Hugo Filipe Vieira Duarte

Licenciado em Ciências da Engenharia Eletrotécnica
e de Computadores

Utilização de sensores bioelétricos em modelos de redes de Petri IOPT
- Aplicação ao controlo de um quadricóptero -

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e Computadores

Orientador: Prof. Doutor Luís Filipe dos Santos Gomes, FCT/UNL

Júri:

Presidente: Prof. Luís Filipe Figueira Brito Palma

Vogais: Prof. Luís Filipe dos Santos Gomes

Prof. Anikó Katalin da Costa

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Março, 2016

Utilização de sensores bioelétricos em modelos de redes de Petri IOPT
- Aplicação ao controlo de um quadricóptero-

Dedicado à Sociedade cientifica, pelo que ela representa, por um mundo melhor.

Agradecimentos,

Em primeiro lugar, gostaria de expressar o meu agradecimento ao Professor Luís Gomes pela sua disponibilidade, pelos conselhos e sua orientação ao longo desta dissertação. Estou muito agradecido por me ter cedido todo o material necessário na elaboração deste projeto, também por me ter dado a oportunidade de trabalhar como investigador no *GRES* (*R*&D Group on Reconfigurable and Embedded Systems*)* e ainda pela possibilidade de elaboração de artigo para revista de comunidade científica internacional. Agradecido também ao meu colega Mestre Rogério Campos Rebelo por ter participado e acompanhado este trabalho.

Agradecido também à Professora Anikó Costa pela sua ajuda preciosa ao longo do meu percurso escolar.

Um grande obrigado a todos os docentes, amigos e colegas da instituição *FCT* da Universidade Nova de Lisboa, que direta ou indiretamente contribuíram para o meu percurso académico. Em especial, aos meus amigos Mestre Pedro Amaral, Tiago Lamelas, Mestre Ana Patrícia, Nuno Laje, Dr. Tiago Maria, Duarte Saraiva, Mestre João Gião, Pedro Alves, e muitos mais, por me terem acompanhado durante a dissertação como em quase todo o curso.

Também quero agradecer o apoio e dedicação dos meus Pais, Sr. Sérgio Duarte e Sra. Ana Paula, e da minha querida irmã, Sra. Inês Filipa, que sempre me incutiram o desejo de terminar o que eles com muito custo me proporcionaram. Estou eternamente grato por todo o bem que me fizeram e o vosso amor incondicional.

Um muito obrigado à minha namorada, Ana Rita Calheiros, que me deu forças para continuar sempre que a vontade e a inspiração faltaram.

Os trabalhos associados a esta dissertação beneficiaram de resultados obtidos no projeto de investigação *Petri-Rig* (Ambiente de desenvolvimento de sistemas embutidos baseado em redes de Petri), com bolsa financiada pela fundação *FCT (*Fundação para a Ciência e a Tecnologia*)* com a referência *BI_61_15*_Petri-*Rig*/*PTDC/EEI-AUT/2641/2012*.

Resumo

Até hoje, o uso de eletroencefalogramas, eletromiogramas e eletro-oculogramas foram direcionados para auxiliar, aumentar ou reparar a ação cognitiva humana ou as suas funções sensoriais ou motoras, mantendo-se um distanciamento em relação às restantes aplicações onde estas tecnologias podem ser potencialmente utilizadas.

Nesta dissertação foi implementado um controlador especificado através de um modelo de redes de Petri IOPT aplicado ao controlo de um quadricóptero, utilizando um conjunto de sinais bioelétricos obtidos de um capacete EPOC, fabricado pela Emotiv. Foi desenvolvida uma biblioteca de funções de aquisição e processamento dos sinais bioelétricos apta a ser reutilizada em diferentes aplicações.

Como resultado, os controladores especificados em modelos de redes de Petri IOPT e gerados pelo ambiente de desenvolvimento IOPT-Tools podem utilizar sinais giroscópicos ou bioelétricos (estes dependentes de expressões faciais, estados de emoção ou pensamentos específicos), para definir as ações de controlo de qualquer dispositivo ou aparelho eletrónico.

Palavras-chave: *Eletroencefalograma, Eletro-oculograma, Eletromiograma, Giroscópio, Interface Cérebro-Computador, Redes de Petri, IOPT, Controlador Digital, quadricóptero.*

Abstract

Until now, the use of electroencephalogram, electromyogram and electrooculogram was directed to help, enhance or repair the human cognitive action or its sensory or motor functions, maintaining a distance in relation to the other applications where these technologies could be potentially used.

This work implemented a digital controller specified through a model of IOPT Petri nets applied to the control of a quadcopter, using a set of bioelectric signals obtained from EPOC helmet, made by Emotiv. A library of functions was developed for acquisition and processing bioelectric signals amenable of being reutilized in different applications.

As a result, the controllers specified using IOPT Petri nets models and generated by IOPT-Tools can use gyroscopic or bioelectric signals (these dependent on facial expressions, emotion states or specific thoughts) to set control actions on any system or electronic device.

Keywords: *Eletroencephalogram, Eletrooculogram, Eletromyogram, Gyroscope, Brain-Computer Interface, Petri Nets, IOPT, Digital Controller, Quadcopter.*

Índice

INTRODUÇÃO 1
1.1 ENQUADRAMENTO E MOTIVAÇÃO 1
1.2 ÂMBITO E OBJETIVOS 2
1.3 ESTRUTURA DO DOCUMENTO 4
FUNDAMENTOS E TECNOLOGIAS 7
2.1 O CÉREBRO HUMANO, OS SINAIS NEURONAIS E AS SUAS ONDAS CEREBRAIS 7
2.1.1 Cérebro Humano 8
2.1.2 Neurónios 9
2.1.3 Ondas Cerebrais 10
2.2 ANÁLISE DOS SINAIS ELÉTRICOS 12
2.2.1 Conceito de Eletroencefalografia 13
2.2.2 Conceito de EOG e EMG 13
2.3 CONCEITO DE SENSOR GIROSCÓPIO 15
2.4 VISÃO GERAL DA INTERAÇÃO HUMANO-SISTEMA 15
2.5 INTERFACES CÉREBRO-COMPUTADOR 16
2.5.1 Características principais das ICC's 17
2.5.2 Tipos de ICC's 18
2.5.3 Alguns ICC's não invasivos no mercado 19
2.5.4 Limitações dos ICC's atuais 19
2.5.5 O futuro dos ICC's 20
2.5.6 Trabalhos Relacionados 21
2.6 21
2.7 REDES DE PETRI 21
2.7.1 Redes de Petri IOPT (Input-Output Place-Transition) 22
2.8 NAVEGAÇÃO DE QUADRICÓPTEROS 24
AMBIENTES DE DESENVOLVIMENTO E EXECUÇÃO 27
3.1 EMOTIV EPOC 27
3.1.1 Descrição do Hardware 28

3.1.2 Descrição de Softwares ... 30
3.2 QUADRICÓPTERO ... 36
3.2.1 Aspetos Técnicos ... 36
3.2.2 Descrição do Software ... 36
3.3 FERRAMENTAS DE DESENVOLVIMENTO DE SOFTWARE ... 38
3.3.1 IOPT-TOOLS ... 38
3.3.2 Microsoft Visual Studio ... 41
AQUISIÇÃO E PROCESSAMENTO DE SINAIS ... 43
4.1 DESCRIÇÃO DA SOLUÇÃO PROPOSTA ... 43
4.1.1 Estrutura Hierárquica ... 44
4.1.2 Funções e variáveis de Controlo ... 44
4.1.3 Ficheiros de controlo ... 45
4.1.4 Ficheiros de Comunicação ... 46
4.1.5 Algoritmo EMOTIV ... 46
4.2 AÇÕES DE CONTROLO ... 48
4.2.1 Implementação de eventos ... 48
4.2.2 Implementação de estados ... 53
4.2.3 Implementação Comunicativa ... 56
4.3 PROJETO DE DESENVOLVIMENTO DE SOFTWARE ... 57
IMPLEMENTAÇÃO DO SISTEMA DE CONTROLO ... 59
5.1 DESCRIÇÃO DA SOLUÇÃO PROPOSTA ... 59
5.1.1 Proposta de controlador ... 61
5.1.2 Arquitetura de Navegação ... 62
5.2 MODELAÇÃO DO SISTEMA ... 65
5.2.1 Definição/Especificação do Sistema ... 66
5.2.2 Complexidade Modular ... 66
5.2.3 Simulação e execução da Rede ... 71
5.2.4 Validação/Verificação dos Estados ... 72
5.2.5 Geração automática de código ... 72
5.3 APLICAÇÃO ICC ... 73
5.3.1 Código fonte adicionado ao projeto ... 73
5.3.2 Interface Gráfica ... 73
5.3.3 Interface de comandos ... 74
5.4 TROCA DE MENSAGENS ... 75
5.4.1 Camada de comunicação EPOC-Computador ... 75
5.4.2 Camada de comunicação Computador-Quadricóptero ... 75
5.4.3 Comunicação entre programas ... 75
5.4.4 Regime de Troca de Mensagens ... 77
5.5 APLICAÇÃO DO QUADRICÓPTERO ... 78

5.5.1 Implementação do controlo do quadricóptero *78*
5.5.2 Comandos do Utilizador *79*
5.5.3 Interface gráfica de Simulação *80*
5.6 RESULTADOS E DISCUSSÃO 83
5.6.1 Descrição dos testes práticos *83*
5.6.2 Resultados Experimentais *84*
CONCLUSÕES E TRABALHO FUTURO 87
6.1 CONTRIBUIÇÕES DA DISSERTAÇÃO 88
6.2 ADVERSIDADES 88
6.3 TRABALHO FUTURO 89
REFERÊNCIAS 91
ANEXOS 97
A. TIPOS DE ABORDAGEM TECNOLÓGICA CAPAZ DE MEDIR A ATIVIDADE CEREBRAL E AS SUAS DESVANTAGENS PRINCIPAIS NA PESQUISA ICC [7]. 97
B. TABELA DE COMPARAÇÃO DE ICC'S COMERCIAIS (ADAPTADO [46]). 98
C. AVALIAÇÃO DO CONFORTO FEITA POR UTILIZADORES DOS PRODUTOS QUE RECOLHEM DADOS VIA EEG, DISPONÍVEIS NO MERCADO [7]. 99
D. ASPETOS TÉCNICOS DO CAPACETE EMOTIV EPOC [9]. 99
E. MANUAL DE PROGRAMAÇÃO DE FUNÇÕES DA BIBLIOTECA "EPOCONTROL" 100
F. MANUAL DE PROGRAMAÇÃO DE FUNÇÕES DA APLICAÇÃO "DRONEPOC" 107
G. MANUAL DE PROGRAMAÇÃO DE FUNÇÕES DA APLICAÇÃO "ARDRONE" 110
H. FLUXO DE PROCEDIMENTOS DO UTILIZADOR PARA REALIZAÇÃO DE EXPERIÊNCIA. 112
I. ALGORITMO DE DECISÃO DE EVENTOS 113

Lista de Figuras

Figura 1: Fluxo de processamento da interface cérebro-computador, ilustrando a aquisição de sinal, fases de processamento e exemplos de aplicações do utilizador final. 3
Figura 2: Diagrama de tecnologias do projeto. 3
Figura 3: Ilustração das áreas e funções cognitivas do cérebro [3] 8
Figura 4: Ilustração das áreas cognitivas simétricas do cérebro [4]. 9
Figura 5: Ilustração dos tipos base de neurónios cerebrais [3]. 10
Figura 6: Gráficos que representam as ondas *Delta, Theta, Alpha* e *Beta* [6]. 11
Figura 7: Ilustração da posição dos pontos diferença [7]. 14
Figura 8: Exemplo de interação Humano-Sistema (adaptado de [15]). 16
Figura 9: Modelo de evolução autómata com uma contribuição autodisciplinar (adaptado de [16]). 16
Figura 10: Funcionamento de um ICC. 17
Figura 11: Elementos principais de um ICC. 18
Figura 12: Exemplo de uma Rede de Petri [26]. 23
Figura 13: Sistema de coordenadas com os ângulos *Roll, Pitch* e *Yaw* [27]. 25
Figura 14: Sistemas de eixos e movimentos possíveis do "*AR.Drone*" [28]. 25
Figura 15: Capacete EPOC da EMOTIV [30]. 28
Figura 16: Capacete e acessórios [31]: 1 – Emotiv EPOC, 2 – Solução Salina, 3 – Elétrodos. 29
Figura 17: Posição correta do capacete [32]. 29
Figura 18: Posições do sistema internacional 10-20 [17]. 30
Figura 19: Menu principal do Emotiv *Control Panel*. 31
Figura 20: Leitura dos sinais expressivos no Emotiv *Control Panel*. 31
Figura 21: Leitura dos sinais Emotivos no Emotiv *Control Panel*. 32
Figura 22: Leitura dos sinais cognitivos no Emotiv *Control Panel*. 32
Figura 23: Leitura dos sinais *EEG* descaracterizada no Emotiv *TestBench*. 33
Figura 24: Leitura dos sinais Giroscópicos descaracterizados no Emotiv *TestBench*. 33
Figura 25: Diagrama da integração do *SDK* e do software da Emotiv com uma aplicação. 34
Figura 26: Interface da aplicação *AR.Freeflight* via *smartphone* para o utilizador. 37
Figura 27: Interface da aplicação *FreeFlight3* via *tablet* para o utilizador. 37
Figura 28: Fluxo do funcionamento típico da IOPT-Tools [25]. 38
Figura 29: Página inicial da *GRES*. 39
Figura 30: Menu principal das IOPT-Tools. 39
Figura 31: Arquivos de código fonte gerado das IOPT-Tools. 40
Figura 32: *Login* do depurador das IOPT-Tools. 41

FIGURA 33: PROCESSO VISUAL DO DEPURADOR DAS IOPT-TOOLS EM FUNCIONAMENTO. 41
FIGURA 34: FLUXO DE PROCESSAMENTO ICC, ILUSTRANDO A AQUISIÇÃO DE SINAL E FASES DE PROCESSAMENTO, COM ÊNFASE NA SOLUÇÃO *SDK "*EPOC*CONTROL"*. 44
FIGURA 35: FUNÇÕES E VARIÁVEIS DE CONTROLO APTAS A SEREM CHAMADAS A CONTROLAR UM SISTEMA. 45
FIGURA 36: COLEÇÃO DE FICHEIROS DEDICADOS AO CONTROLO E ENTRADA DE SISTEMAS. 46
FIGURA 37: COLEÇÃO DE FICHEIROS DEDICADOS À COMUNICAÇÃO INTRAPROCESSOS. 46
FIGURA 38: COLEÇÃO DE FICHEIROS *SDK* EMOTIV EPOC. 46
FIGURA 39: COMPORTAMENTO IDEAL DE UMA APLICAÇÃO QUE UTILIZA O EPOC [32]. 47
FIGURA 40: PROTOCOLO DE TREINO EXPRESSIVO [32]. 50
FIGURA 41: PROTOCOLO DE TREINO COGNITIVO [32]. 50
FIGURA 42: TIPOS DE EVENTO NO TREINO COGNITIVO. 51
FIGURA 43: TIPOS DE EVENTO NO TREINO EXPRESSIVO. 51
FIGURA 44: TREINO "PUSH" E "LIFT" EXECUTADO NA APLICAÇÃO. 52
FIGURA 45: CONJUNTO DE FICHEIROS NECESSÁRIOS À CONSTRUÇÃO DE UM PROJETO COM O EMOTIV EPOC. 57
FIGURA 46: DIAGRAMA DE TECNOLOGIAS DO PROJETO, BASEADO EM *IHS*, POSSÍVEL ATRAVÉS DA ADIÇÃO DO ALGORITMO DE TRADUÇÃO FORNECIDO NO CAPÍTULO ANTERIOR. 60
FIGURA 47: ESTRUTURA DE INTEGRAÇÃO DO SISTEMA. 61
FIGURA 48: FLUXO DE IMPLEMENTAÇÃO. 62
FIGURA 49: REPRESENTAÇÃO DO MODO DE CONTROLO *TAKE OFF* E *LANDING* DO QUADRICÓPTERO. 64
FIGURA 50: REPRESENTAÇÃO DO MODO DE CONTROLO DE SUBIDA, DESCIDA, MOVIMENTAÇÃO PARA A ESQUERDA E DIREITA DO QUADRICÓPTERO. 64
FIGURA 51: REPRESENTAÇÃO DO MODO DE CONTROLO DE ROTAÇÃO ESQUERDA OU DIREITA E AVANÇO OU RECUO DO QUADRICÓPTERO. 65
FIGURA 52: METODOLOGIA ADOTADA NA CONSTRUÇÃO DA *RDP* [38]. 65
FIGURA 53: ESQUEMA GLOBAL DO SISTEMA. 66
FIGURA 54: MODELO DA REDE DE PETRI DO PROJETO. 67
FIGURA 55: REPRESENTAÇÃO DO COMPORTAMENTO DO SENSOR HORIZONTAL. 69
FIGURA 56: REPRESENTAÇÃO DO COMPORTAMENTO DO SENSOR VERTICAL. 69
FIGURA 57: REPRESENTAÇÃO DO COMPORTAMENTO DO SINAL *WINKRIGHT()* 70
FIGURA 58: APLICAÇÃO DA FUNÇÃO *BLINK() DA RDP*. 71
FIGURA 59: AMBIENTE DE SIMULAÇÃO DA REDE NAS IOPT-TOOLS. 71
FIGURA 60: GERAÇÃO DE ESPAÇO DE ESTADOS. 72
FIGURA 61: CONSOLA *WIN32* COM AS SAÍDAS DA *RDP* E OS SINAIS DO EPOC. 74
FIGURA 62: INTERFACE DE COMANDOS DO PROJETO "*DRON*EPOC". 74
FIGURA 63: MENSAGEM ENTRE AS DUAS APLICAÇÕES COM AS AÇÕES DE CONTROLO. 77
FIGURA 64: SOLUÇÃO DO PROJETO "*ARDRONE* " NO AMBIENTE *VISUAL STUDIO*. 78
FIGURA 65: CLASSES DE CONTROLO DE ENTRADA DO *DRONE* NO AMBIENTE *VISUAL STUDIO*. 78
FIGURA 66: TECLAS DO TECLADO USADAS COMO INTERFACE DE COMANDO DO QUADRICÓPTERO. 80
FIGURA 67: AMBIENTE PRINCIPAL DE NAVEGAÇÃO DO QUADRICÓPTERO. 81
FIGURA 68: INTERFACE DE VÍDEO *STREAM* DO QUADRICÓPTERO COM A RESPETIVA LINHA DE COMANDOS E O VÍDEO *STREAM* GERADO. 82
FIGURA 69: TECLAS DO TECLADO USADAS COMO INTERFACE DE COMANDO DO QUADRICÓPTERO E DO EPOC. 83
FIGURA 70: AMBIENTE GRÁFICO DE UTILIZADOR. 84

FIGURA 71: EXPERIÊNCIA PRÁTICA. 85

Lista de Tabelas

Tabela 1: Banda de frequências e anotações [5]. 10
Tabela 2: Resumo de alguns tipos de métodos utilizados (adaptado de [7]). 13
Tabela 3: Tipos de Evento do EPOC[32]. 49
Tabela 4: Estados 53
Tabela 5: Tradução dos estados recebidos da expressão facial do utilizador 54
Tabela 6: Tradução do estado mental do utilizador. 55
Tabela 7: Configurações em informação do e/ou para o estado do sistema 56
Tabela 8: Suporte de comunicação. 56
Tabela 9: Comandos e funções de entrada e saída do sistema. 63
Tabela 10: Legenda das entradas e saídas da Rede de Petri 68
Tabela 11: Legenda das funções da interface de comandos referentes ao "*DronEpoc*". 74

Lista de Códigos Fonte

Código 1: Inicialização das variáveis de estado, evento e *profile.* 47
Código 2: Código fonte da conexão ao Emotiv *EPOC.* 100
Código 3: Código fonte que apaga os treinos gravados. 100
Código 4: Código fonte que utiliza uma assinatura existente. 101
Código 5: Código fonte que grava uma assinatura 101
Código 6: Código fonte que permite afetar a sensibilidade de todos os sinais cognitivos. 101
Código 7: Código fonte que permite afetar a sensibilidade de cada um dos quatro sinais cognitivos. 102
Código 8: Funções definidas em *interrupt().* 102
Código 9: Pseudo código da receção da resposta aos eventos no *EPOC.* 103
Código 10: Pseudo código utilizado ao aceitar ou rejeitar um treino cognitivo 104
Código 11: Pseudo código da estrutura da função *newAction()* que processa os eventos 106
Código 12: *Pseudo código* na inicialização do canal *socket.* 107
Código 13: Função que envia mensagens através de um endereço. 107
Código 14: *Includes* a fazer no ficheiro *net_io.cpp.* 107
Código 15: Chamada de Funções de inicialização. 108
Código 16: Chamada de funções de entrada. 108
Código 17: Chamada das funções de saída. 109
Código 18: Função que define o tempo entre cada ciclo 109
Código 19: Função evocada ao sair do ciclo da rede. 109
Código 20: Código fonte das teclas *standard* utilizadas no controlo do *Drone.* 110
Código 21: Código fonte dos comandos enviados pela aplicação *"DronEpoc"* utilizados na aplicação *"ARDrone"* para controlar do *Drone.* 110
Código 22: Pseudo código da classe *AsynchronousSocketListener.* 111
Código 23: Código fonte da variável que recebe a mensagem passada pelo cliente ("*DronEPOC"*) 111

Abreviaturas

- ***API*** – *Application Programming Interface*
- ***BCI*** – *Brain-Computer Interface*
- ***DLL*** – *Dynamic Linked Libraries*
- ***EEG*** – Electroencefalografia
- ***EMG*** – Electromiografia
- ***EOG*** – Electrooculografia
- **EPOC** – Capacete da Emotiv
- ***ERP*** – *Event-related potencial*
- ***GRES*** - *R&D Group on Reconfigurable and Embedded Systems*
- ***GUI*** – *Graphic User Interface*
- ***HSI*** – *Human system interaction*
- ***Hz*** – *Hertz,* unidade de medida para frequências
- ***IDE*** – *Integrated Development Environment*
- ***IP*** – *Internet Protocol*
- **IOPT** - *Input-Output Place-Transition*
- **IOPT-Tools** – IOPT ferramenta *WEB*
- ***PNML*** - *Petri Net Markup Language*
- ***TCP*** – *Transmission Control Protocol*
- ***RdP*** – Rede de Petri
- ***SDK*** – Software *Developer Kit*
- ***UAV*** – *Unmanned aerial vehicle*

1

Introdução

"Biofeedback is a new way of learning, a way of relearning, or realizing for the first time, what the body already knows."

Maxwell Cade, British scientist

1.1 Enquadramento e Motivação

No cérebro humano existem biliões de neurónios que compõem o córtex cerebral e que respondem a diferentes atividades e características particulares como cores e movimentos, sons e palavras, sensações e cheiros, e onde distintos grupos de neurónios, ativos ou inativos, dão origem a uma experiência consciente. Durante a última década registaram-se desenvolvimentos notáveis de algoritmos que decifram essa atividade neuronal.

À medida que é quebrada a fase de perceção eletrónica destes sinais, fase que refere a compreensão dos padrões de eletroencefalografia produzidos pelos grupos de neurónios do cérebro de um indivíduo, é mapeada a atividade cerebral contribuindo com soluções de controlo em aplicações.

Não obstante, o ser humano foi vivendo adaptando-se à evolução e ao progresso do mundo que o rodeia, nos mais diversos setores, e onde atualmente reside lado a lado com uma enorme oferta de tecnologias. Os rápidos avanços nas ciências neuronal e eletrónica tornaram possível a comunicação bidirecional

com o cérebro, dando origem ao aparecimento de interfaces cérebro computador (*ICC*[1]) em regimes de interação humano-sistema (IHS[2]) reproduzindo uma ligação comunicativa direta entre o cérebro e um dispositivo externo.

Considera-se cada vez mais importante para o futuro no desenvolvimento de sistemas embutidos que se consiga incluir este tipo de abordagens.

1.2 Âmbito e Objetivos

O primeiro objetivo desta dissertação passa por construir um *Kit*[3] de desenvolvimento de software (*SDK*[4]), inserido no processamento de sinal descrito na Figura 1, para ser utilizado em conjunto com modelos de redes de Petri IOPT e com a ferramenta das IOPT's[5], permitindo aos modelos de controladores gerados integrarem esta nova biblioteca e serem capazes de usar um capacete de aquisição de dados de modo a participar nas instruções do modelo.

Para isso é então explorado o uso do capacete Emotiv EPOC, portador de elétrodos que registam a atividade elétrica com recurso a métodos de estudo *EEG*[6]*/EMG*[7]*/EOG*[8], operando numa forma não-evasiva, com o objetivo de tornar qualquer sistema de controlo capaz de usar padrões de reconhecimento diferenciados sob a forma de expressões faciais, estados de emoção ou pensamentos específicos de um indivíduo.

A sua finalidade é a de descrever os sinais de entrada nos modelos de redes de Petri, com função de comando de entrada de qualquer dispositivo ou aparelho eletrónico para o qual o sistema de controlo for construído.

---

*1 Brain Computer Interface (*BCI*), em Inglês.*
2 Human System Interaction (HSI) em Inglês..
3 Conjunto de funcionalidades.
4 Software *Development Kit.*
5 Input-Output Place Transition.
6 Electroencephalography.
7 Electromyography.
8 Electrooculography.

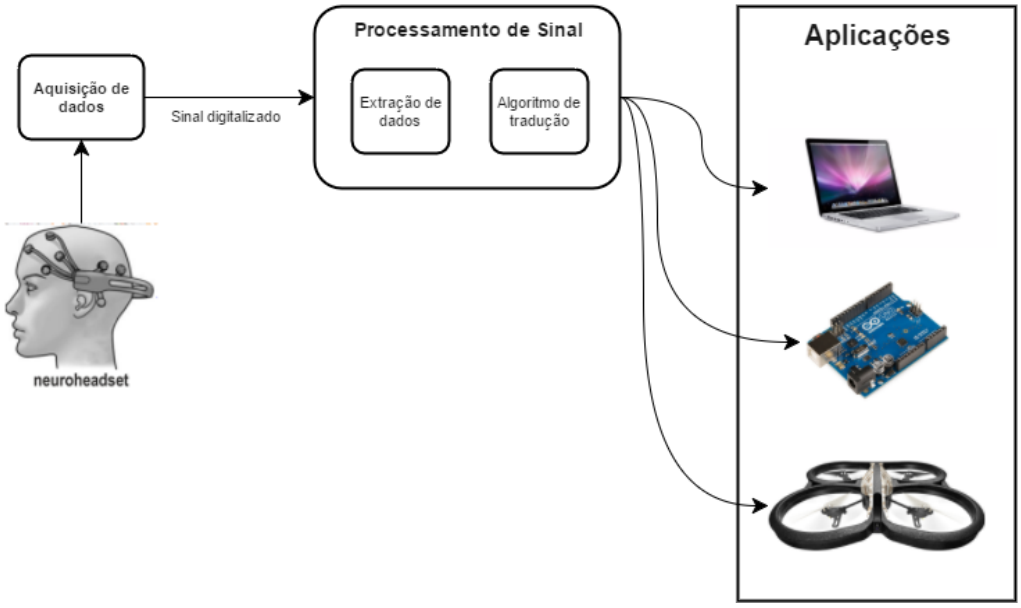


Figura 1: Fluxo de processamento da interface cérebro-computador, ilustrando a aquisição de sinal, fases de processamento e exemplos de aplicações do utilizador final.

Entretanto, o segundo objetivo deste trabalho passa por apresentar uma solução prática, representada pela Figura 2, a partir da biblioteca desenvolvida anteriormente, para dirigir um quadricóptero, para efeitos de validação. O modelo é construído com recurso a redes de Petri IOPT, o código de execução gerado pela ferramenta IOPT-Tools, integrando sinais vindos do capacete de modo a controlar a sua navegação.

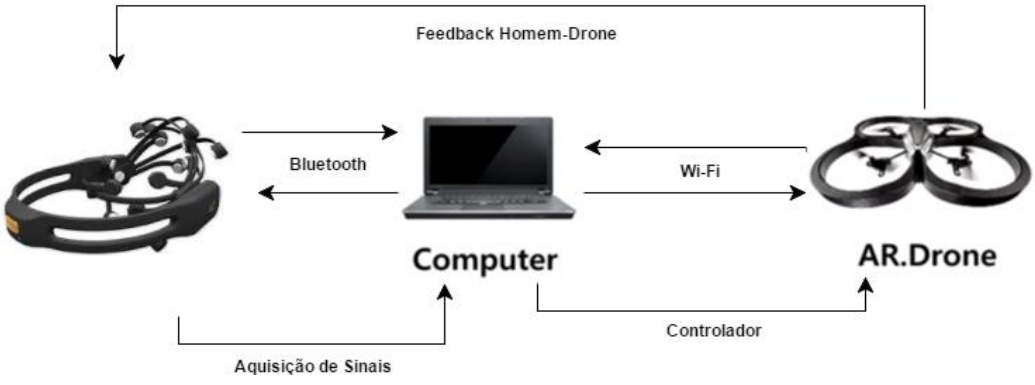


Figura 2: Diagrama de tecnologias do projeto.

Numa última fase é simulado o estado do modelo, ao mesmo tempo que é simulado graficamente o comportamento real específico. Em seguida é analisado o cenário do modelo enquanto é permitido ensaiar o suporte de controlo e avaliar o feedback da navegação do veículo.

Apesar da solução final do projeto implicar o controlo de navegação de um quadricóptero é importante realçar que este exemplo de aplicação serve de validação prática do modelo, sendo que o propósito final será permitir abordar qualquer tipo de aplicação ou sistema, sendo apresentadas algumas possíveis soluções de futuro ao longo do documento.

Por fim é fornecido um suporte eletrónico que inclui documentação, código e utilitários para que técnicos de desenvolvimento de software e hardware consigam utilizar esta abordagem no momento de desenvolver as suas aplicações de acordo com um padrão de desenvolvimento de sistemas modelados por Redes de Petri.

1.3 Estrutura do Documento

Esta dissertação está dividida em seis capítulos. O primeiro capítulo pretende fazer reconhecer o interesse, o propósito do trabalho a desenvolver.

O segundo capítulo aborda os conceitos gerais sobre sinais neuronais, aquisição de dados via *EEG*, *EMG*, *EOG e* sensores giroscópicos, conceitos e soluções de interface no ramo da neurociência e em geral, o formalismo de redes de Petri e ainda conceitos de navegação de quadricópteros, tudo isto antes de se proceder ao desenvolvimento do projeto, de forma a ser feito um enquadramento prévio das temáticas abordadas. Ainda são referidos alguns projetos realizados no âmbito desta área de investigação.

O terceiro capítulo tem como objetivo dar a conhecer as ferramentas necessárias no decorrer deste trabalho, identificando o capacete Emotiv EPOC tanto ao nível de hardware como os seus ambientes de desenvolvimento e execução e descrevendo os aspetos importantes sobre o quadricóptero *Ar.Drone*. Por fim é feita uma descrição da plataforma de desenvolvimento IOPT-Tools e alguns comentários sobre as linguagens usadas no decorrer do projeto.

O quarto capítulo conta com o trabalho e a estrutura desenvolvida no *Kit* de desenvolvimento de software a ser utilizado em conjunto com as IOPT-Tools.

O quinto capítulo descreve a abordagem utilizada no desenvolvimento do protótipo, a interação do capacete com o controlador, as camadas de comunicação entre dispositivos, a estrutura de controlo do quadricóptero e as interfaces concebidas, e ainda os procedimentos tomados durante os testes práticos, comentando os resultados obtidos.

O sexto capítulo contém conclusões sobre todo o trabalho desenvolvido, críticas aos resultados menos bons e intenções nos futuros projetos.

Por fim incluem-se as referências e anexos ao projeto, contendo manuais de utilização das interfaces criadas, tabelas e fluxogramas de suporte ao documento.

2

Fundamentos e Tecnologias

" A inteligência é o único meio que possuímos para dominar os nossos instintos."

Sigmund Freud, Austrian neurologist

Tendo em conta o objetivo de criar um *Kit* de desenvolvimento de Software para operar em conjunto com as IOPT-Tools, e posteriormente desenvolver um sistema capaz de usar os sinais produzidos no cérebro para controlar a navegação de um *drone*, existiu a necessidade de: conhecer melhor como funciona a atividade elétrica gerada pelos neurónios, mencionar os métodos utilizados para adquirir os sinais cerebrais, perceber o que se espera de um giroscópio, explicar o que são *IHS*'s, ficar a conhecer os aspetos mais importantes da área dos ICC's e alguns trabalhos já realizados relacionados com interfaces cérebro-computador, entender o que as redes de Petri representam e ainda compreender o sistema de navegação de um quadricóptero.

2.1 O Cérebro Humano, os sinais Neuronais e as suas Ondas Cerebrais

A ciência que estuda o cérebro e as suas funções é conhecida como Neurociência. Dentro desta área duas são essenciais no desenvolvimento deste tipo de tecnologias, a Psicologia que estuda os pensamentos e comportamentos humanos, e a Neurofisiologia que examina a saúde da atividade cerebral [1].

2.1.1 Cérebro Humano

O cérebro é o centro do sistema nervoso do ser humano. Ele contém todos os centros de recetores e interpretadores sensoriais, de controlo de movimentos, de análise de informação, de experiências emocionais, etc. Os centros destas atividades estão localizadas em diferentes partes do córtex cerebral (Figura 3). Apesar disso não parece existir uma localização específica do controlo consciente. Esta consciência é fruto da interação de todos os sistemas que se encontram no cérebro [2][3].

A consciência é uma qualidade que pertence à mente, considerando abranger qualificações como subjetividade, autoconsciência, sapiência, e a capacidade de compreender a relação entre si e um ambiente. Embora o pensamento lógico e racional seja muitas vezes atribuído ao hemisfério esquerdo e as atividades intuitivas e criativas sejam atribuídas ao hemisfério direito, como descrito na Figura 4, na maioria das pessoas é bastante simétrico, ou melhor, os dois hemisférios estão interligados, e mesmo que haja atividades cerebrais que podem ser claramente localizadas num dos hemisférios, a maior parte do tempo os dois hemisférios estão ativos [2][4].

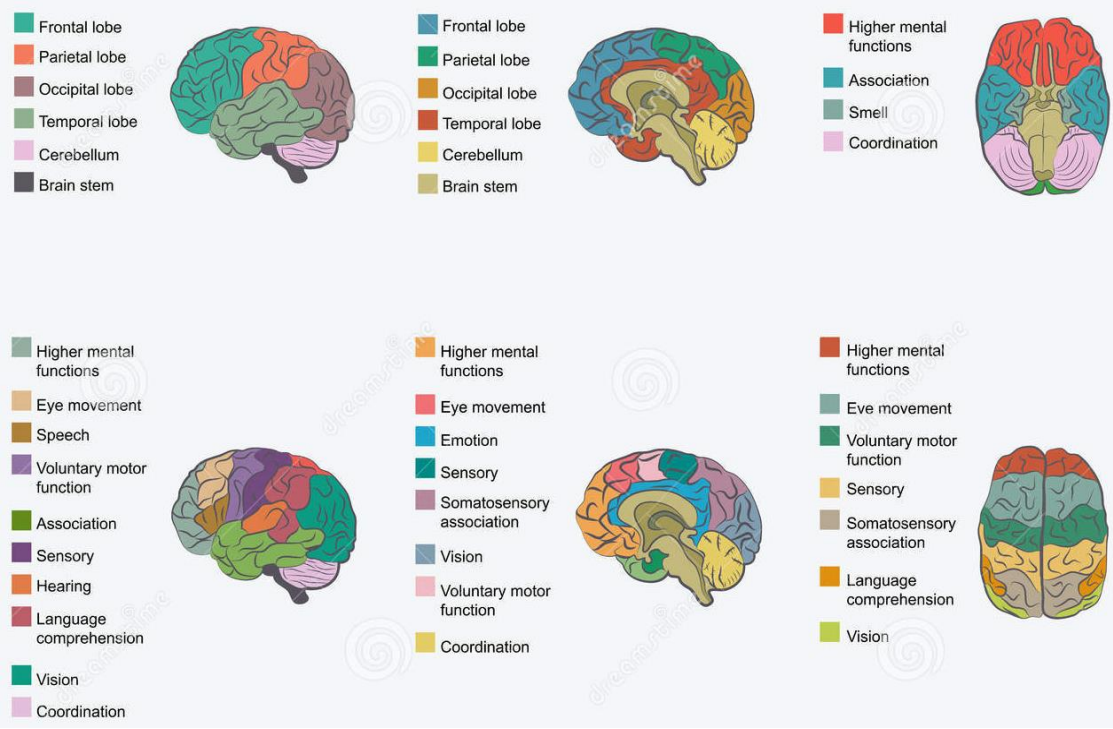


Figura 3: Ilustração das áreas e funções cognitivas do cérebro [3].

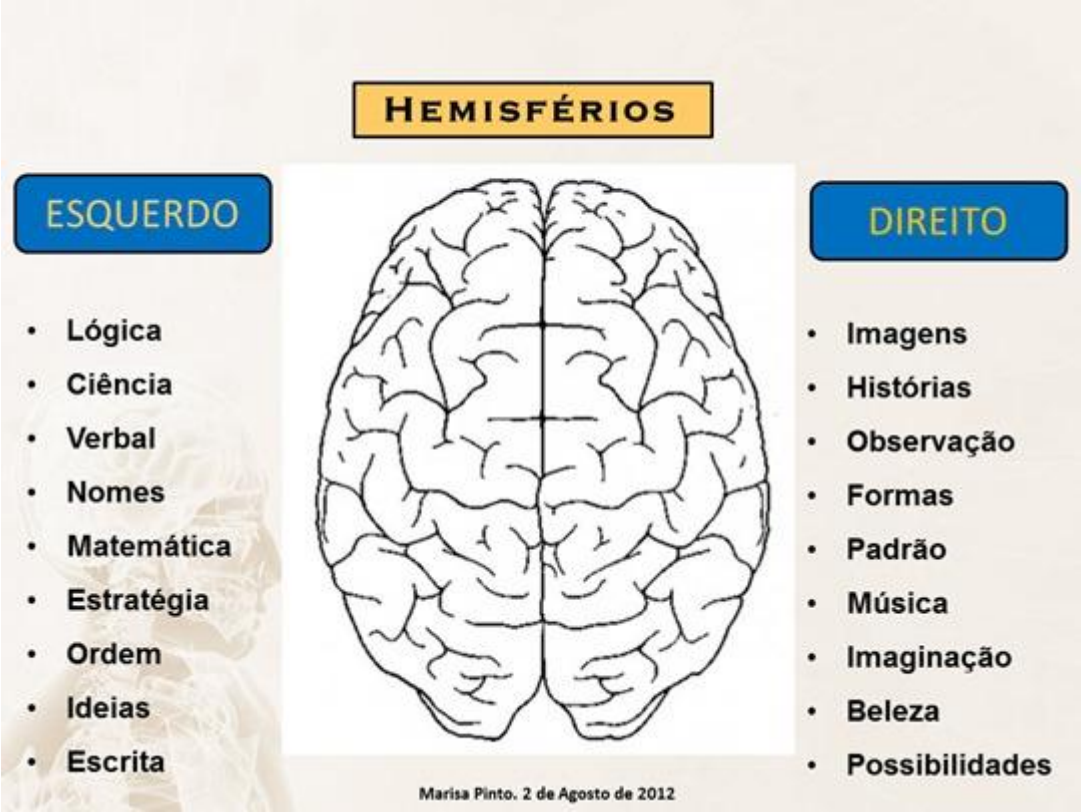


Figura 4: Ilustração das áreas cognitivas simétricas do cérebro [4].

2.1.2 Neurónios

Um neurónio é uma célula nervosa do cérebro, com uma representação ilustrada na Figura 5. O cérebro humano é composto por aproximadamente 80 a 100 bilhões destes neurónios, com a capacidade de receber e transmitir sinais eletroquímicos, variando entre o estado ativo e inativo em consequência da variação do seu potencial elétrico [3]. Esta atividade elétrica pode ser observada através de eletroencefalografia de forma não-invasiva.

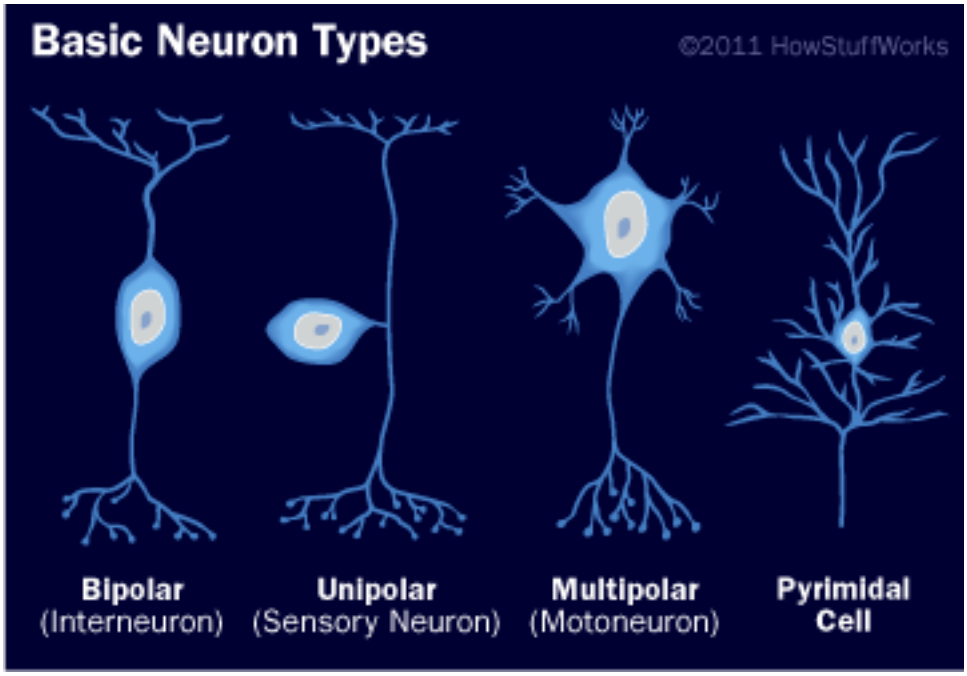


Figura 5: Ilustração dos tipos base de neurónios cerebrais [3].

2.1.3 Ondas Cerebrais

O campo elétrico do cérebro emite sinais em diferentes faixas de frequências e com diferentes valores de potência, como representado na Tabela 1 e na Figura 6 [2][5][6].

Tabela 1: Banda de frequências e anotações [5].

Banda de frequência	Notação	Frequência de alcance (*Hz*)
Delta	δ	<4
Theta	θ	4-8
Alpha	α	8-13
Beta	β	13-25
Gamma	γ	>25

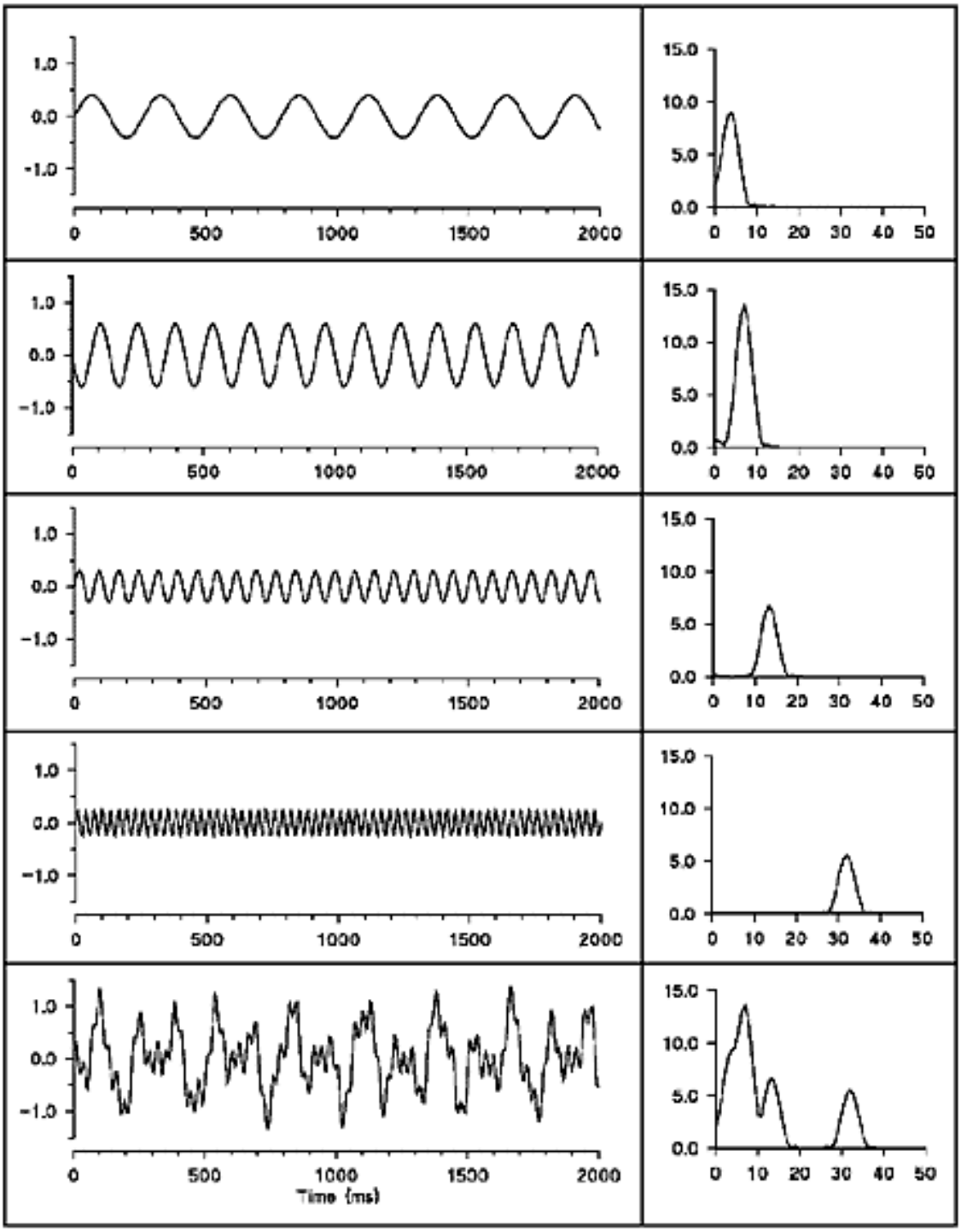


Figura 6: Gráficos que representam as ondas *Delta, Theta, Alpha* e *Beta* [6].

Ondas *Gamma* - Uma das características principais deste tipo de ondas é o sincronismo da atividade em muitas áreas do cérebro. Não são fáceis de detetar devido à sua fraca potência.

Ondas *Beta* - Representam a consciência atenta, a atenção para com exterior e o ambiente que o rodeia, o raciocínio lógico, consciente e analítico. Estas ondas de alta frequência ainda relatam inquietação, *stress*, ansiedade e espirito crítico.

Ondas *Alpha* - São ondas que representam um estado relaxado, ou sentidos como audição, cheiro, gostos, etc. São muito importantes em combinação com outras ondas cerebrais devido à sua ligação com ondas de frequências mais baixas do subconsciente.

Ondas *Theta* - Estas ondas representam o subconsciente. São observadas durante o sono, meditação, ou durante picos de experiências e estados criativos.

Ondas *Delta* - São as ondas cerebrais de frequência mais baixa e representam o inconsciente. Encontramos este tipo de ondas num estado de sono profundo, como também em várias combinações com outras ondas cerebrais. Elas podem representar a intuição, a curiosidade, etc.

Os diferentes estados de consciência do ser humano podem ser descritos com combinações destes conjuntos de ondas. Na maioria das vezes o que é registado não é apenas uma categoria de ondas cerebrais, mas sim uma combinação de ondas cerebrais que interagem entre si.

2.2 Análise dos Sinais Elétricos

Existem diversos processos que possibilitam analisar as atividades elétricas de origem no cérebro, sendo que deles divergem as amplitudes e frequências. A própria gama de frequências também se sobrepõe em certa medida, sendo esse um componente significativo, por exemplo no sinal obtido através de eletroencefalografia, crucial para se fazer considerações peculiares na sua análise. A Tabela 2 apresenta uma visão geral sobre alguns tipos de processos e suas gamas de amplitude/frequência [7].

Tabela 2: Resumo de alguns tipos de métodos utilizados (adaptado de [7]).

Signal	Amplitude Range		Frequency Range(Hz)	
	from	to	from	to
EEG	2 μV	100 μV	0.5	100
EOG	10 μV	5 mV	0	100
EMG	50 μV	5 mV	2	500
ECG	1 mV	10 mV	0.05	100

2.2.1 Conceito de Eletroencefalografia

Eletroencefalografia (*EEG*) trata-se de uma análise neurológica que regista e quantifica a atividade elétrica produzida naturalmente no cérebro, obtendo uma visão da sua condição atual. O processo passa por registar as alterações nos potenciais elétricos em diferentes zonas do cérebro humano, colocando elétrodos no couro cabeludo do indivíduo, ou tomando uma forma invasiva, amplificando o seu sinal de maneira significativa. A frequência, largura de banda e amplitude do sinal nas diversas zonas do cérebro permitem estabelecer padrões únicos e característicos. Esta atividade contínua é distributivamente espacial em torno da superfície do cérebro e pode ser descrita como não estacionária. *Event-related Potencials (ERP)* é um dos tipos de análise feita por uma *EEG* à resposta medida do cérebro quando o indivíduo é exposto a determinados estímulos, cognitivos ou por eventos sensoriais [8].

2.2.2 Conceito de *EOG* e *EMG*

Os eletromiogramas (EMG) e os eletro-oculogramas (EOG) são técnicas distintas, enquanto os *EMG* leem sinais gerados pelo movimento muscular, os *EOG* leem sinais gerados pelo movimento dos olhos.

Eletro-oculografia (EOG) - São potenciais electro oculares utilizados para monitorizar os movimentos dos olhos. O sinal resultante é chamado eletro-oculograma. Este método mede a importante fonte de perturbação originária da polaridade elétrica do globo ocular, como é ilustrado na Figura 7 [8][7][9].

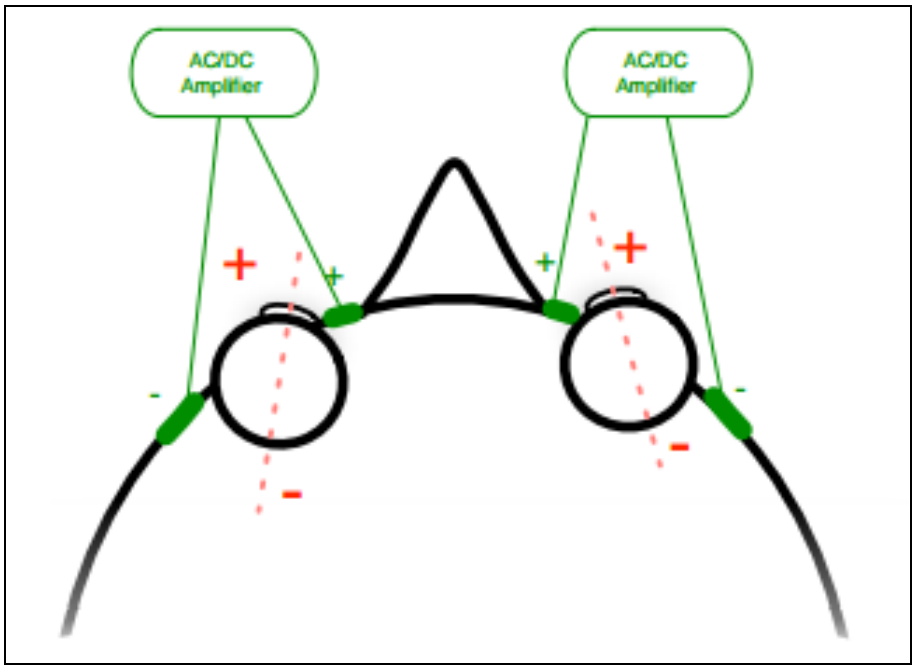


Figura 7: Ilustração da posição dos pontos diferença [7].

O efeito é um reflexo do campo elétrico induzido que se move quando os olhos se movem nas suas órbitas [10]. Este tipo de atividade funciona numa faixa de frequência ampla, sendo a máxima frequência inferior a 4 *Hz* [11].

Eletromiografia (EMG) - Deteta a potência elétrica gerada pelas células musculares durante a contração muscular, quando estas células são eletricamente ativadas, ou seja, reflete a tensão feita pelos músculos da cara, neste caso de estudo, representando atividades neuromusculares. Esta atividade tem uma ampla faixa de frequência, sendo máxima a frequências superiores a 30 *Hz* [11].

Estudos demonstraram que a atividade das *EMG's* e *EOG's* podem gerar sinais que afetam fenómenos neurológicos utilizados num sistema de ICC*'s*, sendo por isso necessário e importante ter este tipo de métodos presente num sistema como este. Ou seja, possuem características distintivas utilizadas tanto para reconhecer e extrair sinais considerados ruído com algoritmos de remoção de ruído, como também são facilmente utilizados para classificar os sinais como parte de controlo de ICC's [8][11]. Estas duas técnicas são aproximadamente iguais entre diferentes utilizadores de ICC*'s,* isto porque os seus dados são obtidos por grupos iguais de sinais de saída faciais e musculares.

2.3 Conceito de sensor Giroscópio

O **giroscópio** é um sensor interno, inercial, que avalia a orientação e rotação sobre os parâmetros internos da posição relativamente a um eixo de referência, cuja propriedade de precisão giroscópica é de grande utilidade no controle de movimento rotacional [12][13].

Além de ser integrado em aplicações como bússolas, aeronaves, sistemas de posicionamento global (*GPS*[9]), etc., os giroscópios foram introduzidos atualmente em produtos eletrónicos de consumo em massa, onde os fabricantes o incorporaram na tecnologia mais moderna. A integração do giroscópio neste tipo de dispositivos permite assim o reconhecimento de movimento num espaço tridimensional [12].

2.4 Visão geral da Interação Humano-Sistema

Conhecido como ***IHS*** (*HSI* em Inglês), trata-se de uma matéria que relaciona diferentes áreas da ciência. A interação entre os humanos e as máquinas acontece através da interface do utilizador, como se demonstra na Figura 8, e que recorre a uma topologia de regras descritas na Figura 9. Esta pretende focar-se em três princípios [14]:

Aprendizagem – facilidade de uso que cada utilizador atravessa na sua interação com um sistema, obtendo o melhor desempenho possível.

Flexibilidade – diversidade de interações existentes entre utilizadores e sistemas na troca de informações.

Robustez – nível de suporte oferecido ao utilizador como resposta aos objetivos pretendidos.

Resumidamente baseia-se numa disciplina voltada para o estudo, conceção, construção e implementação de sistemas com interação humana.

[9] *Global Positioning System.*

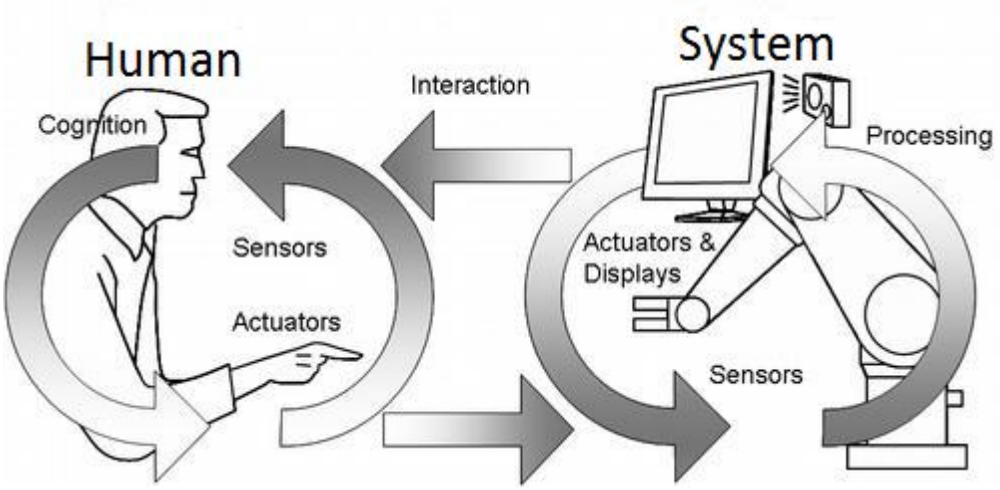


Figura 8: Exemplo de interação Humano-Sistema (adaptado de [15]).

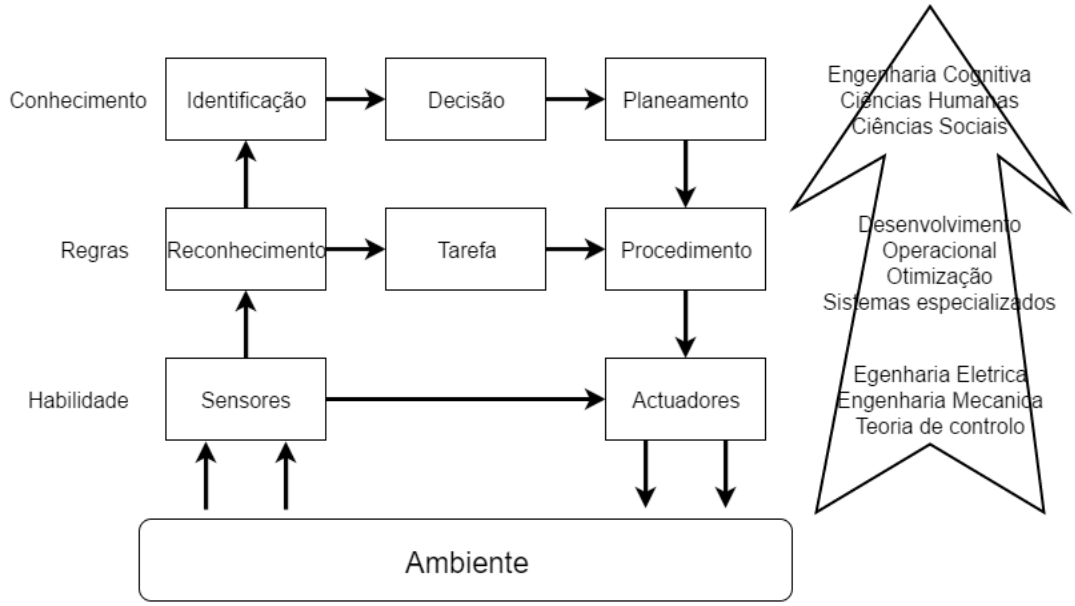


Figura 9: Modelo de evolução autómata com uma contribuição autodisciplinar (adaptado de [16]).

2.5 *Interfaces Cérebro-computador*

Um **ICC**[1] (*BCI* em Inglês) é uma tecnologia que gira em torno do campo da neurociência, sendo descrito como uma via de comunicação direta entre o cérebro e o computador, como ilustrado na Figura 10, onde lê e analisa diversos tipos de ondas cerebrais e as converte em ações integradas num sistema/computador [17][18].

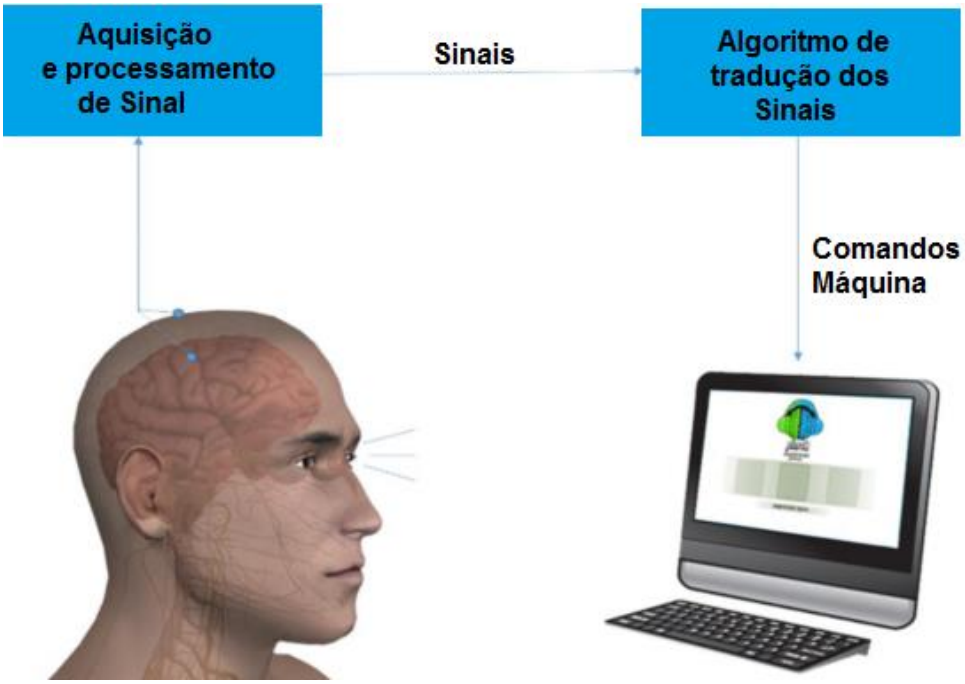


Figura 10: Funcionamento de um ICC.

Em ICC's que usam *EEG's* o processo de análise de sinal pode variar, desde o potencial ser evocado por um estímulo externo (*ERP*[10]) ou analisando-o na sua ausência de estímulo, designando-se por *EEG* espontâneo ou potencial espontâneo [17].

Nos últimos anos, os sistemas ICC tornaram-se significativamente mais práticos e precisos, através do uso de tecnologia de processamento de sinal e algoritmos de aprendizagem [18]. O processo padrão de identificação de pensamento baseia-se no desempenho do algoritmo de classificação utilizado. Os dados obtidos por *EEG* contêm ruído, influenciado principalmente por movimentos musculares, movimentos dos olhos, piscar de olhos, tornando-se por vezes difícil de identificar o sinal [5].

2.5.1 Características principais das ICC's

Existem quatro componentes definidos para o correto funcionamento de um ICC, como representa a Figura 11, seguida com a respetiva legenda.

[10] Event-Related Potencial.

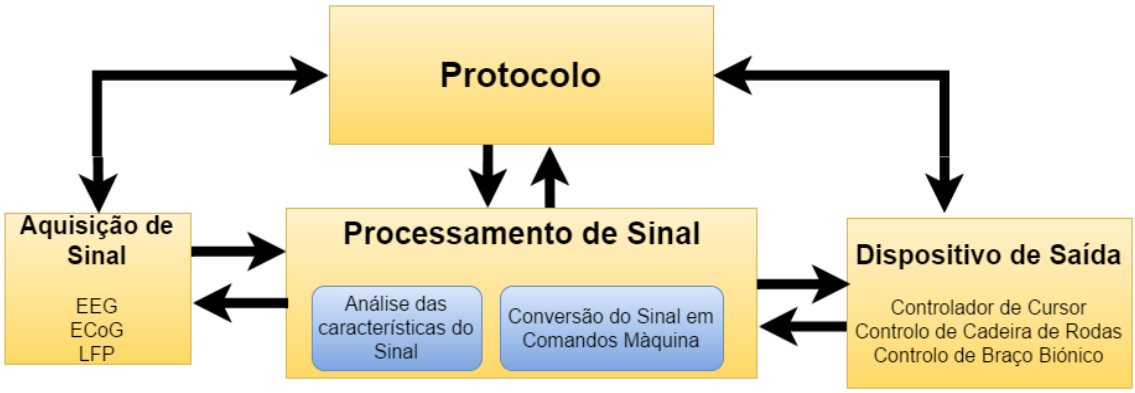


Figura 11: Elementos principais de um ICC.

Aquisição de sinal – O sistema capta o sinal proveniente do cérebro, praticamente em tempo real. A atividade do cérebro é normalmente medida por diferenças de potenciais adquiridos por elétrodos [17].

Processamento de sinal – Conversão da informação em bruto, ou seja, análise das características do sinal retirando a informação útil do sinal. De seguida a máquina usa o sinal previamente filtrado no processo anterior e transforma-o em comandos que possam ser compreendidos pelo dispositivo de saída [17].

Dispositivo de saída – Ações realizadas num dispositivo de saída através de um ICC, como por exemplo o movimentar de um cursor no ecrã de um computador, fazer controlar um quadricóptero, controlar um braço robótico, conduzir uma cadeira de rodas ou mesmo controlar processos fisiológicos tais como mexer um membro [17][18].

Protocolo – Regras implementadas para comunicação entre os componentes. Um fator importante para o correto funcionamento de um ICC é o protocolo utilizado [17].

2.5.2 Tipos de ICC's

O principal fator de classificação das ICC's diz respeito à técnica usada para captar os sinais emitidos pelo cérebro. No anexo A está disponível um quadro com alguns exemplos de tecnologia que medem a atividade cerebral, explicitando algumas das suas desvantagens.

Não invasivos - As ICC's não invasivas são uma abordagem que já se provou bastante útil no desenvolvimento de formas de comunicação. Baseia-se

na maior parte dos casos em *EEG's* e tem a grande vantagem de não expor o utilizador a uma cirurgia. Contudo, as técnicas não invasivas têm canais de comunicação com capacidade limitada, por vezes insuficiente, e outras limitações descritas mais à frente [17].

Invasivos - As ICC's invasivas utilizam elétrodos intracranianos para aquisição dos dados, o que permite um sinal com pouco ruído e um alto nível de capacidade de controlo sobre os dispositivos. Este tipo de técnicas são potencialmente perigosas pois requerem a implantação de elétrodos na massa cinzenta do cérebro que pode levar a infeções e lesões teciduais permanentes [17].

2.5.3 Alguns ICC's não invasivos no mercado

Atualmente existem no mercado alguns modelos de ICC's, na sua maioria baseados em capacetes *EEG*. No anexo B está afixado uma tabela representativa de alguns desses produtos, seguido de um gráfico no anexo C que representa a avaliação feita por parte dos utilizadores do conforto de cada um [7].

2.5.4 Limitações dos ICC's atuais

Apesar da investigação deste campo ser relativamente recente, nas últimas duas décadas fizeram-se grandes avanços na investigação deste tipo de tecnologia. A precisão dos sistemas ICC aumentou significativamente e com isso o tempo de treino necessário para os usar foi reduzido [17].

Apesar destes avanços na tecnologia há algumas questões que continuam por resolver. A maioria dos sistemas ICC não estão preparados para o uso contínuo na vida diária, necessitando de um esforço excessivo de configuração e calibração, e alguma dificuldade no momento de utilização [17].

Apesar de ser das formas mais fáceis de ler os sinais cerebrais, a *EEG* requer que os elétrodos sejam humedecidos e que o contacto entre o elétrodo e o couro cabeludo seja o ideal [17], não sendo esse um aspeto positivo.

Ponderadas todas as opções, a escolha do ICC para o projeto a desenvolver recaiu no modelo EPOC da Empresa Emotiv, pelo facto de possuir múltiplos elétrodos e um *SDK* bastante completo [17].

2.5.5 O futuro dos ICC's

A aplicabilidade mais ampla de tecnologia ICC para uso recreativo requer melhoramentos que não estiveram em foco nas pesquisas feitas para utilização médica. Nos pontos seguintes serão explicadas as melhorias que têm estado em foco nas pesquisas sobre ICC's para que esta tecnologia possa ser adotada para aplicações lúdicas de uso diário [17].

Elétrodos secos - Na maioria dos ICC's atualmente no mercado utilizam elétrodos que necessitam de ser humedecidos, o que faz com que a sua preparação consuma demasiado tempo. Esta é uma das principais razões para que esta tecnologia ainda não tenha sido adotada em massa. Os elétrodos dependem de um gel que pode secar facilmente, o que compromete a leitura dos sinais *EEG*. A inclusão de elétrodos secos flexíveis em tecidos têxteis poderá ser uma realidade no futuro para facilitar a aplicação desta tecnologia [17][18].

Redução da necessidade de Treino - Na abordagem para estabelecer uma comunicação ICC no qual este não dependa de quaisquer vias de saída neuromusculares, o utilizador tem que aprender a controlar os seus potenciais corticais, o que exige formação intensiva do lado do utilizador. Sistemas ICC baseados na deteção de potenciais relacionados com estímulos externos normalmente requerem menos treino por parte dos utilizadores, mas têm como desvantagem o facto de dependerem de estímulos externos o que faz com que o utilizador não seja completamente independente no controlo do ICC. A redução do tempo de treino será portanto um dos objetivos na investigação dos próximos anos [17][18].

Redução do tempo de Calibração - Mesmo que seja possível reduzir o tempo de treino do utilizador através do uso de um ICC baseada em estímulos externos, continua a existir uma etapa demorada de preparação onde implica a calibração do sistema para as características cerebrais do utilizador. Têm sido feitas algumas tentativas para evitar esta etapa de calibração através do uso de técnicas com algoritmos de aprendizagem e inteligência artificial, especificamente desenvolvidos para este fim, o que faz prever que no futuro seja possível utilizar ICC's sem ter que passar por este procedimento [18][19].

2.5.6 Trabalhos Relacionados

Hoje em dia existem alguns projetos onde se estuda o uso de ICC's para controlo de próteses, de braços robóticos, de carros ou até mesmo coisas tão simples como comandar o cursor do computador. Houve então a necessidade de fazer um levantamento de trabalhos de investigação realizada e semelhantes a alguns aspetos deste trabalho, de onde foram selecionados os seguintes trabalhos, considerando os exemplos seguintes os mais relevantes:

- "EEG control of an electric wheelchair for disabled persons": Este projeto descreve um método de controlo inteligente para controlar uma cadeira de rodas elétrica por pessoas com deficiência, com recurso ao Emotiv EPOC [20].

***- "Cognitive Efficiency in Robot Control by* Emotiv EPOC":** Este projeto consiste num protótipo de um pequeno veículo com rodas onde é implementado um controlador de estados mentais com recurso ao Emotiv EPOC. O estudo observou diferentes pessoas com idades entre 14 a 30 anos onde se registou 72,65% de precisão, enquanto as pessoas portadoras de deficiência foi de 82%, em média. O trabalho desenvolvido investigou as falhas sobre a fiabilidade cognitiva do dispositivo, com base nas observações dos utilizadores e pesquisas relacionadas [9].

- "Electroencephelograph Based Brain Machine Interface for Controlling a Robotic Arm": Neste projeto utiliza-se um Emotiv EPOC para enviar sinais de controlo para um braço robótico com 7 graus de liberdade. Esta pesquisa pretendeu ainda incorporar e aperfeiçoar os processos de alguns algoritmos de classificação e deteção através de um aumento da percentagem da carga de treino [21].

2.6

2.7 Redes de Petri

Rede de Petri (*RdP*) trata-se de um formalismo base de modelação de sistemas, amplamente utilizadas em diferentes áreas de aplicação [22][23]. Define graficamente a estrutura de sistemas distribuídos através de representações ma-

temáticas, sendo utilizado em modelos de sistemas, dispondo de técnicas de análise robustas, e possuindo representação gráfica, bem como matemática [24].

Como já foi referido, o objetivo deste trabalho pretende contribuir para os ambientes de desenvolvimento de sistemas, podendo ser encarada como um complemento à ferramenta IOPT-Tools, mais sucintamente, contribuindo para, por exemplo, modelar sistemas de automação, fazendo uso deste tipo de formalismo, modelos estes representados em *PNML (*Petri *Nets Markup Language)* [22]. Neste documento pretende-se então aplicar as características de uma classe de redes de Petri direcionada para a modelação de sistemas embutidos, denominada de ***Input-Output*** **Place Transition** (IOPT).

2.7.1 Redes de Petri IOPT (Input-Output Place-Transition)

A classe IOPT emprega os padrões base das redes de Petri, lugares, transições e arcos. Além disso, as IOPT's também permitem a definição de sinais de entrada, sinais de saída, eventos de entrada, eventos de saída e *arrays* contendo tabelas de dados, entre outras definições [25][23].

Os **sinais de entrada** são utilizados para obter informações do mundo exterior, por exemplo, para ler o estado de sensores, botões da interface do utilizador, ou mesmo ler sinais de outros sistemas [25].

Os **sinais de saída** podem ser utilizados para manipular atuadores mecânicos, iluminar *LED's,* para criar interfaces de utilizador ou enviar informações para outros sistemas [25].

Os **eventos** são normalmente associados com mudanças nos valores de sinal. Eventos de entrada são acionados por mudanças nos sinais de entrada e Eventos de saída causam mudanças nos sinais de saída. Um controlador IOPT pode esperar por eventos de entrada específicos e reagir em conformidade, alterando o valor dos sinais de saída. Os sistemas complexos podem ainda ser implementados utilizando vários subsistemas IOPT que comunicam por meio de sinais e eventos [25].

Um **arco** é representado por uma linha com uma seta que define a direção do arco. Um arco que começa num lugar e termina numa transição é chamado

de arco de entrada. Um arco que parte de uma transição e termina num lugar é chamado de arco de saída [25].

A **transição** só pode disparar quando todos os lugares ligados aos arcos de entrada têm um número de marcas maiores ou iguais à inscrição do arco correspondente. Quando essa condição for satisfeita, a transição está habilitada. Além disso, as transições também podem ser associadas aos eventos de entrada e às condições de guarda. Quando todas as condições de guarda são verdadeiras e os eventos de entrada acontecerem, a transição está apta a disparar. Uma transição só será disparada quando é simultaneamente apta e habilitada [25].

Os **lugares** são utilizados para armazenar as marcas. Um lugar com uma ou mais marcas é considerado marcado [25].

A evolução de um modelo de rede de Petri é definida pelo comportamento das transições. Quando acontecem certas condições, uma transição pode disparar, remover marcas de determinados lugares e adicionar novas marcas para outros lugares, de acordo com os arcos conectados a essa transição [25]. A Figura 12 ilustra o aspeto de uma rede de Petri.

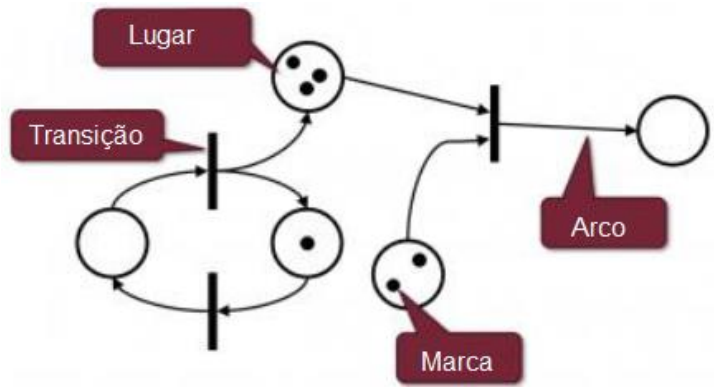


Figura 12: Exemplo de uma Rede de Petri [26].

Esta classe de redes de Petri é o elo comum num conjunto de ferramentas de desenvolvimento chamado **IOPT-Tools**, que incluem um editor gráfico, um analisador de espaço de estados para a verificação das propriedades, resolução de conflitos, geradores de código automático, simuladores, entre outros. Neste sentido, o objetivo principal proposto de usar a classe IOPT das redes de Petri e as ferramentas associadas é apoiar todo o fluxo de desenvolvimento do projeto, desde a especificação até à execução do sistema [22].

Contrariamente a outras classes de redes de Petri, as IOPT-Tools foram projetados especificamente com o objetivo de geração automática de código para a implementação de controladores de sistemas embutidos e outros sistemas digitais. Como consequência, um modelo IOPT é executado em etapas, com um sistema de *clock's*[11] preciso (ciclo de *clock* para implementações de hardware) [25].

2.8 Navegação de Quadricópteros

O **Quadricóptero** (*quandcopter*[12]) é um veículo aéreo não tripulado (*UAV*[13]), também chamado de ***drone***. O controlo de um quadricóptero é um problema bastante interessante devido à sua complexidade. É composto por seis graus de liberdade (três eixos de rotação e três eixos de translação) e apenas quatro entradas (velocidade angular de cada hélice). Além disso, a dinâmica sobre este tipo de aparelhos deve oferecer flexibilidade e robustez nos movimentos, como por exemplo, os algoritmos de controlo podem ser implementados de modo a manter a sua estabilidade mesmo que alguma das hélices que controle um eixo de rotação perca a sua funcionalidade. Por outro lado, sendo um veículo aéreo o algoritmo de controlo tem que considerar o amortecimento [27].

A orientação do quadricóptero é dada pelo *Roll (Φ), Pitch (θ)* e *Yaw (ψ)*, ângulos definidos como rotações com o respetivo eixo X, eixo Y e eixo Z, ilustrados na Figura 13 e Figura 14 [27].

[11] *Sinal utilizado para coordenar ações em circuitos eletronicos.*
[12] *Quadricoptero em inglês.*
[13] *Unmanned aerial vehicle.*

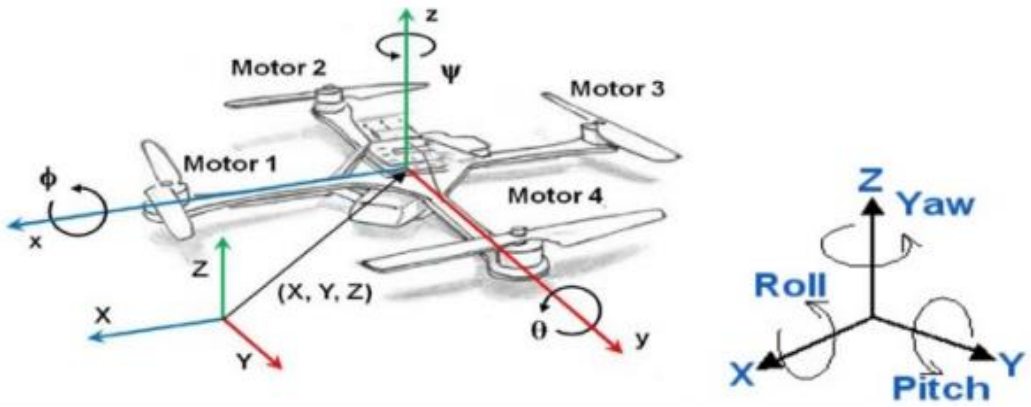


Figura 13: Sistema de coordenadas com os ângulos *Roll, Pitch* e *Yaw* [27].

Figura 14: Sistemas de eixos e movimentos possíveis do "*Ar.Drone*" [28].

No mercado existem todo o tipo de preços de quadricópteros, dependendo do desempenho dos microcontroladores. A sua aplicabilidade vai desde produtos profissionais, militares a fins lúdicos [27].

3

Ambientes de Desenvolvimento e Execução

"Ao longo dos próximos 30 anos, vamos cada vez mais integrar a tecnologia aos nossos corpos para fins recreativos e informativos. Um adolescente em 2044 vai se espantar o quanto nosso corpo está livre de tecnologia em 2014."

Ravin Agrawal, diretor da Exploratorium

Este capítulo referência o capacete Emotiv EPOC utilizado na aquisição e processamento de dados, o quadricóptero *Ar.Drone* como teste de validação do projeto, e ainda as ferramentas de modelação e programação de desenvolvimento de software empregado na nossa abordagem ao problema.

3.1 Emotiv EPOC

O capacete da Emotiv EPOC é um produto de alta resolução, com a capacidade de aquisição e processamento de neuro-sinais, que usa um conjunto de sensores para registar sinais elétricos produzidos pelo cérebro e detetar padrões de ondas cerebrais sob a forma de pensamentos, sentimentos e expressões, bem como a direção da cabeça no eixo vertical e horizontal [29].

Figura 15: Capacete EPOC da EMOTIV [30].

Trata-se de um dispositivo móvel e fácil de usar. O segredo presente neste dispositivo, representado na Figura 15, está no uso de elétrodos onde se aplicam técnicas de eletroencefalografia, eletromiografia e eletro-oculografia, captando e processando ondas cerebrais *Gamma*, *Delta, Theta, Alpha* e *Beta*. Os sinais são gravados e transmitidos para um sistema através de uma tecnologia sem fios *Bluetooth*. O dispositivo dispõe de uma bateria que assegura o uso durante 12 horas de operação contínua. Além disso, o dispositivo Emotiv EPOC tem sensores giroscópios que detetam a orientação espacial do dispositivo [29]-[31].

3.1.1 Descrição do Hardware

É utilizada uma solução salina nos elétrodos para reduzir a impedância entre eles e o couro cabeludo. A sua concentração de sais na água afeta diretamente a sua condutividade elétrica: quanto mais iões dissolvidos, maior a condutividade elétrica (sob temperatura constante) [31]. Na Figura 16 estão representados o capacete, a solução salina e os elétrodos utilizados.

Figura 16: Capacete e acessórios [31]: 1 – Emotiv EPOC, 2 – Solução Salina, 3 – Elétrodos.

A vantagem de usar vários elétrodos possibilita aceder a mais do que uma localização do cérebro, o que permite um maior conjunto de funções [17].

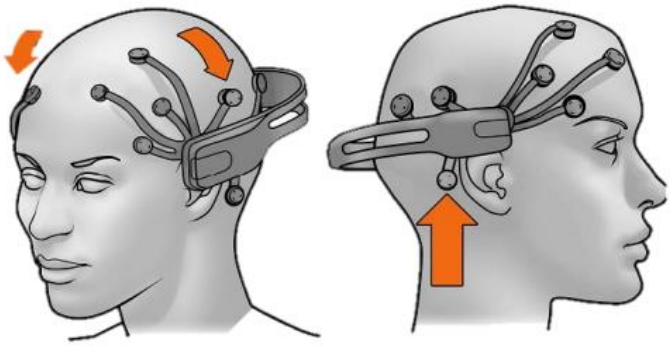


Figura 17: Posição correta do capacete [32].

O Emotiv EPOC requer ser colocado numa posição particular, como exibe a Figura 17. Composto por 14 elétrodos com uma frequência de amostragem de 128 *Hz* situados nas posições *AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8 e AF4* do sistema internacional *10-20* descrita na Figura 18, adquire as ondas cerebrais diretamente a partir do contacto com o couro cabeludo do utilizador. Posteriormente processa as ondas cerebrais e transmite-as sem fios para um computador ou dispositivo [17]. No anexo D estão disponíveis as características do capacete Emotiv EPOC do fabricante.

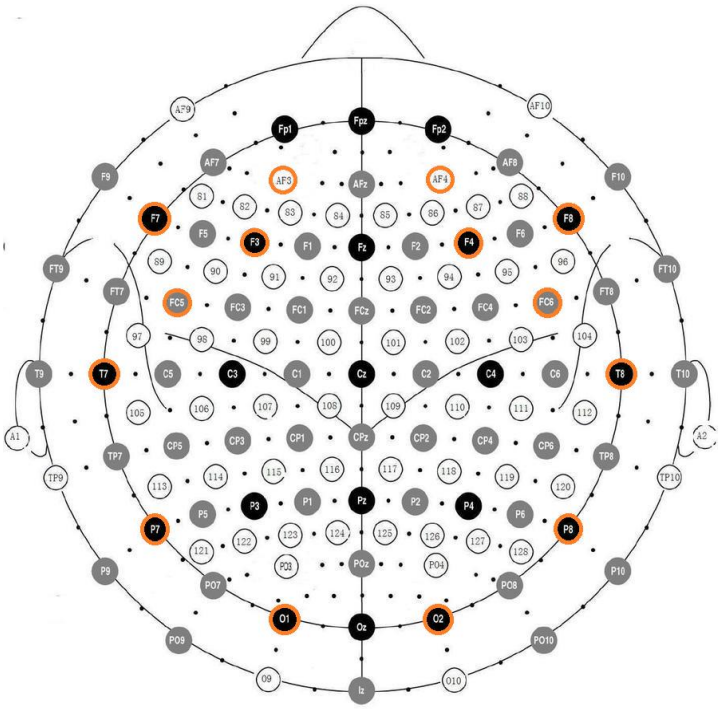


Figura 18: Posições do sistema internacional 10-20 [17].

3.1.2 Descrição de Softwares

No momento da aquisição deste aparelho foi também adquirido através do seu fabricante um *SDK* com um conjunto de vários softwares, ambos bastante completos para os programadores poderem tirar partido das funcionalidades do Emotiv EPOC.

3.1.2.1 Aplicações

O "Emotiv *Control Panel*", apresentado na Figura 19, é uma ferramenta que torna possível ver graficamente o estado de ligação do Emotiv EPOC ao computador, incluindo se todos os elétrodos estão a funcionar de forma correta [17].

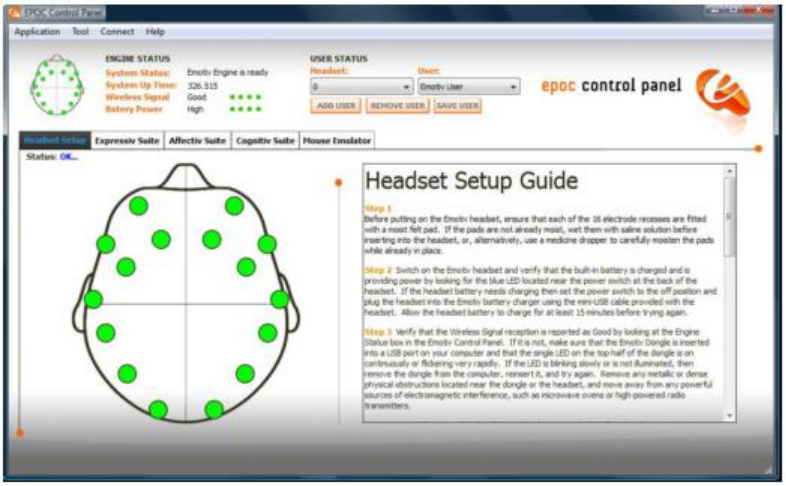


Figura 19: Menu principal do Emotiv *Control Panel*.

Este contém submenus onde mostra atividade dos sensores giroscópicos, os sinais Emotivos, um avatar onde podemos testar as próprias expressões faciais e ainda gravar estados mentais em forma de pensamentos lógicos, ao ponto de fazermos mexer um cubo em *3D*, como demonstram as Figura 20, Figura 21 e Figura 22.

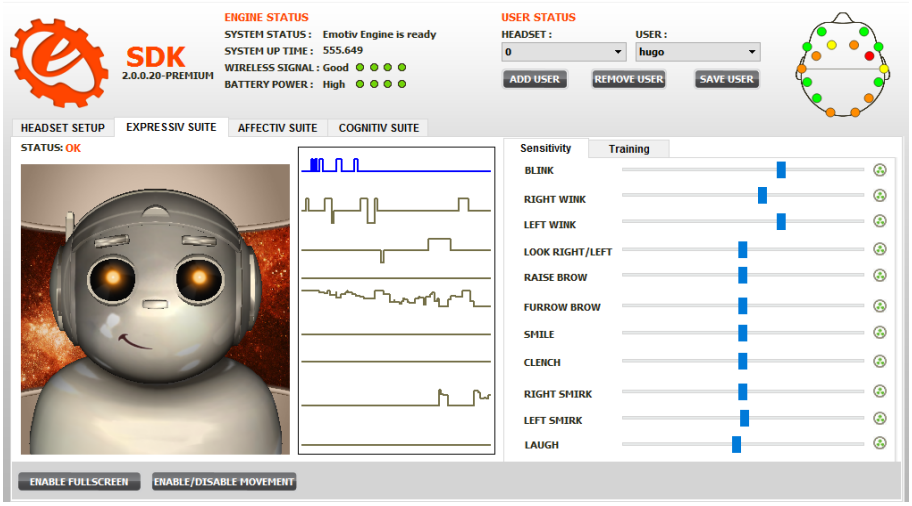


Figura 20: Leitura dos sinais expressivos no Emotiv *Control Panel*.

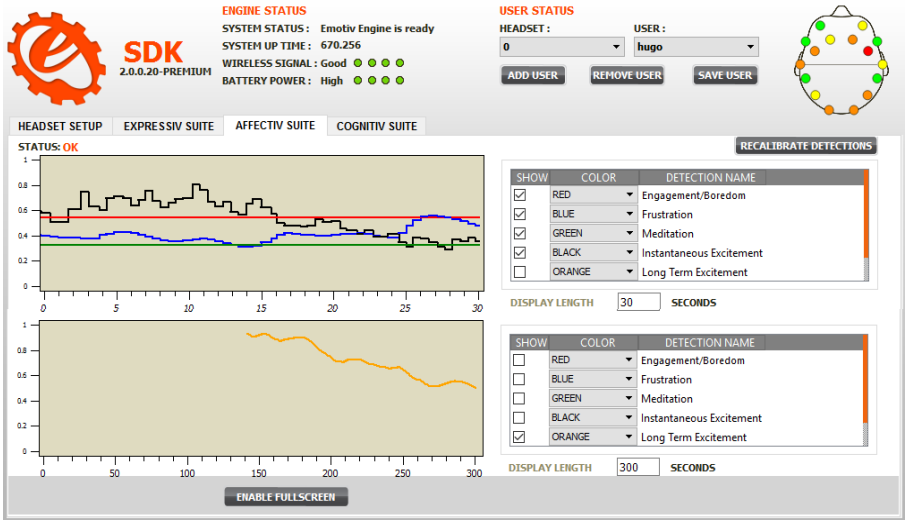


Figura 21: Leitura dos sinais Emotivos no Emotiv *Control Panel*.

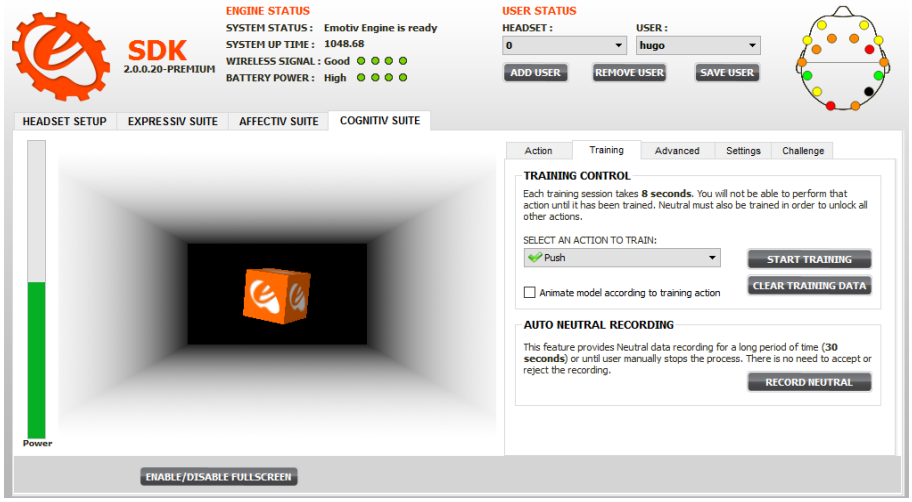


Figura 22: Leitura dos sinais cognitivos no Emotiv *Control Panel*.

Outro programa explorado no decorrer dos trabalhos foi o Emotiv *TestBench*, um programa muito detalhado e com o grafismo bastante lúdico, permitindo analisar cinco tipos de gama de onda em cada elétrodo e gerar os dados descaracterizados para um ficheiro, representado na Figura 23, bem como gerar o sinal dos giroscópios representado na Figura 24.

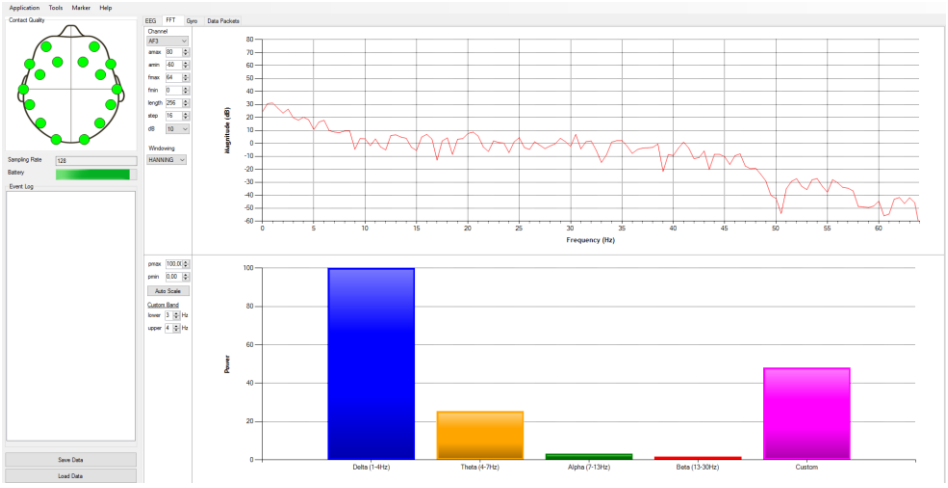


Figura 23: Leitura dos sinais *EEG* descaracterizada no Emotiv *TestBench*.

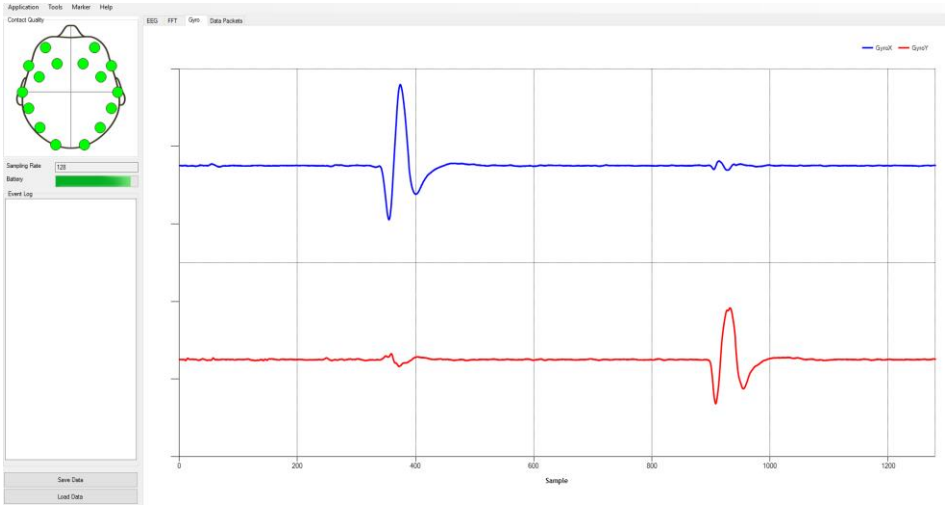


Figura 24: Leitura dos sinais Giroscópicos descaracterizados no Emotiv *TestBench*.

Por fim, ainda foram testados programas como o "*Mind Your OSC's*", "*BrainVisualiser*", "Emotiv EPOC *Brain Activity Map*", "*EmoComposer*" e o "*EmoKey*", todos bastantes interessantes e didáticos.

3.1.2.2 *Kit* de desenvolvimento

Após a receção e processamento dos dados obtidos a partir do Emotiv EPOC é criada uma estrutura temporária, na qual são guardadas todas as leituras obtidas, como a Figura 25 representa. A biblioteca disponibilizada no final do documento oferece uma grande variedade de funções que permitem o controlo e processamento dos dados recolhidos, facilitando assim o desenvolvimento de aplicações que tirem partido de todas as funcionalidades que o Emotiv EPOC oferece [17].

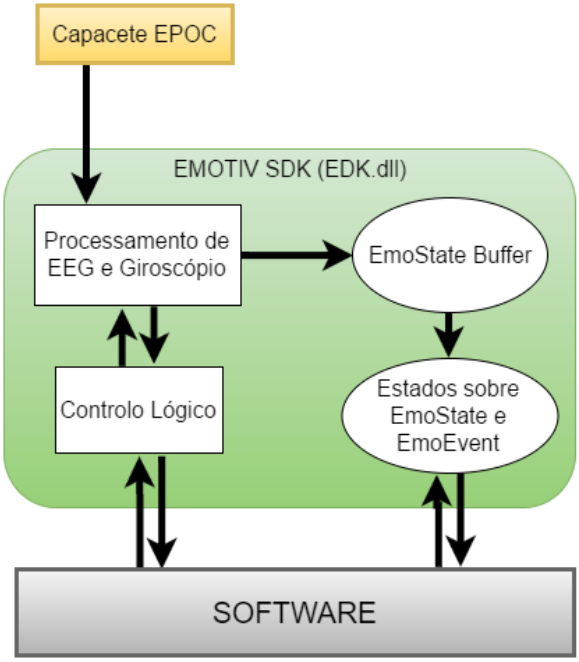


Figura 25: Diagrama da integração do *SDK* e do software da Emotiv com uma aplicação.

3.1.2.2.1 Sinais Expressivos

Utiliza os elétrodos do capacete para coletar dados *EMG* de músculos faciais, detetando os movimentos laterais dos olhos através do sinal *EOG*, e a intensidade é calculada utilizando um algoritmo de localização para detetar as combinações musculares, permitindo ao utilizador mapear expressões faciais [29][33]. O capacete tem a capacidade de detetar este tipo de sinais em menos de 10 milissegundos.

3.1.2.2.2 Sinais Emocionais

Está relacionado com o estado emocional do utilizador, nomeadamente com o entusiasmo, tédio, atenção e meditação. Usa sinais cerebrais fortemente filtrados para remover os componentes musculares presentes. Todas as deteções são feitas com base na distribuição de frequências cerebrais características e a atividade de grupos de neurónios correlacionados com toda a superfície do cérebro. Estas deteções também usam um algoritmo adaptativo para ajustar a escala de modo a atender a gama de base e nível do utilizador de cada emoção [29]. Os níveis de entusiasmo são obtidos tanto instantaneamente como ao longo do tempo, sendo possível avaliar o estado do utilizador em tempo real ou num determinado intervalo de tempo [17].

3.1.2.2.3 Sinais Cognitivos

Permite a associação de determinados padrões cerebrais através de comandos do *SDK* [17]. Admite alguns componentes musculares na assinatura, a fim de permitir que os utilizadores inexperientes possam obter resultados razoáveis (é possível que existam alguns sinais cerebrais que sofram de atenuações devido às tensões musculares, mas não expulsa a sua existência)[29]. Quando os dados *EEG* são lidos, os movimentos musculares fornecem alguns dados extra e são contados como ruído que geralmente é filtrado [9][21]. O EPOC permite o uso de quatro estados cognitivos em simultâneo, apesar de ser extremamente difícil ao utilizador conseguir manter o controlo de tal situação [17]. Portanto, ao contrário das expressões, não existe nenhuma assinatura universal que vá funcionar em indivíduos diferentes simultaneamente. É necessário criar uma assinatura para cada utilizador individual, chamando as funções de treino adequadas, implicando manipulações apropriadas aos eventos. Os novos utilizadores exigem, geralmente, muita prática, a fim de evocar de forma confiável o sinal desejado, e alternar entre os estados mentais, consequência do treino de cada sinal cognitivo. Como tal, é imperativo que um utilizador se torne mestre com uma única ação antes de tentar ativar duas ações simultâneas, e assim por diante. Durante todo o processo de formação, é importante manter a qualidade do sinal de *EEG* e a consistência da aparência mental associada à ação a ser treinada.

Os utilizadores devem abster-se de movimentos e devem relaxar a sua cara, a fim de limitar outras fontes potenciais de interferência com o seu sinal *EEG*, devido aos sinais provenientes de *EMG* e *EOG*.

3.1.2.2.4 Sensor giroscópico

O giroscópio tem dois eixos, X e Y, através do qual é imitida a intensidade do sinal ao utilizador. Como apresentado na Figura 24, o eixo X relata os movimentos horizontais da cabeça, sendo o valor negativo indicador do movimento esquerdo da cabeça e o valor positivo o movimento direito da cabeça. Por outro lado, o eixo Y identifica movimentos verticais da cabeça, um valor positivo corresponde a um movimento para cima da cabeça e um valor negativo indica um movimento para baixo da cabeça [34].

3.2 Quadricóptero

Nesta dissertação é utilizada para teste de validação do projeto um quadricóptero da empresa *Parrot SA*, cujo modelo é o *Ar.Drone* versão *2.0*, capaz de um bom controlo de precisão e recursos automáticos de estabilização. Durante a elaboração deste projeto recorreu-se ao guia disponibilizado pela *Parrot* [35].

3.2.1 Aspetos Técnicos

Este quadricóptero tem um processador *ARM cortex A8*, com *1 Gb de RAM*, tecnologia *Wi-Fi*, *USB* incorporado e funciona com o sistema operativo *LINUX*. O seu sistema de controlo é composto por uma arquitetura de hardware embutido e tem integrado acelerómetros que correspondem a três eixos direcionais, giroscópios que correspondem a dois eixos de rotação, e ainda um eixo que recorre a um giroscópio de precisão. Utiliza um altímetro com sensibilidade até aos 6 metros e um sensor de pressão. Ainda é portador de duas camaras, uma de estabilização de voo, e uma outra para fins lúdicos *HD 720p*. Ao nível mecânico é composto por quatro motores *brushless inrunner* com 14.5*W* de potência a 28.500 *RPM*[14]. Tem um sistema de queda de emergência por software e controlador eletrónico de motor. Este quadricóptero é capaz de atingir os 18 *Km/h*, pesa menos que 0.45 quilogramas e tem um tempo de voo estimado de 12 minutos [28].

3.2.2 Descrição do Software

Este quadricóptero tem a vantagem de ter sido construído de modo que fosse possível qualquer um consultar, examinar ou modificar o seu código, código aberto, onde se torna possível programar uma aplicação para comunicar com o servidor alojado no *drone*. Numa primeira fase foram utilizados alguns softwares para conseguir testar o voo, nomeadamente o "*AR.Freeflight*" [28], *o* "*FreeFlight*" [36], o "*QGroundControl*" [37] e o "*AutoFlight*". Foi analisada a capacidade de voo do *drone* de modo a prever alguma situação mais delicada. De seguida foi desenvolvida uma versão *beta* disponível na comunidade, o "*ARDrone*".

[14] *Rotações por minuto.*

Na Figura 26 e Figura 27 estão representadas as interfaces dos softwares *AR.FreeFlight* e *FreeFlight3.* São aplicações primárias gratuitas utilizadas para voar e pilotar o *AR.Drone* para *smartphones/tablets*. Compatível com *AR.Drone* e *AR.Drone* 2.0 [28][36].

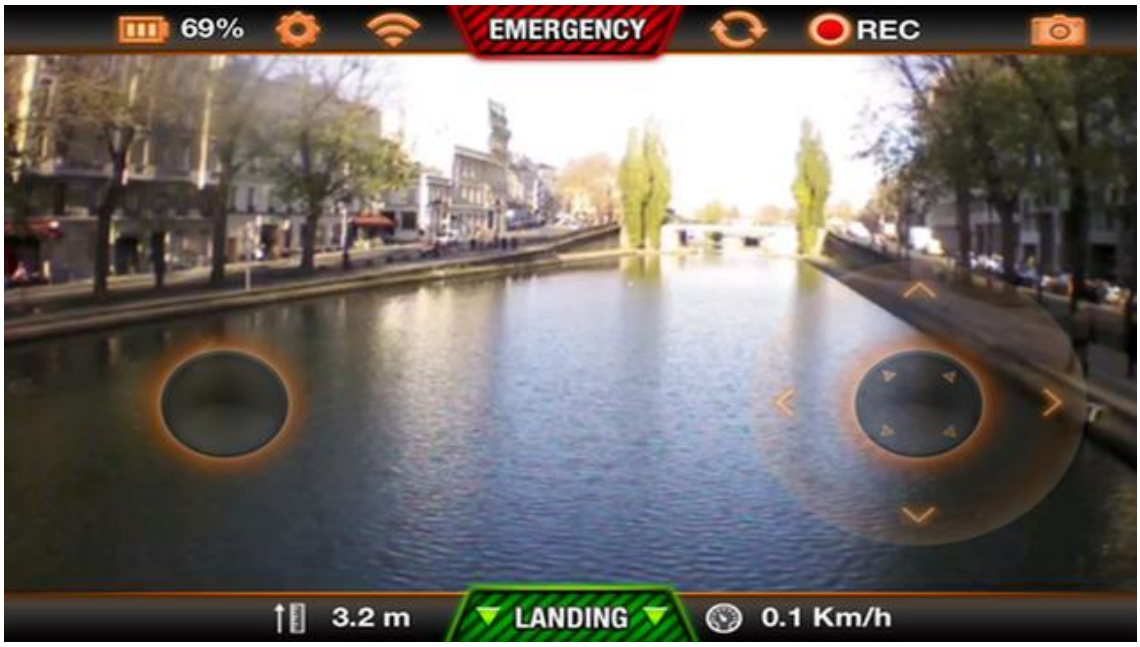


Figura 26: Interface da aplicação *AR.Freeflight* via *smartphone* para o utilizador.

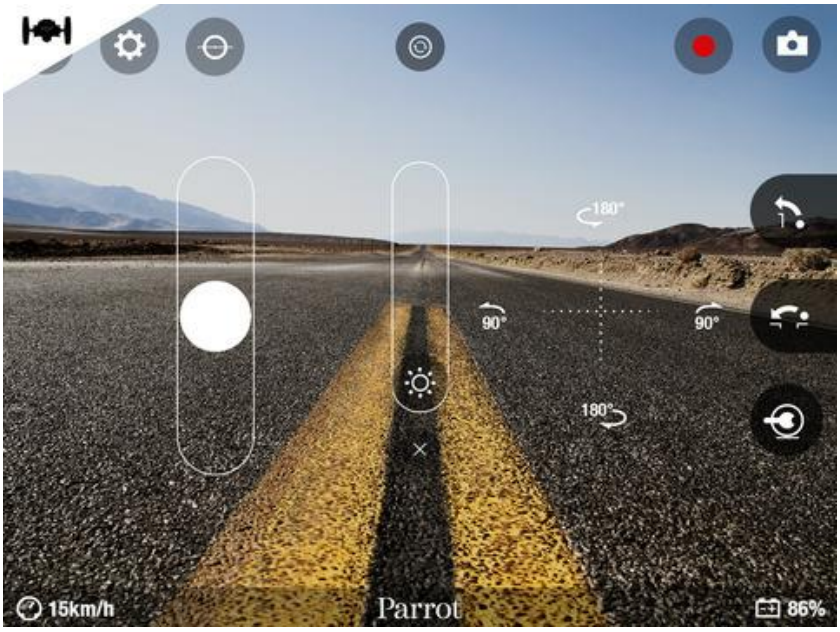


Figura 27: Interface da aplicação *FreeFlight3* via *tablet* para o utilizador.

3.3 Ferramentas de desenvolvimento de Software

Toda atividade do projeto foi desenvolvida com recurso às ferramentas IOPT-Tools e *MS Visual Studio*, descritas a seguir.

3.3.1 IOPT-TOOLS

O ambiente de desenvolvimento das **IOPT-Tools** foi o escolhido nesta dissertação porque oferece uma cadeia completa de ferramentas para apoiar o desenvolvimento e teste de sistemas de controladores, aplicações de automação industrial e outros sistemas digitais, disponível *online* sob uma *GUI*[15] na *Web* [25][38][39][40].

Os sistemas são especificados usando modelos IOPT e a sua estrutura consiste num editor, um simulador para testar o comportamento do sistema, um gerador de espaço de estados, um sistema de consulta para automatizar a verificação de propriedades e, finalmente, uma ferramenta de geração automática de vários tipos de linguagens (*C, VHDL, JavaScript*), utilizado para gerar código a implementar em dispositivos físicos ou implementar controladores embutidos [25].

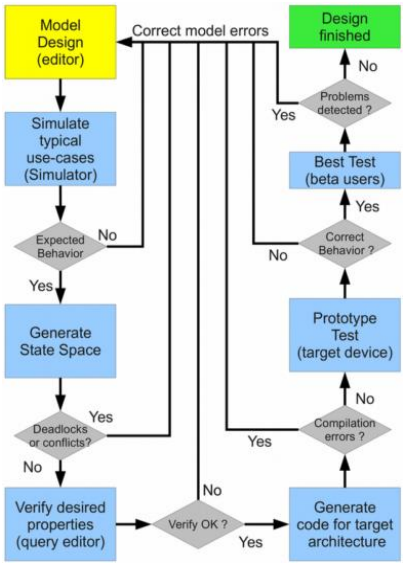


Figura 28: Fluxo do funcionamento típico da IOPT-Tools [25].

[15] Graphical User Interface.

Como é apresentado na Figura 28, esta ferramenta segue um fluxo de desenvolvimento rigoroso. As ferramentas de simulação e verificação de modelos permitem a deteção precoce de erros de projeto, permitindo a correção da maioria dos erros antes de atingir a fase de implementação do protótipo, levando a ciclos de desenvolvimento mais rápidos. As ferramentas de geração automática de códigos eliminam a necessidade de escrever manualmente o código de baixo nível, com exceção de algumas operações específicas dos dispositivos, minimizando erros durante a fase de desenvolvimento. A ferramenta representada na Figura 29 e Figura 30 pode ser usada gratuitamente e está disponível *online* em *http://gres.uninova.pt* [25] [38].

Figura 29: Página inicial da *GRES*[16].

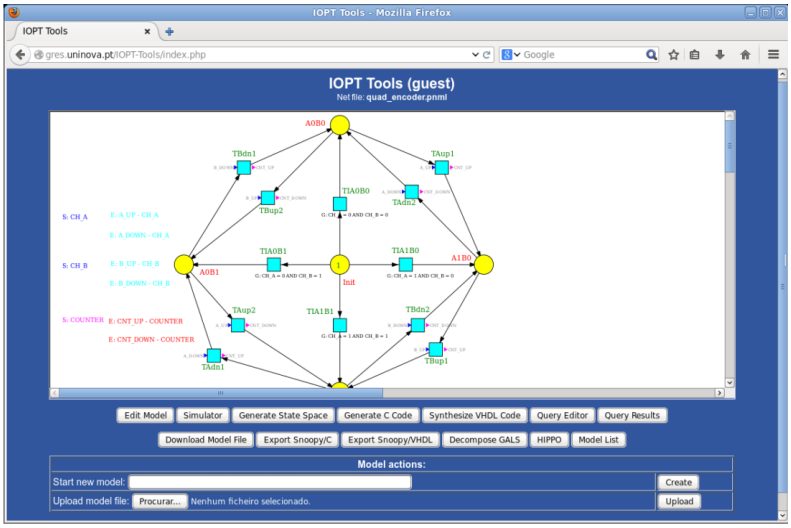


Figura 30: Menu principal das IOPT-Tools.

[16] *R&D Group on Reconfigurable and Embedded Systems*

3.3.1.1 Geração automática de código C

As IOPT-Tools contêm três ferramentas de geração automática de código (*C, VHDL* e *JavaScript)* para implementação de controladores de execução do comportamento de um modelo IOPT [25]. A ferramenta de geração automática de código *C* produz um arquivo compactado contendo os arquivos de código-fonte referidos na Figura 31 [41].

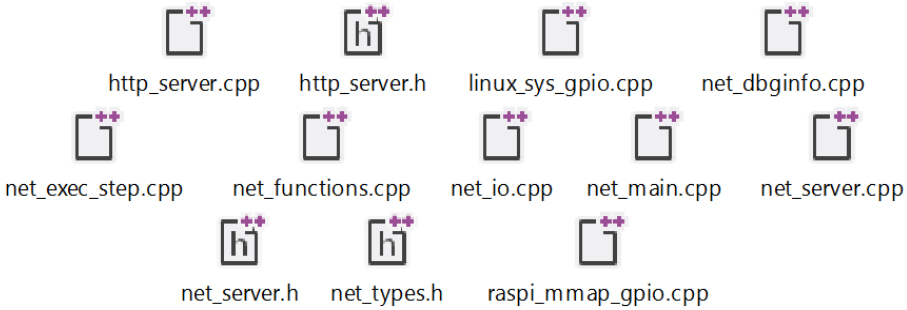


Figura 31: Arquivos de código fonte gerado das IOPT-Tools

3.3.1.2 Depurador

A ferramenta das IOPT-Tools permite controlar remotamente a execução do modelo IOPT em tempo de execução. Devido às contribuições deste trabalho, também passou a ser possível fazê-lo a partir de máquinas Windows, com recurso a uma comunicação *socket* e endereçado a um porto específico. Isto torna possível, no momento da execução do nosso programa, comunicar com as IOPT-Tools e fazer o utilizador poder ter uma informação gráfica do estado de marcação do modelo IOPT e dos eventos em tempo real, como apresentado na **Erro! A origem da referência não foi encontrada.** e na **Erro! A origem da referência não foi encontrada.**.

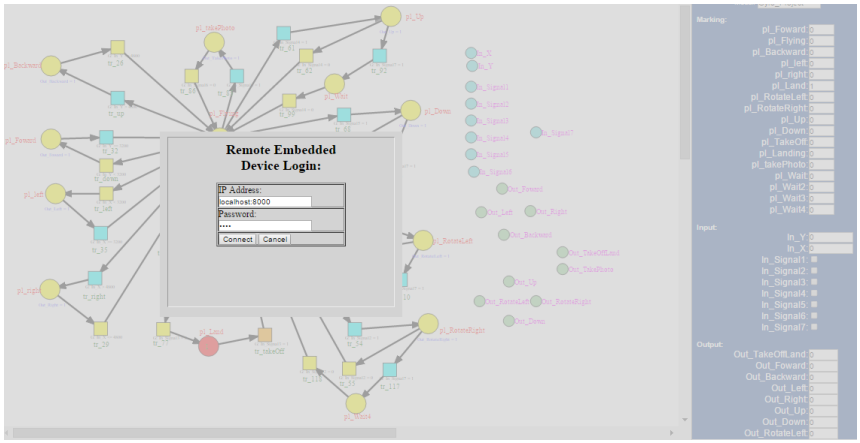


Figura 32: *Login* do depurador das IOPT-Tools.

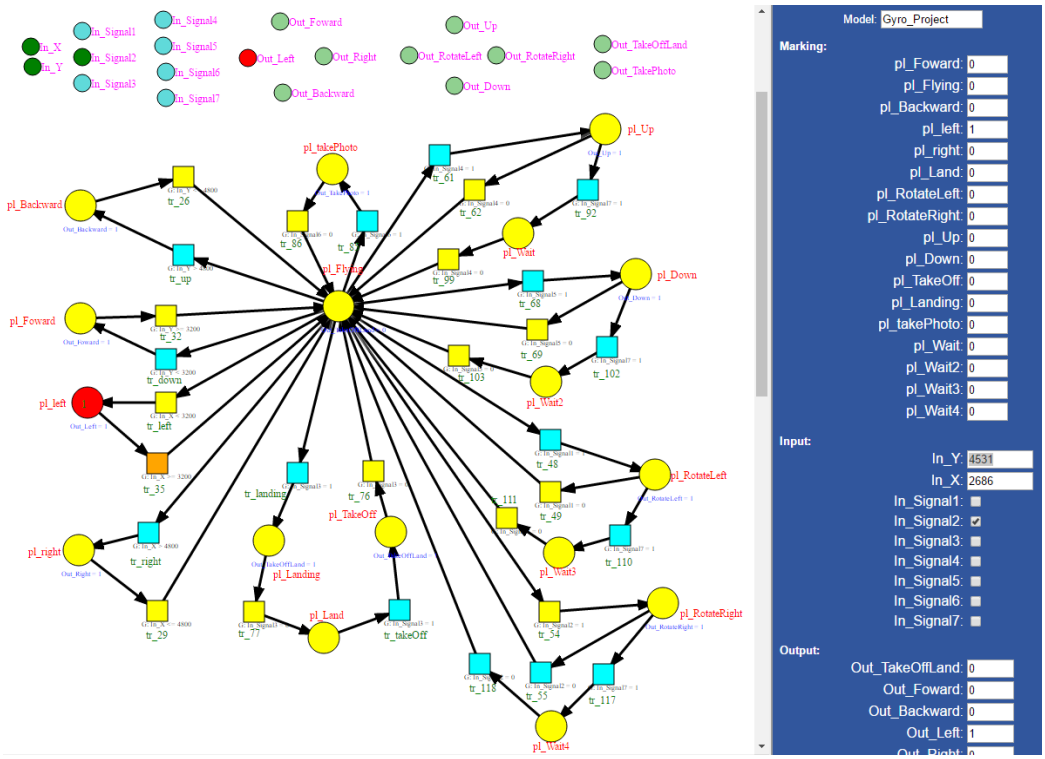


Figura 33: Processo visual do depurador das IOPT-Tools em funcionamento.

3.3.2 Microsoft Visual Studio

Trata-se de um ambiente de desenvolvimento integrado *(IDE[17])* dedicada à programação *DOTNET*, linguagens *Visual Basic*, *C*, *C++*, *C# (CSharp)* e *J#*

[17] *Integrated Development Environment*

(JSharp), possuindo diversas edições no mercado. Permite que se crie com muita rapidez aplicações que são passíveis de proporcionar experiências de utilizador com a mais alta qualidade e riqueza [42]. Os conceitos de linguagem seguintes são lembrados devido à integração total deles nesta dissertação.

Linguagem C - Trata-se de uma linguagem de propósito geral, sendo adequada à programação estruturada. As suas características são: portabilidade, modularidade, compilação separada, recursos de baixo nível, geração de código eficiente, confiabilidade, regularidade, simplicidade e facilidade de uso [43].

Linguagem C++ - É uma linguagem de programação genérica que suporta os paradigmas da programação estruturada. Grande parte de código C pode ser perfeitamente compilado em C++, existindo algumas pequenas diferenças sintáticas e semânticas entre as linguagens, ou códigos que exibem comportamentos diferentes em cada linguagem [44].

Linguagem C# - Trata-se de uma linguagem orientada a objetos, que neste trabalho será utilizado para criar uma aplicação no sentido cliente-servidor fornecendo um editor de código avançado, interfaces de utilizador convenientes e muitas outras ferramentas para facilitar o desenvolvimento da aplicação [42].

4

Aquisição e Processamento de Sinais

"A integração de nossos mundos físicos e virtual trará experiências mais ricas e conectividade com a população global."

Phil Wiser, diretor de tecnologia da Hearst.

No presente capítulo vai ser descrito a primeira parte do trabalho proposto e executado nesta dissertação. Em primeiro lugar, foi feita uma análise da estrutura hierárquica necessária na criação da biblioteca, devido às prioridades de funcionamento do capacete. De seguida são apresentadas as ações de controlo criadas, possíveis de serem usadas em programação e desenvolvimento de software. Para finalizar, pretende-se mostrar onde e como integrar esta biblioteca com modelos expressos através de redes de Petri, com o recurso à ferramenta IOPT-Tools.

4.1 Descrição da solução proposta

Foi proposto desenvolver uma biblioteca capaz de ser integrada em qualquer projeto de desenvolvimento de software, ou ainda diretamente na ferramenta IOPT-Tools, e onde houvesse interesse no uso de expressões faciais, emoções, pensamentos e orientação da cabeça, tudo sob a forma de sinais de

entrada. Deste modo e usando o capacete EPOC foi estudada e desenvolvida uma biblioteca que apresenta uma hipótese de uso para qualquer tipo de aplicações.

4.1.1 Estrutura Hierárquica

A solução está resumidamente descrita na Figura 34 passando por criar uma biblioteca, denominada por "EPOC*ontrol*" que faz uso do algoritmo utilizado pela Emotiv, que recolhe dados captados pelos elétrodos e os processa de maneira a descrever a ocorrência, de modo a ativar sinais de entrada em aplicações do utilizador, criando assim uma interação cognitiva.

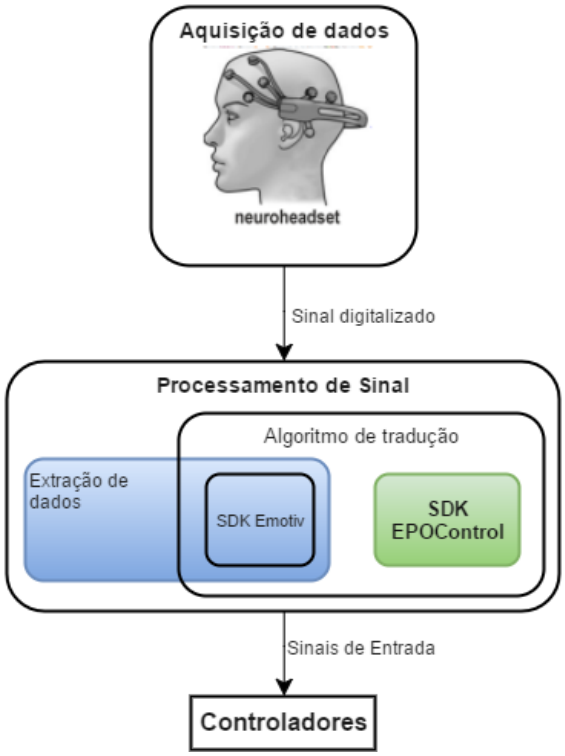


Figura 34: Fluxo de processamento ICC, ilustrando a aquisição de sinal e fases de processamento, com ênfase na solução *SDK* "EPOC*ontrol*".

4.1.2 Funções e variáveis de Controlo

Estão representados na Figura 35 as funções e variáveis de controlo desenvolvidas na biblioteca, e que permitem ao utilizador interagir num sistema de controlo caso sejam evocados e definidos como sinais de entrada do sistema. Representados a verde estão os casos recorrentes à orientação do capacete EPOC, em amarelo estão representados os casos que definem as expressões fa-

ciais, a roxo estão os casos dos sinais cognitivos que para além de variarem de utilizador para utilizador estes necessitam de um treino prévio. Nos casos a cinzento, trata-se de ferramentas de ação e controlo dos estados do capacete EPOC.

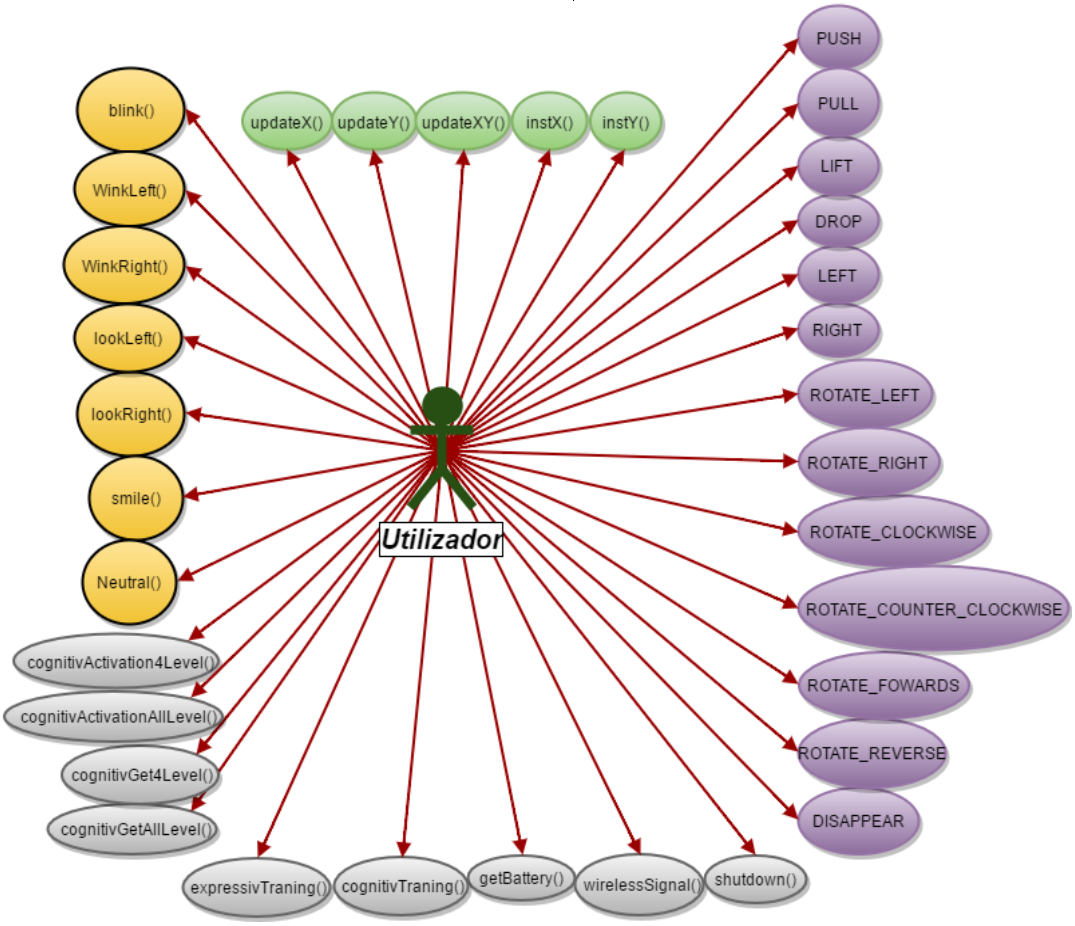


Figura 35: Funções e variáveis de controlo aptas a serem chamadas a controlar um sistema.

Para melhor se poder compreender as funções e varáveis descritas, as Tabela 4, Tabela 5, Tabela 6 e Tabela 7 do capítulo 4.2.2 descrevem a utilidade de cada uma.

4.1.3 Ficheiros de controlo

Para controlar a chamada de funções do algoritmo Emotiv e o fluxo das ações de controlo utilizados recursivamente foram criados os ficheiros mencionados na Figura 36.

Figura 36: Coleção de ficheiros dedicados ao controlo e entrada de sistemas.

4.1.4 Ficheiros de Comunicação

Para permitir que a aplicação desenvolvida comunique com outra solução, como por exemplo uma aplicação que controle um *drone*, foi criada uma classe correspondente, como apresenta a Figura 37, de modo a ser possível enviar o estado de cada variável do sistema.

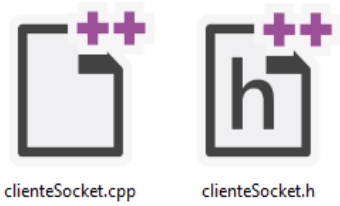


Figura 37: Coleção de ficheiros dedicados à comunicação intraprocessos.

4.1.5 Algoritmo EMOTIV

Esta secção apresenta conceitos-chave na utilização da Emotiv *SDK* para desenvolvimento de software que seja compatível com o capacete Emotiv. Trata-se de uma classe que permite interações entre aplicações externas e o mecanismo de deteção da Emotiv.

Para utilizar o *SDK* do Emotiv EPOC foi necessário fazer a importação para a biblioteca dos sete ficheiros da Figura 38 que contêm as classes que descodificam a informação obtida pelo Emotiv EPOC e que fornecem de forma bastante simplificada a informação desejada [17].

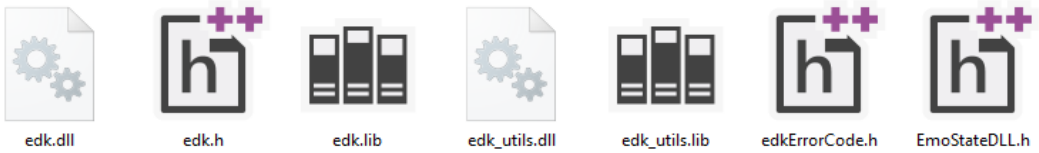


Figura 38: Coleção de ficheiros *SDK* Emotiv EPOC.

No seguimento da utilização desta biblioteca recorremos a um guia de utilização fornecido pela Emotiv que enumera as classes e funções, sendo a comu-

nicação EPOC com a aplicação bidirecional. Na maior parte das funções iremos encontrar um prefixo "*ES_*" que representa estado, senão "*EE_*" que representa evento [45].

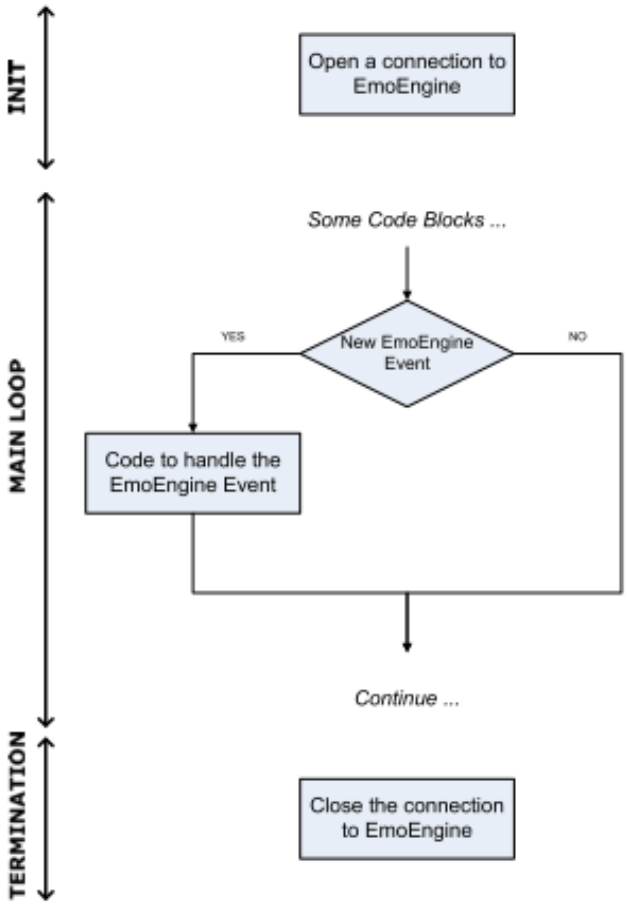


Figura 39: Comportamento ideal de uma aplicação que utiliza o EPOC [32].

A Figura 39 apresenta um fluxograma de alto nível para aplicações que usam o EPOC. As primeiras instruções a serem executadas correspondem à inicialização de três variáveis, representadas no Código 1, uma do tipo evento, outra que representa um *buffer* de estados e uma que representa o perfil de utilizador *profile*, e será com essas que será feita a comunicação geral com o capacete.

```
EmoEngineEventHandle hEvent = EE_EmoEngineEventCreate();
EmoStateHandle eState = EE_EmoStateCreate();
EmoEngineEventHandle eProfile = EE_ProfileEventCreate();
```

Código 1: Inicialização das variáveis de estado, evento e *profile*.

4.2 Ações de Controlo

Devido à complexidade da biblioteca apenas irão ser descritas neste documento as linhas gerais essenciais do que foi definido, nomeadamente um conjunto de funções a serem chamadas a representar eventos, estados do sistema e a comunicação apropriada.

4.2.1 Implementação de eventos

Estas operações foram criadas com o objetivo de comunicar com o EPOC e executar instruções, permitindo um fluxo de funcionamento correto.

Durante a **inicialização**, deverá ser estabelecida uma conexão com o EPOC, através da função *initEpoc()* descrita no Anexo 0, onde garante a conexão ao capacete, verificação do estado dessa comunicação, o registo de utilizador através de funções como *loadUser()* ou *cognitivResetActions()*,descritos no Anexo 0, e um intervalo de tempo necessário para consequentemente se poder executar um ciclo capaz de processar qualquer evento ou mudança de estado que surjam.

Para o **período de atividade** foram pré-estabelecidas funções estritamente necessárias ao seu correto funcionamento. Por um lado é indispensável estar em constante comunicação com o EPOC, através da função *newAction()* descrita no Anexo 0, responsável por interpretar os eventos do capacete, onde escuta as respostas recebidas. Pretende-se em cada rotina identificar os estados correntes do capacete, pertencentes ao conjunto observado na Tabela 3, procurando mediante cada estado executar os eventos definidos.

EmoEngine events	Hex Value	Description
EE_UserAdded	0x0010	New user is registered with the EmoEngine
EE_UserRemoved	0x0020	User is removed from the EmoEngine's user list
EE_EmoStateUpdated	0x0040	New detection is available
EE_ProfileEvent	0x0080	Notification from EmoEngine in response to a request to acquire profile of an user
EE_CognitivEvent	0x0100	Event related to Cognitiv detection suite. Use the EE_CognitivGetEventType function to retrieve the Cognitiv-specific event type.
EE_ExpressivEvent	0x0200	Event related to the Expressiv detection suite. Use the EE_ExpressivGetEventType function to retrieve the Expressiv-specific event type.
EE_InternalStateChanged	0x0400	Not generated for most applications. Used by Emotiv Control Panel to inform UI that a remotely connected application has modified the state of the embedded EmoEngine through the API.
EE_EmulatorError	0x0001	EmoEngine internal error.

Tabela 3: Tipos de Evento do EPOC[32].

Por outro lado existe a função *interrupt()*descrita no Anexo 0, que permite ao utilizador recorrer instantaneamente ao conjunto limitado de operações criadas que regulam o sistema, como parar o programa, aceder aos níveis de bateria do capacete e do sinal *wireless* de comunicação. Este conjunto conta com funções de natureza modelar sobre os sinais (funções *expressivActivationLevel()*, *cognitivActivationAllLevel()* e *cognitivActivation4Level()*),descrita no Anexo 0, onde permitem ao utilizador conhecer e modificar o valor de sensibilidade dos sinais expressivos e cognitivos. Neste conjunto ainda se encontram as funções *expressivTraining()* e *cognitivTraining()*, descrita no Anexo 0, absolutamente necessárias caso o indivíduo queira criar novos sinais, tendo atenção que por cada registo de treino ter-se-á de comunicar com o capacete várias vezes, seguindo adequadamente a estrutura modelo demonstrada na Figura 40 e Figura 41 descrevendo o processo a aplicar e as manipulações apropriadas aos eventos recebidos.

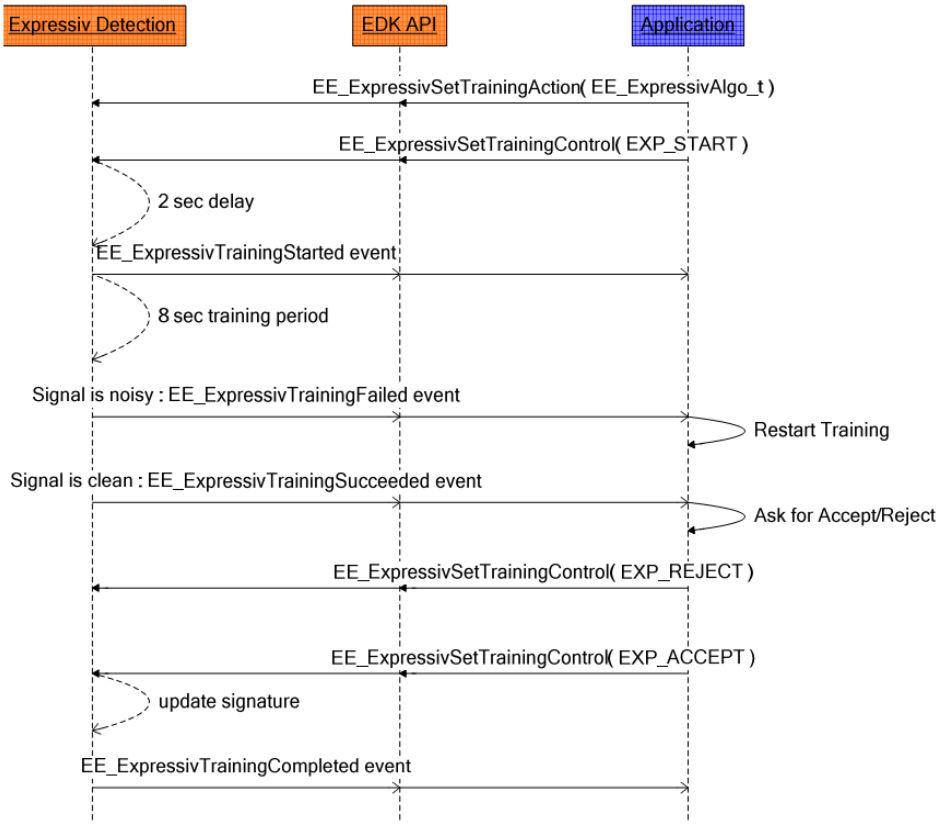


Figura 40: Protocolo de treino expressivo [32].

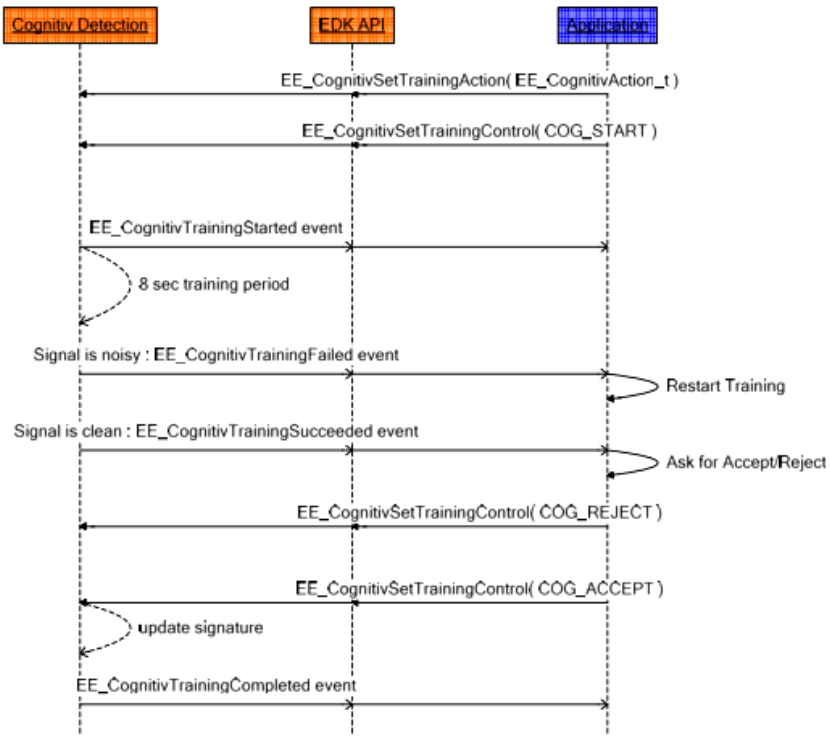
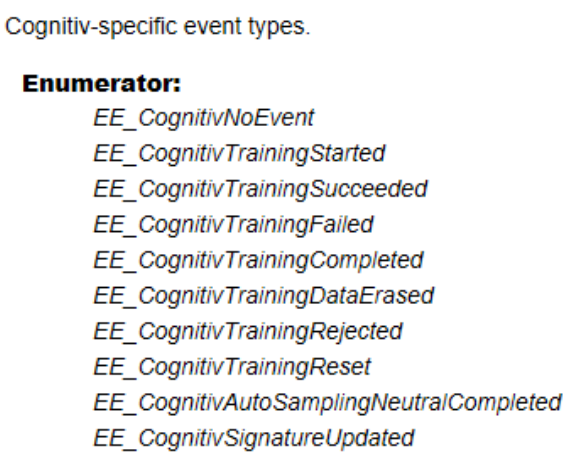


Figura 41: Protocolo de treino cognitivo [32].

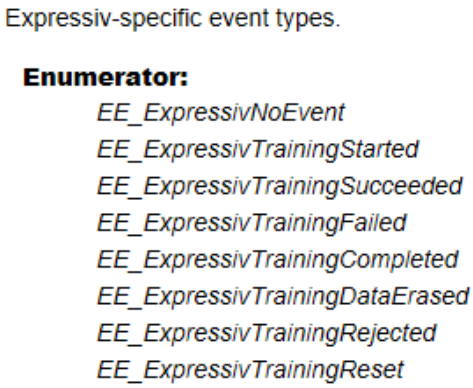
É necessário entender que existe um conjunto de eventos possíveis de ser recebidos pelo capacete, como apontam as Figura 42 e Figura 43.


Cognitiv-specific event types.

Enumerator:

EE_CognitivNoEvent
EE_CognitivTrainingStarted
EE_CognitivTrainingSucceeded
EE_CognitivTrainingFailed
EE_CognitivTrainingCompleted
EE_CognitivTrainingDataErased
EE_CognitivTrainingRejected
EE_CognitivTrainingReset
EE_CognitivAutoSamplingNeutralCompleted
EE_CognitivSignatureUpdated

Figura 42: Tipos de evento no treino cognitivo.


Expressiv-specific event types.

Enumerator:

EE_ExpressivNoEvent
EE_ExpressivTrainingStarted
EE_ExpressivTrainingSucceeded
EE_ExpressivTrainingFailed
EE_ExpressivTrainingCompleted
EE_ExpressivTrainingDataErased
EE_ExpressivTrainingRejected
EE_ExpressivTrainingReset

Figura 43: Tipos de evento no treino expressivo.

Em seguida, a Figura 44 apresenta uma sessão que demonstra uma experiência com o comando de treinos *cognitivTraining()* disponível numa consola, para entender melhor o processo de treino acima descrito.

```
CognitivDemo> set_actions 0 push lift
==> Setting Cognitiv active actions for user 0...

CognitivDemo>
Cognitiv signature for user 0 UPDATED!
CognitivDemo> training_action 0 push
==> Setting Cognitiv training action for user 0 to "push"...

CognitivDemo> training_start 0
==> Start Cognitiv training for user 0...

CognitivDemo>
Cognitiv training for user 0 STARTED!
CognitivDemo>
Cognitiv training for user 0 SUCCEEDED!
CognitivDemo> training_accept 0
==> Accepting Cognitiv training for user 0...

CognitivDemo>
Cognitiv training for user 0 COMPLETED!
CognitivDemo> training_action 0 neutral
==> Setting Cognitiv training action for user 0 to "neutral"...

CognitivDemo> training_start 0
==> Start Cognitiv training for user 0...

CognitivDemo>
Cognitiv training for user 0 STARTED!
CognitivDemo>
Cognitiv training for user 0 SUCCEEDED!
CognitivDemo> training_accept 0
==> Accepting Cognitiv training for user 0...

CognitivDemo>
Cognitiv training for user 0 COMPLETED!
CognitivDemo>
```

Figura 44: Treino "PUSH" e "LIFT" executado na aplicação.

Por fim, no momento em que o utilizador quer **terminar** o programa é necessário chamar a função *shutdown()* descrita no Anexo 0, onde é possível encontrar funções dedicadas a garantir que a comunicação com o EPOC é desligada convenientemente. Apesar de existir um grande conjunto de sinais previamente reservados a qualquer utilizador, e caso este crie novos sinais, é necessário recorrer à função *saveUser()* descrita no Anexo 0, onde é proporcionado um fluxo controlado de ordens de modo a ser possível reservar uma assinatura. Deste modo o utilizador pode utilizar a assinatura que contém os sinais gravados sempre que for desejável. No caso de haver uma comunicação *socket* com o software esta função também irá permitir fazer *closeSocket()* de modo a que a aplicação que esteja à espera de mensagens possa não ter que ficar bloqueada.

4.2.2 Implementação de estados

A implementação deste tipo de funções irá permitir ao sistema de controlo usar como sinal de entrada a informação gerada por este serviço.

Os estados lidos pelos sensores giroscópicos são recebidos pelas funções representadas na Tabela 4, contribuindo para uma maior interação em sistemas de controlo.

Tabela 4: Estados

Nome da Função	Retorna
UpdateY()	Devolve as coordenadas no eixo vertical (devolve valores negativos quando o capacete aponta para baixo, devolve valore positivos quando o capacete se encontra a apontar para cima).
UpdateX()	Devolve as coordenadas no eixo Horizontal (devolve valores negativos quando o capacete aponta para a esquerda, devolve valore positivos quando o capacete se encontra a apontar para a direita).
instY()	Devolve a potência do sinal instantânea de acordo com uma escala definida no sentido vertical (devolve valores negativos quando o capacete aponta para baixo, devolve valore positivos quando o capacete aponta para cima).
instX()	Devolve a potência do sinal instantânea de acordo com uma escala definida no sentido horizontal (devolve valores negativos quando o capacete aponta para a esquerda, devolve valore positivos quando o capacete aponta para a direita).

As funções representadas na Tabela 5 identificam o estado facial do utilizador, sendo que este tipo de estado será igual para qualquer utilizador, devido ao facto de serem observados maioritariamente através de *EOG* e *EMG*.

Tabela 5: Tradução dos estados recebidos da expressão facial do utilizador.

Nome da Função	Retorna
blink()	Verifica se os olhos estão a piscar (devolve um se estiverem a piscar, devolve zero se não estiverem)
winkleft()	Verifica se o olho esquerdo pisca (devolve um se verdadeiro, devolve zero se não piscar)
winkright()	Verifica se o olho direito pisca (devolve um se verdadeiro, devolve zero se não piscar)
lookLeft()	Verifica se os olhos apontam para a esquerda (devolve um se verdadeiro, devolve zero se não olharem)
lookRight()	Verifica se os olhos apontam para a direita (devolve um se verdadeiro, devolve zero se não olharem)
smile()	A operação retorna 1 se registar uma potência superior a 0.5 numa escala de zero a um, com intervalos de 0,1.

Como foi referido anteriormente, qualquer sinal cognitivo requer um processo de treino inicial (função *cognitivTraining()*), a fim de reconhecer quando é identificada uma atividade consciente por parte de um utilizador igual a uma das ações representadas na Tabela 6 suportadas pelo EPOC.

Tabela 6: Tradução do estado mental do utilizador.

Nome da Variável	Retorna
PUSH	Devolve 1 se for detetado o pensamento empurrar, caso contrário devolve 0.
PULL	Devolve 1 se for detetado o pensamento puxar, caso contrário devolve 0.
LIFT	Devolve 1 se for detetado o pensamento subir, caso contrário devolve 0.
DROP	Devolve 1 se for detetado o pensamento descer, caso contrário devolve 0.
LEFT	Devolve 1 se for detetado o pensamento esquerda, caso contrário devolve 0.
RIGHT	Devolve 1 se for detetado o pensamento direita, caso contrário devolve 0.
ROTATE_LEFT	Devolve 1 se for detetado o pensamento rodar à esquerda, caso contrário devolve 0.
ROTATE_RIGHT	Devolve 1 se for detetado o pensamento rodar à direita, caso contrário devolve 0.
ROTATE_CLOCKWISE	Devolve 1 se for detetado o pensamento rodar no sentidos ponteiros do relógio, caso contrário devolve 0.
ROTATE_COUNTER_CLOCKWISE	Devolve 1 se for detetado o pensamento rodar no sentido contrário ao dos ponteiros do relógio, caso contrário devolve 0.
ROTATE_FORWARDS	Devolve 1 se for detetado o pensamento rodar para frente, caso contrário devolve 0.
ROTATE_REVERSE	Devolve 1 se for detetado o pensamento rodar para trás, caso contrário devolve 0.
DISAPPEAR	Devolve 1 se for detetado o pensamento desaparecer, caso contrário devolve 0.

Para que seja possível o utilizador ter acesso ao estado do sistema durante uma operação foram criadas funções que o permitem observar o estado de algumas condições, bem como alterar certas instruções, representadas na Tabela 7.

Tabela 7: Configurações em informação do e/ou para o estado do sistema

Nome da função	Retorna
cognitivGet4Level()	Devolve a informação de cada um dos níveis de sensibilidade de todos os sinais cognitivos.
cognitivActivation4Level()	Permite afetar o nível de sensibilidade de cada um dos sinais cognitivos.
cognitivGetAllLevel()	Devolve a informação do nível de sensibilidade de todos os sinais cognitivos.
cognitivActivationAllLevel()	Permite afetar o nível de sensibilidade de todos os sinais cognitivos.
getBattery()	Devolve a informação do estado de bateria do capacete EPOC.
WirelessSignal()	Devolve a informação do sinal sem fios.
expressivTraining()	Permite fazer um registo de expressões faciais.
cognitivTraining()	Permite fazer até quatro registos de pensamentos cognitivos.
shutdown()	Permite desconectar o capacete, encerrando-o de forma correta ao mesmo tempo que grava toda a informação do utilizador.

4.2.3 Implementação Comunicativa

Foi decidido que seria de grande utilidade futura, e de grande utilidade para o projeto descrito no capítulo 5, incluir nesta biblioteca funções que se dedicassem à comunicação de sinais de saída do software com outro software, na mesma máquina, de maneira a ser possível poder testar numa aplicação. Na Tabela 8 temos as funções principais a evocar para esse efeito.

Tabela 8: Suporte de comunicação.

Tipo de comunicação	Nome da função
Sockets	initSocket()
	Send()

Desta feita, foi considerado ser necessário, no início de qualquer aplicação com este perfil, correr a função *initSocket()* com objetivo de criar um canal *socket* pronto a escutar mensagens através de um endereço na própria máquina, e recorrer à função *Send()* no final de cada ciclo de atividade, de maneira a enviar uma mensagem por esse mesmo canal, contendo como argumentos o nome do canal, a mensagem e o tamanho de mensagem.

4.3 Projeto de Desenvolvimento de Software

Para concluir esta fase tem-se agora um arquivo de ficheiros que permite suportar qualquer projeto que venha a precisar do capacete Emotiv EPOC, e para isso apenas ter-se-á que integrar esta biblioteca, com o aspeto que se vê na Figura 45, onde contém todos os ficheiros necessários ao seu funcionamento.

EPOControl.rar	02/03/2016 15:56	Arquivo WinRAR	2 075 KB
IncludeEpoc.rar	14/01/2016 12:05	Arquivo WinRAR	1 037 KB
clienteSocket.h	11/02/2016 16:46	C/C++ Header	2 KB
Controlador.h	07/03/2016 14:25	C/C++ Header	13 KB
edk.h	04/10/2015 19:36	C/C++ Header	38 KB
edkErrorCode.h	18/10/2012 08:51	C/C++ Header	3 KB
EmoStateDLL.h	17/05/2013 08:28	C/C++ Header	20 KB
clienteSocket.cpp	11/02/2016 22:30	C++ Source	3 KB
Controlador.cpp	07/03/2016 14:22	C++ Source	46 KB
Manual de utilizador.txt	16/03/2016 17:16	Documento de tex...	3 KB
edk.dll	31/07/2015 09:52	Extensão da aplica...	2 152 KB
edk_utils.dll	31/07/2015 09:52	Extensão da aplica...	1 938 KB
EDK_APIREF.chm	15/10/2013 10:32	Ficheiro de Ajuda ...	295 KB
edk.lib	15/10/2013 10:32	Object File Library	842 KB
edk_utils.lib	10/12/2012 06:48	Object File Library	3 KB

Figura 45: Conjunto de ficheiros necessários à construção de um projeto com o Emotiv EPOC.

O *SDK* contém dois ficheiros de formato *".dll"* (*Dynamic Linked Libraries*) com o nome *"edk.dll"* e *"edk_utils.dll"* que devem ser integradas na diretoria *debug* do projeto, onde se encontram os executáveis do projeto. Por fim é possível aceder a uma informação mais detalhada deste *SDK* no ficheiro "*EDK_APIREF.chm*", e ainda o "Manual de utilizador", ambos disponíveis na biblioteca "EPOC*ontrol*".

5

Implementação do Sistema de Controlo

"A theory can be proved by experiment; but no path leads from experiment to the birth of a theory."

Albert Einstein, theoretical physicist

Com o desenvolvimento de uma gramática de sinais, funções e regras propostos no capítulo 4 anterior, e com o motivo de proporcionar uma mais-valia à ferramenta IOPT-Tools, pretende-se demonstrar a razão que se prende na sua utilização e nas múltiplas aplicações possíveis, propondo como exemplo a criação de um projeto baseado num assistente de controlo cerebral para quadricópteros com base num paradigma *IHS*. Para isso foi desenvolvido um modelo de controlo do sistema nas IOPT-Tools e dois softwares, uma aplicação ICC de comunicação com o capacete e uma aplicação de navegação do quadricóptero. Estudou-se ainda uma arquitetura de mensagens de modo a ser possível as aplicações comunicarem entre si.

5.1 Descrição da solução proposta

O objetivo deste capítulo passa por utilizar o capacete EPOC, captando a orientação da cabeça, expressões faciais e os próprios sinais neuronais, proces-

sando-os de forma a condizer às instruções dos comandos utilizados para controlar um quadricóptero, como é ilustrado na Figura 46.

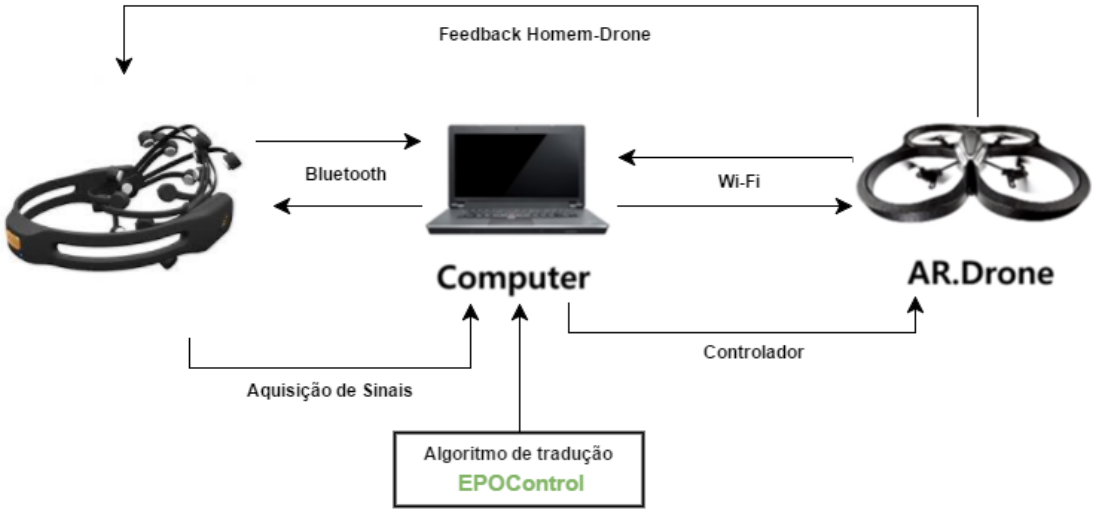


Figura 46: Diagrama de tecnologias do projeto, baseado em *IHS*, possível através da adição do algoritmo de tradução fornecido no capítulo anterior.

Numa primeira fase foi criado um programa que lida com a biblioteca desenvolvimento de software referida no capítulo 4 e com a biblioteca gerada pela ferramenta das IOPT's onde é modelado o sistema *IHS* de navegação de um quadricóptero.

De seguida foi criado um programa que controla o quadricóptero. Este programa tem o objetivo de procurar conectar-se via *Wi-Fi* com o servidor alojado no próprio quadricóptero e daí enviar os comandos de navegação, imprimidos pelo teclado e/ou pelos dados captados pelo capacete EPOC, retornando o estado comportamental do aparelho.

Finalmente foi concebido a ideia do projeto fazer estas duas aplicações funcionarem inteiramente no sentido cliente servidor na mesma máquina, a primeira comportando-se como cliente capaz de comunicar com o servidor que por sua vez permite comandar o quadricóptero.

Como já foi explicado os sinais neuronais são adquiridos utilizando a ferramenta desenvolvida no capítulo anterior. Após alguns ciclos de treino mental os sinais medidos são interpretados como padrões que estão associados com os sentidos. Uma vez que um padrão é definido este é relacionado com um co-

mando do controlador do sistema do quadricóptero que transforma as mensagens em atuação, como direção ou movimento.

Devido à complexidade do projeto foi necessário abordar o problema com recurso à Figura 47 onde podemos imaginar a estrutura do sistema de modo a podermos seguir os passos corretos e necessários.

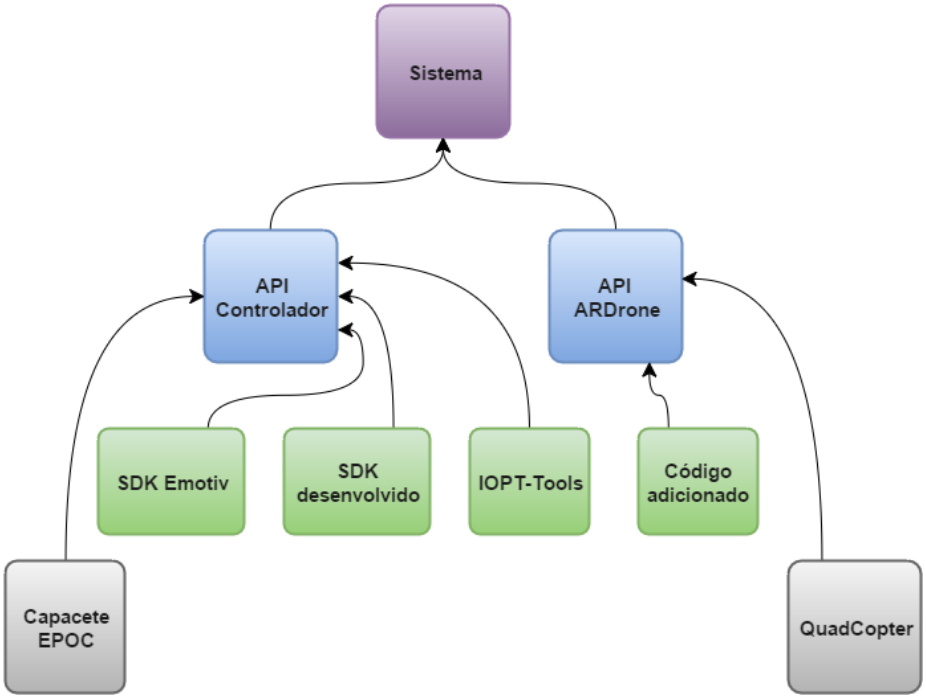


Figura 47: Estrutura de integração do sistema.

O sistema não é mais que a comunicação entre as duas aplicações (*API*[18]) representadas, com a finalidade de conseguir entregar no período mais curto possível uma mensagem de comando.

5.1.1 Proposta de controlador

No momento de integrarmos todas estas tecnologias é necessário entender como e quais serão os passos a tomar referentes ao projeto que irá permitir produzir a solução desejada. Na Figura 48 está representado um regime de implementação, sendo aqui consideradas classes e bibliotecas que resultam de cada tecnologia sendo assim integrados em projetos distintos, resultando na obrigação de uma comunicação unidirecional entre eles.

[18] *Application Programming Interface*

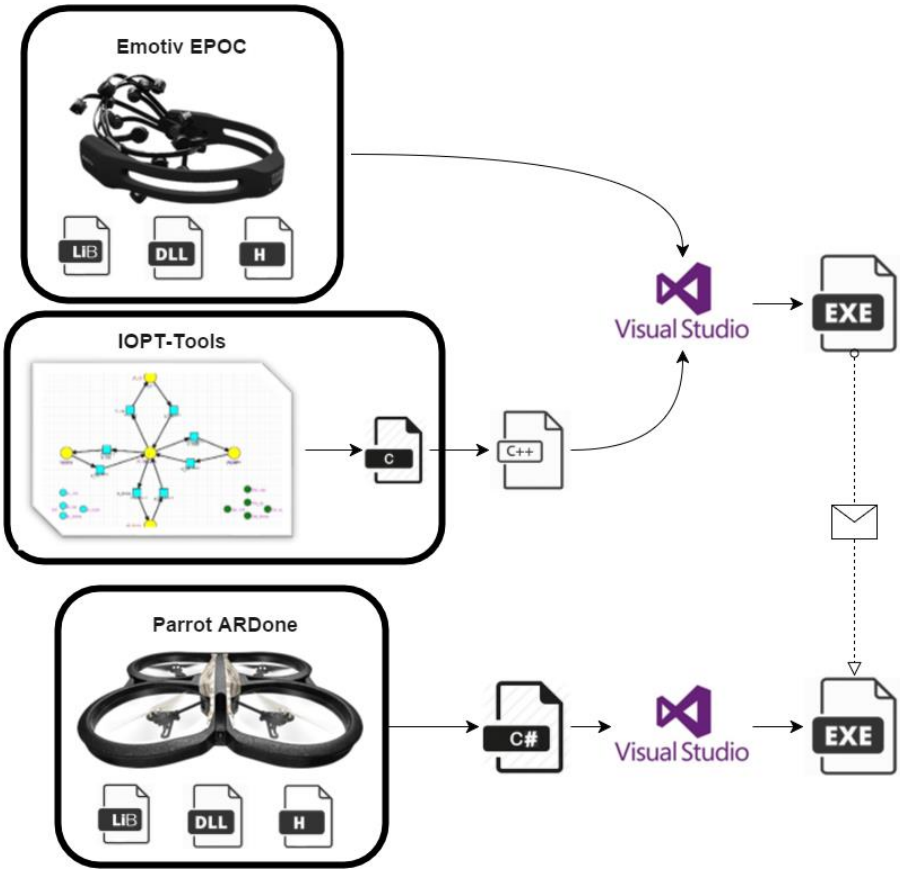


Figura 48: Fluxo de implementação.

5.1.2 Arquitetura de Navegação

De seguida são propostas as entradas e saídas do sistema de controlo. De acordo com os comandos de navegação do quadricóptero é feito então um relacionamento com os sinais de entrada que o capacete EPOC nos permite detetar. Na Tabela 9 é descrito a abordagem ao problema, contudo a solução a adotar poderia ser perfeitamente diferente, apenas os comandos de navegação são estritamente os definidos, enquanto os sinais de controlo de entrada poderiam ser outros que estejam indentados no *kit* desenvolvido no capítulo anterior.

Tabela 9: Comandos e funções de entrada e saída do sistema.

Sensores		**Atuadores**	
Evento	**Função**	**Ordem de voo**	**Interface**
Piscar do olho direito	*winkRight()*	**Levantar voo/Aterrar**	Tecla "SPACE"
Orientação da Cabeça na horizontal	*UpdateX()*	**Esquerda**	Tecla "A"
		Direita	Tecla "D"
Orientação da Cabeça na vertical	*UpdateY()*	**Avançar**	Tecla "W"
		Recuar	Tecla "S"
Olhar para a direita	*lookRight()*	**Rodar para a direita**	Tecla "L"
Olhar para esquerda	*lookLeft()*	**Rodar para a esquerda**	Tecla "J"
Pensamento de subida	*LIFT*	**Cima**	Tecla "W"
Pensamento de descida	*DROP*	**Baixo**	Tecla "S"
Piscar o olho esquerdo	*winkLeft()*	**Tirar fotografias**	*n.d.*
Piscar de olhos	*Blink()*	**Estabilização**	*n.d.*
n.d.[19]	*n.d.*	**Emergência**	Tecla "E"
n.d.	*n.d.*	**Calibração do quadricóptero**	Tecla "F"

O funcionamento prático de todas as funções descritas na Tabela 9 estão representadas na Figura 49, Figura 50 e na Figura 51.

[19] *Significa que não está disponível.*

Figura 49: Representação do modo de controlo *Take off* e *Landing* do quadricóptero.

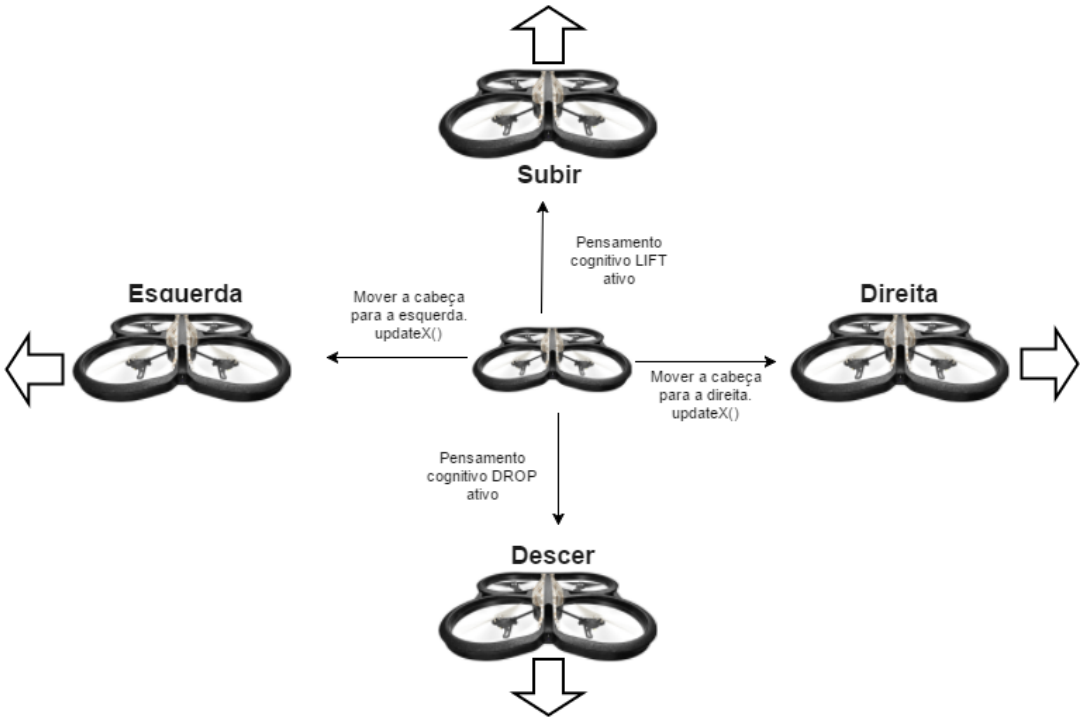


Figura 50: Representação do modo de controlo de subida, descida, movimentação para a esquerda e direita do quadricóptero.

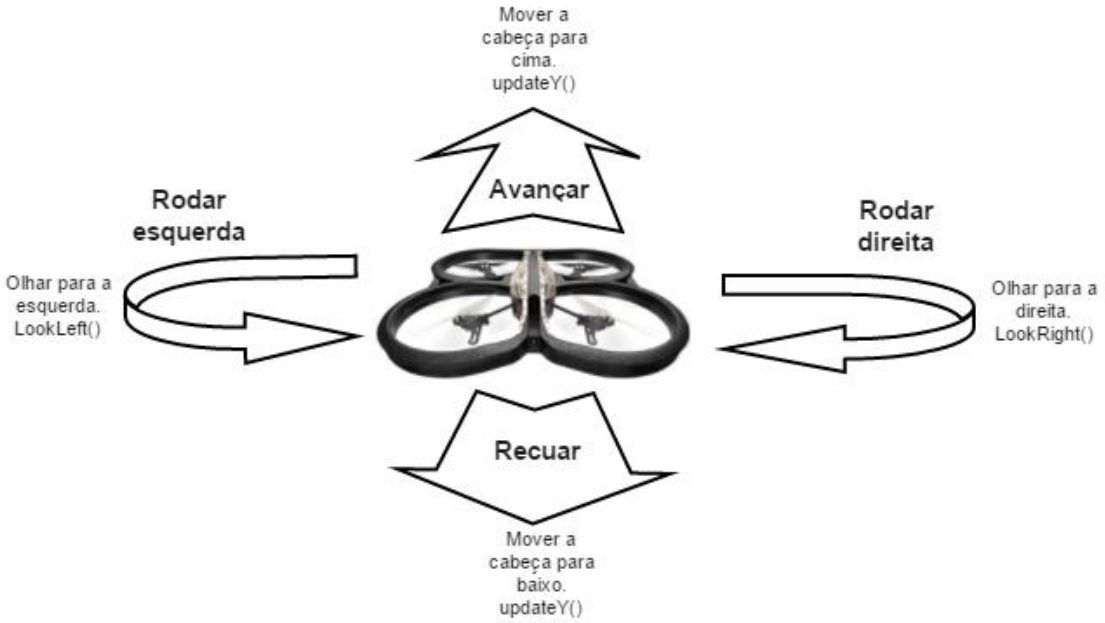


Figura 51: Representação do modo de controlo de rotação esquerda ou direita e avanço ou recuo do quadricóptero.

Sempre que o utilizador pensa num destes movimentos descritos anteriormente é lançado um evento que é recebido pelos vários controlos - menus, teclados e outros – que os faz movimentar para a direita, esquerda etc. No anexo I apresenta uma figura que representa o algoritmo do esquema de controlo base dos sinais do capacete com a representação da troca de mensagens.

5.2 *Modelação do sistema*

Seguindo a estratégia de desenvolvimento de sistemas especificados com redes de Petri, apresentada na Figura 52, são definidos os passos a seguir.

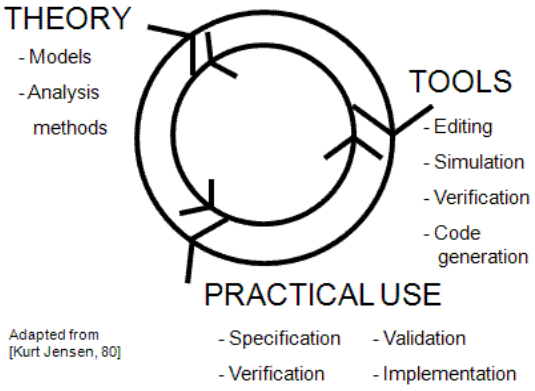


Figura 52: Metodologia adotada na construção da *RdP* [38].

5.2.1 Definição/Especificação do Sistema

A partir da solução proposta é analisado o problema sob a forma do esquema da Figura 53 aplicando o formalismo das redes de Petri. É importante analisar o sistema de controlo em cascata de modo a identificar os sinais de entrada e saída do sistema.

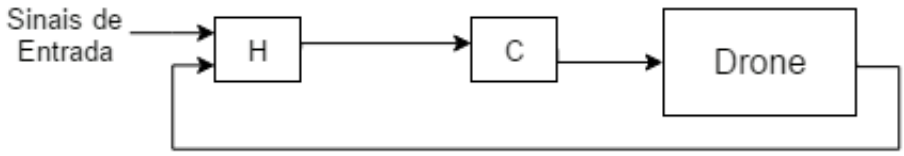


Figura 53: Esquema global do sistema.

5.2.2 Complexidade Modular

Primeiro editou-se o principal modelo do sistema pretendido, representado na Figura 54, através do editor de modelos da ferramenta das IOPT's, onde permite então definir os lugares pretendidos, as transições e arcos. Este modelo é criado com base na complexidade do sistema, partindo dos sinais de entrada e saída que foram previamente idealizados na Tabela 9. De seguida relacionou-se os lugares desenhados na *RdP* com os sinais de entrada (azul) e saída (verde), como apresenta a Tabela 10.

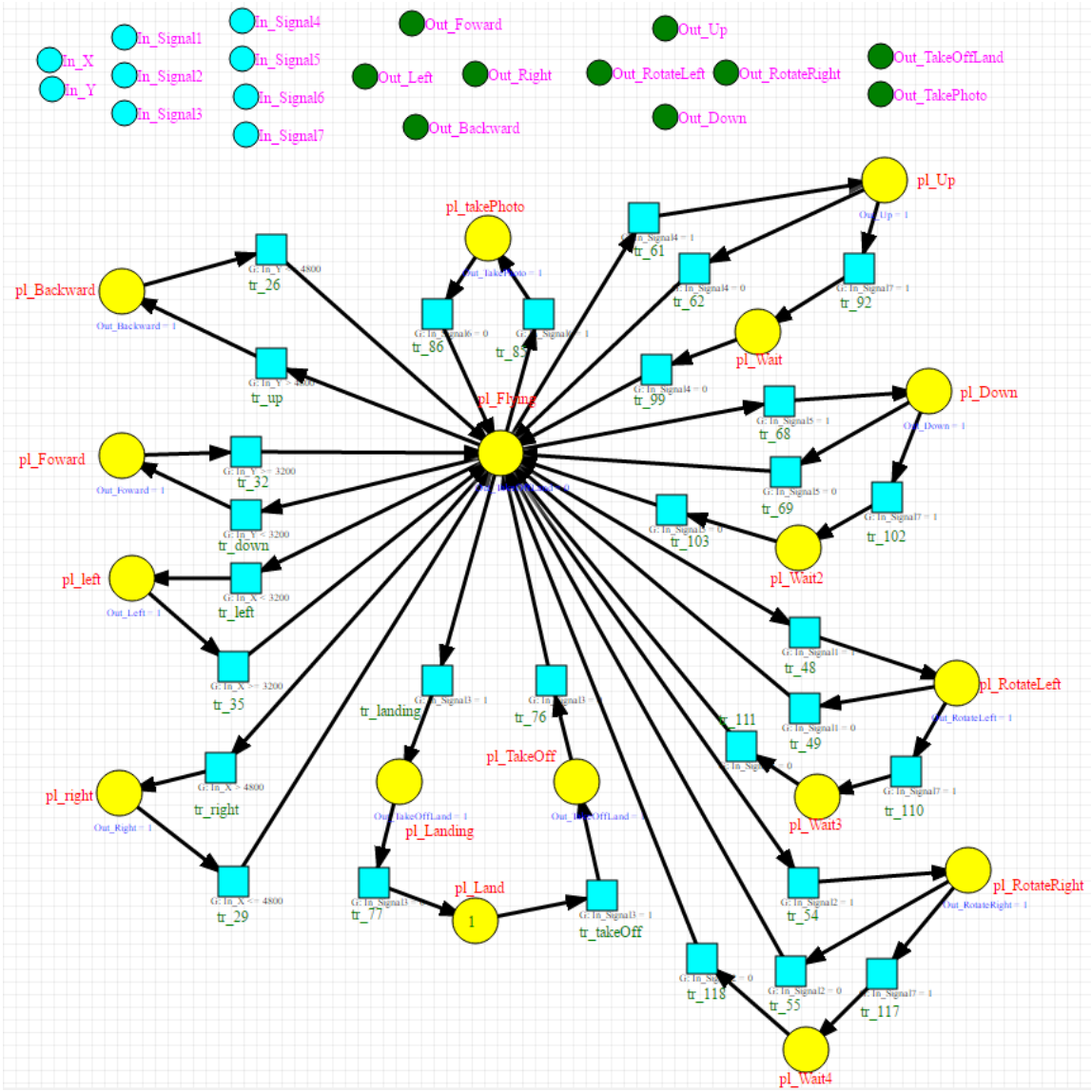


Figura 54: Modelo da Rede de Petri do projeto.

Tabela 10: Legenda das entradas e saídas da Rede de Petri.

Ação	Entradas	Saídas	Função
Direção Horizontal da cabeça	In_X	Out_left	Movimento para os Lados
		Out_Right	
Direção Vertical da cabeça	In_Y	Out_Forward	Movimento para frente e Trás
		Out_Backward	
Direção dos Olhos para a Esquerda	In_Signal1	Out_RotateLeft	Rodar para a esquerda
Direção dos Olhos para a Direita	In_Signal2	Out_RotateRight	Rodar para a direita
Piscar Olho Direito	In_Signal3	Out_TakeOffLand	Levantar ou aterrar
Pensamento de Subida	In_Signal4	Out_up	Subir
Pensamento de descida	In_Signal5	Out_Down	Descer
Piscar Olho Esquerdo	In_Signal6	Out_TakePhoto	Tirar Foto
Piscar dos dois Olhos	In_Signal7	*n.a*	Estabilizar

Devido ao facto da rede de Petri gerar apenas valores positivos (*unsight int*)[20] e o sensor giroscópico horizontal do capacete EPOC ler valores negativos para a direção esquerda, e na fase de testes ter-mos observado o angulo mais confortável, então foi necessário tomar atenção da criação de um *off-set*[21] considerado suficiente para o movimento da cabeça do utilizador, como mostra a Figura 55, de maneira a ser feia a compensação, concluindo que um valor de 4000 seria o ideal para ser representado o valor estático deste sensor, sendo que é ativada abaixo dos 3000 ou acima dos 5000.

[20] *Numero natural inteiro positivo.*
[21] *Compensar, desfasamento, desvio.*

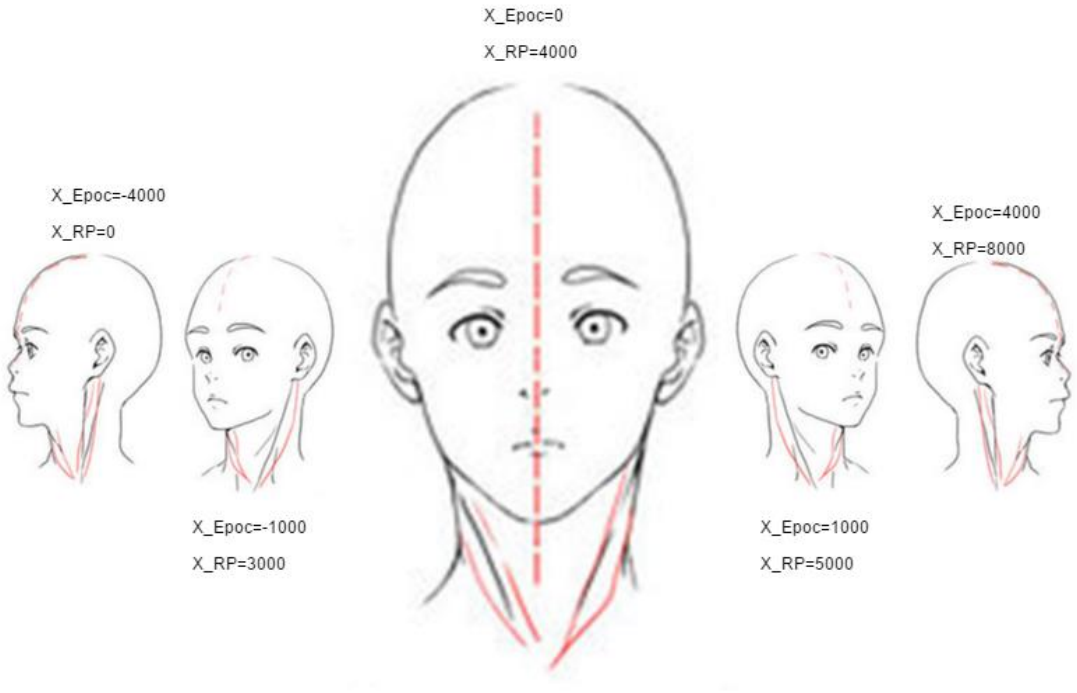


Figura 55: Representação do comportamento do sensor Horizontal.

No caso do sensor vertical foi verificada a mesma situação, quando o capacete toma uma direção no sentido para baixo retorna valores negativos, tendo sido utilizada a mesma solução, representado na Figura 56. A posição estática encontra-se nos 4000, sendo que é ativada abaixo dos 3000 ou acima dos 5000.

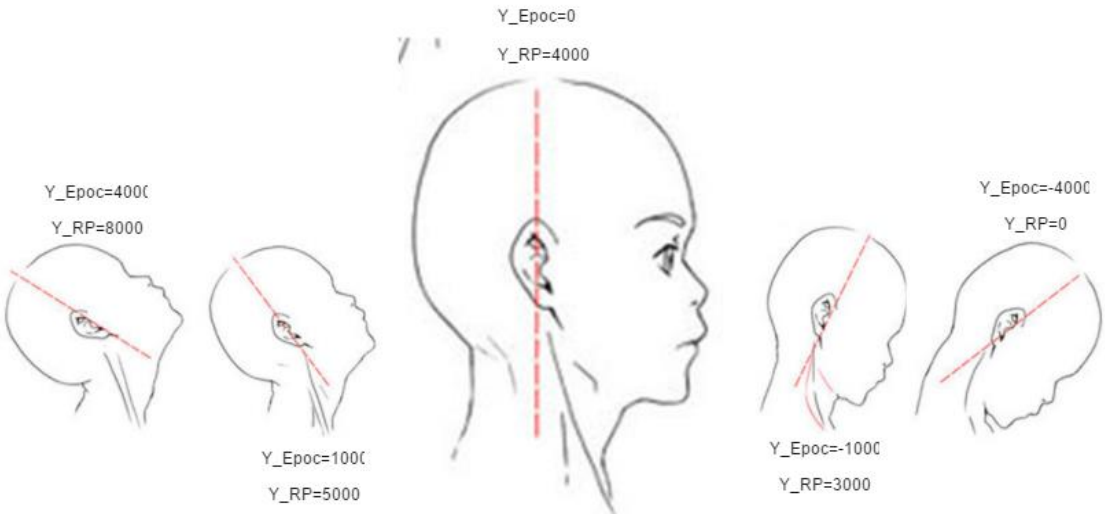


Figura 56: Representação do comportamento do sensor Vertical.

Durante a fase de testes foi evidente que quando é captado um sinal expressivo este é detetado várias vezes apenas com um impulso. No caso de

querer-mos levantar o quadricóptero ou aterrar com o piscar de olho direito (função ***WinkRight()***) e sendo a mesma função para as duas operações adotámos a técnica demonstrada na Figura 57, que passa por ser necessário registar o evento ativo e depois nulo.

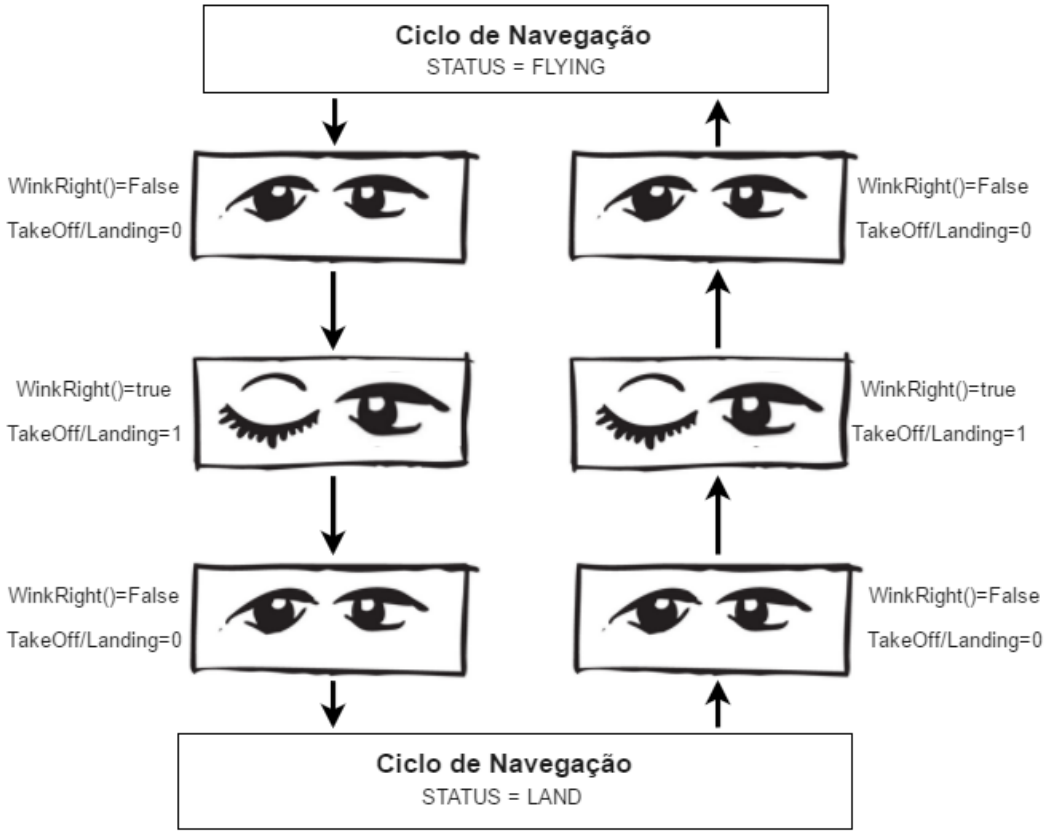


Figura 57: Representação do comportamento do sinal *WinkRight()*

Devido à sensibilidade de certos sinais captados pelo capacete, por vezes foi verificado a ocorrência de situações em que os sinais ficavam presos no mesmo lugar. Para que fosse executável lidar com estes casos de forma segura no momento de voo recorreu-se à função ***blink()*** para permitir sair desses estados, como é demonstrado na Figura 54 e Figura 58, onde é possível verificar que em certas situações, mesmo que uma transição continue a aceitar um sinal ativo, o sinal *blink()* é capaz de mudar a sua condição de lugar.

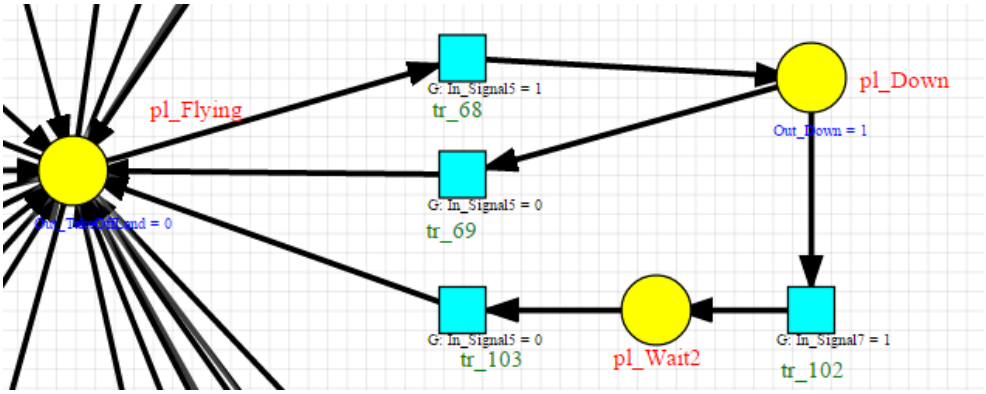


Figura 58: Aplicação da função *blink() da RdP*.

5.2.3 Simulação e execução da Rede

Depois que o modelo foi concebido com êxito, ele pode ser executado no simulador para verificar se ele se comporta corretamente em cada caso de uso esperado, como apresentado na Figura 59.

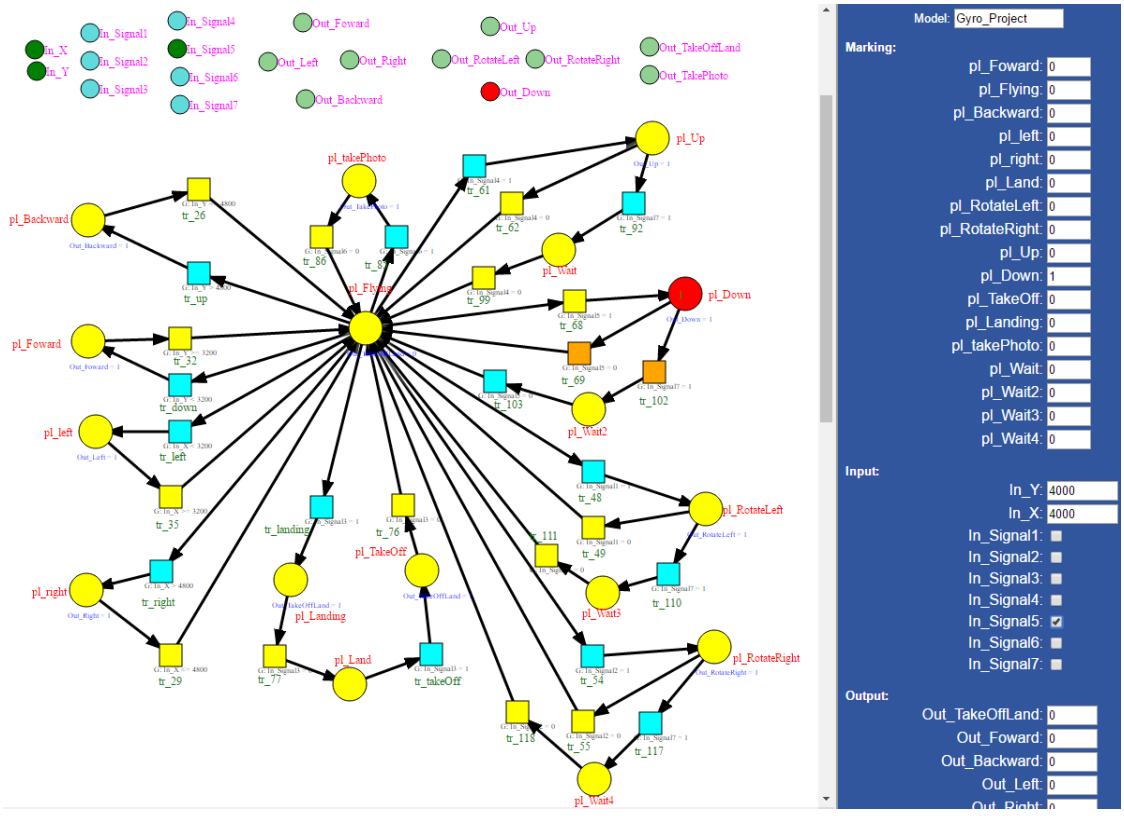


Figura 59: Ambiente de simulação da rede nas IOPT-Tools.

Verificou-se que o modelo projetado pela ferramenta IOPT comporta-se conforme o esperado em todos os casos de uso, ou seja, com diferentes sequências de entrada.

5.2.4 Validação/Verificação dos Estados

Para ajudar a detetar situações em que sequências inesperadas, como sinais de entrada ou interações do utilizador, possam levar a resultados indesejados é evocado a verificação de estados representada na Figura 60, uma operação com um modelo de verificação, composta pela ferramenta de geração de espaço de estados automático, e um sistema de consulta fornecendo a capacidade de detetar tais erros e verificar as propriedades importantes do sistema.

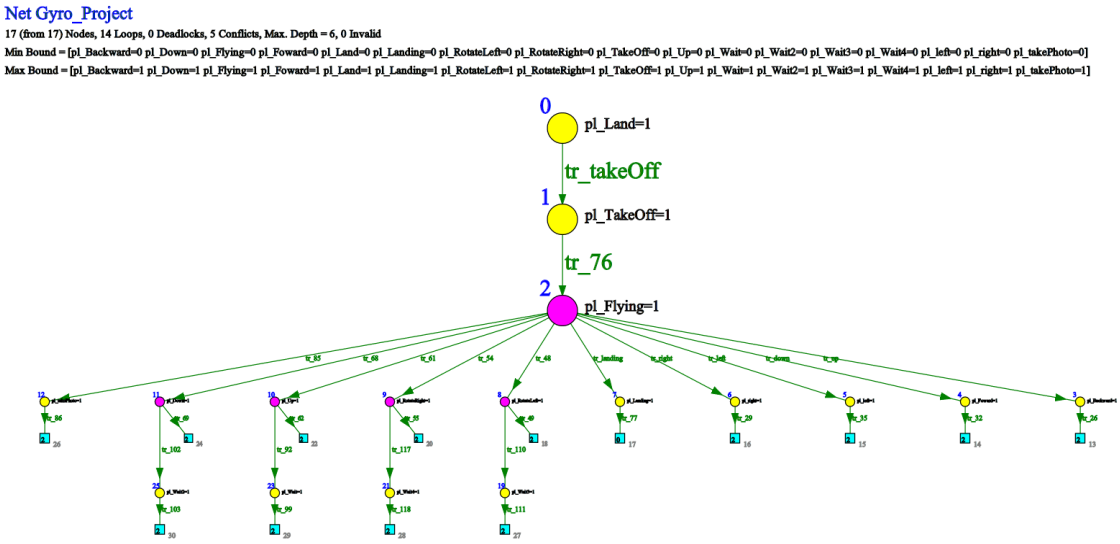


Figura 60: Geração de Espaço de Estados.

5.2.5 Geração automática de código

É produzida pela ferramenta um arquivo de ficheiros tal como ilustrado na Figura 31. Por razões de compatibilidade, o código gerado que utiliza regras de sintaxe *ANSI C*, no momento da inserção do projeto em *C++*, em cada ficheiro no nosso caso foi substituído a terminação por "*.cpp*". Todo o código produzido consegue ser diretamente compilado na maioria dos sistemas, exceto o arquivo "*net_io.cpp*" que sofre de uma adaptação à nossa arquitetura devido às entradas e saídas do sistema [25].

5.3 *Aplicação* ICC

Nesta fase do projeto foi criada um programa, intitulado de "*Dron*EPOC", escrito na linguagem *C++* através da plataforma *MS Visual Studio*, integrando o *SDK* "EPOC*ontro*l" desenvolvido no capítulo 4 e os ficheiros gerados da *RdP* referidos na Figura 31. Por fim é adicionado no ficheiro *net_io.cpp*, como já referido anteriormente, as inicializações da rede, do EPOC, os sinais de entrada e saída, tudo o que compete à aplicação fazer.

5.3.1 Código fonte adicionado ao projeto

O arquivo *net_io.cpp* gerado pela ferramenta das IOPT's compreende cinco funções: *InitialzeIO()*, *GetInputSignals()*, *PutOutputSignals()*, *LoopDelay()* e *finishExecution()*, que devem ser preenchidas manualmente pelo programador usando código específico, incluído a chamada das bibliotecas no inicio do ficheiro, apresentadas no Anexo 0 [25].

Na função de **Inicialização** será necessário configurar o capacete como também iniciar a comunicação com a aplicação "*ARDrone*", através das funções descritas no anterior capítulo e no Anexo 0.

Durante o **ciclo de atividade** da aplicação são executadas as funções *GetInputSignals()* e *PutOutputSignals()* descritos no Anexo 0, onde participam as funções *interrupt()* e *newAction()*, que leem os sinais atuais do capacete como *inputs* do modelo, cujos *Outputs* atualizados são enviados através da função *Send()*.

No momento que a função *FinishExecution()*, descrita no Anexo 0, é executada o programa **termina**, desligando conforme o procedimento que a arquitetura do EPOC deseja, gravando as capacidades do utilizador atual, gerando um ficheiro com toda a informação necessária.

5.3.2 Interface Gráfica

As saídas do sistema e os estados do capacete são impressos numa consola ao fim de cada ciclo de modo a conseguirmos observar as instruções a serem executadas, como demonstra a Figura 61. No decorrer dos testes foi permitido também utilizar o depurador da ferramenta das IOPT's representado na **Erro! A**

origem da referência não foi encontrada. de modo a observar o comportamento da rede.

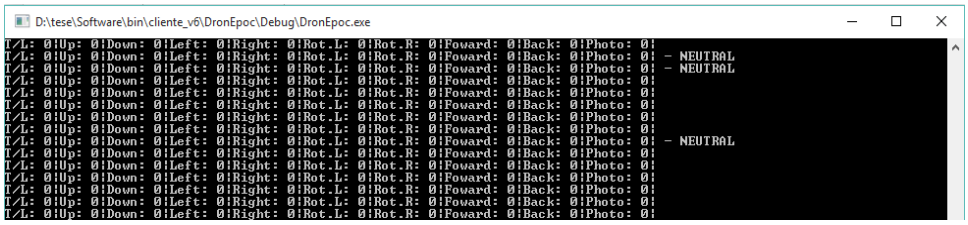


Figura 61: Consola *Win32* com as saídas da *RdP* e os sinais do EPOC.

De modo a conseguir obter um *feedback* em tempo real, da qualidade de cada elétrodo, usaremos o software Emotiv *Control Pannel*, representada na Figura 20, e o *TestBench* da Figura 23, em toda a execução.

5.3.3 Interface de comandos

De maneira a conseguir evocar as funções de chamada foram reservadas algumas teclas do teclado apresentadas na Figura 62 com a respetiva legenda na Tabela 11.

Figura 62: Interface de comandos do projeto "*Dron*EPOC".

Tabela 11: Legenda das funções da interface de comandos referentes ao "*DronEpoc*".

Comandos	Instrução
Letra "P"	Pausa ao programa.
Letra "T"	Treinar expressões faciais e/ou pensamentos.
Letra "V"	Sinal Sem fios do Capacete.
Letra "B"	Nível de bateria do Capacete.
Letra "N"	Leitura e introdução da sensibilidade de todos os sinais.
Letra "M"	Leitura e introdução da sensibilidade de cada sinal.
Números de 1-9 Tecla "Enter"	Instruções.
Letra "END"	Encerrar o programa.

5.4 Troca de Mensagens

No momento do projeto onde são gerados os dois executáveis responsáveis pelo sistema de controlo foi indispensável criar a camada de comunicação entre eles, isto é, foi necessário garantir um tipo de comunicação suficientemente robusto. Com base na solução proposta para este projeto foram calculadas as melhores soluções viáveis para esta camada, resumindo-se os tipos funcionais possíveis.

5.4.1 Camada de comunicação EPOC-*Computador*

A comunicação entre o capacete Emotiv EPOC e a aplicação "*Dron*EPOC" anteriormente proposta acontece de um modo bidirecional, havendo troca de mensagens, pergunta e resposta, entre estes dois dispositivos. É um tipo de comunicação sem fios, com uma tecnologia de comunicação *Bluetooth*. Acrescenta-se o facto de que se a distância entre os dois for minimamente elevada (1 metro) o sinal da ligação perde-se facilmente.

5.4.2 Camada de comunicação *Computador*-Quadricóptero

A comunicação estabelecida entre a aplicação de controlo do quadricóptero e o próprio é feita do mesmo estilo, sem fios, mas com uma tecnologia via *Wi-Fi*. O *drone* transporta, para além do módulo de memória e do controlador logico, uma placa de rede *Wi-Fi*, de maneira a permitir a comunicação com um dispositivo de controlo de navegação. Ocasionalmente é necessário fazer um *reset* à placa do quadricóptero, devido ao excesso de tentativas de comunicação com o aparelho.

5.4.3 Comunicação entre programas

A comunicação feita entre os executáveis das aplicações concebidas podia ser feita de inúmeras formas, tendo sido necessário analisar o melhor protocolo a usar depois de definido o seu objetivo. Conhecido como comunicações entre processos (*IPC*[22]), os sistemas operativos fornecem mecanismos que permitem

[22] - inter-process communication

que processos possam trocar dados entre si. Geralmente este tipo de mecanismos usa como base um cliente e um servidor. Nos parágrafos seguintes são referidos alguns métodos existentes, cuja utilização varia de acordo com a o requisito do sistema, como o desempenho, modularidade, etc.

Sample File - Trata-se de um ficheiro de texto registado no disco que um programa cria inicialmente, escreve, grava e fecha. Sempre que um processo abre o ficheiro para ler ou escrever, mais nenhum terá permissão para conseguir aceder enquanto estiver aberto.

Message Queue - Semelhante ao fluxo de dados de um "*socket*", trata-se de um método implementado pelo sistema operativo *Windows*, onde permite que vários processos leiam e escrevam numa fila de mensagens sem ser diretamente ligados, sofrendo de uma ordem de prioridades. Infelizmente pouco desenvolvida em certas linguagens.

NamedPipe - Método capaz de implementar um canal de comunicação através de um arquivo no sistema ao invés da entrada e saída padrão. Vários processos podem ler e gravar o arquivo. Utiliza um comportamento *FIFO (First-In-First-Out).*

PipeMessage - Método que lida com um fluxo de dados bidirecional entre dois processos, interligados através da entrada e saída padronizada de informação e onde se lê um carácter de cada vez. Este tipo de abordagem foi realizado no projeto, mas devido ao período curto da taxa de envio de dados necessário no controlo do *drone* tornou-se ineficaz. Isto acontece devido ao facto do tipo de comunicação *FILO (First-In-Last-Out)*, observando-se um atraso exponencial entre o comando destinado à ação reação do *drone*.

Sockets - Esta foi a abordagem adotada na versão final deste projeto. O método especificado neste protocolo é definido pelas aplicações que estabelecem a conexão, constituído por um mecanismo cliente servidor. Consiste numa chamada inicial de um endereço *IP* na camada de rede seguido de um enquadramento básico da mensagem, em camadas de transporte *TCP (Transmission Control Protocol)*. Passa a existir uma conexão persistente, com um fluxo de dados enviados através de uma interface de rede. O servidor fica assim capaz de ler a última

mensagem enviada pelo cliente, quantas vezes for necessária, devido ao seu comportamento.

5.4.4 Regime de Troca de Mensagens

Como já foi referenciado anteriormente, a comunicação entre as duas aplicações (do controlador do capacete EPOC e do *drone*) é então feita através de *sockets*, recorrendo a uma variável reservada, sendo composta por dez carateres binários, cada um correspondente a uma ação de controlo, como é ilustrado na Figura 63.

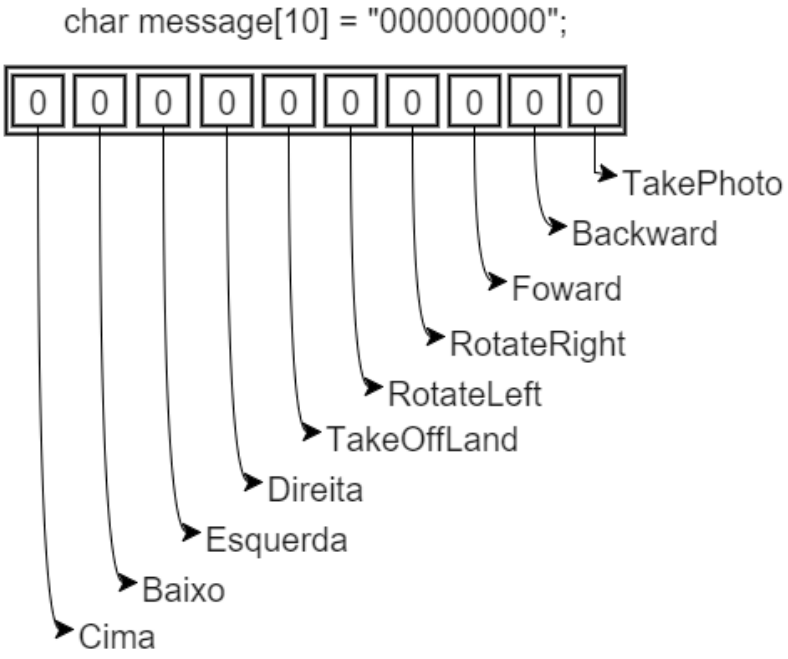


Figura 63: Mensagem entre as duas aplicações com as ações de controlo.

Ao fim de cada ciclo de execução da rede de Petri esta variável é atualizada com as saídas do modelo, sendo que Zero representa o estado inativo e o valor Um representa o estado ativo da ação de controlo. Esta variável é então enviada através de um endereço e recebida no lado da aplicação que controla o *drone.* No anexo I está disponível um fluxograma de controlo de troca de mensagens entre as aplicações (“algoritmo de decisão de eventos”).

5.5 Aplicação *do* Quadricóptero

Devido à necessidade de validar o modelo de controlo concebido não apenas de uma forma gráfica e abstrata, recorreu-se ao uso de um quadricóptero para promover uma experiência real.

5.5.1 Implementação do controlo do quadricóptero

O nome dado à aplicação usada para comandar o quadricóptero foi "*ARDrone*". O projeto foi realizado em *MS Visual Studio* numa linguagem *C#*. A solução representa um conjunto de vários projetos e classes, como é indicado na Figura 64.

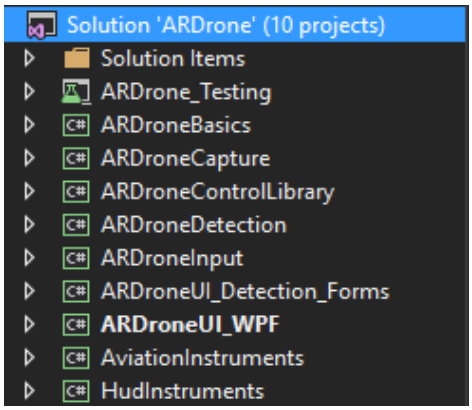


Figura 64: Solução do projeto "*ARDrone* " no ambiente *Visual Studio*.

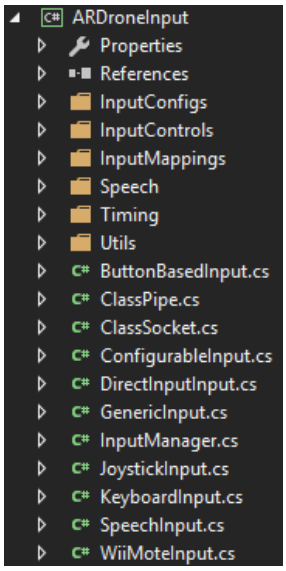


Figura 65: Classes de controlo de entrada do *drone* no ambiente *Visual Studio*.

A classe *ARDroneUI_WPF* corresponde à *GUI***Erro! Marcador não definido.** da aplicação, enquanto a classe *ARDroneInput*, representada na Figura 65 contém o código fonte referente às instruções vindas do teclado, e outras duas classes dedicados à comunicação entre aplicações na mesma máquina.

A classe *KeyboardInput* tem vários objetivos. Primeiro escuta todas as teclas imprimidas no teclado e compara-as com as teclas reservadas aos comandos utilizados para navegar o quadricóptero, como representado no Código 20 do Anexo G, executando-as caso isto se verifique. Outro objetivo é escutar, num canal *socket*, as mensagens recebidas, controlando a navegação do quadricóptero como se trata-se das teclas do teclado, apresentado no Código 21 no Anexo G.

A escolha feita no tipo de comunicação foi definida pelas necessidades que existem em controlar o quadricóptero. Trata-se de um objeto que deve responder em tempo real no momento que é enviado um comando, por exemplo, se o *drone* estiver a ir em frente ao encontro de um obstáculo o utilizador deve enviar um comando para que se desvie. Esse comando não pode sofrer de atrasos. Ora nos diferentes tipos de tecnologias de comunicação entre aplicações, o s*ocket* permite ao *drone* escutar no canal a última mensagem enviada pela aplicação que usa o capacete. Caso usássemos *PipeMessage*, implementada e descrita no Anexo G, as mensagens seriam reservadas em lista e o *drone* iria aceder à mensagem com mais tempo de vida. Recorreu-se então à classe *AsynchronousSocketListener* que através de um *socket* descrito anteriormente cria um canal *TCP/IP,* como apresentado no Código 22 do Anexo G, que permite a comunicação de estações através de um endereço, neste caso denominado *"127.0.0.1"* com a porta *"8888"*, procurando, sempre que é possível, receber a última mensagem devolvida. Esta Comunicação permite a troca de mensagens, através da classe *mensagem*, ***c***riada apenas para reservar uma *string* de carateres representada no Código 23 do Anexo G, utilizada para alojar as mensagens recebidas pelo *socket*.

5.5.2 Comandos do Utilizador

Tal como foi referido anteriormente, a aplicação *"ARDrone"* lida com a navegação do quadricóptero recorrendo ao teclado. Na Figura 66 estão descri-

tos os comandos usados neste projeto possíveis de ser acedidos a qualquer altura da experiência, tendo como propósito a total segurança do aparelho e do meio envolvente.

Figura 66: Teclas do teclado usadas como interface de comando do quadricóptero.

5.5.3 Interface gráfica de Simulação

No decorrer da implementação do sistema de controlo de navegação foi necessário observar as saídas do sistema aplicadas aos comandos do quadricóptero para poder inicialmente modelar algum tipo de erro de navegação, como também abordar da melhor forma um voo real. Para isso foi necessário implementar uma *GUI* capaz de simular um voo e mais tarde servir de interface de comando dos voos práticos, na presença tanto do teclado como dos comandos de segurança, bem como o uso desenvolvido do capacete EPOC (principal interesse do projeto).

A aplicação "*ARDrone*" concede as indicações necessárias quando é executado, e caso o quadricóptero não tenha sido detetado, ele perguntará se é desejado dar continuidade ao processo, permitindo o uso do ambiente de controlo exibido na Figura 67 como um simulador.

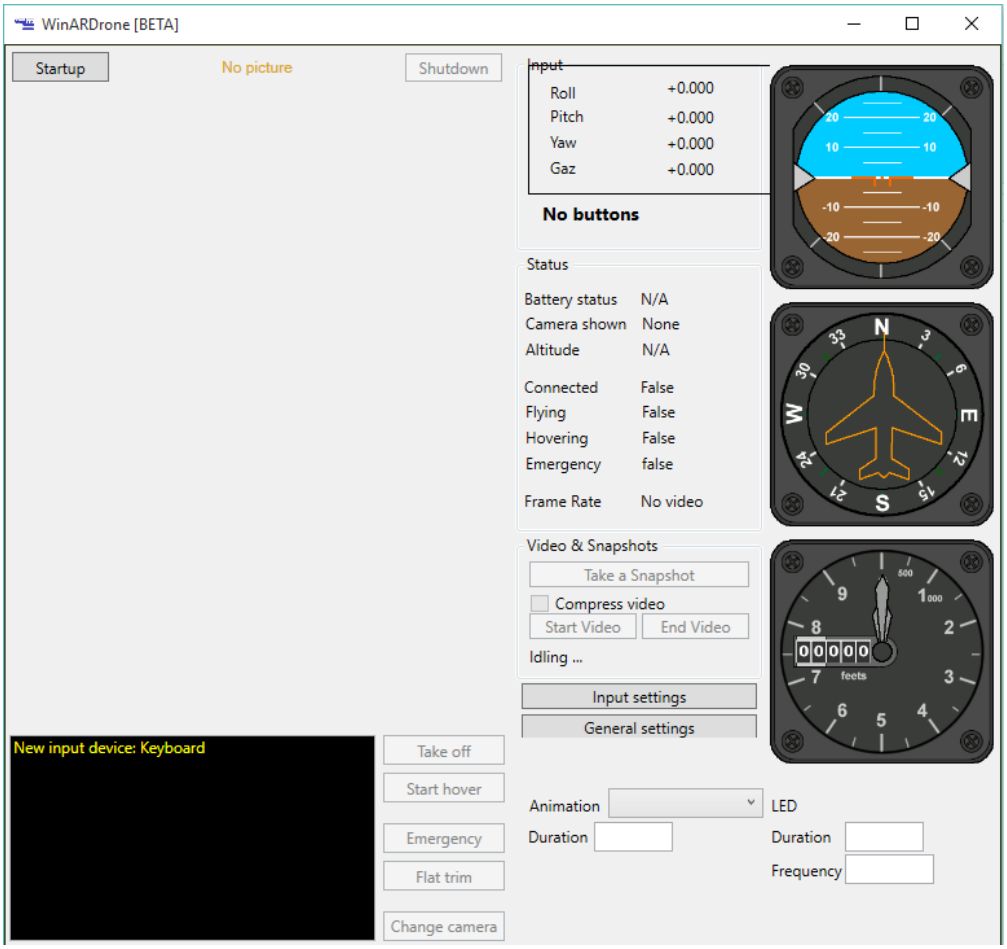


Figura 67: Ambiente principal de navegação do quadricóptero.

É de realçar que o programa apenas começa a executar os comandos de navegação do quadricóptero caso a aplicação ICC seja executada. Isto acontece devido ao facto do programa ficar à escuta num endereço *TCP* que será só iniciado através do *socket* criado a partir da aplicação "*Dron*EPOC".

A aplicação "*ARDrone*" tem até aqui apenas uma componente de simulação. No momento que esta aplicação faz "*StartUp*" na rede gerada pelo quadricóptero então esta inicia uma conexão, e torna possível dirigi-lo.

Entretanto, sendo o quadricóptero portador de uma camara capaz de captar um vídeo em *stream*, com o sentido da direção de voo, foi explorada e desenvolvida uma solução capaz de trazer ao utilizador a possibilidade de observar em tempo real o que o quadricóptero "observa". Nesse sentido, foi criada uma diretoria "Software/*FFmpeg*" com um executável "*ffplay.exe*" capaz de aceder ao servidor através de *TCP*, como apresenta a Figura 68. Devido à quali-

dade da imagem a transmissão sofre de um atraso, sendo necessário aumentar o *framerate* da transmissão e diminuir o tamanho de dados "*probesize*" a analisar, de modo a não sofrer desse mesmo *delay*[23] no momento de voo.

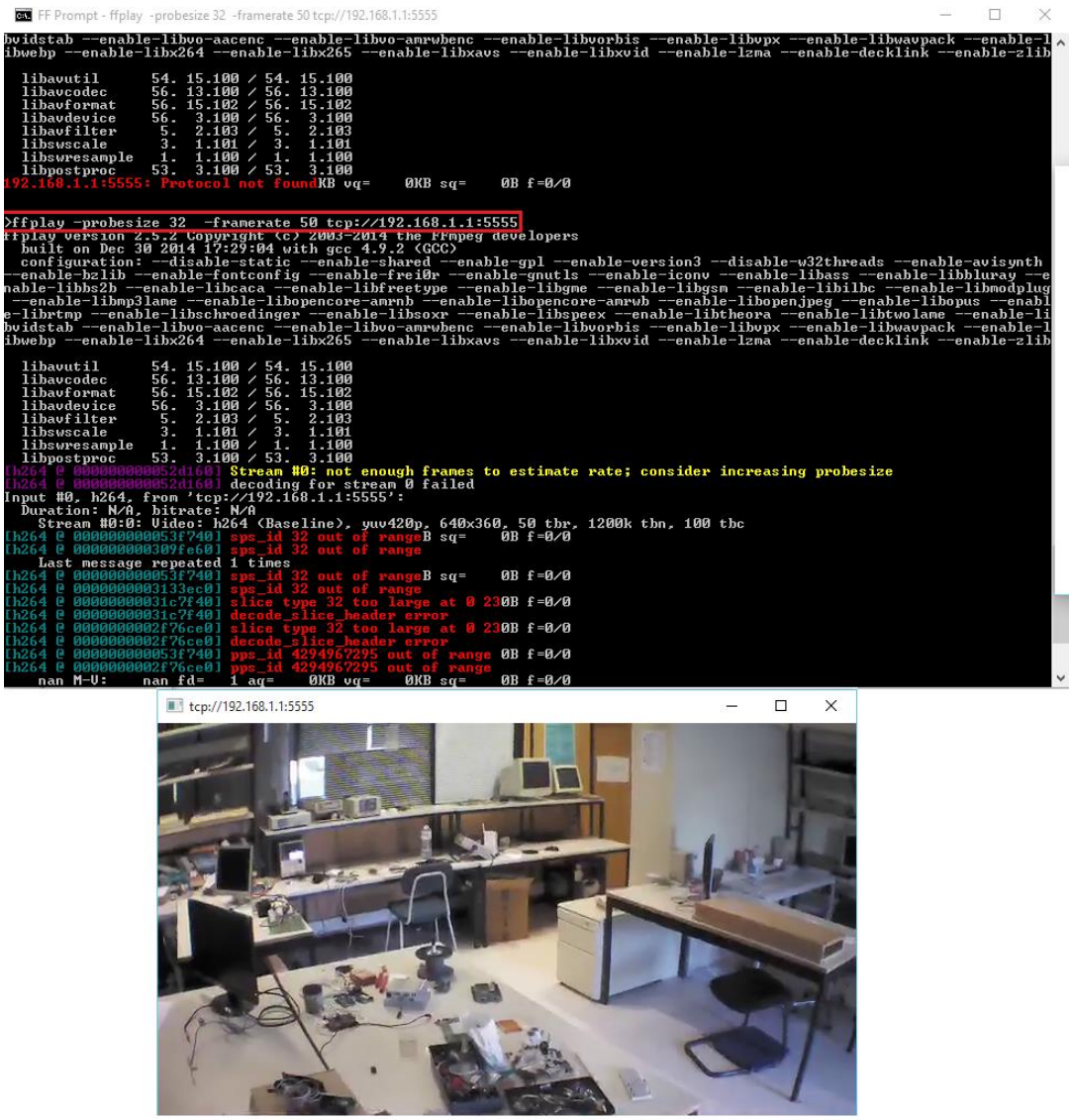


Figura 68: Interface de vídeo *stream*[24] do quadricóptero com a respetiva linha de comandos e o vídeo *stream* gerado.

[23] *Atraso temporal.*
[24] *Envio de informação multimídia em tempo real.*

5.6 Resultados e discussão

De acordo com a solução proposta foi posto em prática o projeto desenvolvido, executando o sistema de controlo gerado pela ferramenta das IOPT's integrado com a biblioteca de desenvolvimento de software, de modo que a aplicação que permite controlar o quadricóptero comunicasse com o EPOC. Por sua vez foram relatados alguns resultados dos vários ensaios feitos ao longo dos últimos meses e na fase final da dissertação, isto para que fosse possível garantir tornar a versão final das aplicações do sistema o mais estáveis possível.

5.6.1 Descrição dos testes práticos

O ensaio proposto sugere um voo controlado do quadricóptero, onde implicasse diferentes utilizadores recorrerem apenas ao capacete EPOC para conseguirem controlar a sua navegação, completando um percurso definido. Este percurso implicou a utilização de todas as funções que a aplicação dispõe. No decorrer dos testes existiu um utilizador que utilizou um teclado (Figura 69), cujas funções foram descritas anteriormente na Tabela 9 e Tabela 11, e uma interface gráfica como apresenta a Figura 70, garantindo a não ocorrência de situações críticas de voo para o que o rodeia.

É necessário que utilizador se guie pelo fluxograma disponível no anexo H ("Fluxo de procedimentos do utilizador para realização de experiência") no momento de testar o projeto e viver a sua experiência de voo.

Figura 69: Teclas do teclado usadas como interface de comando do quadricóptero e do EPOC.

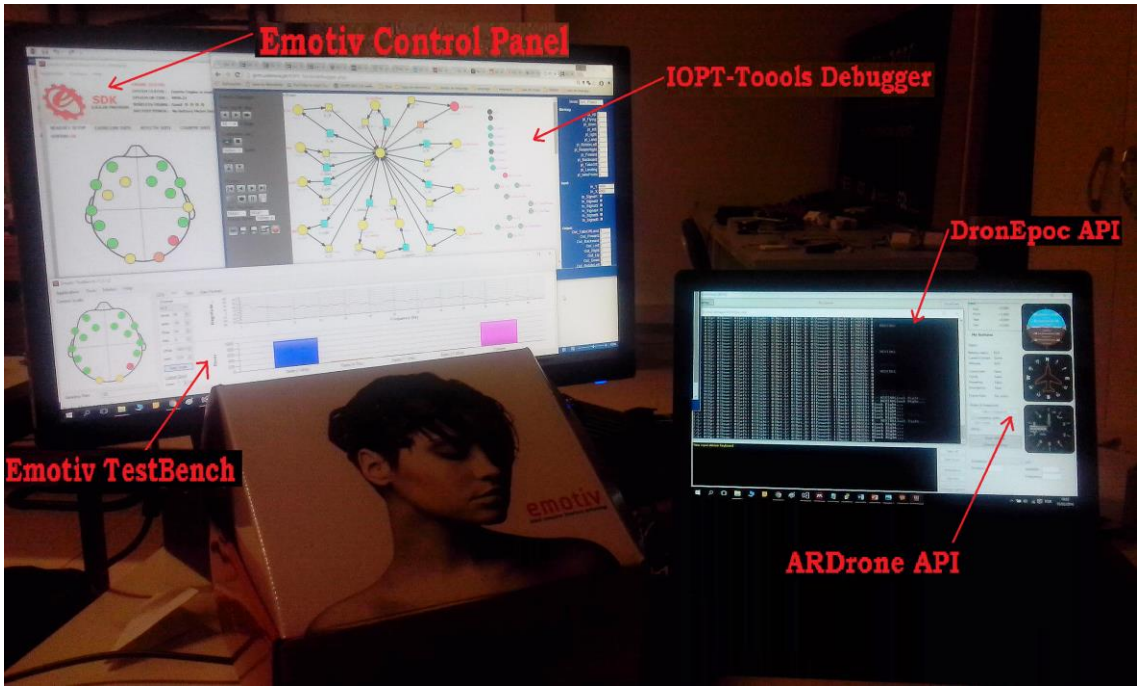


Figura 70: Ambiente gráfico de utilizador.

5.6.2 Resultados Experimentais

Na avaliação ao software deste projeto, constituído pelos métodos descritos ao longo do documento, foram realizados testes experimentais durante os dois últimos meses, como é ilustrado na Figura 71, de modo a validar a solução proposta.

Foi possível garantir um voo controlado com as funções que recorriam ao sensor giroscópico sem quaisquer problemas, tendo sido considerado um sucesso. Funções como descolar e aterrar o quadricóptero através de expressões faciais por parte do utilizador mostraram-se igualmente perfeitas. A utilização de várias expressões faciais com diferentes funções de navegação foram a chave para um voo controlado do quadricóptero. Por último a utilização dos sinais cognitivos mostraram ter alguma qualidade quando o utilizador despendia o seu tempo no treino neuronal. A grande vantagem para isso foi certamente a criação e utilização de assinaturas entre utilizadores para este tipo de sinais, definindo assim uma exclusividade entre cada um com padrões distinto.

É importante salientar que o modelo apresentado no decorrer deste documento, representado pela Figura 54, foi o resultado de constantes mutações

derivado às questões técnicas ligadas aos sinais captados que implicariam um controlo do sinal mais minucioso ou de outro modo limitado pela necessidade de resposta ao quadricóptero em tempo-real.

Figura 71: Experiência prática.

6

Conclusões e Trabalho Futuro

"In the end, your success will speak for itself"

Patrick Bet-David, financial adviser

Chegado ao fim desta dissertação, do estudo de tecnologias em torno dos ICC's e do desenvolvimento de um *Kit* que enriquece uma ferramenta assistente à modelização de sistemas de controlo, é possível entender a carência existente nesta matéria até há pouco tempo, e ao mesmo tempo, o grande desenvolvimento tecnológico que começa a surgir.

Este trabalho prova a sua grande vantagem ao permitir o uso do capacete EPOC e usar os sinais captados traduzidos em quatro abordagens distintas (deteção da orientação da cabeça, deteção de expressões faciais, deteção de estados emocionais e a deteção de estados mentais intencionais) permitindo assim afetar ou comandar sistemas eletrónicos digitais interagindo de forma natural, como exemplo, ajuste dinâmico do nível de dificuldade dos jogos em resposta ao aumento de frustração, ajuste da música do agrado da pessoa, controlo de iluminação e outros fatores ambientais em resposta a estados de espírito, controlo de veículos, aplicações, etc.

Tendo como uma excelente experiência o desenvolvimento do projeto de controlo de navegação do *drone* através do capacete, é possível garantir o sucesso de projetos que possam ser realizados sob a mesma linha definida neste tra-

balho, pois a aplicação deste tipo de tecnologia neste caso prova a sua capacidade minuciosa de controlo e decisão. Comprovou-se o êxito de muitas das funcionalidades propostas, algumas carências complexas do sistema e ainda algumas futuras recomendações a ter em consideração no futuro deste projeto.

Ao nível dos sinais captados pelo capacete EPOC conseguimos concluir no decorrer dos ensaios que o teste de validação proposto demonstrou a infinita margem de progresso no âmbito desta área, nomeadamente nas inúmeras aplicações que poderão ser propostas no futuro com um tipo de abordagem destas.

Também foi possível concluir com esta investigação que as questões éticas e sociais decorrentes na utilização de ICC's são muitas e incluem principalmente autonomia, identidade, propriedade, privacidade, igualdade e responsabilidade, sendo que estas precisam ainda de ser respondidas, e no foro sociológico das tecnologias de informação é interessante estudar o reconhecimento deste tipo de tecnologias por parte da sociedade no futuro.

6.1 Contribuições da Dissertação

Nesta dissertação concebeu-se uma biblioteca que reúne as fontes necessárias às IOPT-Tools de modo a modelar controladores digitais capazes de usar sinais neuronais e giroscópicos do capacete EPOC nos eventos de controlo. Deste modo, o trabalho desenvolvido fornece ainda contribuições com testes experimentais, tendo em conta um projeto com um número limitado de casos de estudo dedicados ao controlo de um quadricóptero num espaço *tridimensional* elaborando assim um tipo de solução de validação para softwares desenvolvidos no uso deste tipo de abordagens.

6.2 Adversidades

A partir da análise ao capacete utilizado no estudo torna-se evidente que este tipo de abordagem tecnológica deve ser capaz de superar obstáculos como a anulação do sinal de ruído. Os sinais neuronais interessantes a captar estão por vezes "escondidos" no ruído, presente em redor do ambiente em estudo, ou seja, o sinal que pretendemos captar, registar e analisar, na maioria é de potên-

cia negativa comparado com a amplitude do sinal exterior, como por exemplo, frequências de sinais de telemóvel, de aparelhos eletrónicos, do próprio batimento cardíaco ou tensão dos músculos. É importante garantir no futuro uma melhor aquisição do sinal pretendido e não ter que submeter o indivíduo em condições fora da sua condição normal ou levando-o ao método invasivo. Portanto o uso desta tecnologia necessita também de ser enquadrado numa realidade que melhor inclua a fácil utilização do aparelho.

Outro dos problemas em redor à consistência do sinal registado advém da viagem de um sinal tão frágil através dos elétrodos e dos fios até o amplificador, gerando ao longo deste processo uma quantidade de ruído suficientemente maligna. Usar um transístor em lugar de um elétrodo comum é um salto qualitativo, sem precedentes, já com provas científicas fortes que indicam um melhor resultado.

6.3 Trabalho Futuro

Apesar de tudo o que foi realizado até ao momento, existe uma série de melhorias e novas funcionalidades que podem melhorar o trabalho realizado e abrir portas a novos projetos. Como tal, para trabalho futuro no âmbito desta tese sugere-se:

- ✓ Conclusão da biblioteca desenvolvida no âmbito dos sinais emocionais, do *layout* da *GUI* dos comandos dedicada ao capacete, da gestão de perfis de utilizador e por fim uma integração com realidade virtual.
- ✓ Elaboração de um projeto de desenvolvimento de aplicação sob uma cadeira de rodas movida através do capacete Emotiv EPOC, destinado a deficientes.
- ✓ Desenvolvimento de uma segunda abordagem ao capacete Emotiv EPOC, esta de mais baixo nível, com o objetivo de recolher os dados dos sinais lidos e processá-los, através da produção de um algoritmo, necessitando de uma análise minuciosa ao sinal, e relacionando diferentes estados do mesmo, nas diversas plataformas aprendidas ao longo do curso como a ferramenta *MatLab*. Posteriormente, aplicação de técnicas, como algoritmos de aprendizagem e

inteligência artificial, lógica difusa, heurísticas, entropias, probabilidades, etc.

- ✓ Um levantamento detalhado de outras futuras aplicações onde haja interesse em utilizar os sinais propostos neste trabalho para ativar eventos de controlo em sistemas digitais possíveis de serem modelados por *RdP*. Exemplos de áreas: *Egames*, psicologia, pesquisa aerospacial, defesa militar, medicina.
- ✓ Um estudo sociológico onde sejam levantadas questões éticas nas abordagens a ter em relação ao uso abusivo e aberto do indivíduo de ICC*'s*, e o seu impacto social.
- ✓ O desenvolvimento de uma nova abordagem ao quadricóptero na construção de um novo *firmware*, instalação essa feita via *FTP (file transfer protocol)* com o servidor e iniciando-o via *TELNET* e/ou *SSH (Secure Shell)*, permitindo assim controlar o aparelho com auxílio de novos dispositivos de controlo.

7

Referências

[1] "Human Brain: Facts, Anatomy & Mapping Project." [Online]. Available: http://www.livescience.com/29365-human-brain.html. [Accessed: 26-Jan-2016].

[2] L. Hirschfeld and S. Gelman, "Mapping the mind," *LAWRENCE A. HIRSCHFELD*, no. University of Michigan, 1994.

[3] E. van der Spoel, M. P. Rozing, J. J. Houwing-Duistermaat, P. Eline Slagboom, M. Beekman, A. J. M. de Craen, R. G. J. Westendorp, and D. van Heemst, *The Human Body*, vol. 7, no. 11. 2015.

[4] Marisa Pinto, "O Cérebro Humano: Conceitos Básicos | Psicologia para o Futuro," 2012. [Online]. Available: https://psicologiaparaofuturo.wordpress.com/2012/08/02/o-cerebro-humano-conceitos-basicos/. [Accessed: 02-Mar-2016].

[5] M. M. Amarasinghe. K, D.Wijayasekara, "EEG based brain activity monitoring using Artificial Neural Networks," *… Syst. Interact. (HSI …*, no. University of Idaho, pp. 61–66, 2014.

[6] M. Jobert, "Recording EEG and EP Data." [Online]. Available: https://journals.prous.com/journals/servlet/xmlxsl/pk_journals.xml_summary_pr?p_JournalId=6&p_RefId=832&p_IsPs=Y. [Accessed: 17-Mar-2016].

[7] von André Hoffman, "EEG Signal Processing and Emotiv's Neuro Headset," *Arbeit*, p. 49, 2010.

[8] J. J. Vidal, "Real-time detection of brain events in EEG," *Proc. IEEE*, vol. 65, no. 5, pp. 633–641, 1977.

[9] P. Chowdhury, S. S. Kibria Shakim, M. R. Karim, and M. K. Rhaman, "Cognitive efficiency in robot control by Emotiv EPOC," *2014 Int. Conf.*

Informatics, Electron. Vision, ICIEV 2014, pp. 1–6, 2014.

[10] J. J. Vidal, "Toward direct brain-computer communication.," *Annu. Rev. Biophys. Bioeng.*, vol. 2, pp. 157–80, 1973.

[11] M. Fatourechi, A. Bashashati, R. K. Ward, and G. E. Birch, "EMG and EOG artifacts in brain computer interface systems: A survey," *Clin. Neurophysiol.*, vol. 118, no. 3, pp. 480–494, 2007.

[12] L. T. G. Silva, "Modelagem, simulação e controle de um VANT quadrirotor," *Proj. Grad. apresentado*, no. Universidade Federal do Rio de Janeiro, p. 65, 2015.

[13] M. I. Ribeiro, "Sensores em Robótica." http://users.isr.ist.utl.pt/~mir/pub/sensores.pdf, Lisboa,Institute for Systems and Robotics, p. 2, 2004.

[14] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4. pp. 237–285, 1996.

[15] "ETH - Sensory-Motor Systems Lab - Mission." [Online]. Available: http://www.sms.hest.ethz.ch/mission. [Accessed: 15-Feb-2016].

[16] G. A.Boy, *The Handbook Of Human-Machine Interaction*, ASHGATE e-. Florida Institute of Technology, USA: NASA Kennedy Space Center, USA, 2011.

[17] T. J. Faria, "Interfaces cérebro-computador - Utilização do Emotiv EPOC para controlar software lúdico," Instituto Superior de Engenharia do Porto, 2014.

[18] B. Blankertz, M. Tangermann, C. Vidaurre, S. Fazli, C. Sannelli, S. Haufe, C. Maeder, L. Ramsey, I. Sturm, G. Curio, and K. R. Müller, "The Berlin brain-computer interface: Non-medical uses of BCI technology," *Front. Neurosci.*, vol. 4, no. DEC, pp. 1–17, 2010.

[19] H. Cecotti, "A self-paced and calibration-less SSVEP-based brain-computer interface speller," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 18, no. 2, pp. 127–133, 2010.

[20] F. Ben Taher, N. Ben Amor, and M. Jallouli, "EEG control of an electric wheelchair for disabled persons," *2013 Int. Conf. Individ. Collect. Behav. Robot. - Proc. ICBR 2013*, pp. 27–32, 2013.

[21] E. B. B. M. I. for C. a R. A. W. Ouyang, K. Cashion, and V. K. A. I. P. R. W. S. for C. and A. 2013 I. A. Asari, "Electroencephelograph based brain machine interface for controlling a robotic arm," *Wenjia Ouyang*, 2013.

[22] R. W. Ferreira, "Comunicações intra- e inter-circuito de componentes especificados com Redes de Petri," Universidade Nova de Lisboa, 2010.

[23] L. Gomes, J. P. Barros, A. Costa, and R. Nunes, "The input-output place-transition petri net class and associated tools," *IEEE Int. Conf. Ind. Informatics*, vol. 1, no. Vienna, Autria, pp. 509–514.

[24] W. Reisig and G. Rozenberg, *Lectures on Petri Nets~I: Basic Models : Advances in Petri Nets*, no. 1491. 1998.

[25] F. Pereira, F. Moutinho, and L. Gomes, "IOPT Tools User Manual," vol. 2014, no. C, pp. 1–50, 2014.

[26] B. Bogsch, "Activity Diagrams to Petri Nets," 2012. [Online]. Available: https://wiki.eclipse.org/VIATRA2/Activity_Diagrams_to_Petri_Nets. [Accessed: 02-Mar-2016].

[27] C. B. Daniel Gheorghiță*, Ionuț Vîntu, Letiția Mirea, "Quadcopter Control System Modelling and Implementation," *Int. Conf. Syst. Theory, Control Comput.*, pp. 421–426, 2015.

[28] Parrot.com, "AR.Drone 2.0 official site – AR.Freeflight: Fly with iPhone and iPad, videos record & replay in HD, Share on YouTube & Facebook," 2013. [Online]. Available: http://ardrone2.parrot.com/. [Accessed: 28-Jan-2016].

[29] E. EPOC, "Headset and software setup for your Emotiv EPOC neuroheadset," *SpringerReference*. pp. 1–19, 2003.

[30] Emotiv Systems, "Emotiv EPOC Quick Start Guide." p. 2.

[31] K. Holewa and A. Nawrocka, "Emotiv EPOC neuroheadset in brain - Computer interface," *Proc. 2014 15th Int. Carpathian Control Conf. ICCC 2014*, pp. 149–152, 2014.

[32] Emotiv Company, "Emotiv Software Development Kit," 2012.

[33] S. Grude, M. Freeland, C. Yang, and H. Ma, "Controlling mobile Spykee robot using Emotiv Neuro headset," *Control Conf. (CCC), 2013 32nd Chinese*, pp. 5927–5932, 2013.

[34] E. J. Rechy-Ramirez, H. Hu, and K. McDonald-Maier, "Head movements based control of an intelligent wheelchair in an indoor environment," *2012 IEEE Int. Conf. Robot. Biomimetics*, pp. 1464–1469, 2012.

[35] M. A. Wycoff and W. G. Skogan, "User Guide," *Inter-university Consort. Polit. Soc. Res.*, pp. 1–10, 1996.

[36] "FreeFlight 3 on the App Store." [Online]. Available: https://itunes.apple.com/us/app/freeflight-3/id889985763?mt=8. [Accessed: 29-Jan-2016].

[37] "Home - QGroundControl GCS." [Online]. Available: http://www.qgroundcontrol.org/. [Accessed: 10-Feb-2016].

[38] "IOPT tools." [Online]. Available:

http://gres.uninova.pt/iopt_usermanual.pdf. [Accessed: 29-Jan-2016].

[39] L. Gomes, F. Moutinho, and F. Pereira, "'IOPT-TOOLS – A web based tool framework for embedded systems controller development using Petri nets,'" *FPL'2013 – Int. Conf. F. Program. Log. Appl.*, pp. 2–4, 2013.

[40] L. Gomes, F. Moutinho, F. Pereira, J. Ribeiro, A. Costa, and J.-P. Barros, "'Extending Input-Output Place-Transition Petri nets for Distributed Controller Systems development,'" *ICMC'2014 - Int. Conf. Mechatronics Cont*.

[41] F. Pereira, F. Moutinho, and L. Gomes, "'IOPT-Tools - Towards cloud design automation of digital controllers with Petri nets,'" *ICMC'2014 - Int. Conf. Mechatronics Control*, no. Jinzhou, China, pp. 3–5, 2014.

[42] Microsoft, "Visual Studio - Microsoft Developer Tools," 2015. [Online]. Available: https://www.visualstudio.com/. [Accessed: 29-Jan-2016].

[43] Calle, *Introdução à linguagem C*, Versão 2.0. 2009.

[44] P. Baltarejo and J. Santos, *Apontamentos de Programação em C/C++*. Departamento de Engenharia Informática, 2006.

[45] Emotiv, "'EDK_APIREF'Emotiv Software Development Kit Documentation." 2013.

[46] "Comparação de ICC's comerciais." [Online]. Available: https://en.wikipedia.org/wiki/Comparison_of_consumer_brain%E2%80%93computer_interfaces#Comparison. [Accessed: 13-Aug-2015].

[47] "iWinks - Control Your Dreams." [Online]. Available: https://iwinks.org/. [Accessed: 28- Jan -2016].

[48] "Epoc." [Online]. Available: https://emotiv.com/epoc.php. [Accessed: 28- Jan -2016].

[49] "Insight." [Online]. Available: https://emotiv.com/insight.php. [Accessed: 28- Jan -2016].

[50] "FocusBand - Mind Training Headset, proven, intuitive & rugged." [Online]. Available: http://www.ifocusband.com/. [Accessed: 28- Jan -2016].

[51] "Brainwave Monitoring EEG Headband | Melon." [Online]. Available: http://thinkmelon.com/. [Accessed: 28- Jan -2016].

[52] Muse, "MUSE ™ | Meditation Made Easy." [Online]. Available: http://www.choosemuse.com/. [Accessed: 28- Jan -2016].

[53] "MyndPlay Ltd." [Online]. Available: http://www.myndplay.com/products.php. [Accessed: 28- Jan -2016].

[54] "{32bit OpenBCI Kit (8-channel)}. {OpenBCI} website," *[Online] Accedido: 26-Ene-2016*. [Online]. Available:

http://shop.openbci.com/collections/frontpage/products/openbci-32-bit-board-kit. [Accessed: 28- Jan -2016].

[55] "Ganglion Board Kit (4-channel) – OpenBCI Online Store." [Online]. Available: http://shop.openbci.com/collections/frontpage/products/pre-order-ganglion-board. [Accessed: 28-Jan-2016].

[56] "OpenBCI 16-channel R&D Kit – OpenBCI Online Store." [Online]. Available: https://openbci.myshopify.com/collections/frontpage/products/openbci-16-channel-r-d-kit. [Accessed: 28-Jan-2016].

8

Anexos

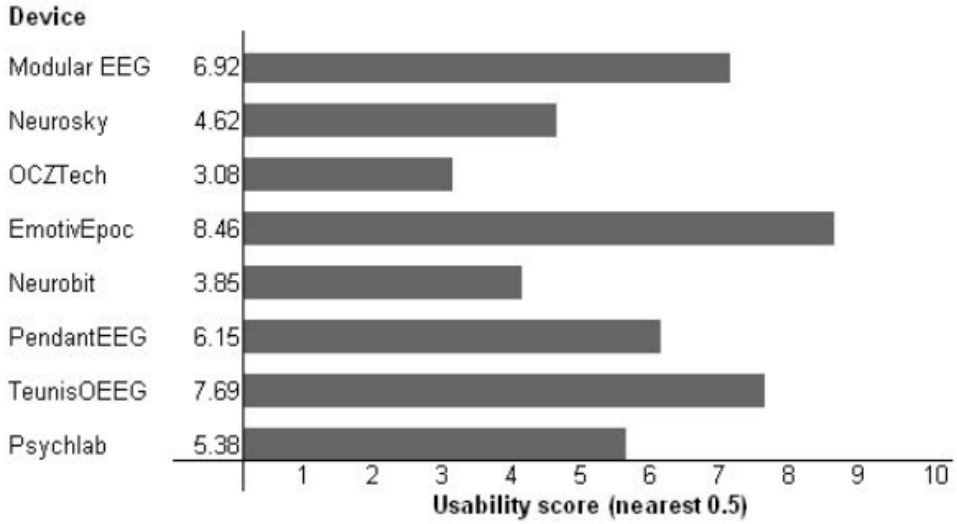
A. Tipos de abordagem tecnológica capaz de medir a atividade cerebral e as suas desvantagens principais na pesquisa ICC [7].

Technology	Primary Disadvantage
Electrocorticogram (ECoG)	Highly invasive, surgery
Magneto-encephalography (MEG)	Extremely expensive
Computed Tomography (CT)	Only anatomical data
Single Photon Emission Computerized Tomography (SPECT)	Radiation exposure
Positron Emission Tomography (PET)	Radiation exposure
Magnetic Resonance Imaging (MRI)	Only anatomical data
Functional Magnetic Resonance Imaging (fMRI)	Extremely expensive
Event-Related Optical Signal / Functional Near-Infrared (EROS/fNIR)	Still in infancy, currently expensive

B. Tabela de Comparação de ICC's *Comerciais (adaptado* [46]*).*

Dispositivos	Preço	Número de elétrodos	Interpretadores de sinais:	Produto
Aurora Dream Headband[47]	$199	1	Frontal *EEG*/*EOG* sensor, open SDK, embedded 4-stage sleep tracking, 2 LEDs near the eyes	iwinks
Emotiv EPOC[48]	$399– 499	14	3 mental states (based on brainwaves), 13 conscious thoughts, facial expressions, head movements (sensed by 2 gyros)	Emotiv Systems
Emotiv Insight[49]	$299	5		Emotiv Lifescience
iFocusBand[50]	$500	1	8 mental states, facial tension, eye movement & quiet eye	iFocusBand
Melon Headband[51]	$149	4	L+R hemisphere pre-frontal *EEG*	Melon
Mindball	$20,000	1	1 mental state	Interactive poductline
Mindlex	$50	1	1 mental state	Mattel(Neurosky
MindSet	$199	1	2 mental states (based on 4 brainwaves), eye-blinks	NeuroSky
MindWave	$99.95	1	2 mental states (based on 4 brainwaves), eye-blinks	NeuroSky
Muse[52]	$299	4	7 sensors; 5 front (2 active, 2 DRL, 1 reference), 2 active behind ears	InteraXon
MyndPlay Brain-Band[53]	$158	1	8 *EEG* bands	MyndPlay
OpenBCI 32bit Board[54]	$499	8	*EEG*, *EMG*, EKG, and accelerometer data	OpenICC
OpenBCI Ganglion Board[55]	$99	4	*EEG*, *EMG*, EKG, and accelerometer data	OpenICC
OpenBCI R&D Kit[56]	$899	16	*EEG*, *EMG*, EKG, and accelerometer data	OpenICC
Star Wars Force Trainer	$45	1	1 mental state	Uncle Milton
XWave headset	$90	1	8 *EEG* bands	PLX Devices

C. Avaliação do conforto feita por utilizadores dos produtos que recolhem dados via *EEG,* disponíveis no mercado [7].



D. Aspetos técnicos do capacete Emotiv EPOC [9].

Sl.no.	Emotiv Neuroheadset	
	Specification key	***Specifications***
1	Number of channels	14 channels with CMS/DRL references)
2	Sampling method	Sequential sampling, Single ADC
3	Channel names	AF3, AF4, F3, F4, F7, F8, FC5, FC6, P3 (CMS), P4 (DRL), P7, P8, T7, T8, O1, O2
4	Sampling rate	~128Hz but internally it is 2048Hz
5	Resolution	16 bits (14 bits effective) 1 LSB = 0.51µV
6	Dynamic range (input referred)	256mVpp
7	Bandwidth	0.2 - 45Hz and digital notch filters at 50Hz and 60Hz
8	Coupling mode	AC coupled
9	Connectivity	Proprietary wireless, 2.4GHz band
10	Impedance measurement	Contact quality using patented system
11	Battery Life	12 hours
12	Battery Type	Li-Poly

E. Manual de Programação de Funções da biblioteca "EPOControl"

Este manual foi elaborado com o intuito de fornecer ao utilizador os passos e informações necessários para poder comunicar com o capacete EPOC e chamar as funções criadas na biblioteca "*EPOControl*".

Função *initEPOC()* - Esta operação é necessariamente chamada no início do programa e está representada no Código 2, sendo executada uma única vez, fazendo a ligação ao EPOC. Executa um conjunto de funções obrigatórias, desde a conexão ao capacete, verificação do seu estado, intervalo de tempo necessário ao software do capacete para poder executar convenientemente e ainda o registo de utilizador.

```
EE_EngineConnect();
EE_GetBaseProfile( eProfile );
CapaceteOn();
load();
loadUser();
```

Código 2: Código fonte da conexão ao Emotiv *EPOC.*

A partir do momento em que é feita a ligação, o *SDK* da Emotiv irá colocar as ações do utilizador num *buffer*. O software tem de aceder a esse buffer regularmente para utilizar a informação do Emotiv EPOC e processar os eventos.

Função *cognitivResetActions()* - Esta operação é evocada quando é necessário enviar ao EPOC a ordem de apagar os treinos gravados do utilizador, descrita no Código 3.

```
EE_EmoEngineEventGetUserId( hEvent, &userID );
EE_CognitivSetTrainingControl( userID, COG_ERASE );
EE_CognitivSetTrainingControl( userID, COG_RESET );
```

Código 3: Código fonte que apaga os treinos gravados.

Função *loadUser()* - Esta operação pretende utilizar uma assinatura para atualizar um utilizador, através das funções representadas no Código 4, recorrendo a um ficheiro que outrora foi gravado.

```
EE_EmoEngineEventGetUserId( hEvent, &userID );
EE_LoadUserProfile( userID, "test.emu" );
```

Código 4: Código fonte que utiliza uma assinatura existente.

Função *saveUser()* - Esta exercício faz percorrer um fluxo controlado de ordens de modo a ser possível criar e reservar uma assinatura, demonstrado no Código 5.

```
EE_GetUserProfileSize( eProfile, &profileSize );
EE_EmoEngineEventGetUserId( hEvent, &userID );
EE_GetUserProfile( userID, eProfile );
result = EE_GetUserProfileBytes( eProfile, profileBuffer, profileSize );
SetUserProfile( userID, profileBuffer, profileSize );
(…)
// case EE_CognitivTrainingCompleted:
      EE_SaveUserProfile( &userID, "test.emu" );
(…)
// stopEPOC(){
      (…)
      EE_SaveUserProfile( &userID, "test.emu" ); }
```

Código 5: Código fonte que grava uma assinatura.

Função *cognitivActivationAllLevel()* - Esta função representada no Código 6 enquanto mostra o valor de sensibilidade dos sinais cognitivos recebe o valor para ativar um novo nível de sensibilidade que irá afetar todos os sinais cognitivos, sensibilidade essa de um a sete.

```
cognitivGetAllLevel();
EE_EmoEngineEventGetUserId( hEvent, &userID );
EE_CognitivSetActivationLevel( userID, level );
```

Código 6: Código fonte que permite afetar a sensibilidade de todos os sinais cognitivos.

Função *cognitivActivation4Level()* - Esta função representada no Código 7 enquanto mostra o valor de sensibilidade dos sinais cognitivos recebe quatro valores para ativar um novo nível de sensibilidade que irá afetar cada um os sinais cognitivos, sensibilidade essa de um a dez.

```
cognitivGet4Level();
EE_EmoEngineEventGetUserId( hEvent, &userID );
EE_CognitivSetActionSensitivity( userID, Level1, Level2, Level3, Level4 );
```

Código 7: Código fonte que permite afetar a sensibilidade de cada um dos quatro sinais cognitivos.

Função *interrupt()* - Esta função, representada no Código 8, permite ao utilizador aceder a várias funções a partir do teclado, nomeadamente fazer *pause* ao programa, afetar as sensibilidades dos sinais personalizados, verificar os níveis de bateria e sinal *wireless* do EPOC e ainda criar expressões ou pensamentos específicos para um utilizador.

```
cognitivGet4Level();
cognitivActivation4Level();
CognitivGetAllLevel();
cognitivActivationAllLevel();
getBattery();
WirelessSignal();
cognitivTraining();
expressivTraining();
```

Código 8: Funções definidas em *interrupt()*.

Função *cognitivTraining()* - Esta função é absolutamente necessária caso o indivíduo queira usar sinais cognitivos. Deve ser chamada no princípio do programa, tendo atenção que por cada registo de treino ter-se-á de comunicar com o capacete várias vezes, acompanhando adequadamente a estrutura modelo demonstrada na Figura 41, para melhor compreender a sequência de chamada.

O treino cognitivo pode ser configurado para reconhecer e distinguir entre um máximo de quatro ações distintas. Por cada treino é essencial transmitir ao capacete as ordens necessárias para se conseguir proceder ao correto registo e aplicar as manipulações apropriadas aos eventos.

Foram incluídos ao longo da sequência do protocolo **atrasos**, antes e depois dos comandos de treino, para que não houvesse lugar a atrasos feitos por parte do utilizador. Com isto evita-se a formação de ruídos indesejáveis nos sinais, resultantes da transição de um estado mental "*neutro*" para outra condição mental desejada.

O diagrama de sequência apresentado na Figura 41 descreve o processo de realização do treino cognitivo para uma ação particular qualquer. É necessário entender que existe um conjunto de eventos possíveis de ser recebidos pelo capacete, como exibe o Figura 42.

O primeiro sinal a ser registado será o *"Neutral"*, que representa o estado apático do utilizador. Este sinal não faz parte do conjunto de sinais que o utilizador pretende encontrar, mas é imprescindível para uma boa leitura de estados.

O Código 9 ilustra o procedimento para extrair informações específicas do evento cognitivo do EPOC.

```
EmoEngineEventHandle hEvent = EE_EmoEngineEventCreate();
int state = EE_EngineGetNextEvent( hEvent );
if ( state == EDK_OK ) {
EE_Event_t eventType = EE_EmoEngineEventGetType( hEvent );
switch ( eventType ) {
(…)
Case EE_CognitivEvent:
      {
      EE_CognitivEvent_t eventType = EE_CognitivEventGetType( hEvent );
            break;
      }
```

Código 9: Pseudo código da receção da resposta aos eventos no *EPOC.*

Antes do início de uma sessão de treino, o tipo de ação deve ser definido com as funções *EE_CognitivSetTrainingAction*() e *EE_CognitivSetTrainingControl()*. Em *EmoStateDLL.h*, o tipo enumerado *EE_CognitivAction_t* define todas as ações cognitivas que são atualmente suportadas (*COG_PUSH, COG_LIFT*, etc.).

EE_CognitivSetTrainingControl(userId, COG_START) pode então ser chamado para iniciar o treino sobre a ação de destino. Em *EDK.h*, o tipo *EE_CognitivTrainingControl_t* define as constantes de comando de controlo do treino.

Se o treino é iniciado, o evento *EE_CognitivTrainingStarted* é enviado do EPOC para a aplicação quase que imediatamente. (nota: O utilizador deve visualizar e/ou imaginar a ação apropriada antes de enviar o comando *COG_START*). A atualização de formação começará após este evento.

Depois de aproximadamente oito segundos de período de treino, dois eventos possíveis são enviados a partir do EPOC para a aplicação.

• *EE_CognitivTrainingSucceeded*: Se a qualidade do sinal *EEG* durante a sessão de treino foi suficientemente boa para atualizar os algoritmos de assinatura treinados, ele entrará num estado de espera para confirmar a atualização do treino.

• *EE_CognitivTrainingFailed*: Se a qualidade dos sinais durante a sessão de treino não forem suficientes para atualizar a assinatura treinada o processo de treino será reiniciado automaticamente, e o utilizador deve ser convidado a iniciar o treino novamente.

Quando a sessão de treino acabar (sinal *EE_CognitivTrainingSucceeded* recebida), o utilizador deve aceitar ou rejeitar a sessão conforme o Código 10 mostra.

```
EE_CognitivSetTrainingControl(userID, COG_ACCEPT);
EE_CognitivSetTrainingControl(userID, COG_REJECT);
```

Código 10: Pseudo código utilizado ao aceitar ou rejeitar um treino cognitivo

O utilizador pode querer rejeitar a sessão de treino, isto se não foi capaz de evocar ou manter um estado mental consistente durante toda a duração do período de treino. A resposta do utilizador é então submetida ao EPOC com o argumento *COG_ACCEPT* ou *COG_REJECT*. Se a formação for rejeitada, então a aplicação deve esperar até que ele recebe o evento *EE_CognitivTrainingRejected* antes de reiniciar o processo de treino. Se a formação for aceite, o capacete irá reconstruir o treino na assinatura do utilizador, e um evento *EE_CognitivTrainingCompleted* será emitido pela aplicação. Note-se que este processo de construção de assinatura pode demorar até vários segundos, dependendo dos recursos do sistema, o número de ações que estão sendo treinados, e o número de sessões de treino gravadas para cada ação.

Função *expressivTraining()* - Esta função é necessária caso o utilizador queira criar novas expressões faciais, diferentes das que foram padronizadas e referidas nos casos de uso com a Figura 35. Deve ser chamada no princípio do programa, funcionando de um modo muito semelhante à função *cognitivTraining()* e deverá seguir adequadamente a estrutura modelo demonstrada na

Figura 40, para que se possa compreender a sequência de chamada das operações necessárias.

No momento que uma aplicação chama esta função prevê-se cumprir uma comunicação com o capacete EPOC muito semelhante à função cognitivTraining*()*, sendo que nesta são evocadas diferentes operações.

Função *newAction()* - Esta é a função responsável por interpretar os eventos do Emotiv EPOC, numa rotina representada no Código 11 onde escuta as respostas enviadas pelo EPOC. Esta função pretende operar em conjunto com o capacete de maneira a identificar os estados correntes do capacete, procurando mediante cada estado executar os eventos definidos.

De modo a que uma aplicação comunique com o capacete, o programa necessita de verificar regularmente os novos eventos e endereça-los de acordo com a sua função. A função começa por consultar o EPOC para obter o evento atual invocando *EE_EngineGetNextEvent(hEvent)*. Usando a função *EE_EmoEngineEventGetType()* observa-se que estado o EPOC regista. Como exibe a Tabela 3 existem vários estados.

Se o resultado obtido usando *EE_EmoEngineEventGetType()* for do tipo de evento *EE_EmoStateUpdated*, então foi detetado um novo evento para um utilizador específico (função *EE_EmoEngineEventGetUserID()*). A função *EE_EmoEngineEventGetEmoState(eEvent,eState)* é então usada para copiar as informações atuais a partir do evento para segurar no *buffer EmoState.* É necessário aceder às informações do buffer em curtos espaços de tempo para que não haja desfasamentos entre os pensamentos do utilizador e o comportamento do software. Note-se que *EE_EngineGetNextEvent()* irá retornar *EDK_NO_EVENT* se os novos eventos foram iguais à chamada anterior. O utilizador também verifica se são retornados outros erros com *EE_EngineGetNextEvent()* para lidar com possíveis problemas que são relatados pelo EPOC.

Caso o tipo de evento for *EE_EmoStateUpdated*, e depois de ser feito EE*_EmoEngineEventGetEmoState(eEvent,eState)* é possível detetar resultados, como estados específicos recuperados a partir desse estado chamando as funções de estado correspondentes definidas em *EmoState.h*. Por exemplo, para

aceder à deteção de olhos fechados, *ES_ExpressivIsBlink(Estate)* deve ser utilizado.

Então para concluir, caso existam eventos a processar, esta função do *SDK* dispara um evento que é recebido pelo software e que por sua vez chama a função que interpreta as ações do utilizador e as converte em ações do software.

```
EmoEngineEventHandle eEvent = EE_EmoEngineEventCreate();
EmoStateHandle eState= EE_EmoStateCreate();
unsigned int userID= 0;
while (...) {
int state = EE_EngineGetNextEvent( eEvent );
if (state == EDK_OK) {
      EE_Event_t eventType = EE_EmoEngineEventGetType( eEvent );
      EE_EmoEngineEventGetUserId( eEvent, &userID );
      if ( eventType == EE_EmoStateUpdated ) {
        EE_EmoEngineEventGetEmoState( eEvent, eState );
        logEmoState( userID, eState, writeHeader );
        writeHeader = false;
      }
   }
}
```

Código 11: Pseudo código da estrutura da função *newAction()* que processa os eventos.

Função *shutdown()* - Esta função é evocada antes do final do programa, onde *EE_EngineDisconnect()* é chamado para terminar a ligação com o EPOC e liberar recursos associados a essa ligação. O utilizador também deve chamar *EE_EmoStateFree()* e *EE_EmoEngineEventFree()* para libertar a memória alocada no buffer *EmoState* e *EmoEngineEventHandle*. No caso de haver uma comunicação *socket* com o software esta função também irá permitir fazer *closeSocket()* de modo a que a aplicação que esteja à espera de mensagens possa não ter que ficar bloqueada.

Função *initSocket()* - Esta função deve ser declarada no início do programa e tem como objetivo criar um canal *socket* pronto a escutar mensagens através de um endereço na própria máquina, como demonstra o Código 12.

```
error = WSAStartup( version, &wsa );
cliente = socket( AF_INET, SOCK_STREAM, 0 );
server.sin_addr.s_addr = inet_addr( IP );
server.sin_family = AF_INET;
server.sin_port = htons( PORT );
if (connect( cliente, ( struct sockaddr *) &server, sizeof(server)) < 0){
      perror( "connect failed. Error\n" );
      getchar();
      system( "CLS" );
      }
else{
      printf("Cliente Connected...\n");
       }
```

Código 12: *Pseudo código* na inicialização do canal *socket*.

Função *Send()* - Operação que permite enviar uma mensagem pelo canal, contendo como argumentos o nome do canal, a mensagem e o tamanho de mensagem.

```
send(cliente, message, strlen(message), 0);
```

Código 13: Função que envia mensagens através de um endereço.

F. Manual de Programação de Funções da aplicação "DronEpoc"

Include – o conjunto de alíneas apresentadas no Código 14 deve ser colocado no início do ficheiro *net_io.cpp*. Elas representam as bibliotecas necessárias ao programa. Qualquer função evocada depois terá origem nestas classes.

```
#include <WinSock2.h>
#include <stdlib.h>
#include "net_types.h"
#include <iostream>
#include "Controlador.h"
#include "edk.h"
#include "EmoStateDLL.h"
#include <stdio.h>
#include <conio.h>
#include <tchar.h>
#include <windows.h>
#include "clienteSocket.h"
```

Código 14: *Includes* a fazer no ficheiro *net_io.cpp*.

Função *InitializeIO()* - É executada apenas uma vez quando o programa é inicializado, de modo a, neste caso em particular e como representa no Código 15, inicializar e configurar o capacete. É necessário também reproduzir a operação *socket,* onde principia a comunicação com a aplicação "*ARDrone*" através de um endereço *IP (Internet Protocol)*.

```
InitEPOC();
initSocket();
```

Código 15: Chamada de Funções de inicialização.

Função *GetInputSignals()* - É executado no início de cada etapa de execução do ciclo, para ler o valor atual de sinais de entrada, que no nosso caso do estão representados no Código 16.

```
// blink(), Winkleft(), lookLeft(), Winkright(), lookRight(),smile()
// PUSH, PULL, LIFT, DROP, LEFT, RIGHT, ROTATE_LEFT, ROTATE_RIGHT,
// ROTATE_CLOCKWISE, ROTATE_COUNTER_CLOCKWISE, ROTATE_FORWARDS,
// ROTATE_REVERSE, DISAPPEAR
interrupt();
newAction();
inputs->In_Y = UpdateY();
inputs->In_X = UpdateX();
inputs->In_Signal1 = lookLeft();
inputs->In_Signal2 = lookRight();
inputs->In_Signal3 = WinkRight();
inputs->In_Signal4 = LIFT;
inputs->In_Signal5 = DROP;
inputs->In_Signal6 = WinkLeft();
inputs->In_Signal7 = blink();
```

Código 16: Chamada de funções de entrada.

Foi adicionado a função *interrupt(),* para que o utilizador consegue parar o programa ou mesmo chamar uma instrução através do teclado. A instrução *newAction()* procura saber se em cada ciclo foi registado algum evento. Por fim cada entrada da rede é igualada ao tipo de sinal que pretendemos.

Função *PutOutputSignals()* - É chamada antes e depois de terminar a etapa de execução, para atualizar todos os sinais de saída. Conforme apresenta a Código 17 a função *printf()* imprime na consola o comportamento dos estados do capacete e da *RdP*.

```
/****************************** OUTPUT ******************************/
printf("\nT/L: %d|Up: %d|Down: %d|Left: %d|Right: %d|Rot.L: %d|Rot.R: %d|
Forward: %d|Back: %d|", place_out->Out_TakeOffLand, place_out->Out_Up,
place_out->Out_Down, place_out->Out_Left, place_out->Out_Right,
place_out->Out_RotateLeft, place_out->Out_RotateRight,
place_out->Out_Forward, place_out->Out_Backward);
/****************************** SOCKETS ******************************/
if (place_out->Out_Up == 1) message[0] = '1';
       else message[0] = '0'; //comando I
if (place_out->Out_Down == 1) message[1] = '1';
       else message[1] = '0'; //comando K
if (place_out->Out_Left == 1) message[2] = '1';
       else message[2] = '0'; //comando A
if (place_out->Out_Right == 1) message[3] = '1';
       else message[3] = '0'; // comando D
if (place_out->Out_TakeOffLand == 1) message[4] = '1';
       else message[4] = '0'; // comando Space
if (place_out->Out_RotateLeft == 1) message[5] = '1';
       else message[5] = '0'; // comando J
if (place_out->Out_RotateRight == 1) message[6] = '1';
       else message[6] = '0'; // comando L
if (place_out->Out_Forward == 1) message[7] = '1';
       else message[7] = '0'; // comando W
if (place_out->Out_Backward == 1) message[8] = '1';
       else message[8] = '0'; // comando S
if (place_out->Out_TakePhoto == 1) message[9] = '1';
       else message[9] = '0'; // Take Photo
send( cliente, message, strlen(message), 0 );
```

Código 17: Chamada das funções de saída.

Função *LoopDelay()* – Função que faz cada ciclo da rede demorar um tempo definido pelo utilizador. Devido à comunicação com o capacete foi proposto implementar um *delay* conforme o Código 18 para que não surgisse problemas dessa comunicação:

```
Sleep(15);// Sem Drone -> sleep 5 || Com drone -> sleep 15
```

Código 18: Função que define o tempo entre cada ciclo.

Função *FinishExecution()* - No final de cada ciclo a função do Código 19 é chamada, e caso do utilizador quiser desligar a aplicação esta irá desligar conforme o procedimento que a arquitetura do EPOC deseja, gravando as capacidades do utilizador atual gerando um ficheiro com toda a informação necessária.

```
return shutdown(); //desligar o programa com a tecla "END"
```

Código 19: Função evocada ao sair do ciclo da rede.

G. Manual de Programação de Funções da aplicação "ARDrone"

Classe *KeyboardInput:*

```
protected override InputMapping GetStandardMapping()
{
ButtonBasedInputMapping mapping = new
ButtonBasedInputMapping(GetValidButtons(), GetValidAxes());

mapping.SetAxisMappings("A-D", "W-S", "J-L", "K-I");
mapping.SetButtonMappings("Space", "Space","F", "E");

return mapping;
}
```

Código 20: Código fonte das teclas *standard* utilizadas no controlo do *Drone.*

```
AsynchronousSocketListener.StartListening();
if (mensagem.CharMessageFromClient[0] == '1')buttonsPressed.Add("I");
if (mensagem.CharMessageFromClient[1] == '1')buttonsPressed.Add("K");
if (mensagem.CharMessageFromClient[2] == '1')buttonsPressed.Add("A");
if (mensagem.CharMessageFromClient[3] == '1')buttonsPressed.Add("D");
if (mensagem.CharMessageFromClient[4] == '1')buttonsPressed.Add("Space");
if (mensagem.CharMessageFromClient[5] == '1')buttonsPressed.Add("J");
if (mensagem.CharMessageFromClient[6] == '1')buttonsPressed.Add("L");
if (mensagem.CharMessageFromClient[7] == '1') buttonsPressed.Add("W");
if (mensagem.CharMessageFromClient[8] == '1') buttonsPressed.Add("S");
if (mensagem.CharMessageFromClient[9] == '1') recorder.takePhoto();
```

Código 21: Código fonte dos comandos enviados pela aplicação *"DronEpoc"* utilizados na aplicação *"ARDrone"* para controlar do *Drone.*

Classe *ClassPipe* - Esta classe cria um servidor para um canal corrente denominado *Pipe*. Este canal pretende receber de um cliente mensagens, enquanto este servidor acede a essas mensagens por ordem de chegada, através do método *First-In-Last-Out (FILO)*. No caso deste projeto não é aconselhável usar este tipo de abordagem apesar de ter sido integrada, isto porque esta abordagem foi considerada muito antes da abordagem aos *Socket*.

Classe *AsynchronousSocketListener* :

```
IPHostEntry ipHostInfo = Dns.Resolve(Dns.GetHostName());
IPAddress ipAddress = ipHostInfo.AddressList[0];
IPEndPoint localEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
8888);
// Create a TCP/IP socket.
Socket listener = new Socket(AddressFamily.InterNetwork, Socket-
Type.Stream, ProtocolType.Tcp);
// Bind the socket to the local endpoint and listen for incoming connec-
tions.
try
{
     listener.Bind(localEndPoint);
     listener.Listen(1);
     bool a = true;
     while (a)
     {
         // Set the event to nonsignaled state.
         allDone.Reset();
         // Start an asynchronous socket to listen for connections.
         Console.WriteLine("Waiting for a connection...");
         listener.BeginAccept(
             new AsyncCallback(AcceptCallback),
             listener);
         // Wait until a connection is made before continuing.
         a = !allDone.WaitOne();
     }
```

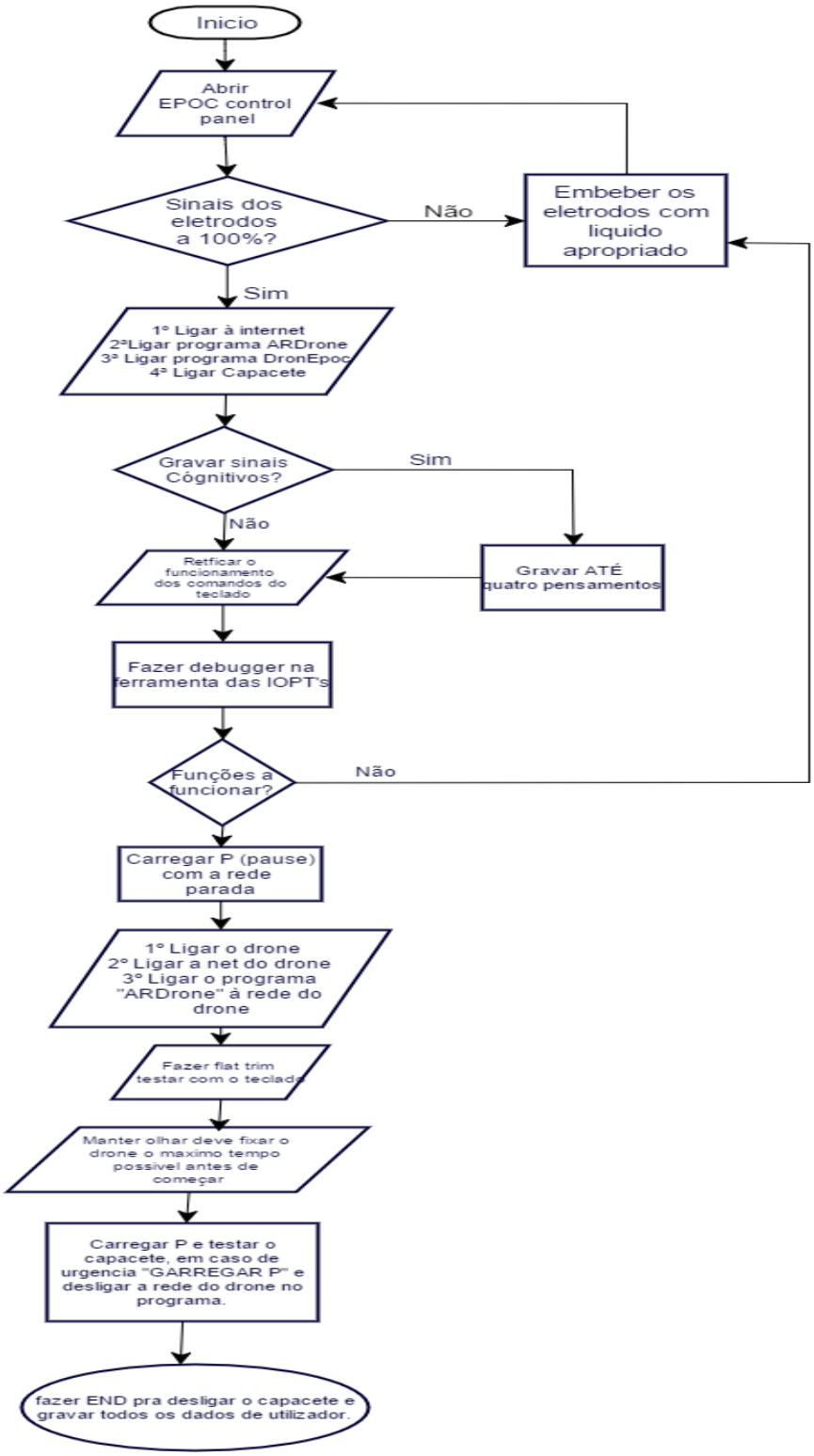
Código 22: Pseudo código da classe *AsynchronousSocketListener*.

Classe *mensagem* - Criada apenas para reservar uma *string* de carateres representada no Código 23, é utilizada para alojar as mensagens recebidas pelo *socket*.

```
public class mensagem
{
       public static char[] CharMessageFromClient = new char[10];
}
```

Código 23: Código fonte da variável que recebe a mensagem passada pelo cliente (“*DronEPOC*”).

H. Fluxo de procedimentos do utilizador para realização de experiência.



I. Algoritmo de decisão de eventos.