Universidade de Aveiro, Departamento de Eletrónica, Telecomunicações e Informática, 2017

Modelos Gráficos Probabilísticos no Reconhecimento de Objetos

Autores:

Luís Miguel Santos Castro, nº 76641

Miguel Filipe Carvalhais dos Santos de Matos, nº 76686

Pedro Nuno Gomes Gusmão, nº77867

Tiago Alexandre Melo Almeida, nº 76366

Yuriy Muryn, nº76373

Modelos Gráficos Probabilísticos no Reconhecimento de Objetos

Relatório de Projeto em Informática da Licenciatura de Engenharia Informática da Universidade de Aveiro, realizado por Luís Miguel Santos Castro, Miguel Filipe Carvalhais dos Santos de Matos, Pedro Nuno Gomes Gusmão, Tiago Alexandre Melo Almeida e Yuriy Muryn sob a orientação de Luís Seabra Lopes, Professor Associado do Departamento de Electrónica, Telecomunicações e Informática.

Palavras-Chave

Probabilistic Graphical Model

Conditional Random Field

Plane-Based Map

Machine Learning

Computer Vision

Resumo

Este projeto centra-se na aplicação de uma técnica de reconhecimento de objetos baseada num modelo gráfico probabilístico treinado com amostras sintéticas ou reais. Pode ser útil em várias áreas profissionais, principalmente na área da robótica.

O reconhecimento dos objetos é feito com base nas caraterísticas que estes apresentam e as relações que estabelecem entre si. O facto de haver uma certa organização na localização dos diversos objetos (um ecrã normalmente está em cima de uma mesa) facilita o reconhecimento, pois essa informação pode ser introduzida na forma de conhecimento semântico que nos permitem gerar amostras sintéticas para treinar o modelo. O foco do trabalho é a utilização de amostras sintéticas e não de amostras reais, por não ser necessária a recolha de observações reais, por ser um processo mais prático e por nos poupar uma quantidade de tempo significativa.

Foi ainda implementado um processo que, a partir de uma câmara Kinect e com um conjunto de bibliotecas existentes, consegue extrair de um cenário os planos presentes e as suas características.

Neste documento, apresentamos as tecnologias envolvidas e o fundamento teórico para o objetivo proposto. De seguida, descrevemos a arquitetura e analisamos a implementação da nossa solução.

Testámos a nossa solução em vários cenários, nas salas Universidade de Aveiro e contra *datasets* produzidos na Universidade de Málaga.

Índice

Lista de Figuras

Abreviações

BN	Bayes network
CRF	Conditional Random Fields
DETI	Departamento de Eletrónica Telecomunicações e Informática
LEI	Licenciatura em Engenharia Informática
MLE	Maximum Likelihood Estimation
MRF	Markov Random Field
MRPT	Mobile Robot Programming Toolkit
OWL	Ontology Web Language
PbMap	Plane-based Map
PGM	Probabilistic Graphical Model
RDF	Resource Description Framework
UPGMpp	Undirected Probabilistic Graphical Models in C++

Capítulo 1

Este capítulo serve para contextualizar o trabalho desenvolvido e indicar a organização do relatório.

1 Introdução

O nosso projeto consiste no reconhecimento de objetos com base nas caraterísticas que estes apresentam e as relações que estabelecem entre si. O facto de haver uma certa organização na localização dos diversos objetos (por exemplo, um ecrã normalmente está em cima de uma mesa) facilita esse reconhecimento.

Em vez de utilizarmos amostras reais, o que pressupõe uma recolha exaustiva das mesmas, geramos amostras sintéticas para treinar um modelo probabilístico. Para demonstrar este funcionamento usamos uma Kinect para, a partir de imagens RGB-D (imagem RGB com profundidade), obter os planos e as suas caraterísticas. Cada plano identificado corresponde assim a um objeto a ser identificado.

1.1 Contexto

Este projeto está integrado na Licenciatura em Engenharia Informática (LEI) lecionada na Universidade de Aveiro. Foi desenvolvido durante o segundo semestre do ano letivo 2016/2017 na cadeira de Projeto em Informática.

Este trabalho foi desenvolvido com a orientação do Prof. Luís Seabra Lopes e baseia-se na tese de doutoramento "*Probabilistic Techniques in Semantic Mapping for Mobile Robotics*" [1] por J. R.Ruiz Sarmiento da Universidade de Málaga, onde o nosso orientador foi júri.

1.2 Motivação

O reconhecimento de objetos é uma área bastante estudada nos dias de hoje. Existem muitas técnicas de o fazer e tem inúmeras aplicações, sendo que a principal é na área da Robótica.

Com isso, a nossa motivação foi explorar uma destas técnicas, nomeadamente a descrita em [1], e aprender mais sobre os conceitos de *machine learning*.

1.3 Objetivos

O principal objetivo deste projeto é o reconhecimento de objetos sabendo as suas caraterísticas e as relações que estes estabelecem entre si. O segundo objetivo é demonstrar este funcionamento recorrendo à uma Kinect para extrair estas caraterísticas e relações.

1.4 Estrutura do documento

Este relatório está organizado da seguinte forma:

Capítulo 2 - Neste capítulo será feita uma referência para os trabalhos relacionados, às tecnologias usadas e será feita uma introdução teórica aos tópicos abordados no trabalho.

Capítulo 3 - Neste capítulo pretendemos dar a conhecer e descrever detalhadamente a solução utilizada para o reconhecimento de objetos.

Capítulo 4 - Neste capítulo iremos demonstrar as soluções que foram implementadas e como podem ser usadas.

Capítulo 5 - Neste capítulo serão apresentados os resultados obtidos.

Capítulo 6 - Neste capítulo será apresenta a conclusão.

Capítulo 2

Neste capítulo será feita uma referência para os trabalhos relacionados, às tecnologias usadas e será feita uma introdução teórica aos tópicos abordados no trabalho.

2 Estado de Arte

Existe uma literatura muito extensa sobre o reconhecimento de objetos. Em muitos casos este reconhecimento só tem em conta as características visuais de objetos isolados, sem considerar a informação das relações que estes objetos possam estabelecer entre si.

A abordagem que nós seguimos explora, para além das características de cada objeto, a informação contextual entre estes. Esta abordagem está descrita em [1]–[4] e usa os Modelos Gráficos Probabilísticos (PGMs), nomeadamente os Campos Condicionais Aleatórios (CRF), para gerir esta informação contextual.

Para os PGMs, a representação das amostras pode surgir de várias formas, provindo de um *dataset* ou da captura de imagens de um cenário. Na nossa abordagem utilizamos a captura de imagens do tipo RGB-D, imagem RGB com profundidade, a partir de uma Kinect. Esta abordagem foi baseada no artigo [5]. Nesta abordagem, após a captura de imagens, é feita uma reconstrução do cenário e posteriormente a identificação dos planos contidos no cenário para a extração das características desses planos (por exemplo, área e orientação) e relações (por exemplo, se estão próximos). Para este fim foram utilizadas uma biblioteca para a extração dos planos (MRPT módulo PbMap) e uma aplicação para obtenção da posição e orientação da câmara (Kinect-6D-SLAM.)

2.1 Trabalho Relacionado

2.1.1 “Probabilistic Techniques in Semantic Mapping for Mobile Robotics”

A componente principal do nosso trabalho é baseada na tese de doutoramento de J. R.Ruiz Sarmiento [1], bem como alguns artigos que o mesmo publicou [2]–[4]

Na tese primeiramente é nos apresentado uma técnica para reconhecimento de objetos, seguidamente, o autor apresenta uma solução conjunta do problema de reconhecimento de objetos e de divisões, onde o objetivo seria reconhecer os objetos e o cenário onde eles estão. A abordagem conjunta ao problema de reconhecimento

de objetos e divisões possibilita que os objetos ajudem a identificar corretamente as divisões e vice-versa.

No nosso projeto nós só abordamos o primeiro problema, isto é, problema de reconhecimento de objetos.

2.1.2 “Fast place recognition with plane-based maps”

A extração dos planos contidos no cenário foi baseada em [5]. Do artigo apenas uma parte foi considerada, parte essa referente à extração de planos a partir de imagens RGB-D. Esta parte consiste na construção de um *plane-based map* (PbMap) do cenário capturado na imagem RGB-D para posterior extração das características dos planos e suas relações.

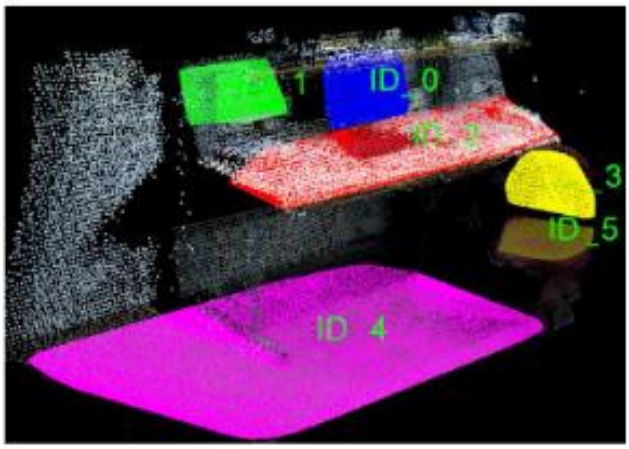


Figura 1 - Exemplo demonstrativo da deteção dos planos [4]

A Figura 1 representa o resultado final da aplicação do algoritmo de *plane-based mapping* apresentado no artigo, podemos ver que vários planos foram identificados e a estes estão associadas determinadas características, como por exemplo, área, orientação, entre outros.

2.2 Fundamento Teórico

2.2.1 Conhecimento Semântico (Ontologia)

Muitos sistemas tradicionais de reconhecimento de objetos dependem apenas dos dados sensoriais que recolhem. Em diversos cenários existem padrões na organização de objetos, muitas vezes devido à funcionalidade dos mesmos (por exemplo, uma televisão encontra-se à frente de um sofá). Torna-se possível explorar essas relações contextuais na forma de conhecimento semântico, o que possibilita um aumento na *performance* do sistema que contrasta com as criações tradicionais.

Ao atribuirmos conhecimento semântico ao sistema de reconhecimento de objetos (em vez de existir apenas a recolha sensorial) limitamos o conjunto de objetos

possíveis para o reconhecimento e permitimos ao sistema explorar certas relações contextuais que, consequentemente, melhoram os resultados.

Este conhecimento semântico é fornecido por humanos, naturalmente na forma de uma ontologia. Uma ontologia é utilizada como uma forma de representar um conjunto de classes, as suas propriedades e as relações entre elas num determinado domínio. No nosso caso, é fácil associar o domínio ao cenário (por exemplo ume escritório), as classes aos objetos (por exemplo, Cadeira), as propriedades às caraterísticas dos objetos (por exemplo, área) e as relações referem-se às relações de posição entre os objetos (por exemplo, sobreposição).

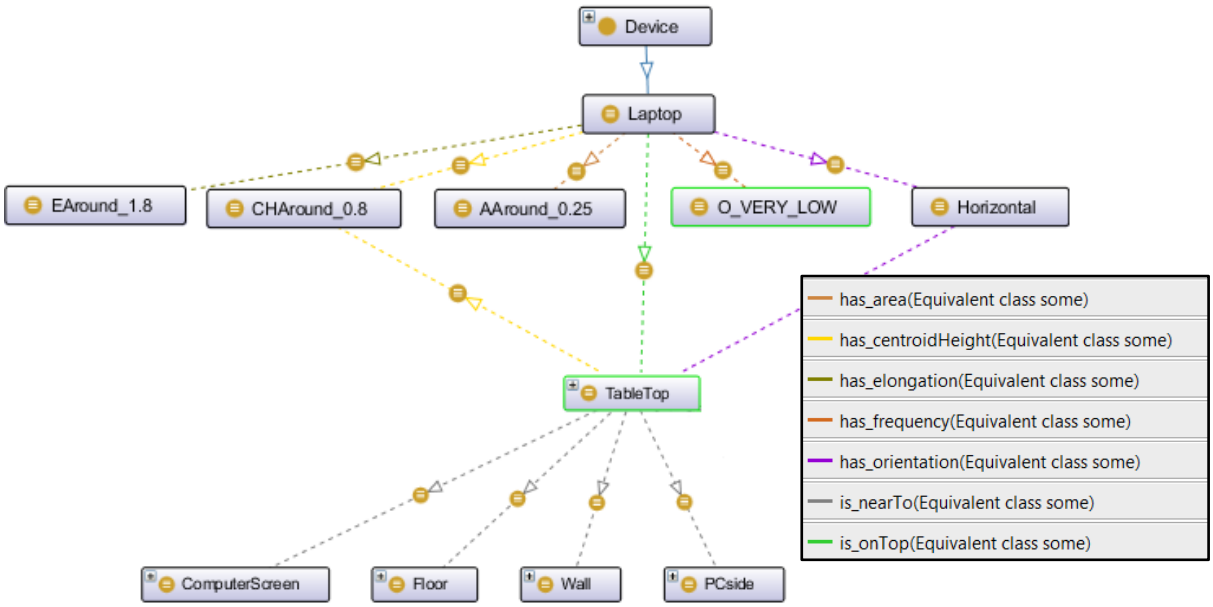


Figura 2 - Representação da parte de uma ontologia.

Na Figura 2 podemos observar que por exemplo, o *Laptop* tem uma área de $0.25m^2$ e está em cima da TableTop que possui uma orientação horizontal.

2.2.2 Modelo Gráfico Probabilístico

Muitas tarefas necessitam que pessoas ou sistemas autónomos raciocinem e retirem conclusões baseadas na informação disponível. O *framework* de PGM oferece uma abordagem para essa tarefa, permitindo a criação de modelos para interpretação e manipulação dessa informação podendo inferir conclusões. Estes modelos também podem aprender automaticamente através de informação de treino, permitindo assim a utilização deste mecanismo em problemas cujo a construção manual do modelo é difícil ou até mesmo impossível [6].

Os PGMs têm inúmeras aplicações em *machine learning*, *computer vision*, *natural language processing* e *computational biology*. Um PGM é representado por um grafo $G = (V, E)$. O conjunto V representa as variáveis aleatórias do problema e frequentemente são referenciadas como nós, enquanto que as arestas E representam

variáveis aleatórias que são dependentes de alguma forma. Podemos então modelar uma grande quantidade de variáveis aleatórias e interações complexas entre elas. Também suporta a execução de técnicas de inferência probabilística para prever valores de variáveis aleatórias.

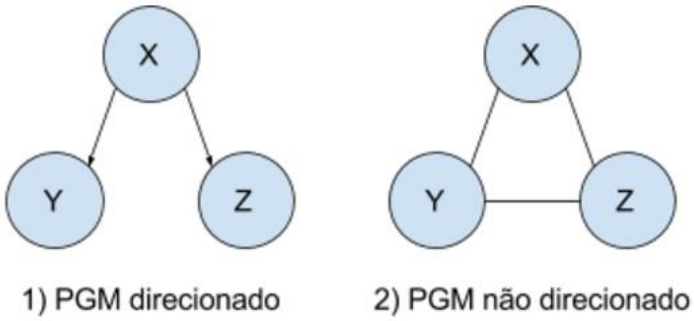


Figura 3 - PGMs direcionados e não direcionados

Os PGMs podem ser agrupados em modelos direcionados (Figura 3, 1) e não direcionados (Figura 3, 2). No caso dos modelos direcionados existem as Redes de *Bayes* (BN) que modelam as dependências entre nós através de arestas direcionadas que representam relação de causalidade, este tipo de modelo tem sido utilizado em diagnósticos médicos, biologia, previsão do tempo etc. Por outro lado, os modelos não direcionados, como por exemplo os campos aleatórios de *Markov* (MRF), usam arestas não direcionadas para definir relações simétricas entre variáveis aleatórias.

Pegando no problema inicial de reconhecimento de objetos com base nas caraterísticas e relações entre estes, podemos observar que um modelo não direcionado é o mais indicado, pois as relações entre objetos não dependem da direção. Numa relação de proximidade, por exemplo, tanto faz se a relação é de A para B ou de B para A.

Como referido anteriormente também são consideradas as caraterísticas dos objetos como informação observada para os classificar. Nesta perspetiva o mais indicado é a utilização de CRF, que permite condicionar as variáveis aleatórias sobre um conjunto de informações observadas, neste caso as caraterísticas dos objetos.

Depois será utilizado um exemplo para ajudar a definir e compreender MRF e CRF, mas antes e para ajudar a contextualizar podemos dizer que MRF pretende calcular $P(y)$ sendo y uma variável aleatória podendo assumir diversos estados enquanto que a CRF pretende calcular a $P(y|x)$ onde x representa um conjunto de observações associadas à variável aleatória y.

2.2.2.1 Exemplo ilustrativo para definir MRF e CRF

(Este exemplo foi retirado e adaptado de [1]*)*

Considerando a seguinte família composta pelo Bob (B), a Sofia (S), a Alice (A) e a Teresa (T), e o problema de modelar o estado emocional dos elementos desta família, isto é, se estão felizes (1) ou tristes (0).

Sendo os quatro elementos da família seres humanos sociáveis, podemos assumir que a felicidade destes depende da interação uns com os outros. Deste modo a relação entre os elementos da família também deve ser tida em conta para modelar o estado emocional destes. Sendo assim, podemos utilizar o MRF para tentar modelar o estado emocional desta família.

Assumindo as seguintes relações entre eles:

- Teresa só comunica com o Bob
- Alice, Sofia e Bob comunicam entre eles.

Dada esta informação podemos construir o modelo gráfico representado na Figura 4.

Podemos observar que por exemplo a felicidade da Teresa é influenciada e influencia o Bob. Em vez de calcular a probabilidade P(y), sendo o y uma distribuição possível, por exemplo y = [A=1, B=0, S=1, T=1], pode-se partir este problema em pedaços mais pequenos através de fatores, também chamados de potenciais. que correspondem a funções definidas sobre diferentes partes do grafo. Normalmente existem dois tipos de fatores, os fatores unários e os emparelhados (também chamados *pairwise*).

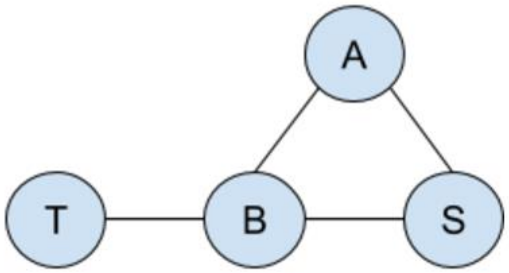


Figura 4 - Representação do modelo MRF para o exemplo de felicidade

Os fatores unários representam a possibilidade desses nós estarem num determinado estado, neste caso como já foi referido é o estado de felicidade (feliz ou infeliz).

	$\phi(A)$	$\phi(B)$	$\phi(S)$	$\phi(T)$
Infeliz (0)	0.3	0.4	0.4	0.5
Feliz (1)	0.7	0.6	0.6	0.5

Tabela 1 - Fator unário das quatro variáveis aleatórias. ϕ(y) representa um fator unário aplicado a uma determinada variável aleatória.

Observando a Tabela 1 podemos constatar que a Alice tem maior possibilidade estar num estado emocional de felicidade do que a Teresa.

Os fatores *pairwise* definem a possibilidade de dois nós estarem numa determinada combinação de estados.

$\phi(A,B) = \phi(A,S) = \phi(B,S)$		
Infeliz (0)	Infeliz (0)	1
Infeliz (0)	Feliz (1)	0.5
Feliz (1)	Infeliz (0)	0.5
Feliz (1)	Feliz (1)	1

$\phi(B,T)$		
Infeliz (0)	Infeliz (0)	1
Infeliz (0)	Feliz (1)	0.9
Feliz (1)	Infeliz (0)	0.9
Feliz (1)	Feliz (1)	1

Tabela 2 - ϕ(y1,y2) representa um fator pairwise aplicado a um par de variável aleatória conectadas no grafo.

Observando a Tabela 2 podemos constatar que existe uma maior possibilidade local do Bob estar com o mesmo estado de felicidade do que a teresa do que estar com um estado diferente.

É possível notar que estes valores não representam probabilidades, visto que a soma destes não é igual a 1, mas sim afinidades, possibilidades.

Neste simples exemplo podemos ver que a representação de $P(y)$ através de MRF, ou seja, utilizando os fatores unários e *pairwise* para simplificar a distribuição probabilística do problema não nos simplificou o trabalho visto que, se tivéssemos definido exaustivamente $P(y)$ seria equivalente a $2^4 = 16$ probabilidades, mas também é evidente que se o número de variáveis aleatórias aumenta-se, isto é, se o y = 20 e se os estados possíveis continuassem a ser binários usando o mesmo exemplo feliz ou infeliz, a definição exaustiva de P(y) seria igual a $2^{20} \simeq 10^6$ probabilidades, e aqui uma abordagem MRF iria facilitar o problema.

Continuando a abordagem com MRF, então o cálculo de $P(y)$ segundo o teorema de *Hammersley-Clifford* [7], a probabilidade $P(y)$ pode ser fatorizada sobre Grafo G como um produto dos fatores ϕ.

$$P(y) = \frac{1}{Z}\prod_{c \in C} \phi(y_c) \quad \text{(eq 1)}$$

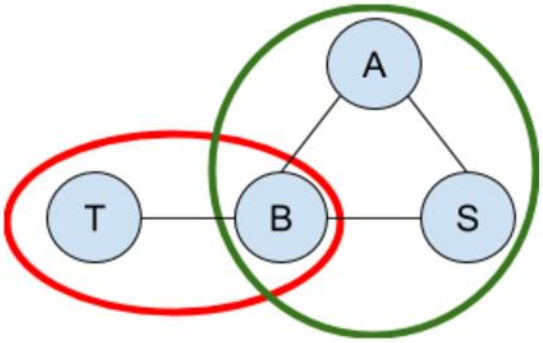


Figura 5 - Representação dos maximal cliques no grafo.

C é um conjunto de *maximal cliques* do grafo G, a Figura 5 representa esse conjunto para este exemplo, e podemos observar que o grafo contém dois *maximal cliques* $c_1 = \{T, B\}\, e\, c_2 = \{B, A, S\}$. A constante Z é também chamada de função de partição que permite fazer a normalização de modo que $\sum\ p(y) = 1$, para todos os possíveis valores de y, pelo que a computação da função de partição é necessária para o cálculo de $P(y)$ para qualquer atribuição possível, tendo por base esta informação podemos definir Zda seguinte forma.

$$Z = \sum_{y} \prod_{c \in C} \phi(y_c\ )$$

Tendo em mente que o grafo tem dois *maximal cliques*, $c_1 = \{B, T\}\, e\, c_2 = \{B, A, S\}$, podemos reescrever a equação 1 aplicada ao nosso exemplo.

$$P(A, B, T, S) = \frac{1}{Z} \cdot \phi(B, T) \cdot \phi(A, B, S)$$

Como é evidente pela equação em cima, os fatores que envolvem muitas variáveis não são trivialmente definidos, neste caso $\phi(A, B, S)$, por isso uma medida comum utilizada é considerar apenas os fatores unários e emparelhados do grafos e não utilizar com um maior número de variáveis, sendo assim equação 1 pode ser reescrita da seguinte forma.

$$P(y) = \frac{1}{Z} \cdot \prod_{i \in Vertices} \phi(y_i) \cdot \prod_{i-j \in Arestas} \phi(y_i, y_j)$$

Que aplicando ao nosso exemplo ficamos com a seguinte expressão para $P(y)$.

$$P(y) = \frac{1}{Z} \cdot \phi(A) \cdot \phi(B) \cdot \phi(S) \cdot \phi(T) \cdot \phi(A, B) \cdot \phi(A, S) \cdot \phi(B, S) \cdot \phi(B, T)$$

Agora para calcular $P(y)$só nos falta a constante (função de partição) Z.

	P(y) não normalizado	P(y) normalizado
P(A=0,B=0,S=0,T=0)	0.024	0.053186
P(A=0,B=0,S=0,T=1)	0.0216	0.047867
P(A=0,B=0,S=1,T=0)	0.009	0.019945
P(A=0,B=0,S=1,T=1)	0.0081	0.01795
P(A=0,B=1,S=0,T=0)	0.0081	0.01795
P(A=0,B=1,S=0,T=1)	0.009	0.019945
P(A=0,B=1,S=1,T=0)	0.01215	0.026925
P(A=0,B=1,S=1,T=1)	0.0135	0.029917
P(A=1,B=0,S=0,T=0)	0.014	0.031025
P(A=1,B=0,S=0,T=1)	0.0126	0.027922
P(A=1,B=0,S=1,T=0)	0.021	0.046537
P(A=1,B=0,S=1,T=1)	0.0189	0.041884
P(A=1,B=1,S=0,T=0)	0.0189	0.041884
P(A=1,B=1,S=0,T=1)	0.021	0.046537
P(A=1,B=1,S=1,T=0)	0.1134	0.251302
P(A=1,B=1,S=1,T=1)	0.126	0.279224
Constante Z	0.45125	

Tabela 3 - Cálculo da probabilidade conjunta para o exemplo da felicidade.

Com base na Tabela 3 conseguimos observar o valor correspondente à constante Z (função de partição) é 0.45125 e com este valor já podemos calcular as diferentes distribuições de probabilidades, que estão apresentadas na coluna da direita.

A definição dos fatores tanto unários como emparelhados é complicada visto que a felicidade de uma pessoa depende de múltiplos outros aspetos, como por exemplo, horas de sono, sucesso no trabalho, horas gastas com a família entre outras, estes aspetos deveriam ser tidos em conta no modelo. Seria possível incluir estes novos aspetos num grafo MRF como variáveis aleatórias adicionais, mas iria aumentar complexidade do problema. Podemos então utilizar os CRFs que evitam a necessidade da criação de mais variáveis aleatórias para modelar estes aspetos através do cálculo da probabilidade condicional de y sabendo os aspetos observados, que são referidos como caraterísticas (Figura 6).

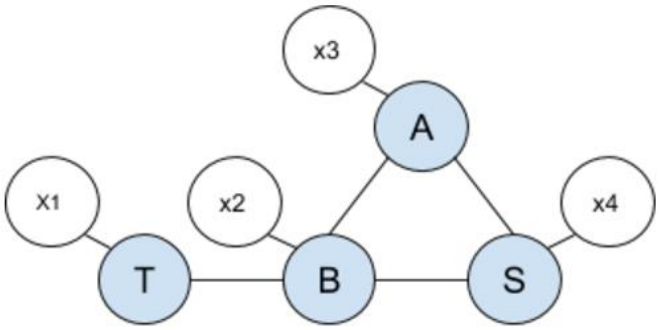


Figura 6 - Representação da CRF incluindo as caraterísticas observadas para cada variável aleatória

O CRF pretende calcular a $P(y|x)$ sendo x um vetor contendos as caraterísticas (aspetos) observadas e em vez de os fatores de cada observação serem definidos à mão, eles são parametrizados a partir de um vetor de pesos θ que é aprendido durante a fase de treino da CRF. Podemos calcular $P(y|x)$da seguinte forma:

$$P(y|x;\theta) = \frac{1}{Z(x,\theta)} \prod_{c \in C} \phi(y_c, x_c, \theta_c) \text{ (eq 2)}$$

A parametrização dos fatores pode ser formulada de diferentes maneiras dependendo da aplicação, mais a frente vamos ver um exemplo de formulação usada para o reconhecimento de objetos.

2.2.2.2 Aprendizagem automática do modelo

Como foi referido, muitos dos modelos são difíceis de construir à mão. O termo construir está se a referir à definição dos fatores como foi feito no exemplo anterior, as Tabelas 1 e 2. Desta forma, uma abordagem muito recorrente é deixar que seja o modelo a definir esses mesmo fatores. A este processo dá-se o nome treino da CRF.

O treino de um modelo CRF para um dado domínio requer a estimação dos parâmetros de peso θ, de modo a maximizar a probabilidade da equação 2 dado um determinado *dataset* (conjunto de dados) de treino $D = [d^1, \ldots d^m]$, o que equivale a seguinte equação.

$$max_\theta L_p(\theta : D) = max_\theta \prod_{i=1}^{m} p(y^i|x^i;\theta) \text{ (eq 3)}$$

A otimização presente na equação 3 é conhecida por *Maximum Likelihood Estimation* (MLE) que como já foi referido significa estimar os pesos θ para onde o cálculo da probabilidade da equação 1 é máximo para um dado domínio, ou seja, *dataset*. Ainda no contexto da equação 3 o *dataset* é composto com um total de m amostras e cada amostra $d^i = (y^i, x^i)$ contém um vetor de caraterísticas observadas x^ie o correspondente *ground truth* que corresponde à classificação de y^i, seguindo o exemplo corresponderia a dizer se a pessoa está feliz ou infeliz dadas as caraterísticas em x.

2.2.2.3 Inferência probabilística

Tendo a CRF treinada e a sua representação em grafo para um dado problema, esta pode ser explorada através de métodos de inferência. Nesta perspetiva existem dois tipos de inferências relevantes, *Maximum a Posteriori* (MAP) *query* e a *Marginal query.*

Neste contexto temos mais interesse na MAP que permite encontrar a atribuição mais provável de $\hat{y}$ para a variável aleatória y, isto é:

$$\hat{y} = arg\ max_y P(y|x;\theta) \text{ (eq 4)}$$

2.2.2.4 CRF aplicadas ao problema de reconhecimento de objetos

O problema de reconhecimento de objetos está ligado à área da *computer vision* tem o objetivo de encontrar ou identificar objetos em imagens ou vídeo. As CRFs têm sido cada vez mais utilizados como base de soluções que tentam solucionar este problema.

Como já foi referido em cima nós vamos aplicar as CRFs com o objetivo de classificar os objetos presentes em uma cena, isto é, será calculada a $P(y|x)$ sendo o $x = [x_1, x_2, \ldots, x_n]$o conjunto de observações dos objetos no cenário onde se encontram, e o $y = [y_1, y_2, \ldots, y_n]$ são as variáveis aleatórias que podem assumir qualquer estado do conjunto L, que representa as classes (nomes) dos objetos possíveis. valores de L

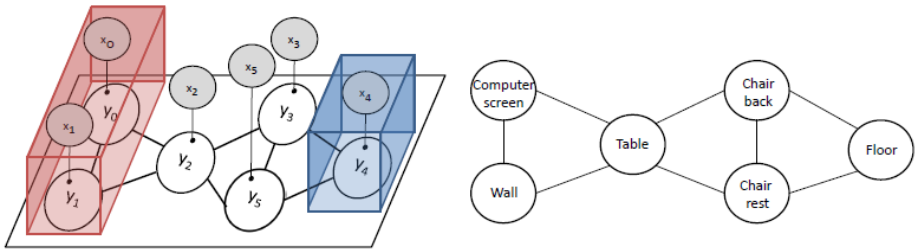


Figura 7 - Representação da CRF lado esquerdo e os correspondentes objetos lado direito [4]

Relembrando a teoria apresentada em cima podemos expressar a factorização da $P(y|x)$ sobre um grafo, neste caso a CRF, como o produto dos seus fatores, como está formulado na equação 2.

Neste trabalho, para o reconhecimento de objetos serão apenas considerados dois tipos de fatores, os unários e os emparelhados (*pairwise)*. Os fatores unários codificam o conhecimento sobre as propriedades do objeto, neste caso corresponde à possibilidade de classificação do objeto tendo em conta as observações. Os fatores emparelhados codificam o conhecimento sobre as relações dos objetos, que neste caso corresponde à possibilidade de dois objetos terem uma determinada classificação (nome de classe atribuído).

As observações do objeto são usadas no cálculo do fator unário e emparelhado, neste trabalho consideramos estas possíveis observações que podem ser decompostas em unárias e emparelhadas chamando-se caraterísticas unárias e emparelhadas.

Caraterísticas unárias	Caraterísticas emparelhadas
Altura até ao chão	Perpendicularidade
Orientação	Relação de sobreposição
Área	Distância vertical entre centros
Elongação	

Tabela 4 - Caraterísticas unárias e emparelhadas

Pegando agora a equação 2 e relembrando que foi dito que a parametrização dos fatores depende do tipo de aplicação a que se destina, no contexto de reconhecimento de objetos, podemos definir o fator unário $\phi(y_i, x_i, \theta\ )$da seguinte forma.

$$\phi(y_i, x_i, \theta\ ) = \sum_{l \epsilon L} \delta(y_i = l) \cdot \theta_l \cdot f(x_i)$$

Onde $f(x_i)$ é a função que computa o vetor de características unárias do objeto x_i, $\theta_l$corresponde ao vetor de pesos para cada classe l obtido durante a fase de treino, e $\delta(y_i = l)$ é a função *Kronecker delta*, que toma o valor de 1 quando $y_i = l$ e 0 nos restantes casos. Tendo por base a Figura 7 e o objeto com o ID = 0 que representa um *computer screen* que tem associada a observação x_0, neste caso a função $f(x_i)$ tomaria os seguintes valores $f(x_0)$ = [1.06,0,0.17,1.83], onde a ordem destas caraterística é a mesma definida na Tabela 4.

Por outro lado, o fator emparelhado pode ser definido da seguinte forma.

$$\phi(y_i, y_j, x_i, x_j, \theta\ ) = \sum_{l1 \epsilon L} \sum_{l2 \epsilon L} \delta(y_i = l_1, y_j = l_2) \cdot \theta_{l1l2} \cdot g(x_i, x_j)$$

Onde $g(x_i, x_j)$ é a função que computa o vetor de caraterísticas emparelhadas entre observações x_i e x_j, θ_{l1l2}é o vetor de pesos para o par das classes $l1$ e $l2$.

O produto dos fatores também pode ser expresso por meios de modelos *log-linear*.

$$\varphi(\cdot) = e^{-\epsilon(\cdot)}$$

sendo $\epsilon(\cdot)$ a chamada de função de energia. A representação logarítmica assegura que a distribuição probabilística é sempre positiva, sendo que os fatores podem tomar qualquer valor real [6].

A equação 2 pode, portanto, ser reescrita no espaço logarítmico da seguinte forma.

$$P(y|x;\theta) = \frac{1}{Z(x,\theta)} \cdot e^{-\epsilon(y,x,\theta)}$$

Onde a função de energia $\epsilon(\cdot)$ é representada juntamente com o fator unário e emparelhado.

$$\epsilon(y,x,\theta) = \sum_{i \,\epsilon\, Vertices} \phi(y_i, x_i, \theta) + \sum_{i \,\epsilon\, Arestas} \phi(y_i, y_j, x_i, x_j, \theta)$$

O treino e a indiferença é representado pelas equações 2 e 3.

2.3 Tecnologias

Nesta secção apresentamos uma introdução às tecnologias utilizadas no desenvolvimento do projeto.

2.3.1 Ontologia (Protégé/OWL)

Para a construção das ontologias utilizamos a ferramenta Protégé [8], que foi desenvolvida pelo *Stanford Center* for *Biomedical Informatics Research* na *Stanford University School of Medicine*.

Protégé fornece uma interface gráfica para a definição das ontologias gerando um ficheiro resultante. O formato do ficheiro escolhido é o ***RDF/XML*** com a terminação ***owl***.

2.3.2 Treino e Inferência com PGMs

2.3.2.1 UGM

“UGM é um conjunto de funções Matlab que implementam várias tarefas em modelos gráficos probabilísticos não direcionados de dados discretos com potenciais unários e emparelhados (*pairwise*)” [9].

Este tem implementação de métodos de treino e inferência com CRF e MRF.

Inicialmente, estávamos a desenvolver o nosso projeto usando estas funções, mas posteriormente deparámo-nos com problemas de que UGM não suporta a construção de grafos com diferente número de variáveis aleatórias, isto é, só conseguimos criar e treinar grafos com um número fixo de variáveis aleatórias. Sendo que no problema de reconhecimento de objetos o número de variáveis aleatória é diferente em cada cenário, teríamos de implementar esta funcionalidade no UGM, pelo que decidimos utilizar a biblioteca UPGMpp.

2.3.2.2 UPGMpp

Undirected Probabilistic Graphical Models in C++ (UPGMpp ou UPGM++) é uma biblioteca *open-source* para construir, treinar e gerir PGMs [3], [10], [11] .

Esta biblioteca foi desenvolvida pelo José Raúl Ruiz Sarmiento (autor da tese pela qual nos guiamos) na Universidade de Málaga e é mais adequada para o problema de reconhecimento de objetos visto que não possui as limitações que UGM tinha e é escrita em C++.

2.3.3 Aplicação para obtenção das *poses* (Kinect6DSLAM)

O Kinect6DSLAM [12] é uma aplicação, feita pelo Miguel Algaba Borrego, que permite comparar cada *frame* com o capturado anteriormente para o cálculo da posição e orientação da câmara em cada captura. Os *frames* são capturados através de uma câmara Kinect.

2.3.4 Análise e Processamento de Imagens (MRPT-PbMap)

MRPT

Empowering C++ development in robotics

A biblioteca Mobile Robot Programming Toolkit (MRPT) é uma biblioteca multiplataforma *open source* em C++, desenvolvida na Universidade de Málaga. Esta biblioteca ajuda no desenvolvimento robótico por investigadores em áreas relacionadas como a localização, mapeamento e visão por computador [13].

Desta biblioteca utilizamos o módulo PbMap [5], [14]. Este módulo, dado um conjunto de *frames* (imagens *point cloud)* e respetiva posição e orientação da câmara faz a reconstrução do cenário e identifica os planos presentes que correspondem a um objeto ou a uma parte deste. Após inferência, utilizamos o PbMap para a apresentar os planos no cenário reconstruído com a respetiva identificação. Apenas planos que respeitem os mínimos/máximos de certas características, definidas num ficheiro de configuração, são considerados. Por exemplo, apenas planos de médias/grandes dimensões, isto é, com grande quantidade de pontos é que são considerados, ignorando planos referentes a objetos de reduzidas dimensões, como por exemplo telemóveis.

Esta biblioteca foi utilizada devido à satisfação das nossas necessidades e pelo uso do autor na tese.

Consideramos outras possibilidades, mas o PbMap mostrou-se superior em termos do reconhecimento de planos, devido à reconstrução do cenário fornecer, no final, uma *point cloud* mais completa. Também verificamos que o cálculo de características seria mais fácil de fazer, estando alguns desses cálculos já implementados.

Capítulo 3

Neste capítulo pretendemos dar a conhecer e descrever detalhadamente a solução utilizada para o reconhecimento de objetos

3 Reconhecimento de Objetos

Como foi referido no Capítulo 1, o objetivo deste projeto passa por apresentar uma solução para o problema de reconhecimento dos objetos. A imagem seguinte representa a arquitetura da nossa solução de forma generalizada.

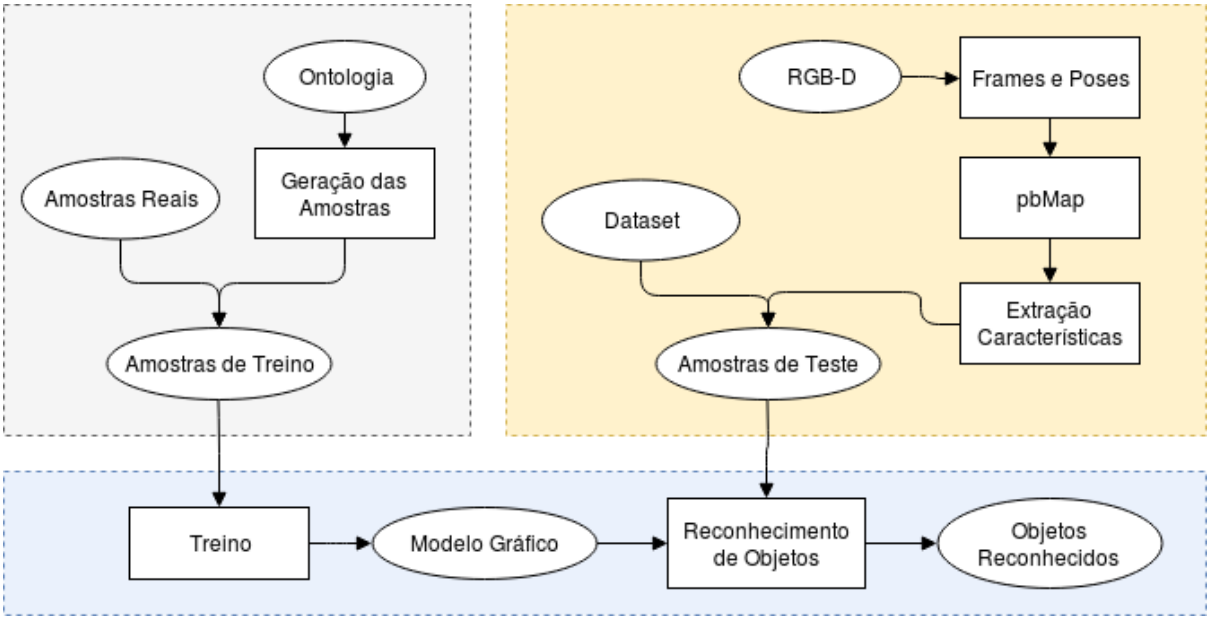


Figura 8 - Arquitetura generaliza para reconhecimento de objetos.

Na Figura 8 podemos identificar 3 blocos que podemos chamar de módulos:

- Modelo Gráfico Probabilístico, representado pelo bloco a azul.
- Amostras de treino, representado pelo bloco a cinzento (esquerda).
- Amostras de teste, representado pelo bloco a laranja (direita).

O módulo principal desta arquitetura é o módulo Modelo Gráfico Probabilístico representado a azul. Este módulo representa o pipeline de funcionamento da nossa solução. Isto é, representa a sequência de ações e processos necessários para ser possível o reconhecimento de objetos, portanto este é o único módulo que não pode ser totalmente substituído ou alterado pois é nele que assenta toda a arquitetura.

O módulo de amostras de treino é responsável por tratar e arranjar ou gerar as amostras de treino para o módulo Modelo Gráfico Probabilístico. Este módulo pode ser substituído caso exista uma solução melhor para arranjar ou gerar amostras num futuro. O único requisito é as amostras de treino seguirem a estrutura definida.

O módulo de amostras de teste é responsável por tratar e arranjar as amostras reais com a finalidade de serem analisadas pelo módulo Modelo Gráfico Probabilístico para poder reconhecer os objetos nelas contidos. Este módulo também pode ser substituído caso exista uma solução melhor para arranjar amostras reais num futuro.

Como é facilmente visível por esta breve introdução a arquitetura foi desenhada de forma modular. Isto porque, ao longo do tempo a tecnologia tende a evoluir e, por exemplo, daqui a uns anos pode ser possível extrair características de objetos em imagens RGB e então vai se poder criar um novo módulo de amostras de teste que implementa essa nova tecnologia e substituir ou até mesmo adicionar ao sistema esse novo módulo e o sistema será capaz de reconhecer objetos em imagem RGB.

3.1 Módulo Modelo Gráfico Probabilístico

Este módulo tem como objetivo encapsular o funcionamento da CRF, tanto o seu treino como a sua inferência, dando uma vista simplificada do processo de reconhecimento de objetos.

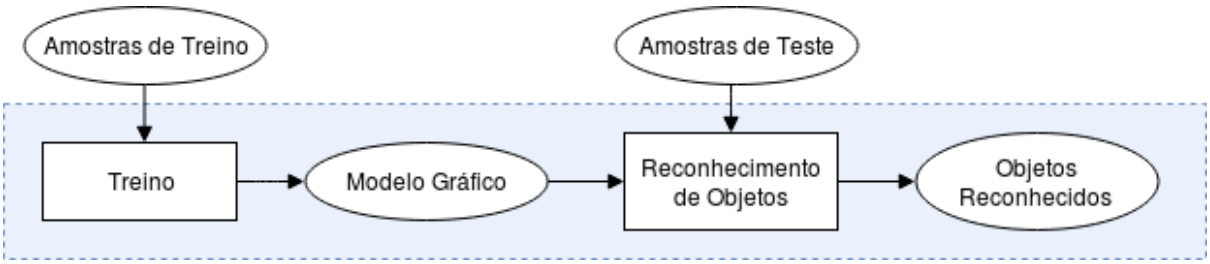


Figura 9 - Arquitetura do módulo Modelo Gráfico Probabilístico.

A Figura 9 demonstra a arquitetura deste módulo. Mais precisamente é possível observar uma cadeia de processos que culmina no reconhecimento de objetos.

Inicialmente temos a fase de treino, cujo o objetivo é treinar a CRF a partir de um conjunto de amostras de treino, o resultado deste processo é a CRF já treinada. Tendo a CRF treinada começa a segunda parte que corresponde à fase de inferência que permite a partir de uma amostra real calcular os objetos presente nela.

Uma amostra equivale a um cenário com vários objetos que têm as suas próprias características e podem estar em diversas posições.

3.1.1 Fase de treino do modelo

A fase de treino corresponde ao cálculo dos valores dos pesos θ da equação 3 do Capítulo 2 para a CRF a partir de um conjunto de amostras de treino, que serão posteriormente utilizados na inferência.

Nesta fase são recebidas como *input* um conjunto de amostras de treino que já foram pré-processadas pelo módulo amostras de treino.

A amostra de treino deve respeitar o seguinte formato:

- Identificação numérica dos objetos presentes no cenário que a amostra está a representar, isto corresponde ao *ground truth.*
- Cada objeto deve possuir informação sobre as suas características, isto é, a sua área, elongação, entre outros.
- Representação em grafo do cenário, onde os objetos são os vértices e as arestas representam a proximidade se estiverem, isto é, dados dois vértices que correspondem a dois objetos, se estes estiverem perto então existe uma aresta entre estes dois vértices.
- Indicação das arestas que estabelecem uma relação de sobreposição.

3.1.2 Fase de inferência sobre o modelo

Na fase de inferência, tendo por base a CRF treinada (isto é, o conjunto de pesos θ) e uma amostra real, são calculados os objetos presentes nessa amostra recorrendo à equação 4 do Capítulo 2.

Nesta fase são recebidos como *input* a CRF já treinada e a amostra de teste que foi obtida no módulo Amostras de análises.

A amostra de teste deve respeitar o seguinte formato:

- Cada objeto deve possuir informação sobre as suas características, isto é, a sua área, elongação etc…
- Representação em grafo do cenário, onde os objetos são os vértices e as arestas representam a proximidade se estiverem, isto é, dados dois vértices que correspondem a dois objetos, se estes estiverem perto então existe uma aresta entre estes dois vértices.
- Indicação das arestas que estabelecem uma relação de sobreposição.

Em contraste com a amostra de treino, esta não possui a identificação dos objetos presentes no cenário pois o objetivo é que a CRF treinada consiga reconhecer quais são os objetos presentes nesta amostra de teste.

3.2 Módulo Amostras de treino

Este módulo é responsável pela criação das amostras de treino segundo o formato especificado em 3.1.1.

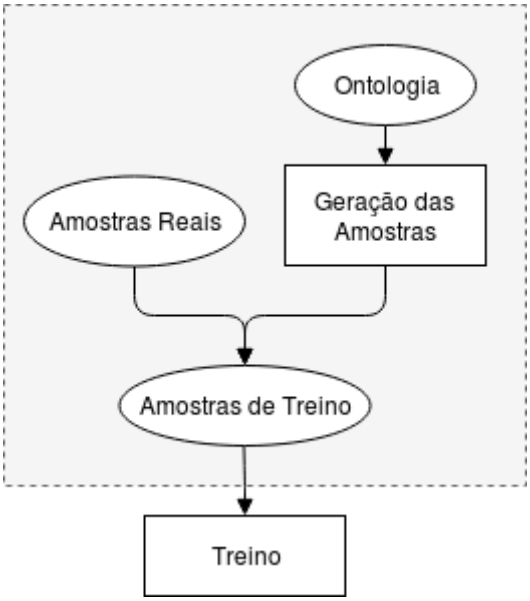


Figura 10 - Arquitetura do módulo Amostras de treino

Com base na Figura 10, que demonstra a arquitetura do módulo, é possível observar que este módulo utiliza duas opções para a criação de amostras de treino, sendo a primeira vertente baseada em amostras reais e a segunda vertente consiste na sintetização das amostras de treino por meio de uma ontologia.

3.2.1 Amostras reais

Amostras reais correspondem a cenários extraídos diretamente do mundo sem que tenha sido feito qualquer tipo de processamento. Estas correspondem, por exemplo, a imagens RGB-D.

Tendo agora a amostra real, esta é analisada e durante este processo é recolhida toda a informação referente aos objetos que fazem parte desta, bem como a relação de proximidade entre eles e as suas características unárias e emparelhadas que já foram referenciadas no Capítulo 2 Tabela 4.

O resultado do processo de análise da amostra real é então colocado no formato de amostra de treino descrito em 3.1.1.

É de realçar que como as amostras de treino serão utilizadas no treino da CRF, será necessário existir um número consideravelmente destas, na ordem das centenas, dependendo do número e variabilidade dos objetos. Para facilitar a recolha deste número consideravelmente grande de amostras podem ser utilizados *datasets*.

Podemos ver o *dataset* como um conjunto de amostras reais. Estas amostras podem estar ou não processadas. Por exemplo pode ser um *dataset* que apresenta um enorme conjunto de imagens RGB-D e neste caso será necessário aplicar o processo de análise a cada uma das imagens e só depois fazer a conversão para amostra de treino. Por outro lado, pode ser um *dataset* já processado onde a parte de análise já

vem feita e, portanto, só seria necessário colocar as amostras reais no formato de amostras de treino.

3.2.2 Amostras sintéticas

Geração de amostras sintéticas tem como objetivo evitar o trabalho demorado da recolha de amostras reais e a sua conversão para amostras de treino.

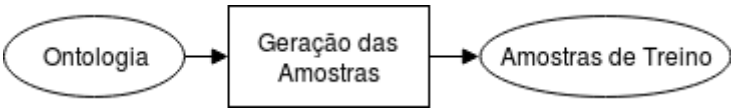


Figura 11 - Processo de geração de amostras sintéticas

A geração de amostras sintéticas pressupõe um conjunto de processos, tal como mostrado na Figura 11.

Primeiramente é criada uma ontologia que encapsula o conhecimento humano sobre um determinado cenário genérico onde se pretende fazer o reconhecimento de objetos.

Como referido no capítulo 2 na secção do conhecimento semântico, uma ontologia é uma forma de definir e representar um conjunto de objetos, as suas caraterísticas e as suas relações num dado cenário, esta informação tem como finalidade a geração de amostras de treino na forma de cenários sintéticos. A inclusão dos objetos no cenário sintético depende da frequência de ocorrência dos objetos no ambiente definido (*has_frequencyOfOcurrence*). As caraterísticas (*has_area**,** has_elongation**,*** entre outros) são materializadas de acordo com o seu valor médio e desvio padrão, que é calculado com base na variância definida na ontologia, seguindo a seguinte distribuição gaussiana (Figura 12).

Distribuição Gaussiana: *N(μ, σ)*, na qual **μ** é o valor médio da caraterística (e.g. área) e **σ** o seu desvio padrão

Depois as relações contextuais entre os objetos são estabelecidas de acordo com as suas relações de proximidade (*is_nearTo*) e as frequências associadas (por exemplo, se na ontologia um assento de uma cadeira, *chairSeat*, estiver perto da parte do lado de uma mesa, *tableSide*, com uma frequência elevada, é muito provável que sejam colocados perto um do outro nos cenários sintéticos).

As caraterísticas das relações estabelecidas são calculadas com base nas caraterísticas de cada objeto. Por exemplo, para calcular a diferença entre alturas de centroide entre dois objetos são utilizadas as alturas respetivas de cada objeto. Outra relação que é considerada é a *is_on,* que determina se dois objetos estão sobrepostos, isto é, se um está por cima ou por baixo do outro, como definido na ontologia.

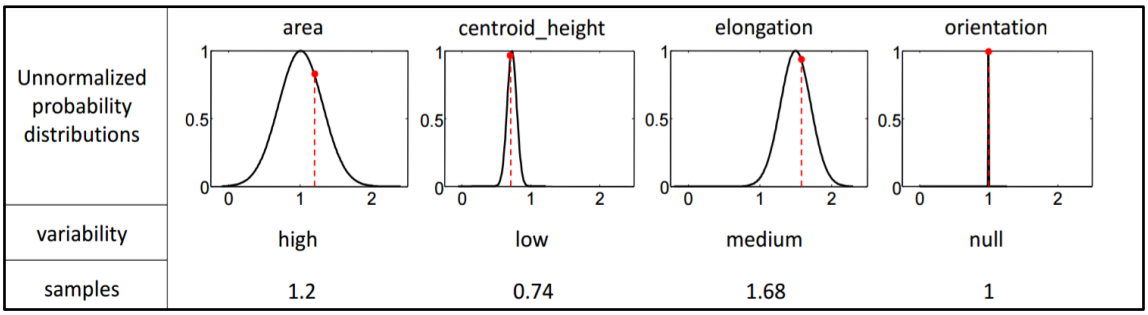


Figura 12 - Geração das caraterísticas dos objetos segundo uma distribuição normal definida na ontologia. [1]

Com isto, dá-se início ao processo de geração de amostras segundo um algoritmo que utiliza a informação contida na ontologia para criar os cenários sintéticos que já estão no formato de amostras de treino descritas em 3.1.1.

O pseudo-código do algoritmo de geração das amostras encontra-se no Apêndice 3.

3.2.3 Quando se deve usar amostras reais ou sintéticas

Parece claro que a geração de amostras sintéticas apresenta vantagens sobre a recolha de amostras reais. Mas nem sempre é verdade e é por esse motivo que nós decidimos incluir a possibilidade de treino com amostras reais. Este tópico serve para explicar quando é que se deve utilizar amostras reais ou sintéticas para treino.

Para esclarecer esta dúvida o mais fácil será demonstrar quando as amostras reais deveriam ser utilizadas. Como podemos ver no algoritmo apresentado em cima, as características dos objetos são geradas segundo uma distribuição gaussiana, isto pressupõe que as características dos objetos sejam sempre ligeiramente semelhantes e tendem para um valor médio.

Imaginemos o seguinte caso em que temos um conjunto de mesas num cenário e estas mesas são de duas categorias: mesas altas, com altura de 1 metro, e mesmas baixas, com altura de 0.2 metros. Se quiséssemos juntar estas mesas num objeto na ontologia, teríamos de dizer que a média da altura ao chão seria, por exemplo, 0.64 e que a variação é muito alta para obter um desvio padrão de por exemplo 0.48.

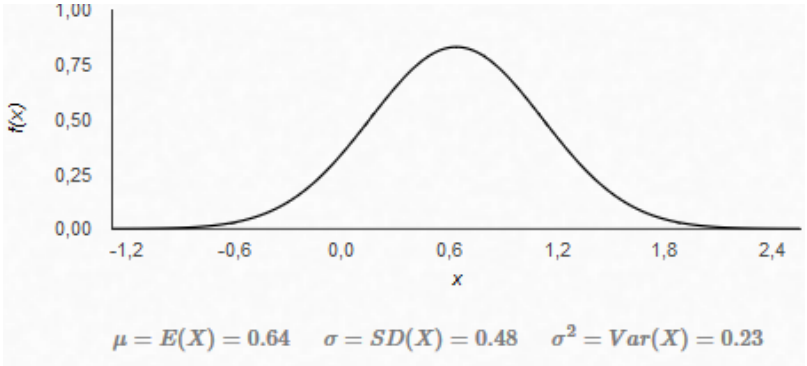


Figura 13 - Representação da distribuição normal para o exemplo das duas mesas.

Como podemos ver as mesas que seriam geradas provavelmente iriam ter as características da altura ligeiramente erradas, o que iria resultar em que o nosso modelo gráfico tivesse mais dificuldade em identificar as mesas reais, pois ele treinou com mesas que em média tinham sempre 0.6 metros de altura (Figura 13).

Este problema poderia ser facilmente corrigido caso fossem criados dois objetos mesa alta e mesa baixo e cada um destes já poderia ser representado por uma distribuição gaussiana válida. Esta prática seria uma má ideia porque se existissem mais casos semelhantes o número de objetos irá escalar rapidamente o que certamente irá afetar os resultados finais, e por outro lado a única diferença entre o objeto mesa alta e baixa, seria apenas a característica de altura sendo que as outras seriam muito semelhantes. Pelo que neste caso a melhor opção seria o treino com amostras reais que, apesar de ser um processo mais trabalhoso, os resultados finais seriam melhores.

Felizmente este é um caso raro visto que os objetos do dia a dia tendem a ser sempre semelhantes, isto é, as suas características podem ser corretamente codificadas em termos de distribuição gaussiana.

3.3 Módulo Amostras de teste

Este módulo tem como objetivo criar as amostras de teste no formato descrito em 3.1.2, que contém os objetos a serem reconhecidos pela CRF.

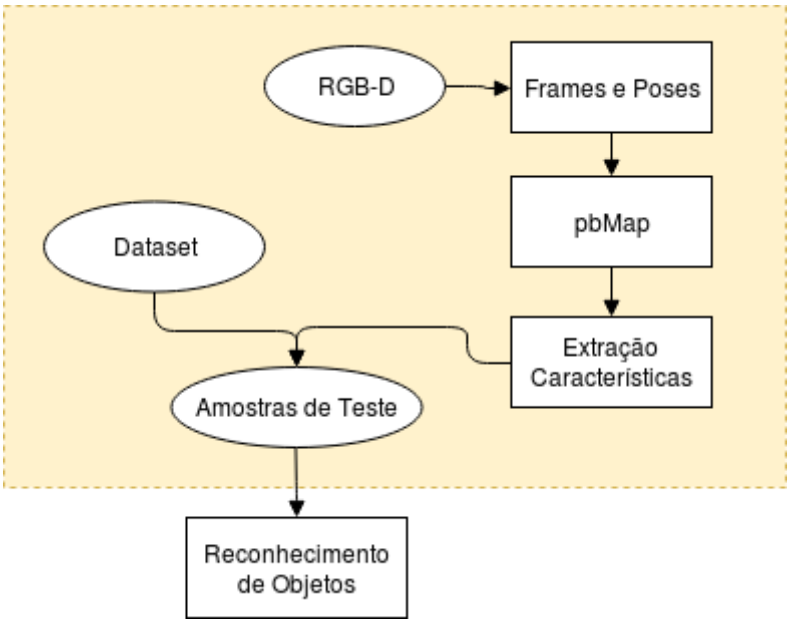


Figura 14 - Arquitetura do módulo Amostras de teste

A Figura 14 demonstra a arquitetura deste módulo e de maneira semelhante ao módulo Amostras de treino, este também tem duas perspetivas. A primeira tem como objetivo criação de amostras a partir de um *dataset* e a segunda perspetiva demonstra detalhadamente uma sequência de processos para, a partir de várias imagem RGB-

D, obter-se a amostra de teste que contém os objetos das imagens a serem identificados.

3.3.1 Amostras a partir de *Datasets*

O objetivo principal de utilizar amostras a partir do *dataset* é de reconhecer objetos e verificar de maneira automática os resultados da previsão, ou seja, quantos acertou/error.

Esta funcionalidade permite testar facilmente o desempenho das CRFs para uma grande quantidade de objetos e cenários facilitando assim a deteção de falhas no modelo de dados. Também oferece um meio para comparar diferentes CRFs para um mesmo problema específico de reconhecimento de objetos.

A criação das amostras de testes a partir do *dataset* segue um fluxo de processos igual ao demonstrado em 3.2.1, isto é, terá de ser feita uma análise a todas as amostras do *dataset* para convertê-las em amostras de teste no formato indicado em 3.1.1. Neste caso também é necessário incluir o *ground truth* para ser possível comparar os resultados previstos com os resultados reais.

3.3.2 Amostras a partir de imagens RGB-D

Nós decidimos incluir também uma perspetiva de criação de amostras a partir de imagens RGB-D, porque nós planeamos utilizar o sensor Kinect como ferramenta para adquirir amostras de reais para teste como vai ser demonstrado na Capítulo 4. No entanto a arquitetura desenhada é suficientemente genérica para a partir de imagens RGB-D gerar amostras de teste.

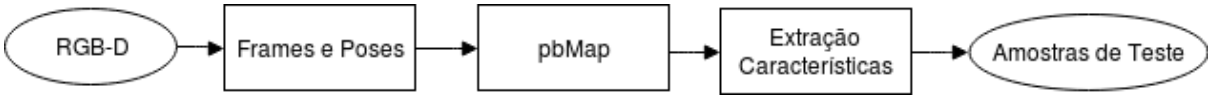


Figura 15 - Sequência de processos para a criação de amostra de teste a partir de imagens RGB-D.

Na Figura 15 observamos 4 passos fundamentais para a criação de amostras de teste a partir de imagens RGB-D.

3.3.2.1 RGB-D

O primeiro passo corresponde à captura da imagem RGB-D, esta pode ser feita de várias formas sendo mais comum através da Kinect. Uma vez a captura feita, vamos chamar às imagens resultantes de observações.

Dependendo dos algoritmos utilizados mais a frente pode ou não ser necessário a captura de várias imagens em sequência.

Na figura 16, está representado um exemplo de uma imagem RGB-D, onde do lado esquerdo temos a imagem RGB normal, e do lado direito a imagem de profundidade.

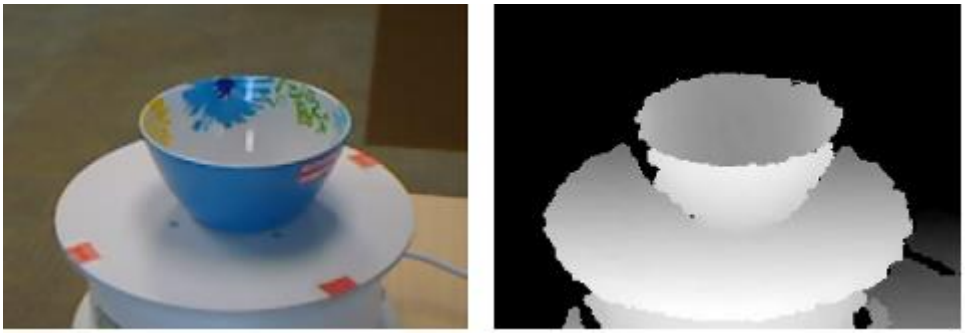


Figura 16 - Exemplo de uma imagem RGB-D [15]

3.3.2.2 *Frames* e *Poses*

Este segundo passo só é aplicado caso sejam necessárias várias imagens RGB-D em sequência para representar o cenário.

Tendo um conjunto de observações capturadas é agora necessário, através da odometria visual calcular a posição e orientação do sensor a quando da sua captura. Em contraste com o cálculo a odometria visual, esta posição também pode ser obtida através do auxílio de outro sensor específico para a tarefa.

Cada observação deve ser emparelhada com a sua posição e orientação do sensor respetiva, que a partir de agora vamos chamar de *pose*.

3.3.2.3 Plane-Based Mapping

Nós para o reconhecimento de objetos em três dimensões, providos das capturas RGD-B, optamos por tentar reconhecer os planos associados aos objetos, aqui existem outras alternativas poderiam ser seguidas.

Nesta fase deve ser utilizado um algoritmo ou software capaz de fazer o *plane-based mapping* de uma imagem RGB-D ou de várias imagem RGB-D com a respetiva *pose*.

Plane-Based Mapping corresponde à identificação de superfícies planas no cenário correspondente às observações RGB-D.

3.3.2.4 Extração das caraterísticas

Esta fase pressupõe que os planos nas imagens RGB-D já foram encontrados e agora com base nesses planos iremos extrair as características unárias descritas no capítulo 2, Tabela 4. Seguidamente é calculada a relação de proximidade entre os planos e em termos de características emparelhadas, também definidas na mesma tabela, só é necessário extrair a de sobreposição visto que as outras são facilmente calculadas.

Finalmente tendo os planos, as suas respetivas caraterísticas e relações podemos criar a amostra de teste segundo o formato descrito em 3.1.2, onde um plano corresponde a um objeto a ser identificado.

Capítulo 4

Neste capítulo iremos demonstrar as soluções que foram implementadas e como podem ser usadas.

4 Implementação

Visto que tanto a biblioteca UPGMpp [10] como MRPT [13] estavam escritos na linguagem C++, nós também decidimos trabalhar em C++ em todo o nosso projeto.

O módulo Modelo Gráfico Probabilístico do capítulo 3 é a base de todo o nosso sistema. As operações de criação, treino e inferência de CRFs já estão implementadas na biblioteca UPGMpp. Contudo, esta biblioteca é bastante complexa e tem muitas funcionalidades que nós não utilizamos. Por outro lado, faltam alguns componentes como, por exemplo, a geração de amostras sintéticas.

Para facilitar o uso desta biblioteca e para complementá-la nós criámos uma biblioteca que chamamos de PI_ObjectRecognition. Esta trabalha em cima da UPGMpp e implementa várias novas funcionalidades que no geral facilitam o processo de reconhecimento de objetos.

Para demonstrar como se usa a biblioteca PI_ObjectRecognition, foram desenvolvidas várias aplicações:

- PI_ObjectRecognition_Demo, que permite treinar o CRF com amostras sintéticas ou reais;
- Módulo Kinect demonstra visualmente em três dimensões o processo de captura do cenário, processamento do mesmo e reconhecimento dos objetos lá encontrados. De notar que este módulo é um conjunto de três outras aplicações PI_KinectGrabber, DemoSlam e PI_PbMap_Demo.

Tanto a biblioteca como as aplicações encontram-se no nosso repositório da Universidade de Aveiro [16].

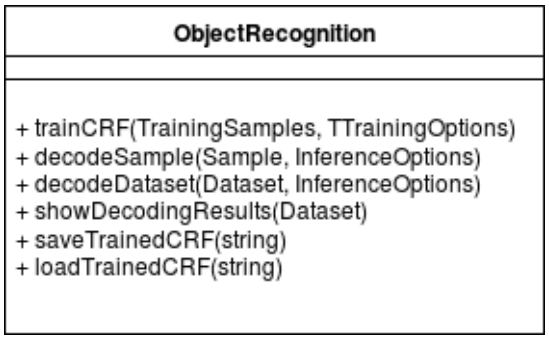
4.1 Biblioteca PI_ObjectRecognition

Esta biblioteca corresponde à implementação do módulo **Modelo Gráfico Probabilístico** e do módulo **Amostras de treino** descritos no Capítulo 3 e pode ser separada em dois componentes principais: treino e inferência.

A classe principal desta biblioteca é a ObjectRecognition e é comum às duas partes.

4.1.1 ObjectRecognition

A classe ObjectRecognition é o componente principal da nossa biblioteca. Esta permite treinar a CRF, guardar a CRF treinada num ficheiro, ler a CRF treinada de um ficheiro e fazer a descodificação de uma amostra ou *dataset*.



4.1.2 Treino

Para treinar a CRF é preciso obter primeiro as amostras de treino. Estas amostras de treino são guardadas na classe TrainingSamples e podem ser obtidas a partir de um *dataset* (terá de estar definido na classe Dataset) ou geradas a partir de uma ontologia.

No caso de usar a classe Dataset, é preciso preenche-la com as amostras, que devem ser definidas na classe Sample. Um exemplo da estrutura da amostra encontra-se no Apêndice 4.

No caso de se usar a ontologia, esta pode ser definida no Protégé (que gera um ficheiro owl RDF/XML) e posteriormente usado o *parser* que criámos, OwlXmlParser, ou então diretamente no código C++ (Apêndice 1).

Depois de ter a ontologia, é preciso estipular um conjunto de fatores para prosseguir com a geração sintética das amostras. Estes fatores encontram-se na estrutura SyntheticOptions e pode ser definido:

- ***numberSamples***: número total de amostras a gerar;
- ***standardDeviation***: função que mapeia a variância (Variance) para um valor concreto de desvio padrão. Assim, um valor que tenho a média 2 e a variância U_MEDIUM, será mapeado numa normal com média 2 e desvio padrão 1 (2*0.5). Os valores por defeito são:
  U_VERY_HIGH=1, U_HIGH=0.75, U_MEDIUM=0.5, U_LOW=0.25, U_VERY_LOW=0.1 e U_NULL=0;
- ***frequencyOfOccurrence****:* função que mapeia a frequência com que o objeto aparece (FrequencyOfOccurrence). Estes valores estão na escala de 0 a 100.
  Os valores por defeito são:
  O_ALWAYS=100, O_VERY_HIGH=90, O_HIGH=70, O_MEDIUM=50, O_LOW=30 e O_VERY_LOW=10;
- ***frequencyOfRelations***: função que mapeia a frequência com que a relação aparece (PairwiseFrequency). Estes valores estão na escala de 0 a 100.
  Os valores por defeito são:
  P_ALWAYS=100, P_VERY_HIGH=90, P_HIGH=80, P_MEDIUM_HIGH=65, P_MEDIUM=50, P_MEDIUM_LOW=40, P_LOW=20 e P_VERY_LOW=10;

- ***unaryFeaturesNameOrder***: Ordem das características geradas. Os nomes devem coincidir com os nomes estipulados na ontologia. O utilizador tem de fornecer sempre esta lista.
- ***functionsPairwise***: Lista de funções que se destinam a ser implementadas pelo utilizador para o cálculo das caraterísticas emparelhadas. Caso não sejam definidas, estas não serão usadas na geração. Um exemplo de uso encontra-se no Apêndice 2.

Tendo a ontologia e as opções sintéticas, as amostras de treino são geradas com a função generateSyntheticSamplesFromOntology(OntologyConcept *root, SyntheticOptions syntheticOptions) do TrainingSamples.

Por fim, antes de começar a treinar, é necessário definir as opções de treino (usámos diretamente a estrutura TTrainingOptions da biblioteca UPGMpp). Aqui é possível definir quantas *threads* serão utilizadas durante o treino, número máximo de iterações, entre outros.

Na classe TrainingSamples podem ser definidas outras opções de treino na estrutura TrainingSamplesOptions, como:

- ***nodeBias / edgeBias***: o *bias* é uma caraterística que está sempre a 1, e é incluído para refletir os efeitos no estado que são independentes das caraterísticas. Nestas opções indica-se se é ou não para usar este *bias* nos nós ou na ligações, respetivamente.
- ***nodeMultipliers / edgeMultipliers***: vetor de multiplicadores de cada caraterística. É usado para dar maior importância a certas caraterísticas dos nós ou das ligações, respetivamente.

Para treinar é usada a função trainCRF(TrainingSamples& samples, TTrainingOptions upgmpp_training_options) da classe ObjectRecognition. Esta função é apenas uma abstração da função de treino da UPGMpp, sendo que apenas se transfere o conteúdo para estruturas apropriadas da biblioteca UPGMpp e treina-se com essa.

Depois de acabar o treino, a CRF treinada pode ser guardada num ficheiro.

4.1.3 Inferência

Depois de ter a CRF treinada é possível usá-la para reconhecer objetos. Como já mostrámos antes, este reconhecimento pode ser feito com um *dataset* ou com uma amostra concreta (no nosso caso retirada com ajuda da Kinect).

A estrutura de uma amostra, classe Sample na nossa biblioteca, pode ser encontrada no Apêndice 4. A inferência pode ser feita logo sobre essa amostra ou, no caso de ser um conjunto de amostras, é usada a classe Dataset que encapsula um conjunto de Samples.

Para proceder à inferência é preciso ainda definir as opções de inferência. Estas encontram-se na estrutura InferenceOptions. Tal como nas opções de geração sintética, é necessário indicar functionsPairwise, funções de cálculo das caraterísticas emparelhadas. Tal como as TrainingSamplesOptions definidas no ponto anterior, InferenceOptions têm os parâmetros de nodeBias/edgeBias e nodeMultipliers/edgeMultipliers. É possível definir ainda, com inferenceAlgorithm, o algoritmo de inferência a ser usado e passar as opções de inferência da biblioteca UPGMpp no campo upgmppInferenceOptions.

Os resultados da inferência são guardados em inferenceResults de cada objeto da classe *Sample*.

4.2 Aplicação PI_ObjectRecognition_Demo

Para demonstrar algumas das funcionalidades da biblioteca PI_ObjectRecognition que foi desenvolvida a aplicação PI_ObjectRecognition_Demo. Nesta é possível gerar as amostras de treino a partir da ontologia definida em Protégé e treinar a CRF com essas amostras. Também é possível, partindo da CRF já treinada, fazer inferência com um *dataset* já processado para testar o desempenho do nosso modelo.

4.3 Demo PI_Kinect

A demonstração aqui apresentada tem como objetivo a partir de um sensor Kinect capturar diversas observações, isto é, imagens RGB-D de um determinado cenário, e seguidamente visualizar em três dimensões o cenário com os objetos nele reconhecidos.

Esta demonstração corresponde à implementação do módulo Amostras de teste, mais especificamente a segunda perspetiva descrita no ponto 3.3.2 do capítulo 3.

A sequência de passos descritas no ponto 3.3.2 correspondem a várias aplicações individuais.

1º passo: Captura de várias imagens RGB-D corresponde à aplicação PI_KinectGrabber.

2º passo: Cálculo da posição e orientação da câmara corresponde à aplicação PI_DemoSlam

3º e 4º passo, execução do algoritmo de *plane-based mapping* e extração de características corresponde à aplicação PI_Pb

Map_Demo primeira parte.

Para além destes passos ainda criámos uma visualização gráfica em três dimensões do cenário resultante já com os resultados da previsão por parte da CRF, que corresponde à segunda parte do PI_PbMap_Demo.

4.3.1 Aplicação PI_KinectGrabber

A aplicação PI_KinectGrabber permite ao utilizador recolher diversas observações de um cenário real de diferentes perspetivas através da utilização de um sensor Kinect.

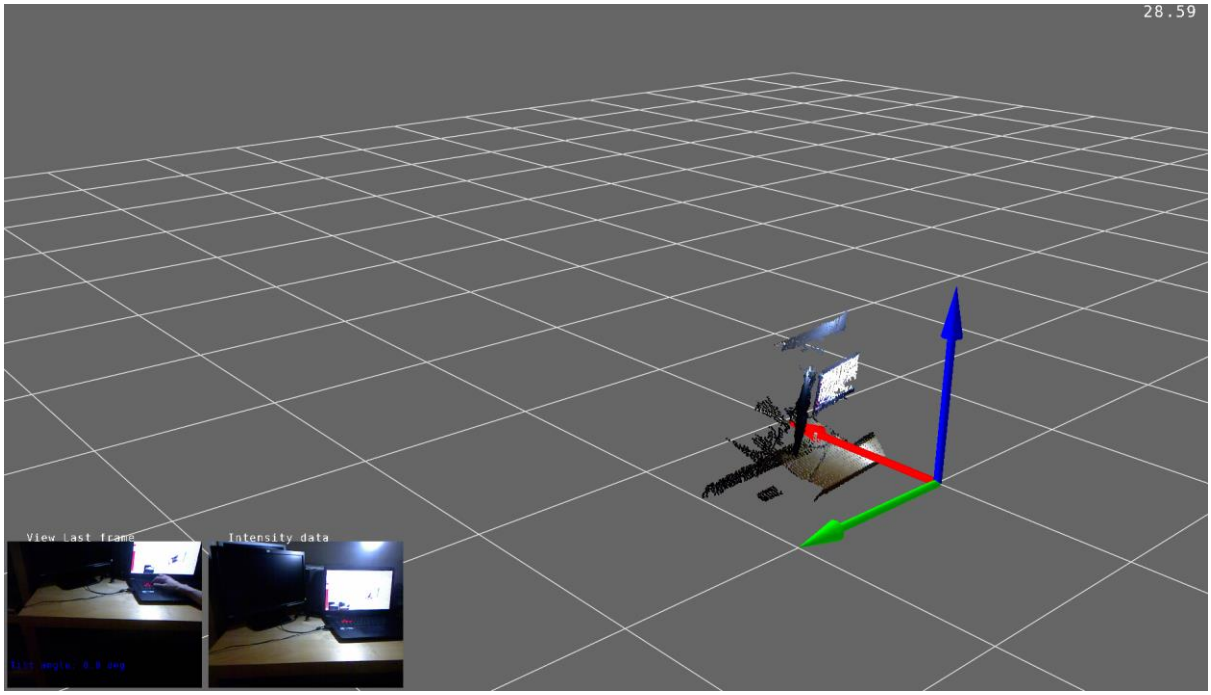


Figura 17 - Aplicação PI_KinectGrabber

A Figura 17 representa o ambiente de trabalho da aplicação e podemos observar que no centro encontra-se a *point cloud* que está a ser obtida em tempo real e no canto inferior esquerdo estão imagens RGB. Para auxiliar a captura de várias observações em sequência foi adicionada a imagem do lado esquerdo que permite visualizar a última imagem RGB tirada pela aplicação.

Em termos de comandos o utilizador pode:

- Carregar na tecla “E”, que corresponde a tirar uma observação (*frame*).
- Carregar na tecla “W”, que corresponde a aumentar o ângulo de inclinação do sensor Kinect.
- Carregar na tecla “S”, que corresponde em diminuir o ângulo de inclinação do sensor Kinect.
- Carregar na tecla “Esc”, que permite terminar a captura.

Quando a captura termina, a aplicação antes de terminar guarda todas as observações tiradas num ficheiro “rawlog” comprimido. Este ficheiro é um tipo de ficheiro da biblioteca MRPT que permite guardar observações capturadas por sensores.

4.3.2 Aplicação DemoSlam

Como já referido esta aplicação é responsável pelo cálculo da posição e orientação do sensor Kinect. Nós não utilizamos diretamente a aplicação *Kinect6DSlam*, em vez disso nós convertemos essa aplicação numa biblioteca mais leve "KinectSlamAdpter", isto é, que só utiliza uma pequena parte das funcionalidades do *Kinect6DSlam* o que diminuiu drasticamente as bibliotecas necessárias para a instalação.

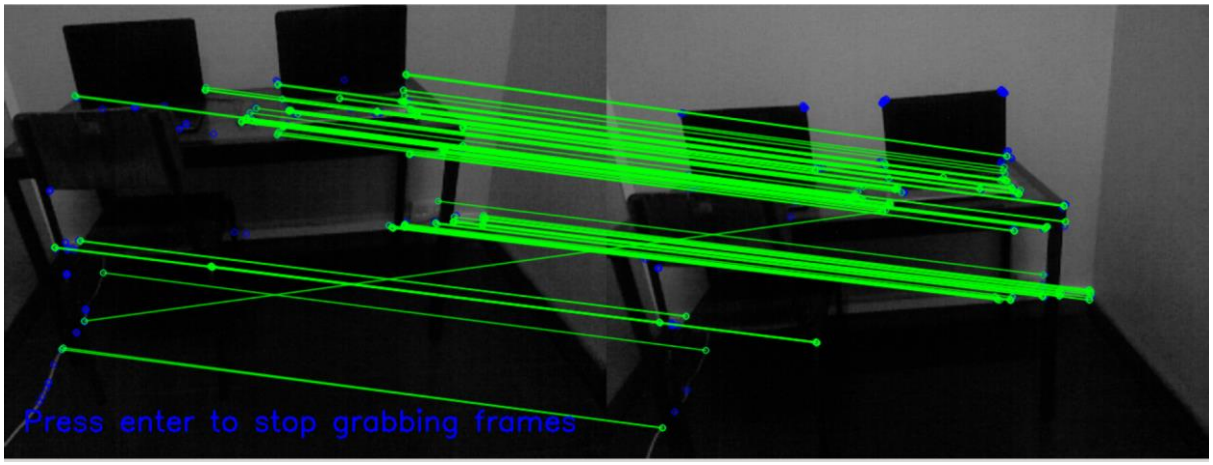


Figura 18 - Execução da aplicação PI_DemoSlam

Em termos de execução a aplicação PI_DemoSlam é a mesma que o Kinect6DSlam, só que é utiliza a biblioteca KinectSlamAdpter. Esta aplicação permite-nos, a partir do *rawlog* obtido como resultado da aplicação PI_KinectGrabber, obter a *point cloud* (".pcd") e pose (".pose") correspondentes a cada observação.

A Figura 18 demonstra visualmente o algoritmo implementado na aplicação *Kinect6DSlam*, que foi adaptado para a biblioteca "KinectSlamAdpter".

4.3.3 Aplicação PI_PbMapDemo

Como foi referido, a aplicação PI_PbMapDemo encontra-se dividida em duas partes. A 1º parte, de nome "reconstrução", é responsável pela reconstrução do cenário numa única representação visual em três dimensões e da posterior aplicação do algoritmo de *plane-based mapping* para deteção dos planos e extração das características. A 2º parte, de nome "visualização", tem como objetivo, a partir dos planos e características calculadas, inferir através de uma CRF treinada os objetos que estão representados no cenário e demonstrá-los visualmente.

4.3.3.1 Reconstrução

A aplicação PI_PbMapDemo recebe os seguintes argumentos como *inputs*:

- Localização das *point cloud* (*frames*) e as respetivas *poses*, geradas pela aplicação anterior..
- Número de observações a serem processadas, este parâmetro torna-se interessante caso se pretenda processar menos farmes do que os que existem na localização dada.

Através dos pares de *point cloud* e *pose* e do algoritmo de *plane-based Mapping* implementado pela biblioteca MRPT-PbMap é efetuada a reconstrução do cenário. O

algoritmo junta as diferentes *point clouds* numa única através da aplicação da translação e rotação indicada no ficheiro *pose*, à medida que é feita esta junção os planos vão sendo detetados segundo as limitações definidas no ficheiro de configuração ("configPbMap.ini"). Para finalizar a deteção, ainda tratamos de desconsiderar os planos repetidos, tratando-se estes de planos iguais, mas com orientação contrária, que são muitas vezes detetados incorretamente pelo MRPT-PbMap, este processo está indicado nas Figuras 19 e 20.

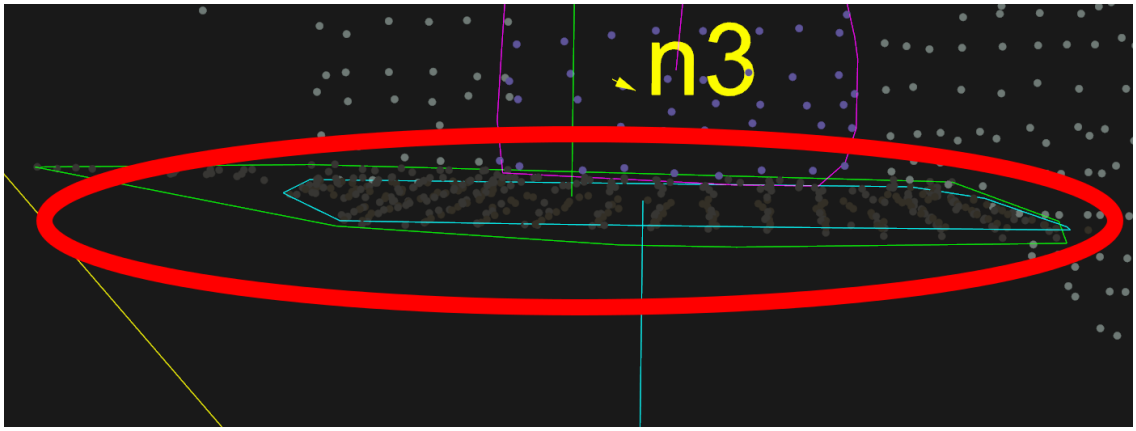


Figura 19 - Demostração de dois planos iguais na reconstrução com PbMap

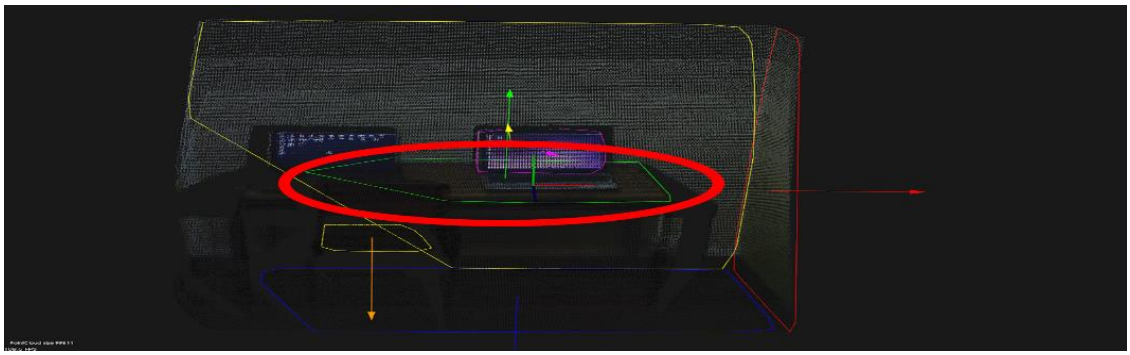


Figura 20 - Correção dos planos sobrepostos no PbMap

Nota as observações capturadas com o PI_KinectGrabber estão num referencial com orientação diferente ao utilizado pela biblioteca MRPT-PbMap, pelo que é necessário aplicar uma rotação de (180º) no eixo dos X à *pose* original da *point cloud* antes de ser aplicada a translação e rotação a esta como é óbvio.

$$Matrix_{novaPose} = Matrix_{rotação\ X\ de\ \Pi} \cdot Matrix_{PoseOriginal}$$

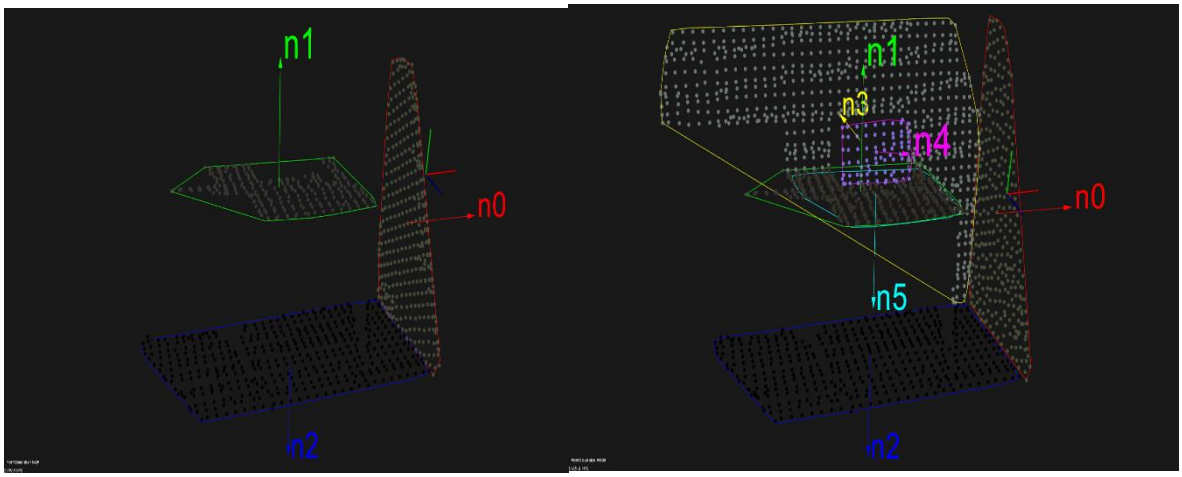


Figura 21 - Exemplo da reconstrução e deteção dos planos pelo PbMap.

No final do processo de reconstrução e deteção de planos (Figura 21) dá-se início ao processo de extração de características dos planos.

As características calculadas são mais uma vez, orientação, altura da centroide do chão, a área e elongação. A área e a elongação são calculadas pela biblioteca MRPT_PbMap, a orientação indica se o objeto tem orientação vertical ou não e é calculada a partir do vetor normal do plano, com o qual verificamos se a componente y é maior que 0.5 para o caso de ser vertical, caso contrário é horizontal. O cálculo da altura da centroide do chão é necessário encontrar o plano mais baixo que irá corresponder ao chão, ficando com altura de 0 m e a altura dos restantes planos é calculado a partir da distância vertical do seu centro até intercetar o plano que foi considerado como sendo o chão, este cálculo é representado pela seguinte equação:

$$altura_{centroid} = \frac{P_x \cdot a + P_y \cdot b + P_z \cdot c + d}{\sqrt{(a)^2 + (b)^2 + (c)^2}}$$

Sendo a equação do plano definida $ax + by + cz + d = 0$ e o Ponto P (x, y, z) que corresponde ao centro do plano.

Orientation	Centroid_z	Area	Elongation
1	0,72526	0,34783	2,32114
0	0,00000	1,46145	2,86637
1	0,51358	0,39803	1,85308
0	0,89651	0,96934	1,70350
0	0,74833	0,58519	1,47532
1	0,96364	0,12571	1,00000
1	0,93444	0,21259	1,52702
1	1,05757	0,23680	3,22358
0	0,39841	0,13140	1,00000
0	0,03006	0,12952	2,03183
1	0,46934	0,50543	1,45506

Tabela 5 - Exemplo de matriz com representação das características dos planos

Como é facilmente visível, a altura ao chão, não é uma das melhores características a ser utilizada, pois caso o plano do chão não seja detetado, irá existir um plano que será considerado como sendo o chão e as alturas de todos os objetos ficaram em relação a esse objeto que não é efetivamente o chão, levando a valores incorretos na componente "centroid_z". Este é um problema que nós garantimos que nunca acontece aquando da recolha dos dados, uma solução seria a não utilização desta característica e optar por outras.

As relações que calculamos são a relação de proximidade e de sobreposição. É assumida como uma relação de proximidade se dois planos estiverem a uma certa distância (*default* 0.20 m). A distância entre os planos é calculada pelo MRPT-PbMap através do método isPlaneNearby().

A relação que define se um objeto está em cima ou por baixo de outro é definida se a distância entre os dois planos é inferior a 0.10 m e se ambos os planos não têm orientação vertical.

Edges	1	1	1	1	1	1	1	2	2	2	4	4	4	4	4	4	5	5	5	5	5	5	6	7	7	7	8	9
	2	3	4	5	6	9	10	3	9	10	5	6	7	8	9	11	6	7	8	9	10	11	7	8	9	11	11	10
isOn	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Tabela 6 - Exemplo de matrizes com representação das relações entre planos e sobreposição

Com este conjunto de características (Tabelas 5 e 6) é então criada a amostra de teste.

4.3.3.2 Visualização

A 2ª parte da aplicação PI_PbMapDemo utiliza como *input* que vem da 1º parte:

- *Point cloud* reconstruída.
- Planos reconhecidos no PbMap
- Amostra de teste com as características calculadas.

A aplicação recorre à biblioteca PI_Object_Recognition para aplicar a inferência sobre a amostra de teste criada na 1ª parte. Seguidamente é executada a visualização da *point cloud* reconstruída e são aplicados os contornos respetivamente aos planos detetados durante a 1ª parte (Figura 22).

Finalmente são colocadas as *labels* no centro dos planos representados, isto é, o nome dos objetos detetados como texto em três dimensões (Figura 23).

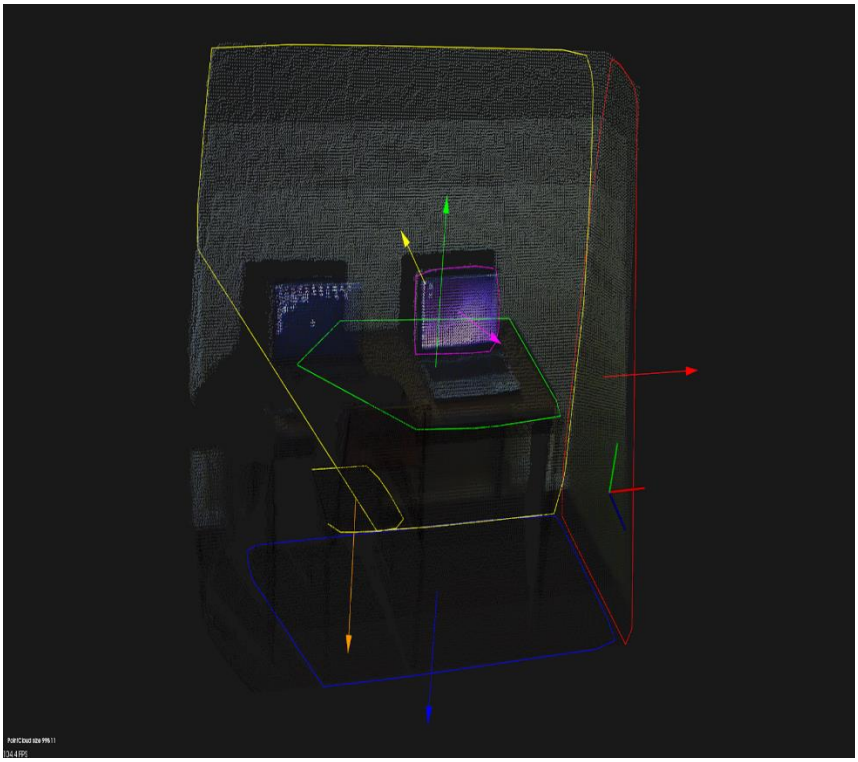


Figura 22 - Visualização da point cloud reconstruída e dos planos detectados.

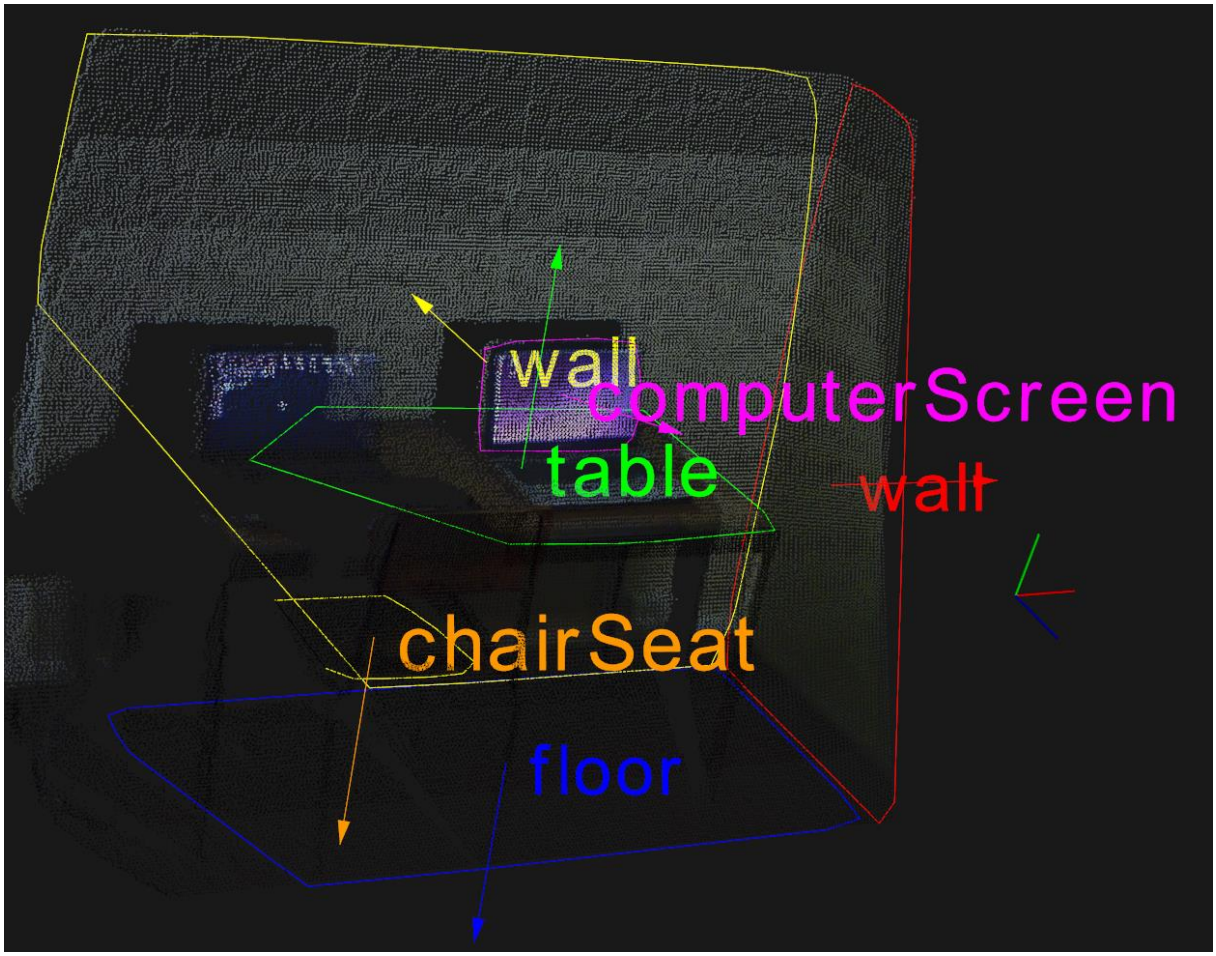


Figura 23 - Representação final da point cloud com a inferência feita.

4.4 Instalação

A instalação e configuração das bibliotecas necessárias pode ser um tanto quanto desafiante visto são bibliotecas em c++ e estão constantemente a sofrer atualizações. Para facilitar o processo de instalação foram criados uns *shell scripts* de auxílio à instalação e podem ser encontrados no repositório do projeto [16]. No entanto vamos listar as dependências necessárias.

Todo o trabalho foi desenvolvido sobre o sistema operativo Ubuntu 14.04, sendo que noutros sistemas operativos o processo de instalação ou as dependências podem ser diferentes.

O código foi desenvolvido utilizando o IDE netbeans 8.2, pelo que pode ser utilizado para visualizar ou até mesmo compilar/executar o código fonte.

Dependências comuns às aplicações criadas:

- Eigen3, biblioteca C++ para aplicação de álgebra linear e operações com matrizes e vetores.
- Boost, coleção de bibliotecas C++ que estendem a funcionalidade da linguagem C++.

4.4.1 Biblioteca PI_ObjectRecognition

Dependências necessárias para esta biblioteca:

- UPGMpp

Para instalar basta executar “sudo make install”. Esto comando compila os ficheiros C++ e encarrega-se da *linkagens* das biblioteca de que depende e instala os ficheiros *headers* em /usr/local/include/PI_ObjectRecognition e a biblioteca em /usr/local/lib.

4.4.2 Aplicação PI_ObjectRecognition_Demo

Dependências necessárias:

- PI_ObjectRecognition

Para executar pode ser utilizado o IDE netbeans e aconselha-se para a modificação das MACROS ou então basta compilar com “make” o executável vai para a pasta dist/(depende do compilador).

O programa tem duas opções de execução com -t onde treina a CRF segundo uma ontologia ou -d utiliza última CRF treinada para testar a sua performance contra um *dataset* à escolha.

4.4.3 Demonstração PI_Kinect

Sendo que a quantidade de dependências a instalar é muito grande só serão referenciadas as mais importantes e recomenda-se a execução do script para instalação.

- Visualization Toolkit (Vtk) versão 5.8, biblioteca C++ para processamento de imagem e visualização
- Point CLoud Library (PCL) versão 1.7, biblioteca C++ para processamento de point clouds usados em visão computacional
- Opencv
- MRPT
- PI_ObjectRecognition

Capítulo 5

Neste capítulo serão apresentados os resultados obtidos.

5 Resultados e discussão

Serão agora demonstrados e discutidos os resultados recolhidos referente às aplicações descritas no ponto 4.2 e 4.3 do capítulo 4.

Primeiramente, vamos abordar os resultados referentes à aplicação PI_ObjectRecognition_Demo e seguidamente vamos apresentar os resultados referentes à demonstração PI_Kinect.

5.1 Resultados da aplicação PI_ObjectRecognition_Demo

Com esta aplicação foram realizados dois testes. O primeiro tinha como objetivo testar o desempenho de uma CRF treinada com base em amostras sintéticas geradas a partir de uma ontologia contra o *dataset* da UMA-Offices [17]. O segundo tinha por objetivo determinar qual o valor mínimo de amostras de teste que seriam necessárias gerar para conseguirmos obter resultados satisfatórios, isto é, semelhantes ao do primeiro teste.

5.1.1 Teste contra o dataset da UMA-Offices

O *dataset* UMA-Offices é composto por um total de 25 cenários capturados através de um sensor RGB-D, e já está processado pelo que a conversão para amostras de teste foi direta e não precisou de existir o passo de análise. Ao longo do *dataset* podem ser encontrados 11 objetos diferentes, que estão listados na Tabela 7.

1	Chão (floor)
2	Parede (wall)
3	Tampo mesa (table)
4	Lado mesa (tableSide)
5	Costas da cadeira (chairBack)
6	Assento da cadeira (chairSeat)

7	Monitor (computerScreen)
8	Portátil (laptop)
9	Parte frente torre (PCfront)
10	Parte lateral torre (PCside)
11	Parte lateral do armário(cupboardSide)

Tabela 7 - Identificação dos nomes dos objetos presente no dataset e entre parênteses está correspondente label que nós escolhemos para os identificar nas demos.

O *dasaset* também possuía as seguintes características referentes aos objetos: orientação, altura ao chão, área e elongação; estas coincidem com as características unárias consideradas no ponto 2.2.2 do capítulo 2.

5.1.1.1 Definição das ontologias

O primeiro passo para a realização deste teste consistiu na criação de uma ontologia que encapsulam os objetos referidos no *dataset*.

Neste ponto foram utilizadas duas abordagens para a criação da ontologia, uma abordagem mais simplistas e uma mais minuciosa.

Na abordagem simplista, mais especificamente aquando da definição na ontologia das características dos objetos, tentamos imaginar as dimensões dos objetos que estariam no *dataset* tendo por base o seu nome, por exemplo, o chão nós consideramos como sendo uma superfícies bastante grande e, portanto, tem uma área considerável, e o mesmo se aplica às relações de proximidade de objetos e até mesmo a frequência com que estes aparecem nos cenários. Resumidamente, não foram consideradas quaisquer tipos de medidas físicas e a ontologia é baseada 100% no conhecimento humano da pessoa que a construiu.

Por outro lado, na abordagem mais minuciosa já foram tidos em consideração as características dos objetos referenciadas no *dataset*. Esta abordagem como seria de esperar é mais demorada. Consiste em ir ao *dataset* recolher um número arbitrário de característica de um determinado objeto e calcular o valor médio dessa característica entre os objetos selecionados e o respetivo desvio padrão. Este valor médio e desvio padrão era então inserido na ontologia.

5.1.1.2 Resultados

A CRF foi treinada com a seguinte configuração:

- número de amostras sintéticas = 1000
- *nodeLambda* = 1
- *edgeLambda* = 8

Para a análise do desempenho da CRF recorremos às métricas de macro *precision* e *recall*.

A precisão de uma dada classe de objeto c_i é definida pela percentagem de objetos reconhecidos que pertencente a essa classe. Considerando $recognized(c_i)$ como sendo um conjunto de objetos reconhecidos que pertencem à classe c_i e considerando $groundtruth(c_i)$ como o conjunto correto de objetos de uma classe c_i presente no *dataset*. Podemos definir precisão da seguinte maneira [4].

$$precision(c_i) = \frac{|recognized(c_i) \cap groundtruth(c_i)|}{|recognized(c_i)|}$$

O *recall* de uma dada classe do objeto c_i expressa a percentagem dos objetos que que pertencem à classe c_i e são reconhecidos como membros desta.

$$recall(c_i) = \frac{|recognized(c_i) \cap groundtruth(c_i)|}{|groundtruth(c_i)|}$$

As métricas apresentadas em cima estão associadas a uma única classe de objetos, para sabermos a performance da CRF para todas as classes utiliza-se a macro precisão e *recall,* que corresponde a fazer a média.

$$macro\_precision(c_i) = \frac{\sum_{i \in L} precision(c_i)}{|L|}$$

$$macro\_recall(c_i) = \frac{\sum_{i \in L} recall(c_i)}{|L|}$$

Sendo $|L|$número total de classes de objetos.

Serão apresentados agora os resultados obtidos para a ontologia criada com uma abordagem mais minuciosa, e só foram considerados os 7 primeiros objetos do *dataset*, pois constatamos que os restantes objetos eram muito raros sendo que alguns nem faziam parte deste.

Na Figura 24 está representada a matriz de confusão referente à previsão dos objetos nas 25 amostras do *dataset* para os primeiros 7 objetos.

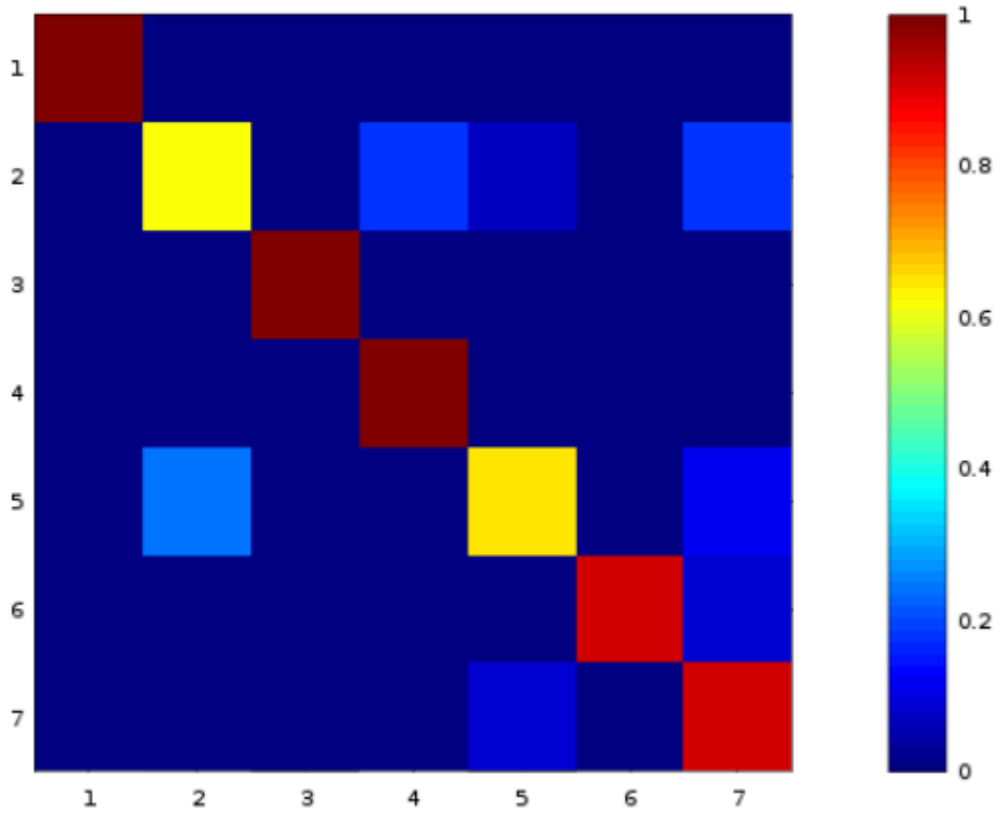


Figura 24 - Matriz de confusão com UMA-Offices dataset

As matriz de confusão tem uma dimensão de 7 por 7, sendo que as linhas representam os objetos reais presentes no *dataset* e as colunas os objetos identificados no processo de inferência. A representação a cores dos retângulos está em percentagem e demonstra a precisão sobre uma determinada classe. Por exemplo podemos observar que o objeto 1, que corresponde ao chão, teve uma precisão de 100% enquanto que o objeto 2 (parede) teve uma precisão de 60%, pois foi identificado como sendo parede, mas também foi identificado como sendo lado da mesa, costas da cadeira e monitor.

Por outro lado, temos o objeto 6 que corresponde ao assento da cadeira que teve uma precisão de 90% e teve um *recall* de 100%. Como é possível ver, sempre que algum objeto foi previsto como sendo assento da cadeira, a previsão estava correta.

Os resultados da *macro_precision* e *macro_recall* são 82% e 87% respetivamente.

De notar que muitos dos objetos identificados incorretamente devesse ao facto da má extração das características e reconstrução dos cenários a quando da criação do *dataset*. No caso da parede que foi o objeto em que tivemos piores resultados, muitas das vezes as suas características geométricas não se assemelhavam com uma parede, sendo esta do tamanho ou ainda mais pequena que um monitor, devido possivelmente a um erro do algoritmo de *plane-based mapping*.

Tendo estes aspetos em consideração os resultados obtidos podemos considerar como muito bons, aliás durante a preparação da demo PI_Kinect nós capturamos

vários cenários no Departamento de Eletrónica Telecomunicações e Informática (DETI) na Universidade de Aveiro, e criamos um *dataset* onde tivemos o cuidado de excluir cenários onde consideramos que o algoritmo de *plane-based mapping* tenha falhado a que chamamos DatasetUA. Num total de 6 cenários apresenta uma *macro_precision* de 100% e *macro_recall* de 100%, para os mesmo 7 objetos, utilizando uma ontologia também especifica. Pode ser encontrada mais informação no Apêndice 6.

Os resultado até aqui apresentados foi utilizando a ontologia criada segundo uma abordagem minuciosa.

A outra ontologia não a consideramos para os resultados pois desde cedo observamos que estávamos a obter resultados muito inferiores à expectativa. Isto é, inferiores a 50% tanto de *macro_precision* como de *macro_recall.* No entanto decidimos incluir para tentar explicar os motivos dos resultados tão inferiores ao esperado.

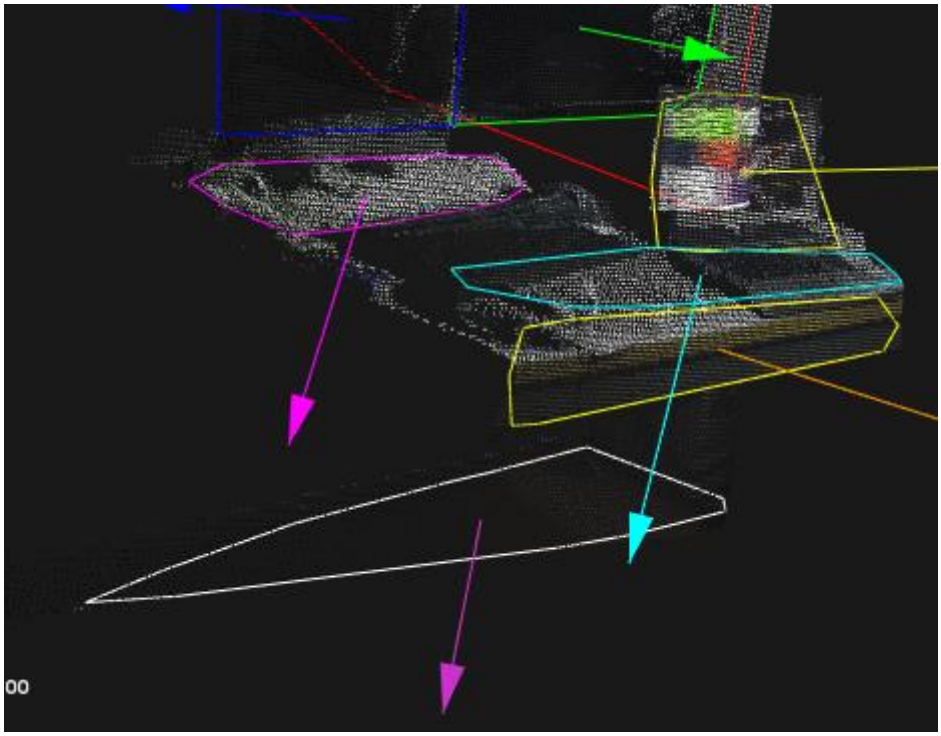


Figura 25 - Representação de uma cenário do dataset

O grande motivo dos resultados serem tão baixos deve-se ao facto de que o nosso conhecimento humano não ir de encontra com as características geométricas dos objetos representados no *dataset*. Por exemplo, como já foi referido, nós consideramos o chão como sendo um plano de grandes dimensões, mas na realidade, nos cenários do *dataset,* o chão correspondia a um plano de dimensões muito inferiores ao esperado como é possível ver na Figura 25.

5.1.2 Teste para obter número mínimo de amostras

Este teste tem como objetivo tentar relacionar os resultados obtidos em termos de *macro_precision e macro_recall* com o número de amostras sintéticas que são necessárias gerar para treinar a CRF.

Serão utilizados a ontologia minuciosa e o *dataset* UMA-Offices apresentados no ponto 5.1.1.

E configuração de treino utilizada foi a seguinte:

- *nodeLambda* = 2
- *edgeLambda* = 10

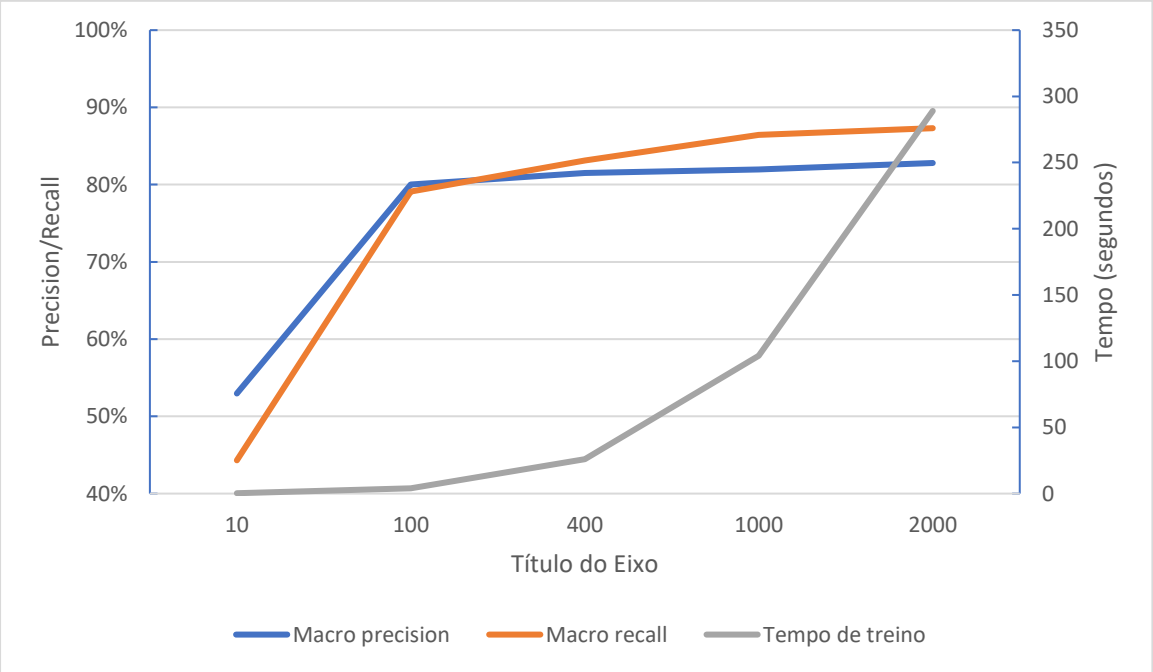


Figura 26 - Comparação de macro precision/recall e tempo de treino em função do número de amostras geradas.

Podemos ver na Figura 26 que a partir das 1000 amostras o tempo perdido na fase de treino não justifica a melhoria dos resultados e também é possível ver que o gráfico está a tender para se tornar constante, pelo que o segredo para obter melhores resultados passaria por arranjar melhores amostras de treino.

5.2 Resultados da demonstração PI_Kinect

A organização desta demonstração já foi apresentada anteriormente no ponto 4.3. Pelo que nesta secção serão demonstrados os resultados finais obtidos.

Esta demonstração foca-se para ser utilizada no contexto de sala de aulas do DETI da Universidade de Aveiro e inclusive foi exibida no Students@DETI.

A ontologia criada é diferente da criada para o *dataset* UMA-Office e encontra-se no apêndice 5.

A configuração de treino foi a seguinte:

- *nodeLambda* = 5
- *edgeLambda* = 15
- *nodeMultiplier* = [50,20,5,10,1]

A configuração referente ao *nodeMultiplier* já foi explicada no ponto 4.1.2. Seguidamente serão apresentados os resultados obtidos utilizando os cenários capturados por este conjuntos de aplicações que compõem o PI_Kinect.

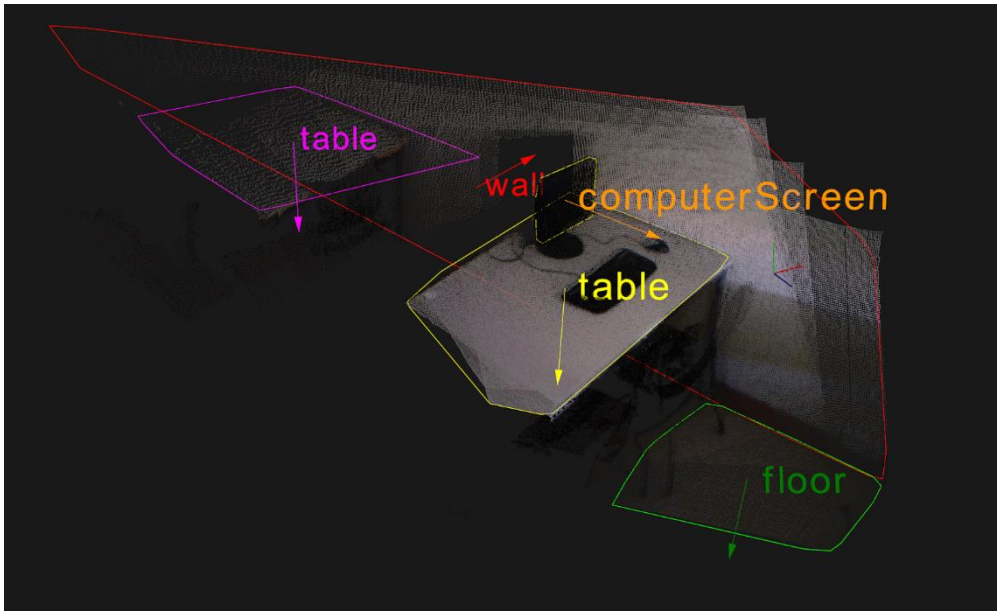


Figura 27 - Resultado final do cenário correspondente a uma sala do DETI

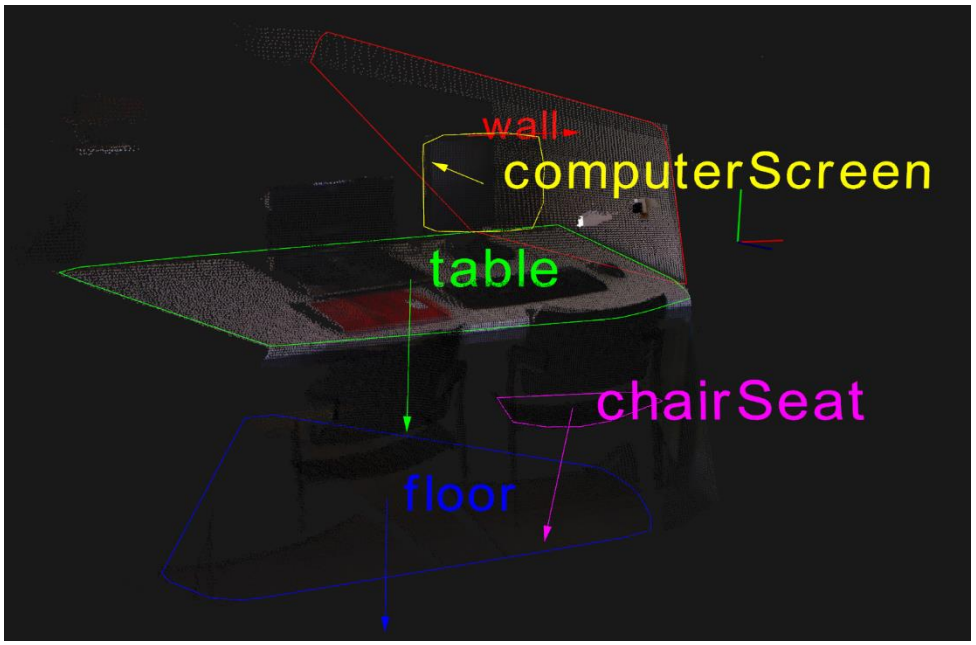


Figura 28 - Resultado final do cenário correspondente a outra sala do DETI

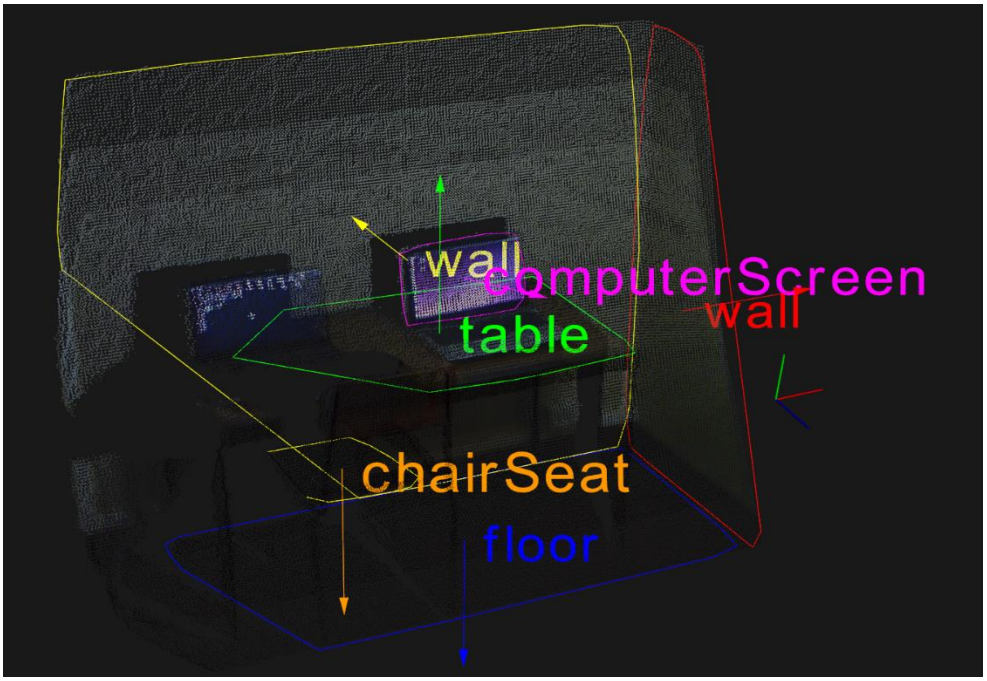


Figura 29 - Resultado final do cenário de demonstração durante o Students@DETI

Como é possível ver em todos os cenários representados (Figuras 27-29) fomos capazes de reconhecer todos os objetos presentes nestes. Relembrando que a CRF treinada foi a mesma para todos os cenários de teste aqui apresentados.

Capítulo 6

6 Conclusão

Este projeto apresenta uma solução para o problema do reconhecimento dos objetos, que une conhecimento semântico com modelos gráficos probabilísticos.

Durante a realização deste projeto, foi construída uma biblioteca em C++ que permite a criação, treino e inferência de modelos gráficos probabilísticos dedicados ao problema de reconhecimento dos objetos.

Foi implementado um algoritmo para a geração de amostras sintéticas tendo por base uma Ontologia, onde são descritos os objetos e as características destes através de elicitação humana.

Os modelos gráficos probabilísticos criados e treinados pela nossa biblioteca foram testados contra o UMA-Offices *dataset*, que conta com um total de 25 cenários de escritório, onde utilizando as métricas *precision* e *recall* obtivemos, respetivamente, 82% e 87%.

Ainda foi criada uma demonstração que consistia em, a partir de um sensor Kinect, recolher observações do cenário. Posteriormente passa-se à deteção dos planos e das suas características geométricas e recorrendo ao modelo probabilístico são visualizados os resultados que consistem nos planos a serem identificado como objetos.

Com base ainda na demonstração, foi construído um *dataset* dos cenários que corresponde às salas do DETI da Universidade de Aveiro. Esse *dataset* tem um total de 6 cenários e o modelo gráfico probabilístico obteve resultados, utilizando as mesma métricas que o UMA-Offices, de 100% *precision* e 100% *recall*.

É considerado como trabalho futuro o reconhecimento do ambiente (quarto, escritório, cozinha, entre outros), o reconhecimento em tempo real e a não dependência da captura do chão para a obtenção da altura do chão pois é uma característica a ser analisada aquando da identificação de objetos.

Referências

[1] J. R. Ruiz-Sarmiento, “Probabilistic Techniques in Semantic Mapping for Mobile Robotics,” University of Málaga, 2016.

[2] J. R. Ruiz-Sarmiento, C. Galindo, and J. Gonzalez-Jimenez, “Scene Object Recognition for Mobile Robots Through Semantic Knowledge and Probabilistic Graphical Models,” *Expert Syst. Appl.*, vol. 42, no. 22, pp. 8805–8816, 2015.

[3] J. R. Ruiz-Sarmiento, C. Galindo, and J. Gonzalez-Jimenez, “UPGMpp: a Software Library for Contextual Object Recognition,” in *3rd. Workshop on Recognition and Action for Scene Understanding (REACTS)*, 2015.

[4] J. R. Ruiz-Sarmiento, C. Galindo, and J. Gonzalez-Jimenez, “Exploiting Semantic Knowledge for Robot Object Recognition,” *Knowledge-Based Syst.*, vol. 86, pp. 131–142, 2015.

[5] E. Fernández-Moral, W. Mayol-Cuevas, V. Arévalo, and J. González-Jiménez, “Fast place recognition with plane-based maps,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013, pp. 2719–2724.

[6] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. 2009.

[7] J. M. Hammersley and P. E. Clifford, “Markov random fields on finite graphs and lattices,” *em Unpubl. Manuscr.*, 1971.

[8] “Protégé.” [Online]. Available: http://protege.stanford.edu/. [Accessed: 24-Jun-2017].

[9] “UGM.” [Online]. Available: http://www.cs.ubc.ca/~schmidtm/Software/UGM.html. [Accessed: 24-Jun-2017].

[10] “UPGMpp.” [Online]. Available: http://mapir.isa.uma.es/mapirwebsite/index.php/mapir-downloads/201-the-upgmpp-library.html. [Accessed: 24-Jun-2017].

[11] “GitHub UPGMpp.” [Online]. Available: https://github.com/jotaraul/upgmpp. [Accessed: 24-Jun-2017].

[12] “GitHub KinectSLAM6D.” [Online]. Available: https://github.com/MiguelAlgaba/KinectSLAM6D. [Accessed: 24-Jun-2017].

[13] “MRPT.” [Online]. Available: http://www.mrpt.org/. [Accessed: 24-Jun-2017].

[14] “PbMap algorithm.” [Online]. Available: http://www.mrpt.org/pbmap. [Accessed: 24-Jun-2017].

[15] “RGB-D.” [Online]. Available: https://rgbd-dataset.cs.washington.edu/imgs/rgbd.png. [Accessed: 24-Jun-2017].

[16] “CodeUA Repository projeto_lei_2017.” [Online]. Available:

http://code.ua.pt/projects/projecto_lei_2017/repository. [Accessed: 24-Jun-2017].

[17] “UMA-Offices Dataset.” [Online]. Available: http://mapir.isa.uma.es/mapirwebsite/index.php/mapir-downloads/244-uma-offices.html/. [Accessed: 24-Jun-2017].

Apêndice

1 - Exemplo de Ontologia em C++

```
OntologyConcept *object = new OntologyConcept("Object");
OntologyConcept *furniture = new OntologyConcept("Furniture", object);
OntologyConcept *building = new OntologyConcept("Building", object);
OntologyConcept *device = new OntologyConcept("Device", object);
OntologyConcept *table = new OntologyConcept("Table", furniture);

OntologyInstance *floor = new OntologyInstance("Floor", building);
OntologyInstance *wall = new OntologyInstance("Wall", building);
OntologyInstance *tableTop = new OntologyInstance("TableTop", table);
OntologyInstance *computerScreen = new OntologyInstance("ComputerScreen", device);

// floor unary
floor->setFrequencyOfOccurrence(O_VERY_HIGH);
floor->addUnaryFeature("has_orientation", Variance{0, U_NULL});
floor->addUnaryFeature("has_centroidHeight", Variance{0, U_NULL});
floor->addUnaryFeature("has_area", Variance{1.5, U_HIGH});
floor->addUnaryFeature("has_elongation", Variance{2.17, U_LOW});
// floor pairwise
floor->addRelationsFrequency(wall, P_LOW);
floor->addRelationsFrequency(tableTop, P_VERY_LOW);

// wall unary
wall->setFrequencyOfOccurrence(O_VERY_HIGH);
wall->addUnaryFeature("has_orientation", Variance{1, U_NULL});
wall->addUnaryFeature("has_centroidHeight", Variance{1, U_LOW});
wall->addUnaryFeature("has_area", Variance{1, U_HIGH});
wall->addUnaryFeature("has_elongation", Variance{2, U_LOW});
// wall pairwise
wall->addRelationsFrequency(floor, P_LOW);
wall->addRelationsFrequency(tableTop, P_VERY_HIGH);
wall->addRelationsFrequency(computerScreen, P_HIGH);

// computerScreen unary
computerScreen->setFrequencyOfOccurrence(O_HIGH);
computerScreen->addUnaryFeature("has_orientation", Variance{1, U_NULL});
computerScreen->addUnaryFeature("has_centroidHeight", Variance{1.02, U_NULL});
computerScreen->addUnaryFeature("has_area", Variance{0.23, U_MEDIUM});
computerScreen->addUnaryFeature("has_elongation", Variance{1.45, U_LOW});
// computerScreen pairwise
computerScreen->addPairwiseFeature("is_onTop", tableTop, P_VERY_HIGH);
computerScreen->addRelationsFrequency(wall, P_HIGH);
computerScreen->addRelationsFrequency(tableTop, P_ALWAYS);
```

2 - Exemplo do uso de SyntheticOptions

```
namespace PairwiseFunctionsTrain{
	double perpendicularity(OntologyInstance* instance1,VectorXd& feat1,OntologyInstance* instance2,VectorXd& feat2){
		// indice do feature segue a ordem do vetor unaryFeatureNameOrder
		return (feat1(0) - feat2(0) == 0) ? 0 : 1;
	}
	double heightDistance(OntologyInstance* instance1, VectorXd& feat1, OntologyInstance* instance2, VectorXd& feat2){
		return abs((float)feat1(1) - (float)feat2(1));
	}
	double isOn(OntologyInstance* instance1,VectorXd& feat1,OntologyInstance* instance2,VectorXd& feat2){
		int r = rand() % 100 ;
		int prob1 = frequeniaRelacoes(instance1->getPairwiseFeature("is_onTop", instance2));
		int prob2 = frequeniaRelacoes(instance2->getPairwiseFeature("is_onTop", instance1));
		if (prob1 > r || prob2 > r)
			return 1;
		else
			return 0;
	}
}

double desvioPadrao(Variance var) {
	switch (var.freq) {
		case U_VERY_HIGH:
			return var.value ;
		case U_HIGH:
			return var.value * 0.75;
		case U_MEDIUM:
			return var.value * 0.50;
		case U_LOW:
			return var.value * 0.25;
		case U_VERY_LOW:
			return var.value * 0.10;
		case U_NULL:
			return 0;
	}
}

SyntheticOptions syntheticOptions;
syntheticOptions.unaryFeaturesNameOrder = {"has_orientation", "has_centroidHeight", "has_area", "has_elongation"};
syntheticOptions.functionsPairwise.push_back(PairwiseFunctionsTrain::perpendicularity);
syntheticOptions.functionsPairwise.push_back(PairwiseFunctionsTrain::heightDistance);
syntheticOptions.functionsPairwise.push_back(PairwiseFunctionsTrain::isOn);
syntheticOptions.standardDeviation = desvioPadrao;
syntheticOptions.numberSamples = 1000;
```

3 - Pseudo-código do algoritmo de geração da amostra

Requere: ti: todas as instâncias

```
1: var amostra ← criar
2: para (instância ∈ ti)
3:       var random
4:       se (freq_ocorrer(instância) >random ) então
5:             var objeto← criar
6:             para cada( c ∈ caraterísticas_unarias(instância) )
7:                   adicionar(objeto,distrib_normal(c))
8:             adicionar(amostra,objeto)
9:
10: para (O1  ∈ objetos(amostra))
11:      para (O2  ∈ objetos(amostra))
12:            var random
13:            se (freq_relacao(O1, O2) >random) então
14:                  var relacao ←  criar
15:                  para cada( c ∈ caraterísticas_emparilhadas(O1,O2))
16:                        adicionar(relacao,c)
17:                  adicionar(amostra,relacao)
```

A função “adicionar(x1,x2)” permite adicionar o objeto x2 ao objeto x1.

O duplo ciclo nas linhas 10 e 11 deve ser feito com atenção, isto é, caso o O_1= A e O_2= B seja verificado já não é necessário verificar O_1=B e O_2=A, ficando assim com uma ordem de complexidade $O(\frac{n^2}{2})$.

4 - Formato da classe *Sample*

```
Sample *sample = new Sample(5, 4);
sample->nodeFeatures << 0, 0.745944,   1.20098,      1.68726,
                        1, 0.523566,   0.255477,     1.72744,
                        0, 0,          1.49268,      2.54707,
                        1, 0.359525,   0.268375,     2.8157,
                        0, 0.446692,   0.129895,     1;
sample->adjacency << 0,  1,  0,  1,  1,
                     1,  0,  1,  1,  0,
                     0,  1,  0,  1,  1,
                     1,  1,  1,  0,  1,
                     1,  0,  1,  1,  0;
Eigen::MatrixXi isOn_relation(5, 5);
isOn_relation.setZero();
sample->relations.push_back( isOn_relation );
sample->groundTruth << 2, 3, 0, 3, 5;
```

5 - Ontologia para DatasetUA em c++

```
OntologyConcept *object = new OntologyConcept("Object");
OntologyConcept *furnature = new OntologyConcept("Furniture", object);
OntologyConcept *bulding = new OntologyConcept("Building", object);
OntologyConcept *device = new OntologyConcept("Device", object);
OntologyConcept *chair = new OntologyConcept("Chair", furnature);
OntologyConcept *table = new OntologyConcept("Table", furnature);
OntologyConcept *pc = new OntologyConcept("PC", device);
//instancias
OntologyInstance *floor = new OntologyInstance("Floor", bulding);
OntologyInstance *wall = new OntologyInstance("Wall", bulding);
OntologyInstance *tableTop = new OntologyInstance("TableTop", table);
OntologyInstance *tableSide = new OntologyInstance("TableSide", table);
OntologyInstance *computerScreen = new OntologyInstance("ComputerScreen", device);
OntologyInstance *chairBack = new OntologyInstance("ChairBack", chair);
OntologyInstance *chairSeat = new OntologyInstance("ChairSeat", chair);

//floor unary
floor->setFrequencyOfOccurrence(O_VERY_HIGH);
floor->addUnaryFeature("has_orientation", Variance{0, U_NULL});
floor->addUnaryFeature("has_centroidHeight", Variance{0, U_NULL});
floor->addUnaryFeature("has_area", Variance{1.5, U_HIGH});
floor->addUnaryFeature("has_elongation", Variance{2.17, U_LOW});
//floor pairwise
floor->addRelationsFrequency(wall, P_LOW);
floor->addRelationsFrequency(tableTop, P_VERY_LOW);
floor->addRelationsFrequency(tableSide, P_MEDIUM);
```

```
//wall unary
wall->setFrequencyOfOccurrence(O_VERY_HIGH);
wall->addUnaryFeature("has_orientation", Variance{1, U_NULL});
wall->addUnaryFeature("has_centroidHeight", Variance{1, U_LOW});
wall->addUnaryFeature("has_area", Variance{1, U_VERY_HIGH});
wall->addUnaryFeature("has_elongation", Variance{3, U_HIGH});
//wall pairwise
wall->addRelationsFrequency(floor, P_LOW);
wall->addRelationsFrequency(tableTop, P_VERY_HIGH);
wall->addRelationsFrequency(tableSide, P_MEDIUM_HIGH);
wall->addRelationsFrequency(computerScreen, P_HIGH);

// tableTop unary
tableTop->setFrequencyOfOccurrence(O_ALWAYS);
tableTop->addUnaryFeature("has_orientation", Variance{0, U_NULL});
tableTop->addUnaryFeature("has_centroidHeight", Variance{0.62, U_NULL});
tableTop->addUnaryFeature("has_area", Variance{1, U_LOW});
tableTop->addUnaryFeature("has_elongation", Variance{1.5, U_LOW});
// tableTop pairwise
tableTop->addRelationsFrequency(floor, P_VERY_LOW);
tableTop->addRelationsFrequency(wall, P_VERY_HIGH);
tableTop->addRelationsFrequency(tableSide, P_ALWAYS);
tableTop->addRelationsFrequency(computerScreen, P_ALWAYS);

// tableSide unary
tableSide->setFrequencyOfOccurrence(O_LOW);
tableSide->addUnaryFeature("has_orientation", Variance{1, U_NULL});
tableSide->addUnaryFeature("has_centroidHeight", Variance{0.4, U_LOW});
tableSide->addUnaryFeature("has_area", Variance{0.15, U_MEDIUM});
tableSide->addUnaryFeature("has_elongation", Variance{8, U_HIGH});
// tableSide pairwise
tableSide->addRelationsFrequency(floor, P_MEDIUM);
tableSide->addRelationsFrequency(wall, P_MEDIUM_HIGH);
tableSide->addRelationsFrequency(tableTop, P_ALWAYS);

// computerScreen unary
computerScreen->setFrequencyOfOccurrence(O_VERY_HIGH);
computerScreen->addUnaryFeature("has_orientation", Variance{1, U_NULL});
computerScreen->addUnaryFeature("has_centroidHeight", Variance{0.66, U_NULL});
computerScreen->addUnaryFeature("has_area", Variance{0.1, U_MEDIUM});
computerScreen->addUnaryFeature("has_elongation", Variance{1.45, U_LOW});
// computerScreen pairwise
computerScreen->addPairwiseFeature("is_onTop", tableTop, P_VERY_HIGH);
computerScreen->addRelationsFrequency(wall, P_HIGH);
computerScreen->addRelationsFrequency(tableTop, P_ALWAYS);

// chairBackseat unary
chairBack->setFrequencyOfOccurrence(O_MEDIUM);
chairBack->addUnaryFeature("has_orientation", Variance{1, U_NULL});
chairBack->addUnaryFeature("has_centroidHeight", Variance{0.57, U_LOW});
chairBack->addUnaryFeature("has_area", Variance{0.15, U_MEDIUM});
chairBack->addUnaryFeature("has_elongation", Variance{1.13, U_LOW});

// chairSeat unary
chairSeat->setFrequencyOfOccurrence(O_MEDIUM);
chairSeat->addUnaryFeature("has_orientation", Variance{0, U_NULL});
chairSeat->addUnaryFeature("has_centroidHeight", Variance{0.43, U_LOW});
chairSeat->addUnaryFeature("has_area", Variance{0.05, U_MEDIUM});
chairSeat->addUnaryFeature("has_elongation", Variance{3.40, U_LOW});
```

6 - Resultados com o DatasetUA

Estes resultados foram incluídos no apêndice, visto que o *dataset* só possui 6 cenários e não consideramos que esse número fosse suficiente para ficar nas partes dos resultados, no entanto achamos importante fazer referência. O *dataset* está declarado no ficheiro "UADataset.cpp" dentro da aplicação PI_ObjectRecognition_Demo.

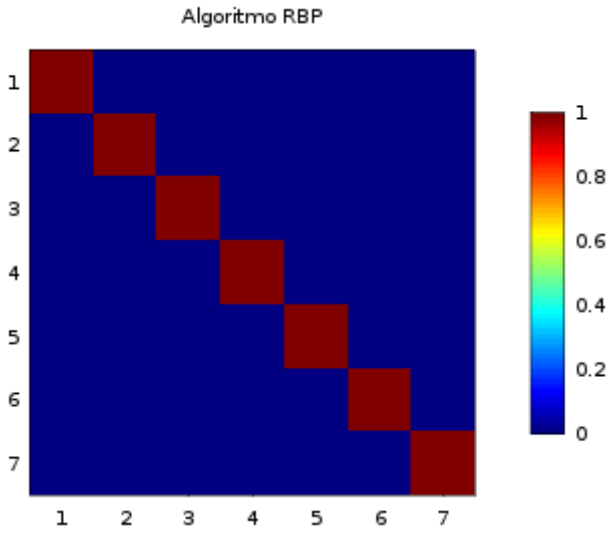


Figura 30 - Matriz de confusão do DatasetUA

Como é possível ver pela Figura 30, a CRF treinada teve uma desempenho de 100% tanto de *macro_precision* como de *macro_recall*.