**Faculdade de Engenharia da Universidade do Porto**


# Planeamento Cooperativo de Tarefas e Trajectórias em Múltiplos Robôs

**Pedro Luís Cerqueira Gomes da Costa**

Dissertação no âmbito do Doutoramento em Engenharia Electrotécnica e de Computadores

**Orientador: Prof. Dr. António Paulo Gomes Mendes Moreira**

Março de 2011

Para a Susana,
o Bruno
e a Ana

Se fiz descobertas valiosas, foi mais por ter paciência do que qualquer outro talento.

(Isaac Newton)

# Agradecimentos

Ao meu orientador, Prof. Dr. António Paulo Gomes Mendes Moreira, pela confiança e apoio dados durante a execução do trabalho.

Ao meu irmão Paulo Costa, que na qualidade de chefe da equipa 5DPO, muitas contribuições deu para o desenvolvimento da tese.

À Susana, pela imensa paciência e enorme ajuda.

Aos restantes elementos da equipa 5DPO, Prof. Dr. Armando Sousa e Paulo Marques, que sem eles o projecto nunca teria ido para a frente.

Ao Tiago Nascimento pelo seu apoio na edição de artigos científicos.

À Faculdade de Engenharia da Universidade do Porto pelo apoio técnico disponibilizado.

Aos meus pais e restantes familiares, por toda a compreensão e ajuda.

# Resumo

Hoje em dia o planeamento de trajectórias não se resume ao domínio dos robôs. Muitas novas áreas de aplicação prática, tais como: animação por computador, jogos, mundos virtuais, procedimentos cirúrgicos, informática para fins biomédicos e protótipos virtuais surgiram e fizeram com que o desenvolvimento desta área acelerasse significativamente nos últimos anos. Este desenvolvimento ocorre com avanços significativos em muitas direcções diferentes.

No planeamento de trajectórias existem muitas restrições originadas pelo ambiente em que ocorre a acção, o que torna mais ou menos complexo esse planeamento. Uma dessas restrições é o problema da acção decorrer em tempo real com obstáculos dinâmicos imprevisíveis, em que o tempo de execução do planeamento é um aspecto crucial.

Neste trabalho foi realizado o desenvolvimento e a implementação de um novo algoritmo de planeamento de trajectórias, que funciona num ambiente de tempo real com obstáculos dinâmicos. Para plataforma de teste escolheu-se o futebol robótico devido à sua elevada dificuldade e complexidade. Neste ambiente existem vários robôs para controlar e vários obstáculos a contornar com uma dinâmica muito imprevisível. A abordagem aqui adoptada foi a decomposição em células aproximadas, com a utilização do algoritmo de pesquisa A*. Foram desenvolvidas e implementadas diversas modificações à abordagem escolhida, de modo a melhorar o seu desempenho, sendo a principal preocupação melhorar o tempo de execução das trajectórias e reduzir o número de colisões.

Foi ainda desenvolvido e implementado um algoritmo para suavizar as trajectórias geradas, com a intenção de conseguir dotar as trajectórias de percursos capazes de serem executadas pelos robôs com velocidades elevadas. Foi preocupação constante na construção do algoritmo ter um tempo de execução muito baixo para não afectar o tempo de planeamento de trajectória que é um ponto crítico, em situações de tempo real.

# Abstract

Today, the trajectory planning is not just of the domain of robots. Many new practical application areas such as, computer animation, games, virtual worlds, surgical procedures, biomedical information technology and virtual prototypes, emerged and led the development on this area at a pace that has accelerated in recent years. This development is full of significant advances in multiple directions.

The environment in which the action occurs imposes many restrictions to the trajectory planning problem than can turn it into a very complex problem. Some of those restrictions are the real time pressure from having to obtain a solution while the the action is taking place and the case where some obstacles possess unpredictable dynamics making it impossible to know their location in the future.

This work undertook the development and implementation of a new trajectory planning algorithm that operates in a real-time environment with dynamic obstacles. Robotic soccer was chosen as a test platform due to its high complexity and difficulty. In this environment, there are various robots to control and navigate around various obstacles with very unpredictable dynamics. The approach taken here followed the concepts of cell decomposition and used the search algorithm A *. Several modifications to this approach were developed and implemented, chosen to improve its performance. The main objective being to optimize the execution time for following the chosen paths while also reducing the number of collisions.

An algorithm to smooth the generated trajectories so these trajectories could be performed by robots traveling at high speed was further developed and implemented . There was a continuous concern in the construction of the algorithms to ensure that their execution required a short computational time so that the real time critical restrictions were met during the online path planning.

# Índice

# Lista de Figuras

# Lista de Gráficos

# Lista de Tabelas

# Lista de Abreviaturas

| | |
|---|---|
| 2D | Duas dimensões |
| 3D | Três dimensões |
| 5DPO | Equipa de futebol robótico da Faculdade de Engenharia da Universidade do Porto |
| ACO | *Ant Colony Optimization* |
| AD* | Algoritmo AD* |
| AG | Algoritmos genéticos |
| A* | Algoritmo A* |
| BFS | *Breadth-first search* |
| CPU | *Central processing unit* |
| $C_{espaço}$ | Espaço de configuração |
| $C_{livre}$ | Espaço de configuração livre |
| $C_{obstáculos}$ | Espaço de configuração ocupado pelos obstáculos |
| DC | *Direct current* |
| D* | Algoritmo D* |
| E | Conjunto de ligações do *roadmap* |
| ERRT | *Extended Rapidly-exploration Random Tree* |
| EST | *Expansive-Space Tree* |
| E* | Algoritmo E* |
| FEUP | Faculdade de Engenharia da Universidade do Porto |
| FIFA | *Fédération Internationale de Football Association* |
| IDDFS | *Iterative deepening depth–first search* |
| Mod5 | Modificação das curvas com 5 nós |
| Mod7 | Modificação das curvas com 7 nós |
| ODE | *Open Dynamics Engine* |
| PC | Personal computer |
| PID | Controller *proportional–integral–derivative controller* |
| PRM | *Roadmap* probabilístico |
| PSO | *Particle Swarm Optimization* |
| RRT | *Rapidly-exploration Random Tree* |
| SBL | *Single-Query, Bidirectional and Lazy Collision* |
| SimTwo | Software de Simulador |

SN *Subgoal network*

SSL *Small-size robot league*

UDP *User Datagram Protocol*

VG *Visibility graph*

XML *Extensible Markup Language*

# Capítulo 1

# Introdução

## 1.1 Contexto e Motivação

Quando se fala actualmente em planeamento de trajectórias não se está a referir apenas robôs a desviarem-se de obstáculos, mas sim uma vasta área de outras aplicações, o que fez nos últimos anos, disparar o desenvolvimento de estudos nesta área do conhecimento. Tais áreas são, por exemplo, os jogos de computadores, a animação por computadores, os mundos virtuais, a biologia molecular e os procedimentos cirúrgicos, entre outros.

Os programadores de jogos desde há muito que trabalham esta área. Os trabalhos destes programadores nem sempre foram reconhecidos pela comunidade académica, por serem demasiados práticos, mas mais recentemente a comunidade académica começou a olhar com outros olhos para esta área, que em muitos aspectos está muito avançada, e vários trabalhos de investigação focando a área de jogos foram desenvolvidos como por exemplo Khantanapoka e Fischer, 2009 [1, 2]. Desde jogos em 3D, tipo *personal shooter* Figura 1.1 c) e jogos em plataformas 3D, Figura 1.1 a) até aos jogos de estratégia em tempo real, Figura 1.1 b), o desenvolvimento do planeamento de trajectórias foi uma área em que os programadores investiram muito por forma a torná-los mais realistas e atractivos, devido à exigência imposta pelo público e pela necessidade de gráficos cada vez mais realistas.


Figura 1.1 - a) jogo 3D b) jogo de estratégia em tempo real c) *personal shooter*

A animação por computador, Figura 1.2 a), e o mundo virtual, Figura 1.2 b), utilizam o planeamento de trajectórias para que todo o planeamento de movimentos dos chamados actores, que

podem ser pessoas, veículos, animais ou qualquer outro objecto, ocorra no cenário Junfeng, 2010 e Muzhou, 2009 [3, 4].


Figura 1.2 - a) animação por computador b) realidade virtual

Em áreas como a biologia molecular, o planeamento do movimento também faz parte do seu mundo. No movimento inerente às interacções moleculares, quanto melhor se prever esse movimento com simulações dinâmicas de moléculas melhor se entende por exemplo a sequência genética, o que poderá ser mais um passo para se desenvolverem melhores medicamentos baseados em descobertas genéticas. Cortes, 2005 [5] desenvolveu um trabalho nesta área.

Em procedimentos cirúrgicos hoje em dia existem vários robôs que auxiliam e fazem com que o trabalho de um médico esteja muito mais facilitado. Na Figura 1.3 mostra-se o robot Da Vinci Surgery apresentado por Kypson, 2003 [6] utilizado em diversos tipos de operações.

Figura 1.3- Robô de cirurgia Da Vinci

Estas áreas ajudaram a que o desenvolvimento do planeamento de trajectórias crescesse muito rapidamente nos últimos anos graças à exploração em vários segmentos diferentes do planeamento da trajectória. Assim , os cenários e restrições que existem para o planeamento de

trajectórias são dos mais variados possíveis e tal situação faz com que haja o tal rápido desenvolvimento. As restrições e os cenários possíveis são:

- Robôs holonômicos[1], que são os robôs que conseguem mover-se em qualquer direcção, como por exemplo os robôs omnidireccionais.
- Robôs não-holonômicos[2], que têm restrições em algum dos seus movimentos, como por exemplo os robôs tipo carros, que só conseguem andar em frente ou para trás.
- A existência de múltiplos robôs que cooperam entre si para as mais diversas tarefas.
- A existência de obstáculos parados ou com movimento previsível ou com movimento desconhecido e imprevisível.
- A existência de obstáculos que são deformáveis é mais uma restrição que torna o problema mais complicado.
- As restrições dinâmicas do objecto em movimento.

Estes cenários e restrições entram, hoje em dia, nos problemas estudados na área de planeamento de trajectórias sendo ainda mais complicado quando se pretende pôr esse estudo em prática, numa situação real. Quando se passa para o chamado tempo real, muita da teoria de planeamento deixa de ser possível de executar. O trabalho aqui exposto pretende estudar a aplicação do planeamento de trajectórias numa situação de tempo real bem definida e das mais exigentes: o futebol robótico.

## 1.2 Objectivos e contribuições

O principal objectivo deste trabalho é o desenvolvimento e implementação de um algoritmo de planeamento de trajectórias, num ambiente dinâmico em tempo real, com aspectos inovadores, pretendendo minimizar as colisões e a duração do trajecto com tempos de processamento muito baixos. Procurando-se uma situação difícil e de elevada complexidade, recorreu-se ao problema do futebol robótico. Neste trabalho foi possível implementar e testar as soluções desenvolvidas num ambiente que permite comparar metodologias diferentes. Nesse ambiente existem vários robôs para controlar e vários obstáculos a contornar com dinâmicas imprevisíveis.

Como contribuições principais da tese teremos:

[1] O termo holonômico significa "universal", "integral", "integrável" ( literalmente: holo = o todo, conjunto, totalidade - nomia =

[2] Definem-se como não-holonômicos sistemas com dimensão finita onde algum tipo de restrição é imposta a um ou mais estados do sistema. Estas limitações podem ser provocadas pela conservação do momento angular, condições impostas pela impossibilidade de deslocar em uma ou mais direcções,

- Desenvolvimento e implementação de modificações nos algoritmos e nas abordagens normalmente utilizadas, com o objectivo de melhorar o desempenho dos robôs. Com esta abordagem inovadora conseguiu-se melhorar as trajectórias quer no aspecto do tempo de execução do algoritmo como também e, principalmente, nos número de colisões.
- Desenvolvimento e implementação de um algoritmo para suavizar as trajectórias geradas. Este algoritmo consegue dotar as trajectórias de percursos mais capazes de serem executados pelos robôs com velocidades elevadas. Frequentemente a trajectória gerada não era totalmente seguida pelo robô por causa da sua dinâmica. A introdução deste algoritmo visa fazer com que o robô possa seguir com maior facilidade e precisão as trajectórias geradas.

## 1.3 Estrutura da tese

No capítulo 2 serão apresentadas as ideias base relativas aos métodos existentes para planeamento de trajectórias. Primeiro será abordado o espaço de configuração e por fim os métodos para o planeamento serão apresentados: Algoritmos *Bug*, *Roadmap*, decomposição em células, campo potencial, programação matemática e outros métodos.

O capítulo 3 começa com uma breve apresentação de vários algoritmos de pesquisa e o porquê da escolha do A* como o algoritmo a utilizar. Em seguida é aprofundado o método utilizado para o planeamento de trajectórias e mostrada a sua implementação. O método utilizado foi a decomposição por células fixas, com o algoritmo de pesquisa A*.

No capítulo 4 será apresentada a plataforma de teste utilizada, o porquê da sua utilização e a descrição dos robôs tanto ao nível de hardware como de software.

O capítulo 5 apresenta o simulador utilizado e serão apresentadas as razões de se utilizar um simulador em vez dos robôs reais. Serão ainda identificados os parâmetros que validam o simulador, de modo a que se possa usar o simulador com a garantia que o robô real se comporta de uma forma muito similar.

No capítulo 6 vão ser descritas as alterações propostas ao algoritmo A*. Essas alterações visam dar ao $C_{espaço}$ alguma da dinâmica dos obstáculos, diminuir o tempo de processamento, controlar colisões e controlar a direcção de chegada. Serão também apresentados os resultados dos testes efectuados, tanto em ambiente de simulação como com robôs reais, de modo a comprovar as melhorias registadas.

No capítulo 7 vai ser descrita a suavização efectuada nas trajectórias. Primeiro será explicado o porquê dessa suavização e quais serão as suas consequências. Em seguida será

demonstrado o processo a efectuar e serão apresentados os resultados, tanto em ambiente de simulação como com robôs reais, que comprovam as melhorias conseguidas.

Por fim, no capítulo 8 serão apresentados as conclusões e o trabalho futuro.

# Capítulo 2

# Planeamento de trajectórias

Neste capítulo serão apresentados os métodos mais utilizados para planear as trajectórias e quais as suas aplicações. Inicialmente alguns conceitos serão explicados e será abordado o espaço de configuração. Por fim serão abordados os métodos para planear as trajectórias.

## 2.1 Conceitos

Existem três conceitos importantes a reter desde já:

- planeamento do caminho (*path planning*)
- planeamento de trajectórias (*trajectory planning*)
- planeamento de movimentos (*motion planning*)

O planeamento do caminho descreve em termos geométricos ou matemáticos qual o caminho desde o ponto inicial até ao ponto de destino, evitando colidir com obstáculos.

O planeamento de trajectórias representa esse caminho em função do tempo, ou seja, diz para cada instante de tempo o local onde deve estar posicionado o robô.

O planeamento dos movimentos entra em conta com as restrições cinemáticas e dinâmicas do robô.

Estes três conceitos são muitas vezes confundidos na literatura. Também neste trabalho foi utilizado a expressão planeamento de trajectórias, embora o termo correcto seja planeamento do caminho. A decisão de utilizar a expressão de planeamento de trajectórias advém do facto desta expressão ser mais facilmente reconhecida no português corrente.

### 2.1.1 Factores para escolha de um método

Quando se pretende escolher qual o método a utilizar para o planeamento de uma trajectória, vários aspectos têm de ser considerados, tais como:

- O tipo de optimização pretendida, se se pretende optimizar o comprimento da trajectória, o tempo que demora a executar a trajectória (tempo de execução da trajectória), a energia consumida ou outra variável qualquer.
- A complexidade computacional é outro aspecto a ter em conta, uma vez que muitos dos métodos demonstrados teoricamente acabam por não serem possíveis de implementar

ou porque não existe memória suficiente ou porque o tempo de execução do algoritmo é muito elevado.

- Se o método é completo, completo em resolução ou probabilisticamente completo.
  - O método diz-se completo se encontra sempre uma solução quando ela existe e indica que não há solução quando esta não existe.
  - O método diz-se completo em resolução se existe uma solução para uma determinada discretização do ambiente.
  - O método diz-se probabilisticamente completo se a probabilidade de encontrar uma solução converge para 1 à medida que o tempo tende para o infinito.
- Se o método é *offline* ou *online*. Um método diz-se *offline* se constrói a solução baseado no modelo do ambiente, ou seja, antes ser iniciada a trajectória. Se for *online* o método constrói a solução à medida que o robô o executa.
- Se existe um robô ou múltiplos robôs, isto é, se se pretende encontrar uma trajectória ou várias trajectórias.
- Se os obstáculos estão parados, em movimento previsível ou em movimento desconhecido. A maior parte dos métodos funciona para obstáculos parados, mas quando os obstáculos estão em movimento não conseguem funcionar principalmente por causa do tempo de execução ser elevado.
- Se o robô é holonômico ou não-holonômico, isto é, se pode mover-se em qualquer direcção, como por exemplo os robôs omnidireccionais, ou se tem restrições nas direcções de movimento, como por exemplo um carro.
- Se as restrições dinâmicas do robô entram no planeamento, ou não (por exemplo a aceleração e velocidade máxima).
- Se obstáculos são deformáveis ou não.

Estes aspectos devem entrar em consideração para a escolha do método a utilizar, por isso é importante sempre que possível simplificar o modelo de modo a ser mais fácil o método tratar o problema.

Nesta dissertação o método utilizado é completo em resolução, sendo o objectivo principal o tempo de execução da trajectória. O método é online, os obstáculos estão em movimento desconhecido, não são deformáveis e o robô é omnidireccional.

## 2.2 Espaço de Configuração

Um dos problemas ao planear uma trajectória e os movimentos do robô é que é preciso um mapa do ambiente onde está inserido esse robô. Esse mapa serve para uniformizar os ambientes para os modelos.

A configuração de um robô é um conjunto de parâmetros que especificam a sua posição no sistema. O espaço de configuração de um sistema com um robô móvel é o espaço de todas as configurações possíveis do sistema e denomina-se por $C_{espaço}$. A sua dimensão é igual ao número de parâmetros que define a configuração e chama-se graus de liberdade. Denomina-se espaço livre a parte do $C_{espaço}$ que não é ocupada por obstáculos e representa-se por $C_{livre}$. Este conceito foi usado pela primeira vez por Lozano-Perez e Wesley em 1979 [7], e tornou-se muito importante na resolução do problema de planeamento de trajectórias.

A posição e a orientação de um robô num plano no espaço físico pode ser especificado por três parâmetros $(x,y,\theta)$ no $C_{espaço}$ com rotação e dois parâmetros $(x,y)$ sem rotação. Essa transformação desde o espaço físico para o $C_{espaço}$ faz com que o robô passe a ser tratado por um ponto. Isto é, no $C_{espaço}$ o robô diminui-se para um ponto e aumentam-se os obstáculos com o raio do robô. O problema do planeamento da trajectória no espaço físico passa a ser encontrar um caminho para o ponto que representa o robô em que todos os pontos de caminho têm de ser livres.

### 2.2.1 Métodos para calcular o $C_{obstáculos}$

Os $C_{osbtáculos}$ podem ser calculados de diversos métodos, como indicado em Ahuja,1992[8]. Em seguida serão apresentados sete dos métodos mais comuns:

#### 2.2.1.1 Point of evalution (Ponto de evolução)

Para cada configuração possível do robô é determinado se esta intercepta o obstáculo e assim determina-se se pertence ao $C_{obstáculos}$. Este método é o mais simples mas o mais ineficiente.

#### 2.2.1.2 Minkowski set difference

Este método foi descrito pela primeira vez por Lozano-Pérez em 1983 [9], e considera $Mdiff(A,B) = \{a-b \mid a \in A, b \in B\}$ em que A e B são dois conjuntos de pontos. Considerando o robô um objecto rígido sem rotação, o $C_{osbtáculo}$ é a união do *Minkowski set difference* entre as áreas ocupadas pelos obstáculos e o robô. Por exemplo na Figura 2.1, o ponto de referência do objecto A não pode ser colocado na região sombreada. A região cinzenta clara é o Mdiff(A,B) a cinzenta mais escura é o obstáculo. Se o ponto de referência do objecto A estiver dentro dessa área então existe intersecção entre eles.

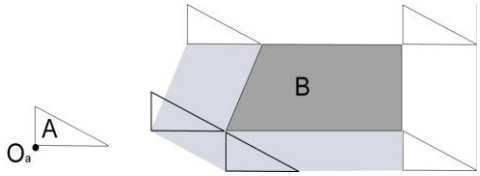



Figura 2.1 - Exemplo de uma Mdiff(A,B)

#### 2.2.1.3 Boundary equations (Equações limite)

Neste método encontram-se as equações matemáticas que definem quando o objecto toca nos obstáculos. Para uma representação poliédrica essas equações de restrição são derivadas a partir do vértice e das bordas de contactos entre o robô e os obstáculos. Representar os $C_{obstáculos}$ com equações de fronteiras pode ser muito complexo, normalmente estes casos servem para testar se uma configuração particular está em $C_{livre}$. McCarthy em 1989 [10] descreveu o método para obter a equação de fronteira de um manipulador plano e Hwang em 1990 [11] descreveu o método para um manipulador em 3D.

#### 2.2.1.4 Sweep volume method (Método de varrimento do volume)

Este método é usado principalmente para os manipuladores, situação em que se constrói uma área em que o robô pode mover-se sem colisão, ou seja $C_{livre}$. Essa área é construída fixando um dos parâmetros da configuração do robô e guardando os valores dos outros parâmetros em que o robô está em $C_{livre}$. Este processo é repetido para todos os parâmetros. Este método torna-se computacionalmente complexo quando os graus de liberdade são maiores que 3 como indicou Ahuja, em 1992 [8]. A estratégia de busca sequencial é uma variação deste método e Guo em 1992 [12] demonstrou essa estratégia.

#### 2.2.1.5 Needle method (Método da agulha)

Este método cria os $C_{obstáculos}$ fixando todos os parâmetros excepto um dos parâmetros de configuração e usa as equações de fronteira para detectar as colisões. Como resultado guardam-se os intervalos do parâmetro variável em que resultou em colisão. O $C_{obstáculo}$ é representado como um conjunto de intervalos discretos. Foi implementado por Lozano-Pérez em 1983 [9, 13].

#### 2.2.1.6 Jacobian-based method(Método baseado na matriz Jacobiana)

Este é um método para calcular blocos de $C_{livre}$ ou de $C_{obstáculos}$. O Jacobiano J de um robô é a matriz que relaciona o deslocamento dx de um ponto do robô com a mudança na configuração do robô dq. Isto é, dx=J(q).dq. Para o robô na configuração q, o máximo de |J(q)| é B(q). Se a mínima distância entre o robô em q e todos os obstáculos é D , então a esfera centrada em q com raio D/B(q) é $C_{livre}$. Se definir uma distância mínima $D_o$ entre dois objectos que se sobreponham para se

separarem, então pode-se calcular uma esfera com centro em q e raio $D_o/B(q)$ que é um $C_{obstáculo}$. Este método foi utilizado por Branicky em 1990 e por Connolly em 1990 [14, 15] .

#### 2.2.1.7 Templates (Modelos)

Os obstáculos podem ser decompostos em formas mais simples, tais como pontos e linhas, enunciado por Branicky em 1990 [14]. Essas formas mais simples são parametrizadas e a união dessas formas geram o $C_{obstáculos}$. Este método funciona bem para graus de liberdade menores que 4.

## 2.3 Métodos para planeamento de trajectórias

Muitos métodos foram desenvolvidos para o planeamento de trajectórias, sendo que alguns são aplicáveis a vários tipos de problemas de planeamento de trajectórias mas outros são muito difíceis de aplicar. Os métodos podem ser divididos em seis categorias:

- Algoritmos *Bug*,
- *Roadmap*,
- Decomposição em células,
- Campo potencial,
- Programação matemática
- e outros métodos.

Estas categorias não são mutuamente exclusivas, ou seja, existem combinações entre elas de modo a tirar melhor partido das vantagens que cada uma tem. Em seguida será dada uma breve explicação de cada uma das categorias e as suas aplicações mais recentes, mantendo sempre o foco no planeamento de trajectória em robôs reais em particular no nosso ambiente de teste, o futebol robótico.

### 2.3.1 Algoritmo *Bug*

Os algoritmos *Bug* surgiram para resolver os casos em que o ambiente é desconhecido, isto é, o robô tem de chegar ao destino e não tem conhecimento nenhum do ambiente global e utiliza os sensores para ir descobrindo o que encontra. Nos algoritmos *bug* não se constroem mapas. Este algoritmo assume que o robô é um ponto e que sabe sempre a sua localização em relação ao ponto de origem.

Em seguida serão apresentados alguns dos algoritmos mais conhecidos e algumas aplicações em robôs reais, começando pelos primeiros algoritmos desenvolvidos nesta categoria, passando depois para algoritmos mais recentes e que têm implementação em casos práticos. James Ng em 2010 [16] compilou quase todos os algoritmos desta categoria.

#### 2.3.1.1 Bug 1

O algoritmo *Bug* 1 descrito pela primeira vez por Stepanov em 1987 [17] foi o primeiro algoritmo desta categoria a ser desenvolvido. A algoritmo resume-se aos seguintes passos:

- Ir em direcção ao destino
- Se um obstáculo for encontrado dá a volta ao obstáculo e guarda o local em que fica mais perto do destino
- No fim da volta deve ir para ao ponto mais perto e continua em direcção ao destino

Na Figura 2.2 está um exemplo da implementação do algoritmo, e como se pode constatar não é muito eficaz. Ter que dar sempre uma volta ao obstáculo para decidir aonde deve sair não é a melhor solução, mas deve-se realçar o mérito de ter sido o primeiro desta categoria a ser desenvolvido.

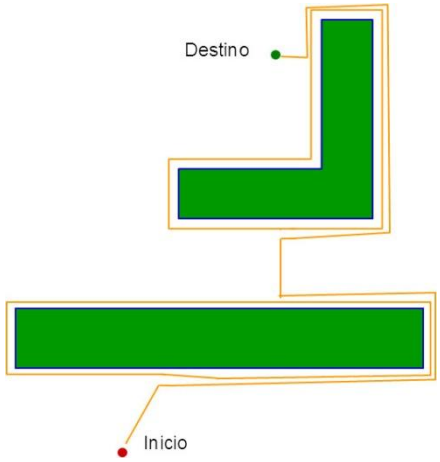



Figura 2.2 - Exemplo de *Bug* 1

#### 2.3.1.2 Bug 2

No algoritmo *Bug* 2 de Stepanov, 1987[17], que é um melhoramento do *Bug 1*, os passos implementados são os seguintes:

- Ir em direcção ao destino no segmento de recta que une o ponto inicial e o ponto de destino.
- Se um obstáculo for encontrado dar a volta ao obstáculo até voltar a encontrar o segmento de recta
- Continuar pela recta em direcção ao destino

Na Figura 2.3 está um exemplo da implementação do algoritmo, e como se pode constatar já se comporta melhor que o *Bug 1*, dado que já não precisa dar a volta toda ao obstáculo.

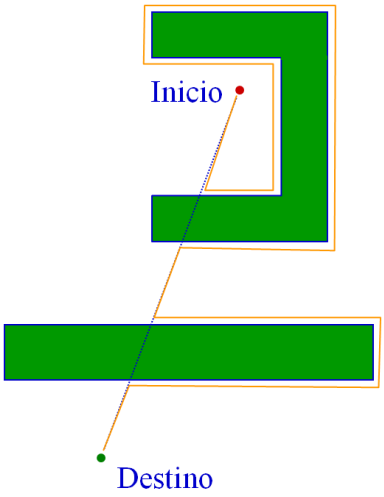



Figura 2.3 - Exemplo do *bug* 2

#### 2.3.1.3 Tangent Bug (Besouro tangente)

O algoritmo *Tangent Bug* de Kamon, de 1998 [18] utiliza sensores de distância para construir um grafo com os vértices dos obstáculos alcançáveis pelos sensores em cada momento, escolhendo qual o vértice que deve de ir, como mostra a Figura 2.4. Este algoritmo foi implementado num robô real por Kamon,em 1998 [18].

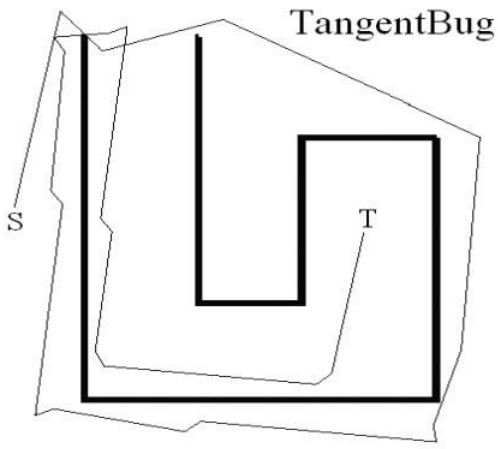



Figura 2.4 – Exemplo do Algoritmo *Tangent Bug*

#### 2.3.1.4 DistBug

O algoritmo *DistBug* de Rivlin, de 1997 [19] também utiliza sensores de distância mas a sua utilização é para saber quando é que pode abandonar o contorno do obstáculo e ir direito para o destino. Na Figura 2.5 constata-se que o contorno do obstáculo é abandonado mal se possa andar um bocado em direcção ao destino. Este algoritmo também foi implementado com sucesso num robô real por Rivlin, em 1997 [19].

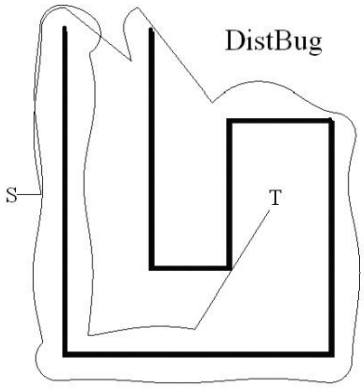



Figura 2.5 - Algoritmo *Distbug*

#### 2.3.1.5 Bug 2+

Este algoritmo é um melhoramento do algoritmo *Bug 2* apresentado por Antich,em 2009 [20] nos casos em que só passa para a recta se não tiver passado por pontos da recta mais próximos do destino, como se pode constatar na Figura 2.6. No *Bug 2* a primeira vez que encontrou a recta, que une o inicio ao destino, tentou ir directo para o destino, só que voltou a um local que já tinha passado. No *Bug 2+*, não se passou para recta porque já sabia não ia ter sucesso.

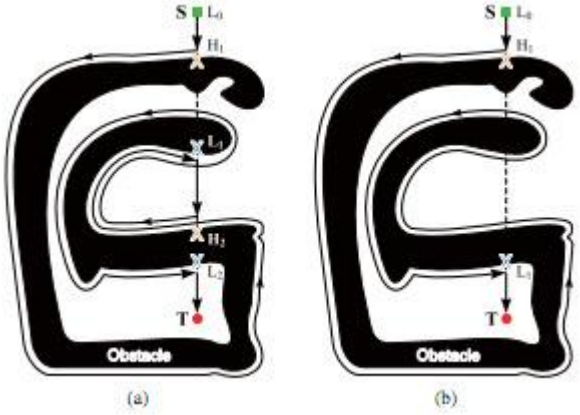



Figura 2.6 - a) Bug2 b) Bug2+

### 2.3.2 Roadmap

Nesta abordagem o $C_{espaço}$ é reduzido para uma rede de dimensão 1 que pertence ao $C_{livre}$, isto é, o *roadmap* está embebido no espaço livre do $C_{espaço}$ como indicado em Canny, 1988 e Latombe, 1991[21, 22]. No algoritmo *roadmap* existem nós e ligações entre nós que podem ter um significado físico, ou seja, os nós podem significar uma localização e as ligações correspondem ao caminho entre essas localizações.

Assim nesta abordagem o planeamento de trajectórias resume-se a um problema de pesquisa em grafo. Essa pesquisa pode ser executada pelas técnicas usuais de pesquisa em grafo. No

capítulo 3 serão abordadas algumas dessas técnicas. O problema central desta abordagem é a construção do *roadmap*. Em seguida serão apresentados algumas das técnicas usadas para a construção do *roadmap*.

#### 2.3.2.1 Visibility graph ( VG)

É normalmente usado um $C_{espaço}$ a duas dimensões (2D) em que os nós são os vértices dos obstáculos e as ligações entre nós só existem se os vértices estão visíveis um para o outro, isto é, existe uma recta que une os vértices e que não passa por nenhum obstáculo como indicado em Wesley, 1979 e Latombe, 1991 [7, 22]. O ponto inicial (ponto referência do robô) e o ponto de destino são também nós e têm de estar ligados a pelo menos um nó para existir um caminho, como se pode ver no exemplo da Figura 2.7. Existem $O(n^2)$ ligações que podem ser construídas em $O(n^2)$ tempo em que n é o número de nós, Asano, 1985 [23]. Um dos problemas desta técnica é que o caminho gerado passa nos limites do espaço livre, como se verifica na Figura 2.7, o que faz com que com pequenos erros no controlo do robô possam existir muitas colisões.

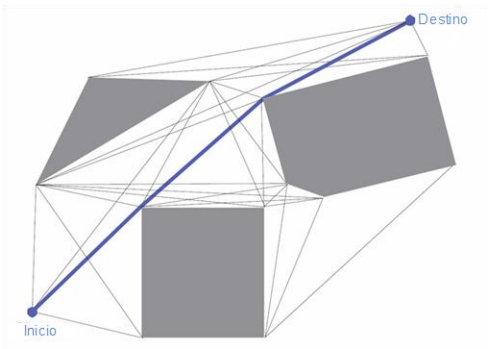



Figura 2.7 - Exemplo de uma trajectória gerada com um mapa em VG

#### 2.3.2.2 Diagrama Voronoi (DV)

O diagrama Voronoi de Aurenhammer, 1991 [24] é o conjunto de pontos que estão equidistantes de dois ou mais obstáculos. O espaço é dividido em regiões e em cada região só existe um obstáculo. Qualquer ponto na região está mais perto do obstáculo dessa região do que de qualquer outro obstáculo. Ao contrário do VG, qualquer caminho gerado pelo DV está muito afastado dos obstáculos. O diagrama é composto por curvas quando as arestas dos polígonos são usadas para calcular a equidistância. Um DV diferente foi desenvolvido por Canny, 1985 [25] que só contém linhas rectas e utiliza uma medida de distância que não é a distância euclidiana.

Para encontrar o caminho no DV deve-se ligar o ponto inicial até ao diagrama de Voronoi, depois encontrar dentro do diagrama o melhor caminho e por fim ligar o diagrama até o destino, como mostra a Figura 2.8.

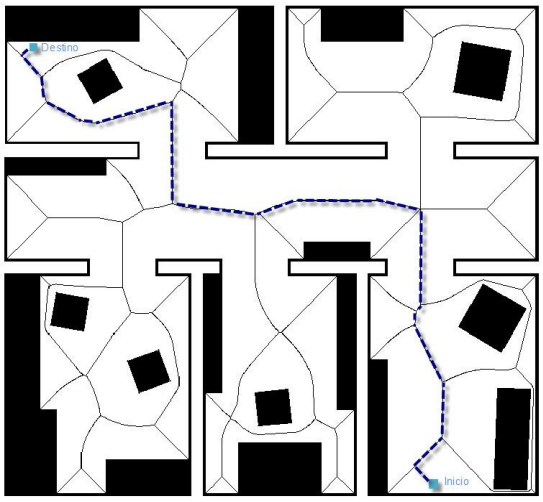



Figura 2.8 - Exemplo de um caminho gerado num mapa DV

O diagrama de Voronoi para n obstáculos pode ter O(n) ligações e pode ser construído em O(n) tempo  Preparata, 1985 [26]. Para o cálculo do DV vários trabalhos desenvolvidos apresentam diversas soluções como são os casos de Preparata, 1985, Ahuja, 1983, Kirkpatrick, 1979, Drysdale,1981 e Scilling, 1989 [26-30].

#### 2.3.2.3 Método da silhueta

Este método foi desenvolvido por Canny, 1988[31] para objectos em espaços dimensionais elevados. O método consiste em projectar os objectos numa dimensão inferior e traçar as curvas da fronteira da projecção, daí o nome de silhueta. A silhueta é projectada recursivamente até reduzir a dimensão a um plano 2D. Este método tende a gerar, tal como o VG, caminhos ao longo dos limites dos obstáculos. Uma variação deste algoritmo foi implementado por  Lin, 1993 [32]. A Figura 2.9 mostra um pequeno exemplo de um mapa em silhueta de um objecto com um buraco no meio.

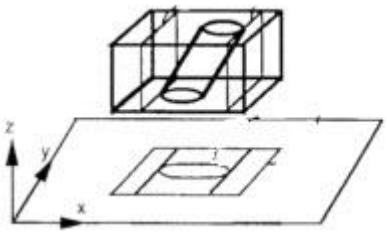



Figura 2.9 - Silhueta de um objecto com um buraco no meio

#### 2.3.2.4 Subgoal network (SN)

Este método foi desenvolvido por Faverjon, 1987 [33] e aperfeiçoado por Hwang, 1992 [34] e não constrói o $C_{obstáculos}$ , em vez disso constrói uma lista de configurações alcançáveis desde a configuração inicial. Quando a configuração destino é encontrada o planeamento de trajectória

fica resolvido. A acessibilidade de uma configuração vinda de outra configuração é decidida por um algoritmo chamado operador local. Inicialmente o operador local é usado para testar se o destino é atingível ou não. Se o destino não for atingível uma lista de candidatos intermédios são gerados, através de uma heurística, que são chamados *subgoals*. O operador local é usado para determinar qual dos *subgoals* é atingível e guarda essa informação na lista. Este processo é repetido até o destino ser alcançado. A Figura 2.10 mostra o método, com o operador local a mover-se na diagonal. A eficácia deste método depende do operador local utilizado e do gerador de s*ubgoals*. O VG é um método *subgoal* em que o operador usado é sempre em frente e os *subgoals* são os vértices dos obstáculos. Noutro exemplo, em Warren, 1991[35] foi apresentado como gerador de *subgoals* um vector desde o centro do obstáculo até à fronteira do obstáculo.

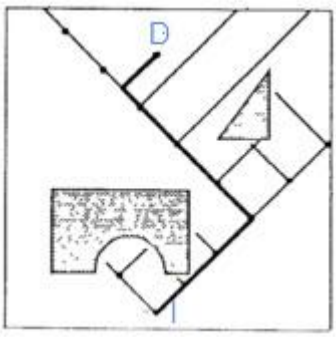



Figura 2.10 - Exemplo de SN com operador local a mover-se na diagonal

#### 2.3.2.5 Roadmap Probabilístico (PRM)

O *Roadmap* probabilístico foi inicialmente descrito por Kavraki, 1996 [36] e é resultado do trabalho independente de vários grupos: Kavraki, 1995, Kavraki, 1998, Overmars, 1992, Svestka, 1995, Svestka, 1993, Kavraki, 1995, Kavraki, 1996 e Latombe, 1998 [37-44]. O PRM baseia-se na construção de um *roadmap* de uma forma probabilística, que pode ser descrita da seguinte forma:

O PRM é dividido em duas fases:

- A primeira fase corresponde à construção do *roadmap*, designada por fase de aprendizagem ( *Learning phase*)
- Na segunda fase procura-se o caminho desde o início até ao destino dentro do *roadmap* criado e designa-se por fase de investigação (*Query Phase*).

A fase de aprendizagem, por sua vez pode ser dividida em 3 partes:

A primeira parte é a da construção do *roadmap* que se pode resumir nas seguintes fases:

- Começar com um *roadmap* vazio R.
- Gerar uma configuração aleatória pertencente a $C_{livre}$, que é adicionada como nó no *roadmap*, sendo N o conjunto de nós do *roadmap*.

- Para esse nó seleccionar um conjunto de nós vizinhos (nós vizinhos são aqueles que estão até uma distância máxima pré-definida).
- Testar para cada nó vizinho se existe uma ligação usando um planeamento local
- Se sim acrescentar essa ligação ao *roadmap*, sendo E o conjunto de ligações do *roadmap*.
- Repetir estes passos enquanto for necessário, ou seja, até se considerar que o *roadmap* está suficientemente preenchido para se possa gerar um caminho.

Na Figura 2.11, o círculo a tracejado é o limite dos vizinhos, o novo nó vai ser ligado a todos os nós gerados que estão dentro do círculo à distância máxima.

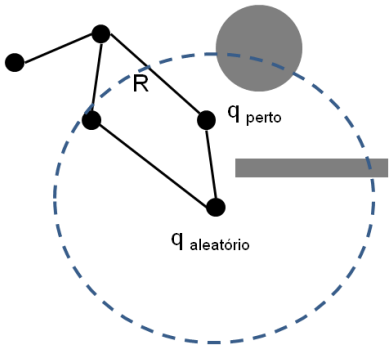



Figura 2.11 - Exemplo das ligações de um novo nó

- A segunda parte é a da expansão que serve para melhorar o *roadmap*, principalmente nas regiões consideradas mais difíceis. Essas regiões são as que têm normalmente poucos espaços livres, o que faz que normalmente tenham poucos nós e como consequência poucas ligações. Nesta parte existem as seguintes fases:
  - Encontrar os nós que estão em regiões difíceis, usando uma função heurística, normalmente designada por W(n). Existem várias opções para a função heurística, as mais utilizadas são as seguintes:
    - Inversamente proporcional ao número de nós dentro de uma distância pré-definida até n.
    - Inversamente proporcional à distância desde do nó até ao nó vizinho mais perto que não esteja ligado a ele.
  - Expandir usando *random-bounce walks* ou voltando a repetir a construção do *roadmap* mas só para as regiões chamadas difíceis. O *random-bounce walks* consiste nas seguintes operações para um determinado nó:

- Escolher uma direcção aleatória de movimento em $C_{espaço}$
- Movimentar-se nessa direcção até atingir um obstáculo
- Escolher uma nova direcção aleatória, quando se atinge um obstáculo
- Repetir até que o caminho possa ser ligado a outro nó
- Armazenar o caminho gerado

- A terceira parte é opcional e visa simplificar o *roadmap*, ou seja, serve para retirar os nós que são redundantes de modo que a fase de encontrar o caminho seja mais rápida.

A fase de investigação tem os seguintes passos:

- Dado o ponto inicial encontrar um caminho livre até um dos nós do *roadmap*.
- Dado o ponto de destino encontrar um caminho livre até um dos nós do *roadmap*.
- Encontrar um caminho desde o nó que liga o ponto de início até ao nó que liga ao ponto de destino.

Estes métodos demonstram bons resultados empíricos, isto é, funcionam razoavelmente bem em termos práticos. No entanto estes métodos não são completos, são chamados de probabilisticamente completos, isto é, têm uma probabilidade a tender para 1 de descobrir a trajectória quando o tempo despendido a procurar a solução tende para infinito. Estes métodos têm ainda a desvantagem de, em casos de não existir trajectória possível, o método pode nunca terminar. Na Figura 2.12 pode-se verificar que normalmente os nós ficam distribuídos ao longo de todo o espaço. Uma estimativa é dada por Kavraski, 1996 [36] para a probabilidade de encontrar uma trajectória assumindo que existe trajectória.

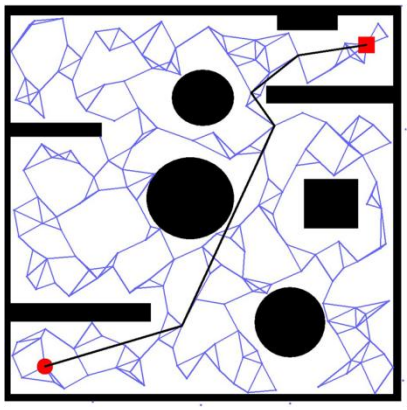


Figura 2.12 - Exemplo de utilizar um PRM

Outro dos problemas ao utilizar o PRM é quando existem passagens muito estreitas em que a probabilidade de existirem nós nessas regiões é quase nula o que faz com que seja difícil, mesmo utilizando a técnica *random-bounce walks*, de encontrar a passagem. Para tentar aliviar o problema várias variantes e extensões do método foram propostas, tais como aumentar a probabilidade de existirem nós à volta dos obstáculos, Amato, 1998 [45] , Boor, 1999 [46] e hsu, 2005 [47], ou nos pontos médios do $C_{livre}$ Wilmarth, 1999 [48], ou com a dilatação do espaço livre, Hsu, 2006 [49] e Saha, 2005 [50] ou com filtro de visibilidade desenvolvido, Siméon, 2000 [51], ou guardando informação quando se está a construir o *roadmap* de modo a reduzi-lo e povoar as regiões mais complicadas Burns, 2005 [52] ou o *Expansive-Space Tree* (EST) de Hsu, 1997 [53] que consiste em decompor o espaço de modo a poder garantir que as passagens estreitas possam ter nós.

Várias variantes foram também desenvolvidas com o objectivo de se minimizar o tempo de execução, como por exemplo o algoritmo *Lazy* PRM de Bohlin, 2000 [54] que tenta minimizar o número de verificações de colisões ou atrasar a verificação até ao ponto em que seja absolutamente necessário e também por Sanchez, em 2002 [55] que utiliza a mesma técnica tendo sido utilizado em ambientes dinâmicos, ver [56].

O *Rapidly-exploration Random Tree* (RRT) de LaValle, 2000 [57], consiste não em guardar o nó gerado aleatoriamente mas em guardar o nó que fica na direcção do nó aleatório com o nó mais perto, mas com uma distância pré-definida (incremento), como se pode constatar na Figura 2.13.

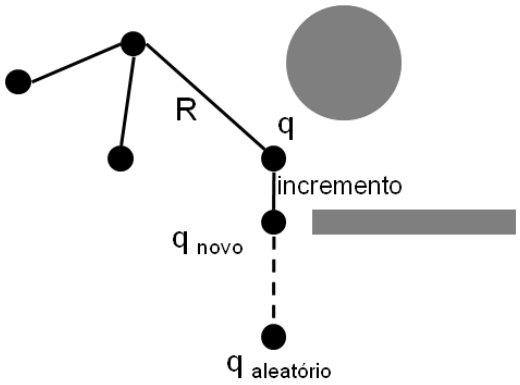



Figura 2.13 - Exemplo da expansão no RRT

A Figura 2.14 mostra como fica *roadmap* utilizando um RRT e verifica-se que os nós não ficam tão dispersos e principalmente não existem caminhos redundantes, o que faz com que a pesquisa do caminho seja executada muito mais rapidamente.

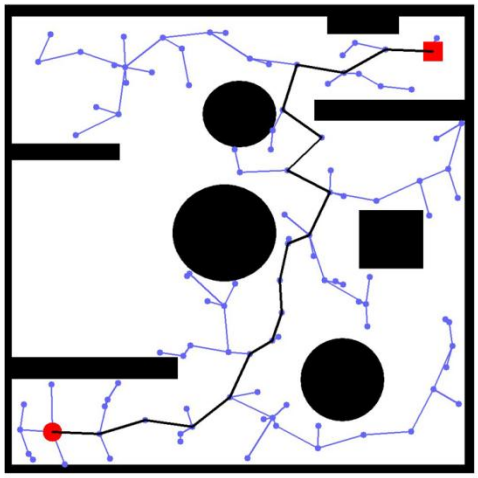


Figura 2.14 - Exemplo de um caminho usando RRT

Este algoritmo foi implementado e melhorado num robô numa ambiente desconhecido por Liu , 2008 [58] e num carro num ambiente urbano, Kuwata, 2009 [59].

O ERRT (*Extended* RRT) deriva do RRT e foi introduzido 2002 no futebol robótico por Bruce, 2002 [60], para situações em que temos um ambiente com obstáculos em movimento. Para utilizar o RRT nestes casos é preciso estar constantemente a calcular novos caminhos para cada instante, visto que o $C_{espaço}$ utilizado no RRT não entra com o parâmetro tempo, logo é um ambiente estático. O ERRT parte do princípio que os novos caminhos não serão muito diferentes que o anterior, logo guarda a informação do caminho anterior e utiliza essa informação, quando está a construir o novo *roadmap*. Na construção do *roadmap* os novos nós a introduzir no *roadmap* têm um probabilidade de mais alta de serem nós que fizeram parte do caminho anterior. Existe uma

probabilidade alta de se poder fazer o novo caminho muito parecido com o anterior, o que vem confirmar que a cada instante a variação dos caminhos é mínima.

O DRRT (*Dynamic* RRT) também utilizado no futebol robótico [61], parte do pressuposto igual ao ERRT, que em instantes sucessivos os caminhos não serão muito diferentes. Neste algoritmo o *roadmap* mantém-se constante em cada instante. O que se vai verificar é, como os obstáculos se moveram, se existem nós e ligações nos novos locais com obstáculos e eliminá-los do *roadmap*. Em seguida reconstrói-se (expansão de nós) à volta da região onde existiu essa eliminação.

O *Single-Query, Bidirectional and Lazy Collision* (SBL) de Sanchez, 2002 [55], consiste em construir duas redes a partir do ponto inicial e a partir do ponto destino, focando no espaço livre acessível a partir desses pontos, quando existe um grande espaço livre dá incrementos maiores e quando existem áreas estreitas os incrementos são mais pequenos, por fim utiliza o *Lazy* PRM para detectar as colisões e diminuir o tempo de execução. Este algoritmo foi implementado num robô com uma configuração similar a um automóvel por Balakirsky, 2010 [62].

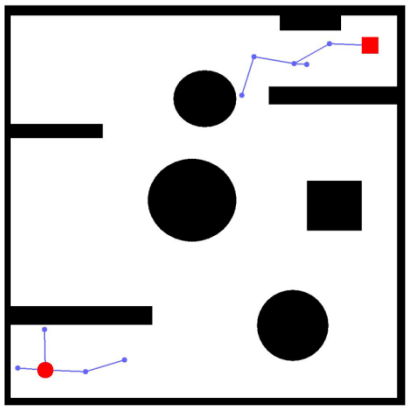


Figura 2.15 - Exemplo da construção de duas redes: uma a partir do ponto inicial e outra a partir do ponto destino

### 2.3.3 Decomposição em células

Nesta abordagem o $C_{espaço}$ é dividido num conjunto de células, efectuando-se o cálculo se a célula está livre ou não e a sua ligação às células vizinhas, criando-se um grafo. Tal como no *roadmap* esta solução resume-se a pesquisar um grafo. O caminho é encontrado pesquisando o grafo e utilizam-se algoritmos de pesquisa em grafos (no capítulo 3 alguns desses algoritmos serão apresentados). Os passos a percorrer são: primeiro aloca-se o ponto inicial a uma célula e o ponto de destino a outra célula e procura-se a sequência de células de modo a chegar de uma à outra. Este método pode ser dividido em:

- Decomposição em células exactas
- Decomposição em células aproximadas

#### 2.3.3.1 Decomposição em células exactas

Na decomposição em células exactas o $C_{espaço}$ vai representar exactamente o mundo real, isto é, as células representam ou espaços livres ou espaços ocupados. Os métodos de decomposição têm de criar células de geometria simples para que facilmente se possa calcular o caminho de duas configurações de uma célula, de modo a que seja simples encontrar as células vizinhas e o caminho. Duas das decomposições mais usadas são: polígonos convexos e trapézios.

##### 2.3.3.1.1 Polígonos convexos

Nesta decomposição as células são polígonos convexos em que seus vértices são os vértices dos obstáculos, tal como se verifica na Figura 2.16. As ligações representam as células adjacentes e os pontos médios dos limites das células são os pontos a partir de onde se constroem os caminhos.

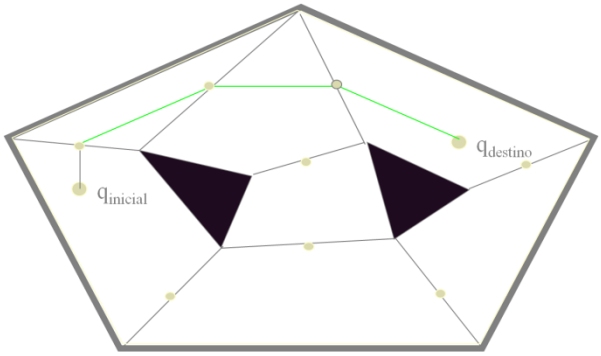



Figura 2.16 - Exemplo de uma decomposição por polígono

##### 2.3.3.1.2 Trapézios

Na decomposição por trapézios cada vértice dos obstáculos contém uma linha na vertical, essas linhas é que vão ser as arestas das células. A Figura 2.17 mostra uma decomposição para 2 obstáculos.

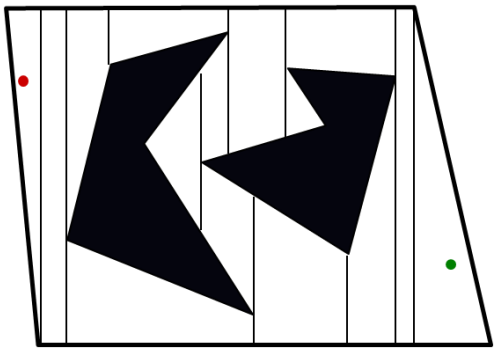


Figura 2.17 - Exemplo de uma decomposição por trapézios

#### 2.3.3.2 Decomposição em células aproximadas

Nesta decomposição o espaço não representa na exactidão o espaço real, porque as células podem neste caso ter três estados: livres, ocupadas ou parcialmente ocupadas. As células são normalmente as figuras geométricas mais simples, como o quadrado, o que faz com que seja simples e rápido de construir o $C_{espaço}$. Um dos problemas gerado pode ser o caminho não ser encontrado mesmo quando exista. Por isso o tamanho das células influência o resultado.

Várias formas de decomposição são encontradas, sendo as mais usadas a célula fixa e *quadtree* para os espaços em 2D.

##### 2.3.3.2.1 Celula fixa

Com a decomposição em célula fixa as células têm sempre um tamanho pré-definido e divide-se o $C_{espaço}$ em quadrados desse tamanho como mostra a Figura 2.18. Este método será o usado na abordagem apresentada nesta dissertação, no capítulo seguinte será apresentada essa decomposição em detalhe.

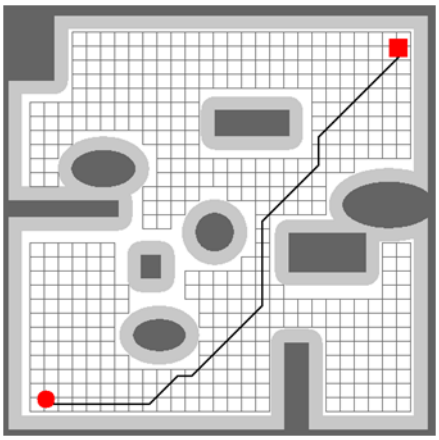


Figura 2.18 - Exemplo de uma decomposição com célula fixa

#### 2.3.3.2.2 Decomposição em Quadtree

Nesta decomposição começa-se por dividir o $C_{espaço}$ em 4 células iguais e sempre que uma célula não pertencer ao $C_{livre}$, volta-se a dividir em 4. Repete-se recursivamente até um limite mínimo do tamanho das células pré-definido. Na Figura 2.19 pode-se visualizar essa divisão. Com esta decomposição o $C_{espaço}$ à volta dos obstáculos fica mais definido. No entanto esta solução é mais complicada de implementar computacionalmente. De acrescentar que esta solução tem como vantagem principal baixar o número de células o que pode provocar baixar o tempo de execução do algoritmo de pesquisa.

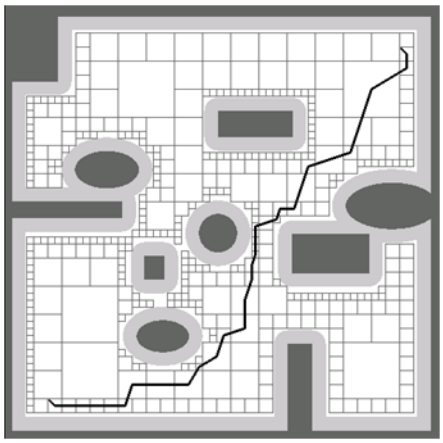


Figura 2.19 - Exemplo de uma decomposição Quadtree

### 2.3.4 Campos de potencial

Em 1984, os campos de potencial foram pela primeira vez investigados em dois projectos distintos de Arimoto, 1984 e Pavlov, 1984[63, 64] . No ano seguinte os campos potenciais foram usados para controlar um manipulador e no planeamento de trajectórias de um robô por Khatib,

1986 [65] e ao longo dos anos seguintes vários trabalhos foram desenvolvidos usando campos de potencial.

A abordagem dos campos de potencial no planeamento de trajectórias baseia-se no uso da física dos potenciais eléctricos como heurística para encontrar a trajectória. O robô é representado por um ponto que se move num espaço com obstáculos. Esse ponto é considerado uma partícula de carga positiva que está a ser influenciado por um campo de potencial artificial. Os obstáculos e o ponto de destino é que vão criar o campo de potencial artificial. Os obstáculos comportam-se como partículas de carga positivas que criam campos repulsivos e o ponto de destino comporta-se como uma partícula de carga negativa que cria um campo atractivo. O somatório desses campos é o campo de potencial artificial. A combinação de campos atractivos com campos repulsivos força o robô a ir até ao ponto de destino evitando obstáculos, num cenário ideal.

Nesse cenário ideal o campo de potencial tem um mínimo no ponto de destino, não tem mínimos locais, que é o grande problema deste tipo de abordagem, e junto dos obstáculos o campo tende para infinito.

Nas subsecções seguintes será explicada sucintamente a matemática por de trás desta abordagem.

#### 2.3.4.1 Função potencial

O movimento da partícula vai ser gerado por uma função potencial.

Essa função é real e diferenciável:

$$U: \mathbb{R}^m \rightarrow \mathbb{R} \qquad (2.1)$$

podendo ser vista como a energia e o seu gradiente como uma força.

$$\Delta U(q) = DU(q)^T = \left[\frac{\partial U}{\partial q_1}, \ldots, \frac{\partial U}{\partial q_m}\right]^T \qquad (2.2)$$

O somatório dos vários campos resulta em:

$$U(q) = U_{destino}(q) + \sum U_{obstaculos}(q) \qquad (2.3)$$

O gradiente é a força que actua numa partícula, neste caso, o robô. Essa força designa-se por força de campo artificial, sendo que é esta a força que determina o movimento do robô.

$$F(q) = -\nabla U(q) \qquad (2.4)$$

#### 2.3.4.2 Potencial atractivo

O campo potencial atractivo criado pelo ponto do destino normalmente é representado por:

$$U_{destino}(q) = \frac{1}{2}\xi\|q - q_{destino}\|^2 \quad (2.5)$$

$$F_{atração}(q) = -\xi(q - q_{destino}) \quad (2.6)$$

em que $\xi$ é coeficiente de ganho.

A força atrai o robô para o destino e quanto mais afastado o robô estiver maior é a força, tendendo para zero quando se aproxima do destino.

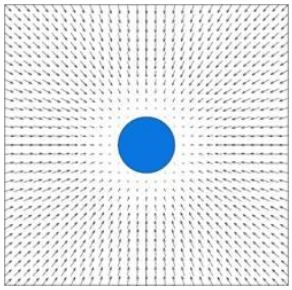


Figura 2.20 - Exemplo do campo gerado no ponto de destino

#### 2.3.4.3 Potencial repulsivo

O campo potencial repulsivo criado pelo obstáculo, normalmente é representado pela função parabólica:

$$U_{repulsão}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2, & \rho(q) \leq \rho_0 \\ 0, & \rho(q) > \rho_0 \end{cases} \quad (2.7)$$

$$\mathrm{F_{repulsão}(q)} = \begin{cases} -\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\left(\frac{1}{\rho(q)}\right)^2 \frac{\partial\rho(q)}{\partial x}, & \rho(q) \leq \rho_0 \\ 0, & \rho(q) > \rho_0 \end{cases} \quad (2.8)$$

em que $\eta$ é o coeficiente de ganho, $\rho_0$ é a distância de influência do obstáculo e $\rho(q)$ é a distância mínima do robô ao obstáculo.

A força de repulsão repele o robô, evitando que colide com o obstáculo, quanto mais perto está, maior é a sua influência no robô, sendo nula quando o robô está a uma distância superior a que se considera de limite de influência.

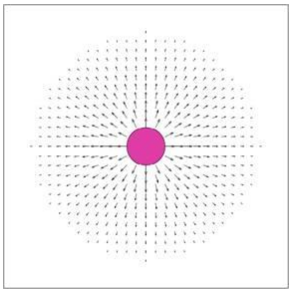


Figura 2.21 - Exemplo do campo gerado por um obstáculo

A combinação das duas forças gera o campo de potencial. Se o objectivo for simplesmente saber o próximo passo do robô não olhando para trajectória completa, então simplesmente o robô terá de seguir o sentido da força resultante no seu ponto. Se o objectivo for gerar a trajectória completa, então é necessário usar algum algoritmo para a encontrar, sendo que um dos mais utilizados é o algoritmo do gradiente descendente. Nesse algoritmo começa-se pelo ponto inicial e move-se no sentido oposto do gradiente resultante (sentido da força resultante). Em seguida vai-se para um novo ponto e nesse ponto repete-se o movimento até que o gradiente seja 0. Na Figura 2.22, uma trajectória foi gerada para chegar ao destino desviando-se de um obstáculo.

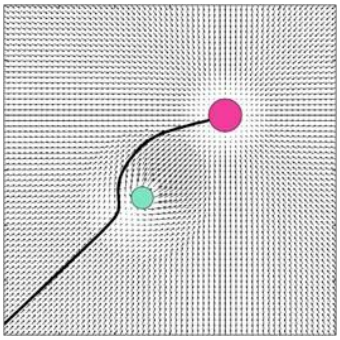


Figura 2.22 - Exemplo de uma trajectória gerada pelo campo de potencial artificial.

#### 2.3.4.4 Problema dos mínimos locais

Apesar do campo potencial ser um método muito popular existem situações especiais em que devido à posição do robô, dos obstáculos e do destino, o robô pode não conseguir chegar ao destino, Su, 2010 [66]. Os seguintes dois casos são os mais usuais:

- Quando o robô, o obstáculo e o destino estão alinhados e o obstáculo está entre o robô e o destino, se a repulsão for maior ou igual que a atracção, o robô nunca consegue chegar ao destino.
- Quando devido à acção de múltiplos robôs, de obstáculos e do ponto de destino, a força resultante for nula, isto é, temos a existência de um mínimo local e o robô fica parado nesse ponto.

Este último problema é o principal ponto fraco do método que faz desencorajar o seu uso. No entanto várias soluções foram estudadas ao longo dos anos e as duas principais categorias de resolução foram seguidas:

1. Utilizar campos de potenciais que não tenham mínimos locais a não ser no destino
2. Desenvolver métodos para fugir aos mínimos locais.

No 1º grupo estão incluídos os seguintes métodos:

- Connolly em 1990 [15] utiliza pela primeira vez funções harmónicas através da equação de Lapace para eliminar o mínimo local. Utilizando os limites dos obstáculos como condições limites na equação de Laplace, resulta uma função harmónica que não tem mínimo local
- O campo de potencial generalizado desenvolvido por Krog, em 1984 [67] que entra em consideração não só a posição do robô mas também com a sua velocidade. Se o robô se mover paralelamente aos limites de um obstáculo esse obstáculo não gera nenhuma força repulsiva no robô.
- O *vortex field* desenvolvido por Medio, em 1991 [68] substitui a força repulsiva por uma força que se comporta como um furacão, o campo potencial fica como um redemoinho.

No 2º grupo estão incluídos os seguintes métodos:

- Canny, 1998 [69] utilizou o *random walk* para sair do mínimo local. Após ter saído, volta ao método normal.
- Rimom, 1992 [70], elimina o problema do mínimo local "enchendo a região de mínimo local" com uma nova função potencial.
    - Outro dos métodos utilizados é o de seguir a parede (*Wall following*) de Yun, 1997 [71], isto é, quando encontra num mínimo local, o algoritmo muda para ir até o obstáculo mais perto e contorná-lo. Depois volta a utilizar o campo potencial.
    - Chang, em 1996 [72] usa várias funções de potencial para quando um mínimo local é encontrado, transformando esse mínimo noutro campo de potencial, com uma função diferente, esse local pode já não ser mínimo.

Ao longo dos anos este método foi muito utilizado em muitas áreas, sendo muito popular no futebol robótico. Em Peng,2004 [73] foram utilizados campos de potenciais preditivos, em Xinying, 2006 [74] utilizou-se, além dos campos de potenciais, também algoritmos genéticos, em

Su, 2010 [66] utilizou-se campos de potenciais *Fuzzy* e em Hongyan, 2006 [75] utilizou-se campos de potenciais caóticos.

### 2.3.5 Programação matemática

Este método baseia-se na formulação do planeamento da trajectória num problema matemático de optimização de modo a encontrar uma curva do início até ao destino minimizando uma determinada função custo: comprimento do caminho, tempo de execução, etc. Os obstáculos são representados por um conjunto de desigualdades. Como se está perante equações não lineares com muitas desigualdades provenientes de restrições, para se resolver o problema utilizam-se métodos numéricos. O problema desta abordagem é o facto de ser quase impraticável, isto é, torna-se muito difícil passar para a sua execução em tempo real. Vários estudos foram feitos: Sakamoto em 1993 [76] e Johnson em 1996 [77] utilzaram curvas B-*spline*, Bazaz em 1998 [78] utilizou curvas 3*cubic-spline*, Vaz em 2004 [79] utilizou programação semi-infinita e Lengagne em 2010 [80] utilizou polinómios de Taylor.

### 2.3.6 Outras abordagens

Com a crescente necessidade de resolver cada vez mais problemas mais complexos com graus de liberdade cada vez maiores apareceram novos métodos oriundos de outras áreas. De seguida serão apresentados brevemente alguns desses métodos, mostrando-se algumas das suas aplicações.

Uns dos métodos que tem evoluído nos últimos anos é o *Velocity Obstacles,* utilizado pela primeira vez em Fiorini, 1998 [81]. Neste método é definido o conjunto de todas as velocidades de um robô em que irá resultar numa colisão, nalgum momento temporal, assumindo que o obstáculo mantem a velocidade actual, O planeamento dos movimentos pretende encontrar as velocidades que estão fora desses conjuntos que garantem que não vão existir colisões. Várias variações foram desenvolvidas ao longo dos últimos anos tais como *common velocity obstacles,* Abe, 2001 *[82], recursive probabilistic velocity obstacles* Fulgenzi, 2007[83], *reciprocal velocity obstacles,*Berg,2008[84], *generalized velocity obstacles,* Berg,2009[85]. Este método é muito utilizado em simulações de multidões, apresenta no entanto dois problemas: primeiro os obstáculos parados são contornados pelas bordas do obstáculo o que faz com que o robô não seja rápido, como se pode constatar no vídeo do *paper* do Berg[86], o segundo problema é o tempo de processamento, como se constata em Berg[85].

A utilização de redes neuronais no planeamento de trajectórias foi introduzida em Zacksenhouse, 1988 [87]. Este método já foi aplicado numa grande variedade de sistemas, devido a

ser de fácil implementação, desde o planeamento de trajectórias num robô em ambientes de duas dimensões de Canny, 1988[88] até aos manipuladores de Zhao, 2010 [89].

O algoritmo genético (AG) foi usado pela primeira vez no planeamento de trajectórias por Arimoto, 1984 [63]. Em Canny, 1988 [31] foram usados algoritmos genéticos para planear múltiplas trajectórias. Em Zacksenhouse, 1988 [87] e em Skewis, 1990[90] estes algoritmos foram aplicados num robô real e em Dadios, 2001 [91] o AG foi aplicado num grupo de robôs num ambiente 3D. No futebol robótico, foi utilizado em Xinying, 2006, Xinying, 2007, Shih-Wen, 2010 e Song, 2010 [74, 92-94].

O método *Ant Colony Optimization* (ACO) é inspirado no comportamento das formigas quando estão a farejar por comida e têm de descobrir o caminho mais eficiente para a ela chegar. Este método foi introduzido no planeamento de trajectórias em Rivlin, 1997 [19] e foi desenvolvido em vários trabalhos tais como Zhao, 2010, Haibin, 2008 e Xiaoyong, 2009 [89, 95, 96]. No futebol robótico, foi utilizado em Juing-Shian, 2009 e Kunli, 2010 [97, 98].

O método *Particle Swarm Optimization* (PSO) é inspirado na capacidade dos bandos de pássaros, cardumes de peixes e dos rebanhos de animais, de se adaptarem ao seu ambiente, isto é, encontrarem fontes ricas de alimentos e evitar os predadores através de uma "partilha de informação" de modo a desenvolver uma vantagem evolutiva. Foi aplicado pela primeira vez em Li, 2006 [99]. No futebol robótico foi utilizado por, Shih-Wen, 2010 e Li, 2006 [93, 99]

Em Kim, 2003 [100] foi utilizada pela primeira vez a lógica *fuzzy* no planeamento de trajectórias. A lógica *fuzzy* é utilizada preferencialmente em conjunto com outros métodos tais como as redes neuronais Borenstein, 1989 e Xinying, 2006 [71, 74] e campos potenciais Kavraki, 1998[44] . No futebol robótico foi utilizado por Dadios, 2002, Dadios, 2001 e Guangshun, 2008 [91, 101, 102].

# Capítulo 3

# Algoritmos de Pesquisa e Implementação do A*

Neste capítulo serão apresentados alguns algoritmos de pesquisa. O algoritmo A*, o algoritmo escolhido para ser implementado, será descrito em detalhe assim como as suas propriedades. Serão ainda abordados os principais problemas que podem surgir na implementação do algoritmo A* e tentar-se-á rebater o mito de que o algoritmo A* é muito lento. Para terminar o capítulo será efectuada a descrição do processo usado no cálculo de trajectórias no futebol robótico.

## 3.1 Algoritmos de pesquisa

No capítulo anterior foram descritas as abordagens mais utilizadas para o planeamento de trajectórias. Em algumas dessas abordagens o $C_{espaço}$ é transformado em grafos que têm nós com ligações entre eles. Nessa categoria de abordagens estão os *roadmap* e a decomposição em células.

Para descobrir a trajectória recorrendo a estas abordagens deve-se pesquisar o grafo à procura do melhor caminho. Por isso é muito importante a correcta escolha do algoritmo que vai executar essa tarefa, não só por causa da sua capacidade em descobrir ou não a solução, mas principalmente porque este tem de ser eficiente. O tempo de execução do algorítmo é crítico, no mundo da robótica não se pode ficar segundos à espera de uma solução.

### 3.1.1 Algoritmos sem informação

Os primeiros algoritmos a resolver o problema são os chamados algoritmos de pesquisas exaustivas. Estes algoritmos não utilizam nenhuma informação que ajude a encontrar o destino. Seguem uma ordem pré-definida e tentam encontrar o destino. Só quando encontram o destino é que sabem que chegaram à solução, nos momentos anteriores não sabem sequer se estão perto ou não. Dentro desta categoria existem muitos algoritmos, os mais reconhecidos são:

- a pesquisa por profundidade
- a pesquisa por largura
- a pesquisa por profundidade limitada
- a pesquisa por aprofundamento iterativo

Em seguida serão apresentados cada um dos algoritmos e para cada um serão indicados alguns factores que explicam a sua eficácia e eficiência. Os factores são:

- Completo ou não, isto é, se encontra uma solução sempre que esta exista
- Complexidade de tempo, que indica a quantidade de tempo necessário para resolver o problema. Normalmente está relacionado com o número de nós expandidos
- Complexidade de espaço, que indica o máximo de nós guardados em memória
- Optimalidade, se encontra sempre a solução de mais baixo custo.

A complexidade de tempo e de espaço são medidas usando os seguintes termos:

b – número máximo de ligações para um nó

d – profundidade da solução óptima ( profundidade representa o número de nós )

m – máxima profundidade do $C_{espaço.}$

#### 3.1.1.1 Pesquisa por profundidade

O algoritmo de pesquisa por profundidade, o *depth-first search* (DFS), é um algoritmo de pesquisa que explora os nós o mais longe possível antes de recuar. Isto é, para cada nó só explora a 1ª ligação que encontra e assim sucessivamente até encontrar o destino ou um nó sem ligações. Se encontrar um nó sem ligações volta atrás até encontrar um nó que tenha mais ligações para explorar. O algoritmo é completo para o caso de b ser finito, a complexidade de tempo é $O(b^m)$, a complexidade de espaço é O(b.m), são guardados todos os nós em memória e não é óptimo.

A Figura 3.1 mostra como é executada a pesquisa, em que a numeração dos nós indica a ordem pela qual foram expandidos.

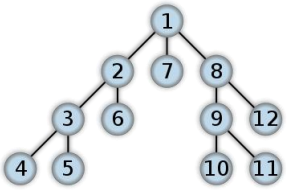



Figura 3.1 - DFS numa árvore

#### 3.1.1.2 Pesquisa por largura

O algoritmo de pesquisa por largura, breadth-first search (BFS), é um algoritmo de pesquisa que explora todas as ligações de um nó antes de passar para outro nó e assim sucessivamente até encontrar o destino. O algoritmo é completo para o caso de b ser finito, a complexidade de tempo é $O(b^{d+1})$, a complexidade de espaço é $O(b^{d+1})$, são guardados todos os nós em memória e não é óptimo, excepto no caso em que o custo é sempre igual para todas as ligações.

A Figura 3.2 mostra como é executada a pesquisa com este algorítmo, em que a numeração dos nós indica a ordem pela qual foram expandidos.

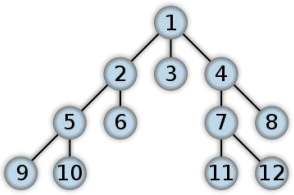



Figura 3.2 - BFS numa árvore

#### 3.1.1.3 Pesquisa por aprofundamento limitado

O algoritmo de pesquisa por aprofundamento limitado, *depth–limited search* (DLS), é um algoritmo de pesquisa que executa o DFS com um limite imposto à profundidade a que se pode pesquisar, isto é, se o algoritmo chegou ao limite imposto à partida, mesmo que o nó tenha ligações possíveis de serem expandidas, o algoritmo não expande mais, volta para trás. O algoritmo é completo para o caso de b ser finito e para o caso de o destino estar dentro do limite imposto, a complexidade de tempo é $O(b^p)$ em que p é o limite, a complexidade de espaço é O(b.p), são guardados todos os nós em memória e não é óptimo, excepto no caso em que o custo é sempre igual para todas as ligações.

#### 3.1.1.4 Pesquisa por aprofundamento iterativo

O algoritmo de pesquisa por aprofundamento iterativo, *iterative deepening depth–first search* (IDDFS), é um algoritmo de pesquisa que executa o DLS as vezes que forem necessárias, em que o limite de profundidade vai ser aumentado, até encontrar o destino. O algoritmo é completo para o caso de b ser finito, a complexidade de tempo é $O(b^m)$, a complexidade de espaço é $O(b^d)$, são guardados todos os nós em memória e não é óptimo, excepto no caso em que o custo é sempre igual para todas as ligações.

### 3.1.2 Algoritmos com heurística

Os algoritmos com heurística começaram a ganhar notoriedade graças à sua capacidade de reduzir muito o seu tempo de execução, quando comparados com os algoritmos sem informação. Estes algoritmos usam alguma informação para escolher a sequência de pesquisa. Dentro desta categoria existem muitos algoritmos, os mais reconhecidos são:

- Algoritmo Dijkstra's
- Algoritmos do tipo guloso
- Algoritmo A*.

#### 3.1.2.1 Algoritmo Dijkstra's

O algoritmo Dijkstra's, descrito pela primeira vez por Dutch [103] em 1956 é um algoritmo de pesquisa que entra em conta com a distância percorrida. O algoritmo vai explorar sempre o nó

que está mais perto do ponto inicial em cada iteração. A partir do nó inicial, o nó seguinte será o que está a uma distância mais curta, e assim sucessivamente até encontrar o destino. O algoritmo é completo para o caso de b ser finito, a complexidade de tempo é $O(b^m)$, a complexidade de espaço é $O(b^d)$, são guardados todos os nós em memória e não é óptimo, excepto no caso em que o custo é sempre igual para todas as ligações.

#### 3.1.2.2 Algoritmos do tipo guloso

Os algoritmos do tipo guloso, *Greedy Search*, são algoritmos de pesquisa que entram em conta com a informação de quanto falta para alcançar o destino. Em cada instante a melhor solução é encontrada, isto é, são optimos locais. Os algoritmos vai explorar sempre o nó que está mais perto do destino em cada iteração. A partir do nó inicial, o nó seguinte será o que está a uma distância mais curta do destino, e assim sucessivamente até encontrar o destino. O algoritmo é completo para o caso de b ser finito, a complexidade de tempo é $O(b^m)$, a complexidade de espaço é $O(b^m)$, são guardados todos os nós em memória e não é óptimo.

#### 3.1.2.3 Algoritmo A*

O algoritmo A* funciona essencialmente como o Dijkstra's, mas entra em conta não só com a informação dos nós que estão mais perto mas também com a informação de quanto falta para alcançar o destino. Ou seja, este algoritmo pesquisa primeiro os nós que considera que são mais promissores. O algoritmo é completo, óptimo e a complexidade de tempo e de espaço depende da heurística, principalmente da qualidade da função heurística Este algoritmo é o mais eficiente dos apresentados. Este foi o algoritmo escolhido para ser implementado nesta tese, por isso na secção seguinte será descrito ao pormenor.

### 3.1.3 Algoritmos recentes

Os algoritmos descritos anteriormente funcionam bem quando o problema a resolver é um grafo completo e estático. Quando se trata de cenários em tempo real, em que o $C_{espaço}$ está em constante alteração, os algoritmos têm de estar constantemente a refazer o planeamento. No caso do algoritmo A*, a solução passa por refazer o planeamento desde o início. O algoritmo A* ao longo do tempo ganhou a fama de ser computacionalmente muito lento, e por isso de pouca utilidade em cenários em tempo real. Por esse motivo começaram a ser desenvolvidos outros algoritmos a partir do algoritmo A* para serem implementados em tempo real. Esses algoritmos foram desenvolvidos com a premissa de que não havia necessidade de resolver o planeamento desde de o início da trajectória, sendo só necessário reajustar a trajectória. Dentro desses algoritmos existem o IDA*,ARA*,AD*, MA*, SMA*, D*, *Focused* D*, D* *lite*, E* e outros.

Em relação ao algoritmo A* estes algoritmos ou geram a mesma solução que é a óptima, ou geram uma solução sub-óptima, isto é, uma solução pior. As vantagens destes algorítmos são que o MA* e o SMA* usam menos memória, o IDA*e o ARA* executam em menos tempo e os restantes têm a vantagem de em alguns casos em tempo real executarem em menos tempo que o algoritmo A*.

Em 1984 Pearl [104] apresentou a versão IDA*, que consiste em definir-se um limite k para a função f(), usando a mesma conceito usado no IDDFS. O algoritmo executa A* em que não se expande os nós que tenham f(n) >k. Se o algoritmo não encontrou o destino incrementa-se o k e volta-se a executar o algoritmo A*. Esta situação repete-se as vezes necessárias até encontrar o destino ou até esgotar-se um determinado tempo predefinido. Se o algoritmo termina sem encontrar o destino, a solução final será a solução mais perto deste.

O algoritmo MA* foi adaptado a partir do A* por Chakrabarti, em 1989 [105]. Este algoritmo consiste em retirar os nós menos promissores, ou seja, com f mais elevados da lista aberta, quando a memória está cheia, para dar espaço a novos nós. Quando a lista aberta e a lista fechada chegam a um número predefinido de nós, o próximo nó a entrar na lista aberta faz com que o nó com o f mais elevado seja retirado da lista. O antecessor desse nó retirado fica com a informação actualizada do f.

Em 2002, Rong Zhou e Eric A. Hansen [106], simplificaram o algoritmo MA* de modo a ser mais eficiente, e para isso utilizaram uma estrutura de dados mais eficiente e resultou no SMA*, mas o conceito geral é o igual ao MA*.

O algoritmo ARA* desenvolvido por Likhachev, 2003 [107] é um tipo de algoritmo chamado *anytime*, que se baseia no facto de que se pode construir soluções sub-óptimas em tempos reduzidos. No algoritmo A* consegue-se reduzir o tempo de execução utilizando uma heurística inflacionada chamada *weighted* A*. Assim, o algoritmo ARA* constrói uma solução sub-óptima e vai melhorando enquanto existir tempo disponível. Existem ainda vários outros algoritmos baseados nesse conceito desenvolvidos por Bonet 2001, Zhou 2002, Edelkamp 2001, Rabin 2000 e Chakrabarti 1998 [108-112].

O D* desenvolvido por Stentz, 1994 [113], o *Focused* D* também desenvolvido por Stentz[114], o D* *lite* desenvolvido por Koenig, 2002 [115] e o *Delayded* D*[116] são algoritmos que fazem o reajustamento da trajectória sem a necessidade de refazer todo o processo desde o início. No início da sua execução, estes algoritmos constroem uma trajectória exactamente como o algoritmo A*. O algoritmo D* *lite* tem uma pequena diferença que é construir a trajectória do destino para o início, em vez de ser do início para o destino.

No instante seguinte em que é necessário refazer a trajectória, o algoritmo A* refaz tudo desde o início. Os outros algoritmos se não houver alteração dos $C_{espaço}$, não fazem nada, ou seja, mantêm a trajectória. Se houver alteração dos $C_{espaço}$, estes algoritmos vão refazer a parte que tem alterações propagando essas alterações a todos os nós que são afectados. Cada algoritmo tem a sua própria maneira de propagar as alterações.

O algoritmo AD* apresentado por Likhachev, 2005 [117] é uma junção dos conceitos de *anytime* e do D* *lite*, ou seja, consegue utilizar o conceito de reajustamento do D* *lite* cada vez que melhora a sua heurística.

Em [118] foi demonstrado que para algumas tarefas de navegação o D* *lite* é mais eficiente do que o D*. E em [119] o *Delayded* D* foi considerado mais eficiente do que D* *lite* em alguns cenários em que a distância entre o início e o destino é grande e as mudanças ocorridas são aleatórias. Tipicamente em cenários em que as alterações ocorrem perto do robô, os algoritmos de reajustamento são muito mais eficazes do que o algoritmo A* [119]. Quando existem muitas alterações ou as alterações são perto do destino, esses algoritmos deixam de ser eficazes, isto é, fica mais rápido refazer do início as trajectórias [119].

## 3.2 Escolha do algoritmo

No capítulo seguinte será apresentada a plataforma de teste utilizada nesta tese, o futebol robótico. Neste caso específico o planeamento de trajectória é em tempo real em ambiente dinâmico, em que os obstáculos e o ponto de destino estão em constante mudança. É ainda muito importante salientar desde já o facto de ser preciso gerar 10 trajectórias independentes em cada instante. A explicação deste valor será apresentada no capítulo seguinte.

A abordagem utilizada para o planeamento de trajectórias foi a de decomposição em células aproximadas com células fixas. Assim é preciso escolher o algoritmo de pesquisa que possa responder às necessidades exigidas.

Perante o facto de as alterações ao $C_{espaço}$ que podem ocorrer, tanto podem ser próximo do robô como próximo do destino, e que o destino está constantemente a mudar, a escolha recai facilmente sobre o A*, visto que acaba de ser mais eficiente do que os algoritmos de reajustamento. Outro aspecto muito importante a ter em atenção é o facto de serem precisas 10 trajectórias, e como nos algoritmos de reajustamento todas as informações ficam armazenadas logo é preciso guardar em memória 10 vezes mais informação do que no algoritmo A*.

## 3.3 Algoritmo A*

O algoritmo A* é um algoritmo que foi descrito pela primeira vez por Peter Hart, Nils Nilsson, e Bertram Raphael [120], em 1968, e é um algoritmo de procura num grafo, designado

melhor primeiro. Este algoritmo consiste em calcular o caminho mais curto num grafo entre o nó inicial e o nó final. O algoritmo utiliza uma função heurística f(n) que determina qual a ordem pela qual deve procurar os nós de modo a encontrar o melhor caminho, o mais rápido possível. A função utilizada é a soma de duas funções (Figura 3.3):

g(n) – É uma função, que pode ou não ser uma função heurística, que representa o custo actual desde do nó início até ao nó n.

h(n) – É uma função heurística que faz a estimativa do custo de ir do nó n até ao nó final.

$$f(n) = g(n) + h(n) \tag{3.1}$$

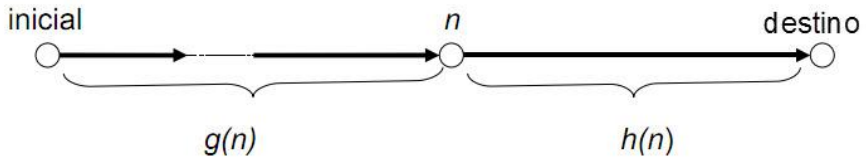



Figura 3.3 - Representação de f(n), g(n) e h(n)

O algoritmo A* foi descrito inicialmente da seguinte forma:

1) Inicia-se com o nó inicial, calcula-se o seu $f(n_{inicial})$ e coloca-se esse nó numa lista de nós designada por lista aberta.

Os procedimentos seguintes são repetidos até se chegar ao nó de destino ou até a lista aberta ficar vazia, o que implica que não existe solução.

2) Escolhe-se o melhor nó n (aquele que tem a função de custo f(n) mais baixa) que se encontra na lista aberta. Este nó passa a ser considerado já processado, deixa de existir na lista aberta e passa a existir numa lista designada por lista fechada que contém todos os nós já processados.

3) A partir daí vão-se avaliar todos os nós adjacentes (nós que estão directamente ligados ao nó n), que podem ser um dos seguintes casos:

a) Serem inseridos na lista aberta se ainda lá não existirem e não pertencerem à lista fechada, com a informação do seu $f(n_i)$ e do seu antecessor, designado por Pai, ou seja o nó procedente de onde veio até chegar a ele.

b) Caso existam na lista aberta, vai-se determinar se a função f é menor que o valor da função f que estava anteriormente associada ao nó, isto é, verificar se seguindo este percurso a função custo é menor. Sendo este o caso, modifica-se o pai do nó e o valor de f.

c) Se pertencer à lista fechada verifica se a função f é menor por este caminho do que quando esse nó foi processado. Se assim for, esse nó passa a pertencer outra vez à lista aberta.

A trajectória resultante é encontrada começando com o nó final e a partir dele vê-se qual é o seu pai. Em seguida verifica-se nesse nó o seu Pai e assim sucessivamente até se chegar ao nó inicial.

Este algoritmo, tal como está apresentado, não tem uma boa eficiência. Esta inferência deve-se ao facto de se ter de visitar o mesmo nó várias vezes. Como veremos na secção das propriedades, dando-se o caso de termos uma heurística consistente, podemos eliminar a parte 3c), isto é, não é necessário voltar a abrir um nó depois de tê-lo posto na lista fechada. Esta circunstância torna o algoritmo muito mais eficiente e como será demonstrado continua óptimo, isto é, será a solução de menor custo.

Uma vez desenvolvida a apresentação do algoritmo A*, torna-se igualmente necessário proceder a uma especificação detalhada da terminologia adoptada no mesmo:

O - A lista aberta contém os nós que ainda não foram seleccionados e que podem vir a ser escolhidos.

C - A lista fechada contém os nós já processados, isto é, nós que já saíram da lista aberta, uma vez que já foram analisados.

Star(n) – Representa o conjunto de nós adjacentes do nó n, isto é, todos os nós que estão ligados ao nó n.

$c(n_1,n_2)$ – É o custo de passar do nó $n_1$ para nó $n_2$

f(n) = g(n) + h(n) é a função custo do caminho mais curto desde o nó de início até ao nó final em que passa pelo nó n

g(n) – É o custo actual desde o início até ao nó n.

h(n) – É a função heurística de custo que faz a estimativa de ir do nó n até ao nó final.

### 3.3.1 Pseudocódigo do Algoritmo

1. Adicionar o nó de partida a O
2. Repetir
   2.1. Escolher o $n_{melhor}$ (nó melhor) de O tal que $f(n_{melhor}) \leq f(n) \ \forall n \in O$
   2.2. Remover o $n_{melhor}$ de O e adicionar a C

2.3. Se $n_{melhor}$ = nó final , terminar o algoritmo
2.4. Para todos n $\in$ Star($n_{melhor}$) fazer o seguinte:
2.4.1. Se (n $\notin O$ ) e (n $\notin C$ ) então adiciona o nó n a O
2.4.2. Se (n $\in O$ ) então se g($n_{melhor}$ ) + c($n_{melhor}$,n) < g(n) então alterar o pai do nó n para $n_{melhor}$
2.4.3. Se (n $\in C$ ) então se g($n_{melhor}$ ) + c($n_{melhor}$,n) < g(n) então alterar o pai do nó n para $n_{melhor}$ e passar n para a lista aberta.
3. Até que O esteja vazio

## 3.4 Propriedades

Veremos seguidamente algumas das propriedades que tornam o algoritmo A* fiável e eficaz. Em [120] foi provado que o algoritmo é completo e que é admissível perante a escolha de uma heurística admissível e/ou consistente.

### 3.4.1 Completo

O algoritmo A* diz-se completo, já que encontra sempre uma solução, caso exista. Como o algoritmo só termina quando:

- ou chegou ao destino, isso implica que existe uma solução,
- ou quando a lista aberta está vazia, o que implica que houve uma busca por todos os nós alcançáveis

tem-se, portanto, a garantia de que se consegue obter sempre a solução.

### 3.4.2 Admissível

O algoritmo A* é admissível ou óptimo se a função heurística h for admissível. Isso acontece se a função nunca sobrestimar o custo de chegar ao destino.

$$h(n) <= h_m(n) \quad \forall n \tag{3.2}$$

$h_m(n)$ é o menor custo de n até ao destino

No caso de se pretender visitar o nó uma única vez, é necessário garantir que a função h seja monótona e consistente.

O nó destino é sempre encontrado através do caminho óptimo. Esta afirmação pode ser comprovada através da contradição. Isto é, pode-se provar que é impossível chegar ao nó destino através de um caminho sub-óptimo.

**Demonstração:**

Assume-se que se chega ao destino através de um caminho sub-óptimo com o custo do caminho a ser de f(d), em que d é o nó destino. Se é um caminho sub-óptimo então f(d) > f*(d) em que f*(d) é o caminho óptimo. Para o caminho sub-óptimo ter sido escolhido, significa que d foi seleccionado em vez de n (um nó do caminho óptimo) da lista aberta. Se o nó n não foi escolhido em relação ao nó d, verifica-se que o algoritmo escolhe sempre o nó com o f menor, logo f(n) ≥ f(d). Como h é admissível, então f*(d) ≥ f(n) ( o custo actual é sempre maior ou igual ao custo estimado pela heurística). Destas duas afirmações implica que f*(d) ≥ f(d), o que implica f*(d) = f(d), logo o algoritmo A* nunca escolhe um caminho sub-óptimo.

### 3.4.3 Consistente

Uma função h é considerada consistente se, para qualquer par de nós adjacentes a seguinte fórmula for verdadeira:

$$h(n) \leq c(n,m)+h(m) \text{ para } \forall n,m \text{ em que } \exists c(n,m) \tag{3.3}$$

Isto é, para todos nós que exista ligação entre eles, a estimativa de chegar ao destino é sempre menor ou igual ao custo de chegar ao nó adjacente, mais a estimativa desse nó ao destino.

### 3.4.4 Complexidade

O tempo de execução do algoritmo A* depende da heurística, principalmente da qualidade da função heurística. Na pior heurística possível (h(n)=0) verifica-se que a complexidade é exponencial $O(2^n)$ , no melhor cenário ($h(n)=h_m(n)$) a complexidade é linear O(n).

## 3.5 Problemas de implementação

Na implementação do algoritmo A* existem vários aspectos a ter em consideração, que podem afectar significativamente o desempenho do algoritmo A*:

- a velocidade do CPU,
- a limitação da memória,
- o tipo de estrutura de dados a usar para as listas aberta e fechada e qual a heurística a utilizar.

A velocidade do CPU e a velocidade de acesso à memória vão limitar o tempo de execução do algoritmo, pelo que, logicamente, quanto mais rápida for a velocidade do CPU e da velocidade de acesso à memória, mais rápido será o tempo de execução.

A memória pode ser um factor importante a ter em consideração, visto que é necessário guardar a informação dos nós e da função f(n). Por isso, se o grafo for muito grande pode dar-se o problema da memória não ser suficiente. Para o caso do futebol robótico tanto a memória como o tempo de execução não serão problemas.

### 3.5.1 Heurísticas

Existem várias funções heurísticas a usar num mapa de células. Convém usar heurísticas simples, isto é, que seja rápido descobrir o valor e heurísticas o mais reais possíveis. Quanto mais próximas dos valores reais, melhores serão os resultados.

#### 3.5.1.1 Distância Manhattan

Nesta heurística calcula-se a distância só se movendo em duas direcções: na horizontal e na vertical.

$$h(n) = D \times ( abs(n_x - destino_x) + abs(n_y - destino_y)) \quad (3.4)$$

A heurística representa o número de células a percorrer na horizontal mais o número de células a percorrer na vertical de modo a alcançar o destino.

A Figura 3.4 apresenta um exemplo em que se move na horizontal e na vertical, nesse exemplo o h(n) = 3+2 = 5.

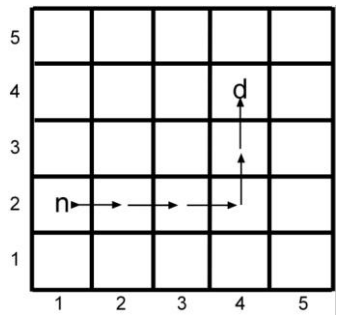



Figura 3.4 - Heurística distância Manhattan

#### 3.5.1.2 Distância Diagonal

Nesta heurística calcula-se a distância movendo-se: na vertical, na horizontal e na diagonal.

$$h_{diagonal}(n) = \min(abs(n_x - destino_x), abs(n_y - destino_y)) \quad (3.5)$$

$$h_{direitos}(n) = \max(abs(n_x - destino_x), abs(n_y - destino_y)) \quad (3.5)$$

$$h(n) = D_2 \times h_{diagonal}(n) + D \times (h_{direitos}(n) - h_{diagonal}(n))) \quad (3.6)$$

em que

$h_{diagonal}$ representa o número de passos que se executa na diagonal

$h_{direitos}$ representa a número máximo de passos que se executa numa determinada direcção

$D_2$ – é o peso de andar na diagonal

D – é o peso de andar na outra direcção

A Figura 3.5 apresenta um exemplo em que se move na horizontal e na diagonal, nesse exemplo o $h(n) = D_2 \times 1 + D \times 2$

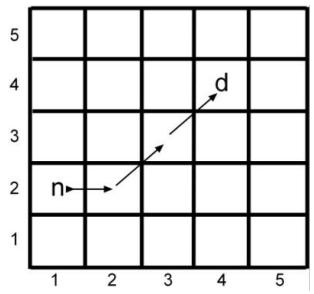



Figura 3.5 - Heurística Diagonal

#### 3.5.1.3 Distância euclidiana

Nesta heurística pode-se mover em qualquer direcção e utiliza-se a distância euclidiana.

$$h(n) = D \times \sqrt{(n_x - destino_x)^2 + (n_y - destino_y)^2} \quad (3.7)$$

representa a distância euclidiana em linha recta até ao destino

A Figura 3.6 apresenta um exemplo em que se move na horizontal e na diagonal.

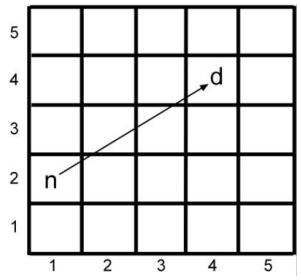



Figura 3.6 – Heurística euclidiana

## 3.6 Estrutura de Dados

Outro problema a ter em consideração é a estrutura das listas. Grande parte do tempo de execução do algoritmo é devido ao tempo despendido a retirar e a colocar os nós nas listas e a escolher o nó com o valor de f() mais baixo. Várias estruturas podem ser usadas tais como:

- *arrays* ordenados,
- listas ligadas ordenadas,
- tabelas *Hash*,
- *binary heaps*,

Em seguida apresenta-se uma tabela que representa a complexidade associada ao tempo de inserir, de eliminar e de procurar um elemento em cada uma das estruturas pode ser consultada em [121].

Tabela 3.1 - Tabela que relaciona os tipos de estrutura de dados com o tempo despendido em tarefas

| | ***Array* Ordenados** | **Listas ligadas ordenadas** | **Tabelas *Hash*** | ***Binary heaps*** |
|---|---|---|---|---|
| **Procura** | O(1) | O(n) | O(n) | O(1) |
| **Inserir** | O(n) | O(1) | O(1) | O(log n) |
| **Eliminar** | O(n) | O(1) | O(1) | O(log n) |

Na escolha do tipo de estrutura deve-se ter em conta não só a quantidade de elementos que serão manuseados, mas principalmente que tipos de tarefas ocorrem mais regularmente. No caso do algoritmo A* verifica-se que as tarefas de procurar, inserir e eliminar são executados na mesma ordem de grandeza.

Nas implementações iniciais utilizaram-se *array* ordenados, tendo posteriormente sido implementado o *binary heaps*, com resultados muito melhores. Com esta solução o tempo de execução do algoritmo decresceu cerca de 60%.

## 3.7 Implementação

Para se conseguir utilizar o algoritmo A* no cálculo de trajectórias é conveniente definir alguns dos parâmetros a adaptar.

Em primeiro lugar utilizou-se a decomposição em células aproximadas em células fixas. Isto é, divide-se o mapa do ambiente, neste caso o campo de jogo, em células.

Assim, cada célula representa um nó, sendo que cada um deles tem uma ligação correspondente a um dos nós adjacentes, o que implica um determinado custo. Este custo é traduzido, neste caso, pela distância métrica entre os nós.

Em seguida, considera-se o robô para o qual se pretende encontrar a trajectória e que é representado pelo nó inicial e ocupa um só nó, sendo esse nó o centro geométrico do robô visto por cima. O nó destino é o ponto para o qual o robô pretende ir. Todos os outros robôs da mesma equipa, bem como da equipa adversária são os obstáculos. Como o robô é representado por uma única célula, os obstáculos têm de ser maiores de modo a abranger a largura do robô, como se verifica na Figura 3.7. Esses obstáculos são representados por circunferências de raio igual à soma do raio do obstáculo mais o raio do robô, para o qual se pretende a trajectória.

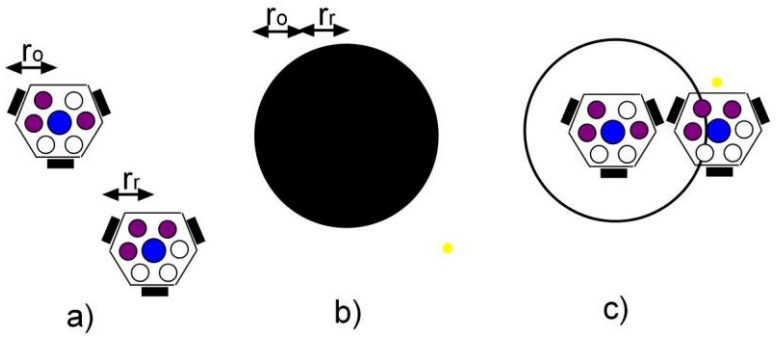



Figura 3.7 - a) robô real b) representação no $C_{espaço}$ c) Explicação do aumento do obstáculo

Na Figura 3.8 as células a preto representam os obstáculos, a célula a amarelo indica o ponto inicial e a azul o ponto de destino.

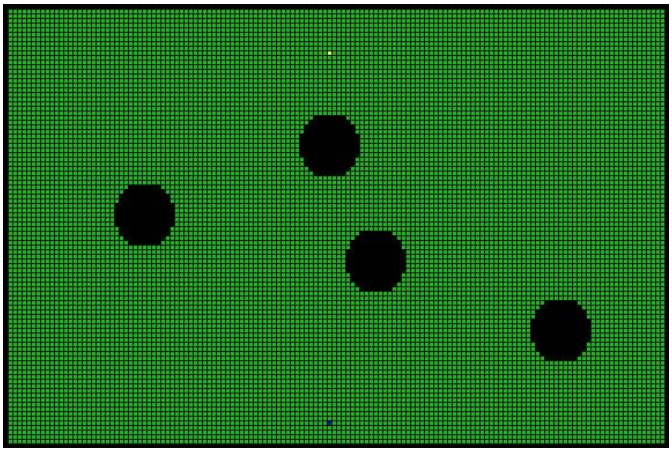


Figura 3.8 - $C_{espaço}$ criado no software

### 3.7.1 Heurística

A heurística utilizada foi a Euclidiana, mas introduziu-se um ganho K na fórmula

$$h(n) = K \times D \times ((n.x\text{-}final.x)^2 + (n.y\text{-}final.y)^2)^{1/2} \quad (3.8)$$

Com K > 1

Este ganho K serve para dar mais importância aos nós que estão mais perto do destino. Como K > 1, o h será maior, o que tem como consequência que o valor da função f() vai aumentar proporcionalmente à distância ao destino. Logo as células que estão mais perto têm um acréscimo do f menor das que estão mais longe, como consequência esses nós ficam melhor posicionados para serem escolhidos. Por exemplo quando se tem dois nós na lista aberta com os seguintes valores de g e h:

Tabela 3.2 - Exemplo da consequência de se utilizar o k

| nós | g | h | f com k = 1 | f com k = 1.3 |
|---|---|---|---|---|
| **1** | 1 | 5 | 6 | 7.5 |
| **2** | 3.5 | 3 | 6.5 | 7.4 |

O nó menor com K = 1 é o primeiro mas quando se passa para K = 1.3 o segundo nó que está mais perto do destino passa a ter um f menor.

A vantagem em aumentar o K é que vão ser analisados menos nós, e como consequência o tempo de processamento vai ser menor.

A desvantagem é que ao utilizar o K > 1, a heurística deixa de ser admissível, o algoritmo deixa de ser óptimo, logo o caminho encontrado pode não ser o caminho óptimo.

A Figura 3.9 mostra os nós das listas fechada e aberta que foram analisados no primeiro caso para K=1 e no segundo caso para K = 1.5.

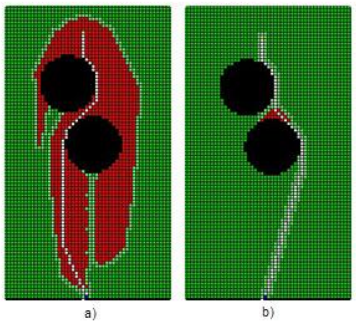



Figura 3.9 - a) $C_{espaço}$ com K = 1 b) $C_{espaço}$ com K = 1.5 - o vermelho representa nós pertencentes à lista fechada, o verde representa os nós pertencentes à lista aberta e o branco representa a trajectória encontrada

Tabela 3.3 - Dados relativos aos dois casos K=1 e K=1.5

| | **Tempo de processamento** | **Comprimento da trajectória** | **Nº de nós processados** |
|---|---|---|---|
| **K=1** | 1.04 ms | 2.76 m | 1214 |
| **K=1.5** | 0.30 ms | 2.77 m | 94 |

Como se pode verificar existe uma diferença muito grande ao nível do tempo de processamento entre os dois casos,. Para além disso, a trajectória encontrada é completamente diferente apesar de o comprimento da trajectória ser quase igual. A diferença de tempo de processamento resulta do número de nós processados.

É preciso encontrar o ganho K que melhor combine os dois aspectos, ou seja, que faça baixar o tempo de processamento, mas que não se distancie muito do caminho óptimo.

Foram efectuadas várias simulações em diferentes situações com o ganho K a variar entre 1 e 2. Das várias das simulações efectuadas as mais representativas, ou seja, as que influenciavam mais as trajectórias são as seguintes:

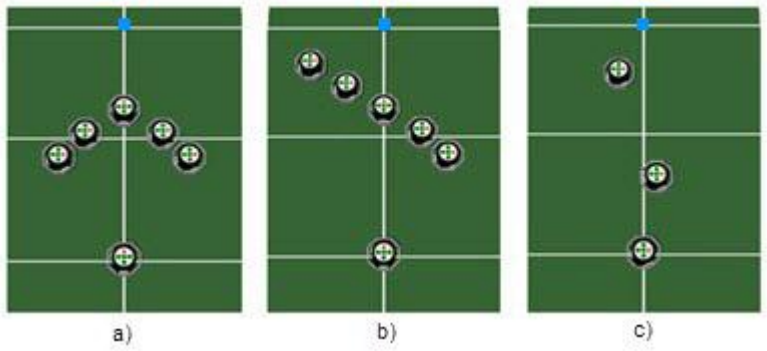



Figura 3.10 - a) 1ª situação em que existe o beco sem saída b) caminho mais curto não é o mais óbvio c) ziguezague entre obstáculos

O robô que se encontra mais abaixo nas imagens é o robô para o qual se pretende encontrar a trajectória até ao ponto azul, que se encontra a 2 metros de distância.

Na primeira imagem têm-se a clássica situação de existir um beco sem saída. Na segunda imagem o caminho mais curto, efectivamente, é indo pela direita, porque se for em frente ou pela esquerda encontra obstáculos, embora estes se situem mais próximos do destino. Por fim na terceira imagem, existem dois obstáculos, de modo que o caminho mais curto é um ziguezague por entre eles.

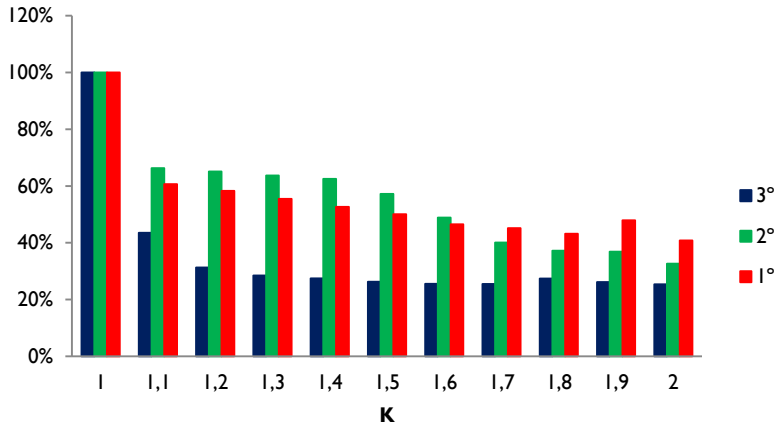



Gráfico 3.1 - Tempo de processamento em relação ao tempo de processamento do K = 1

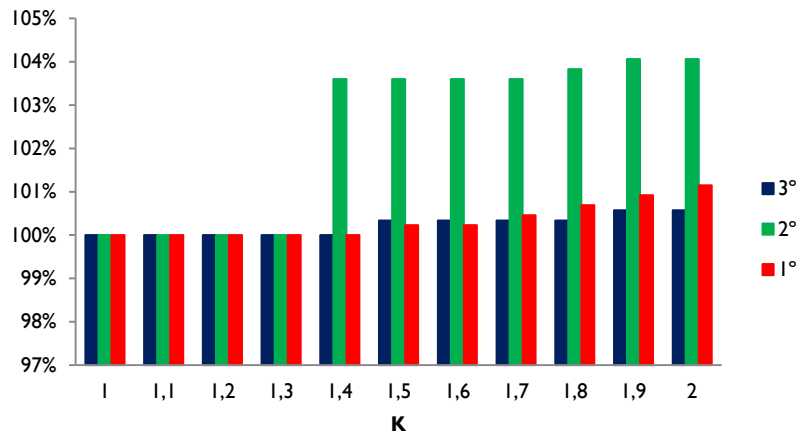



Gráfico 3.2 – Comprimento da trajectória em relação ao comprimento de K=1

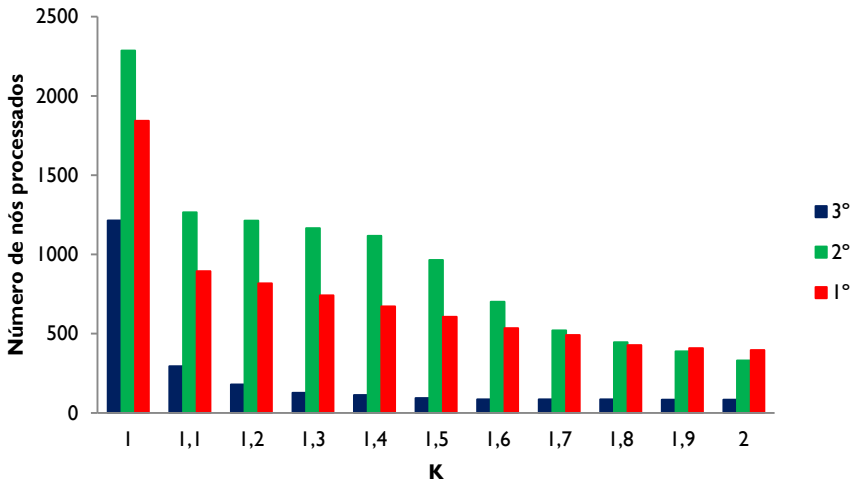



Gráfico 3.3 - Número de nós processados nas 3 situações

Da Gráfico 3.3 conclui-se que à medida que o K aumenta o número de nós diminui significativamente. Essa diminuição é muito mais significativa quando se passa de K = 1 para K = 1.1. Como consequência, o tempo de processamento segue o mesmo comportamento, ou seja, à medida que K aumenta o tempo diminui (Gráfico 3.1). Em qualquer situação o tempo diminuiu à medida que se aumenta o K, sendo que passar de K = 1 para K = 1.1 é a solução que obtém um ganho maior. Na situação 3 é onde existe uma diferença maior entre o K = 1 e K = 1.1.

Em relação à trajectória (Gráfico 3.2) verifica-se que até K = 1.3 o caminho mantém-se e que, em K = 1.4, na segunda situação houve um ligeiro aumento. Daí em diante as trajectórias aumentam. De realçar que na segunda situação a partir de K = 1.4 a trajectória encontrada é completamente diferente da trajectória óptima. A trajectória óptima é pela direita e a encontrada para K ≥ 1.4 foi pela esquerda.

Perante estes resultados e tendo em conta que um dos factores críticos é o tempo de processamento, dado os robôs estarem constantemente em movimento, foi escolhido o valor de K=1.3. Assim, as trajectórias encontradas não são óptimas mas estão muito perto de o ser e o tempo de processamento é muito mais baixo do que para valores abaixo de 1.3. O valor de K=1.4, apesar de ter tempos de processamentos ainda melhores, já começa a perder qualidade na trajectória, por isso foi descartada essa hipótese.

### 3.7.2 Tamanho das células

O tamanho das células vai ter implicações na precisão da trajectória, no tempo de processamento e na precisão dos obstáculos. Quanto menor for o tamanho das células melhor vai ser a precisão, o que logicamente aumenta o número de células necessárias e, em consequência, o tempo de processamento aumenta. Em virtude disso é necessário arranjar um compromisso. Foram testados valores entre 0.01 m e 0.1 m, para robôs e obstáculos com diâmetros aproximados de 0.4 m.

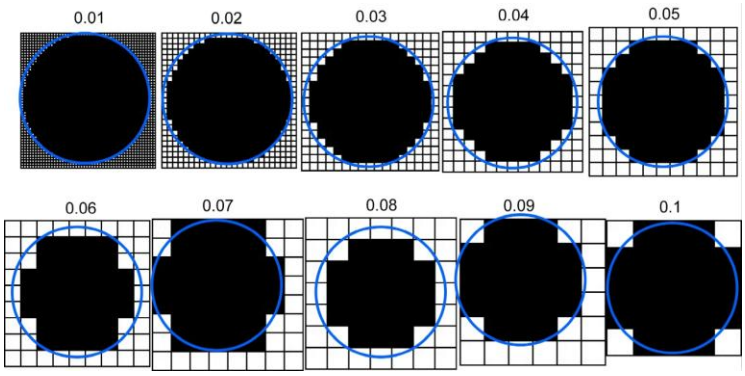



Figura 3.11 - Decomposição aproximada para células de tamanho fixo para valores entre 0.01 e 0.1

Na Figura 3.11 está representado um obstáculo para cada tamanho de célula e a circunferência azul indica o limite do obstáculo com raio igual à soma dos raios dos robôs, ou seja, indica como devia ser representado se fosse uma decomposição exacta. Pode-se verificar na figura que a partir de 0.04 m começa a ser muito difícil representar uma circunferência através do preenchimento de células, pelo que serão testados valores até 0.04 m.

Para os testes efectuados utilizou-se o caso número 3 da secção anterior, o caso do zigue-zague, com K=1.

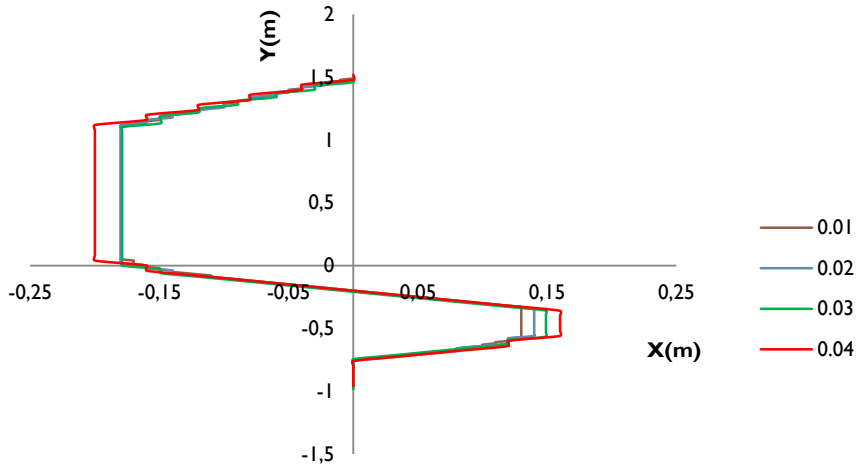



Gráfico 3.4 - Trajectórias da 3ª situação

Tabela 3.4 - Evolução do tempo de processamento e comprimento da trajectória em relação ao tamanho das células

| **Tamanho das Células** | 0.01 m | 0.02m | 0.03m | 0.04m |
|---|---|---|---|---|
| **Tempo de processamento** | 14.01 ms | 2.11 ms | 1.17 ms | 0.72 ms |
| **Comprimento** | 2.76 m | 2.76 m | 2.77 m | 2.80 m |

À medida que as células aumentam o tempo de processamento vai diminuindo e a sua trajectória vai ficando cada vez mais distante da óptima. Existe uma grande diferença entre passar o tamanho das células de 0.01m para 0.02m ao nível do tempo de processamento sem que se note prejuízo do comprimento da trajectória. Passar para 0.03m já implica um ligeiro prejuízo, mas continua-se a melhorar substancialmente no tempo de processamento. Mas ao passar para 0.04m, o ganho obtido no tempo de processamento começa a não compensar a degradação da trajectória. Como consequência escolheu-se o tamanho das células de 0.03m.

Estes valores encontrados para o tamanho das células são específicos para estes robôs e obstáculos, isto é, o tamanho é perfeitamente escalável para robôs maiores ou menores. Tal como foi referido, o tamanho das células influencia o número de células que por sua vez influencia o tempo de processamento e a precisão das trajectórias. Com robôs maiores, as células têm de ser maiores para o tempo de processamento não aumentar exponencialmente. Esta variável é outra das variáveis que é preciso ter muito cuidado e que influencia, em muito, a capacidade do algoritmo A*. Como se pode constatar, uma escolha errada do tamanho das células pode levar a ter-se uma implementação do algoritmo inviável.

## 3.8 Exemplo

Em seguida apresenta-se um pequeno exemplo de como funciona o algoritmo A* com seguintes características:

- Dimensão do $C_{espaço}$ de 7X6
- Ponto inicial na célula (2,3)
- Ponto destino na célula (5,2) de cor vermelho
- Obstáculos nas células (3,1), (4,1), (4,2), (4,3), (4,4) e (3,4)
- Heurística utilizada é a distância euclidiana com D = 1 e K = 1.3
- Células a verde significam que estão na lista aberta
- Células cinzentas significam que estão na lista fechada
- As setas dentro das células apontam para o seu Pai.
- G representa o valor do custo actual desde o início até à célula correspondente
- H representa o valor da função heurística de custo que faz a estimativa de ir da célula correspondente até ao ponto destino.
- F=G+H

O ponto inicial é adicionado à lista aberta e é calculado o F

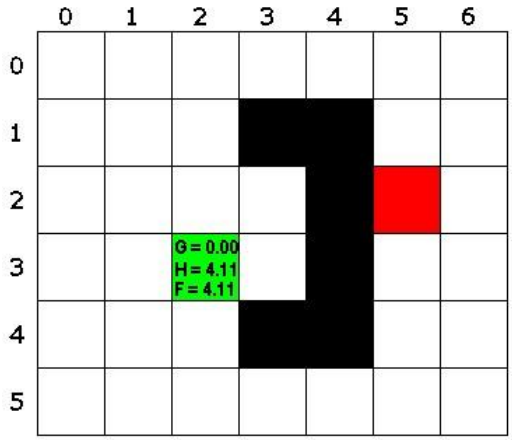



Figura 3.12 - $C_{espaço}$ com os valores de F,G e H no início

Tabela 3.5 - Listas aberta e fechada no início

| Lista Aberta | Lista Fechada |
|---|---|
| (2,3) | |

**Iteração 1**

O ponto inicial é removido da lista aberta e é inserido na lista fechada, uma vez que é a célula com melhor F.

Todas as células adjacentes, que não sejam obstáculos e que não estejam na lista fechada, são adicionadas à lista aberta ((1,2),(2,2),(2,3),(3,3),(2,4),(1,4) e (1,3)). São então calculados os F dessas células. Por exemplo para a célula (1,2) tem-se:

G = G(2,3) +c[(2,3),(1,2)] = 0 + 1.41=1.41

H = 1.3 x 4 = 5.2

F = G+H = 6.61

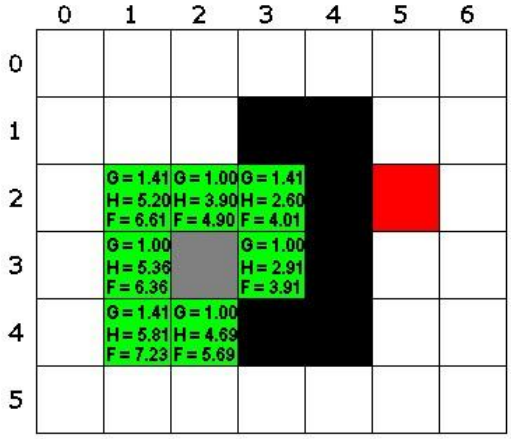



Figura 3.13 - $C_{espaço}$ com os valores de F,G e H 1ª iteração

Tabela 3.6 - Listas aberta e fechada na 1ª Iteração

| Lista Aberta | Lista Fechada |
|---|---|
| (1,2) | (2,3) |
| (2,2) | |
| (3,2) | |
| (3,3) | |
| (2,4) | |
| (1,4) | |
| (1,3) | |

**Iteração 2**

A célula (3,3) que tem o menor F é removida da lista aberta e inserida na lista fechada. Essa célula tem como Pai a célula (2,3).

Não existem células adjacentes que não sejam obstáculos e que não pertençam à lista fechada, por isso nenhuma célula é acrescentada à lista aberta.

As células adjacentes ((2,2),(3,2) e (2,4)) já pertencem à lista aberta. Em consequência vai-se verificar se o novo F é menor que o antigo F e no caso de tal se verificar actualiza-se o F e muda-se o Pai. Neste caso, as três células não são actualizadas. Por exemplo para a célula (2,2):

G = G(3,3) +c[(3,3),(2,2)] = 1 + 1.41=2.41

H = 1.3 x 3 = 3.9

F = G+H = 6.31

Este F é maior que o F antigo da célula (2,2) logo não é alterado

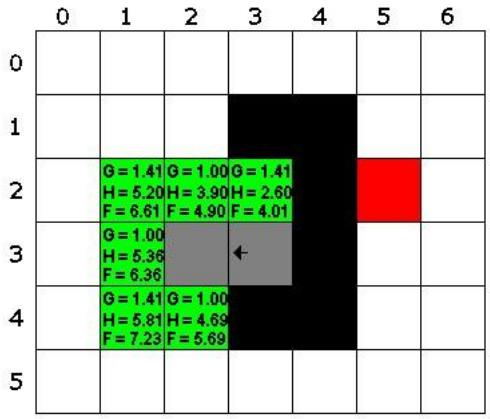



Figura 3.14 - $C_{espaço}$ com os valores de F,G e H 2ª iteração

Tabela 3.7 - Listas aberta e fechada na 2ª Iteração

| Lista Aberta | Lista Fechada |
|---|---|
| (1,2) | (2,3) |
| (2,2) | (3,3) |
| (3,2) | |
| (2,4) | |
| (1,4) | |
| (1,3) | |

**Iteração 3**

A célula (3,2) tem o menor F, logo é removida da lista aberta e inserida na lista fechada. Essa célula tem como Pai a célula (2,3).

A célula (2,1) é a única que não é obstáculo e não pertence a listas abertas ou fechadas, logo é inserida na lista aberta e calculado seu F.

A célula adjacente (2,2) já pertence à lista Aberta, mas não tem o F menor logo não existem actualizações.

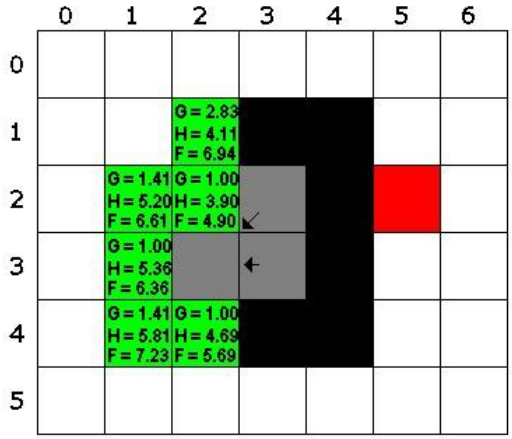



Figura 3.15 - $C_{espaço}$ com os valores de F,G e H 3ª iteração

Tabela 3.8 - Listas aberta e fechada na 3ª Iteração

| Lista Aberta | Lista Fechada |
|---|---|
| (1,2) | (2,3) |
| (2,2) | (3,3) |
| (2,4) | (3,2) |
| (1,4) | |
| (1,3) | |
| (2,1) | |

**Iteração 4**

A célula (2,2) com o menor F é removida da lista aberta e inserida na lista fechada. Essa célula tem como Pai a célula (2,3).

A célula (1,1) é a única que não é obstáculo e não pertence a listas abertas ou fechadas, logo é inserida na lista aberta e calculado seu F.

As células adjacentes ((1,2) e (1,3)) já pertencem à lista aberta, mas nenhuma tem o F menor logo não existem actualizações.

A célula adjacente (2,1) pertence à lista aberta e tem um F menor logo são actualizados os seus valores, tal como o seu Pai.

G = G(2,2) +c[(2,2),(2,1)] = 1 + 1 = 2

H = 4.11

F = G+H = 6.11

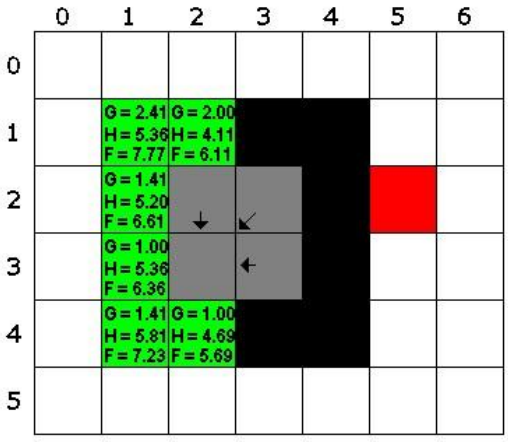



Figura 3.16 - $C_{espaço}$ com os valores de F,G e H 4ª iteração

Tabela 3.9 - Listas aberta e fechada na 3ª Iteração

| Lista Aberta | Lista Fechada |
|---|---|
| (1,2) | (2,3) |
| (2,2) | (3,3) |
| (2,4) | (3,2) |
| (1,4) | (2,2) |
| (1,3) | |
| (2,1) | |

Após a realização de mais dez iterações consegue-se encontrar o ponto destino.

**Iteração 14**

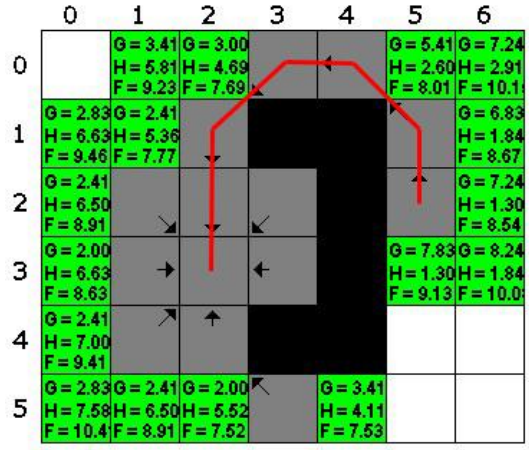



Figura 3.17 - $C_{espaço}$ com os valores de F,G e H 14ª iteração

Tabela 3.10 - Listas aberta e fechada na 14ª Iteração

| Lista Aberta | Lista Fechada |
|---|---|
| (1,1) | (2,3) |
| (2,5) | (3,3) |
| (1,5) | (3,2) |
| (1,0) | (2,2) |
| (2,0) | (2,4) |
| (0,2) | (2,1) |
| (0,3) | (1,3) |
| (0,4) | (1,2) |
| (0,1) | (3,0) |
| (4,5) | (3,5) |
| (0,5) | (1,4) |
| (5,0) | (4,0) |
| (6,0) | (5,1) |
| (6,1) | (5,2) |
| (6,2) | |
| (6,3) | |
| (5,3) | |

Na lista fechada podem-se verificar todas as células que foram processadas: houve 14 iterações e o tempo de processamento foi de 0.09 $ms^{-1}$. Para se obter o caminho é necessário ir à célula destino (5,2) e ver qual é o seu Pai, neste caso é a Célula (5,1); depois ir à célula (5,1) ver o seu Pai (4,0) e assim sucessivamente até chegar à célula inicial. Na Figura 3.17 o traço vermelho representa o caminho encontrado através do algoritmo. Este caminho tem como comprimento 6.82 m.

Em seguida são apresentados outros dois exemplos para mostrar a influência do K e do tamanho das células.

**Exemplo 1**

Com K = 1, na situação em que se garante que a heurística é admissível e consistente, verifica-se que o número de iterações aumenta para 18 e o tempo de processamento aumenta para 0.01 $ms^{-1}$. O comprimento do caminho encontrado é igualmente 6.82 m.

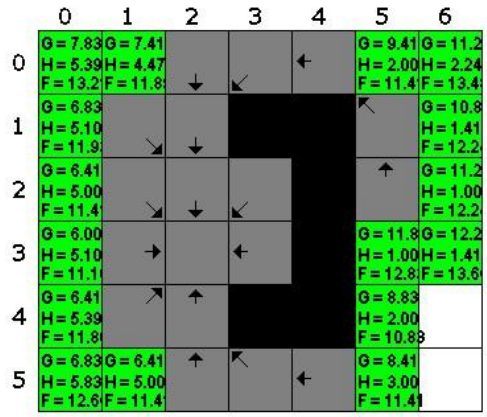



Figura 3.18 - $C_{espaço}$ com K=1

**Exemplo 2**

Diminuindo o tamanho das células para metade, passa-se a ter o quádruplo de células. Verifica-se, então, que o número de iterações aumenta para 38 e o tempo de processamento aumenta para 0.02 $ms^{-1}$. O comprimento do caminho encontrado é igualmente 6.82 m.

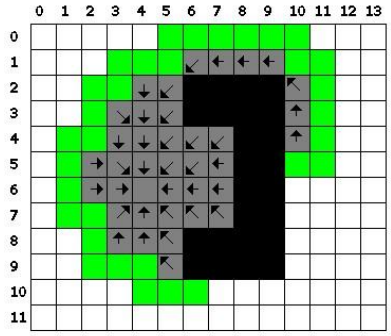



Figura 3.19 - $C_{espaço}$ com células com metade do tamanho

# Capítulo 4

# Plataforma de teste utilizada

Neste capítulo será apresentada a plataforma de teste utilizada nesta tese, o porquê da sua utilização e a descrição dos robôs construídos tanto ao nível de hardware como de software.

## *4.1* Robocup

O *Robocup[122]* é uma iniciativa internacional desenvolvida para promover a investigação e a educação na Robótica e na Inteligência Artificial. Através de alguns problemas comuns mas com grande complexidade, várias tecnologias são exploradas, inovadas e partilhadas através de encontros científicos anuais. O *Robocup* veio trazer referências científicas a estes domínios, ou seja, permite que as soluções e as técnicas empregues sejam mais facilmente analisadas e comparadas.

O problema inicial a tratar e com maior impacto é o jogo de futebol e através dessa escolha conseguiu-se incorporar vários factores que contribuíram decisivamente para o sucesso da iniciativa. Com o futebol robótico tem-se robôs a jogar futebol e várias áreas de investigação são incorporadas como: projecto de agentes autónomos, colaboração de multi-agentes, estratégia, computação em tempo-real, electrónica, microprocessadores, robótica, controlo, processamento de imagem, fusão sensorial e planeamento de trajectórias, entre outras. O futebol robótico serve também para educar e entreter o público em geral, dada a afinidade com o desporto rei a nível mundial, o que permite uma maior aproximação da tecnologia com um público indiferenciado. Outro dos aspectos a ter em conta é que o factor competição implica um maior progresso. A competição permite e incentiva a troca de ideias, o que garante, em última instância, uma maior rapidez e eficácia na resolução de problemas. Em consequência destas trocas de conhecimentos e experiências, as tecnologias e algoritmos resultantes são usados depois nas várias áreas da indústria ou na resolução de outros problemas completamente diferentes.

A federação do *RoboCup* definiu a seguinte meta para a iniciativa inicial: “No ano de 2050, uma equipa de robôs autónomos humanóides, ser capaz de vencer a equipa humana campeã do mundo de futebol de 11”. Este desiderato, hoje em dia, está ainda muito longe de acontecer, embora tenham sido dados passos importantes nesse sentido, de modo que algum dia seja possível que essa meta venha a ser alcançada.

Desde 1998 que uma equipa de docentes, estudantes e investigadores da Faculdade de Engenharia da Universidade do Porto (FEUP) projecta e constroi robôs para participar na Liga pequena, formando a equipa 5DPO, e têm como objectivo o desenvolvimento de tecnologias e algoritmos para a área da robótica. Ao longo dos anos vários projectos foram desenvolvidos e não só referentes à liga pequena, mas também à liga média, à simulação, aos humanoides e à liga normalizada.

Desde 2000 que as Universidades do Porto e de Aveiro uniram esforços e criaram a equipa FCPortugal dedicada à simulação. Essa cooperação foi reforçada a partir de 2009 com a criação do FCPortugal SPL para participar na Standard Platform League. Em consequência desses projectos foram publicados trabalhos de vária ordem, como *papers*, teses e relatórios de projectos industriais desenvolvidos. As áreas em que esses trabalhos produziram impacto foram as seguintes:

- Sistemas multi-agentes e coordenação em sistemas de multi-agentes.[123-130]
- Futebol robótico e cooperação entre robôs[131-140]
- Simulação baseada em agentes[141-145]
- Análise de jogo, raciocínio estratégico e táctico de modelos[146-148]
- Monitorização de multi-agentes e de multi-robôs[149, 150]
- Fusão sensorial[123, 151-154]
- Processamento de imagens em tempo real[155-161]

O reconhecimento do trabalho desenvolvido foi igualmente validado através das várias participações honrosas nas diversas competições ao longo dos anos. Desde 1998, ano em que a equipa 5DPO iniciou a sua participação na iniciativa Robocup, foi alcançado em dois anos o 2º lugar no campeonato da Europa ( *German Open*) e em três anos o título de campeão da Europa. Em relação aos campeonatos do Mundo, foi alcançado um 5º, um 3º e um 2º lugares. Em Portugal não existe mais nenhuma instituição do ensino superior nesta área de competição (liga pequena).

A FC Portugal, das Universidades do Porto e de Aveiro, teve prestações brilhantes ao longo dos anos. Foi campeão do mundo, em 2000 e campeão europeu em 2000 e 2001 na simulação 2D. Foi campeão do mundo em 2002, 2º em 2003 e 2004 na simulação de treinador. Na simulação de resgate foi campeão europeu em 2006. E na simulação 3D foi campeão do mundo em 2006 e campeão da Europa em 2006 e 2007.

A equipa FC Cambada da Universidade de Aveiro dedica-se à Liga dos médios e desde 2003 conseguiram ser campeões do mundo em 2008 e ficar em 3º lugar em 2009, 2010 e 2011.

### *4.1.1 RoboCup Soccer* (Futebol Robótico)

O *RoboCup Soccer* está dividido em várias ligas, sendo que essa divisão visa colocar um conjunto de desafios próprios a cada uma delas. Isto é, apesar de o objectivo principal em cada liga ser o mesmo - ter uma equipa de robôs para ganhar um jogo de futebol - em diferentes ligas, temos diferentes problemas que determinam diferentes resultados de sucesso.

#### 4.1.1.1 Liga de simulação

Jogo de futebol simulado num computador, entre duas equipas de onze elementos cada. Esta liga visa particularmente os sistemas multi-agentes, estratégia e a Inteligência Artificial. Existem duas subligas: 2D, 3D.


Figura 4.1 - a) Simulação em 2D b) Simulação em 3D

#### 4.1.1.2 Small-size robot league (SSL) (Liga de robôs pequenos)

Nesta liga os robôs têm menos de 18 cm de diâmetro e jogam num campo de 6.05 m x 4.05 m. Cada equipa tem até 5 robôs. Esta liga visa essencialmente o controlo rápido e preciso, em que os robôs e a bola podem atingir velocidades elevadas, actualmente na ordem dos 2 a 3 $ms^{-1}$. Esta liga é a plataforma de teste utilizada neste trabalho.

#### 4.1.1.3 Middle-size robot league (MSL) (Liga dos robôs médios)

Nesta liga jogam os chamados robôs médios, que não podem ter mais que 50 cm de diâmetro, num campo de 12 m x 18 m. Cada equipa tem até 5 robôs. Nesta liga cada robô é autónomo, ou seja, o robô tem os sensores e um computador incorporado e usam a rede *wireless* para comunicar entre si. Nesta liga, o essencial para um bom desempenho é a parte electromecânica e uma boa capacidade de localização do robô no campo de futebol.

Figura 4.2 - a) Small Size League b) Middle Size League

#### 4.1.1.4 Standard League (Liga normalizada)

Nesta liga todas as equipas usam o mesmo robô e, como consequência, o software é que é desenvolvido. Não existe controlo externo, isto é, os robôs são autónomos. Inicialmente os robôs eram os *four-legged Sony AIBO* e desde 2008 passaram a ser os *humanoid Aldebaran Nao*. Nesta liga o essencial é o controlo dos movimentos dos robôs bípedes.

#### 4.1.1.5 Humanoid league (Liga Humanóide)

Esta liga utiliza robôs bípedes, construídos pelas equipas, que concorrem em duas subcategorias: *Kid-size e Teen-size*. A distinção entre estas categorias tem a ver com o tamanho dos robôs. Além do domínio do controlo dos movimentos dos robôs, é essencial nesta liga haver uma boa mecânica e electrónica.

Figura 4.3 - a) Standard League b) Humanoid League

Com o sucesso do *Robocup Soccer* começaram a aparecer novos desafios para novos domínios e tecnologias, tais como:

### *4.1.2 RoboCupRescue* (Liga de Resgate)

Com este projecto, que consiste no resgate de pessoas e bens em desastres naturais, são envolvidos um elevado número de pessoas e organizações num ambiente desconhecido e perigoso.

A pesquisa neste projecto envolve vários domínios científicos tais como coordenação em equipa de multi-agentes, construção de robôs para procura e resgate, informação de infra-estrutura, simuladores e sistemas de apoio à decisão. Esta categoria está dividida em duas ligas:

#### 4.1.2.1 Rescue Robot League (Liga de robôs para resgate)

Nesta liga é necessário construir os robôs de modo a que consigam movimentar-se nos mais variados locais, muitas vezes em ambientes de muito difícil acesso. À medida que os robôs vão avançando, vão dando informação do local e de potenciais perigos.

#### 4.1.2.2 Rescue Simulation League (Liga de simulação para resgate)

Esta liga visa as situações de emergência em que é necessário prever, planear e coordenar buscas e as diversas cooperações humanas. Com esta liga algumas áreas da Inteligência Artificial são exploradas, tais como planeamento multi-agente, planeamento em *realtime/anytime*, agentes heterogéneos, planeamento robusto.

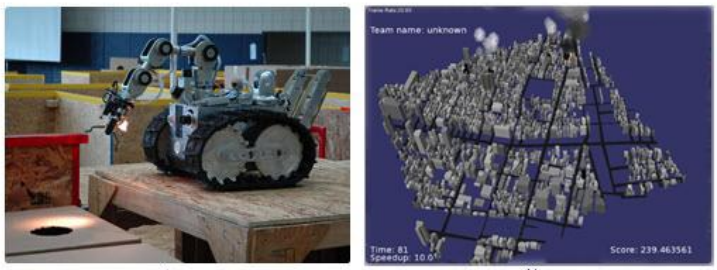



Figura 4.4 - a) Rescue Robot League b) Rescue Simulation League

#### 4.1.2.3 RoboCup@Home

Esta liga baseia-se em aplicações do chamado dia-a-dia do Homem e tem por objectivo promover o desenvolvimento de robôs que vão ajudar dentro de uma habitação. Os seguintes domínios vão ser aperfeiçoados e testados: cooperação e interacção humano-robôs, navegação e mapeamento em ambientes dinâmicos, visão, reconhecimento de objectos, manipulação de objectos, comportamentos adaptativos e de integração.

Figura 4.5 – Robocup@Home

### 4.1.3 RoboCupJunior

É um projecto que visa promover a robótica educativa para crianças e jovens. É-lhes dada a possibilidade de com alguns recursos começarem a estar envolvidas em projectos com electrónica, software, hardware e trabalho em equipa. No âmbito deste projecto criaram-se três competições: Futebol, Dança e Resgate.


Figura 4.6 - RoboCupJunior a) Dança b) Futebol

## 4.2 Liga Pequena (SSL)

Esta dissertação está inserida no projecto de uma equipa que visa a construção dos robôs para a liga pequena.

Nesta liga há duas equipas que jogam num campo (Figura 4.7), sendo que cada uma delas tem uma ou mais câmaras de filmar por cima do campo. A informação de vídeo é enviada para um computador que processa a imagem de modo a retirar os dados das posições e velocidades de cada robô e da bola. De seguida o computador envia essa informação para outro computador que faz de treinador que por sua vez envia as ordens para os robôs através de frequência de rádio. Tal como nas outras ligas, não há intervenção humana durante o decorrer do jogo e, consequentemente, os robôs devem ser capazes de evoluir em campo de uma forma perfeitamente autónoma.

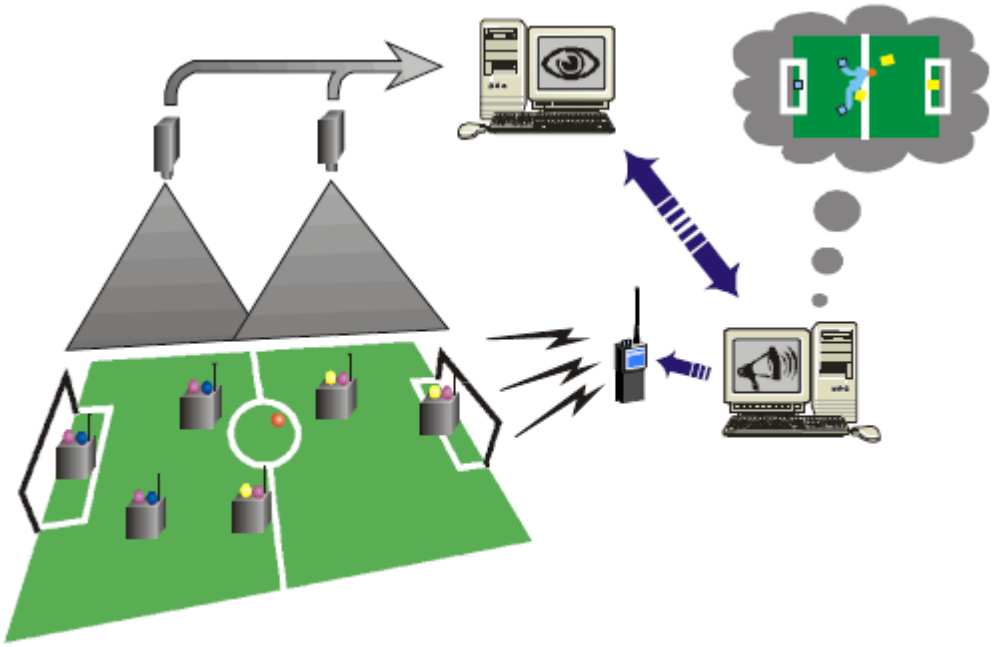


Figura 4.7 – Sistema de uma equipa de SSL

As principais regras desta liga para 2010 são as seguintes:

- As dimensões do campo são de 6,05m por 4,05 m
- A bola é uma bola de golfe cor de laranja
- Cada equipa no máximo tem 5 robôs em que um é o guarda-redes (este robô tem funções específicas e diferenciadas dos restantes)
- O robô não pode exceder a largura de um cilindro de 18cm de diâmetro e a altura de 15cm.
- Existe um árbitro principal humano e um auxiliar que controla a *Referee Box*: trata-se de uma aplicação informática responsável por enviar os comandos do árbitro para os computadores das equipas que estão a jogar e também por fazer *logs* dos jogos.
- Cada partida tem a duração de 20 minutos dividida em 2 partes.
- As regras do jogo seguem as regras da FIFA para o futebol de 11, com algumas excepções, como por exemplo não existir fora-de-jogo.

### 4.2.1 Robôs

Ao longo dos vários anos do projecto, diferentes robôs foram projectados e construídos pela equipa 5DPO. Quase todos os anos foi necessário projectar e construir novos robôs. Em algumas situações tornou-se necessário fazer actualizações profundas. Essa premência deveu-se à tremenda evolução existente. Para se conseguir acompanhar a permanente transformação dos robôs das outras equipas e mostrar-se à altura em momentos de competição foi necessário melhorar constantemente aspectos como a velocidade e a aceleração dos robôs. Nos robôs de 1998 que nos valeram um honroso 3º lugar no campeonato do mundo, por exemplo, a velocidade máxima dos nossos robôs era de 20 $cms^{-1}$. Até 2007, ano em que a equipa ficou em 2º no campeonato do mundo, os robôs tiveram de evoluir para uma velocidade máxima de 1,5 $ms^{-1}$ . Na Figura 4.8 tem-se a noção da evolução física dos robôs ao longo do tempo.


Figura 4.8 - Evolução dos Robos desde 1998 até 2007

Nas seguintes secções serão apresentados os robôs nos quais foram feitos os testes apresentados na tese, e que são a última evolução projectada e construída no laboratório de robótica do Departamento de Engenharia Electrotécnica e de Computadores (DEEC) da FEUP.

### 4.2.2 Hardware

Ao nível do hardware os robôs são constituídos por 3 rodas omnidireccionais, acopladas a 3 motores *brushless* de 12 W (*EC 45 Flat motor da maxon motors*). Graças aos sensores *Hall* disponíveis nos motores fica associado a cada roda um *encoder*. Cada robô tem uma placa de controlo onde está presente o sistema de controlo para os três motores e a bomba elevadora de tensão para o condensador que alimenta os *kickers* (dispositivos de chuto da bola) assim como os *drives* das bobines dos dois *kickers*. Há ainda uma placa de comunicação onde estão presentes dois emissores-receptores de rádio que permitem a comunicação entre os robôs e o PC exterior que comanda a movimentação dos robôs. Adicionalmente, há um sensor de infravermelhos que permite um canal de comunicações adicional e independente do rádio e ainda uma barreira infravermelha que detecta a proximidade da bola ao *kicker* e pode ser usado para o disparar.

Os robôs, em si mesmos, não são autónomos: simplesmente executam comandos vindos por rádio para controlar a velocidades dos motores e os *kicks* (chutos na bola).

#### 4.2.2.1 Roda-motor-*encoder*

Os três conjuntos roda-motor-*encoder* (Figura 4.9) estão conectados a uma placa de controlo que implementa um controlador PID da velocidade de rotação do motor, recorrendo a um microcontrolador de 8 bits (*ATMega8 da Atmel*). A placa de controlo inclui também os *drivers* de potência com limitação de corrente para os motores. O microcontrolador mede as velocidades das rodas recorrendo aos sinais dos sensores de efeito de *Hall* que são também usados para comutar as fases dos motores *brushless*. A redução entre o motor e a roda é conseguida por um conjunto de engrenagens que implementam uma redução de 1/10.

Figura 4.9 - Roda –moto-*encoderr*

#### 4.2.2.2 Baterias

A solução encontrada para a alimentação dos robôs foi a utilização de dois pacotes de baterias com 5 elementos cada que fornecem 6 *volts* cada um. Estes elementos estão ligados em série para fornecer 12 *volts* para a alimentação dos motores e do sistema de carga dos condensadores dos *kickers*. Tendo sido considerado que a electrónica tem um consumo reduzido face aos outros consumos, o sistema electrónico é alimentado a 5V através regulador *low-drop* (MAX883) a partir de um único pacote de baterias.

#### 4.2.2.3 Kicker

Os dispositivos de chuto (Figura 4.10) são baseados num solenóide que num caso puxa uma pá que imprime um movimento ascendente à bola e no outro caso empurra um batente que, atingindo a bola, lhe dá um movimento horizontal. Este dispositivo de chuto utiliza energia acumulada em condensadores para um impulso com força variável sobre a bola, conforme o tempo que o solenóide está alimentado.

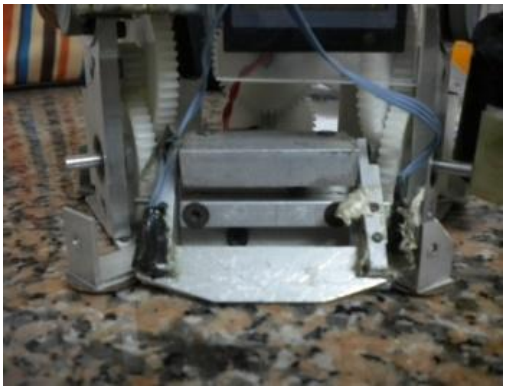


Figura 4.10 - Kicker

#### 4.2.2.4 Rádio

Os regulamentos da SSL impõem que os robôs sejam capazes de comunicar em duas frequências distintas. Isso é conseguido recorrendo a dois módulos de rádio, BIM433 e BIM418, que operam respectivamente nas frequências de 433MHz e 418MHz (Figura 4.11). O microcontrolador de 8 bits (ATMega8 da Atmel) responsável por implementar o canal de comunicações pode seleccionar um ou outro canal, ou ainda se os activa simultaneamente. Essa capacidade adicional permite, quando é possível recorrer a ela, aumentar a fiabilidade da comunicação pois minimiza o efeito de interferências externas. O protocolo implementado permite minimizar a latência da comunicação pois foi optimizado para esta aplicação.


Figura 4.11 - Rádio

### 4.2.3 Software

Ao nível do software, existem dois programas a funcionar em paralelo no decorrer do jogo, o programa da visão e o programa da decisão.

No programa de visão o objectivo é receber os sinais de vídeo das duas câmaras (solução adoptada pela equipa 5DPO) e processar a informação de modo a poder retirar os dados necessários para o outro programa, a saber, qual é a posição dos robôs e da bola. A bola é facilmente encontrada na imagem, visto que é o único objecto da cor laranja. Os robôs são encontrados sem ambiguidades,

uma vez que cada equipa tem nos seus robôs uma parte de cima especial e diferente. Uma equipa tem um círculo no topo dos robôs da cor azul e a outra equipa de cor amarela. Cada equipa pode usar ainda uma estratégia adicional para identificar individualmente cada um dos robôs. O método de identificação usado pela equipa 5DPO foi a posicionamento de seis círculos à volta do círculo central, como mostra a Figura 4.12, que podem ser da cor verde ou rosa e a sequência deles permite não só identificar o robô, mas também orientar o robô, isto é, ficar a saber para que lado está virada a parte da frente do robô.

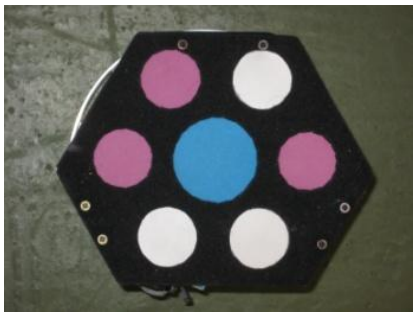


Figura 4.12 - Topo do Robô

O sistema de visão actual é uma herança que tem as suas origens no trabalho desenvolvido em 1998. Desde essa altura este sistema tem vindo a ser aperfeiçoado, de modo a que se possa ter a melhor precisão possível. Actualmente, o erro existente no sistema de visão da equipa 5DPO é inferior a 1 cm em todo o campo. É igualmente importante conseguir o máximo de *frames* possíveis processadas de modo a que o atraso seja minimizado. Isto é, como os robôs e a bola se movem a velocidades na ordem dos 2 a 3 $ms^{-1}$ é importante ter o programa, no mínimo, a processar a 25 *frames* por segundo. Entre cada *frame* uma bola pode ter-se movido 12 cm, o que faz que com um atraso de 40 ms, o erro passe a ser significativo.

No programa de visão (Figura 4.13) pode-se calibrar as cores e ajustar as dimensões do campo e dos robôs. Quando o sistema de visão já tiver identificados os dados pretendidos (posição dos robôs e da bola) envia por UDP para o sistema de decisão essa informação.

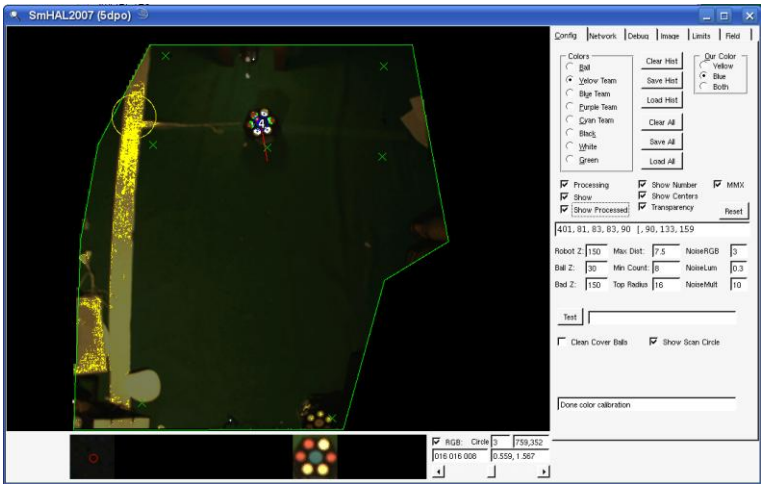



Figura 4.13 - Programa de visão

O objectivo do programa de decisão é receber o posicionamento dos robôs e da bola, e, com base nessa informação, escolher a melhor estratégia para a continuação do jogo. Isto é, o programa deve escolher quais são os pontos no campo para onde devem ir cada um dos robôs. Cabe assim ao programa de decisão decidir se um determinado robô vai ter com a bola ou se deve defender uma linha de passe, se deve cobrir a baliza ou ir para um local que dê linha de passe a outro robô. Depois de saber para onde devem de ir os robôs, o programa de decisão manda via rádio para cada um deles as velocidades que cada roda deve ter. A cada 40 ms é efectuado novo cálculo de todo o processo, ou seja, o programa de decisão recebe os dados de posicionamento novamente e tenta achar a melhor estratégia para esses novos pontos.


Figura 4.14 - Programa de decisão

O sistema de decisão está dividido em 3 níveis ao nível de arquitectura, tal como mostra a Figura 4.15.

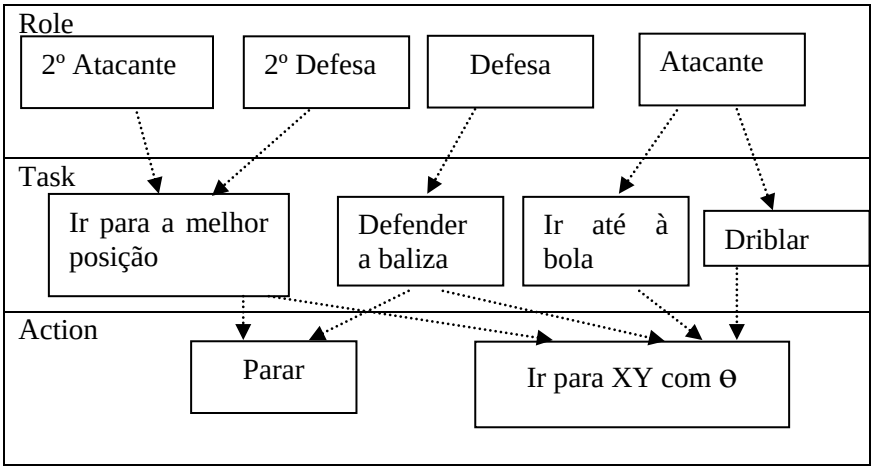



Figura 4.15 - Arquitectura do sistema de decisão

Na camada superior está o "*role*" que define o papel de cada robô. Os papéis possíveis são:

Tabela 4.1 - Papel de cada robô

| | |
|---|---|
| Atacante | Único papel que procura a bola e remata |
| Defesa | Serve para defender o caminho da bola para a baliza |
| 2º Defesa | Serve para defender os passes |

| | |
|---|---|
| 2º Atacante | Serve para se posicionar no espaço vazio |
| Marcador de penalidades | Serve para marcar grandes panalidades |

O programa de decisão vai procurar o melhor robô para fazer de atacante e o melhor robô para fazer de defesa e assim sucessivamente. Os robôs podem mudar de "*role*" a qualquer instante, num momento um robô pode ser defesa e no momento seguinte passar a ser atacante. O único robô que não muda de "*role"* é o guarda-redes que é sempre o mesmo robô. A procura do melhor robô para cada papel, segue sempre a mesma ordem. Em primeiro lugar descobre-se o robô melhor posicionado para ser atacante. Este processo de descoberta depende do papel anterior de cada robô e principalmente do caminho mais curto para chegar à bola. Em seguida procura-se o melhor robô para fazer de defesa. Depois escolhe-se o robô para fazer de 2º defesa e por fim o robô que faz de 2º atacante.

Após se terem atribuído os papéis a cada robô, o programa de decisão deve passar à camada seguinte e determinar a "*task*" de cada um, a tarefa que cada um deve executar. A cada papel podem estar associadas várias tarefas:

| | |
|---|---|
| Atacante | Interceptar a bola<br>Ir até à bola<br>Driblar com a bola |
| Defesa | Defender a baliza |
| 2º Defesa | Ir para a melhor posição |
| 2º Atacante | Ir para a melhor posição |

Na última camada do programa de decisão, após se ter decididoo a *task* de cada um dos robôs, escolhe-se qual a "*Action*" que cada robô deve executar para se cumprir cada "*task*". Existem duas "*Actions*": a acção de Stop, ou seja simplesmente parar, e a acção de ir para um determinado ponto XY, com um determinado ângulo ɵ. Nesta última acção são calculadas quais as velocidades das rodas a mandar para o robô.

## 4.3 Conclusão

Ao longo deste capítulo descreveu-se o ambiente de estudo deste trabalho. É um ambiente dinâmico, em que existe dez robôs, divididos em duas equipas, a jogar futebol num espaço limitado. Ou seja, temos um ambiente conhecido, em que conhecemos as posições dos obstáculos e as suas velocidades em cada instante, sendo que esses obstáculos estão sempre em movimento imprevisível, a velocidades que podem ser elevadas.

A análise da estrutura do sistema de decisão planeia 10 trajectórias distintas de modo a atribuir os papéis aos diferentes robôs, definidos para cada instante do jogo. Para decidir qual é o robô atacante calculam-se os caminhos para cada um dos diferentes robôs da equipa (que são 4) de modo a chegar à bola. Em seguida calculam-se os caminhos para os restantes 3 robôs de modo a ver qual é o melhor robô para ficar como defesa. Depois calcula-se o caminho para decidir qual o robô que irá para a posição de 2º defesa e por fim o caminho para o robô que será o 2º atacante.

Ao serem necessários 10 planeamentos de trajectórias, o tempo de cálculo dessas trajectórias passa a ser crítico. Esse tempo vai entrar nas contas para o cálculo do atraso entre a imagem captada e os sinais de velocidade a chegarem às rodas. Por isso, o tempo de cálculo do algoritmo de planeamento de trajectórias passa a ser um dos aspectos muito importantes a ter em consideração.

# Capítulo 5

# Simulador

Neste capítulo serão explicadas as razões de se utilizar um simulador. De seguida o simulador utilizado será descrito bem como o modelo do robô. Na parte final do capítulo será identificado os parâmetros do robô simulado e validado o simulador.

A introdução de um simulador neste trabalho tem como principal objectivo ter um ambiente estável, que não afecte o que se está a testar [162]. Para ter um ambiente estável é importante que todos os factores envolventes não se alterem, de modo a não afectarem os resultados. Vários factores podem perturbar este ambiente estável, desde o desgaste mecânico do robô, passando pelas perturbações da visão (condições diferentes de luminosidade em dias diferentes podem afectar a recolha dos valores de posicionamento de cada um dos intervenientes) e principalmente o aspecto mais crítico de utilizar robôs reais é a sua fonte de alimentação, isto é, as baterias à medida que se vão descarregando vão influenciando o desempenho do robô. Essa descarga implica que as velocidades começam a diminuir, o que implica que seja necessário manter as baterias sempre a um nível constante. Isso implica também que se gaste muito tempo a carregá-las. O tempo necessário para o *setup* de cada teste com robôs reais contribui também para que o simulador seja muito mais rápido. Logo o tempo também é um factor a ter em conta, é muito mais rápido fazer testes num simulador do que nos robôs reais.

Por isso nesta dissertação foi utilizado um simulador, mas primeiro foi validado, como se verá neste capítulo. Os restantes testes foram feitos no simulador.

## 5.1 Simulador

O simulador utilizado foi o SimTwo[163], que foi criado pelo Prof. Dr. Paulo Costa, um docente da Faculdade de Engenharia da Universidade do Porto, que tem como objectivo ser um sistema de simulação realista onde fosse possível implementar vários tipos de robôs:

- Robôs móveis com diferentes configurações:
  - Diferenciais;
  - Com rodas omnidirecionais.
- Robôs Manipuladores;
- Robôs Quadrúpedes;

- Robôs Humanóides;
- Qualquer tipo de robô terrestre que possa ser descrito como uma mistura de articulações rotativas e rodas clássicas ou omnidirecionais;
- Veículos mais-leves-que-o-ar com ou sem hélices para propulsão.

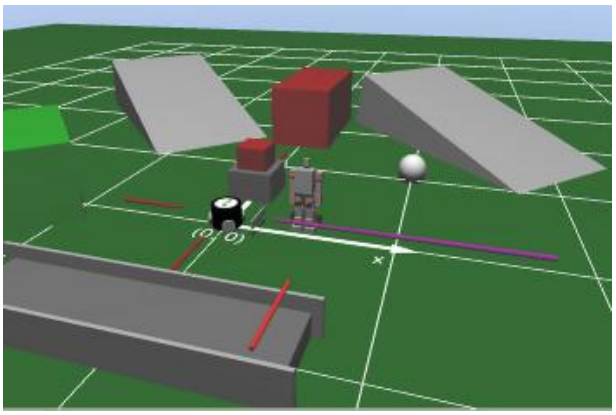


Figura 5.1 - Simulador SimTwo

A principal preocupação do simulador é a parte realista ao nível da dinâmica. Outra parte simulada com elevado realismo são os motores e respectivos controladores. Estes modelos tentam capturar elementos não lineares que estão invariavelmente presentes nos motores de todos os robôs.

Na representação 3D do simulador não existe uma grande preocupação com foto realismo, serve essencialmente para permitir uma observação do comportamento do sistema em tempo real.

## 5.2 Componentes do robô

Esse realismo implementado no SimTwo é obtido decompondo o robô num sistema de corpos rígidos, articulações, sensores e sistemas de accionamento (os motores eléctricos). Cada uma das suas componentes tem as suas propriedades associadas.

Os corpos rígidos são expressos através das suas propriedades físicas:

- Forma;
- Massa;
- Momentos de inércia;
- Atritos das superfícies;
- Elasticidade.

Certas articulações típicas: eixo, *cardan* e calha são definidos explicitamente e podem ter associados um sistema de accionamento e sensores.

O sistema de accionamento pode ser constituído por um motor DC, opcionalmente com caixa redutora, e um controlador. O controlador pode ser de vários tipos: um PID aplicado ao sinal de posição ou de velocidade ou uma realimentação de estado.

Em relação aos motores, de modo a recriar o máximo de realismo, vários elementos lineares e não lineares são definidos. As propriedades que se podem definir são:

- Resistência dos enrolamentos do motor;
- Indutância dos enrolamentos do motor;
- Constante de binário do motor;
- Tensão máxima aplicável ao motor;
- Corrente máxima aplicável ao motor.

As rodas são definidas através das suas propriedades físicas e de dois coeficientes de atrito. Têm as seguintes propriedades:

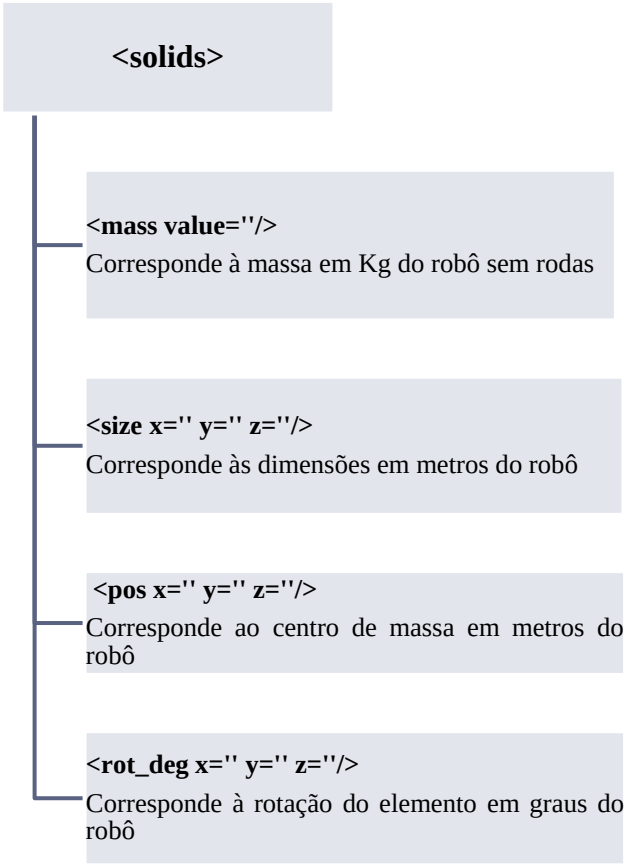
- Massa da roda;
- Raio da roda;
- Largura da roda;
- Distância do centro da roda ao centro do robô;
- Ângulo do eixo da roda à frente do robô;
- Constante de atrito viscoso no eixo da roda;
- Binário de atrito de *Coulomb* no eixo da roda;
- Atritos da roda com o chão segundo as distâncias longitudinais e radiais.

## 5.3 Elementos base dos robôs

No simulador SimTwo para definir o modelo do robô utiliza-se uma sintaxe XML e guarda-se essa informação num ficheiro. Esse modelo é construído especificando uma série de elementos base que irão definir a sua forma e dinâmica. Esses elementos são os seguintes:

| | |
|---|---|
| **<solids>** | Representam um elemento físico com uma dada forma (Paralelepípedo, Esfera, Cilindro) |
| **<wheels>** | Conjunto Eixo com motor, controlador e *encoder* ligado a um cilindro |
| **<shells>** | Conjunto de placas que formam a carapaça do robô. Com elas pode-se dar uma forma mais complexa ao robô. Servem, neste caso, principalmente para resolver uma limitação do motor de dinâmica de corpos rígidos usado (ODE) que não implementa colisões entre cilindros. |

Dentro do elemento <solids> encontram-se os seguintes elementos e respectivos parâmetros:



Dentro do elemento <wheels> encontram-se os seguintes elementos e respectivos parâmetros:

**<wheels>**

**<omni>**
**<tyre mass='' radius='' width='' centerdist=''/>**
Corresponde a uma roda omnidireccional com os parâmetros:
- *mass corresponde à massa da roda, em kg.*
- *radius e width correspondem, respectivamente, ao raio e à espessura da roda, em metros.*
- *centerdist corresponde à distância da roda ao centro do robô, em metros.*

**<motor ri='' li='' ki='' vmax='' imax='' active=''/>**
Corresponde ao motor DC com os parâmetros:
- *ri* corresponde à resistência interna, *em* Ω
- *li* corresponde à indutância dos enrolamentos, *em H.*
- *ki* corresponde à constante que relaciona o binário produzido pelo motor com a corrente que por ele passa, *em Nm/A.*
- *vmax* corresponde à tensão máxima no motor, *em V*
- *imax* corresponde à corrente máxima permitida pelo motor, *em A*

**<gear ratio=''/>**
Corresponde a uma caixa redutora em que o parâmetro *ratio é a relação de transmissão.*

**<controller mode='' kp='' ki='' kd='' kf='' active='' period=''/>**
Corresponde ao controlador com os seguintes parâmetros:
- *mode diz respeito ao tipo de controlador*
- *kp, ki, kd e kf correspondem aos ganhos do controlador*
- *period corresponde ao período de amostragem do controlador*

**<friction bv='' fc=''/>**
Corresponde ao modelo de atrito dinâmico e estático não linear associado a cada eixo com os seguintes parâmetros:
- bv corresponde à constante de atrito viscoso, em Nm/rad/s
- fc corresponde ao binário de atrito de *Coulomb Ctg (em Nm)*

Dentro do elemento <shells> encontram-se os seguintes elementos e respectivos parâmetros:

**<shells>**

**<size x='' y='' z=''/>**
Corresponde às dimensões, em metros

**<pos x='' y='' z=''/>**
Corresponde ao posicionamento, em metros

**<rot_deg x='' y='' z=''/>**
Corresponde à rotação do elemento em graus

## 5.4 Modelo do Robô

Para construir o modelo do robô utilizado na simulação usou-se um cilindro para a parte do corpo; as rodas utilizadas são omnidireccionais com um controlador PID de velocidade; e cuboídes como superfícies usadas para representar superfícies de contacto.

### 5.4.1 Corpo

Em relação ao corpo do robô, os valores utilizados foram os medidos do robô real:, o peso do robô sem rodas é de 1.6 Kg, o seu tamanho tem um diâmetro igual a 0.15 m e altura de 0.135 m. Para calcular o centro de massa efectuou-se uma pequena experiência com o objectivo de determinar a altura do centro de massa do robô. O robô foi colocado na horizontal, sobre uma barra de suporte, como mostrado na Figura 5.2. Quando o robô se encontra na horizontal e equilibrado, foi medida a distância entre a barra de suporte e a base do robô. Assim encontra-se a altura do centro de massa que é 6 cm.

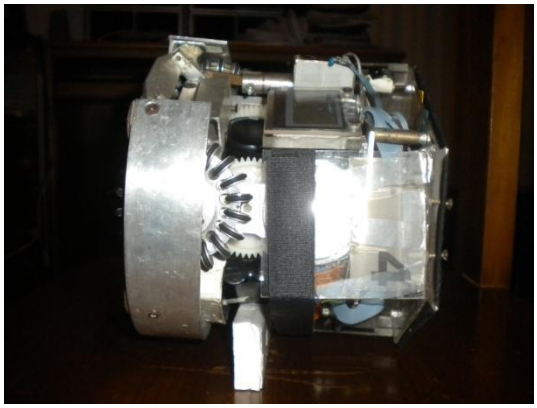


Figura 5.2 - Experiência para calcular o centro de massa

A parte do corpo fica com a seguinte configuração:

```
<solids>
  <cylinder>
    <mass value='1.6'/>
    <size x='0.075' y='0' z='0.135'/>
    <pos x='0' y='0' z='0.06'/>
    <rot_deg x='0' y='0' z='0'/>
  </cylinder>
</solids>.
```

### 5.4.2 Sistema de actuação

Esta secção engloba as rodas e os motores, com as respectivas caixas redutoras e controladores. O modelo é constituído por três rodas omnidireccionais desfasadas entre elas de 120º.

```
<omni/>
  <wheel>
    <axis angle='-60'/>
  </wheel>
  <wheel>
    <axis angle='60'/>
  </wheel>
  <wheel>
    <axis angle='180'/>
  </wheel>
```

Os valores dos parâmetros das rodas utilizados foram os valores medidos do robô real:o peso de cada roda é de cerca de 0.195 Kg, o raio é 0.0325 m, com a largura de 0.015 m e que dista do centro do robô 0.07 m.

*<tyre mass='0.195' radius='0.0325' width='0.015' centerdist='0.07'/>*

Os motores são DC e para ponto de partida os valores da resistência interna e constante $k_i$ foram retirados da especificação técnica do motor real. Esses valores foram respectivamente 1.73 Ω e 0.0104 Nm/A. O valor de $V_{max}$ corresponde à tensão de alimentação utilizada, isto é, 12 V e o valor de $I_{max}$ corresponde ao limite de corrente imposto pelo controlador usado no motores reais que é de 2 A.

*<motor ri='1.73' ki='1.04e-2' vmax='12' imax='2' active='1'/>*

A caixa redutora tem uma relação de 1/10, ou seja, para cada volta da roda é preciso dar 10 voltas no motor.

*<gear ratio='10'/>*

O controlador implementado no robô real é um controlador PID discreto que visa manter a velocidade das rodas no valor de referência. No simulador existe também este tipo de controlador, mas os parâmetros são ligeiramente diferentes dos parâmetros do controlador implementado no robô real. De seguida será explicada a relação entre os parâmetros de ambos os controladores.

O controlador real tem a seguinte configuração:

$$V_{out} = K_p \varepsilon + K_i \int \varepsilon dt + K_d \frac{d\varepsilon}{dt} + K_f \omega_{ref} \tag{5.1}$$

$$\varepsilon = \omega_{ref} - \omega \tag{5.2}$$

O controlador simulado apresenta a seguinte configuração:

$$V_{out} = K'_p \varepsilon' + K'_i \int \varepsilon' dt + K'_d \frac{d\varepsilon'}{dt} + K'_f \omega'_{ref} \tag{5.3}$$

$$\varepsilon' = \omega'_{ref} - \omega' \tag{5.4}$$

A diferença entre ambos reside no facto de o valor de ω ser medido de diferentes maneiras nos dois casos. No robô real é medido em impulsos do *encoder* por ciclo e no simulado é medido em rad/s. Assim a relação é dada por:

$$\omega = \frac{NT}{2\pi}\omega' \tag{5.5}$$

em que N é o número de impulsos por revolução da roda e T é o período de amostragem do controlador.

Usando as equações (5.2),(5.4) e (5.5), obtém-se:

$$\varepsilon = \frac{NT}{2\pi}\varepsilon' \tag{5.6}$$

Substituindo na equação (5.1) os valores das equações (5.5) e (5.6), obtém-se:

$$V_{out} = K_p \frac{NT}{2\pi}\varepsilon' + K_i \int \frac{NT}{2\pi}\varepsilon' dt + K_d \frac{NT}{2\pi}\frac{d\varepsilon'}{dt} + K_f \frac{NT}{2\pi}\omega'_{ref} \tag{5.7}$$

Pode-se relacionar os ganhos de ambos os controladores, equações (5.3) e (5.7)

$$K'_p\varepsilon' + K'_i\int \varepsilon' dt + K'_d \frac{d\varepsilon'}{dt} + K'_f\omega'_{ref} = K_p \frac{NT}{2\pi}\varepsilon' + K_i \int \frac{NT}{2\pi}\varepsilon' dt + K_d \frac{NT}{2\pi}\frac{d\varepsilon'}{dt} + K_f \frac{NT}{2\pi}\omega'_{ref} \tag{5.8}$$

Deste modo consegue-se relacionar os parâmetros de cada controlador:

$$K'_p = \frac{NT}{2\pi}K_p \tag{5.9}$$

$$K'_i = \frac{NT}{2\pi}K_i \tag{5.10}$$

$$K'_d = \frac{NT}{2\pi}K_d \tag{5.11}$$

$$K'_f = \frac{NT}{2\pi}K_f \tag{5.12}$$

Com os valores reais conseguem-se estimar os valores para os parâmetros do simulador. O período de amostragem do controlador real é de 10 ms. O número de impulsos por revolução da roda é 256. Os valores dos parâmetros do controlador real são os seguintes:

Tabela 5.1 - Valores dos parâmetros do controlador real

| $K_p$ | $K_i$ | $K_d$ | $K_f$ |
|---|---|---|---|
| **0.4** | 0 | 0 | 0.1 |

Usando os valores dos parâmetros do controlador real nas equações (5.9), (5.10), (5.11) e (5.12) estimam-se os valores dos parâmetros do simulador:

$$K'_p = \frac{256 \times 0.01}{2\pi} \times 0.4 = 0.16 \tag{5.13}$$

$$K'_i = \frac{256 \times 0.01}{2\pi} \times 0 = 0 \tag{5.14}$$

$$K_d^{'} = \frac{256 \times 0.01}{2\pi} \times 0 = 0 \quad (5.15)$$

$$K_f^{'} = \frac{256 \times 0.01}{2\pi} \times 0.1 = 0.04 \quad (5.16)$$

No controlador do simulador o parâmetro *mode* representa o tipo de controlador e o parâmetro *period* é período de amostragem.

*<controller mode='pidspeed' kp='0.16' ki='0' kd='0' kf='0.04' active='1' period='10'/>*

O modelo de atrito que está implementado no simulador utiliza os parâmetros *bv* e *fc*. Esses parâmetros foram ajustados como se verá na secção seguinte.

*<friction bv='' fc=''/>*

### 5.4.3 Superfícies de colisão

Tal como anteriormente foi dito o modo de implementar as colisões é obtido através das *shells*. Neste caso foram inseridos à volta do cilindro, a fazer um tipo de pára-choques, 8 cuboides com a largura de 0.01 m, o comprimento 0.1m e altura de 0.05 m.

```
<shells>
 <cuboid>
  <size x='0.01' y='0.1' z='0.05'/>
  <pos x='0.12' y='0' z='0'/>
  <rot_deg x='0' y='0' z='0'/>
 </cuboid>
    <cuboid>
  <size x='0.01' y='0.1' z='0.05'/>
  <pos x='-0.12' y='0' z='0'/>
  <rot_deg x='0' y='0' z='0'/>
 </cuboid>
   <cuboid>
  <size x='0.1' y='0.01' z='0.05'/>
  <pos x='0' y='0.12' z='0'/>
  <rot_deg x='0' y='0' z='0'/>
 </cuboid>
  <cuboid>
```

```
    <size x='0.1' y='0.01' z='0.05'/>
    <pos x='0' y='-0.12' z='0'/>
    <rot_deg x='0' y='0' z='0'/>
  </cuboid>
    <cuboid>
    <size x='0.09' y='0.01' z='0.05'/>
    <pos x='0.085' y='0.085' z='0'/>
    <rot_deg x='0' y='0' z='135'/>
  </cuboid>
    <cuboid>
    <size x='0.09' y='0.01' z='0.05'/>
    <pos x='0.085' y='-0.085' z='0'/>
    <rot_deg x='0' y='0' z='45'/>
  </cuboid>
    <cuboid>
    <size x='0.09' y='0.01' z='0.05'/>
    <pos x='-0.085' y='0.085' z='0'/>
    <rot_deg x='0' y='0' z='45'/>
  </cuboid>
    <cuboid>
    <size x='0.09' y='0.01' z='0.05'/>
    <pos x='-0.085' y='-0.085' z='0'/>
    <rot_deg x='0' y='0' z='135'/>
  </cuboid>
</shells>
```

No anexo A encontra-se o ficheiro XML completo do robô

O robô ficou com o aspecto que mostra a Figura 5.3.

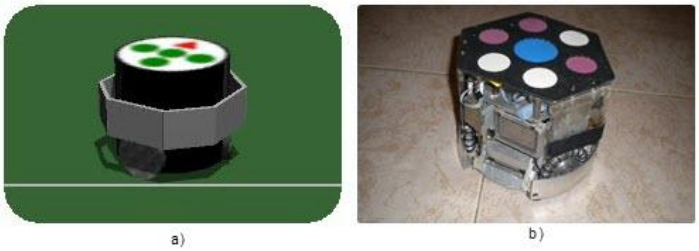



Figura 5.3 – a) Robô no simulador b) Robô real

### 5.4.4 Validação do simulador

De modo a validar o simulador, alguns dos parâmetros foram ajustados de modo a que o simulador consiga simular o mais real possível o robô real. Os parâmetros bv e fc do elemento *friction* e o ki do motor foram ajustados.

O processo de estimação implementado foi o seguinte:

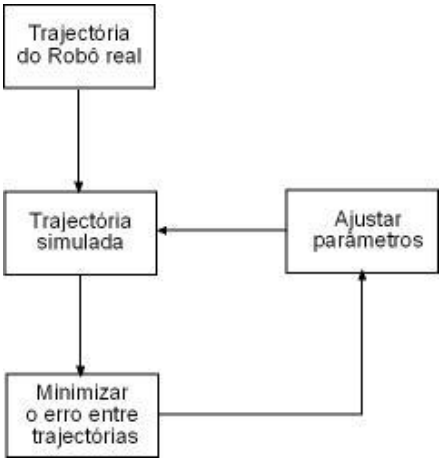



Figura 5.4 - Processo de estimação

Dada uma trajectória do robô real, ajustam-se os parâmetros de modo a minimizar o erro entre as trajectórias. Neste caso utilizaram-se duas trajectórias do robô real:

1º caso – O robô executa um circunferência em que a roda de trás está parada e as duas restantes estão a andar a velocidades iguais com sentidos contrários.

2º caso – O robô vai do ponto "a" até ao ponto "b" em linha recta, utilizando o controlador de movimento.

Para cada caso foram feitas 10 corridas e seleccionou-se a trajectória que tem o erro menor em relação à média das 10. Deste modo, ficou-se com a trajectória que melhor representa a situação.

Os erros que se tentaram minimizar foram:

- o quadrado da diferença entre os ângulos θ(t)
- o quadrado da diferença entre as coordenadas x(t)
- o quadrado da diferença entre as coordenadas y(t) de cada trajectória.

$$erro = \sum_{k=0}^{N} \left( (\theta(k)_s - \theta(k)_r)^2 + (x(k)_s - x(k)_r)^2 + (y(k)_s - y(k)_r)^2 \right) \quad (5.17)$$

No simulador os parâmetros a ajustar foram variando de um valor inicial até a um máximo com um dado intervalo. Estes valores foram baseados nos valores estimados a partir do robô real. O procedimento para descobrir os parâmetros ideais foi de uma pesquisa exaustiva, isto é, testaram-se todos os valores devido ao facto deste problema ter muitos mínimos locais. Esta foi a razão pela qual não foi utilizado nenhum algoritmo para minimizar o erro. Com a pesquisa exaustiva garante-se que se encontra efectivamente o mínimo global. Esta solução mostra também uma das grandes potencialidades de se utilizar o simulador: era totalmente impraticável realizar uma pesquisa exaustiva de valores no robô real.

Tabela 5.2 - Valores de cada parâmetro:

| Parâmetros | Valor inicial | Valor máximo | Intervalo |
|---|---|---|---|
| **bv** | 0 | 0.01 | 0.001 |
| **fc** | 0 | 0.1 | 0.005 |
| **Ki(motor)** | 0 | 0.1 | 0.001 |

Tabela 5.3 - Valores optimizados

| Parâmetros | Valor |
|---|---|
| **bv** | 0.002 |
| **fc** | 0.01 |
| **Ki(motor)** | 0.013 |

No 1º Caso obtiveram-se os seguintes gráficos:

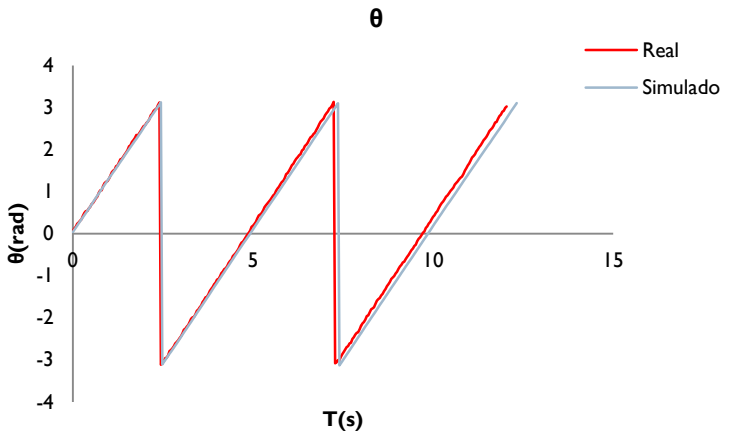



Gráfico 5.1 - Evolução dos ângulos Real e Simulado ao longo da trajectória para o 1º caso

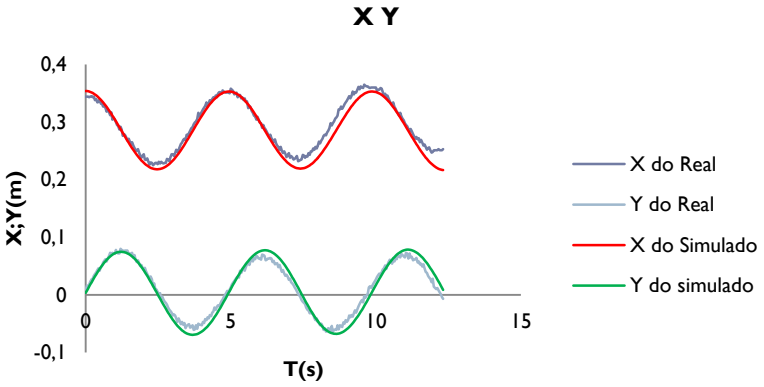



Gráfico 5.2 - Evolução do x e do y ao longo da trajectória para o 1º caso

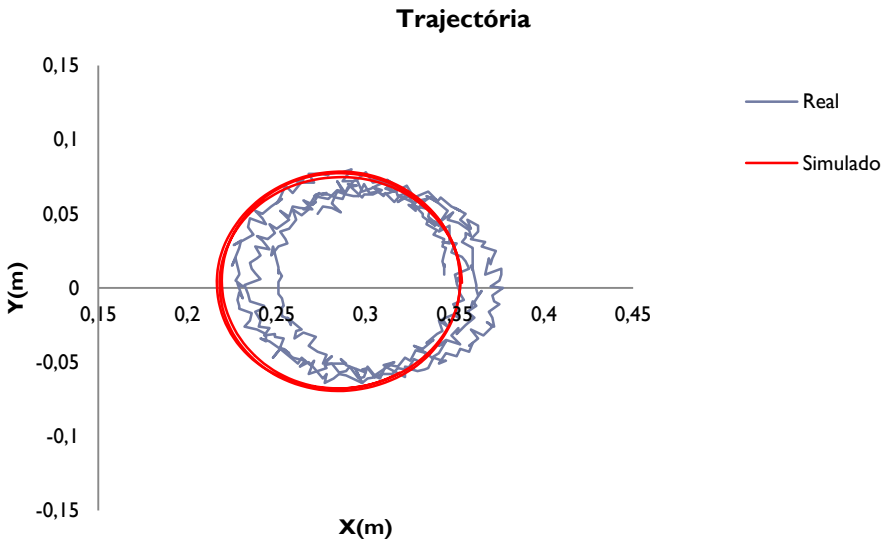



Gráfico 5.3 - Trajectória do 1º caso

Neste caso o único comentário a acrescentar é que o robô real tem um ligeiro deslizamento para a direita, enquanto que no simulador esse deslizamento não existe. Esse ligeiro deslizamento deve-se ao facto natural do sistema roda-motor não ter comportamentos exactamente iguais nas três rodas.

No 2º Caso obteve-se o seguinte gráfico:

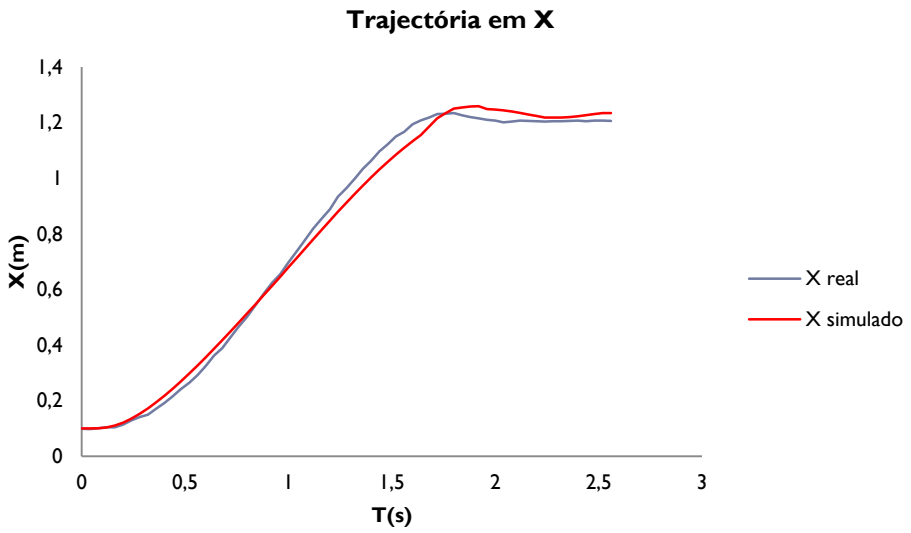



Gráfico 5.4 - Evolução do x ao longo da trajectória para o 2º caso

Como a trajectória é um linha recta em cima do eixo dos x, o gráfico não necessita do y e do θ. Pelo gráfico pode-se constatar a semelhança entre o simulado e o real.

De modo a validar estes valores dos parâmetros executaram-se outros dois casos para verificar o erro.

3º caso

Igual ao 1º caso mas com velocidade superior. Os resultados obtidos foram os seguintes:

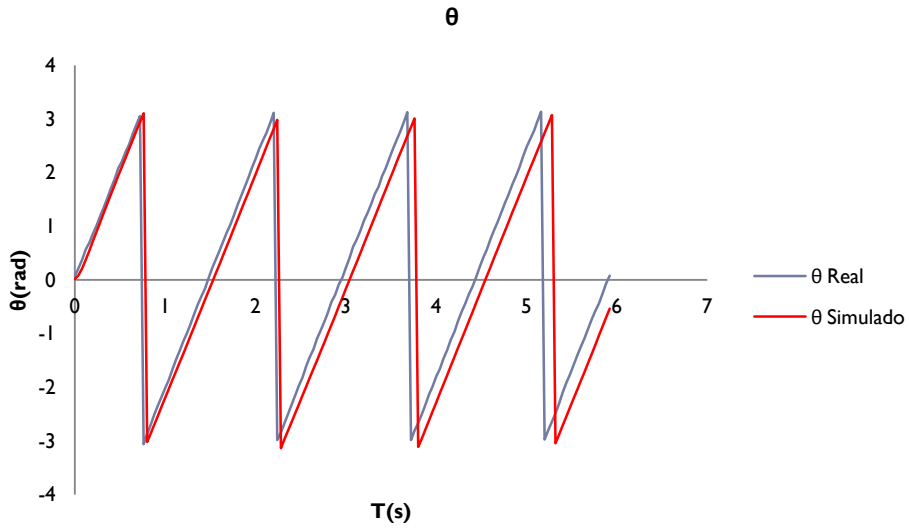



Gráfico 5.5 - Evolução dos ângulos Real e Simulado ao longo da trajectória para o 3º caso

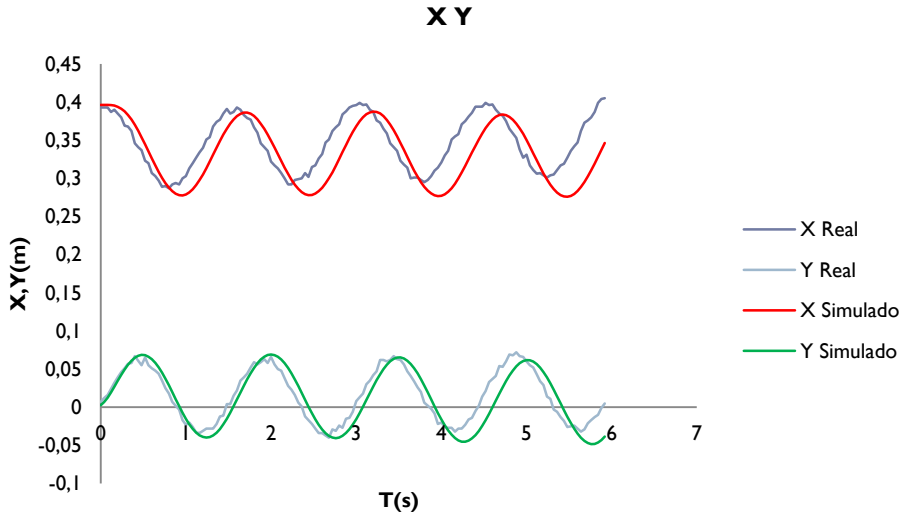



Gráfico 5.6 - Evolução do x e do y ao longo da trajectória para o 3º caso

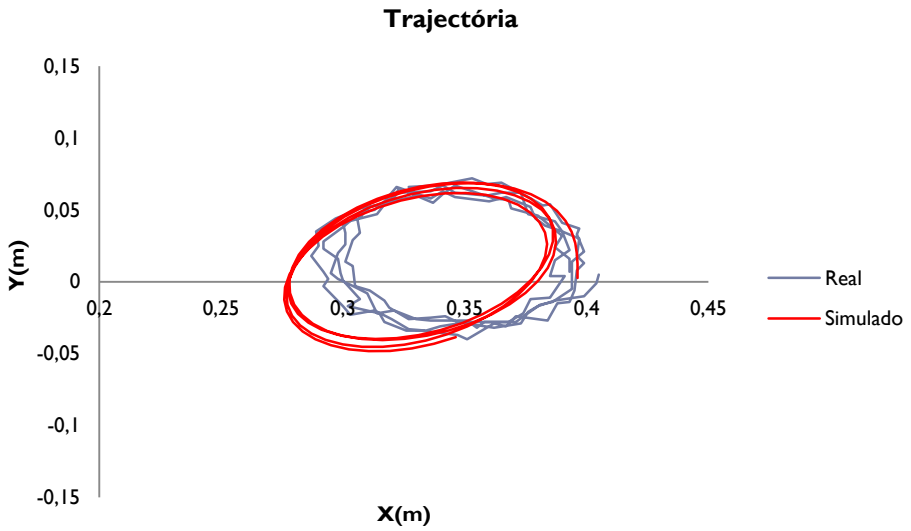



Gráfico 5.7 – Trajectória para o 3º caso

4º Caso

No 4º caso o robô vai do ponto "a" até ao ponto "b" em linha recta, mas encontra um obstáculo em x = 0.75 m. As trajectórias geradas tanto pelo real como pelo simulador (Gráfico 5.8) mostram que o simulador consegue seguir o real.

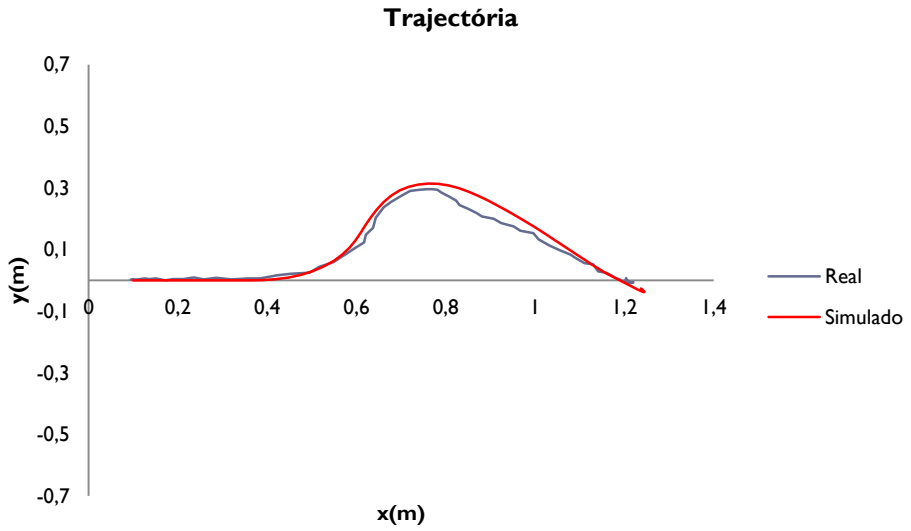



Gráfico 5.8 - Trajectória para o 4º caso

Tabela 5.4 - Erro quadrático médio dos 4 casos

| | **Erro Quadrático Médio** |
|---|---|
| **1º Caso** | 0.00708 |
| **2º Caso** | 0.00132 |
| **3º Caso** | 0.10125 |
| **4º Caso** | 0.00104 |

## 5.5 Arquitectura de Controlo dos robôs

O modelo de controlo implementado no SimTwo é constituído por dois níveis. O nível mais baixo é o controlador dos motores que, por omissão, corre com um período de 10ms. A uma frequência mais baixa (período de 40ms, por omissão) tem-se a oportunidade de implementar um controlador de alto nível, mais complexo, recorrendo a uma função implementada num *script* ou então remotamente. Neste caso foi criada uma outra aplicação que controla remotamente os robôs. Nessa aplicação está implementado o algortimo A* e as suas modificações. A Figura 5.5, mostra o software implementado. O software comunica com o simulador SimTwo ( zona verde da figura ), recebe os dados que o SimTwo envia ( dados relativos ao posicionamento e velocidades do robô e obstáculos) e envia os dados ( referências das velocidades do robô e obstáculos) ( zona vermelha da figura). A zona laranja do software indica as modificações que se pretende activar e os respectivos valores de cada uma das modificações. A zona azul indica as características iniciais, relativamente ao posicionamento e velocidades do robô e dos obstáculos. A zona amarela mostra os dados relativos ao tempo total da execução da trajectória, ao comprimento da trajectória e ao tempo médio do algoritmo. A zona preta mostra os dados da trajectória (x,y e as células). A zona roxa é a zona de gravar em ficheiros os resultados da trajectória e mostra o mapa. E por fim as zonas verde claro, azul claro e cor de rosa foram utilizadas para estimar os parâmetros do simulador.

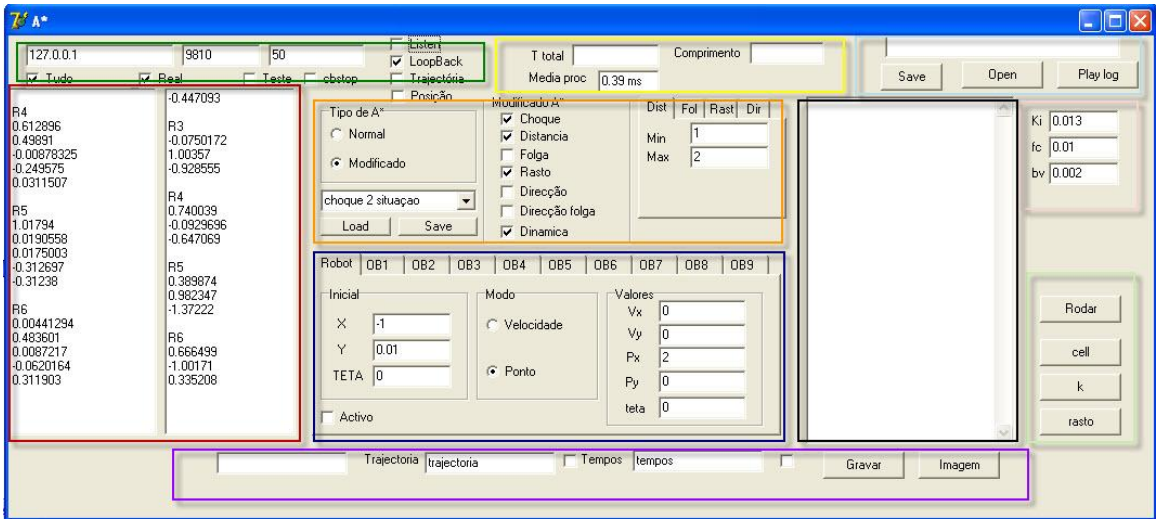



Figura 5.5 - Software de controlo

## 5.6 Conclusão

Os erros quadráticos médios dos 1º, 2º e 4º casos são muito baixos, o erro do 3º caso é mais elevado e deve-se em parte ao deslizamento do robô real. Mas esse deslizamento, em termos de trajectórias que um robô executa normalmente, não vai afectar significativamente os resultados.

Por isso os resultados obtidos são muito satisfatórios, e é preciso ter em conta que o erro é acumulativo, ou seja, um pequeno desvio no inicio da trajectória afecta a trajectória toda e faz com que o erro aumente. Nos próximos capítulos o simulador será utilizado para validar as melhorias efectuadas no algoritmo A*.

# Capítulo 6

# Alteração ao $C_{espaço}$

Neste capítulo vão ser descritas as alterações introduzidas no $C_{espaço}$, alterações que visam dar ao $C_{espaço}$ alguma da dinâmica dos obstáculos, diminuir o tempo de processamento, minimizar colisões e controlar a direcção de chegada. Para cada uma das alterações será apresentado o resultado obtido de modo a comprovar a melhoria introduzida. Em anexo encontra-se os fotogramas das simulações. Na parte final serão utilizados os robôs reais para comprovar a melhoria alcançada.

Uns dos conceitos que importa novamente salientar é, como o ambiente é dinâmico, com obstáculos a movimentarem-se aleatoriamente, o objectivo é encontrar o caminho mais rápido em vez do caminho mais curto. Tal como acontece quando se anda de carro, o caminho mais curto muitas das vezes não é o caminho mais rápido. Esta situação ocorre devido ao facto de em certas situações a velocidade a que ocorre o movimento ser mais baixa. Em consequência, o objectivo é optimizar a rapidez a alcançar o destino, optimizar o tempo de processamento e evitar colisões. O algoritmo A* optimiza o caminho mais curto, assim é preciso introduzir alterações no $C_{espaço}$ para conseguir melhorar a performance da rapidez.

## 6.1 Pontos de Colisão

O $C_{espaço}$ tal como foi descrito no capítulo 3 contém a localização dos obstáculos, numa posição fixa, determinada pelo instante em que foi capturada a informação. Essa informação normalmente ignora a velocidade do obstáculo.

Um dos modos de usar essa informação num mapa de células é calcular o possível ponto de colisão em função da velocidade do nosso robô e das velocidades e direcções dos movimentos dos obstáculos. Em cada cálculo de trajectória assume-se que os obstáculos têm velocidades constantes, adquiridas nesse instante e que o robô tem uma velocidade média constante. Para o obstáculo tem-se as seguintes equações:

$$x_o(t) = x_{oi} + v_{ox}t \tag{6.1}$$

$$y_o(t) = y_{oi} + v_{ox}t \tag{6.2}$$

em que

$x_{oi}$ representa a posição inicial x do obstáculo

$y_{oi}$ representa a posição inicial y do obstáculo

$v_{ox}$ representa a velocidade x do obstáculo

$v_{oy}$ representa a velocidade y do obstáculo

Para o robô tem-se as seguintes equações:

$$x_r(t) = x_{ri} + v_{xr}t \tag{6.3}$$

$$y_r(t) = y_{ri} + v_{yr}t \tag{6.4}$$

em que

$x_{ri}$ representa a posição inicial x do robô

$y_{ri}$ representa a posição inicial y do robô

$v_{xr}$ representa a velocidade x do robô

$v_{yr}$ representa a velocidade y do robô

O ponto de colisão será quando:

$$x_o(t) = x_r(t) \tag{6.5}$$

$$y_o(t) = y_r(t) \tag{6.6}$$

Fica-se com

$$x_{oi} + v_{ox}t = x_{ri} + v_{xr}t \tag{6.7}$$

$$y_{oi} + v_{oy}t = y_{ri} + v_{yr}t \tag{6.8}$$

Sabendo-se que

$$v_{med} = \sqrt{vxr^2 + vyr^2} \tag{6.9}$$

Com as equações (6.7) e (6.8) e resolvendo-se em ordem a $v_{xr}$ e $v_{yr,}$ fica-se com:

$$v_{xr} = \frac{x_{ri} - x_{oi} - v_{ox}t}{t}, \forall\, t > 0 \tag{6.10}$$

$$v_{yr} = \frac{y_{ri} - y_{oi} - v_{oy}t}{t}, \forall\, t > 0 \tag{6.11}$$

Substituindo na equação (6.10) fica-se com:

$$v_{med}^2 = \left(\frac{x_{ri} - x_{oi} - v_{ox}t}{t}\right)^2 + \left(\frac{y_{ri} - y_{oi} - v_{oy}t}{t}\right)^2 \tag{6.12}$$

Com:

$$a = x_{ri} - x_{oi} \tag{6.13}$$

$$b = y_{ri} - y_{oi} \tag{6.14}$$

$$c = v_{ox} \tag{6.15}$$

$$d = v_{oy} \tag{6.16}$$

$$e = v_{med} \tag{6.17}$$

Substituindo na equação (6.12) de modo a simplificar a resolução fica-se com:

$$e^2 = \left(\frac{a - dt}{t}\right)^2 + \left(\frac{b - ct}{t}\right)^2 \tag{6.18}$$

Resolvendo em ordem a t:

$$e^2t^2 = a^2 - 2adt + d^2t^2 + b^2 - 2bct + c^2t^2 \tag{6.19}$$

$$t = \frac{2ad + 2bc \pm \sqrt{(-2ad-2bc)^2 - 4\left(d^2 + c^2 - e^2\right)\left(a^2+b^2\right)}}{2\left(d^2 + c^2 - e^2\right)}, \forall\, t > 0 \tag{6.20}$$

Através da equação (6.20) encontra-se o tempo em que vai haver a colisão. Substituindo t nas equações (6.1) e (6.2) encontra-se o potencial ponto de colisão entre o nosso robô e cada um dos adversários. É este ponto de colisão que é introduzido no mapa como obstáculo e não a posição actual do obstáculo, como é possível verificar-se pela Figura 6.1.

Em seguida serão apresentados dois exemplos demonstrativos da vantagem de se utilizarem os pontos de colisão.

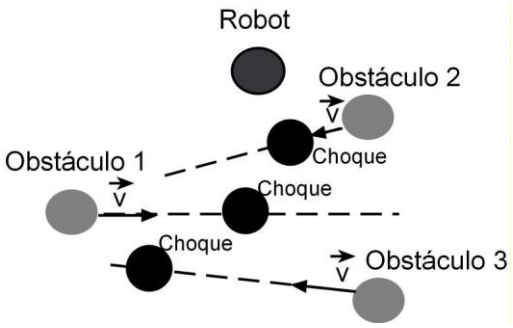



Figura 6.1 - Exemplo dos pontos de choque

### 6.1.1 1ª Situação – Obstáculo em direcção ao robô

Na 1ª situação o robô dirige-se em linha recta e um obstáculo encontra-se a meio do caminho a dirigir-se na direcção do robô, sendo importante de salientar que o obstáculo é cego, isto é, não evita obstáculos.


Figura 6.2 - Pontos de colisão 1ª situação - Visualização do estado inicial

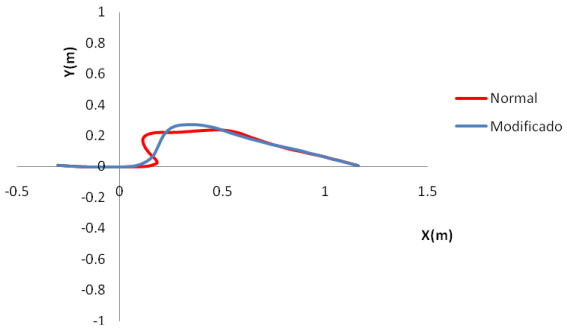



Gráfico 6.1 - Pontos de colisão 1ª situação - Trajectórias executadas

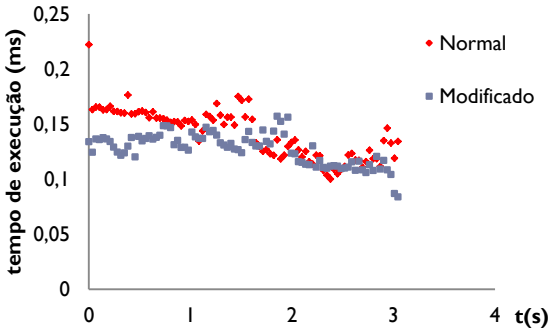



Gráfico 6.2 – Pontos de colisão 1ª situação - Tempo de processamento ao longo da trajectória

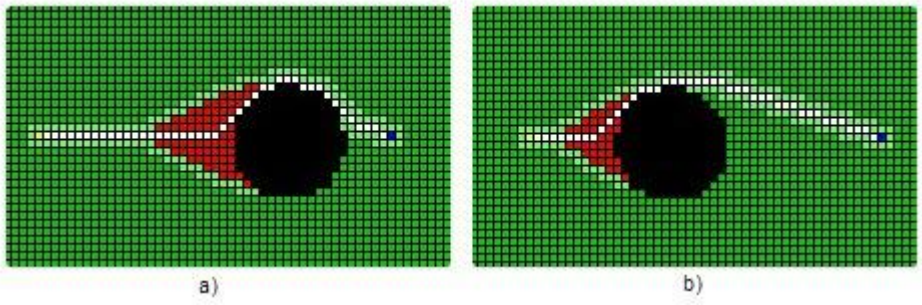



Figura 6.3 – Pontos de colisão 1ª situação - Trajectória gerada pelo A* no 1º instante a) modo normal b) modo modificado

Tabela 6.1 - Pontos de colisão 1ª situação - Resultados

| | Tempo de médio de processamento (ms) | Duração (s) | Colisões |
|---|---|---|---|
| **Normal** | 0.10 | 3.098 | 1 |
| **Modificado** | 0.09 | 2.899 | 0 |

Na situação normalmente utilizada (obstáculos dinâmicos) o robô não consegue evitar a colisão com o obstáculo, não conseguindo virar a tempo de modo a evitar o obstáculo. Ao implementar-se esta nova abordagem com a inclusão do ponto de colisão, e como se pode verificar no Gráfico 6.1, a trajectória já prevê o local de colisão conseguindo evitá-lo. Como consequência o robô consegue chegar mais rápido, neste caso concreto cerca de 200 ms. O tempo de processamento diminui muito ligeiramente, 0.01 ms. Verifica-se no Gráfico 6.2 que essa diminuição é consequência da colisão no modo normal. No modo modificado como passou sem colisão, chega mais rápido e como deixou de ter um obstáculo pela frente mais cedo o processamento foi mais rápido. As Figura 6.3 a) e b) mostram que em virtude do ponto de colisão encontrado ser mais próximo do robô, ele vai reagir mais rápido e assim consegue evitar chocar com o obstáculo.

### 6.1.2 2º Situação – 5 Obstáculos

Esta situação visa mostrar qual é o verdadeiro impacto no tempo de processamento quando o número de obstáculos é mais elevado. Nesta situação serão 5 os obstáculos em que alguns se cruzam com a possível trajectória do robô.

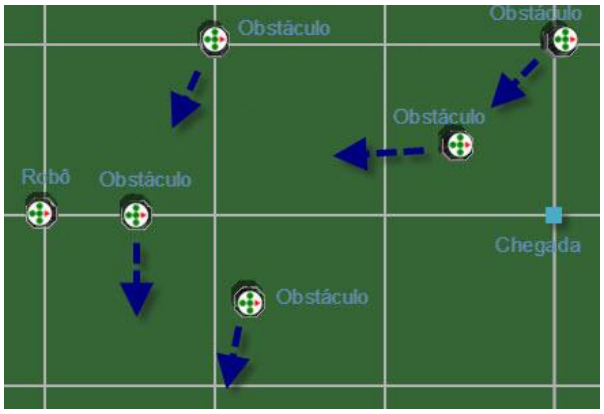



Figura 6.4 - Pontos de colisão 2 ª situação - visualização do estado inicial

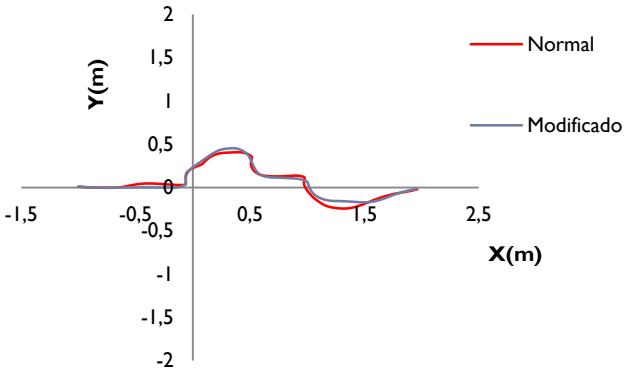



Gráfico 6.3 – Pontos de colisão 2ª situação - trajectórias executadas

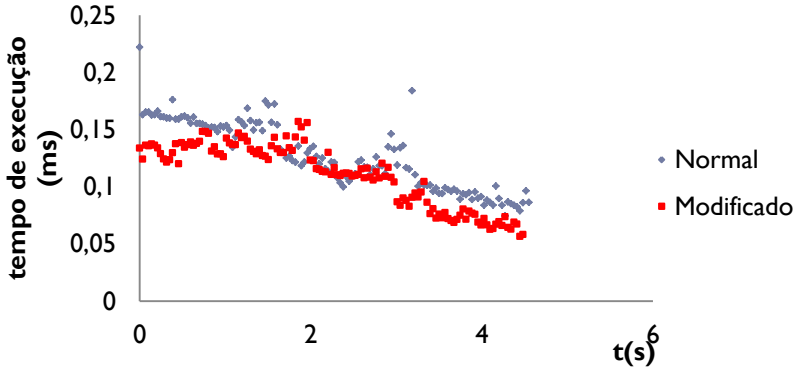



Gráfico 6.4 - Pontos de colisão 2ª situação – Tempo de processamento ao longo da trajectória

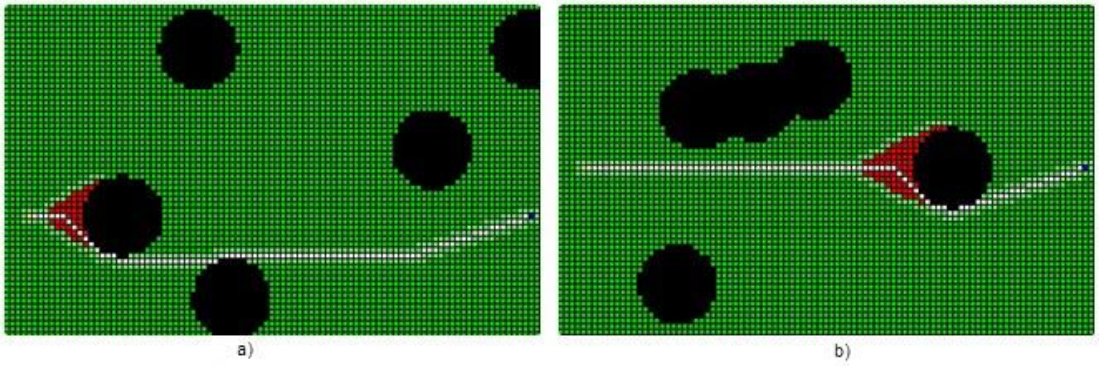



Figura 6.5 - Pontos de colisão 2ª situação - Trajectória gerada pelo A* no 1º instante a) modo normal b) modo modificado

Tabela 6.2 - Pontos de colisão 2ª situação - Resultados

| | Tempo médio de processamento (ms) | Duração (s) | Colisões |
|---|---|---|---|
| Normal | 0.15 | 5.329 | 3 |
| Modificado | 0.16 | 5.102 | 0 |

Neste caso verificou-se que o tempo de processamento médio continuou a ser ligeiramente superior, na ordem dos 0.01 ms. No caso normal existiram 3 colisões, com a implementação da alteração deixaram de existir e como consequência a duração do trajecto baixou em cerca de 227 ms. As Figura 6.5 a) e b) mostram como os pontos de colisões alteraram completamente a trajectória.

Em conclusão pode-se afirmar que a grande vantagem desta alteração é na diminuição do número de colisões, uma vez que consegue prever o ponto de encontro, como consequência o tempo total da trajectória também diminui. Este ganho é conseguido com um ligeiro aumento do tempo de processamento.

## 6.2 Alterações ao $C_{espaço}$ e aos custos

Foram desenvolvidas ainda mais quatro alterações do algoritmo com o objectivo de recriar, tanto quanto possível, a dinâmica dos obstáculos num $C_{espaço}$ estático. Desta forma tenta-se superar algumas das lacunas detectadas ao utilizar-se este algoritmo dado que diminuindo-se o tempo de processamento controla-se as colisões e a direcção de chegada. As denominações das quatro alterações são:

- Zona de Folga – modificação da folga
- Rasto – modificação do rasto
- Direcção – modificação da direcção
- Distância – modificação da distância

### 6.2.1 Zona de Folga

Nesta alteração é criada uma zona à volta do ponto de colisão de modo a que se tenha uma margem de segurança. Essa margem de segurança serve para evitar colisões ou para incentivá-los. Esta margem não é para pôr o obstáculo maior ou menor, mas sim para dar ao mapa um factor de risco maior, ou seja, nessa zona tem-se um custo maior que o custo normal de um ponto do mapa (esse custo é o custo de vir de um nó adjacente para este nó). Nesta alteração, à volta do obstáculo vai existir uma zona em que, à medida que se afasta do centro, o custo vai diminuindo. O custo da célula nessa zona vai passar a ser o custo normal multiplicado por um factor que é inversamente proporcional à distância do centro do obstáculo. Esse factor varia conforme o gráfico, inicialmente é *Cfolga* e no limite o é 1.

O custo de passar de um nó para um nó na folga é igual a:

$$c(n_1,n_c) = c(n_1,n_2) \cdot C \quad (6.21)$$

Sendo $c(n_1,n_2)$ o custo de passar do nó n1 para nó nc, sem custos extras.

C representa o factor multiplicativo extra que aumenta o custo e é extraído da Figura 6.6 b)

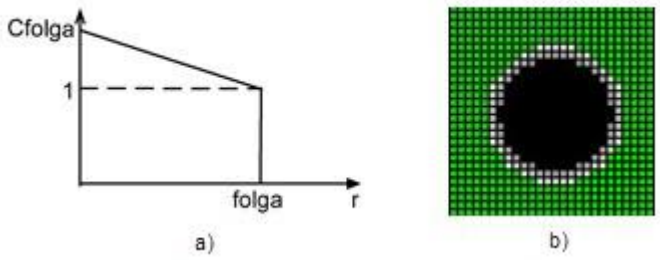



Figura 6.6 - a) Gráfico que representa a variação do factor multiplicativo do custo em função do comprimento da folga Cfolga representa o máximo factor de custo e folga representa máximo comprimento b) representação do obstáculo no mapa de células.

A Figura 6.6 b) mostra que a preto está o obstáculo, isto é, são as células por onde a trajectória nunca pode passar. Em tons de cinzento estão as células que fazem parte da folga, cinzento-escuro indica que o custo é maior. A diferença entre esta alteração e aumentar o obstáculo é que ao aumentar o obstáculo, está-se a obrigar que a trajectória nunca possa passar pelo aumento extra, enquanto que com esta zona de folga tem-se a possibilidade de passar sabendo que se tem um custo maior, mas pode-se dar o caso de compensar esse custo. Ou seja, nos casos do robô poder passar ao lado da folga, a folga funciona como um aumento do obstáculo. Quando o robô não tem alternativa pode passar pela zona de folga.

Para descobrir os valores de *Cfolga* e de *folga* óptimos, foram efectuadas 5 tipos de simulações com 1, 2 e 5 robôs em que para cada tipo de simulação existiam varias simulações com diferentes de posições do obstáculos. Essas simulações foram executadas de modo a minimizar as colisões, o principal critério a ter em conta nesta alteração. Em relação aos restantes critérios o objectivo é não sofrer alterações significativas, isto é, que, o tempo de processamento não aumente muito, e a trajectória possa, nos casos que evita a colisão, diminuir e nos restantes casos não aumente. Os valores encontrados foram os seguintes:

Tabela 6.3 - Dados relativos às constantes da modificação da zona de folga

| | |
|---|---|
| Cfolga | 1.2 |
| folga | 0.06 m |

Em seguida serão apresentadas 3 situações das situações simuladas que melhor representam as principais influências desta alteração. Nestas simulações a alteração do ponto de colisão foi desactivada de modo a se saber que os resultados devem-se única e exclusivamente à alteração da zona de folga.

#### 6.2.1.1 1ª Situação – Obstáculo em direcção ao robô

Será igual ao 1º caso da alteração do ponto de colisão, em que o robô dirige-se em linha recta e um obstáculo encontra-se a meio do caminho a dirige-se na sua direcção.

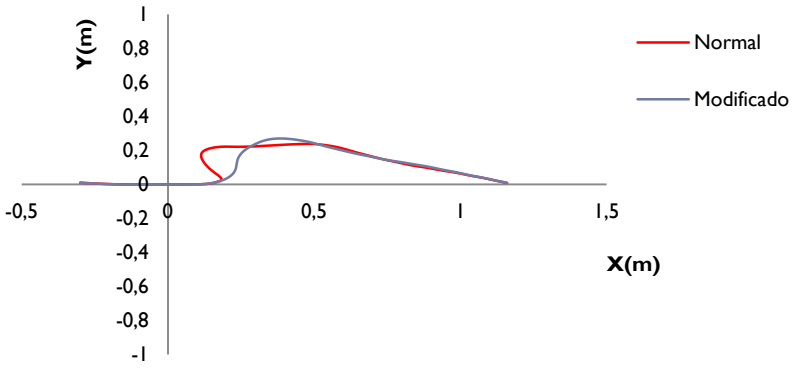



Gráfico 6.5 – Zona de folga 1ª situação - Trajectórias executadas

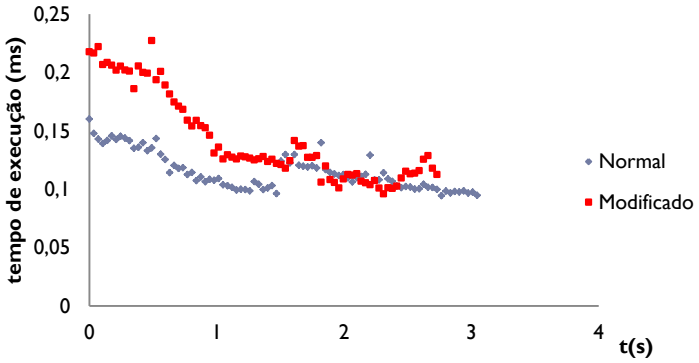



Gráfico 6.6 - Zona de folga 1ª situação - Tempo de processamento ao longo da trajectória

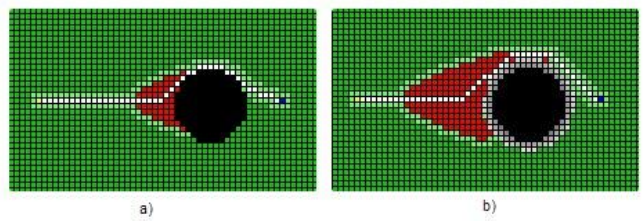



Figura 6.7 - Zona de folga 1ª situação - Trajectória gerada pelo A* no 1º instante a) modo normal b) modo modificado

Tabela 6.4 - Zona de folga 1ª situação - Resultados

| | Tempo médio de processamento (ms) | Duração(s) | Colisões |
|---|---|---|---|
| **Normal** | 0.10 | 3.098 | 1 |
| **Modificado** | 0.13 | 2.791 | 0 |

Como já se tinha verificado, nesta situação o robô não consegue evitar a colisão com o obstáculo. Ao implementar a zona de folga, e como se pode verificar na Figura 6.7, o algoritmo evita a zona de folga e assim passa a ter uma trajectória mais larga o que possibilita passar sem chocar. Como consequência o robô consegue chegar muito mais rápido, neste caso concreto cerca de 300 ms. O tempo de processamento é aumentado como se constata no Gráfico 6.6 do momento inicial até à colisão; na alteração o tempo de processamento é maior, isso deve-se ao facto de com o aumento do obstáculo mais os nós, o algoritmo tem de processar como se verifica na Figura 6.7

#### 6.2.1.2 2ª Situação – Obstáculo parado

Igual à 1º situação, mas o obstáculo fica parado. Esta situação serve para mostrar como se comporta esta alteração em relação aos obstáculos parados.


Figura 6.8 - Zona de folga 2 ª situação - visualização do estado inicial

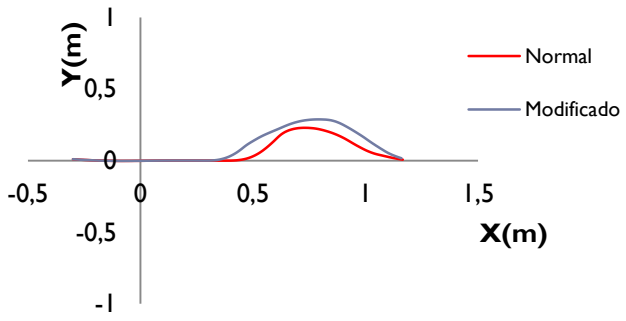



Gráfico 6.7 - Zona de folga 2ª situação - Trajectórias executadas

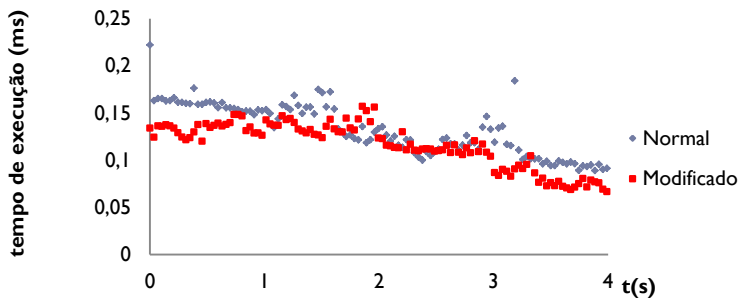



Gráfico 6.8 - Zona de folga 2ª situação - Tempo de processamento ao longo da trajectória

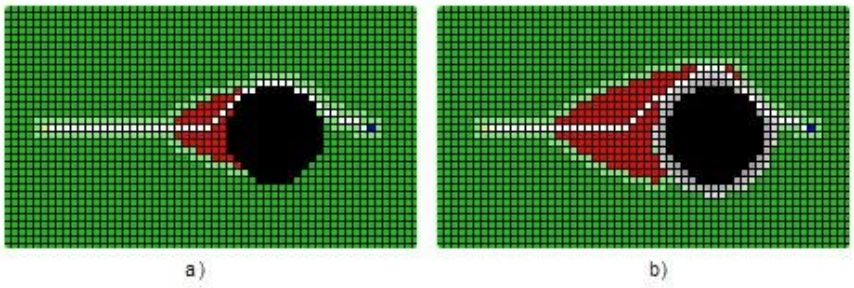



Figura 6.9 - Zona de folga 2ª situação - Trajectória gerada pelo A* no 1º instante a) modo normal b) modo modificado

Tabela 6.5 - Zona de folga 2ª situação - Resultados

| | Tempo médio de processamento (ms) | Duração (s) | Colisões |
|---|---|---|---|
| **Normal** | 0.09 | 2.641 | 0 |
| **Modificado** | 0.12 | 2.675 | 0 |

Com a existência da zona de folga, e a trajectória podendo passar ao lado dela, o obstáculo passa, para o algoritmo, a ser aparentemente maior e como consequência o tempo de processamento aumenta. O robô passa a ter um trajecto maior e o tempo de chegar ao destino também aumenta. O aumento do tempo de processamento no 1º instante sofreu um aumento porque o algoritmo teve de ir procurar mais nós. Esta constatação verifica-se na Figura 6.9, com o aumento do número das células vermelhas.

#### 6.2.1.3 3º Situação – Passagem estreita

Este exemplo serve para mostrar se o robô passa por espaços estreitos, em que a única possibilidade seja passar dentro da zona de folga.

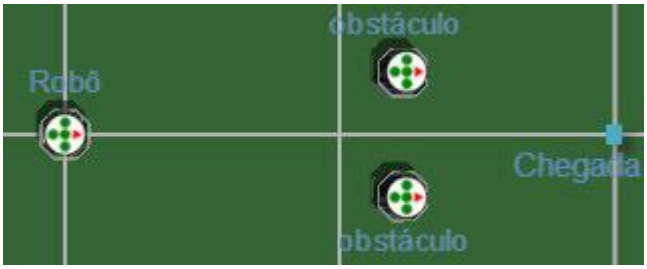



Figura 6.10 - – Zona de folga 3 ª situação - visualização do estado inicial

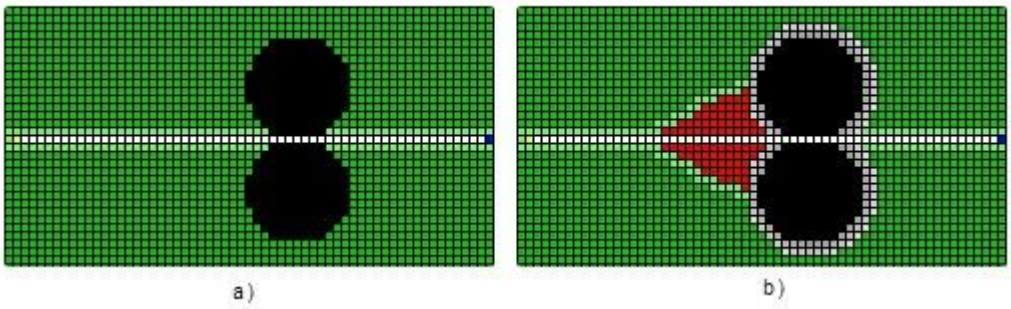



Figura 6.11 - de folga 2ª situação - Trajectória gerada pelo A* no 3º instante a) modo normal b) modo modificado

Tabela 6.6 - Zona de folga 3ª situação - Resultados

| | **Tempo de médio de processamento (ms)** | **Duração(s)** | **Colisões** |
|---|---|---|---|
| **Normal** | 0.09 | 2.913 | 0 |
| **Modificado** | 0.14 | 2.914 | 0 |

Verificou-se que, mesmo com a alteração, o robô executa a mesma trajectória. Apesar de existir a zona de folga a trajectória consegue ir pelo meio dos obstáculos. Esta é a distância mínima que dois obstáculos podem estar de modo a que o robô consiga passar pelo meio deles, tanto para o normal como para a modificação da folga. Isto é, a alteração não influencia em termos de trajectória estas situações. A única consequência é o aumento do tempo de processamento em virtude de ir testar mais nós.

Também é possível fazer o contrário, ou seja, diminuir o tamanho real dos obstáculos e com isso a folga e consegue-se forçar a passagem por locais onde de outra maneira era impossível. No futebol robótico podem existir situações que tal se revele uma vantagem.

Pode-se dizer que a alteração é prejudicial quando existem obstáculos parados, mas pouco altera as suas trajectórias. Com obstáculos em movimento consegue-se evitar muitas colisões, o que

faz com que a alteração compense, uma vez que o ambiente é dinâmico. O tempo de processamento aumenta ligeiramente, em consequência de ter de visitar mais nós.

### 6.2.2 Distância

Esta alteração faz com que quanto mais longe está o ponto de colisão, menor vai ser o obstáculo, ou seja, o obstáculo vai perdendo importância à medida que se afasta como mostra a Figura 6.12. Quando um obstáculo está longe não interfere com a trajectória a curto e médio prazo e é bastante provável que altere a sua velocidade de deslocamento. Como estamos a calcular a trajectória várias vezes por segundo o que se passa a partir de uma certa distância não vai influenciar a nossa trajectória a curto prazo. E quanto menos obstáculos estiverem no nosso mapa, mais rápido será o processamento visto que visitará menos nós.

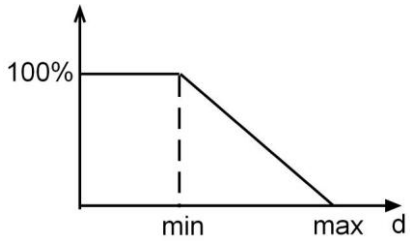



Figura 6.12 - Tamanho do obstáculo em função da distância em que: *d* é a distância do robô ao obstáculo; *min* é a distância em que se considera que começa a perder importância; *max* é a distância em que consideramos que não tem importância nenhuma.

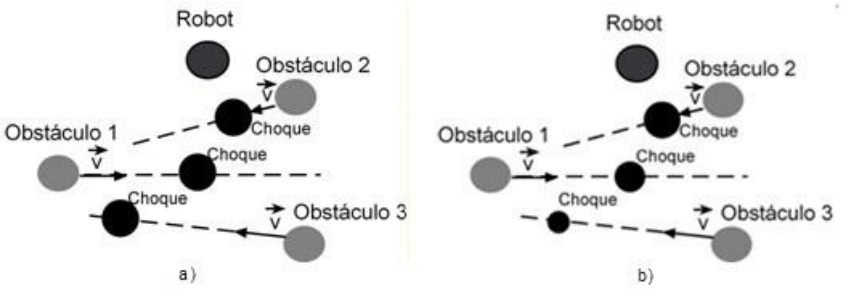



Figura 6.13 - a) Pontos de colisão sem modificação da distância b) pontos de colisão com a modificação da distância

De modo a descobrir quais os valores de *min* e de *max* óptimos, foram efectuadas 2 tipos de simulações com 1 e 5 robôs em que para cada tipo de simulação existiam várias simulações com diferentes posições do obstáculos, com os obstáculos aproximarem-se do robô. Essas simulações

foram executadas de modo a minimizar o tempo de processamento sem influenciar a trajectória, o principal critério a ter em conta nesta alteração. Os valores encontrados foram os seguintes:

Tabela 6.7 - Parâmetros da distância

| | |
|---|---|
| min | 1 m |
| max | 2 m |

Nestas simulações as alterações anteriores foram desactivadas de modo a se saber que os resultados devem-se única e exclusivamente à alteração da distância.

#### 6.2.2.1 1ª situação – Obstáculo parado

Na 1ª situação o robô dirige-se em linha recta e um obstáculo encontra-se parado no meio do caminho.

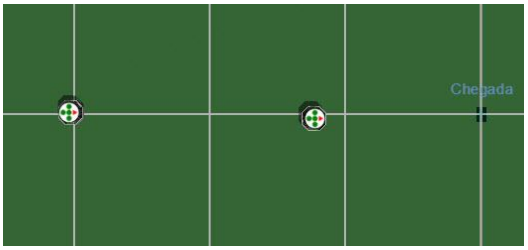



Figura 6.14 - Distancia 1 ª situação - visualização do estado inicial

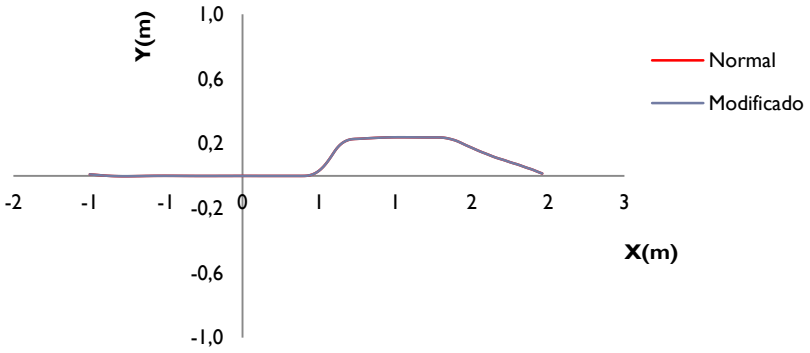



Gráfico 6.9 - Distancia 1ª situação - Trajectórias executadas

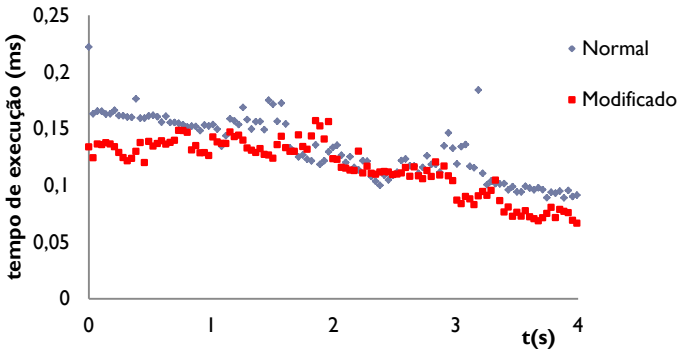



Gráfico 6.10 - Distância 1ª situação - Tempo de processamento ao longo da trajectória

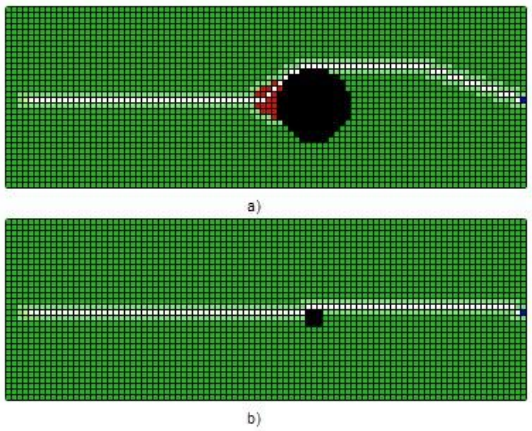



Figura 6.15 - Distância 1ª situação - Trajectória gerada pelo A* no 1º instante a) modo normal b) modo modificado

Tabela 6.8 - Distância 1ª situação - Resultados

| | Tempo médio de processamento (ms) | Duração(s) | Colisões |
|---|---|---|---|
| Normal | 0.13 | 4.409 | 0 |
| Modificado | 0.11 | 4.413 | 0 |

As trajectórias são exactamente iguais, o que se altera é o tempo de processamento - Gráfico 6.10. Verifica-se na Figura 6.15, que os obstáculos mais longes ficam mais pequenos e como consequência os nós visitados são em menor número, logo o tempo de processamento baixa.

#### 6.2.2.2 2º Situação – 5 obstáculos

A 2ª situação será igual ao segundo caso dos pontos de colisão em que vão existir 5 obstáculos em movimento.

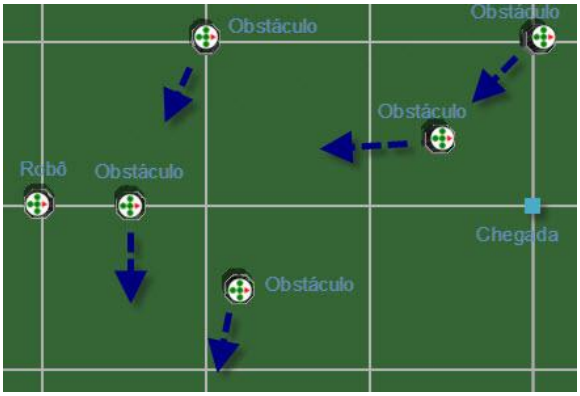



Figura 6.16 - Distância 2 ª situação - visualização do estado inicial

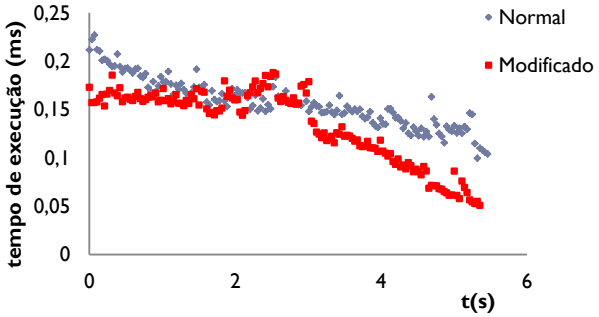



Gráfico 6.11 - Distância 2ª situação - Tempo de processamento ao longo da trajectória

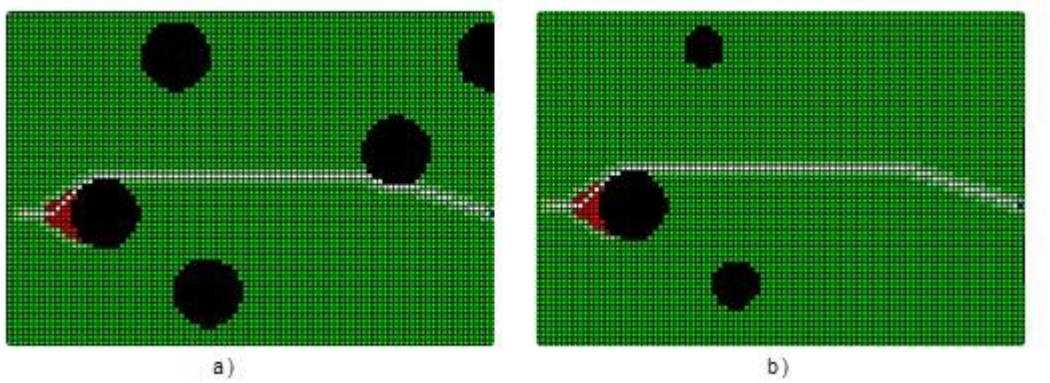



Figura 6.17 - Distância 2ª situação - Trajectória gerada pelo A* no 1º instante a) modo normal b) modo modificado

Tabela 6.9 - Distância 2ª situação - Resultados

| | Tempo médio de processamento (ms) | Duração (s) | Colisões |
|---|---|---|---|
| **Normal** | 0.15 | 5.321 | 3 |
| **Modificado** | 0.09 | 5.324 | 3 |

As trajectórias continuam exactamente iguais, o mesmo número de colisões e a mesma duração, o que muda é o tempo de processamento como se constata no Gráfico 6.11.

O único objectivo da alteração é baixar o tempo de processamento de modo a que o algoritmo essencialmente só se preocupa com a trajectória a curto prazo.

### 6.2.3 Rasto

Esta alteração tem por objectivo evitar que o robô em alguns casos seja levado numa direcção menos favorável. Estes casos, como por exemplo o da Figura 6.18, acontecem porque os obstáculos são considerados objectos parados, e como consequência em cada instante a trajectória óptima para aquele instante não é a melhor para o trajecto global. Na Figura 6.19 as imagens a) e b) mostram como a trajectória óptima é passar pela esquerda, como essa é a direcção do movimento do obstáculo, o robô não consegue passar e como consequência é *arrastado*, neste caso para cima, até ao instante em que ( Figura 6.19 c)) a trajectória passa a ser pela parte direita do obstáculo. Este arrastamento é maior quanto mais próximas forem as velocidades do robô e do obstáculo.

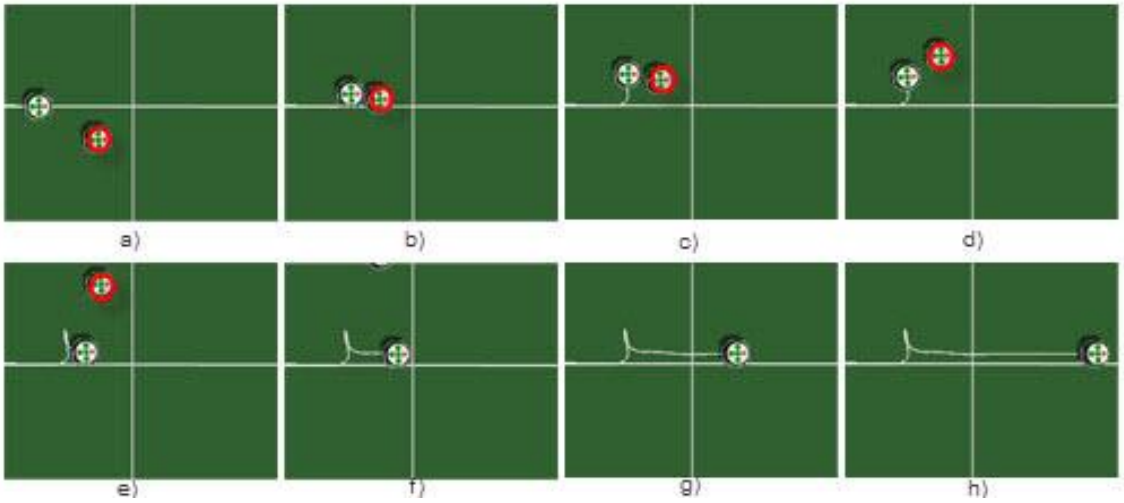



Figura 6.18 - Deslizamento do robô, por causa do obstáculo

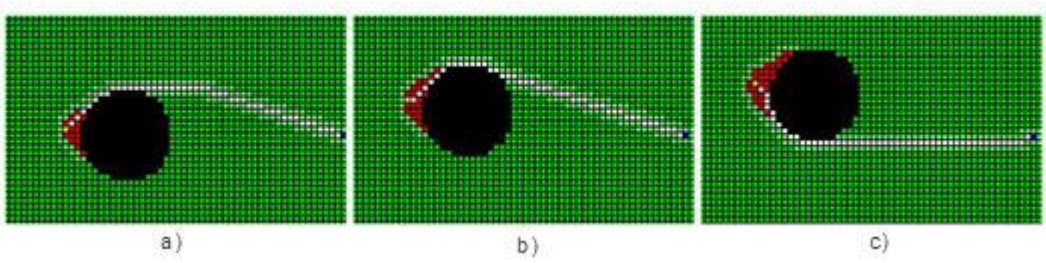



Figura 6.19 - 3 instantes do deslizamento

Esta alteração visa dar ao mapa estático uma ideia dos movimentos dos obstáculos. Consiste em criar uma zona, com direcção e sentido da velocidade do obstáculo e proporcional à sua velocidade. Nessa zona as células passam a ter um custo maior. Inicialmente foi pensada como uma elipse (Figura 6.20 a)) e depois passou a ser um cone (Figura 6.20 b)). Ambas as situações deram resultados parecidos, a única diferença foi o tempo de processamento para cada uma. O cone tem um tempo de processamento mais baixo que a elipse.


Figura 6.20 - a) zona em elipse b) zona em cone

O comprimento do cone varia conforme a velocidade do obstáculo, quanto maior é a velocidade do obstáculo maior é o comprimento ( Gráfico 6.12 a)). O custo da célula passa a ser o custo normal multiplicado por um factor que é inversamente proporcional à distância do centro do obstáculo ( Gráfico 6.12 b)).

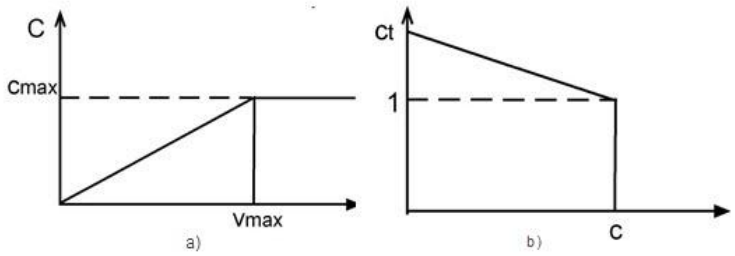



Gráfico 6.12 - a) Gráfico que relaciona a velocidade do obstáculo com o factor multiplicativo do custo em que: Cmax é o máximo comprimento do cone; Vmax é a máxima velocidade; Ct é o máximo factor de custo de b) gráfico que relaciona o factor com a distância ao centro do obstáculo.

Um aspecto importante é que o cone deixa de existir se o robô estiver inserido no cone, este aspecto é importante porque evita que o robô fuja completamente do obstáculo depois de ganhar a posição.

Foram efectuadas 2 tipos de simulações com 1 obstáculo em cada: o primeiro tipo de simulação é igual à situação 1 e 2 que se vai analisar, para o qual se testou o obstáculo mais perto e mais longe. O 2º tipo é quando o ângulo de encontro é cerca de 45º. Estas simulações permitem encontrar os valores de *Cmax*, *Vmax* e *Ct* para minimizar o tempo de execução da trajectória. Os valores encontrados foram os seguintes:

Tabela 6.10 - Parâmetros do rasto

| **Cmax** | **0.60 m** |
|---|---|
| **Vmax** | 1 ms |
| **Ct** | 1.35 |

Nestas simulações as alterações anteriores foram desactivadas excepto o ponto de colisão porque esta alteração é a que tem mais significado com o ponto de colisão activo.

#### 6.2.3.1 1ª Situação- Obstáculo intercepta a trajectório do robô

O robô em linha recta é interceptado por um obstáculo que passa à frente.


Figura 6.21 - Rasto 1 ª situação - visualização do estado inicial

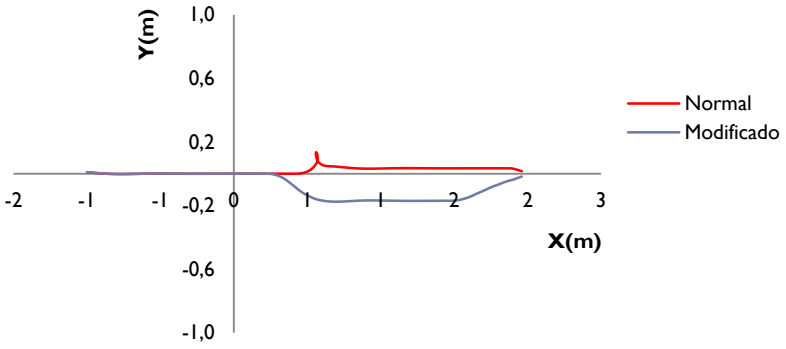



Gráfico 6.13 - Rasto 1ª situação - Trajectórias executadas

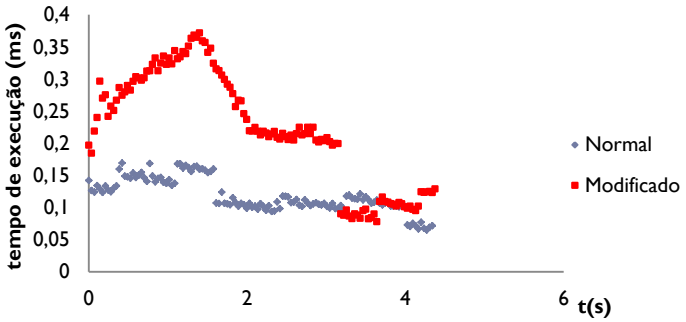



Gráfico 6.14 - Rasto 1ª situação - Tempo de processamento ao longo da trajectória

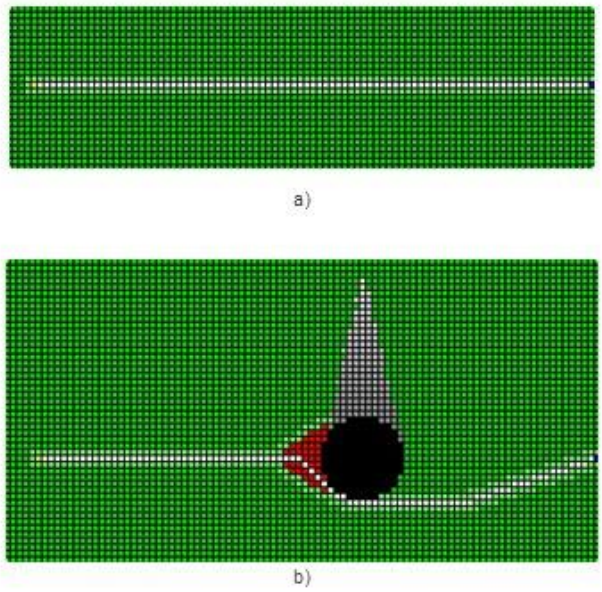



Figura 6.22 - Rasto 1ª situação - Trajectória gerada pelo A* no 1º instante a) modo normal b) modo modificado

Tabela 6.11 - Rasto 1ª situação - Resultados

| | Tempo médio de processamento (ms) | Duração(s) | Colisões |
|---|---|---|---|
| **Normal** | 0.11 | 4.731 | 1 |
| **Modificado** | 0.22 | 4.302 | 0 |

Este é o exemplo descrito no início desta secção, em que o robô é arrastado pelo obstáculo. Com a introdução da alteração, o robô já não tenta passar pela esquerda mas passa pela direita e como consequência a trajectória passa ser outra e é executada muito mais rapidamente. Além disso é também evitado uma colisão. O tempo de processamento aumenta, como se verifica no Gráfico 6.14.

#### 6.2.3.2 2ª Situação – Obstáculo intercepta a trajectória do robô com ponto inicial mais longe

Igual à 1ª situação mas o obstáculo está ligeiramente mais longe, de modo a mostrar o que acontece quando o robô chega primeiro que o obstáculo.

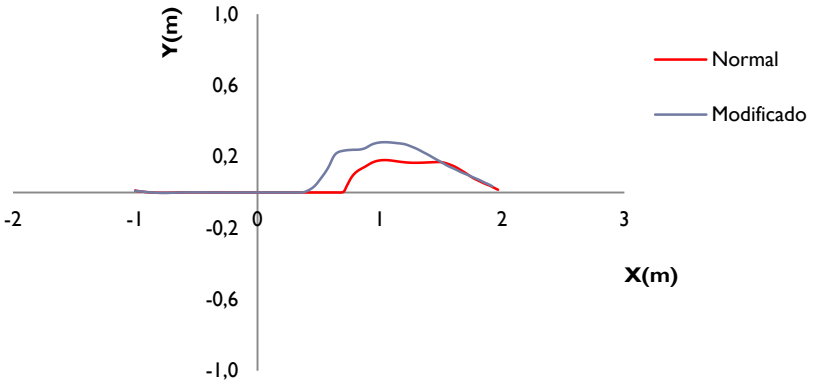



Gráfico 6.15 - Rasto 2ª situação - Trajectórias executadas

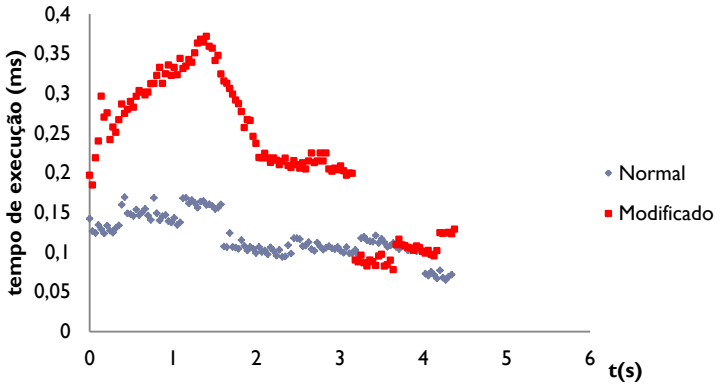



Gráfico 6.16 - Rasto 2ª situação - Trajectória gerada pelo A* no 1º instante a) modo normal b) modo modificado

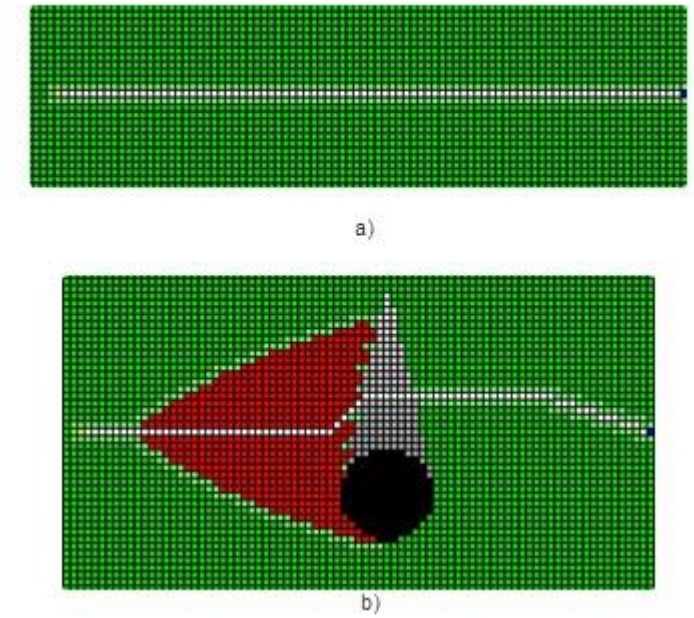



Figura 6.23 - Rasto 2ª situação - Trajectória gerada pelo A* no 1º instante a) modo normal b) modo modificado

Tabela 6.12 - Rasto 2ª situação - Resultados

| | Tempo médio de processamento (ms) | Duração(s) | Colisões |
|---|---|---|---|
| **Normal** | 0.11 | 4.286 | 1 |
| **Modificado** | 0.21 | 4.376 | 0 |

Em ambas as situações o robô passa pelo lado esquerdo, mas com esta alteração evita a colisão. Como consequência de evitar a colisão, a trajectória é mais larga, por isso a duração é ligeiramente maior. O tempo de processamento, apresentado na **Error! Reference source not found.**, antes de o obstáculo ser ultrapassado é muito superior, mas dentro dos valores razoáveis.

#### 6.2.3.3 3ª Situação – 5 obstáculos

A 3ª situação será igual ao segundo caso dos pontos de colisão e da distância em que vão existir 5 obstáculos em movimento.

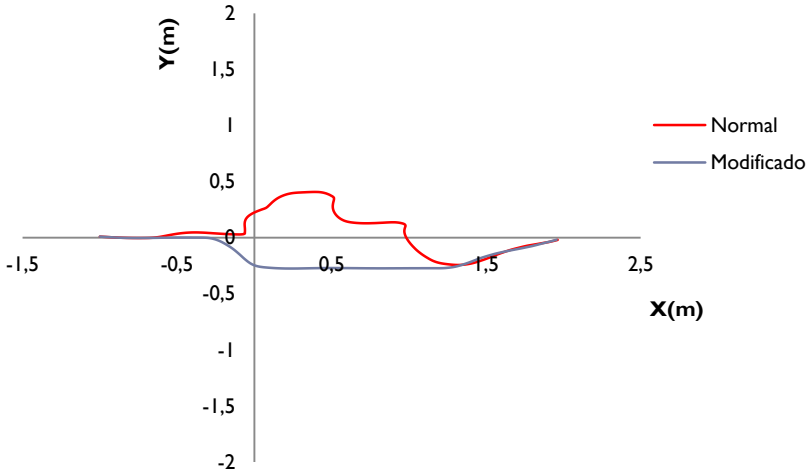



Gráfico 6.17 - Rasto 3ª situação - Trajectórias executadas

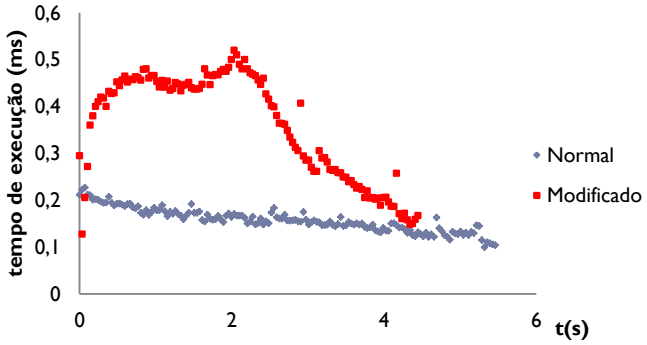



Gráfico 6.18 - Rasto 3ª situação - Trajectória gerada pelo A* no 1º instante a) modo normal b) modo modificado

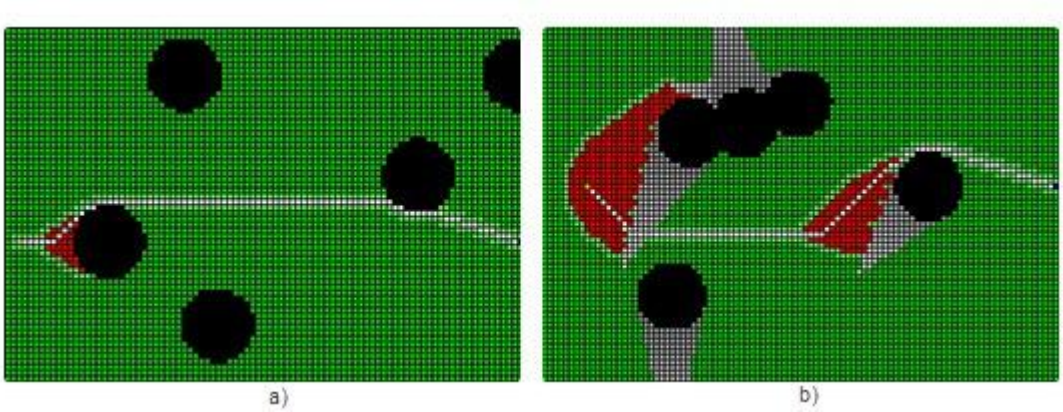



Figura 6.24 - Rasto 3ª situação - Trajectória gerada pelo A* no 1º instante a) modo normal b) modo modificado

Tabela 6.13 - Rasto 3ª situação - Resultados

| | Tempo médio de processamento (ms) | Duração(s) | Colisões |
|---|---|---|---|
| **Normal** | 0.15 | 5.321 | 3 |
| **Modificado** | 0.46 | 4.398 | 0 |

O robô faz uma trajectória completamente diferente com um tempo de execução muito mais baixo, o tempo de processamento é que aumenta também razoavelmente.

Esta alteração consegue eliminar o "arrastamento" mas como consequência o tempo de processamento também aumenta. No entanto esse aumento fica dentro de valores comportáveis.

### 6.2.4 Direcção

Esta alteração tenta fazer com que o robô chegue ao ponto de chegada pela direcção pretendida. Sem esta alteração a trajectória do robô para chegar ao ponto de chegada é uma incerteza, não sabemos por que lado o robô vai chegar ao ponto de chegada. Esta introdução veio corrigir esse problema. O objectivo não é minimizar o tempo de execução, nem evitar colisões ou diminuir o tempo de processamento, é simplesmente conseguir-se que o algoritmo consiga gerar uma trajectória que inclua a direcção de chegada ao destino. Esta situação é algo que não estava contemplado no algoritmo A*.

Considera-se que podemos ter dois casos:

- No primeiro caso a que se chama de "direcção obrigatória", que obriga o robô a chegar pela direcção pretendida (Figura 6.25). Por isso à volta do ponto de chegada é considerado obstáculo exceptuando na direcção pretendida.

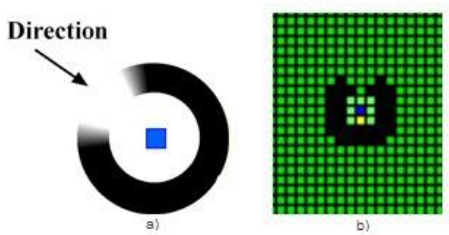



Figura 6.25 - a) Obstáculo com direcção obrigatória b) obstáculo com direcção obrigatória no mapa

- No segundo caso, chamado de "direcção pretendida", temos uma direcção preferida, em que a trajectória calculada pode chegar ao ponto de chegada por essa direcção ou não, dependendo se compensar ou não. Neste caso, como se pode ver na (Figura 6.26), à volta do ponto de chegada existem nós, com custos maiores à medida que nos afastamos da direcção preferida.

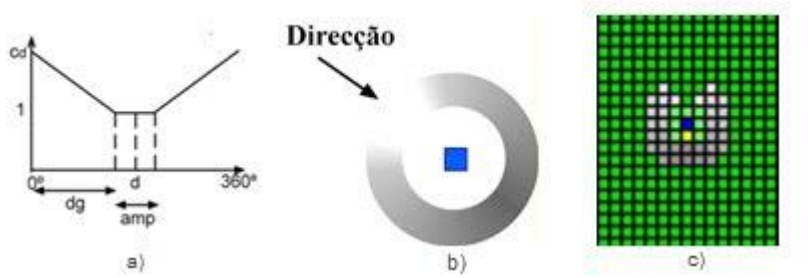



Figura 6.26 – a) Gráfico que relaciona o ângulo com o factor de custo: em que *amp* é zona de segurança que define o espaço sem custos e o Cd é o máximo factor de custo b) Obstáculo com "direcção pretendida" c) obstáculo com "direcção pretendida" no mapa

Foram efectuadas 2 tipos de simulações iguais à situação 1 e 2 desta modificação, em que para cada uma delas a direcção de chegada foi modificada de 10º em 10º. Estas simulações permitem encontrar os valores de *amp* e *Cd* de modo a minimizar o tempo de execução da trajectória. Os valores encontrados foram os seguintes:

Tabela 6.14 - Parâmetros da alteração direcção

| | |
|---|---|
| **amp** | 0.5 rad |
| **Cd** | 1.3 |

Nestas simulações as alterações anteriores foram desactivadas de modo a se saber que os resultados devem-se única e exclusivamente à alteração da direcção.

#### 6.2.4.1 1ª Situação – Sem obstáculos

Serve essencialmente para mostrar a diferença entre os dois tipos de alterações. O robô vai em linha recta para o ponto de destino sem obstáculos.

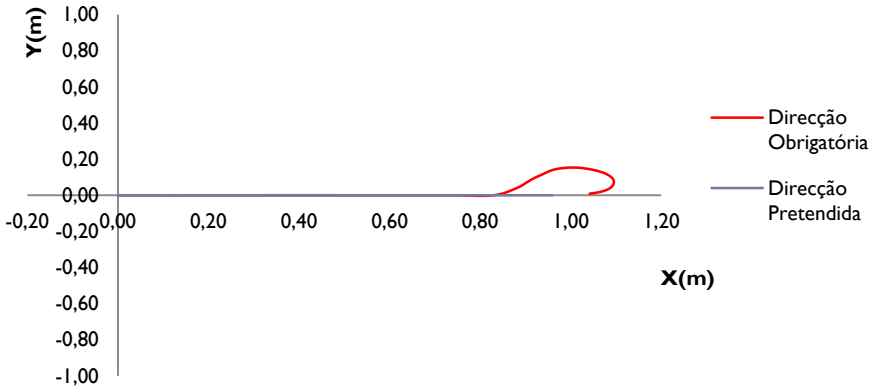



Gráfico 6.19 -Direcção 1ª situação - Trajectórias executadas

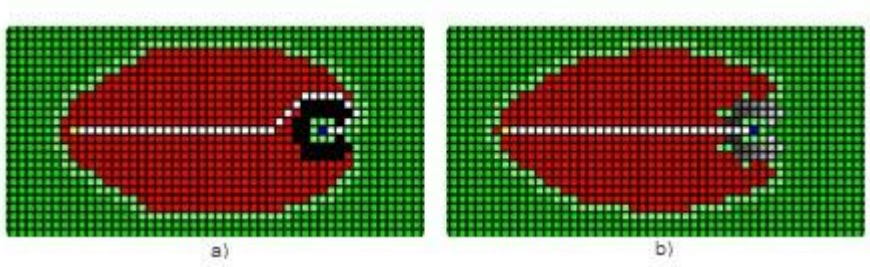



Figura 6.27 - Direcção 1ª situação - Trajectória gerada pelo A* no 1º instante a) Direcção obrigatória b) Direcção Pretendida

Tabela 6.15 - Direcção 1ª situação - Resultados

| | Tempo médio de processamento (ms) |
|---|---|
| **Normal** | 0.09 |
| **Direcção pretendida** | 0.11 |
| **Direcção obrigatória** | 0.15 |

Como se pode constatar, quando se utiliza a alteração obrigatória o robô dá a volta ao ponto de destino para chegar com a direcção pedida e com a alteração pretendida já não compensa ir dar a

volta. O tempo de processamento aumenta ligeiramente. Pela Figura 6.27, confirma-se que os nós processados aumentaram, logo a tempo de processamento aumenta.

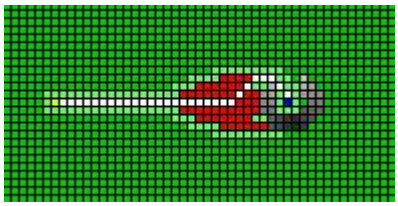


Figura 6.28 - Direcção 1ª situação - Trajectória gerada pelo A* no 1º instante em que o ponto de chegada tem a direcção de 85º

Só a partir do pi/2 rads de diferença é que o robô com a alteração implementada vai dar a volta. (Figura 6.28)

#### 6.2.4.2 2ª Situação – Obstáculo parado

Serve para mostrar que o ponto de chegada pode influenciar a trajectória. O robô encontra um obstáculo parado no meio do caminho, no 1º caso a direcção de chegada é de 90º e no 2º caso é de –90º.

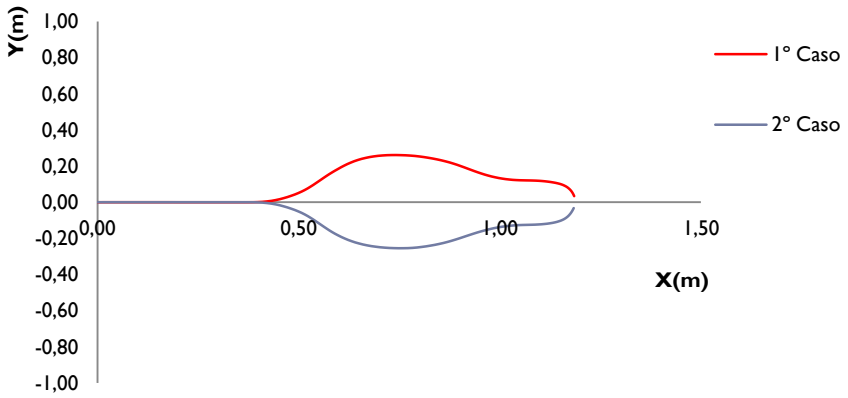



Gráfico 6.20 - Direcção 2ª situação - Trajectórias executadas

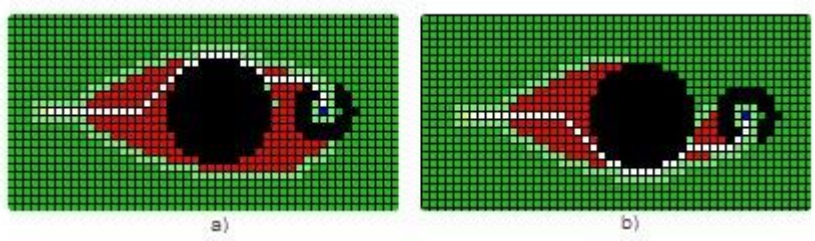



Figura 6.29 - Direcção 2ª situação - Trajectória gerada pelo A* no 1º instante a) 1º caso b) 2º caso

Como se pode verificar as trajectórias variam conforme a direcção pretendida de chegada.

No caso concreto de futebol robótico esta alteração vai ser controlada pelo programa que faz de "treinador". Assim em situações como a do atacante que está a tentar chegar à bola, interessa que chegue à bola na direcção certa, por exemplo, já pronto para rematar. Já os defesas quando estão a defender o mais importante é chegar à bola primeiro e só depois então é chegar na direcção ideal.

## 6.3 Caso com todas as alterações

Nesta secção vão-se utilizar simultaneamente as alterações do ponto de colisão, da distância, da folga e do rasto na situação dos 5 obstáculos, para mostrar o funcionamento de todas em simultâneo.

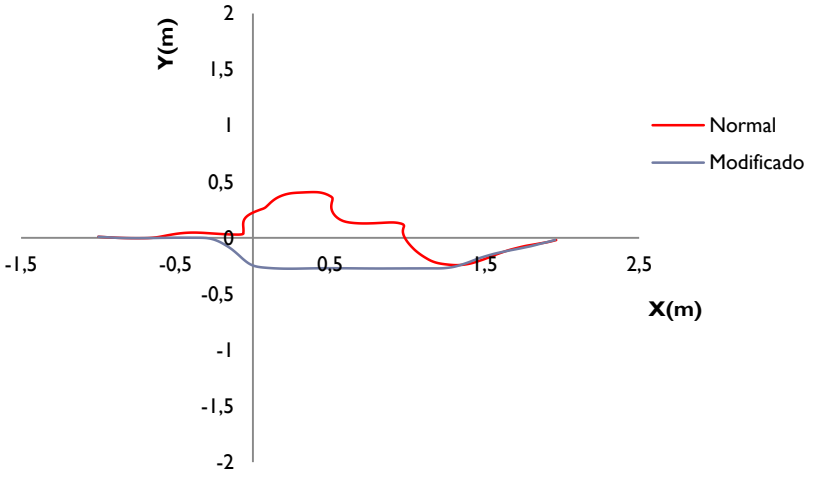



Gráfico 6.21 - Trajectórias executadas

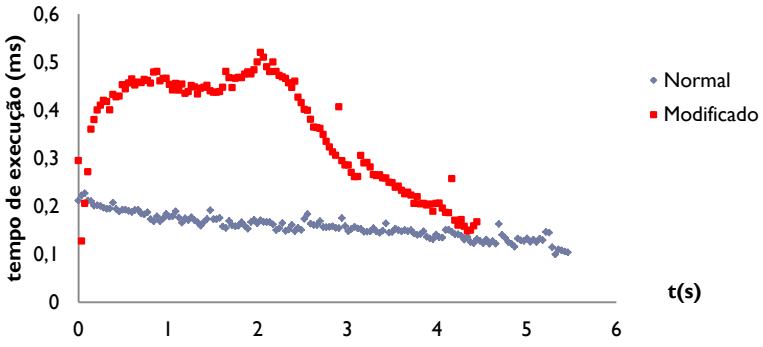



Gráfico 6.22 - Tempo de execução

Tabela 6.16 - Resultados

| | Tempo médio de processamento (ms) | Duração (s) | Colisões |
|---|---|---|---|
| **Normal** | 0.12 | 5.321 | 3 |
| **Modificado** | 0.35 | 4.330 | 0 |

O que se pode constatar é que se consegue uma perfomance na trajectória muito boa, com um aumento do tempo de processamento. Pode-se verificar que o aumento detectado no tempo de processamento na alteração de rasto é diminuido quando se introduz a alteração distância.

## 6.4 Robôs reais

Nesta secção serão apresentados 4 casos em que se utilizam os robôs reais, de modo a consolidar a validação da simulação e das alterações implementadas. Estes casos são as situações que representam melhor as melhorias introduzidas. Cada caso foi executado 5 vezes, a duração é a média das cinco e o gráfico é da que está mais próximo da média. Em todos os casos usou-se todas as alterações e no CD em anexo estão os vídeos.

### 6.4.1 1ª Situação – Obstáculo parado

No meio da trajectória o robô tem um obstáculo parado

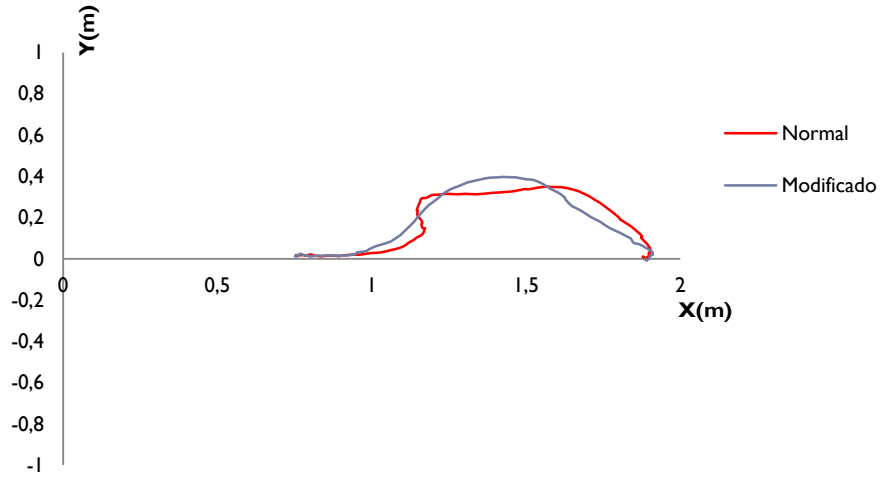



Gráfico 6.23 Robôs reais - 1ª situação - Trajectórias executadas

Tabela 6.17 - Robôs reais 1ª situação - Resultados

| | Duração (s) | Ganho | Colisões |
|---|---|---|---|
| **Normal** | 3.080 | | 0 |
| **Modificado** | 2.640 | 14.3% | 0 |

Neste caso, em que o obstáculo está parado, existe uma melhoria em virtude da alteração da zona de folga que faz com que a trajectória fique mais larga e como consequência o robô consegue executar com maior velocidade, o que permite uma duração menor.

Este exemplo é ligeiramente diferente do simulado porque o obstáculo está mais longe, o robô chega com mais velocidade ao obstáculo e como consequência tem mais dificuldades em contorná-lo. Em obstáculos mais pertos os ganhos diminuem chegando a existir situações como representado na 2ª situação da zona de folga na simulação em que possam não existir ganhos.

### 6.4.2 2ª situação – Obstáculo em direcção ao robô

No meio da trajectória o robô tem um obstáculo a dirigir-se em sentido contrário com velocidade de 0.4 $ms^{-1}$ .

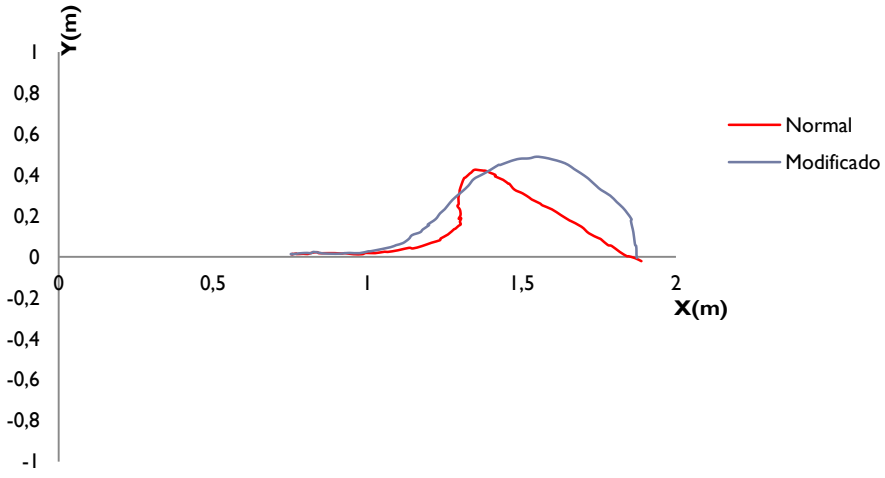



Gráfico 6.24 - Robôs reais - 2ª situação - Trajectórias executadas

Tabela 6.18 - Robôs reais 2ª situação - Resultados

| | Duração (s) | Ganho | Colisões |
|---|---|---|---|
| **Normal** | 3.284 | | 0 |
| **Modificado** | 2.763 | 15.8% | 0 |

Neste caso verifica-se tal como na simulação que o robô consegue desviar-se atempadamente e como consequência ganha tempo com essa manobra. As alterações ponto de colisão e rasto são as alterações que contribuem para essa melhoria. Para velocidades superiores nos obstáculos, quando não existem alterações o robô não consegue evitar o obstáculo, com alterações o robô consegue evitar o obstáculo até velocidades de 0.9 $ms^{-1}$.

### 6.4.3 3ª situação – Obstáculo a intersectar a trajectória do robô

No meio da trajectória o robô tem um obstáculo a passar à frente do robô com velocidade de 0.8 $ms^{-1}$ .

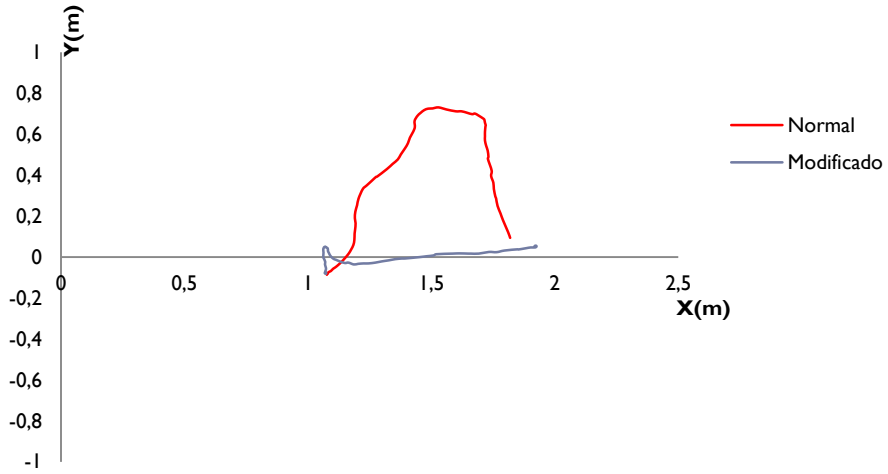



Gráfico 6.25 - Robôs reais - 3ª situação - Trajectórias executadas

Tabela 6.19 - Robôs reais 3ª situação - Resultados

| | Duração (s) | Ganho | Colisões |
|---|---|---|---|
| **Normal** | 2.963 | | 0 |
| **Modificado** | 2.205 | 25.6% | 0 |

O ganho é elevado neste tipo de casos em que o robô executa trajectórias completamente diferentes, neste caso o robô sem alterações esforça-se para passar à frente do obstáculo. Enquanto no modificado o robô passa por trás do obstáculo e logicamente ganha muito tempo com essa operação.

### 6.4.4 4ª situação – Obstáculo a intersectar a trajectória do robô

No meio da trajectória o robô tem um obstáculo a passar à frente do robô com velocidade de 0.8 $ms^{-1}$ , em que as posições iniciais são diferentes da situação anterior.

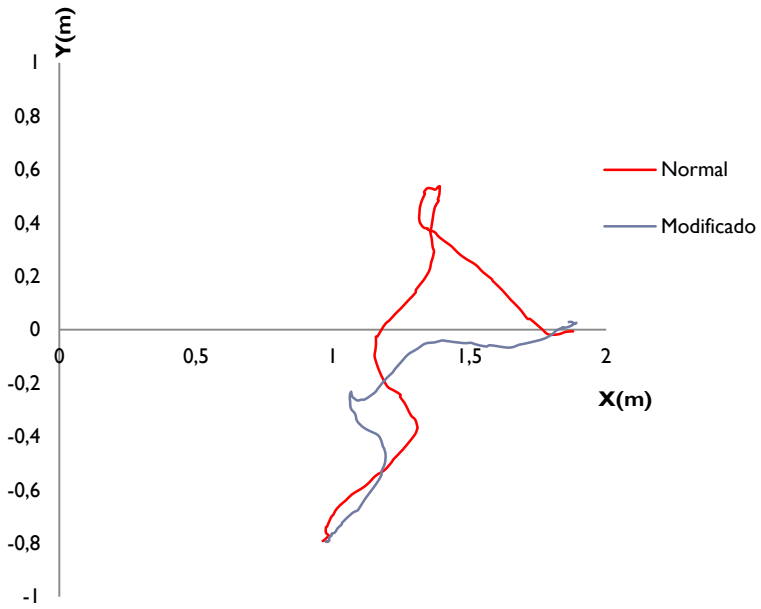



Gráfico 6.26 - Robôs reais - 4ª situação - Trajectórias executadas

Tabela 6.20 - Robôs reais 4ª situação - Resultados

| | Duração (s) | Ganho | Colisões |
|---|---|---|---|
| **Normal** | 4.125 | | 0 |
| **Modificado** | 2.846 | 31.0% | 0 |

Situação muito parecida com anterior, mas neste caso o robô é arrastado durante muito tempo sem nunca conseguir passar à frente do obstáculo. Com as alterações muito mais rapidamente o robô dá-se conta que é melhor passar por trás do obstáculo.

## 6.5 Conclusão

Neste trabalho foi possível implementar um algoritmo A* num cenário de tempo real com obstáculos em movimento. Este algoritmo é executado em tempos na ordem dos 0.20 ms, ou seja, em tempos muito abaixo de 1 ms. Qualquer outros algoritmos da familia, o D*, D* *lite*, E* ou AD* não encontram trajéctórias melhores, encontram as mesmas trajectórias. Por isso prova-se que o algoritmo A* pode ser usado com sucesso em cenário de tempo real com obstáculos em movimento.

Conseguiu-se com a implementação destas alterações ao algoritmo original A* melhorar as trajectórias no aspecto do tempo de execução e principalmente nas colisões. O tempo de processamento, como consequência da implementação das alterações, foi aumentado mas para

valores aceitáveis. No ambito do Futebol Robótico, e como já foi referido no capítulo 4, em cada instante o software que faz de "treinador" executa 10 vezes o algoritmo de gerar trajectórias o que faz com que o tempo de processamento não possa ser superior 10 ms para as 10 vezes, porque é o tempo disponivel para não haver atrasos. A média para gerar uma trajectória será de 1ms. No trabalho efectuado esses valores foram conseguidos, sendo de realçar que esse tempo vai variar conforme a distância a que se encontra o destino e a quantidade de obstáculos que se encontra pelo meio do caminho. Estes testes foram efectuados num computador com 5 Anos ( no Anexo B encontram-se as características deste), se fossem realizados num computador da última geração os resultados do tempo de processamento seriam ainda melhores.

Na ultima secção ficou mostrado, com a utilização de robôs do futebol robotico da liga pequena, que as alterações produzem os efeitos pretendidos e que a simulação espelha a realidade.

Estas modificações não são exclusivas do algoritmo A*, visto que não são alterações ao algoritmo A*, mas sim ao $C_{espaço}$ e aos custos da heuristica. Por isso podem ser implementadas em qualquer algoritmo que utilize este tipo de $C_{espaço}$. Isto é, o D*, D* *lite*, o E*, AD* e os outros algortimos desta familia podem utilizar estas modificações com o mesmo sucesso aqui verificado.

# Capítulo 7

# Alteração das Trajectórias

## 7.1 Introdução

Neste capítulo é descrita a suavização efectuada nas trajectórias. Primeiro será explicado o porquê dessa suavização e quais serão as suas consequências. Em seguida será demonstrado o processo a efectuar, serão apresentados os resultados que comprovarão as melhorias conseguidas tanto no simulador como com os robôs reias e as conclusões a tirar.

## 7.2 Suavização

Ao utilizar o algoritmo A* para gerar trajectórias, a dinâmica do robô não entra nesse planeamento de trajectórias, o que faz com que o robô em algumas situações não consiga seguir exactamente essas trajectórias. Quando a trajectória tem mudanças bruscas de direcção, o robô tem que desacelerar para conseguir seguir essa trajectória. Tal como quando uma pessoa corre, ao fazer uma curva de 90º ou abranda ou alarga a trajectória. Ao desacelerar, para conseguir efectuar as trajectórias, o caminho a percorrer torna-se mais demorado. Logo, em alguns casos o caminho mais curto não é o caminho mais rápido. No futebol robótico e em muitas outras áreas, o caminho mais rápido é o caminho pretendido, algo que muitos algoritmos de planeamento de trajectórias não contemplam, tal como a decomposição em células com o algoritmo A*.

No exemplo do Gráfico 7.1 em que se tem um robô a desviar-se de um obstáculo parado, pode-se constatar que no momento de se desviar do obstáculo o robô teve de desacelerar para conseguir executar a trajectória. É essa desaceleração que faz com que o caminho não seja o mais rápido. O Gráfico 7.2 mostra que a trajectória planeada pelo algoritmo A* não é a que o robô executa; no momento de se desviar do obstáculo, o robô não consegue responder com a rapidez pretendida.

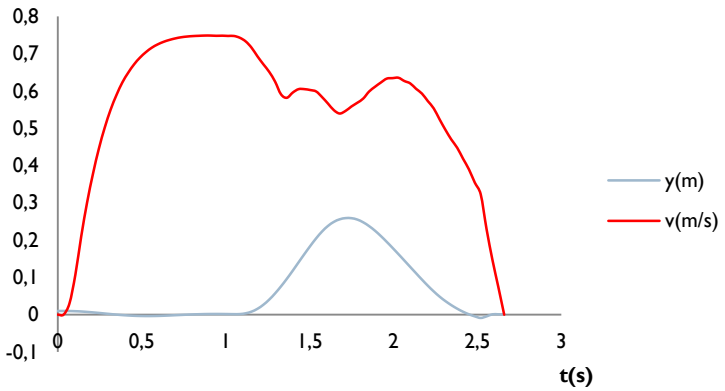



Gráfico 7.1 - Velocidade e posição no eixo dos y do Robô com um obstáculo parado

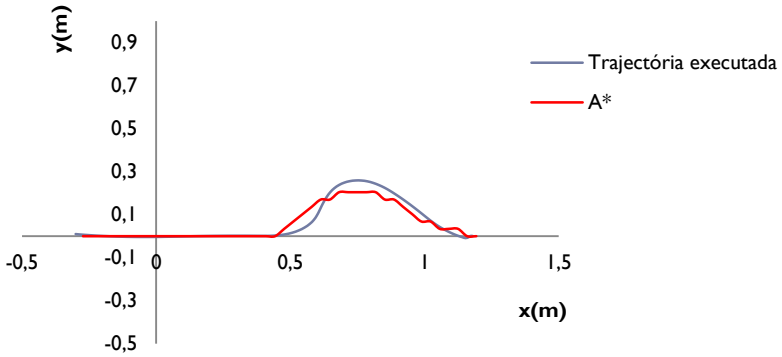



Gráfico 7.2 - Trajectória executada e trajectória planeada pelo algoritmo A*

## 7.3 Processo de alteração da trajectória

Os objectivos a atingir são os mesmos do capítulo anterior, isto é, encontrar o caminho mais rápido, sem que o tempo de processamento seja comprometido e sem que as colisões, no mínimo, aumentem. A preocupação foi encontrar uma solução que, principalmente, não comprometesse o tempo de processamento.

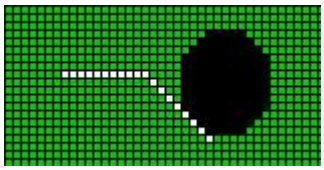


Figura 7.1 - Nós da trajectória quando encontra um obstáculo

No algoritmo A* as mudanças de direcção quando o robô encontra um obstáculo são de 45º como o exemplo da Figura 7.1. O objectivo é diminuir esse ângulo dentro do possível. A

trajectória do robô só faz ângulos superiores a 45º, se a isso for obrigada, ou seja, para contornar obstáculos. Nesses casos não se consegue suavizar a trajectória por causa do obstáculo.

Quando se está a alterar uma trajectória a principal preocupação é que a nova trajectória não intercepte os obstáculos. Para evitar esse aspecto, foram utilizadas as informações das células do algoritmo A*, isto é, a trajectória só pode conter nós fechados do algoritmo. Esses nós são os que foram processados, logo em alguma altura do algoritmo foram considerados nós promissores e principalmente sabe-se que não contêm obstáculos.

O processo de suavizar a trajectória passa por percorrer os nós que fazem parte dessa trajectória desde o ponto de destino até ao ponto inicial e verificar as mudanças de direcção. Como foi descrito no capítulo 3, cada nó tem a indicação de qual é o seu pai (nó), isto é, de onde veio a trajectória. Essa indicação é dada por um número que indica qual é o nó, como mostra a Figura 7.2.Esse número vai ser muito útil para identificar as mudanças de direcções.

| | | |
|---|---|---|
| 5 | 6 | 7 |
| 4 | P | 0 |
| 3 | 2 | 1 |

Figura 7.2 - Numeração dos Pais do nó P

### 7.3.1 Modificação das curvas com 5 nós (Mod5)

Na primeira implementação verificou-se a existência de dois conjuntos de nós:

1. Três nós numa direcção seguidos de dois nós a perfazer um ângulo de 45º, como mostra a Figura 7.3. Esta identificação encontra-se recorrrendo ao número do pai. Sendo $P_1$ o número do pai do nó 1 (primeiro nó da sequência encontrado), a sequência encontra-se quando se verifica a seguinte condição:

$$P_1 = P_2 \wedge P_3 = P_4 \wedge (|P_2 - P_3| = 1 \vee |P_2 - P_3| = 7) \qquad (7.1)$$

que significa que o pai do nó 1 e o pai do nó 2 têm a mesma direcção, que o pai do nó 3 e o pai do nó 4 têm a mesma direcção e que para haver mudança de direcção do pai do nó 2 com o pai do nó 3 a diferença entre eles é de 1 ou 7.

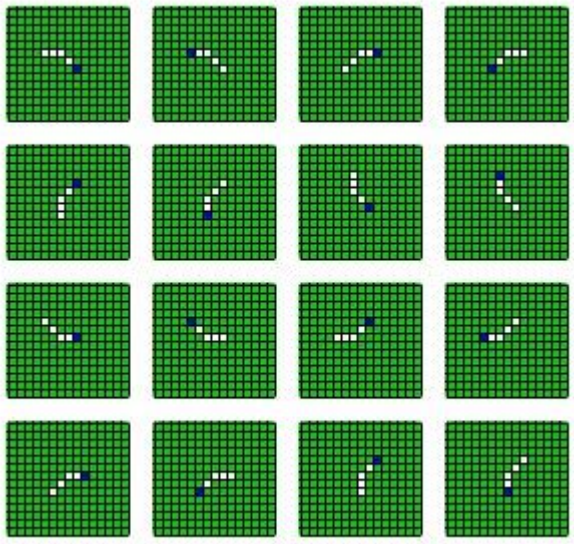


Figura 7.3 - Todas as sequências possíveis para 3 nós mais 2

O nó a azul significa que é o primeiro nó. Por exemplo no caso de encontrar a primeira sequência da Figura 7.3, fica-se com os seguintes valores para os pais dos nós:

Tabela 7.1 - Valores dos pais dos nós da sequência

| $\mathbf{P_1}$ | $\mathbf{P_2}$ | $\mathbf{P_3}$ | $\mathbf{P_4}$ |
|---|---|---|---|
| **5** | 5 | 4 | 4 |

2. Três nós numa direcção seguido de um nó a perfazer um ângulo de 45º como mostra a Figura 7.4. A sequência é encontrada quando se verifica a seguinte condição:

$$P_1 = P_2 \wedge (|P_2 - P_3| = 1 \vee |P_2 - P_3| = 7) \qquad (7.2)$$

Este conjunto de nós é igual ao anterior mas só tem 4 nós. Por isso a fórmula é igual, mas não tem $P_3$=$P_4$.

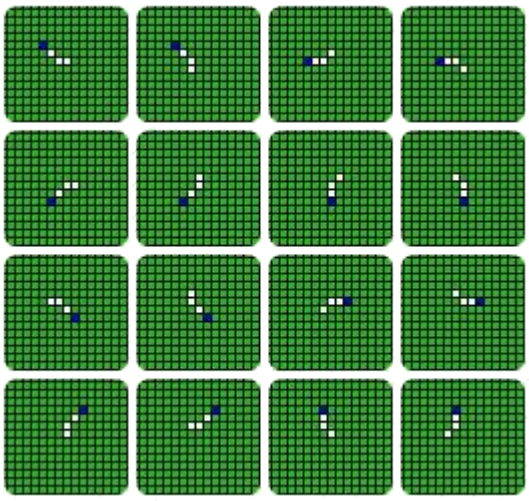


Figura 7.4 - Todas as sequências possíveis para 3 nós mais 1

Quando essas mudanças são encontradas vão-se substituir esses nós por uns novos nós em que a mudança de direcção fica mais suave. Deve-se ter em atenção que essa mudança só pode acontecer se os novos nós pertencerem à lista de nós fechados.

1. Para a primeira sequência a mudança pode-se processar de 2 formas:

a) A sequência passa a ser:

Tabela 7.2 - Nova sequência de nós para a 1ª forma tipo a

| $\mathbf{P_1}$ | $\mathbf{P_{n2}}$ | $\mathbf{P_{n3}}$ | $\mathbf{P_{n4}}$ |
|---|---|---|---|
| $\mathbf{P_3}$ | $P_2$ | $P_3$ | $P_2$ |

A primeira linha da Tabela 7.2, representa os pais dos nós da sequência nova, em que $P_{n2}$ significa pai do novo nó 2. A segunda linha representa o novo valor numérico do respectivo pai. Esta mudança é para qualquer uma das sequências encontradas deste tipo. Assim a sequência fica organizada do seguinte modo:

- O nó 1 passa a ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 2 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo.
- O nó 3 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 3 antigo
- O nó 4 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo.

- 

Na Figura 7.5 está exemplificada uma possível mudança, na parte a) está representado a sequência encontrada e na parte b) a nova sequência.

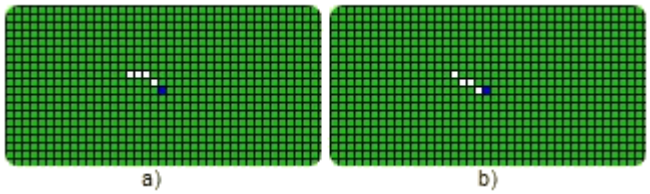



Figura 7.5 - Exemplo de uma mudança da 1º forma tipo a: a) sequência encontrada b) nova sequência

b) A nova sequência passa a ser:

Tabela 7.3 - Nova sequência de nós para a 1ª forma tipo b

| $\mathbf{P_1}$ | $\mathbf{P_{n2}}$ | $\mathbf{P_{n3}}$ | $\mathbf{P_{n4}}$ |
|---|---|---|---|
| $\mathbf{P_3}$ | $P_2$ | $P_2$ | $P_3$ |

Assim a sequência fica organizada do seguinte modo:

- O nó 1 passa a ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 2 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo.
- O nó 3 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo
- O nó 4 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 3 antigo.

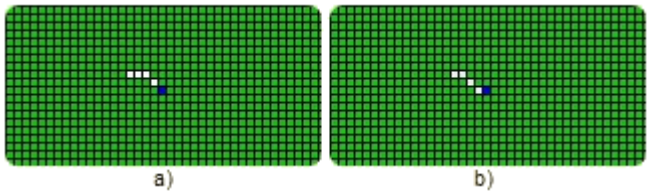



Figura 7.6 - Exemplo de uma mudança da 1ª forma tipo b: a) sequência encontrada b) nova sequência

2. Para a segunda sequência a mudança processa-se do modo seguinte.

Tabela 7.4 - Nova sequência de nós para a 2ª forma

| $\mathbf{P_1}$ | $\mathbf{P_{n2}}$ | $\mathbf{P_{n3}}$ |
|---|---|---|
| $\mathbf{P_3}$ | $P_2$ | $P_2$ |

Assim a sequência fica organizada do seguinte modo:

- O nó 1 passa a ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 2 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo.
- O nó 3 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo

Na Figura 7.7, está exemplificada uma possível mudança:

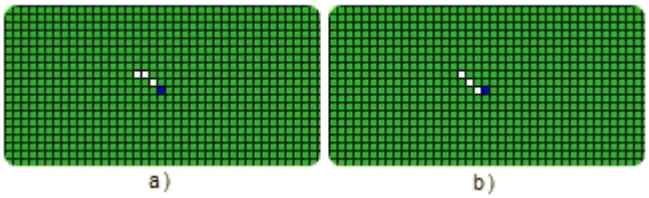



Figura 7.7 - Exemplo de uma mudança da 2ª forma: a) sequência encontrada b) nova sequência

### 7.3.2 Modificação das curvas com 7 nós (Mod7)

Na segunda implementação o número de nós vai ser maior, são 7 nós que se podem testar e verificam-se 3 conjuntos de nós:

1. Três nós numa direcção seguido de quatro nós a perfazer um ângulo de 45º como mostra a Figura 7.8. A sequência encontra-se quando se verifica o seguinte:

$$P_1 = P_2 \wedge (P_3 = P_4 = P_5 = P_6) \wedge (|P_2 - P_3| = 1 \vee |P_2 - P_3| = 7) \qquad (7.3)$$

Este conjunto de nós é igual ao anterior mas com mais um nó. Por isso a fórmula é igual, mas acrescenta-se $P_5$=$P_6$.

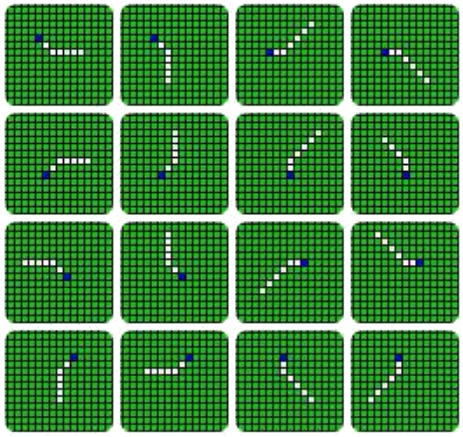


Figura 7.8 - Todas as sequências possíveis para 3 nós mais 4

2. Três nós numa direcção seguido de três nós a perfazer um ângulo de 45º como mostra a Figura 7.9. A sequência encontra-se quando se verifica o seguinte:

$$P_1 = P_2 \wedge (P_3 = P_4 = P_5) \wedge (|P_2 - P_3| = 1 \vee |P_2 - P_3| = 7) \qquad (7.4)$$

Este conjunto de nós é igual ao anterior mas com mais um nó. Por isso a fórmula é igual, mas acrescenta-se $P_4$=$P_5$.

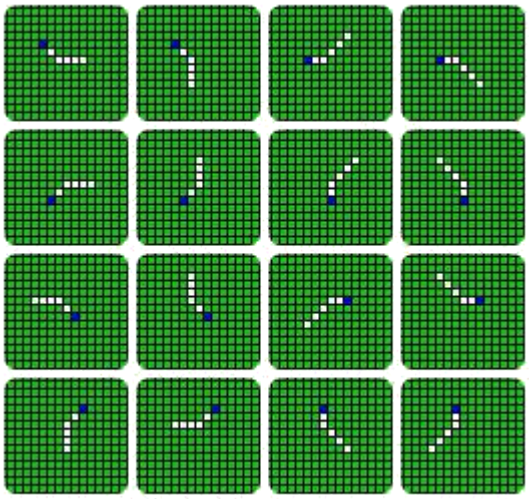


Figura 7.9 - Todas as sequências possíveis para 3 nós mais 3

3. Igual ao conjunto a) da modificação 1, isto é, três nós numa direcção seguido de dois nós a perfazer um ângulo de 45º.

Os novos nós para as sequências encontradas são:

1. Na terceira sequência possível, a nova sequência de nós vai ser:

Tabela 7.5 - Exemplo de uma mudança da 3º forma

| $\mathbf{P_1}$ | $\mathbf{P_{n2}}$ | $\mathbf{P_{n3}}$ | $\mathbf{P_{n4}}$ | $\mathbf{P_{n5}}$ | $\mathbf{P_{n6}}$ |
|---|---|---|---|---|---|
| $\mathbf{P_3}$ | $P_3$ | $P_2$ | $P_3$ | $P_3$ | $P_2$ |

- O nó 1 passa a ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 2 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 3 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo
- O nó 4 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 5 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 6 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo.

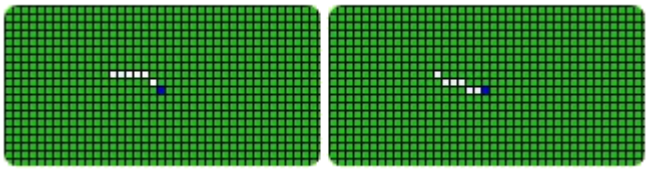


Figura 7.10 - Exemplo de uma mudança da 3º forma

2. Na segunda sequência vão existir dois tipos de modificação possíveis.
   a) A sequência passa a ser:

Tabela 7.6 - Nova sequência de nós para a 2ª forma tipo a

| $\mathbf{P_1}$ | $\mathbf{P_{n2}}$ | $\mathbf{P_{n3}}$ | $\mathbf{P_{n4}}$ | $\mathbf{P_{n5}}$ |
|---|---|---|---|---|
| $\mathbf{P_3}$ | $P_2$ | $P_3$ | $P_3$ | $P_2$ |

Assim a sequência fica organizada do seguinte modo:

- O nó 1 passa a ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 2 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo.

- O nó 3 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 3 antigo
- O nó 4 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 5 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo.

Na Figura 7.11, está exemplificada uma possível mudança.

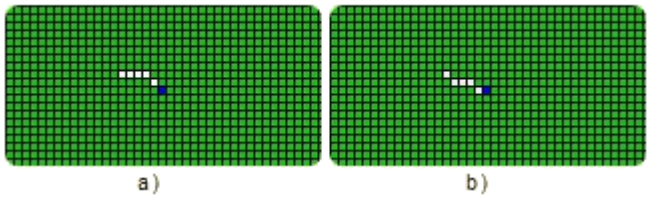



Figura 7.11 - Exemplo de uma mudança da 2º forma tipo a: a) sequência encontrada b) nova sequência

b) A sequência passa a ser:

Tabela 7.7 - Nova sequência de nós para a 2ª forma tipo b

| $\mathbf{P_1}$ | $\mathbf{P_{n2}}$ | $\mathbf{P_{n3}}$ | $\mathbf{P_{n4}}$ | $\mathbf{P_{n5}}$ |
|---|---|---|---|---|
| $\mathbf{P_3}$ | $P_3$ | $P_2$ | $P_3$ | $P_2$ |

Assim a sequência fica organizada do seguinte modo:

- O nó 1 passa a ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 2 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 3 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo.
- O nó 4 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 3 antigo.
- O nó 5 que passa a ser um novo nó, vai ter como valor numérico para o pai o valor do pai do nó 2 antigo.

Na Figura 7.12, está exemplificada uma possível mudança.

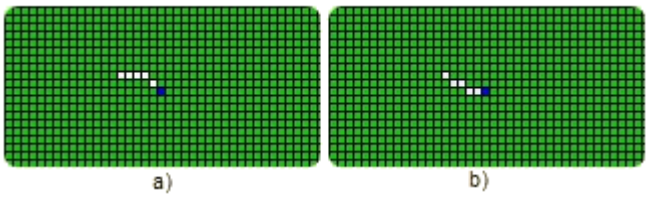



Figura 7.12 - Exemplo de uma mudança da 2ª forma tipo b: a) sequência encontrada b) nova sequência

3. A terceira sequência é igual à sequência da modificação 1.

## 7.4 Algoritmo da suavização de trajectória

Este processo é repetido até se processar uma trajectória do fim ao início sem alterações.

A diferença entre as duas modificações é que na modificação 1 o novo ângulo da mudança pode ir até 26.56º e na modificação 2 pode chegar aos 18.43º, como se pode comprovar pela Figura 7.13

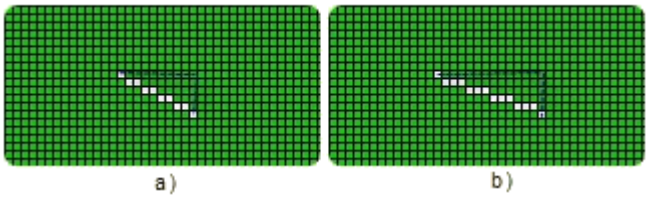



Figura 7.13 – Ângulo máximo a) da Mod5 b) da Mod7

O fluxograma do algoritmo é o seguinte:

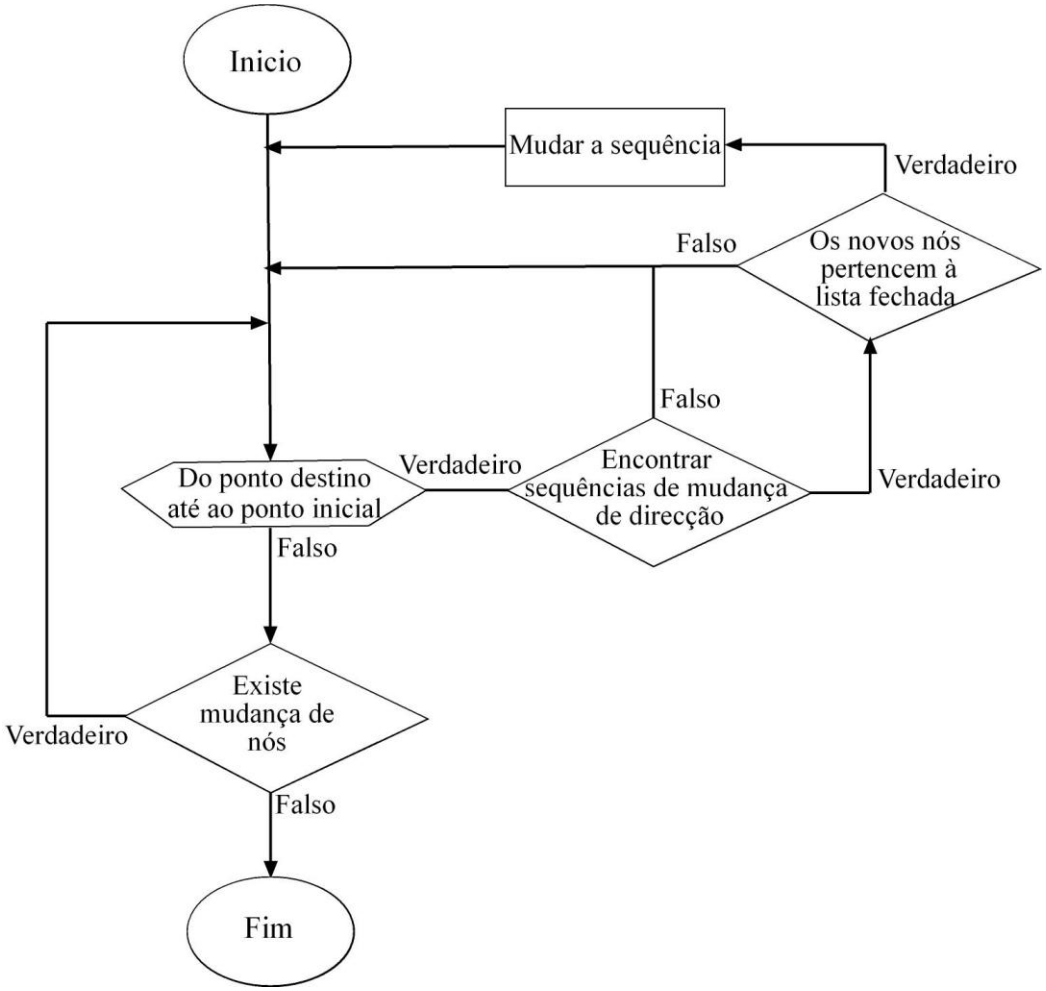



Figura 7.14 - Fluxograma do algoritmo

Na Figura 7.15 está representada a evolução da trajectória devido ao Mod7. Verifica-se que só à quarta passagem é que não houve mudança de nós. Na primeira passagem houve uma mudança da 1ª forma. Na segunda e terceira é 3ª forma que é aplicada.

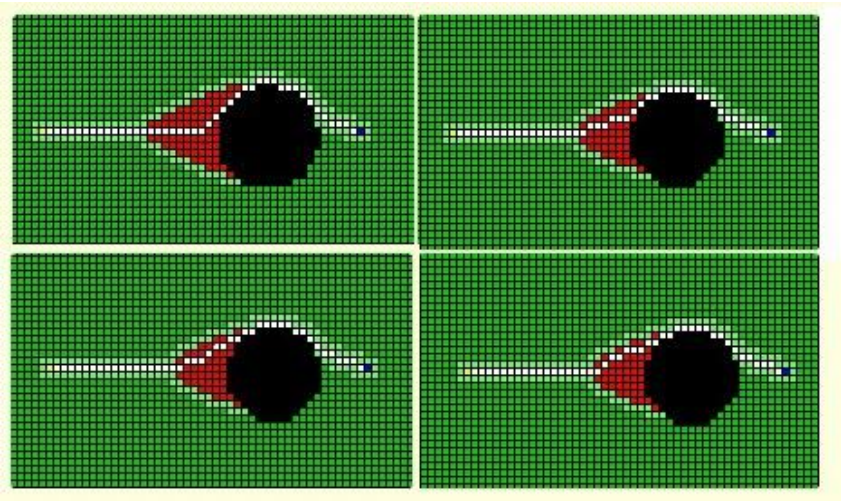


Figura 7.15 - Evolução da trajectória devido à Mod7

## 7.5 Resultados em simulação

Em seguida serão ilustradas várias simulações para verificar a sua validade. São apresentadas 4 simulações que tentam representar as principais relevâncias desta alteração. Servem também para decidir qual a modificação que melhor resultado obtém.

### 7.5.1 1º Situação – Obstáculo parado

O robô tem de passar por um obstáculo que se encontra parado no meio de caminho

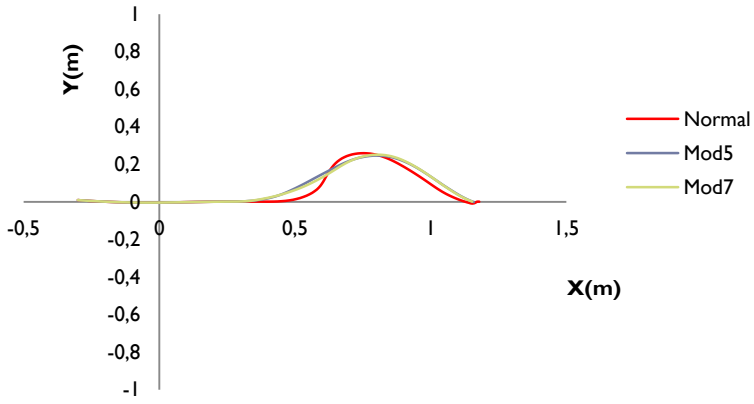



Gráfico 7.3 - Trajectória executada na 1ª situação

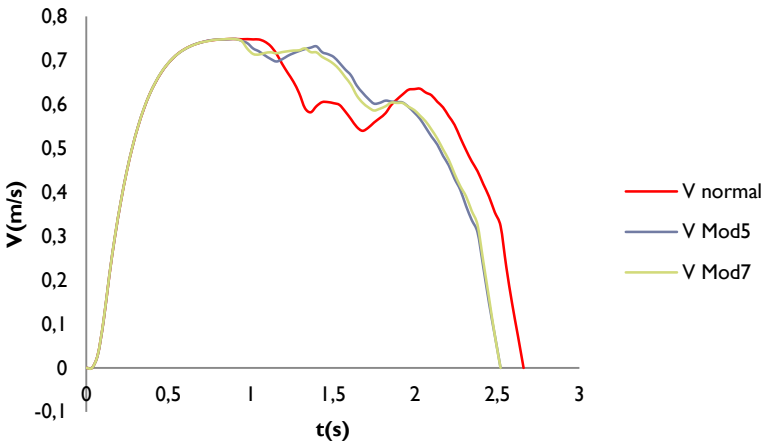



Gráfico 7.4 - Velocidades do robô na 1ª situação

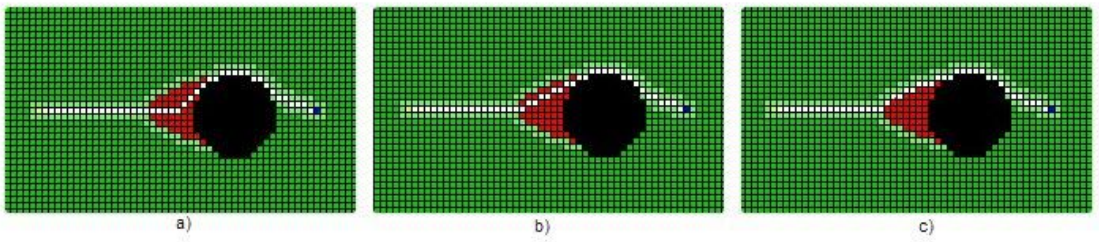



Figura 7.16 - Trajectórias geradas no primeiro instante a) Normal b) Mod5 c) Mod7

Tabela 7.8 - Resultados da 1ª situação

| | **Tempo médio de processamento (ms)** | **Duração (s)** | **Colisões** |
|---|---|---|---|
| **Normal** | 0.09 | 2.640 | 0 |
| **Mod5** | 0.10 | 2.404 | 0 |
| **Mod7** | 0.10 | 2.401 | 0 |

Verifica-se pela Figura 7.16 que a trajectória tinha inicialmente uma mudança de direcção de 45º e passou para cerca 26º no Mod5 e para o Mod7 é ligeiramente mais inclinada no início. Na 2ª modificação o robô começou a evitar o obstáculo mais cedo, como consequência a velocidade diminuiu pouco e o tempo de chegada ao destino foi mais rápido cerca de 230ms para ambas as modificações. O tempo de processamento manteve-se praticamente inalterável, ou seja, consegue-se melhorar a rapidez do caminho sem perder tempo de processamento em ambas as modificações.

### 7.5.2 2º Situação – Obstáculo parado com trajectória na diagonal

O robô tem de passar por um obstáculo que se encontra parado no meio de caminho, só que o trajecto não é na horizontal mas na diagonal.

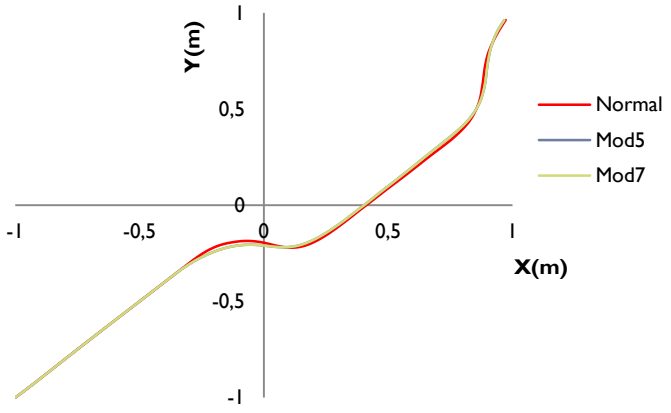



Gráfico 7.5 - Trajectória executada na 2ª situação

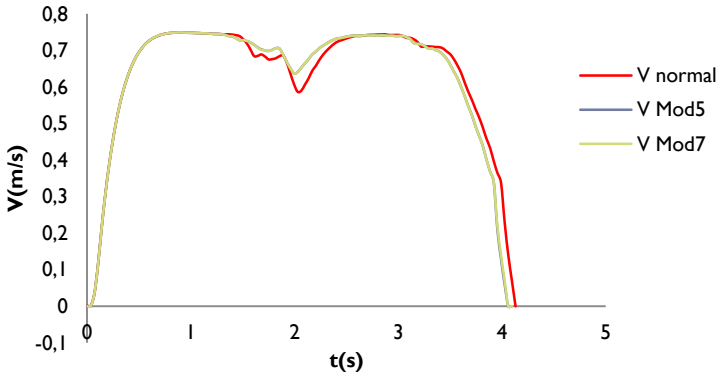



Gráfico 7.6 - Velocidades do robô na 2ª situação

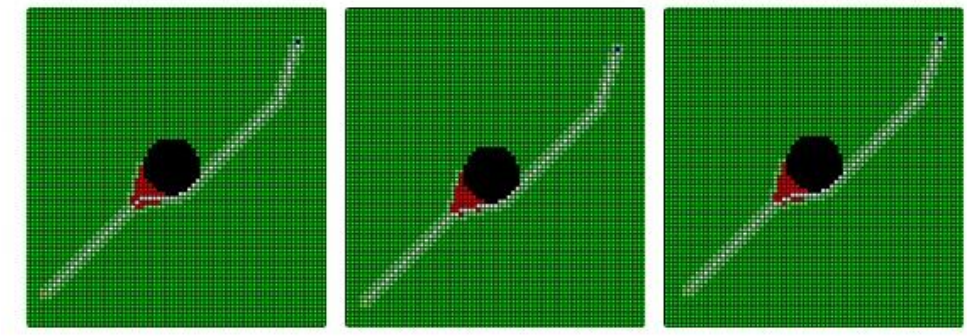


Figura 7.17 - Trajectórias geradas no primeiro instante a) Normal  b) Mod5 c) Mod7

Tabela 7.9 - Resultados da 2ª situação

| | Tempo médio de processamento (ms) | Duração(s) | Colisões |
|---|---|---|---|
| **Normal** | 0.07 | 3.931 | 0 |
| **Mod5** | 0.07 | 3.857 | 0 |
| **Mod7** | 0.07 | 3.858 | 0 |

Ao utilizar uma trajectória na diagonal verifica-se que não se consegue suavizar tanto como na horizontal em virtude de existirem menos nós fechados. Ocorreu uma ligeira melhoria na duração do trajecto, mas em menor escala do que na situação anterior. Ambas as modificações comportaram-se de igual modo, como se pode ver pela trajectória (Gráfico 7.5) como pela duração da trajectória.

### 7.5.3 3ª Situação – Obstáculo parado com alteração da direcção activa

Este caso serve para se analisar a alteração de indicar uma direcção no ponto de destino em simultâneo com a suavização da trajectória

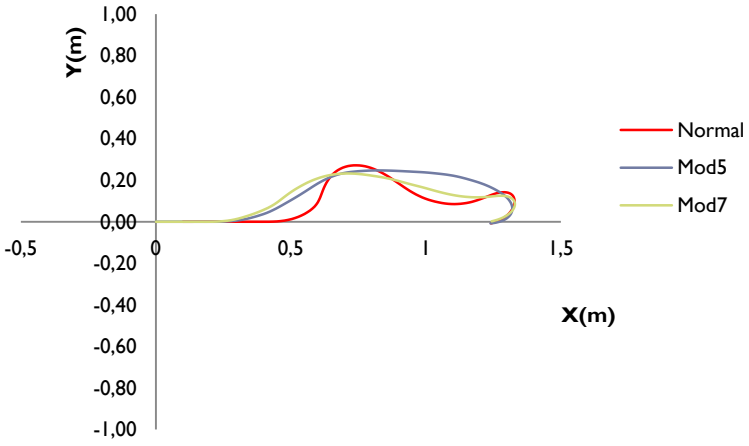



Gráfico 7.7 - Trajectória executada na 3ª situação

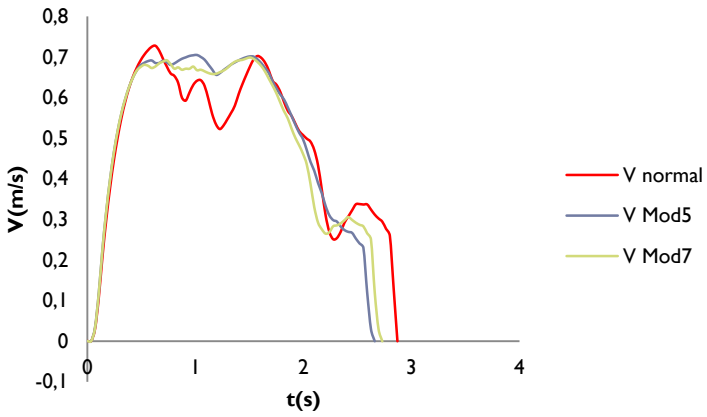



Gráfico 7.8 - Velocidades do robô na 3ª situação

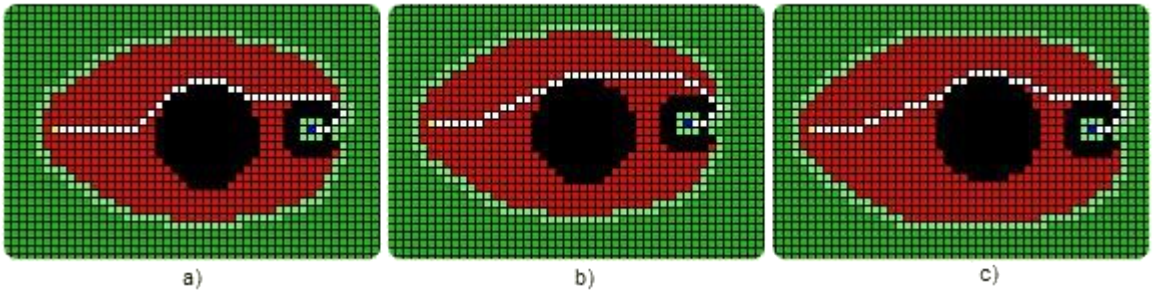



Figura 7.18 - Trajectórias geradas no primeiro instante a) Normal b) Mod5 c) Mod7

Tabela 7.10 - Resultados da 3ª situação

| | Tempo médio de processamento (ms) | Duração (s) | Colisões |
|---|---|---|---|
| **Normal** | 0.07 | 2.827 | 0 |
| **Mod5** | 0.07 | 2.578 | 0 |
| **Mod7** | 0.07 | 2.620 | 0 |

Como se constata na Figura 7.19 a trajectória foi modificada em duas ocasiões e como consequência a duração do caminho baixou muito, e as trajectórias ficaram muito diferentes uma da outra. Nesta situação a Mod5 obteve melhores resultados que a Mod7, em virtude de conseguir fazer uma aproximação mais eficaz ao ponto de chegada depois de passar o obstáculo.

### 7.5.4 4º Situação - 5 obstáculos

Nesta situação existem 5 obstáculos mas com o obstáculo assinalado a vermelho ligeiramente mais a baixo, para obrigar a mudar a trajectória. Este exemplo vai servir para mostrar que juntamente com as alterações do capítulo anterior, os resultados serão melhores.

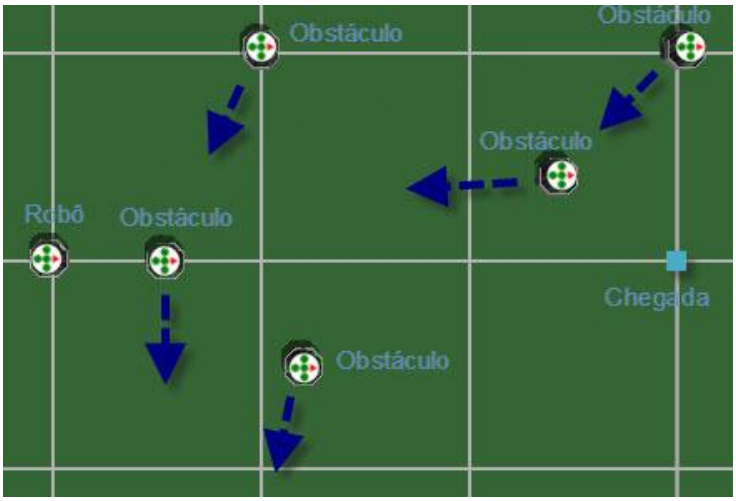



Figura 7.19 - Imagem da posições iniciais da 4ª situação

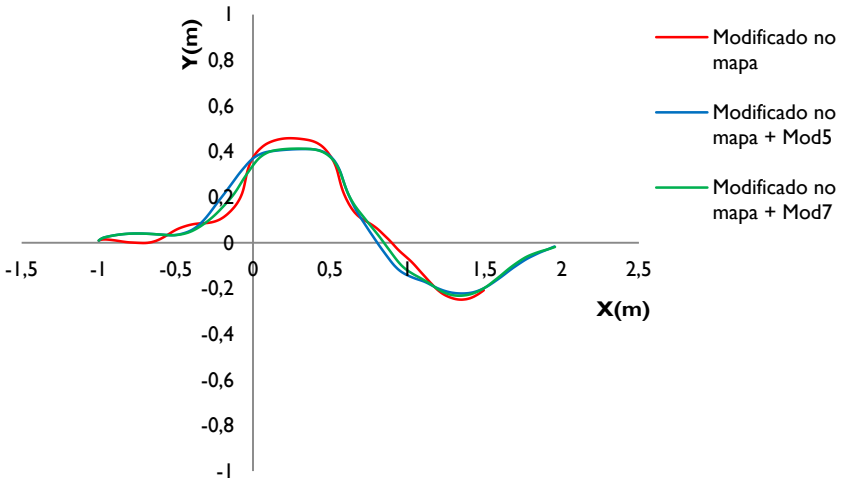



Gráfico 7.9 - Trajectória executada na 4ª situação

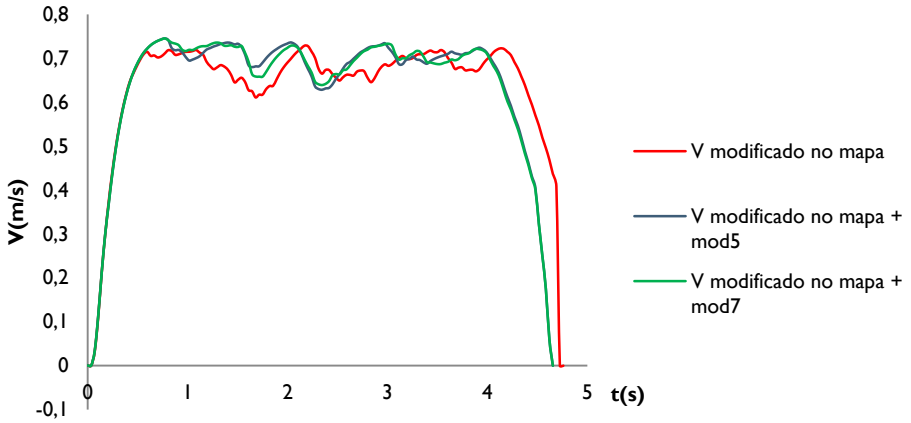



Gráfico 7.10 - Velocidades do robô na 4ª situação

Tabela 7.11 - Resultados da 4ª situação

| | Tempo médio de processamento (ms) | Duração(s) | Colisões |
|---|---|---|---|
| **Modificações no mapa** | 0.34 | 4.603 | 0 |
| **Modificações no mapa + Mod5** | 0.34 | 4.449 | 0 |
| **Modificações no mapa + Mod7** | 0.34 | 4.451 | 0 |

A rapidez de execução melhora quando se utilizam todas as alterações desenvolvidas. A velocidade ao longo do trajecto, como se constata na Gráfico 7.10, diminui pouco, e consequentemente o robô chega muito mais rápido ao destino. Não existem diferenças significativas entre a Mod5 e a Mod7. O tempo de processamento continua a ser insignificante.

## 7.6 Resultados com robôs reais

Tal como no capítulo anterior, nesta secção foram utilizados os robôs reais para validar os resultados. Serão apresentadas 3 situações que representam melhor as melhorias introduzidas. Cada caso foi executado 5 vezes, a duração é a média das cinco e o gráfico é da que está mais próximo da média. Em todos os casos utilizaram-se as alterações ao $C_{espaço}$ e a suavização de trajectória Mod5 e no CD em anexo estão os vídeos..

### 7.6.1 1ª situação – Obstáculo parado

No meio da trajectória o robô tem um obstáculo parado

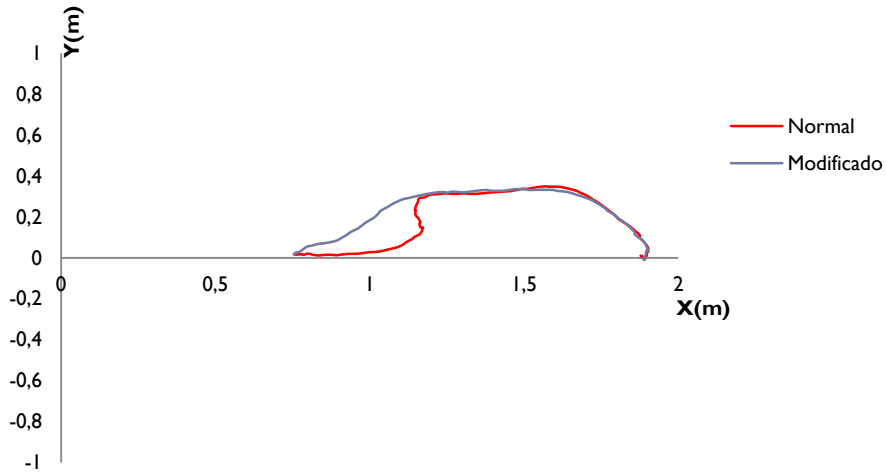



Gráfico 7.11 - Robôs reais - 1ª situação - Trajectórias executadas

Tabela 7.12 - Robôs reais 1ª situação - Resultados

| | Duração (s) | Ganho | Colisões |
|---|---|---|---|
| **Normal** | 3.080 | | 0 |
| **Modificado** | 2.509 | 18.5% | 0 |

Nesta situação existe uma melhoria significativa, a suavização da trajectória faz com que a trajectória fique mais larga e como consequência o robô consegue executar com maior velocidade, o que permite uma duração menor. Houve um ganho em relação às alterações do $C_{espaço}$, do capitulo anterior, de cerca de 4.9%.

### 7.6.2 2ª situação – Obstáculo em direcção ao robô

No meio da trajectória o robô tem um obstáculo a dirigir-se em sentido contrário com velocidade de 1 $ms^{-1}$ .

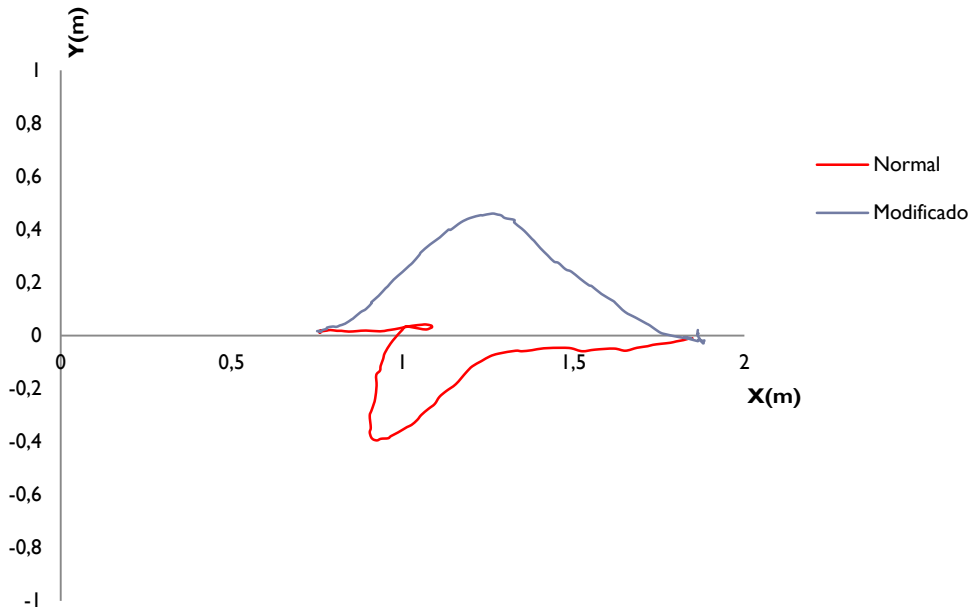



Gráfico 7.12 - Robôs reais - 2ª situação - Trajectórias executadas

Tabela 7.13 - Robôs reais 2ª situação - Resultados

| | Duração (s) | Ganho | Colisões |
|---|---|---|---|
| **Normal** | 4.187 | | 1 |
| **Modificado** | 2.673 | 36.2% | 0 |

Com a suavização o robô consegue evitar obstáculos com velocidades superiores a 1 $ms^{-1}$ . O ganho é elevado mas é consequência da colisão que acontece sem as alterações. Por isso a grande vantagem é que se consegue evitar colisões com obstáculos com a mesma ordem de grandeza de velocidade.

### 7.6.3 3ª situação – Obstáculo a intersectar a trajectória do robô

No meio da trajectória o robô tem um obstáculo a passar à frente do robô com velocidade de 0.8 $ms^{-1}$ .

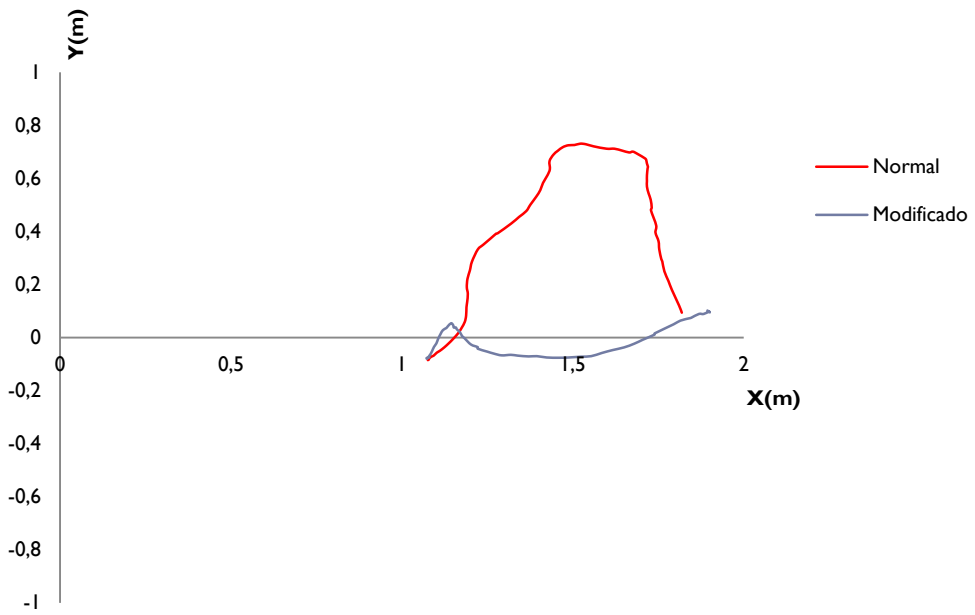



Gráfico 7.13 - Robôs reais - 3ª situação - Trajectórias executadas

Tabela 7.14 - Robôs reais 3ª situação - Resultados

| | **Duração (s)** | **Ganho** | **Colisões** |
|---|---|---|---|
| **Normal** | 2.963 | | 0 |
| **Modificado** | 2.123 | 28.3% | 0 |

Esta situação é igual à 3ª do capitulo anterior, o ganho é ligeiramente superior com a introdução da suavização do que só com as modificações do $C_{espaço}$.

## 7.7 Conclusão

Com pouco esforço computacional foi possível melhorar a qualidade da trajectória em termos de rapidez, tanto para o Mod5 como para o Mod7. O tempo de processamento é praticamente nulo para ambas modificações. Com estas modifcações consegue-se garantir que as novas trajectórias evitam obstáculos e mantêm velocidades mais elevadas.

Só na 3ª situação é que o Mod5 é ligeiramente melhor que o Mod7. Nas restantes situações ambos portam-se de igual modo. Por isso, o Mod5 é o eleito para ser utilizada.

Neste processo de suavização de trajectórias, quanto mais nós fechados existirem melhor vai ser essa suavização, uma vez que só existe suavização se os novos nós gerados pretencerem aos nós fechados.

Os resultados demonstrados tanto na simulação como com os robôs reais permitem afirmar que essas alterações são suficientes para uma melhoria significativa.

# Capítulo 8

# Conclusões

Neste trabalho foi abordado o problema do planeamento de trajectórias num ambiente dinâmico em tempo real. A plataforma de teste utilizada foi o futebol robótico devido à sua complexidade e exigência. Trata-se de uma situação em que existem vários robôs para controlar e vários obstáculos a contornar com dinâmicas imprevisíveis. A tese foi estruturada de forma a apresentar no final de cada capítulo as conclusões dos resultados obtidos. Será agora apresentada uma síntese do trabalho realizado e as suas conclusões principais:

Foi estudado o desenvolvimento e a implementação de um algoritmo de planeamento de trajectórias num ambiente dinâmico em tempo real. A abordagem adoptada foi a decomposição em células aproximadas, com a utilização do algoritmo de pesquisa A*. O tempo médio de execução do algoritmo é da ordem de grandeza das poucas décimas de milissegundos, ou seja, tempos suficientemente baixos para que o algoritmo possa ser implementado em tempo real. No caso estudado o software que faz de "treinador" necessita de efectuar o planeamento de 10 trajectórias em cada instante. Também nesta situação os tempos de execução do algoritmo não influenciam o desempenho da solução final. Com esta implementação rebate-se o mito de que o algoritmo A* é muito lento, e demonstra-se claramente que é um algoritmo com potencial para ser usado em muitas outras aplicações. O algoritmo A*, com estas modificações desenvolvidas, pode ser usado com o mesmo potencial que outros algoritmos desenvolvidos mais recentemente como são os casos dos algoritmos D*, D* *lite*, E* e AD*.

Foi introduzido um simulador neste trabalho. Esta decisão teve como principal objectivo proporcionar um ambiente estável de desenvolvimento e teste, que não afecte o objecto do estudo. Foi ainda possível validar o simulador de modo a que este possa ser usado como ferramenta de teste. Fica assim provado que os simuladores realistas são uma ferramenta cada vez mais indispensável para estes trabalhos.

Foram desenvolvidas e implementadas várias modificações na abordagem utilizada, com o objectivo de melhorar o desempenho dos robôs no caso concreto do futebol robótico. Essas modificações foram executadas no $C_{espaço}$ gerado pela decomposição em células aproximadas e nos custos da heurística usada no algoritmo A*. As modificações introduzidas foram 5:

- introdução dos pontos de colisão

- modificação do rasto
- modificação da distância
- modificação da folga
- modificação da direcção

Estas modificações visam essencialmente o seguinte:

- Entrar com a informação da velocidade dos obstáculos no $C_{espaço}$ - isso foi conseguido com a introdução dos pontos de colisão e a modificação do rasto;
- Dar pouca importância às áreas que estão longe do robô. Com ambientes tão dinâmicos como o futebol robótico não vale a pena estar preocupado com o que está afastado porque rapidamente o $C_{espaço}$ é alterado. Esta característica foi conseguida com a introdução da modificação da distância, que indica essencialmente que os obstáculos a partir de uma certa distância deixam de ser considerados como obstáculos. Com isso consegue-se reduzir o tempo de execução do planeamento.
- Dar uma folga aos obstáculos de modo a evitar ainda mais as colisões, sem prejudicar o planeamento de trajectórias. Isso foi conseguido com a modificação da folga, em que à volta do obstáculo os custos da heurística eram maiores, por isso a trajectória normalmente passa ao lado dessas células mas se for necessário também consegue passar por essas células.
- A introdução da modificação da direcção faz com que se possa indicar por que lado se pretende chegar ao ponto de destino. Esta situação é conseguida com a introdução de obstáculos à volta do ponto de destino excepto na direcção pretendida. Esta alteração não melhora a solução em relação ao tempo de execução da trajectória e ao tempo de processamento mas permite novas possibilidades ao gerar trajectórias.

Conseguiu-se com a implementação destas alterações à abordagem utilizada melhorar as trajectórias no aspecto do tempo de execução e principalmente nas colisões. O tempo de processamento, como consequência da implementação das alterações, foi aumentado mas ligeiramente.

Estas modificações não são exclusivas do algoritmo A*, visto que não são alterações ao algoritmo A*, mas sim ao $C_{espaço}$ e aos custos das heuristicas. Por isso podem ser implementas em qualquer algoritmo que utilize este tipo de $C_{espaço}$. Isto é, os algoritmos D*, D* *lite*, o E*, AD* e outros desta familia podem utilizar estas modificações com o mesmo sucesso aqui verificado.

Foi ainda desenvolvido e implementado um algoritmo para suavizar as trajectórias geradas, com a intenção de conseguir dotar as trajectórias de percursos capazes de serem executados por robôs com velocidades elevadas. Como em alguns casos a trajectória gerada não era totalmente seguida pelo robô por causa da sua dinâmica, a introdução deste algoritmo conseguiu fazer com que o robô acompanhasse melhor a trajectória o que se traduz em ganhos consideráveis no tempo de execução. O algoritmo tem um tempo de processamento muito baixo, o que não afecta o tempo total de processamento da trajectória.

Foram compravadas todas melhorias resultantes das alterações implementadas e do algoritmo desenvolvido para suavizar as trajectórias com a utilização dos robôs da liga pequena, ou seja, com a utilização de robôs reais. Essa implementação serviu igualmente para consolidar a validação do simulador.

## 8.1 Trabalhos futuros

Para o futuro existem dois caminhos que se podem seguir: implementar a solução em ambientes similares ao estudado e verificar até que ponto é que em ambientes totalmente diferentes o algoritmo funciona. O trabalho a desenvolver no futuro passa assim por:

- Implementar a solução na equipa de robôs da liga média. Neste caso o ambiente é um ambiente similar ao do trabalho realizado.
- Introduzir a solução em ambientes com espaços e mapas desconhecidos, em que o ambiente vai sendo reconhecido através dos sensores. Pretende-se assim verificar até que ponto o algoritmo A* e as modificações introduzidas dão resposta a este problema.
- Verificar até que ponto o algoritmo funciona em $C_{espaços}$ com graus de liberdade mais elevados, como por exemplo em manipuladores, Neste caso pretende-se verificar até que ponto o tempo de execução deixa de ser aceitável, visto que vão existir muitos mais nós para processar.
- Implementar no espaço de configuração o $\theta$, ou seja, verificar até que ponto introduz melhorias o facto de se considerar a direcção do robô.

# Referencias

[1] Khantanapoka, K. and K. Chinnasarn. *Pathfinding of 2D & 3D game real-time strategy with depth direction A* algorithm for multi-layer*. in *Natural Language Processing, 2009. SNLP '09. Eighth International Symposium on*. 2009:

[2] Fischer, L.G., R. Silveira, and L. Nedel. *GPU Accelerated Path-Planning for Multi-agents in Virtual Environments*. in *Games and Digital Entertainment (SBGAMES), 2009 VIII Brazilian Symposium on*. 2009.

[3] Yao, J., et al., *Path Planning for Virtual Human Motion Using Improved A* Star Algorithm.* Information Technology: New Generations, Third International Conference, 2010: p. 1154-1158.

[4] Xiong, M., et al., *A Rule-Based Motion Planning for Crowd Simulation.* Cyberworlds, International Conference 2009: p. 88-95.

[5] Cortes, J., *A path planning approach for computing large-amplitude motions of flexible molecules.* Bioinformatics, 2005. **21**: p. 116-125.

[6] Kypson, A.P. and L. Wiley Nifon, *Robotic Cardiac Surgery.* Journal of Long-Term Eff ects of Medical Implants, 2003. **13**(6): p. 451-464.

[7] Lozano-Pérez, T. and M.A. Wesley, *An algorithm for planning collision-free paths among polyhedral obstacles.* Communications of the ACM, 1979. **22**: p. 560-570.

[8] Hwang, Y.K. and N. Ahuja, *Gross Motion Planning - a survey.* ACM Computing Surveys, 1992. **24**: p. 219-291.

[9] Lozano-Pérez, T., *Spatial planning: a configuration space approach.* IEEE transactions on computers, 1983. **C-32**: p. 108-120.

[10] Ge, Q. and J.M. McCarthy, *Equations for boundaries of joint obstacles for planar robots*, in *IEEE international conference on robotics and automation*. 1989. p. 164-169.

[11] Hwang, Y.K., *Boundary equations of configuration obstacles for manipulators*, in *IEEE international conference on robotics ans automation*. 1990. p. 298-303.

[12] Gupta, K.K. and Z. Guo, *Motion planning for many degrees of freedom: sequencial search with backtracking*, in *IEEE international conference on robotics ans automation*. 1992. p. 2328-2333.

[13] Lozano-Pérez, T., *A simple motion-planning algorithm for general robot manipulators.* IEEE Journal of Robotics and Automation, 1987. **RA-3(3)**: p. 224-238.

[14] Branicky, M.S. and W.S. Newman. *Rapid computation of configuration space obstacles*. in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*. 1990.

[15] Connolly, C.I., J.B. Burns, and R. Weiss. *Path planning using Laplace's equation*. in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*. 1990.

[16] James, N., *An Analysis of Mobile Robot Navigation Algorithms in Unknown Environments*, in *School of Electrical, Electronic and Computer Engineering*. 2010, The University of Western Australia.

[17] Lumelsky, V. and A. Stepanov, *Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shapes.* Algorithimica, 1987. **2**: p. 403-430.

[18] Kamon, I., E. Rimon, and E. Rivlin, *Tangentbug: a range-sensor based navigation algorithm.* International journal of robotics research, 1998. **17(9)**: p. 934-953.

[19] Kamon, I. and E. Rivlin, *Sensory-based motion planning with global proofs.* IEEE transactions on robotics ans automation, 1997. **13**: p. 814-822.

[20] Antich, J., A. Ortiz, and J. Minguez. *A Bug-inspired algorithm for efficient anytime path planning*. in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. 2009.

[21] Canny, J.F., *Constructing roadmaps of semi-algebraic sets I: completeness.* Artificial intelligence, 1988. **37**: p. 203-222.

[22] Latombe, J.C., *Robot motion planning*. 1991, Boston, MA: Kluwer Academic Publishers.

[23] Asano, T., et al. *Visibility-polygon search and euclidean shortest paths*. in *Foundations of Computer Science, 1985., 26th Annual Symposium on*. 1985.

[24] Aurenhammer, F., *Voronoi diagrams - a survey of a fundamental geometric structure.* ACM Computing Surveys, 1991. **23**: p. 345-405.

[25] Canny, J., *A Voronoi method for the piano-movers problem*, in *IEEE international conference on robotics and automation*. 1985. p. 530--535.

[26] Preparata, F.P. and M.I. Shamos, *Computational Geometry: An Introduction*. 1985, New York: Spinger-Verlag.

[27] Ahuja, N. and B.J. Schachter, *Pattern models*. 1983, New York: Wiley.

[28] Kirkpatrick, D.G. *Efficient computation of continuous skeletons*. in *Foundations of Computer Science, 1979., 20th Annual Symposium on*. 1979.

[29] Drysdale, D.T. and R.L. Lee, *Generalization of Voronoi Diagrams in the Plane.* SIAM Journal on Computing, 1981. **10**(1): p. 73-87.

[30] Takahashi, O. and R.J. Schilling, *Motion planning in a plane using generalized Voronoi diagrams*, in *IEEE international conference on robotics and automation*. 1989. p. 143-150.

[31] Canny, J.F., *The complexity of robot motion planning.* MIT Press, 1988.

[32] Canny, J.F. and C. Lin, *An opportunistic global path planner.* Algorithimica, 1993. **10**: p. 102-120.

[33] Faverjon, B. and P. Tournassoud. *A local based approach for path planning of manipulators with a high number of degrees of freedom*. in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*. 1987.

[34] Chen, P.C. and Y.K. Hwang, *SANDROS: a motion planner with performance proportional to task dificulty*, in *IEEE international conference on robotics ans automation*. 1992. p. 2346-2353.

[35] Warren, C., *A vector based approach to robot path planning*, in *IEEC international conference on robotics and automation*. 1991. p. 1021-1026.

[36] Kavraki, L.E., et al., *Probabilistic roadmaps for path planning in high-dimensional configuration spaces.* IEEE transactions on robotics ans automation, 1996. **12(4)**: p. 566-580.

[37] Kavraki, L.E., et al., *Randomized query processing in robot motion planning*, in *ACM symposium on theory of computing*. 1995. p. 353-362.

[38] Kavraki, L.E., et al., *Randomized query processing in robot path planning.* Journal of computer and systems sciences, 1998. **57(1)**: p. 50-60.

[39] Overmars, M.H., *A random approach to motion planning*. 1992, Utrecht University: The Netherlands.

[40] Overmars, M. and P. Svestka, *A probabilistic learning approach to motion planning*, in *Algorithmic foundations of robotics (WAFR)*, D.H. K. Goldberg, J. C. Latombe and R. Wilson, Editor. 1995, A. K. Peters. Ltd. p. 19-37.

[41] Svestka, P., *A probabilistic approach to motion planning for car-like robots*. 1993, Utrecht University: Utrecht, the Netherlands.

[42] Kavraki, L.E., *Random networks in configuration space for fast path planning*. 1995, Stanford University: Stanford.

[43] Kavraki, L.E., M. Kolountzakis, and J.C. Latombe, *Analysis of probabilistic roadmaps for path planning*, in *IEEE international conference on robotics and automation*. 1996. p. 302-3026.

[44] Kavraki, L.E. and J.C. Latombe, *Probabilistic roadmaps for robot path planning*, in *Pratical motion planning in robotics: current aproaches and future challenges*, K.G.a.A.P.d. probil, Editor. 1998, John Wiley: West Sussex, England. p. 33-53.

[45] Amato, N.M., et al., *OBPRM: An obstacle-based PRM for 3D workspaces.* In Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR), 1998: p. 155-168.

[46] Boor, V., M.H. Overmars, and A.F. van der Stappen, *The Gaussian sampling strategy for probabilistic roadmap planners.* In Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 1999: p. 1018–1023.

[47] Hsu, D., G. Sanchez-Ante, and Z. Sun, *Hybrid PRM sampling with a cost-sensitive adaptive strategy.* In Proc. IEEE Int. Conf. Robot Autom., 2005.

[48] Wilmarth, S.A., N.M. Amato, and P.F. Stiller, *MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space.* In Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 1999: p. 1024–1031.

[49] Hsu, D., et al. *Multi-level free-space dilation for sampling narrow passages in PRM planning*. in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. 2006.

[50] Saha, M. and J.C. Latombe, *Finding narrow passages with probabilistic roadmaps: The small step retraction method.* in Proc. IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, 2000.

[51] Simeon, T., J. Laumond, and C. Nissoux, *Visibility-based probabilistic roadmaps for motion planning.* J. Advanced Robotics, 2000. **14**(6): p. 477–494.

[52] Burns, B. and O. Brock, *Sampling- based motion planning using predictive models.* in Proc. IEEE Int. Conf. on Robotics and Automation, 2005.

[53] Hsu, D., J.C. Latombe, and R. Motwani, *Path planning in expansive configuration spaces*, in *IEEE conference on robotics and automation*. 1997. p. 2719-2726.

[54] Bohlin, R. and L.E. Kavraki, *Path planning using Lazy PRM.* In Proc. IEEE Int. Conf. Robot Autom. (ICRA), 2000: p. 521-528.

[55] Sanchez, G. and J.C. Latombe, *On delaying collision checking in PRM planning – application to multi-robot coordination.* Int. J. of Robotics Research 2002. **21**(1): p. 5–26.

[56] Jaillet, L. and T. Simeon. *A PRM-based motion planner for dynamically changing environments*. in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. 2004.

[57] Kuffner, J.J. and S.M. LaValle, *RRT-connect: an efficient approach to single-query path planning*, in *IEEE international conference on robotics and automation*. 2000. p. 995-1001.

[58] Liu, C.-a., et al. *Mobile robot path planning based on an improved rapidly-exploring random tree in unknown environment*. in *Automation and Logistics, 2008. ICAL 2008. IEEE International Conference on*. 2008.

[59] Kuwata, Y., et al., *Real-Time Motion Planning With Applications to Autonomous Urban Driving*. Control Systems Technology, IEEE Transactions on, 2009. **17**(5): p. 1105-1118.

[60] Bruce, J. and M. Veloso. *Real-time randomized path planning for robot navigation*. in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. 2002.

[61] Fergunson, D., N. Kalra, and A. Stentz, *Replanning with RRTs.* Robotics Institute Carnegie Mellon University, 2009.

[62] Balakirsky, S. and D. Dimitrov. *Single-query, Bi-directional, Lazy roadmap planner applied to car-like robots*. in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. 2010.

[63] Arimoto, F. and S. Miyazaki, *Sensory feedback based on the artificial potential for robots.* Budapest :Proc. 9th IFAC, 1984.

[64] Pavlov, V.V. and A.N. Voronin, *The method of potential functions for coding of constraints of the external space in a intelligent mobile robot.* Soviet Automat Control 1984. **6**.

[65] Khatib, O., *Real-time obstacle avoidance for manipulators and mobile robots.* International journal of robotics research, 1986. **5**: p. 90-98.

[66] Su, W., R. Meng, and C. Yu. *A Study on Soccer Robot Path Planning with Fuzzy Artificial Potential Field*. in *Computing, Control and Industrial Engineering (CCIE), 2010 International Conference on*. 2010.

[67] Krogh, B.H., *A generalised potential field approach to obstacle avoidance control.* Proceedings of SME Conference on Robotics Research, 1984: p. Paper No. MS84-484.

[68] De Medio, C. and G. Oriolo, *Robot obstacle avoidance using vortex fields.* In: Advances in Robot Kinematics, Stifter S. & Lenarcic J. (Ed.), 1991: p. 227–235.

[69] Canny, J.F., *The complexity of robot motion planning.* MIT Press, 1998.

[70] Rimon, E. and D.E. Koditschek, *Exact robot navigation using artificial potential functions.* IEEE Transaction on Robotic and Automation, 1992. **8**(5): p. 501-518.

[71] Borenstein, J. and Y. Koren, *Real-time obstacle avoidance for fact mobile robots.* Systems, Man and Cybernetics, IEEE Transactions on, 1989. **19**(5): p. 1179-1187.

[72] Jen-Hui, C. *Potential-based modeling of two dimensional workspace using several source distributions*. in *Multisensor Fusion and Integration for Intelligent Systems, 1994. IEEE International Conference on MFI '94.* 1994.

[73] Peng, H., Y. Peng, and L. Zuojun. *Robot soccer path planning research based on predictive artificial potential field*. in *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on*. 2004.

[74] Xinying, X., X. Jun, and X. Kerning. *Path Planning and Obstacle-Avoidance for Soccer Robot Based on Artificial Potential Field and Genetic Algorithm*. in *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*. 2006.

[75] Hongyan, S., et al. *Chaotic Potential Field Method and Application in Robot Soccer Game*. in *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*. 2006.

[76] Sakamoto, K. and A. Kawamura. *Trajectory planning using optimum solution of variational problem*. in *Power Conversion Conference, 1993. Yokohama 1993., Conference Record of the*. 1993.

[77] Johnson, C.G. and D. Marsh. *Modelling robot manipulators in a CAD environment using B-splines*. in *Intelligence and Systems, 1996., IEEE International Joint Symposia on*. 1996.

[78] Bazaz, S.A. and B. Tondu. *3-cubic spline for online Cartesian space trajectory planning of an industrial manipulator*. in *Advanced Motion Control, 1998. AMC '98-Coimbra., 1998 5th International Workshop on*. 1998.

[79] Vaz, A.I.F., E.M. Fernandes, and M.P.S. Gomes, *Robot trajectory planning with semi-infinite programming.* European Journal of Operational Research, 2004. **153**(3): p. 607-617.

[80] Lengagne, S., et al. *Generation of dynamic motions under continuous constraints: Efficient computation using B-Splines and Taylor polynomials*. in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. 2010.

[81] Fiorini, P. and Z. Shiller, *Motion planning in dynamic environments using velocity obstacles.* Int. Journal of Robotics Research, 1998. **17**(7): p. 760–772.

[82] Abe, Y. and M. Yoshiki, *Collision avoidance method for multiple autonomous mobile agents by implicit cooperation*, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 01)*. 2001: New York, N.Y. p. 1207–1212.

[83] Fulgenzi, C., A. Spalanzani, and C. Laugier, *Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid*, in *IEEE International Conference on Robotics and Automation (ICRA 07)*. 2007: New York, N.Y. p. 1610–1616.

[84] Berg, J., M. Lin, and D. Manocha, *Reciprocal velocity obstacles for real-time multi-agent navigation*, in *IEEE International Conference on Robotics and Automation (ICRA 08)*. 2008: New York, N.Y. p. 1928–1935.

[85] Wilkie, D., J. Berg, and D. Manocha, *Generalized velocity obstacles*, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009: New York, N.Y.

[86] Berg, J.v.d., M.C. Lin, and D. Manocha. *http://gamma.cs.unc.edu/RVO/*.

[87] Zacksenhouse, M., R.J.P. deFigueiredo, and D.H. Johnson. *A neural network architecture for cue-based motion planning*. in *Decision and Control, 1988., Proceedings of the 27th IEEE Conference on*. 1988.

[88] Canny, J.F., *Some algebraic and geometric computations in PSPACE*, in *20th ACM sysmposium on the theory of computing*. 1998. p. 460-469.

[89] Zhao, S. and M. Li. *Path Planning of Inspection Robot Based on Ant Colony Optimization Algorithm*. in *Electrical and Control Engineering (ICECE), 2010 International Conference on*. 2010.

[90] Lumelsky, V.J. and T. Skewis, *Incorporating range sensing in the robot navigation function.* IEEE Transactions on systems, man and cybernetics, 1990. **20**: p. 1058-1068.

[91] Dadios, E.P. and O.A. Maravillas. *Fuzzy logic controller for micro-robot soccer game*. in *Industrial Electronics Society, 2001. IECON '01. The 27th Annual Conference of the IEEE*. 2001.

[92] Xinying, X., et al. *Immune Evolution Algorithm Based Dynamic Path Planning Approach for the Soccer Robot*. in *Granular Computing, 2007. GRC 2007. IEEE International Conference on*. 2007.

[93] Shih-Wen, C., et al. *Application of the GA-PSO with the fuzzy controller to the robot soccer*. in *SICE Annual Conference 2010, Proceedings of*. 2010.

[94] Song, D.-l. and Y.-l. Li. *An obstacle-avoidance path-planning in robot soccer based on Refined Genetic Algorithms*. in *Information Science and Engineering (ICISE), 2010 2nd International Conference on*. 2010.

[95] Haibin, D., Y. Yaxiang, and Z. Rui. *UCAV path planning based on Ant Colony Optimization and satisficing decision algorithm*. in *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*. 2008.

[96] Xiaoyong, Z., et al. *A Rescue Robot Path Planning Based on Ant Colony Optimization Algorithm*. in *Information Technology and Computer Science, 2009. ITCS 2009. International Conference on*. 2009.

[97] Juing-Shian, C., W. Kuo-Yang, and S. Ming-Yuan. *The optimization of the application of fuzzy ant colony algorithm in soccer robot*. in *Information and Automation, 2009. ICIA '09. International Conference on*. 2009.

[98] Kunli, Z., et al. *Improved Ant Colony Algorithm based on cellular Automata for obstacle avoidance in robot soccer*. in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*. 2010.

[99] Li, W., et al. *Obstacle-avoidance Path Planning for Soccer Robots Using Particle Swarm Optimization*. in *Robotics and Biomimetics, 2006. ROBIO '06. IEEE International Conference on*. 2006.

[100] Kim, S., J. Rusell, and K. Koo, *Construction robot path-planning for earthwork operations.* Journal of computing in civil engineering, 2003: p. 97-104.

[101] Dadios, E.P. and O.A. Maravillas. *Cooperative mobile robots with obstacle and collision avoidance using fuzzy logic*. in *Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on*. 2002.

[102] Guangshun, Y. and R. Qian. *A Control Algorithm for Soccer Robot's Moving Based on Fuzzy Control*. in *Computing, Communication, Control, and Management, 2008. CCCM '08. ISECS International Colloquium on*. 2008.

[103] Dijkstra, E.W., *A note on two problems in connexion with graphs*. Numerische Mathematik 1959. **1**: p. 269–271.

[104] Pearl, J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. 1984: Addison-Wesley.

[105] Chakrabarti, P., et al., *Heuristic search in restricted memory.* Artificial Intelligence, 1989. **47**: p. 197–221.

[106] Zhou, R. and E.A. Hansen, *Memory-Bounded A* Graph Search.* FLAIRS Conference, 2002: p. 203-209.

[107] Likhachev, M., G. Gordon, and S. Thrun, *ARA*: Anytime A* with provable bounds on sub-optimality.* In Advances in Neural Information Processing Systems., 2003.

[108] Bonet, B. and H. Geffner, *Planning as heuristic search.* . Artificial Intelligence, 2001. **129**(1-2): p. 5-33.

[109] Zhou, R. and E. Hansen, *Multiple sequence alignment using A*.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 2002.

[110] Edelkamp, S., *Planning with pattern databases.* In Proceedings of the European Conference on Planning, 2001.

[111] Rabin, S., *A* speed optimizations.* In DeLoura, M., ed., Game Programming Gems, 2000: p. 272-287.

[112] Chakrabarti, P., S. Ghosh, and S. DeSarkar, *Admissibility of AO* when heuristics overestimate.* Artificial Intelligence, 1998. **34**: p. 97–113.

[113] Stentz, A., *Optimal and efficient path planning for unknown and dynamic environments.* in Proc. IEEE Int. Conf. Robot Autom. , 1994: p. 3310-3317.

[114] Stentz, A., *The focused D* algorithm for real time replanning.* in Proc. IEEE Int. Conf. Artif. Intell., 1995: p. 1652-1659.

[115] Koenig, S. and M. Likhachev, *Fast replanning for navigation in unknown terrain.* IEEE Transactions Robot, 2005. **21**(3): p. 354-363.

[116] Ferguson, D. and A. Stentz, *The Delayed D* Algorithm for Efficient Path Replanning.* Proceedings of the IEEE International Conference on Robotics and Automation, 2005: p. 2045 - 2050.

[117] Likhachev, M., et al., *Anytime Dynamic A*: An Anytime, Replanning Algorithm.* In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2005.

[118] Koenig, S. and M. Likhachev, *Improved fast replanning for robot navigation in unknown terrain.* In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). 2002.

[119] Ferguson, D., M. Likhachev, and A. Stentz, *A guide to heuristicbased path planning.* in Proc. of the Workshop on Planning under Uncertainty for Autonomous Systems, Int. Conf. on Automated Planning and Scheduling (ICAPS'05), 2005.

[120] Nilsson, N., *Principles of Artificial Intelligence.* . 1990: Tioga Publishing Company,.

[121] Aho, A.V., J. Ulman, and J. Hopcroft, *Data Structures and Algorithms*: Addison-Wesley.

[122] Available from: http://www.robocup.org/.

[123] Lau, N., et al. *Multi-Robot Team Coordination Through Roles, Positioning and Coordinated Procedures*. in *Proc.IEEE/RSJ Int. Conf. on Intelligent Robots and Systems – IROS 2009*. 2009. St. Louis, USA.

[124] Lau, N., et al., *Roles, Positionings and Set Plays to Coordinate a MSL Robot Team* in *14th Port. Conf. on Artificial Intelligence, EPIA'2009*. 2009, Springer: Aveiro. p. 323-337.

[125] Alves, P., N. Lau, and L.P. Reis. *Map Visiting In RoboCup Rescue Simulations*. in *Proceedings of IROBOT 2008 - 3rd International Workshop on Intelligent Robotics*. 2008. Lisbon University Institute – ISCTE.

[126] Mota, L. and L.P. Reis, *A Common Framework for Cooperative Robotics: an Open, Fault Tolerant Architecture for Multileague RoboCup Teams*, in *International Conference on Simulation Modeling and Programming for Autonomous Robots (SIMPAR 2008)*. 2008: Venice, Italy. p. 171-182.

[127] Lau, N. and L.P. Reis, *FC Portugal - High-level Coordination Methodologies in Soccer Robotics,*, in *Robotic Soccer*, I.E.a.P. Pedro Lima, Editor. 2007: Vienna, Austria. p. 167-192.

[128] Reis, L.P. and N. Lau, *COACH UNILANG – A Standard Language for Coaching a (Robo) Soccer Team*, in *RoboCup2001 Symposium: Robot Soccer World Cup V,*, S.C.a.S.T. Andreas Birk, Editor. 2002: Berlin. p. 183-192.

[129] Reis, L.P., N. Lau, and E.C. Oliveira, *Situation Based Strategic Positioning for Coordinating a Team of Homogeneous Agents*, in *Balancing Reactivity and Social Deliberation in Multi-Agent System – From RoboCup to Real-World Applications*, J.W.a.E.P. M.Hannebauer, Editor. 2001. p. 175-197.

[130] Conceição, A.S., et al., *Architecture of Cooperation for Multi-Robot Systems*, in *1st IFAC Workshop on Multivehicle Systems (MVS'06)*. 2006: Bahia Conv. Center, Salvador, Brazil.

[131] Sousa, M., R. Braga, and L.P. Reis, *Intellwheels MMI: A Flexible Interface for an Intelligent Wheelchair*, in *RoboCup Symp2009*, M.L. Jacky Baltes, T.Naruse, S.Shiry, Editor. 2009, Springer: Graz, Austria.

[132] Lima, J.L., et al., *Humanoid Realistic Simulator – The Servomotor Joint Modeling*, in *ICINCO 2009–6th Int. Conf. on Informatics in Control, Automation and Robotics*. 2009: Milan, Italy. p. 396-400.

[133] Almeida, R., L.P. Reis, and A.M. Jorge, *Analysis and Forecast of Team Formation in the Simulated Robotic Soccer Domain*, in *14th Port. Conf. on Artificial Intelligence, EPIA'2009*. 2009, Springer: Aveiro. p. 239-250.

[134] Oliveira, H.P., et al., *Precise Modeling of a Four Wheeled Omni-directional Robot*, in *8th Conference on Autonomous Robot Systems and Competitions - ROBOTICA2008*. 2008: Aveiro. p. 57-62.

[135] Oliveira, H.P., et al., *Dynamical Models for Omni-direccional Robots with 3 and 4 Wheels*, in *ICINCO 2008 -International Conference on Informatics in Control, Automation and Robotics*. 2008: Funchal, Madeira, Portugal. p. 189-196.

[136] Mota, L. and L.P. Reis, *An Elementary Communication Framework for Open Co-operative RoboCup Soccer Teams*, in *4th International Conference on Informatics in Control, Automation and Robotics - ICINCO 2007 – 3rd International Workshop on Multi-Agent Robotic Systems*. 2007: Angers, France. p. 97-101.

[137] Reis, L.P., et al., *FC Portugal: Development and Evaluation of a New RoboCup Rescue Team*, in *1st IFAC Workshop on Multivehicle Systems (MVS'06)*. 2006: Salvador, Brazil.

[138] Costa, P., et al., *5dpo Team Description*, in *Robocup-99: Robot Soccer World Cup III*, V.e. al., Editor. 2000. p. 663-666.

[139] Costa, P., et al., *5dpo-2000 Team Description*, in *Robocup-99: Robot Soccer World Cup III*, V.e. al., Editor. 2000. p. 754-757.

[140] Moreira, A.P., P. Costa, and P. Costa. *Real-time path planning using a modified A* algorithm*. in *Proceedings of ROBOTICA 2009 - 9th Conference on Mobile Robots and Competitions*. 2008.

[141] Picado, H., et al., *Automatic Generation of Biped Walk Behavior Using Genetic Algorithms*, in *10th International Work-Conference on Artificial Neural Networks*. 2009, Springer: Salamanca, Spain.

[142] Lima, J.L., et al., *Humanoid Robot Simulator: A Realistic Dynamics Approach*, in *8th Portuguese Conference on Automatic Control, Controlo 2008*. 2008: Vila Real, Portugal. p. 485-490.

[143] Lima, J.L., et al., *Humanoid robot simulation with a joint trajectory optimized controller*, in *ETFA 2008. IEEE International Conference on Emerging Technologies and Factory Automation*. 2008: Hamburg, Germany. p. 986 – 993.

[144] Lau, N., L.P. Reis, and J. Certo, *FC Portugal 2001 Team Description: Configurable Strategy and Flexible Teamwork*, in *RoboCup-2001: Robot Soccer World Cup V*, S.C.a.S.T. A.Birk, Editor. 2002. p. 515-518.

[145] Reis, L.P. and N. Lau, *FC Portugal Team Description: RoboCup 2000 Simulation League Champion*, in *RoboCup-2000: Robot Soccer World Cup IV*, T.B. P.Stone, G.Kraetzschmar, Editor. 2001: Berlin. p. 29-40.

[146] Ramirez, M.I., P.M. Abreu, and L.P. Reis. *Using a Datawharehouse to Extract Knowledge from ROBOCUP Teams*. in *Proceedings of ICEIS 2008 - 10th Int. Conf. on Enterprise Information Systems*. 2008.

[147] Certo, J., N. Lau, and L.P. Reis, *A Generic Multi-Robot Coordination Strategic Layer*, in *RoboComm 2007 - First International Conference on Robot Communication and Coordination*. 2007: Athens, Greece.

[148] Certo, J., et al., *FCPx: A Tool for Evaluating Teams' Performance in RoboCup Rescue Simulation League*, A.a.R.-G. Gelbukh, C.Special Issue: Advances in Artificial Intelligence, Research in Computing Science, Editor. 2007. p. 137-148.

[149] Lau, N., L.P. Reis, and J. Certo. *Multi-Level, Functional, Spatial and Temporal Agent's Reasoning Debugging*. in *Proc. 13th Port.Conf. on AI, EPIA 2007, New Trends in Artificial Intelligence*. 2007. Guimarães, Portugal.

[150] Lau, N., L.P. Reis, and J. Certo, *Understanding Dynamic Agent's Reasoning*, in *Progress in Artificial Intelligence, 13th Port. Conf. on AI, EPIA 2007*. 2007: Guimarães, Portugal. p. 542-551.

[151] Silva, J., et al., *Sensor and Information Fusion applied to a Robotic Soccer Team*, in *RoboCup Symposium*, M.L. J.Baltes, T.Naruse, S.Shiry, editors, Editor. 2009, Springer: Graz, Austria.

[152] Silva, J., et al., *Obstacle Detection, Identification and Sharing on a Robotic Soccer Team*, in *14th Port. Conf. on Artificial Intelligence, EPIA'2009*. 2009, Springer: Aveiro. p. 350-360.

[153] Silva, J., et al., *Ball Sensor Fusion and Ball Interception Behaviours for a Robotic Soccer Team*, in *IROBOT'08 - 3rd International Workshop on Intelligent Robotics*. 2008: Lisboa, Portugal. p. 25-36.

[154] Moreira, A.P.G.M., A. Sousa, and P. Costa, *Vision Based Real-Time Localization of multiple Mobile Robots*, in *FSR2001– International Conference on Field and Service Robotics*. 2001: Ulm, Finland. p. 103-106.

[155] Neves, A.J.R., et al., *Autonomous Configuration of Parameters in Robotic Digital Cameras*, in *Proc.4th Iberian Conf.on Pattern Recog. Image Analysis, ibPRIA 2009*. 2009: Póvoa do Varzim, Portugal

[156] Martins, D.A., A.J.R. Neves, and A.J. Pinho, *Real-Time Generic Ball Detection in RoboCup Domain*, in *IROBOT'08 - 3rd International Workshop on Intelligent Robotics*. 2008: Lisboa, Portugal. p. 37-48.

[157] Cunha, B., et al., *Obtaining the Inverse Distance Map from a Non-SVP Hyperbolic Catadioptric Robotic Vision System*, in *RoboCup Symposium, In: RoboCup 2007: Robot Soccer World Cup XI*. 2007. p. 417-424.

[158] Moreira, P., L.P. Reis, and A.A. Sousa, *Best Multiple-View Selection for the Visualization of Urban Rescue Simulations.* IJSIMM - International Journal of Simulation Modelling, 2007. **5**(4): p. 167-173.

[159] Reis, L.P. *Robust Vision Algorithms for Quadruped Soccer Robots*. in *CompImage 2006 – Comp. Modelling of Objects Represented in Images: Fundamentals Methods and Applications*. 2006. Coimbra, Portugal: Taylor & Francis.

[160] Moreira, P.M., L.P. Reis, and A.A. Sousa. *Visualization Optimization: Application to the RoboCup Rescue Domain*. in *SIACG 2006 – Ibero American Symp. in Computer Graphics*. 2006. Santiago de Compostela, Spain.

[161] Costa, P., et al., *Tracking and Identifying in Real Time the Robots of a F-180 Team*, in *Robocup-99: Robot Soccer World Cup III*, V.e. al., Editor. 2000. p. 286-291.

[162] Martinoli, A., *Swarm Intelligence in Autonomous Collective Robotics: From Tools to the Analysis and Synthesis of Distributed Collective Strategies*, in *DI-EPFL*. 1999: Lausanne.

[163] Costa, P.; Available from: http://paginas.fe.up.pt/~paco/index.php?n=SimTwo.SimTwo.

# Anexos

# Anexo A

Ficheiro XML do simulador SimTwo, que define o robô:

```
<?xml version="1.0" ?>
<robot>
  <kind value='omni3'/>
  <solids>
    <cylinder>
      <ID value='1'/>
      <mass value='1.9'/>
      <size x='0.075' y='0' z='0.135'/>
      <pos x='0' y='0' z='0.08'/>
      <rot_deg x='0' y='0' z='0'/>
      <color_rgb r='128' g='0' b='255'/>
      <!--<texture name='MatFeup' scale='6'/>-->
      <texture name='MatBallTriangle' scale='6'/>
    </cylinder>
  </solids>

  <wheels>
    <default>
      <omni/>
      <tyre mass='0.2' radius='0.0325' width='0.015' centerdist='0.06'/>
      <surface mu='1' mu2='0.001'/>
      <axis angle='0'/>
      <motor ri='1.73' ki='1.3e-2' vmax='12' imax='2' active='1'/>
      <gear ratio='10'/>
      <friction bv='2e-3' fc='1e-2'/>
      <encoder ppr='256' mean='0' stdev='0'/>
      <controller mode='pidspeed' kp='0.16' ki='0' kd='0.01' kf='0.05' active='1' period='10'/>
      <color_rgb r='128' g='0' b='128'/>
    </default>
    <wheel>
      <axis angle='-60'/>
    </wheel>
    <wheel>
      <axis angle='60'/>
    </wheel>
    <wheel>
      <axis angle='180'/>
    </wheel>
  </wheels>

 <shells>
    <cuboid>
      <size x='0.005' y='0.08' z='0.05'/>
      <pos x='0.085' y='0' z='0'/>
```

```
      <rot_deg x='0' y='0' z='0'/>
      <color_rgb r='128' g='128' b='128'/>
    </cuboid>
   <cuboid>
      <size x='0.005' y='0.08' z='0.05'/>
      <pos x='-0.085' y='0' z='0'/>
      <rot_deg x='0' y='0' z='0'/>
      <color_rgb r='128' g='128' b='128'/>
    </cuboid>
 <cuboid>
      <size x='0.08' y='0.005' z='0.05'/>
      <pos x='0' y='0.085' z='0'/>
      <rot_deg x='0' y='0' z='0'/>
      <color_rgb r='128' g='128' b='128'/>
    </cuboid>
<cuboid>
      <size x='0.08' y='0.005' z='0.05'/>
      <pos x='0' y='-0.085' z='0'/>
      <rot_deg x='0' y='0' z='0'/>
      <color_rgb r='128' g='128' b='128'/>
    </cuboid>
<cuboid>
<size x='0.065' y='0.005' z='0.05'/>
      <pos x='0.063' y='0.063' z='0'/>
      <rot_deg x='0' y='0' z='135'/>
      <color_rgb r='128' g='128' b='128'/>
    </cuboid>
<cuboid>
<size x='0.065' y='0.005' z='0.05'/>
      <pos x='0.063' y='-0.063' z='0'/>
      <rot_deg x='0' y='0' z='45'/>
      <color_rgb r='128' g='128' b='128'/>
    </cuboid>
<cuboid>
<size x='0.065' y='0.005' z='0.05'/>
      <pos x='-0.063' y='0.063' z='0'/>
      <rot_deg x='0' y='0' z='45'/>
      <color_rgb r='128' g='128' b='128'/>
    </cuboid>
<cuboid>
<size x='0.065' y='0.005' z='0.05'/>
      <pos x='-0.063' y='-0.063' z='0'/>
      <rot_deg x='0' y='0' z='135'/>
      <color_rgb r='128' g='128' b='128'/>
    </cuboid>
  </shells>
</robot>
```

# Anexo B

Descrição das caracteristicas do computador utilizado na elaboração do trabalho:

**CPU** : Intel(R) Core(TM)2 CPU  T7200  @ 2.00GHz

**Memoria** (RAM): 2048 MB

**Placa de som**: Realtek HD Audio output

**Placa gráfica**: ATI Mobility Radeon X1600 Screen Resolution: 1280 X 800 - 32 bit

**Adaptadores de rede**: Intel(R) PRO/Wireless 3945ABG Network Connection - Packet Scheduler Miniport | Bluetooth Personal Area Network - Packet Scheduler Miniport | Realtek RTL8169/8110 Family Gigabit Ethernet NIC - Packet Scheduler Miniport

**CD / DVD Drives**: D: TSSTcorpCD/DVDW TS-L632D

**Disco Rigido**: C:  97.7GB | E:  195.3GB | F:  172.8GB

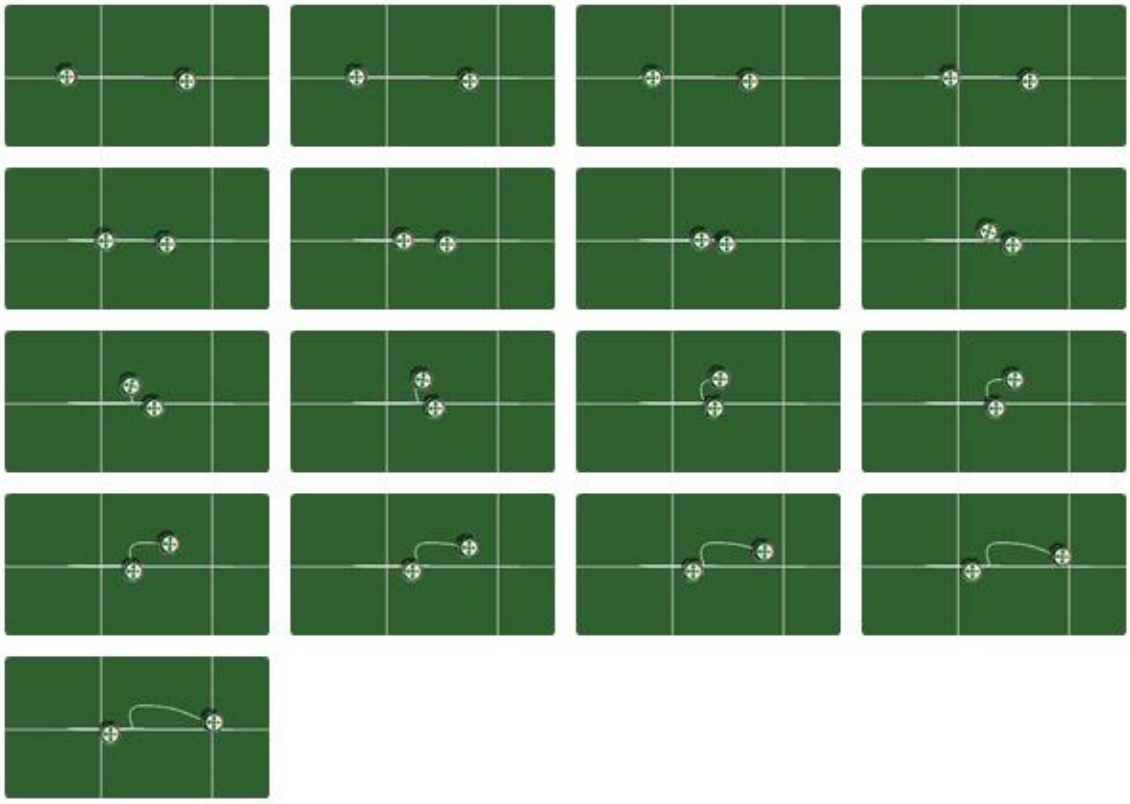
**USB**: 5 portas.

**Firewire (1394):** 1 porta.

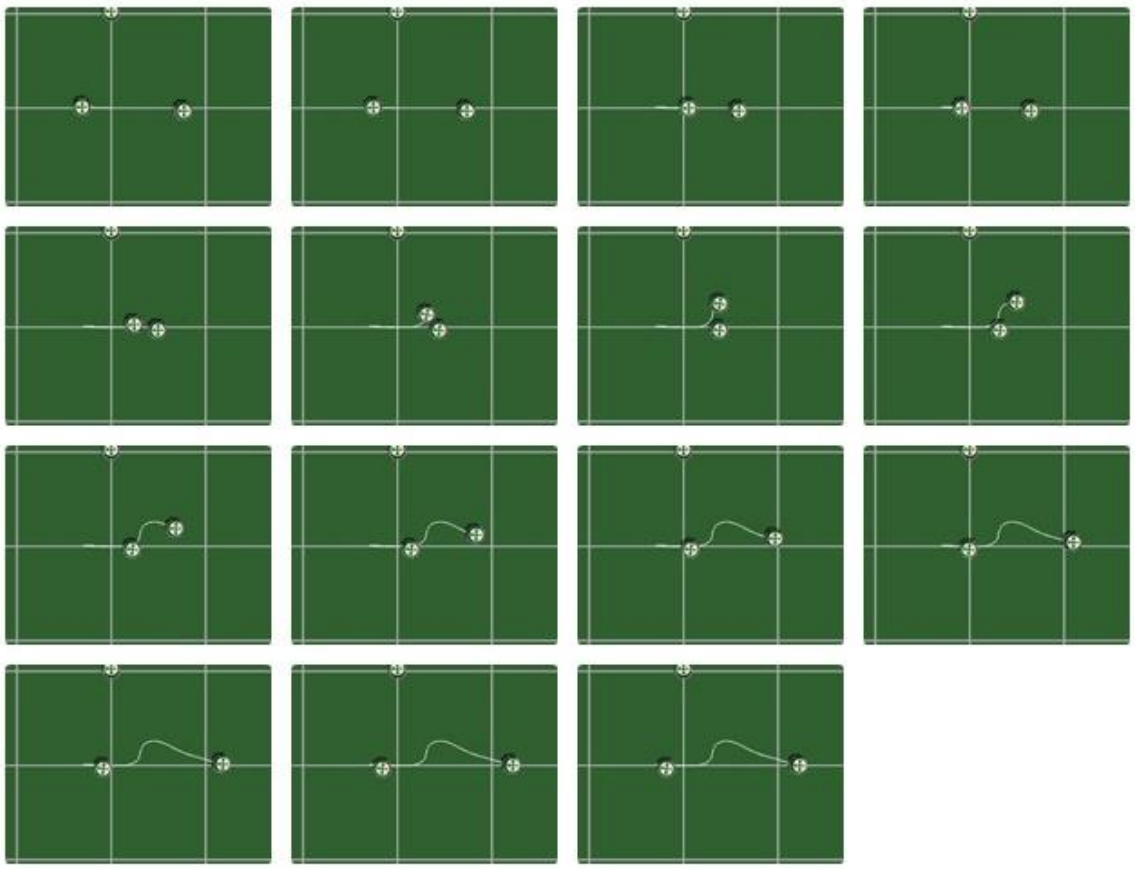
**Windows**: Windows XP Professional, Version 5.1.2600, Service Pack 3, 32 Bit.

# Anexo C

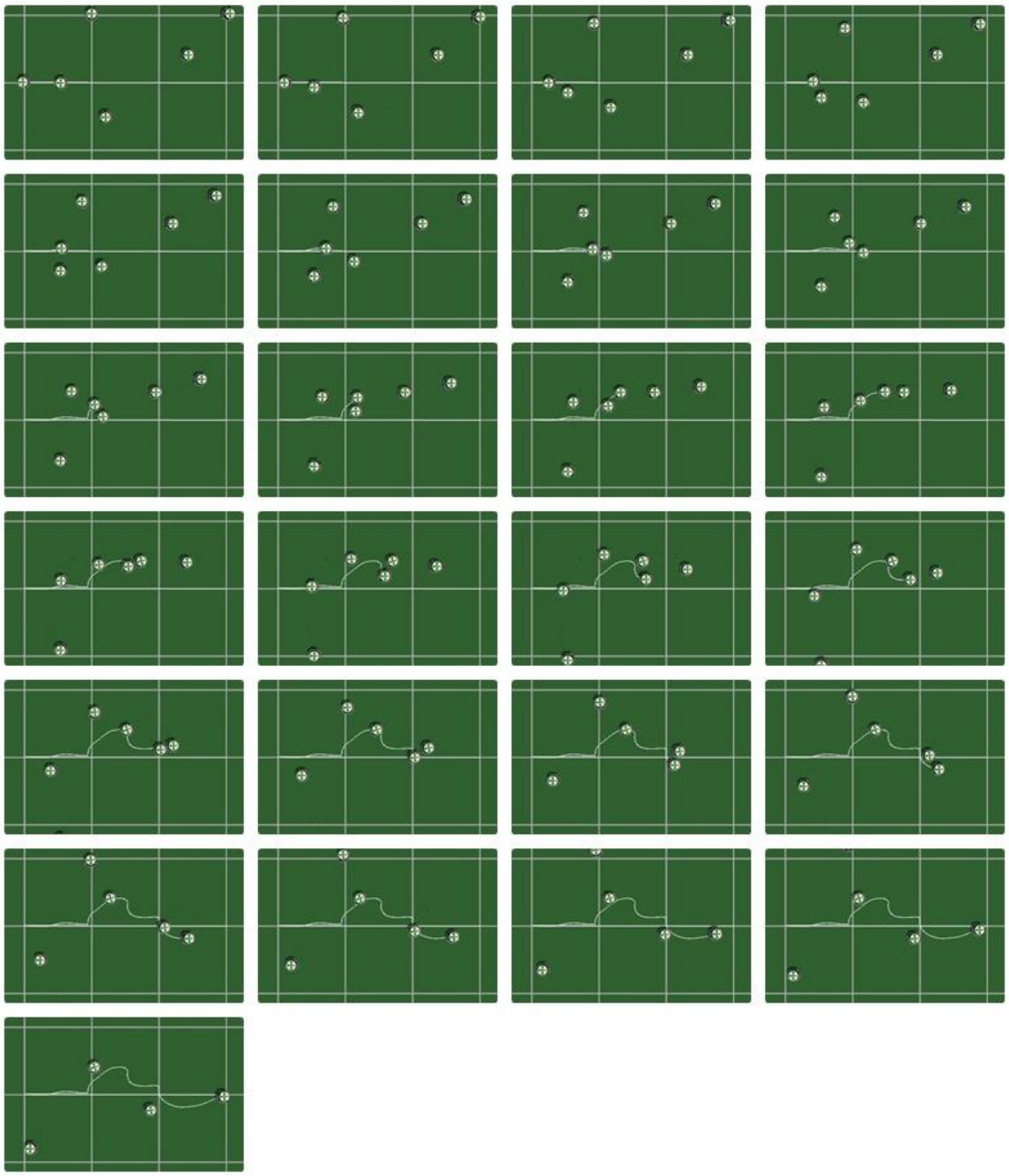
Ponto de Colisão – 1º Situação -Fotogramas da simulação Normal



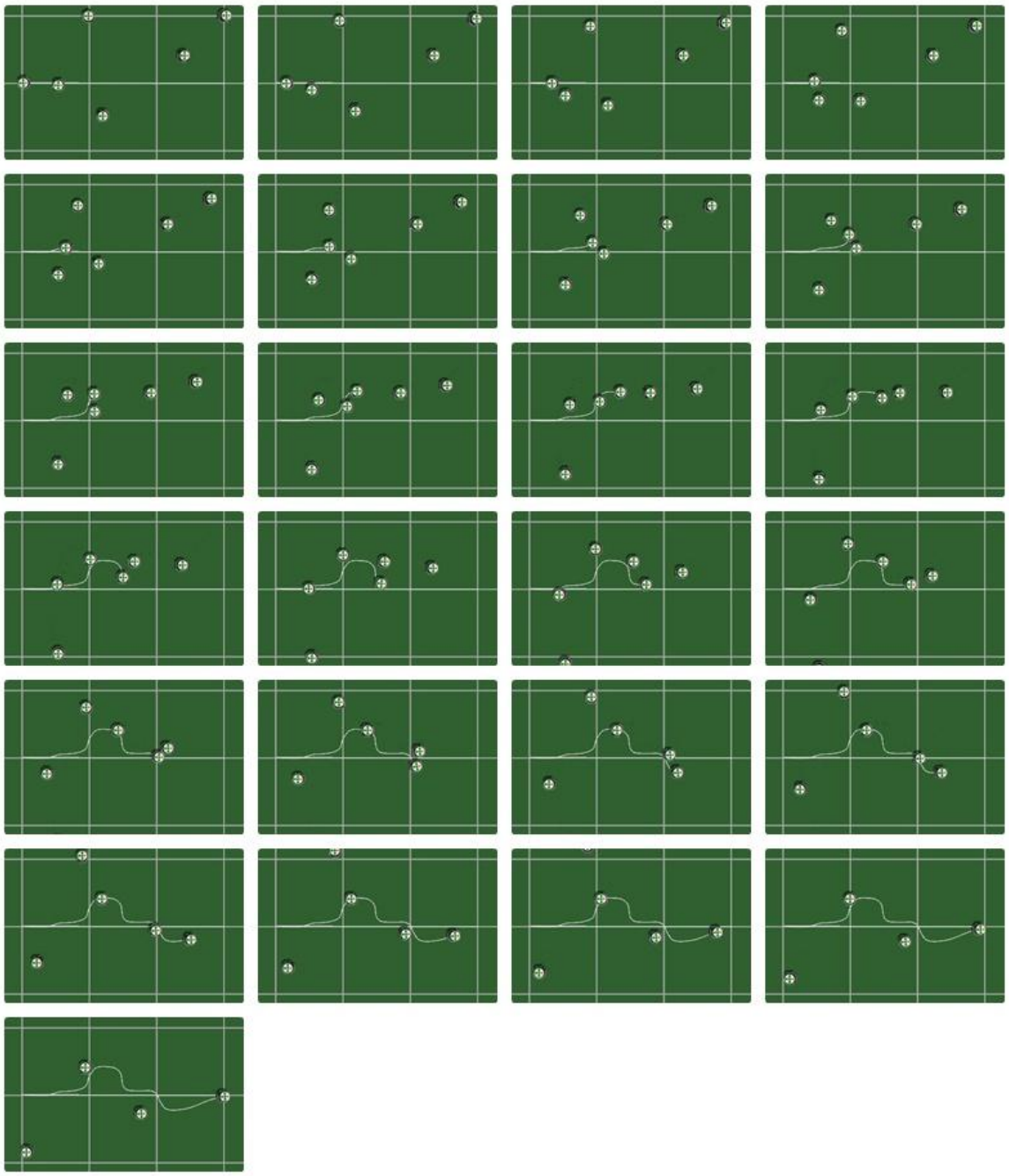
Ponto de Colisão – 1º Situação -Fotogramas da simulação Modificado



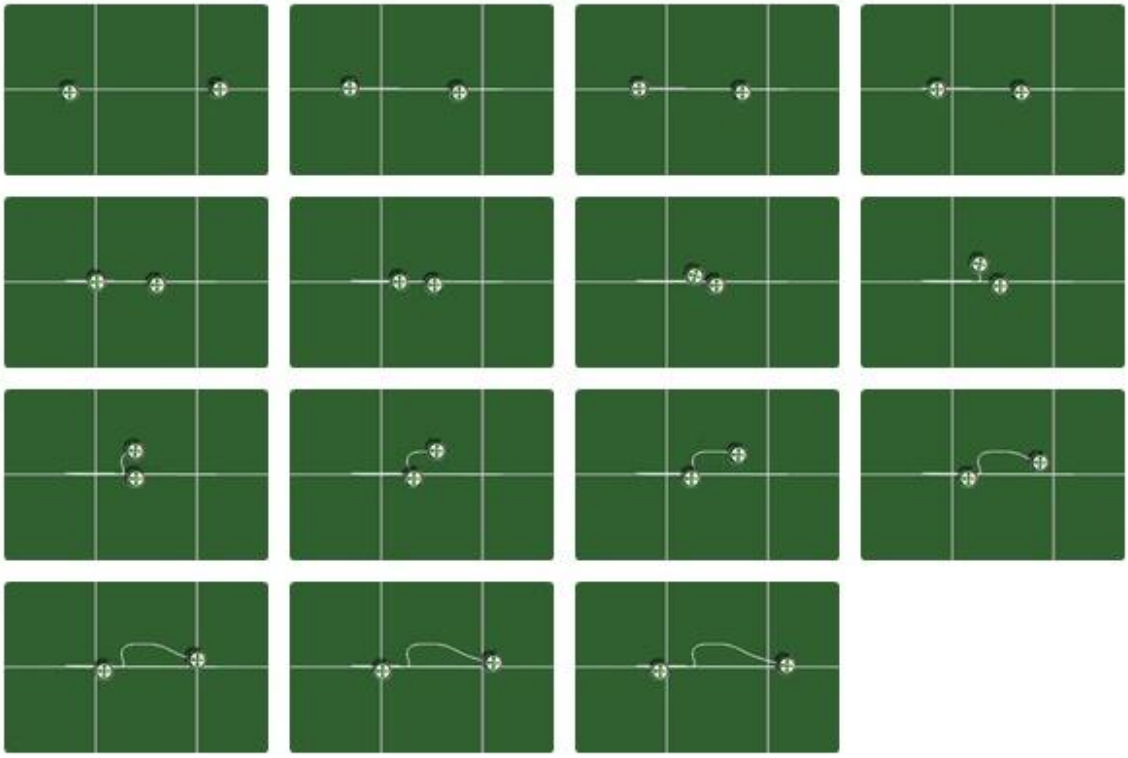
Ponto de Colisão – 2º Situação -Fotogramas da simulação Normal



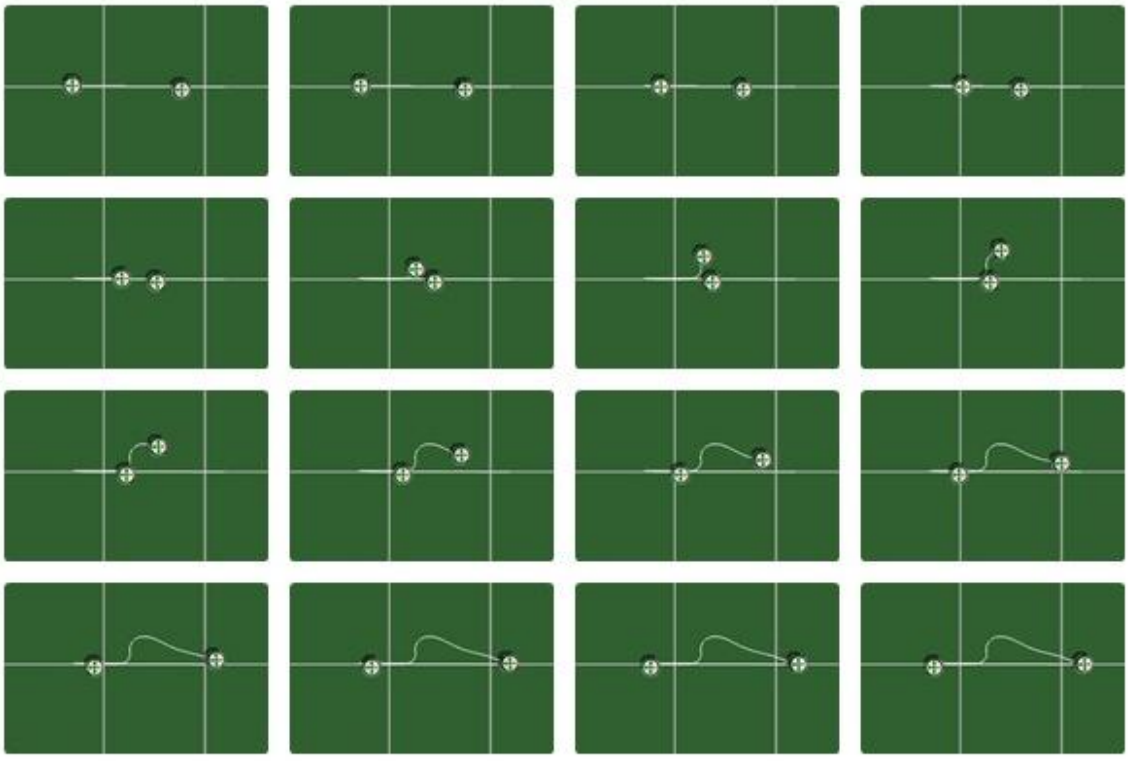
Ponto de Colisão – 2º Situação -Fotogramas da simulação Modificado



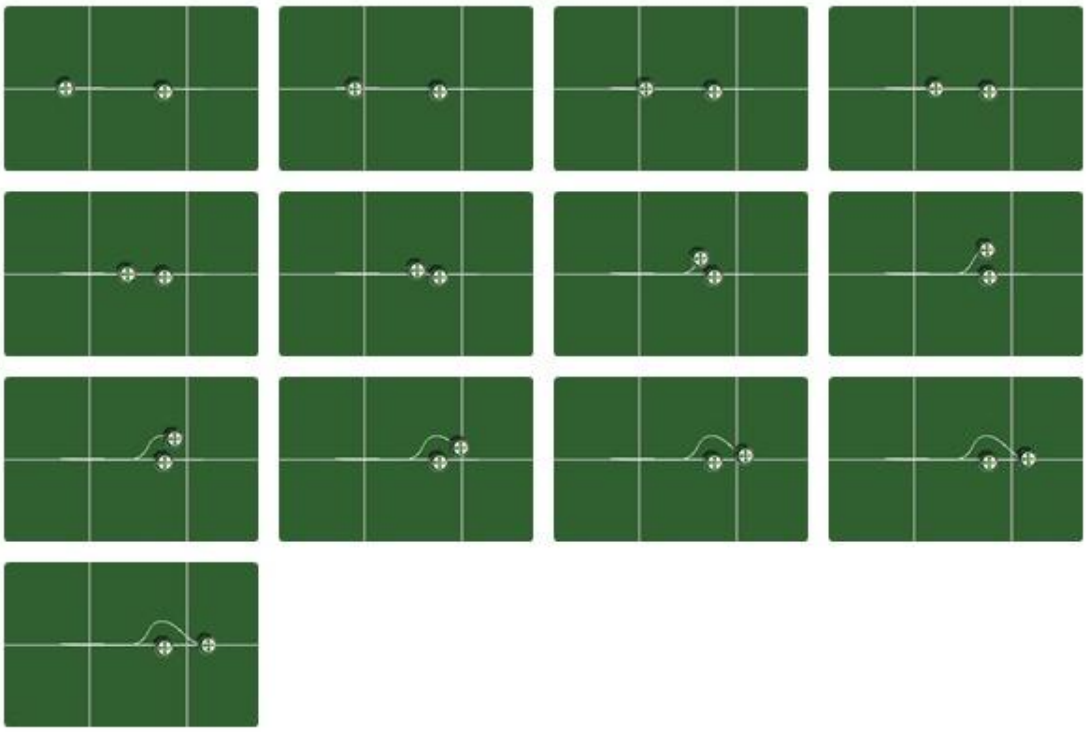
Zona de folga – 1º Situação -Fotogramas da simulação Normal



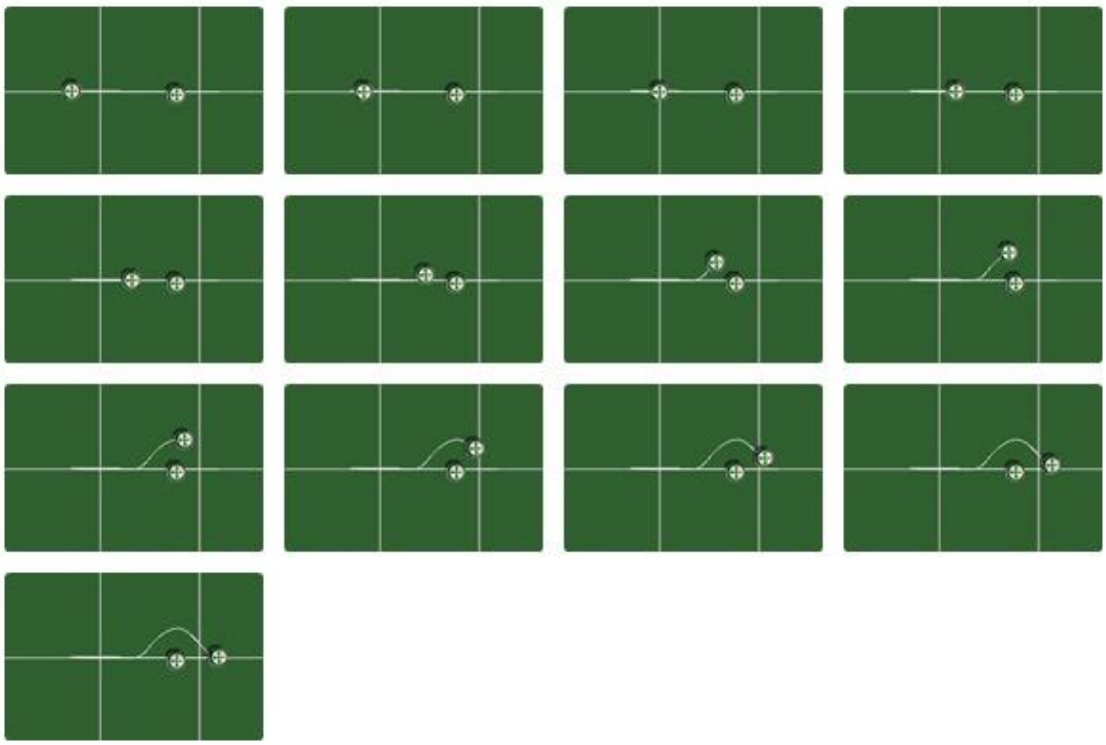
Zona de folga – 1º Situação -Fotogramas da simulação Modificado



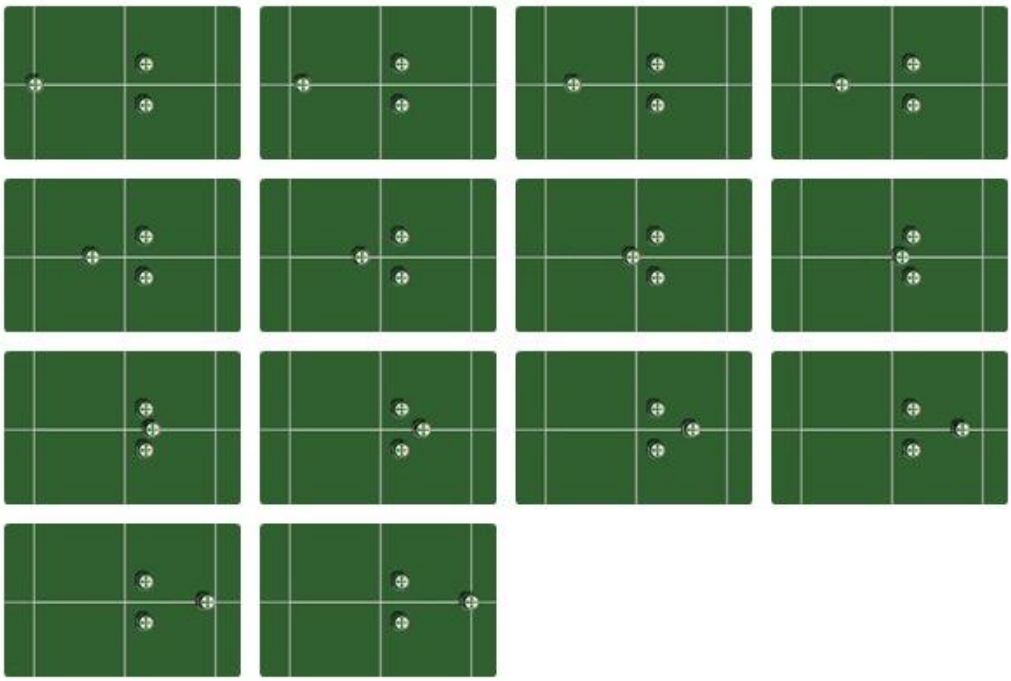
Zona de folga – 2º Situação -Fotogramas da simulação Normal



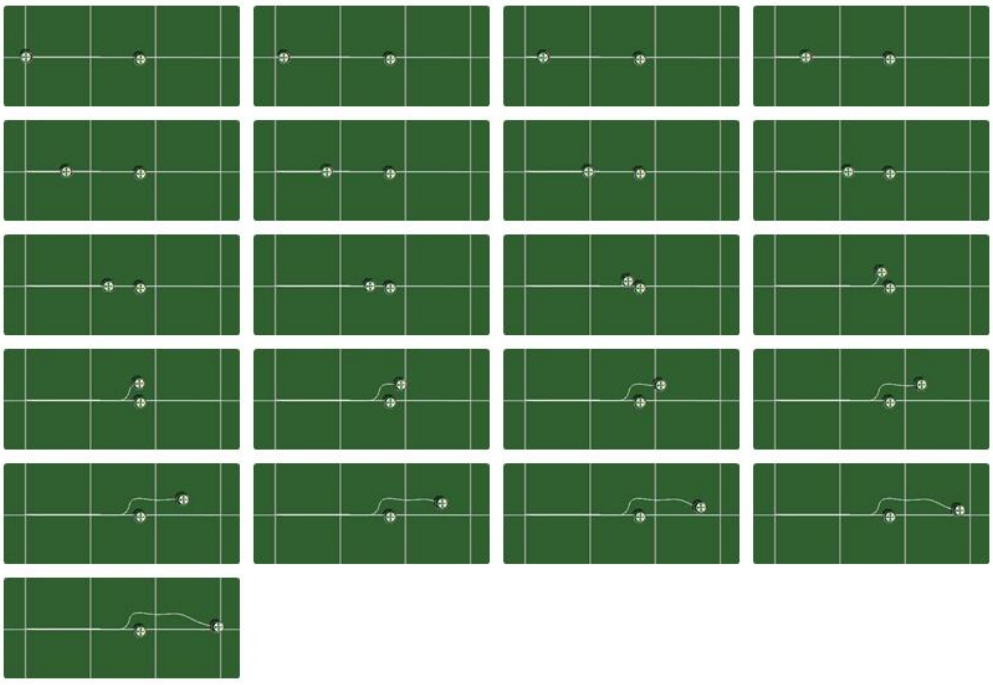
Zona de folga – 2º Situação -Fotogramas da simulação Modificado



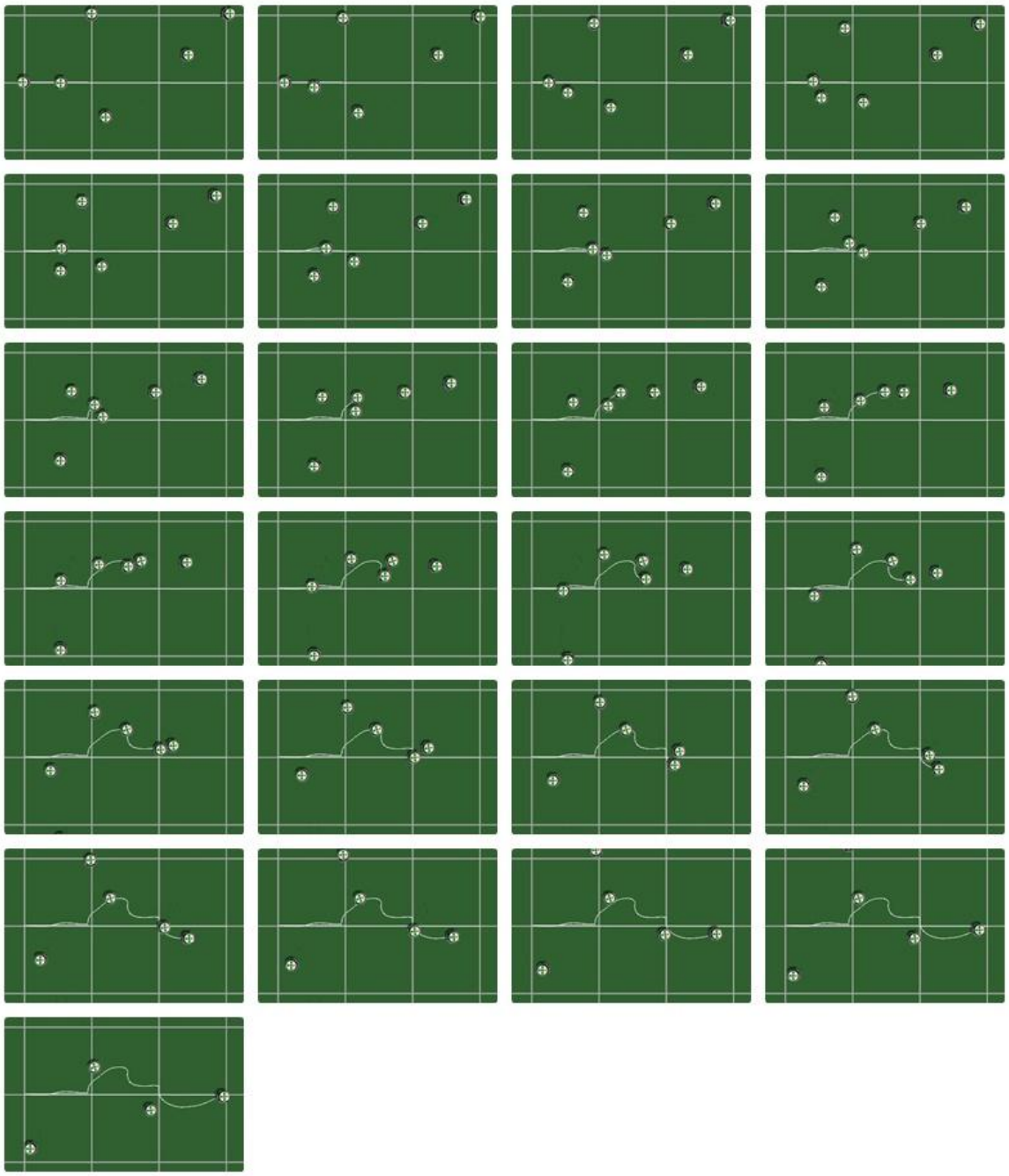
Zona de folga – 3º Situação -Fotogramas da simulação Normal e Modificado



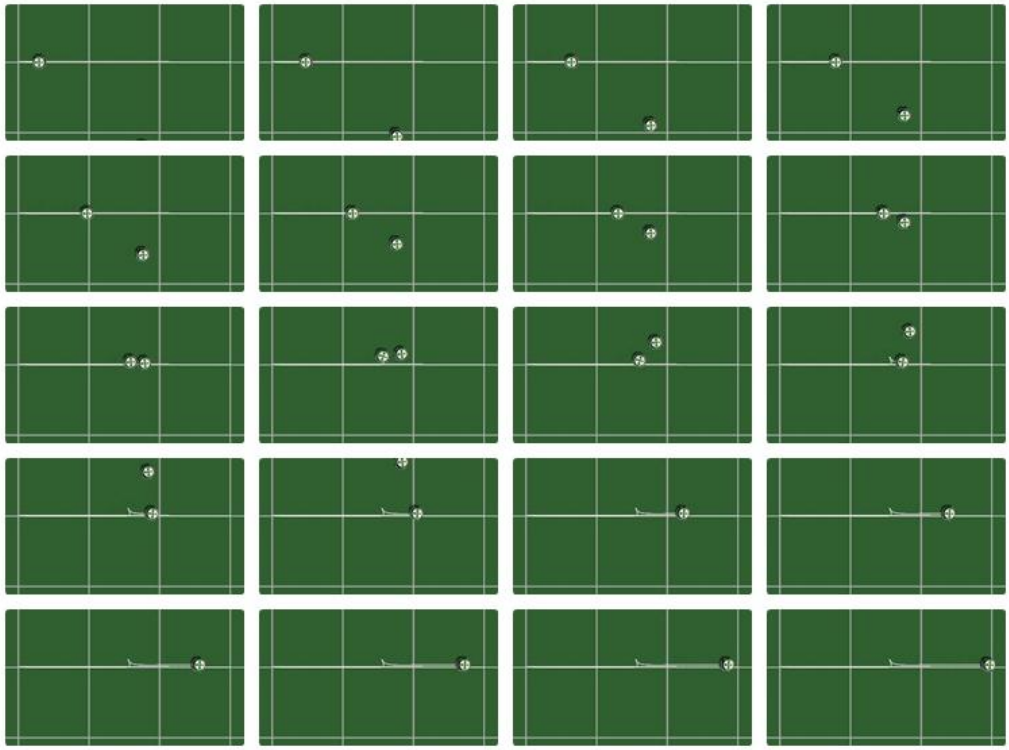
Distancia -– 1º Situação -Fotogramas da simulação Normal e Modificado



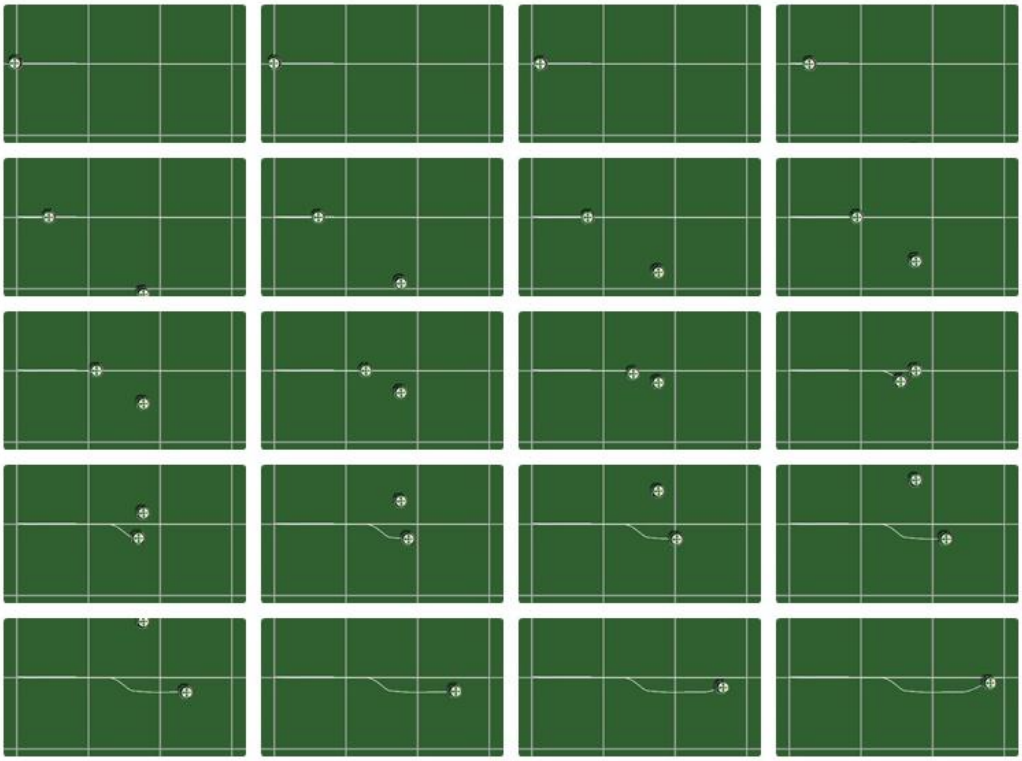
Distancia – 2ª Situação -Fotogramas da simulação Normal e Modificado



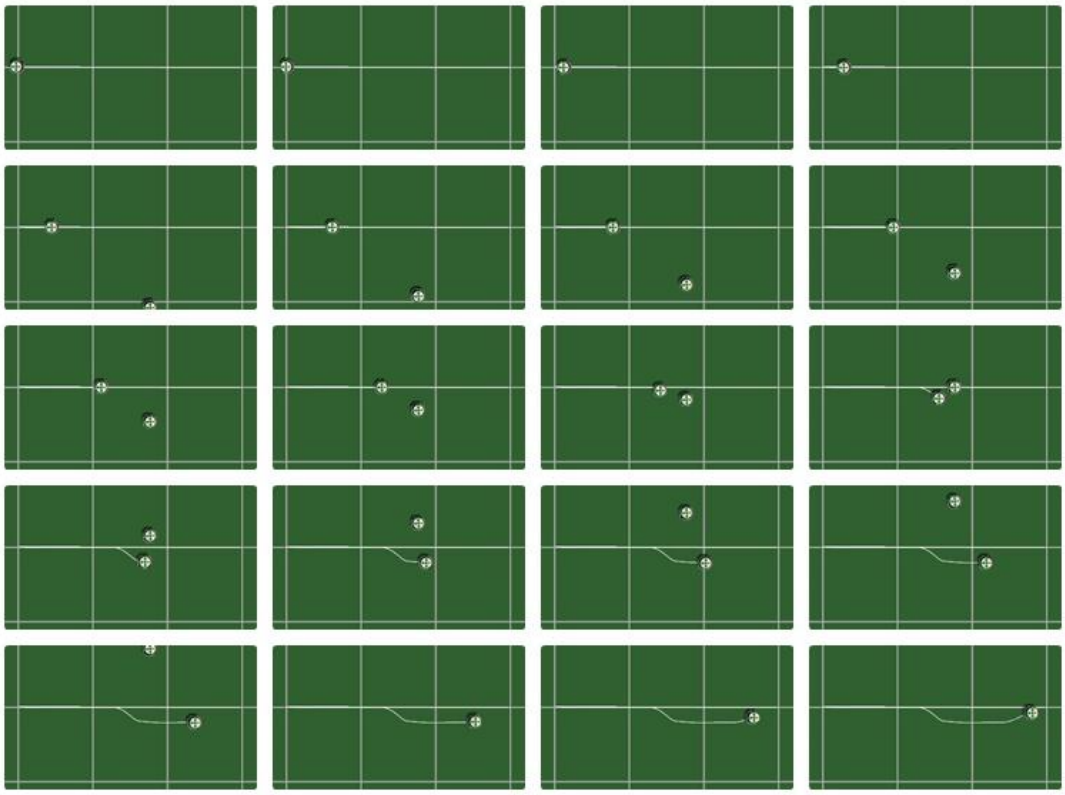
Rasto – 1ª Situação -Fotogramas da simulação Normal



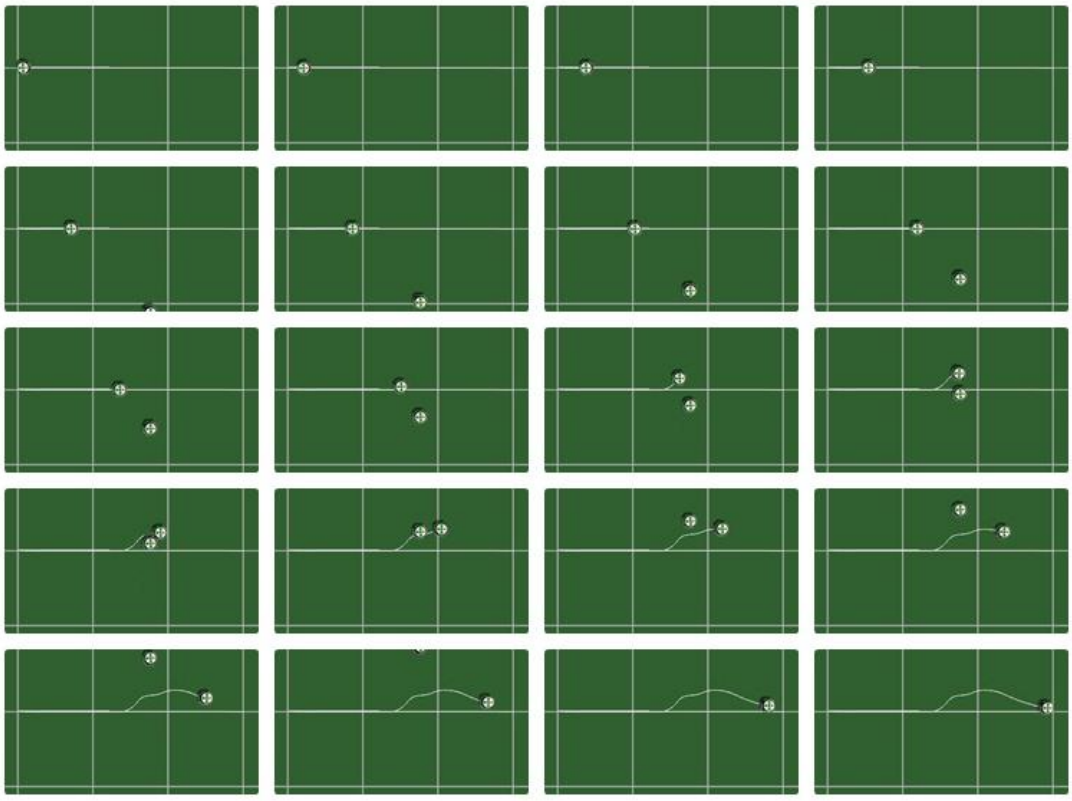
Rasto – 1ª Situação -Fotogramas da simulação Modificado



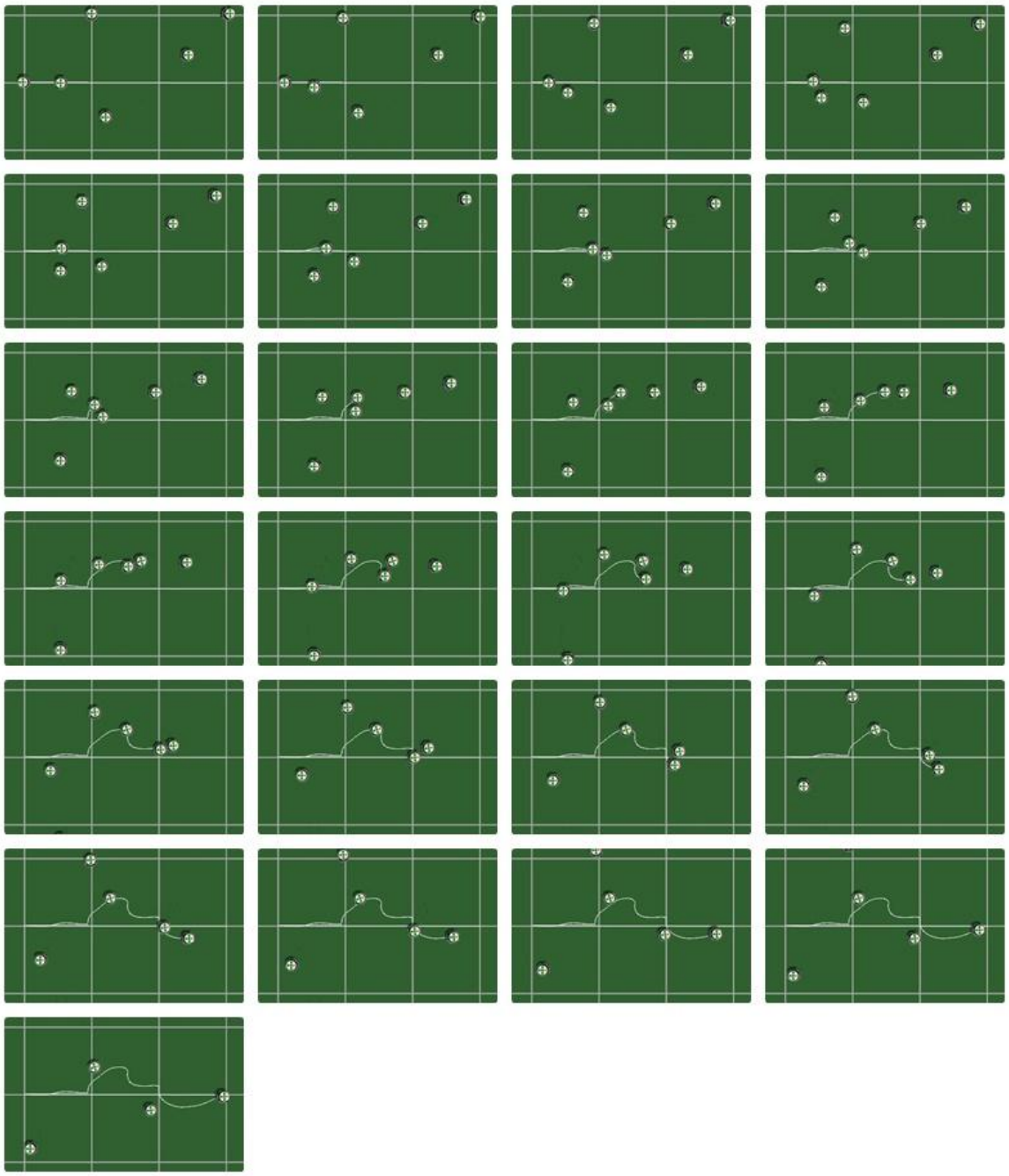
Rasto – 2ª Situação -Fotogramas da simulação Normal



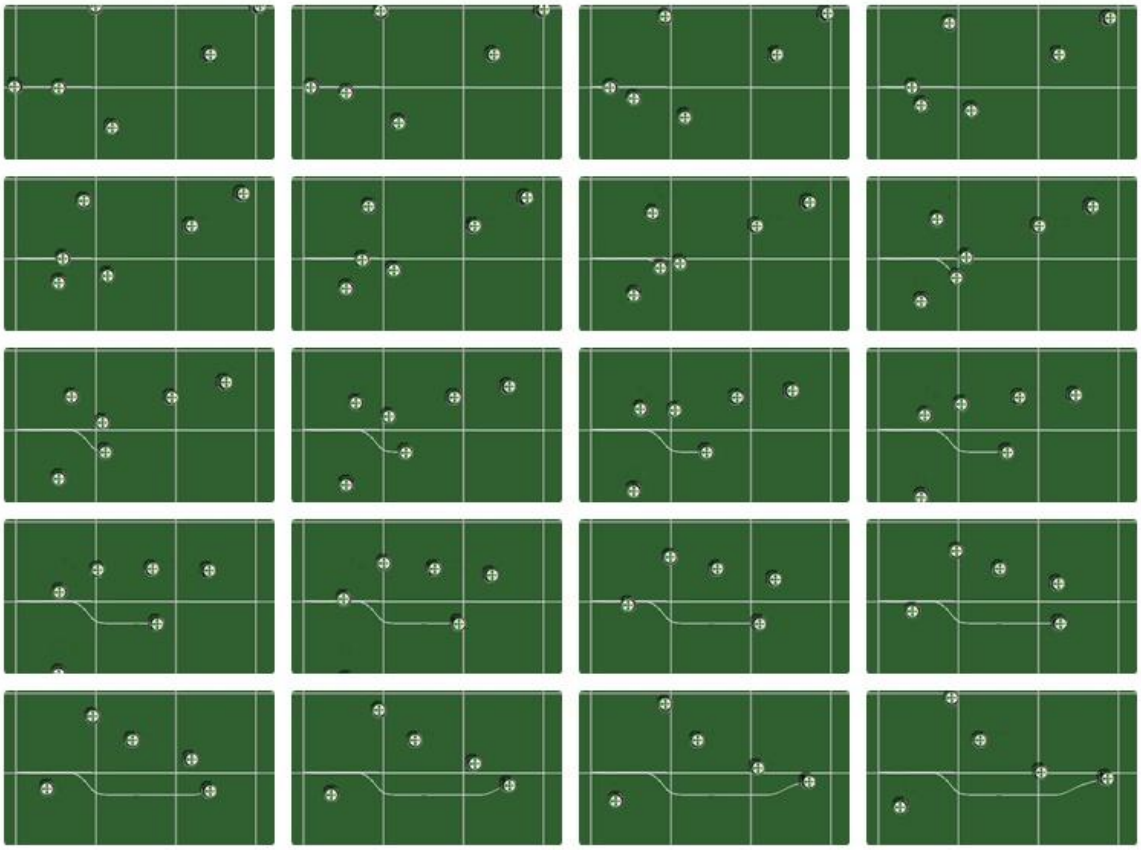
Rasto – 2ª Situação -Fotogramas da simulação Modificado



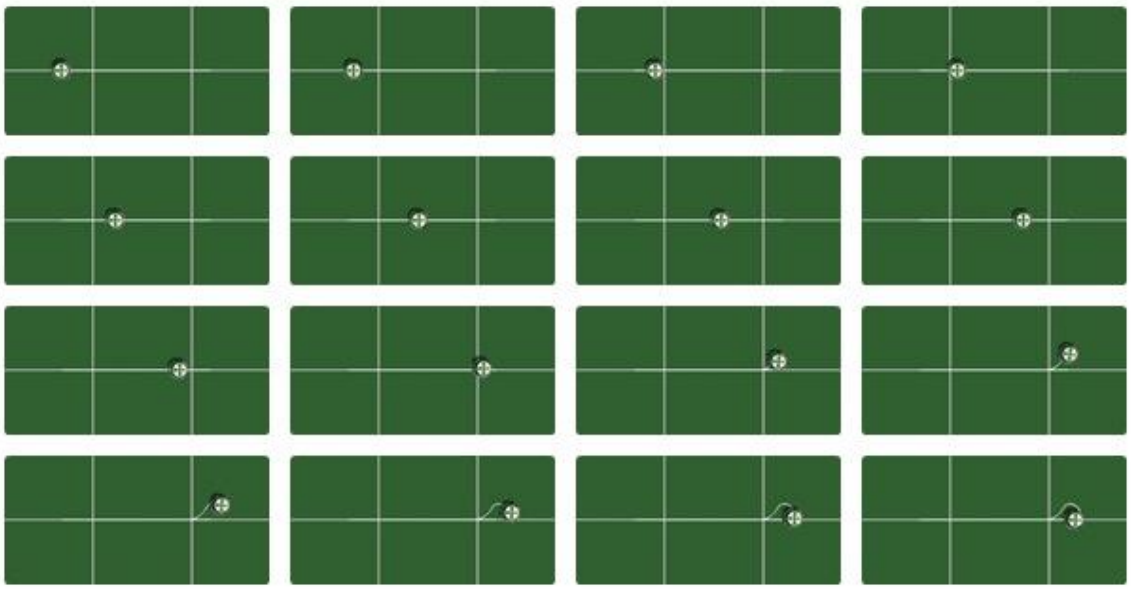
Rasto – 3ª Situação -Fotogramas da simulação Normal



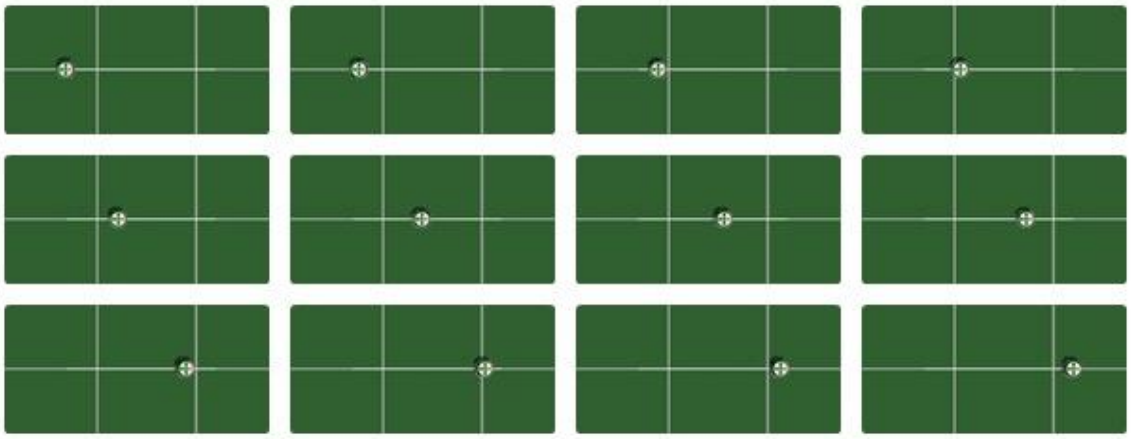
Rasto – 3ª Situação -Fotogramas da simulação Modificado



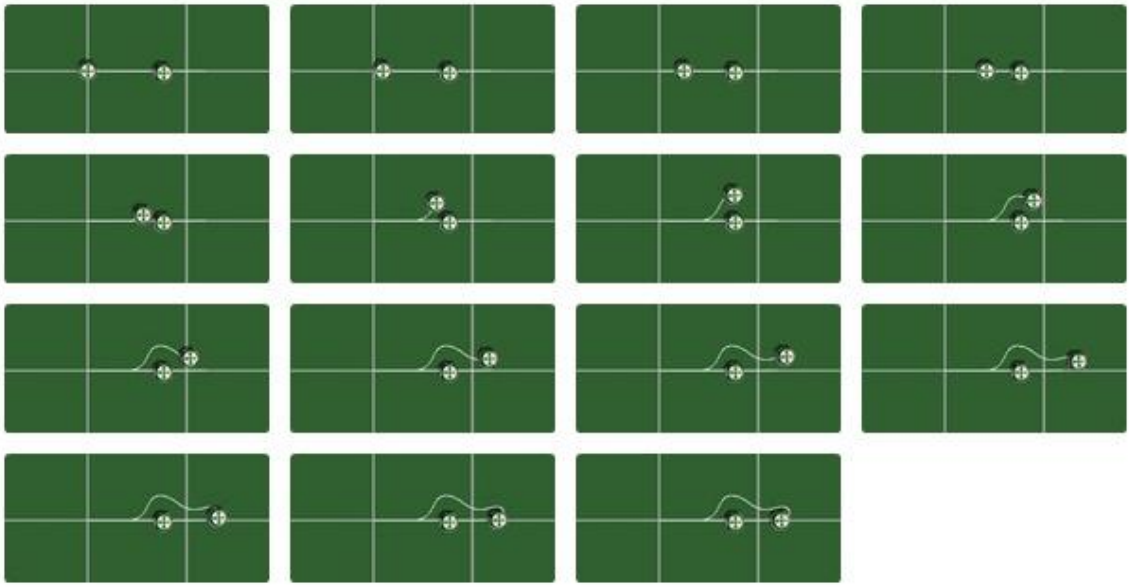
Direcção 1ª Situação -Fotogramas da simulação Direcção obrigatória



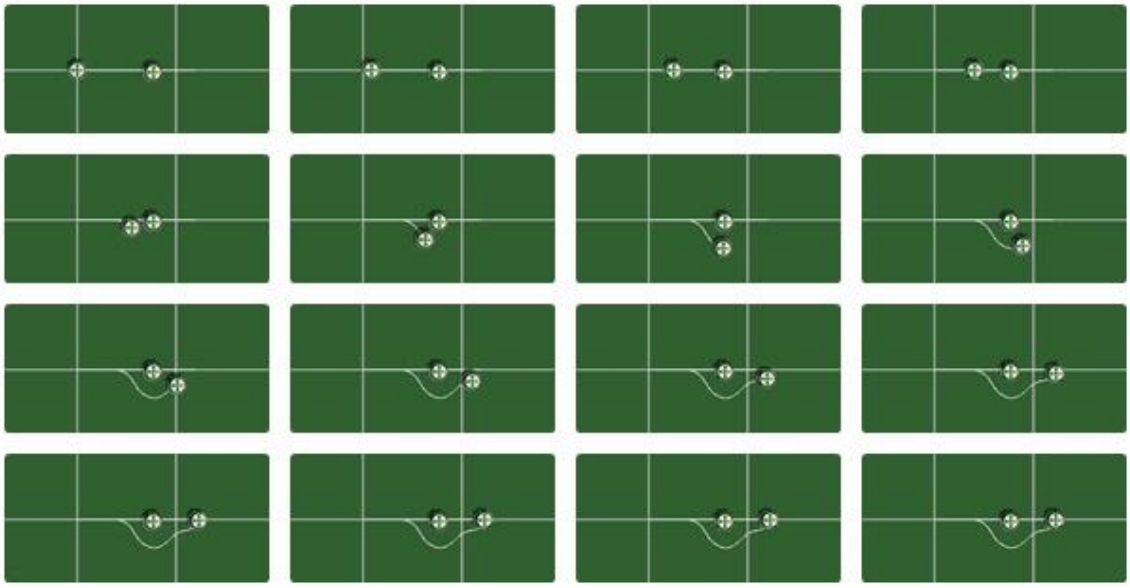
Direcção 1ª Situação -Fotogramas da simulação Direcção pretendida



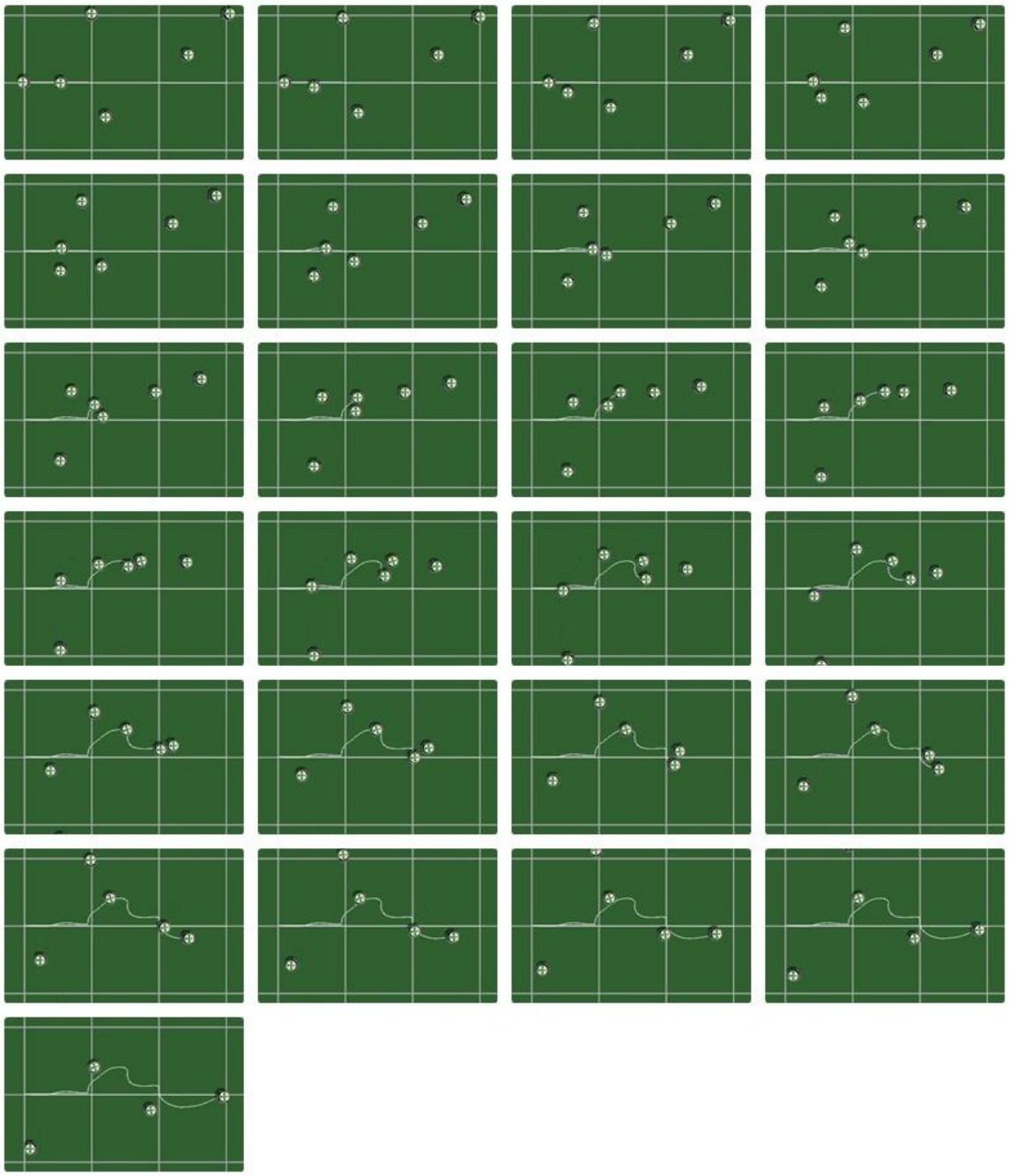
Direcção 2ª Situação -Fotogramas da simulação  1ª Caso



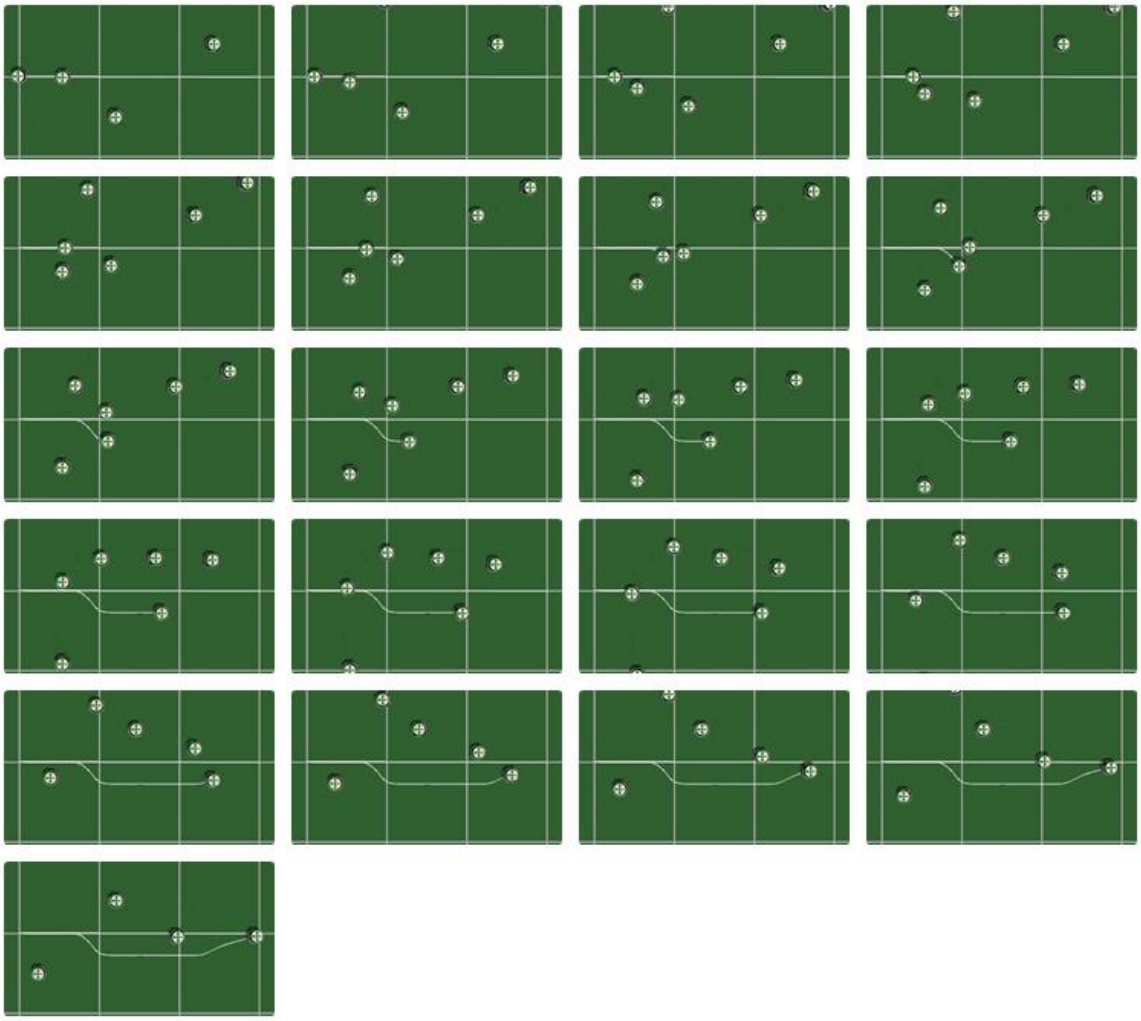
Direcção 2ª Situação -Fotogramas da simulação  2ª Caso



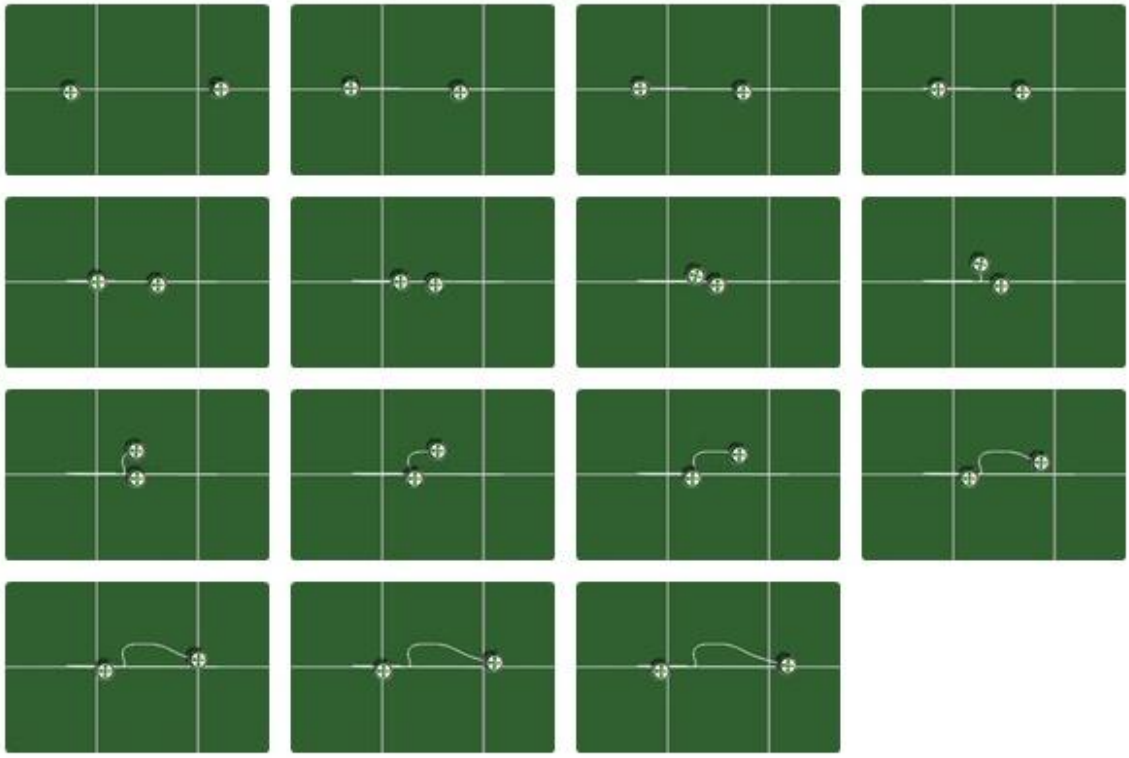
Todos as modificações Fotogramas da simulação  Normal



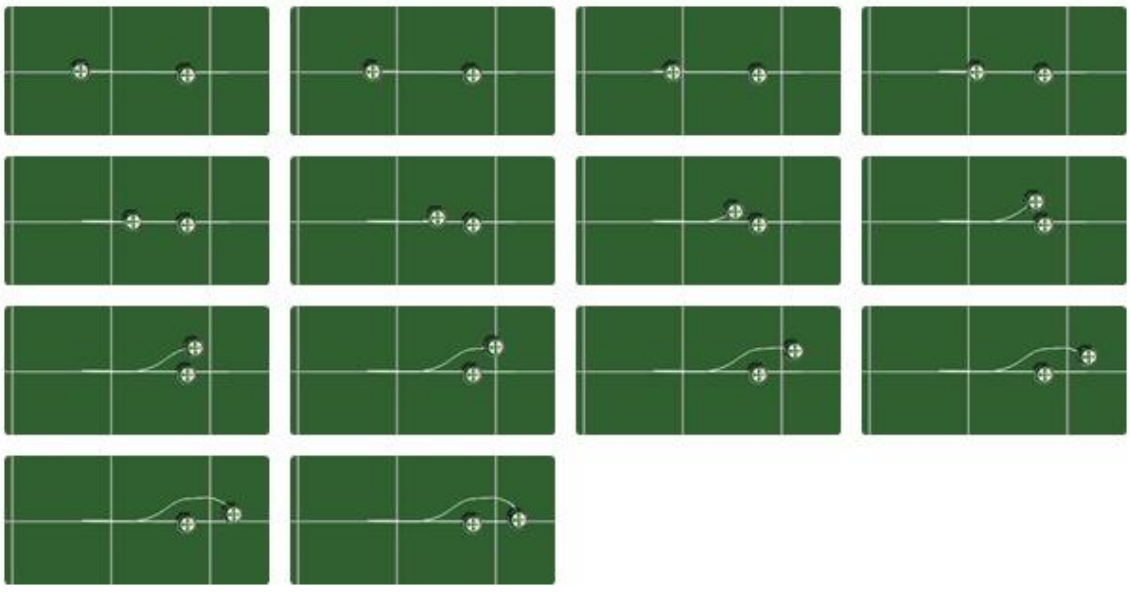
Todos as modificações Fotogramas da simulação  Modificado



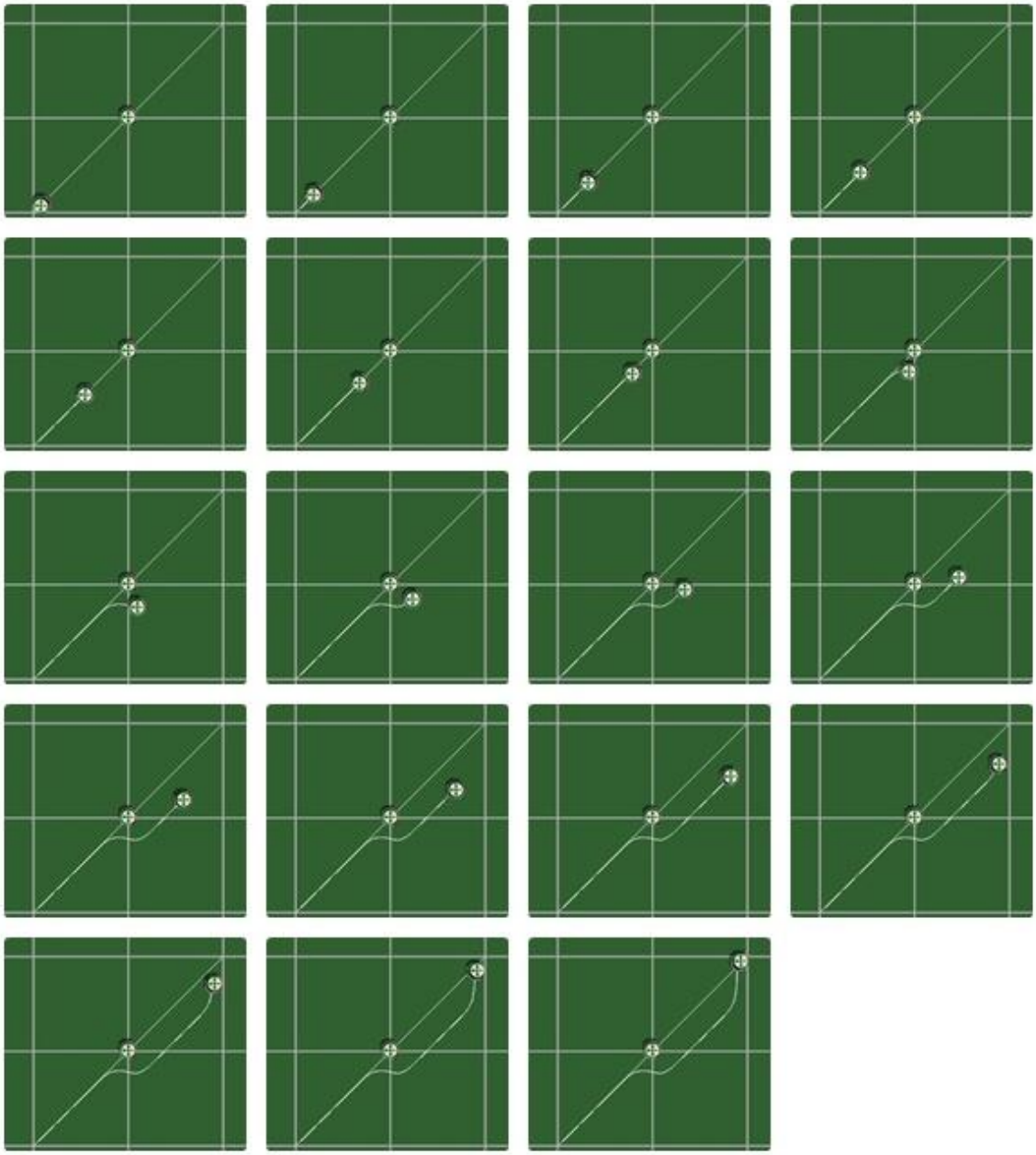
Alteração de trajectórias – 1ª Situação - Fotogramas da simulação – Normal



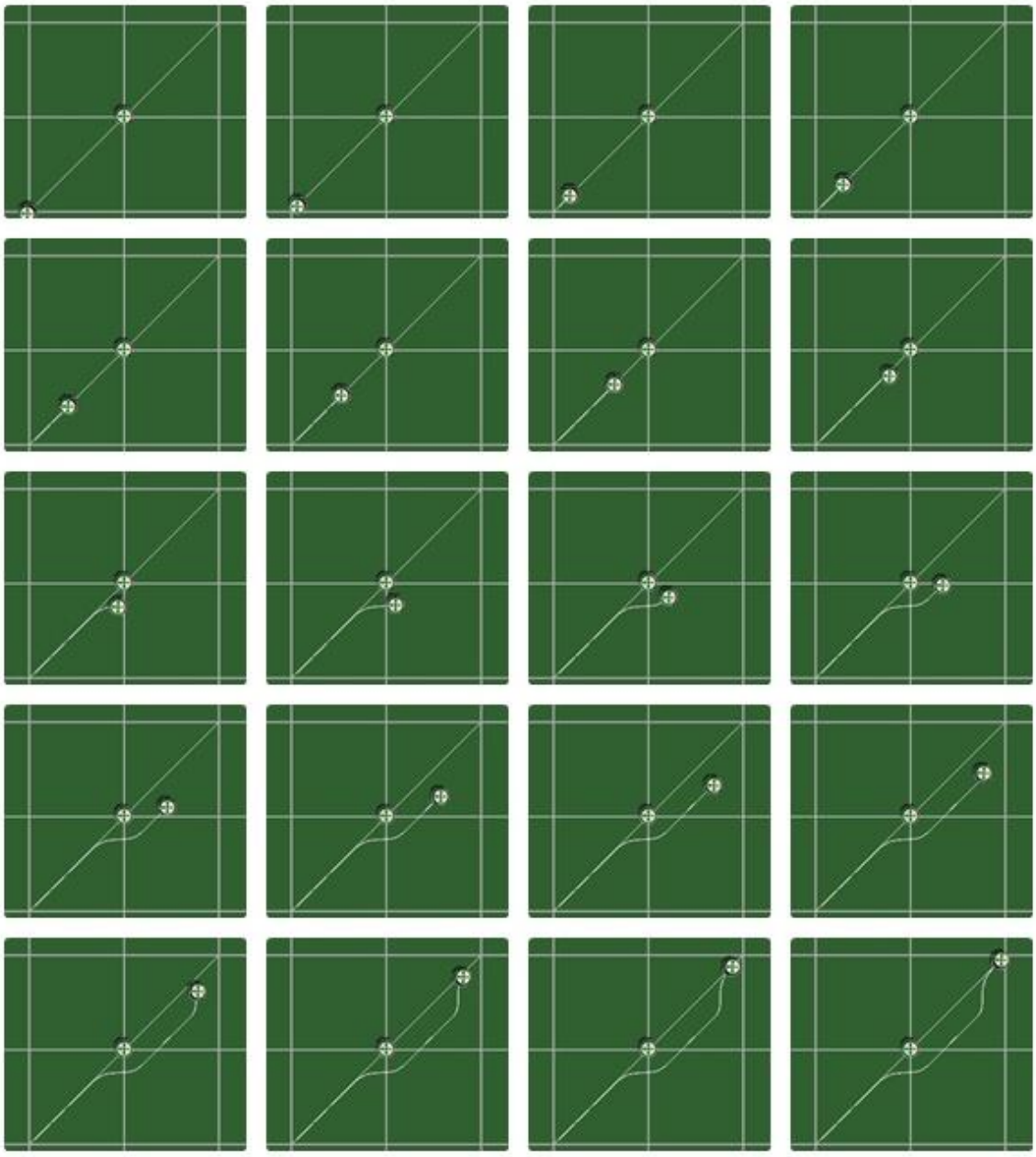
Alteração de trajectórias – 1ª Situação - Fotogramas da simulação – MOD5



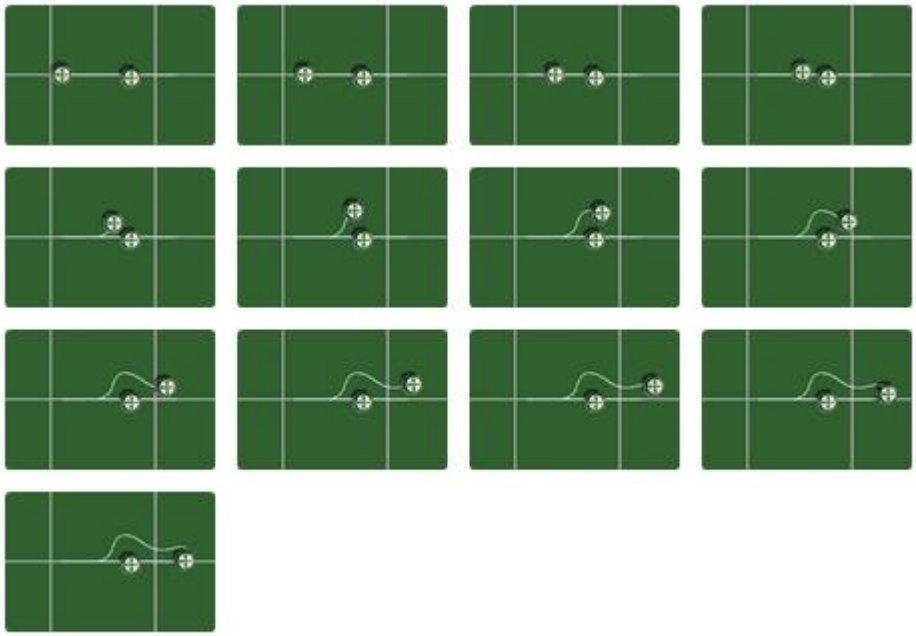
Alteração de trajectórias – 2ª Situação - Fotogramas da simulação – Normal



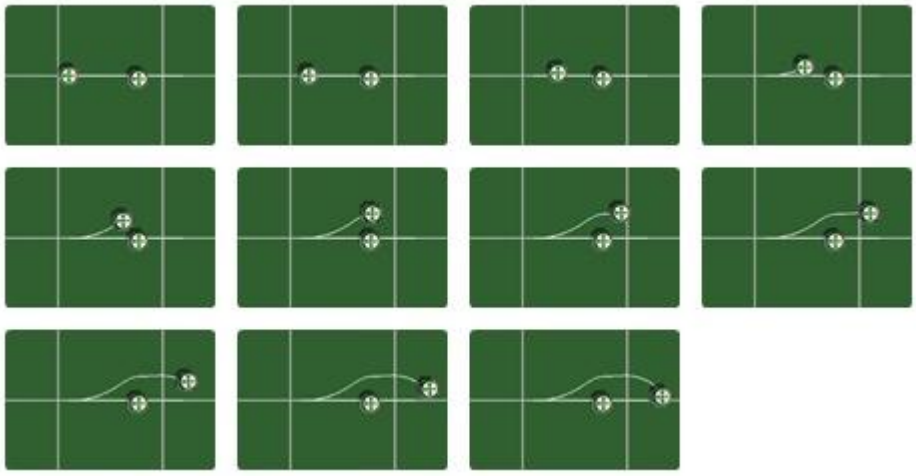
Alteração de trajectórias – 2ª Situação - Fotogramas da simulação – MOD5



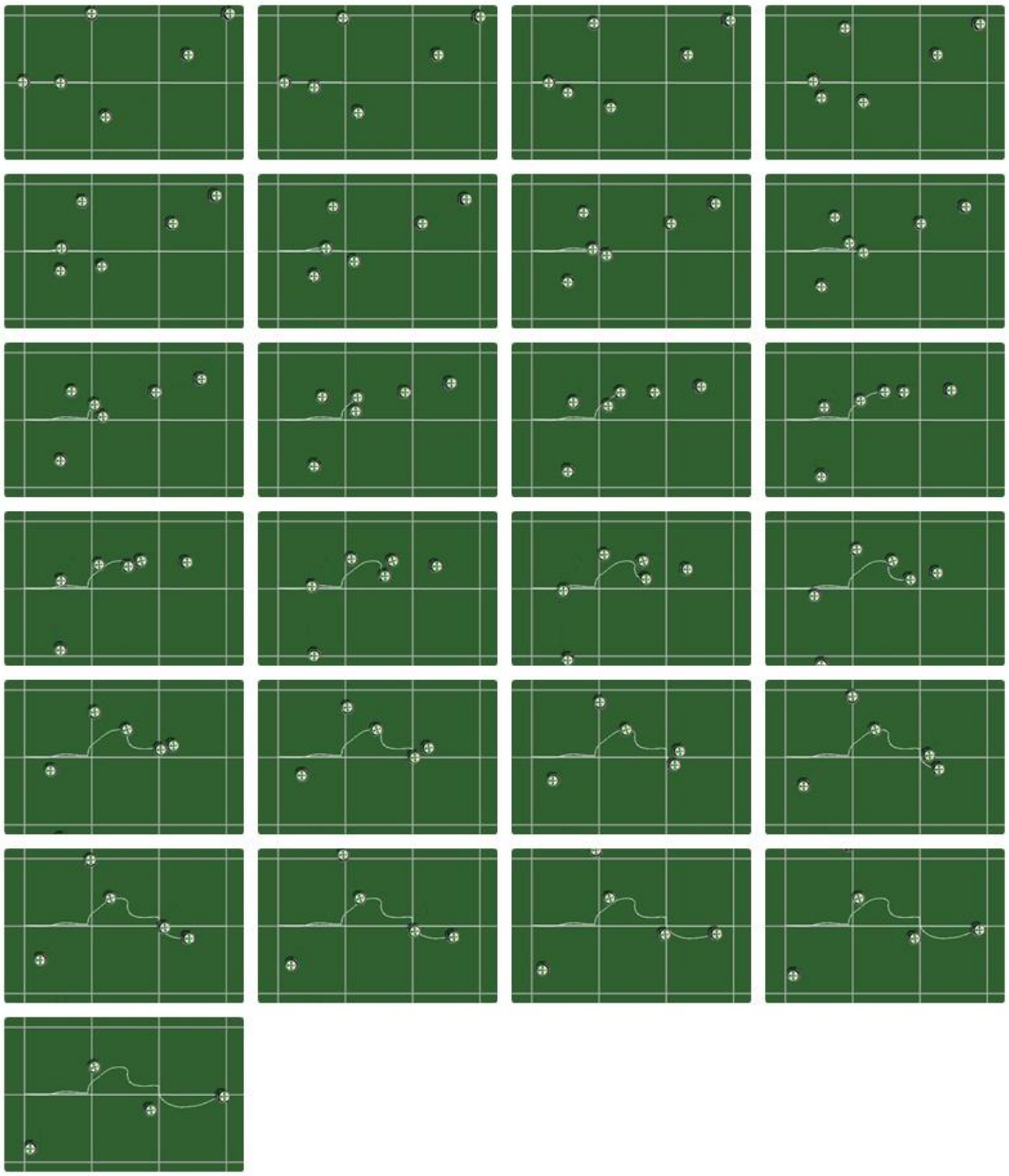
Alteração de trajectórias – 3ª Situação - Fotogramas da simulação – Normal



Alteração de trajectórias – 3ª Situação - Fotogramas da simulação – MOD5



Alteração de trajectórias – 3ª Situação - Fotogramas da simulação – Normal



Alteração de trajectórias – 3ª Situação - Fotogramas da simulação – MOD5