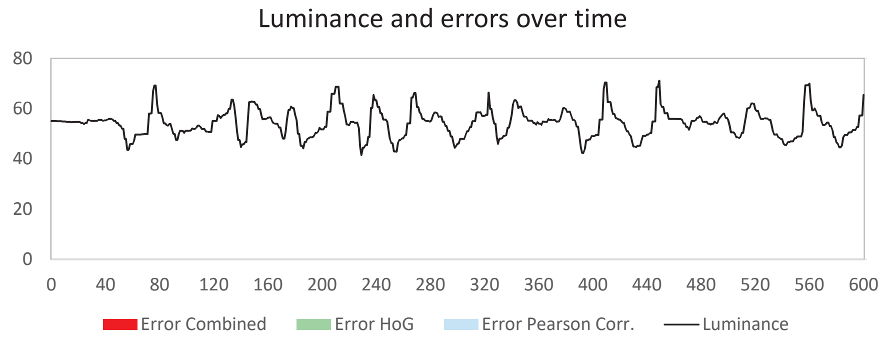
			Errors		***Luminance***
Object:	Liquid-level	**HoG+Pearson:**	0	**Max:**	78.69
State:	L0	**HoG:**	0	**Min:**	50.10
		Pearson:	0	**Average:**	62.14
				SD:	5.84



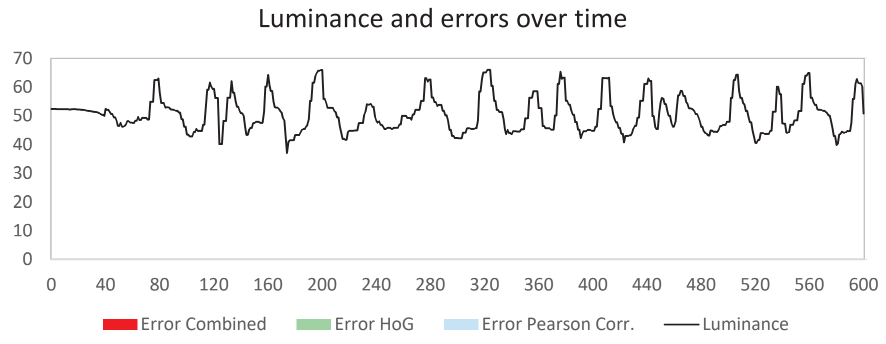
			Errors		***Luminance***
Object:	Liquid-level	**HoG+Pearson:**	0	**Max:**	74.63
State:	L1	**HoG:**	0	**Min:**	45.68
		Pearson:	0	**Average:**	57.72
				SD:	5.15



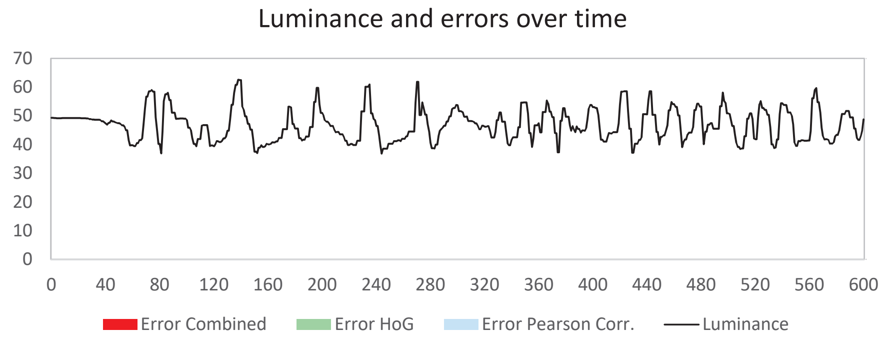
			Errors		***Luminance***
Object:	Liquid-level	**HoG+Pearson:**	0	**Max:**	71.12
State:	L2	**HoG:**	0	**Min:**	41.53
		Pearson:	0	**Average:**	54.15
				SD:	5.39



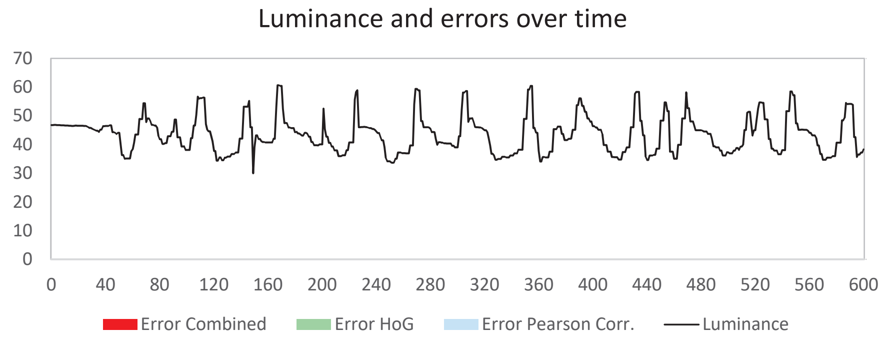
			Errors		***Luminance***
Object:	Liquid-level	**HoG+Pearson:**	0	**Max:**	65.99
State:	L3	**HoG:**	0	**Min:**	37.07
		Pearson:	0	**Average:**	50.61
				SD:	6.14



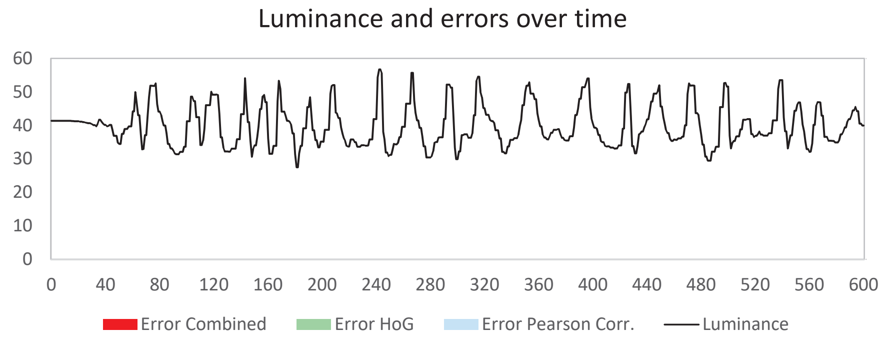
			Errors		***Luminance***
Object:	Liquid-level	**HoG+Pearson:**	0	**Max:**	62.59
State:	L4	**HoG:**	0	**Min:**	36.79
		Pearson:	0	**Average:**	46.86
				SD:	5.49



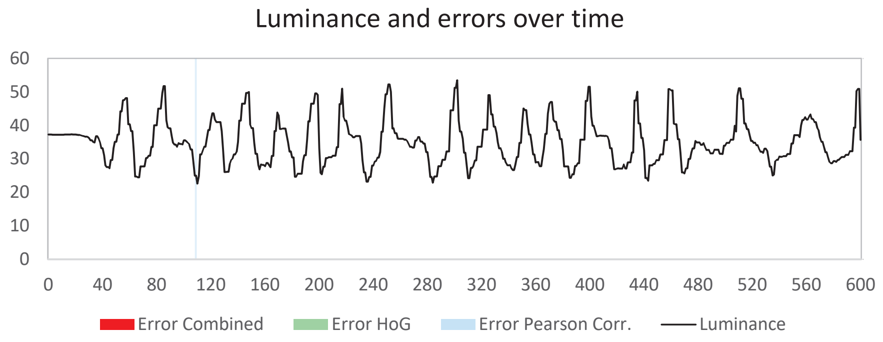
			Errors		*Luminance*
Object:	Liquid-level	**HoG+Pearson:**	0	**Max:**	60.63
State:	L5	**HoG:**	0	**Min:**	29.96
		Pearson:	0	**Average:**	43.39
				SD:	6.31



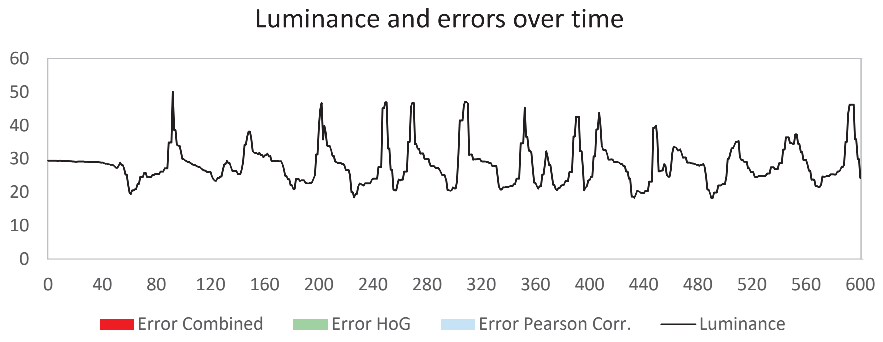
			Errors		*Luminance*
Object:	Liquid-level	**HoG+Pearson:**	0	**Max:**	56.73
State:	L6	**HoG:**	0	**Min:**	27.47
		Pearson:	0	**Average:**	39.90
				SD:	6.11



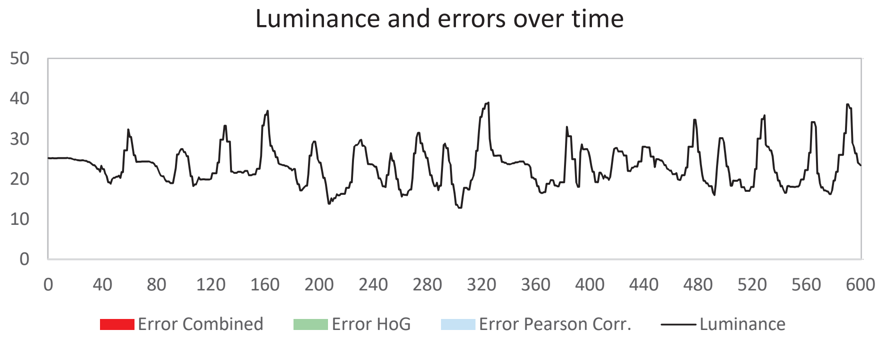
			Errors		***Luminance***
Object:	Liquid-level	**HoG+Pearson:**	0	**Max:**	53.48
State:	L7	**HoG:**	0	**Min:**	22.59
		Pearson:	2	**Average:**	34.97
				SD:	6.62



			Errors		***Luminance***
Object:	Liquid-level	**HoG+Pearson:**	0	**Max:**	50.09
State:	L8	**HoG:**	0	**Min:**	18.28
		Pearson:	0	**Average:**	28.24
				SD:	5.75

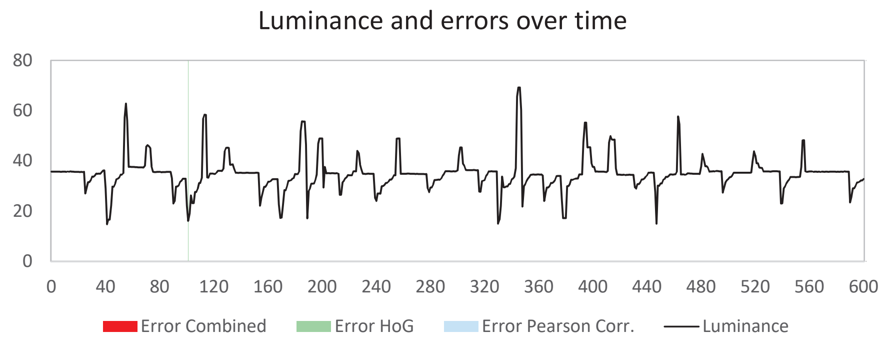


			Errors		***Luminance***
Object:	Liquid-level	**HoG+Pearson:**	0	**Max:**	39.04
State:	L9	**HoG:**	0	**Min:**	12.81
		Pearson:	0	**Average:**	23.14
				SD:	4.80

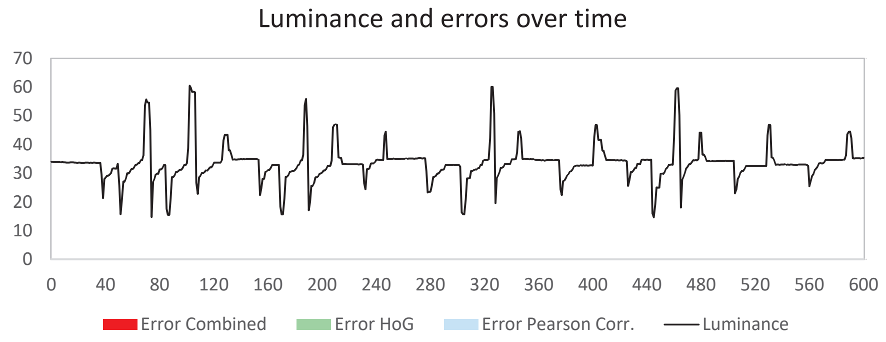


A.9 Door angle

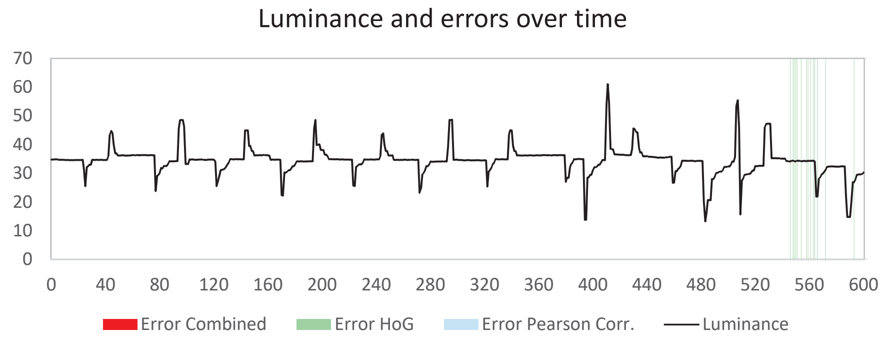
			Errors		*Luminance*
Object:	Door	**HoG+Pearson:**	0	**Max:**	69.24
State:	Angle00	**HoG:**	1	**Min:**	14.75
		Pearson:	0	**Average:**	34.79
				SD:	6.61



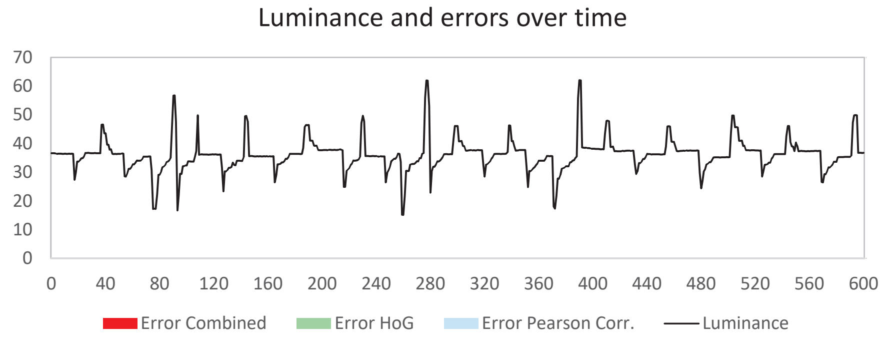
			Errors		***Luminance***
Object:	Door	**HoG+Pearson:**	0	**Max:**	60.48
State:	Angle01	**HoG:**	0	**Min:**	14.59
		Pearson:	0	**Average:**	33.40
				SD:	6.29



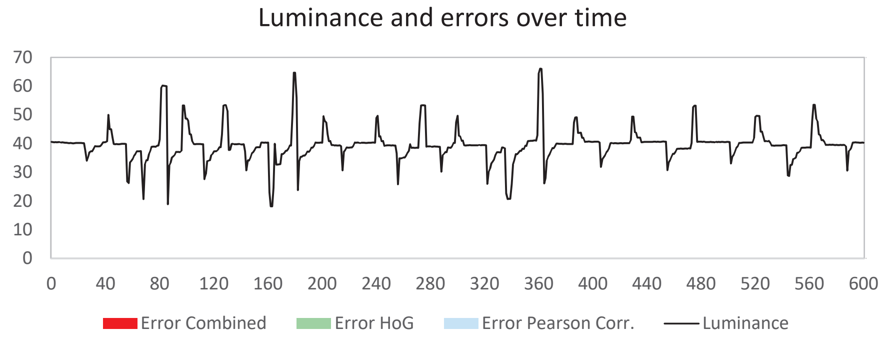
			Errors		***Luminance***
Object:	Door	**HoG+Pearson:**	0	**Max:**	61.05
State:	Angle02	**HoG:**	14	**Min:**	13.21
		Pearson:	0	**Average:**	34.43
				SD:	4.99



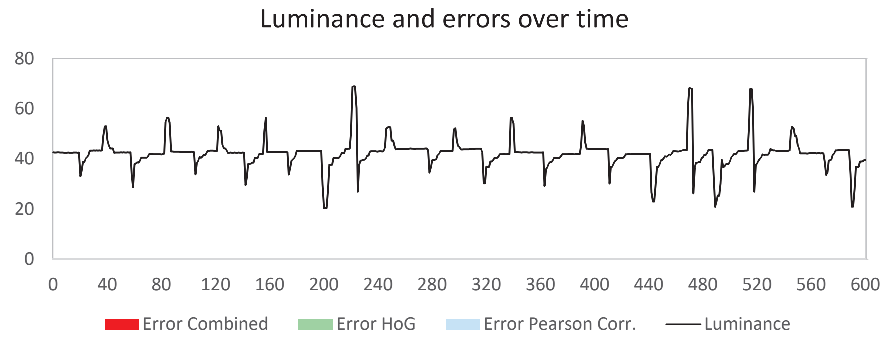
			Errors		*Luminance*
Object:	Door	**HoG+Pearson:**	0	**Max:**	62.07
State:	Angle03	**HoG:**	0	**Min:**	15.15
		Pearson:	0	**Average:**	36.08
				SD:	5.55



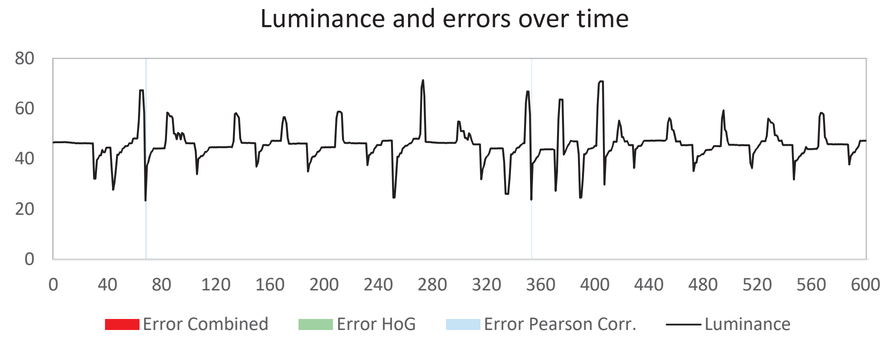
			Errors		*Luminance*
Object:	Door	**HoG+Pearson:**	0	**Max:**	66.07
State:	Angle04	**HoG:**	0	**Min:**	18.10
		Pearson:	0	**Average:**	39.54
				SD:	5.74



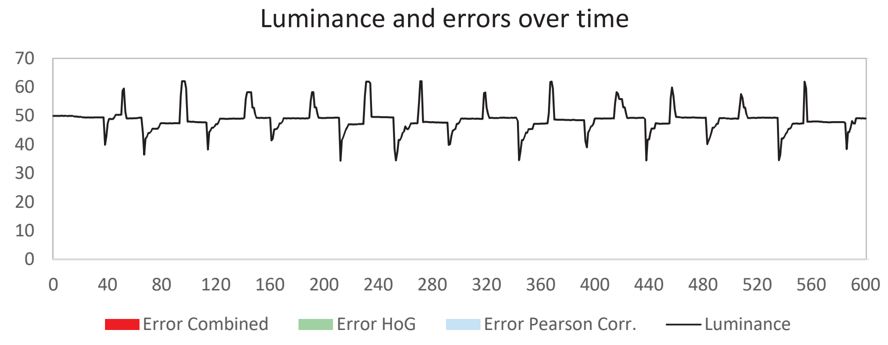
		Errors		*Luminance*	
Object:	Door	**HoG+Pearson:**	0	**Max:**	68.95
State:	Angle05	**HoG:**	0	**Min:**	20.30
		Pearson:	0	**Average:**	42.17
				SD:	5.77



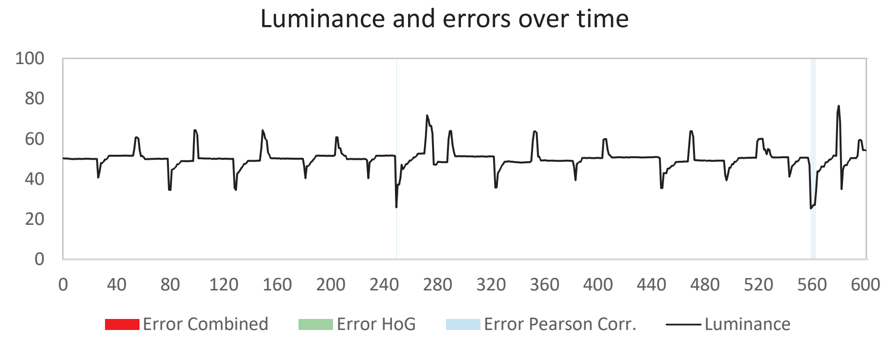
		Errors		*Luminance*	
Object:	Door	**HoG+Pearson:**	0	**Max:**	71.35
State:	Angle06	**HoG:**	0	**Min:**	23.37
		Pearson:	2	**Average:**	45.80
				SD:	6.21



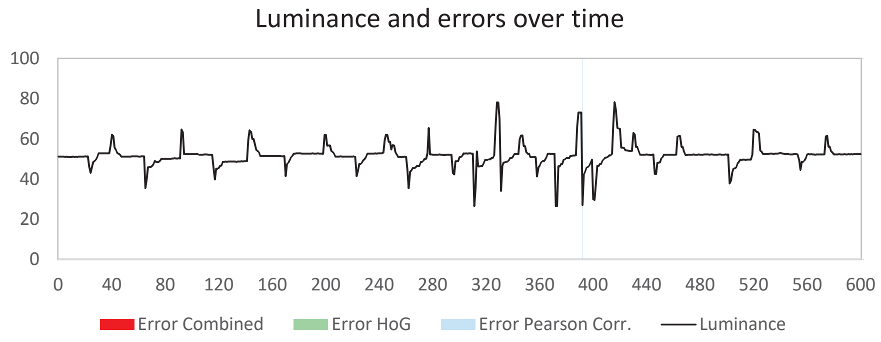
			Errors		*Luminance*
Object:	Door	**HoG+Pearson:**	0	**Max:**	62.09
State:	Angle07	**HoG:**	0	**Min:**	34.32
		Pearson:	0	**Average:**	48.58
				SD:	3.87



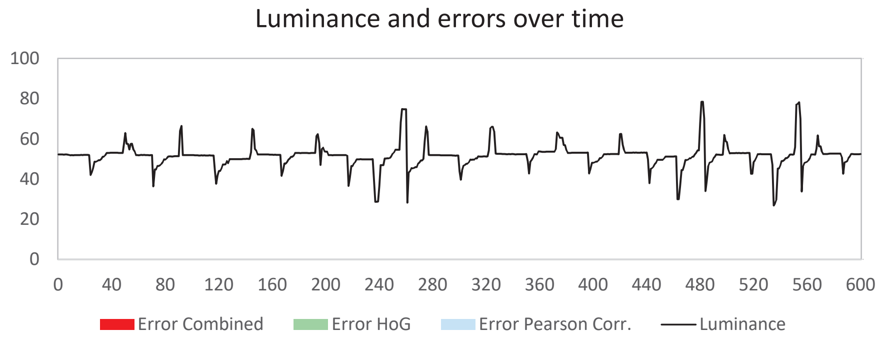
			Errors		*Luminance*
Object:	Door	**HoG+Pearson:**	0	**Max:**	76.41
State:	Angle08	**HoG:**	0	**Min:**	25.31
		Pearson:	5	**Average:**	50.08
				SD:	5.14



		Errors		***Luminance***	
Object:	Door	**HoG+Pearson:**	0	**Max:**	78.22
State:	Angle09	**HoG:**	0	**Min:**	26.53
		Pearson:	1	**Average:**	51.56
				SD:	5.52

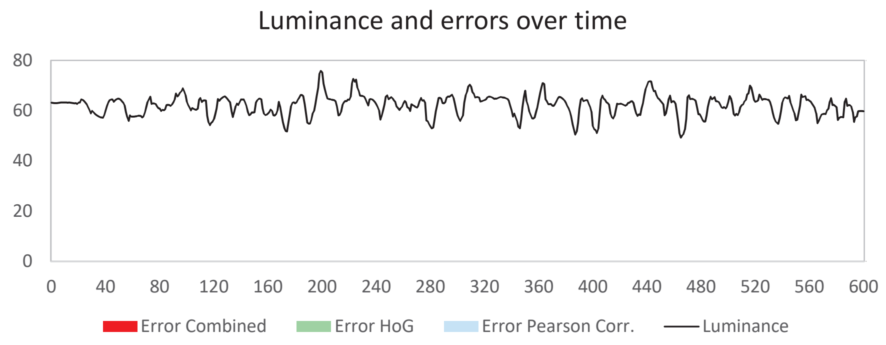


		Errors		***Luminance***	
Object:	Door	**HoG+Pearson:**	0	**Max:**	78.47
State:	Angle10	**HoG:**	0	**Min:**	26.74
		Pearson:	0	**Average:**	51.50
				SD:	5.94

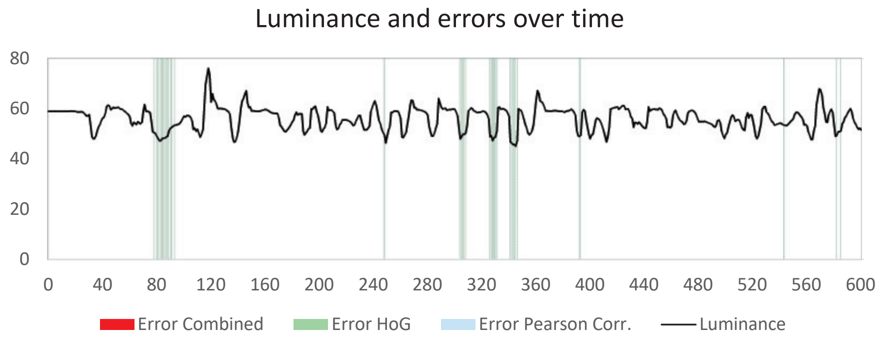


A.10 Colour and presence of a towel

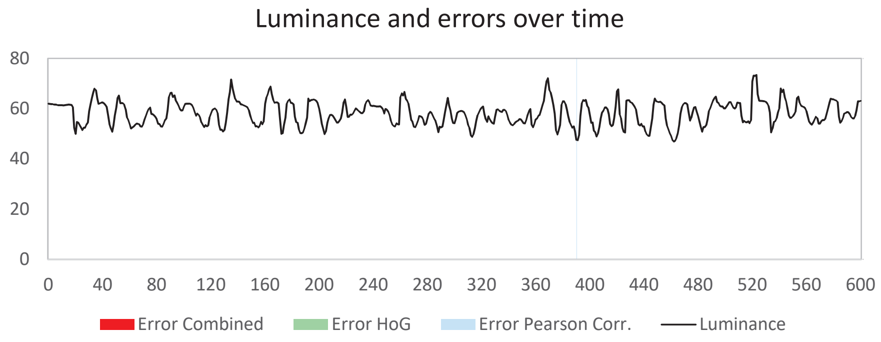
			Errors		***Luminance***
Object:	Towel	**HoG+Pearson:**	0	**Max:**	75.82
State:	None	**HoG:**	0	**Min:**	49.25
		Pearson:	0	**Average:**	62.03
				SD:	3.98



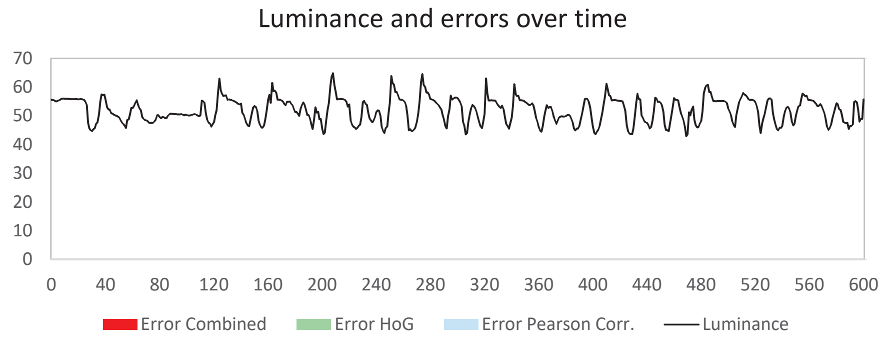
			Errors		*Luminance*
Object:	Towel	**HoG+Pearson:**	0	**Max:**	75.98
State:	Blue	**HoG:**	37	**Min:**	45.04
		Pearson:	0	**Average:**	56.10
				SD:	4.42



			Errors		*Luminance*
Object:	Towel	**HoG+Pearson:**	0	**Max:**	73.41
State:	Green	**HoG:**	0	**Min:**	46.89
		Pearson:	1	**Average:**	58.17
				SD:	4.67



		Errors		*Luminance*	
Object:	Towel	**HoG+Pearson:**	0	**Max:**	64.85
State:	Red	**HoG:**	0	**Min:**	42.88
		Pearson:	0	**Average:**	51.87
				SD:	4.25

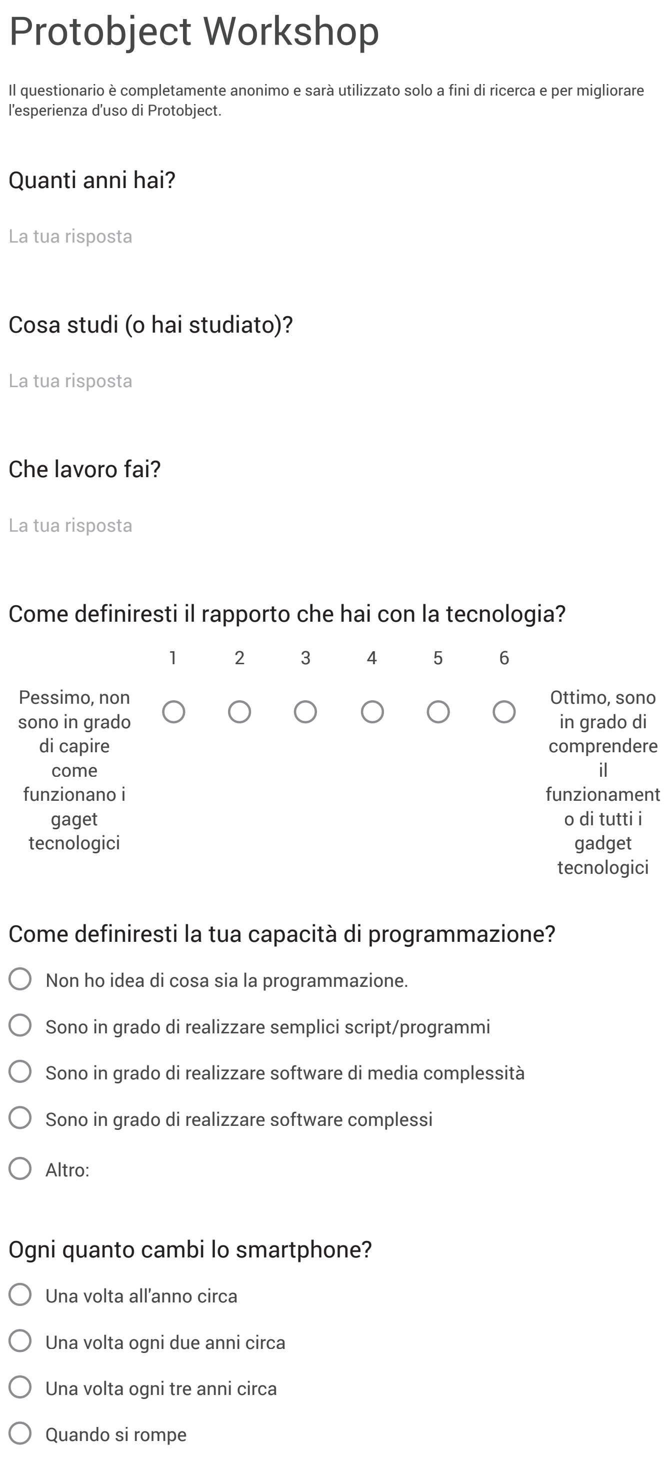


Appendix B

Questionnaire

B.1 The questionnaire

The questionnaire is in Italian.



Protobject Workshop

Il questionario è completamente anonimo e sarà utilizzato solo a fini di ricerca e per migliorare l'esperienza d'uso di Protobject.

Quanti anni hai?

La tua risposta

Cosa studi (o hai studiato)?

La tua risposta

Che lavoro fai?

La tua risposta

Come definiresti il rapporto che hai con la tecnologia?

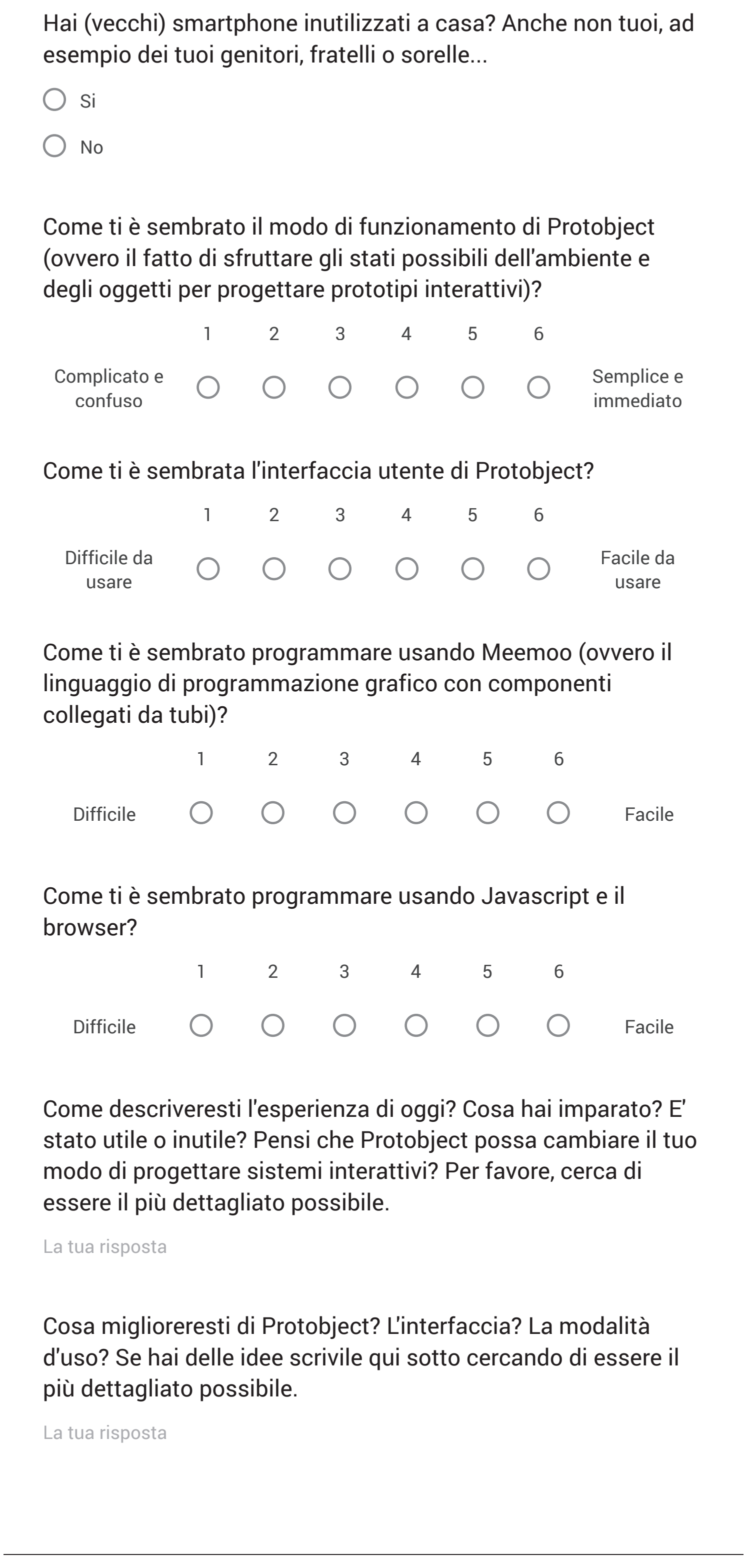
	1	2	3	4	5	6	
Pessimo, non sono in grado di capire come funzionano i gaget tecnologici	○	○	○	○	○	○	Ottimo, sono in grado di comprendere il funzionament o di tutti i gadget tecnologici

Come definiresti la tua capacità di programmazione?

- ○ Non ho idea di cosa sia la programmazione.
- ○ Sono in grado di realizzare semplici script/programmi
- ○ Sono in grado di realizzare software di media complessità
- ○ Sono in grado di realizzare software complessi
- ○ Altro:

Ogni quanto cambi lo smartphone?

- ○ Una volta all'anno circa
- ○ Una volta ogni due anni circa
- ○ Una volta ogni tre anni circa
- ○ Quando si rompe



Hai (vecchi) smartphone inutilizzati a casa? Anche non tuoi, ad esempio dei tuoi genitori, fratelli o sorelle...

○ Si

○ No

Come ti è sembrato il modo di funzionamento di Protobject (ovvero il fatto di sfruttare gli stati possibili dell'ambiente e degli oggetti per progettare prototipi interattivi)?

	1	2	3	4	5	6	
Complicato e confuso	○	○	○	○	○	○	Semplice e immediato

Come ti è sembrata l'interfaccia utente di Protobject?

	1	2	3	4	5	6	
Difficile da usare	○	○	○	○	○	○	Facile da usare

Come ti è sembrato programmare usando Meemoo (ovvero il linguaggio di programmazione grafico con componenti collegati da tubi)?

	1	2	3	4	5	6	
Difficile	○	○	○	○	○	○	Facile

Come ti è sembrato programmare usando Javascript e il browser?

	1	2	3	4	5	6	
Difficile	○	○	○	○	○	○	Facile

Come descriveresti l'esperienza di oggi? Cosa hai imparato? E' stato utile o inutile? Pensi che Protobject possa cambiare il tuo modo di progettare sistemi interattivi? Per favore, cerca di essere il più dettagliato possibile.

La tua risposta

Cosa miglioreresti di Protobject? L'interfaccia? La modalità d'uso? Se hai delle idee scrivile qui sotto cercando di essere il più dettagliato possibile.

La tua risposta

B.2 The responses

User responses are in Italian.

Quanti anni hai?	Cosa studi (o hai studiato)?	Che lavoro fai?	Come definiresti il rapporto che hai con la tecnologia?	Come definiresti la tua capacità di programmazione?	Ogni quanto cambi lo smartphone?	Hai (vecchi) smartphone inutilizzati a casa? Anche non tuoi, ad esempio dei tuoi genitori, fratelli o sorelle...	Come ti è sembrato il modo di funzionamento di Protobject (ovvero il fatto di sfruttare gli stati possibili dell'ambiente e degli oggetti per progettare prototipi interattivi)?	Come ti è sembrata l'interfaccia utente di Protobject?	Come ti è sembrato programmare usando Meemoo (ovvero il linguaggio di programmazione grafico con componenti collegati da tubi)?	Come ti è sembrato programmare e usando Javascript e il browser?	Come descriveresti l'esperienza di oggi? Cosa hai imparato? E' stato utile o inutile? Pensi che Protobject possa cambiare il tuo modo di progettare sistemi interattivi? Per favore, cerca di essere il più dettagliato possibile.	Cosa miglioreresti di Protobject? L'interfaccia? La modalità d'uso? Se hai delle idee scrivile qui sotto cercando di essere il più dettagliato possibile.
55	Architettura	Architetto	4	Non ho idea di cosa sia la programmazione.	Una volta ogni tre anni circa	Si	5	5	6	2	Molto utile. Provero' a usarlo per le mie idee in campo illuminotecnico.	Ve lo dirò quando avrò provato da sola!
21	Web designer	Programmatore	5	Sono in grado di realizzare software di media complessità	Una volta ogni due anni circa	Si	4	5	4	5	Credo che con la giusta applicazione e il giusto lavoro sia possibile, migliorare e creare qualcosa che possa realmente cambiare la vita, si è sulla buona strada.	Il memoo editor, in certi casi non è molto semplice, manca di affordance in punti quasi importanti, un aiuto in più magari per chi non è molto pratico sarebbe ottimo.
27	Ingegneria informatica	Web Developer	5	Sono in grado di realizzare software di media complessità	Una volta ogni due anni circa	Si	5	4	3	6	Ho imparato che si può ottenere un idea (o risultato) da qualunque cosa, come in questo caso reinventare tutto utilizzando i vecchi smartphone	
29	Teoria e tecnologia della comunicazione	Designer	4	Sono in grado di realizzare semplici script/programmi	Una volta ogni tre anni circa	No	4	2	5	2	E stato uno stimolo molto interessante e sto ancora pensando ad una situazione in cui sperimentare con Protobject. Sicuramente credo che adesso sarei meno spaventata se dovessi trovarmi di fronte all'esigenza di creare un prototipo di questo tipo.	Sicuramente migliorerei l'interfaccia perché con una buona user experience potrebbe essere ancora più semplice e intuitivo. La presentazione forse non ha del tutto chiarito il valore/senso di Protobject.
57	architettura	docente	4	Non ho idea di cosa sia la programmazione.	Quando si rompe	Si	4	4	5	3	Mi sono fatto un'idea più precisa sulle possibilità di programmazione	Lo renderei più chiaro
28	Design&Engineering	Student at the moment	6	Sono in grado di realizzare semplici script/programmi	Una volta ogni tre anni circa	Si	6	4	4	4	The concept and idea is charmy and useful. Its chalenging for innivation. I am thinking already how and where can apply it.	Possibility Of using it unlimited
22	Design del prodotto industriale	Studente | Designer	5	Sono in grado di realizzare semplici script/programmi	Una volta ogni tre anni circa	Si	5	3	6	5	Un'esperienza interessante, potrebbe essere interessante utilizzarlo nella parte metaprogettuale per far comprendere un concept o per una prima prototipazione di un sistema interattivo.	L' interfaccia di Protobject potrebbe essere migliorata nella UI, per rendere il prodotto ancora più appetibile ad una range di utenti maggiore e con differenti background.
33	Informatica	Sviluppatore software	5	Sono in grado di realizzare software complessi	Una volta ogni tre anni circa	Si	5	4	5	6	Difficile rispondere, in quanto non ho mai progettato/sviluppato un sistema interattivo. L'esperienza è stata interessante, mi è piaciuta sia l'idea che il buon interfacciamento con javascript che permette ad un programmatore di perdere poco tempo alla logica e di dedicarsi alla progettazione. Faccio solo un po' di fatica ad immaginare degli impieghi pratici davvero utili in fase di effettivo utilizzo e non di progettazione, però probabilmente in ambito industriale potrebbe apportare miglioramenti in tema di sicurezza, efficienza energetica, ecc..	L'interfaccia potrebbe essere leggermente migliorata, ma già un manuale d'uso sarebbe sufficiente.
40	Elettrotecnica e Sociologia	Manager	3	Ne comprendo la logica base	Quando si rompe	Si	4	6	3	4	Da "non addetto ai lavori" ho trovato il momento interessante ed intutitivo	Non possedendo le competenze specifiche non mi addentro in consigli pratici

27	design del prodotto	ux designer	6	Sono in grado di realizzare semplici script/programmi	Quando si rompe	Si	4	1	4	4	Molto interessante però ho trovato macchinosa la spiegazione generale. Consiglierei per le prossime esperienze di portare degli esempi più puntuali di utilizzo all'interno del processo di design di un sistema interattivo. Esempio: voglio realizzare un sensore per controllare un neonato, provo prima con protobject le interazioni, poi realizzo un prototipo con Arduino (esempio). Deve essere più chiaro che protobjet non sostituisce ma aiuta la ricerca e il processo.	Le icone erano poco chiare, troppe finestre.
28	product design	product designer	4	Non ho idea di cosa sia la programmazione.	Una volta ogni tre anni circa	No	4	3	3	2	Workshop è stato molto interessante. Penso che Protobjec potrebbe avere una possibilità diessere usato per miei progetti di arredamento.	Interfaccia non è tanto intuitiva alle persone non abituate alle logiche di programmazione.
39	Perito elettronico - Facoltà di Sociologia	project manajer makerspace	6	Sono in grado di realizzare semplici script/programmi	Una volta ogni due anni circa	Si	5	5	5	4	Progetto interessante e di facile utilizzo, quindi approcciabile anche da persone non esperte. Proveremo ad utilizzarlo per creare dei progetti di interaction design e testarne a nsotra volta la risposta da parte degli utenti finali.	Qualche messaggio di conferma dell'avvenuta registrazione degli stati e dell'inserimento delle etichette. La parte di programmazione dovrebbe essere supportata da tooltips per aiutare i non programmatori.
24	Ingegneria Informatica	web developer	6	Sono in grado di realizzare semplici script/programmi	Una volta ogni tre anni circa	No	6	6	5	6		
25	Design		5	Sono in grado di realizzare semplici script/programmi	Quando si rompe	Si	5	5	5	4	Sicuramente sarà possibile utilizzare le funzionalità di Protobject al fine di prototipare un sistema interattivo di base. Probabilmente provando a conoscere meglio le funzionalità e le possibilità che il programma offre, sarà possibile creare sistemi più complessi. Basandomi sugli esempi visti in aula, sembra che il sistema possa venire in aiuto solo per notificare qualche azione "non normale" secondo la logica programmata, quindi non risolve il problema né lo anticipa. Perché non pensare di "prevenire" il problema notificando su azioni che potrebbero compiersi in determinate situazioni x? (Ad esempio il sistema rileva che fuori piove. Prima che esci di casa, potrebbe arrivarti una notifica suggerendoti di prendere l'ombrello che è ANCORA sulla sedia).	Per chi ha conoscenza nulla o basilare sulla programmazione, il sistema può risultare complicato a prima vista, dato che alcune funzionalità sono "nascoste" o raggiungibili attraverso short code. Il fatto di dover "addestrare" il sistema ad ogni spostamento involontario o volontario della telecamera, rende l'operazione più lunga: non sarebbe possibile aumentare la sensibilità dell'area presa in considerazione, ed evitare di registrare ogni volta l'area da considerare? Ad esempio mantenendo il focus su quell'oggetto. è vero che non è richiesto l'acquisto di alcun hardware aggiuntivo ma basta utilizzare un vecchio telefono...ma fino a quale versione di Android deve essere aggiornato il telefono vecchio? Inoltre bisogna ricordarsi di tenere la batteria del device che ospita la telecamera ad un buon livello di carica (o sempre sotto carica), ed avere una connessione internet sempre attiva.
35	Architettura + Product Design	Architetto	6	Sono in grado di realizzare semplici script/programmi	Quando si rompe	No	6	6	6	2	è stato molto utile, ed è molto interessante il software/ app	manca fare un app per iphone

28	Design	Designer consulente	3	sono in grado di capire i processi ma non di programmare	Una volta ogni due anni circa	Si	5	4	4	1	L'esperienza e' stata sicuramente molto interessante per una persona creativa con voglia di conoscere e esprimere le sue idee, senza sapere come funzionano alcune parti tecniche di un oggetto. Ho capito la potenzialita' della tecnologia e quanto puo' cambiare il mestiere del designer grazie a queste nuove interfacce. Penso che in un secondo momento un approccio come Protobject possa portare innovazione vera creando un ponte di comunicazione tra designer e programmatori, un cambio di paradigma importantissimo nel nuovo scenario della progettazione. Mi piacerebbe poter approffondire l'argomento e vedere ulteriori sviluppi della piattaforma, anche in contesti di sperimentazione progettuale spontanea delle potenzialita' del programma.	Mi piacerebbe poter interagire con Protobject senza saper programmare, redendo il passaggio tra la lettura del mondo reale e il computer in maniera piu' intuitiva e diretta. A lunga andata penso una piattaforma di questo tipo potrebbe diventare un punto di passaggio di know how e conoscenza tra programmatori e designer, condividendo saperi e vocabolari in comune. Come un dizionario di saperi e passaggio di conscenze mirato alla co-progettazione.
29	Master in product design	Studente	5	Sono in grado di realizzare semplici script/programmi	Quando si rompe	No	6	5	5	5	È stato molto utile e ho imparato programmazione semplice con un software e un smartphone e mi ha dato una esperienza nuova.	
46		Liberoprofessionista	6		Una volta all'anno circa	Si	6	6	3	3	E stato molto utile	

Bibliography

[ABC$^+$09] Carmelo Ardito, Paolo Buono, Maria Francesca Costabile, Rosa Lanzilotti, and Antonio Piccinno. A tool for wizard of oz studies of multimodal mobile systems. In *Proceedings of the 2nd Conference on Human System Interactions, HSI*, pages 341–344, 2009.

[ADB$^+$99] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *International Symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer, 1999.

[AK09] Hideki Aoyama and Yu Kimishima. Mixed reality system for evaluating designability and operability of information appliances. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 3(3):157–164, 2009.

[ALF06] Robert Adelmann, Marc Langheinrich, and Christian Floerkemeier. Toolkit for bar code recognition and resolving on camera phones-jump starting the internet of things. *GI Jahrestagung (2)*, 94:366–373, 2006.

[AM04] Gregory D Abowd and Elizabeth D Mynatt. Designing for the human experience in smart environments. *Smart environments: technologies, protocols, and applications*, pages 151–174, 2004.

[AR06] Douglas G Altman and Patrick Royston. The cost of dichotomising continuous variables. *Bmj*, 332(7549):1080, 2006.

[Ban09] Massimo Banzi. O'reilly media. *Inc. Getting Started with Arduino*, 10:978–1, 2009.

[BCCP09] Monica Bordegoni, Umberto Cugini, Giandomenico Caruso, and Samuele Polistina. Mixed prototyping for product assessment: a reference framework. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 3(3):177–187, 2009.

[BCF06] Monica Bordegoni, Giorgio Colombo, and Luca Formentini. Haptic technologies for the conceptual and validation phases of product design. *Computers & Graphics*, 30(3):377–390, 2006.

[BDME$^+$11] Thomas Binder, Giorgio De Michelis, Pelle Ehn, Giulio Jacucci, Per Linde, and Ina Wagner. *Design things*. MIT Press, 2011.

[Bel16] Alessio Bellino. Protobject: a sensing tool for the rapid prototyping of ubicomp systems. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 257–260. ACM, 2016.

[BLM03] Michel Beaudouin-Lafon and Wendy Mackay. Prototyping tools and techniques. *Human Computer Interaction-Development Process*, pages 122–142, 2003.

[BTL$^+$13] Tegawendé F Bissyandé, Ferdian Thung, David Lo, Lingxiao Jiang, and Laurent Réveillere. Popularity, interoperability, and impact of programming languages in 100,000 open source projects. In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, pages 303–312. IEEE, 2013.

[CC04] SH Choi and AMM Chan. A virtual prototyping system for rapid product development. *Computer-Aided Design*, 36(5):401–412, 2004.

[Che99] Elaine Chen. Six degree-of-freedom haptic system for desktop virtual prototyping applications. In *Proceedings of the First International Workshop on Virtual Reality and Prototyping*, pages 97–106, 1999.

[CMP$^+$11] Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Gabriele Sprega, Donato Barbagallo, and Chiara Francalanci. Dashmash: a mashup environment for end user development. In *International Conference on Web Engineering*, pages 152–166. Springer, 2011.

[CS15] Federico Cabitza and Carla Simone. Building socially embedded technologies: Implications about design. In *Designing socially embedded technologies in the real-world*, pages 217–270. Springer, 2015.

[CTG99] CK Chua, SH Teh, and RKL Gay. Rapid prototyping versus virtual prototyping in product design and manufacturing. *The International Journal of Advanced Manufacturing Technology*, 15(8):597–603, 1999.

[CY09] Giorgos Cheliotis and Jude Yew. An analysis of the social structure of remix culture. In *Proceedings of the fourth international conference on Communities and technologies*, pages 165–174. ACM, 2009.

[DAK03] Christan Duriez, Claude Andriot, and Abderrahmane Kheddar. Interactive haptics for virtual prototyping of deformable objects: snap-in tasks case. *feedback*, 20:28, 2003.

[DAM+11] Phillip S Dunston, Laura L Arns, James D Mcglothlin, Gregory C Lasker, and Adam G Kushner. An immersive virtual reality mock-up for design review of hospital patient rooms. In *Collaborative design in virtual environments*, pages 167–176. Springer, 2011.

[Dan05] Peter Danholt. Prototypes as performative. In *Proceedings of the 4th decennial conference on Critical computing: between sense and sensibility*, pages 1–8. ACM, 2005.

[DAS01] Anind K Dey, Gregory D Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, 16(2):97–166, 2001.

[DE] Carl Doersch and Alexei Efros. Improving the hog descriptor.

[Dew13] John Dewey. *Logic-The theory of inquiry*. Read Books Ltd, 2013.

[Dou04a] Paul Dourish. What we talk about when we talk about context. *Personal and ubiquitous computing*, 8(1):19–30, 2004.

[Dou04b] Paul Dourish. *Where the action is: the foundations of embodied interaction*. MIT press, 2004.

[DT05] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[Fet11] Ian Fette. The websocket protocol. 2011.

[FO02] Jerry Alan Fails and Jr Dan Olsen. Light widgets: interacting in everyday spaces. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 63–69. ACM, 2002.

[FO03] Jerry Fails and Dan Olsen. A design tool for camera-based interaction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 449–456. ACM, 2003.

[GF01] Saul Greenberg and Chester Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 209–218. ACM, 2001.

[GL85] John D Gould and Clayton Lewis. Designing for usability: key principles and what designers think. *Communications of the ACM*, 28(3):300–311, 1985.

[Gos12] A Ardeshir Goshtasby. Similarity and dissimilarity measures. In *Image registration*, pages 7–66. Springer, 2012.

[GPT15] Thomas Grill, Ondrej Polacek, and Manfred Tscheligi. Conwiz: The contextual wizard of oz. *Journal of Ambient Intelligence and Smart Environments*, 7(6):719–744, 2015.

[GPZ05] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and computer applications*, 28(1):1–18, 2005.

[HKB$^+$06] Björn Hartmann, Scott R Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 299–308. ACM, 2006.

[HVSS12] Steve Hodges, Nicolas Villar, James Scott, and Albrecht Schmidt. A new era for ubicomp development. *IEEE Pervasive Computing*, 11(1):5–9, 2012.

[HYLDG03] Chen-Je Huang, E Yi-Luen Do, and D Gross. Mousehaus table: A physical interface for urban design. In *16th Annual ACM Symposium on User Interface Software and Technology (UIST)*, pages 41–42, 2003.

[JHM04] Wesley M Johnston, JR Hanna, and Richard J Millar. Advances in dataflow programming languages. *ACM Computing Surveys (CSUR)*, 36(1):1–34, 2004.

[JW07] Giulio Jacucci and Ina Wagner. Performative roles of materiality for collective creativity. In *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, pages 73–82. ACM, 2007.

[KB07] Martin Kaltenbrunner and Ross Bencina. reactivision: a computer-vision framework for table-based tangible interaction. In *Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 69–74. ACM, 2007.

[KHC$^+$13] Yoshihiro Kawahara, Steve Hodges, Benjamin S Cook, Cheng Zhang, and Gregory D Abowd. Instant inkjet circuits: lab-based inkjet printing to support rapid prototyping of ubicomp devices. In *Proceedings of the*

2013 ACM international joint conference on Pervasive and ubiquitous computing, pages 363–372. ACM, 2013.

[Kim90] S. Kim. Interdisciplinary cooperation. In *In B. Laurel (ed.) The Art of Human–Computer Interface Design*. Addison-Wesley, 1990.

[LBHH15] Gierad Laput, Eric Brockmeyer, Scott E Hudson, and Chris Harrison. Acoustruments: Passive, acoustically-driven, interactive controls for handheld devices. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2161–2170. ACM, 2015.

[Les08] Lawrence Lessig. *Remix: Making art and commerce thrive in the hybrid economy*. Penguin, 2008.

[LLW$^+$15] Gierad Laput, Walter S Lasecki, Jason Wiese, Robert Xiao, Jeffrey P Bigham, and Chris Harrison. Zensors: Adaptive, rapidly deployable, human-intelligent sensor feeds. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1935–1944. ACM, 2015.

[Low99] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[LS66] Claude Lvi-Strauss. *The savage mind*. University of Chicago Press, 1966.

[LW98] Rüdiger Lainer and Ina Wagner. Connecting qualities of social use with spatial qualities. In *International Workshop on Cooperative Buildings*, pages 191–203. Springer, 1998.

[MAWI07] Dan Maynes-Aminzade, Terry Winograd, and Takeo Igarashi. Eyepatch: prototyping camera-based interaction through examples. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 33–42. ACM, 2007.

[MO14] Can Mihci and Nesrin Ozdener. Programming education with a blocks-based visual language for mobile application development. *International Association for Development of the Information Society*, 2014.

[MP62] Maurice Merleau-Ponty. Phenomenology of perception, trans. colin smith, 1962.

[MP14] Mary Lou Maher and Pearl Pu. *Issues and applications of case-based reasoning to design*. Psychology Press, 2014.

[MPW08] Valérie Maquil, Thomas Psik, and Ina Wagner. The colortable: a design story. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 97–104. ACM, 2008.

[MVSC05] Ji-Ye Mao, Karel Vredenburg, Paul W Smith, and Tom Carey. The state of user-centered design practice. *Communications of the ACM*, 48(3):105–109, 2005.

[O$^+$12] Forrest Oliphant et al. Meemoo: Hackable web app framework. 2012.

[OST13] Makoto Ono, Buntarou Shizuki, and Jiro Tanaka. Touch & activate: adding interactivity to existing objects using active acoustic sensing. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 31–40. ACM, 2013.

[OWM15] Lora Oehlberg, Wesley Willett, and Wendy E Mackay. Patterns of physical design remixing in online maker communities. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 639–648. ACM, 2015.

[P$^+$96] Miller Puckette et al. Pure data: another integrated computer music environment. *Proceedings of the second intercollege computer music concerts*, pages 37–41, 1996.

[Puc91] Miller Puckette. Combining event and signal processing in the max graphical programming environment. *Computer music journal*, 15(3):68–77, 1991.

[QBVG08] Till Quack, Herbert Bay, and Luc Van Gool. Object recognition for the internet of things. In *The Internet of Things*, pages 230–246. Springer, 2008.

[RG04] Michael Rohs and Beat Gfeller. *Using camera-equipped mobile phones for interacting with real-world objects.* na, 2004.

[ROPT05] Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen. Contextphone: A prototyping platform for context-aware mobile applications. *IEEE pervasive computing*, 4(2):51–59, 2005.

[RSP15] Yvonne Rogers, Helen Sharp, and Jenny Preece. *Interaction Design: Beyond Human - Computer Interaction.* Wiley Publishing, 4rd edition, 2015.

[Sat01] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001.

[SAW94] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90. IEEE, 1994.

[SC93] Daniel Salber and Joëlle Coutaz. Applying the wizard of oz technique to the study of multimodal systems. In *International Conference on Human-Computer Interaction*, pages 219–230. Springer, 1993.

[SCG$^+$11] Anna Spagnolli, Nicola Corradi, Luciano Gamberini, Eve Hoggan, Giulio Jacucci, Cecilia Katzeff, Loove Broms, and Li Jönsson. Eco-feedback on the go: Motivating energy awareness. *Computer*, 44(5):38–45, 2011.

[Sch83] Donald A Schön. *The reflective practitioner: How professionals think in action*, volume 5126. Basic books, 1983.

[SCH13] Valkyrie Savage, Colin Chang, and Björn Hartmann. Sauron: embedded single-camera sensing of printed physical user interfaces. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 447–456. ACM, 2013.

[SDA99] Daniel Salber, Anind K Dey, and Gregory D Abowd. The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 434–441. ACM, 1999.

[SHH$^+$15] Valkyrie Savage, Andrew Head, Björn Hartmann, Dan B Goldman, Gautham Mysore, and Wilmot Li. Lamello: Passive acoustic sensing for tangible input components. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1277–1280. ACM, 2015.

[SPD$^+$10] Adam C Siegel, Scott T Phillips, Michael D Dickey, Nanshu Lu, Zhigang Suo, and George M Whitesides. Foldable printed circuit boards on paper substrates. *Advanced Functional Materials*, 20(1):28–35, 2010.

[STG03] Reinhard Sefelin, Manfred Tscheligi, and Verena Giller. Paper prototyping-what is it good for?: a comparison of paper-and computer-based low-fidelity prototyping. In *CHI'03 extended abstracts on Human factors in computing systems*, pages 778–779. ACM, 2003.

[TGF09] Anthony Tang, Saul Greenberg, and Sidney Fels. Exploring video streams using slit-tear visualizations. In *CHI'09 Extended Abstracts on Human Factors in Computing Systems*, pages 3509–3510. ACM, 2009.

[Tha01] John Thackara. The design challenge of pervasive computing. *interactions*, 8(3):46–52, 2001.

[TVH02] Stefan Thomke and Eric Von Hippel. Innovators. *Harvard business review*, 80(4):74–81, 2002.

[UPYHB+16] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L. Littman. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 3227–3231, New York, NY, USA, 2016. ACM.

[VSH+12] Nicolas Villar, James Scott, Steve Hodges, Kerry Hammil, and Colin Miller. . net gadgeteer: a platform for custom devices. In *International Conference on Pervasive Computing*, pages 216–233. Springer, 2012.

[Wal] Andy Walton. Life expectancy of a smartphone. http://smallbusiness.chron.com/life-expectancy-smartphone-62979.html. Accessed: 2016-10-07.

[Wan02] G Gary Wang. Definition and review of virtual prototyping. *Journal of Computing and Information Science in engineering*, 2(3):232–236, 2002.

[WBHP12] Karl Willis, Eric Brockmeyer, Scott Hudson, and Ivan Poupyrev. Printed optics: 3d printing of embedded optical elements for interactive devices. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 589–598. ACM, 2012.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[Win97] Terry Winograd. The design of interaction. In *Beyond calculation*, pages 149–161. Springer, 1997.

[WM07] Ron Wakkary and Leah Maestri. The resourcefulness of everyday design. In *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, pages 163–172. ACM, 2007.

[ZB06] Tobias Zimmer and Michael Beigl. Awareoffice: Integrating modular context-aware applications. In *26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'06)*, pages 59–59. IEEE, 2006.

[ZWPG03] F Zorriassatine, C Wykes, R Parkin, and N Gindy. A survey of virtual prototyping techniques for mechanical product development. *Proceedings of the institution of mechanical engineers, Part B: Journal of engineering manufacture*, 217(4):513–530, 2003.