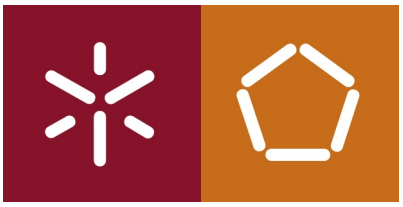

Universidade do Minho
mestrado integrado em engenharia informática

Sistemas Autónomos

Sistemas Inteligentes

(2º Semestre / 4º Ano)

TP1 - Programação de Robôs - Competição Robocode

GRUPO 10
Diogo Braga (a82547)
Pedro Ferreira (a81135)
Ricardo Caçador (a81064)
Ricardo Veloso (a81919)

Março, 2020

Conteúdo

1 Introdução

O presente relatório foi elaborado no âmbito do primeiro trabalho prático da Unidade Curricular de Sistemas Autónomos, inserida no perfil de especialização em Sistemas Inteligentes.

Neste trabalho prático é explorado um dos sub-domínios associados à Inteligência Artificial: programação de robots. Umas das formas mais simples de introduzir este tema, é através da utilização da plataforma `Robocode`.

Na ótica dos sistemas autónomos, os robots encapsulam processos de simulação de comportamento individuais, de grupo e sociais. Assim, num primeiro momento, era pretendido que fosse programado um robot capaz de circum-navegar três obstáculos e regressar à posição inicial. Além disso, deveria ser implementado um odómetro que medisse a distância total do percurso realizado pelo robot. De seguida, procurando explorar a vertente social e o ambiente de batalhas disponibilizado pelo `Robocode`, era pretendido que fosse desenvolvida uma equipa de robots capaz de combater contra outras equipas. Esta tarefa permitiu consolidar tópicos relacionados com a área de Agentes Inteligentes, sendo a equipa abstraída como um Sistema Multiagente. Deste modo, implementamos técnicas de cooperação, de execução de tarefas em grupo e de tomada de decisão.

O relatório encontra-se organizado da seguinte forma: na secção 2 é descrita a implementação do odómetro; na secção 3 é descrito o processo de circum-navegação; a elaboração de uma equipa de robots, assim como a estratégia inerente a esta são descritas na secção 4; por fim, conclui-se na secção 5.

2 Odómetro

Um dos objetivos deste trabalho prático passava pela implementação de um odómetro que permitisse avaliar a distância percorrida por um determinado robô, durante uma corrida. Neste caso, a corrida correspondia à circum-navegação de três outros robôs, com partida e chegada na posição (18, 18). Para efeitos de comparação e análise de precisão do odómetro desenvolvido, foi disponibilizado um outro pela equipa docente: o `StandardOdometer`. Era pretendido que os valores medidos por cada um deles fossem idênticos.

A distância total percorrida pode ser obtida de uma forma acumulativa, isto é, a cada *turn* incrementamos o valor de uma variável na distância que o robô se deslocou. Para tal, é necessário registar as diferentes posições. No caso do `StandardOdometer`, implementado sobre forma de uma `Condition`, as diferentes posições são registadas aquando da execução do método `test()`. Procurando uma alternativa de implementação, a solução desenvolvida pelo grupo (classe `CustomOdometer`), passa por captar as mudanças de posição através do método `onStatus(StatusEvent event)`, que é chamado a cada turno executado pelo robô:

```
1 void onStatus(StatusEvent event){
2     RobotStatus rs = event.getStatus();
3     Position current = new Position( rs.getX(), rs.getY());
4     myOdometer.registerPosition(current);
5 }
```

No método `registerPosition(Position p)`, é calculada a distância euclidiana entre a posição passada como argumento e a posição anterior do robô, guardada num *array* que armazena todas as posições registadas até ao momento. O valor resultante é, então, somado a uma variável `totalDistance`. Além disso, o construtor da classe `CustomOdometer` recebe como argumento um `AdvancedRobot` de forma a que lhe seja adicionado `CustomEvent`. Este, por sua vez, representará o final da corrida:

```
1     robot.addCustomEvent(
2             new Condition(name) {
3                 @Override
4                 public boolean test() {
5                     return !isRunning && raceCompleted;
6                 }
7             }
8     );
```

Desta forma, conseguimos saber quando imprimir o valor final da distância percorrida durante a corrida e quando podemos parar de registar novas posições no odómetro.

Em suma, tendo em conta a frequência com que são registadas as posições ocupadas pelo robô, o valor medido pelo `CustomOdometer` não só é bastante preciso como próximo daquele que é registado pelo `StandardOdometer`, conforme se pode verificar na seguinte figura:

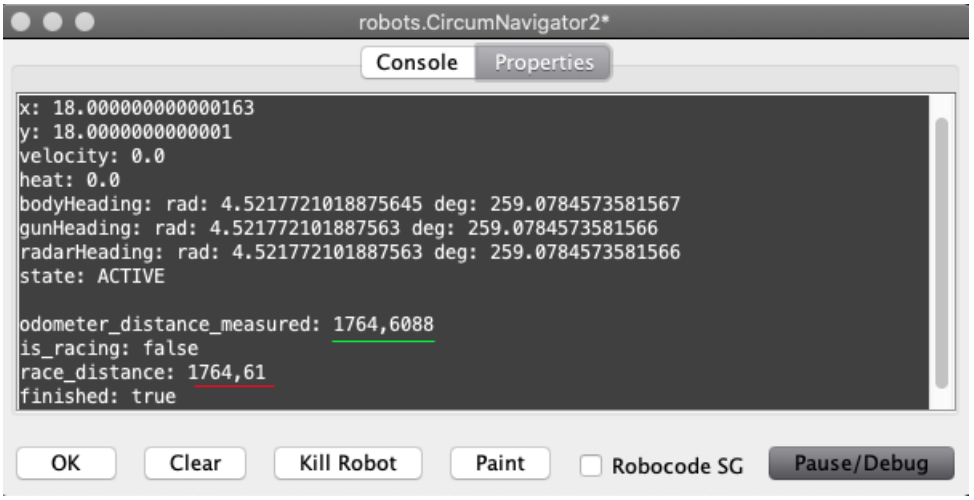

Figura 1: Valores medidos pelos odómetros; A verde - CustomOdometer; A vermelho - StandardOdometer.

3 Implementação de técnicas de planeamento de trajetórias

Neste tralho prático era também pretendido que fosse implementado um robô que, partindo da posição (18,18), realizasse uma navegação por fora da área demarcada por três obstáculos, no sentido dos ponteiros do relógio, terminando na posição de partida, sendo o objetivo principal minimizar a distância percorrida nesse trajeto. Assim, podemos desde já diferenciar entre dois momentos distintos na tarefa do robô: a deslocação para a posição inicial (descrita na secção 3.1) e a circum-navegação dos três obstáculos (abordada na secção 3.2).

3.1 Posicionamento inicial

Na inicialização de cada ronda o robô surge numa posição aleatória e, para iniciar a navegação pelos obstáculos, este deve deslocar-se até ao canto inferior esquerdo do mapa, mais concretamente a posição (18,18).

Para tal ser possível executar, o robô deve rodar de forma a direcionar-se para o canto inferior esquerdo, e depois percorrer a distância a que está dessa posição.

Desta forma, são executados os seguintes passos:

1. Identificação das coordenadas finais (18,18);
2. Identificação das coordenadas atuais do robô;
3. Cálculo da distância entre os dois pontos;
4. Utilizando o teorema de pitágoras, cálculo do ângulo complementar, como explicado na figura 4 (beta);
5. Rotação consoante o ângulo calculado (considerar `getHeading()`, pois o robô não inicia exatamente nos 0 graus (norte), e assim essa diferença é tida também em consideração);
6. Movimentação no sentido frontal da distância, que coincide com a posição final.

Caso seja encontrado algum obstáculo pelo caminho, o robô realiza um pequeno movimento para a esquerda de forma a conseguir desenvencilhar-se e continuar o seu caminho em direção à posição inicial.

```
1  void goTo(double toX, double toY){
2      double fromX = getX();
3      double fromY = getY();
4      double distance =  distanceBetween2Points(fromX, fromY, toX, toY);
5
6      double complementaryAngle = pythagorasTheorem(fromX, fromY, toX, toY);
7
8      double angleToTurn = 180-complementaryAngle;
9
10     turnLeft(normalRelativeAngleDegrees(angleToTurn + getHeading()));
11
12     ahead(distance);
13 }
```

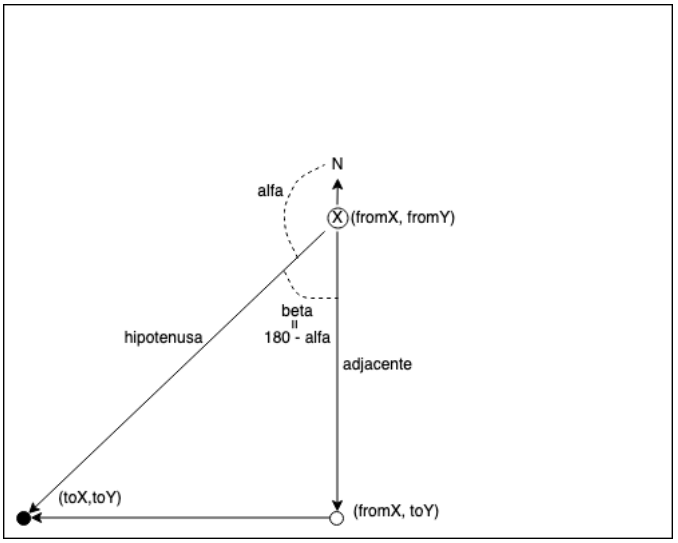

Figura 2: Teorema de pitágoras aplicado à deslocação do robô

```
1  double pythagorasTheorem(double fromX, double fromY, double toX, double toY){
2      double hipotenusa = distanceBetween2Points(fromX, fromY, toX, toY);
3      double adjacente = distanceBetween2Points(fromX, fromY, fromX, toY);
4      double cosB = adjacente/hipotenusa;
5
6      double acosB = java.lang.Math.acos(cosB);
7
8      double acosBDegrees = (180/java.lang.Math.PI)*acosB;
9
10     return acosBDegrees;
11 }
```

3.2 Algoritmo de navegação

Após posicionar o robô na posição inicial, este tem que realizar a navegação circular pelos três obstáculos, sem embater em nenhum deles.

Para este problema foi criada uma solução que se baseia no evento **onScannedRobot**.

Desta forma, são executados os seguintes passos:

1. Posicionamento do robô em direção a norte;
2. Rotação do radar para a direita de forma a detetar o primeiro robô;
3. Rotação do veículo de forma a enquadrar-se com o robô detetado, deixando uma margem de 10 graus à esquerda de forma a não embater contra este;
4. Movimentação até ficar ao lado do robô a circum-navegar (`e.getDistance()`).

5. Fazer uma ligeira curva até detetar o robô seguinte.
6. Ao fim de contornar os 3 robôs, deslocação para a posição inicial.

Apresentamos então uma imagem explícita da abordagem.

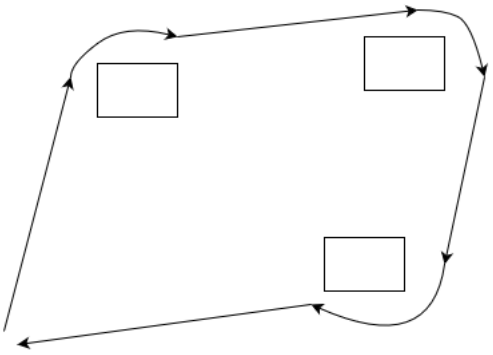

Figura 3: Estratégia de circum-navegação dos robôs

3.3 Resultados

Após a implementação deste algoritmo, foi possível retirar algumas conclusões no que diz respeito à eficácia da estratégia.

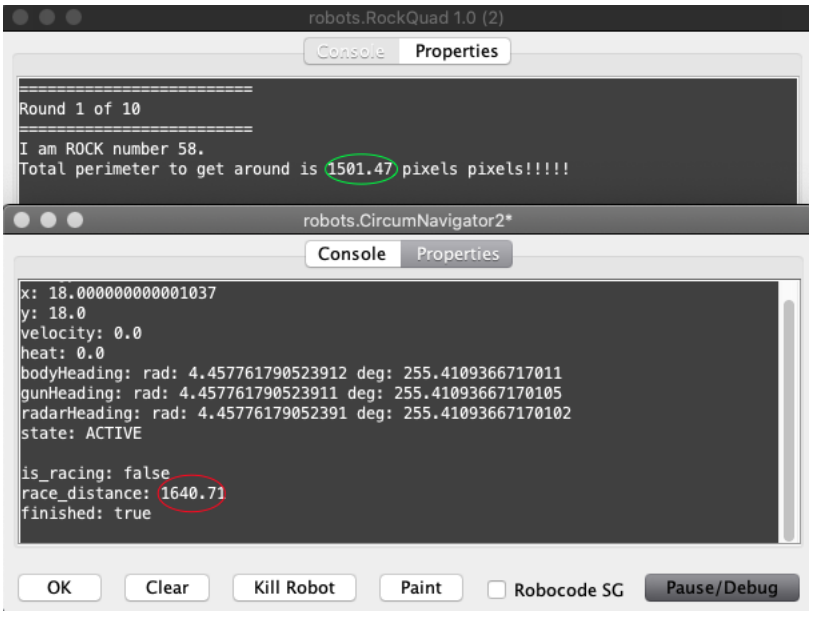

Figura 4: Comparação entre distância percorrida e perímetro dos 3 robôs

Para este caso em específico podemos calcular uma eficácia de cerca de 91.5%, o que representa um valor aproximado à média dos resultados obtidos em experiências anteriores.

4 Formação de equipas com comportamentos sociais

Nesta secção, será detalhada a abordagem seguida para a elaboração de uma equipa de robôs, assim como a estratégia utilizada por esta. Assim, na secção 4.1 serão descritos todos os cenário de comunicação e cooperação entre os membros da equipa, que definem a estratégia geral de combate. De seguida, na secção 4.2 será descrita a movimentação utilizada por todos os robôs perante o ataque a um inimigo. Procede-se com a descrição detalhada de cada um dos tipos de robôs desenvolvidos: na secção 4.3 o `TeamLeader`; na secção 4.4 o `DroidRobot`; na secção 4.5 o `HelperRobot`; por fim, na secção 4.6 o `RunawayRobot`.

4.1 Estratégia geral

Inicialmente, foi feita uma pesquisa exaustiva no que diz respeito à API do Robocode. Esta pesquisa permitiu constatar a existência de robôs com mais energia, que seriam uma mais valia no contexto da batalha. No entanto, esta classe de robôs, denominados por `Droids`, apresenta uma debilidade que consiste na inexistência de *scanner*. Posto isto, poderíamos programar robôs desta classe dotando-os de comportamentos elementares e de uma estratégia que fosse independente da utilização do *scanner*, ou então, com recurso à comunicação entre os membros da equipa, superar a barreira provocada pela ausência de um *scanner* através da difusão de informação por parte dos membros da equipa que apresentavam capacidade de percepcionar o ambiente envolvente.

Tendo em vista a construção de um ambiente cooperativo entre os robôs, optamos pela segunda opção supramencionada. Além disso, procurando intensificar a cooperação entre os membros da equipa, optamos por colocar os `Droids` sob controlo de um robô `TeamLeader`, com intuito de existir coordenação de ações entre ambos. A estratégia delineada passa pelo `TeamLeader` e os `Drois` atacarem o mesmo inimigo, aumentando assim as hipóteses de o eliminar. A escolha de qual inimigo atacar é feita segundo o procedimento que de seguida se explicita. Primeiramente, o `TeamLeader` requisita informação sobre a posição dos `Droids`. De seguida, regista a posição de todos os inimigos através do seu *scanner*. Com base neste par de informações, é escolhido como alvo o inimigo que se encontre mais próximo dos `Droids` e do `TeamLeader`. Por fim, o `TeamLeader` ordena aos `Droids` que ataquem o inimigo escolhido. Finda a seleção de um alvo comum, é necessário que os `Droids` recebam informação atualizada sobre o estado deste, pois, só assim, serão capazes de utilizar uma estratégia de ataque coerente e eficaz. Ora, mais uma vez, o `TeamLeader`, cujo radar se encontra em rotação permanente, é o responsável pelo envio desta informação.

Numa primeira fase, realizamos várias batalhas com um equipa constituída por três `Droids` e um `TeamLeader`. Como seria de esperar, rapidamente se constatou um grande dependência no `TeamLeader`, uma vez que sem este os `Droids` não conseguiam reagir. Deste modo, surgiu a ideia de criar um robô que pudesse, não só ocupar o cargo de *team leader* caso o `TeamLeader` morresse, mas também protege-lo enquanto este estivesse vivo. Chamamos a este robô `HelperRobot` e as suas principais caraterísticas passam pelo facto de agir independentemente do restantes robôs, mas alterar a sua estratégia individual em prol da defesa do `TeamLeader`, quando este for atacado por um inimigo.

Com a introdução do `HelperRobot` conseguimos diminuir a probabilidade de a equipa ficar sem *team leader*. No entanto, esse cenário continua a ser possível pelo que implementamos uma estratégia extra para garantir que os `Droids` não permanecem imóveis. Assim, caso o `TeamLeader` e o `HelperRobot` morram, os `Droids` adotam o comportamento de um `WallsRobot`, isto é, deslocam-se ao longo das paredes do mapa, disparando contra inimigos com os quais colidam. Desta forma, prolongarão o seu tempo de vida e conseguirão escapar a ataques de alguns inimigos, fazendo com que estes gastem energia de forma desnecessária.

Em suma, a equipa desenvolvida pelo grupo, para efeitos de torneio, é composta por um

`TeamLeader` que age em conjunto com dois `Droids` e é apoiado por um `HelperRobot`. Esta equipa exibe comportamentos sociais de cooperação, implementados tendo por base a comunicação entre os robôs.

4.2 Movimentação dos robôs

Todos os robôs realizam o mesmo género de movimentos, sendo por isso estes unificados e explicados nesta secção.

A imobilidade do robô é evitável em qualquer situação, tendo em conta que este fica muito mais propício a sofrer ataques caso não esteja em constante movimentação.

Desta forma, o movimento base escolhido foi em arco. Realizando este movimento circular, a posição do robô fica ainda mais difícil prever, ao contrário do que aconteceria com um movimento em linha reta.

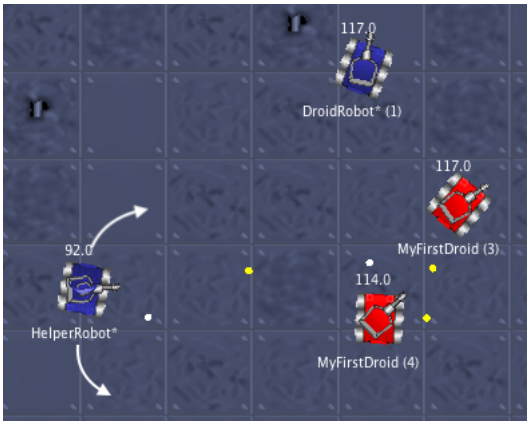

Figura 5: Movimentação em arco

Para tal ser executado, o robô tem que se encontrar numa posição perpendicular ao *target* definido (`target.getBearing()+90`).

Além da movimentação ser em arco, de 5 em 5 *ticks* o robô vai alterando a direção no qual o realiza, criando ainda maior imprevisibilidade (`moveDirection *= -1;`).

De forma a tornar o ataque mais perigoso, o movimento é realizado com aproximação progressiva ao alvo selecionado (`- (15 * moveDirection)`).

```
1    setTurnRight(target.getBearing() + 90 - (15 * moveDirection));
2
3    if (getTime() % 5 == 0) {
4        moveDirection *= -1;
5        setAhead(4000 * moveDirection);
6    }
```

Caso embata num obstáculo (velocidade nula), a direção é, de igual forma, alterada.

```
1     if (getVelocity() == 0)
2         moveDirection *= -1;
```

4.3 TeamLeader

O `TeamLeader` realiza a maior parte das suas funções após scanear um inimigo. Estas funções dependem do estado em este se encontra. Os estados são:

- **Modo Planeamento** - Na fase de planeamento o `TeamLeader`, inicialmente, recolhe todas as informações relativas ao estado do jogo ou seja, as posições dos inimigos e as posições dos próprios colegas de equipa. Após obter todas estas informações, escolhe o inimigo mais desejável para atacar através de uma função que determina o inimigo mais próximo de todos os robôs da equipa e, de seguida, coordena e realiza o ataque.
- **Modo Combate** - Na fase de combate o `TeamLeader` simplesmente coordena e ataca em conjunto com os `Droids` o inimigo selecionado. Além disso, envia constantemente informação atualizada sobre esse inimigo aos `Droids`, para que estes o possam atacar corretamente. Após a morte do inimigo selecionado, o `TeamLeader` volta ao modo de planeamento para poder voltar a definir uma nova estratégia.

4.4 DroidRobot

O `DroidRobot` apenas segue as ordens do `TeamLeader`. Enquanto o `TeamLeader` está vivo, aguarda por informações sobre o inimigo a atacar e procede a atacá-lo através da movimentação explicada na secção **4.2**.

Adicionalmente caso o `HelperRobot` e o `TeamLeader` morram, o `DroidRobot` inicia um movimento que consiste em circular à volta do mapa, junto às paredes, sem realizar qualquer disparo para poupar energia e tentar esgotar o tempo. Desta forma, em situações muito específicas, é possível alcançar a vitória através do desgaste dos robôs da equipa adversária.

4.5 HelperRobot

Este robô pode funcionar em três modos. São estes:

- **Independente:** modo inicial e mais frequente do robô no jogo; possui o radar em rotação permanente de forma a manter informação atualizada sobre os inimigos; seleciona o inimigo mais próximo e efetua os disparos atacando num movimento em arco, com aproximação progressiva.
- **Ajuda ao líder:** se o `TeamLeader` for atacado, é enviada uma mensagem para o `HelperRobot` contendo informação sobre o inimigo responsável pelo ataque; este dá total prioridade a essa ação e ataca o robô identificado, de forma a providenciar alguma proteção ao líder da equipa; depois de realizado o ataque, volta ao modo independente.
- **Líder da equipa:** caso o `TeamLeader` inicialmente definido tenha sido derrotado, o `HelperRobot`, através do método `onRobotDeath(RobotDeathEvent e)`, deteta a sua morte; a partir desse momento, o `HelperRobot` passa a comandar os `Droids` para atacar um inimigo, disparando também nesse sentido.

4.6 RunawayRobot

Este robô, como o próprio nome indica, é um robô cuja sua principal função é fugir dos inimigos, procurando perturbar as ações deles. Assim, foram implementados vários mecanismos de forma a tornar o movimento deste robô o mais eficaz de forma a que os inimigos tivessem problemas a acertar os tiros nos robôs. Um `RunawayRobot` começa por ir ter com um inimigo e depois foge deste. Durante a sua fuga, procura esquivar-se aos ataques do inimigo, e vai disparando sobre o inimigo, até que este morra.

Apesar de tudo e depois de fazermos alguns testes chegamos à conclusão que a utilização deste robô na equipa não aumentava as nossas chances de vencer pelo que não faz parte da equipa principal.

5 Conclusão

A realização deste trabalho permitiu ao grupo iniciar-se na programação de robots. Na primeira tarefa, começamos por circum-navegar três obstáculos parados, tentando minimizar a distância total percorrida. Este desafio permitiu-nos ambientar ao desenvolvimento na plataforma `Robocode`, assim como às caraterísticas de um robot. Na segunda tarefa, com a elaboração de uma equipa completa, pudemos aprofundar o nosso conhecimento em `Robocode` e, adicionalmente, explorar e aplicar conceitos relacionados com Sistemas Multiagente. De facto, a equipa desenvolvida exibe comportamentos sociais de cooperação (partilha de informação entre os membros, coordenação de ataques e tomada de decisão sobre qual inimigo atacar). Todos estes comportamentos têm por base a plataforma de comunicação disponibilizada pelo `Robocode`, entre `TeamRobots`.

Tendo em conta os resultados obtidos, na primeira tarefa, fomos capazes de circum-navegar os três obstáculos com uma eficácia de 91.5%. Quanto à segunda tarefa, a equipa por nós desenvolvida consegue derrotar sistematicamente a `SampleTeam`, mesmo estando em desvantagem numérica. No entanto existem alguns problemas evidentes, tais como a questão de *friendly fire* e a existência de colisões entre membros da mesma equipa.