Universidade Autónoma de Lisboa

Sistemas de Robótica 2017-2018

Professor Daniel de Matos Silvestre

1 Conceitos Essenciais

Nesta secção faz-se a revisão dos principais conceitos matemáticos necessários para a cadeira de Sistemas de Robótica.

1.1 Álgebra Linear

Nesta secção vamos utilizar $x \in \mathbb{R}^n$ como vector e $A \in \mathbb{R}^{n \times n}$ como matriz em todos os resultados e definições apresentados.

Para uma matriz quadrada A, pode-se definir valores próprios λ e respectivos vectores próprios v como os escalares e vectores que satisfazem a equação:

$$Av = \lambda v$$

o que intuitivamente define uma base para perceber o que acontece ao produto Ax. Para o caso de uma matriz com todos os valores próprios $\lambda_i, i = 1, \cdots n$ também temos os vectores próprios à esquerda que satisfazem:

$$w_i^\intercal A = \lambda_i w_i^\intercal$$

e que são mutuamente ortogonais com os v_i, isto é:

$$w_i^\intercal v_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{caso contrário.} \end{cases}$$

Temos então a igualdade:

$$A = \sum_{i=1}^{n} \lambda_i v_i w_i^\intercal$$

o que nos diz que o mapa linear defindo por Ax pode ser entendido escrevendo o x na base identificada pelo conjunto dos vectores próprios e escalando essa coordenada pelo respectivo valor próprio.

Outro ponto de interesse é a solução de um sistema de equações do tipo:

$$Ax = b \iff x = A^{-1}b$$

onde A^{-1} identifica a matriz inversa de A que satisfaz $A^{-1}A = AA^{-1} = I$ com I como a matriz identidade. A existência de matriz inversa está relacionada com os valores próprios da matriz serem todos diferentes de zero de forma a que o mapa linear não faça igual a zero todos os vectores ao longo de uma ou mais direcções.

Para um mapa linear A podemos também definir o seu espaço nulo (em inglês denominado também por kernel):

$$\text{null}(A) := \{v \in \mathbb{R}^n : Av = 0\}$$

e o espaço das colunas que é dado por:

$$\text{Im}(A) := \{Ax : x \in \mathbb{R}^n\}.$$

Por fim falta mencionar que estes dois espaços satisfazem $\dim(\text{Im}(A)) + \dim(\text{null}(A)) = n$ pelo Teorema Rank-Nullity, onde usamos $\dim(S)$ para denominar a função que dado um sub-espaço linear devolve o número de vectores linearmente independentes que compõem a base desse sub-espaço S.

1.2 Sistemas Dinâmicos Lineares

Nesta secção vamos apenas introduzir a formulação em espaço de estados de um sistema linear invariante no tempo (LTI - Linear Time-Invariant). Podemos escrever para o caso em tempo contínuo t:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \tag{1}$$

onde $\dot{x}(t) := \frac{\partial x(t)}{\partial t}$. Para o caso de tempo discreto k temos de forma equivalente:

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) + Du(k) \end{cases} . \tag{2}$$

Para ambos os casos (1) e (2) podemos escrever as soluções das equações à custa apenas do estado inicial $x(0) = x_0$ como:

$$x(t) = e^{At}x_0 + \int_0^t e^{A-\tau}Bu(\tau)d\tau$$
$$y(t) = Ce^{At}x_0 + \int_0^t Ce^{A-\tau}Bu(\tau)d\tau + Du(t)$$

e

$$x(k) = A^k x_0 + \sum_{\tau=0}^{k-1} A^{k-1-\tau}Bu(\tau)$$
$$y(k) = CA^k x_0 + \sum_{\tau=0}^{k-1} CA^{k-1-\tau}Bu(\tau) + Du(k)$$

que é estável (isto é o estado vai para zero) se $\mathrm{Re}[\lambda_i(A)] < 0, \forall i \leq n$ para o caso contínuo e $|\lambda_i(A)| < 1, \forall i \leq n$ para o discreto.

1.3 Sistemas Dinâmicos não Lineares

A importância do estudo de sistemas lineares advém de que podemos localmente aproximar sistemas não lineares por lineares e desenhar controladores que funcionam apenas numa região perto do ponto de operação.

Um sistema não linear pode ser escrito na forma mais simples como:

$$\dot{x}(t) = f(x(t))$$

ou

$$x(k+1) = f(x(k))$$

respectivamente para o caso continuo e discreto. A função não linear $f(\cdot)$ pode ser aproximada recorrendo à série de Taylor:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

onde a é o valor à volta do qual queremos aproximar, $n!$ representa o factorial de n e $f^n(a)$ é a derivada de ordem n da função f avaliada no ponto a. Um formato semelhante pode ser utilizado para o caso em que o $x(t)$ é um vector. Para obtermos um sistema linear poderíamos considerar utilizar apenas o primeiro e segundo termos da série.

1.4 Probabilidades

Muito sucintamente revemos apenas os conceitos fundamentais necessários à discussão relacionada com a cadeira. Uma variável aleatória é definida

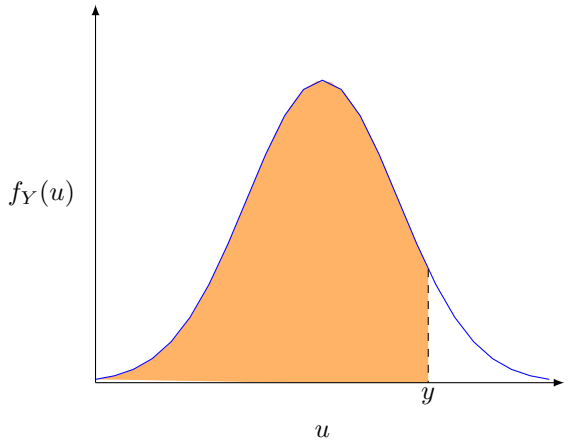


Figure 1: Ilustração da pdf e como se relaciona com a cdf da distribuição Normal.

através da sua distribuição:

$$X \sim D(p)$$

para uma certa distribuição D com parâmetros p. Uma das que mais vamos utilizar será a distribuição normal (variável contínua) $X \sim \mathcal{N}(\mu, \sigma^2)$ com valor esperado μ e variância σ^2. Na figura 1.4 está representada a distribuição normal a título de exemplo.

Para variáveis contínuas, como o exemplo na figura 1.4, é importante introduzir o conceito de função densidade de probabilidade (pdf de probability density function em inglês) que corresponde ao conceito de função de probabilidade para variáveis discretas. Embora no caso contínuo seja mais difícil de identificar dado que a $\mathbb{P}[X = x] = 0$ para qualquer valor de x, a ideia subjectiva é que traduz a concentração de probabilidade ao longo do domínio onde é definida a variável (isto é, o seu suporte).

Mais interessante é normalmente a função que dá a probabilidade dentro de um intervalo. Podemos calcular recorrendo à função de distribuição (cdf do inglês cumulative distribution function) que se equipara à função cumulativa de probabilidade para o caso discreto. Na figura 1.4, a cdf devolve a área sombreada por baixo da pdf. Portanto, se tivermos uma pdf de X, $f_X(x)$, e uma cdf $F_X(x)$, estas relacionam-se no caso contínuo:

$$\mathbb{P}[X \leq x] = F_X(x) = \int_{-\infty}^{x} f_X(t)dt$$

enquanto que no discreto toma o formato:

$$\mathbb{P}[X \leq x] = F_X(x) = \sum_{x_i \leq x} \mathbb{P}[X = x_i] = \sum_{x_i \leq x} p(x_i)$$

e em ambos os casos $\mathbb{P}[a \leq X \leq b] = F_X(b) - F_X(a)$.

A probabilidade de um evento pode também ser calculada se existir informação sobre outros eventos. Nesse caso estamos a definir o conceito de probabilidade condicionada:

$$\mathbb{P}[A|B] = \frac{\mathbb{P}[A \cap B]}{\mathbb{P}[B]}$$

e à custa podemos identificar que eventos independentes satisfazem $\mathbb{P}[A|B] = \mathbb{P}[A]$ e de forma semelhante $\mathbb{P}[B|A] = \mathbb{P}[B]$. Ou seja, eventos independentes são tais que saber informação sobre um deles não acrescenta nada sobre o outro. Neste caso, podemos também através de alguma manipulação chegar à conclusão que eventos independentes satisfazem $\mathbb{P}[A \cap B] = \mathbb{P}[A]\mathbb{P}[B]$.

Apenas faltam introduzir dois teoremas. O teorema da probabilidade total permite escrever a probabilidade de um evento à custa de uma partição de todo o espaço de eventos em n distintos $\{B_n : n = 1, 2, 3, \cdots\}$ que sejam disjuntos, ou seja $\forall i, j : B_i \cap B_j = \emptyset$, e que utiliza a expressão para a união

$$\mathbb{P}[A \cup B] = \mathbb{P}[A] + \mathbb{P}[B] - \mathbb{P}[A \cap B].$$

O teorema da probabilidade total na sua formulação mais simples pode ser escrito como:

$$\mathbb{P}[A] = \sum_n \mathbb{P}[A \cap B_n]$$

ou em alternativa

$$\mathbb{P}[A] = \sum_n \mathbb{P}[A|B_n]\mathbb{P}[B_n].$$

A formula também pode ser aplicada para calcular uma probabilidade condicionada na forma:

$$\mathbb{P}[A|C] = \sum_n \mathbb{P}[A|C \cap B_n]\mathbb{P}[B_n|C].$$

Na área da robótica, o interesse é normalmente estimar uma variável do nosso sistema à custa de algumas medidas efectuadas pelos sensores a bordo. É também de interesse um resultado que nos permita calcular a probabilidade de um certo estado sabendo as medidas com base na probabilidade das medidas sabendo o estado. Temos então o Teorema de Bayes que assume a formulação

$$\mathbb{P}[X|Y] = \frac{\mathbb{P}[Y|X]\mathbb{P}[X]}{\mathbb{P}[Y]}$$

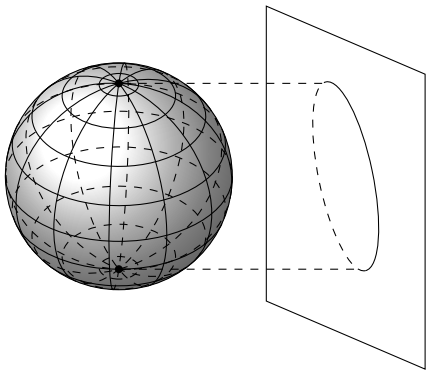


Figure 2: Ilustração da projeção de um objecto em 3D numa fotografia.

ou de forma mais extensa para os eventos $\{X_i\}$:

$$\mathbb{P}[X_i|Y] = \frac{\mathbb{P}[Y|X_i]\mathbb{P}[X_i]}{\sum_j \mathbb{P}[Y|X_j]\mathbb{P}[X_j]}.$$

2 Imagem

Uma imagem é o resultado de uma conversão da luz recebida no receptor da câmara. A luz é amostrada em diversos pontos que são denominados por pixéis e guardados numa matriz $A \in \mathbb{R}^{n \times m}$. O número de pixéis numa imagem é a resolução, ou seja, quantas unidades mais pequenas de luz são guardadas na imagem. Um ponto fundamental é perceber que uma fotografia é uma projecção do mundo real num plano 2D tal como ilustrado na figura 2.

De seguida fazemos a revisão de alguns algoritmos relacionados com imagem que são importantes no contexto da robótica.

2.1 Algoritmos para Visão

2.1.1 Detecção de arestas

A detecção de arestas tem como base a diferença de cor entre os vários pixéis e pode ser útil em vários casos como: localizar o horizonte, os cantos de objectos, determinar formas, etc. O pseudo-código é apresentado no algoritmo 1 seguinte.

Uma das questões mais fundamentais é como escolher o valor de $threshold$. No caso de se escolher um valor muito alto apenas alguns pixéis serão marcados e talvez seja difícil de reconstruir as arestas. No sentido contrário e há muitos elementos marcados que não fazem parte.

Algorithm 1 Detecção de arestas.

Require: Matriz $A \in \mathbb{R}^{n \times m}$ contendo os valores de cor da imagem.
Ensure: Identifica a 1 os pixéis pertencentes a arestas.

1: **for each** pixel (i, j) em A **do**
2: /* *Comparar o pixel mais escuro e claro entre os vizinhos de i,* $V(A, i, j)$ */
3: **if** $\max V(A, i, j) - \min V(A, i, j) > threshold$ **then**
4: /* *Marcar o pixel* (i, j) *como aresta* */
5: $A_{ij} = 1$
6: **else**
7: $A_{ij} = 0$
8: **end if**
9: **end for**
10: **return** A

2.1.2 Detecção de forma

Utilizando o algortimo 1 podemos construir exemplos mais sofisticados. Num primeiro exemplo estamos apenas interessados em descobrir formas simples no algoritmo 2.

Algorithm 2 Detecção de formas.

Require: Matriz $A \in \mathbb{R}^{n \times m}$ contendo os valores de cor da imagem.
Ensure: Identifica as formas.

1: **for each** pixel (i, j) em A **do**
2: /* *Obter as arestas* */
3: $A =$ Algoritmo 1
4: /* *Calcular vector de direcção actual e médio* */
5: $v = (i, j) - (k, \ell)$
6: $v_{\text{avg}} = \text{avg}(v_{\text{avg}}, v)$
7: /* *Se a variação do ângulo for grande* */
8: **if** $\text{ang}(v, v_{\text{avg}}) > threshold$ **then**
9: /* *Contar nova aresta* */
10: $count = count + 1$
11: **end if**
12: **end for**
13: **return** Forma consoante base de dados.

São possíveis outros algoritmos com base em thresholds para a cor como classificação de pixéis ou descoberta de objectos com base numa só cor.

2.1.3 Redes neuronais

Recentemente tem havido desenvolvimentos na área de redes neuronais que permitem utilizar para a área de visão. Nesta secção é apresentado apenas o modelo mais simples do perceptrão ilustrado na figura 3.

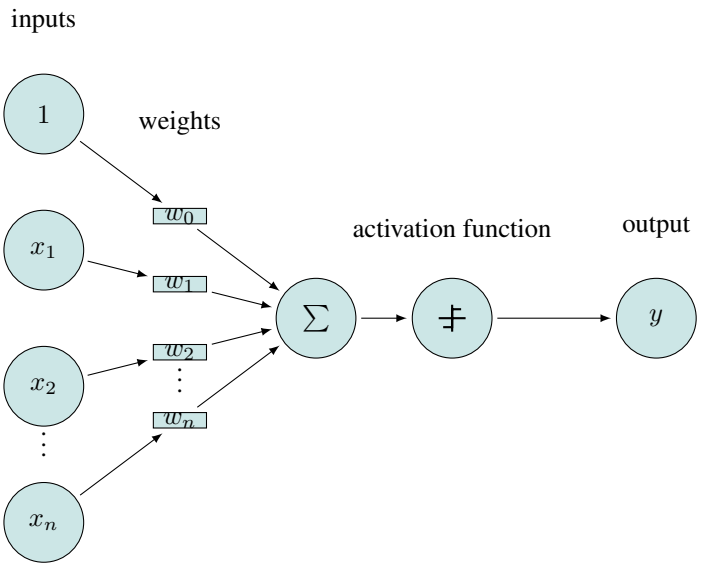


Figure 3: Esquema do perceptrão.

Algorithm 3 Back-propagation.

Require: Training Set T, network $\mathcal{R}$, taxa aprendizagem α e $iterations_{\max}$.
Ensure: Trained network.

/* *Initializar pesos* */
$\mathcal{R} = initWeights(\mathcal{R})$
/* *Back-propagation* */
for each $i = 1$ to $iterations_{\max}$ **do**
    /* *Obter input* i */
    $input_i \leftarrow getInput(T)$
    /* *Calcular output* */
    $output_i \leftarrow forwardPropagate(input_i, \mathcal{R})$
    /* *Calcular erro* */
    $error_i \leftarrow backwardPropagateError(input_i, output_i, \mathcal{R})$
    /* *Actualizar os pesos* */
    $\mathcal{R} \leftarrow updateWeights(input_i, output_i, \mathcal{R}, \alpha)$
end for
return $\mathcal{R}$

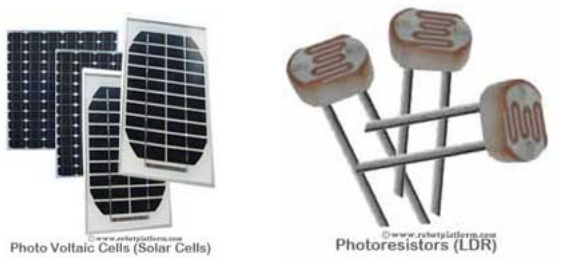


Figure 4: Sensores de luz (células fotovoltaicas à esquerda e foto-resistências à direita).

Uma rede neuronal pode ser treinada com base em dados usando um algoritmo aprendizagem supervisionada como por exemplo o back-propagation que se ilustra no algoritmo 3.

3 Sensores e Actuadores

Na construção de um robot introduzimos na Secção 2 alguns mecanismos relacionados com a utilização de imagem para capturar informação sobre o mundo envolvente de forma a desenhar controladores e algoritmos para robótica. Nesta secção mencionamos alguns dos restantes sensores e actuadores que serão úteis.

Do ponto de vista da orientação e navegação (excluindo por exemplo a necessidade de determinar luz, som, temperatura, etc para uma função específica), sensores são essenciais para reunir variáveis como:

- posição do veículo $x \in \mathbb{R}^3$;
- velocidade linear $v \in \mathbb{R}^3$;
- velocidade angular $w \in \mathbb{R}^3$;
- mapa $A \in \mathbb{R}^{n \times m}$;
- atitude $S \in SO(3)$, angulos de Euler, etc;
- aceleração linear $a \in \mathbb{R}^3$;
- aceleração angular $\alpha \in \mathbb{R}^3$;
- etc.

3.1 Sensores

3.1.1 Sensores de luz

Os sensores de luz mais utilizados são foto-resistências e células foto-voltaicos apresentados na figura 4.

Figure 5: Sensores de som.

Figure 6: Sensores de temperatura.

O meio de funcionamento de um sensor de luz como uma foto-resistência é um circuito construído num semi-condutor que aquando da incidência de luz conduz mais electrões. Desta forma a variação de tensão permite medir a existência de luz.

3.1.2 Sensores de som

Os sensores de som são tipicamente microfones que convertem ondas sonoras em valores de voltagem proporcional à intensidade do som. Este tipo de sensores estão representados na figura 5.

Para robots mais complexos, microfones podem ser utilizados para detecção de voz e discurso e utilizar isso para as suas actividades ou para o seu controlo. A implementação de sensores de som é mais complicada que sensores de luz pois estes geram um intervalo de tensão menor e por isso necessitam de amplificadores e circuitos adicionais.

3.1.3 Sensores de temperatura

Nalgumas situações os robots necessitam de monitorizar a temperatura ambiente e transmitir para a entidade que gere a missão. Exemplos destes são termo-resistências e outros. Um exemplo está mostrado na figura 6.

Alguns destes funcionam de uma forma parecida com os sensores de luz onde a resistência à passagem de electrões é menor quanto maior for a temperatura.

Figure 7: Sensores de contacto.

3.1.4 Sensores de contacto e proximidade

A navegação de um robot num ambiente requer evitar obstáculos. Uma alternativa é utilizar sensores de contacto como push buttons ou semelhantes que funcionam como um interruptor. Quando existe contacto com objectos, estes sensores (um exemplo pode ser visto na figura 7) são activados permitindo a identificação de obstáculos.

Existem exemplos de sensores de contacto capacitivos que reagem apenas a contacto com o ser humano como os presentes nos ecrã dos smartphones. Uma desvantagem é a detecção de um objecto apenas aquando do toque. Em alternativa podemos implementar sensores de proximidade. Sensores deste tipo funcionam emitindo um sinal electromagnético ou campo magnético e analizando o sinal recebido. Três tipos são bastante utilizados:

Infravermelhos Um emissor de infravermelhos emite um sinal que é reflectido quando encontra um obstáculo que é posteriormente medido no receptor. Também é possível medir distâncias com base em infravermelhos;

Ultrasom Através de um som de alta frequência e da recepção do eco é possível detectar a proximidade de objectos ou até medir a distância;

Foto-resistências Este sensor de luz pode ser utilizado para medir proximidade de objectos que tapam a recepção de luz.

3.1.5 Sensores de distância

Sensores de distância funcionam de forma idêntica aos de proximidade pelo que podemos sumarizar os seguintes:

Ultrasom Dada a descrição de como funciona um sensor de ultrasom para detectar proximidade de objectos pode ser utilizado para medir distâncias com base de que no ar o som viaja a cerca de 344 m/s. Medindo o tempo de round-trip calcula-se a distância ao obstáculo;

Infravermelho estes sensores utilizam triangulação para medir o ângulo de recepção do sinal reflectido e determinar a distância com base em triangulação;

Laser range De forma semelhante aos ultrasom, funciona medindo a latência de propagação do laser no percurso de ida e reflexão para o obstáculo. Particularmente interessante para longas distâncias dada a velocidade da luz;

Câmaras stereo Câmaras com mais que uma lente das quais é possível retirar informação sobre distância.

3.1.6 Navegação e posicionamento

A tarefa mais fundamental num robot que se desloque é determinar a sua posição e navegar pelo mundo envolvente. Alguns dos métodos têm limitações no que toca a serem utilizados no interior ou exterior. Exemplo de sensores utilizados em navegação e posicionamento:

GPS - Global Positioning System utilizado em ambientes exteriores serve para determinar a posição e velocidade com base em triangulação de sinais enviados de satélites orbitando a terra;

Bússola digital permite obter medições de direcção com base no campo magnético da terra;

Localização através de Visão Usando features artificalmente colocadas ou naturais é possível obter a localização e deslocamento do robot;

Acelerómetro equipamento que permite medir a aceleração em que direcção que pode ser usada para determinar a posição;

Giroscópio mede a taxa de rotação em torno de um eixo que pode ser utilizado sem ter de depender da força de gravidade;

IMU - Inertial Measurement Unit Combina 2 ou mais sensores entre acelerómetros, giroscópios, magnómetros, etc para medir a orientação, velocidade e força gravítica.

3.1.7 Outros sensores

Em vários robots consoante a sua função podem ser encontrados muitos outros sensores:

Pressão Especialemente importantes para medir pressão em mãos robóticas;

Voltagem Detecção de diferenças de potencial ampliando de forma a ser utilizado para alguma função;

Corrente Equivalente ao de voltagem mas para a corrente;

Humidade ;

Gas ;

Potenciómetros ;

etc .

3.2 Actuadores

Recentemente são fruto de investigação e desenvolvimento alguns actuadores passivos que guardam a energia para a depois aplicarem quando necessário. Nesta disciplina estamos interessados apenas em actuadores activos (motores). Os motores são utilizados para criar movimento linear ou rotacional e estaremos apenas interessados em motores eléctricos sejam eles DC (Direct Current) ou AC (Alternate Current) que funcionam com corrente directa ou alternada.

4 Sistemas de Navegação Inercial

A tarefa mais simples que podemos desenhar para um robot é a sua deslocação no espaço em duas dimensões se terrestre ou em três se considerarmos veículos aéreos ou submarinos. A realização desta tarefa requer o desenho de um Sistema de Navegação Inercial (SNI) se não houver a possibilidade de receber de uma unidade externa a localização e orientação do nosso robot (seja GPS no exterior ou sistemas de localização interna com base em câmaras ou outros).

Um SNI utiliza as variáveis do nosso sistema de robótica:

- posição $x(t)$;
- velocidade $v(t)$;
- aceleração $a(t)$;
- atitude $R(t)$.

Para já excluímos aceleração e velocidade angular. A utilização de uma IMU e algum processamento com base na relação entre estas variáveis:

$$\dot{x}(t) = v(t)$$
$$\dot{v}(t) = a(t)$$

Em conjunto temos então o sumário de um SNI na Figura 4.

Embora a tarefa do SNI pareça simples como integrar algumas equações diferenciais, a verdade é que os sensores utilizados têm ruído. Mesmo em condições favoráveis, a posição tem um drift bastante acentuado por estar a somar repetidos erros com o novo ruído. Idealmente uma fusão

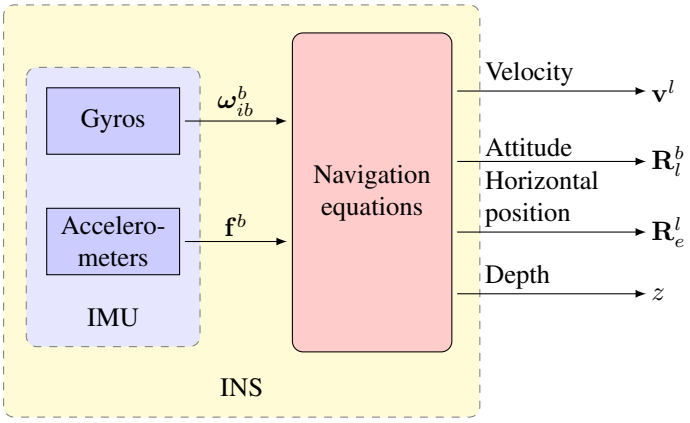


Figure 8: Ilustração de um Sistema de Navegação Inercial.

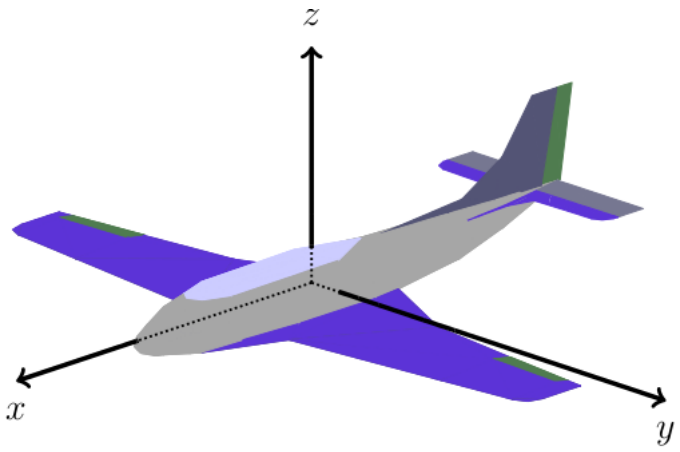


Figure 9: Referencial no corpo de um avião.

destas medidas permite melhores resultados, contudo o tópico está fora do âmbito da disciplina. As várias variáveis já foram introduzidas, embora seja necessário mais algum detalhe no tratamento da atitude.

4.1 Atitude

A função das variáveis de atitude é determinar a orientação do robot no espaço em função do referencial definido do corpo e do referencial inercial. Um exemplo do referencial do corpo é apresentado na Figura 9. Idealmente, o responsável pela sua definição deverá respeitar que todos os eixos seja ortogonais e seguindo a ordem do produto externo.

No formato anglo-saxónico, as rotações em torno do eixo x, y e z são denominadas de *roll*, *pitch* e *yaw*, respectivamente. Dada a sua definição, as rotações não são comutativas pelo que assumiremos a ordem xyz.

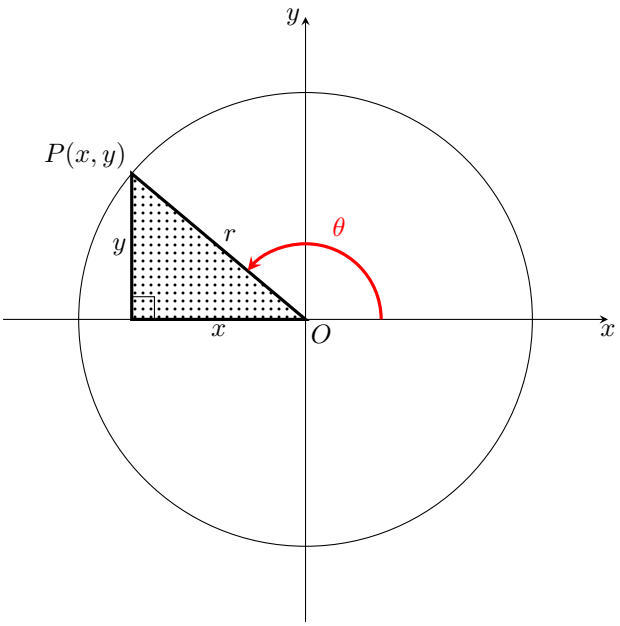


Figure 10: Rotação em $\mathbb{R}^2$.

Vamos começar por investigar uma rotação em $\mathbb{R}^2$.

A Figura 4.1 ilustra uma rotação a partir do referencial até um ponto com coordenadas (x, y). Usando trigonometria temos o seguinte sistema de equações:

$$\begin{bmatrix} x \\ y \end{bmatrix} = r \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_{\text{inicial}} \\ y_{\text{inicial}} \end{bmatrix} \tag{3}$$

sendo as coordenadas originais do ponto $(x_{\text{inicial}}, y_{\text{inicial}})$. A norma do vector que vai da origem ao ponto deverá ser constante, pelo que temos $r = 1$. A matriz de rotação é a que aparece na (3). O eixo na Figura 4.1 pode ser colocado em 3 dimensões considerando que o eixo z é ortogonal aos restantes e que não aparece visível dado a ponto de vista superior. Dessa forma temos então que a rotação encontrada em $\mathbb{R}^2$ é na verdade a rotação em torno do eixo do z e que ao longo desse eixo qualquer ponto não sofre alteração. Dessa forma temos:

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4}$$

Prosseguindo de forma semelhante ao feito na (4) mas com os restantes

eixos temos as duas matrizes de rotação simples em falta

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}, R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}.$$

Qualquer rotação pode ser descrita à custa de sucessivas rotações simples de tal forma que $R = R_z R_y R_x$.

5 Mobilidade

Esta secção é dedicada aos vários tipos de mobilidade para robots e à introdução de alguns modelos simples para veículos terrestres que serão o objecto de exemplo noutras componentes da disciplina.

5.1 Tipos de locomoção

Existem naturalemente várias formas de locomoção para robots dependendo do meio onde se aplicam. Vamos apresentar as várias opções tendo atenção a vantagens e desvantagens.

5.1.1 Rodas

A utilização de rodas é o meio de locomoção mais utilizado e aparece em vários formatos sendo o mais comum 3 ou 4 rodas. No caso de 3 rodas, duas são motrizes e a terceira é omnidirecional e serve para estabilidade.

Vantagens

- tipicamente low-cost;
- desenho e construção simples;
- possibilidade de escolher entre várias medidas consoante qualquer necessidade;
- usando 6 rodas podemos substituir outros mecanismos como as lagartas;
- possibilidade de serem feitas em vários materiais;
- ideal para uma primeira abordagem à robótica.

Desvantagens

- pode perder tração;
- pequena área de contacto.

5.1.2 Lagartas

Essencialmente seguindo a utilização em veículos militar todo-o-terreno, são boas utilizações em terreno irregular ou arenoso. O facto de existir bastante contacto reduz a perda de tração e ajuda a distribuir o peso do robot.

Vantagens

- contacto constante com o chão reduz a possibilidade de perder tração;
- peso distribuido de forma igual sobre o meio de locomoção.

Desvantagens

- a viragem causa uma força lateral que pode danificar ou o terreno ou causar desgaste;
- pouca oferta de diferentes soluções;
- maior complexidade mecânica.

5.1.3 Pernas

Um robot movido por pernas tem como objectivo uma maior flexibilidade mas também uma aparência mais humana. As configurações mais populares são com 2, 4 e 6 pernas e são capazes de lidar com terrenos mais adversos.

Vantagens

- parecido com o movimento orgânico natural;
- pode evitar grandes obstáculos e andar e terrenos muito irregulares.

Desvantagens

- aumento da complexidade mecânica, electrónica e de programação;
- menor relação bateria/consumo;
- maior custo.

5.1.4 Aéreos

Existem vários tipos de locomoção incluindo asas e hélices. A escolha prende-se com a maior manobrabilidade ou melhor desempenho das baterias.

Vantagens

- Excelente para vigilância.

Desvantagens

- acidentes são tipicamente mais desastrosos.

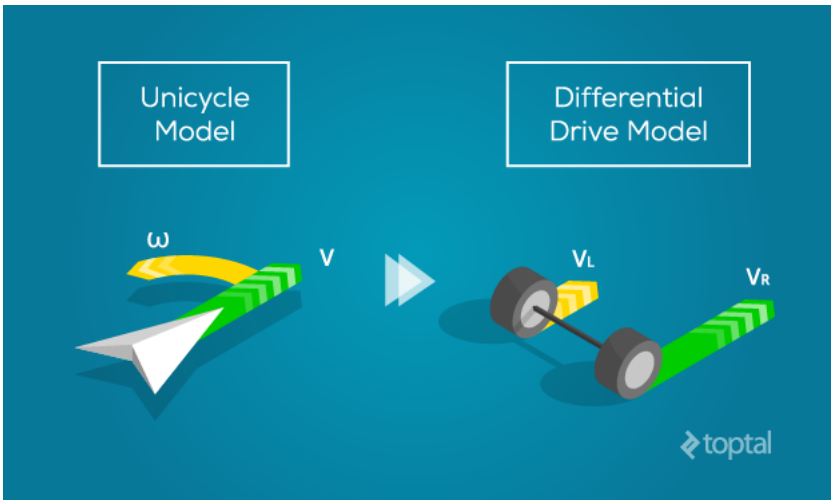


Figure 11: Ilustração do modelo diferencial e do uniciclo.

5.1.5 Aquáticos

No meio aquático a maior diferença é a técnica utilizada para submergir: usando compartimentos de ar comprimido, propulsores, barbatanas ou até asas. A propulsão é tipicamente efectuada através de propulsores a hélice.

Vantagens

- maior aplicabilidade possível dado que a maior parte do planeta é coberta por mar.

Desvantagens

- facilidade de perda do equipamento seja por falha de comunicações, ficar preso ou afundar;
- profundidades maiores que 10 metros implicam grandes investimentos.

6 Modelos de cinemática

Nesta secção iremos estudar o modelo diferencial e do uniciclo apresentados na Figura 11.

As variáveis de interesse para escrever este modelo são:

- L - separação entre rodas;
- R - raio das rodas;
- θ - ângulo de direcção a partir do eixo do x;

- u_ℓ - velocidade angular aplicada à roda esquerda;
- u_r - velocidade angular aplicada à roda direita.

6.1 Modelo diferencial

A magnitude do vector de velocidade do robot (isto é quanta distância por unidade de tempo é que é percorrida) é dada pela média das velocidades angulares aplicadas a cada roda e proporcional ao respectivo raio. Utilizando trigonometria podemos calcular que porção do vector de velocidade é convertido em deslocamento ao longo de x e y.

A velocidade de rotação é proporcional à diferença entre as velocidades angulares das rodas e do raio das rodas, mas inversamente proporcional com a distância entre estas. Utilizando todas estas informações chegamos ao modelo diferencial para o deslocamento do robot

$$\begin{aligned} \dot{x} &= \frac{R}{2}(u_\ell + u_r)\cos\theta \\ \dot{y} &= \frac{R}{2}(u_\ell + u_r)\sin\theta \\ \dot{\theta} &= \frac{R}{L}(u_r - u_\ell). \end{aligned} \tag{5}$$

A equação (5) significa que a velocidade em translação é $v = \frac{u_\ell + u_r}{2}$ enquanto que a rotação é dado por $\omega = u_r - u_\ell$.

6.2 Modelo Uniciclo

Um robot com um modelo diferencial como em (5) pode ser simplificado utilizando um modelo de um uniciclo e fazendo a posterior conversão entre velocidade e rotação e velocidade angular das rodas.

Para tal começamos por definir:

$$\begin{aligned} u_v = R\frac{u_\ell + u_r}{2} &\iff u_r = \frac{2}{R}u_v - u_\ell \\ u_\omega = \frac{R}{L}(u_r - u_\ell) &\iff -u_\ell = \frac{L}{R}u_\omega - u_r \end{aligned}$$

e que substituindo u_ℓ da segunda na primeira, obtém-se

$$u_r = \frac{2}{R}u_v + \frac{L}{R}u_\omega - u_r$$

o que resulta nas equações finais que convertem entre os dois modelos:

$$\begin{aligned} u_r &= \frac{1}{R}u_v + \frac{L}{2R}u_\omega \\ u_\ell &= \frac{1}{R}u_v - \frac{L}{2R}u_\omega. \end{aligned}$$

7 Filtro de Kalman

Após termos introduzido sensores e actuadores e modelos de dinâmica de um robot, interessa referir que num cenário realista, existe ruído e perturbações que afectam a capacidade de termos certeza das variáveis do nosso sistema. Nesta secção introduzimos o filtro de Kalman como solução para estimar as variáveis de interesse do robot em questão.

O filtro de Kalman assume que a dinâmica de um sistema é representada pelo seguinte modelo:

$$\begin{aligned} x_k &= F_k x_{k-1} + B_k u_k + w_k \\ z_k &= H_k x_k + v_k \end{aligned} \tag{6}$$

onde:

- x_k é o estado;
- F_k é a matriz da dinâmica;
- B_k é a matriz que define a entrada de actuação;
- u_k é o vector de actuação;
- w_k é um vector de perturbações que tem distribuição $\mathcal{N}(0, Q_k)$;
- H_k é uma matriz que define os sensores;
- v_k é um vector de ruído com distribuição $\mathcal{N}(0, R_k)$.

Temos duas fases do funcionamento do filtro. Uma fase de previsão onde a incerteza em relação ao estado anterior é propagada com a dinâmica e uma fase de update onde existe uma média entre a previsão e a distribuição dada pelas medidas dos sensores.

Fase de Previsão

Nesta fase aplicamos directamente a dinâmica do estado da equação (6) e obtemos a estimativa (valor esperado) e a incerteza (dada pela matriz de covariância):

$$\begin{aligned} \hat{x}_{k|k-1} &= F_k \hat{x}_{k-1|k-1} + B_k u_k \\ P_{k|k-1} &= F_k P_{k-1|k-1} F_k^\intercal + Q_k \end{aligned}$$

Fase de Update

Nesta fase começamos por medir a *inovação* fornecida pelos sensores através da estimativa e da covariância:

$$\begin{aligned} \tilde{y}_k &= z_k - H_k \hat{x}_{k|k-1} \\ S_k &= H_k P_{k|k-1} H_k^\intercal + R_k \end{aligned}$$

e, evitando mostrar os cálculos dado fugir ao âmbito da cadeira, o ganho óptimo é

$$K_k = P_{k|k-1} H_k^\intercal S_k^{-1}. \tag{7}$$

A estimativa já após o update (ou seja a estimativa à posteriori) é dada por:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k$$
$$P_{k|k} = (I - K_k H_k) P_{k|k-1} (I - K_k H_k)^\intercal + K_k R_k K_k^\intercal$$

para um ganho genérico K_k. Usando o ganho em (7) é possível simplificar a expressão da matriz de covariância à posteriori para:

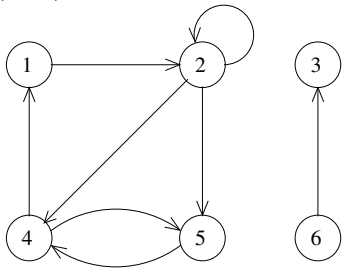
$$P_{k|k} = (I - K_k H_k) P_{k|k-1}.$$

Uma implementação ilustrativa pode ser encontrada em https://www.cs.utexas.edu/ teammco/misc/

8 Procura em Grafos

8.1 Definição de grafos

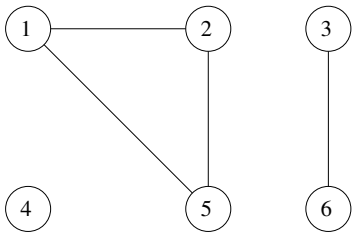
- Um grafo $G = (V, E)$ consiste num conjunto finito de *nós* V e um conjunto finito de *ligações* E.
  - *Grafo direcional*: E é um conjunto ordenado de pares de vertices (u, v) onde $u, v \in V$



V = {1, 2, 3, 4, 5, 6}

E = {(1,2), (2,2), (2,4), (2,5)
(4,1), (4,5), (5,4), (6,3)}

  - *Grafo não direcional*: E é um conjunto não ordenado de pares $\{u, v\}$ where $u, v \in V$



V = {1, 2, 3, 4, 5, 6}

E = {{1,2}, {1,5}, {2,5}, {3,6}}

- O *grau* de um nó num grafo não direcional é dado pelo seu número de vizinhos.

Figure 12: Conversão do mapa para o formato de grafo incluíndo paredes como obstáculos.

- Existe uma definição semelhante para grafos direcionais fazendo a distinção do grau de entrada e saída.
- Um *caminho* de u_1 para u_2 é uma sequência de nós $< u_1{=}v_0, v_1, v_2, \cdots, v_k{=}u_2 >$ de forma a que $(v_i, v_{i+1}) \in E$ (ou $\{v_i, v_{i+1}\} \in E$)
  - Dizemos que u_2 é *alcançável* a partir de u_1
  - O *tamanho* do caminho é k
  - É descrito como um *ciclo* se $v_0 = v_k$
- Um grafo não direcional é dito *ligado* se para cada par de nós existe um caminho
- Grafos aparecem em muitas aplicações, como por exemplo:
  - Árvores ($|E| = |V| - 1$)
  - Identificar conectividade ou dependências (planos de uma caso, páginas web e as suas ligações, ou seja, topologia da internet)
- Em alguns cenários as ligações (u, v) têm pesos $w(u, v)$, por exemplo
  - Rede de estradas (distâncias)
  - Redes de computadores (capacidade)

Do ponto de vista da cadeira de robótica, uma das aplicações mais úteis de grafos (para além de modelarem acções e estados) é a possibilidade de representar o mapa onde o robot se desloca. De uma forma simples, cada célula que o robot poderá ocupar será um nó no grafo. A definição de grafo é ideal para representar obstáculos como deslocações entre nós que não são possíveis (ou seja não colocando uma ligação entre dois nós). Um exemplo disto mesmo aparece na Figura 12.

Num mapa de maior dimensão, esta conversão seria como demonstrado na Figura 13.

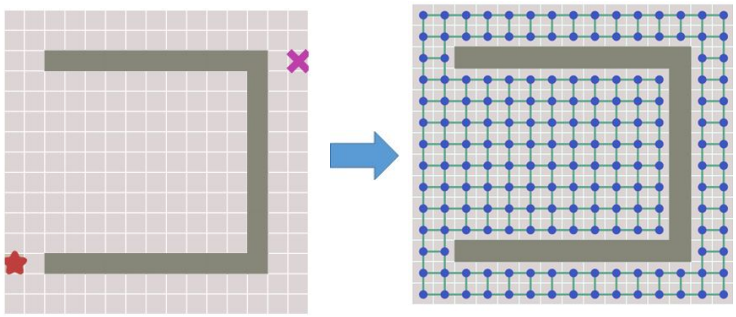


Figure 13: Conversão do mapa para o formato de grafo incluíndo paredes como obstáculos.

8.2 Algoritmos de procura em grafos sem pesos

- Há duas abordagens standard e simples de pesquisar ao longo de um grafo de forma sistemática
  - Breadth-first-search (BFS)
  - Depth-first-search (DFS)

8.2.1 Breadth-first search (BFS)

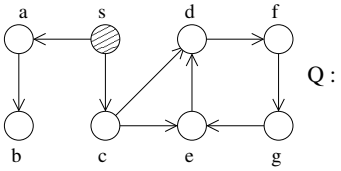
- Ideia principal:
  - Iniciar num nó s e visitar,
  - todos os nós com distância 1,
  - Seguindo com todos os nós de distância 2,
  - Seguindo com todos os nós de distância 3,
  - $\vdots$
- BFS calcula a distância do *caminho mais curto* em número de nós desde s para qualquer outro nó.
- Para ilustrar o funcionamento do algoritmo BFS, atribuímos cores aos passos do algoritmo
  - *Branco* aquando do início,
  - *Cinzento* após visitarmos o nó mas antes de visitar todos os seus vizinhos,
  - *Preto* após visitar o nó e os seus vizinhos (todos os vizinhos estão marcados a cinzento).
- Utilizamos uma fila Q para manter todos os nós ainda marcados a cinzento—aqueles que já visitámos mas não estão descartados.

- Algoritmo:

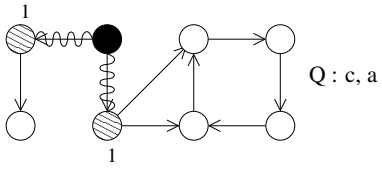
```
BFS(s)
    color[s] = cinzento
    d[s] = 0
    adicionaFila(Q, s)
    WHILE Q não é vazio DO
        retiraFila(Q, u)
        FOR (u, v) ∈ E DO
            IF color[v] = branco THEN
                color[v] = cinzento
                d[v] = d[u] + 1
                parent[v] = u
                adicionaFila(Q, v)
            END
            color[u] = preto
    END
```

- Exemplo (para grafos direcionais):

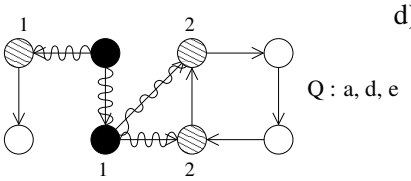
a)



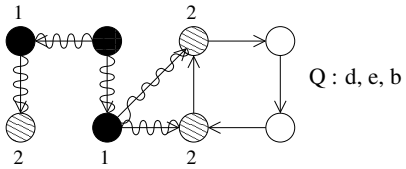
b)



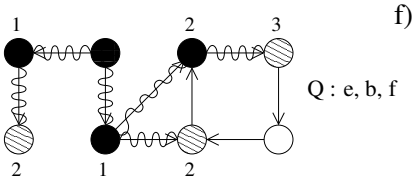
c)



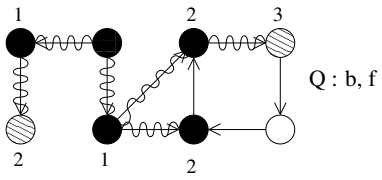
d)



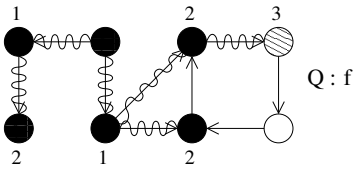
e)



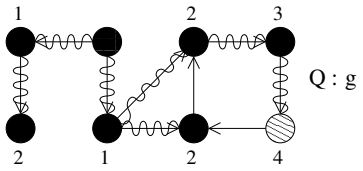
f)



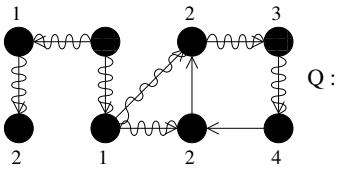
g)



h)



i)



- Nota:
  - parent[v] forma uma árvore; *BFS-tree*.
  - $d[v]$ guarda o comprimento do menor caminho entre s e v.
  - Podemos utilizar parent[v] para achar o caminho mais curto de s para qualquer nó.
- Se o grafo não é ligado teremos de iniciar o processo em todos os nós de forma a garantir que todas as componentes do grafo são testadas.

```
FOR each vertex u ∈ V DO
    IF color[u] = branco THEN BFS(u)
END
```

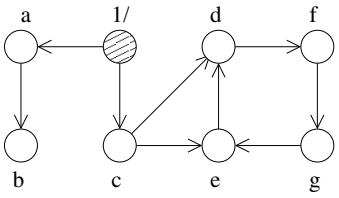
8.2.2 Depth-first search (DFS)

- Se usarmos uma pilha em vez de uma fila Q temos uma alternativa ao BFS que priviligia a profundidade do grafo em vez da largura:
  - Vamos o "mais profundo possível",
  - Voltamos atrás até encontrarmos nós não visitados,
  - Vamos novamente o "mais profundo possível",
  - $\vdots$
- Podemos então escrever o DFS usando o mesmo algoritmo do BFS mas com uma pilha em vez de uma fila
- Algorithm:

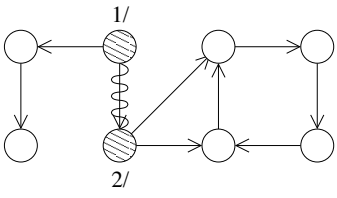
```
DFS(u)
    color[u] = cinzento
    d[u] = time
    time = time + 1
    FOR (u, v) ∈ E DO
        IF color[v] = branco THEN
            parent[v] = u
            DFS(v)
        END
    END
    color[u] = preto
    f[u] = time
    time = time + 1
```

- Exemplo:

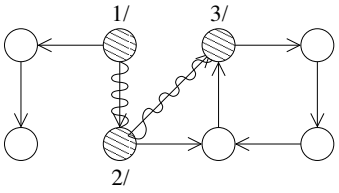
a)



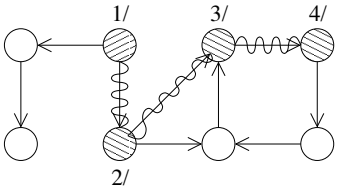
b)



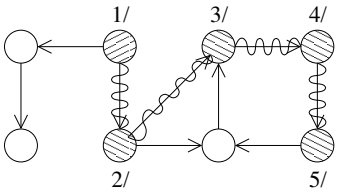
c)



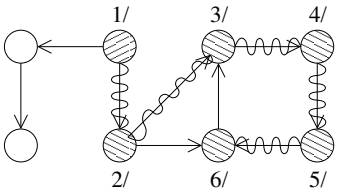
d)



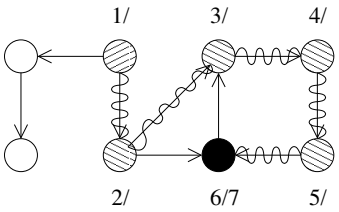
e)



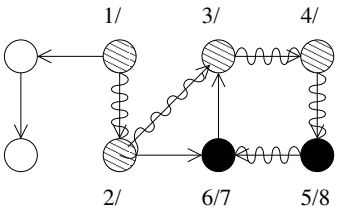
f)



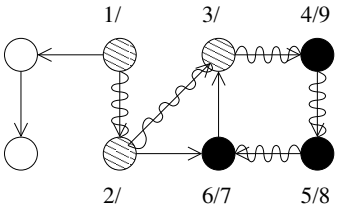
g)



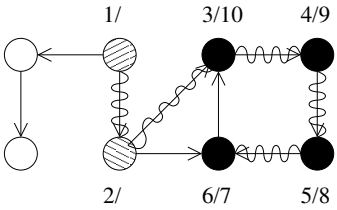
h)



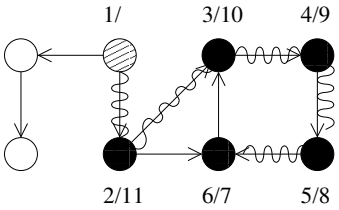
i)



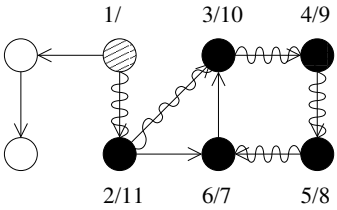
j)



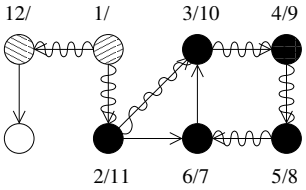
k)



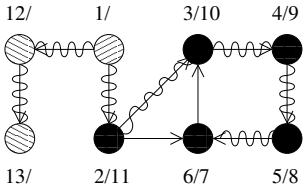
l)



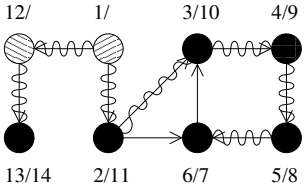
m)



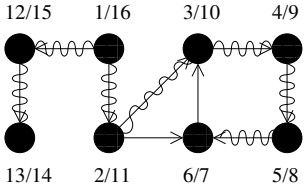
n)



o)



p)



- Como anteriormente, parent[v] forma uma árvore; *DFS-tree*
  - Nota: If u é descendente de v na DFS-tree então $d[v] < d[u] < f[u] < f[v]$

Uma implementação do BFS e do DFS pode ser encontrada em:

https://cs.stanford.edu/people/abisee/tutorial/bfsdfs.html

onde podemos verificar como cada algoritmo passa pelos vários nós no grafo. Inclusivé podemos testar o exemplo com obstáculos.

8.2.3 Algoritmo Greedy

Algoritmos do tipo *greedy* (que traduzido directamente significa gananciosо) procuram optimizar uma certa heurística em cada passo. Exemplos de heurísticas são funções que medem distância, erro, etc e que nos dão uma indicação se uma certa acção está na direcção de melhorar ou não o estado actual.

Para o caminho mais curto podemos utilizar:

- Distância Euclidiana — norma do vector que vai do estado actual ao destino;
- Distância de Manhattan — soma das coordenadas do vector que vai do estado actual ao destino;
- entre outras.

O algoritmo proposto é testar um novo nó do grafo cuja heurística utilizada seja a menor. Este algoritmo pode retornar soluções aproximadas e mais longas do que a óptima. Um exemplo ilustrativo pode ser

encontrado em:

https://cs.stanford.edu/people/abisee/tutorial/greedy.html

8.2.4 A*

O algoritmo A* foi desenvolvido por Peter Hart, Nils Nilsson e Bertram Raphael em Stanford em 1968 com o objectivo de permitir ao robot Shakey the Robot navegar numa sala com obstáculo de forma mais eficiente. Naturalmente, o algoritmo combina a rapidez de uma estratégia greedy com o mecanismo do BFS.

A principal diferença está no facto de que A* escolhe que nó n visitar a seguir como o nó que minimiza a soma:

$$\text{custo}(s, n) + \text{heuristica}(n)$$

onde custo(s, n) é o custo de ir desde o nó inicial s até ao n e heuristica(n) é o valor da heurística de distância entre n e o destino d.

Uma ilustração do seu funcionamento em comparação com o BFS e o greedy pode ser vista em:

https://cs.stanford.edu/people/abisee/tutorial/astar.html

A vantagem principal de A* é que é bastante mais rápido que BFS em casos comuns mas retorna o valor óptimo desde que:

$$\text{heuristica}(n) \leq \text{menor\_caminho}(n, destino).$$

8.3 Algoritmos de procura em grafos com pesos

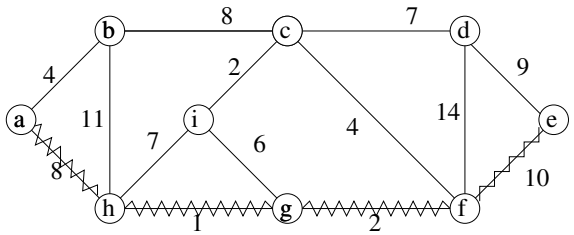
Na secção anterior considerámos apenas algoritmos em que cada ligação entre os nós tinha o peso fixo de uma unidade. Vamos agora considerar o caso mais genérico em que atribuímos pesos diferentes.

8.3.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra é a versão para um grafo com pesos do BFS. Em cada iteração vai visitar e incrementar os respectivos custos dos vizinhos do nó com menor custo até ao momento. Um exemplo interactivo para ver a comparação entre o BFS e o Dijkstra pode ser visto em

https://cs.stanford.edu/people/abisee/tutorial/dijkstra.html

onde se percebe claramente que o BFS segue o caminho mais curto mas não o caminho com o custo menor, enquanto que o Dijkstra prefere valorizar o custo. Quando pretendemos o menor custo estamos a referir no exemplo seguinte ao caminho entre os nós a e e (comprimento 21)



O algoritmo pode ser dado em pseudo-código:

Dijkstra(s)

FOR each $v \in V$ DO

$d[v] = \infty$

INSERT(Q, v, ∞)

$S = \emptyset$

$d[s] = 0$

CHANGE($Q, s, 0$)

WHILE Q not empty DO

$u =$ DELETEMIN(Q)

$S = S \cup \{u\}$

FOR each $e = (u, v) \in E$ with $v \in V \setminus S$ DO

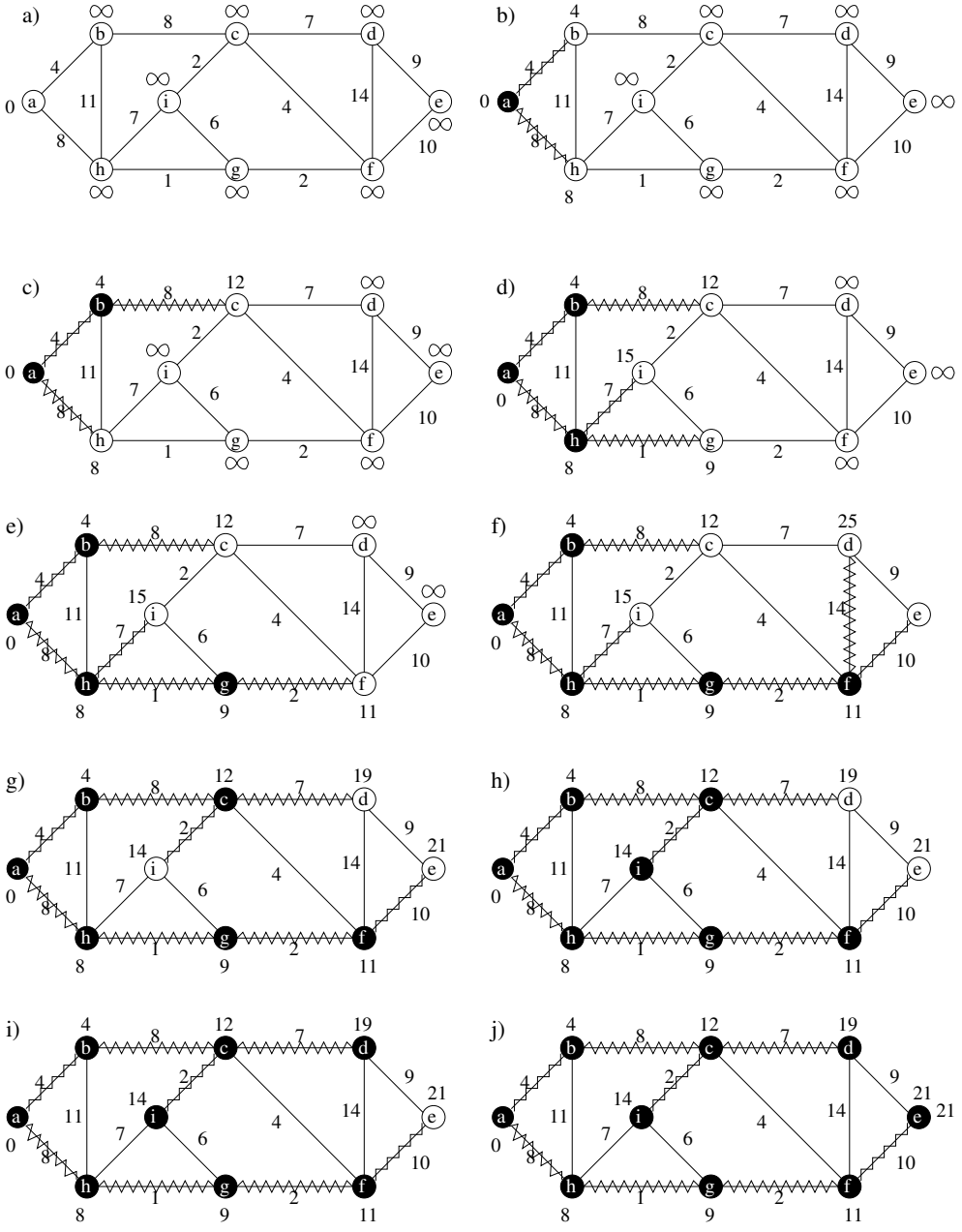
IF $d[v] > d[u] + w(u, v)$ THEN

$d[v] = d[u] + w(u, v)$

CHANGE($Q, v, d[v]$)

parent[v] $= u$

Exemplo:


a)
b)
c)
d)
e)
f)
g)
h)
i)
j)
Vertex in S
Vertex in V \ S

8.3.2 A* com pesos

O algoritmo A* sem pesos considerava o nó cujo número de passos desde a source e o número de passos estimados até ao destino era o menor. Para modificar para grafos com pesos temos apenas de considerar minimizar:

$$\mathrm{cost}(s, n) + \mathrm{heuristica_c ost}(n)$$

onde agora a função $\mathrm{cost}(s, n)$ mede a soma dos pesos do caminho desde s até ao nó n e $\mathrm{heuristica_c ost}(n)$ mede o valor aproximado da soma dos pesos do nó n até ao destino d.

Uma comparação entre o Dijkstra e A* para grafos com pesos pode ser encontrada em:

https://cs.stanford.edu/people/abisee/tutorial/customize.html

Desta forma temos que o Dijkstra é óptimo mas lento enquanto que o A* é óptimo mas mais rápido.

9 Controladores

Existem muitas técnicas de desenhar controladores para sistemas dinâmicos, dos quais sistemas de robótica são uma pequena parte. No seguimento de outras disciplinas do curso (onde foram estudados sistemas de uma variável e controladores do tipo PID - Proportional Integral Derivative), aproveitamos para introduzir de uma forma mais superficial a teoria de funções de Lyapunov.

Uma função de Lyapunov sobre o estado x do nosso sistema satisfaz as seguintes características:

$V(x) \geq 0$ A intuição por trás desta condição é que a função de Lyapunov irá medir a *energia* do nosso estado;

$V(x) = 0 \iff x = 0$ Queremos que a energia seja zero apenas quando o sistema atingiu o ponto desejado;

$\lim_{x \to \infty} V(x) = \infty$ esta condição é necessária para evitar que a função esteja a mascarar o crescimento do estado.

Se tivermos uma função $V(x)$ nas condições acima e que $\dot{V}(x) < 0$ então temos que $x \to 0$.

Com base no conceito de funções de Lyapunov e de optimização, vimos sem recorrer a provas, uma ferramenta disponível por exemplo no

Matlab. Um controlador óptimo para um sistema linear pode ser dado pelo LQR (Linear Quadratic Regulator) que minimiza:

$$\int_0^\infty x^\intercal Q x + u^\intercal R u + 2x^\intercal N u dt \tag{8}$$

onde:

- a matriz Q estabelece a ponderação dada à energia do estado;
- a matriz R faz o equivalente para a energia do sinal de controlo;
- a matriz N para explicitar energia entre termos cruzados do estado com o controlo.

O resultado dessa minimização é um controlador do tipo $u = -Kx$, que pode ser obtido em Matlab através da função:

```
1    [K,S,e] = lqr(SYS,Q,R,N);
```

onde as matrizes Q, R e N são as mesmas em (8), a variável SYS é um sistema dinâmico no Matlab e o output corresponde ao seguinte:

K ganho do controlador;

S solução da equação de Riccati (que está relacionado com a optimização feita);

e são os novos valores próprios após o controlo, i.e., da dinâmica $A - BK$.

10 Trajectórias

Nesta secção vamos estudar alguns dos algoritmos utilizados para a definição de trajectórias. Muitos deles podem ser vistos como a aplicação de algoritmos de procura em grafos no grafo correspondente ao mapa onde o robot se encontra.

Começamos por assumir que o nosso robot é holonómico:

Definição 1 (Robot holonómico)**.** *Um robot holonómico é tal que a quantidade de graus de liberdade do seu movimento é maior ou igual à quantidade total de graus de liberdade.*

Para robots holonómicos podemos ignorar quaisquer restrições e utilizar os seguintes algoritmos:

Campo potencial artificial robot desloca-se na direcção que minimiza o potencial dado por uma função a definir (do tipo algoritmo greedy);

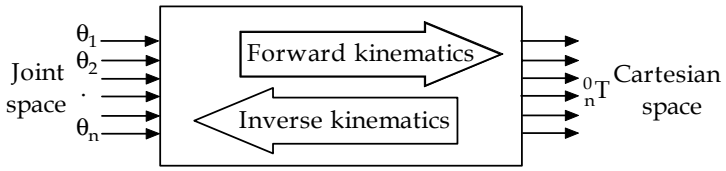


Figure 14: Desenho esquemático sobre a cinemática.

planeamento por grid este é semelhante ao que vimos na pesquisa por grafos onde o mapa é convertido num grafo e depois podemos executar por exemplo o $A^\star$ para encontrar o caminho;

planeamento por amostra para evitar desenhar o grafo completo, uma amostra de pontos é retirada e apenas esse grafo é utilizado na procura;

planeamento por recompensa algoritmos como processos de decisão de markov treinam com base em experiência de forma a tomarem escolhas no próximo passo que maximizam as recompensas futuras (maiores por ficarem mais perto do destino);

planeamento por machine learning Usando por exemplo uma rede neuronal, é treinada com base em tentativas anteriores e outras trajectórias de forma a tomar a decisão do caminho a escolher.

11 Cinemática

A cinemática lida com a transformação entre as variáveis que definem os vários elementos interligados por junções num robot e as respectivas coordenadas cartesianas. Um desenho esquemático está representado na Figura 14.

Como podemos observar, existem duas formas de descobrir as equações da cinemática de um braço robótico. A primeira opção é usar a cinemática directa, onde seguimos a ordem de interligações e braços desde o corpo do robot até à extremidade final. Numa outra abordagem, podíamos definir o ponto final da sequência de braços e achar as configurações que fazem sentido, isto é, a cinemática inversa. Nesta disciplina, vamos ver apenas a cinemática directa, tendo apenas o conhecimento que também existe a inversa dado que esta normalmente involvem muito mais cálculos.

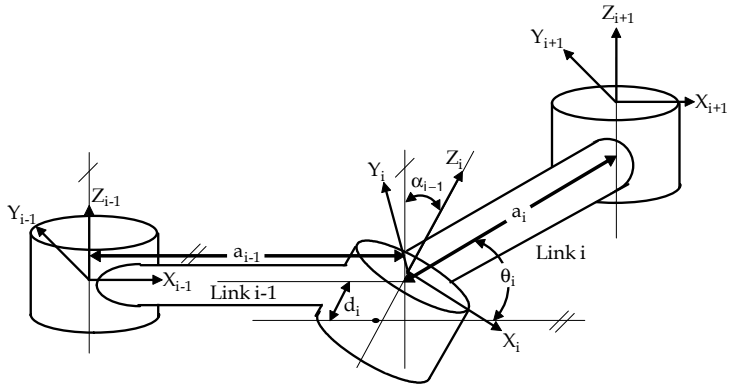


Figure 15: Desenho esquemático sobre a cinemática.

11.1 Cinemática directa

Vamos utilizar o exemplo na Figura 15 na derivação da transformação feita através da cinemática directa. Algo que à primeira vista poderia causar estranheza será o facto de tratarmos algo em $\mathbb{R}^3$ usando o espaço de $\mathbb{R}^4$, mas o objectivo é poder fazer translações e rotações usando apenas multiplicações de matrizes.

Como podemos observar na Figura 15, existem duas rotações dadas pelos respectivos ângulos α_{i-1} and θ_i em torno do eixo do x e z respectivamente. Aparecem também duas translações causadas pelas dimensões do braço. Sendo assim temos que a transformação da base $i-1$ para a primeira junção i tem a seguinte transformação:

$$
\begin{aligned}
T_i^{i-1} &= R_x(\alpha_{i-1})D_x(a_{i-1})R_z(\theta_i)D_z(d_i) \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_{i-1} & -\sin\alpha_{i-1} & 0 \\ 0 & \sin\alpha_{i-1} & \cos\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1}d_i \\ \sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

o que significa que podemos escrever $T^{\text{inicial}}_{\text{final}} = T^{\text{inicial}}_1 T^1_2 \cdots T^{n-1}_{\text{final}}$ e que

tem o formato:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

para uma matriz de rotação R dada pelas entradas r_{ij} e a posição dada por p_x, p_y and p_z.

12 Restrições Não-holonómicas

Intuitivamente, já referimos na secção sobre trajectórias o que é um robot holonómico. Recuperemos a definição genérica de um sistema robótico. Estes são governados por equações diferenciais do tipo:

$$\dot{x} = f(x)$$

onde a variável x contém diversas variáveis do sistema. Se identificarmos as variáveis correspondendo aos vários graus de liberdade (tipicamente rotações e translações). A título de exemplo um drone de 4 hélices tem 6 graus de liberdade correspondendo às rotações em torno de x, y, e z e as respectivas translações. De notar que o facto de a rotação poder ser positiva ou negativa não aumenta o número de graus de liberdade. De notar que os graus de liberdade podem ser menos que o espaço onde o robot está. Por exemplo se no modelo diferencial as rodas funcionarem à mesma velocidade, existe apenas um grau de liberdade pois o robot poderá apenas deslocar-se para a frente ou para trás numa única direcção.

Se de forma explicita considerarmos $q_1, \cdots, q_n$ como as várias variáveis do sistema, então o facto de um robot holonómico ser capaz de movimentar-se ao longo de cada um dos seus graus de liberdade pode ser descrito matematicamente como:

$$g(q_1, \cdots, q_n, t) = 0. \tag{9}$$

Qualquer robot cujas restrições não possam ser escritas no formato de (9), diz-se que é não-holonómico.

13 Materiais Avançados

Os alunos interessados podem querer aprofundar os seus conhecimentos no que toca a problemas de estimação relativas ao estado do sistema a partir das medidas vindas do sensor (semelhante ao conseguido através do filtro de Kalman) ou na área de controlo de múltiplos veículos. Os

alunos podem escolher qualquer dos tópicos para executar o trabalho adicional.

Tópicos relacionados com a convergência de formações de robots estão nos artigos [1], [2], [3] e [4].

Como visto durante a Unidade Curricular, a estimação do estado do sistema dinâmico representando o sistema robótico em causa é um papel fundamental, para o qual foi introduzido o filtro de Kalman como método estocástico. Tópicos relacionados com a estimação no sentido determinístico estão disponíveis em [5], [6], [7] e [8].

A modelação não prevê a existência de erros ou falhas tanto nos sensores como atuadores que precisam de ser detectados. Utilizando os métodos de estimação podem ser definidos algoritmos de detecção de falhas como os exemplos em [9], [10], [11], [12] e [13].

Muitos algoritmos utilizados em software ou hardware podem ser modelados usando técnicas semelhantes às apresentadas para os modelos dinâmicos de sistemas robóticos como por exemplo em [14] e [15].

Um último exemplo mostra como é possível descobrir a fonte de um comportamento em que os nós de uma formação seguem um determinado algoritmo de propagação em [16].

References

[1] D. Antunes, D. Silvestre, and C. Silvestre, "Average consensus and gossip algorithms in networks with stochastic asymmetric communications," in *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pp. 2088–2093, Dec 2011.

[2] D. Silvestre, P. Rosa, J. P. Hespanha, and C. Silvestre, "Finite-time convergence policies in state-dependent social networks," in *American Control Conference (ACC), 2015, Chicago, Illinois, USA.*, pp. 1041–1046, July 2015.

[3] D. Silvestre, P. Rosa, J. P. Hespanha, and C. Silvestre, "Set-consensus using set-valued observers," in *American Control Conference (ACC), 2015, Chicago, Illinois, USA.*, July 2015.

[4] D. Silvestre, J. P. Hespanha, and C. Silvestre, "Broadcast and gossip stochastic average consensus algorithms in directed topologies," *IEEE Transactions on Control of Network Systems*, pp. 1–1, 2018.

[5] D. Silvestre, P. Rosa, J. P. Hespanha, and C. Silvestre, "Self-triggered set-valued observers," in *European Control Conference (ECC)*, pp. 3647–3652, July 2015.

[6] D. Silvestre, P. Rosa, J. P. Hespanha, and C. Silvestre, "Fault detection for LPV systems using set-valued observers: A coprime factorization approach," *Systems & Control Letters*, vol. 106, pp. 32 – 39, 2017.

[7] D. Silvestre, P. Rosa, J. P. Hespanha, and C. Silvestre, "Set-based fault detection and isolation for detectable linear parameter-varying systems," *International Journal of Robust and Nonlinear Control*, vol. 27, no. 18, pp. 4381–4397, 2017. rnc.3814.

[8] D. Silvestre, P. Rosa, J. P. Hespanha, and C. Silvestre, "Self-triggered and event-triggered set-valued observers," *Information Sciences*, vol. 426, pp. 61 – 86, 2018.

[9] D. Silvestre, P. Rosa, R. Cunha, J. Hespanha, and C. Silvestre, "Gossip average consensus in a byzantine environment using stochastic set-valued observers," in *52nd IEEE Conference on Decision and Control*, pp. 4373–4378, Dec 2013.

[10] D. Silvestre, P. Rosa, J. P. Hespanha, and C. Silvestre, "Finite-time average consensus in a byzantine environment using set-valued observers," in *American Control Conference (ACC), 2014*, pp. 3023–3028, June 2014.

[11] D. Silvestre, P. Rosa, J. P. Hespanha, and C. Silvestre, "Distributed fault detection using relative information in linear multi-agent networks," *IFAC-PapersOnLine*, vol. 48, no. 21, pp. 446–451, 2015. 9th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 20, Paris, 2-4 September 2015.

[12] D. Silvestre, J. P. Hespanha, and C. Silvestre, "Fault detection for cyber-physical systems: Smart grid case," in *23rd International Symposium on Mathematical Theory of Networks and Systems (MTNS), Hong-Kong, China.*, July 2018.

[13] D. Silvestre, P. Rosa, J. P. Hespanha, and C. Silvestre, "Stochastic and deterministic fault detection for randomized gossip algorithms," *Automatica*, vol. 78, pp. 46 – 60, 2017.

[14] D. Silvestre, J. Hespanha, and C. Silvestre, "A pagerank algorithm based on asynchronous gauss-seidel iterations," in *2018 Annual American Control Conference (ACC)*, pp. 484–489, June 2018.

[15] D. Silvestre, J. Hespanha, and C. Silvestre, "Desynchronization for decentralized medium access control based on gauss-seidel iterations,"

[16] H. Hao, D. Silvestre, and C. Silvestre, “Source localization and network topology discovery in infection networks,” in *37th Chinese Control Conference (CCC), Wuhan, China.*, July 2018.