FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Sistema de Localização Indoor para o robô telemóvel "Robobo"

João Luís Vila Chã Torres

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Professor Doutor Armando Jorge Miranda de Sousa

26 de Novembro de 2018

Resumo

A robótica é, hoje em dia, uma área em grande evolução. Contudo, existe uma diferenciação muito grande na oferta. A maioria dos robôs preparados para crianças acabam por não ser simultaneamente adequados para jovens e adultos. Adicionalmente, os robôs utilizados para ensino superior e investigação são demasiado complexos para serem manipulados por idades mais jovens.

O robô denominado *Robobo* foi desenvolvido com o objetivo de mudar o panorama apresentado, permitindo a utilização em ambiente de ensino superior, mas disponibilizando também modos acessíveis aos mais jovens. Assim, a principal característica deste robô é a diversidade de opções disponíveis para programação e utilização, que atravessam todo o espectro de competências. O *Robobo* tem duas partes complementares, uma base móvel designada de *ROB* e um *smartphone* Android, denominado de *OBO*. Estes dois componentes dotam o sistema de características interessantes e até superiores a alguns robôs educacionais disponíveis no mercado.

Com o sistema *Robobo* o utilizador contacta com praticamente todas as vertentes da robótica, desde a sensorização até à imagem. Porém, à altura da escrita deste documento não existe uma solução de localização para o *Robobo*. Com a massificação da automatização de processos e dispositivos a que se assiste na atualidade esta lacuna revela-se limitante da autonomia de um robô que se pretende poder vir a ter comportamentos desafiantes.

A presente dissertação pretende colmatar a lacuna da localização no ambiente *Robobo*, com um sistema interno e automático baseado em visão e utilizando marcadores planares *Aruco*, tão pouco intrusivos quanto possível.

O sistema de localização desenvolvido está integrado no software Android do OBO e foi desenvolvido em Java e C++. Para facilidade de utilização futura, o sistema de localização desenvolvido pode posteriormente ser utilizado como uma biblioteca por outra aplicação para comando do Robô. Todo este processo assenta numa *framework*, a *ROBOBO Framework*, sendo a mesma responsável por todas as interações entre as duas partes complementares do *ROBOBO*.

De acordo com os testes realizados, o sistema desenvolvido permite obter uma localização com um erro inferior a 0.1 m, com um erro máximo angular de cerca de 3 graus para visualizações de marcadores aruco verticais (numa parede) de cerca de 10 cm a uma distância de cerca de 2 metros. Da mesma forma, a localização com base nos mesmos marcadores na horizontal (no chão) resulta em erros da ordem dos 2.5 cm e 3 graus para uma distância de 90cm.

É ainda proposta alguma caracterização da qualidade da localização a nível da quantidade de localizações com sucesso - aproximadamente todas as frames capturadas a menos de 2 metros produzem localização. É ainda apresentada uma proposta de modelo para o desvio padrão das medidas de localização.

Abstract

Robotics is now a rapidly evolving area. However, there is a very large differentiation in the existing offer. Most child-friendly robots are not suitable for young people and adults. At the same time, robots used for higher education and research are extremely complex to be manipulated by younger ages.

The Robobo robot was developed with the aim of changing the panorama presented, allowing the use in a higher education environment, but also providing modes accessible to the young. Thus, the main characteristic of this robot is the diversity of options available for programming and use, which cross the entire spectrum of skills. The *Robobo* has two complementary parts, a mobile base called *ROB* and an Android smartphone called *OBO*. These two components form a system with interesting features, even superior to a large portion of the educational robots available in the market.

With the *Robobo* system the user contacts virtually every aspect of robotics, from sensor to image. However, at the time of writing, there is no localization solution for *Robobo*. With the massification of the automation of processes and devices that is currently observed, this gap is a limitation of autonomy.

The present dissertation intends to bridge the location gap in the *Robobo* environment, with an internal, automatic, vision-based system using Aruco planar markers.

The localization system developed is integrated into OBO's Android software and was developed in Java and C ++. For ease of use in the future, the developed localization system can later be used as a library by another application to command the Robot. This whole process is based on a *framework*, the *ROBOBO Framework*, and it is responsible for all the interactions between the two complementary parts of *ROBOBO*.

According to the tests performed, the developed system allows for a location with an error of less than 0.1 m with a maximum angular error of about 3 degrees for views of vertical ARUCO markers of about 10 cm at a distance of about 2 meters. Likewise, location based on the same markers on horizontals results in errors of the order of 2.5 cm and 3 degrees at a distance of 90cm.

This work also includes the number of successful localizations per time and a tentative model for the quality of the localization, the standard deviation for the localization.

Agradecimentos

Este documento é dedicado aos meus pais, por toda a dedicação e preocupação que sempre tiveram comigo. Por toda a liberdade que me deram para escolher o meu ainda curto caminho. Por me terem transmitido todos os valores que fazem de mim aquilo que sou hoje.

Aos meus irmãos, por me distraírem em todas as oportunidades e todos os dias me transmitirem a felicidade que só a inocência da infância é capaz.

À Carolina, pela paciência ao longo destes meses. Por estar sempre pronta a apoiar-me. Por me acalmar nos momentos em que sentia que estava tudo perdido e por me alertar a seguir, quando a pressão aliviava e acabava por relaxar.

Aos meus amigos, por fazerem das nossas saídas um momento em que me esquecia de todo o trabalho e por não falarem no tema proibido.

Por fim, ao meu orientador, Professor Armando Sousa, por toda a ajuda, tempo e orientação ao longo deste projeto.

João Torres

"To the optimist, the glass is half full.
To the pessimist, the glass is half empty.
To the engineer, the glass is twice as big as it needs to be"

Unknown

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ROS	Robot Operating System
API	Application Program Interface (Interface de Programação de Aplicações)
RSS	Received Radio Signal Strength
AOA	Angle Of Arrival (Ângulo de Chegada)
LED	Light-Emitting Diode (Díodo Emissor de Luz)
IR	Infrared Radiation (Radiação Infravermelha)
GPS	Global Positioning System (Sistema de Posicionamento Global)
DC	Direct Current (Corrente Contínua)
USB	Universal Serial Bus
TOF	Time Of Flight
IDE	Integrated Development Environment (Ambiente de Desenvolvimento Integrado)
JNI	Java Native Interface

Capítulo 1

Introdução

1.1 Enquadramento

A Robótica é um tópico central na educação STEM, principalmente pelas suas característica multidisciplinares e práticas. Um projeto de robótica envolve normalmente, conhecimentos de mecânica, eletrónica e programação. Estes estão por sua vez dependentes de capacidades técnicas de base, como a matemática e física. O objetivo de um sistema robótico é resolver problemas concretos e reais. O facto de se guiar por objetivos práticos é favorável à aprendizagem, despertando nos alunos índices de atenção e motivação superiores, que se traduzem em ganhos ao nível do conhecimento adquirido.

O *Robobo* [1] [2] é um robô móvel desenvolvido por investigadores do MINT [3], na Universidade da Coruña.

É composto por duas partes: a base móvel, e o *smartphone*.

A base dota o conjunto de mobilidade, quer a nível de deslocação, quer de orientação do *smartphone*, uma vez que existe uma unidade *pan & tilt*, que permite ângulos próximos de 180º e 90º para a horizontal e vertical, respetivamente. Existem também luzes *LED* na base, assim como sensores *IR* para medição de distância a obstáculos e distância ao solo.

O *smartphone*, para além de ser a unidade de processamento, contém inúmeros sensores que podem ser utilizados, nomeadamente as câmaras, os sensores de proximidade, luminosidade e temperatura, giroscópio, acelerómetro e magnetómetro.

A nível de programação é uma plataforma extremamente completa uma vez que contempla todos os níveis de competências.

Para a iniciação, é possível a programação por blocos via *Scratch* ou *Blockly*.

A plataforma é também compatível com *ROS* podendo ser programada como um nó ROS remoto.

Contudo, os programas criados utilizando os métodos anteriores correm num computador remoto.

De forma a correr os programas criados de forma independente no robô está disponível uma terceira opção, denominada "programação nativa". Este método baseia-se na *Robobo Framework*,

uma *framework* de desenvolvimento de robótica desenvolvida especificamente para o *Robobo*. Utilizando esta *framework* em conjunto com linguagem Java e a *API Android* é possível programas nativos para o *Robobo*, adicionando assim funcionalidades ao robô que podem correr de forma autónoma, sem necessidade de outro dispositivo.

Todos estes fatores fazem com que a plataforma *Robobo* seja bastante interessante para ser utilizada em educação, nomeadamente na iniciação de robótica.

1.2 Objetivos e Motivação

Neste momento, uma das principais lacunas do *Robobo* prende-se com o sistema de localização.

É possível utilizar os sensores disponíveis no *smartphone*, contudo estes revelam-se ineficazes na grande maioria dos cenários de utilização da plataforma. Por exemplo, os módulos de *GPS* encontrados nos *smartphones* não permitem a localização no interior de edifícios, sendo este o cenário de utilização mais comum para o *Robobo*.

Assim, e tendo em conta que a localização é um dos mais importantes desafios da robótica, esta lacuna revela-se importante, afastando alguns utilizadores de uma parte importante da realidade da área.

A dissertação realizada tem como principal objetivo dotar o *Robobo* de um sistema de localização *indoor*. Pretende-se que o sistema a desenvolver cumpra alguns requisitos, nomeadamente: seja barato, fácil de implementar e fiável.

Como objetivo secundário desta dissertação pretende-se validar o sistema configurando o *Robobo* para se movimentar dentro de uma arena.

1.3 Estrutura do Documento

Para além da introdução, esta dissertação contém mais 5 capítulos. No capítulo 2 é descrito o estado da arte de componentes de interesse para a realização da dissertação, tal como robôs para educação e técnicas utilizadas para localização *indoor* de robôs móveis. No capítulo 3 são apresentadas as decisões que foram tomadas ao nível de projeto do sistema desenvolvido, as suas componentes e a interação entre as mesmas. No capítulo 4 são descritas as ferramentas utilizadas, a arquitetura implementada e os algoritmos utilizados no desenvolvimento do sistema. No capítulo 5 é apresentado o sistema de localização no seu estado final, casos de estudo e discutidos os resultados obtidos. No capítulo 6 são apresentadas as conclusões de todo o trabalho, as contribuições do mesmo e o trabalho futuro que poderá ser desenvolvido no âmbito da presente dissertação.

Capítulo 2

Revisão Bibliográfica e Conceitos Fundamentais

Neste capítulo é apresentado o estado da arte de diversas componentes de interesse para a presente dissertação. Para tal, são estudadas as diferentes opções de robôs para educação existentes no mercado. De seguida é descrito o robô em torno do qual assenta o trabalho e, por último são estudados diferentes métodos de localização *indoor* de robôs móveis.

2.1 Robôs Educacionais

A tabela 2.1 mostras as características de um conjunto representativo de sistemas robóticos utilizados para educação.

Tabela 2.1: Comparativo de características de robôs educacionais. Adaptado de [1]

	NAO	**LEGO EV3**	**DASH & DOT**	**MBOT RANGER**	**THYMIO II**
CPU	Atom Z530 1.6 Ghz Cpu	Arm926ej-S Core@300 Mhz	Arm Cortex-M0	Arduino Mega 2560	PIC24 32 MHz
Sensor de Distância	Sonar	Sonar IR	IR	Sonar	IR
Módulo Inercial	Sim	Acelerómetro	Não	Acelerómetro	Não
Som	Coluna Microfone	Coluna	Coluna Microfone	Buzzer Microfone	Coluna Microfone
Câmara	Sim (1280x960)	Não	Não	Não	Não
Sensor tátil / Ecrã	Sim (head)	Não	Não	Não	Não
Comunicações	WIFI USB	Bluetooth USB	Bluetooth	Bluetooth USB	WIFI USB
Preço (euros)	6.000	350	230	170	130

Como se percebe, pela análise da tabela, a maioria dos sistemas utiliza sensores simples (maioritariamente Ultrasons e Infravermelhos), plataformas de comunicação básicas e um poder computacional reduzido. A escolha deste tipo de hardware básico, aquando do desenvolvimento, pode ser explicada por razões monetárias, mas não só. Alguns deste sistemas foram desenhados para

serem configuráveis, com elementos simples que possam ser facilmente modificados e até substituídos em caso de avaria. Contudo, a simplicidade de hardware diminui o nível de aplicabilidade em problemas reais e interessantes para os alunos. No passado, talvez isto não fosse um problema, mas os alunos interagem diariamente com sistemas robóticos e apercebem-se que os sistemas utilizados para educação estão afastados da realidade e atrasados em termos de funcionalidades e aplicações possíveis.

Neste contexto, o ROBOBO foi desenvolvido com o objetivo de ser uma alternativa aos sistemas existentes para educação.

2.2 *Robobo*

Figura 2.1: Base móvel *Robobo* e *smartphone*. Adaptado de [1]

O *Robobo* é um robô educacional composto por duas partes complementares: uma base móvel e um smartphone conectado à base. Esta integração dota o conjunto de algumas características fundamentais:

- Mobilidade. Pela utilização da base móvel
- Baixo custo. Comparativamente à oferta existente para robôs móveis educacionais com características semelhantes
- *Updates* contínuos. A tecnologia dos *smartphones* está em constante evolução, como tal o *Robobo* pode ser melhorado através de um *upgrade* no *smartphone*, tanto a nível de *software* como a nível de *hardware* através da substituição por outro com características melhoradas.

- Tecnologia avançada. Para além dos sensores e atuadores presentes na base, estão também disponíveis todas as tecnologias presentes num *smartphone* típico.

A base móvel do Robobo pode ser adquirida por um valor a rondar os 450 €, para uma única base. Sendo também necessário um smartphone para complementar a mesma, de forma a ser possível utilizar o sistema.

Na figura 2.1 é apresentada a base móvel do sistema *Robobo* acompanhada de um *smartphone*.

2.2.1 Hardware

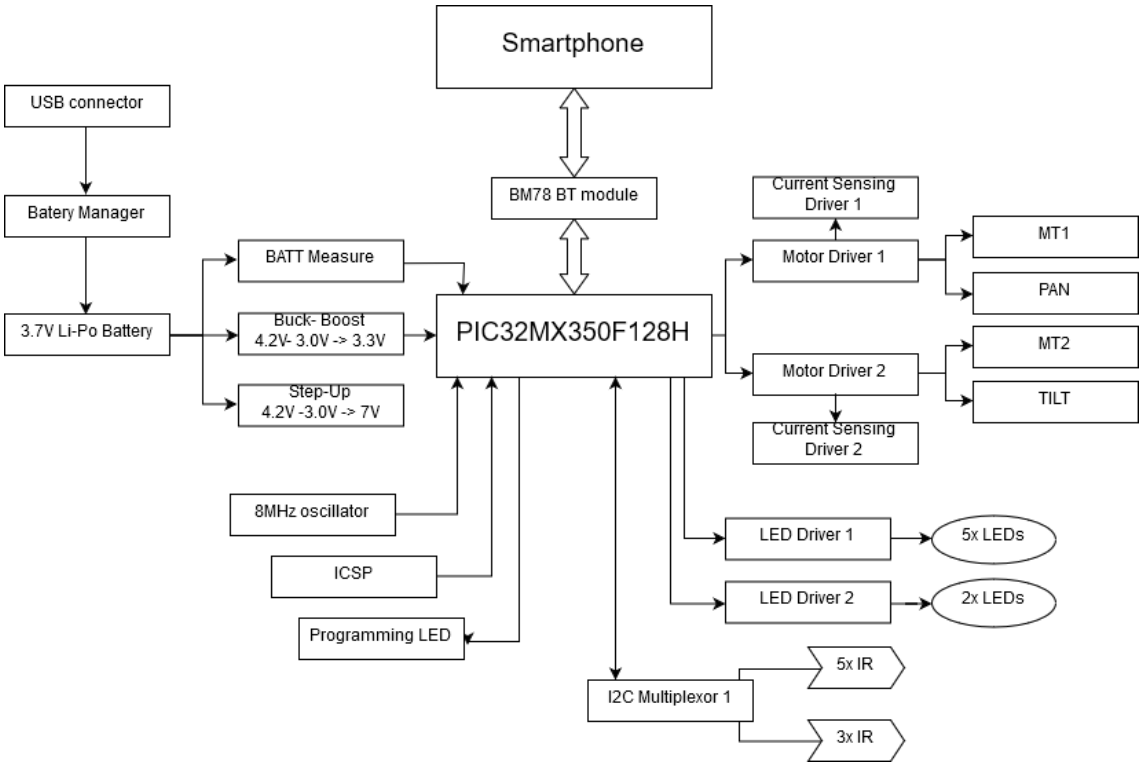

Figura 2.2: Arquitetura Eletrónica do *Robobo*. Adaptado de [1]

Na figura 2.2 é apresentada a arquitetura de hardware do *Robobo* . Os dois componentes principais do *Robobo* são a base móvel (ROB) e o *smartphone* (OBO). São suportados todos os *smartphones* que cumpram os seguintes requisitos:

- Sistema Operativo *Android* 5.1 ou superior
- Ecrã com dimensões entre 4.7in e 5.5in
- Espessura máxima de 95mm

A inclusão do *smartphone* no conjunto permite a utilização do hardware indicado na Tabela 2.2:

Tabela 2.2: Características concedidas ao *Robobo* pelo *smartphone*

Sensor de distância	Sim
Módulo inercial	Acelerómetro Giroscópio Magnetómetro
Som	Altifalantes Microfone
Câmara	2 de alta resolução
Sensor Tátil	Sim
Ecrã	Sim, de alta resolução
Comunicações	4G Wi-fi USB Bluetooth

Os componentes fundamentais da base são:

- 2 motores DC com redução mecânica nas rodas (75:1 com *encoder*)
- 2 *hobby servo* DC para *pan and tilt* (75:1 e 1000:1 respetivamente)
- Micro-controlador PIC32 responsável por controlar as rotinas dos sensores e atuadores da base e comunicação com plataforma
- Módulo *Bluetooth* para comunicação entre a base e o *smartphone*
- 9 *LEDs RGB*
- 9 sensores infravermelhos para medição de distâncias ao solo e a objetos

Na figura 2.3 está indicada a localização dos sensores e atuadores apresentados anteriormente.

2.2.2 Software

A arquitetura de *software* do *Robobo* assenta numa biblioteca chamada *Robobo Framework* que contém os mecanismos necessários para correr os módulos no *smartphone Android* assim como a comunicação entre a base e o *smartphone*.

O *Robobo* pode ser programado com diferentes linguagens, dependendo do nível de competências, existindo três opções:

- Programação por blocos
- Programação ROS
- Programação Nativa

De seguida apresentam-se as características destas possibilidades.

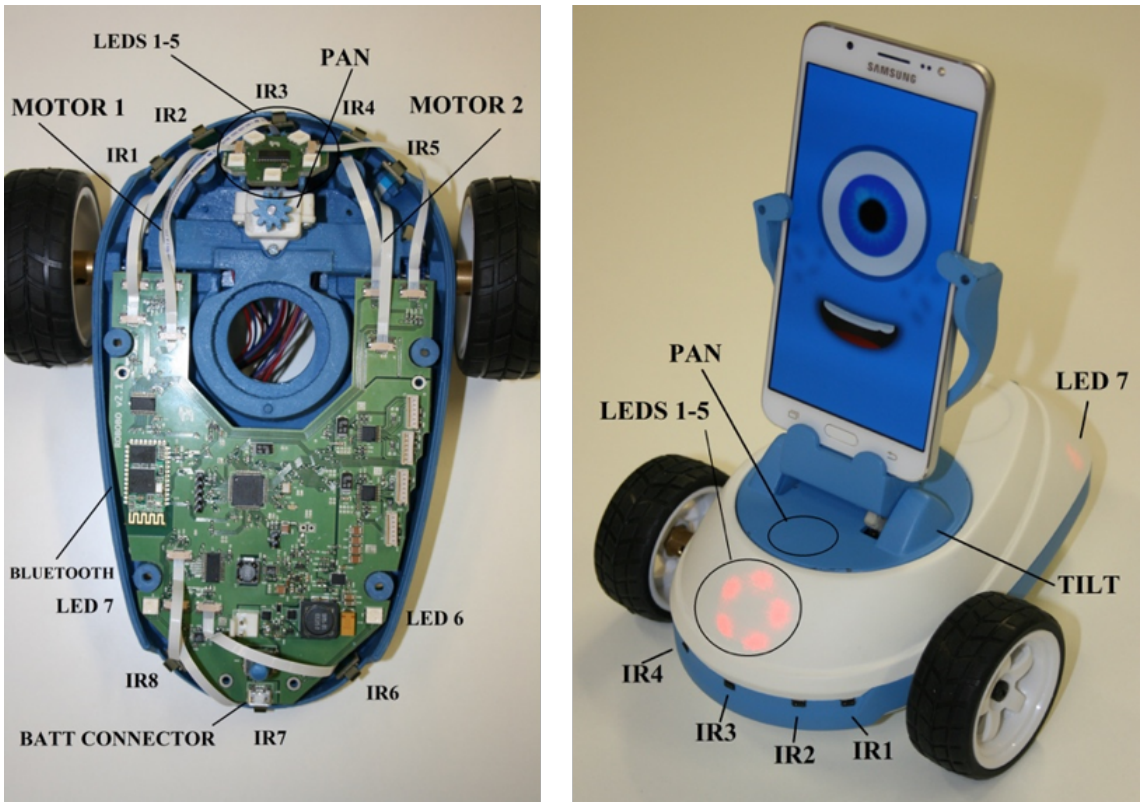

Figura 2.3: Sensores e Atuadores presentes na base Rob. Adaptado de [1]

2.2.2.1 Programação por Blocos

Este é o método de programação mais simples e recomendado para utilizadores sem experiência de programação. Oferece ao utilizador a possibilidade de escrever programas através da junção de peças de *puzzle* que representam diferentes operações. É baseado na ferramenta *ScratchX* que permite controlar o comportamento do *Robobo* usando blocos simples, que podem ser conectados de forma a criar comportamentos mais complexos [4].

Para que se possa controlar o sistema pelo computador é necessária a instalação prévia da aplicação Robobo disponível para Android. Neste caso a comunicação entre o computador e o *smartphone* é efetuada pela Internet. Na figura 2.5 é apresentada a interface de programação *ScratchX*.

2.2.2.2 Programação ROS

O sistema *Robobo* é compatível com ROS [5], através da aplicação *Android Robobo Developer*. Após a instalação desta aplicação no *smartphone* o sistema passa a funcionar como um nó, que pode ser utilizado por outros nós *ROS*.

Contudo, é necessário que o programa esteja a correr num dispositivo externo (pc, Rpi, p.e.) que vai comunicar com o *smartphone*.

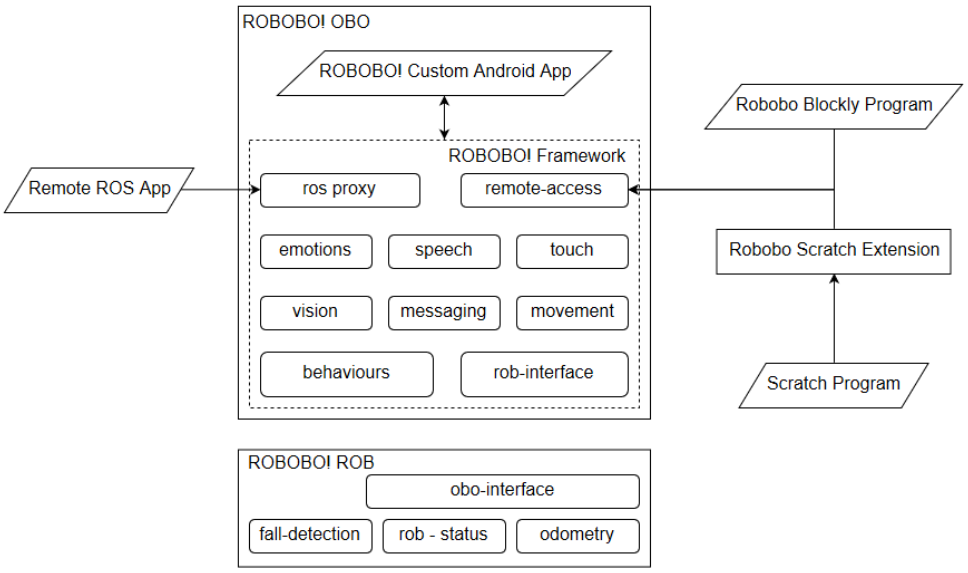

Figura 2.4: Arquitetura de Software do *Robobo*. Adaptado de [1]

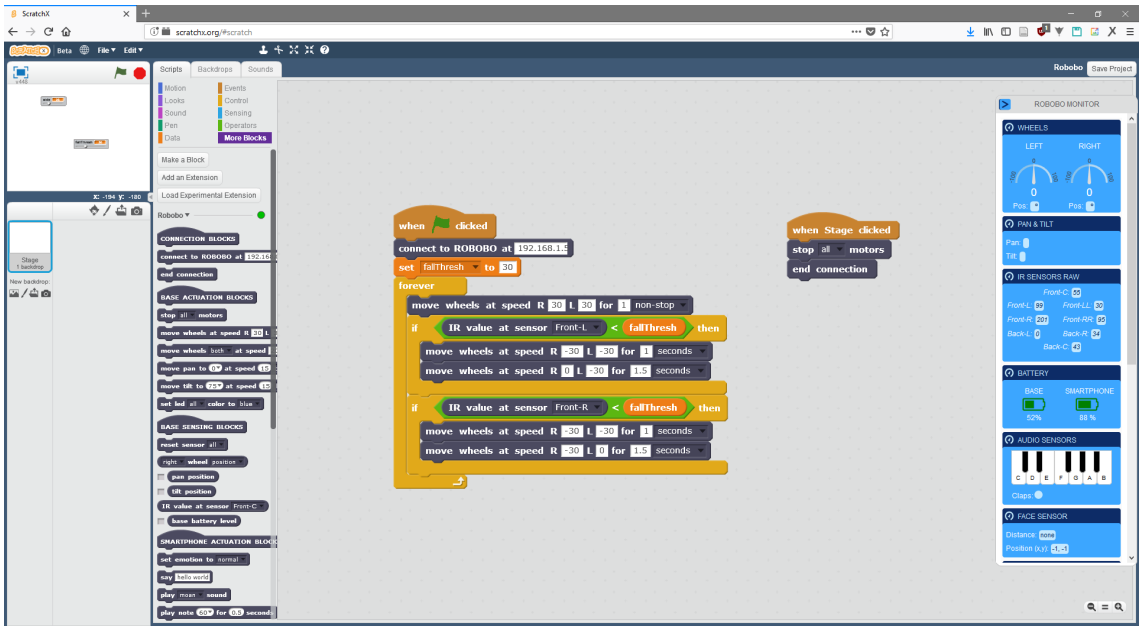

Figura 2.5: Exemplo da interface de programação *ScratchX*

De forma a normalizar e simplificar esta comunicação existe um pacote *ROS* que especifica o formato de mensagens a utilizar. Estão disponíveis dois tipos de mensagens: ***status*** e ***commands***. Os *status* são utilizados para o robô publicar informação acerca do estado dos seus sensores num

determinado momento. Os *commands* servem para enviar ordens ao robô, como movimento de rodas, variação dos ângulos do *pan & tilt* [6].

2.2.2.3 Programação Nativa

O *Robobo* tem como base a *Robobo Framework*, desenvolvida especificamente para este robô e construída sobre o sistema operativo Android . Esta *framework*, também é responsável pelos modos de programação anteriormente referidos, embora de forma menos percetível para o utilizador[7]. Essa interação pode ser identificada na Figura 2.4. Utilizando esta *framework* em conjunto com linguagem Java e Android é possível criar programas nativos para o robô.

Os programas nativos, ao contrário daqueles desenvolvidos em *Scratch* e *ROS*, permitem adicionar comportamentos e funcionalidades ao robô. Funcionalidades essas que podem ser executas autonomamente, sem necessidade de hardware externo (computador, etc.).

Existem dois tipos de programas nativos que podem ser criados utilizando a *Robobo Framework*: *Robobo Apps* e *Robobo Modules*.

As *Robobo Apps* são a forma mais simples de programar nativamente o robô. Permite adicionar comportamentos, denominados de *Behaviour Modules*, ao *Robobo*. Estes comportamentos podem ser, por exemplos, jogos. Contudo, a criação destes comportamentos está limitadas às funcionalidades existentes na *framework*.

Para adicionar funcionalidades ao robô é necessário desenvolver um novo *Robobo Module*. Estes módulos, podem posteriormente ser utilizados por outros programadores, integrando-os nas suas Robobo Apps ou mesmo novos módulos.

2.3 Localização *Indoor* de robôs móveis

Existem inúmeros métodos para localizar robôs móveis em espaços interiores. Contudo, e tendo em conta as capacidades limitadas do *Robobo* a nível de sensorização apenas é interessante analisar técnicas que sejam: baratas, simples e em que as necessidade de sensorização não ultrapassem o disponível num *smartphone*.

Na base de todos os métodos de localização está a medição em tempo-real de um ou mais parâmetros, como ângulos ou distâncias [8]. Através desta medição é obtida a localização do objeto, relativamente a um ou mais pontos fixos, cuja localização é conhecida. Estas medições são baseadas nas propriedades físicas de sinais eletromagnéticos ou ultrassónicos, como a velocidade ou atenuação. A posição do objeto é então calculada utilizando as medições efetuadas e a localização das posições fixas conhecidas. Existem quatro técnicas e métodos principais para calculo e estimação de localizações *indoor*.

- As técnicas baseadas em alcance utilizam geometria para estimar a posição do objeto relativamente a transmissores fixos. Uma vez que a velocidade de propagação de sinais eletromagnéticos e ultrasónicos é conhecida, a diferença de tempo entre o envio e receção de

um sinal permite calcular a distância entre o recetor e o transmissor. O conhecimento da distância a três ou mais sensores permite obter a posição do objeto.

- As técnicas de análise de cena envolvem a recolha e processamento de imagens ou características eletromagnéticas recolhidas na posição do objeto-alvo. A análise das características eletromagnéticas medidas e comparação com um modelo fornece a localização do objecto-alvo utilizando técnicas de mapeamento. Um sistema de localização pode também identificar características e padrões numa *stream* de dados de vídeo para determinar a localização. Se existirem *tags* espalhadas no ambiente, a perspetiva e localização das mesmas numa imagem de vídeo também permite calcular a posição aproximada da câmara.

- As técnicas de *Dead Reckoning* utilizam estimativas de velocidade dadas por sensores. Sabendo a localização inicial de um robô e a sua *pose*(velocidade e rotação) é possível estimar a sua posição atual. Os dados necessários são o resultado da integração das informações de vários sensores como *encoders*, acelerómetros ou giroscópios.

Nos últimos anos têm sido propostas, testadas e avaliadas bastantes técnicas de localização *indoor*. Contudo, ainda não existe uma combinação destas tecnologias *indoor* que forneça utilização comparável ao *GPS* para *outdoor* [9].

2.3.1 Métodos Baseados em *Wi-Fi*

A ideia da utilização de *Wi-Fi* para estimação de localização de utilizadores móveis já existe há alguns anos [10]. Os avanços na tecnologia dos aparelhos móveis já permitem uma localização relativamente precisa de utilizadores com recurso a dispositivos sem-fios, em espaços *indoor* onde o sinal de GPS não está disponível. Uma forma prática e bastante utilizada é utilizar os sinais *Wi-Fi* recebidos pelo utilizador dos diferentes *access points* . A maioria das técnicas de localização *indoor* utilizam valores de *RSS* e modelos de propagação de sinais rádio para localizar o utilizador[11][12].

As técnicas mais utilizadas são:

- Baseadas em *RSS* e lateração. Combinam as medições de *RSS* de um cliente para vários *access points* com modelos de propagação, de forma a obter a distância aos *access points*. Utilizando técnicas de Trilateração, a posição estimada do cliente é calculada, relativamente à posição conhecida dos *access points*[13].

- Baseadas em *fingerprinting*. Estas técnicas utilizam apenas os dados de RSS. Numa primeira fase são medidos os dados de RSSI em relação a vários *access points*, assim como a posição do cliente durante essas medições. Estes dados são guardados numa base de dados. Na fase de localização os dados atuais de RSSI são comparados com a base de dados retornando a posição de maior proximidade encontrada. Esta é considerada a posição estimada do cliente. A maior limitação destes sistemas é a suscetibilidade a mudanças no ambiente, como a adição ou remoção de mobília. Estas alterações provocam alterações na *fingerprint*

do local, obrigando a atualizações da base de dados[14]. O Anyplace [15] é um sistema de posicionamento *Wi-Fi*, gratuito e *open-source* que permite o mapeamento de espaços interiores.

- Ângulo de chegada.O *AOA* é o ângulo com que o sinal chega ao recetor. Este ângulo é calculado através da medição da diferença de tempo entre a chegada do sinal a diferentes antenas, normalmente utilizando o algoritmo MUSIC[16]. Spotfi [17], ArrayTrack [18] e LTEye [19] são soluções propostas que utilizam este método.

- Tempo de transmissão. O *TOF* é tempo que o sinal demora para se propagar desde o transmissor até ao recetor. Este método assenta na velocidade praticamente constante das ondas de rádio-frequência na maioria dos meios. Assim, este método não é tão dependente do ambiente como aqueles baseados em *RSS*[20]. O *TOF* apenas estima a distância do cliente aos *access points*. Uma técnica de trilateração continua a ser necessária para obter a posição estimada do cliente. O maior desafio destes métodos são problemas relacionados com a sincronização[20].

2.3.2 Métodos Baseados em *Bluetooth*

A localização não era um dos objetivos previstos aquando do desenvolvimento do protocolo *Bluetooth*[21][22]. Após a standardização como *IEEE 802.15.1* [23] começaram a ser desenvolvidos sistemas para utilização da tecnologia para posicionamento e localização[24][25][26]. O *Bluetooth* não fornece localizações exatas, mas sim medidas de proximidade, normalmente a *beacons*.

A ideia principal é registar a força do sinal emitido por 3 ou mais *beacons*, cujas posições no espaço real são conhecidas. Estas medições utilizam o RSS, que indica as perdas que o sinal sofreu no caminho entre o emissor e o recetor. Com o RSS podemos obter as distâncias aos *beacons* e utilizar multilateração para obter a posição aproximada do dispositivo [27].

A utilização de *beacons* baseados em *Bluetooth Low Energy* [28] provou ser bastante precisa na localização de dispositivos em espaços interiores [29]. Um dos protocolos mais utilizados é o *iBeacon*, desenvolvido pela *Apple* [30]. Existem inúmeros transmissores - *beacons* - que utilizam este protocolo para fazer *broadcast* do seu identificador para dispositivos próximos, como *smartphones*. Caso os dispositivos recetores sejam compatíveis com a tecnologia podem executar diversas ações quando se encontram na proximidade dos *beacons*. A mensagem enviada pelos *beacons* pode ser utilizada para determinar a distância ao dispositivo, contribuindo assim para a localização do dispositivo recetor [31][32].

2.3.3 *Dead Reckoning*

O processo de *Dead Reckoning* consiste no cálculo da posição atual de um objeto com base numa posição anterior conhecida e os valores da sua direção e direção, conhecidos ou estimados. Conhecendo a posição absoluta do objecto - *fix* - num momento anterior e conhecendo, ou

estimando, a sua velocidade e sentido de movimento é possível estimar a sua posição atual.

Contudo, esta técnica está sujeita a erros significantes. Para determinar a posição de forma precisa é necessário conhecer a velocidade e direção em todos os momentos do movimento entre a posição de partida e a posição atual. Por exemplo, se for utilizado o número de rotações de uma roda para calcular a distância percorrida, podem ser introduzidos erros devidos a irregularidades na superfície ou derrapagens da roda. Estes fatores criam uma discrepância entre a distância real percorrida e a distância que se assume que foi percorrida por cada rotação. Como cada posição estimada é relativa à posição anterior, estes erros são cumulativos.

2.3.3.1 Hodometria

O método de utilizar sensores de movimento, como *encoders* para estimar a posição de um robô é chamado **hodometria**. É um dos métodos mais utilizados para estimar a posição de um robô, apesar de normalmente ser combinado com medidas de posicionamento absolutas [33][34].

Os *encoders* rotativos convertem movimentos de rotação em impulsos. Por cada volta completa do seu eixo geram uma quantidade exata de impulsos. A contagem do número de impulsos permite quantificar a rotação realizada. Normalmente os *encoders* rotativos são caracterizados pelo seu valor de distância por impulso, C , dado por:

$$C = \frac{\pi D}{N} \tag{2.1}$$

onde D é o diâmetro da roda e N o número de impulsos por volta completa.

A distância percorrida por cada uma das rodas pode então ser calculada

$$d_R = C_R n_R \tag{2.2}$$

$$d_L = C_L n_R \tag{2.3}$$

O processo apresentado é válido para *encoders* rotativos incrementais. Estes apenas quantificam a rotação do eixo desde a alimentação. Por outro lado os *encoders* absolutos indicam a posição absoluta atual do eixo.

2.3.3.2 Hodometria visual

A **hodometria visual** é uma variante do método de *Dead Reckoning* que utiliza a informação de uma ou mais câmaras para estimar a posição de um agente, por exemplo um robô. Funciona através da estimação incremental do movimento do agente com base na análise das alterações nas imagens das câmaras, provocadas pelo movimento do agente [35].

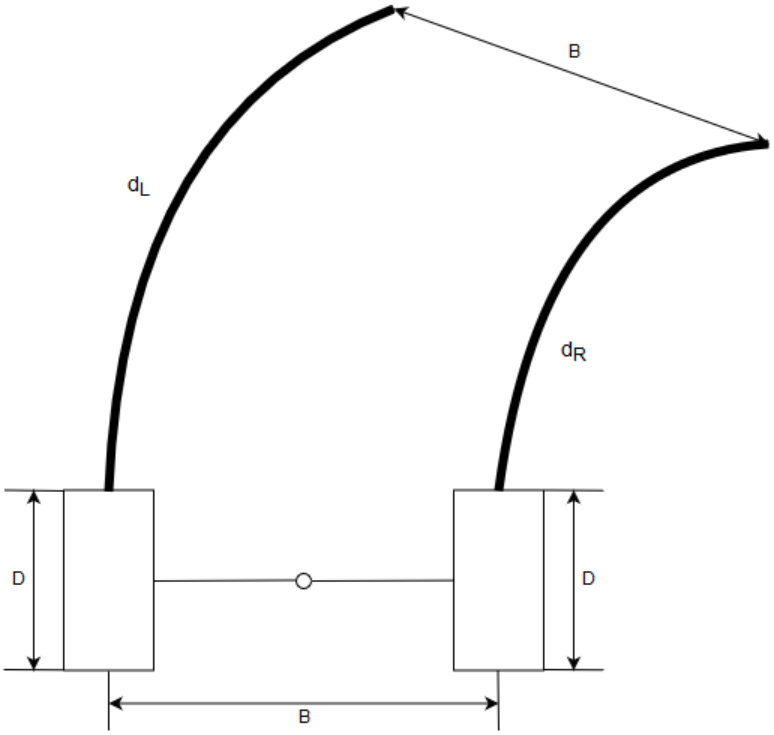

Figura 2.6: Representação do deslocamento de um robô

2.3.4 Métodos Baseados em características visuais

As técnicas baseadas em características visuais utilizam interpolação de imagens recolhidas pela câmara do dispositivo em bases de dados. Estas bases de dados contêm imagens recolhidas anteriormente no local, a posição do dispositivo, assim como características detetadas na imagem. Estas características normalmente são superfícies planares, como paredes e portas. Com a interpolação da sequência de imagens recolhidas *online* na base de dados criada *offline* é possível obter uma aproximação da localização do dispositivo [36].

2.3.5 Métodos Baseados em Artificial Beacons/Landmarks

A posição de um dispositivo móvel com câmara pode ser determinada através da descodificação de coordenadas de marcadores visuais. Nestes sistemas de localização, os marcadores são colocados em posições conhecidas e as coordenadas dessa posição codificadas no próprio marcador. O cálculo do ângulo entre o marcador e o dispositivo permite estimar a posição do dispositivo. Esta posição é relativa ao marcador. Porém, combinando com a posição conhecida do marcador a localização do dispositivo pode ser estimada [37].

Os sistemas de marcadores visuais podem ter complexidades bastante distintas. Os mais simples devem ser alinhados manualmente pelo utilizador e permitem a codificação de grandes quantidades de dados, como os *QR codes* [38]. Porém, os mais interessantes do ponto de vista de localização são aqueles que fornecem a posição relativa à câmara e orientação da *tag*. Uma das

utilizações mais comuns destes sistemas é a realidade virtual, que desencadeou o desenvolvimento de sistemas que atingiram a popularidade, como o *ARToolkit* [39] e o *ARTag*[40].

2.3.5.1 AprilTag

O ***AprilTag*** [41] é um sistema completamente aberto e documentado que permite a criação de *tags* compostos por códigos binários 2D. Os três sistemas indicados anteriormente têm como base dos seus marcadores uma borda negra. Contudo, o *ARToolkit* permite a existência de padrões aleatórios de pixeis no interior dessa borda, aumentando assim a base de dados de padrões e o poder computacional necessário para encontrar e distinguir padrões.

O sistema ***AprilTag 2*** [42] introduz melhorias de performance relativamente ao seu predecessor, conseguindo também uma melhor precisão de localização do que o *ARTag*. Tem também a vantagem de fornecer uma implementação do detetor de *tags* completamente *open-source*.

Na figura 2.7 apresentam-se exemplos de marcadores *AprilTag*.

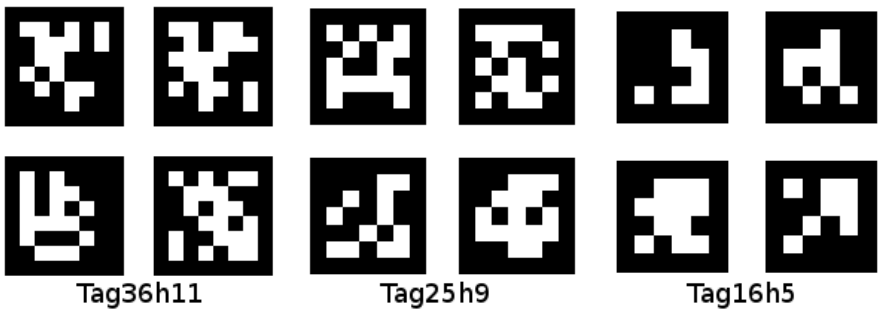

Figura 2.7: Exemplos de marcadores *AprilTag*

2.3.5.2 Aruco

O sistema Aruco foi desenvolvido com o propósito de solucionar o problema da escolha de dicionários e melhorar o processo de deteção de marcadores. Apresenta um algoritmo para geração de dicionários de marcadores configuráveis em tamanho e número de bits, de forma a maximizar a distância entre marcadores e o número de transições de bits. A distância entre dois marcadores é definida por

$$D(m_i, m_j) = \min_{k\varepsilon\{0,1,2,3\}} \left\{ H(m_i, R_k(m_j)) \right\} \tag{2.4}$$

A função H é a distância de Hamming entre dois marcadores, definida como a soma das distâncias de Hamming entre cada par de palavras. A função R_k é um operador que roda a grelha $k * 90^\circ$ no sentido contrário ao dos ponteiros do relógio. Assim, a função D é a distância de Hamming *rotation-invariant* entre os dois marcadores.

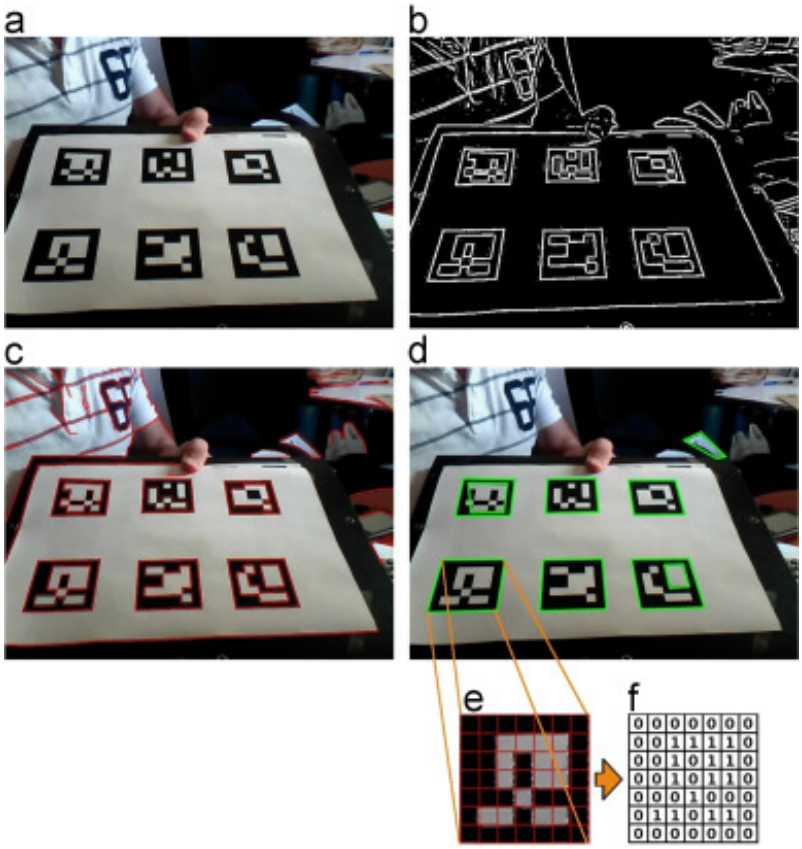

Figura 2.8: Processamento de imagem para deteção dos marcadores. Adaptado de [43]

O processo de deteção de marcadores é composto por vários passos:

- Segmentação da imagem: aplicação de um *threshold* adaptativo local (ver figura 2.8(b))

- Extração de contornos e filtragem: aplicação de método Suzuki e Abe para extração de contornos (ver figura 2.8(c)); aproximação poligonal usando método Douglas-Pecker; seleção dos polígonos com 4 vértices e remoção dos polígonos interiores em caso de proximidade de contornos (ver figura 2.8(d))

- Extração do código do marcador: cálculo da matriz de homografia para remoção da perspetiva; aplicação de *threshold* usando método Otsu; divisão da imagem numa grelha (ver figura 2.8(e)); atribuição de de valor 0 ou 1 a cada elemento da grelha conforme a maioria dos pixeis no seu interior sejam pretos (0) ou brancos (1) (ver figura 2.8(f)); confirmação da presença da borda preta

- Identificação do marcador e correção de erros: obtenção de 4 identificadores, 1 para cada rotação; caso algum seja encontrado no dicionário o marcador é considerado válido;

2.4 Conceitos de visão computacional

Tendo em conta a importância da visão computacional nesta dissertação, é importante esclarecer os principais conceitos desta área. De seguida são explicados os conceitos de distorção e calibração de câmaras.

2.4.1 Distorção

Existem duas formas de distorção normalmente presentes em imagens capturadas por câmaras: distorção radial e distorção tangencial.

Distorção Radial

Trata-se de uma distorção radialmente simétrica, em que o nível de distorção está relacionado com a distância ao centro ótico da câmara. Na verdade é uma alteração na ampliação. Se a ampliação diminui com o aumento da distância ao centro ótico, é chamada distorção em barril. Por outro lado se a ampliação aumenta com a mesma variação da distância, o efeito denomina-se distorção em rolamento. A figura 2.9 mostra um exemplo das duas distorções apresentadas.

Figura 2.9: Representação dos efeitos de distorção de barril e distorção de rolamento

Se considerarmos o eixo ótico como a origem de uma imagem $f(i,j)$ então:

$$i' = i(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.5}$$

$$j' = j(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.6}$$

onde $f'(i',j')$ é a imagem corrigida, $r = \sqrt{i^2 + j^2}$ a distância ao centro ótico e k_1, k_2, k_3 são parâmetros que descrevem a distorção.

Distorção tangencial

Ocorre quando a lente não está perfeitamente paralela ao plano da imagem e resulta numa ampliação irregular. Contudo, neste caso a ampliação varia de um lado do plano de imagem para o outro.

Assumindo novamente que a origem da imagem coincide com o eixo ótico, a distorção tangencial pode ser modelada por

$$i' = i + (2p_1ij + p_2(r^2 + 2i^2) \tag{2.7}$$

$$j' = j + (2p_2ij + p_1(r^2 + 2j^2) \tag{2.8}$$

onde $r = \sqrt{i^2 + j^2}$ representa a distância ao eixo ótico e p_1, p_2 são parâmetros que descrevem a distorção.

O conjunto dos parâmetros indicados anteriormente $(k_1, k_2, p_1, p_2, k_3)$ denomina-se coeficientes de distorção e dependem apenas dos componentes físicos utilizados no sistema da câmara (como lentes e sensor) e posicionamento dos mesmos. São independentes da resolução utilizada. Assim, mantêm-se constantes enquanto o sistema físico da câmara se mantiver inalterado.

2.4.2 Calibração

Um passo extremamente importante para recolher informação precisa de imagens é o processo de calibração da câmara. Este processo permite calcular a distância focal, as coordenadas do centro da câmara, os coeficientes de distorção (parâmetros intrínsecos) e até a posição relativa da câmara em relação a outro referencial, como outra câmara ou um marcador planar(parâmetros extrínsecos). Para uma melhor compreensão dos parâmetros intrínsecos e extrínsecos são apresentadas as figuras 2.10 e 2.11, respetivamente.

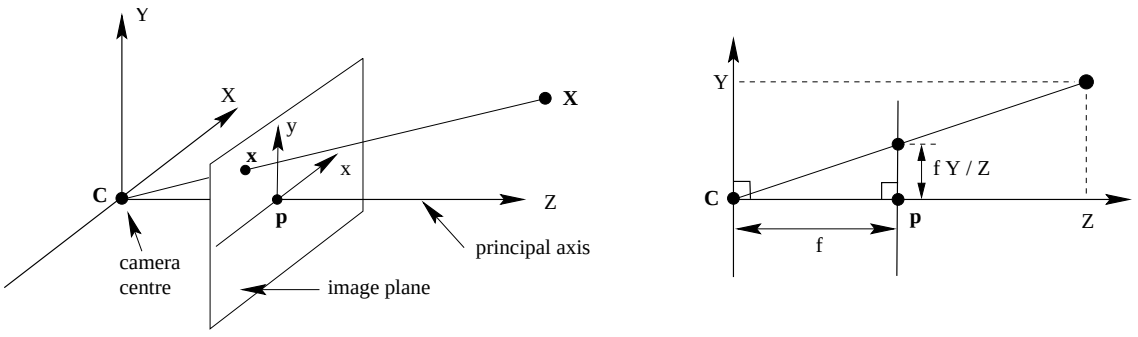

Figura 2.10: Representação dos parâmetros intrínsecos, onde f é o ponto focal e p é o ponto principal, onde o eixo ótico interseta o plano da imagem

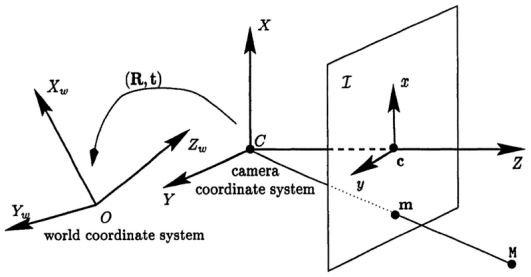

Figura 2.11: Representação dos parâmetros extrínsecos, onde c é o centro da imagem, M é um ponto no referencial mundo e m o ponto correspondente de M no plano da imagem

A conjugação destes parâmetros permite a criação das matrizes intrínseca K e extrínseca $[R \mid t]$:

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

$$[R \mid t] = \begin{bmatrix} r_1 1 & r_1 2 & r_1 3 & t_1 \\ r_2 1 & r_2 2 & r_2 3 & t_2 \\ r_3 1 & r_3 2 & r_3 3 & t_3 \end{bmatrix} \tag{2.10}$$

que por sua vez permitem a transformação de coordenadas 3D de um referencial externo em coordenadas 2D de uma imagem. Essa projeção é definida por

$$x = K[R \mid t]X \tag{2.11}$$

onde x são as coordenas dos pixeis da imagem(2D), Ka matriz intrínseca, R a matriz de rotação, t o vetor de translação e X as coordenadas do referencial externo(3D).

Na sua forma prática, o processo de calibração consiste na captação de várias imagens contendo um objeto de calibração conhecido, em diferentes poses (posições e rotações). Os parâmetros do modelo são calculados através de transformações matemáticas baseadas nas equações X. Uma vez calibrada a câmara, as distorções existentes nas imagens seguintes são removidas através por aplicação das equações indicadas anteriormente.

Capítulo 3

Projeto do Sistema de Localização

Em qualquer projeto de engenharia, antes de se implementar uma solução é necessário projetar e formular a mesma. Nesse processo é fundamental analisar os requisitos do sistema a desenvolver e conjugar os mesmos com as diferentes opções disponíveis para cumprir esses requisitos. Desse modo é possível criar um conceito do sistema baseado nas alternativas selecionadas.

Neste capítulo apresentam-se as decisões tomadas no projeto do sistema.

Inicialmente são apresentados os requisitos do sistema a desenvolver, seguidos de uma análise das alternativas disponíveis para o cumprimento dos mesmos. Finalmente é feita uma formulação final do conceito do sistema.

3.1 Requisitos do Sistema

A fase inicial com mais importância de todos os projetos passa pelo levantamento dos requisitos a cumprir pelo sistema. Estes são essencialmente funcionalidades requeridas pelo cliente, mas também podem incluir sugestões dadas pelo projetista. Para que se atinja um produto final de qualidade é importante que este processo passe por várias iterações e se baseie numa comunicação articulada entre o projetista e o cliente. Assim, a primeira fase do projeto foi um levantamento dos requisitos, em conjunto com o Professor Doutor Armando Sousa:

- O sistema deve ser de utilização simples para os utilizadores - fornecer localização sem obrigar a operações complexas no ambiente de programação;
- Fornecer distância, ângulo e identificação do marcador avistado;

O sistema deve ainda depender tão pouco quanto possível do exterior e ser tão pouco intrusivo quanto possível.

3.2 Abordagens de Solução

Como explicado na secção 2.2.2 existem vários métodos disponíveis para programação do *Robobo*. Esta foi a primeira decisão a ser tomada no projeto do sistema. Foi então necessário

decidir qual dos métodos apresentados seria utilizado ao longo deste trabalho. Para tal foram analisadas e comparadas as características dos métodos disponíveis. A tabela 3.1 apresenta essa comparação.

Tabela 3.1: Comparação dos métodos de programação do *Robobo*

	Independência de Hardware Externo	**Acesso Imagens Câmara**	**Adição de funcionalidades à Framework**
Blocos	Não	Não	Não
ROS	Não	Sim	Não
Nativa	Sim	Sim	Sim

Analisando a tabela 3.1 é possível verificar que apenas os métodos de programação nativa e ROS são de interesse para o projeto, pois permitem a utilização de imagens provenientes das câmaras do telemóvel.

Comparando as duas alternativas supra-indicadas, as principais diferenças residem na necessidade de processamento externo do método baseado em ROS e na possibilidade de criação de novas funcionalidades do método nativo. Conjugando estas diferenças com o requisito de desenvolver um sistema simples, independente de um computador externo e que possa ser utilizado por outros utilizadores selecionou-se o método de programação nativa.

Selecionada a plataforma e arquitetura base do sistema partiu-se para a análise das abordagens possíveis, no que diz respeito à deteção e localização dos marcadores.

A decisão de utilizar a biblioteca ARUCO foi tomada, ainda na fase de preparação desta dissertação. Durante a pesquisa foram encontradas diversas implementações do algoritmo, das quais se destacam duas: uma desenvolvida em C++ pelo criador original [44], e outra resultante da adaptação da primeira para correr de forma nativa em Java [45].

Inicialmente, optou-se por utilizar a opção desenvolvida em Java. Contudo, no seguimento do trabalho houve necessidade de mudar para a alternativa (código fonte original). Esta mudança deveu-se ao facto da opção selecionada inicialmente contemplar apenas as funcionalidades mais básicas da biblioteca. Com esta implementação apenas era possível detetar o marcador e calcular o seu ID. Os procedimentos mais avançados, como calibração e obtenção dos vetores de rotação e translação, não estavam implementados. Tendo em conta a elevada importância desses procedimentos para o trabalho, optou-se por abandonar esta opção e utilizar a implementação original, em C++.

3.3 Conceito do Sistema

O sistema conceptualizado baseia-se na integração da biblioteca Aruco com a ROBOBO Framework. A *framework*, acessível através de uma aplicação Android é responsável pela captação das imagens provenientes da câmara do *smartphone*. A tarefa de identificar marcadores na imagem e obter a posição do robô em relação aos mesmos fica a cargo da componente do projeto

desenvolvida em linguagem C onde foi utilizada a biblioteca Aruco. O conceito funcional do sistema está apresentado na figura 3.1.

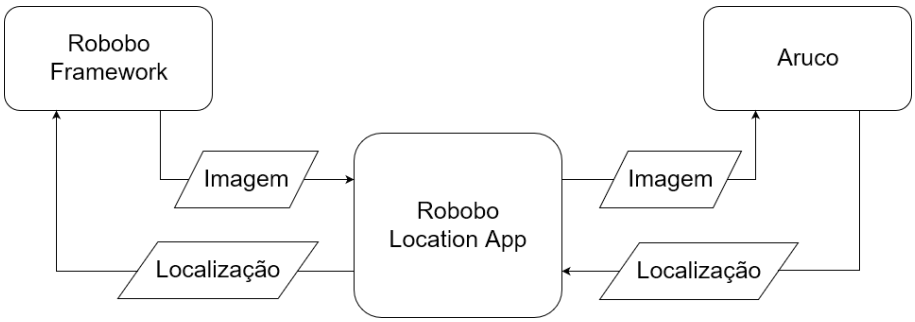

Figura 3.1: Diagrama de funcionamento do Robobo Location Module

Capítulo 4

Módulo de Localização

Neste capítulo apresentam-se o projeto e as ferramentas utilizadas na implementação do sistema. Começa-se por fazer uma descrição do ambiente de desenvolvimento, nomeadamente as ferramentas, bibliotecas e tecnologias que foram utilizadas. É apresentada a arquitetura do sistema e destacadas as funcionalidades implementadas.

4.1 Ambiente de Desenvolvimento - Ferramentas

Tendo em conta que o formato de solução escolhido se baseia na programação Nativa foi necessário selecionar o ambiente de desenvolvimento a utilizar. Existem inúmeras soluções para programação com a plataforma Android como *target*. Contudo, visto que a documentação disponível para programação nativa do *Robobo* [7] se baseia na ferramenta Android Studio, esta foi a escolhida.

4.1.1 Android Studio

O Android Studio[46] é o IDE oficial da Google. É um IDE bastante completo, que oferece todas as funcionalidades necessárias para desenvolver aplicações *Android* sem ser necessário sair da ferramenta. É baseado em IntelliJ IDEA [47], um editor de código para Java. As funcionalidades que mais se destacam para a criação de aplicações Android são:

- Ferramentas de debugging
- Simulação de dispositivos e teste em dispositivos virtuais
- Integração com sistemas de controlo de versão
- Monitorização de Logs do sistema

O simulador incluído permite ao programador configurar diversas características do dispositivo virtual como o tamanho de ecrã, a versão de Android e a RAM.

Outra funcionalidade de grande importância é o *Android Device Monitor*. Permite monitorizar tanto dispositivos reais ligados ao computador, como dispositivos virtuais.

Uma vez que o autor, à data de início desta dissertação, não possuía experiência de programação para ambiente Android, o processo de ambientação e estudo da plataforma foi demorado.

4.1.2 Bibliotecas e tecnologias utilizadas

4.1.2.1 Robobo Framework

Ultrapassada a fase de adaptação ao ambiente de desenvolvimento, foi iniciado o contacto com a Robobo Framework. Como ponto de partida, foi utilizada a documentação oficial de programação nativa para o Robobo [7].

Contudo, e após diversas tentativas de implementação, chegou-se à conclusão que a documentação existente se encontrava ligeiramente desatualizada em alguns pontos fundamentais, nomeadamente no projeto de Android Studio indicado como ponto de partida. Para solucionar estes problemas foram desenvolvidos contactos com os responsáveis pelo projeto Robobo. Desses contactos resultou uma melhor compreensão da *framework*, assim como uma atualização da documentação.

Posteriormente foram realizados ensaios para ambientação à *Robobo Framework*, onde foram testadas as funcionalidades que se sabia necessárias à realização desta dissertação, como a captação e processamento de imagens provenientes da câmara do *smartphone*. Neste ponto, foi encontrado um novo contratempo. Detetou-se que a *framework* não permite controlar sobre o *layout* do ecrã do telemóvel. A única alteração permitida é a modificação das expressões faciais do *Robobo*. Estas expressões são apresentadas no ecrã, durante a execução de todos os programas. Uma vez que o sistema projetado possui uma grande componente de visão, a possibilidade de controlar o ecrã, nomeadamente para apresentação de imagens, era de grande importância para a fase de desenvolvimento.

Assim, foram desenvolvidos novos contactos com os responsáveis do projeto do robô para solucionar esta contrariedade. Infelizmente, não foi indicada nenhuma solução válida, tendo sido indicado que o controlo do ecrã é distribuído numa biblioteca interna cujo código ainda não foi tornado público.

Tendo em conta a importância da observação de imagens neste projeto foi tomada a decisão de desenvolver uma aplicação Android de origem, onde fosse possível implementar funcionalidades, que depois de testadas seriam implementadas sobre a *Robobo framework*. O desenvolvimento desta aplicação, usada apenas para desenvolvimento, revelou-se demorado, principalmente pela reduzida experiência em programação para Android do autor.

Na figura 4.1 é apresentada uma captura de ecrã da aplicação desenvolvida, onde é possível ver um marcador. Sobre o marcador detetado são apresentados os eixos do mesmo, assim como a sua identificação.

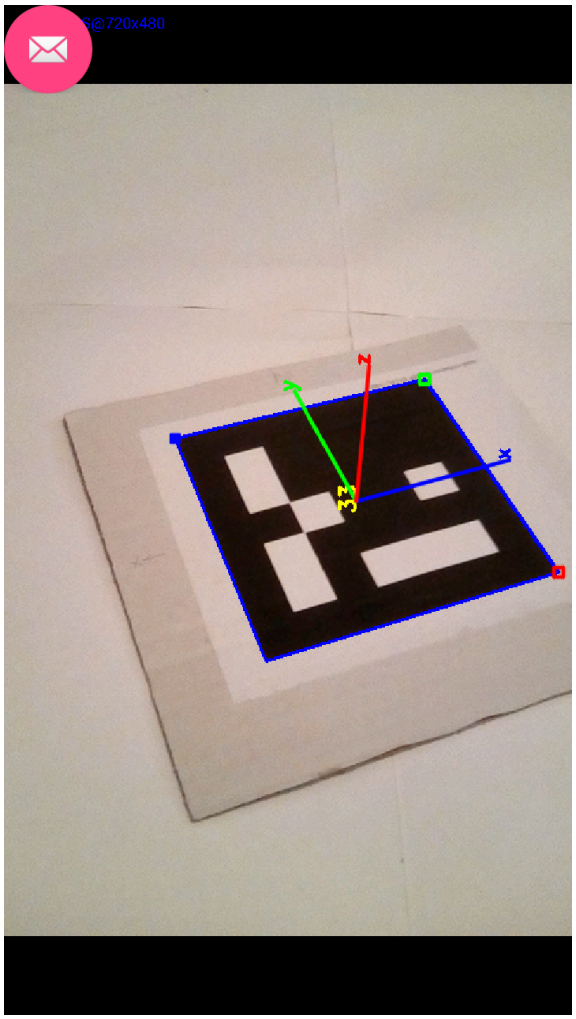

Figura 4.1: Captura de ecrã da aplicação criada para desenvolvimento - Resolução: 480 x 720, Abertura: f/2.8

4.1.2.2 Aruco

A biblioteca *Aruco*, apresentada na subsecção 2.3.5 é utilizada para criar o dicionário de marcadores, para detetar os marcadores e para determinar a posição da câmara no referencial dos marcadores.

Contudo, após uma análise aprofundada das funcionalidades desta biblioteca percebeu-se que a função *getCameraLocation()*, que deveria fornecer as coordenadas da câmara no referencial do marcador, na realidade não tinha o comportamento desejado. Assim, foi necessário proceder a algumas alterações, cujo resultado é apresentado de seguida.

Na versão da biblioteca utilizada (ver. 3.0) a função, que não se encontra completamente desenvolvida (e apresenta avisos, no código-fonte) não cumpre o seu objetivo, devolvendo valores que não correspondem ao pretendido. Estas incorreções foram reportadas por utilizadores da biblioteca, sendo que durante o desenvolvimento desta dissertação não foram corrigidas por parte

```
cv::Point3f CameraParameters::getCameraLocation(cv::Mat Rvec, cv
    ::Mat Tvec){
        cv::Mat Rmat(3, 3, CV_32FC1);
        cv::Rodrigues(Rvec, Rmat);

        cv::Mat camera_position = -Rmat.t() * Tvec;

        return cv::Point3f(camera_position.at<float>(0, 0),
         camera_position.at<float>(1, 0), camera_position.at<
            float>(2, 0));
    }
```

Listagem 4.1: Método para obtenção da posição do câmara no referencial do marcador

do programador original. Assim, procedeu-se às alterações necessárias para que a função possa ser útil .

O método recebe como parâmetros os vetores de rotação e translação. Para obter a matriz de rotação, aplica-se o método de Rodrigues ao vetor de rotação. Finalmente, a posição da câmara no referencial do marcador é obtida por uma operação resultante da manipulação da fórmula indicada na subsecção 2.4.2.

Contudo, para o valor de posição ser útil, é necessário conhecer também a orientação da câmara relativamente ao marcador. Para isso, foi criado um método, semelhante ao apresentado na listagem 4.1, que devolve o ângulo entre a projeção do eixo ótico da câmara no referencial do marcador e o eixo X do marcador. O funcionamento desta função é apresentado no Algoritmo 1.

ALGORITMO 1
Calculo do ângulo entre a projeção do eixo da câmara no plano do marcador e o eixo X do marcador

```
function GET ORIENTATION( Mat rotationVector, Mat translationVector, flag
markerIsVertical)
    RotationMatrix ← RODRIGUES(rotationVector)
    InvertedRotMat ←invert RotationMatrix
    cameraAxis ← [0 0 1]
    TraslatedCameraAxis ← CameraAxis - translationVector
    FinalCameraAxis ← InvertedRotMat ∗ TranslatedCameraAxis
    X ← FinalCameraAxis(0,0)
    if markerIsVertical then
        Z ← FinalCameraAxis(0,2)
        angle ← -ATAN2(Z,X)
    else
        Y ← FinalCameraAxis(0,1)
        angle ← -ATAN2(-Y,X)
    end if
    return angle
end function
```

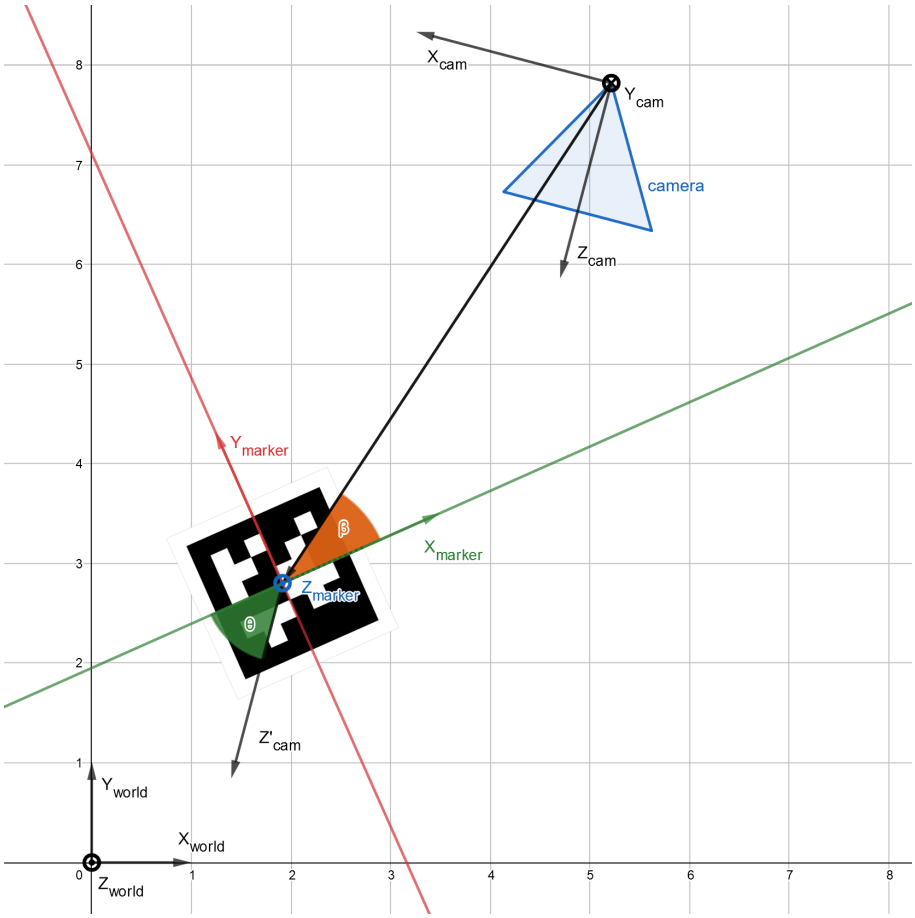

Figura 4.2: Esquema dos ângulos utilizados para cálculo da orientação

A figura 4.2 apresenta um esquema da visualização superior dos ângulos utilizados para o cálculo da localização. O ângulo β representa o ângulo de avistamento do marcador. Este ângulo depende diretamente da posição do robô no referencial cartesiano do marcador e pode ser utilizado caso se pretenda a utilização de coordenadas polares para a posição do robô. Contudo, utilizando apenas este ângulo e a distância ao marcador apenas se obtém a posição do robô. Tal como indicado anteriormente, de modo a obter uma localização é também necessário calcular a orientação. De forma a obter a orientação do robô, o eixo da câmara (Z_{cam}) é projetado no referencial do marcador. O ângulo entre esta projeção e o eixo -X do marcador, θ, representa o ângulo da câmara em relação ao marcador.

Conjugando os dois valores calculados (posição e orientação) obtém-se a localização do robô.

4.1.3 Java Native Interface

Uma vez que a biblioteca escolhida para deteção dos marcadores está desenvolvida em C++ foi necessário encontrar uma forma de utilizar a mesma com a *framework* do Robobo, desenvolvida em Java, para Android.

A JNI é uma framework de Java que fornece uma interface entre o código na Java Virtual Machine (JVM) e código nativo, principalmente compilado em C/C++. Esta interface permite a invocação de métodos nativos a partir de objetos Java. Reciprocamente permite também invocação de métodos Java pelo ambiente nativo.

Contudo, apesar de objetos Java e os seus dados serem acessíveis pelo ambiente nativo, o inverso não acontece, e não é fornecido qualquer mecanismo que permita o acesso direto a dados pertencentes ao ambiente nativo. Assim, a JNI apresenta-se como uma ferramenta poderosa, que remove limitações aos programadores. O diagrama de funcionamento da JNI está representado na figura 4.3.

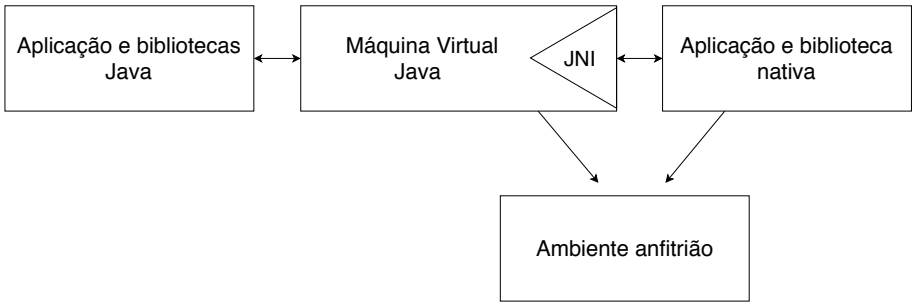

Figura 4.3: Arquitetura da Java Native Interface (JNI). Adaptado de [48]

Na listagem 4.2 apresenta-se um exemplo simples da utilização da JNI no sistema desenvolvido.

```
JNIEXPORT jstring JNICALL
    Java_example_myaruco_MainActivity_jniGetLog(JNIEnv *env,
    jobject){
        string str = LogStream.str();
        LogStream.str(std::string());
        return env->NewStringUTF(str.c_str());
}
```

Listagem 4.2: Exemplo de utilização de JNI

O código apresentado exemplifica o formato de uma função JNI. Neste caso, a função retorna uma *string Java*, indicado por *jstring*. O nome da função é *jniGetLog* sendo que o texto que a precede refere-se ao ficheiro que a pode chamar. Neste caso um ficheiro *Java*, localizado em *example/myaruco/*, e nome *MainActivity*. Os parâmetros fazem referência ao ambiente virtual Java que fez a chamada da função. Este método foi utilizado ao longo do projeto para ter acesso ao *Log* da parte nativa do projeto.

4.2 Arquitetura

A arquitetura final do sistema é apresentada na figura 4.4.

O módulo desenvolvido, *Robobo Localization Module*, é constituído por duas partes, sendo uma desenvolvida em *Java* e outra em C, que comunicam pela *JNI*.

A componente *Java* do módulo, é responsável por captar as imagens da câmara e receber os pedidos de Localização.

Por forma a obter a imagem mais recente proveniente da câmara do telemóvel, implementou-se um módulo *CameraListener*, que permite subscrever notificações do *CameraModule*. Após a sua iniciação e subscrição ao módulo da câmara, no lançamento do Localization Module, o *listener* é notificado sempre que existe uma nova imagem disponível.

Nesse momento, a imagem é convertida num formato *mat* para poder ser utilizada na biblioteca *Aruco*. É também adicionado um *timestamp*, para referência temporal da imagem. Finalmente a imagem é passada à componente nativa desenvolvida em C++, através da *JNI*.

A componente desenvolvida em C++, identificada na figura 4.4 como *Native Localization* é responsável pelas funções de processamento da imagem. Inicialmente a biblioteca *Aruco* (com as alterações indicadas na subsecção 4.1.2.2) é utilizada para encontrar marcadores na imagem. Caso seja detetado um marcador, é calculada a posição da câmara no referencial dos marcadores, assim como a sua orientação. Estes valores, que definem a localização do *Robobo*, são devolvidos à componente *Java* do módulo, onde ficam armazenados, juntamente com a *timestamp* correspondente ao momento é que a imagem que deu origem aos mesmos foi adquirida.

Esta arquitetura, permite manter sempre uma localização disponível e pronta a ser acedida por outros módulos.

O funcionamento das componentes *Java* e *C++* do módulo é apresentado nos algoritmos 2 e 3.

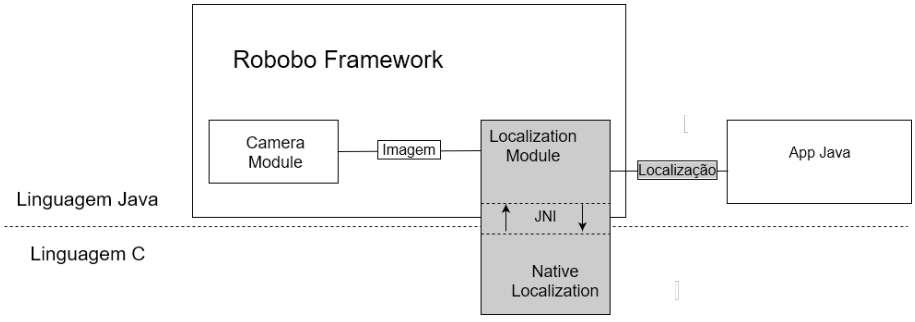

Figura 4.4: Arquitetura do Robobo Localization Module

Na figura 4.5 é apresentado o funcionamento de uma aplicação que utiliza o sistema desenvolvido.

ALGORITMO 2
Componente Java do Location Module

```
upon event Startup do
    CAMERA MODULE. SUBSCRIBE(CameraListener)
upon event CameraListener_onNewFrame do
    mat ← camera frame;
    timestamp ← current time;
    pose ← JNIPROCESSCAMERAFRAME(mat)

function GETCURRENTLOCATION()
    return location,timestamp,quality
end function
```

ALGORITMO 3
Componente C do Location Module

```
function JNIPROCESSCAMERAFRAME(mat)
    DETECTMARKERS(mat)
    if marker_detected then
        position ←GETPOSITION(Rvec, Tvec)
        orientation ←GETORIENTATION(Rvec, Tvec)
        pose ← position,orientation
        return pose
    else
        return error_no_markers_found
    end if
end function
```

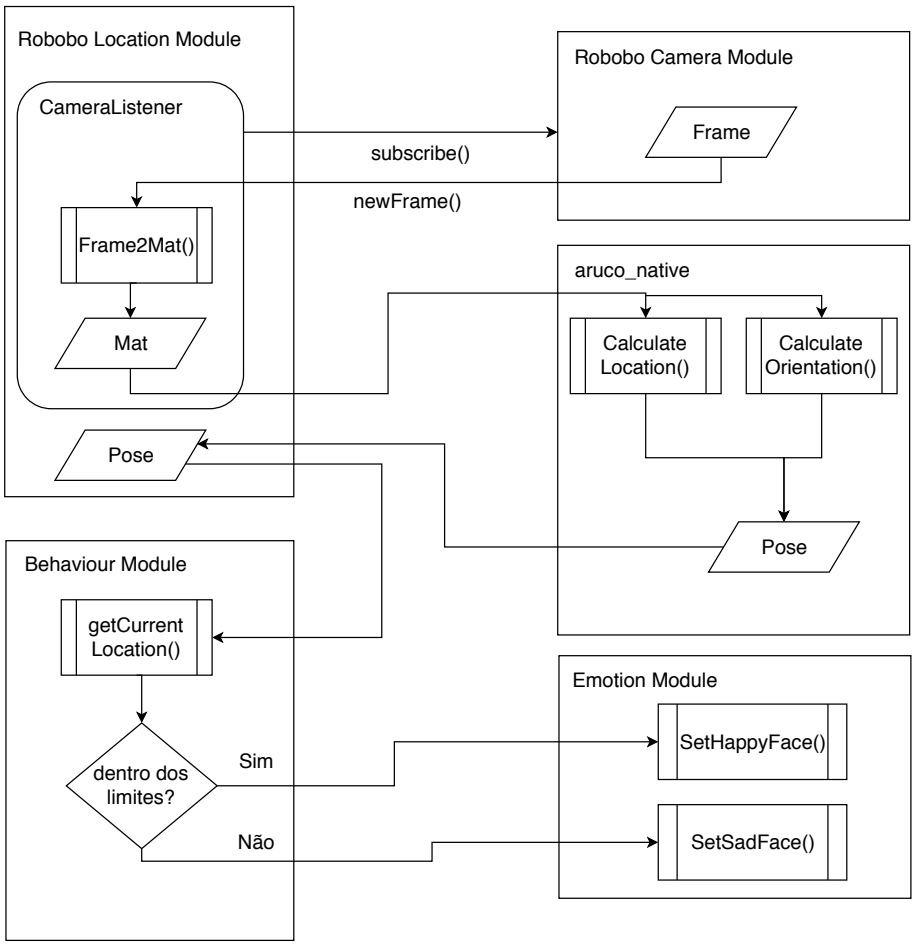

Figura 4.5: Diagrama de funcionamento do Robobo Location Module

4.3 Funcionalidades Implementadas

De forma a cumprir os requisitos do sistema foram implementadas as seguintes funcionalidades:

- Localização do *Robobo* no referencial xyz dos marcadores e consequente localização no mundo
- Localização em coordenadas polares referentes ao centro dos marcadores
- Determinação de modelo para a qualidade da localização obtida

Capítulo 5

Utilização e Resultados

A validação de qualquer projeto de engenharia passa pela verificação dos seus requisitos. Por sua vez, os requisitos são normalmente verificados com a utilização do sistema. Muitas vezes essa é a única forma de garantir que o mesmo funciona e cumpre os requisitos.

Assim, neste capítulo faz-se uma validação do sistema desenvolvido. Começa-se por descrever os testes realizados. De seguida são apresentados os resultados das mesmas e por fim é apresentado um modelo para determinar a qualidade da localização obtida.

5.1 Testes Realizados

De forma a validar o sistema desenvolvido foram realizados diversos testes. Estes têm como objetivo a avaliação de características como a precisão, exatidão e fiabilidade do sistema. Por outro lado, permitem recolher os dados necessários para a criação de um modelo. Esse modelo poderá ser posteriormente implementado no sistema de forma a calcular a qualidade da localização obtida.

O primeiro teste foi realizado com o objetivo de comparar a distância calculada pelo sistema com a distância real, ambas referentes ao marcador. A diferença entre estes dois valores dá-nos o valor do erro da medição e permite analisar a exatidão do sistema. A figura 5.1 representa o método utilizado para a realização deste teste. Para este teste utilizou-se um marcador com a dimensão de 9,7 cm, colocado na vertical. Teste realizado com luminosidade suave e homogénea.

Foi realizado um teste semelhante, para marcadores colocados na horizontal. O método utilizado está representado na figura 5.2.

Os valores resultantes destes testes, apresentam-se no gráfico 5.3. Pela análise do gráfico pode-se observar que o sistema apresenta uma exatidão satisfatória para o tamanho do marcador utilizado. É importante destacar também a grandeza da distância que o sistema permite medir com marcadores verticais, neste caso de 3 metros.

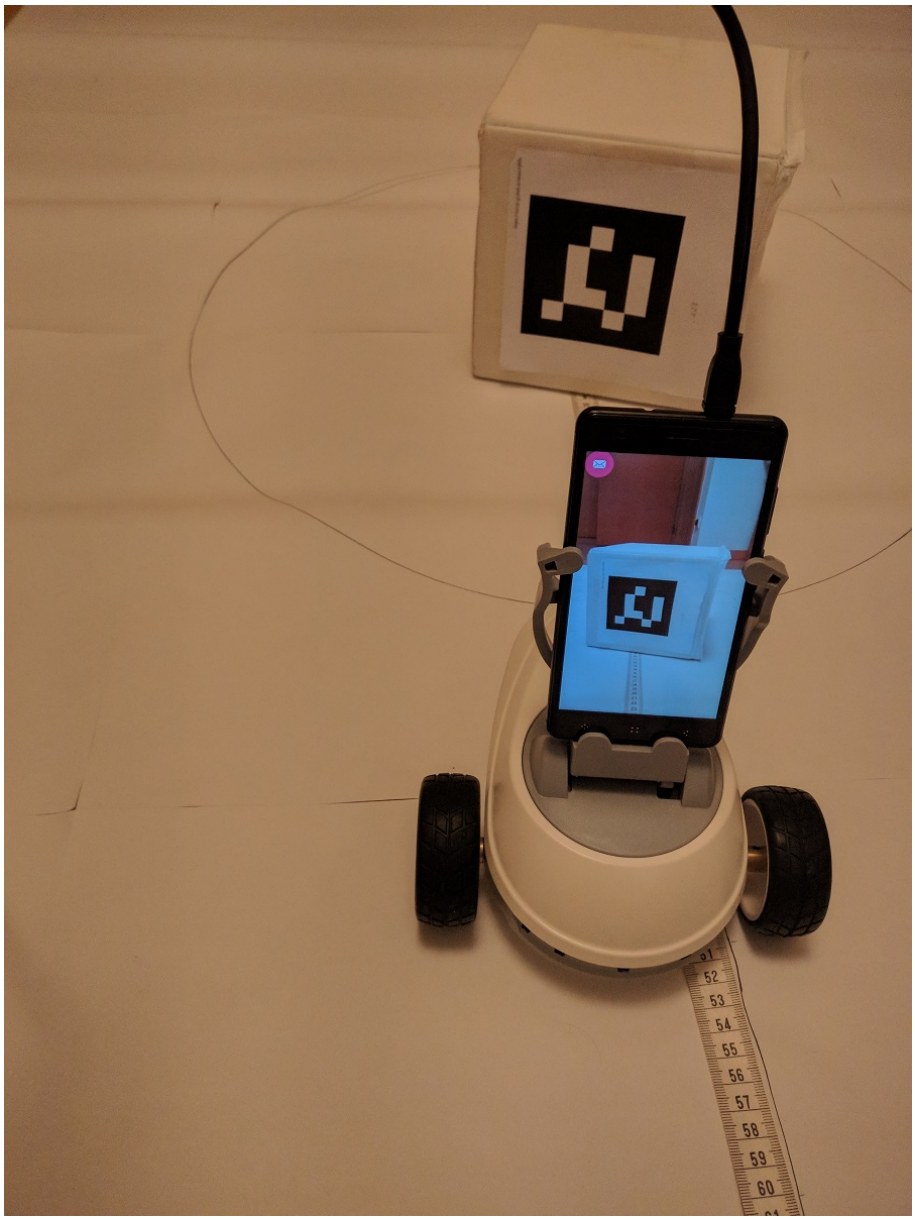

Figura 5.1: Teste para medição da distância com marcador vertical

De forma análoga, foi realizado um teste para comparação do ângulo calculado pelo sistema com o ângulo real, entre o marcador e o eixo da câmara. Os métodos utilizados para marcadores verticais e horizontais são apresentados nas figuras 5.4 e 5.5, respetivamente. Os valores obtidos nestes testes são apresentados na figura 5.6. Pela análise do gráfico percebe-se que a ordem de grandeza dos erros é bastante reduzida.

Foi também realizado um teste para medição da fiabilidade das deteções dos marcadores pelo

Figura 5.2: Teste para medição da distância com marcador horizontal

sistema. Neste teste, com o marcador a uma distância fixa, foram contados o número de marcadores detetados durante um período de tempo de 10 segundos. O mesmo procedimento foi repetido para diferentes distâncias e para marcadores colocados na horizontal e vertical. Os resultados deste teste são apresentados na figura 5.7, para marcadores verticais e horizontais. Pela análise do gráfico é possível verificar que os marcadores verticais apresentam uma fiabilidade de deteção muito superior quando existe um aumento da distância. Os marcadores horizontais apenas apresentam fiabilidade de deteção para distâncias até cerca de 1 metro. Os marcadores colocados na vertical, por outro lado, mantêm a fiabilidade para distâncias na ordem dos 3 metros.

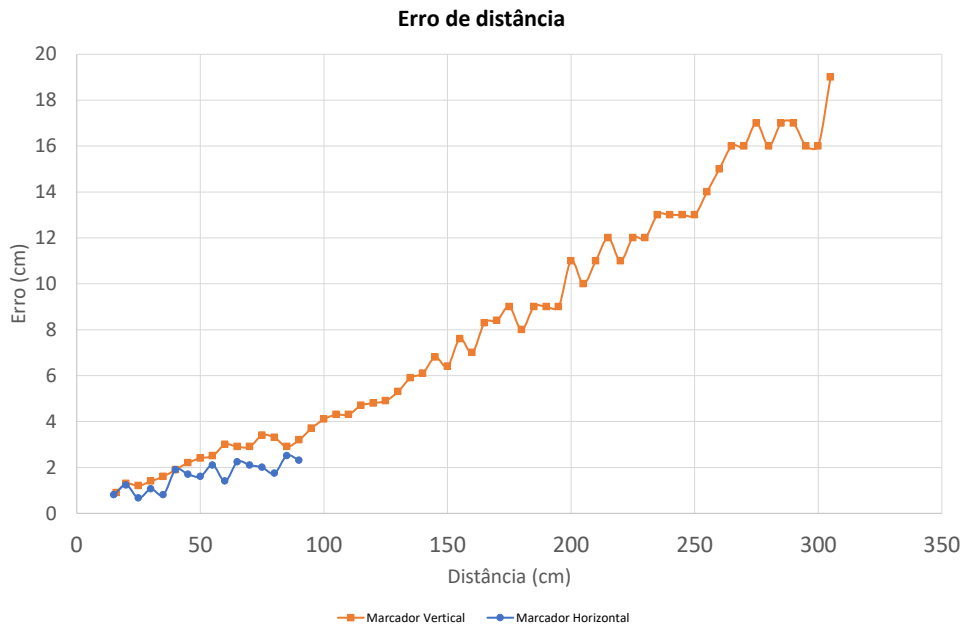

Figura 5.3: Erro de distância ao marcador- Valores obtidos para marcadores com 9,7cm

Por forma a analisar a precisão do sistema, foi efetuado um novo teste, em que foi calculada a localização do robô fixo, em relação a um marcador também fixo. Este procedimento foi executado de forma cíclica, até se obterem 1500 medições. Repetiu-se o processo para várias distâncias e ângulos. De seguida foi calculado o desvio-padrão médio das medições de distância e ângulo. Este valor permite ter uma ideia da precisão do sistema para os diferentes valores de ângulo e distância. Os valores obtidos para marcadores colocados na vertical são apresentados nas figuras 5.8 e 5.9, e nas figuras 5.10 e 5.11 para marcadores colocados na horizontal.

5.2 Modelo para qualidade da localização

Um valor importante em qualquer sistema de localização é a qualidade da mesma. Tendo em conta a utilização de marcadores visuais optou-se por relacionar a qualidade da observação dos mesmos com a qualidade da observação. De forma a quantificar essa qualidade foram analisados os valores do desvio-padrão angular e da distância, em função do número de pixeis do lado menor

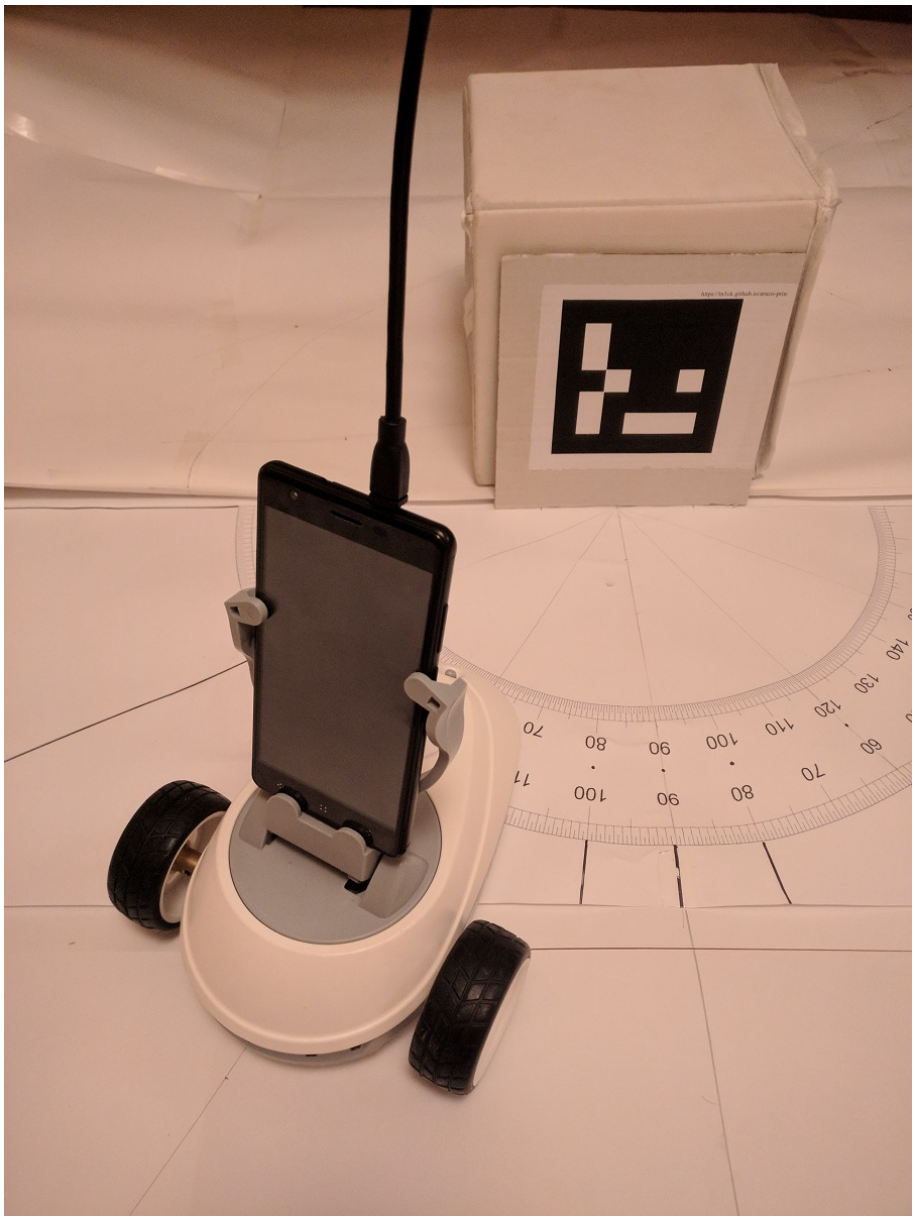

Figura 5.4: Teste para medição do ângulo com marcador vertical

do marcador, na imagem. Os valores estão apresentados nas figuras 5.12 e 5.13, para a distância e ângulo, respetivamente.

Tomando os valores do desvio-padrão como uma medida da qualidade da medida, são utilizadas as seguintes equações, para cálculo da qualidade da medição:

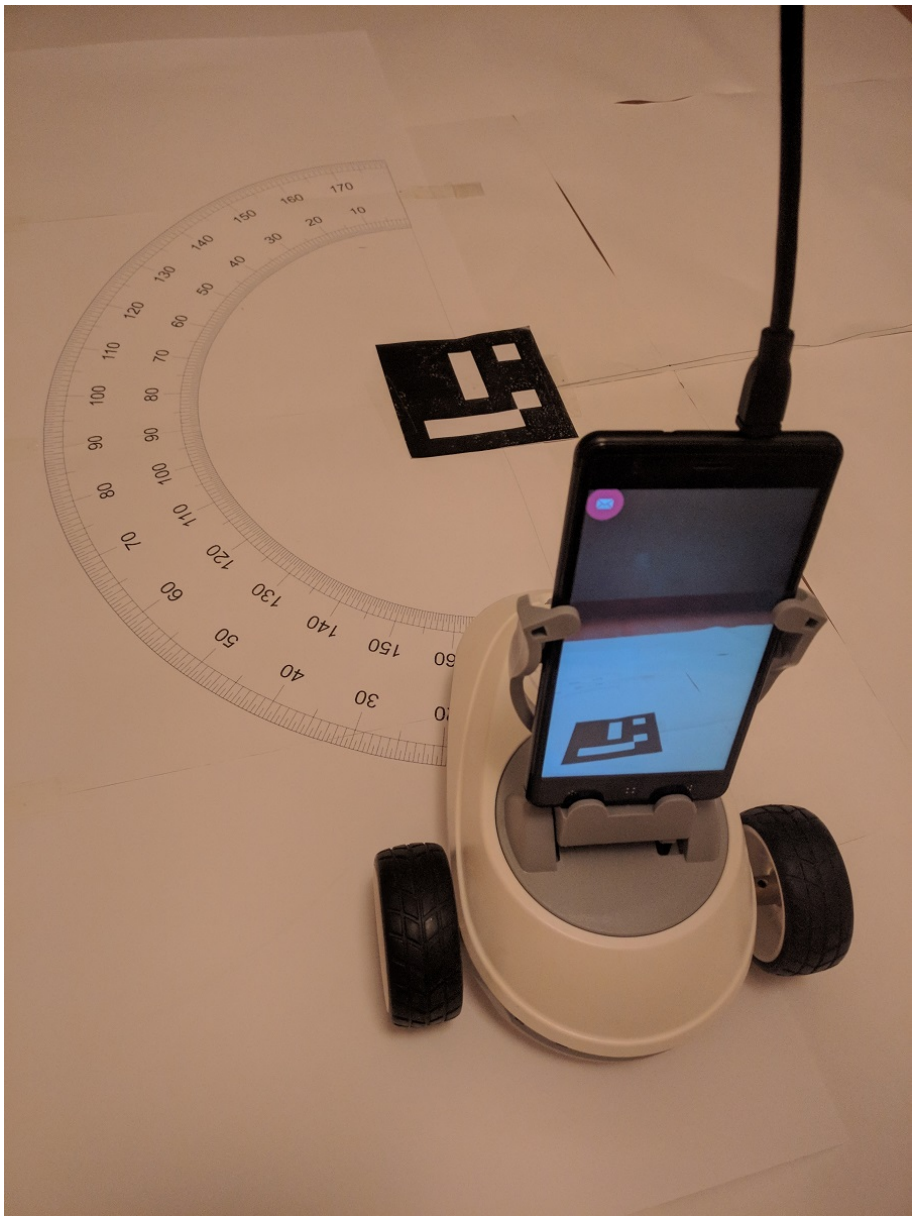

Figura 5.5: Teste para medição do ângulo com marcador horizontal

Modelo para marcador aruco de 9.7cm na vertical (marcador na parede):

$$\sigma_{dist}(l_{min}) = -2 \times 10^{-05} \times l_{min} + 0,0021 \quad [m] \tag{5.1}$$

$$\sigma_{ang}(l_{min}) = -0,0047 \times l_{min} + 0,6422 \quad [^{\circ}] \tag{5.2}$$

Modelo para marcador aruco de 9.7cm na horizontal (marcador no chão):

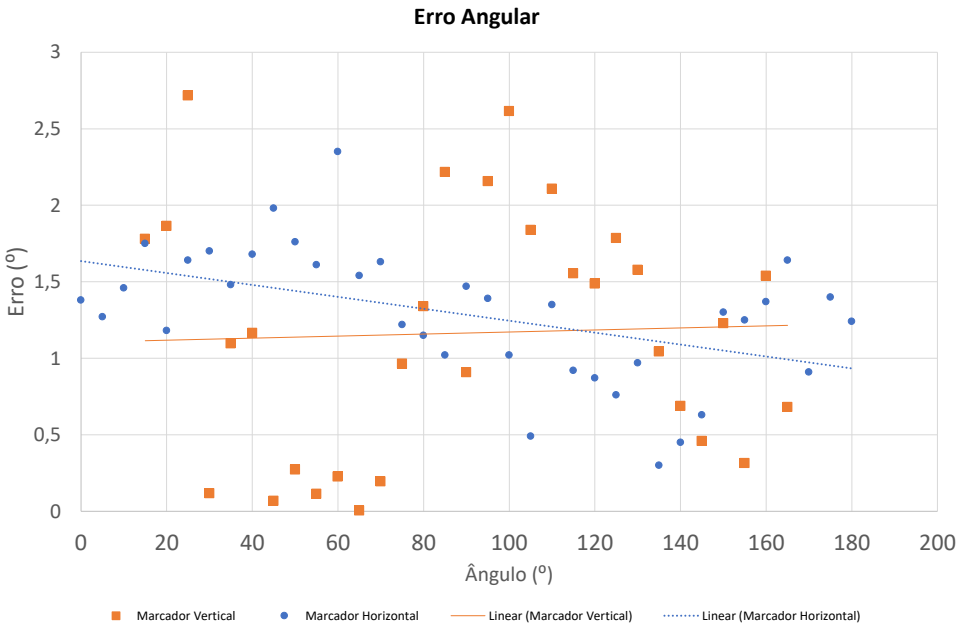

Figura 5.6: Erro angular ao marcador- valores obtidos para marcador com 9,7cm

$$\sigma_{dist}(l_{min}) = -7 \times 10^{-06} \times l_{min} + 0,0014 \quad [m] \tag{5.3}$$

$$\sigma_{ang}(l_{min}) = -0,00111 \times l_{min} + 0,2255 \quad [^{\circ}] \tag{5.4}$$

onde l_{min} representa o comprimento do menor lado do marcador na imagem, em pixeis; os valores σ_{dist} são em metros e os valores σ_{ang} em graus.

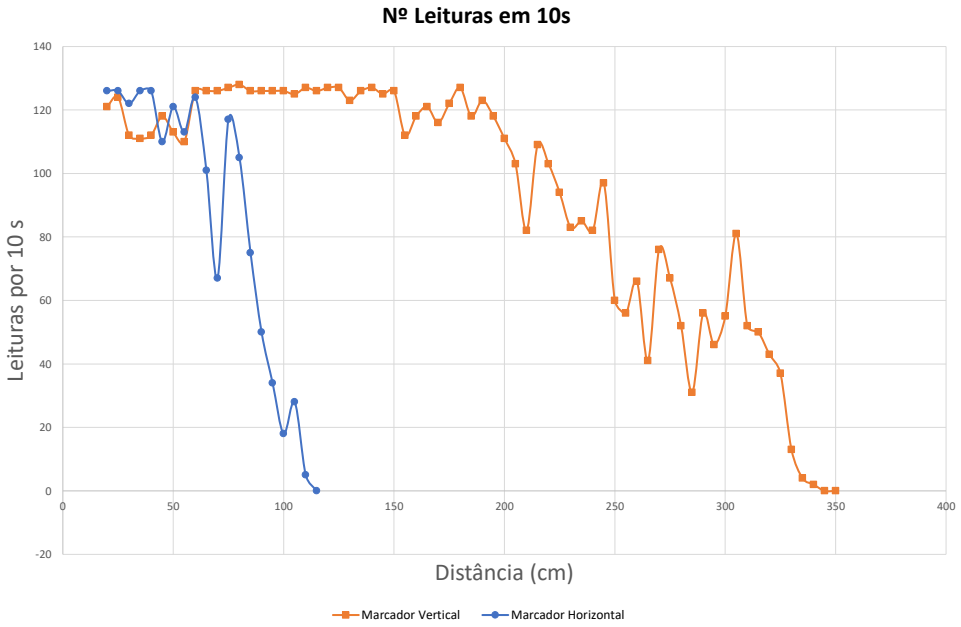

Figura 5.7: Leituras por 10 segundos - Número de leituras registadas durante 10 segundos, em função da distância ao marcador

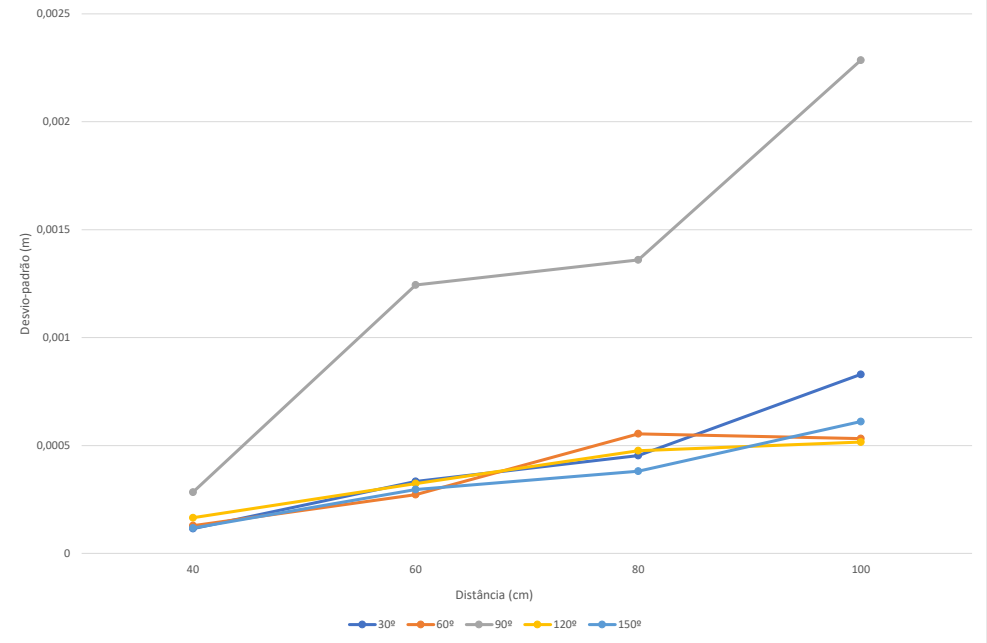

Figura 5.8: Desvio-padrão da distância com marcador vertical

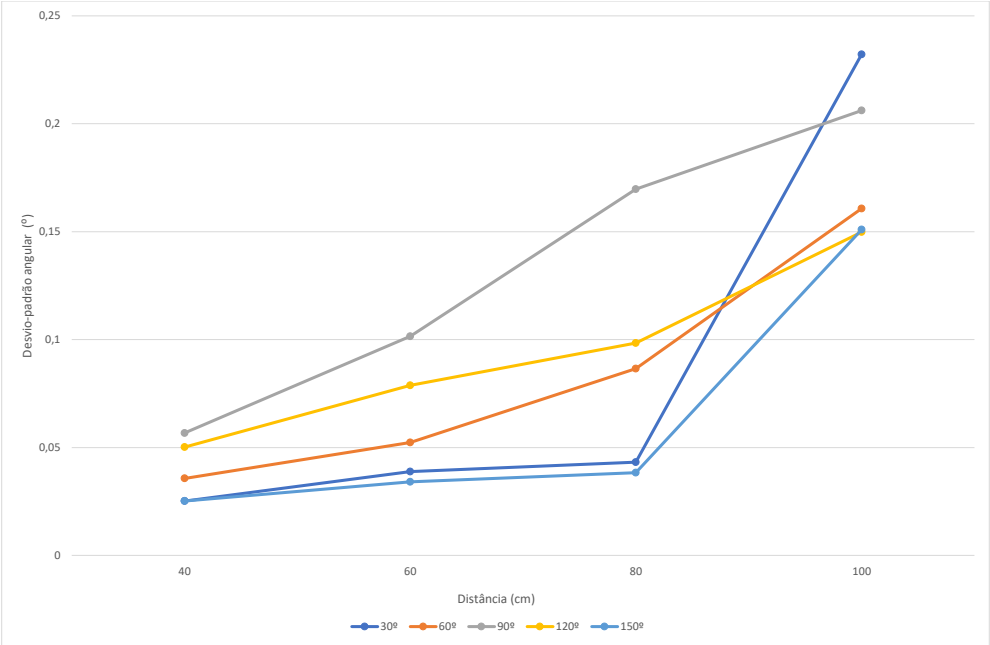

Figura 5.9: Desvio-padrão angular com marcador vertical

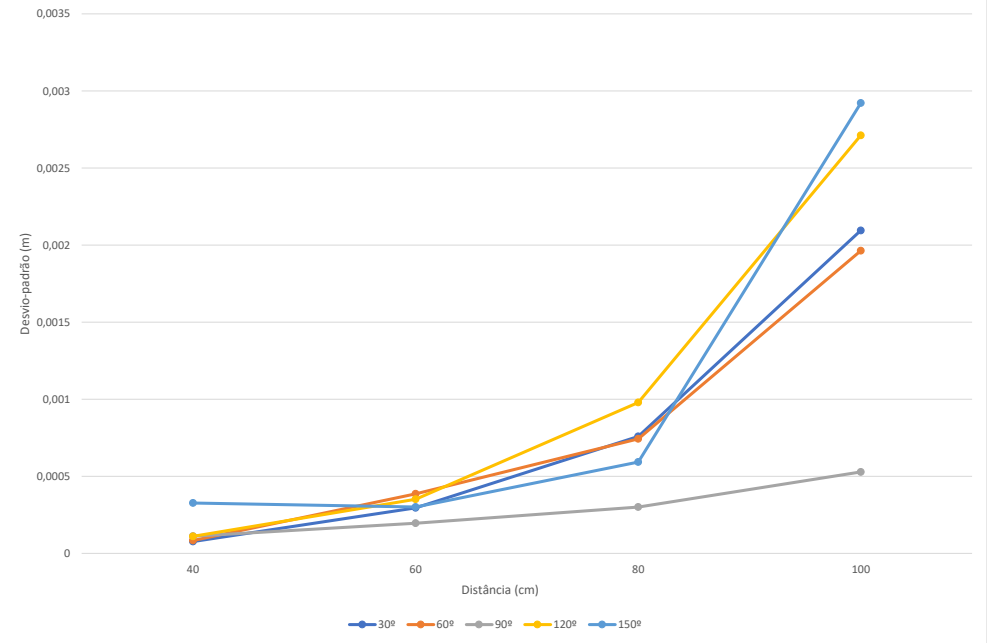

Figura 5.10: Desvio-padrão da distância horizontal

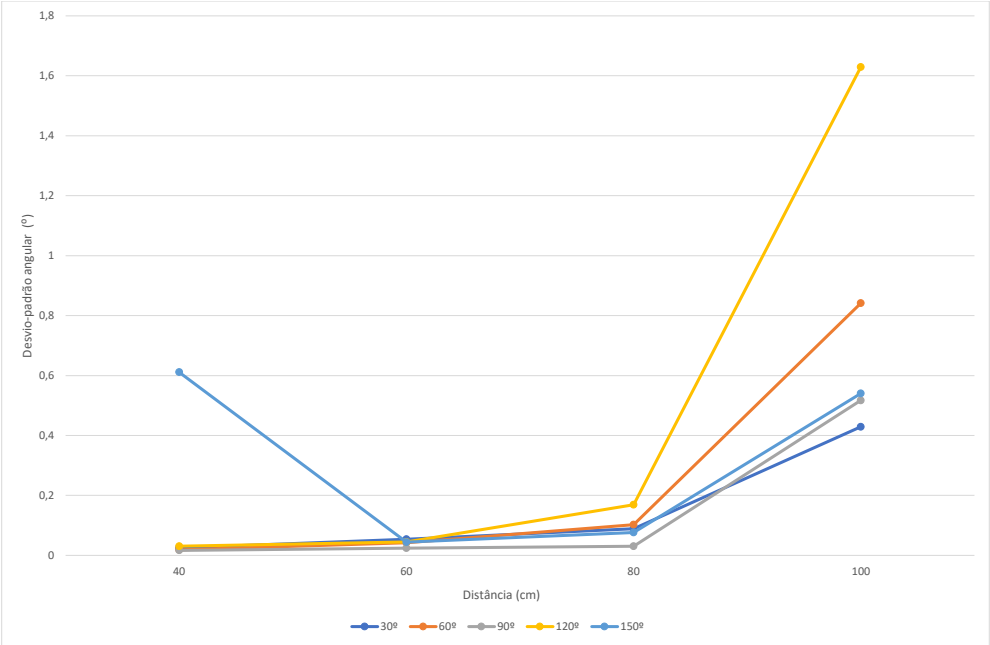

Figura 5.11: Desvio-padrão angular com marcador horizontal

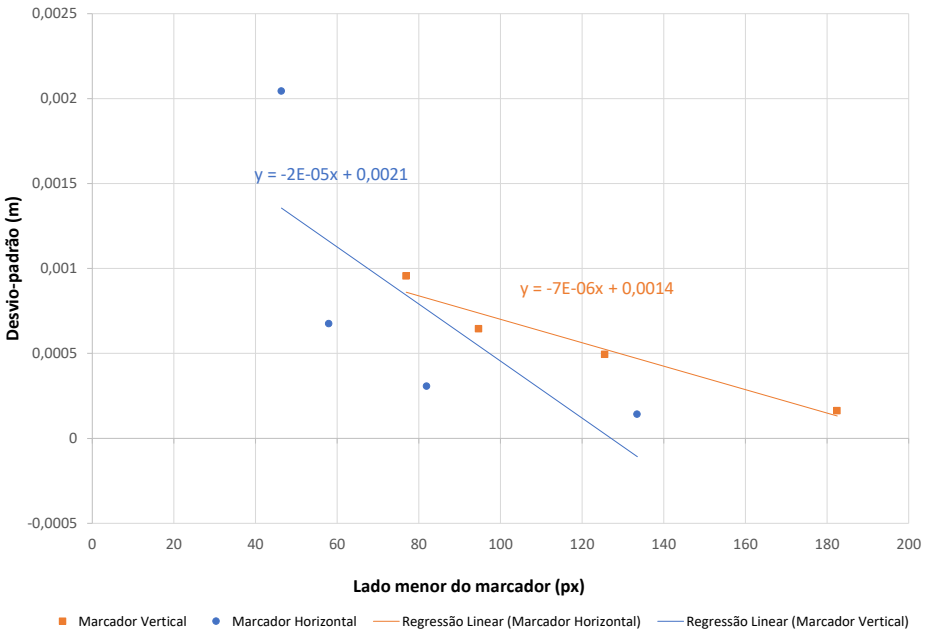

Figura 5.12: Desvio-padrão de distância em função do lado menor do marcador na imagem

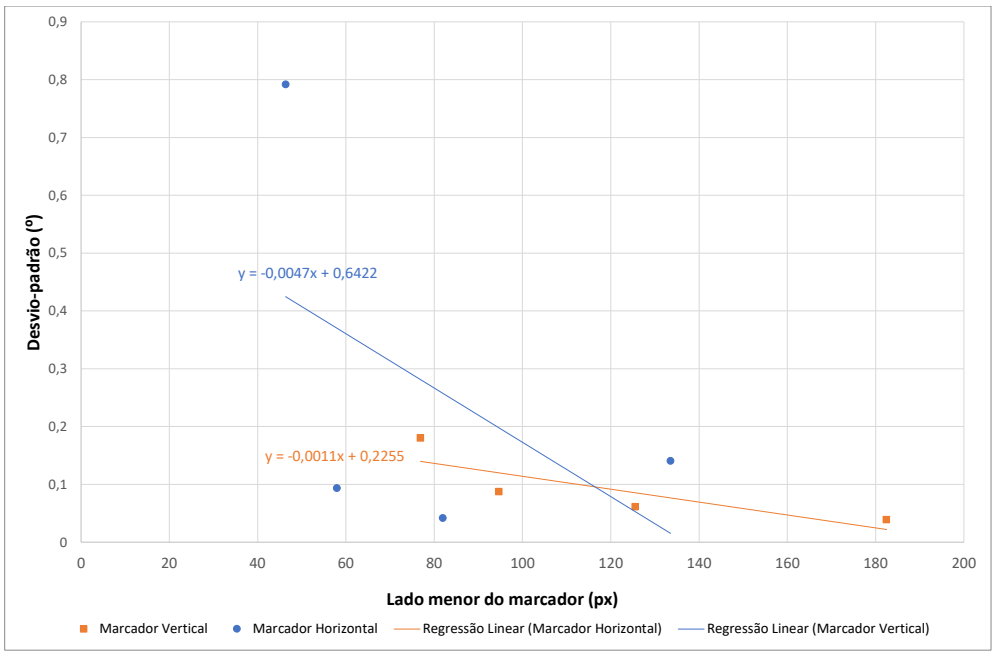

Figura 5.13: Desvio-padrão angular em função do lado menor do marcador na imagem

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Satisfação dos Objectivos

No que diz respeito aos objetivos desta dissertação, os mesmos foram atingidos. Foi criado um sistema independente de hardware externo, que permite a localização do *Robobo*, utilizando como referência marcadores planares *Aruco*. O sistema desenvolvido apresenta a forma de um *Robobo Module*, que pode ser incorporado por outros utilizadores nas suas aplicações. De acordo com os testes realizados, o sistema criado permite a obtenção de uma localização com erro inferior a 0,1m e erro máximo angular de 3 graus para visualizações de marcadores Aruco verticais de 10cm a uma distância de 2 metros. Da mesma forma, a utilização dos mesmos marcadores na vertical resulta em erros da ordem dos 3 cm e 2.5 graus para uma distância de 90cm.

6.2 Contribuições

Com o sistema desenvolvido ao longo desta dissertação, foi adicionada uma funcionalidade de localização *indoor* e baseada em visão ao ecossistema *Robobo*. Com esta adição torna-se possível a utilização de referências posicionais nas aplicações desenvolvidas pelos utilizadores do robô. No geral, com o sistema de localização desenvolvido, o contacto com o *Robobo* aproxima-se mais da utilização de sistemas robóticos mais complexos e que geralmente estão fora do alcance da maioria dos utilizadores.

Apesar de não estar planeada inicialmente, também existiu uma contribuição para a biblioteca Aruco, com a correção do método para obtenção do posicionamento da câmara, no referencial de um marcador observado.

6.3 Trabalho Futuro

Um sistema de localização robótico pode sempre melhorado, pelo aumento de complexidade e funcionalidades. No caso do sistema desenvolvido, pretendeu-se que este fosse de utilização

simples e que não dependesse de competências prévias do utilizador. Contudo, é possível melhorar o sistema, nomeadamente com:

- Desenvolvimento de solução para utilização nos modos de programação Scratch e ROS
- Sincronização com dados de hodometria
- Integração de dados provenientes de múltiplos marcadores
- Utilização simultânea das duas câmaras
- Permitir utilização de marcadores verticais e horizontais em simultâneo
- Utilização de lente olho de peixe
- Introdução de sistema de criação e carregamento de mapas previamente criados
- Introdução de condução autónoma, por exemplo utilizando algoritmos SLAM (Simultaneous localization and mapping)

Referências

[1] Francisco Bellas, Martin Naya, Gervasio Varela, Luis Llamas, Moises Bautista, Abraham Prieto, e Richard J. Duro. Robobo: The Next Generation of Educational Robot. Em Anibal Ollero, Alberto Sanfeliu, Luis Montano, Nuno Lau, e Carlos Cardeira, editores, *ROBOT 2017: Third Iberian Robotics Conference*, volume 694, páginas 359–369. Springer International Publishing, Cham, 2018. DOI: 10.1007/978-3-319-70836-2_30. URL: http://link.springer.com/10.1007/978-3-319-70836-2_30.

[2] The Robobo Project. URL: https://therobomboproject.com/en/.

[3] MINT. URL: http://www.mintforpeople.com/.

[4] Repositórios de programação por blocos para ROBOBO na plataforma BitBucket. URL: https://bitbucket.org/mytechia/robobo-programming/wiki/Blocks.md.

[5] ROS.org | Powering the world's robots. URL: http://www.ros.org/.

[6] Repositórios de programação ROS para ROBOBO na plataforma BitBucket. URL: https://bitbucket.org/mytechia/robobo-programming/wiki/ROS.md.

[7] Repositórios de programação nativa para ROBOBO na plataforma BitBucket. URL: https://bitbucket.org/mytechia/robobo-programming/wiki/Native.md.

[8] J. Hightower e G. Borriello. Location systems for ubiquitous computing. *Computer*, 34(8):57–66, Aug 2001. doi:10.1109/2.940014.

[9] Dimitrios Lymberopoulos, Jie Liu, Xue Yang, Romit Roy Choudhury, Vlado Handziski, e Souvik Sen. A Realistic Evaluation and Comparison of Indoor Location Technologies: Experiences and Lessons Learned. Em *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, IPSN '15, páginas 178–189, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2737095.2737726, doi:10.1145/2737095.2737726.

[10] Qiang Yang, Sinno Jialin Pan, e Vincent Wenchen Zheng. Estimating Location Using Wi-Fi. *IEEE Intelligent Systems*, 23(1):8–13, 2008.

[11] Waltenegus Dargie e Christian Poellabauer. *Fundamentals of wireless sensor networks: theory and practice*. John Wiley & Sons, 2010.

[12] Yongguang Chen e Hisashi Kobayashi. Signal strength based indoor geolocation. Em *Communications, 2002. ICC 2002. IEEE International Conference on*, volume 1, páginas 436–439. IEEE, 2002.

[13] P. Bahl e V. N. Padmanabhan. RADAR: an in-building RF-based user location and tracking system. Em *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 2, páginas 775–784 vol.2, 2000. doi:10.1109/INFCOM.2000.832252.

[14] Moustafa Youssef e Ashok Agrawala. The <Emphasis Type="Italic">Horus</Emphasis> location determination system. *Wireless Networks*, 14(3):357–374, Junho 2008. URL: https://link.springer.com/article/10.1007/s11276-006-0725-7, doi:10.1007/s11276-006-0725-7.

[15] AnyPlace | Indoor Information Service. URL: https://anyplace.cs.ucy.ac.cy/.

[16] Mahmoud Mohanna, Mohamed L. Rabeh, Emad M. Zieur, e Sherif Hekala. Optimization of MUSIC algorithm for angle of arrival estimation in wireless communications. *NRIAG Journal of Astronomy and Geophysics*, 2(1):116–124, Junho 2013. URL: http://www.sciencedirect.com/science/article/pii/S209099771300031X, doi:10.1016/j.nrjag.2013.06.014.

[17] Manikanta Kotaru, Kiran Joshi, Dinesh Bharadia, e Sachin Katti. SpotFi: Decimeter Level Localization Using WiFi. Em *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, páginas 269–282, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2785956.2787487, doi:10.1145/2785956.2787487.

[18] Jie Xiong e Kyle Jamieson. Arraytrack: a fine-grained indoor location system. Usenix, 2013.

[19] Swarun Kumar, Ezzeldin Hamed, Dina Katabi, e Li Erran Li. LTE Radio Analytics Made Easy and Accessible. Em *Proceedings of the 6th Annual Workshop on Wireless of the Students, by the Students, for the Students*, S3 '14, páginas 29–30, New York, NY, USA, 2014. ACM. URL: http://doi.acm.org/10.1145/2645884.2645891, doi:10.1145/2645884.2645891.

[20] S. Lanzisera, D. Zats, e K. S. J. Pister. Radio Frequency Time-of-Flight Distance Measurement for Low-Cost Wireless Sensor Localization. *IEEE Sensors Journal*, 11(3):837–845, Março 2011. doi:10.1109/JSEN.2010.2072496.

[21] Bluetooth Technology Website. URL: https://www.bluetooth.com/.

[22] Specification of the bluetooth system, version 1.0b. URL: http://grouper.ieee.org/groups/802/15/Bluetooth/core_10_b.pdf.

[23] 802.15.1 Offsite Links Page. URL: http://grouper.ieee.org/groups/802/15/Bluetooth/.

[24] A. Kotanen, M. Hannikainen, H. Leppakoski, e T. D. Hamalainen. Experiments on local positioning with Bluetooth. Em *Proceedings ITCC 2003. International Conference on Information Technology: Coding and Computing*, páginas 297–303, Abril 2003. doi:10.1109/ITCC.2003.1197544.

[25] J. Hallberg, M. Nilsson, e K. Synnes. Positioning with Bluetooth. Em *10th International Conference on Telecommunications, 2003. ICT 2003.*, volume 2, páginas 954–958 vol.2, Fevereiro 2003. doi:10.1109/ICTEC.2003.1191568.

[26] G. Anastasi, R. Bandelloni, M. Conti, F. Delmastro, E. Gregori, e G. Mainetto. Experimenting an indoor bluetooth-based positioning service. Em *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, páginas 480–483, Maio 2003. doi:10.1109/ICDCSW.2003.1203598.

[27] Mateusz Korona, Matija Mandić, Vukoman Pejić, e Daniel Sila\d ji. Bluetooth Indoor Positioning. 2015.

[28] Bluetooth SIG. Bluetooth specification version 4.0, Junho 2010.

[29] Ramsey Faragher e Robert Harle. An analysis of the accuracy of bluetooth low energy for indoor positioning applications. Em *Proceedings of the 27th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS+'14)*, páginas 201–210, 2014.

[30] Apple. URL: https://www.apple.com/.

[31] Qusai Alhumoud. *Using iBeacon for Navigation and Proximity Awareness in Smart Buildings*. PhD Thesis, WORCESTER POLYTECHNIC INSTITUTE.

[32] I. A. Ovchinnikov, A. S. Smirnov, e A. M. Tolstaya. Indoor-Navigation Using Internal Sensors of Devices Running Apple iOS and iBeacon Technology. *International Journal of Tomography & Simulation*^{TM}, 30(3):1–13, Julho 2017. URL: http://ceser.in/ceserp/index.php/ijts/article/view/4987.

[33] Bong-Su Cho, Woo-sung Moon, Woo-Jin Seo, e Kwang-Ryul Baek. A dead reckoning localization system for mobile robots using inertial sensors and wheel revolution encoding. *Journal of Mechanical Science and Technology*, 25(11):2907–2917, Novembro 2011. URL: http://link.springer.com/10.1007/s12206-011-0805-1, doi:10.1007/s12206-011-0805-1.

[34] KyuCheol Park, Hakyoung Chung, Jongbin Choi, e Jang Gyu Lee. Dead reckoning navigation for an autonomous mobile robot using a differential encoder and a gyroscope. Em , *8th International Conference on Advanced Robotics, 1997. ICAR '97. Proceedings*, páginas 441–446, Julho 1997. doi:10.1109/ICAR.1997.620219.

[35] D. Scaramuzza e F. Fraundorfer. Visual Odometry [Tutorial]. *IEEE Robotics Automation Magazine*, 18(4):80–92, Dezembro 2011. doi:10.1109/MRA.2011.943233.

[36] Nguyen Xuan Dao, Bum-Jae You, e Sang-Rok Oh. Visual navigation for indoor mobile robots using a single camera. Em *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, páginas 1992–1997, Agosto 2005. doi:10.1109/IROS.2005.1545494.

[37] Sergi Arias Bellot. Visual tag recognition for indoor positioning. Junho 2011. URL: http://upcommons.upc.edu/handle/2099.1/12668.

[38] Chung-Hua Chu, De-Nian Yang, e Ming-Syan Chen. Image Stablization for 2d Barcode in Handheld Devices. Em *Proceedings of the 15th ACM International Conference on Multimedia*, MM '07, páginas 697–706, New York, NY, USA, 2007. ACM. URL: http://doi.acm.org/10.1145/1291233.1291394, doi:10.1145/1291233.1291394.

[39] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, e Dieter Schmalstieg. Pose Tracking from Natural Features on Mobile Phones. Em *Proceedings of the 7th*

IEEE/ACM International Symposium on Mixed and Augmented Reality, ISMAR '08, páginas 125–134, Washington, DC, USA, 2008. IEEE Computer Society. URL: http://dx.doi.org/10.1109/ISMAR.2008.4637338, doi:10.1109/ISMAR.2008.4637338.

[40] M. Fiala. ARTag, a fiducial marker system using digital techniques. Em *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, páginas 590–596 vol. 2, Junho 2005. doi:10.1109/CVPR.2005.74.

[41] Edwin Olson. AprilTag: A robust and flexible visual fiducial system. Em *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, páginas 3400–3407. IEEE, 2011.

[42] John Wang e Edwin Olson. AprilTag 2: Efficient and robust fiducial detection. Em *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, páginas 4193–4198. IEEE, 2016.

[43] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, e M. J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, Junho 2014. URL: http://www.sciencedirect.com/science/article/pii/S0031320314000235, doi:10.1016/j.patcog.2014.01.005.

[44] Francisco J. Romero-Ramirez, Rafael Muñoz-Salinas, e Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76:38–47, Agosto 2018. URL: http://www.sciencedirect.com/science/article/pii/S0262885618300799, doi:10.1016/j.imavis.2018.05.004.

[45] Sidney Berg. Aruco augmented reality library developed for android., Maio 2018. original-date: 2015-04-26T20:41:53Z. URL: https://github.com/sidberg/aruco-android.

[46] Android Studio Website. URL: https://developer.android.com/studio/.

[47] IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains. URL: https://www.jetbrains.com/idea/.

[48] Sheng Liang. *The Java Native interface: programmer's guide and specification*. The Java series. Addison-Wesley, Reading, Mass, 1999.