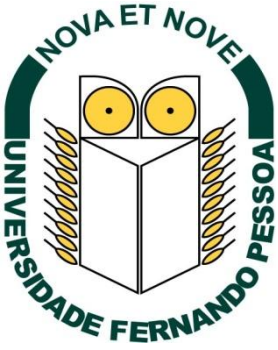
LoCoBoard: Quadro Interactivo de Baixo Custo Recorrendo a Algoritmos de Visão por Computador

Christophe Pinto de Almeida Soares

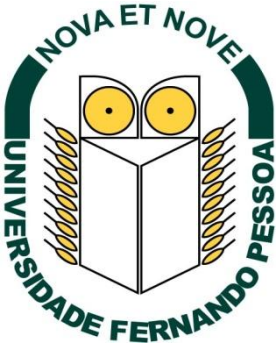


Universidade Fernando Pessoa
Faculdade de Ciência e Tecnologia
Praça 9 de Abril, 349, 4249-004 Porto, Portugal

Outubro de 2009

LoCoBoard: Quadro Interactivo de Baixo Custo Recorrendo a Algoritmos de Visão por Computador

Christophe Pinto de Almeida Soares



Universidade Fernando Pessoa
Faculdade de Ciência e Tecnologia
Praça 9 de Abril, 349, 4249-004 Porto, Portugal

Outubro de 2009

LoCoBoard: Quadro Interactivo de Baixo Custo recorrendo a Algoritmos de Visão por Computador

Christophe Pinto de Almeida Soares

Licenciado em Engenharia Informática pela Universidade Fernando Pessoa

Dissertação apresentada à Universidade Fernando Pessoa como parte dos requisitos para obtenção do grau de Mestre em Computação Móvel, orientada pelo Professor Doutor Rui Silva Moreira e co-orientada pelo Professor Doutor José Torres.

Universidade Fernando Pessoa
Faculdade de Ciência e Tecnologia
Praça 9 de Abril, 349, 4249-004 Porto, Portugal

Outubro de 2009

Resumo

Na actual era digital, a adopção de interfaces naturais entre o homem e a máquina, torna-se cada vez mais pertinente. Na educação, em particular, a utilização de ferramentas interactivas para melhorar as práticas pedagógicas, auxiliar a compreensão de conceitos complexos e permitir o trabalho colaborativo, constitui uma vantagem inequívoca. Em particular os quadros interactivos (QI) são uma ferramenta muito útil cujo uso está cada vez mais disseminado por vários níveis de ensino. Neste aspecto, a panóplia de soluções comerciais disponíveis é vasta, mas o custo associado é geralmente elevado, para equipar de forma generalizada os estabelecimentos de ensino, principalmente nos países de economia mais débil.

Este trabalho propõe neste contexto, um sistema de quadro interactivo de código aberto, com requisitos de hardware muito reduzidos (i.e., um computador com uma câmara de vídeo WEB, um videoprojector e um dispositivo emissor de infravermelhos com o formato de caneta), o que poderá contribuir para a massificação do seu uso.

Neste trabalho apresenta-se a estrutura física e lógica de um protótipo de Quadro Interactivo de baixo custo designado por LoCoBoard (*Low Cost Interactive Board*). O desenvolvimento deste sistema baseou-se na implementação de vários módulos, desde a captura de imagens, pré-processamento das frames, detecção e seguimento de pontos de interesse (PI) e distribuição de coordenadas através do protocolo *Tangible UI Object Protocol* (TUIO). Apresenta-se ainda a avaliação de desempenho dos diferentes algoritmos de detecção de pontos de interesse uma vez que estes fazem parte de um dos módulos centrais da arquitectura do sistema. Finalmente, estrutura-se ainda uma ferramenta de análise que nos permite comparar o protótipo LoCoBoard com sistemas afins.

Abstract

In the current digital age, the integration of natural interfaces between humans and machines is becoming important than ever. This is particularly relevant in education as the utilisation of interactive tools can provide clear advantages by improving teaching practices, facilitating the comprehension of complex concepts and permitting collaborative work. More particularly, interactive whiteboards are very useful tools that are being increasingly used in various levels of education. There is a vast catalogue of business solutions available for interactive whiteboards. However, these solutions are often costly which can deter their full implementation in all types of schools, especially in countries with more fragile economies.

In this context, this dissertation proposes the adoption of an open source interactive whiteboard, which have low-cost hardware requirements such as a webcam-equipped computer, a video projector and an infrared device. Such solution will easy the access to interactive whiteboards and consequently increase its widespread.

This dissertation discusses the physical and logical structure of a low cost interactive framework prototype called LoCoBoard (Low Cost Interactive Board). The development of this system was based on the implementation of several modules including the capture of images, pre-processing of frames, detection and tracking of points of interest (POI) and coordinated distribution through the Tangible UI Object Protocol (TUIO). Additionally, the performance of different algorithms, used to detect points of interest, are evaluated and presented, as they are part of one of the core modules of the system. Finally, an analysis framework is used for comparing the LoCoBoard prototype with related systems.

Résumé

Dans l'ère numérique actuelle, l'adoption d'interfaces naturelles entre l'homme et la machine prend une place de plus en plus importante. C'est le cas dans l'éducation où l'utilisation d'outils interactifs s'avère d'une très grande utilité que ce soit pour améliorer les pratiques pédagogiques, aider à la compréhension de concepts complexes ou faciliter le travail collaboratif. Les tableaux interactifs (TI) par exemple sont des outils de plus en plus répandu dans le milieu de l'éducation et ce à tous les niveaux (e.g., Ecole, Collège, Universités). Il existe d'ailleurs à ce jour un grand éventail de solutions disponibles, mais leurs coûts généralement élevés ne permettent pas d'équiper toutes les institutions et plus particulièrement dans les pays pauvres.

Cette thèse présente donc dans ce contexte un prototype de tableau interactif à bas prix, basé sur une solution *open-source* et composé d'un hardware simple (un ordinateur, une caméra vidéo, un vidéoprojecteur et un émetteur infrarouge IR) tout ceci dans le but de le rendre facilement réutilisable et surtout abordable à tous. Il s'intitule LoCoBoard (*Low Cost Interactive Whiteboard*).

Toute la structure logique et physique de ce prototype est expliquée dans cette thèse. La conception de ce système repose sur le développement de divers éléments comme la capture d'image, le prétraitement d'images, la détection et le suivi (cf. tracking) de points d'intérêt (PI) et la distribution de coordonnées à travers le protocole Tangible UI Object Protocol (TUIO). Pour mettre en évidence la performance des différents algorithmes développés pour détecter les points considérés d'intérêt capturés par la caméra vidéo, une étude comparative a été effectuée et est présenté dans cette thèse. Ces algorithmes constituent l'un des éléments essentiels de l'architecture du système LoCoBoard. Pour finir, les caractéristiques du système LoCoBoard sont détaillées par le biais d'une analyse comparative avec celles d'une série de systèmes identiques.

Aos Meus Pais

Agradecimentos

O mestrado consiste numa etapa de desenvolvimento a nível pessoal, cultural e profissional. Nesta fase, sinto ter atingido mais uma das metas estabelecidas precedentemente, mas no entanto, devido ao espírito de ambição, perdurarão sempre muitas outras a alcançar.

A possibilidade de finalizar e concretizar este ciclo, deve-se não só ao aluno, mas também a todas as pessoas que o acompanharam e contribuíram de forma positiva e benéfica ao longo desta caminhada. É a estas que quero agradecer.

Aos Professores Doutor José Torres, co-orientador da dissertação, e Doutor Pedro Sobral quero agradecer, pela partilha de conhecimentos, por todo o tempo que dispensaram e por toda a atenção prestada na elaboração da aplicação que deu origem a este trabalho, permitindo a sua finalização com sucesso.

Ao Professor Doutor Rui Silva Moreira, orientador da dissertação, pelas mesmas razões que foram acima mencionadas, bem como, também pela ajuda que foi prestada na elaboração desta tese. Também quero agradecer-lhe em particular pelo acompanhamento que me prestou e por continuar a estimular o meu interesse pelo conhecimento e pela vida académica.

Aos meus pais que sempre me souberam apoiar no meu percurso académico, sem eles não seria o que sou hoje. Agradeço-lhes todos os valores, que me souberam desde sempre transmitir e continuarei a ser fiel aos mesmos.

O meu sincero agradecimento a todas as pessoas que não referi anteriormente, mas que contribuíram de alguma forma, para a concretização desta dissertação.

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

API	Acrónimo para ***A**pplication **P**rogramming **I**nterface*, este é a uma definição de métodos implementados que podem ser reutilizados por outros programadores através da evocação duma função.
ASCII	Acrónimo para ***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange*, este é uma codificação de caracteres de sete bits baseada no alfabeto inglês.
BLOB(s)	Acrónimo para ***B**inary **L**arge **Ob**ject*, este caracteriza conjuntos de píxeis com características comuns que se conseguem isolar numa imagem.
C++	Linguagem de Programação Orientada a Objectos.
CPU	Este termo serve para designar o processador.
Cross-Platform	Este termo serve para designar uma aplicação que é suportada por vários sistemas operativos.
Intel® IPP	***I**ntel **I**ntegrated **P**erformance **P**rimitives* são bibliotecas da Intel que permitem optimizar o processamento de aplicações.
NUI	***N**atural **U**ser **I**nterface*, é uma comunidade que investiga tecnologias ligadas as possíveis novas

	interfaces de comunicação entre o Homem e a Maquina.
OS / SO	***O**perative **S**ystem* / **S**istema **O**perativo.
OSC	***O**pen **S**ound **C**ontrol* é um padrão de mensagens estabelecido para comunicações entre computadores e dispositivos multimédia.
OpenCV	***Open** Source **C**omputer **V**ision* é uma biblioteca de visão por computador criada pela Intel.
PARC	*Xerox **P**alo **A**lto **R**esearch **C**enter* é uma empresa americana conhecida por seus trabalhos de investigação nas áreas de tecnologias de informação.
PI(s)	**P**onto(s) de **I**nteresse(s), este termo serve para qualificar a presença de um ponto considerado como uma interacção valida pela câmara.
TUIO	***T**angible **UI O**bject Protocol* é um protocolo usado na transmissão de coordenadas de Pontos de Interesse.
Windows, Mac OS X, Linux	Estes termos servem para qualificar diferentes sistemas operativos que constituem propriedade de diferentes empresas.

Capítulo 1: Introdução

Ao longo dos últimos anos, a evolução das tecnologias de informação e comunicação tem sido notável, no entanto, a sua utilização nas diferentes actividades tem tido diferentes ritmos. Na educação, por exemplo, o objecto central da sala de aula é ainda o quadro que, geralmente, não faz parte da era digital. Isto deve-se principalmente aos elevados custos que acarretam os quadros interactivos digitais existentes no mercado que impedem a sua adopção generalizada em estabelecimentos de ensino com recursos limitados. Contudo, os quadros interactivos têm uma série de qualidades inexistentes nos quadros comuns, promovendo uma maior interacção e discussão dos conceitos em sala de aula, por via da utilização de recursos multi-mediáticos.

Hoje em dia existem componentes de *hardware* de baixo custo que permitem criar um sistema de quadro interactivo digital acessível e com funcionalidade comparável à das ofertas comerciais.

1.1 Objectivos

O objectivo deste trabalho consiste na criação de uma ferramenta de software, que implemente um quadro interactivo, com recurso a um computador normal, equipado com uma câmara de vídeo WEB comum, ou seja, um quadro interactivo de baixo custo que designamos LoCoBoard (*Low Cost Interactive Board*). Pretende-se que esta ferramenta utilize algoritmos de visão na detecção das interacções com o quadro (via apontadores infravermelhos). A plataforma deve ajustar-se às condições do ambiente de projecção e retornar as coordenadas dos pontos de interesses, para qualquer aplicação que delas faça uso. O propósito é mostrar que com hardware comum e software apropriado é possível obter um sistema de quadro interactivo genérico, barato e útil em sala de aula. Este sistema está organizado consoante demonstrado na Figura 1.

A aplicação, que nos propomos conceber tem por objectivo principal ser compatível com diversos sistemas operativos (*Cross-Platform*), por exemplo Windows, Linux ou mesmo Mac OS X.

Um dos aspectos mais importantes neste tipo de sistemas é o suporte à detecção de interacções, através da análise das imagens recolhidas pela câmara de vídeo. Esta é efectuada por algoritmos de detecção de regiões – *Binary Large Objects* (*BLOBs*), i.e., um conjunto de píxeis com características comuns que se conseguem isolar numa imagem. Para além de analisar estes algoritmos, este trabalho apresenta também uma avaliação comparativa dos mesmos. Do mesmo modo seguir-se-á uma comparação entre o nosso sistema e um equivalente, o CCV (Tbeta 2008).

Figura 1: Ciclos de Processamento do Sistema

Propomo-nos também integrar nesta aplicação o protocolo TUIO (reacTIVision 2009), permitindo uma maior interoperabilidade com as aplicações em flash existentes, criadas pela comunidade *Natural User Interface* (NUI Group 2008). A reutilização deste protocolo traz mais-valias, como, a interoperabilidade com qualquer sistema operativo através da infra-estrutura de rede, a separação entre a aquisição e o tratamento dos PI e o suporte fornecido às aplicações existentes compatíveis com o TUIO.

1.2 Estrutura do Relatório

Esta dissertação encontra-se organizada em seis capítulos: a introdução; o estado da arte, quadros interactivos; a definição dos algoritmos utilizados; o sistema concebido; uma avaliação ao sistema e por último as conclusões.

No primeiro capítulo introduzimos os conceitos base da nossa dissertação, defendendo os motivos que nos levaram a realizar este trabalho e os objectivos a que nos propomos.

De seguida encontraremos o estado da arte onde serão apresentados os sistemas existentes e será feita uma revisão bibliográfica dos conceitos relevantes nesta área. Assim como uma abordagem inicial ao problema e uma justificação da investigação que nos propomos realizar.

No terceiro capítulo serão apresentados os algoritmos de detecção de pontos de interesse e de análise do fluxo de vídeo essenciais ao funcionamento da aplicação.

No quarto capítulo será apresentada a arquitectura da aplicação do quadro interactivo. Analisaremos ainda os mecanismos de calibração geométrica e técnicas de melhoria da imagem.

No quinto capítulo será feita uma avaliação comparativa da nossa aplicação. O desempenho e usabilidade da aplicação serão analisados em comparação com ferramentas de sistemas equivalentes.

Encerramos a dissertação no sexto capítulo com as conclusões e trabalho futuro.

Capítulo 2: Quadros Interactivos

Este capítulo procura fazer uma revisão do estado da arte, constituindo uma reflexão sobre a questão dos quadros interactivos, as tecnologias que os constituem e sobre sistemas similares existentes no mercado.

2.1 Enquadramento

Ao longo do tempo, foi notória a evolução dos computadores e dos seus componentes, no entanto, este ritmo de evolução não se verificou nas interfaces de comunicação entre o Homem e a máquina. Por exemplo, a evolução da capacidade de processamento dos computadores pessoais foi considerável (1980 – 5-10 MHz / 1MB de RAM, 2009 – 2-4GHz / 1TB de RAM, segundo (Risley 2001)), mas a forma de interacção com o computador manteve-se a mesma.

Analisando as máquinas que existiam antigamente e as existentes na actualidade, verificamos que poucas foram as alterações que estas sofreram ao nível das suas interfaces de comunicação com o utilizador. O que nos leva a pensar se as interfaces hoje usadas continuam a ser as mais adequadas face a era digital que atravessamos. Segundo (Ballmer 2009):

> "*Na era digital, a adopção de interfaces naturais entre o Homem e a máquina torna-se cada vez mais pertinente. [...] Interfaces Naturais serão usadas em cada dispositivo. Fala, gestos e escrita manual serão parte integrante da interacção com os computadores, televisões e telemóveis. Continuará a fazer sentido utilizar o teclado e o rato como únicos meios de interacção?*"

Neste excerto, é revelada a vontade de mudar e alterar as soluções existentes. Surge um novo termo para as interfaces. Estas são agora denominadas de "Interfaces Naturais". São designadas desta forma, por serem interfaces mais próximas dos utilizadores, quebrando as fronteiras dos limites da máquina, em que existe uma maior proximidade e interacção com o Homem. Na última década, a interacção entre o Homem e a máquina, tem sofrido uma revolução, com a aparição de computadores com monitores sensíveis ao toque (e.g., *Tablets PCs*) e dispositivos com interface multi-toque. (Cordis 2009) refere que a inovação, leva-nos a repensar o paradigma existente, criado à cerca de trinta e seis anos atrás, pela Xerox PARC, o WIMP (1973). Este corresponde a um acrónimo para *Windows, Icons, Menus and Pointers (Associated Universities Inc. 1996)* e serve para definir o padrão das interfaces gráficas, usadas na altura, pelo nosso ambiente e pelas nossas aplicações. Continuando a referir (Cordis 2009), não é pela tecnologia que não evoluímos as nossas interfaces. Voz, gestos, toque, sensores de movimento ou actuadores, entre outros, já se encontram disponíveis. A evolução deve ir no sentido de facilitar, cada vez mais, a interacção entre os humanos e o computador, permitindo que nos adaptemos a novas interfaces, fazendo com que a dependência do teclado e do rato seja, cada vez mais, atenuada. Este autor justifica o atraso desta evolução, devido à complexidade existente em encontrar interfaces que se adaptem à morfologia e à neurologia humana, de forma a tornar mais simples o desempenho das suas tarefas. Mas para que tal se verifique com sucesso, é necessário investigar neste sentido, o que requer tempo.

Uma evolução recente nesta área aconteceu com a introdução dos novos telemóveis, integrando sistemas de toque ou sistemas baseados em gestos, por exemplo, o caso do *iPhone* da Apple. Através de uma interface redesenhada pela Apple, o seu *SO*, oferece a possibilidade do utilizador manipular e interagir com o seu telemóvel, simplesmente através do contacto digital, sendo possível, executar tarefas como o *Zoom In* (i.e., ampliar imagem), através de um movimento de *pinch* com dois dedos. Outra evolução notada foi ao nível das consolas de jogos, com a introdução da consola Wii da Nintendo. Através do seu inovador comando de jogo, o *Wiimote,* foi possível quebrar uma das barreiras até agora existentes no comando tradicional. A interacção entre o mundo virtual, a personagem do jogo e o ser humano era sempre realizada pressionando teclas

num comando. Com a aparição do *Wiimote*, foi possível detectar os movimentos do utilizador e reproduzi-los nas personagens, que nos representam no jogo em questão. Por exemplo, é possível num jogo de ténis, com o comando na nossa mão, simular movimentos que faríamos com uma raquete numa situação real. No mesmo sentido, estão soluções ainda em desenvolvimento, como é o caso do Projecto Natal da Microsoft (Gates 2009; 2009b), que tenta através de uma câmara entender o mundo real, isto é, baseando-se nas acções dos utilizadores e nos seus movimentos, manipular o ambiente virtual que é um jogo. Será então possível, guiar um carro através da simulação da manipulação de um volante, por parte do utilizador. Em resposta ao lento desenvolvimento das interfaces, foi criado um projecto conhecido por OpenInterface (Nigay 2008). Este projecto tem como objectivo fornecer uma plataforma *open-source* para o desenvolvimento de interfaces que se comunicam de forma inteligente através de várias modalidades (e.g., o PC estimula os sentidos do ser humano). Centra-se na interacção natural entre o humano e a máquina e vice-versa.

Como verificamos, existem possibilidades de melhorar as interfaces existentes em diversos sectores, no entanto, decidimos dirigir a nossa atenção no que poderia ser feito no âmbito da educação. Numa sala de aula, o elemento principal é o quadro, pois é nele que os alunos centram toda a sua atenção e o docente sintetiza e explica a matéria a leccionar durante a exposição da aula.

A evolução dos quadros da sala de aula ao longo dos anos não foi muito evidente. De forma geral, continuamos a usar quadros a giz ou a marcadores. Um quadro interactivo segundo (Ekbutechnology n.d.), sustentado em (Burden 2002; Miller et al. 2005; The National Centre for Languages 2007), é uma superfície sensível ao toque, onde podemos ver uma imagem projectada ou reflectida, ao estilo de um ecrã. É possível interagir com essa superfície através da mão ou de uma caneta, podendo os dados ser nela directamente manipulados, sem ter de recorrer ao computador. O software deverá permitir salvaguardar as tarefas que foram realizadas ou as anotações que foram acrescentadas.

Segundo (Miller et al. 2005), um quadro interactivo pode ser usado como um quadro tradicional, permitindo-nos acrescentar-lhe interactividade e flexibilidade, o que faz dele uma tecnologia especial. (Ekbutechnology n.d.) afirma que, como professor, pretende tornar a matéria mais interessante e atractiva aos alunos, a fim de lhes estimular a vontade de aprender e participar. Um quadro interactivo, à imagem do que é um computador, permite-nos navegar pela internet e promover a toda a turma conteúdos, que tradicionalmente eram divulgados, através do uso do computador pessoal. Desta forma, conseguimos reduzir o tempo de pesquisa, e promover um maior tempo de estudo. Por exemplo, numa aula de geografia através de software, como o *Google Earth* é possível identificar os países, as cidades e até mesmo ir ao detalhe e consultar alguns edifícios. A informação torna-se acessível de forma rápida e simples a toda a turma. A interactividade pode também trazer algumas vantagens nas aulas ministradas à distância, caso sejam criadas aplicações, que permitam uma interacção semelhante ao ambiente de trabalho remoto. O facto de poder salvaguardar a informação escrita no quadro, também pode constituir uma mais-valia para a turma, permitindo ao professor relembrar um conceito já explicado anteriormente, bastando-lhe para isso encontrar no histórico a apresentação correspondente.

Um quadro interactivo não é uma ferramenta com a mesma simplicidade de uso que o quadro tradicional. Isso deve-se às múltiplas soluções que um sistema destes pode oferecer. Esta complexidade é um factor a ter em conta para o professor e também para os alunos, daí ser necessária uma boa formação prévia do docente, a fim de que fique apto a ensinar os alunos a usar o quadro interactivo. Os elementos apresentados anteriormente em (Ekbutechnology n.d.), permitem-nos ter uma ideia das vantagens destes sistemas numa sala de aula. Existem três tipos de quadros diferentes:

- O quadro tradicional, constituído por marcador a tinta ou giz, é o mais usado, mas não fornece interactividade.
- O quadro com tela, que permite o uso de videoprojector, é uma evolução do modelo anterior, porque permite-nos alargar o ambiente do nosso computador a toda a sala de aula, através da projecção na tela. Neste sistema, a interactividade não está patente, porque não é possível interagir directamente na projecção, pois

qualquer interacção deverá ser feita no próprio computador. Este sistema já é frequentemente usado pelos docentes na prática pedagógica.

- O quadro interactivo é uma solução mais completa que a anterior, porque permite ao professor interagir directamente na projecção, sem necessitar de recorrer ao seu computador. Este torna a tarefa de leccionar mais fácil e intuitiva, tanto para os alunos como para o docente.

Tabela 1: Comparação dos Preços de Quadros Interactivos [Silva, M., 2008]

Modelos de Quadros Interactivos	***Preços***
eBeam Integral 65	~800 €
InterWrite 1071	~1150 €
Promethan Activboard 95 Studio	~1900 €
SMARTBoard ESP680-N	~3400 €
SMARTBoard 2000i	~7100 €

Tendo por base um estudo realizado por (Silva 2008, pp.13-16), construímos uma tabela recapitulativa de um estudo dos preços de mercado dos Quadros Interactivos, ver Tabela 1. Podemos justificar que a panóplia de soluções comerciais disponíveis é vasta, mas o custo associado é geralmente elevado, para permitir equipar de forma generalizada os estabelecimentos de ensino, principalmente nos países de economia mais débil. Assim sendo, faria sentido continuar a evoluir as soluções encontradas, com o propósito de as tornar mais acessíveis. Desta forma, poderíamos permitir massificar o uso de quadros interactivos e tirar proveito das mais-valias, que foram atrás referidas.

As tecnologias usadas nos quadros interactivos comerciais encontram-se, segundo (Silva 2008, pp.6-8), sintetizadas na Tabela 2. Identificamos várias diferenças entre as tecnologias usadas, sendo que, uma delas consiste na necessidade de ter uma superfície de contacto específica, isto é, necessitamos de um quadro específico preparado para suportar uma tecnologia, como são os casos da tecnologia Resistiva, Electromagnética e Capacitiva. Os outros sistemas são mais transversais e permitem a sua adaptação às superfícies tradicionalmente usadas. No caso da tecnologia de leitura óptica de infra-

vermelhos, existem situações que podem necessitar da personalização da superfície de contacto (consulte Tabela 4), com o objectivo, de tornar o nosso sistema mais eficiente.

Tabela 2: Diferentes Tecnologias usadas em Quadros Interactivos [Silva, M., 2008]

Tecnologias	***Características***
Resistiva (Silva 2008, p.6)	O sistema é composto por duas placas separadas por uma camada de ar. Quando é executado um toque ou pressão na camada superior, ocorre um contacto eléctrico entre as duas superfícies. Desta forma, identificamos o ponto de contacto. Este sistema não suporta funções multi-toque. Tem resoluções de cerca de 4000 píxeis.
Electromagnética (Silva 2008, p.6)	O sistema é composto por uma superfície de contacto que contém um conjunto de fios, que interagem na presença de uma bobine presente na caneta. Existe um conjunto de sensores magnéticos no quadro que lêem as anteriores alterações devido à caneta. Estes sistemas suportam resoluções à volta dos 30000 píxeis.
Capacitiva (Silva 2008, p.7)	Como na tecnologia electromagnética anteriormente referida, existe um barramento de fios presentes por trás da superfície de contacto, no entanto, o sistema agora é capaz de reagir a qualquer objecto que entre em contacto com o ecrã.
Leitura óptica de infravermelhos (Silva 2008, p.7)	Com a tecnologia de infra-vermelhos, embora existam diferentes concepções possíveis, o seu funcionamento é similar. É usada uma câmara, a fim de capturar os Pontos Infra-vermelhos ou as sombras. As diferentes concepções tendem a optimizar a detecção dos toques na superfície de contacto.
Ultra-sons (Silva 2008, p.8)	Este sistema funciona por leitura e sinalização de sinais ultra-sónicos. Cada vez que a caneta entra em contacto com a superfície, esta emite um sinal e dois receptores colocados em cada um dos cantos da superfície de contacto, conseguem processar a distância, em que a caneta se encontra de ambos, recorrendo ao tempo de propagação e à velocidade. O ponto de intersecção das duas distâncias corresponde ao local onde se encontra a caneta.

O nosso projecto, tem como principal objectivo, obter uma solução com custo reduzido por isso não consideramos como possível linha de pesquisa a manipulação da superfície de contacto. A tecnologia de ultra-sons seria uma opção válida, mas não é uma solução que permita mobilidade. Depois de instalada, esta requer a fixação dos sensores nos cantos do quadro e a necessidade de ter uma caneta emissora de sinais ultra-sónicos. Por estes motivos, decidimos então explorar a solução das tecnologias de

leitura de sinais de infravermelhos, tentando manter e reaproveitar ao máximo o material já existente na sala de aula.

Quanto menor for a mudança de *hardware*, mais fácil será a sua adopção por parte dos utilizadores. Segundo (NUI Group Authors 2009, p.3) as soluções baseadas em sensores ópticos ou de luz (i.e., câmaras), permitem-nos criar uma larga escala de dispositivos multi-toque. Assim, as soluções desenvolvidas nesta área tornam-se atractivas devido à sua escalabilidade, baixo custo e facilidade de instalação.

Segundo (NUI Group Authors 2009, p.3), a luz IR foi escolhida pelas suas características próprias. Este tipo de luz só é perceptível pelas câmaras comuns. No entanto, estas estão muitas vezes equipadas com um filtro de luz IR. Ao substituir este filtro por um que só deixa passar a luz IR, obtemos uma câmara que deixa de ser sensível a todas as luzes do ambiente, mas que fica habilitada a detectar os pontos de luz IR. Sendo assim, conseguimos diminuir a quantidade de ruído e limitar a visão do ambiente da nossa câmara. No final, obtemos um sistema sensível a variações de fontes de luz IR.

Cada vez mais, face às dificuldades económicas que estamos a atravessar, é necessário encontrar novas soluções menos dispendiosas. Este é o desafio a que nos propomos com este projecto, cujo objectivo consiste em criar um sistema de quadro interactivo de código aberto e de baixo custo. Os requisitos de *hardware* são baixos a fim de permitir massificar o seu uso, a saber:

- Um computador. Este elemento é necessário para processar e tornar a superfície interactiva. A superfície não é nada mais que a imagem projectada do ambiente de trabalho do computador usado.
- Uma câmara (*webcam*). Que irá permitir ao computador através da aplicação desenvolvida "ver e sentir o ambiente", de forma com que a aplicação possa tomar em consideração as acções pretendidas pelos utilizadores.
- Um videoprojector. Este elemento é necessário para expandirmos o ambiente do nosso computador a uma larga escala visível, por todos os utilizadores.

- Um apontador de infravermelhos. Este é o elemento que a câmara irá identificar ou visualizar no ambiente.

É importante salientar que o projecto que nos propomos desenvolver não é de todo um concorrente a nível do desempenho com sistemas de quadros interactivos comerciais. O que nos propomos é conceber uma solução usável e com um custo reduzido.

2.2 Ferramentas Tecnológicas

O OpenCV (Bradski & Kaehler 2008) é uma biblioteca de código aberto sob licença BSD, isto é, livre para uso académico ou comercial. Esta contém rotinas básicas de processamento de imagem apropriadas ao desenvolvimento de aplicações, na área da Visão por Computador. Com base nesta biblioteca é possível adquirir imagens através de uma câmara e efectuar o seu processamento, de forma a obter informação sobre pontos de interesse nas mesmas. A área da visão por computador tem registado evoluções consideráveis nos últimos anos, com aplicações nas áreas médicas, industriais (e.g., robótica) e na engenharia aeroespacial (e.g., veículos autónomos).

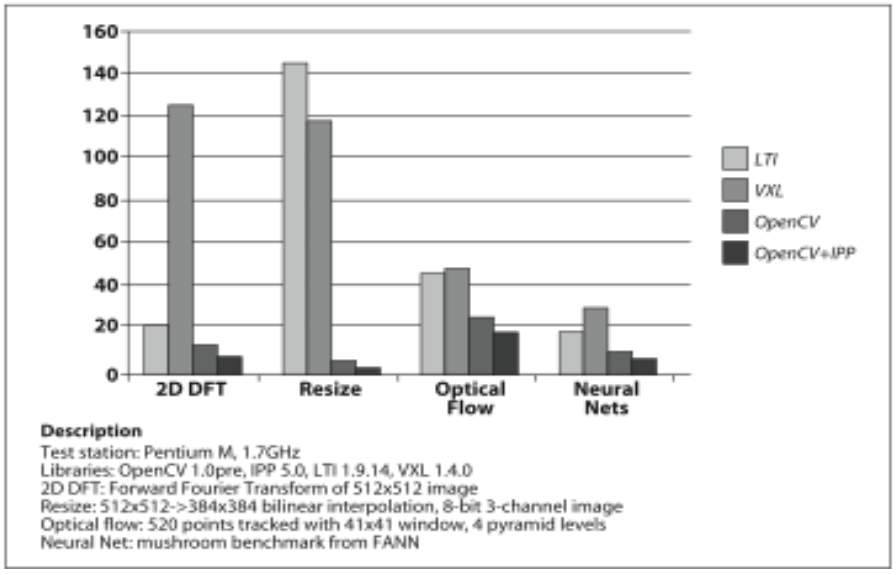


Figura 2: Comparação entre Diferentes Bibliotecas de Visão [Bradski & Kaehler 2008, p.8]

Na obra (Bradski & Kaehler 2008, p.7), é realçada a mais-valia de usar o OpenCV em detrimento de outras bibliotecas, quando se trata de analisar imagens em tempo real. A fim de comprovar o mesmo, foi realizada a experiência presente na Figura 2. Foram usadas duas bibliotecas de visão, a LTI e a VSL comparadas com a biblioteca OpenCV. Esta última, pode ser utilizada com e sem *Intel Integrated Performance Primitives* (Intel® 2009). Com base nestas bibliotecas, foram realizados quatro diferentes testes com o objectivo de avaliar o seu desempenho. O tamanho de cada uma das barras da figura é proporcional ao tempo de execução de uma determinada tarefa com uma dada biblioteca.

Constatou-se que, em todos os casos, a biblioteca OpenCV apresenta melhores resultados.

Do mesmo modo, consideramos o OpenCV, uma biblioteca fiável por ser utilizada por grandes empresas como é o caso da Intel, IBM, Microsoft, Sony, Siemens, Google, entre outras e centros de investigação como é Stanford, MIT, CMU, Cambridge, INRIA (dados obtidos segundo (Vadim Pisarevsky 2007, p.7)). Esta biblioteca encontra-se em desenvolvimento contínuo como podemos ver na Figura 3, com o intuito de se tornar cada vez mais eficaz em qualquer sistema operativo. A última versão estável (*v2.0*) foi lançada no passado dia 30 de Setembro 2009.

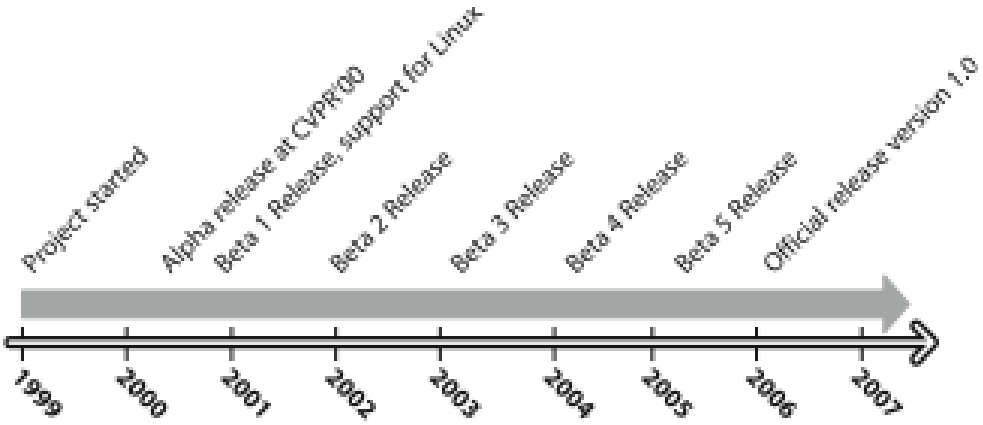


Figura 3: Evolução do OpenCV [Bradski & Kaehler 2008, p.7]

O OpenCV é a base de algumas aplicações e bibliotecas (i.e., *framework*) conhecidas, como é o caso do Touché (Kaindl 2006), Touchlib (Touchlib n.d.) ou o CCV (Tbeta 2008). Estas serão definidas mais detalhadamente na secção 2.3.

A biblioteca OpenCV possui a arquitectura presente na Figura 4, acompanhada da sua devida legenda e definição dos módulos que a compõe na Figura 5. O *HighGUI* é usado para a aquisição de imagens, criação de janelas e acompanhamento dos eventos do rato ou teclado. Este módulo permite adquirir as imagens ou o fluxo de vídeo a ser analisados pela biblioteca. Seguidamente o *CXCORE* é o módulo que contém as operações disponíveis e possíveis nas matrizes (i.e., transformações ou operações), sendo estas usadas para a representação de imagens. É neste módulo, que podemos gerir a memória usada pela biblioteca. O módulo *CV* permite-nos desencadear um conjunto de operações nas estruturas de imagens que possuímos, isto é, alterações da escala de cor (e.g., conversões para *grayscale*), *Blur*, entre outras. Estas operações pretendem simplificar a tarefa de detecção e processamento de imagem. Encontram-se também disponíveis métodos, que nos permitem fazer seguimento de objectos e reconhecimento de rosto por exemplo. Por fim, o módulo *MLL* é constituído por um conjunto de algoritmos, que facilitam as tarefas de estudos estatísticos. Este pode ser usado em aplicações com uma componente de inteligência artificial.

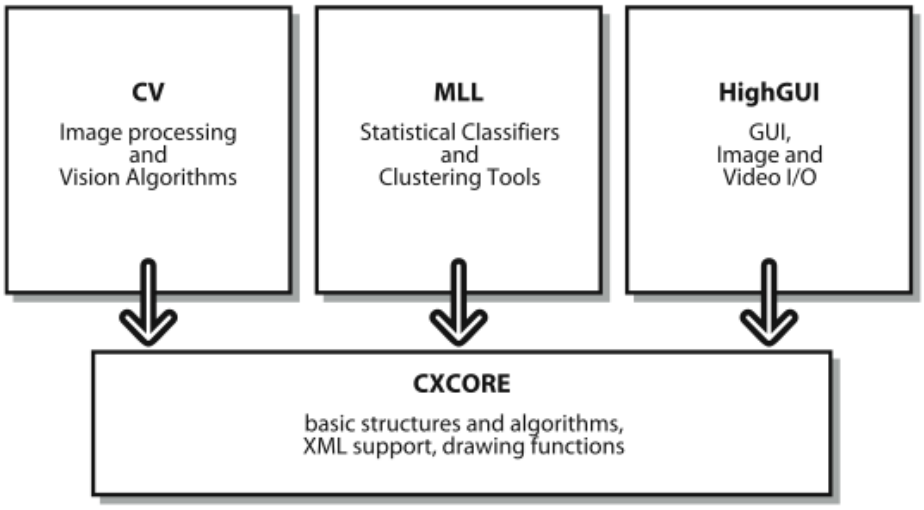


Figura 4: Arquitectura da Biblioteca OpenCV [Bradski & Kaehler 2008, p.13]

É com base nesta biblioteca, que foi desenvolvida a aplicação do quadro interactivo de baixo custo. Não serão reutilizados todos os métodos que a ferramenta dispõe, isto porque pretendemos reproduzir e estudar os diferentes comportamentos de algoritmos que iremos desenvolver. Serão então usadas as funções básicas da biblioteca OpenCV, como é a captura e tratamento de imagem. Métodos como a detecção de con-

tornos ou reconhecimento de padrões não serão utilizados. Pretende-se uma solução de código aberto, simples e que permita ilustrar a forma de trabalhar a partir de um fluxo de vídeo ou de imagens em tempo real.

CXCORE
Contains data structures, matrix algebra, data transforms, object persistence, memory management, error handling, and dynamic loading of code as well as drawing, text and basic math.

CV
Contains image processing, image structure analysis, motion and tracking, pattern recognition, and camera calibration.

Machine Learning (ML)
Contains many clustering, classification and data analysis functions.

HighGUI
Contains user interface GUI and image/video storage and recall.

Figura 5: Definição dos módulos que compõem a Arquitectura da Biblioteca OpenCV [Bradski & Kaehler 2008, p.11]

Recorremos também a uma função da *framework* QT (NOKIA 2008), devido a ausência no OpenCV de uma função que nos permita controlar o ponteiro do rato deslocando-o para coordenadas definidas. A escolha do QT (consulte a sua arquitectura na Figura 6) deve-se ao facto desta ser uma biblioteca funcional em Windows, Mac OS X, e Linux (i.e., *cross-platform*), permitindo-nos através dum único método controlar o rato independentemente do sistema operativo usado. Outra mais-valia reside na sua fácil integração com as linguagens C/C++ e Java facilitando a sua adopção. Por outro lado esta permite-nos com maior facilidade, a longo prazo, implementar uma interface para a nossa aplicação que seja funcional em qualquer um dos sistemas operativos.

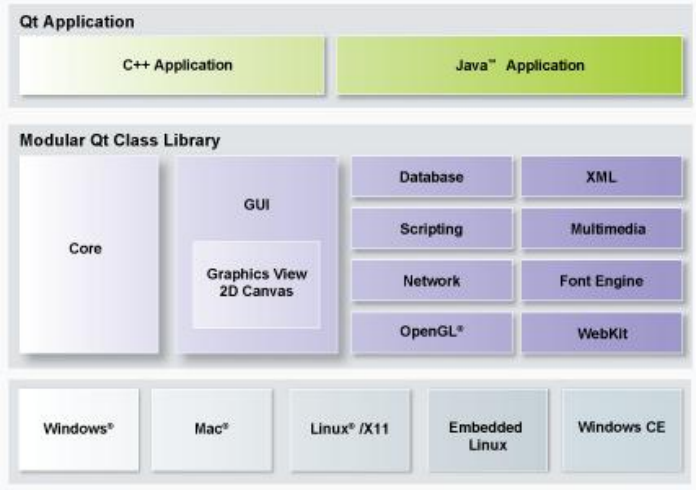


Figura 6: Framework QT [(QT Nokia n.d.)]

Foi ainda utilizado um conjunto de classes denominado de *OSCpack* (Bencima 2006), a fim de integrar na nossa aplicação o protocolo TUIO (TUIO.org n.d.; reacTIVision 2009). Segundo (Bencima 2006), estas classes têm implementado métodos que permitem fazer o *packing* e *unpacking* dos pacotes *Open Sound Control (The Center For New Music and Audio Technology (CNMAT) 2002)*. Pode-se reutilizar este código em qualquer implementação, tanto de um servidor como de uma aplicação com suporte às mensagens com o padrão OSC. Este foi criado com o intuito de formar um padrão único de mensagens de forma a serem reutilizadas entre computadores no uso de aplicações ligadas a área da multimédia. Podemos considerar o *OSCpack* como uma biblioteca de código que nos proporcionará de forma simples enviar, receber e analisar pacotes de mensagem TUIO.

2.3 Sistemas Relacionados

Existem outras ferramentas disponíveis para esta implementação (Tabela 3). Estes conjuntos de bibliotecas são todos ligados à área de multi-toque. Embora a nossa aplicação não seja especificamente pensada com um suporte nativo para multi-toque, esta poderá ser uma possível linha de trabalho para o futuro. Cada uma delas tem por base a visão por computador, recorrendo a uma câmara para detecção de *Binary Large Object* (*BLOB*) no ambiente.

Existem algumas plataformas com características semelhantes às do sistema aqui exposto. São os casos da Touchlib (Wallin 2008; Touchlib n.d.), e da Tbeta (Tbeta 2008). Estas plataformas utilizam o OpenCV para efectuar detecção dos pontos de interesse nas imagens captadas pela câmara. Esta decisão, como veremos, tem consequências na carga de processamento necessária (consulte a secção 5.1.5 que se encontra na página 71). Verificamos através da análise do código fonte do Tbeta, que este recorre à função *cvFindContours* de forma linear, acompanhado de salto nas linhas (*y*). Destacamos também a vantagem de reutilizar o protocolo existente TUIO (reacTIVision 2009), desenvolvido para estes sistemas, o qual permite reportar coordenadas dos pontos de interesse (PI) ou *BLOBs* através de mensagens num formato comum.

Tabela 3: Tecnologias & Bibliotecas usadas em Tecnologias Multi-toque [NUI Group Authors, 2009]

Tecnologias	***Características***
BBTouch (NUI Group Authors 2009, p.59)	BBTouch é uma solução *open-source*, para ambiente Mac OS X, baseada em tecnologia de detecção e seguimento por câmara para mesas multi-toque (Smith 2008).
Bespoke Multi-touch Framework (NUI Group Authors 2009, p.59)	Este é uma *framework* que permite a sua reutilização, na criação de interfaces multi-toque. Pode ser utilizada e manipulada para conceber a aplicação pretendida. Usa um sistema de envio de eventos baseados em OSC (Varcholik 2008).
reacTIVision (NUI Group Authors 2009, p.59)	Esta é uma *framework, open-source* e *cross-platform* usada como um sistema de *tracking* a padrões afixados em objectos. (reacTIVision 2009).
Community Core Vision (CCV) (NUI Group Authors 2009, p.60)	Esta plataforma, primeiro conhecida por Tbeta, é uma solução *cross-platform*, na área de visão por computador e multi-toque. Usa como entrada um fluxo de vídeo, devolve dados sobre os PI (tamanho e posição) e os seus eventos, baseados em OSC (Tbeta 2008).
Touché (NUI Group Authors 2009, p.60)	Esta é uma plataforma *open-source*, para *tracking* de *BLOB* em mesas multi-toque. Só funciona em ambiente Mac OS X (Kaindl 2006).
Touchlib (NUI Group Authors 2009, p.60)	Esta é uma biblioteca, que nos permite criar uma interface multi-toque. Este permite a detecção e seguimento de *BLOBs* IR. Reporta os eventos de interacção (*finger down, moved, released*). Funciona em ambiente Windows, embora existam esforços realizados no sentido de adaptá-lo a outros ambientes (Touchlib n.d.).

A Figura 7 (reacTIVision 2009) ilustra o funcionamento do sistema reacTIVision que permite através da utilização de uma câmara reportar *BLOBs* de radiação infra-vermelha. O sistema pode ser divido em duas máquinas distintas através do uso de mensagens TUIO (reacTIVision 2009), podendo estar uma máquina a fazer detecção de pontos e outra dedicada à projecção e interacção com o utilizador. Dessa forma podemos propiciar um balanceamento de carga entre os dois computadores. Do mesmo modo conseguimos simplificar, isolando o problema que consiste na detecção da interacção.

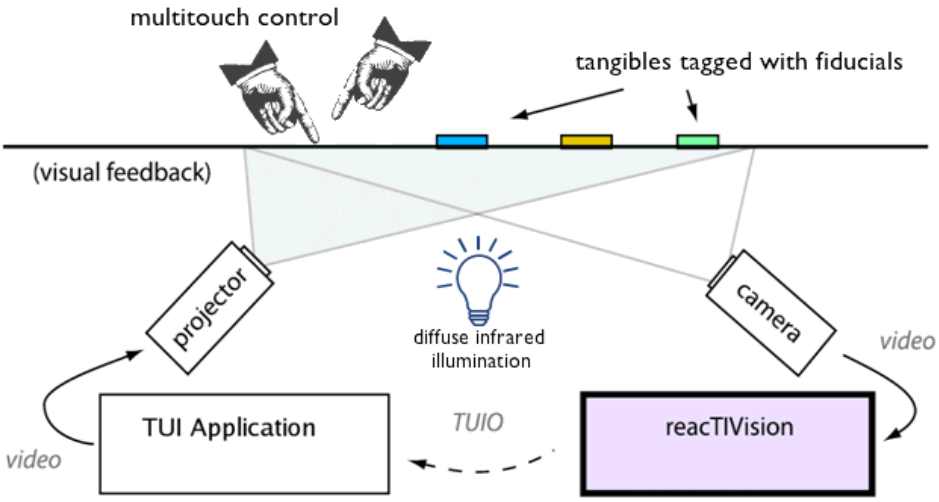


Figura 7: ReacTIVision Framework Diagram [Reactable - ReacTIVision, 2008]

O artigo (Bovermann et al. 2005) é dedicado ao TUIO, e mostra que este protocolo é simples e fiável e foi particularmente concebido para ir de encontro às necessidades das novas interfaces com o utilizador (e.g., multi-toque). Este é um protocolo que serve para reportar as posições de um objecto (ou das digitais) numa superfície de toque. Possui ainda a capacidade para transmitir gestos (e.g., ângulo de movimento ou aceleração do PI) que possam ser executados por parte do utilizador. A real mais-valia do uso deste protocolo em qualquer aplicação deve-se ao facto de poder separar, a aquisição (principal objecto do nosso trabalho de investigação) da interpretação dada às coordenadas (e.g., controlo do rato). Dessa forma qualquer pessoa que tenha devido conhecimento do formato de mensagem do protocolo será capaz de conceber uma aplicação idêntica aos exemplos em flash propostos (Tbeta 2008). O programador terá a liberdade de reutilizar a linguagem a seu gosto, sendo sempre preferidas linguagens como o Java e o flash de forma a manter uma compatibilidade entre todos os sistemas operativos. O flash é uma solução privilegiada por permitir de forma simples fornecer suporte através da rede e por sustentar acções para mais do que um PI. Actualmente os sistemas operativos não suportam mais do que um cursor, o que limita a utilização do multi-toque. Mas com a aparição dos ecrã-tácteis estão a ser repensados estes conceitos de forma, a permitir uma integração de gestos, um pouco ao estilo do que é hoje em dia possível nos telemóveis de nova geração, como é o iPhone. No mesmo artigo estão definidos os formatos das mensagens do TUIO (ver Figura 8) a ser enviados em cada *frame*.

```
/tuio/[profileName] alive [list of active sessionIDs]

/tuio/[profileName] set [sessionID parameterList]

/tuio/[profileName] fseq [int32]
```

Figura 8: Formato de Mensagens do TUIO [Bovermann et al. 2005]

O *profileName* corresponde a um perfil de mensagem que deverá estar de acordo com os perfis existentes (consulte a Figura 9).

2D Interactive Surface

```
/tuio/2Dobj set s i x y a X Y A m r
/tuio/2Dcur set s x y X Y m
/tuio/2Dblb set s x y a w h f X Y A m r
```

2.5D Interactive Surface

```
/tuio/25Dobj set s i x y z a X Y Z A m r
/tuio/25Dcur set s x y z X Y Z m
/tuio/25Dblb set s x y z a w h f X Y Z A m r
```

3D Interactive Surfaces

```
/tuio/3Dobj set s i x y z a b c X Y Z A B C m r
/tuio/3Dcur set s x y z X Y Z m
/tuio/3Dblb set s x y z a b c w h d v X Y Z A B C m r
```

Figura 9: Perfis de mensagens do TUIO [Bovermann et al. 2005]

O *sessionID* é o identificativo correspondente e o atribuído a cada um dos *BLOBs* reconhecidos. A primeira linha serve para reportar para cada PI as suas características. Estas podem ser consultadas na Figura 10.

s	sessionID, temporary object ID, int32
i	classID (e.g. marker ID), int32
x, y, z	position, float32, range 0...1
a, b, c	angle, float32, range 0..2PI
w, h, d	dimension, float32, range 0..1
f, v	area/volume, float32, range 0..1
X, Y ,Z	movement vector (motion speed & direction), float32
A, B, C	rotation vector (rotation speed & direction), float32
m	motion acceleration, float32
r	rotation acceleration, float32
P	free parameter, type defined by OSC packet header

Figura 10: Características por PI no TUIO [Bovermann et al. 2005]

Numa segunda fase, será enviada a lista dos *Id* (i.e., número identificativo por cada PI) que se encontram activos. Por fim, é enviado um número que corresponde à sequência de imagem, o qual permite identificar mensagens obsoletas.

Por fim, destaca-se o projecto desenvolvido por Johnny Lee (Lee n.d.; Lee 2008), como sendo um dos pioneiros na concepção de quadros interactivos de baixo custo, reutilizando um comando da Nintendo, o *Wiimote*. Este dispositivo permite a detecção de pontos IR e, por bluetooth, reporta as respectivas coordenadas ao dispositivo emparelhado. Este tipo de sistema possui algumas limitações, das quais se destaca o facto de o dispositivo constituir propriedade intelectual do criador da Nintendo, permanecendo os algoritmos usados desconhecidos dos programadores. Outro inconveniente é o facto da precisão da detecção estar limitada pela resolução da câmara embutida na *Wiimote*: apenas 128x96 (Silva 2008, p.23). Por outro lado, a utilização da câmara de vídeo disponível na maioria dos computadores recentes é sempre preferível à utilização de um periférico adicional. Esta solução concebida por Johnny Lee tem a desvantagem de não ser portável, daí Uwe Schmidt ter concebido uma aplicação equivalente, mas sustentada em Java (Schmidt 2008).

Em (He et al. 2002), os autores realçam a necessidade de evolução do modelo tradicional do uso de um quadro. Propõe uma solução baseada numa câmara, num microfone e num computador, para conseguirem armazenar num documento digital as várias partes de uma reunião. Este documento é pesquisável, permitindo assim rever e ouvir partes da reunião que sejam do nosso interesse. O intuito é permitir aos participantes focar-se exclusivamente nos objectivos da reunião, sem se preocuparem em tirar notas e evitando o esquecimento de tarefas ou ideias discutidas.

Os autores de (Hanning Zhou et al. 2004), pretendem uma migração suave entre o mundo digital e os sistemas tradicionais, isto é, permitir a uma câmara distinguir a nova informação que foi escrita, quer esta esteja num quadro interactivo (meio digital), ou num quadro a giz (meio tradicional). Podemos com este modelo ter um suporte para ambos os métodos, quer estes sejam mais inovadores ou tradicionais. Algumas técnicas abordadas neste artigo (e.g., *Visual Echo Cancelation*), podem revelar-se importantes,

nomeadamente as técnicas de calibração, que permitem, por sua vez gerar previsões. Os autores conseguem prever em que posição e de que forma, a imagem projectada irá ser visualizada pela câmara. Esta informação é valiosa, porque permite ao algoritmo focar a sua atenção em determinadas áreas consideradas de interesse. É necessário calibrar a cor, porque quando uma imagem é projectada pode-se tornar mais escura ou mais clara, dependendo das condições de luminosidade ou mesmo do projector. A perspectiva da projecção também difere consoante a distância entre o projector e a área projectada, sendo necessário, criar uma escala que nos permita mapear a resolução entre a imagem real (i.e., PC) e a projectada.

A *Natural User Interface* (NUI Group) foi criada em 2006, com o intuito de investigar a interactividade entre a relação homem-máquina. O seu objectivo principal seria criar aplicações úteis para as comunidades artísticas, comerciais ou mesmo educativas. Este grupo oferece um conjunto de informação aos programadores, que estejam interessados na aprendizagem e na partilha de novos métodos e conceitos relacionados com *Human Computer Interaction* (HCI). Esta foca-se em diversas áreas como é a realidade aumentada, reconhecimento de voz/escrita/gestos, *touch computing* e visão por computador (NUI Group Authors 2009, p.ii).

Tabela 4: Diferentes Técnicas para Concepção de um Modelo Multi-toque

Sigla	***Denominação***	***Referencia***	***Img. no Anexo 1***
FTIR	Frustrated Total Internal Reflection	(NUI Group Authors 2009, fig.2)	Fig. 1
DI	Diffused Illumination	(NUI Group Authors 2009, fig.3)	Fig. 2
LLP	Laser Light Plane	(NUI Group Authors 2009, fig.4)	Fig. 3
DSI	Diffused Surface Illumination	(NUI Group Authors 2009, fig.5)	Fig. 4
LED-LP	Led Light Plane	(NUI Group Authors 2009, fig.6)	Fig. 5

Segundo (NUI Group Authors 2009, p.2), multi-toque é um conjunto de técnicas de interacção, que permitem aos utilizadores do computador controlar as aplicações gráficas através do toque digital. São sistemas constituídos por uma superfície sensível ao toque, a qual pode ser um monitor, um *touchpad*, uma mesa ou até mesmo uma parede. Este tipo de sistemas tem a vantagem de suportar a detecção de múltiplos pontos. Esta comunidade investigou possíveis formas de construir sistemas capazes de detectar múltiplos pontos tanto a nível do *hardware* como do *software*. As soluções apresentadas por estes, são mais completas, do que as que nos propomos a desenvolver, pois já suportam multi-toque. Nós só pretendemos, numa primeira fase, desenvolver uma aplicação sensível a um único ponto.

Existem diferentes tipos de modelos, sugeridos pelo NUI, a fim de criar interfaces multi-toque. Apresentamos os principais modelos na Tabela 4. Estas informações foram encontras em (NUI Group Authors 2009, pp.9-19). Estes sistemas apresentados pela NUI proporcionam uma melhoria significativa no desempenho de aplicações com leitura de IR, mas no entanto obrigam a manipulação da superfície de toque por parte do utilizador. No caso do nosso sistema optamos pela não manipulação da superfície de forma a manter a simplicidade de adopção do sistema por parte do utilizador. No entanto achamos importante referir que estas são soluções viáveis e aconselháveis, de forma a proporcionar um sistema com melhor desempenho, nomeadamente a nível da detecção do PI. Estas superfícies actuam como amplificadores da mancha IR, proporcionando a melhor detecção possível do ponto pela câmara. As diferenças existentes entre cada um dos modelos devem-se à posição da câmara, das fontes emissoras de luz IR, ou até mesmo do tipo de vidro usado para a propagação. Caso se queira adoptar uma destas soluções, é de referir que estas serão compatíveis com o nosso sistema.

Capítulo 3: Algoritmos de Detecção e Seguimento de PIs

Neste capítulo descrevem-se os algoritmos correspondentes ao bloco "Detecção e *Tracking* de *BLOBs* na *Frame*" apresentado na Figura 1. Estes algoritmos descrevem o modo como é realizada a análise de cada imagem, para a detecção da posição de interacção no formato de coordenadas cartesianas (*x*, *y*). Os algoritmos serão comparados posteriormente no capítulo de avaliação.

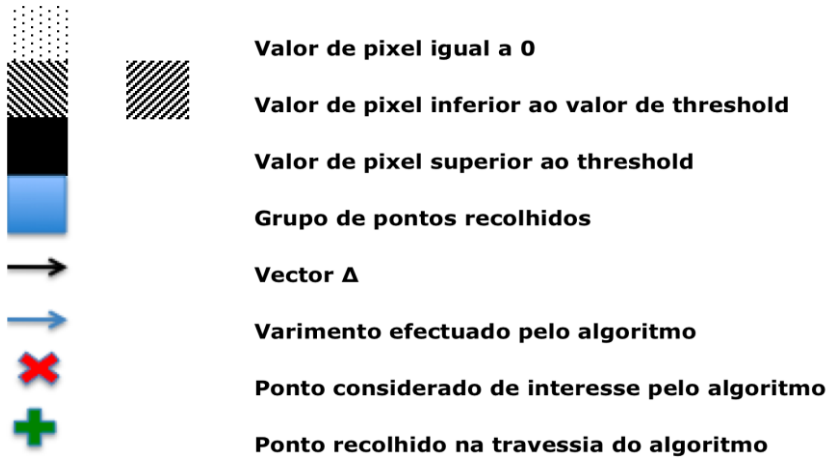


Figura 11*: Legenda da Sinalética usada na Definição dos Algoritmos*

A detecção e seguimento de PIs baseiam-se na análise de imagens representadas por uma matriz de píxeis. Cada píxel é constituído por três valores, representando as componentes, vermelho, verde e azul do espaço de cor RGB. Convertendo uma imagem para uma escala de cinzentos (*grayscale*), passamos a ter apenas um valor por cada píxel. No processamento em tempo real de cada imagem, esta transformação revela-se fundamental sendo que a análise pode ser até três vezes mais rápida, do que com imagens a cores. Por outro lado, a aplicação fica impossibilitada de reconhecer e distinguir cores diferentes. Outro aspecto a ter em conta é a distribuição das coordenadas do PI. Estas são reportadas tendo como origem o ponto (*0,0*), correspondente ao canto superior esquerdo de cada imagem adquirida da câmara de vídeo. A coordenada máxima é defi-

nida pela (*largura*, *altura*) da imagem. Dependendo da resolução da câmara, podemos trabalhar com imagens de diferentes tamanhos o que, por um lado, pode ser benéfico porque contribui para melhorar a precisão da detecção mas, por outro lado, aumenta a utilização do CPU.

Segue-se uma breve descrição dos cinco algoritmos de detecção e seguimento de *BLOBs* gerados pela caneta IR. Podemos ver na Figura 11 a sinalética que será usada na descrição de todos os algoritmos. As funcionalidades de detecção e seguimento foram evoluindo com o seu nível de desenvolvimento:

1º Inicialmente, pretendia-se suportar apenas um PI, retornando a coordenada do centróide (A1);
2º Posteriormente, adicionou-se suporte para múltiplos pontos, todos recolhidos através de uma pesquisa linear, ou seja, percorrendo píxel após píxel toda a *frame* (A5);
3º De seguida, no modelo de detecção de um só ponto introduziu-se a noção de previsão, que permite à aplicação, em cada momento, prever onde irá aparecer o PI numa nova *frame*, conhecendo o seu deslocamento e começando a partir dessa previsão uma pesquisa em modo espiral (A4 – primeira versão *spiral*).
4º Numa quarta fase, melhorou-se o algoritmo de previsão em espiral, restringindo a pesquisa a uma subárea. Embora o modelo seja eficaz para espirais de pequenas dimensões, torna-se menos eficiente do que a pesquisa linear quando "N" (i.e., níveis da espiral) ultrapassa aproximadamente oito níveis (consulte Figura 63) (A4 – segunda versão *spiral2*).
5º Na última fase, concebeu-se um algoritmo baseado em A1, mas implementando saltos na pesquisa linear (A2). Do mesmo modo implementou-se uma variante ao algoritmo anterior no processamento do centróide (A3).

3.1 Algoritmo Simples de Detecção – A1

Neste algoritmo é efectuado um varrimento da imagem por linhas, como é ilustrado na Figura 12, de modo a encontrar um valor superior a um determinado limiar (*threshold*), ou um valor máximo aceitável definido pelo utilizador na fase de inicialização. Este algoritmo recolhe as coordenadas de todos os pontos que têm um valor superior ao *threshold* e devolve a média dos valores em *x* e *y*.

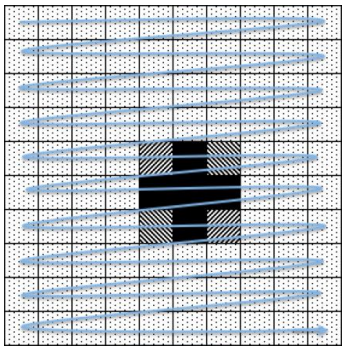


Figura 12: Travessia linear no Algoritmo Simples (A1)

O elemento principal deste algoritmo encontra-se resumido nos dois ciclos *for* descritos abaixo (consulte a Figura 13). São percorridos todos os píxeis da imagem e caso estes tenham um valor superior a um determinado limiar (cf., *threshold*) as suas coordenadas em *x* e *y* são armazenadas num vector.

```
for (int row = 0; row < workFrame->height; ++row) {


      for (int column = 0; column < workFrame->width; ++column) {
            if (workFrame->imageData[(row * workFrame->width) + column]
            >= threshold) {
                  xValues.push_back(column);
                  yValues.push_back(row);
            }
      }
}
```

Figura 13: Excerto do Algoritmo Simples A1

Após esta tarefa, procede-se ao cálculo do centróide. Para isso, usaremos uma função que irá fazer a média dos valores em *x* e em *y* (ver Figura *14*).

```
for (int i = 0; i < (int) xValues.size(); i++) {
      xTemp = xTemp + (int) xValues.at(i);
      yTemp = yTemp + (int) yValues.at(i);
}
center.x = (int) (xTemp / xValues.size());
center.y = (int) (yTemp / yValues.size());
```

Figura 14*: Excerto do Algoritmo Centróide recorrendo a Médias*

3.2 Algoritmo Simples de Detecção com Salto – A2

Este algoritmo tem um funcionamento semelhante ao anterior, mas permite definir um factor de salto S. Desta forma, entre duas leituras de píxeis, o algoritmo ignora S-1 píxeis. No processamento da coordenada do PI usamos o mesmo método, ou seja, reportamos a média dos valores recolhidos em *x* e *y*. Quanto menor for o valor de S, mais preciso será o cálculo do centro do *BLOB* a detectar.

No que toca à implementação com o anterior algoritmo, as variações são poucas. Há o acréscimo de um factor *step*, que corresponde a um valor inteiro, o qual foi enviado como parâmetro ao evocar a função (ver Figura *15*). Este factor de salto é aplicado entre colunas, contudo também poderá ser válido aplicar o mesmo em linhas. Alerta-se para o caso de usar o factor de salto nas linhas, o seu valor deverá ser significativamente menor do que nas colunas, isto porque irá ser ignorada muita informação que se poderá revelar posteriormente importante, para definir com precisão as coordenadas do ponto de interesse. No processamento do centróide, usa-se a mesma função que foi vista anteriormente (ver Figura *14*).

```
for (int row = 0; row < workFrame->height; ++row) {
      for (int column = 0; column < workFrame->width; column += step)
{
            if (workFrame->imageData[(row * workFrame->width) + column]
            >= threshold) {
                  xValues.push_back(column);
                  yValues.push_back(row);
            }
      }
}
```

Figura 15*: Excerto do Algoritmo Simples com Salto (A2)*

3.3 Algoritmo Simples de Detecção com Salto v2 – A3

As diferenças entre este algoritmo e o anterior visam reduzir o custo de processamento de cada imagem, em casos específicos, nomeadamente, quando existe um PI na *frame* e quando este não esteja na coordenada (*largura, altura*). O algoritmo começa a travessia da *frame*, usando como valor o salto que foi definido pelo utilizador. Quando é encontrado um píxel com valor superior ao *threshold*, interrompe a pesquisa e aplica nesta área um algoritmo para determinar quais as coordenadas do PI (ver Figura 16).

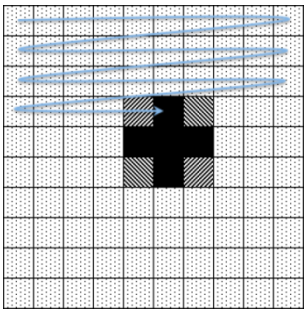


Figura 16: Primeiro Passo do Algoritmo com Salto v2 (A3)

Para encontrar o centro faz-se uma pesquisa nas quatro direcções possíveis, com origem no ponto encontrado. Primeiro pesquisa-se na horizontal, e determina-se o máximo e o mínimo nessa linha. Depois aplica-se uma pesquisa na vertical com origem no eixo dos *x*, o valor médio entre o mínimo e máximo encontrados anteriormente, e no eixo dos *y* o mesmo valor do ponto original. Com a mesma técnica, encontra-se um máximo e um mínimo na vertical. Os dois valores médios dão-nos o centro do PI (ver a Figura 17).

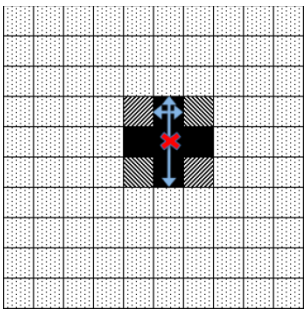


Figura 17: Segundo Passo do Algoritmo com Salto v2 (A3)

A implementação deste algoritmo é diferente dos anteriores, por incluirmos uma instrução de *return* dentro do nosso ciclo *for* (ver Figura 18). Isto é, não necessitamos de percorrer todos os píxeis presentes na imagem para retornar o centróide. Caso se encontre um píxel com valor superior ao *threshold* iremos tentar identificar o seu centróide, através da função *findCenter*. Se não existir nenhum píxel com um valor superior ao limiar, será retornado às coordenadas dum ponto fora da escala, isto é *(-1,-1)*.

```
for (int count = 0; count < max; count++) {
      if (workFrame->imageData[count] >= threshold) {
            int column = (int) (count % workFrame->width);
            int row = (int) (count / workFrame->width);
            coordinate temp = { column, row };
            coordinate center = findCenter(temp, workFrame);
            return center;
      }
}


coordinate center = { -1, -1 };
return center;
```

Figura 18: Excerto do Algoritmo com Salto v2 (A3)

A implementação do *findCenter* (ver Figura 19) é a imagem daquilo que foi explicado acima, tentando aproximar o centróide dum *BLOB* que seja de uma certa forma simétrico, como é o caso de um círculo, rectângulo, quadrado, entre outros. Primeiro, efectuando um varrimento na horizontal e, depois, a partir do valor médio encontrado entre o máximo e o mínimo na horizontal, ele irá efectuar um varrimento na vertical. O valor estimado do centróide é guardado na variável *center*.

```
int pointer = (temp.y * frame->width + temp.x);//adjust pointer
for (int p = pointer; p < ((temp.y + 1) * frame->width - 1)
&& frame->imageData[p] >= threshold; xMax++, p++);
for (int p = pointer; p > ((temp.y) * frame->width) && frame-
>imageData[p]>= threshold; xMin--, p--);

int distance = xMax - xMin;
center.x = temp.x + int(distance / 2);

pointer = (temp.y * frame->width + center.x);  //adjust pointer
for (int p = pointer; p < (frame->height + center.x - 1)
&& frame->imageData[p] >= threshold; yMax++, p += frame->width);
for (int p = pointer; p > (center.x) && frame->imageData[p] >=
threshold; yMin--, p-= frame->width);
distance = yMax - yMin;
center.y = temp.y + int(distance / 2);
```

Figura 19: Excerto do Algoritmo Centróide em Cruz

3.4 Algoritmo com Previsão e Pesquisa em Espiral – A4

Este algoritmo propõe uma abordagem diferente dos anteriores pois foca-se não tanto na detecção mas mais no seguimento de um PI. O algoritmo deverá apresentar melhores resultados quando um ponto se movimentar (operação de *drag* do dispositivo apontador) ao longo de imagens consecutivas. Usamos neste algoritmo um vector, que guarda a informação sobre o deslocamento, ou seja, a diferença de posições entre duas imagens consecutivas. Consideramos as seguintes três *frames* consecutivas: F1, F2 e F3. Todas contêm um PI em movimento, que tem por coordenadas, respectivamente, p1 = (*x1, y1*), p2 = (*x2, y2*) e p3 = (*x3, y3*). O valor do vector deslocamento do PI entre F1 e F2, Δs, é calculado segundo a fórmula 1.

$$\Delta s = (\Delta x, \Delta y) = (x2\text{-}x1, y2\text{-}y1) \qquad (1)$$

Podemos então reutilizar Δs, juntamente com uma pesquisa em espiral, para encontrar rapidamente as coordenadas do PI em F3. Como Δs tem o valor do último deslocamento, prevemos que este se mantenha em *frames* consecutivas, pressuposto que é válido a não ser que termine, abruptamente, a operação de arrastamento (*drag*). A estimação para as novas coordenadas do PI em F3 será calculada segundo a fórmula 2.

$$\hat{p}_3 = p2 + \Delta \boldsymbol{s} = (x2+\Delta x, y2+\Delta y) \qquad (2)$$

Com base nesta previsão, iremos iniciar a nossa pesquisa na *frame*, não no tradicional ponto (*0,0*), mas sim na previsão que acabamos de calcular. Usamos um mecanismo de espiral com a finalidade de aproximar a solução e de encontrar um ponto com valor superior ao *threshold*, o mais rapidamente possível. O mecanismo para pesquisa em espiral tradicional seria muito intensivo no uso do CPU, por isso, optou-se por pré-construir uma tabela (*lookup-table*) com os deslocamentos prévios em *x* e em *y*, definindo o movimento de uma espiral como se pode constatar na Figura 21.

No funcionamento global deste algoritmo, descrito na Figura 20, conseguimos identificar duas posições anteriores de PI em frames precedentes e, com base nessas, calculamos a previsão e efectuamos uma pesquisa em espiral.

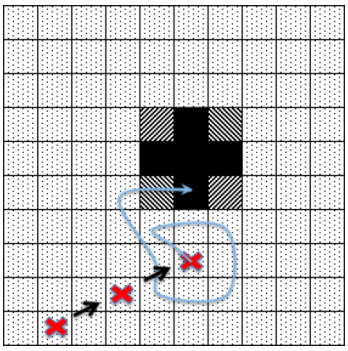


Figura 20: Ilustração do Funcionamento do Algoritmo em Espiral (A4)

Verificamos que quando um ponto desaparece, a expansão da espiral a toda a imagem pode revelar-se ineficiente, quando comparada com o método de pesquisa linear. Os motivos estão relacionados com o armazenamento em memória da imagem. Numa pesquisa em espiral efectuamos repetidamente vários saltos entre zonas de memória, enquanto que numa pesquisa linear analisamos zonas de memória consecutivas, permitindo que o processo seja desempenhado mais rapidamente. Por este motivo, decidimos limitar a sua expansão com um parâmetro que corresponde ao número máximo N, isto é, à profundidade permitida de expansão da espiral. Quando o algoritmo não encontra nada no limite da espiral e ainda permanece uma área da imagem por percorrer, recorremos a uma abordagem de pesquisa linear, a fim de verificar a presença ou ausência de um PI nos restantes píxeis.

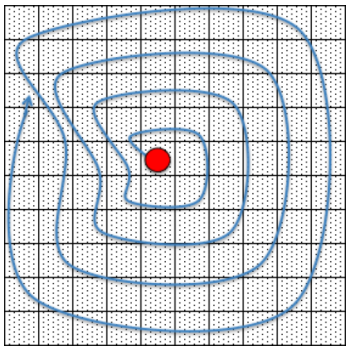


Figura 21: Descrição do Movimento da Espiral

No que se refere à implementação deste algoritmo, tivemos diferentes versões, sempre com o intuito de melhorar o desempenho do mesmo. A versão final que atingimos, propõe na fase de inicialização da aplicação, processar uma *lookup-table*. Esta, irá conter os valores correspondentes às deslocações em *x* e *y*, a aplicar num determinado ponto de referência para obter um movimento em espiral. A tabela é construída de forma a respeitar o movimento da espiral, descrito na Figura 21. Contudo, é possível aplicar alternativas. Uma delas consiste em acrescentar um factor salto entre os pontos da espiral, evitando percorrer consecutivamente todas as possibilidades contínuas da espiral. Este factor salto, não deve, de forma alguma, ser superior ao tamanho do *BLOB*, para não correr o risco da pesquisa em espiral não ser bem sucedida. Outra técnica a que se pode recorrer, mas que não se encontra de momento implementada, é a do movimento em espiral usando fórmulas trigonométricas (*Cosinus* e *Sinus*). A *lookup-table* é construída através da função *Distance* (ver Figura 22), na qual podemos enviar como parâmetro um *Integer* que corresponde ao número de níveis da espiral que queremos pré-processar. O *jumpValue* corresponde ao factor salto que pode ser aplicado. Por defeito o valor deste factor é *1*.

```
do {  for (i = seX; i <= idX; i += jumpValue) {
            c.x = i;
            c.y = seY;
            vDistance.push_back(c);
      }
      for (i = seY + 1; i <= idY; i += jumpValue) {
            c.x = idX;
            c.y = i;
            vDistance.push_back(c);
      }
      for (i = idX - 1; i >= seX; i -= jumpValue) {
            c.x = i;
            c.y = idY;
            vDistance.push_back(c);
      }
      for (i = idY - 1; i > seY; i -= jumpValue) {
            c.x = seX;
            c.y = i;
            vDistance.push_back(c);
      }
      seX--;
      seY--;
      idX++;
      idY++;
      circles++;
} while (circles <= max);
```

Figura 22: Excerto do Algoritmo de construção da Lookup-Table

Ao iniciar o algoritmo, verifica-se que é possível executar uma previsão, no entanto, esta situação nem sempre é válida. Para isso, ele primeiramente precisa que exista um ponto presente na última imagem. Do mesmo modo, será necessário que exista já um Vector previsão Δs válido. Logo, será necessário que nas duas últimas imagens, antes da que será agora analisada, existam dois pontos de interesse, um em cada imagem. Caso não exista nenhuma previsão possível, o algoritmo irá optar por efectuar um varrimento linear, como visto nos algoritmos anteriores. Por outro lado, se for possível ter uma previsão viável do próximo ponto na imagem a ser analisada, ele irá efectuar uma pesquisa na presente imagem, tendo por base o valor da previsão e recorrendo a *lookup-table* para as deslocações. A pesquisa em espiral é feita na função *spiral2* (ver Figura 23), que recebe nos seus parâmetros a imagem a analisar e a previsão. A função *spiral* corresponde à primeira versão da implementação não recorrendo a *lookup-table*. Durante a execução, esta vai processando as coordenadas do próximo ponto a explorar.

```
for (int i = 0; i < (int) vDistance.size(); i++) {
      int pointer = (y + vDistance.at(i).y) * workFrame->width + (x
            + vDistance.at(i).x);
      if (workFrame->imageData[pointer] >= threshold) {
            temp.x = (int) pointer % workFrame->width;
            temp.y = (int) pointer / workFrame->width;
            return temp;
      }
}
temp = singlePointProcessPTR(workFrame, step);
return temp;
```

Figura 23: Pesquisa em Espiral – Spiral v.2

Neste caso, o algoritmo não toma em consideração valores como a aceleração entre pontos consecutivos encontrados anteriormente. Poderá ser uma mais-valia em próximas versões considerar este parâmetro, podendo usá-lo na estimativa da previsão e consequentemente diminuindo o tempo da pesquisa e respectivo custo de processamento. Salientam-se dois aspectos na Figura 23, por um lado quando é encontrado um ponto com valor superior ou igual ao *threshold*, este será processado *a posteriori* pela função *findCenter* (ver Figura 19), para determinar qual o centróide do *BLOB* identificado. Por outro lado, quando não é encontrado nenhum PI na expansão da espiral, limitada a N níveis, deverá ser executado um dos algoritmos anteriores da pesquisa linear, de forma eficiente, para identificar se existe ou não, algum PI na restante área da imagem.

3.5 Algoritmo Multiponto – A5

Este algoritmo permite, num simples varrimento linear da imagem, recolher informação sobre todos os pontos de presentes, como podemos ver na Figura 24.

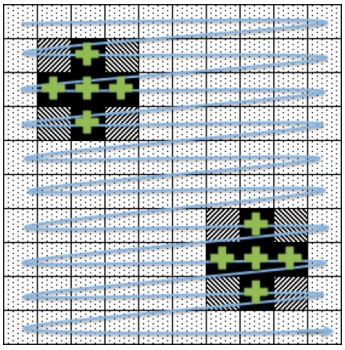


Figura 24: Travessia e Captura dos PI no Algoritmo Multiponto (A5)

De seguida, o algoritmo analisa os valores recolhidos e determina diferentes grupos de forma a saber quantos pontos tem em cada imagem. Por fim, investiga estes grupos para conhecer quais são as coordenadas dos seus centróides, como é ilustrado na Figura 25. O algoritmo descrito foi adaptado do algoritmo relatado por Erik van Kempen (Kempen 2008). O autor, em (Kempen 2008), salienta a eficiência temporal desta técnica usada para o reconhecimento de vários pontos de interesse, num só varrimento de imagem.

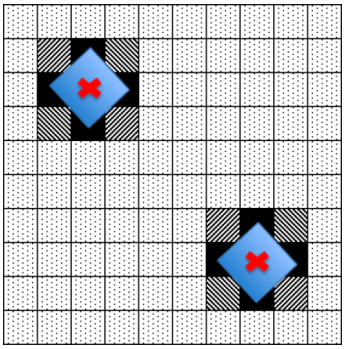


Figura 25: Definição dos Grupos e Processamento dos Centróides

No que se refere à implementação deste algoritmo, podemos dizer que se encontra dividida em três fases. A primeira consiste na análise da *frame* e na recolha de todas as coordenadas dos pixéis com valor superior ao *threshold* (ver Figura 26).

```
for (int row = 0; row < frame->height; ++row) {
    for (int column = 0; column < frame->width; ++column) {
      unsigned char byte = (unsigned char) frame-
>imageData[(row*frame->width) + column];
      if (byte >= threshold) {
            int start = column;
            for (; byte >= threshold; byte=(unsigned char)frame-
>imageData[(row * frame->width)+column],++column);
            int stop = column - 1;
            lineBlob lineBlobData = { start, stop, blobCounter, false
};
            imgData[row].push_back(lineBlobData);
            blobCounter++;
}}}
```

Figura 26: Primeira Fase do A5 [Kempen, E. V., 2008]

```
for (int row = 0; row < (int) imgData.size(); ++row) {
    for (int entryLine1=0;entryLine1<(int) imgData[row].size();
++entryLine1){
      for (int entryLine2 = 0; entryLine2 < (int) imgData[row +
1].size(); ++entryLine2) {
            if (!((imgData[row][entryLine1].max < imgData[row +
1][entryLine2].min) || (imgData[row][entryLine1].min> imgData[row +
1][entryLine2].max))) {
                  if (imgData[row + 1][entryLine2].attached == false) {
                        imgData[row + 1][entryLine2].blobId=
imgData[row][entryLine1].blobId;
                        imgData[row + 1][entryLine2].attached = true;
                  } else{
                        imgData[row][entryLine1].blobId= imgData[row +
1][entryLine2].blobId;
                  imgData[row][entryLine1].attached = true;
}}}}}
for (int row = 0; row < (int) imgData.size(); ++row) {
    for (int entry = 0; entry < (int) imgData[row].size(); ++entry) {
      if (blobs.find(imgData[row][entry].blobId) == blobs.end()) {
            coordinate min = { imgData[row][entry].min, row };
            coordinate max = { imgData[row][entry].max, row };
            coordinate center = { 0, 0 };
            blob blobData = { min, max, center };
                  blobs[imgData[row][entry].blobId] = blobData;
      } else {
      if (imgData[row][entry].min<
blobs[imgData[row][entry].blobId].min.x)
            blobs[imgData[row][entry].blobId].min.x=
imgData[row][entry].min;
      else if (imgData[row][entry].max>
blobs[imgData[row][entry].blobId].max.x)
            blobs[imgData[row][entry].blobId].max.x=
imgData[row][entry].max;
            if (row < (int) blobs[imgData[row][entry].blobId].min.y)
                  blobs[imgData[row][entry].blobId].min.y = row;
            else if (row > (int)
blobs[imgData[row][entry].blobId].max.y)
                  blobs[imgData[row][entry].blobId].max.y = row;}}}
```

Figura 27: Segunda Fase do A5 [Kempen, E. V., 2008]

Numa segunda fase, o algoritmo analisa todos os pontos que foram recolhidos, e determina se esses são constituintes do mesmo *BLOB* ou PI. Para isso, verifica os pontos presentes em linhas ou colunas juntas. Podemos comparar esta fase a um agrupamento das coordenadas, que foram anteriormente recolhidas (ver Figura 27).

Por último, o algoritmo identifica o centróide de cada um dos *BLOBs* reconhecidos na imagem (ver Figura 28). A *Macro MIN_SIZE* designa um valor predefinido para o tamanho mínimo que cada um dos *BLOB* deverá ter para ser considerado como possível PI. O centróide que corresponde a cada um dos PI tem as suas coordenadas armazenadas na *Struct Coordinate*. Aqui, o centróide está armazenado na variável *temp*.

```
for (map<unsigned int, blob>::iterator i = blobs.begin(); i != 
blobs.end(); ++i) {
    (*i).second.center.x = (*i).second.min.x + ((*i).second.max.x-
(*i).second.min.x) / 2;
    (*i).second.center.y = (*i).second.min.y + ((*i).second.max.y-
(*i).second.min.y) / 2;
    int size = ((*i).second.max.x - (*i).second.min.x) * 
((*i).second.max.y - (*i).second.min.y);
    cleanVectorResults();
    if (size > MIN_SIZE) {
      coordinate temp = { (*i).second.center.x, (*i).second.center.y
};
      results.push_back(temp);
    }
}
```

Figura 28: Terceira Fase do A5 [Kempen, E. V., 2008]

Este algoritmo poderá revelar-se importante, caso queiramos trabalhar com uma aplicação que possa fornecer suporte para multi-toque. Esta não foi uma das características a que nos propusemos inicialmente, na implementação do quadro interactivo, contudo será sempre possível ver através da implementação deste algoritmo, uma linha de trabalho futuro, permitindo a interacção de diversos usuários ao mesmo tempo no sistema. Deverão do mesmo modo ser repensadas as aplicações, tomando o exemplo das aplicações demonstrativas criadas para multi-toque (Tbeta 2008), hoje existentes, isto devido ao facto dos sistemas operativos ainda não fornecerem um suporte nativo para o multiponto. O único ponto que usamos é o associado ao rato, contudo existem já implementações para de utilizadores, nomeadamente em Linux, que permitem controlar

mais do que um rato no ambiente de trabalho, por exemplo, o sistema Multi-Cursor (Kai Li's Research Group & Wallace n.d.).

A integração da funcionalidade multi-toque envolve repensar as aplicações tradicionais, e alterar os seus comportamentos com a presença de dois ou mais pontos de interacção por parte do utilizador. Este tipo de interfaces, têm um funcionamento similar ao que verificamos nas novas interfaces propostas nos dispositivos móveis como, por exemplo, o iPhone, o qual permite, a partir de dois pontos, proceder a acções como Zoom, Rodar, etc. A ideia a reter, é que aplicações multi-toque, são mais simples do ponto de vista da interacção com o utilizador, mas requerem para o programador, uma análise mais exaustiva dos eventos, a fim de desencadearem determinadas acções. O trabalho de pesquisa e desenvolvimento por parte do programador será muito mais extenso devendo pensar e prever quais serão as acções ou gestos possíveis e permitido ao utilizador um maior nível de interacção com sua aplicação.

3.6 Síntese dos Algoritmos

Apresentamos na Tabela 5, podemos ver uma síntese dos algoritmos de detecção e seguimentos anteriormente descritos.

Tabela 5: Diferenças entre as características dos Algoritmos de Detecção

Características / *Algoritmos*	***Detecção PI***	***Processamento Centróide***	***Quantidade PI***
A1	Varrimento Linear	Sistema de Médias	PI Único
A2	Varrimento Linear com Salto	Sistema de Médias	PI Único
A3	Varrimento Linear com Salto	Sistema em Cruz	PI Único
A4	Varrimento em Espiral	Sistema em Cruz	PI Único
A5	Varrimento Linear	Sistema Adaptado a Grupos	Multiponto

Foram testados diversos algoritmos, a fim de avaliar e estudar os seus diferentes comportamentos, durante a execução da aplicação. A sequência dos quatro primeiros algoritmos foi criada seguindo uma lógica de optimização. O último algoritmo prevê uma evolução, de forma a permitir um suporte no futuro a outros tipos de aplicações, que sejam do género multi-toque. De forma geral, são comparados diferentes tipos de varrimento linear, desde um varrimento linear simples a um composto por saltos. Também se analisa um varrimento em espiral no algoritmo A4. Do mesmo modo, são usados dois modelos de processamento de centróide diferentes, um que define a média através da recolha de pontos (ver Figura *14*) e outro procurando o centróide através de um varrimento em cruz (ver Figura 19).

O próximo capítulo descreve mais detalhadamente a implementação prática do quadro interactivo, no qual estes algoritmos detêm uma importância vital.

Capítulo 4: Implementação do Sistema LoCoBoard

Neste capítulo, apresentamos inicialmente a metodologia usada e os resultados que foram obtidos, seguindo-se uma descrição global da arquitectura e funcionalidade da ferramenta de suporte do quadro interactivo que foi desenvolvido. Serão também descritos os três blocos restantes do ciclo de processos da Figura 1 (Captura Sinal de Vídeo; Pré-processamento da Frame e Reportar as coordenadas).

Seguiu-se uma metodologia essencialmente experimental, que conduziu este trabalho a uma divisão em duas partes. A primeira parte, assente numa revisão bibliográfica que permitiu compreender as noções básicas e os conceitos da ciência *Computer Vision*. Esta análise foi fundamental para a concepção da aplicação LoCoBoard. A segunda parte, baseada na elaboração de um protótipo, ilustra a aplicação dos conhecimentos teóricos que foram estudados. Esta aplicação foi concebida usando um modelo iterativo de prototipagem, primeiro efectuando um trabalho de análise, depois o desenvolvimento e por fim a realização de testes de validação para encontrar os pontos que poderiam ser melhorados ou corrigidos, refinando-se assim o processo. O presente capítulo foca-se essencialmente na segunda parte da investigação, isto é, a concepção e implementação da aplicação.

A revisão bibliográfica inicial permitiu identificar os algoritmos de detecção de pontos de interesse utilizados na ferramenta. A implementação destes algoritmos foi efectuada em C++ e com recurso a duas bibliotecas: o OpenCV (para adquirir e manipular as imagens capturadas) e o QT (para dotar a aplicação de portabilidade, i.e., com funcionalidade *cross-platform*, independentemente do ambiente de execução).

Inicialmente concebeu-se então uma plataforma de recolha de PIs descrita anteriormente, cujos resultados de implementação foram publicados e nos deram confiança e estabilidade da ferramenta (Soares et al. 2009). Esta plataforma foi aberta à comunidade e poderá ser utilizada por outros programadores no desenvolvimento de outras aplicações. No nosso caso, utilizamos esta ferramenta no desenvolvimento de um Quadro Interactivo, que permite de forma simples e eficaz controlar a interacção com um PC através de acções de uma caneta IR sobre as imagens projectadas.

4.1 Descrição Global do Sistema

O Quadro Interactivo desenvolvido é composto por software de código aberto que pode ser instalado num computador vulgar. Este software utiliza uma câmara Web normal (ou qualquer outro tipo de câmara genérica) e detecta dispositivos apontadores que emitam luz infravermelha reflectida sobre uma superfície de projecção (e.g., quadro ou parede branca). A projecção do ecrã do computador pode ser efectuada por um videoprojector simples. A robustez e a simplicidade de utilização foram dois requisitos importantes para que o sistema fosse facilmente utilizado por pessoas sem conhecimentos de informática ou programação.

Para isolar a câmara de fontes de luz parasitas, colocamos à frente da lente um filtro de luz que só deixa passar infravermelhos (e.g., uma película fotográfica dobrada várias vezes). Usamos, como dispositivo apontador, uma caneta com um LED IR montado na extremidade, alimentada por uma pilha acomodada no seu interior.

O funcionamento do sistema inicia-se com uma fase de calibração, para estabelecer um factor escala entre a resolução do quadro e a do computador e podermos assim mapear os movimentos dos PI detectados em movimentos do cursor. O sistema tal como foi descrito permite-nos interagir com o computador, através dos deslocamentos do apontador de IR na parede de projecção, tal como se fosse um rato. O sistema detecta a posição e interpreta a acção que o utilizador pretende efectuar num determinado ponto de projecção.

Podemos ver, no diagrama de blocos da Figura 1, os quatro processos mais importantes efectuados pelo sistema. O primeiro corresponde à aquisição de uma imagem feita por parte da câmara. No segundo aplicam-se filtros à imagem capturada para melhorar a qualidade da mesma; opcionalmente, nesta fase, podem ainda utilizar-se técnicas de remoção de fundo – *background* - para melhorar a qualidade da imagem recolhida pela câmara (com base em funções OpenCV). Estas permitem-nos construir um modelo do *background*, que podemos subtrair a cada nova imagem, obtendo como resultado apenas o *foreground* de cada imagem. Numa terceira fase utilizam-se os algoritmos de detecção e seguimento, abordados mais à frente. Por fim, reportam-se as coordenadas do ponto encontrado ou, caso este não exista, retorna-se o ponto (*-1,-1*).

As secções seguintes abordam detalhadamente cada um dos módulos presentes na Figura 1. Relembra-se apenas que o módulo de detecção e *tracking* de BLOBs na frame já foi apresentado no Capitulo 3.

4.2 Captura de Sinal de Vídeo

Este é o primeiro módulo do nosso diagrama de processos e, tal como os restantes, é essencial ao funcionamento da aplicação. É este, que faz a ponte entre o quadro interactivo e o dispositivo de captura, ou seja, a câmara Web. Esta transição e comunicação são asseguradas pela biblioteca OpenCV. Recorrendo aos métodos de *input* e *output* de vídeo disponibilizados na biblioteca HighGUI do OpenCV, iremos numa primeira fase inicializar o nosso dispositivo de captura (ver Figura 29).

```
capture = cvCreateCameraCapture(CV_CAP_ANY);
```

Figura 29: Inicialização da Captura

O método *cvCreateCameraCapture* devolve um apontador do tipo *CvCapture*, como vemos na Figura 29 (Bradski & Kaehler 2008, p.26 e p.103). Caso este seja diferente de *0*, poderemos iniciar a aquisição de imagens. Existem diferentes tipos de parâmetros, que podem ser enviados à função *cvCreateCameraCapture*. Estes parâmetros dependem do meio pelo qual a câmara se encontra ligada ao computador, o qual é

conhecido por *Camera Domain*. O *CV_CAP_ANY* é usado por defeito quando se usa uma única câmara. Este parâmetro serve para indicar onde é que a biblioteca deve encontrar a câmara. A Tabela 6 apresenta os diferentes valores possíveis e o seu significado. Passando o valor *-1* para a função obriga a aplicação a abrir uma janela com a lista das câmaras disponíveis. Nesta lista, poderemos escolher a câmara desejada.

Tabela 6: Domínios da Câmara [Bradski & Kaelher 2008, p.103]

Camera capture constant	Numerical value
CV_CAP_ANY	0
CV_CAP_MIL	100
CV_CAP_VFW	200
CV_CAP_V4L	200
CV_CAP_V4L2	200
CV_CAP_FIREWIRE	300
CV_CAP_IEEE1394	300
CV_CAP_DC1394	300
CV_CAP_CMU1394	300

Antes de prosseguir com a aplicação, aconselha-se a realização do teste da condição descrita na Figura 30. Caso esta condição seja válida, devemos verificar os parâmetros do método de inicialização da captura.

```
if (!capture) {
    fprintf(stderr, "Could not initialize capturing...\n");
    exit(-1);
}
```

Figura 30: Condição de Teste da Captura

É possível, após ter inicializado o dispositivo de captura, alterar as suas definições, como, por exemplo, a taxa de captura (*FPS*) ou a resolução (*Height*, *Width*), entre outros. A função *cvSetCaptureProperty* possui uma lista das possíveis definições (Bradski & Kaehler 2008, p.104). Contudo, estas funcionalidades não estão disponíveis nos sistemas Mac OS X. Listam-se na Figura 31 algumas das funções que serão usadas com uma directiva de pré-processamento para ambiente Windows. O primeiro parâmetro corresponde a um apontador do tipo *CvCapture*, que foi previamente inicializado; o segundo parâmetro é uma *Macro* que corresponde a um valor *Integer*, referente à pro-

priedade que queremos alterar; por último, temos o novo valor, com o qual queremos predefinir a propriedade de captura. Pode-se revelar importante alterar estes valores, por exemplo, no caso de termos uma câmara com uma resolução máxima muito elevada, isto irá ter um impacto no tamanho da imagem capturada e no número de píxeis que esta contém. Assim sendo, o custo de processamento do algoritmo de detecção é tanto maior quanto maior for o número de píxeis a percorrer. Podemos desta forma, concluir que se não controlarmos o tamanho máximo das imagens adquiridas, podemos sobrecarregar o CPU em demasia. Mas por outro lado, uma resolução melhor permite conseguir uma menor taxa de erro na detecção dos pontos de interesse.

Do mesmo modo, será necessário determinar um equilíbrio na taxa de amostragem adequada, não devendo ser muito baixa para se conseguir fazer o seguimento (*tracking*) do ponto de interesse de forma nítida, mas também não deverá ser muito alta, para não se sobrecarregar o CPU.

```
cvSetCaptureProperty(capture,CV_CAP_PROP_FRAME_WIDTH,captureWidth);
cvSetCaptureProperty(capture,CV_CAP_PROP_FRAME_HEIGHT,captureHeight);
cvSetCaptureProperty(capture,CV_CAP_PROP_FPS, captureFPS);
```

Figura 31: Definição dos Parâmetros de Captura

O último passo corresponde à aquisição de uma imagem através da função *cvQueryFrame* (ver Figura 32), a qual nos devolve um apontador para uma imagem do tipo *IplImage (Bradski & Kaehler 2008, p.19)*. Esta função é usada repetidamente, para se capturar sempre a imagem seguinte do fluxo de vídeo, integrada num ciclo do género *for(;;)* ou *while(1)*.

```
IplImage* frame = cvQueryFrame(capture);
```

Figura 32: Função de Aquisição de Imagens

Aconselha-se o uso da condição de controlo apresentada na Figura 33, para detectar o final do vídeo ou a captura da última imagem válida, uma vez que a função retorna o valor *0* quando não consegue capturar uma nova imagem do dispositivo.

```
if (!frame) {
    printf("\nProgram Fail in image acquisition! bad video (Initialize
Method) \n");
    exit(0);
}
```

Figura 33: Condição de Teste da Aquisição

Por fim, é importante salientar que se deve acrescentar a função *cvWaitKey* (ver Figura 34) no final do ciclo *for(;;)* ou do *while(1) (Bradski & Kaehler 2008, p.95)*. Esta função tem por objectivo, obrigar a aplicação a esperar um número N de milissegundos, à semelhança de um *timeout*. Este número "N" corresponde ao valor passado em parâmetro na chamada de função *cvWaitKey*. Da mesma forma, esta função permite ficar à escuta duma tecla que possa ser premida, retornando, nesse caso, o código *ASCII* correspondente. `int keyPressValue=cvWaitKey(100);  // Timeout 100 ms`

```
if (keyPressValue == 27) {exit(0);}      // Char Escape = 27
```

Figura 34: Função de Timeout

Usa-se essa função, para controlar algumas funcionalidades da aplicação através do teclado como, por exemplo, a finalização da aplicação, que é feita através da verificação da tecla *Escape*.

4.3 Pré-processamento da *Frame*

O pré-processamento da frame capturada corresponde à segunda fase do diagrama de blocos do quadro interactivo. Após ter capturado uma imagem, não podemos de imediato trabalhar com esta sem efectuar um tratamento prévio. Teoricamente seria possível, contudo, na prática, muitas vezes estamos sujeitos a condições do ambiente que não controlamos, como é o caso da captura de imagens num ambiente com muito ruído (e.g., muitas fontes de luz). Embora se use um filtro que só deveria deixar passar luz do tipo IR, muitas vezes há fontes de ruído presente no ambiente (incluindo luz infravermelha, para além da gerada pelo dispositivo apontador), perturbando o bom funcionamento dos algoritmos de detecção. A fim de melhor detectar os Pontos de Interesses, evitando confundi-los com o ruído presente na sala, usou-se um modelo de *background* e *foreground* em cada uma das imagens. Este modelo de imagem de *back-*

ground é adaptativa, isto é, à medida que a aplicação vai sendo executada ele vai aprendendo, detectando e actualizando o modelo de *background*.

Posteriormente efectuamos a subtracção da imagem capturada ao modelo de *background* estimado, obtendo desta forma uma nova imagem constituída somente pela luz IR. A utilização desta técnica, que denominamos *Background Subtraction*, constitui um custo acrescido no uso do CPU e da memória, contudo revelou-se essencial ao bom funcionamento da aplicação em ambientes com fontes de ruído externo. É possível instanciar o nosso modelo escolhendo apenas uma das seguintes duas opções ilustradas na Figura 35, embora tenham sido testados os dois modelos durante a fase de implementação.

```
bg_model = cvCreateGaussianBGModel(frame);
bg_model = cvCreateFGDStatModel(frame);
```

Figura 35: Criação de um Modelo

Será importante realçar que, embora tivessem sido testados dois modelos de background, estes não foram usados ao mesmo tempo. O modelo depois de criado é actualizado através da função *cvUpdateBGStatModel*, (ver Figura 36), passando nos parâmetros uma nova *frame* (que descreve o estado actual do ambiente) e um apontador (para o modelo que pretendemos actualizar).

```
cvUpdateBGStatModel(frame, bg_model);
```

Figura 36: Actualização do Modelo

Depois de ter um modelo de *background* actualizado, obtemos a imagem resultante da subtracção da mesma ao modelo. A função *cvAbsDiff* permite-nos obter o valor absoluto da diferença entre dois *array* de imagens *(Bradski & Kaehler 2008, p.50)*. O modelo de background estimado encontra-se na estrutura *bg_model->background* (consulte a Figura 37).

```
cvAbsDiff(finalFrame, bg_model->background, finalFrame);
```

Figura 37: Diferença Absoluta entre Imagens

Depois de obter uma imagem sem o *background*, usamos a função *cvTreshold* para

percorrer o *array* da imagem e, através de um *threshold*, alterar os valores dos píxeis dependendo das suas posições no *array* (contínuos) e dos seus valores (acima ou abaixo do *threshold*) (Bradski & Kaehler 2008, p.135). O principal objectivo é uniformizar a imagem limpando e retirando píxeis de valores insignificantes para a aplicação. A Figura 38 mostra a forma de evocar o método *cvThreshold*.

```
cvThreshold(finalFrame, finalFrame, value, 255, CV_THRESH_BINARY);
```

Figura 38: Função de Threshold

Existem diferentes tipos de *thresholds* possíveis de aplicar como podemos verificar na Tabela 7. Segundo (Bradski & Kaehler 2008, p.135), cada tipo de *Threshold* corresponde a uma comparação particular entre o i-ésimo píxel fonte (src_i) e o valor de *Threshold* (*T*). Dependendo da relação entre o src_i e *T*, o píxel de destino dst_i, poderá ter o valor *0*, src_i ou o valor máximo (*M*).

Tabela 7: Diferentes tipos de Thresholds [Bradski & Kaelher 2008, p.136]

Threshold type	Operation
CV_THRESH_BINARY	$dst_i = (src_i > T) \; ? \; M : 0$
CV_THRESH_BINARY_INV	$dst_i = (src_i > T) \; ? \; 0 : M$
CV_THRESH_TRUNC	$dst_i = (src_i > T) \; ? \; M : src_i$
CV_THRESH_TOZERO_INV	$dst_i = (src_i > T) \; ? \; 0 : src_i$
CV_THRESH_TOZERO	$dst_i = (src_i > T) \; ? \; src_i : 0$

Contudo, as experiências efectuadas permitiram-nos concluir que em certos casos o modelo de *background* podia não corresponder à realidade. Isto depende da frequência com que actualizamos o nosso modelo (*bg_model*) porque dele depende a imagem de *background*. Por um lado, se treinarmos frequentemente este modelo, iremos obter uma melhor aproximação da realidade, mas sobrecarregando o CPU. Também corremos o risco, do ambiente se alterar de forma inesperada, por exemplo, estando a interagir com uma ferramenta como o *Paint*, alterar o fundo da imagem de preto para branco. Neste caso, o modelo de *background* não irá imediatamente aprender esta variação de estado. Esta aprendizagem irá ser progressiva, daí termos que pensar em alternativas. Uma solução foi calcular um factor através de uma métrica, representativa de uma

imagem, como podemos ver na Figura 40. Cada ponto vermelho identifica um valor que irá ser usado para calcular esse factor. Entre *frames* consecutivas calculámos esse factor que corresponde à média da luminância dos píxeis. Refira-se que os píxeis não são aleatórios, isto é, ao longo da execução do programa serão sempre comparadas entre cada imagem as médias dos mesmos píxeis. Podemos ver abaixo a função *returnMetricValue* (ver Figura 39), responsável pelo cálculo da métrica. Foi criado um vector *testingCoordinates*, com os valores predefinidos dos píxeis a processar, a fim de optimizar a sua execução.

```
float Algorithms::returnMetricValue(IplImage * frame)
{
    float temp=0;
    for(int i =0; i<testingCoordinates.size();i++)
    {
      temp=temp+frame->imageData[testingCoordinates.at(i)];
    }
    printf("\nTemp: %f\n",temp);
    return (temp/testingCoordinates.size());
}
```

Figura 39: Função de Processamento da Métrica

Deste modo, podemos ter uma ideia da variação da imagem, embora esta não seja completamente eficaz porque a métrica usa apenas uma amostra de toda a imagem. Baseando-nos nesta métrica, caso exista uma grande variação entre medidas consecutivas (ver Fórmula 3), iremos reiniciar o nosso modelo *bg_model*. Assim, conseguimos de forma mais rápida, ter um novo modelo de *background*, mais fiel em relação à realidade, do que se esperássemos pela actualização do modelo anterior. Este novo modelo, comporta-se como o anterior e continuará a ser actualizado até que o sistema se aperceba de uma nova variação da métrica e decida então reiniciar o modelo. Contudo, é conveniente não escolher um valor de variação de métrica muito pequeno, senão iremos estar constantemente a reiniciar o nosso *background*. A reinicialização do *background* é feita instanciando de novo o nosso apontador *bg_model*, com o método correspondente ao modelo que usamos, enviando por parâmetro a nova imagem inicial.

Consideramos as duas seguintes *frames* consecutivas: F1 e F2. Ambas contêm uma métrica, que tem como valor, respectivamente, *m1* e *m2*. Esses valores foram cal-

culados segundo o método apresentado na Figura 39. O valor da variação da métrica entre F1 e F2, Δm, é calculado segundo a fórmula (3).

$$\Delta m = | m1\text{-}m2 | \qquad (3)$$

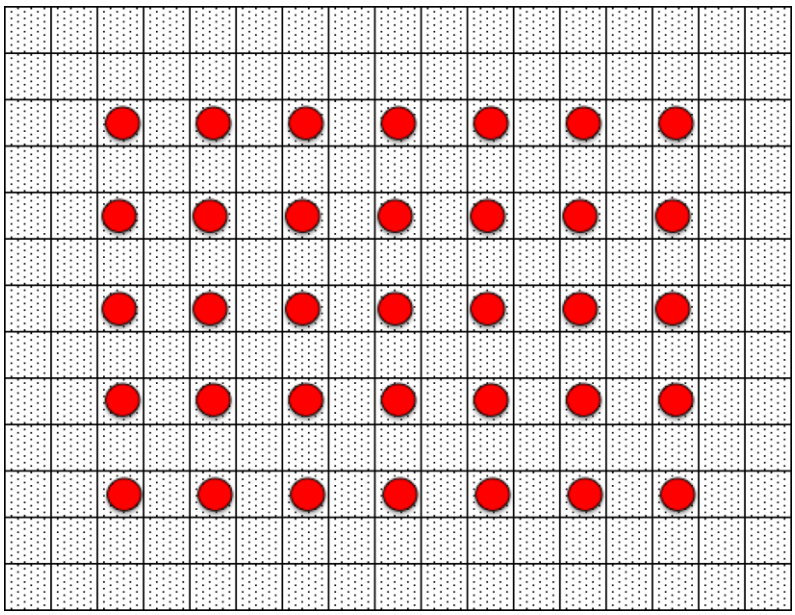


Figura 40: Processamento de uma Métrica por Imagem

Por outro lado, nesta segunda fase, aplicaram-se transformações ao nível da definição da imagem, isto é, a imagem capturada através do método *cvQueryFrame* é uma imagem na norma RGB. A fim de acelerar o processamento do nosso algoritmo é preferível ter um único valor por píxel, do que três valores. No caso de trabalharmos com luz infravermelha, não é necessário distinguir diferenças de cor, daí termos optado por converter as imagens todas para a escala de cinzento. Desta forma, temos um valor que corresponde a uma métrica da intensidade da luz por cada píxel da imagem. Através do método *cvCvtColor (Bradski & Kaehler 2008, p.58),* podemos converter uma imagem numa escala de cor para outra (ver Figura 41), aqui pretendemos a conversão da escala *RGB* para a escala de cinza (*Grayscale*), daí recorrermos ao parâmetro *CV_BGR2GRAY*. É importante salientar, que a imagem na escala de cinza tem de ser criada escolhendo como último parâmetro o valor *1*, o qual corresponde ao número de canais de cor. Por analogia, percebemos que a *originalFrame* tinha o valor *3*, correspondente aos três canais compostos por *Red*, *Green* e *Blue*.

```
workFrame = cvCreateImage(cvSize(frame->width, frame->height),
            IPL_DEPTH_8U, 1);
cvCvtColor(originalFrame, workFrame, CV_BGR2GRAY);
```

Figura 41: Definição e Conversão de uma Imagem para Grayscale

Podem ser aplicadas outras transformações nesta fase, contudo estas são opcionais e deve-se ter cuidado no seu uso, sendo que não são essenciais para o bom funcionamento da aplicação, podem porém contribuir para algumas melhorias. Cada transformação aplicada às imagens representa um custo de CPU acrescido. Quando estamos a lidar com aplicações em tempo real, este factor deve ser tomado em consideração, porque pode introduzir uma taxa de atraso, criando a sensação aos utilizadores que a aplicação está desfasada ou mesmo lenta.

Algumas das transformações possíveis são o espelhamento vertical ou horizontal. Dependendo da posição da câmara, podemos pretender aplicar um *Flip* (ver Figura 42), a fim, de que a imagem com a qual o algoritmo de detecção se encontra a processar, seja de acordo com o que pretendemos, fidedigna à situação real. Denotamos esse erro, quando o sistema começa a reportar coordenadas de pontos, que são o oposto das que pretendemos. Tomando o exemplo do quadro interactivo, podemos ter o apontador de infravermelhos no canto superior direito e o sistema assimilar que se encontra no canto inverso e posiciona-nos o rato no canto superior esquerdo, ou seja, o sistema assimilou o contrário da realidade. Neste caso, devemos considerar que a imagem vista pela câmara está a ser invertida da realidade, logo dever-se-á aplicar um *flip* na vertical, a fim de resolver a situação.

Tabela 8: Parâmetro da Função cvFlip

Espelhamento / *Valor*	***Eixos***
0	Eixos-X
Valor Positivo	Eixos-Y
Valor Negativo	Ambos os Eixos

A seguinte função *cvFlip* (ver Figura 42), no seu último parâmetro, que aqui é designado por *FLIP*, corresponde a um Integer, que define os eixos de espelhamento (Bradski & Kaehler 2008, p.61), como podemos ver na Tabela 8.

```
cvFlip(finalFrame, finalFrame, FLIP);
```

Figura 42: Função Espelho de Imagens

Outra transformação possível é o *Smoothing*, também conhecido por *Blur*. Esta função é normalmente usada para reduzir o ruído presente nas imagens (Bradski & Kaehler 2008, p.109). Esta função, pode ser evocada com a função *cvSmooth* (ver Figura 43), passando como elemento o parâmetro o *Blur* que pretendemos efectuar. Sendo que esta aplicação irá funcionar em tempo real a nossa escolha foi optar por um *blur* simples. Este, altera o valor de um píxel consoante os valores dos seus píxeis vizinhos, desta forma uniformiza uma imagem. No nosso caso, isso pode ser importante, suponhamos na aparição de pontos de luz com tamanhos pequenos, cerca de poucos píxeis, caso os seus vizinhos tenham valores próximos de *0* a função cvBlurr irá também aproximar os valores desses ruídos de *0*, permitindo assim aos algoritmos de detecção, não considerar o ruído como possíveis pontos de interesse.

```
cvSmooth(finalFrame, finalFrame, CV_BLUR, param1, param2);
```

Figura 43: Função Blur de Imagens

É importante salientar que as opções de tratamento de imagem que foram referidas acima são opcionais, como tal a sua utilização depende do utilizador.

4.4 Reportar as Coordenadas

Esta secção tem por objectivo descrever o último elemento do diagrama de blocos, que define o funcionamento do sistema. Esta fase, posterior aos algoritmos de detecção, reporta na forma de coordenadas cartesianas (x, y) o(s) valore(s) estimado(s) do(s) ponto(s) de interesse(s). Com base nestes valores é possível interagir e desencadear acções, de modo a que o sistema interaja com o utilizador. Esta pode ser vista como uma etapa de interpretação lógica, ou mesmo, como a inteligência do nosso sistema. No âmbito do quadro interactivo, iremos definir dois tipos de acções: clique e movimento/arrastamento.

É de salientar que quando se reportam as coordenadas cartesianas, simultaneamente também se pode acrescentar os valores estimados do tamanho e da intensidade do

PI. O tamanho do PI é estimado segundo a fórmula 4. A Figura 44 mostra a forma de identificar os valores *xMax*, *xMin*, *yMax* e *yMin*. Esta medição é estimada, porque recorremos à multiplicação das distâncias absolutas nos eixos do PI com origem no centróide estimado. Desta forma obtemos um tamanho correspondente à área representada a laranja (Figura 44).

$$\text{Tamanho PI} = |\ (xMax-xMin)*(yMax-yMin)\ |\ /\ 2 \quad (4)$$

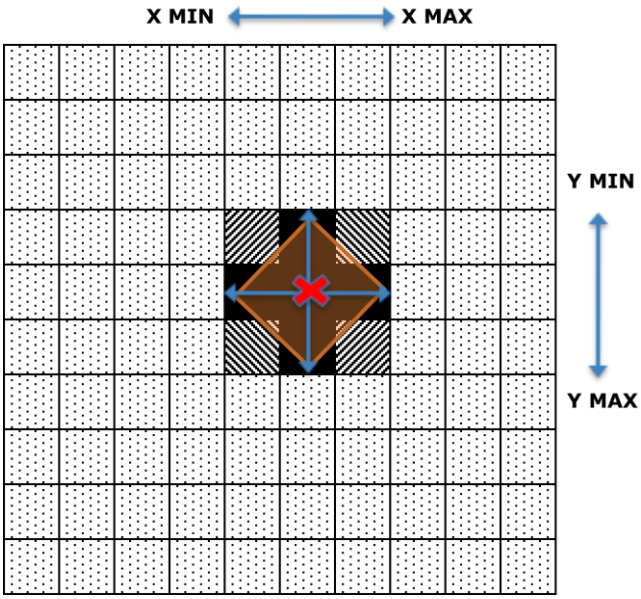


Figura 44: Mecanismo utilizado para estimar o tamanho do PI

Como foi referido anteriormente, este projecto pretende simular um quadro interactivo, onde presentemente se irão usar somente as coordenadas de um único PI, para as interacções com o sistema. Se utilizarmos o algoritmo A5 e se houver um ou mais PI presentes, iremos sempre adoptar por ponto de referência o ponto mais próximo da origem, isto é, o primeiro ponto que seja encontrado no caso da pesquisa em varrimento linear. No futuro, poderá usar-se a informação de vários PI, caso a aplicação concebida o permita ou reutilizando aplicações já concebidas que suportem o TUIO.

O interpretador, encontra-se implementado na função *Interpreter*, baseado nos pontos que foram retornados no formato de coordenadas cartesianas. Este tenta simular o comportamento do rato na sua interacção com o Sistema Operativo.

É de referir ainda que neste caso estamos a trabalhar com duas resoluções diferentes, isto é, temos a resolução presente no monitor do PC (a qual tem uma altura e largura conhecidas) e por outro lado, temos a resolução da imagem projectada pelo retroprojector (esta pode não ser linear). Podemos ter uma imagem projectada perfeitamente rectangular, o que não é frequente, ou então uma figura mais próxima de um quadrilátero. Sendo esta uma figura geométrica, que poderá ter ou não as arestas do mesmo tamanho, será então necessário primeiro mapear os pontos entre a escala da imagem projectada e a correspondente na resolução do nosso computador, como iremos ver na secção seguinte.

4.4.1 Calibração

A calibração é essencial em todos os sistemas de projecção. Existem dois principais tipos de calibração, a calibração geométrica e a calibração de cores. A primeira tem por objectivo encontrar uma matriz de transformação geométrica entre duas imagens; na segunda, a calibração de cor pretende corrigir aspectos da cor, isto é, como a cor do ecrã do computador será projectada numa outra superfície. Ou seja, a imagem projectada irá apresentar tonalidades diferentes e isto depende de várias características, tais como: o estado da lâmpada do retroprojector, da tela ou mesmo da luz ambiental na sala. Para corrigir estes aspectos é necessário calibrar a imagem adquirida pela câmara. Neste caso concreto, não iremos necessitar da calibração de cores porque simplesmente não consideramos as cores como factor de referência na identificação dos PI. A calibração da cor é necessária em sistemas do género do reconhecimento de escrita num quadro tradicional, a fim de estudar e entender a cor da caneta do lápis sobre a superfície. Nestes casos utilizam-se técnicas de *Visual Echo-Cancelation*, de forma a que a câmara consiga distinguir o que faz parte da imagem projectada e o que foi conteúdo acrescentado, isto é, o que o utilizador acrescentou *a posteriori* na imagem projectada. O sistema faz como que uma subtracção do que foi projectado com a imagem capturada pela câmara, resultando unicamente as notas ou acções do utilizador (Hanning Zhou et al. 2004). Neste projecto, foi utilizada apenas uma calibração geométrica, para que as acções executadas na projecção se reflectissem no Sistema Operativo. A Figura 45 mostra uma das possíveis representações da imagem projectada. No centro, observamos um PI, representado por

um ponto vermelho, sendo necessário encontrar um factor escala, a fim de permitir mapear esse PI, na escala do computador. A Figura 46 mostra o resultado da calibração geométrica, onde o PI está já na resolução correspondente à que estamos a utilizar no computador.

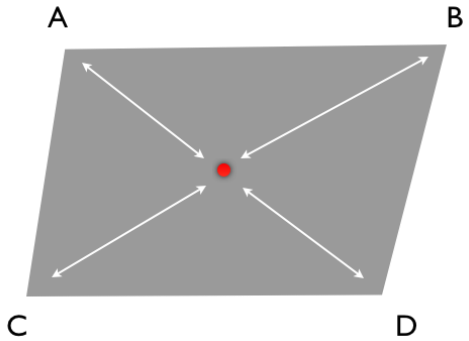


Figura 45: Representação de um possível Quadrilátero correspondente à Projecção

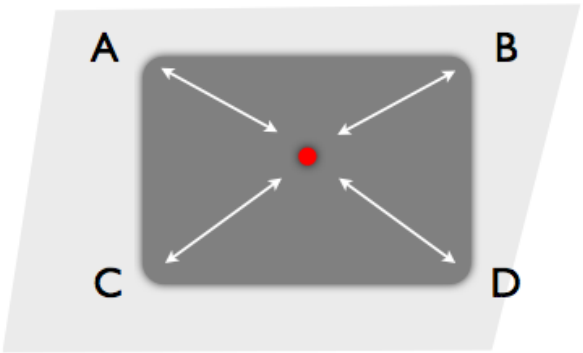


Figura 46: Representação da Projecção e da Resolução Nativa

A implementação da fase de calibração inicia-se com o arranque da aplicação. Após a realização da mesma, não se pode mexer na colocação da câmara nem na zona de projecção. Caso algum destes dois elementos seja movido durante a execução da aplicação, será necessário voltar a proceder a uma nova calibração. A calibração é realizada na função *startCalibrate* (ver Figura 48). Esta executa algumas funções de pré-processamento que foram vistas anteriormente (cf., modelo de *Background Subtraction* ou *Flip* da imagem). Realizamos a calibração geométrica a partir de quatro pontos, cada um localizado em cada um dos quatro cantos da imagem. Os pontos serão colocados, usando um valor *distance* correspondente à distância em *x* e *y* do ponto de referência de cada um dos cantos, este corresponde às setas brancas na Figura 47. Desta forma, pode-

remos correlacionar os valores desses pontos que conhecemos, com os que foram lidos pela câmara. Os pontos serão lidos de forma sequencial, de forma com que o sistema possa distinguir cada um deles. Entre cada leitura será acrescentado um tempo de atraso (i.e., *delay*), para impedir que o sistema leia o mesmo ponto várias vezes.

Figura 47: Colocação dos Pontos de Referência para a Calibração

```
for (int i = 0; i <= 3; i++) {
    img = cvLoadImage(fileName);
    printCircle(references[i], img);
    cvShowImage("Calib", img);
    bool point = false;
    coordinate temp;
    while (!point) {
      frame = cvQueryFrame(capture);
      IplImage* workFrame = NULL;
      workFrame = cvCreateImage(cvSize(frame->width, frame->height),
                  IPL_DEPTH_8U, 1);
      cvCvtColor(frame, workFrame, CV_BGR2GRAY);
      temp = singlePointProcess(workFrame);
      if(temp.x==-1 || temp.y==-1)
            keyPressValue=cvWaitKey(200);
      else{
            if (temp.x > -1 && temp.y > -1 && temp.x < frame->width &&
temp.y< frame->height && temp.x != projection[i - 1].x && temp.y!=
projection[i - 1].y) {
                  cvReleaseImage(&img);
                  cvReleaseImage(&workFrame);
                  cout << '\a' << flush;
                  point = true;
            }
      }
    }
    projection[i] = temp;
    keyPressValue=cvWaitKey(1000);
}
```

Figura 48: Implementação de startCalibrate

Após o sistema ter lido os quatro pontos, procede à construção da matriz transformada, usada para mapear as coordenadas lidas pela câmara na resolução do PC. A escala é inicializada seguindo a função *setScale* (Figura 49). Usa-se a matriz *mat_trf* para efectuar conversões entre as duas resoluções. Inicializa-se uma matriz tridimensional com a função *cvCreateMat*, cujos parâmetros são, por ordem: o número de linhas, de colunas e o seu tipo (Bradski & Kaehler 2008, p.170). Usam-se dois *arrays* de variáveis do tipo *CvPoint2D32f*, com um tamanho igual a quatro (ver Figura 49). Cada um destes *arrays* irá armazenar as posições dos pontos de calibração referidos na Figura 47, sendo que *srcQuad* e *dstQuad*, irão guardar respectivamente os valores obtidos através da leitura, na imagem projectada e os valores reais dos pontos na imagem do PC. Através da função *cvGetPerspectiveTransform*, conseguimos inicializar a matriz transformada (cf., *mat_trf*).

```
CvPoint2D32f srcQuad[4], dstQuad[4];
CvMat * mat_trf;
mat_trf = cvCreateMat(3,3,CV_32FC1);
srcQuad[0].x = projection[0].x; srcQuad[0].y = projection[0].y; // src
Top Left
srcQuad[1].x = projection[1].x; srcQuad[1].y = projection[1].y; // src
Top Right
srcQuad[2].x = projection[2].x; srcQuad[2].y = projection[2].y; // src
Bottom Left
srcQuad[3].x = projection[3].x; srcQuad[3].y = projection[3].y; // src
Bottom Right
dstQuad[0].x = references[0].x; dstQuad[0].y = references[0].y; // dst
Top Left
dstQuad[1].x = references[1].x; dstQuad[1].y = references[1].y; // src
Top Right
dstQuad[2].x = references[2].x; dstQuad[2].y = references[2].y; // src
Bottom Left
dstQuad[3].x = references[3].x; dstQuad[3].y = references[3].y; // src
Bottom Rightset

cvGetPerspectiveTransform(srcQuad, dstQuad, mat_trf);
```

Figura 49: Função para inicialização da Matriz Transformada

Para finalizar, usa-se uma função para transformar os pontos da escala de projecção, na resolução correspondente do monitor. Esta transformação é feita através da função *convertToScale*, que recebe as coordenadas do ponto que queremos converter e devolve os novos valores na resolução adequada (ver Figura 50). Desta vez, usa-se a função *cvPerspectiveTransform*, que com base num ponto e numa matriz transformada,

processa os valores do ponto de destino (Bradski & Kaehler 2008, p.171). As novas coordenadas do ponto, na sua nova escala, vêm armazenadas na variável *temp*.

```
CvMat* src_point = cvCreateMat( 1, 1, CV_32FC2);
CvMat* dst_point = cvCreateMat( 1, 1, CV_32FC2);
float *data_src = src_point->data.fl;
float *data_dst = dst_point->data.fl;

data_src[0] = temp.x; data_src[1] = temp.y; // src point (x,y)

cvPerspectiveTransform(src_point, dst_point, mat_trf);
temp.x=data_dst[0]; temp.y= data_dst[1];
return temp;
```

Figura 50: Função convertToScale

Salienta-se que a função *convertToScale* é invocada cada vez que o algoritmo encontra novas coordenadas de um ponto de interesse, para replicar as acções do utilizador na resolução do ambiente do sistema operativo.

4.4.2 Interpretador do Rato

Obtidas as coordenadas dos pontos de interesse e transformando-as na resolução do nosso sistema operativo, falta analisar o trabalho do interpretador. Este interpretador deve ter inteligência para detectar, a partir das coordenadas de pontos em formato cartesiano, as acções que o utilizador pretende desencadear. Estas acções podem ser cliques de rato, operações de arrastamento ou mesmo movimentos. Do mesmo modo, o interpretador deverá desencadear as acções necessárias para mover o apontador do rato para as coordenadas pretendidas.

Existe uma parte da implementação que é comum a todos os sistemas operativos e que está relacionada com o mecanismo usado pelo sistema para deslocar o apontador do rato. Para isso, usa-se um método disponível na biblioteca QT (NOKIA 2008). Desta forma, conseguimos, em qualquer que seja o ambiente, usar a mesma directiva para mover o rato, simplificando o código. A função *qtMove* é evocada de forma simples, enviando as coordenadas *x* e *y*, correspondentes à posição para onde pretendemos deslocar o ponteiro. Estas devem estar na escala da resolução usada no monitor do computador (ver Figura 51). Uma alternativa foi usar também as directivas de pré-

processamento associadas a métodos próprios do sistema operativo, para deslocar o ponteiro do rato.

```
void Algorithms::qtMove(int x, int y) {
    if (x > -1 && y > -1)
      QCursor::setPos(x, y);
}
```

Figura 51: Função que assegura as Deslocações do Rato

Do mesmo modo, é este interpretador que irá conter as directivas adequadas ao sistema operativo em uso. Por exemplo, não será feito da mesma forma uma acção do rato em ambiente Windows e em Mac OS X (e.g., cliques esquerdo do rato). Usam-se directivas de pré-processamento, a fim de evocar a função adequada. *A posteriori*, estas poderão ser melhoradas, bastando alterar a função que gere os eventos do rato correspondente, não sendo necessário proceder a uma alteração na lógica do interpretador, ficando este independente do sistema operativo.

De momento, o interpretador encontra-se implementado na função *interpreter* (ver Figura 52). Foram implementadas as acções mais básicas, como o clique esquerdo, o clique duplo esquerdo, o arrastamento mantendo o clique esquerdo premido. Nesta fase, ainda não se implementou o clique direito, no entanto, uma das possibilidades de implementação consiste em assimilar um PI fora da área de calibração, isto é, fora da resolução da imagem projectada, como uma vontade do utilizador em efectuar um clique direito.

```
if (keyPress) {
    if (presentPoint.x != -1 && presentPoint.y != -1) {
            if((tryPoint.x + aproximation) >= presentPoint.x
               && (tryPoint.x - aproximation) <= presentPoint.x
               && (tryPoint.y + aproximation) >= presentPoint.y
               && (tryPoint.y - aproximation) <= presentPoint.y )
                  countRightClic++;
                  //KEY PRESS AND MOVING
            } else {
                  //RELEASE KEY
                  keyPress = false;
            }
      } else {
            if (wantToClic && (tryPoint.x + aproximation) >=
presentPoint.x && (tryPoint.x - aproximation) <= presentPoint.x &&
(tryPoint.y + aproximation) >= presentPoint.y && (tryPoint.y -
```

```
aproximation) <= presentPoint.y ) {
                    //KEY PRESS
                    wantToClic = false;
                    keyPress = true;
            } if (lastPoint.x == -1 && lastPoint.y == -1 &&
presentPoint.x != -1 && presentPoint.y != -1) {
                    tryPoint.x = presentPoint.x;
                    tryPoint.y = presentPoint.y;
                    wantToClic = true;
                    keyPress= false;
            }
    } if(countRightClic == VALUE)
    rightClic=true;
    if(rightClic)
    { if(presentPoint.x !=-1 && presentPoint.y !=-1){
            //DOUBLE KEY PRESS ON PRESENT POSITION
      } else{
            //DOUBLE KEY PRESS ON LAST POSITION
      }
      countRightClic=0;
      rightClic=false;
      keyPress = false;
}
```

Figura 52: Interpretador do Rato

A aplicação permite desencadear acções usando as coordenadas dos pontos de interesse recolhidos. Esta poderá num futuro ser trabalhada permitindo, por exemplo, o suporte à interacção e detecção de gestos. Desta forma, poderia por exemplo abrir um *Main Menu*, fazendo um gesto circular. Estas funcionalidades não estão de momento implementadas, mas poderão ser consideradas como possíveis linhas de pesquisa para um trabalho futuro.

4.4.3 Servidor de Eventos

Alguns dos sistemas já existentes (e.g., Tbeta) oferecem a possibilidade de reportar as coordenadas (Touchlib n.d.; Tbeta 2008) através da rede, sobre um formato de mensagem conhecido - o TUIO (reacTIVision 2009). A reutilização de um sistema baseado em mensagens através da rede traz imensas vantagens como, por exemplo, a interoperabilidade ou mesmo a separação de ambas as ferramentas. Assim, temos uma plataforma responsável pela detecção e aquisição de PI e outra para interagir e manipular a aplicação baseada nas coordenadas dos PI. Outra das vantagens, que identificamos sobre a utilização deste tipo de protocolo, é a ausência da necessidade de ter um *driver* e de proceder a emparelhamentos, como era necessário aquando da utilização da *Wiimote*

(Lee n.d.; Schmidt 2008). Outra mais-valia da reutilização de mensagens no formato TUIO, é permitir que o LoCoBoard seja compatível com as inúmeras aplicações, que interagem com o (Touchlib n.d.; Tbeta 2008). As aplicações mais comuns, que funcionam através do TUIO, são desenvolvidas recorrendo ao *ActionScript 3* e *Flash (NUI Group Authors 2009, p.52)*. Esta escolha deve-se ao facto dos programadores disporem de uma ferramenta que permite aos utilizadores criarem aplicações do género *Cross-Platform* para os seus ambientes. O *Flash* permite em combinação com uma biblioteca de Visão por Computador, como é o Touchlib (Touchlib n.d.), CCV (Tbeta 2008) e a recTIVision (reacTIVision 2009), criar sistemas do tipo multi-toque. A comunicação, como referido antes, é assegurada através da rede, usando o TUIO (reacTIVision 2009).

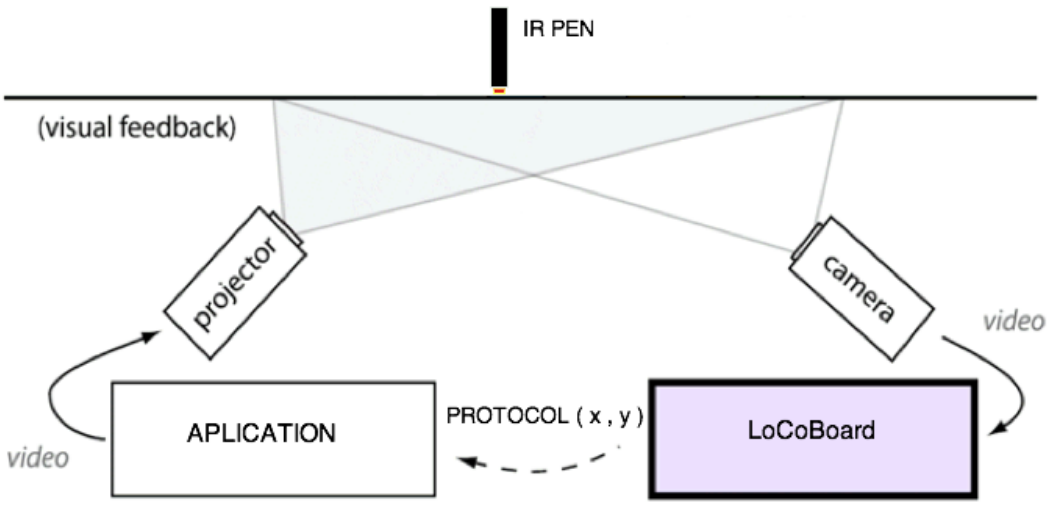


Figura 53: Arquitectura do Sistema reportando coordenadas na Rede

Nesta fase de desenvolvimento do projecto, decidiu-se abrir a possibilidade à implementação de novas aplicações por parte dos programadores, podendo este, sem alterar o funcionamento da já existente da plataforma, reutilizar coordenadas de pontos devolvidos para desenvolverem as suas próprias aplicações. A mais-valia é trazer aos programadores que não possuem conhecimentos na área de visão por computador, a possibilidade de poderem conceber aplicações com base nestas tecnologias. A arquitectura destas plataformas poderá ser semelhante à que foi proposta na Figura 7 e pode ser consultada na Figura 53.

A implementação do protocolo TUIO (Figura 54), obriga a construir uma mensagem com informação do PI em cada nova *frame* processada. Esta mensagem notifica as aplicações dependentes da existência ou ausência de um novo PI. Quando existir um PI, será necessário alterar a escala desse ponto, devido à expressão das coordenadas numa escala que tem como valor mínimo 0.0 e máximo 1.0. Quando não houver PI devemos assinalar o facto, apagando os pontos activos. Por fim, será processada uma mensagem quando evocarmos a função *frame()* da instância *app* do objecto *OSCApp*. A *OSCApp* é uma classe do *Oscpack* (Bencima 2006), utilizada para facilitar a criação, o envio, a recepção e a análise dos pacotes OSC (The Center For New Music and Audio Technology (CNMAT) 2002).

```
float X=0.0,Y=0.0,dX=0.0,dY=0.0;
if(temp.x!=-1 && temp.y!=-1){
    //Convert to scale 0.0 - 1.0
    X=convertToResolutionF(temp.x,screenWidth,1.0);
    Y=convertToResolutionF(temp.y,screenHeight,1.0);
    TouchData activeBlob= {blobID,0,X,Y,0,0,0,0,dX,dY,0};
    //Create one active Point
    app.setActiveFinger(activeBlob);
} else {
    //Delete All Active Points
    app.clearFingers();
}
//Send TUIO Message
app.frame();
```

Figura 54: Envio do PI através do TUIO recorrendo ao OSCPack

Este tipo de interacção permite que inúmeras aplicações possam ser concebidas com base nas coordenadas retornadas como, por exemplo, sistemas de *Tracking*, reconhecimento de gestos ou de forma, de interacção com qualquer superfície.

4.5 Concepção do Apontador de Infravermelhos

Existem várias características a ter em conta na escolha do LED adequado (NUI Group Authors 2009, pp.3-5). Como referimos acima, a luz IR foi escolhida para este projecto, pelas suas características. Permitindo-nos desta forma, ao nível físico, limitar as possibilidades de ruído, diminuindo ao nível de software a carga de processamento de tratamento de imagem. Embora esta última, não possa ser descartada de todo, será sempre necessário efectuar uma análise e um pré-processamento da imagem, porque

nunca conseguimos reunir todas as condições para evitar ruídos. Segundo (NUI Group Authors 2009, pp.4-5), algumas das características a ter em conta para a escolha dos LED emissores de IR, são:

- Comprimento de onda (*wavelength*), deve ter valores compridos entre 780 e 940nm, para ser facilmente distinguido pelas câmaras.
- Intensidade de brilho (*radiant intensity*), o valor mínimo deve ser de 80mw. Quanto maior for este valor melhor será feita a detecção do ponto IR.

Podemos ver na Figura 55 a simples constituição da caneta usada como apontador IR, ao longo das experiências que foram realizadas. A sua constituição é a seguinte: uma pilha AAA com suporte; um interruptor; um LED; um fio condutor.

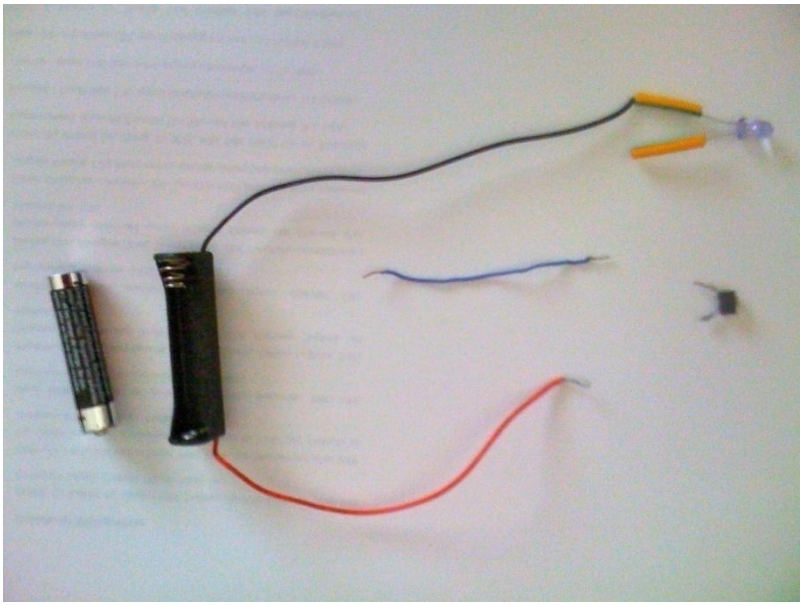


Figura 55: Componentes dos Apontador de Infravermelhos

A constituição deste equipamento, não representa grandes dificuldades, nem ao nível da aquisição nem da montagem. O seu custo é bastante reduzido e o trabalho necessário para realizar um dispositivo destes, não requer nenhuma noção avançada em electrónica.

4.6 Preparação da Webcam

As principais características, a ter em conta pelo sistema de captura dos sinais de infravermelhos estão enunciadas em (NUI Group Authors 2009, pp.5-7). As câmaras embora possam nativamente detectar luz proveniente duma fonte de luz IR, vêem equipadas com um filtro, a fim de atenuar a exposição da luz IR. Poderíamos, caso pretendêssemos obter um resultado melhor para o nosso sistema, manipular a câmara e desmontar a lente de forma a retirar o filtro que atenua a captura da luz IR. Importa referir, que as experiências realizadas neste projecto não alteraram a *webcam* do computador portátil, o que não impediu o bom desempenho do sistema. No entanto, como foi referido, poderia eventualmente melhorar-se o resultado e um comportamento do sistema caso se retirasse o filtro IR. Segundo (NUI Group Authors 2009, pp.6-7), os aspectos a ter em conta na escolha da câmara, que vai fazer a leitura dos pontos IR são:

- A resolução. Quanto maior for a resolução, maior será o número de pixéis capturados, o que nos permitirá ter uma melhor noção dos pontos IR, diminuindo os possíveis erros de precisão. Contudo é de referir, que quanto maior a resolução, maior será o poder de processamento em tempo real para analisar as imagens. As resoluções aconselhadas são de 320x240 píxeis, em mesas multi-toque de pequena dimensões, ou 640x480 para superfícies de maiores dimensões.
- A taxa de amostragem. Corresponde ao número de imagens por segundo, que a câmara é capaz de recolher (*Frame Per Second* - FPS). Este valor pode ser importante nas fases de seguimento de pontos de IR, permitindo uma maior fluidez entre os pontos recolhidos pelo sistema. Um valor aconselhado, para uma boa fluidez do sistema é de 30 FPS. Quanto maior for esse valor, melhor poderá ser a resposta do sistema face às nossas acções. No entanto, isso irá como à semelhança de grandes resoluções, ter um aumento no custo do processamento em tempo real.
- A interface. Existem dois tipos de interface distintas, normalmente usadas a USB e a IEEE 1394 (i.e., *Firewire*). É aconselhado o uso da interface

Firewire, sendo que esta nos oferece um menor *overhead* e menores tempos de latência na transferência da imagem, da câmara para o computador.

Estas são as principais características a ter em atenção. Contudo, deve-se consultar qual o filtro presente na câmara e se esse é facilmente extraível da mesma. Por outro lado, deve-se consultar as características da câmara e ver qual é sua sensibilidade às ondas de luz à volta dos 880nm, sendo nesta zona que se encontra a luz IR. Desta forma, vemos se temos uma câmara ideal para ser usada no nosso sistema. No caso das experiências que foram realizadas, usou-se a câmara integrada no *Macbook*. Esta funciona em VGA, com uma resolução de 640x480, e encontra-se ligada ao sistema pelo meio da interface USB. Não foi retirado o filtro que impedia a passagem da luz IR, para não danificar o equipamento. Foi colocada na frente da lente uma película fotográfica, sendo que esta faz filtro à luz do ambiente deixando somente passar a luz IR.

Após ter descrito a concepção do nosso sistema, iremos proceder a uma fase de avaliações com o intuito de avaliar o comportamento dos algoritmos e de o comparar com sistemas equivalentes.

Capítulo 5: Avaliação e Análise do Sistema

Este capítulo está dividido em três partes. Na primeira, avalia-se de forma sistemática o desempenho de cada um dos algoritmos de detecção de PIs descritos no Capítulo 3. Na segunda parte analisa-se segundo uma lista de critérios, as diferenças e semelhanças entre o LoCoBoard e sistemas equivalentes. Por fim, apresenta-se, de forma sucinta, o funcionamento e facilidade de utilização do LoCoBoard em diferentes ambientes.

5.1 Avaliação dos Algoritmos de Detecção de PIs

Nesta secção descrevem-se os resultados das avaliações quantitativas realizadas com os diferentes algoritmos de detecção de PIs. Procurou-se analisar de forma sistemática o comportamento dos diferentes algoritmos quando sujeitos a diferentes condições de teste, simuladas em vários vídeos.

Os testes efectuados com os diferentes algoritmos realizaram-se sempre na mesma máquina e com as mesmas opções de compilação. Foram criados seis vídeos de teste para posteriormente serem sujeitos aos algoritmos descritos (Tabela 9). Os vídeos incluem vários tipos de interacções, com o apontador de infravermelhos: cliques, movimentos contínuos e uma mistura de ambos. Foram ainda criados vídeos com e sem ruído, ou seja, com e sem a presença de fontes de interferência na imagem de fundo (e.g., luzes, reflexões, etc.).

Na realização dos vídeos procurou-se recriar várias situações de interacção. Por exemplo, durante a simulação de cliques, a interacção de infravermelhos aparece e desaparece em pontos aleatórios da imagem. No movimento, o ponto depois de apare-

cer, continua a deslocar-se pela imagem em movimentos aleatórios. Na simulação de interacções diversificadas, foram conjugados os dois tipos de interacção definidos anteriormente. Por outro lado, foram também introduzidas variações de luz (ruído) nas condições do ambiente, simulando assim interferências na determinação de um ponto de interesse. Os vídeos com ruído, são pré-processados para remover o fundo e isolar o ruído. Os algoritmos descritos anteriormente foram numerados de A1 a A5, de modo a podermos identificá-los nos gráficos decorrentes de cada experiência.

Tabela 9: Diferenças entre as características dos Vídeos utilizados

Condições / *Vídeo*	***Simulação***	***Ambiente***
Vídeo 1	Cliques	Sem Ruído
Vídeo 2	Movimentos	Sem Ruído
Vídeo 3	Diversificado	Sem Ruído
Vídeo 4	Cliques	Com Ruído
Vídeo 5	Movimentos	Com Ruído
Vídeo 6	Diversificado	Com Ruído

De forma a simplificar a estrutura e conteúdo desta secção apresentam-se apenas os gráficos com os resultados das experiências. No anexo 2, poderão, no entanto, ser encontradas as tabelas com os valores medidos e usados na construção dos gráficos.

5.1.1 Primeira Experiência - Desempenho

No primeiro estudo, comparamos o desempenho de cada um dos algoritmos, utilizando 500 imagens como limite da amostragem. Primeiro, medimos a taxa de utilização do processador (CPU) por cada algoritmo, em cada um dos vídeos; depois, calculamos a média desses valores (ver Figura 56). É possível verificar que as diferenças obtidas entre as médias não são muito significativas.

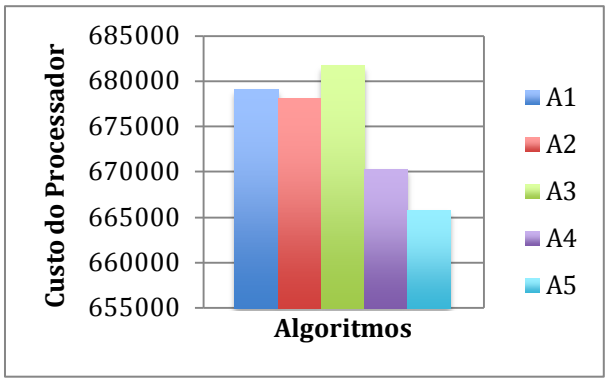


Figura 56: Taxa Média de Utilização do CPU

De forma a medir o desempenho de cada algoritmo, foram criados perfis temporais. Os valores apresentados representam em percentagem, o tempo que o algoritmo foi executado durante a amostragem. Os parâmetros usados foram, no A2 e A3 S=3 (salto), e no A4 N=5 (níveis). Podemos ver, na Figura 57, que os algoritmos com desempenho mais eficiente foram o A3 e o A4 (devido ao seu mecanismo de previsão nos vídeos 2 e 5), seguidos por A2, A1 e A5, sendo este último o menos eficiente.

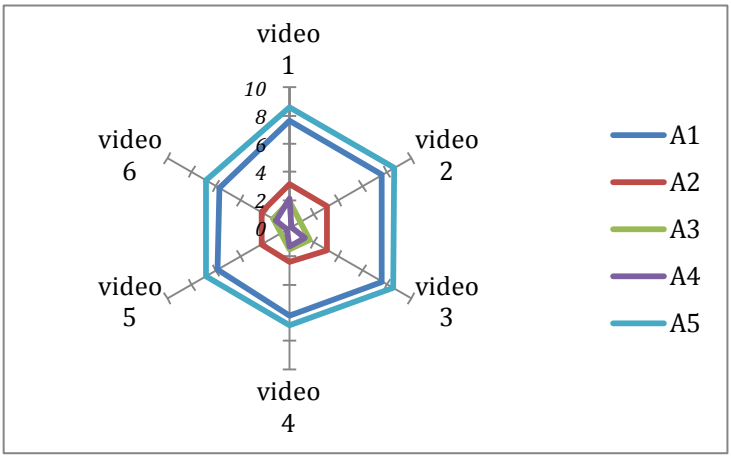


Figura 57: Tempo de Execução dos Algoritmos em cada Vídeo

Nesta experiência verificamos que os algoritmos A2, A3 e A4 são mais eficientes do que os restantes. Contudo estes resultados não podem ser considerados conclusivos nem nos permitam generalizar esta avaliação.

5.1.2 Segunda Experiência - Precisão

Esta segunda fase focou-se sobre a precisão na detecção dos pontos de interesse por parte dos algoritmos. Usamos cinco imagens (500 x 500 píxeis), cada uma com um único PI, do qual conhecemos as coordenadas reais (*ground truth*). Recolhemos então as coordenadas estimadas pelo algoritmo para, posteriormente, as compararmos com as coordenadas reais do PI. Repetimos estes passos para os cinco algoritmos, em cada uma das cinco imagens (ver Tabela 10). Os resultados apresentados correspondem à distância euclidiana (d), entre o ponto estimado e a posição real do PI, calculada pela fórmula (5).

$$\mathbf{d} = \sqrt{\Delta x^2 + \Delta y^2} \qquad (5)$$

Nesta fórmula, Δx corresponde à diferença entre a abcissa estimada e a real e Δy à respectiva diferença entre a ordenada estimada e a real. Esta distância pode ser vista como o erro na estimação do PI.

Nesta experiência aferimos a precisão dos diferentes algoritmos na estimativa dos PI. Quanto menor é o valor da distância euclidiana, mais fidedigna é a estimativa. Segundo esta perspectiva os algoritmos A2 e A5 apresentam os melhores resultados.

Tabela 10: Erro de Estimativa dos PI para os Diferentes Algoritmos

Algoritmos / *Ground Truth*	***A1***	***A2***	***A3***	***A4***	***A5***
Primeiro Ponto	1	0	3	4.12	0
Segundo Ponto	1,41	1	3,16	3,16	1
Terceiro Ponto	1,41	1,41	3,61	3,16	1
Quarto Ponto	1,41	1,41	3,61	3,16	1
Quinto Ponto	1,41	1,41	3,61	3,16	1

5.1.3 Terceira Experiência - Parametrização

Na última fase, analisamos o impacto da mudança de duas variáveis, nos algoritmos três e quatro (A3 e A4). Nesta experiência usamos o vídeo 3, que contém simulações de movimentos e cliques. Possivelmente, este vídeo será o que mais se aproxima dos casos reais, combinando várias situações de utilização.

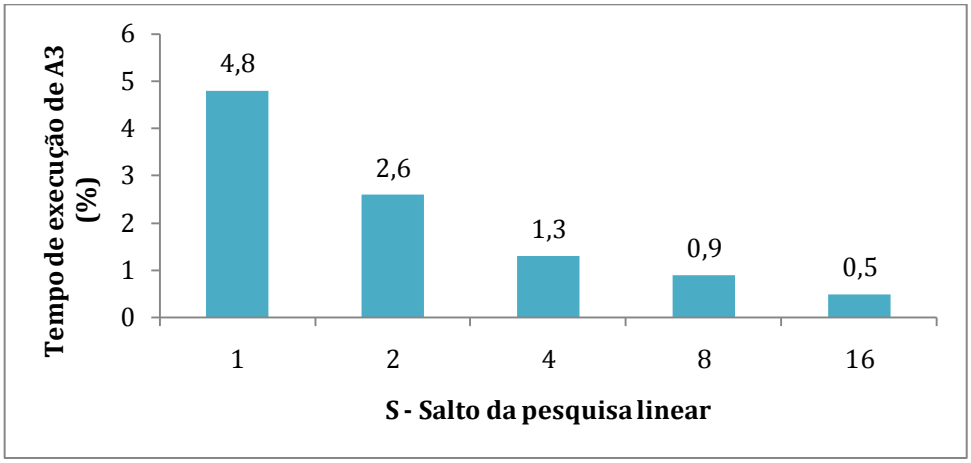


Figura 58: Tempo de Execução de A3 variando o Salto

No algoritmo três (A3), fazemos variar o valor do salto (S), para verificar o impacto no tempo de execução do algoritmo (através de um perfil temporal da execução). A percentagem de tempo obtida corresponde à razão entre o tempo de execução do algoritmo em relação ao tempo de amostragem total (ver Figura 58).

Ao analisar a Figura 59, podemos ver que, como seria previsível, a qualidade de detecção do Algoritmo A3 depende do valor aplicado no salto. Esta experiência foi realizada mantendo as mesmas definições mas analisando outro parâmetro, a noção de precisão, e não o tempo de execução; a precisão é avaliada segundo a definição de distância euclidiana tal como foi o caso na experiência anterior. Podemos concluir que será necessário ter um especial cuidado na selecção do valor de salto isto porque, embora possamos diminuir o tempo de processamento do algoritmo por cada imagem, podemos

da mesma forma diminuir a qualidade com que será feita a detecção de cada um dos pontos de interesses.

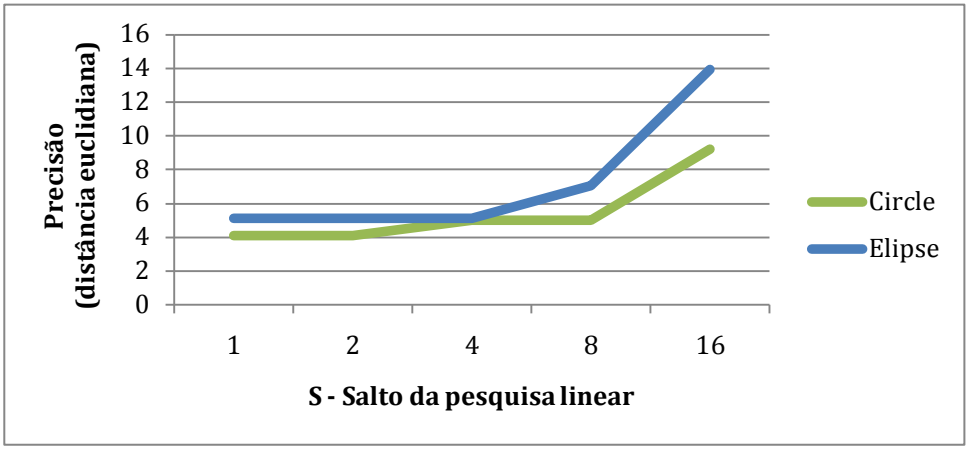


Figura 59: Precisão de A3 na detecção de PI (PI em Circulo e Elipse) variando o Salto

Usaram-se duas formas de PI diferentes na realização desta experiência, uma em círculo e outra em elipse (ver Figura 59) para mostrar que não era pela forma do PI que a precisão diminuía mas sim pela variação do valor S. Outros valores provenientes de experiências realizadas com outras formas de PIs, poderão ser consultadas no Anexo 2.

No algoritmo 4, fizemos variar o tamanho da espiral, aumentando o número de níveis (N) e analisamos qual é o impacto dessa variação na descoberta do PI. Esta corresponde à taxa de sucesso do algoritmo para encontrar o PI. Podemos ver os resultados nas Figura 60, Figura 61 e Figura 62. As diferenças encontradas nestes gráficos correspondem essencialmente às diferenças entre os vídeos utilizados na execução do algoritmo. Escolhemos diferentes vídeos de forma a evidenciar o comportamento e a eficiência deste algoritmo quando confrontado com movimentos de seguimento e não com detecções.

O primeiro vídeo (Figura 60) evidencia que o rácio de sucesso do algoritmo é baixíssimo quando o nível da espiral é baixo. Isto deve-se ao facto de estarmos perante um vídeo com aparições aleatórias de PIs e não de seguimentos/arrastamentos do PI. Nestes casos, o Algoritmo A4 raramente irá ter sucesso na determinação de uma previsão. Contudo, quando aumentamos o valor da espiral, este algoritmo poderá ser bem

sucedido, se a aparição do novo PI se encontrar na expansão da espiral à volta da possível nova previsão. Caso contrário este algoritmo não terá sucesso. Relembramos que quanto menor for o tamanho da captura (altura, largura) e maior for a profundidade de expansão da espiral (níveis), maior será a taxa de sucesso, isto porque efectuamos quase um varrimento em espiral completo à imagem.

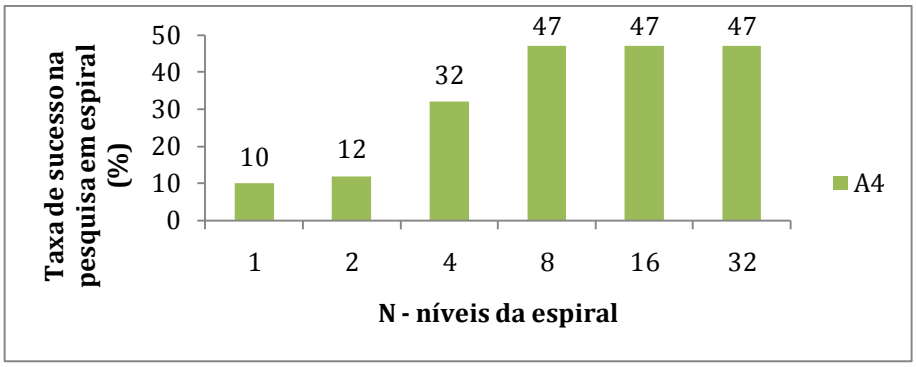


Figura 60: Taxa de Sucesso na Detecção do PI na Espiral – Vídeo 1

No segundo vídeo (Figura 61) os valores de sucesso são mais próximos de 100%, devendo-se isto ao facto do vídeo 2 conter somente a deslocação de um PI ao longo de toda a captura. Por exemplo, o algoritmo atinge os 93% de sucesso com os quatro níveis de profundidade.

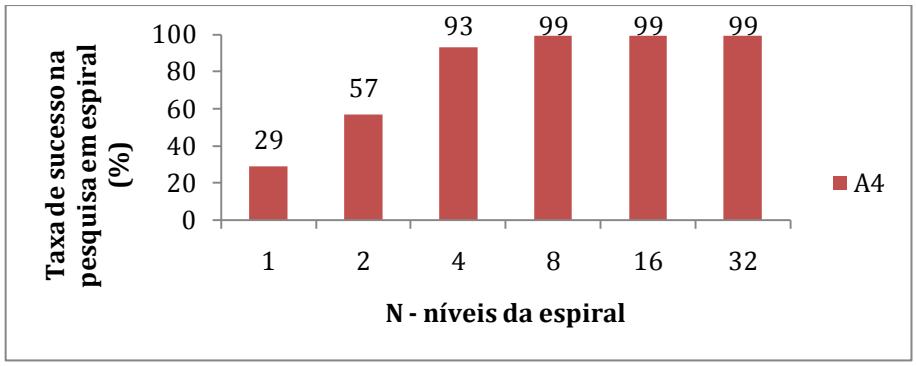


Figura 61: Taxa de Sucesso na Detecção do PI na Espiral – Vídeo 2

Por fim, o terceiro vídeo (Figura 62) corresponde a uma situação mais próxima do caso real, conjugando aparições e deslocações do PI em imagens consecutivas. Esta experiência permitiu-nos obter uma noção mais geral sobre a possível taxa de sucesso de detecção do algoritmo A4 numa situação real, dependendo do número de níveis usados (N). É de referir, no entanto, que esta não era a razão da experiência, ou seja, não se

pretendia provar a taxa de sucesso deste algoritmo perante uma situação de funcionamento mais próxima da realidade, ao invés, pretendia-se ilustrar a evolução da taxa de sucesso consoante o número de níveis (1, 2, ... , 32) e em diferentes situações possíveis (Video1, Video2, Video3).

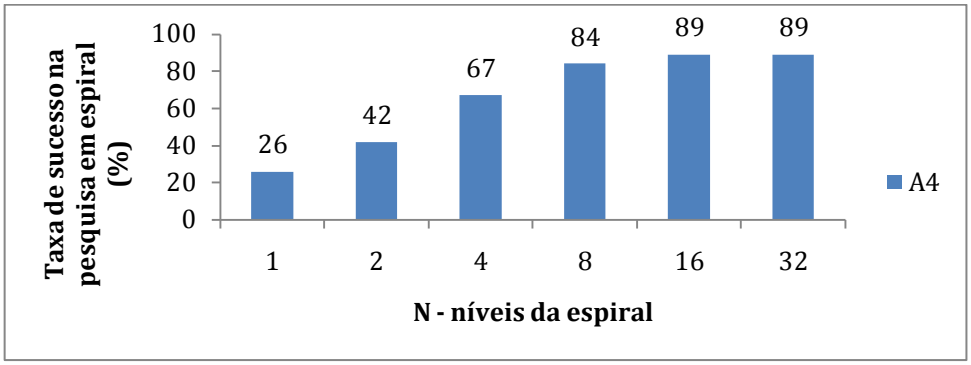


Figura 62: Taxa de Sucesso na Detecção do PI na Espiral – Vídeo 3

Verificamos que, quanto maior for a área percorrida pela espiral, maior é a probabilidade de se encontrar o PI nessa área. Por outro lado, o aumento desta área reflecte-se no aumento do uso do CPU uma vez que a área a percorrer é maior (ver Figura 63). Esta é a razão que nos levou a integrar uma pesquisa linear no algoritmo A4, aquando num determinado número de níveis "N" não fosse encontrado nenhum PI, os restantes pixéis seriam analisados numa pesquisa linear.

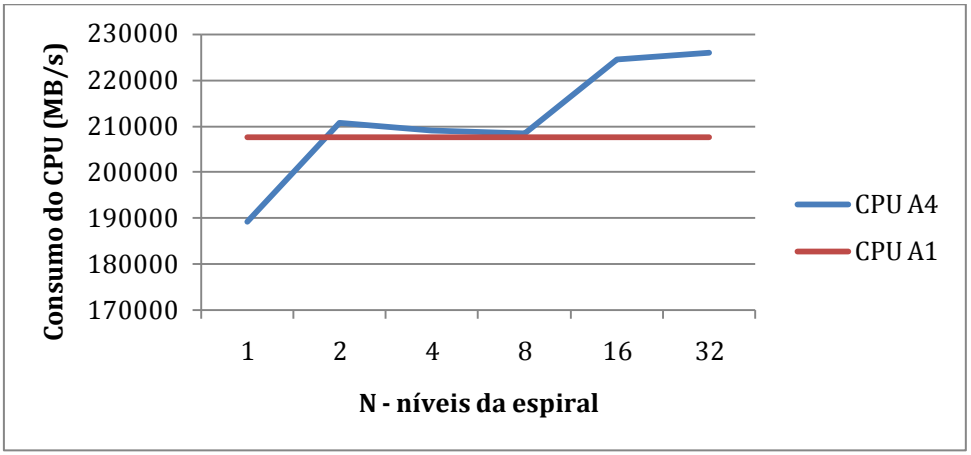


Figura 63: Comparação do consumo de CPU entre uma pesquisa linear (A1) e em espiral (A4) variando o número de níveis

5.1.4 Quarta Experiência - Centróide

A quarta experiência analisa o comportamento da detecção do centróide do PI por cada um dos algoritmos. Esta experiência difere da segunda, em particular, pela forma dos PI. Foram usados seis tipos diferentes de formas de PI (ver Figura 64):

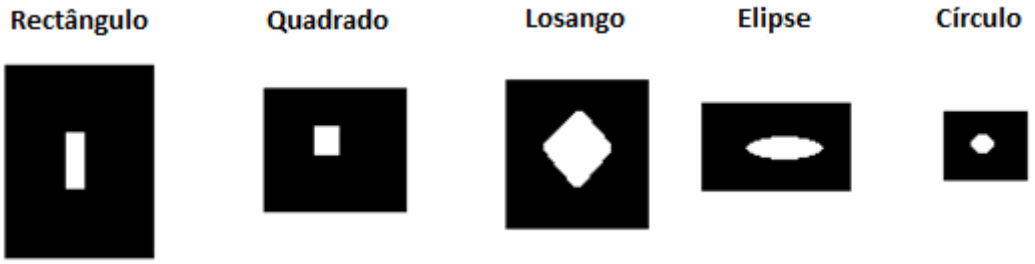


Figura 64: Diferentes representações geométricas usadas para representar o PI

Para verificar a precisão da avaliação, reutilizamos a noção de distância euclidiana, tal como definida na experiência 2 (consulte a fórmula 5). A Figura 65 reflecte os resultados que foram encontrados. Refira-se que não foi incluído o algoritmo A4 nesta experiência por este ser um algoritmo de seguimento e não de detecção. Este algoritmo, como foi referido no capítulo quatro, irá reutilizar um dos algoritmos anteriores para poder retornar as coordenadas do PI detectado. Assim, incluir aqui o algoritmo A4 seria redundante uma vez que os resultados seriam relativos aos algoritmos reutilizados pelo algoritmo A4 quando não pudesse efectuar uma previsão. Verificamos pelo aspecto da curva que a forma do PI afecta a precisão dos algoritmos. Em particular, o algoritmo A3 oferece os piores resultados a nível de precisão. Os algoritmos com os melhores níveis de precisão são o A1 e A5. No entanto, quando estes algoritmos são confrontados com a detecção de um PI cuja forma se aproxima de um losango, as suas taxas de erro aumentam. O algoritmo A2 tem tendência a manter a sua taxa de erro constante e independente das formas do PI. O algoritmo A2 apresenta uma menor taxa de precisão nos PI em forma de losango.

Esta experiência foi realizada para tentar analisar se a forma do PI poderia ou não interferir nos algoritmos de detecção. Embora os resultados não se possam generalizar, percebemos que certas formas de PI podem facultar melhores resultados do que outras.

Isto pode ser relevante na escolha do LED a utilizar no apontador de IR. De uma forma geral, analisando os resultados verificamos que existem menores taxas de erro quando as formas dos PI se aproximam de quadrados ou círculos, exceptuando o ligeiro desvio do algoritmo A1 no quadrado (embora todos os algoritmos revelem uma queda nestas duas formas).

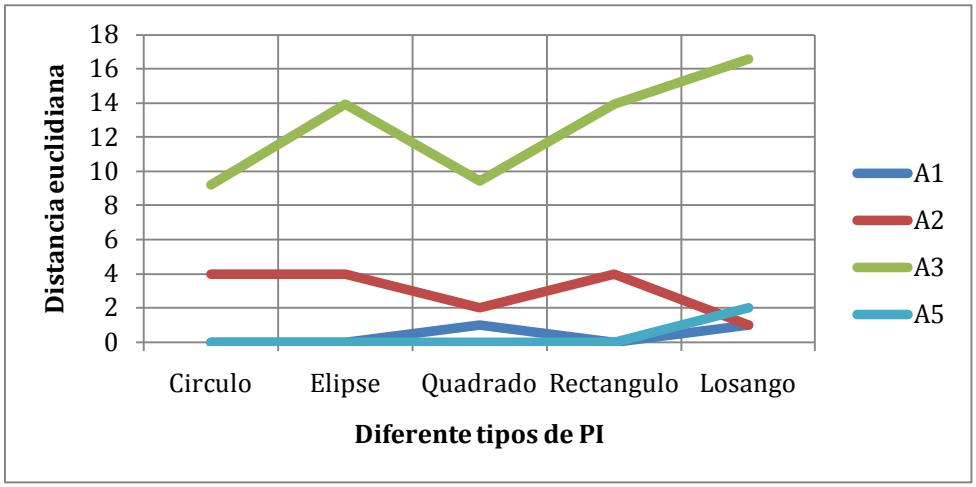


Figura 65: Avaliação da precisão de cada um dos algoritmos em função das formas do PI

5.1.5 Quinta Experiência - Processamento

Esta experiência avalia e compara o custo de processamento consumido por duas aplicações distintas: o CCV e o LoCoBoard. Esta comparação foi realizada com base na recolha de vinte amostras contendo o valor do custo de processamento das aplicações no seu geral. As amostras do consumo de CPU de cada aplicação foram recolhidas através da visualização num gestor de tarefas. Com base nesta recolha criou-se a Tabela 11, que resume o funcionamento da aplicação, de forma genérica.

Tabela 11: Comparação do consumo de CPU entre o LoCoBoard e o CCV (Tbeta)

Sistemas / *Consumo CPU*	***CCV (Tbeta)***	***Nosso Sistema***
Mínimo	64,1 %	14,0 %
Máximo	73,3 %	52,8 %
Média	68,3 %	32,1 %

Com esta experiência pretendeu-se avaliar qual, destes dois sistemas, usa mais CPU durante o processamento da aplicação. Verificou-se que o comportamento do LoCo-Board, comparado com o CCV (Tbeta), é menos dispendioso ao nível dos recursos de CPU usados. Isto deve-se à implementação de algoritmos próprios usados na detecção dos PI, enquanto que o CCV recorre a funções implementadas no OpenCV, por exemplo o *cvFindContours*, fazendo com que este sistema consuma mais recursos. Apesar desta experiência evidenciar que o LoCoBoard é mais eficiente que o CCV a nível de gastos de recursos (CPU), não podemos concluir que o permanecerá a nível de eficácia.

A secção seguinte efectua uma comparação, mediante uma série de critérios, entre o LoCoBoard e alguns sistemas equivalentes.

5.2 Comparação com Sistemas Relacionados

Nesta secção pretende-se avaliar qualitativamente o sistema prototipado em comparação com sistemas afins, segundo critérios bem definidos. Os critérios usados na comparação são os seguintes:

- Custo: este encontra-se definido através de uma escala constituída por três níveis, o baixo que contempla sistemas que tenham um preço inferior a trinta euros; o médio, inferior a cinquenta euros e o elevado, correspondendo a todos os outros. Na avaliação do custo são somente contabilizados os equipamentos extra necessários além do computador e do projector;
- Equipamento: este serve para listar o material necessário, para além do computador e do projector, na construção de um quadro interactivo;
- Portabilidade: esta propriedade diz respeito à plataforma suportada (e.g., Mac ou Windows);
- Interface: esta propriedade permite-nos avaliar se o sistema tem capacidade para suportar múltiplos pontos de interesses ou somente um por imagem;
- TUIO: este critério permite avaliar se a aplicação ou sistema fornece suporte ao envio de mensagens segundo o protocolo TUIO;

- Usabilidade: esta característica refere-se à complexidade da instalação do sistema segundo uma escala constituída por três níveis (cf. simples - caracteriza uma montagem com menos de quatro passos; moderado - com menos de seis passos e complexo - todos os outros. O detalhe dos passos de instalação necessários em cada sistema encontram-se definidos na Tabela 9 do Anexo 3.

Através da análise da Tabela 12, é possível verificar que de todos os sistemas propostos, o LoCoBoard é o que oferece o custo mais baixo, devido ao facto de ser um sistema minimalista que vive das potencialidades dos próprios portáteis, não sendo necessário acrescentar equipamento extra dispendioso. Isto era um dos principais requisitos do sistema, de forma a facilitar a reutilização da aplicação por parte do utilizador comum.

Tabela 12: Comparação entre LoCoBoard e Sistemas equivalentes

Critérios / *Sistemas*	***Custo***	***Equipamento***	***Compatibilidade***	***Interface***	***TUIO***	***Usabilidade***
LoCoBoard	Baixo	- Câmara - Apontador IR - Filtro Pass-IR	- Windows - Mac OS X - Linux	Toque Simples	Suporta	Simples
Johnny Lee Whiteboard (Lee n.d.)	Médio	- Wiimote - Bluetooth - Apontador IR	- Windows	Toque Simples	Não suporta	Moderado
Uweschmidt (Schmidt 2008)	Médio	- Wiimote - Bluetooth - Apontador IR	- Windows - Mac OS X - Linux	Toque Simples	Suporta	Moderado
CCV ou Tbeta (Tbeta 2008)	Baixo	- Câmara - Apontador IR - Filtro pass-IR - Aplicação	- Windows - Mac OS X - Linux	Multi-toque	Suporta	Moderado
Touchlib (Touchlib n.d.)	Baixo	- Câmara - Apontador IR - Filtro pass-IR	- Windows	Multi-toque	Suporta	Complexo

Outro requisito consistia em oferecer a maior compatibilidade com os diferentes SO existentes. Deste facto resulta a aplicação ter sido concebida para funcionar indistintamente nos SO mais comuns. O suporte para clique encontra-se disponível em Mac OS X, Windows e Linux, no entanto, focamos a nossa atenção em particular nas duas primeiras plataformas. Como referimos anteriormente o código está organizado de forma a que seja simples identificar, por intermédio das directivas condicionais (e.g., *#ifdef*), o

código que se destina a cada um dos diferentes SO. No entanto, o suporte através de mensagens TUIO, funcionando através da rede, oferece um comportamento mais transversal aos diferentes SO, logo deverá estar disponível qualquer que seja o SO utilizado.

A nível da interface, o sistema não fornece um suporte nativo ao multi-toque, embora esteja elaborado de forma simples um mecanismo que permita suportar dois PI ao mesmo tempo, isto é, reportar numa mensagem TUIO com os dois PI, para uma aplicação compatível com a mesma. Justificou-se a necessidade de acrescentar o protocolo de mensagens TUIO ao sistema, de forma a permitir a reutilização e integração de várias aplicações existentes (e.g., as demos disponibilizadas pelo CCV).

Por fim, pretendíamos do mesmo modo que a utilização e manipulação do sistema fosse simples e intuitiva a qualquer tipo de utilizador, daí termos concebido uma aplicação que minimizasse os passos necessários para a utilização do sistema. E isto pode ser considerado uma mais-valia para o utilizador, comparando com outros sistemas de funcionamento mais complexo.

Após esta comparação podemos dizer que o sistema desenvolvido oferece um conjunto de vantagens que podem ser atractivas ao público em geral, como sendo: o baixo custo, a simplicidade de instalação e utilização e a compatibilidade com diferentes plataformas e sistemas operativos.

5.3 Funcionamento do Sistema LoCoBoard

Nesta secção pretende-se apresentar e ilustrar de forma sucinta o funcionamento e facilidade de utilização do sistema LoCoBoard. Passa-se a descrever passo a passo a utilização do sistema como num simples manual de utilização.

1. Começa-se por encontrar um filtro que impeça a passagem de qualquer tipo de luz excepto IR. Por exemplo, recorrendo à utilização de uma película fotográfica colocada em frente à câmara (ver Figura 66).

Figura 66: Colocação do Filtro em frente a câmara (IR Pass Filter)

2. Passa-se à execução da aplicação (ver Figura 67), numa consola onde serão apresentadas as diversas opções de utilização possíveis (ver Figura 68).

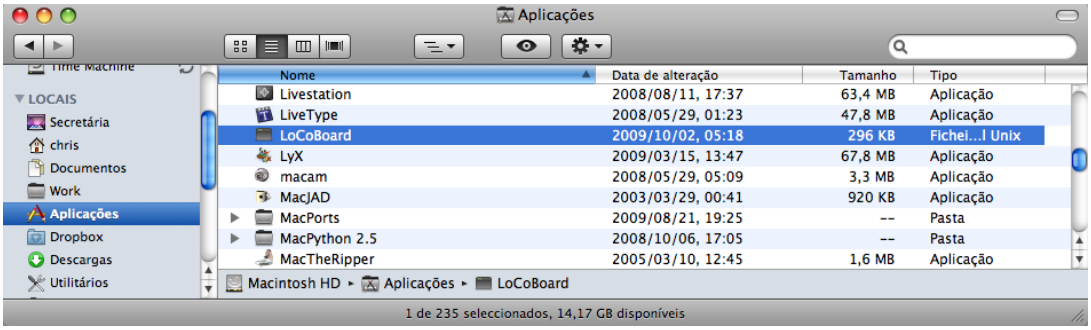


Figura 67: Execução da aplicação LoCoBoard

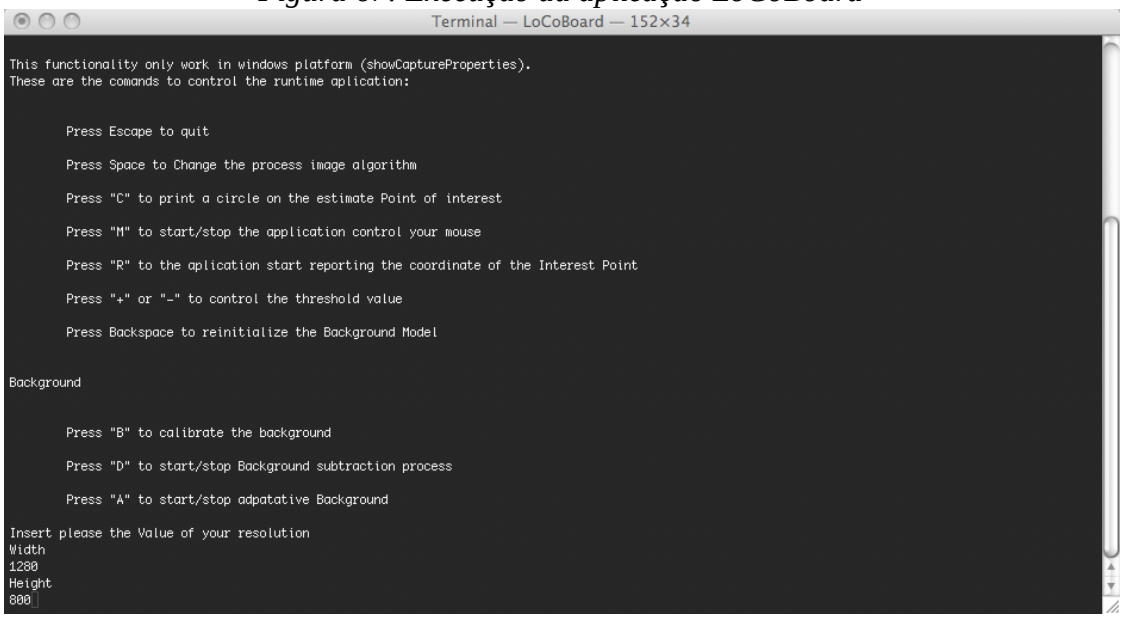


Figura 68: Aplicação LoCoBoard

3. Depois será necessário proceder a fase de calibração onde se efectua a leitura de quatro pontos para que o sistema se adapte à resolução da projecção (ver Figura 69).

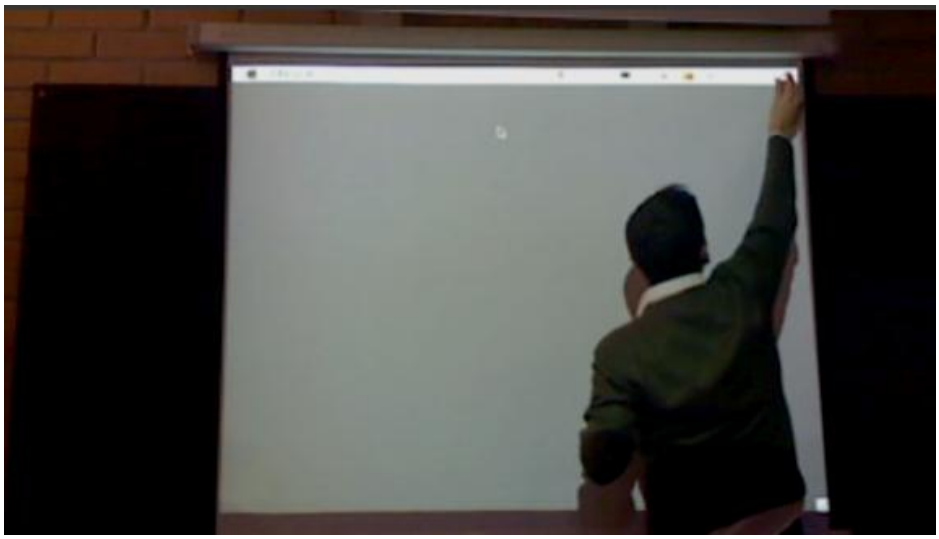


Figura 69: Calibração do LoCoBoard

4. Neste momento a aplicação de detecção encontra-se nesta fase funcional (ver Figura 70) e já iniciou o processo de envio de coordenadas via TUIO. Para iniciar o interpretador (i.e., controlo do rato) deve-se pressionar a tecla "M" como consta nas opções da Figura 68.

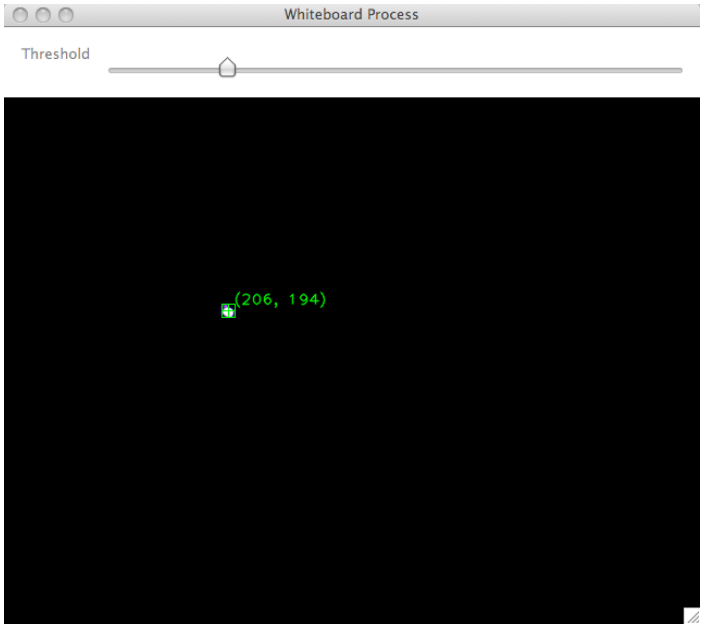


Figura 70: Janela da aplicação LoCoBoard

O sistema LoCoBoard permite ainda interagir com outras aplicações, da mesma forma que faríamos com o rato no computador. Algumas dessas aplicações são: o Google Earth (ver Figura 71) ou um Editor de texto (ver Figura 72).

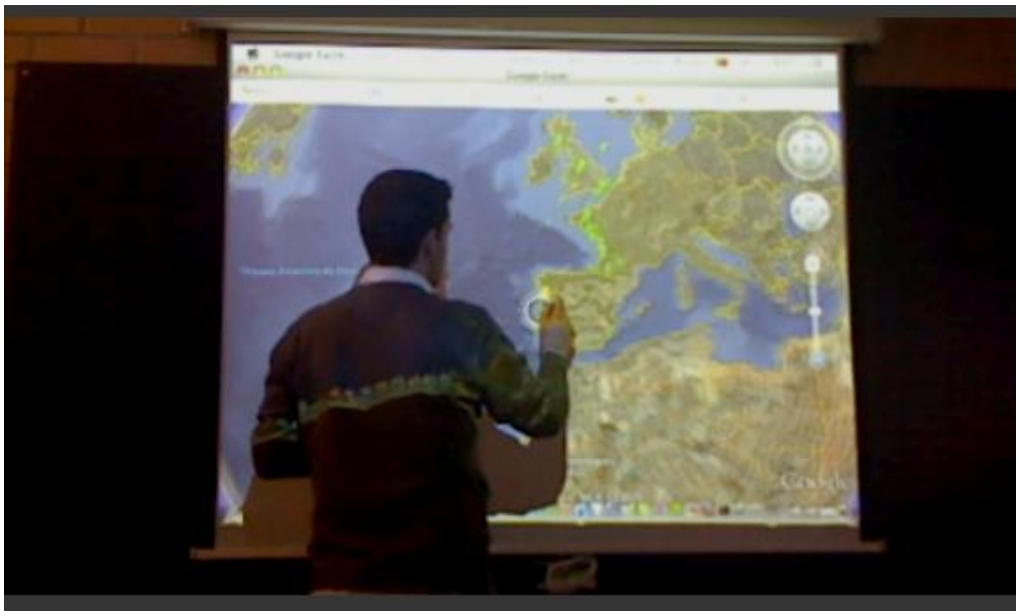


Figura 71: Interacção através do LoCoBoard com o Google Earth

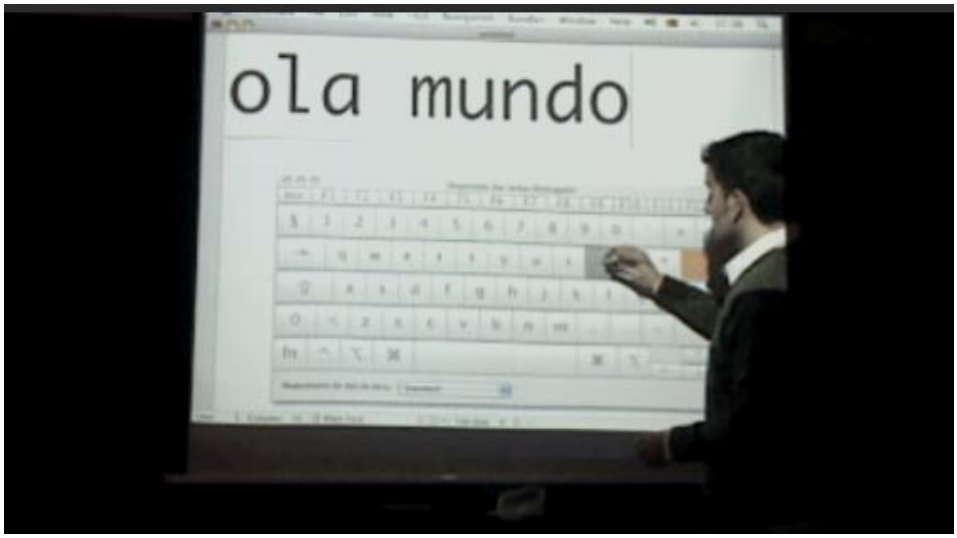


Figura 72: Interacção através do LoCoBoard com um Editor de Texto

A comunicação dos PI localizados através do TUIO é activa quando a aplicação é iniciada. Recorrendo ao servidor *flosc*, implementado em Java e disponibilizado pela comunidade do Tbeta, obtemos um sistema com capacidade para comunicar com qualquer tipo de aplicação que suporte TUIO. A aplicação irá fazer um reconhecimento das

teclas premidas durante a sua execução e poderá ter um comportamento adaptado às acções pretendidas por parte do utilizador. Foram produzidos alguns vídeos ilustrativos do funcionamento do LoCoBoard que podem ser consultados em (2009a). Estes vídeo mostram a facilidade preparação, execução e utilização do sistema.

Capítulo 6: Conclusões e Perspectivas de Desenvolvimento

Este trabalho propõe uma solução simples, económica e eficaz para a implementação de um quadro interactivo que possa satisfazer as necessidades dos utilizadores finais de quadros interactivos, como sejam, quaisquer professores e alunos sem conhecimentos de informática. Neste trabalho apresenta-se um protótipo perfeitamente funcional que combina conhecimentos de várias áreas. Contudo, constitui um trabalho em desenvolvimento que poderá ser complementados por outros investigadores.

Os objectivos inicialmente propostos foram integralmente cumpridos, nomeadamente a concepção de um quadro interactivo de baixo custo, simples, eficiente e prático para poder ser utilizado e personalizado por quaisquer utilizadores em diferentes plataformas (e.g., Windows, Mac, Linux). O protótipo do quadro interactivo apresentado no Capítulo 4 baseou-se num conjunto de algoritmos especificamente desenvolvidos para o efeito e que foram detalhados no Capítulo 3. Por fim, o protótipo foi avaliado segundo diversas perspectivas detalhadas no Capítulo 5. Este protótipo de quadro interactivo está disponibilizado em código aberto para que toda a comunidade académica possa contribuir para a sua evolução.

Um dos componentes principais do quadro interactivo é o módulo de detecção de pontos de interesse (PI). Foi possível, através das diferentes experiências realizadas, avaliar a eficiência dos algoritmos de detecção de PIs desenvolvidos. Uma vez que os resultados dependem do ambiente de utilização, foram utilizados vários vídeos, tentando contemplar várias formas de interacções mais recorrentes, baseadas em seguimentos/arrastamentos ou aparições pontuais de PIs. Da mesma forma, procurou-se também integrar ruído de fundo nestes vídeos (e.g., pontos de luz), procurando simular perturbações no funcionamento dos algoritmos. Com base nas experiências efectuadas, foi pos-

sível estabelecer os fundamentos para a criação de uma plataforma de testes dos algoritmos descritos. Embora as experiências não nos permitam apontar com rigor qual o algoritmo mais adequado para o fim em causa, os resultados apresentados permitiram-nos contudo ter uma ideia dos algoritmos com mais potencialidades.

Futuramente poder-se-á pensar no refinamento da implementação da plataforma e na sua utilização em aplicações concretas, que possam ser utilizadas em sala de aula, de modo a ilustrar o seu funcionamento e a testar e avaliar todo o conjunto em ambientes reais. Outras direcções a explorar, constituem a incorporação plena da tecnologia multi-toque no quadro interactivo.

No decorrer do trabalho foram encontradas algumas das dificuldades, resultantes fundamentalmente do funcionamento das bibliotecas de tratamento de imagem utilizadas (e.g., comunicação entre a câmara, o sistema operativo e a aplicação) que, por vezes, dificultaram o funcionamento do quadro interactivo. Contudo, estes problemas foram totalmente superados, permitindo a este projecto oferecer aos programadores, mais uma alternativa às plataformas existentes. Por outro lado, salienta-se que o código fonte do protótipo está partilhado com toda a comunidade académica, permitindo desta forma dar continuidade ao projecto, melhorá-lo e, eventualmente, contribuir para a melhoria das soluções e plataformas já existentes.

Referências Bibliográficas

2009a. *Low Cost Interactive WhiteBoard (an efficiente substitute for Wii-based Boards) Part 2/2*, UFP, Porto. Portugal. Available at: http://www.youtube.com/watch?v=gXeqTDklWKE [Accessed July 9, 2009].

2009b. *Project Natal*, Available at: http://www.youtube.com/watch?v=g_txF7iETX0 [Accessed July 9, 2009].

Associated Universities Inc., 1996. AIPS++ Glossary. *AIPS Glossary*. Available at: http://www.astron.nl/aips++/docs/glossary/w.html [Accessed July 18, 2009].

Ballmer, S., 2009. Microsoft at International Consumer Electronics Show 2009 Virtual Pressroom. *Ballmer on Natural User Interface*. Available at: http://www.microsoft.com/presspass/events/ces/keynote.aspx?initialVideo=full Keynote [Accessed July 18, 2009].

Bencima, R., 2006. oscpack -- a simple C++ OSC packet manipulation library. Available at: http://www.audiomulch.com/~rossb/code/oscpack/ [Accessed September 8, 2009].

Bovermann, T. et al., 2005. TUIO: A protocol for table-top tangible user interfaces.

Bradski, G. & Kaehler, A., 2008. *Learning OpenCV: Computer Vision with the OpenCV Library* 1st ed., O'Reilly Media, Inc.

Burden, K., 2002. Learning from the bottom up - the contribution of school based practice and research in the effective use of interactive whiteboards for the FE/HE sector, Developemnt Workshop Discussion paper. *Making an Impact Regionally*

Conference. Available at: http://lsda.org.uk/files/lsda/regions/8_Bio_KBurden.pdf.

Cordis, 2009. Ícones, janelas, menus - Quando a informática vai evoluir? Available at: http://www.inovacaotecnologica.com.br/noticias/noticia.php?artigo=icones-janelas-menus-quando-informatica-vai-evoluir&id=010150090717 [Accessed July 18, 2009].

Ekbutechnology, Interactive Whiteboards and My Teaching Goals. Available at: http://ekbutechnology.googlepages.com/interactivewhiteboardsandmyteachinggoals [Accessed July 2, 2009].

Gates, B., 2009. Gates: Natal to bring gesture recognition to Windows too | Beyond Binary - CNET News. Available at: http://news.cnet.com/8301-13860_3-10286309-56.html?tag=mncol;txt [Accessed July 15, 2009].

Hanning Zhou, Zhengyou Zhang & Thomas Huang, 2004. Visual echo cancellation in a projector-camera-whiteboard system. In *2004 International Conference on Image Processing, 2004. ICIP '04.* Singapore, pp. 2885-2888. Available at: http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F9716%2F30680%2F01421715.pdf%3Farnumber%3D1421715&authDecision=-203 [Accessed July 2, 2009].

He, L., Liu, Z. & Zhang, Z., 2002. Why take notes? use the whiteboard capture system. *PROC. ICASSP, 2003*, 2003, 776--779.

Intel®, 2009. Intel® IPP - Open Source Computer Vision Library (OpenCV). Available at: http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-intel-ipp-open-source-computer-vision-library-opencv-faq/ [Accessed July 2, 2009].

Kai Li's Research Group & Wallace, G., Multi-Cursor Window Manager. Available at:

http://multicursor-wm.sourceforge.net/ [Accessed September 28, 2009].

Kaindl, G., 2006. Touché Framework. Available at: http://gkaindl.com/software/touche [Accessed July 2, 2009].

Kempen, E.V., 2008. Blob detection V: growing regions algorithm. Available at: http://geekblog.nl/entry/24 [Accessed July 2, 2009].

Lee, J.C., 2008. Demos Wii Remote hacks | Video on TED.com. Available at: http://www.ted.com/index.php/talks/johnny_lee_demos_wii_remote_hacks.html [Accessed July 2, 2009].

Lee, J.C., Johnny Chung Lee - Projects - Wii. Available at: http://johnnylee.net/projects/wii/ [Accessed July 2, 2009].

Miller, D. et al., 2005. How can the use of an interactive whiteboard enhance the nature of teaching and learning in secondary mathematics and modern foreign languages?'. *Becta*. Available at: * http://partners.becta.org.uk/upload-dir/downloads/page_documents/research/bursaries05/interactive_whiteboard.pdf *.

Nigay, L., 2008. OpenInterface.org. Available at: http://www.openinterface.org/home/ [Accessed July 18, 2009].

NOKIA, 2008. Qt - A cross-platform application and UI framework. Available at: http://www.qtsoftware.com/products/ [Accessed July 23, 2009].

NUI Group, 2008. NUI Group - Natural User Interface Group. Available at: http://www.nuigroup.com/ [Accessed July 2, 2009].

NUI Group Authors, 2009. *Multi-Touch Technologies* 1st ed., Available at: http://nuigroup.com/log/nuigroup_book_1/.

QT Nokia, Huihoo - Qt - Cross-Platform Application Framework. Available at: http://docs.huihoo.com/qt/ [Accessed September 23, 2009].

reacTIVision, 2009. TUIO. Available at: http://www.tuio.org/ [Accessed July 2, 2009].

Risley, D., 2001. A CPU History | PCMech. *A CPU History*. Available at: http://www.pcmech.com/article/a-cpu-history/ [Accessed July 18, 2009].

Schmidt, U., 2008. Wiimote Whiteboard | uweschmidt.org. *Wiimote Whiteboard*. Available at: http://www.uweschmidt.org/wiimote-whiteboard [Accessed July 2, 2009].

Silva, M., 2008. *Projecto de Quadro Interactivo de Baixo Custo Utilizando o Comando da Wii*. Master Thesis. Faculdade de Engenharia da Universidade do Porto - FEUP. Available at: http://paginas.fe.up.pt/~ee06019/DISS/principal.php [Accessed July 2, 2009].

Smith, B.B., 2008. BBTouch. Available at: http://benbritten.com/blog/bbtouch-quick-start/ [Accessed July 2, 2009].

Soares, C. et al., 2009. Um quadro interactivo: Comparação de Algoritmos de visão para detecção de interacções. *Proceedings of CISTI 2009 – 4ª Conferência Ibérica de Sistemas e Tecnologias de Informação*. Available at: http://dei.fe.up.pt/cisti2009/modules/request.php?module=oc_program&action=summary.php&id=97.

Tbeta, 2008. Tbeta. Available at: http://ccv.nuigroup.com/ [Accessed July 2, 2009].

The Center For New Music and Audio Technology (CNMAT), 2002. Introduction to OSC | opensoundcontrol.org. Available at: http://opensoundcontrol.org/introduction-osc [Accessed September 28, 2009].

The National Centre for Languages, 2007. Using the interactive whiteboard in the 14 to 19 classroom. Available at: http://www.cilt.org.uk/14to19/ict/whiteboard/whiteboard.htm.

Touchlib, Touchlib. Available at: http://nuigroup.com/touchlib/ [Accessed July 2, 2009].

TUIO.org, TUIO Protocol Specification 1.1. *TUIO Protocol Specification 1.1*. Available at: http://www.tuio.org/?tuio11 [Accessed September 22, 2009].

Vadim Pisarevsky, 2007. Introduction to OpenCV. Available at: http://unjobs.org/authors/vadim-pisarevsky [Accessed July 22, 2009].

Varcholik, P., 2008. Bespoke Software » Multi-Touch. Available at: http://www.bespokesoftware.org/wordpress/?page_id=41%20/ [Accessed July 21, 2009].

Wallin, D., 2008. Wallin's Webpage - Touchlib. Available at: http://www.whitenoiseaudio.com/touchlib/ [Accessed July 2, 2009].

Anexos

Anexo 1: Descrição das várias concepções de superfícies adequadas ao Multi-toque.

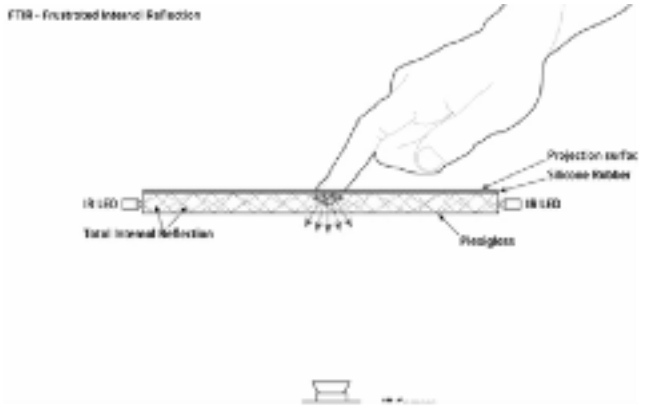


Figura 1: Esquema do funcionamento do FTIR [NUI Group Authors, 2009, p.9]

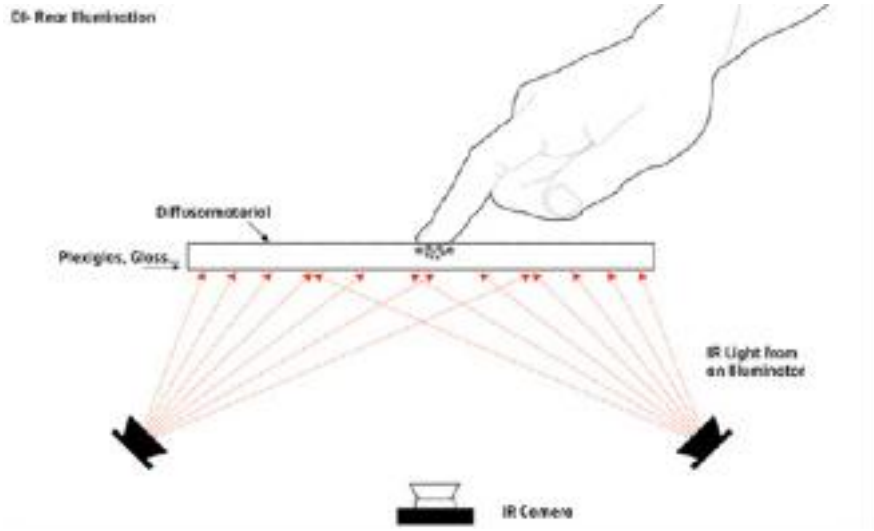


Figura 2: Esquema do funcionamento do Rear DI [NUI Group Authors, 2009, p.14]

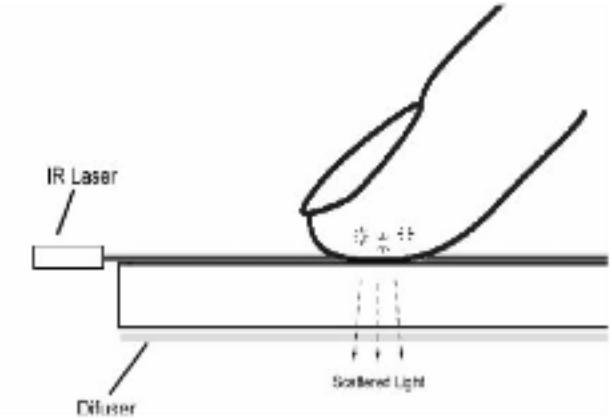


Figura 3: Esquema de funcionamento do LLP [NUI Group Authors, 2009, p.15]

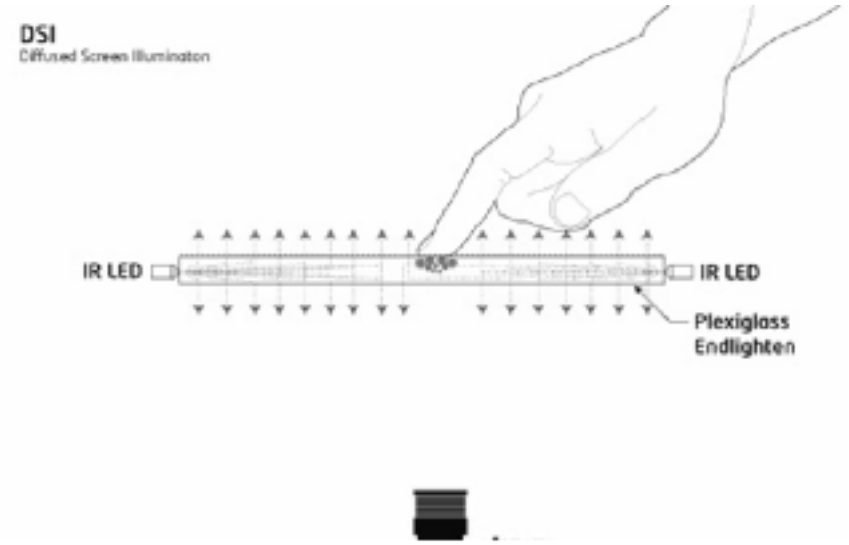


Figura 4: Esquema de funcionamento do DSI [NUI Group Authors, 2009, p.17]

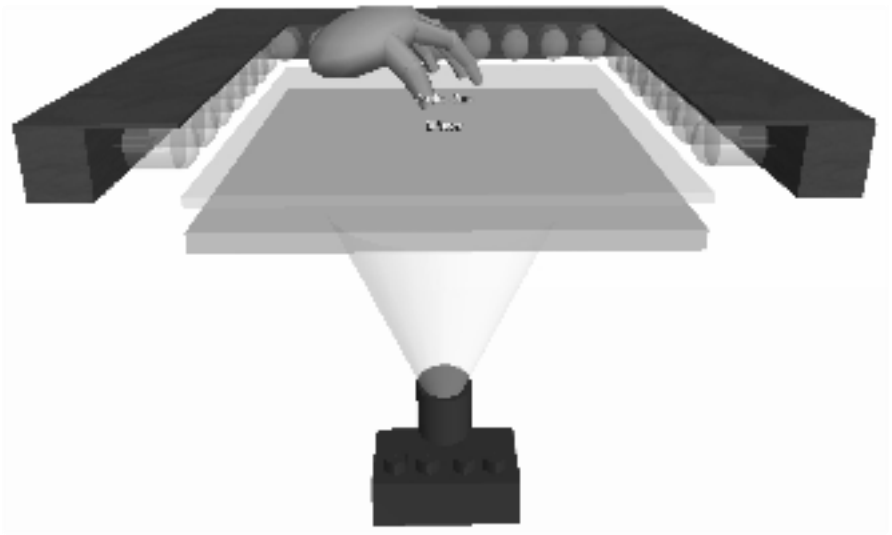


Figura 5: Esquema do funcionamento do LED-LP [NUI Group Authors, 2009, p.18]

Anexo 2: Resultados detalhados das avaliações dos algoritmos

Primeira Experiência

Tabela 1: Valores recolhidos nos testes de avaliação do consumo da aplicação

	video 1	video 2	video 3	video 4	video 5	video 6	Média / Algoritmo
A1	5,50E+05	6,00E+05	5,58E+05	7,73E+05	7,59E+05	7,58E+05	6,79085E+05
A2	5,88E+05	5,60E+05	5,49E+05	8,09E+05	7,93E+05	7,68E+05	6,78026E+05
A3	5,89E+05	5,73E+05	5,85E+05	8,28E+05	8,02E+05	7,74E+05	6,81681E+05
A4	5,67E+05	5,57E+05	5,63E+05	7,73E+05	8,07E+05	8,00E+05	6,70280E+05
A5	5,61E+05	5,29E+05	5,53E+05	7,77E+05	7,71E+05	7,86E+05	6,65776E+05
Média / Video	5,71E+05	5,64E+05	5,61E+05	7,92E+05	7,86E+05	7,77E+05	

Deep=5 Step=3 JumpValue=1 Thresolhd = 50

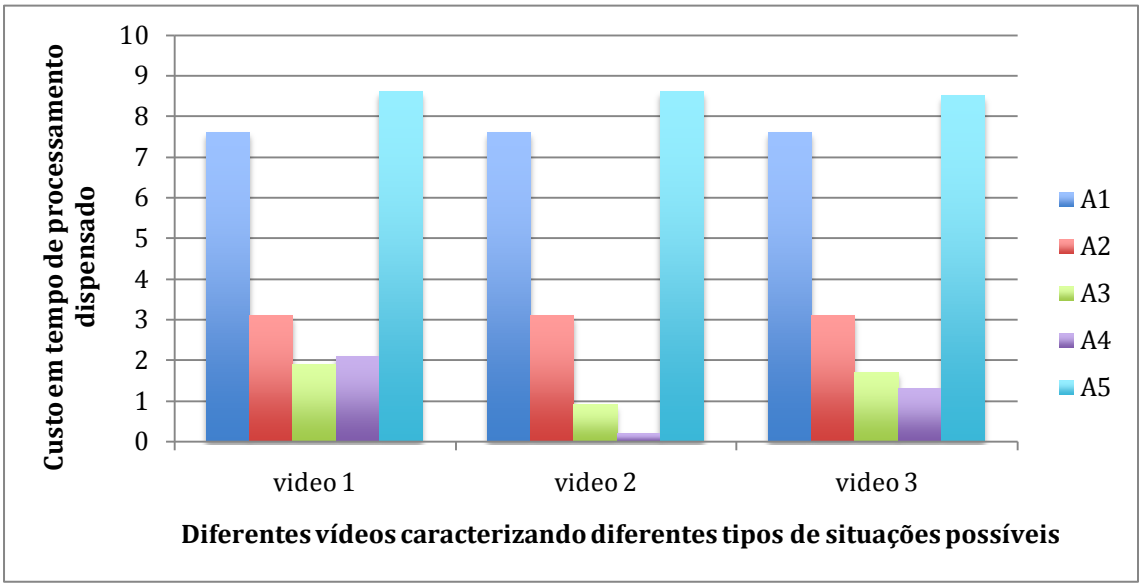


Figura 6: Ilustração do consumo da aplicação entre diferentes Algoritmos do vídeo 1 até ao vídeo 3

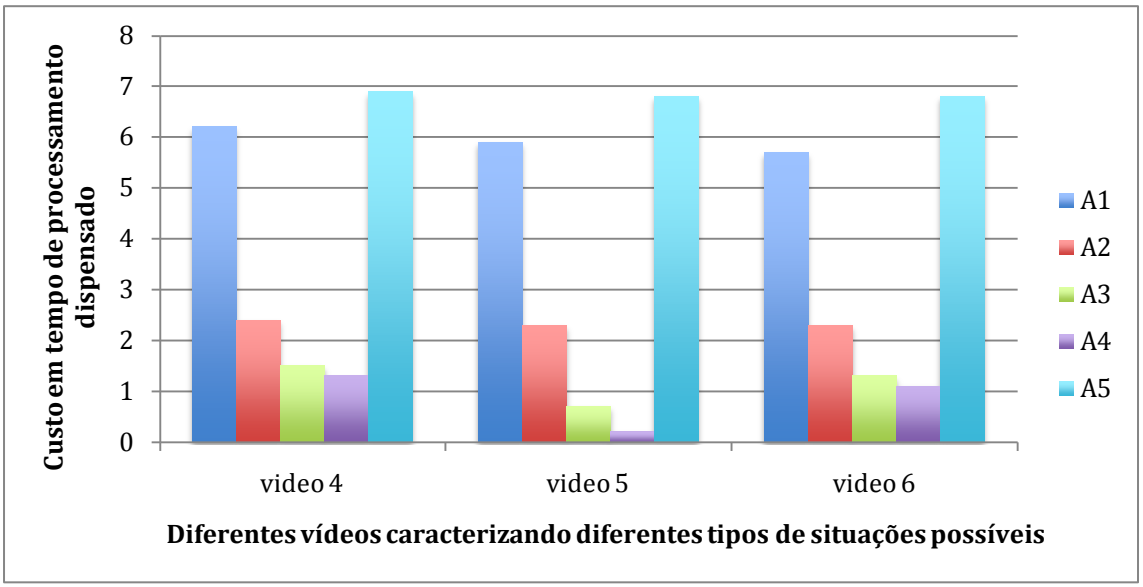


Figura 7: Ilustração do consumo da aplicação entre diferentes Algoritmos do vídeo 4 até ao vídeo 6

Tabela 2: Valores recolhidos através da avaliação da aplicação com o Shark, seguindo diferentes critérios, como o tempo, a carga máxima, mínima, média ou total.

	Limit 500 Frames			Thresolhd = 50			
A1	video 1	video 2	video 3	video 4	video 5	video 6	MEDIA
Total	550318,762	600287,341	557530,841	772984,882	758793,64	757881,677	679084,509
Average	879,103	858,78	926,131	984,694	942,601	848,692	902,617
min	0,922	1,361	3,21	8,087	10,901	18.097	
max	2726,807	2591,846	2563,501	4638,531	4596,191	4.748.096	
time	7,6	7,6	7,6	6,2	5,9	5,7	
	859/11254	852//11202	850/11241	895/14469	859/14560	840/14787	

	Limit 500 Frames			Thresolhd = 50			
A2	video 1	video 2	video 3	video 4	video 5	video 6	MEDIA
Total	587705,493	559812,61	548666,718	809420,856	793499,066	768347,034	678026,2635
Average	961,975	989,068	866,772	1092,336	1005,702	997,853	993,4605
min	0,891	3,558	6,726	36,804	13,159	6,763	
max	2547,485	2655,963	2616,29	4173,669	4195,898	4375,659	
time	3,1	3,1	3,1	2,4	2,3	2,3	
	331/10784	331/10796	326/10654	342/14021	325/14081	324/14264	

	Limit 500 Frames		Step=3	Thresolhd = 50			
A3	video 1	video 2	video 3	video 4	video 5	video 6	MEDIA
Total	589197,51	572525,427	584764,783	827902,71	801824,829	774165,344	681681,427
Average	939,709	985,414	977,868	1140,362	1083,547	939,521	981,641
min	0,653	13,94	0,787	8,771	5,884	15,387	
max	2931,891	2697,357	2905,017	5120,111	4303,534	3638,782	
time	1,9	0,9	1,7	1,5	0,7	1,3	
	201/10418	96/10419	176/10498	214/13841	102/13854	193/14622	

	Limit 500 F.	Deep=5	Step=3	JumpValue=1		Thresolhd = 50	
A4	video 1	video 2	video 3	video 4	video 5	video 6	MEDIA
Total	567152,563	557456,671	563154,486	773408,136	806730,92	800294,092	670280,3495
Average	869,866	891,931	961,014	960,755	979,042	1037,995	960,8845
min	1,581	1,08	21,289	5,365	15,802	10,425	
max	2569,8	2529,297	2653,229	3368,665	4594,251	4829,37	
time	2,1	0,2	1,3	1,3	0,2	1,1	
	219/10486	22/10320	140/10472	183/13881	29/13696	159/14062	

	Limit 500 Frames			Thresolhd = 50			
A5	video 1	video 2	video 3	video 4	video 5	video 6	MEDIA
Total	560982,275	529487,854	552549,683	777233,612	770570,056	785984,076	665776,1655
Average	884,83	851,267	867,425	956,007	920,633	945,829	902,7315
min	1,526	1,984	2,38	7,52	10,931	22,51	
max	2652,606	2865,509	2386,554	4665,222	4543,689	3671,1	
time	8,6	8,6	8,5	6,9	6,8	6,8	
	1036/12077	1059/12281	1035/12198	1074/15538	1071/15709	1070/15816	

	video 1	video 2	video 3	video 4	video 5	video 6
MEDIA GERAL	571071,3206	563913,9806	561333,3022	792190,0392	786283,7022	777334,4446

Segunda Experiência

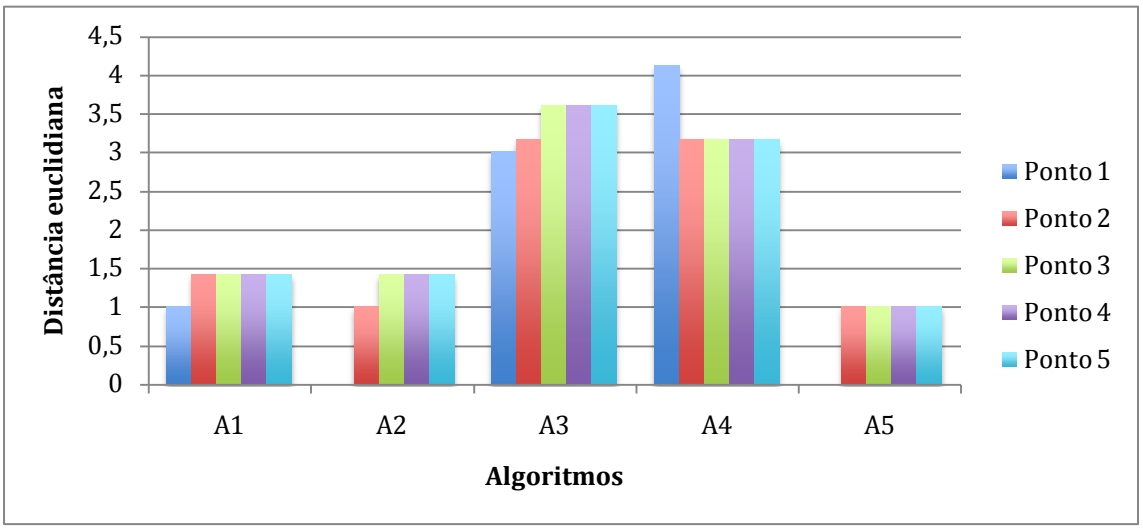


Figura 8: Distância euclidiana representando o erro cometido por cada algoritmo na aproximação de cada um dos pontos

Tabela 3: Valores recolhidos através da avaliação do erro de aproximação cometido por cada um dos algoritmos.

	P1	P2	P3	P4	P5
Ponto (x,y)	82	62	97	393	262	245	396	63	410	398
A1 (x,y)	82	61	96	392	261	244	395	62	409	397
A2 (x,y)	82	62	96	393	261	244	395	62	409	397
A3 (x,y)	82	59	96	390	260	242	394	60	408	395
A4 (x,y)	86	61	100	392	265	244	399	62	413	397
A5 (x,y)	82	62	96	393	261	245	395	63	409	398

	A1	A2	A3	A4	A5
Ponto 1	1	0	3	4,12310563	0
Ponto 2	1,41421356	1	3,16227766	3,16227766	1
Ponto 3	1,41421356	1,41421356	3,60555128	3,16227766	1
Ponto 4	1,41421356	1,41421356	3,60555128	3,16227766	1
Ponto 5	1,41421356	1,41421356	3,60555128	3,16227766	1

Terceira Experiência

Tabela 4: Variação do Salto no A3 e estudo da influência que terá na precisão como no tempo de processamento

A3	Video 3	500 frames				
Jump	1	2	4	8	16	32
time	4,8	2,6	1,3	0,9	0,5	0,2
	535/11069	286/10822	134/10673	94/10579	52/10474	25/10447

Circle

A3	Thresolhd 100						Ponto Real
Jump	1	2	4	8	16	32	
Precisão	4,123105626	4,123105626	5	5	9,219544457	6,08276253	0
x	269	269	271	272	277	274	268
y	198	198	198	199	200	201	202

Quadrado

A3	Thresolhd 100						Ponto Real
Jump	1	2	4	8	16	32	
Precisão	6	6	6	6	6	6	0
x	226	226	226	226	226	226	226
y	206	206	206	206	206	206	212

elipse

A3	Thresolhd 100						Ponto Real
Jump	1	2	4	8	16	32	
Precisão	5,099019514	5,099019514	5,099019514	7,071067812	13,92838828	5,099019514	0
x	239	239	239	243	251	239	238
y	215	215	215	215	215	215	220

elipse 2

A3	Thresolhd 100						Ponto Real
Jump	1	2	4	8	16	32	
Precisão	17	17	16,2788206	15,5241747	15,5241747	12	0
x	247	247	250	251	251	247	247
y	196	196	197	198	198	201	213

losango 217x220

A3	Thresolhd 100						Ponto Real
Jump	1	2	4	8	16	32	
Precisão	17,72004515	17,72004515	17,72004515	17,72004515	16,55294536	16,55294536	0
x	222	222	222	222	224	224	217
y	203	203	203	203	205	205	220

rectangulo 244x148

A3	Thresolhd 100						Ponto Real
Jump	1	2	4	8	16	32	
Precisão	13,92838828	13,03840481	13,03840481	13,92838828	13,92838828	13,92838828	0
x	249	245	245	249	249	249	244
y	135	135	135	135	135	135	148

Tabela 5: Variação da Profundidade da espiral no A4

A4	Video 1	500 frames				
deep	1	2	4	8	16	32
Sucess	10	12	32	47	47	47
Fail	90	88	68	53	53	53

A4	Video 2	500 frames				
deep	1	2	4	8	16	32
Sucess	29	57	93	99	99	99
Fail	71	43	7	1	1	1

A4	Video 3	500 frames				
deep	1	2	4	8	16	32
Sucess	26	42	67	84	89	89
Fail	74	58	33	16	11	11

Tabela 6: Comparação entre o consumo de CPU entre uma pesquisa linear e em espiral, variando o numero de níveis.

	CPU A4	MB/s		
Niveis	Total	Average	Min	Max
1	189366	1015	46	2983
2	210725	829	18	3235
4	209146	846	11,6	2905
8	208374	828	17	3108
16	224392	1034	6	3630
32	225870	934	8	3524

	CPU A1			
	Total	Average	Min	Max
	212632	814	15	3248
	214978	820	8	2913
	203898	853	16	3354
	207689	820	11	4605
	176027	796	19	4375
Media	207689	820	15	3354

Video single touch
20s de amostragem

Quarta Experiência

Tabela 7: Estimativa da precisão por algoritmo variando a forma geométrica do PI

	Coordendadas		
Ref.	x	y	
Circulo	268	202	
Elipse	238	220	
Quadrado	226	212	
Rectangulo	244	148	
Losango	217	220	
A1	x	y	precisão
Circulo	268	202	0
Elipse	238	220	0
Quadrado	225	212	1
Rectangulo	244	148	0
Losango	218	220	1
A2	x	y	precisão
Circulo	272	202	4
Elipse	234	220	4
Quadrado	224	212	2
Rectangulo	240	148	4
Losango	218	220	1
A3	x	y	precisão
Circulo	277	200	9,219544457
Elipse	251	215	13,92838828
Quadrado	234	207	9,433981132
Rectangulo	249	135	13,92838828
Losango	224	205	16,55294536
A4	x	y	precisão
Circulo	277	200	9,219544457
Elipse	251	215	13,92838828
Quadrado	234	207	9,433981132
Rectangulo	249	135	13,92838828
Losango	224	205	16,55294536
A5	x	y	precisão
Circulo	268	202	0
Elipse	238	220	0
Quadrado	226	212	0
Rectangulo	244	148	0
Losango	219	220	2

Threshold 100
Step 2
Deep 8
Jump Value 1

Quinta Experiência

Tabela 8: Recolha de amostras do CPU da nossa aplicação comparativamente com o CCV (ou Tbeta). As 10 primeiras amostras retratam um sistema com o máximo de opções seleccionadas enquanto as outras 10 constituem o sistema no seu funcionamento básico.

	Tbeta	A1	A2	A3	A4	A5
1 amostra	71,8	50,9	50,7	50,5	52,4	52,6
2 amostra	72,7	50,3	49,2	51,7	48,8	50,6
3 amostra	68,8	51,1	51,9	49,2	49,5	54,4
4 amostra	73	50,3	49,5	51,4	51,2	50,1
5 amostra	72,3	53,2	52,2	52,3	49,9	51,1
6 amostra	73,3	52,2	51,8	49,3	49,7	52,1
7 amostra	72	51	52,8	52,3	50,2	52,1
8 amostra	72,8	50,5	51,4	50,2	51,1	52,6
9 amostra	72,3	49,9	52,6	50,3	51,4	51,3
10 amostra	72	49,5	49,4	52,3	49,7	50,4
1 amostra	64,6	14,3	14,3	14,3	14	16,3
2 amostra	64,5	14,4	14	14,4	13,9	16,8
3 amostra	66,1	14,2	14	14,1	14,2	16
4 amostra	64,1	14,5	14,5	14,3	14,2	16,9
5 amostra	67,8	14,8	14,1	14,6	14,3	16,6
6 amostra	65,6	14,3	13,9	14,9	13,9	16,8
7 amostra	65,3	14,4	14	14,4	13,7	16,2
8 amostra	65,8	14,2	14	14	14	16,2
9 amostra	65,7	14,1	13,9	14,4	14	16,1
10 amostra	66,7	14,7	14,2	14,2	14,1	16,8

MIN	64,1	14,1	13,9	14	13,7	16
MAX	73,3	53,2	52,8	52,3	52,4	54,4
MEDIA	68,3	32,15	31,85	32,05	31,55	33,5

	TBETA	Nosso Sistema
MIN	64,1	14
MAX	73,3	52,8
MEDIA	68,3	32,05

Anexo 3: Resultados detalhados da comparação do protótipo LoCoBoard com sistemas equivalentes

Tabela 9: Definição dos passos de montagem envolventes em cada um dos sistemas analisados.

Critérios / *Sistemas*	***Usabilidade***
Sistema LoCoBoard (Soares 2009)	- Ligar a câmara - Ligar a nossa aplicação - Proceder à calibração - Sistema pronto para utilizar
Johnny Lee Whiteboard (Lee n.d.)	- Activar o bluetooth - Ligar e emparelhar o Wiimote com o PC - Ligar a aplicação do Quadro Interactivo - Proceder à calibração - Sistema pronto para utilizar
Uweschmidt (Schmidt 2008)	- Activar o bluetooth - Ligar e emparelhar o Wiimote com o PC - Ligar a aplicação do Quadro Interactivo - Proceder à calibração - Sistema pronto para utilizar
CCV ou Tbeta (Tbeta 2008)	- Ligar a câmara - Ligar o servidor TUIO - Ligar a aplicação do CCV - Proceder à calibração - Ligar a aplicação com suporte a TUIO - Sistema pronto para utilizar
Touchlib (Touchlib n.d.)	- Integração do código fonte num IDE C++ - Compilar o mesmo - Ligar a câmara - Ligar o servidor TUIO - Executar o código compilado - Proceder à calibração - Ligar a aplicação com suporte a TUIO - Sistema pronto para utilizar