Robótica

Relatório do Trabalho Final - NXT

Autores: Alexandra Videira - 37246
João Barata Oliveira – 31559
Pedro Martins - 31501

Docentes Responsáveis:

Regente: José Barata de Oliveira

Responsável Práticas: Eduardo Pinto

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Junho, 2015

Índice

Índice de Figuras

Introdução

A robótica móvel é um tema que hoje em dia é alvo de um grande foco de investigação. Seja em ambiente fabril com AGV responsáveis pelo transporte de peças e produtos entre diversas zonas de produção, como em ambiente doméstico com *robots* móveis que permitem ajudar nas tarefas domésticas, tais como como aspirar uma divisão.

Figura 1 - Exemplo de um AGV de transporte de explosivos

Além destas aplicações, existe uma infinidade de outras possíveis aplicações para um *robot* móvel, entre as quais no domínio da busca e salvamento. Hoje em dia já existem inclusive *robots* comerciais que têm vocação para serem usados em ambientes hostis de busca e salvamento, sendo um deles o Introbot produzido pela Introsys – empresa que tivemos a oportunidade de visitar no âmbito desta cadeira.

Figura 2 - Exemplo de um robô utilizado para busca e salvamento - Introbot

Este trabalho consiste essencialmente no desenvolvimento de um *software* utilizando NXC para a plataforma robótica Lego Mindstorms NXT. A grande finalidade deste trabalho passa não só pela oportunidade que os alunos têm de trabalhar com um *robot* móvel comercial, mas também pela possibilidade de desenvolvimento de algoritmos que permitam resolver um labirinto da forma mais rápida possível. Esse mesmo labirinto pode ser considerado como uma simulação do ambiente desafiante e imprevisível que um verdadeiro *robot* de busca e salvamento irá encontrar durante a sua operação.

Objetivos do trabalho

O objetivo deste trabalho é o desenvolvimento de um programa que permita a um *robot* Lego NXT resolver um labirinto. Ao longo do labirinto existem dois modos: o robot ser ajudado através de linhas de seguimento ou ter de se movimentar em navegação autónoma. Durante o caminho o robot *tem* de conseguir detetar dois tipos de vítimas: o primeiro tipo será indicado por riscas cinzentas transversais à linha preta de seguimento e o segundo tipo será indicado por um gradiente de temperatura reconhecível através da utilização de células de *Peltier*, ou em alternativa resistências de aquecimento. Ao detetar qualquer uma das vítimas, o *robot* deverá parar e assinalar claramente através de sons distintos e mensagens no seu *display* qual o tipo da vítima e qual o número de vítimas desse tipo que encontrou até esse preciso momento.

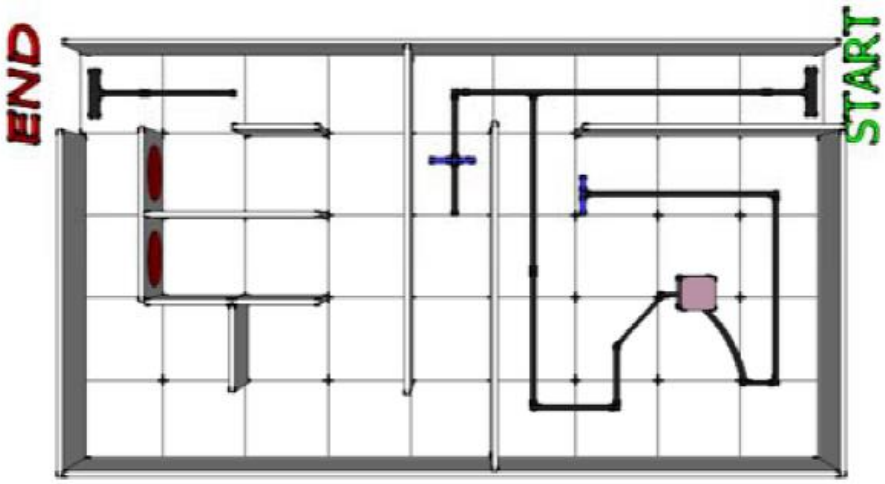

Figura 3 - Exemplo de um possível labirinto

Pelo caminho o *robot* tem de ser capaz de resolver qualquer tipo de curvas da linha de seguimento sem que perca a mesma, inclusive as curvas de 90 graus, bem como vários tipos de intersecções entre linhas tendo o robô de continuar na direção pretendida.

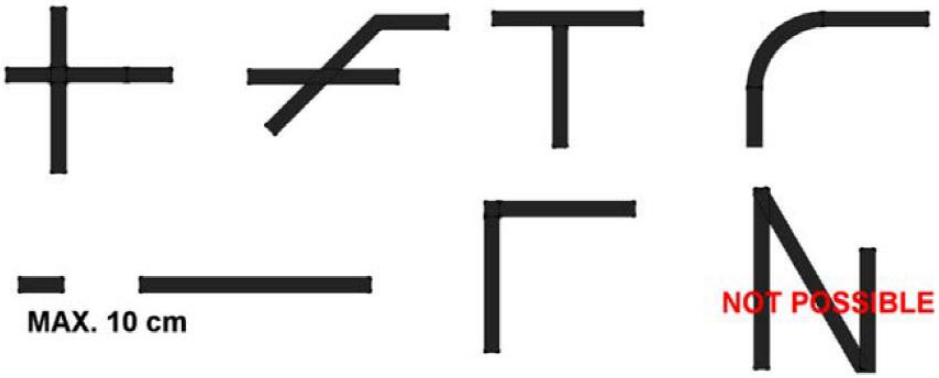

Figura 4 - Exemplos de cruzamentos possíveis

No final do labirinto, o *robot* também tem de ser capaz de identificar que atingiu o seu objetivo através das marcações presentes no final do mesmo e emitir um som que permita informar o utilizador que atingiu o seu objetivo. No seu *display* deverá indicar o tempo que o mesmo demorou a percorrer o labirinto, assim como o número e respetivo tipo de vítimas encontrado durante o percurso realizado.

Enquadramento Teórico

Análise Inicial

Antes de iniciar o desenvolvimento do algoritmo foi necessária a realização de alguma sobre como abordar o problema da resolução do labirinto. No seu decorrer, deparámo-nos com alguns algoritmos que nos ajudaram a melhorar o algoritmo implementado.

Algoritmo para resolução de Labirinto

US Labyrinth Solver

Um dos algoritmos encontrados para conseguir navegar dentro de um labirinto aquando da nossa pesquisa foi o *US Labyrinth Solver*. (1)

Figura 5 - Robot utilizado pelo US Labyrinth Solver

Tal como ilustrado na figura acima, o *robot* é constituído por um sensor ultrassom que está constantemente a rodar 180 graus de maneira a que possa medir a distância entre o *robot* e as paredes que o rodeiam.

Devido às limitações de *hardware* do Lego NXT, a rotação do sensor torna-se muito lenta, visto só ser possível fazer 30 leituras por segundo, o que em movimento se torna rapidamente insuficiente para assegurar a segurança do *robot*. Também com o objetivo de evitar medições incorretas e colisões, o algoritmo tem em conta as médias das leituras realizadas anteriormente de forma a descartar valores implausíveis.

O algoritmo de navegação é simples, visto que o robot sonda a área à sua frente procurando a direção onde tem o obstáculo mais próximo. Após isso, o robot compara essa direção com as duas leituras anteriores, faz uma média da direção e altera a rotação do motor para que não colida com a parede. Para a deteção de possíveis colisões são utilizados sensores de contacto na construção do robot tal como se pode observar na figura 5.

Na nossa opinião, a grande desvantagem da utilização deste algoritmo é o facto de este apenas ser aplicável em locais com paredes estreitas e assim torna a sua aplicação inviável no modo de navegação autónoma implementado no nosso trabalho. Neste caso o objetivo não era descobrir a saída, mas apenas seguir o equivalente ao túnel presente no labirinto do robot. Mas poderia ser uma solução para evitar colidir com as paredes dentro do túnel.

Seguimento de uma parede exterior

Um dos algoritmos mais famosos para resolver um labirinto é o de nos guiarmos por uma parede até chegarmos a uma saída caso ela exista ou então ao ponto onde começamos mas na parede oposto à que inicialmente estávamos a seguir.

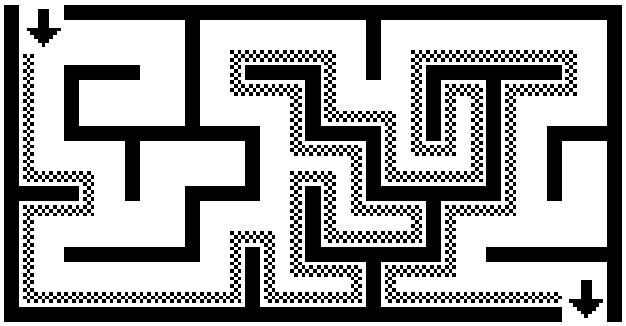

Figura 6 - Exemplo da travesia de um labirinto seguindo a parede direita

No entanto este algoritmo só funciona se o labirinto for simplesmente conexo ou seja se todas as paredes estão ligadas entre si ou à parede exterior do labirinto. No caso especifico do labirinto deste trabalho, o labirinto não é simplesmente conexo devido ao

beco central mas implementando um sistema de mapeamento é possível detetar que houve uma área não visitada no labirinto e ultrapassar assim esta limitação.

Mapeamento do Labirinto

O mapeamento robótico (2) tem sido amplamente estudado desde os anos 1980. O mapeamento pode ser classificado como métrico ou topológico. A abordagem métrica é uma que determina as propriedades geométricas do ambiente em que o robot se encontra ao passo que a abordagem topológica determina as relações entre pontos de interessa no ambiente em que se encontra o robot.

Dado que conhecemos as medidas do labirinto podemos utilizar neste caso uma abordagem topológica e calcular a posição do robot através dos tacómetros dos motores. Uma abordagem que pode ser utilizada é utilizar os sensores de ultrassons para determinar a distância do robot a um obstáculo e de acordo com a sua orientação, e posição, calcular a posição do obstáculo. (3)

Esta abordagem permite ao robot navegar de forma autónoma e mapear o mundo que o rodeia obtendo os resultados vistos na Figura 7.

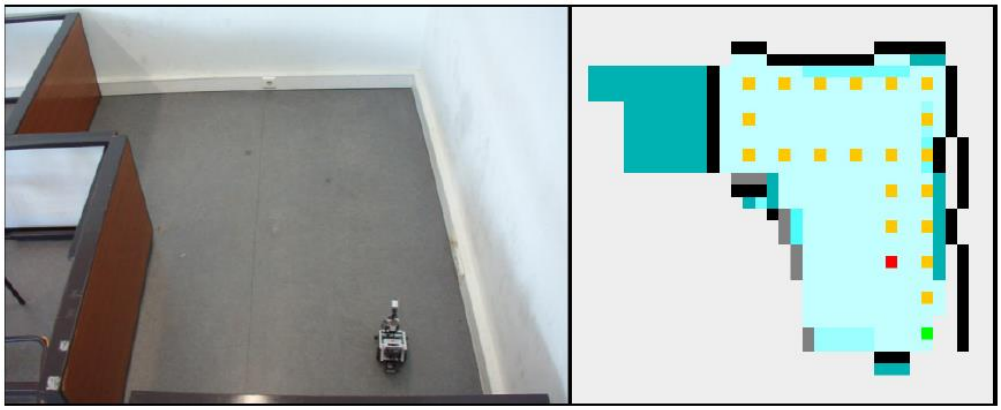

Figura 7 - Exemplo de um ambiente e do mapa obtido pelo NXT. (3)

Características do Robot

Uma das características interessantes deste trabalho foi a total liberdade dada pelo corpo docente na construção do robot a ser utilizado ao longo do trabalho. Isso permitiu que desenvolvêssemos o nosso robot à medida das nossas necessidades.

Figura 8 - Robot construído

A análise dos requisitos funcionais do trabalho levou a que o nosso robot fosse constituído por:

- Um *brick* Lego NXT programado em NXC
- Dois sensores de luminosidade
- Um sensor de ultrassom
- Uma termopilha
- Três motores

Sensores de luminosidade

Os dois sensores de luminosidade são utilizados para detetar a linha preta na primeira parte do percurso, pois o robot segue-a para conseguir realizar o seu caminho, o que permite realizar as curvas e cruzamentos presentes no percurso dado. Estes dois sensores servem ainda para a deteção das vítimas de tipo 1, que são colocadas

perpendicularmente à fita preta que delimita o caminho. Por último, permitem ainda a entrada do robot no túnel também através do seguimento da mesma linha.

Sensor de ultrassom

O sensor de ultrassom permite ao robot medir a distância perante um determinado obstáculo ou parede. Na parte da navegação autónoma, este sensor permite medir as distâncias a cada parede e assim determinar qual a maior delas, o que permite realizar uma viragem de 90 graus para que o robot continue a andar até terminar o seu percurso. Durante a navegação autónoma o robot está constantemente a girar este sensor de forma a poder avaliar a distância a que se encontra da parede e assim deslocar-se paralelamente à mesma ao mesmo tempo que avalia a que distancia se encontra da parede imediatamente em frente.

Termopilha

Este sensor permite medir uma temperatura e assim poder detetar o gradiente de temperatura característico das vítimas de tipo 2. A grande vantagem deste sensor é ele ser capaz de medir uma temperatura a uma certa distância dispensando assim o contacto direto com a fonte de calor.

Motores

Dois dos três motores são utilizados para que o movimento do robot seja possível. O outro motor é utilizado para o movimento do sensor de ultrassom que mede as distâncias conforme referido na secção dedicada ao Sensor de ultrassom.

Software desenvolvido

Arranque e calibração

Assim que se inicia a execução do programa, a tarefa *main* chama a sub-rotina *calibration* cuja implementação pode ser vista na Figura 9. Esta sub-rotina é na prática uma máquina de estados que numa primeira abordagem pede ao utilizador para colocar o robot por cima da linha de partida de forma a estimar uma gama de valores para determinar quando os sensores de seguimento estão por cima da linha de seguimento.

```
sub calibration(){
  int calibrationDone = 0;
  while (calibrationDone < 2){
    mydist.d3 = SensorUS(IN_2);
    mylight.sensor_in3 = Sensor(IN_3);
    mylight.sensor_in4 = Sensor(IN_4);
    if (calibrationDone == 0) {
      TextOut(1, LCD_LINE1, "Calibrate Lines", true);
      if (mydist.d3 < NEAR){
        PlayToneEx(300, 400, 5, FALSE);
        Treshold_Line = mylight.sensor_in3;
        Treshold_Line1 = mylight.sensor_in4;
        TextOut(5, LCD_LINE2, "Done", true);
        NumOut(60, LCD_LINE3, Treshold_Line);
        NumOut(60, LCD_LINE4, Treshold_Line1);
        Wait(2000);
        calibrationDone++;
      }
    }
    mydist.d3 = SensorUS(IN_2);
    if (calibrationDone == 1) {
      TextOut(1, LCD_LINE3, "Calibrate Victims", true);
      if (mydist.d3 < NEAR){
        PlayToneEx(300, 400, 5, FALSE);
        Treshold_Victim = mylight.sensor_in3;
        Treshold_Victim1 = mylight.sensor_in3;
        TextOut(5, LCD_LINE2, "Done", true);
        NumOut(60, LCD_LINE3, Treshold_Victim);
        NumOut(60, LCD_LINE4, Treshold_Victim1);
        Wait(100);
        calibrationDone++;
      }
    }
  }
  do{
    mydist.d3 = SensorUS(IN_2);
  } while (mydist.d3 >= NEAR)

    PlayToneEx(262, 400, 3, FALSE);
  Wait(500);
  PlayToneEx(250, 400, 3, FALSE);
}
```

Figura 9 - Código da sub-rotina de calibração

De seguida o *software* pede ao utilizador para colocar o robot sobre a linha das vítimas novamente de forma a calibrar a gama de valores em que o robot vai assumir que está perante uma vítima.

Depois destes passos realizados a sub-rotina de calibração espera pelo sinal do utilizador ou do árbitro para começar o percurso. Em qualquer das situações referidas anteriormente, o utilizador tem de confirmar a colocação do robot em posição passando a mão à frente do sensor de ultrassons do robot.

Seguimento da linha

Um dos primeiros objetivos do nosso trabalho foi conseguir que o *robot* se guiasse corretamente pela linha de seguimento. Portanto logo desde o início dentro do ciclo de execução presente na tarefa *main* que existe uma chamada à sub-rotina *checkLine* cuja implementação pode ser consultada na Figura 10.

Esta sub-rotina vai ler os valores retornados pelos sensores de luminosidade do robot e vai compara-los com os intervalos obtidos através da calibração. Caso o sensor que detete linha preta seja o da direita, o robot vai virar nessa direção e vice-versa de forma a manter a linha entre os dois sensores.

Enquanto grupo acabamos por decidir fazer uma navegação pelo branco pois a distância entre sensores tornava o robot menos sensível a pequenas variações da linha e assim mantem-se mais tempo a andar em linha reta e menos tempo em correções de trajetória.

Uma questão que é importante referir é que muitas vezes o robot detetava a transição entre branco e linha preta transversal à deslocação do robot como uma “falsa” vítima e portanto esta é a razão de voltar a ler os sensores após detetar vítima, assim evita-se que se detete erradamente uma vítima nesta situação, tornando o programa mais robusto.

```
sub checkLine(){
	int lineCount = 0;

	mylight.sensor_in3 = Sensor(IN_3);
	mylight.sensor_in4 = Sensor(IN_4);

	if ((mylight.sensor_in3 > Treshold_Line - sensorRange && mylight.sensor_in3 < Treshold_Line +
sensorRange) &&
		(mylight.sensor_in4 > Treshold_Line1 - sensorRange && mylight.sensor_in4 < Treshold_Line1 +
sensorRange)){
		moveFwdSyncAC(50, 90, 0);
		mytimer.lineTime = CurrentTick();
	}
	else if (mylight.sensor_in3 > Treshold_Line - sensorRange && mylight.sensor_in3 < Treshold_Line +
sensorRange){
		//Turn Left
		OnFwdSync(OUT_AC, 60, 100);
		while (mylight.sensor_in3 > Treshold_Line - sensorRange && mylight.sensor_in3 <
Treshold_Line + sensorRange && (mylight.sensor_in4 < Treshold_Line1 - sensorRange || mylight.sensor_in4 >
Treshold_Line1 + sensorRange)){
			TextOut(5, LCD_LINE2, "Turning Left", true);
			mylight.sensor_in3 = Sensor(IN_3);
			OnFwdSync(OUT_AC, 60, 100);
		}
		mylight.last_activated = 3;
		mytimer.lineTime = CurrentTick();
	}
	else if (mylight.sensor_in4 > Treshold_Line1 - sensorRange && mylight.sensor_in4 < Treshold_Line1 +
sensorRange){
		//turn right
		OnFwdSync(OUT_AC, 60, -100);
		while (mylight.sensor_in4 > Treshold_Line1 - sensorRange && mylight.sensor_in4 <
Treshold_Line1 + sensorRange && (mylight.sensor_in3 < Treshold_Line - sensorRange || mylight.sensor_in3 >
Treshold_Line + sensorRange)){
			TextOut(5, LCD_LINE2, "Turning Right", true);
			mylight.sensor_in4 = Sensor(IN_4);
			OnFwdSync(OUT_AC, 60, -100);
		}
		mylight.last_activated = 4;
		mytimer.lineTime = CurrentTick();
	}
	else if ((mylight.sensor_in3 > Treshold_Victim - sensorRange && mylight.sensor_in3 < Treshold_Victim
+ sensorRange) && (mylight.sensor_in4 > Treshold_Victim1 - sensorRange && mylight.sensor_in4 <
Treshold_Victim1 + sensorRange)){
		//detect victim, stop and play sound
		Off(OUT_AC);
		mylight.sensor_in3 = Sensor(IN_3);
		mylight.sensor_in4 = Sensor(IN_4);
		if ((mylight.sensor_in3 > Treshold_Victim - sensorRange && mylight.sensor_in3 <
Treshold_Victim + sensorRange) && (mylight.sensor_in4 > Treshold_Victim1 - sensorRange && mylight.sensor_in4 <
Treshold_Victim1 + sensorRange)){
			if ((CurrentTick() - mytimer.victimTime) > 80){
				PlayToneEx(180, 400, 3, FALSE);
				PlayToneEx(180, 400, 3, FALSE);
				Wait(500);
				mytimer.victimTime = CurrentTick();
				victimCount++;
			}
		}
		moveFwdSyncAC(40, 90, 0);
	}
	else{
		OnFwdSync(OUT_AC, 40, 0);
		TextOut(5, LCD_LINE2, "Moving forward");
	}
}
```

Figura 10 - Implementação da sub-rotina checkLines

Obstáculo

Após termos o código que implementa o seguimento da linha, o primeiro obstáculo com que nos deparamos no desenvolvimento do *software* de controlo do robot foi curiosamente o obstáculo no mundo real.

Numa primeira abordagem o robot apenas vencia o obstáculo por um dos lados, mas na versão final do nosso programa o robot é capaz de decidir qual a melhor direção a tomar para contornar o obstáculo.

Devido ao facto de tanto o final do túnel como o obstáculo serem eventos que tem como consequência a proximidade do robot a um objeto, teve de se encontrar uma forma de poder diferenciar estes dois eventos. A solução a que se chegou foi que como o obstáculo se encontra sobre a linha de seguimento, quando o robot roda-se 90 graus, este poderia detetar se cruzou a linha ou não. Por consequência teve de se realizar a rotação em pequenos passos de forma a ir lendo os valores dos sensores de luminosidade e determinar se estamos perante um obstáculo ou uma parede, para atingir este fim implementou-se o código presente na Figura 11.

```
TextOut(5, LCD_LINE5, "d<0");
TextOut(5, LCD_LINE8, "d<0");
TextOut(5, LCD_LINE4, "NOT DETECTED");
isObstacle = 0;
for (i = 0; i<9; i++){
        moveFwdSyncAC(40, 20, 100);
        mylight.sensor_in3 = Sensor(IN_3);
        mylight.sensor_in4 = Sensor(IN_4);
        NumOut(20, LCD_LINE6, mylight.sensor_in3);
        NumOut(20, LCD_LINE7, mylight.sensor_in4);
        if ((mylight.sensor_in3 > Treshold_Line - sensorRange &&
        mylight.sensor_in3 < Treshold_Line + sensorRange) ||
        (mylight.sensor_in4 > Treshold_Line1 - sensorRange &&
         mylight.sensor_in4 < Treshold_Line1 + sensorRange)){
                if (!isObstacle)
                        isObstacle = 1;
                TextOut(5, LCD_LINE4, "IS OBSTACLE!!!");
        }
}
if (!isObstacle)
        TextOut(5, LCD_LINE4, "NOT OBSTACLE!!!");
Off(OUT_AC);
Wait(500);
```

Figura 11 - Deteção da linha de seguimento ao rodar 90º

Após esta rotação de 90 graus, chama-se a sub-rotina *obstacleDetected* em que se passa o parâmetro d que nos indica qual foi a direção em que rodamos. A sub-rotina *obstacleDetected* irá agora apenas contornar o obstáculo e retornar o controlo do robot à tarefa principal de forma a continuar a seguir a linha até ao final do túnel. A implementação desta sub-rotina pode ser vista na Figura 12.

```
//Obstacle handler
sub obstacleDetected(int d){
	TextOut(15, LCD_LINE2, "ObstacleDetected", true);
	if (d < 0){
		NumOut(60, LCD_LINE3, d);
		OnFwdSync(OUT_AC, MAX_AD_VEL, 0);
		Wait(1000);
		RotateMotorEx(OUT_AC, MAX_AD_VEL, 180, -100, true, true);
		OnFwdSync(OUT_AC, MAX_AD_VEL, 0);
		Wait(2200);
		RotateMotorEx(OUT_AC, MAX_AD_VEL, 180, -100, true, true);
		OnFwdSync(OUT_AC, MAX_AD_VEL, 0);
		Wait(500);
		RotateMotorEx(OUT_AC, MAX_AD_VEL, 50, 100, true, true);
	}
	else{
		NumOut(60, LCD_LINE3, d);
		OnFwdSync(OUT_AC, MAX_AD_VEL, 0);
		Wait(1000);
		RotateMotorEx(OUT_AC, MAX_AD_VEL, 180, 100, true, true);
		OnFwdSync(OUT_AC, MAX_AD_VEL, 0);
		Wait(2200);
		RotateMotorEx(OUT_AC, MAX_AD_VEL, 180, 100, true, true);
		OnFwdSync(OUT_AC, MAX_AD_VEL, 0);
		Wait(500);
		RotateMotorEx(OUT_AC, MAX_AD_VEL, 50, -100, true, true);
	}
	OnFwdSync(OUT_AC, MAX_AD_VEL, 0);
}
```

Figura 12 - Implementação da sub-rotina obstacleDetected

Navegação autónoma

Para que o robot fosse capaz de percorrer um labirinto do princípio ao fim decidiu-se implementar o algoritmo de seguir uma das paredes exteriores, por isso assim que o robot deteta que chegou ao fim do túnel e que ao efetuar a rotação implementada na Figura 11 não cruzou a linha, o robot vai chamar a sub-rotina *mazeSolver* que por sua vez inicia uma nova tarefa *getDistances* que mede continuamente as distancias que a sub-rotina *mazeSolver* necessita e que se pode ver implementada na Figura 13. Esta sub-rotina tem também um ciclo que continua até o robot terminar a componente de navegação autónoma do labirinto e que é responsável por processar as distâncias lidas pela tarefa *getDistances*.

```
task getDistances(){
	while (true){

		Acquire(motorBMutex);
		if (lookingLeft){
			Acquire(distanceMutex);
			mydist.d3 = SensorUS(IN_2);
			Release(distanceMutex);
			//turn sensor Left.
			RotateMotor(OUT_B, 60, -90);
			Acquire(distanceMutex);
			mydist.d1 = SensorUS(IN_2);
			Release(distanceMutex);
			//turn sensor Right
			RotateMotor(OUT_B, 60, 90);
			Release(distanceMutex);
		}
		if (lookingRigt){
			Acquire(distanceMutex);
			mydist.d3 = SensorUS(IN_2);
			Release(distanceMutex);
			//turn sensor Left.
			RotateMotor(OUT_B, 60, 90);
			Acquire(distanceMutex);
			mydist.d2 = SensorUS(IN_2);
			Release(distanceMutex);
			//turn sensor Right
			RotateMotor(OUT_B, 60, -90);
			Release(distanceMutex);
		}
		Release(motorBMutex);

	}
}
```

Figura 13 - Implementação da tarefa getDistances

Dentro do ciclo de navegação autónoma verifica-se a distância do robot em relação à parede, caso o mesmo comece a aproximar-se da parede ou pelo contrário comece a afastar-se da parede, há um fator de compensação que é alterado e que vai atuar a percentagem de viragem dos motores de forma a corrigir a trajetória do robot e mante-lo paralelo à parede do labirinto como pode ser visto na Figura 14.

```
//Compensação dos motores
if (d1 < minDist && oldD1 != d1){
        if (compensan < 0){
                compensan = 0;
        }
        else if (compensan < 5){
                compensan = compensan + 1;
        }
        RotateMotorEx(OUT_AC, 20, 90, compensan, true, false);
        OnFwdSync(OUT_AC, 20, compensan);
}
else if (d1 > maxDist && oldD1 != d1){
        if (compensan > 0){
                compensan = 0;
        }
        else if (compensan > -5){
                compensan = compensan - 1;
        }
        RotateMotorEx(OUT_AC, 20, 90, compensan, true, false);
        OnFwdSync(OUT_AC, 20, compensan);
}
```

Figura 14 – Código da compensação da percentagem de viragem dos motores

Quando o robot deteta que a distância que tem livre à sua frente (d3) é insuficiente para prosseguir, o robot vai verificar quais as distancias que tem livres para o lado esquerdo e para o lado direito. Após adquirir estas distancias vai compara-las para se dirigir para o lado com mais espaço livre, na eventualidade de nenhum dos lados ter espaço disponível para manobrar o robot, este vai efetuar uma rotação de 180º para continuar o seu percurso junto à parede. A implementação desta funcionalidade pode ser vista na Figura 15.

Na eventualidade do robot estar a seguir uma parede que faça um angulo de 90 graus a dada altura, o nosso programa vai detectar essa situação e continuar a acompanhar a parede. O código que implementa esta funcionalidade pode ser visto na Figura 16.

```
if (d3 < NEAR){
        Off(OUT_AC);
        lookingRigt = true;
        TextOut(15, LCD_LINE2, "Acquiring Distances");
        Wait(2000);

        Acquire(distanceMutex);
        d1 = mydist.d1;
        d2 = mydist.d2;
        d3 = mydist.d3;
        d4 = mydist.d4;
        Release(distanceMutex);
        if (d2 > d1 && d2 > NEAR_TURN){
                RotateMotorEx(OUT_AC, 20, -90, 0, true, true);
                Wait(50);
                mylight.sensor_in3 = Sensor(IN_3);
                mylight.sensor_in4 = Sensor(IN_4);
                RotateMotorEx(OUT_AC, 20, -150, 100, true, true);
                orientation++;
                if (orientation > 4){
                        orientation -= 4;
                }
        }
        else if (d1 > d2 && d1 > NEAR_TURN){
                RotateMotorEx(OUT_AC, 20, 90, 0, true, true);

                mylight.sensor_in3 = Sensor(IN_3);
                mylight.sensor_in4 = Sensor(IN_4);
                Wait(50);
                RotateMotorEx(OUT_AC, 20, 150, 100, true, true);
                orientation--;
                if (orientation <= 0){
                        orientation += 4;
                }
        }
        else if (d4 > NEAR_TURN){
                //Roda para trás.
                RotateMotorEx(OUT_AC, 20, -90, 0, true, true);
                Wait(50);
                mylight.sensor_in3 = Sensor(IN_3);
                mylight.sensor_in4 = Sensor(IN_4);
                RotateMotorEx(OUT_AC, 20, -300, 100, true, true);
                orientation -= 2;
                if (orientation <= 0){
                        orientation += 4;
                }
        }
        if ((mylight.sensor_in3 > Treshold_Line - sensorRange &&
         mylight.sensor_in3 < Treshold_Line + sensorRange) &&
                (mylight.sensor_in4 > Treshold_Line1 - sensorRange &&
         mylight.sensor_in4 < Treshold_Line1 + sensorRange)){
                end();
                break;

        }
        lookingRigt = false;

}
```

Figura 15 - Deteção de distância insuficiente para prosseguir

Finalmente caso o robot detete no chão uma linha preta transversal vai chamar a sub-rotina *end* que realiza as ações de término do labirinto, esta situação pode ser vista na Figura 17.

```
if (d1 > 40 && d1 < 110){

        RotateMotorEx(OUT_AC, 20, 180, 0, true, true);
        lookingRigt = true;
        Wait(2000);
        Acquire(distanceMutex);
        d1 = mydist.d1;
        d2 = mydist.d2;
        d3 = mydist.d3;
        Release(distanceMutex);
        RotateMotorEx(OUT_AC, 20, 150, -100, true, true);
        Wait(50);
        RotateMotorEx(OUT_AC, 20, 360, 0, true, true);
        TextOut(15, LCD_LINE2, "Checking Wall...");
        Wait(1000);
        Acquire(distanceMutex);
        d1 = mydist.d1;
        d2 = mydist.d2;
        d3 = mydist.d3;
        Release(distanceMutex);
        if (d1 > 40){
                RotateMotorEx(OUT_AC, 20, 150, -100, true, true);
                Wait(50);
                RotateMotorEx(OUT_AC, 20, 360, 0, true, true);
                orientation -= 2;
                if (orientation <= 0){
                        orientation += 4;
                }
        }
        else{
                orientation--;
        }
        lookingRigt = false;
        compensan = 0;

}
```

Figura 16 - Implementação do contorno de parede

```
if ((mylight.sensor_in3 > Treshold_Line - sensorRange && mylight.sensor_in3 < Treshold_Line +
sensorRange) &&
        (mylight.sensor_in4 > Treshold_Line1 - sensorRange && mylight.sensor_in4 <
Treshold_Line1 + sensorRange)){
        end();
        break;

}
```

Figura 17 - Condição de término do labirinto

Conclusão do labirinto

Ao ser detetada a condição de conclusão do labirinto, o robot vai calcular o tempo que demorou a percorrer o mesmo subtraindo o tempo atual ao valor do tempo que foi guardado quando o robot começou o percurso do labirinto.

Após este cálculo, o robot vai exibir no ecrã por tipo o número de vítimas encontradas bem como o tempo total além de emitir um som indicando a conclusão.

A implementação desta funcionalidade pode ser vista na Figura 18.

```
sub end(){
	Off(OUT_AC);
	ClearScreen();
	TextOut(15, LCD_LINE1, "Line Victims:");
	NumOut(40, LCD_LINE2, victimCount);
	TextOut(15, LCD_LINE3, "Temperature Victims:");
	NumOut(40, LCD_LINE4, 0);
	TextOut(15, LCD_LINE5, "Total Time in seconds:");
	mytimer.startTime = CurrentTick() - mytimer.startTime;
	long seconds = mytimer.startTime / 1000;
	NumOut(40, LCD_LINE6, mytimer.startTime);

}
```

Figura 18 - Ações finais do robot

Conclusões

Uma das principais razões pelas quais o grupo gostou da elaboração deste trabalho foi o facto de este permitir obter alguma experiência na área de Robótica Móvel.

Porém, algumas desvantagens fizeram parte da elaboração deste trabalho na construção do robot e implementação do código no mesmo. Um dos motivos pelos quais existe um grande desequilíbrio do robot ao realizar os percursos pretendidos deve-se ao facto de os motores não se conseguirem sincronizar apesar da utilização da função RotateMotorEx('ports', 'speed', 'degrees', 'turnpct', 'sync', 'stop'). Outro grande problema que precisa de ser compensado na implementação do código é o desvio de direção provocado pelas rodas utilizadas. Os sensores de luminosidade também não têm uma grande precisão, o que faz com que o treshold calibrado inicialmente possa não ser de grande referência para a execução do código. A limitação do material e a construção do robot levaram o grupo a perder algumas horas que poderiam ter sido dedicadas a melhorar o algoritmo implementado ou até mesmo na realização do presente relatório.

Apesar de todos os contratempos, o grupo reconhece que foi uma excelente oportunidade poder realizar um projeto enquadrado na Robótica Móvel. Sendo também uma mais-valia o facto de ter sido permitida uma grande liberdade a nível não só da construção do robot, como também do próprio algoritmo implementado.

Bibliografia

1. **Kapusta, Krzystof.** US Labyrinth Solver. *LegoRobot.pl.* [Online] 15 de Fevereiro de 2011. http://legorobot.pl/project.php?pr=USLabyrinthSolver&lg=en.

2. **Nüchter, Andreas.** *3D Robotic Mapping: The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom.* s.l. : Springer-Verlab Berlin Heidelberg, 2009. 978-3-540-89884-9.

3. **Oliveira, Gerardo, et al.** *Environment Mapping using the Lego Mindstorms NXT and leJOS NXJ.* FEUP - Faculdade de Engenharia da Universidade do Porto, Portugal. Porto : s.n.