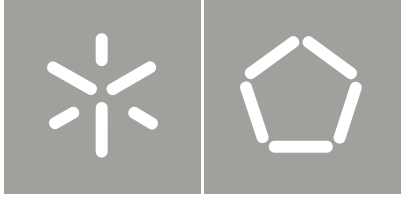
Universidade do Minho
Escola de Engenharia
Departamento de Electrónica Industrial

Tomé Pereira da Silva

Desenvolvimento de Plataforma Móvel para Futebol Robótico

Tomé Pereira da Silva Desenvolvimento de Plataforma Móvel para Futebol Robótico
UMinho | 2010

Setembro de 2010



Universidade do Minho
Escola de Engenharia
Departamento de Electrónica Industrial

Tomé Pereira da Silva

Desenvolvimento de Plataforma Móvel para Futebol Robótico

Tese de Mestrado em
Ciclo de Estudos Integrados Conducentes ao
Grau de Mestre em Engenharia Electrónica Industrial e Computadores

Trabalho efectuado sob a orientação do
Professor Doutor António Fernando Macedo Ribeiro

Setembro de 2010

Genius is one percent inspiration, ninety-nine percent perspiration.

(Thomas A. Edison)

Agradecimentos

Gostaria de expressar o meu reconhecimento e gratidão a algumas pessoas, cujo apoio tornou possível a realização deste trabalho.

Em primeiro lugar, aos meus familiares, em especial aos meus pais Albano Silva e Nazaré Pereira, por todo o apoio e compreensão durante todos os anos de estudo. À minha irmã, Lucília Silva, que sempre me apoiou em todas as fases da realização deste trabalho.

Quero agradecer ao meu orientador, Doutor Fernando Ribeiro, por todo o apoio dado, desde as condições do laboratório, à confiança depositada em mim, à orientação do trabalho e à disponibilidade demonstrada em fases mais complicadas e cruciais.

A toda a equipa de futebol robótico que trabalha ou trabalhou durante todo o tempo em que eu estive presente, nomeadamente, Professor Fernando Ribeiro, Sr. António Sampaio, António Freitas, Rui Simão, Luís Novais, Miguel Oliveira e João Sousa.

Por fim, mas não menos importante, aos meus colegas e amigos, em especial a António Freitas e João Sousa, por todas as criticas construtivas do trabalho, mas principalmente pelo incentivo e apoio dado em todos os momentos menos bons durante o desenvolvimento desta dissertação.

Resumo

A robótica de hoje em dia tem inúmeras aplicações práticas, desde a ajuda prestada ao Homem, até situações em que a precisão e a repetibilidade a torna num grande instrumento de trabalho em diversificadas áreas. Em certos casos, em que o meio ambiente que engloba o agente não é totalmente controlado, este tem que se adaptar ao meio envolvente para finalização da sua determinada tarefa. Esta última situação é a mais complexa, mas é também a situação em que se insere o principal objectivo desta dissertação - a construção de um robô autónomo capaz de jogar futebol.

O trabalho apresentado, engloba tanto a concepção como a construção de um protótipo de um robô futebolista, com *software* capaz de controlar o robô autonomamente, assim como software de apoio às competições. Na construção do robô é analisada desde a estrutura, forma, disposição dos componentes e materiais usados; o software é desenvolvido desde a raiz numa nova estrutura organizada; por fim, mas igualmente importante, é implementado software para a comunicação com *hardware*, para comunicação em rede, processamento de imagem entre outros módulos necessários ao bom funcionamento do robô.

No final, são apresentados alguns aspectos críticos de aperfeiçoamento de todo este trabalho, assim como soluções futuras para os problemas encontrados.

Palavras-Chave: RoboCup®, Robótica, Robô, Linux, C++, Threads, Porta série, USB-I^2C (*Universal serial Interface - Inter-Integrated Circuit*), Visão por computador, processamento de imagem, Segmentação de cores, Calibração de cores, Pesquisa radial e localizada, comunicação *wireless*, Monitorização.

Abstract

Nowadays, robotics has numerous practical applications, from help to humans, to situations where accuracy and repeatability becomes a great tool to work in several different areas. In some cases, when the agent works on uncontrolled environments, he has to adapt itself completely to that environment or to its particular task. This becomes more complex, but it is also the main goal of this thesis - the construction of an autonomous robot, able to play football coping with the RoboCup rules.

This thesis work here presented encloses the robot football player prototype design, with software that can autonomously control the robot, and software to support the competition in which it operates. The robot is analyzed regarding design, structure, shape and components arrangement, as well as the materials used. Software was developed in a new organizational structure from scratch and is also explained on this thesis. Several software modules were created, from the network communication, to hardware control, image processing as well as other modules necessary to manage the real game.

In the last chapters, critical aspects are described and discussed, as well as future solutions to problems encountered during this whole process.

Keywords: RoboCup ®, Robotics, Robot, Linux, C++, Threads, Serial Port, USB-I^2C (Universal Serial Interface - Inter-Integrated Circuit), Computer Vision, Image Processing, Color Segmentation, Color Calibration, Radial and Localized Research, Wireless Communication, Monitoring.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Acrónimos

AP →Access Point

BGRA →Blue Green Reed Alpha

CAMBADA →Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture

CAN →Controller Area Network

COPS →Cooperative Soccer Playing Robots

FPS →Frames per second

GUI →Graphical User Interface

I2C →Inter-Integrated Circuit

IEETA →Instituto de Engenharia Electrónica e Telemática de Aveiro

IP →Internet Protocol

JPEG →Joint Photographic Experts Group

LCD →Liquid Cristal Display

LiPo →Lithium-ion polymer batteries

LUT →Look Up Table

MSL →Middle Size League

openCV →Open Source Computer Vision

RGB →Red Green Blue

RTAI →Real-Time Application Interface

TCP →Transmission Control Protocol

UDP	→User Datagram Protocol
UE	→Unidades Espaciais
USB	→Universal serial Interface
UTP	→Unshielded Twisted Pair
V	→Volt ou vóltio
W	→Watt

Capítulo 1

Introdução

Um r*obô* é um dispositivo mecânico que por vezes se assemelha a um ser humano e é capaz de executar uma variedade de tarefas humanas, muitas vezes complexas, estando ele no comando ou executando as respectivas tarefas para as quais foi programado com antecedência.

O termo, *robô*, foi criado por um escritor Checo *Karel Čapet* [1-3] a partir da palavra *robota* que na sua língua significa “trabalho forçado” [1] .Apesar deste termo apenas ter sido criado em 1921, desde muito cedo foram criados diversos dispositivos autónomos capazes de executar tarefas pré-determinadas.

O primeiro grande exemplo aconteceu muito remotamente em aproximadamente 1400 AC, pelos Babilónios que consistiu na invenção de um mecanismo de relógio de água [4].

No entanto a sua evolução tem acontecido ao longo dos anos e de tal modo que está previsto para em 2050 a realização de um jogo de futebol entre a melhor equipa do mundo da respectiva data e a melhor equipa do mundo de robôs futebolistas[5]. Para impulsionar e desenvolver esta evolução existe anualmente uma competição denominada “*RoboCup*®” [5] em que participam

diversas equipas com diferentes robôs, competindo por um honroso lugar de pódio, fruto da sua dedicação, trabalho, pesquisas, experiências, que vão permitindo passo a passo avanços tecnológicos e científicos significativos. Neste evento existem vários tipos de competições robóticas[6], entre estas, o futebol robótico MSL (*Middle Size League*) [7].

Apesar da ideia generalizada ser apenas uma competição, a participação em eventos deste tipo, leva a que as equipas de investigadores procurem soluções para as dificuldades que encontram à medida que o projecto avança e para que possam levar a bom termo a execução do mesmo projecto antes e durante as competições. Estas soluções são, muitas vezes, respostas a problemas da actualidade e são então aproveitadas pela indústria para a criação e desenvolvimento de novos produtos que tem como objectivo facilitar, ao máximo, a vida do Homem. Um exemplo desta situação é o protótipo e consequente patente da cadeira de rodas omnidireccional desenvolvida pela empresa SAR (Soluções de Automação e Robótica)[8] que derivou de uma ideia e consequente desenvolvimento de um projecto proveniente de uma equipa de futebol robótico.

É no âmbito desta competição que a equipa de futebol robótico da Universidade do Minho, chamada “Minho Team” se insere, assim como esta tese de dissertação.

1.1 Objectivos do Trabalho

O objectivo desta tese de dissertação visa a criação e construção física de um novo robô futebolista, para corrigir alguns aspectos mais críticos que dificultavam o seu desempenho nas competições. Aspectos estes, como o peso actual do robô que deve ser mais leve, mantendo a sua estrutura robusta e o seu baixo centro de massa. A criação de um modelo exequível e fiável para reprodução sistemática quer do robô, quer de peças de substituição.

É proposto também a implementação de um software capaz de calibrar, controlar e executar um programa “Jogo”, garantindo sempre a fluidez do programa com um *frame rate* mínimo de 30 FPS (Frames Per Second) de processamento. O software deve ser construído numa perspectiva de evolução

futura estando bem identificadas e separadas todas as camadas de abstracção para níveis superiores.

O software pedido deve contemplar os seguintes objectivos:

- Comunicação com todo o *hardware* presente
- Comunicação entre os robôs, monitor e *referee box*[1] através da rede
- Aquisição de imagens com câmara *Firewire*
- Segmentação de imagens
- Pesquisa radial
- Pesquisa localizada
- Estrutura de perspectivas de distâncias
- Base de dados
- Campo virtual
- Monitor

1.2 Estrutura do Trabalho

A figura seguinte (Figura 1.2:1) mostra a solução apresentada para o novo robô, em que a sua disposição e materiais são pensados para um óptimo desempenho. Esta solução usa materiais compósitos para uma diminuição substancial do peso. A disposição dos componentes é toda feita na parte inferior do robô para um baixo centro de massa.

[1] No RoboCup®, a *referee box* é a tecnologia utilizada para auxílio do árbitro, em particular para comunicar as decisões de arbitragem para os robôs e manter um registo de jogo.

Figura 1.2:1 - Desenho protótipo do robô futebolista

O *software* criado está implementado segundo as seguintes camadas superficiais apresentadas na figura seguinte (Figura 1.2:2). Deste modo, é possível isolar cada situação e esta ser tratada separadamente sem influenciar camadas de abstracção diferentes. A própria detecção e correcção de problemas são mais fáceis, pois desta forma o problema fica localizado e focado numa única camada. É possível também a evolução de camadas superiores sem conhecimentos profundos de outras camadas, quer inferiores, quer de outra natureza.

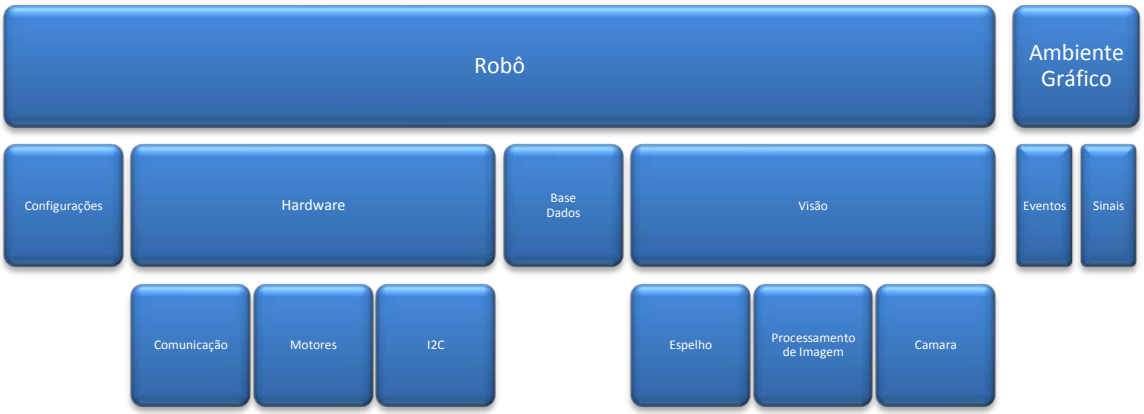


Figura 1.2:2 - Visão superficial sobre o software implementado

Assim, é possível uma evolução controlada e sustentada do software, sem grandes conhecimentos anteriores do software já desenvolvido.

Capítulo 2

Estado da Arte

Actualmente, até a data da escrita desta tese, existem cerca de quinze equipas de futebol robótico da liga média inscritas para a competição da *RoboCup*® em 2010. Este evento, em épocas anteriores, já foi disputado por um maior número de equipas, mas devido ao grau de dificuldade que acresce anualmente e que implica também um grande investimento de recursos materiais, humanos e financeiros torna-se assim, mais uma barreira a ultrapassar pelas equipas para além do desafio científico.

Todas as equipas utilizam diferentes estratégias, abordagens, métodos para resolução dos diversos problemas que existem na concepção de um robô futebolista: desde o tipo do material utilizado, forma do robô, à disposição dos periféricos utilizados, passando pelos dispositivos existentes até ao software segundo as suas ideologias, que pensam ser a mais correcta ou a mais apropriada.

Uma das partes mais importantes de um robô futebolista é a sua programação, mais concretamente a visão, visto que é desta maneira que o robô "vê" e analisa o mundo que o rodeia, se insere e reage segundo o seu modelo. É a parte de todo o processo que ocupa maior tempo computacional,

nunca devendo baixar o seu *frame rate* pois um robô deve tentar reagir em tempo real e nunca em previsão de situações futuras.

De seguida serão analisadas as últimas três equipas que venceram os últimos eventos do RoboCup®, uma das mais importantes e prestigiadas competições de robótica móvel ao nível mundial. A equipa "*RFC Stuttgart*" que venceu em 2009 [9], "CAMBADA " em 2008 [10] e "*Tribots*" que venceram em 2007 e 2006 [11-12].

2.1 RFC Stuttgart

RFC Stuttgart é uma equipa de doze elementos proveniente da Alemanha da Universidade de Stuttgart[13]. É até a data, a equipa vencedora da última edição do RoboCup® 2009 em Graz, (Áustria) [9]. A sua primeira competição foi em 1999 em Estocolmo, (Suécia) [14], com o nome de CoPS (*Cooperative Soccer Playing Robots*).

2.1.1. Estudo do Hardware e estrutura do robô

O seu actual robô, que se apresenta na figura seguinte (Figura 2.1:1) usa uma única estrutura feita em alumínio de 15mm, na qual se vão encaixando as restantes peças [15]. O seu sistema de locomoção é baseado em quatro rodas [15], utilizando motores *brushless* de 200W (Watts) da *Maxon [16],* cada um com o seu respectivo *encoder* e placa controladora. Esta tem um único chuto electromagnético de 400V (Volts)[15] permitindo apenas um chuto, o chamado chuto vertical.

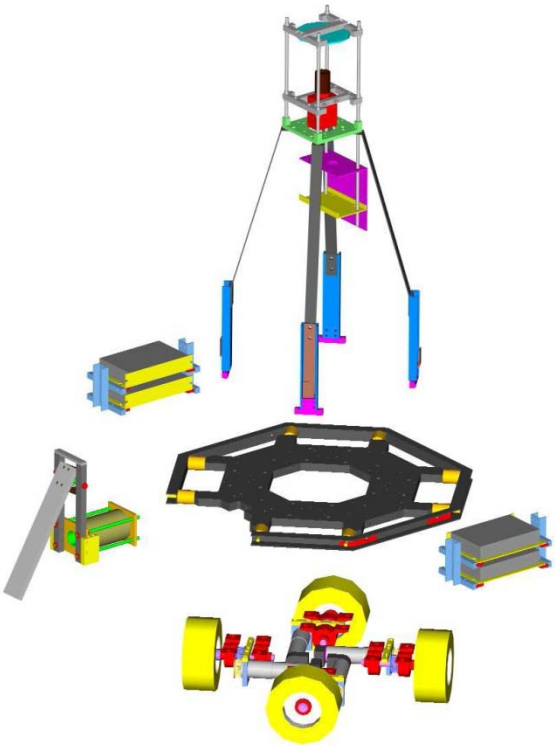


Figura 2.1:1: Desenho tridimensional do robô construído pela equipa RFC Stuttgart [15]

Esta equipa utiliza um portátil Intel® Core™2 Duo com uma câmara FireWire de 400KBits/se e utiliza um barramento CAN (*Controller Area Network*) para comunicar com todos os dispositivos e motores presentes no robô[15]. Para alimentação usa dois conjuntos de baterias, uma de 12V para alimentar o computador, bússola e um *LCD (Liquid Cristal Display)* e o outro conjunto para alimentar o chuto e os motores, sendo esta última de 24V[15].

2.1.2. Estudo do Software

Em relação ao software desenvolvido por esta equipa, é todo ele implementado numa plataforma em Linux e o seu desenvolvimento é feito numa *Framework* de *Qt* em C++ [17].

A organização deste software está representada na figura em baixo (Figura 2.1:2) em que o seu ponto central de funcionamento é o "*World model*" que guarda toda a informação recebida e analisada de vários sensores, "*Sensor data*" e comunicação "*Communication*". Esta informação é então adaptada ao seu mundo modelizado e guardada neste espaço. À informação

recolhida dos vários sensores é feita uma fusão sensorial no "*Data Processor*" que guarda os resultados no "*World model*". Com estes resultados o "*Player*" que representa o tipo de cada jogador, decide a acção a tomar a qual é a comunicada ao "*Navigator*" para controlar esta acção com o trajecto e direcção a seguir [15].

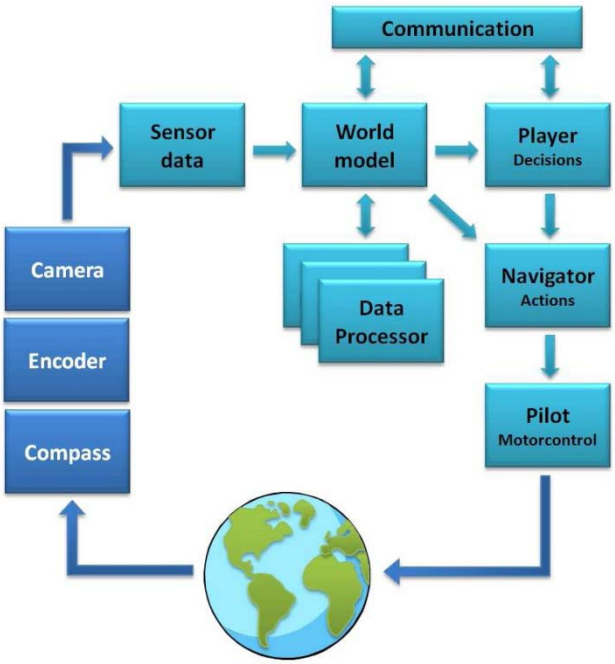


Figura 2.1:2 - Visão superficial sobre o software da equipa RFC Stuttgart [15]

Esta direcção é em coordenadas e é entendida pelo "*Pilot*" que controla os valores a enviar aos respectivos motores e até o chuto deste jogador. Esta equipa utiliza a análise de três sensores, a câmara, os encoders e a bússola.

2.2 CAMBADA

CAMBADA é um acrónimo para *Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture* de uma equipa portuguesa de futebol robótico médio da Universidade de Aveiro [18]. Esta equipa foi criada em 2003 por um grupo de docentes e investigadores do Instituto de Engenharia Electrónica e Telemática de Aveiro (IEETA) e contém agora dezanove elementos. A sua primeira participação no RoboCup® foi em 2004 em Lisboa (Portugal) mas em 2008 conseguiram vencer a prova de futebol robótico da liga média do evento RoboCup® realizado em Suzhou (China) [18].

2.2.1. Estudo do Hardware e estrutura do robô

Esta equipa tem o robô construído em duas bases de alumínio e uma estrutura para suportar a cabeça com a câmara, como mostra a figura seguinte (Figura 2.2:1). A estrutura é circular e é movida por três rodas omnidireccionais [19], com motores de 150W da Maxon [16]. É alimentado por três conjuntos de baterias de 12V, uma para o computador e as outras duas em serie para alimentar os motores, chuto electromagnético, bússola e motor para manuseamento próximo da bola [20]. Possui apenas um único chuto electromagnético que utiliza a energia acumulada até 90V [20].

Utilizam um computador portátil de 12” polegadas com processadores Intel® Core™ 2 Duo e câmaras FireWire [20]. A comunicação com os periféricos é toda feita através de um barramento CAN [20]. Esta equipa possui um módulo independente para leitura da odometria do robô, assim como, uma bússola que é agora substituída por um acelerómetro e giroscópio [20].

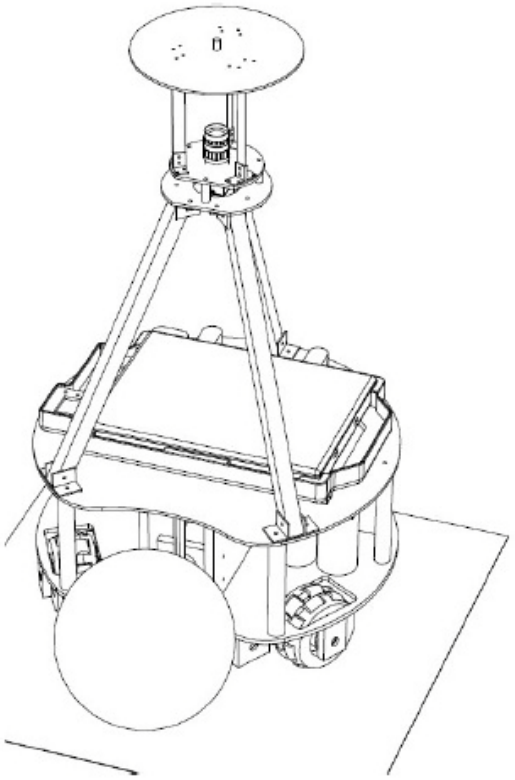


Figura 2.2:1 - Estrutura do robô da equipa CAMBADA [19]

2.2.2. Estudo do Software

O software desenvolvido é todo ele feito numa plataforma em Linux, o seu desenvolvimento é feito numa *Framework* de GTK e é pensado segundo o modelo abaixo apresentado (Figura 2.2:2). Esta equipa implementa uma base de dados em tempo real que é o seu ponto central de todo o processamento [21-22] . Esta implementação é feita através de RTAI (*RealTime Application Interface for Linux*) que é uma camada de abstracção para aplicações em tempo real para o kernel do sistema operativo [23]. Com isto é possível escalonar prioridades de processamento, assim como, garantir tempos críticos para actualização desta base de dados.

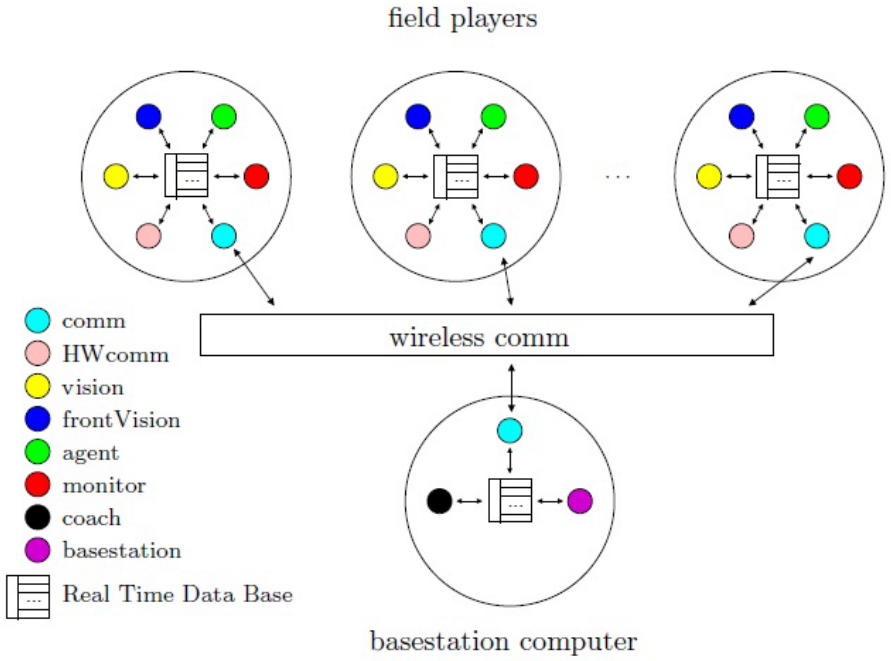


Figura 2.2:2 - Visão superficial do software da equipa CAMBADA [24]

Esta base de dados é acedida para escrita e leitura sincronizada por vários módulos, como a visão, o *hardware*, a comunicação *wireless*, o agente e o monitor [24].

2.3 Brainstormers Tribots

É uma equipa alemã, da Universidade de Osnabrück, do grupo de investigação Neuro-Informáico [25]. Começou a sua experiencia no futebol

robótico em 2002 com quatro robôs protótipos, para em 2004 atingir a sua última versão final deste robô. Antes da participação nas competições de MSL, a equipa participou nas ligas de simulação de futebol, ganhando alguma experiência para as competições de MSL [26-27].

A sua primeira participação no RoboCup® na MSL foi em 2003 em Pádova, Itália, [26-27] mas as suas importantes vitórias aconteceram em 2006 e 2007, em Bremen, Alemanha e Atlanta, Estados Unidos da América respectivamente, onde venceram o torneio de futebol robótico da liga média do RoboCup® [26-27].

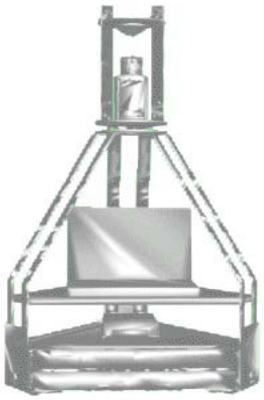
A mesma equipa que começou com dezassete elementos, presentemente, encontra-se com dez elementos e está actualmente a par do melhoramento do software existente, desenvolvendo uma plataforma de robô *open source* que facilitará a novas equipas que pretendam entrar nesta competição [26-27] .

2.3.1. Estudo do Hardware e estrutura do robô

O robô abaixo apresentado (Figura 2.3:1) é o último robô desta equipa presente em competições. A sua última competição foi em 2008 no RoboCup® em Suzhou na China. Este robô tem o formato triangular e usa o mínimo de rodas omnidireccionais, para se movimentar, isto é três. Usa três motores de corrente contínua de 90W da Maxon e uma placa controladora TMC200[2] [28] e consegue atingir velocidades na ordem dos 3.0m/s. Utiliza também um único chuto pneumático que consegue projectar a bola a 3.0 - 4.0 m/s elevando-a até 1m de altura [29].

Esta equipa tem um computador portátil de 1.0Ghz, mas está a substituir este por computadores Mini-ITX Dual Core. Prevêem trocar todos os computadores por processadores Intel® Core™ 2 Duo para a próxima competição. Utilizam duas câmaras Firewire, uma para uma visão omnidireccional e outra para uma visão 3D e reconhecimento de formas na frente do robô. Usam um barramento CAN para ligar todos os dispositivos, assim como, a placa dos motores [29].

[2] TMC200 is a product of the Fraunhofer Institute for Autonomous Intelligent Systems (AIS), Sankt Augustin



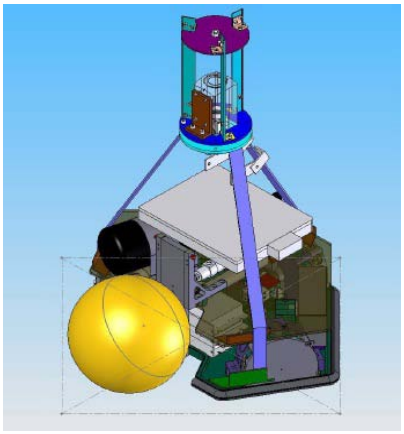


Figura 2.3:1 – (Esquerda) Estrutura usada pela equipa Tribots [30]
Figura 2.3:2 – (Direita) Nova estrutura open source da equipa Tribots [30]

O novo robô no âmbito *open source* é muito parecido com o anterior, mantendo as características principais, mudando apenas alguns aspectos de construção para facilitar a sua construção e montagem por parte de outras equipas [30].

2.3.2. Estudo do Software

O intuito inicial do desenvolvimento do software pela equipa Tribots foi baseado nos princípios de aprendizagem do robô. O software desenvolvido, usa apenas umas única thread que permite modelar o problema de controlar o robô, baseado em decisões de Markov em tempo discreto, que é o princípio básico do *reinforcemente learning [29-31]*.

O software desenvolvido é construído em módulos que permitem facilmente compreender o conteúdo de cada um deles e aplicar diferentes abordagens à mesma situação. Deste modo, é possível comparar e ajustar o nível de aprendizagem de cada um dos módulos em cada acção específica, tornando o robô mais rico em conhecimentos apreendidos.

2.4 Conclusões

Como é natural, cada equipa utiliza os recursos e a forma que pensam ser a mais adequada, no entanto, há que realçar que existem alguns padrões comuns entre estas três equipas analisadas. Todas utilizam câmaras FireWire,

assim como, um barramento CAN para comunicação com todos os periféricos. Em relação ao sistema operativo, todas estas equipas, assim como, a maior parte das equipas da competição utilizam Linux.

Seguidamente apresenta-se uma tabela comparativa (Tabela 1) das principais características das três equipas analisadas, onde se pode facilmente salientar as semelhanças e diferenças entre estas.

Tabela 1 - Tabela comparativa das principais características das equipas analisadas

	RFC Sttutgart	**CAMBADA**	**BrainstormersTribots**
Bases	Alumínio (15mm)	Alumínio	Alumínio
Baterias	NiMh 1 x 12V 9.6Ah (PC) 1 x 24V 3.2Ah (Potência)	NiMh 1 x 9.6V 3.7Ah 2 x 12V 3.7Ah	
Motores	4 x 200W Brushless da *Maxon*	3 x 150W da *Maxon*	3 x 90W da Maxon
Velocidade	5 m/s		3 m/s
Chuto	Electromagnético 400V	Electromagnético 90V	Ar comprimido 3-4 m/s, 1m de altura
CPU	Core 2 Duo 2.2GHz	Intel Core 2 Duo	JVL 1.0GHz (Em substiuição por Core 2 Duo)
Câmara	FireWire 30FPS (400KBits/s)	FireWire	FireWire PFW V500
Barramento	CAN	CAN	CAN
Bússola	HMC 6352	Hitachi HM55B (Em substituição por acelerómetro mais giroscópio)	
Sistema Operativo	Linux	Linux	Linux
Linguagem	C++	C++	C++
Framework	QT4		QT3

Capítulo 2

Existe uma tendência a todas as equipas de ficarem muito parecidas em termos de aspecto e estrutura, uma vez que diversas formas já foram utilizadas e apenas aquelas que se mostram mais eficazes se mantêm ao longo da competição

Capítulo 3

O Robô

Neste capítulo irá ser abordado desde a forma, à construção do robô, passando pelos materiais e técnicas utilizadas na construção das bases, até a disposição destas. Também serão apresentados os periféricos presentes e futuros pois o pensamento desta construção teve sempre em linha de horizonte a evolução desta máquina robótica.

Na sequência deste capítulo, irá ser abordada a comunicação e ligação entre os diversos periféricos, visto que fazem parte integrante desta complexa máquina a que chamamos robô, assim como, a plataforma de desenvolvimento de todo o software.

3.1 Construção do Robô

A construção do robô, continua a manter a mesma forma arredondada[32-33], pois é a forma geométrica que ao mesmo tempo garante maior espaço ocupado em campo, mais área livre para acomodação dos periféricos e assegura o mínimo de pontos críticos, aos quais o robô pode ficar preso em alguma outra estrutura ou robô.

O material utilizado na construção das bases foi um material compósito de fibra de vidro juntamente com resina de epóxi [34]. Para dar o molde que se pretendia foi usado um material de poliestireno extrudido, também vulgarmente conhecido por *roofmate* usado em isolamento térmico de casas. Este material foi moldado numa CNC (Controle Numérico Computadorizado) a laser com formas pretendidas e desenhadas do programa *SolidWorks*® [35]. O material usado no molde apresentou dois problemas, uma vez, que nem sempre conseguiu uma boa absorção da resina, deixando em algumas peças diversos pontos frágeis que facilmente podiam quebrar. Também na junção das lâminas superiores e inferiores ao molde, nem sempre todas as peças ficaram seguras, o que pode provocar problemas em colisões.

As soluções que se apresentam para estas contra partidas são a utilização de um material de molde diferente como madeira de balsa ou contraplacado. A melhor solução é a utilização de espuma de polietileno [36], como mostra a figura seguinte (Figura 3.1:1).

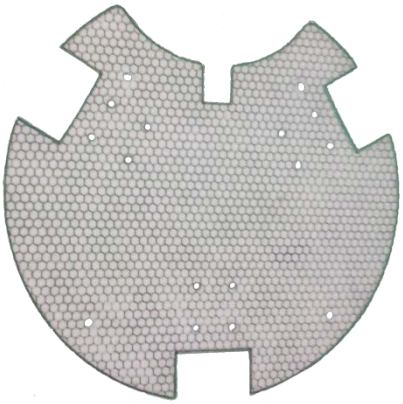


Figura 3.1:1 - Base construída em espuma de polietileno

Estes materiais são mais porosos, levando a uma maior absorção da resina o que dá mais dureza ao material. A moldagem da peça deve apenas ser feita após a construção em bruto do material compósito, pois se esta for feita anteriormente, pode levar a problemas de alinhamentos durante a secagem da resina. Para a evitar que os bordos exteriores das peças se deteriorem, estes devem ser reforçados com um metal rígido ou reforçado com resina epóxi.

A cabeça do robô foi feita em prototipagem rápida [37], o que significa precisão na concepção da peça como se pode verificar na figura seguinte (Figura 3.1:2). Esta se for peça única garante maior estabilidade e rigidez.
A cabeça do robô onde se encontra a visão é segura por quatro pilares ocos de PVC (Policloreto de Vinilo) com um veio roscado dentro de cada tubo de pilar para assegurar que as peças se encontrem bem afixadas e com recursos leves como mostra a figura seguinte (Figura 3.1:2).
Estes pilares são facilmente retirados, separando o robô em duas partes distintas para fácil acomodação e transporte.

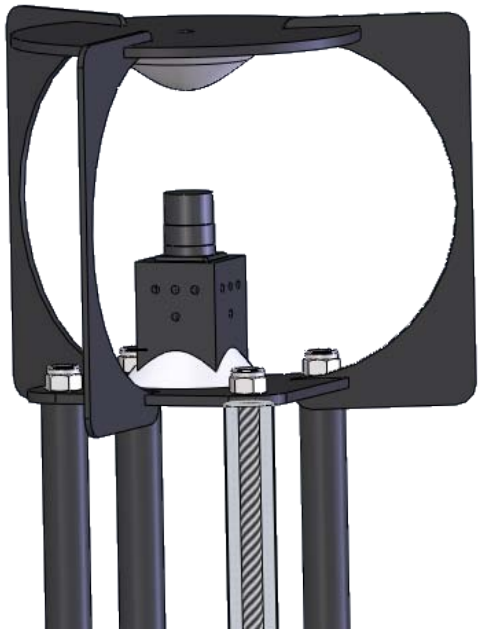


Figura 3.1:2 - Cabeça do robô e suporte da câmara e espelho

O robô é constituído por três bases distintas. A primeira, onde estão seguros os três motores com as respectivas rodas omnidireccionais. Esta base serve também para apoio da bateria de condensadores, das baterias que alimentam o robô e algumas outras placas como mostra a figura seguinte (Figura 3.1:3).

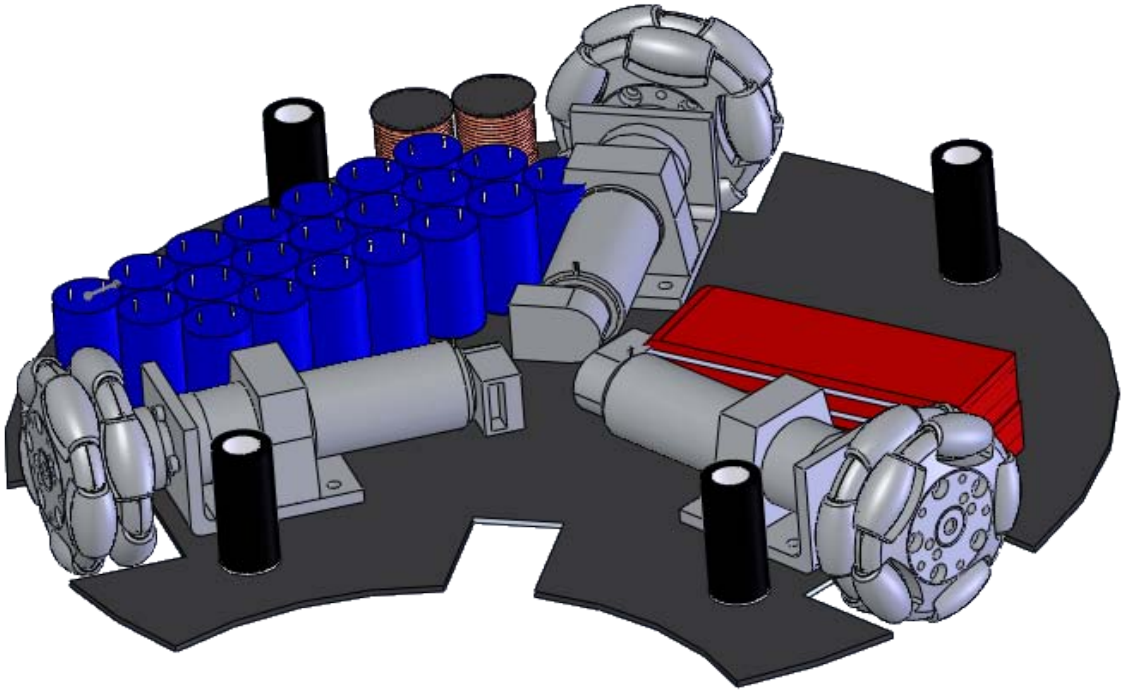


Figura 3.1:3 - Base inferior do robô

A segunda base é onde está apoiada a bobine do chuto por baixo, que executa o chuto da bola, a placa dos chutos, que controla o accionamento das bobines do respectivo chuto, a placa de controlo dos motores, que controla os três motores individualmente, e os receptores infra-vermelhos, que servem para detectar o posicionamento correcto da bola (Figura 3.1:4). Esta base suporta também o computador e a respectiva bateria que estão posicionados verticalmente entre a terceira e a segunda base.

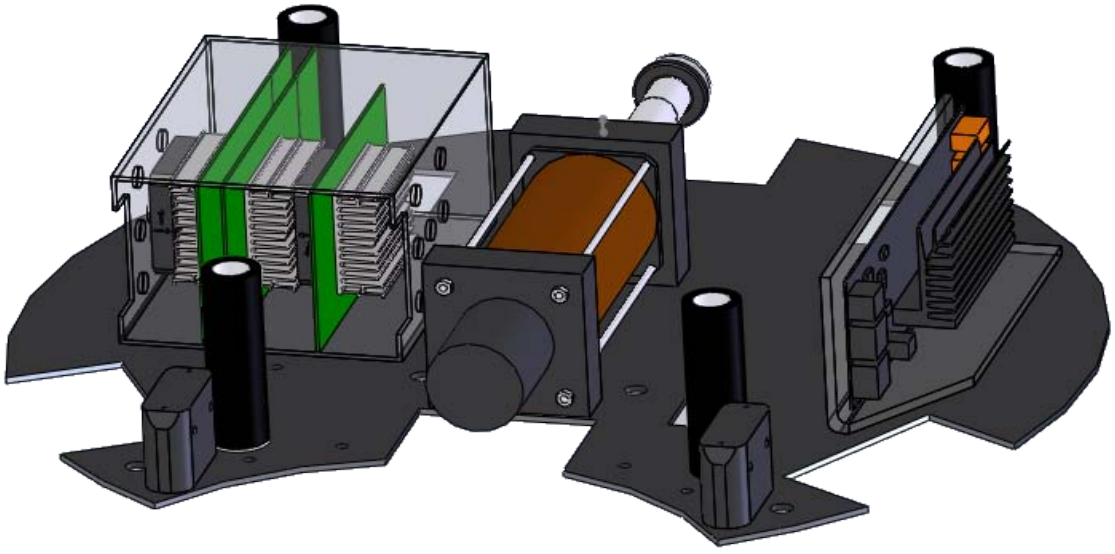


Figura 3.1:4 - Base do centro do robô

Isto é possível pois o computador Intel® Pentium® M de 1000MHz usado utiliza um disco flash que para além de ser mais rápido é *solid state* para protecção em choques. Assim, é possível acomodar ao máximo todo o espaço disponível nas bases do robô.

Seguidamente apresenta-se uma imagem da montagem destas duas bases (Figura 3.1:5), onde se pode verificar o encaixe entre elas, com o uso de veio roscado e tubos de PVC. De notar também o encaixe perfeito e acomodação dos diversos componentes, como das placas dos motores, assim como dos sensores infravermelhos e das rodas, perfeitamente alinhadas.

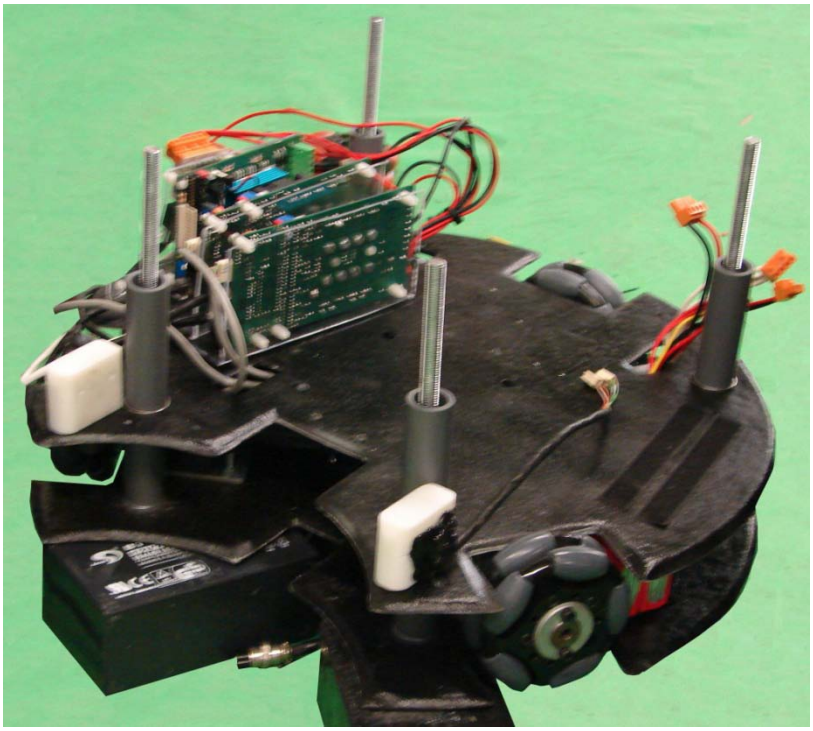


Figura 3.1:5 - Encaixe das duas primeiras bases

A terceira base é onde é apoiado a alavanca do chuto por cima, assim como a sua respectiva bobine de chuto. Esta base serve também para fechar o robô, escondendo toda a electrónica presente dentro deste como mostra a Figura 3.1:6.

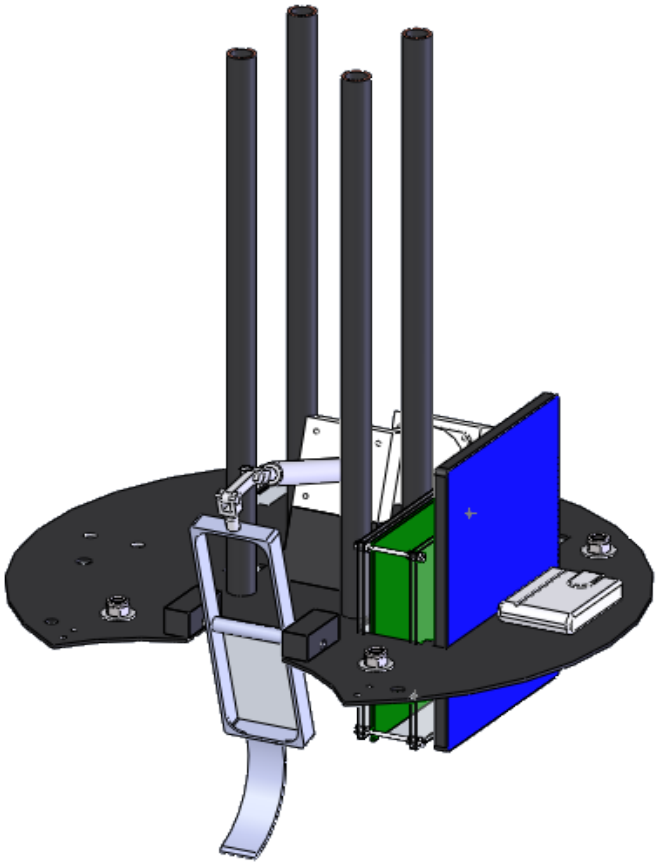


Figura 3.1:6 - Base superior do robô

O resultado final da junção destas três bases encontra-se na figura seguinte, Figura 3.1:7.

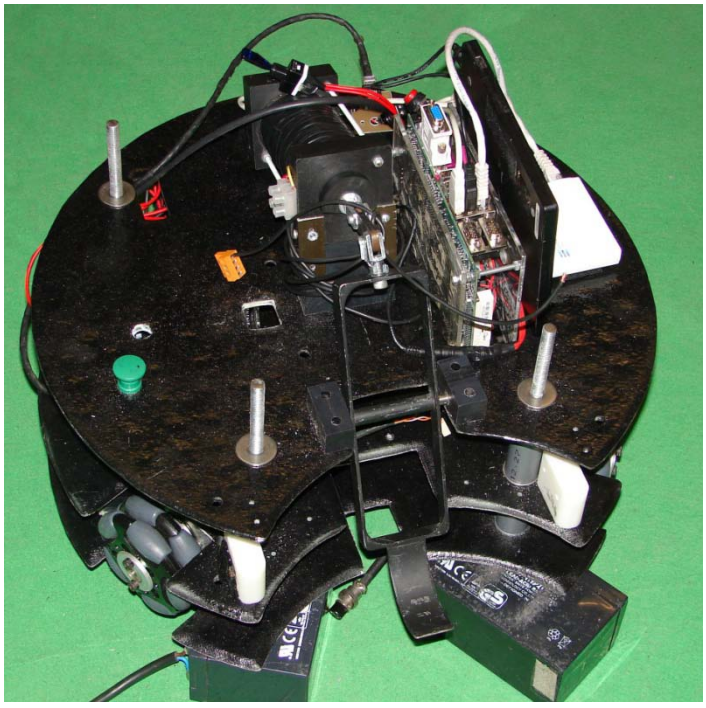


Figura 3.1:7 - Montagem das três bases do robô

Este robô tem duas baterias de lítio, LiPo (Lithium-ion polymer) [38] de 12v, 5Ah ligadas em série, para elevar a tensão para 24V para os motores. Os motores são da *Maxon* e são de 150W [16]. Possui um chuto electromecânico com valores de tensão de 400V já anteriormente desenvolvido pela antiga equipa. Para ligar e comunicar com todos estes dispositivos, é usado um barramento I^2C [39]. Este robô possui uma única câmara *FireWire* Flea® 2 [40] apontada para um espelho omnidireccional para ver todo o ambiente que o rodeia.

Após a montagem de todas estas peças e componentes, o robô protótipo construído fica da seguinte maneira, como mostra a figura seguinte, Figura 3.1:8.

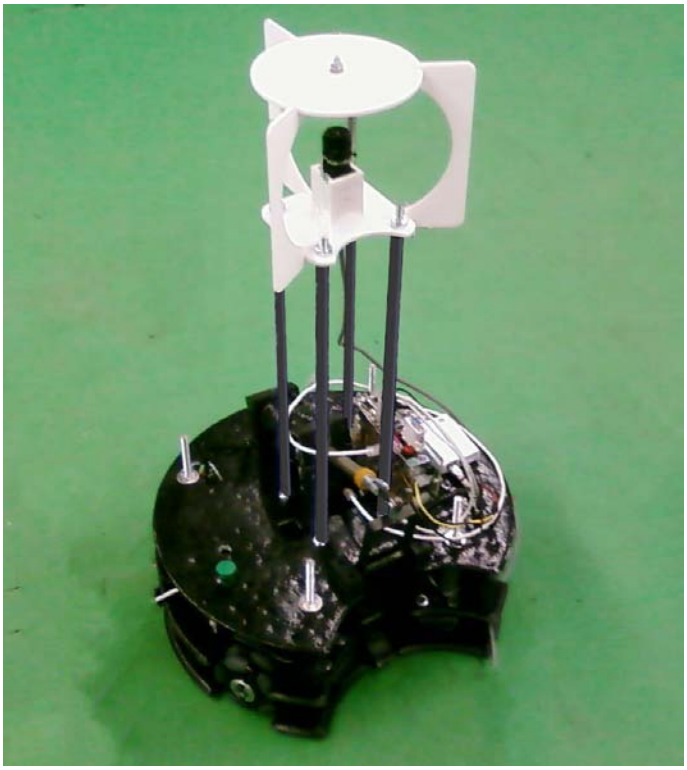


Figura 3.1:8 - Protótipo do robô futebolista

3.2 Comunicação com o hardware

A comunicação com o *hardware* é mais uma parte importante da construção do robô, é através do *hardware* externo ao computador que consegue ler sensores ou activar actuadores para executarem as funções programadas, mais apropriadas ao respectivo momento.

A comunicação com estes dispositivos externos, é actualmente feito através de um barramento I^2C [41] que permite endereçar cada dispositivo e enviar o respectivo comando. Deste modo, é possível ligar todos os dispositivos a dois únicos fios que percorrem todo o robô para a comunicação ser estabelecida de um *master* para os diversos *slaves*. Este tipo de comunicação, no entanto, tem várias falhas, como a falta de um byte de correcção, dado que os dados podem ficar corrompidos durante a transmissão, assim como a impossibilidade de os *slaves* comunicarem informações ao *master* a menos que este tenha pedido a informação e se encontre à espera da respectiva resposta. O conversor utilizado é o conversor USB-I^2C [42] que se apresenta na figura seguinte (Figura 3.2:1).

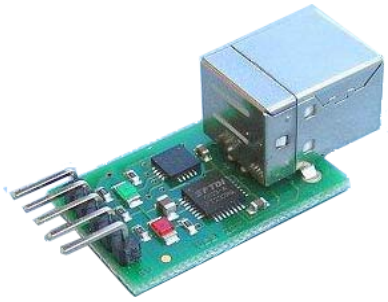


Figura 3.2:1 - Conversor USB-I^2C [42]

Alguns dispositivos mal projectados, quando se encontram desligados bloqueiam o barramento, levando ao bloqueio do respectivo conversor, pois este conversor tem implementado um *firmware* que não possibilita reinicializações bloqueando o seu estado e levando a que este tenha que ser retirado e novamente colocado para retomar a comunicação com os restantes dispositivos. No entanto, na maioria dos casos, estes dispositivos comunicam eficazmente não causando qualquer tipo de problemas.

Mostra-se na figura seguinte as ligações existentes entre os vários dispositivos e o computador (Figura 3.2:2).

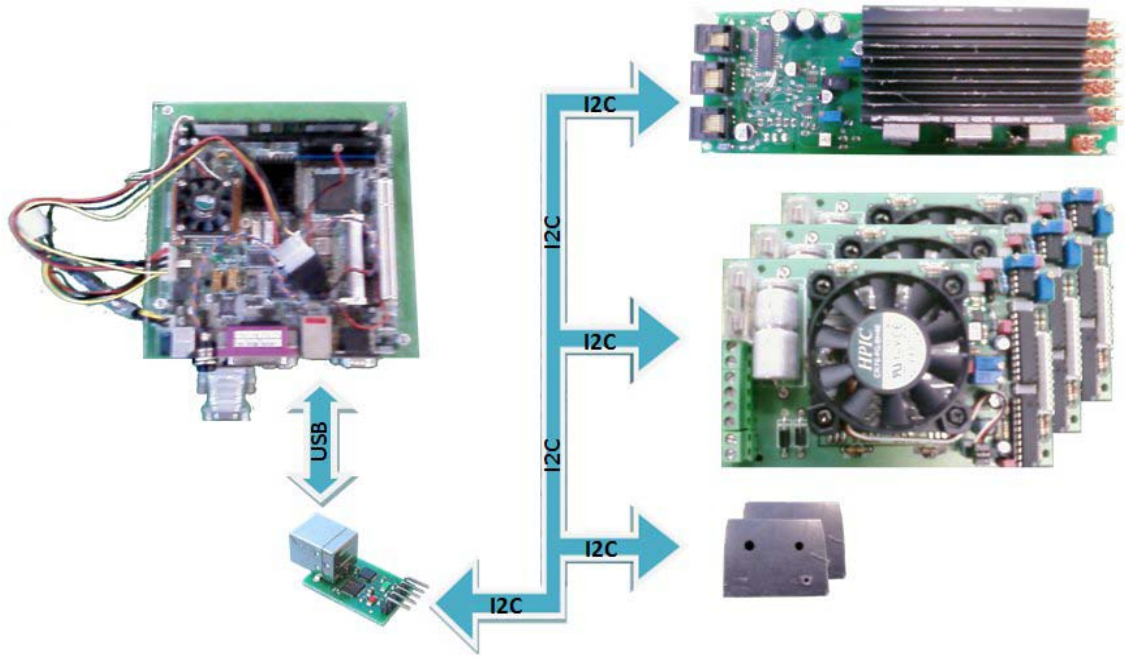


Figura 3.2:2 - Ligações com o hardware

Existe também outro tipo de comunicação, a comunicação entre o computador e a placa dos motores TMC 200 [28] (representada na Figura 3.2:3 seguinte) que é feita através da porta série. Esta comunicação é essencial pois é ela que permite a locomoção do robô.

Figura 3.2:3 - Placa dos motores TMC200 [28]

Num capítulo seguinte será abordada a forma e o algoritmo para estas duas comunicações, assim como, a topologia das respectivas threads da comunicação.

Actualmente encontra-se em desenvolvimento uma nova placa que contém um micro-controlador. A sua única ligação é USB que permite uma fácil interacção com todos os sensores e actuadores, visto que se encontram unicamente ligados a esta placa. Evita-se assim, o uso de barramentos e todas

a complicações que dele advêm. Apresenta-se seguidamente, na Figura 3.2:4 as ligações actuais presentes para o bom funcionamento do robô futebolista.

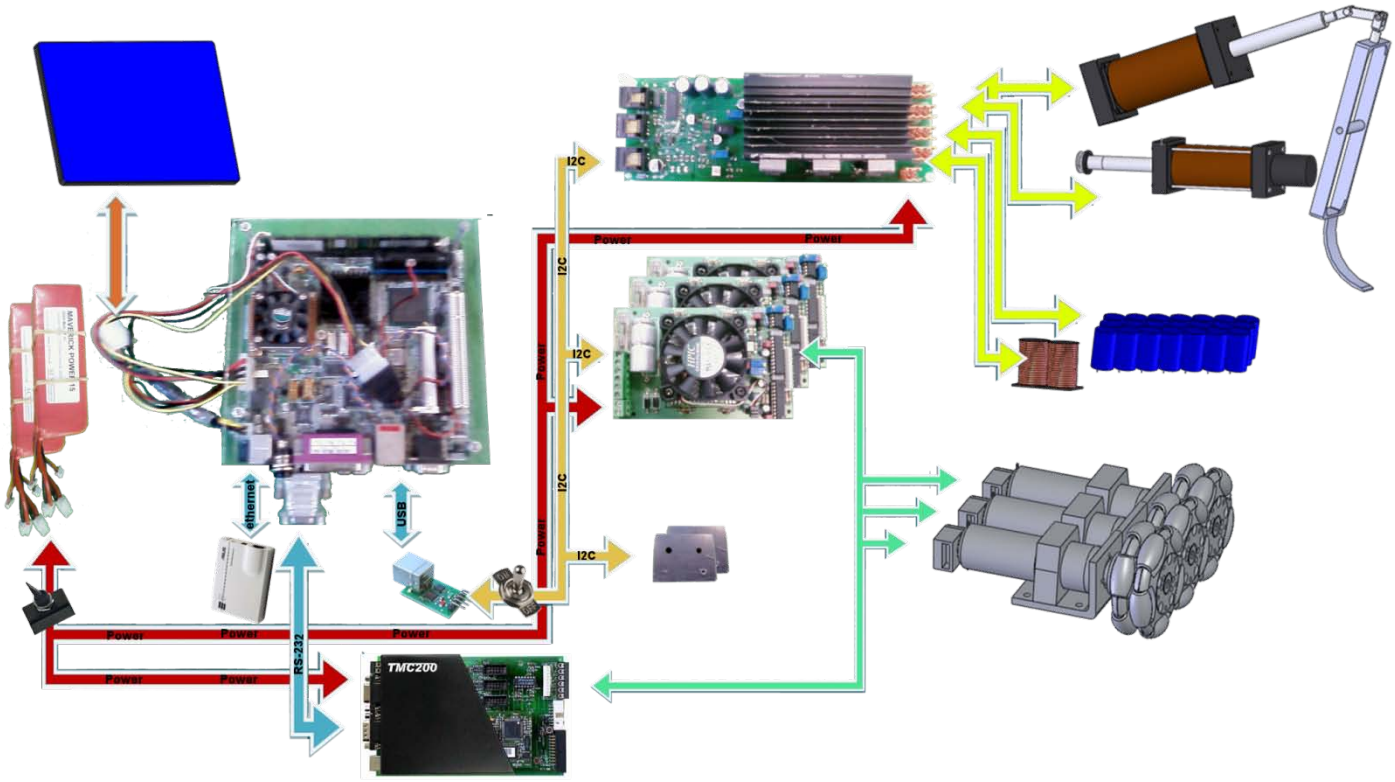


Figura 3.2:4 - Ligações actuais do robô

3.3 Desenvolvimento de *software*

O *software* desenvolvido é todo ele feito em C++ [43] que é uma linguagem orientada ao objecto considerada de médio nível, pois combina o baixo com o alto nível de programação. Esta linguagem foi escolhida para o desenvolvimento dado que ajusta muito bem o baixo nível com a possibilidade de alguma abstracção em alto nível que serve para desenvolvimentos futuros com a sua organização em classes.

Esta programação é feita numa *framework de Qt* sendo um ambiente de desenvolvimento multi-plataforma com ambiente gráfico. Esta *Framework* é *open source* e suporta diversas linguagens, entre elas o C++. Uma das suas inúmeras características é a sua imensa biblioteca que implementa diversas funções e com a existência de um manual muito completo e bem estruturado. Permite também a construção de software com suporte de ambiente gráfico

que ajuda muitas vezes em correcções de erros, assim como, facilita calibrações e é sempre mais interactivo.

Este desenvolvimento é implementado num sistema operativo Linux [44], sendo Debian [45] a distribuição usada, devido a ser um sistema base e ter uma total liberdade de configuração de todo o sistema.

O software desenvolvido para o Robô Futebolista está dividido em duas classes bastante distintas. A classe **CRobo** que implementa todo o código necessário para o robô jogar e outra classe **mainWnd** que contém todo o ambiente gráfico. Esta última quando é instanciada recebe o apontador da classe CRobo que foi anteriormente criada no inicio do programa. Assim é possível controlar os eventos a enviar para o Robô, assim como os resultados provenientes deste. Com esta divisão é também possível arrancar o software com ou sem o suporte ao ambiente gráfico aumentando o poder de processamento ao máximo para os jogos.

Seguidamente, com o desenvolvimento desta dissertação será possível identificar e correlacionar estas classes já apresentadas com outras classes importantes, às quais estas estão directamente ligadas.

Capítulo 4

Comunicação em Rede

Uma parte importante do robô futebolista passa pela comunicação. Esta comunicação pode ser estabelecida entre os próprios robôs, assim como entre o monitor ou treinador e os robôs, ou então entre a *referee box* e o treinador. Esta comunicação tem que obedecer a algumas regras impostas pela organização da competição [46] para garantir que todas as equipas tenham um limite máximo e igual. Esta comunicação tem que ser facilmente estabelecida e garantida a inexistência de falhas ou então que o sistema seja redundante

4.1 Topologia de ligação

As ligações existentes são as impostas e disponibilizadas pela organização como mostra a figura seguinte (Figura 4.1:1). Nestas ligações toda a comunicação *wireless* deve passar pelo AP (*Access Point*) da organização. Existem dois APs, um para a comunicação 802.11a e outra para 802.11b. A comunicação entre robôs é permitida desde que esta seja estabelecida através de um dos APs da organização, evitando assim o uso exagerado de redes *wireless* presentes no recinto e deste modo é também possível controlar a

largura de banda usada por cada equipa que não pode ultrapassar os 20% [46].

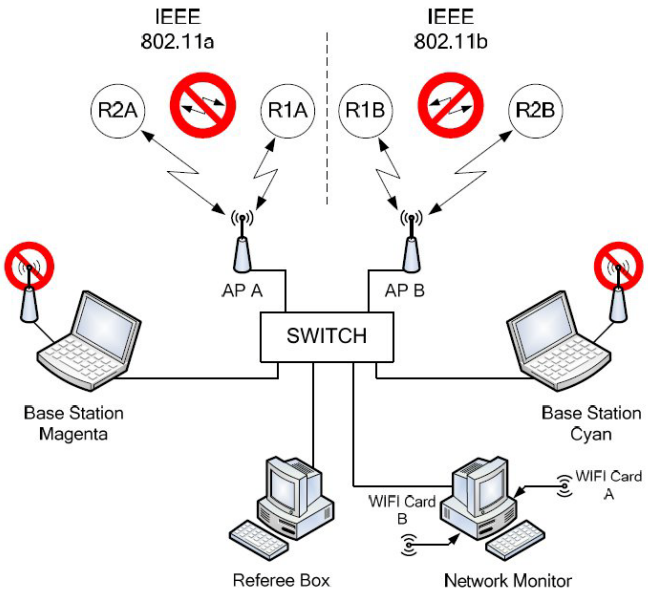


Figura 4.1:1 - Topologias de ligações da rede [46]

A comunicação entre a *Referee Box* e a estação base ou monitor, apenas pode ser estabelecida por cabo, garantindo assim menor erros e maior velocidade de transmissão nesta comunicação.

A todas as equipas é atribuída uma gama de IPs (Internet Protocol) pré-definida. Deste modo evita-se conflitos de IPs, assim como, facilmente se identifica uma respectiva máquina à sua respectiva equipa.

4.2 Protocolos

Cada equipa pode estabelecer comunicações em multicast ou unicast, sendo esta última implementada pela equipa Minho Team. Como foi dito anteriormente cada equipa possui a sua gama de IPs, sendo atribuídos IPs para *unicast* e *multicast*. Foram implementados os dois tipos de protocolos, TCP (Transmission Control Protocol) e UDP (User Datagram Protocol), mais a frente detalhadamente explicados. O protocolo TCP é usado para a comunicação com a *referee box,* pois garante a correcta entrega e recepção do pacote. Como nesta comunicação apenas existe transmissão de dados com uma nova ordem do árbitro, é importante garantir que o dado é recepcionado. O protocolo UDP é usado na transmissão de dados entre robôs ou entre robôs

e monitor. Esta transmissão apesar de ser igualmente importante, como esta é cíclica, o dado é novamente reenviado, poupando largura de banda e mais importante ainda, aumentando a velocidade de transmissão.

4.3 Software

O software desenvolvido para implementar a comunicação com o monitor, assim como com outros robôs, é feito em apenas duas simples classes reutilizáveis. Uma classe **CRede** e outra **CRedeBot**, como se pode ver no diagrama seguinte (Figura 4.3:1). A classe CRedeBot herda características da classe base CRede que por sua vez herda características da classe **QThread** [47] do Qt, passando assim a ser uma classe com thread.

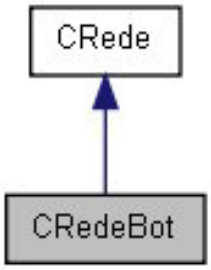


Figura 4.3:1 - Diagrama de classes da rede

A classe base implementa toda a parte de estabelecimento de conexão e desconexão para a comunicação por sockets UDP utilizando novamente as bibliotecas disponíveis pelo Qt. Dependendo do modo desejado e indicado no construtor, esta classe pode enviar e receber, ou apenas enviar ou mesmo apenas só receber dados. Esta classe base implementa um filtro de tramas recebidas, pois só as tramas com assinatura são aceites, criando alguma protecção e fiabilidade dos dados recebidos. É nesta classe que está implementada a thread que espera por eventos de *ready_read*, que indica a chegada de uma nova trama, que está pronta para ser lida, para executar o respectivo processamento dos dados recebidos. Este processamento é feito em funções virtuais para poder ser implementado em outras classes específicas, como a CRedeBot, criando assim, uma maior reutilização da classe. O envio de dados é feito por um temporizador inicialmente configurado

no construtor ou então está sempre disponível uma função pública que permite o envio instantâneo de dados em qualquer altura do fluxo do programa.

Os dados a enviar ou receber estão organizados numa estrutura, num único ficheiro separado (trama.h), possibilitando assim facilmente, o adicionamento ou remoção de parâmetros da trama, sem haver a necessidade de compilar ou ajustar estas classes já funcionais.

Esta abordagem de estrutura destas classes, permitiu a implementação desta comunicação do lado do receptor, entenda-se monitor, muito facilmente, pois bastou simplesmente copiar a classe base e criar uma classe CRedeMonitor igual á CRedeBot, mudando apenas as duas funções virtuais para os respectivos processamentos desejados.

Num capítulo seguinte, no capítulo 6.2.3 sobre o software desenvolvido, vai-se explicar detalhadamente o modo como estas classes de comunicação se interligam e o funcionamento da respectiva thread

Capítulo 5

Processamento de Imagem

A parte mais importante de um ser robô futebolista é o processamento de imagem. Todo o tratamento de processamento sensorial aplicado desde a captura passando pela detecção até à identificação de formas, obstáculos, cores ou pontos específicos. Todo este processo de análise é demorado visto que as imagens adquiridas são grandes e para salientar algum resultado mais importante, é por vezes necessária a aplicação de vários algoritmos sequenciais sobre a mesma imagem.

No seguimento deste capítulo irá ser abordado desde técnicas de captura utilizadas, a diversos métodos e técnicas de processamento desenvolvidos com o intuito de diminuir o tempo de processamento de imagem, aumentando a velocidade de trabalho sobre uma imagem mais recente, uma imagem em tempo real. Este ponto final é importante, pois um robô é um sistema de tempo real, este tem que reagir aos estímulos externos o mais rapidamente possível, diminuindo assim o tempo de resposta, tornando o sistema mais reactivo, para evitar o uso de algoritmos que tenham em conta o atraso, ou algoritmos previsíveis.

5.1 Aquisição de imagens

A captura de imagens é feita com a maior frequência conseguida pela câmara e é implementada numa *thread* separada, a qual vai ser explicada mais em baixo, detalhadamente no capítulo 6.3.2. A dimensão da captura apesar de ser permitido 640 por 480, esta é capturada em 480 por 480 como se pode verificar na figura seguinte (Figura 5.1:1). Deste modo, aumenta-se a velocidade de captura e transmissão das imagens, assim como se diminui o tamanho da imagem a processar, aumentando a velocidade de processamento.

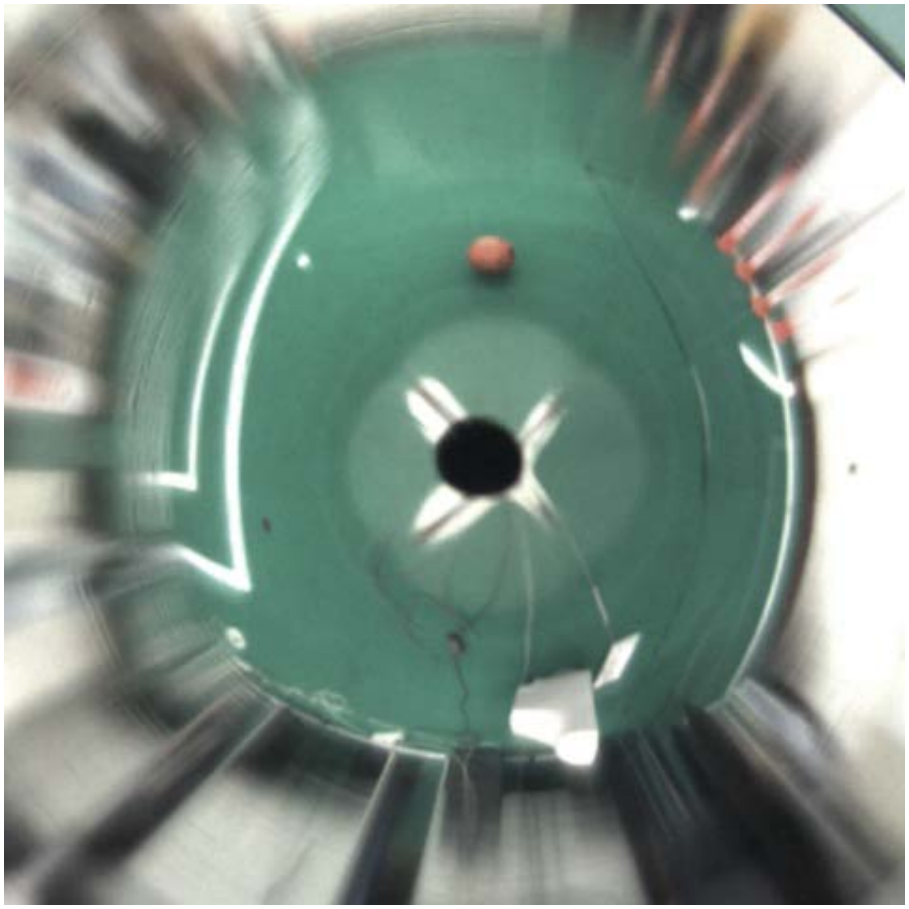


Figura 5.1:1 - Exemplo de uma imagem capturada

Para além da configuração de captura de imagens tem-se todo o controlo sobre as funções da câmara, pois a implementação efectuada que controla desde a captura aos ajustes da câmara, é feita com bibliotecas disponíveis para câmaras FireWire, a segunda versão da biblioteca libdc1394 [48]. A figura seguinte (Figura 5.1:2) mostra o menu de controlo sobre os parâmetros da câmara.

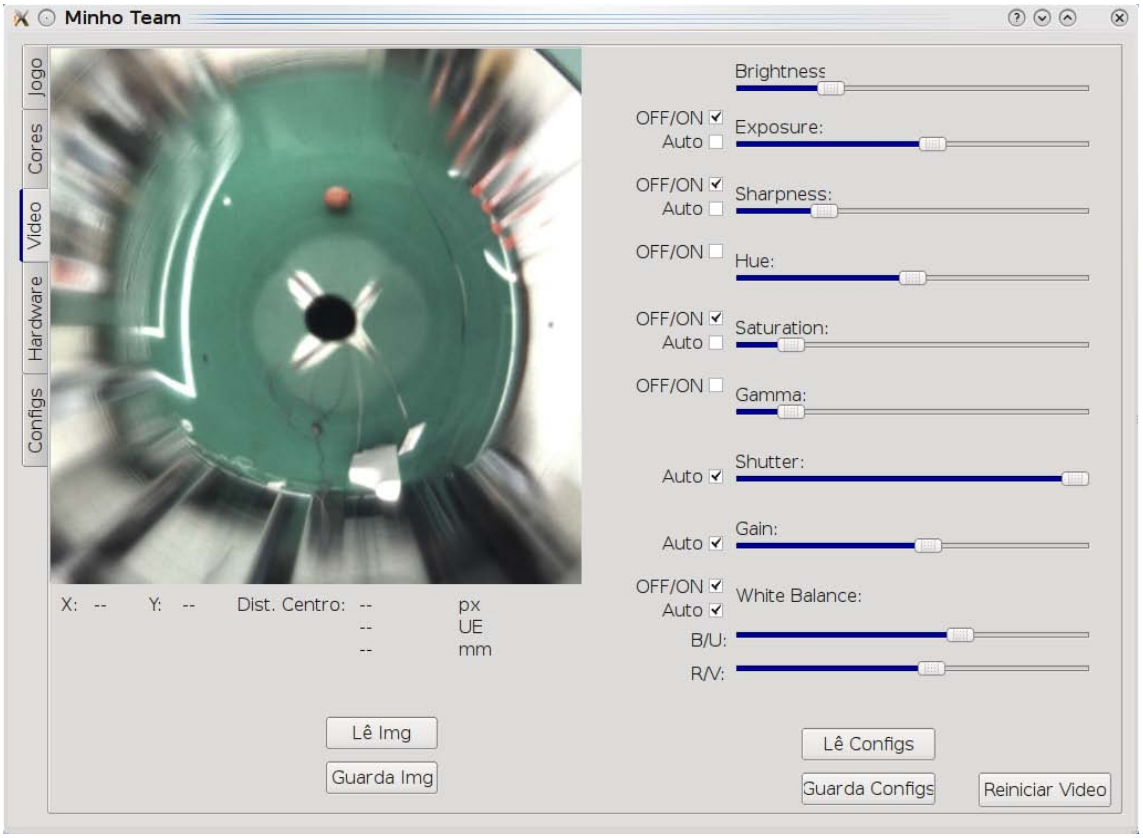


Figura 5.1:2 - Controlo dos parâmetros da câmara

5.2 Segmentação de imagens

A segmentação é o primeiro tratamento a aplicar à imagem capturada. Este processo faz a separação de cores predefinidas, como por exemplo, todos os verdes ou derivados de verdes, verdes mais escuros ou mais claros, assumem-se como o verde pré definido, ou verde puro. Deste modo, consegue-se limitar os inúmeros tons de cor para apenas algumas cores já conhecidas ficando as imagens, com o aspecto da figura seguinte (Figura 5.2:1).

Este processo é feito da seguinte maneira. O objectivo principal foca-se numa tabela com todas as cores e tons existentes, na qual, a cada uma das cores se vai guardar uma letra correspondente à respectiva cor calibrada. Deste modo para cada pixel existe 5 bits de vermelho, 6 bits de verde e 5 de azul. Ou seja, cada pixel pode variar a sua cor e intensidade nestas três componentes entre valores compreendidos entre 0 a 31 para componentes de

5 bits e de 0 a 63 para componentes de 6 bits, para se puder obter a cor desejada

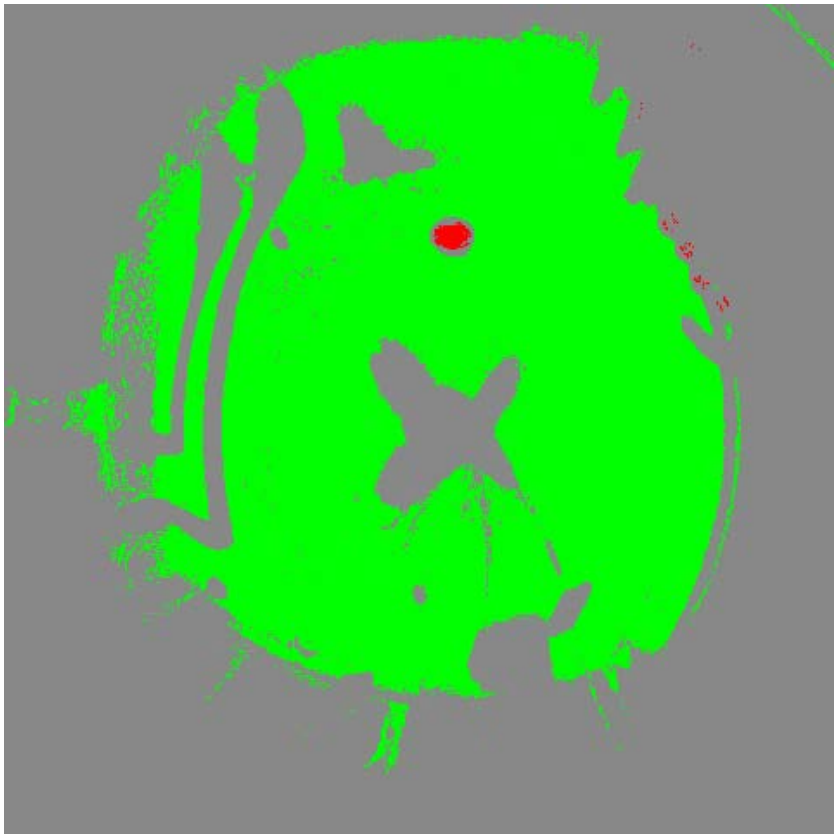


Figura 5.2:1 - Exemplo de uma imagem segmentada

Para a implementação deste algoritmo, criou-se então uma **tabela de caracteres** LUT (*Look Up Table*) com o número máximo de cores existentes 32 x 64 x 32 em que os endereços desta tabela correspondem à junção ordenada dos 16 bits que compõem a respectiva cor e cada caracter vai corresponder a uma cor pura calibrada como mostra a tabela seguinte (Tabela 2)

Tabela 2 - Tabela exemplo da organização da LUT usada

	Endereço (R|G|B)	Cor/Caracter
R=0, G=0, B=0	0	-
R=0, G=0, B=1	1	-
	…	…
R=0, G=55, B=1	1761	G
R=0, G=55, B=2	1762	G
R=0, G=55, B=3	1763	G
	…	…

Deste modo sempre que seja necessário saber se uma cor de um pixel já foi calibrado, ou então, saber qual a sua calibração, basta consultar na tabela o endereço dado pela junção das três componentes da cor RGB (Red Green Blue). Esta LUT inicialmente está toda preenchida por um carácter de cor considerada nula, e que com as calibrações se vai preenchendo as cores desejadas pela calibração para a respectiva cor/caracter puro, subscrevendo o caracter nulo.

Após a calibração este método de segmentação é bastante rápido uma vez que a consulta de uma tabela são instruções muito rápidas, libertando assim, o processamento para partes mais importantes. Para aumentar o desempenho da segmentação, usa-se os 16 bits para definir uma cor. Apesar da câmara apenas disponibilizar os 24 bits, é mais eficaz e mais rápido o uso de 16 bits, mesmo com as instruções para concatenar dos 24 bits recebidos para os 16 bits usados. Esta opção também facilita na calibração, pois apenas os tons mais significativos de cada cor são relevantes, desprezando os bits menos significativos que parecem ter definição a mais para o objectivo proposto.

5.3 Calibração de Cores

A Calibração das cores é uma parte muito importante das configurações antes do inicio das competições e jogos. Por este motivo, a necessidade de um método prático e eficiente leva à implementação de uma aplicação simples para a configuração das cores.

O método implementado de calibração de cada cor é feito à base da pesquisa das cores existentes dentro de um polígono, como mostra a figura seguinte. (Figura 5.3:1). A calibração é feita pela marcação de pontos que definem um polígono numa área específica na qual vão ser capturados todos os pixéis. Dentro dessa mesma área, todas as cores existentes dentro deste polígono vão ser consideradas cores puras para uma cor seleccionada.

Seguidamente apresenta-se um exemplo da calibração das cores verde e branco evidenciando a marcação do polígono para calibração da cor verde.

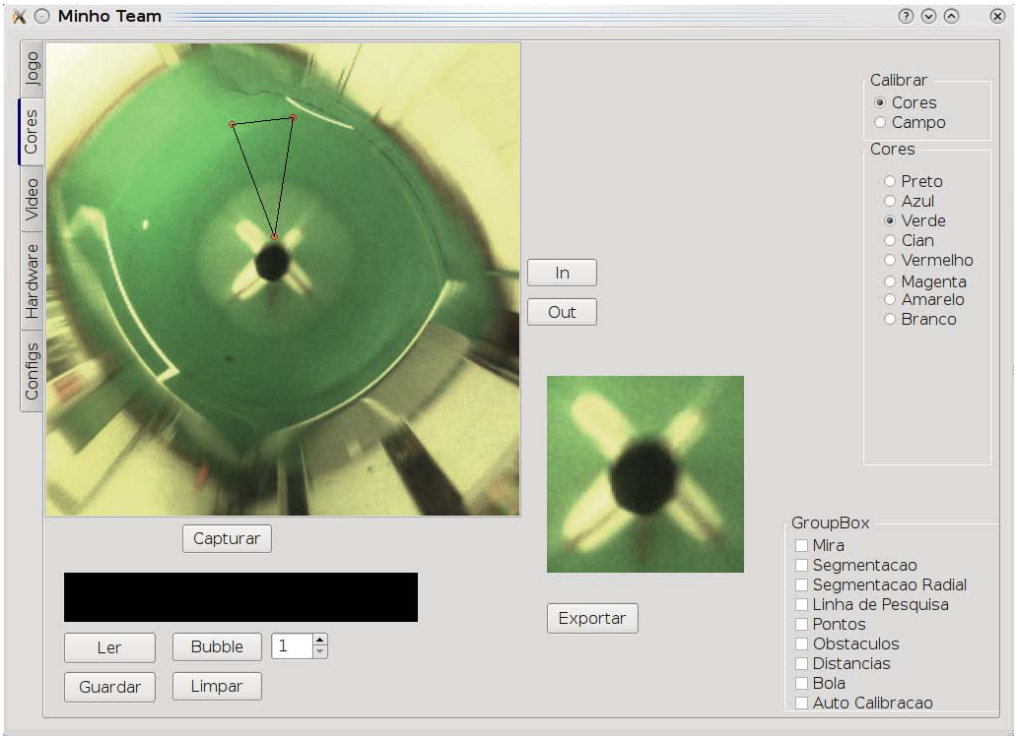
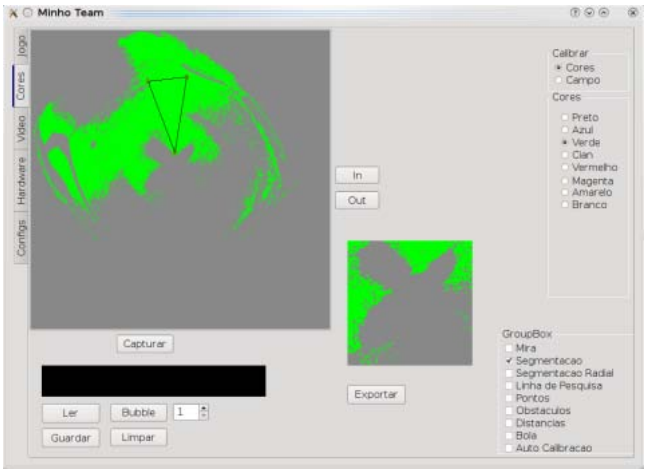


Figura 5.3:1 - Calibração de verdes com polígono

O resultado desta calibração para as cores verdes e brancas com uma única pesquisa dentro do polígono está visível nas duas seguintes imagens, para as cores verdes (Figura 5.3:2) e cores brancas (Figura 5.3:3).



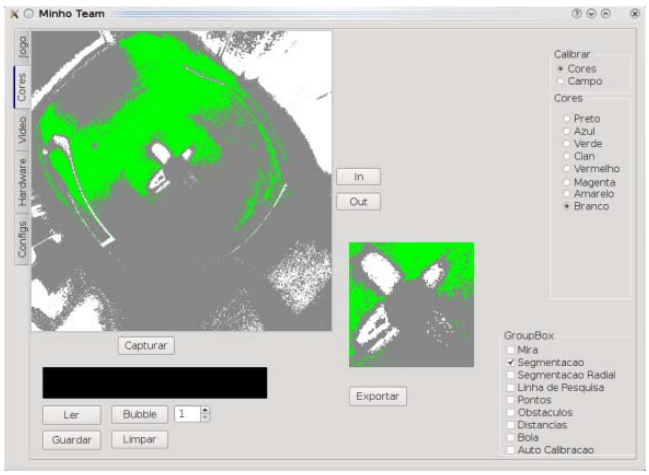


Figura 5.3:2 - (Esquerda) Calibração de verdes
Figura 5.3:3 - (Direita) Calibração de brancos

Após esta calibração existe uma outra aplicação para preenchimento de espaços vazios do cubo RGB calibrado e aumentar a gama de cores dos pixéis que definem uma cor. Este método chamado "*Bubble*" consiste em definir as

cores do cubo RGB mais próximas da cor seleccionada e já calibrada como também pertencentes ao cubo segundo o modelo seguinte (Figura 5.3:4).

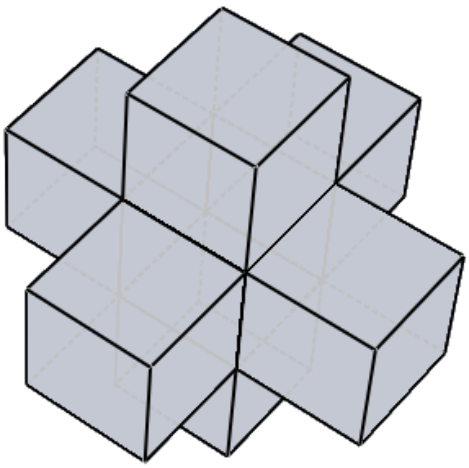


Figura 5.3:4 – Modelo do preenchimento de cores por "*Bubble*"

5.4 Pesquisa radial

A pesquisa radial é um método muito rápido de pesquisa em certas linhas específicas da imagem por pontos de futuro interesse. Deste modo consegue-se limitar a pesquisa para cerca de 16581 pontos, em detrimento dos 230400 pontos totais possíveis. Esta redução de pesquisa para apenas cerca de 7% da imagem, leva a um enorme aumento do poder de processamento.

Após esta pesquisa radial surgem pontos de interesse que futuramente são pesquisados com maior eficácia numa pesquisa localizada que mais a frente irá ser abordada.

Para a execução desta pesquisa radial é construído inicialmente um vector com todas coordenadas interessantes dos pontos da imagem a analisar, como mostra a figura seguinte (Figura 5.4:1).

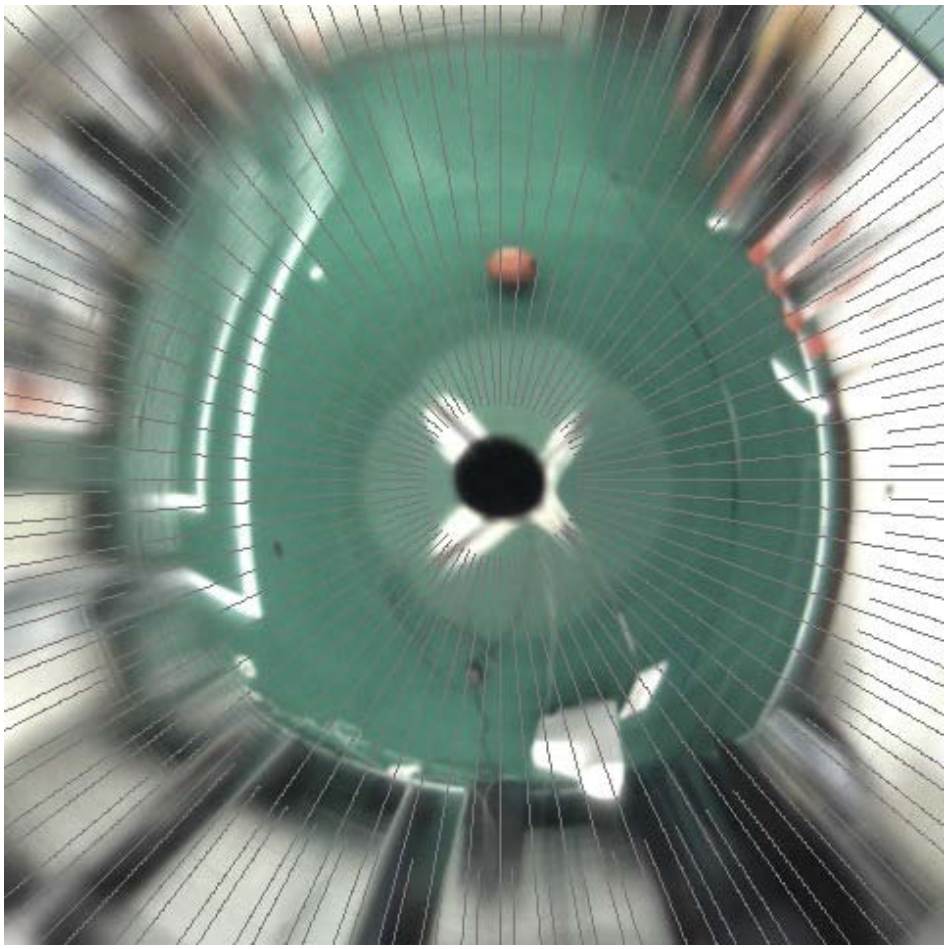


Figura 5.4:1 - Exemplo das linhas de pesquisa a analisar

Em paralelo, em outro vector, são guardados os apontadores para a nova imagem destino que se encontra num formato “campo” Este formato de imagem é três vezes mais pequeno que a imagem original, pois é para este formato que o resultado da segmentação é guardado. Este formato consiste em separar os pixéis analisados em caracteres pertencentes à LUT em vez das suas três componentes de cor, RGB, como mostra a figura seguinte (Figura 5.4:2). Assim, para cada cor segmentada corresponde um só único caracter de 8 bits indicando a cor respectiva em detrimento dos 16 bits das três componentes de cor.

G	G	G	G	G	G	G	G	G	G
G	G	W	W	W	W	W	W	W	W
G	G	W	W	W	W	W	W	W	W
G	G	W	W	G	G	G	G	G	G
G	G	W	W	G	G	G	G	G	G
G	G	W	W	G	G	G	G	R	R
G	G	W	W	G	G	G	G	R	R

Figura 5.4:2 - Excerto de exemplo de uma imagem do tipo "campo"

Deste modo, a pesquisa radial começa com uma segmentação da imagem, como já em cima foi referido, mas apenas às coordenadas especificas guardadas no vector a analisar, colocando o resultado neste novo vector de formato “campo” para se efectuar uma análise mais rápida. Seguidamente apresenta-se um exemplo de uma imagem com segmentação radial com uma pesquisa radial, apenas aos pontos específicos pré definidos (Figura 5.4:3).

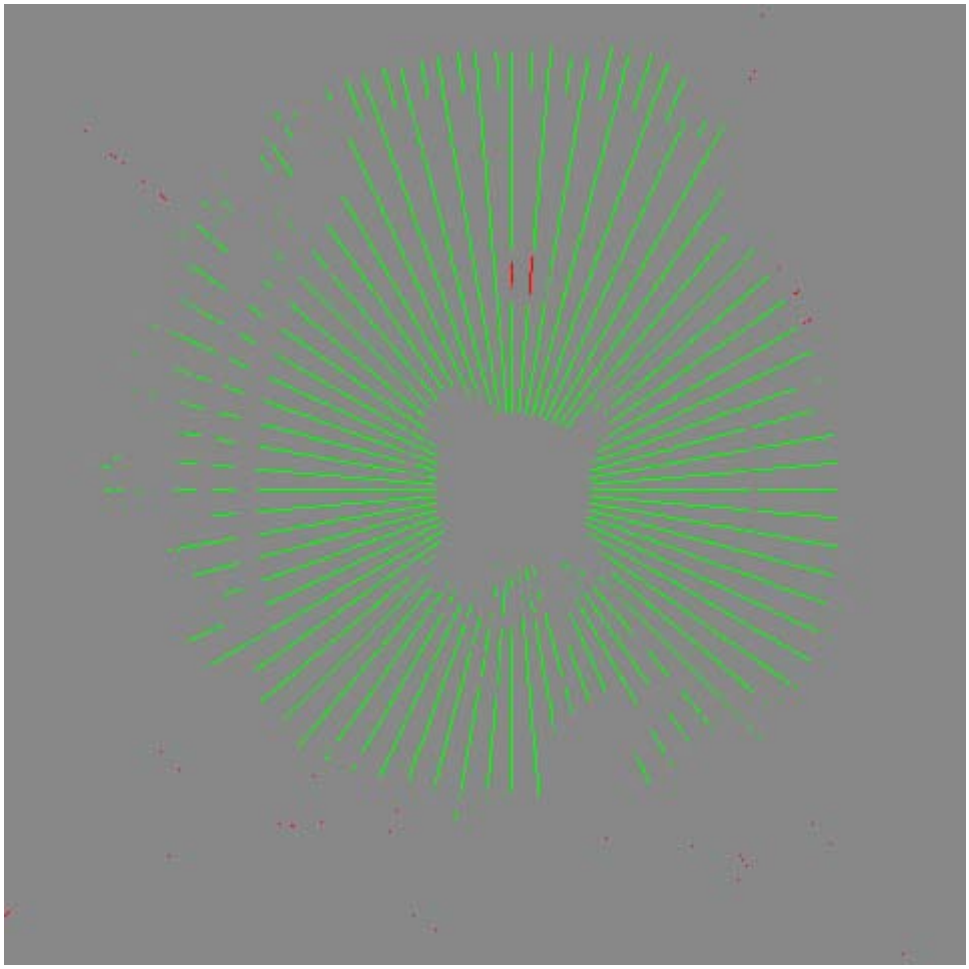


Figura 5.4:3 - Exemplo de uma segmentação com pesquisa radial

Após esta segmentação, é feito então o processamento da imagem para a procura de pontos de interesse. Este processamento é sequencial e apenas sobre as linhas analisadas. Como os endereços de todos os pontos destas linhas estão contidos num vector, basta fazer uma pesquisa sequencial sobre os valores apontados pelo vector que contem as linhas, para encontrar diferenças de verde para branco, como de verde para vermelho, para localizar pontos de interesse de possíveis linhas ou possíveis bolas respectivamente. Os pontos de interesse detectados referentes às bolas, ou seja, as transições de

verde para vermelho, são guardadas em um outro vector auxiliar que contém todos os pontos de interesse encontrados e deste modo agrupa-os consoante a proximidade entre eles. Assim, é possível identificar algumas zonas de interesse para uma procura mais aprofundada, localizada e limitada.

Nesta pesquisa radial também é importante a detecção e marcação de pontos brancos das linhas do campo, como mostra a figura seguinte, Figura 5.4:4

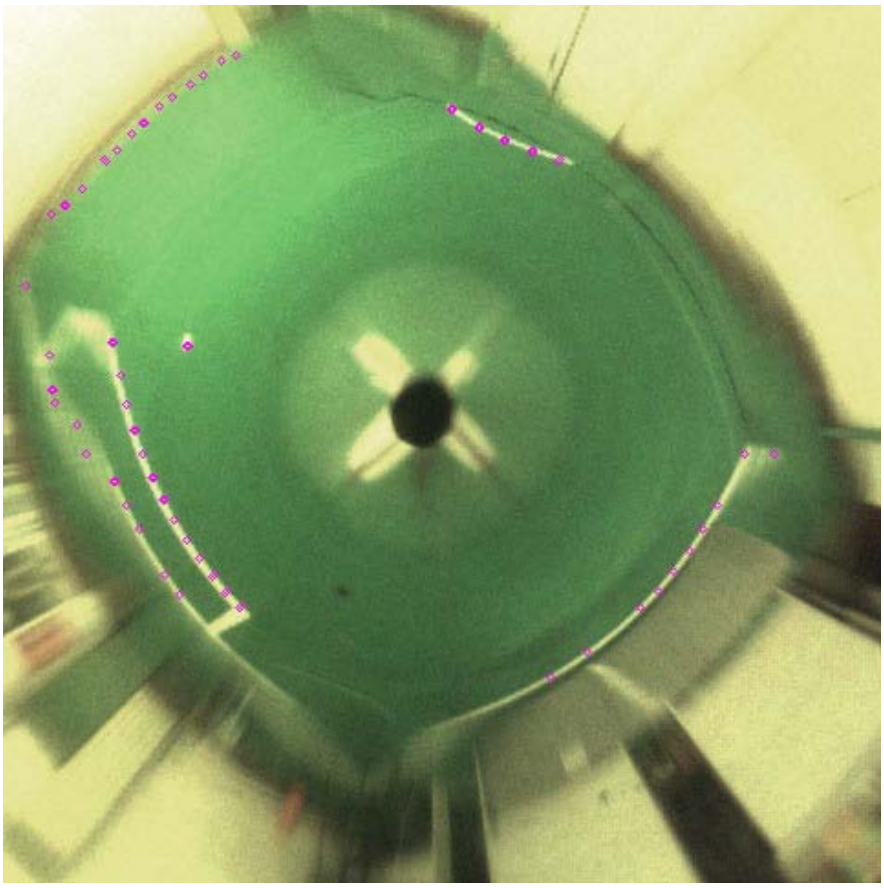


Figura 5.4:4 - Exemplo de detecção de pontos das linhas

Logo é possível evitar-se a pesquisa total da imagem. Com estes pontos brancos das linhas detectados vai ser possível fazer-se a correspondência com os pontos de um campo virtual, mais a frente documentado no capítulo 6.5.

5.5 Pesquisa localizada

Esta pesquisa localizada apenas tem significado após a existência de uma pesquisa radial, pois esta pesquisa localizada baseia-se nos resultados obtidos na pesquisa anterior, a radial, para efectuar uma pesquisa mais

intensa, localizada e limitada em certas áreas. Assim consegue-se um aumento de eficiência no software, aproveitando ao máximo o poder de processamento disponível pela máquina.

Deste modo é então possível nessas determinadas zonas previamente assinaladas como pontos de interesse, efectuar uma procura intensiva por cores, calculando máximos de cores e disposições das cores e até o centro de massa dos objectos. É possível também anular qualquer ponto de interesse mal detectado e que esteja fora do contexto. Seguidamente apresenta-se uma imagem (Figura 5.5:1) onde se pode detectar a marcação correcta de uma bola e com o seu respectivo centro alinhado. Na imagem seguinte vê-se a marcação de duas zonas de pesquisa delimitadas por um quadrado. Uma pesquisa mais localizada nessa zona, detectou e marcou como a possível bola, a bola correcta. Isto é visível pois a bola é delimitada por duas circunferências de raio e centro sobre a bola.

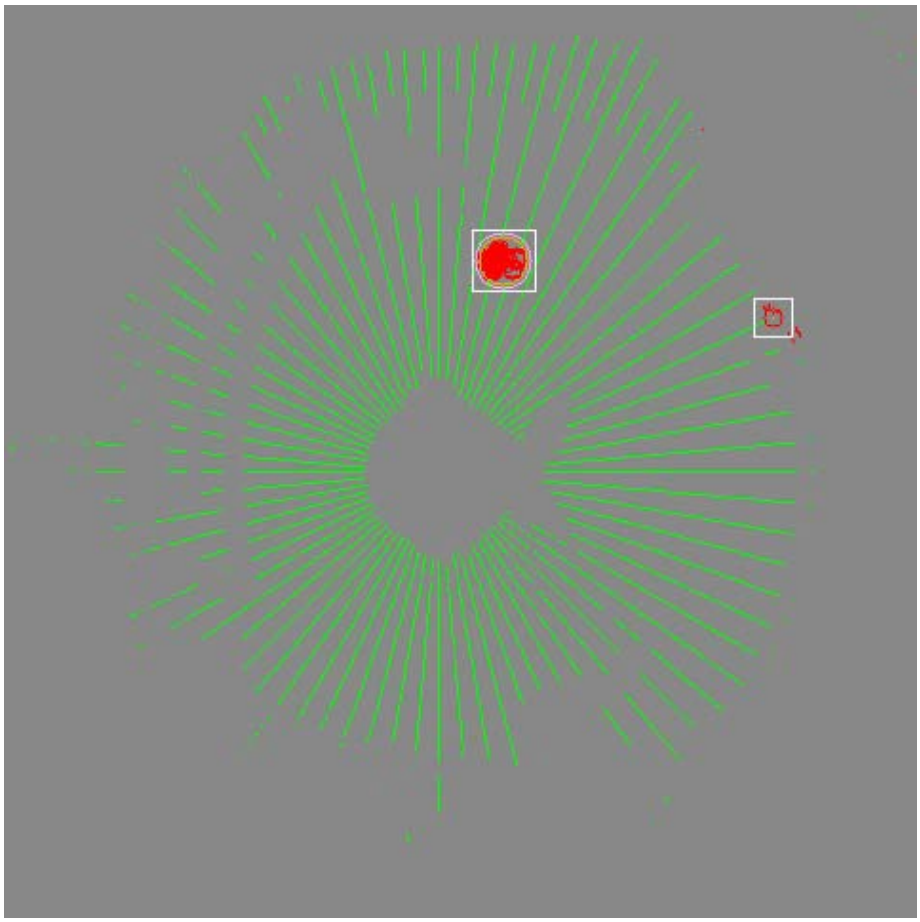


Figura 5.5:1 - Pesquisa localizada de pontos de interesse e marcação correcta de uma bola

Estes dados calculados são posteriormente inseridos numa base de dados que mais a frente será abordada.

5.6 Software desenvolvido

O desenvolvimento e organização do software da parte de processamento de imagem é das partes mais complexas, pois é principalmente à volta de imagens capturadas que todo o robô se movimenta. Deste modo o software tem que ser simples, versátil e facilmente adaptável, quer para alterações de algoritmos quer para novos testes a serem implementados.

Existem bibliotecas livres como o *openCV (Open Source Computer Vision)* [49] da Intel® que são dedicadas ao processamento de imagem. Estas bibliotecas são muito úteis para a análise rápida dos resultados de um algoritmo, mas muito ineficazes, pois necessitam de um encadeamento sequencial de funções muito próprias que introduzem muito atraso para uma plataforma móvel que se quer que reaja a tempo real. Na abordagem apresentada segue o seguinte diagrama de classes, abaixo usado (Figura 5.6:1)

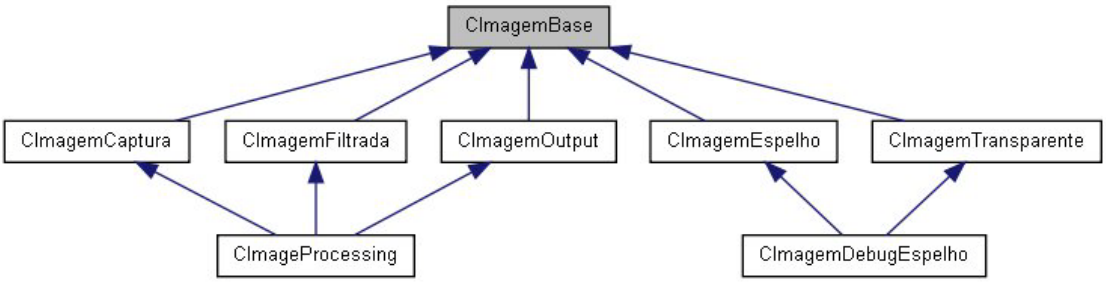


Figura 5.6:1 - Diagrama de classes possíveis para imagens

Deste modo é possível criar e separar pequenas parcelas distintas do processamento de imagem, como o instante que guarda a imagem capturada (CImagemCaptura), o local que guarda a imagem segmentada (CImagemFiltrada) e uma classe própria para retornar os resultados visíveis ao utilizador (CImagemOutput), quer por forma de vídeo, quer por forma de imagem guardada em disco. Com organização neste tipo de estrutura, é possível então criar classes próprias que usem apenas as classes necessárias,

pois existem situações em que não é necessária uma imagem capturada para se fazer o processamento de uma imagem guardada.

A classe base **CImagemBase** contém toda a informação básica de uma imagem, como as dimensões. Todas as seguintes classes herdam desta as principais características para alocarem em disco espaço suficiente e do tipo pretendido para a classe em questão. Por exemplo, a classe **CImagemCaptura** tem um apontador para o início da imagem capturada guardada através da biblioteca libdc1394, que como foi dito anteriormente é a responsável pela captura e ajuste dos parâmetros da câmara. A **CImagemFiltrada** é classe responsável por alocar um espaço de memória dedicado a segmentação para o formato do tipo “campo”. Isto é possível pois todos os atributos estão visíveis às duas classes. Existem também uma classe **CImagemOutput** que é responsável pela amostragem do resultado quer em modo vídeo ou guardando-o no disco.

Finalmente existe uma classe que herda todas estas características que é a **CImageProcessing** e que lhe permite ter acesso à imagem capturada, segmentar esta radialmente ou totalmente para o espaço de memória alocado na classe CImagemFiltrada e onde é efectuado qualquer tipo de algoritmo desejado. Também é possível a esta classe apresentar os resultados de diversas formas pois contém a classe CImagemOutut que é criada para esse propósito.

Seguidamente num capítulo próximo, capítulo 6.3.2, será abordada toda esta estrutura do software existente relativamente ao processamento de imagem. Desde as classes presentes até a funções específicas de cada uma delas, mas principalmente o fluxo do algoritmo usado no processamento de imagem.

Capítulo 6

Software Desenvolvido

Este capítulo é inteiramente dedicado a forma como o software foi organizado e desenvolvido. Seguidamente começar-se-á por descrever a estrutura e organização do software e construir a ponte desta organização com os algoritmos implementados. Esta estrutura é pensada para um progresso controlado e orientado para uma equipa de robôs futebolistas, uma vez que todos os módulos encaixam apenas de uma única maneira como mostra a figura seguinte (Figura 5.6:1), limitando assim os problemas e aumentando a capacidade de resposta de uma equipa bem coordenada com tarefas distintas.

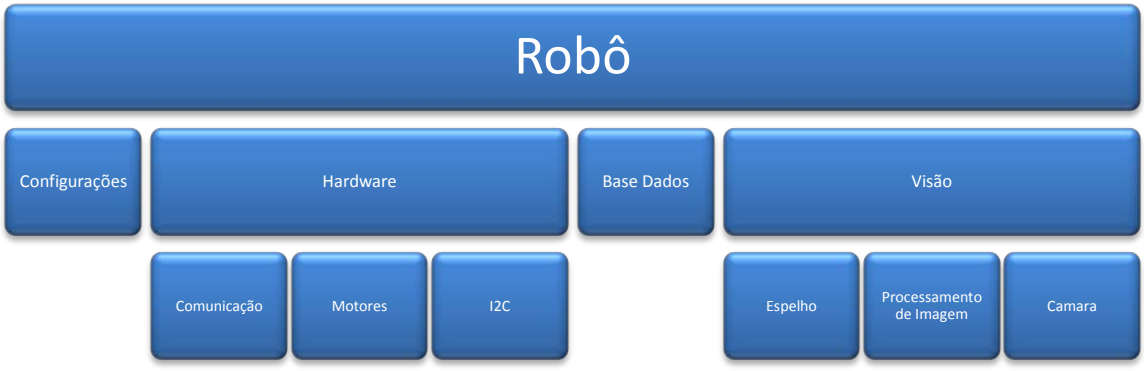


Figura 5.6:1 - Esquema da organização do software desenvolvido

Seguidamente apenas irá ser abordada a classe **CRobo** a classe base e que não necessita do ambiente gráfico para jogar. Esta classe base **CRobo** contém 4 novas classes principais e independentes criadas no construtor desta. Estas novas classes a reter são **CConfigs**, que é responsável pelo armazenamento de configurações. **CHardware**, responsável pelo controlo de todo o hardware presente no robô. **CVideoThread** que controla toda a captura e processamento de imagens e **CBaseDados** que guarda os dados obtidos e/ou calculados.

A figura seguinte (Figura 5.6:2) mostra a interligação entre estas quatro importantes classes e a classe CRobo.

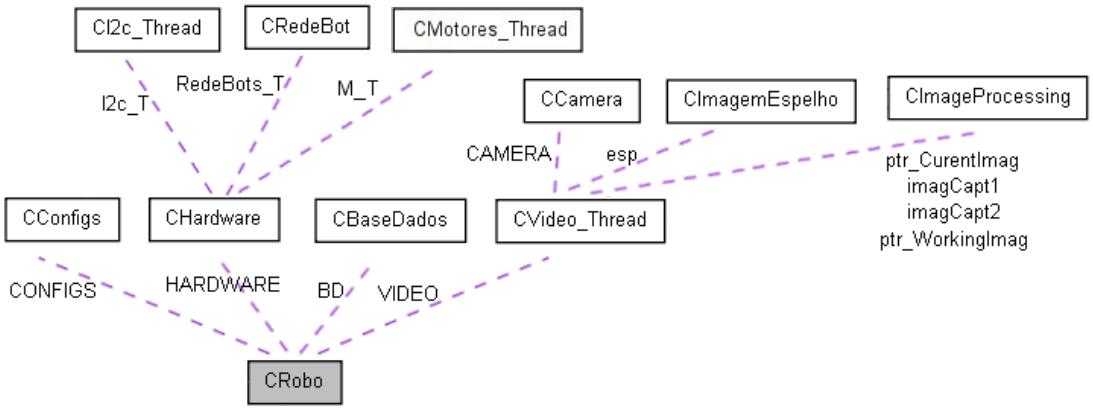


Figura 5.6:2 - Diagrama de classes da classe CRobo

6.1 Classe CConfigs

A classe CConfigs é uma classe dedicada a armazenar em variáveis privadas todos os parâmetros configuráveis, assim como implementar funções públicas para acesso aos dados, e leitura destes valores dos ficheiros de configuração. O excerto do diagrama apresentado a seguir, Figura 6.1:1, mostra a implementação efectuada desta classe base.

Os valores são guardados em disco, em ficheiros de texto que posteriormente são lidos por funções públicas e específicas desta classe. No início do construtor desta classe é efectuada a leitura de todos os parâmetros guardados, para variáveis privadas da classe.

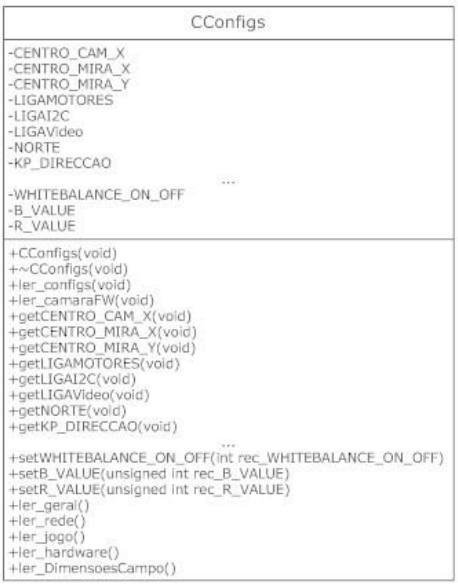


Figura 6.1:1 - Estrutura da classe CConfigs

Existe apenas uma instanciação desta classe pois o objectivo desta, é armazenar os parâmetros num único local específico, de modo a ter uma maior organização destes parâmetros, de maneira mais eficaz e localizada, garantindo assim, que estes estejam disponíveis ao longo de todo o software, não sendo alterados acidentalmente.

6.2 A classe CHardware

A classe CHardware contém suporte a todo o hardware presente como o barramento I2C, a placa dos motores (pela porta serie) e suporte rede. Todos estes dispositivos têm a sua própria classe dedicada, **I2C_Thread**, **CMotores_Thread** e **CRedeBot** respectivamente como mostra a figura seguinte (Figura 6.2:1).

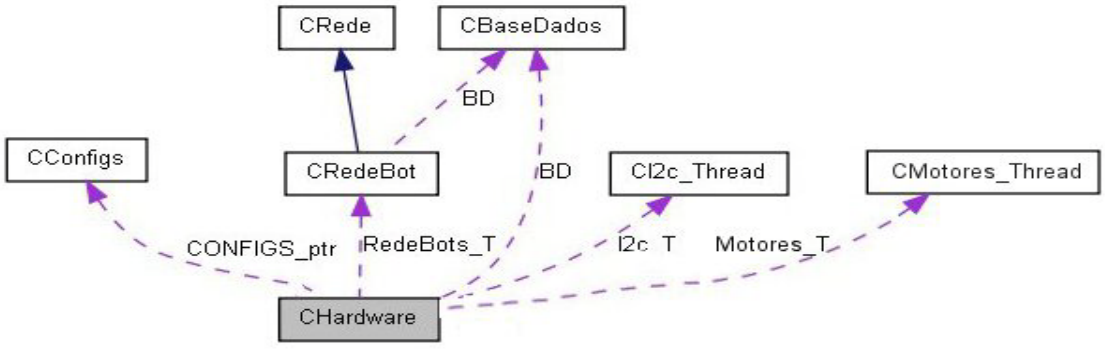


Figura 6.2:1 - Diagrama de classes da classe CHardware

O arranque de cada um destes dispositivos está dependente de *flags* de activação que estão guardadas em variáveis privadas da classe de configurações. Assim, é possível controlar o que se deseja arrancar facilitando algum *debug* especifico ou desabilitando algum dispositivo que não seja necessário. Como exemplo surge a possibilidade de desabilitar a placa dos motores (pela porta serie) quando esta é substituída por uma outra por I2C.

Com uma implementação deste modo desta classe, apenas se consegue ver do exterior as funções públicas de alto nível que permitem enviar comandos pré-definidos, como o chuto com a respectiva força, verificar o estado dos infra-vermelhos para controlar o hardware de baixo nível. Isto é facilmente perceptível analisando a estrutura da classe CHardware apresentada na figura seguinte, Figura 6.2:2.

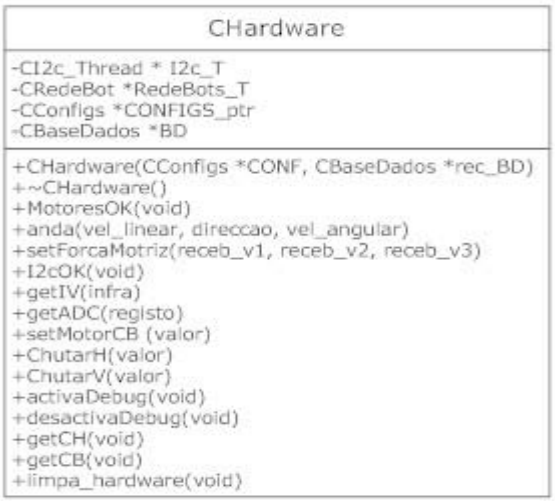


Figura 6.2:2 - Estrutura da classe CHardware

Assim, o baixo nível fica completamente isolado e estanque, evitando possíveis erros posteriores.

Nos próximos subcapítulos irão ser aprofundada as várias componentes da classe CHardware assim como, os algoritmos utilizados para o controlo do seu estado.

6.2.1. A classe CI2C_Thread

A classe CI2C_Thread é uma classe que herda características de uma QThread para poder ser implementada uma thread sobre a classe para controlar todo o barramento I2C não necessitando de estar constantemente em

processamento. Para isso é então utilizado mecanismos de agendamentos para dar o arranque da thread. Estes mecanismos são implementados com temporizadores que iniciam a thread e esta limita-se a verificar as *flags* activas e processar as respectivas funções. Terminada a execução de todas as funções das respectivas *flags* activadas, a thread termina deixando o processador livre.

Existem também mecanismos para o arranque da thread em situações espontâneas, como para chutar. Estas funções apenas activam a *flag* de chuto e arrancam a thread directamente no mesmo instante.

A classe que implementa a thread possui variáveis para guardar todos os valores lidos e também valores a enviar aos dispositivos externos, como por exemplo, a intensidade do chuto. Deste modo tenta-se manter todo o acesso ao barramento o mais separado possível do processo principal. Assim, todos os valores lidos dos diferentes agendamentos ficam guardados em variáveis privadas e específicas que apenas podem ser lidas com as respectivas funções públicas, como mostra do diagrama desta classe na figura seguinte (Figura 6.2:3).

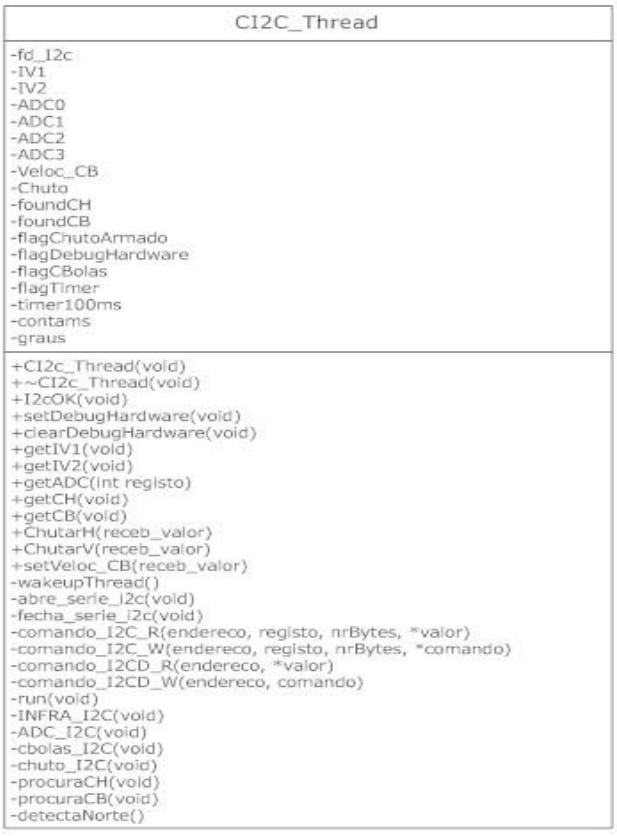


Figura 6.2:3 - Estrutura da classe CI2C_Thread

As variáveis de controlo de dispositivos como a intensidade do chuto e intensidade de velocidade do dispositivo responsável por recolher a bola, apenas podem ser alteradas em funções públicas para esse efeito, permitindo o acesso de uma forma controlada e impedindo que estas variáveis tomem valores não reconhecidos pelos dispositivos.

Nesta thread também está implementado software para controlar uma placa dos motores por I2C. Esta placa necessita de receber instruções novas de 400 em 400ms e devido a esta necessidade existe um temporizador extra para garantir que a placa dos motores nunca perde a comunicação com o software do robô futebolista. Para controlar a velocidade a enviar a cada motor existe uma função “anda” que implementa a cinemática de controlo dos motores. Estes valores calculados para cada motor são guardados em variáveis específicas e privadas da classe que estão constantemente a ser alteradas e a serem enviadas para a placa dos motores.

Seguidamente apresenta-se o fluxograma da thread que controla todo o barramento I2C na Figura 6.2:4

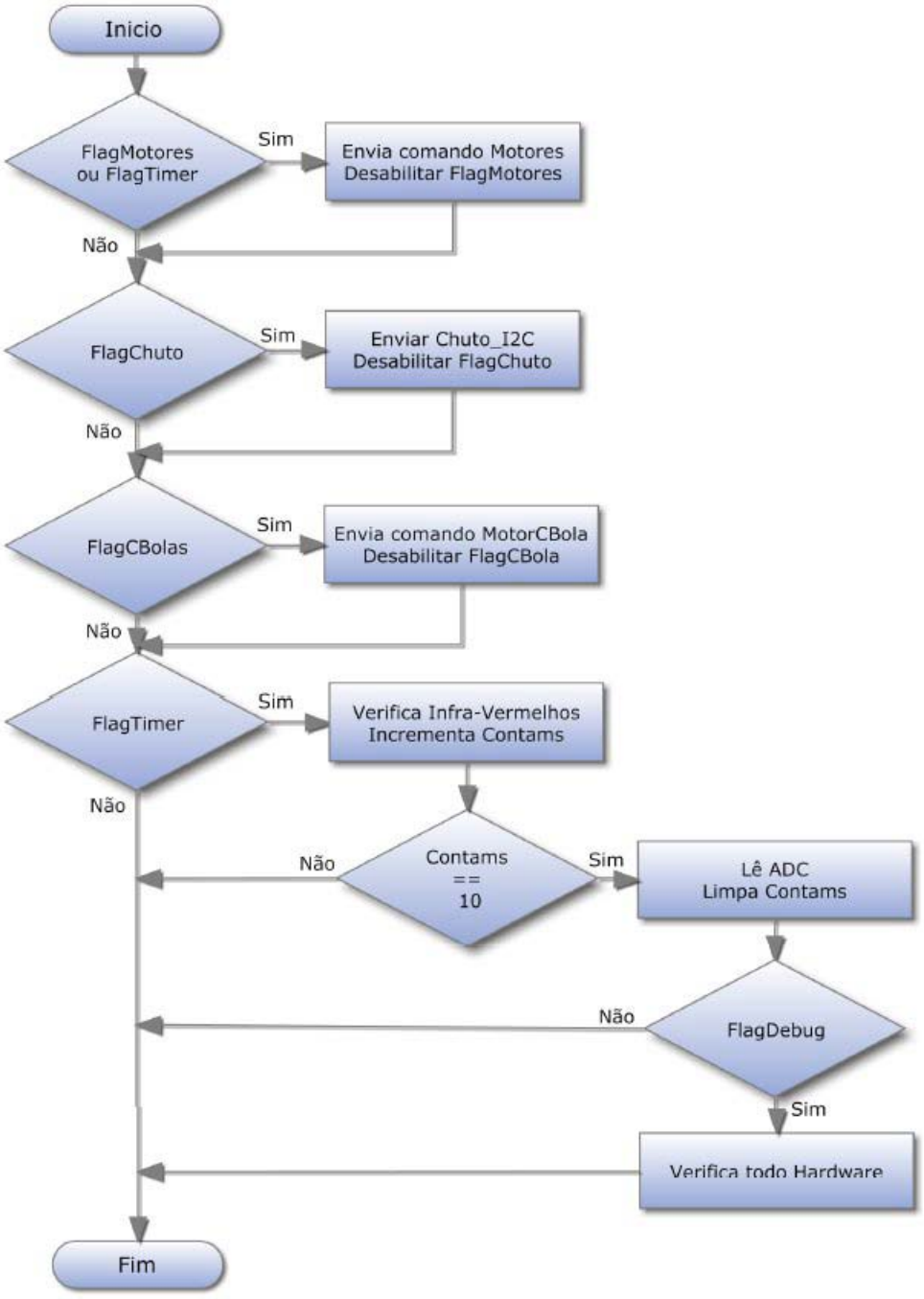


Figura 6.2:4 - Fluxograma de execução da thread de controlo do barramento I^2C

6.2.2. A classe CMotores_Thread

A classe CMotores_Thread também herda as características da classe QThread implementando uma thread simples que se baseia no envio do comando velocidade para a placa dos motores pela porta série. No construtor da thread é estabelecida a comunicação com a placa pela porta série, configurando todos os parâmetros da comunicação. Após bem sucedida esta

comunicação é chamada uma outra função privada desta classe responsável pelo envio das configurações iniciais para a placa, como valores de ganho do controlador PID interno da placa, resolução de encoder, valores de velocidades máximas e mínimas.

Esta thread contém variáveis que guardam os valores de velocidade a enviar a cada motor e que são alterados através de uma função chamada "anda" que implementa a cinemática de controlo dos motores. Estes valores apenas são enviados para os motores quando a thread é acordada, pois esta encontra-se normalmente num estado de *sleep* libertando o processador para cálculos mais importantes. Seguidamente apresenta-se a estrutura do diagrama de classes desta simples classe que implementa toda a comunicação pela porta série com a placa dos motores (Figura 6.2:5).

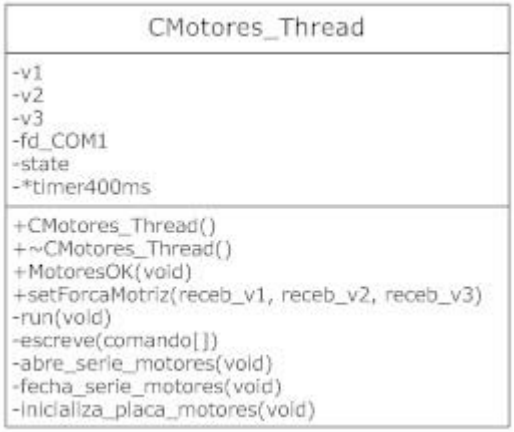


Figura 6.2:5 - Estrutura da classe CMotores_Thread

A thread é acordada no final da função pública "anda" que permite calcular novos valores a enviar à placa. No final desta função arranca-se o respectivo *start* da thread, para os novos valores poderem ser enviados novamente à placa. Existe também um temporizador externo configurável de 400 em 400ms que evita que a placa se desligue, enviando o último comando válido para a placa.

Na figura seguinte (Figura 6.2:6) apresenta-se o fluxograma do código implementado nesta simples thread.

Figura 6.2:6 - Fluxograma da thread dos motores para a placa TCM200

6.2.3. A classe CRedeBot

Esta classe já foi descrita anteriormente no capítulo 4.3 da comunicação em rede, mas no entanto apenas agora faz sentido explicar as funções e variáveis contidas nesta classe, assim como o algoritmo implementado. De salientar que esta classe que controla a comunicação, está inserida no módulo hardware herda características da classe CRede e as suas funções públicas são apenas aquelas necessárias ao envio de tramas novas. Seguidamente apresenta-se o diagrama de classe (Figura 6.2:7), onde se pode verificar o uso de funções virtuais para o processamento dos dados. As variáveis de controlo assim como a estrutura que recebe a trama são privadas à classe, impossibilitando a alteração acidental dos dados recebidos. A classe CRedeBot, tem acesso à base de dados que é onde esta vai inserir os novos dados recebidos.

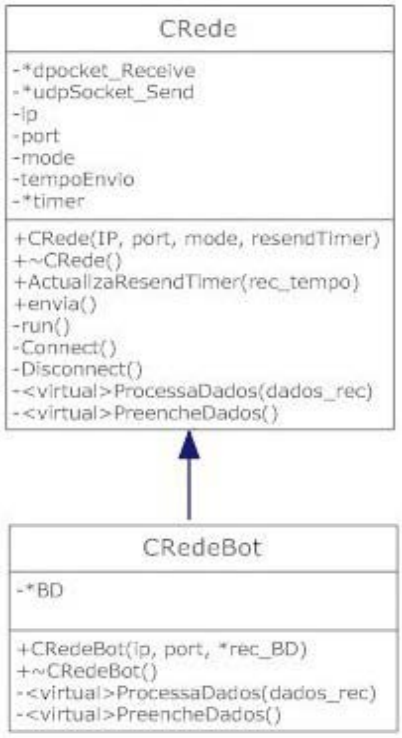


Figura 6.2:7 - Diagrama de classes das classes CRedeBot e CRede

O conceito da classe está anteriormente descrito, no entanto, o software desenvolvido nesta classe segue o seguinte fluxograma (Figura 6.2:8). Esta thread apenas é acordada quando existem novos dados pendentes. Após a leitura de todos os dados é confirmada a autenticidade da trama e os dados são processados e inseridos na base de dados

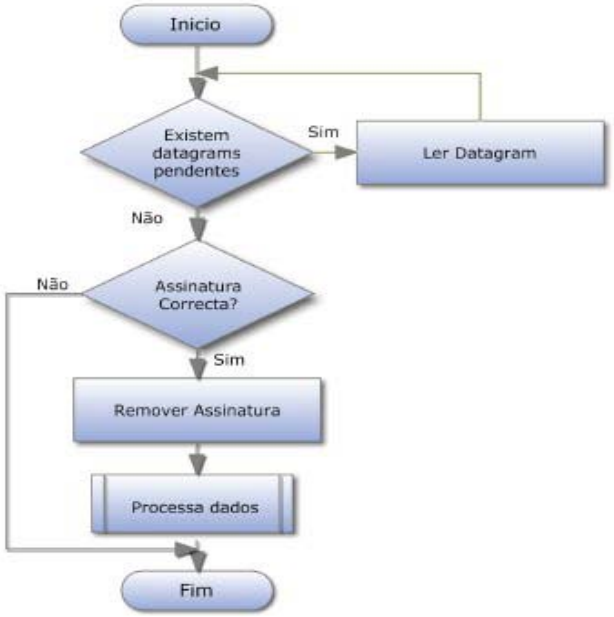


Figura 6.2:8 - Fluxograma da thread rede

Para a implementação deste algoritmo existem as duas classes já anteriormente mencionadas, CRede e CRedeBot. A classe CRede contem a *thread* responsável pela leitura e escrita de dados sem influenciar a normal execução do programa. Nesta classe estão implementadas duas funções virtuais para o processamento dos dados recebidos e enviados. Estas funções vão ser reimplementadas na classe CRedeBot tornando assim o uso destas classes mais versátil podendo estas clases serem reaproveitadas. Na classe CRede, é onde está implementada toda a parte de comunicação com recurso as livrarias do Qt como QUdpSocket [50] ou QTcpSocket [51], assim como a thread com o algoritmo anteriormente descrito.

6.3 A classe CVideoThread

A classe CVideoThread é uma classe que também herda as características da QThread, implementando uma thread que controla todo o vídeo, desde a captura de imagens através da câmara, até à análise e processamento das imagens. Esta thread está a funcionar em ciclo infinito até ao fecho do programa, pois a captura deve ser constante para os dados calculados serem os mais recentes. Apesar de esta thread se encontrar em ciclo infinito, esta tem pequenos tempos de espera, pois existe algum atraso entre o sinal de captura enviado e a imagem após a capturada ser totalmente recebida. Para todo este controlo, esta classe tem acesso à câmara, para controlar as capturas e as imagens a analisar, à imagem processada, pois vai ter o resultado do processamento das imagens capturadas e à base de dados, pois é onde esta classe/thread vai inserir os novos dados obtidos. Isto é conseguido visto que esta classe cria e tem acesso a outras novas classes independentes que são **CCamera**, **CImageProcessing** e **CBaseDados** respectivamente, como mostra a figura seguinte (Figura 6.3:1).

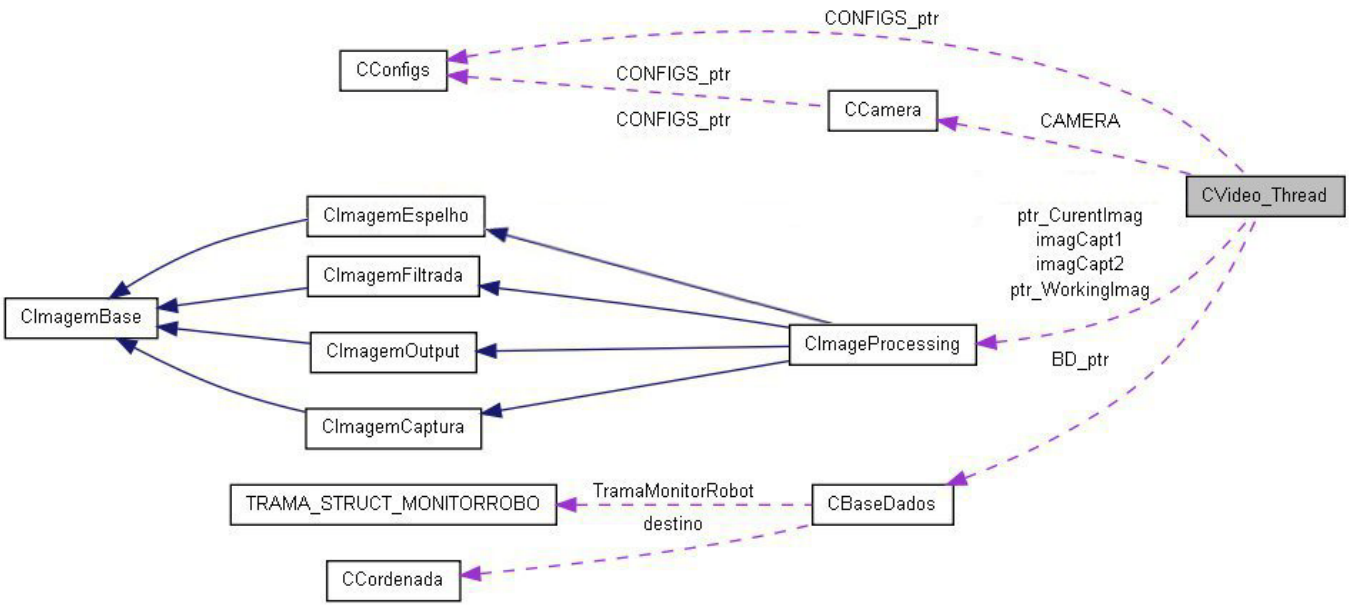


Figura 6.3:1 - Diagrama de classes da classe CVideo_Thread

A estrutura e organização desta classe e as respectivas variáveis é a apresentada na figura seguinte, Figura 6.3:2.

Figura 6.3:2 - Estrutura da classe CVideo_Thread

É nesta classe que é instanciada as classes do tipo CImageProcessing para o processamento de diferentes imagens (imagens pares, impares e imagens lidas de ficheiros), a tabela de filtros e variáveis de *flag* para controlo de estado da thread.

Esta classe tem várias variáveis de controlo do fluxo do programa, sendo possível assim separar entre o estado *debug* e o *run time*, ou seja, quando se pretende visualizar e analisar as imagens adquiridas e processadas, e outro estado, para correr os processos normalmente, ganhando poder de processamento. É possível também controlar a imagem a analisar, pois esta pode ser a imagem par, impar ou uma imagem guardada em ficheiro.

No estado de *debug* é possível guardar a imagem capturada com uma imagem nova, assim como seleccionar o tipo de processamento a efectuar a cada imagem. O tipo de processamento pode variar entre a segmentação total ou radial, procura de bolas e procura de pontos de linhas do campo.

Todo este algoritmo implementado é mais facilmente perceptível no fluxograma seguidamente apresentado (Figura 6.3:3).

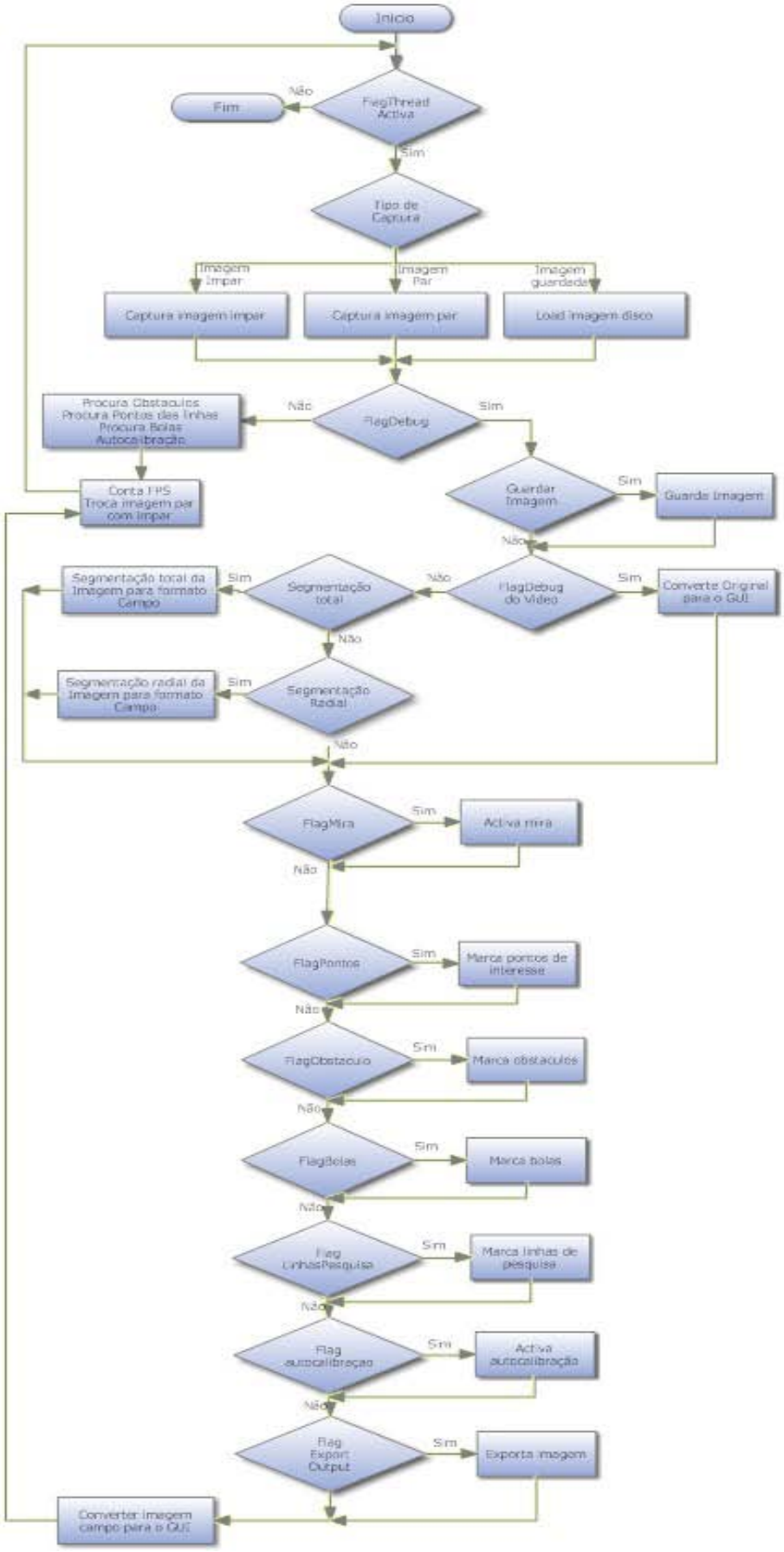


Figura 6.3:3 - Fluxograma da thread de vídeo

6.3.1. A classe CCamera

Esta classe é a classe responsável por todo o controlo da câmara. Desde o ajuste de parâmetros, como o *shutter speed* ou *white balance* até à região de interesse de captura, passando pelo arranque, inicializações da câmara, leitura e escrita das configurações. Existem funções públicas para controlo destes parâmetros que ficam guardados em variáveis privadas da classe. As funções públicas visíveis pelo exterior da classe são apenas as funções de ajuste de parâmetros, inicializações da câmara e uma função para efectuar uma captura. Isto é perceptível na estrutura da classe a seguir documentada na Figura 6.3:4.

Figura 6.3:4 - Estrutura da classe CCamera

Isto é possível porque o controlo da câmara é todo feito com a biblioteca específica para câmaras FireWire, que é a biblioteca libdc1394. Esta classe também é responsável por efectuar a captura de imagens para um espaço de memória alocado pela biblioteca. Após ser efectuada a captura é retornado um apontador para o inicio da imagem capturada que vai ser entendida por outra

classe seguidamente documentada, a classe CImageCapturada no capítulo 6.3.1. Deste modo é agora possível ser feito o processamento sobre a imagem capturada pela classe CImageProcessing que irá ser abordada no tópico seguinte, no capítulo 6.3.2. Após todo o processamento efectuado, é necessário libertar o espaço de memória alocado pela biblioteca da imagem capturada, uma vez que a biblioteca bloqueia o acesso de escrita de novas imagens para esta região, mantendo a integridade da imagem. Isto implica que no fim do processamento efectuado seja chamada uma função específica para este efeito.

6.3.2. A classe CImagemProcessing

Esta classe é a classe principal de todo o processamento de imagem. É esta classe responsável por todo o processamento de imagem feito à imagem capturada, com todos os algoritmos em cima já referidos. Após a thread de vídeo executar as funções públicas de análise de imagem desta classe, os resultados são inseridos numa base de dados que mais em baixo irá ser explicada ao pormenor. Nesta classe estão implementadas as funções de pesquisa radial, pesquisa localizada, segmentação, entre outras como se pode verificar na estrutura desta classe representada na figura seguinte (Figura 6.3:5).

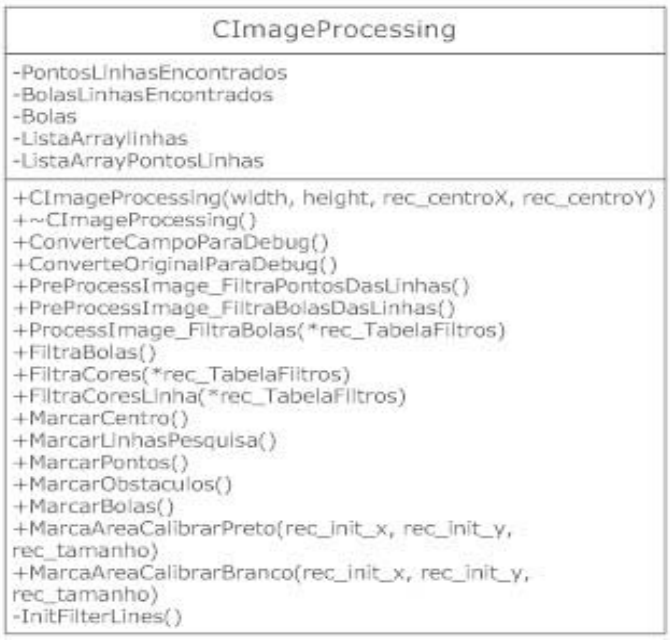


Figura 6.3:5 - Estrutura da classe CImageProcessing

É nesta classe que novas funções de processamento de imagem ou testes devem ser inseridas, pois esta classe herda características das classes, CImagemCaptura, CImagemFiltrada e CImagemOutput, como demonstra a Figura 6.3:6.

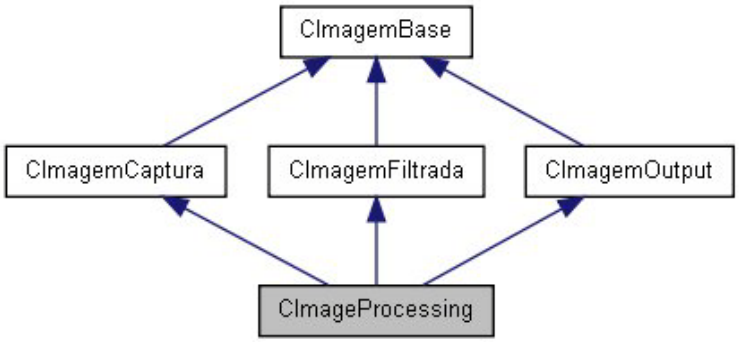


Figura 6.3:6 - Diagrama de classes da classe CImageProcessing

Deste modo mantém-se a estrutura do programa, evitando assim alocações desnecessárias de espaço de memória para imagens, mas principalmente para centralizar todo o processamento de imagem numa classe já com um nível de abstracção superior como demonstrado no diagrama anterior. Caso seja necessário adicionar um novo tipo de imagem do mesmo nível das três mencionadas anteriormente, este novo tipo de imagem encaixa facilmente na estrutura implementada e pode ser usada pela CImageProcessing para futuras aplicações.

6.3.3. A classe CImagemBase

Esta classe é uma classe muito básica e que serve de apoio a todas as outras classes de imagem que vão herdar características desta. Todas as outras classes de imagem necessitam desta classe, pois esta classe tem as definições mais básicas e importantes da imagem, como a dimensão e o centro real da imagem. A estrutura desta classe está representada a seguir, na Figura 6.3:7.

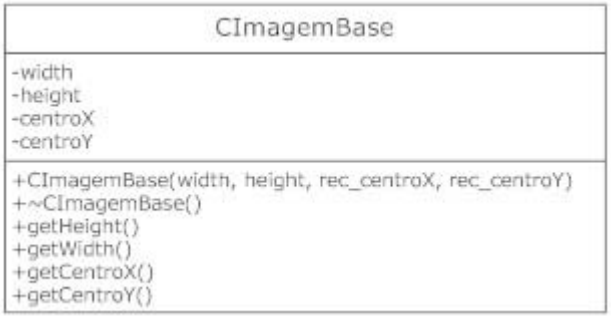


Figura 6.3:7 - Estrutura da classe CImagemBase

6.3.4. A classe CImagemCaptura

Esta classe é a classe responsável pela gestão de memória da imagem capturada. A imagem capturada vai devolver a esta classe um apontador para o inicio da imagem que vai substituir o apontador para o inicio da imagem do tipo openCV aqui criado. Deste modo consegue-se usar a estrutura de imagens do openCV, mas com a imagem capturada pela câmara FireWire e respectiva livraria. Para isto esta classe herda as características da classe CImagemBase e são criadas duas variáveis privadas do tipo apontador, uma para guardar sempre o endereço correcto da nova imagem e outra que corresponde á estrutura da imagem do openCV que contem o apontador para a imagem a ser substituída. Para além destas variáveis, esta classe tem os métodos públicos correspondentes para enviar o novo apontador da imagem, assim como métodos para retornar esse apontador ou posições de memória específicas para um pixel, como mostra a figura seguinte, Figura 6.3:8, da estrutura da classe CImagemCaptura.

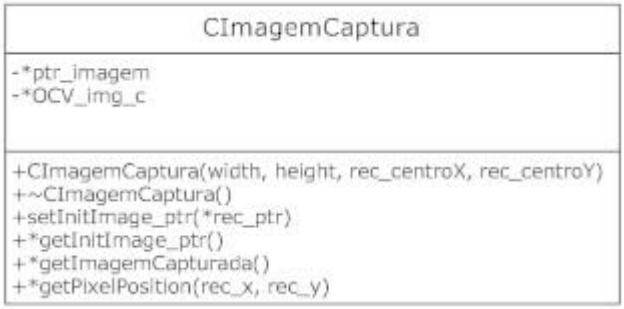


Figura 6.3:8 - Estrutura da classe CImagemCaptura

6.3.5. A classe CImagemFiltrada

Esta classe é a classe responsável pela alocação de memória para guardar o resultado da imagem filtrada. É nesta classe que é implementada o novo tipo de imagem no formato “campo” já anteriormente mencionado. Esta classe herda as características da classe CImagemBase e para além da variável privada contendo a imagem no novo formato, é implementado um método público (LimparFiltrada) para limpar todos os caracteres da imagem, com caracteres de indiferente. A estrutura desta classe está representada na figura seguinte, Figura 6.3:9.

Figura 6.3:9 - Estrutura da classe CImagemFiltrada

6.3.6. A classe CImagemOutput

Esta classe é a classe responsável pela amostragem de resultados. É uma classe de opção que adiciona funções de *debug* e amostragem de resultados às imagens usadas. Esta classe usa uma variável privada que aloca e representa a imagem a ser amostrada. A amostragem final pode ser no formato BGRA (Blue Green Reed Alpha) entendido pelo GUI (*Graphical User Interface*) do Qt ou então em JPEG (*Joint Photographic Experts Group*), guardando a imagem no disco. Para isto são necessários diversos métodos que convertem as diferentes imagens usadas para o formato desejado final, como se pode verificar na figura seguinte, Figura 6.3:10, representativa da estrutura da classe CImagemOutput.

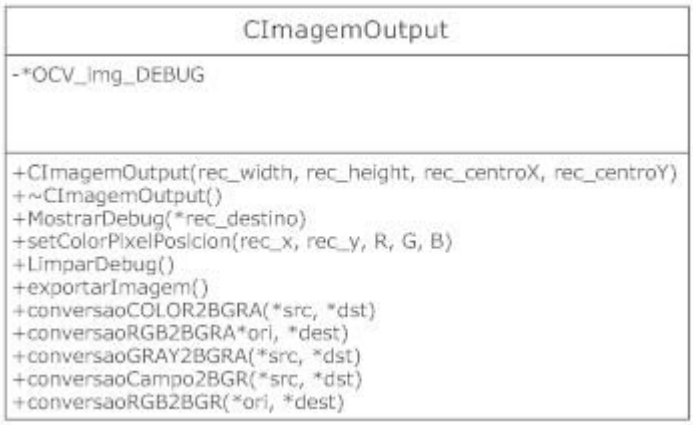


Figura 6.3:10 - Estrutura da classe CImagemOutput

6.3.7. A classe CImagemEspelho

A necessidade desta classe prende-se com o facto de ser necessário converter as distâncias de cada pixel da imagem ao centro do robô para distâncias reais desse mesmo ponto ao centro do robô.

Para isso esta classe contém uma matriz de valores da mesma dimensão da imagem capturada com as respectivas distâncias de cada pixel ao centro do robô. Esta matriz é privada da classe e para se ter acesso a valores desta, existem funções públicas que retornam as distancias, como demonstra a figura seguinte da estrutura desta classe (Figura 6.3:11). Estes valores retornados podem ser em milímetros como em Unidades Espaciais (UE) que são valores inteiros representativos de uma unidade de medida usada futuramente para a localização.

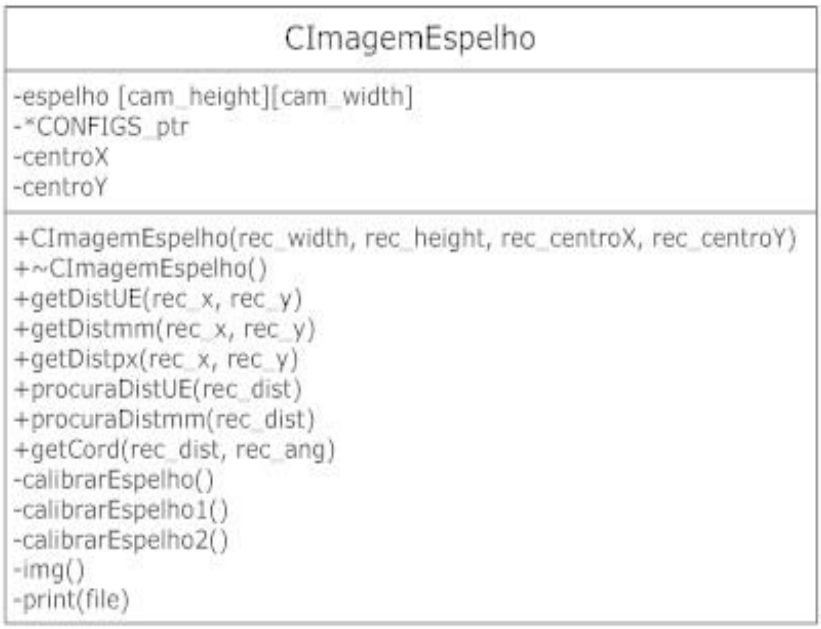


Figura 6.3:11 - Estrutura da classe CImagemEspelho

Deste modo, após o preenchimento desta matriz com a forma do espelho é muito rápido e fácil o acesso a informação da distância desse ponto, pixel, coordenada ao centro do robô.

Inicialmente, no construtor é chamada uma função privada para preencher a matriz com a forma do espelho. Deste modo evita-se a má utilização da matriz como a alteração acidental destes dados.

6.4 A classe CBaseDados

Esta classe tem o objectivo de reter informação do ambiente e do mundo que rodeia o robô. Esta classe recebe dados como coordenadas da localização do robô, da coordenada da bola, e obstáculos. A evolução desta classe é possível e muito fácil, pois é nesta classe que todos os dados estão inseridos e podem ser analisados de diversos modos, levando assim a uma fácil evolução do software. Todos os dados novos obtidos ou calculados são inseridos nesta classe que se limita a armazenar valores recebidos. Assim, é possível saber um último valor inserido, assim como traçar previsões de localizações ou até vectores de movimentação baseada em dados anteriores que também ficam guardados nesta classe.

Os dados inseridos nesta classe, passam sempre por uma pré-avaliação. Com isto é possível rejeitar dados bastante díspares daqueles últimos já inseridos e confirmados como válidos. Esta confirmação é aceite se apenas for possível fisicamente a movimentação de algum objecto dentro deste espaço de tempo. O objectivo deste controlo de inserção de dados é uma tentativa de manter a base de dados íntegra, ou seja, sem dados corruptos ou falsos positivos.

Para isto, esta classe necessita de dois vectores privados para guardar os dados recebidos, assim como algumas variáveis de controlo do estado do robô, como a camisola que usa ou o ultimo comando recebido. São implementados alguns métodos públicos para retorno da posição da bola ou do robô, baseado nos últimos dados recebido e tendo em atenção os dados anteriores e antigos. Foram implementadas funções privadas que verificam a veracidade dos dados recebidos, baseado na velocidade física possível do objecto, tendo em conta o último valor válido. Estes métodos e atributos, estão descritos na figura seguinte (Figura 6.4:1).

Figura 6.4:1 - Estrutura da classe CBaseDados

6.5 A classe CCampo

Esta classe é a classe responsável pela implementação do campo virtual anteriormente mencionado. São usadas as configurações das medidas do campo, para gerar um campo virtual de zeros e uns á escala do campo de jogo numa matriz privada da classe. A implementação das linhas do campo é feita com algoritmos de *[52-56]* pois tem em atenção o *aliasing* provocado nestas. O campo é implementado com uma histerese nas linhas configurável, pois facilita na localização numa primeira fase em que a percepção do espaço ainda é reduzida. Para esta implementação são usadas funções privadas que desenham o campo, estando apenas disponível numa função pública um método para localização retornando a possível posição do robô, como mostra a figura seguinte, da estrutura da classe CCampo, Figura 6.5:1.

Figura 6.5:1 - Estrutura da classe CCampo

O resultado do campo virtual gerado é o mostrado na Figura 6.5:2 em que a vista é bastante afastada. A imagem aparenta estar um pouco esticada, mas é a percepção que é dada, devido ao tipo de letra usada para preencher de zeros e uns não ser perfeitamente quadrada. Apenas estão marcados os pontos brancos existentes em campo, pois são estes os pontos usados para fazer a correspondência entre o campo virtual e os pontos do campo capturados na imagem.

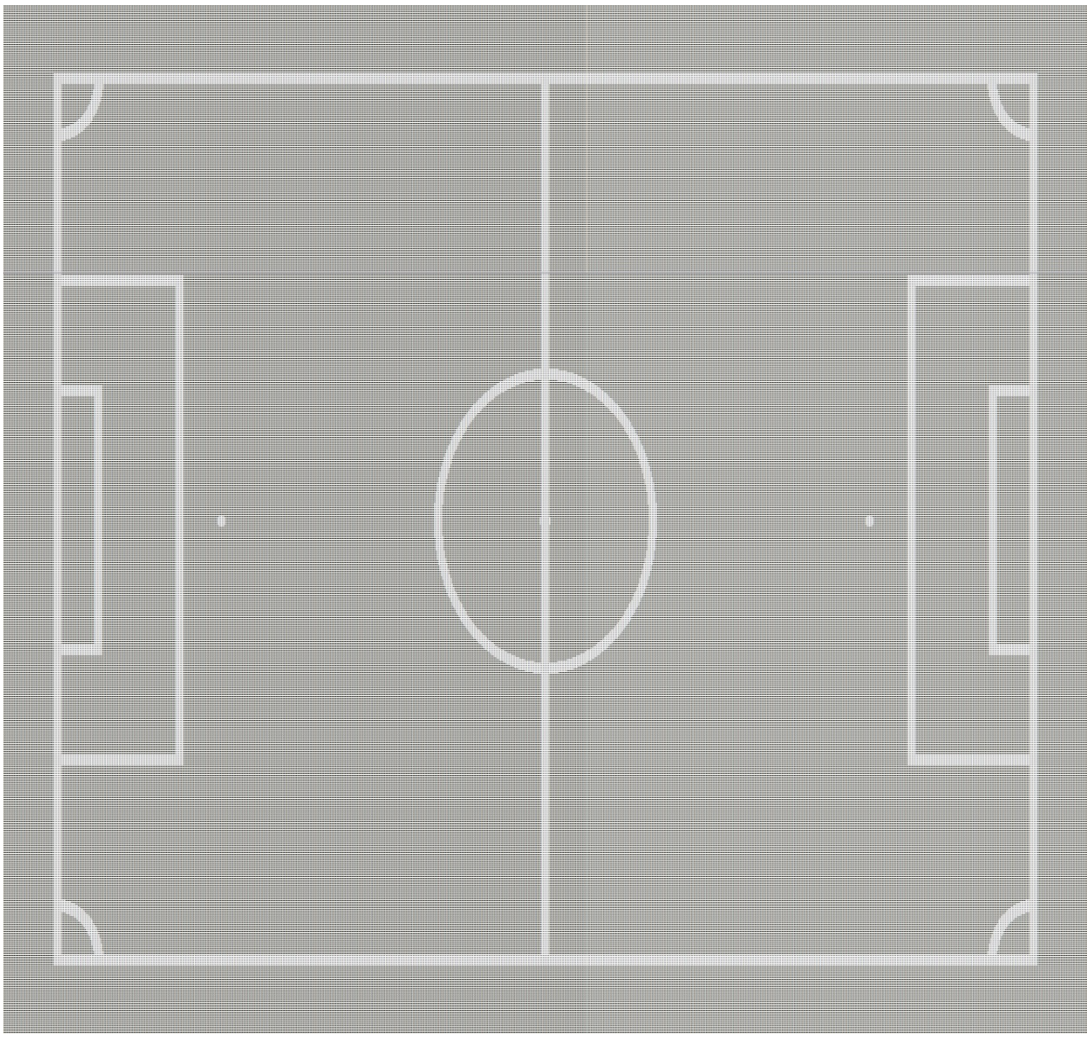


Figura 6.5:2 - Campo virtual gerado (zoom out)

Seguidamente apresenta-se uma imagem mais aproximada do campo na Figura 6.5:3, em que dá para salientar o uso de zeros e uns para a sua construção, assim como o *aliasing* que é retirado.

Figura 6.5:3 - Campo virtual gerado (zoom in)

Capítulo 7

Monitor

Neste capítulo irá ser abordado o software do monitor construído, também conhecido por treinado ou *coach*. Este software é aquele que faz a ponte de ligação o árbitro do jogo que informa a mesa com a *refbox*, e os robôs. Este software pode implementar a gestão de tácticas de jogo, registo dos eventos do jogo, assim como, tomar decisões para jogadas a serem efectuadas. Este tipo de decisão pode ser implementado deste lado do software, pois este contém todas as informações de todos os robôs presentes e pode mais facilmente tomar decisões baseadas nos conhecimentos das posições enviadas por cada robô.

Todas as equipas têm que ter um monitor, visto que é este o único ponto de ligação entre o árbitro/*refbox* e a sua própria equipa. Os dados recebidos através da *refbox* são pré-definidos e disponibilizados antes de cada evento, para que as equipas se possam adaptar.

O software desenvolvido, permite ter acesso à informação relativa de cada robô, assim como, à posição em campo e à posição da bola e ao estado de baterias.

É de notar que à medida que chegam comandos novos da refbox, estes são registados num ficheiro separado e apresentados segundo a sequência de chegada, como mostra a figura seguinte (Figura 6.5:1).

É igualmente visível a comunicação entre o robô 5 e o Monitor, que possibilita ao utilizador controlar os estados principais do robô.

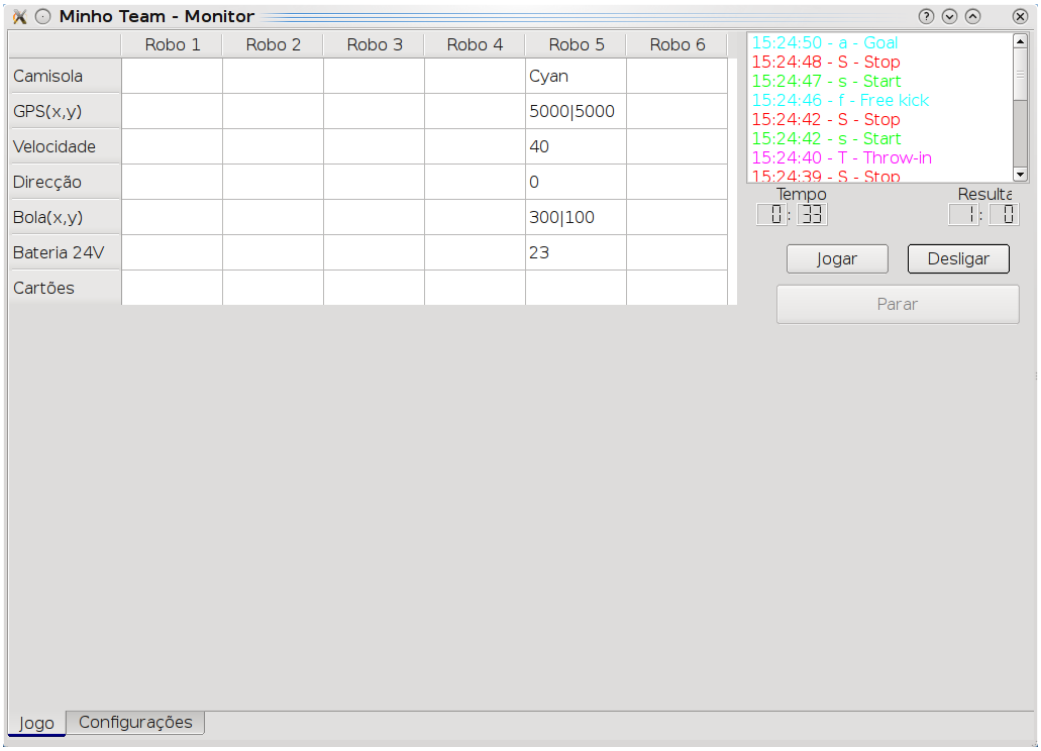


Figura 6.5:1 - Software de monitorização do jogo (Monitor)

Esta monitorização é conseguida, pois em cada trama recebida é recebido um conjunto de parâmetros predefinidos que indicam as informações mais relevantes. Deste modo, é possível detectar alguns problemas localizados durante o jogo.

Dentro deste software é também possível configurar parâmetros como IPs de redes à qual o software se irá ligar, assim como, as dimensões do campo, a cor da camisola da equipa. Estas configurações são necessárias para o início do jogo e devem ser apenas corrigidas num único sítio, como mostra a figura seguinte (Figura 6.5:2).

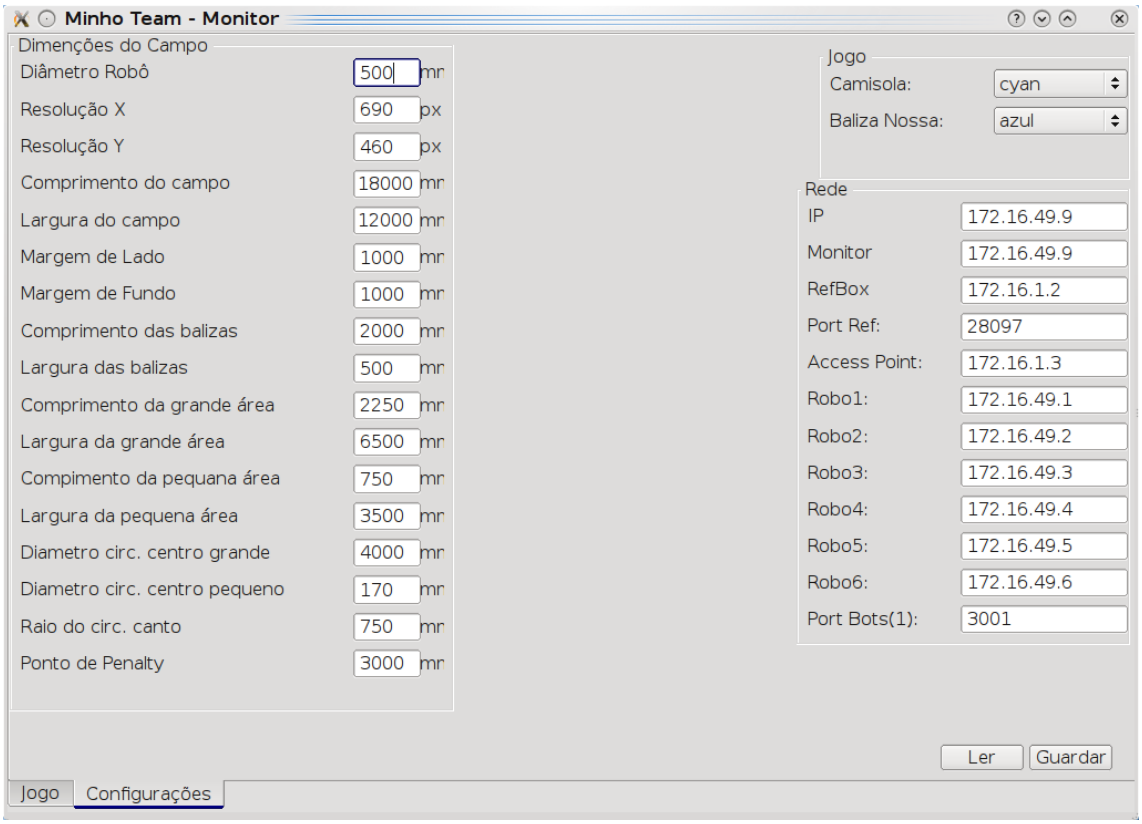


Figura 6.5:2 - Configuração dos parâmetros do Monitor

7.1 Comunicação com Refbox

A comunicação com a Refbox é feita através de um cabo UTP (*Unshielded Twisted Pair*), evitando assim atrasos e diminuindo erros da comunicação. No software desenvolvido, utiliza-se o protocolo TCP/IP para este tipo de comunicação com endereços unicast. No software desenvolvido é reutilizado as classes de rede já previamente construídas e parametrizáveis para uma maior interacção, rapidez e facilidade de desenvolvimento. Este tipo de comunicação é utilizada para a recepção de comandos provenientes do árbitro, que são os comandos principais para o inicio e desenrolar da competição.

7.2 Comunicação com os robôs

A comunicação com os robôs é sempre importante, pois é sempre necessário manter a comunicação entre eles para haver fusão sensorial de todos os dados recebidos. A comunicação com os robôs é toda ela feita

através do monitor, que recebe periodicamente tramas pré definidas dos estados de cada robô, permitindo assim essa fusão sensorial e melhor percepção do mundo que rodeia toda a equipa. A trama enviada é em UDP pois como este envio é cíclico, a perda de uma trama vai ser substituída por uma mais recente, diminuindo a largura de banda e aumentando a velocidade de transmissão. O tipo de informação enviada de cada robô varia desde a sua posição actual, à posição da bola, aos estados internos do próprio robô, como mostra o exemplo seguinte (Figura 7.2:1) da trama enviada do robô para o monitor.

Cod	gpsX	gpsY	BolaX	BolaY	bateria	camisola	velocidadeActual	direccaoActual

Figura 7.2:1 - Exemplo de uma trama enviada do robô para o monitor

Da mesma maneira existe uma trama que também é enviada periodicamente do monitor para todos os robôs, com informações relevantes para o desenrolar da partida. Essa trama é constituída pelo comando emitido pelo árbitro, assim como a estratégia a usar e a camisola da nossa equipa, como mostra a figura seguinte (Figura 7.2:2)

Cod	tecla	estrategia	camisola

Figura 7.2:2 - Exemplo de uma trama entre o monitor e os robôs

Em ambas as tramas segue sempre no seu início uma assinatura que corresponde a um código aceite pelo software para confirmar a veracidade da trama recebida.

7.3 Monitorização de dados

A monitorização de dados é uma parte importante deste software. É importante porque informa aos utilizadores o estado dos robôs. Desde a localização em que estes pensam estar, assim como a localização da bola que estes calculam, até ao estado de processo que envolve o robô e estado do hardware. Estes dados são apresentados numa tabela para de uma forma mais simples e intuitiva, o operador poder controlar facilmente o estado dos robôs.

Seguidamente apresenta-se uma imagem da tabela e dos dados monitorizados na Figura 7.3:1

	Robo 1	Robo 2	Robo 3	Robo 4	Robo 5	Robo 6
Camisola					Cyan	
GPS(x,y)					5000|5000	
Velocidade					40	
Direcção					0	
Bola(x,y)					300|100	
Bateria 24V					23	
Cartões						

Figura 7.3:1 - Exemplo da monitorização do robo

A monitorização é feita através da comunicação a cada robô. Este, na sua trama de comunicação envia o seu estado, como anteriormente, já foi abordado na comunicação com os robôs.

7.4 Registo de eventos de jogo

O registo de eventos de jogo é uma aplicação desenvolvida no software do monitor que é capaz de registar a sequência de caracteres recebidos, assim como, a hora e a ordem pela qual foram enviados. Deste modo, é possível verificar desde a coerência do árbitro que envia os comandos do jogo, testar a reacção dos robôs a cada comando específico. Este registo de eventos é sempre guardado em disco num ficheiro de *log* para uma futura análise dos comandos recebidos e interpretação do jogo.

A imagem seguinte (Figura 7.4:1) mostra um exemplo de sequência de dados recebida da refbox para o monitor e consequentes acções aplicadas.

Figura 7.4:1 - Exemplo de sequência de dados recebidos

Capítulo 8

Conclusões e Perspectivas Futuras

Este trabalho como faz parte integrante do desenvolvimento de uma equipa de robôs futebolistas, não se pode concluir nunca que o trabalho está finalizado. Isto porque o desenvolvimento é continuo e novas ideias surgem chegando a por em questão algumas já implementadas. No entanto, no âmbito desta dissertação, o trabalho desenvolvido foi levado para além daquilo que era pedido, com construção do robô, implementação de um software com fácil possibilidade de evolução, até implementação de software de apoio e desenvolvimento de software com comunicações entre as máquinas. Todo o software desenvolvido tem todo ele uma linha de pensamento alicerçada em bases sólidas e restritas à sua área de acção, e assim, é possível um desenvolvimento mais rápido e focado em certos pontos. É possível futuramente distribuir áreas de trabalho especializada sem que estas sejam influenciadas por outro grupo de trabalho independente.

Ao longo de todo este desenvolvimento surgiram sempre algumas ideias novas ou pontos a melhorar. Desde ideias de construção do robô até ideias de software. Começa-se por se destacar algumas ideias de construção ou materiais por ordem de importância que parecem ser de alguma utilidade.

É importante reforçar as extremidades das bases dos robôs pois existem pontos críticos que podem começar a bilaminar as fibras de vidro. Existem algumas soluções possíveis para isto, desde a colocação de uma capa protectora apenas nas extremidades e a toda a volta das bases, ou então uma capa de protecção relativamente dura à volta do robô, para aguentar os impactos laterais. É também uma possibilidade de substituir as fibras de vidro por fibras de carbono, ganhando assim, mais resistência. A construção de umas novas bases com as novas técnicas apreendidas, evitando assim, alguns erros cometidos anteriormente. Seria muito bom para o robô se fosse possível implementar os dois chutos com um único mecanismo, poupando assim, o peso de mais uma bobine, tornando o robô mais leve e consequentemente mais rápido. A cabeça do robô, ou a parte da visão devem ser feitas num material mais resistente, pois apesar de toda a precisão na construção desta peça, o único ponto fraco é a sua resistência ser bastante débil levando a bastantes oscilações do espelho e consequentemente dificuldades de processamento e alinhamento da câmara. Para segundo plano ficaria a utilização de duas câmaras, não só pela dificuldade inerente ao processamento de ambas e fusão com a visão omni-direccional, mas também, ao baixo poder de processamento que o computador disponibiliza. Isto seria também uma óptima nova aquisição para aumentar o poder de processamento e utilização de algoritmos mais elaborados.

Agora em relação a trabalho futuro dentro da área da programação, existem alguns pontos que poderiam ser retocados ou ideias novas que surgiram para um trabalho futuro. A primeira ideia e de alguma importância foca-se na calibração automática das cores da câmara. É bastante importante manter as cores constantes, evitando assim influências de luz e uma calibração exagerada nas cores para se conseguir detectar todos os tons de cores. Existe também a necessidade de uma calibração automática de distâncias. Existem alguns algoritmos evolutivos que possibilitam esta excentricidade [57-58], pois deste modo é possível a cada robô calibrar a sua forma do espelho em

qualquer altura, independentemente da posição deste. Após estes dois pontos fulcrais é então possível avançar para um algoritmo de localização baseado no método de “Monte-Carlo”[59-64]. Existem outras ideias que surgiram ao longo deste trabalho e que facilitariam ou ajudariam num melhor desempenho, como por exemplo o processamento de imagem estar independente do computador.

Como o processamento de imagem é a parte mais importante e mais pesada de todo o processo, este poderia ser efectuado num GPU de uma gráfica, aumentando assim a sua velocidade e libertando o processador, ou então, ser feito numa placa de desenvolvimento de processamento de imagem separada e exterior.

Uma última futilidade, mas que ajudava bastante em calibrações, é o uso de comandos internos para interpretar o ambiente gráfico desenvolvido, ou seja, a existência de dois programas, um, seria o servidor que é o robô e outro, seria um cliente que se ligaria ao anterior para controlar todo o ambiente gráfico. Deste modo, evitava-se o envio de dados por *wireless* de todo o ambiente gráfico, como imagens e outros pormenores que são irrelevantes para o funcionamento do programa. Assim, conseguia-se uma calibração à distância, mas mais em tempo real, pois evita encriptações e envios de dados desnecessários.

Bibliografia

[1] M. Bellis and About.com:Inventors. *The Robot Story*. Available: http://inventors.about.com/od/rstartinventions/a/RobotStory.htm

[2] M. Bellis and About.com:Inventors. *Timeline of Robots*. Available: http://inventors.about.com/od/roboticsrobots/a/RoboTimeline.htm

[3] Jeff's Robots and Jeffbots. (1997, 10-April-2010). *Ro·bot*. Available: http://www.jeffbots.com/dictionary.html

[4] The History of Computing Foundation. (2007, 10-April-2010). *Timeline of Robotics*. Available: http://www.thocp.net/reference/robotics/robotics.html

[5] The Robocup Federation. (1998-2010, 2-April-2010). *RoboCup*®. Available: www.robocup.org

[6] The Robocup Federation. (1998-2010, 2-April-2010). *About RoboCup*®. Available: http://www.robocup.org/about-robocup/

[7] The Robocup Federation. (2010, 12-April-2010). *RoboCup - Middle Size League Main Page*. Available: http://wiki.msl.robocup-federation.org/wiki/Main_Page

[8] SAR - Soluções de Automação e Robótica. (2010, 10-April-2010). *Cadeira de Rodas Omnidireccional* Available: http://industry.sarobotica.pt/index.php?option=com_content&task=view&id=67&Itemid=84

[9] The Robocup Federation. (2009, 13-April-2010). *Results World Championships 2009: Graz, Austria.*, . Available: http://wiki.msl.robocup-federation.org/wiki/Results_World_Championships_2009:_Graz,_Austria#Soccer_competition

[10] The Robocup Federation. (2008, 13-April-2010). *Results World Championships 2008: Suzhou, China*. Available: http://wiki.msl.robocup-federation.org/wiki/Results_World_Championships_2008:_Suzhou,_China

[11] The Robocup Federation. (2006, 13-April-2010). *RoboCup 2006 - Bremen / Germany*. Available: http://www.robocup2006.org/

[12] The Robocup Federation. (2007, 13-April-2010). *Final Middle Size League Ranks for 2007*. Available:

https://wiki.cc.gatech.edu/robocup/index.php/Middle_Size_League_Schedule

[13] RFC Stuttgart (2010, 12-April-2010). *RFC Stuttgart* Available: http://robocup.informatik.uni-stuttgart.de/rfc/www/

[14] S. Coradeschi, L. Karlsson, P. Stone, T. Balch, G. Kraetzschmar, and M. Asadas, "Overview of RoboCup-99," *AI Magazine* vol. 21, Number 3, p. 8, 2000.

[15] H. Rajaie, U.-P. K¨appeler, O. Zweigle, K. H¨aussermann, A. Tamke, A. Koch, B. Eckstein, F. Aichele, D. DiMarco, A. Berthelot, T. Walter, and P. Levi. (2010, 13-April-2010). RFC Stuttgart, Overview of Hardware and Software. 8. Available: http://robocup.informatik.uni-stuttgart.de/tdp/RobotRfcStuttgart2010.pdf

[16] Maxon Motor AG. (2006, 12-April-2010). *Maxon Motors*. Available: http://www.maxonmotor.com/

[17] RFC Stuttgart. (2010, 13-April-2010). *Control Software of 1. RFC Stuttgart* Available: http://robocup.informatik.uni-stuttgart.de/rfc/www/media/RFCRT_V0.9.tar.bz2

[18] CAMBADA Robotic Soccer - University of Aveiro. (2009, 14-April-2010). *CAMBADA Robotic Soccer*. Available: http://www.ieeta.pt/atri/cambada/

[19] CAMBADA Robotic Soccer University of Aveiro. (2010, 13-April-2010). *CAMBADA Qualification Materials for RoboCup2010: Mechanical Description*. Available: http://www.ieeta.pt/atri/cambada/docs/CAMBADA-mechanical_description-2010.pdf

[20] J. L. Azevedo, M. B. Cunha, A. J. R. Neves, N. Lau, A. Pereira, G. Corrente, F. Santos, D. Martins, N. Figueiredo, J. Silva, J. Cunha, B. Ribeiro, R. Sequeira, L. Almeida, L. S. Lopes, J. M. Rodrigues, and A. J. Pinho. (2010, 13-April-2010). *CAMBADA electrical description 2010*. Available: http://www.ieeta.pt/atri/cambada/docs/CAMBADA-electrical_description-2010.pdf

[21] A. J. R. Neves, J. L. Azevedo, M. B. Cunha, N. Lau, A. Pereira, G. Corrente, F. Santos, D. Martins, N. Figueiredo, J. Silva, J. Cunha, B. Ribeiro, R. Sequeira, L. Almeida, L. S. Lopes, J. M. Rodrigues, and A. J. Pinho. (2010, 13-April-2010). *CAMBADA'2010: Team Description Paper*. Available: http://www.ieeta.pt/atri/cambada/docs/CAMBADA-tdp-2010.pdf

[22] J. L. Azevedo, M. B. Cunha, N. Lau, A. Neves, G. Corrente, F. Santos, A. Pereira, L. Almeida, L. S. Lopes, P. Pedreiras, J. Vieira, D. Martins, N. Figueiredo, J. Silva, N. Filipe, and I. Pinheiro. (2009, 13-April-2010). *CAMBADA'2009: Team Description Paper*. Available: http://www.ieeta.pt/atri/cambada/docs/CAMBADA2009_tdp.pdf

[23] P. Masarati, R. Bucher, H. Mayer, L. Dozio, D. Schleef, D. Gasperini, P. Soetens, P. Mantegazza, M. Neuhauser, and G. Racciu. (2010, 13-April-2010). *RTAI*. Available: https://www.rtai.org/

[24] CAMBADA Robotic Soccer University of Aveiro. (2010, 13-April-2010). *CAMBADA Qualification Materials for RoboCup2010: Software Architecture*. Available: http://www.ieeta.pt/atri/cambada/docs/CAMBADA-software_structure-2010.pdf

[25] Neuro Informatics Group. (2010, 14-April-2010). *Tribots Neuro Informatics Group Web Page*. Available: http://www.ni.uos.de/index.php?id=25

[26] Neuro Informatics Group. (2005, 14-April-2010). *Tribots cronological story*. Available: http://www.ni.uos.de/index.php?id=975&L=0.html

[27] Neuro Informatics Group. (2005, *List of year-by-year innovations*. Available: http://www.ni.uos.de/index.php?id=974&L=0.html

[28] Fraunhofer Institute for Intelligent Analysis and Information System. (2008, 14-April-2010). *TMC200*. Available: http://www.volksbot.de/pdfs/TMC200-en.pdf

[29] R. Hafner, S. Lange, M. Lauer, and M. Riedmiller. (2008, 14-April-2010). *Brainstormers Tribots Team Description*. Available: http://www.ni.uos.de/fileadmin/user_upload/publications/tribotsTDP2008.pdf

[30] R. Hafner, S. Lange, M. Riedmiller, and S. Welker. (2009, 14-April-2010). *Brainstormers Tribots Team Description*. Available: http://www.ni.uos.de/fileadmin/user_upload/publications/tribotsTDP2009.pdf

[31] R. Hafner, S. Lange, M. Lauer, and M. Riedmiller. (2006, 14-April-2010). *Brainstormers Tribots Team Description*. Available: http://www.ni.uos.de/fileadmin/user_upload/publications/tribotsTDP2006.pdf

[32] F. Ribeiro, P. Braga, J. Monteiro, I. Moutinho, P. Silva, and V. Silvas, "New improvements of MINHO Team for RoboCup Middle Size League in 2003," 2003.

[33] F. Ribeiro, I. Moutinho, P. Silva, C. Fraga, and N. Pereiras, "Vision, Kinematics and Game strategy in Multi-Robot Systems like MSL RoboCup," 2004.

[34] Comfibras. (2007, 15-April-2010). *ABC do Fiberglass*. Available: http://www.comfibras.com.br/literatura/1.htm

[35] SolidWorks Corporation. (2010, 15-April-2010). *SolidWorks 3D CAD Design Software*. Available: http://www.solidworks.com/

[36] F. Ribeiro, G. Lopes, T. Silva, A. Freitas, M. Oliveira, and L. Novaiss, "Mobility and agility with an improved sight," 2009.

[37] T. Grimm, *User's guide to rapid prototyping*. Dearborn: Society of Manufacturing Engineers, 2004.

[38] ACM Enterprises. (2009, 14-11-2009). *Maverick Lithium Polymer Battery*. Available: http://www.li-po.co.uk/

[39] Philips Semiconductors. (2000, 15-April-2010). *The I2C-Bus Specification*. Available: http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf

[40] Point Grey Research® Inc. (2010, 15-April-2010). *Flea®2 - Technical Reference Manual*. Available: http://www.ptgrey.com/

[41] F. Ribeiro, P. Silva, I. Moutinho, V. Silva, and N. Pereiras, "Optimization of fast moving robots and implementation of I2C protocol to control electronic devices," 2005.

[42] Robot Electronics. (2009, 15-April-2010). *USB to I2C Communications Module*. Available: http://www.robot-electronics.co.uk/htm/usb_i2c_tech.htm

[43] cplusplus.com. (2000-2010, 16-April-2010). *cplusplus.com - The C++ Resources Network*. Available: http://www.cplusplus.com/

[44] Linux Online Inc. (2010, 16-April-2010). *Linux Online*. Available: http://www.linux.org/

[45] Software in the Public Interest. Inc. (1997-2010, 16-April-2010). *Debian*. Available: http://www.debian.org/

[46] M. Asada, T. Balch, A. Bonarini, A. Bredenfeld, S. Gutmann, G. Kraetzschmar, P. Lima, E. Menegatti, P. Jonker, A. F. Tehrani, T. Nakamura, G. Steinbauer, M. Lauer, Y. Takemura, H. Lu, E. Pagello, F. Ribeiro, T. Schmitt, W.-M. Shen, H. Sprong, S. Suzuki, Y. Takahashi, P. G. Ploeger, F. Schreiber, J. u. v. Eijck, A. Matsumoto, S. S. Ghidary, R. Merry, B. Cunha, D. Lau, S. Ebrahimijam, and O. Zweigles, "Middle Size Robot League Rules and Regulations for 2010," p. 83, 10-03-2010 2010.

[47] Nokia Corporation. (2010, 20-April-2010). *QThread Class Reference*. Available: http://doc.qt.nokia.com/4.6/qthread.html

[48] D. Douxchamps and other contributors. (2000-2010, 20-April-2010). *libdc1394 Homepage*. Available: http://damien.douxchamps.net/ieee1394/libdc1394/

[49] Intel®. (2010, 21-April-2010). *OpenCV*. Available: http://opencv.willowgarage.com/wiki/

[50] Nokia Corporation. (2010, 20-April-2010). *QUdpSocket Class Reference*. Available: http://doc.qt.nokia.com/4.6/qudpsocket.html

[51] Nokia Corporation. (2010, 20-April-2010). *QTcpSocket Class Reference*. Available: http://doc.qt.nokia.com/4.6/qtcpsocket.html

[52] M. Feldman. (1999-2010, 22-April-2010). *Bresenham's Line and Circle Algorithms*. Available: http://www.gamedev.net/reference/articles/article767.asp

[53] Hexar. (2007, 22-April-2010). *Drawing Lines – The Bresenham Algorithm*. Available: http://www.cs.toronto.edu/~smalik/418/tutorial2_bresenham.pdf

[54] C. Flanagan. 22-April-2010). *The Bresenham Line-Drawing Algorithm*. Available: http://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html

[55] T. Ootjers. (1999-2010, 22-April-2010). *Line Drawing Algorithm Explained*. Available: http://www.gamedev.net/reference/articles/article1275.asp

[56] W. Randolph Franklin. (2005, 22-April-2010). *Bresenham Line and Circle Drawing*. Available: http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/bresenham.html

[57] P. Heinemann, F. Streichert, F. Sehnke, and A. Zells, "Automatic Calibration of Camera to World Mapping in RoboCup using Evolutionary Algorithms," *IEEE International Congress on Evolutionary Computing,* p. 8, 2006.

[58] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. New York: Oxford University Press, Inc., 1996.

[59] A. Merke, S. Welker, and M. Riedmiller, "Line Based Robot Localization under Natural Light Conditions," presented at the ECAI 2004 - Workshop on Agents in Dynamic and Real-Time Environments, Valencia, Spain, 2004.

[60] B. Hengst and R. Sheh, "A Fast Vision Sensor Model: Matching Edges With Nightowl," presented at the 2004 Australasian Conference on Robotics and Automation, 2004.

[61] M. Lauer, S. Lange, and M. Riedmillers, "Calculating the Perfect Match: an Efficient and Accurate Approach for Robot Self-Localization," *Springer Berlin / Heidelberg,* vol. 4020/2006, pp. 142-153, 2006.

[62] W. Burgard, D. Fox, D. Hennig, and T. Schmidt, "Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids," presented at the In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 1996.

[63] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "IEEE International Conference on Robotics and Automation (ICRA99)," 1999.

[64] P. Heinemann, J. Haase, and A. Zells, "A Novel Approach to Efficient Monte-Carlo Localization in RoboCup," *Springer Berlin / Heidelberg,* vol. Volume 4434/2009, pp. 322-329, 2009.