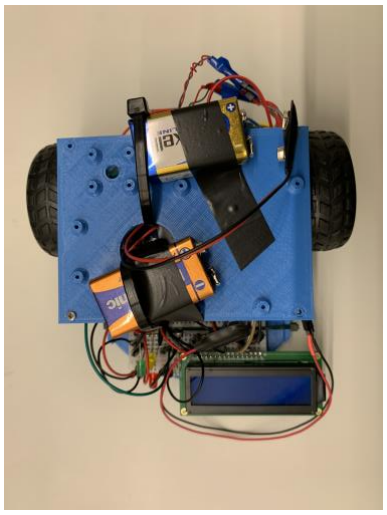
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Relatório do LineBot

ROBOT SEGUIDOR DE LINHA, PROJETO 17

Guilherme Pinheiro (up201703867) | Tomás Araújo (up201704738) | SBMI | 3MIEEC_A3 | I010 | B02 | Janeiro, 2019



Índice

RESUMO **2**

INTRODUÇÃO **2**

DESENVOLVIMENTO DO PROJETO **2**

LCD - LIQUID CRYTAL DISPLAY 3
Inicialização 3
Impressão 3
DESMULTIPLEXADOR 4
PWM – PULSE WIDTH MODULATION 5
Funcionamento 5
Inicialização do PWM 5
Duty-Cycle do PWM 6
ADC - ANALOG DIGITAL CONVERSION 6
Inicialização 6
Conversão 6
TRACKER SENSOR 6
O que é? 6
Princípio do seguidor de linha 7
Inicialização 7
Calibração 7
Processo de normalização do vetor 7
Processo de detecção da posição da linha 8
Algoritmo para o PID 8
Controlar o Robot com base na linha 8
DRIVER DOS MOTORES 9
EEPROM 9
CONTROLO REMOTO 9
No que consiste? 9
Material Usado 9
Procedimento 9
Código 10
FUNCIONALIDADES NÃO REALIZADAS 10

CONSTRUÇÃO FINAL DO ROBOT LINEBOT **11**

PROCEDIMENTO **11**

CONCLUSÃO **11**

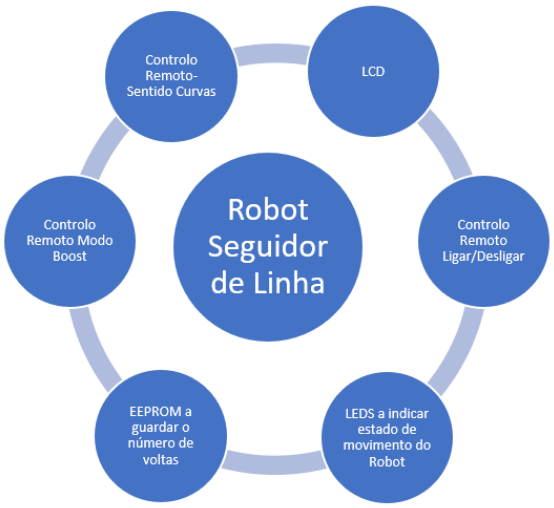
REFERÊNCIAS **12**

Resumo

O nome original do robô é LineBot.

Vamos dar início a este relatório apresentando de uma forma geral o nosso robot e as suas funcionalidades. Posteriormente, nos capítulos seguintes, uma explicação mais detalhada será dada.

Em primeiro lugar, o nosso robot, como se pode ver nas imagens, é composto por um andar extra. Este andar foi colocado por razões especiais e transforma o simples design inicial num "monster" design. Além disto, o nosso projeto não se trata apenas de um seguidor de linha. Em termos de funcionalidades mais básicas, compõe LEDs a indicar o estado do robot (A Calibrar, Andamento ou Parado) e EEPROM a guardar o número de voltas. No que toca a funcionalidades mais complexas temos um LCD e controlo remoto. Este último permite controlar: o sentido das curvas escolhido, o movimento podendo parar ou ativar o robot e por último, permite acionar o modo “fast". Este modo foi criado para aumentar a velocidade do robot.



Introdução

O projeto final, da unidade curricular Sistemas Baseados em Microprocessadores, escolhido foi o projeto 17, o Robot Seguidor de Linha.

Escolhemos projetar este robot, visto que, desde o momento em que foram apresentados os possíveis projetos da unidade curricular que nos intrigava a ideia de criar o nosso próprio robot. Escolhemos o seguidor de linha visto que foi aquele que nos despertou mais interesse, uma vez que vemos recorrentemente ser criado para as mais diversas funcionalidades.

Desenvolvimento do Projeto

Devido ao seguimento do código criado por nós, iremos seguir a ordem dele, para uma mais fácil apresentação das funções do código.

LCD - LIQUID CRYTAL DISPLAY

O LCD usado tem 16 colunas e 2 linhas, com uma luz de fundo azul e letras com cor branca. O display é de 8 bits, no entanto, nós apenas usamos 4 bits.

PINOS LCD	CONEXÃO AO ARDUINO
VSS	GND
VDD	5V
RS	8
R/W	GND
E	9
DB0-DB3	NADA
DB4-DB7	10-13
LED+	5V
LED-	GND

Tabela 1 – Conexão dos pinos do LCD ao Arduíno

Para o código, foi usado uma biblioteca "criada" por nós "lcd_gpta.c". Nesta biblioteca, há uma junção de funções encontradas na internet.

Inicialização

Por questões de segurança, sempre que iniciamos o robot, limpamos o display para a informação não se sobrepor. Essa função encontra-se também na biblioteca referida.

Função: LCD_Clear()

O LCD, tal como outros componentes, necessita de ser inicializado. Essa função encontra-se na biblioteca apresentada anteriormente.

Função: LCD_Init()

Estas duas últimas funções são chamadas na função que inicializa o robot por geral, cujo nome é "init_ALL()".

Impressão

Com essas funções e a configuração dos pinos do LCD na biblioteca, criamos a função presente no código principal, cujo nome é "LCD_Print()". Esta função tem a particularidade de através da recursividade de outras funções, imprimir no LCD o que queremos.

Na primeira linha, está presente a posição que o robot se encontra, isto é: 50, quer dizer que está a ler apenas o sensor do meio; abaixo de 50, quer dizer que os sensores do meio e o da sua esquerda estão a ler a linha preta; a cima de 50, quer dizer que os sensores do meio e o da sua direita estão a ler a linha preta. Além disso, ainda na primeira linha, são apresentadas o número de voltas que o robot deu desde do início.

Na segunda linha, está presente o "modo de curvas" que o robot está configurado, isto é: "esquerda" se no comando foi selecionada a tecla 7, para ele fazer as curvas para a esquerda; "direita" se no comando foi selecionada a tecla 9, para ele fazer as curvas para a direita. Além disso, ainda tem um espaço para o "modo de velocidade" que o robot está configurado, isto

é: “N”, modo normal, anda a uma velocidade base de 50; “F”, modo *fast*, anda na sua velocidade máxima.

Função: LCD_Print()

Parâmetros: tracker_position -> posição do robot face à linha preta

n_voltas -> número de voltas dadas pelo robot

dir -> direção selecionada no comando, se esquerda ou direita

vel -> velocidade selecionada, se normal ou *fast*

DESMULTIPLEXADOR

O desmultiplexador usado foi o SN74LS139AN [1].

Devido a ser um desmultiplexador negado, tornou-se difícil a adaptação a ele. Por só termos 2 portas digitais disponíveis e querermos 3 LEDs e 1 *buzzer*, tivemos de usar um desmultiplexador de 2-4. Através de A e B, conseguimos acender o LED pretendido. O buzzer tornou-se impossível de usar, devido ao facto de o desmultiplexador ser negado, pelo que, nunca conseguiríamos ligá-lo através deste meio.

FUNCTION TABLE

INPUTS			OUTPUTS			
ENABLE	SELECT					
$\overline{G}$	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

H = high level, L = low level, X = irrelevant

Tabela 2 - Tabela retirada da datasheet [1]

Para conseguirmos acender os LEDs, necessitamos de os colocar de forma inversa ao normal, isto é, colocar o perno negativo no pino do desmultiplexador e o perno positivo ligada uma resistência (de 330 ohms) aos 5V.

De modo a escolher as portas, criamos uma função que liga e desliga as portas do desmultiplexador, com base na tabela 2. Se queremos acender a porta 0 (LED Verde), colocamos AB = 00. Se queremos acender a porta 1 (LED Amarelo), AB=01. Se queremos acender a porta 2 (LED Vermelho), AB=10. Para a porta 3, AB=11 e seria para o buzzer.

LED Verde – Indica-nos que o robô está em andamento;

LED Amarelo – Indica-nos que o robô está a calibrar os sensores;

LED Vermelho – Indica-nos que o robô está parado (quando carregamos no botão para parar ou, devido a algum erro, o robô lê branco em todos os sensores).

Função: ligaSaida()

Parâmetros: x -> porta que queremos acender

PWM – PULSE WIDTH MODULATION

PWM, em português, significa modulação de largura de pulso. Através da largura do pulso de uma onda quadrada é possível o controle de potência e de velocidade.

Funcionamento

Considerando uma onda quadrada, temos de conhecer o valor do período da onda e do *duty-cycle* pretendido (largura de pulso propriamente dita). Para o devido funcionamento do PWM, é necessário variar o valor dessa largura.

$$DutyCycle = 100 * \frac{Largura\ de\ Pulso}{Periodo} \quad (3)$$

Onde: DutyCycle: valor em %; Largura de Pulso: tempo em que o sinal está ligado; Período: tempo de ciclo da onda.

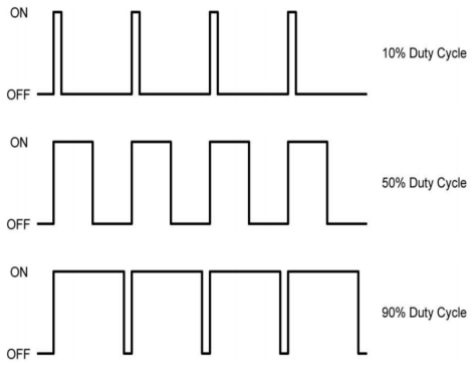


Figura 1 - Diferenças com duty-cycles diferentes

Inicialização do PWM

O registo de configuração, TCCR0A, permite configurar o modo-fast e deixar mudar o duty-cycle pretendido.

TCCR0A=0b10100011; //1010 -> Serve para mudar o duty-cycle; 0011 -> modo fast PWM

O registo de configuração, TCCR0B, define a frequência da onda quadrada do PWM. Ele aceita 7 opções de frequências:

Opção	Frequência
1	62.5 kHz
2	7.81 kHz
3	1.95 kHz
4	976.56 Hz
5	488.28 Hz
6	244.14 Hz
7	61.03 Hz

Nós optamos por escolher a opção 4 ao longo de tudo o trabalho.

Os outros registos de configuração apenas são medidas de segurança importantes.

Função: set_PWM()

Duty-Cycle do PWM

Sempre que queremos alterar o valor do duty-cycle, enviamos para uma função, uma percentagem. Quando maior essa percentagem, "mais rápido" será o motor.

Devido às portas PD5 e PD6 usadas, usamos o OCR0A e o OCR0B para metermos o valor do *duty* que queremos.

Função: set_duty()

ADC - ANALOG DIGITAL CONVERSION

Os microcontroladores foram projetos para trabalhar com informação digital, apenas percebem os valores: "0" e "1" . Neste projeto, precisamos de ler 5 valores analógicos do *tracker sensor* e outro para saber a bateria que o robot ainda tem. Para isso, o ATmega 328p vem equipado com 7 portas analógicas (PC0~PC6).

Inicialização

1. Para alimentar a parte analógica do chip AVR é necessário aplicar tensão ao AVCC, estabelecendo um nível de tensão de referência no pino AREF. Deste modo, garantimos alguma proteção contra o ruído, usando filtro de passa baixo.
2. Desativamos o buffer digital em PC0~PC4
3. No AVR, o ADC precisa estar numa frequência entre 50 e 200kHz. Portanto, precisamos de definir o *prescaller* adequado. Como o nosso AVR é cronometrado em 16MHz, vamos usar um fator de escala de 128, definindo bits ADPS0, ADPS1 e ADPS2 no registro ADCSRA. Isso dá 16000000/128=125kHz do relógio ADC. Por último, permitimos o módulo ADC, definindo bit ADEN no registro ADCSRA.

Função: init_ADC()

Conversão

Antes de tudo, para proteção, usamos ob00001111 (funciona como uma "máscara") que protege o registo ADMUX.

Para chamar a função, precisamos de selecionar o bit da Porta C que pretendemos e ela devolve-nos um valor de 16 bits.

Para realizar a conversão no bit escolhido, definimos o bit ADSC no registo ADCSRA. Este bit permanece *HIGH* até que a conversão esteja completa. No fim da conversão, devolvemos o valor ADC obtido.

Função: read_ADC()

Parâmetros: ADC_channel -> canal que queremos ler o seu valor ADC

TRACKER SENSOR

O que é?

O *Tracker Sensor* tem 5 sensores IR de alta-precisão que combinado com um algoritmo de PID, controla de forma mais estável o robot. [2]

Figura 2 - Tracker Sensor

Princípio do seguidor de linha

O sensor tem 5 saídas analógicas que leem a cor da pista. A cor com maior reflexão infravermelha é o branco, apresenta um sinal de saída maior. Quando o sensor está sobre a linha preta, o valor de saída do sensor será relativamente baixo. Cada sensor IR tem uma funcionalidade diferente, informando o robô que tem de:

- IR1 e IR5 – incrementar o número de voltas;
- IR2 –virar para a esquerda;
- IR3 – continuar em linha reta;
- IR4 – virar para a direita;

Inicialização

No ATmega 328p, iniciamos as portas IR1~IR5 como inputs.

No circuito, IR1~IR5 correspondem às entradas da porta C, C0~C4, respetivamente.

Função: init_TrackerSensor()

Calibração

Sempre que se inicia o robô, é necessário calibrar os 5 sensores para uma melhor leitura. Isto é, "andar com o robô para um lado e para o outro".

A calibração serve para colocar os valores mínimos e máximos em cada sensor. Deste modo, podemos normalizar o vetor.

Função: Calibracao()

Processo de normalização do vetor

Os 5 sensores produzem valores analógicos diferentes para a mesma cor. Além disso, o ambiente em que o robô está inserido também afeta a gama de saída analógica (por exemplo, se a sala está mais escura ou mais clara).

O processo de normalização é importante para reduzir os fatores que afetam diferentes sensores e ambientes. Com base nos valores obtidos de Máximo e Mínimo no processo de calibração, fazemos uma transformação linear:

y = vetor_normalizado[i]
x = ler_sensor[i]

$$y = \frac{(x - Min) * 1000}{Max - Min} \quad \textbf{(1)}$$

Min = minino[i]
Max = maximo[i]

Depois de transformado, o valor de saída irá variar entre 0 e 1000. Pela prática, sabemos que a cor branca > 900 e a cor preta <100.

Função: LerSensor()

Processo de detecção da posição da linha

No fim do processo de normalização, vamos pegar nos valores normalizados para criar a posição da linha. Para isso, apenas precisamos dos valores dos sensores IR2, IR3 e IR4.

y = posicao
value = vetor_normalizado[i]

$$y = \frac{(0 * value1 + 1000 * value2 + 2000 * value3)}{value1 + value2 + value3} \quad (2)$$

Função: PosicaoSensor()

Algoritmo para o PID

A posição obtida anteriormente, é necessária a dividir por 20, para nos ajudar a realizar um melhor algoritmo de PID. Temos que ter sempre em atenção que a linha preta está por baixo do carro. Só dessa forma, o robô poderá seguir a linha sem qualquer percalço. Para isso, o valor da posição deve ser mantido nos 50.

PID dá feedback e regula o erro com base em 3 fatores: P – proporcional; I – integral; D – derivativo. Tivemos por base este algoritmo:

```
proporcional = posicao - 50;
derivada = proporcional - antigo_proporcional;
integral += proporcional;
antigo_proporcional = proporcional;
return (proporcional*KP + derivada*KD + integral*KI);
```

No nosso caso, apenas usamos PD, uma vez que a integral nos daria mais problemas a controlar, portanto KI=0. O KP e o KD variam com base na pista, é preciso sempre os calibrar manualmente.

Função: PID()

Controlar o Robot com base na linha

Pegando em todos os processos anteriores, conseguimos ir ler lendo a linha de forma estável.

O PID() dá-nos o erro da distância á linha. Com base nesse erro, controlamos a velocidade do robot.

Para o motor A, fazemos: $velocidade - erro$. Para o motor B, fazemos: $velocidade + erro$. Esse valor resultante, colocamos no PWM de cada motor, como justificado anteriormente, na secção PWM.

A variável *velocidade* tem dentro o valor ou de 50 ou de 100, dependendo se o robot se encontra no modo “Normal” ou no modo “Fast”.

Nesta parte, vemos ainda quando os sensores IR1 e IR5 detetam a sua linha preta em que incrementam o número de voltas.

Função: LerLinha()

DRIVER DOS MOTORES

O driver usado neste projeto foi o Adafruit TB6612FNG.

Os pins do driver foram todos usados, à exceção do StandBy.

- VM: voltagem para os motores através de uma bateria;

- VCC: voltagem para níveis lógicos, conectado aos 5V do microcontrolador Arduino;

- GND: ligamos sempre tudo ao mesmo *ground;*

- INA1, INA2, INB1, INB2: entradas dos motores, definimos como outputs no código do ATMega, para podermos meter os motores a andar para trás, ou para a frente, ou estarem parados;

- PWMA, PWMB: PWM para cada motor;

- MOTORA: tem 2 outputs controlados pela INA1, INA2 e PWMA;

- MOTORB: tem 2 outputs controlados pela INB1, INB2 e PWMB.

EEPROM

Neste caso específico, a EEPROM [3] apenas foi usada para armazenar o número total de voltas feitas pelo robot.

CONTROLO REMOTO

No que consiste?

O controlo remoto consiste em conseguir controlar o robot à distância. Dar portanto ordens sem a necessidade de o ligar ao computador.

Material Usado

Para esta funcionalidade utilizamos apenas dois componentes: o TSOP 2236 que é um sensor de infravermelhos; e um comando de televisão que é o emissor de infravermelhos.

Procedimento

No que toca ao código, em primeiro, é importante referir que utilizámos uma biblioteca que encontrámos na internet [4].

Com esta biblioteca que descodificava o sinal infravermelhos, o nosso trabalho consistiu em definir 6 botões e escrever uma máquina de estados que é igual para todos os casos.

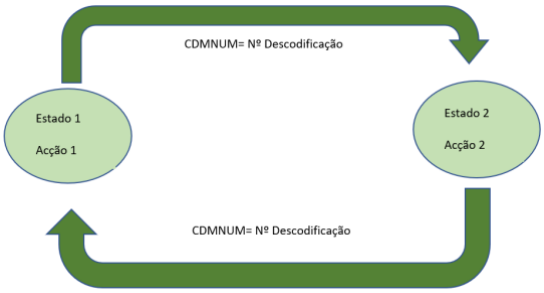
Modos:

- Movimento: um botão para ligar o robot (tecla 1) e outro para parar (tecla 3).

- Sentido das curvas: um botão para escolher o caminho do lado esquerdo numa bifurcação (tecla 7) e um botão para escolher o lado direito (tecla 9).

- Modo: um botão para ativar modo "fast" (tecla do meio grande) e um botão para desativá-lo, ou seja, voltar ao modo normal, com uma velocidade base de 50 (tecla off).

A máquina de estados é desta forma:



Este CDMNUM é o número descodificado pela biblioteca e pode ser diferente por vezes dos números dos comandos (por exemplo pode atribuir o número 1 ao número do *power on* nuns e noutros comandos pode ser a própria tecla 1).

Código

Em primeiro lugar, ver se é recebido um sinal com a função RC5_NewCommandReceived ( ). Se sim, entrar no *if statement*, e se os *starts bits != 3* , quer dizer que recebemos um sinal correto. Recebemos o sinal sob a forma de um número a partir de :

```
uint8_t cmdnum = RC5_GetCommandBits(command);
```

Com esse número, sabemos qual a tecla pressionada e, por isso, entramos no estado que queremos. Nesse específico estado, temos uma especifica ação para realizar, como já foi dito anteriormente.

FUNCIONALIDADES NÃO REALIZADAS

Para além do que desenvolvemos ao longo do projeto, o LineBot iria ter mais duas funcionalidades: um buzzer que dava se ativava quando o robot fizesse uma volta; e a informação da bateria no LCD.

No primeiro caso, não conseguimos, devido ao desmultiplexador ser negado, como referido anteriormente. No segundo caso, não tínhamos mais portas analógicas disponíveis para a sua devida leitura.

Uma das soluções possíveis para estes dois acontecimentos, seria a junção do nosso Arduíno principal com outro, através da Serial Port (RX e TX). Devido á conciliação das outras unidades curriculares, não tivemos o tempo suficiente para desenvolver esta solução. No entanto, será algo que iremos desenvolver, assim que o semestre acabar.

O código para a bateria seria algo do género:

```
int bateria(void) {
  val = read_ADC(5);
  if(val != val_anterior){
     tensao=val*5/1024;
     val_anterior=val;
  }
  return tensao*100/5;
}
```

Construção Final do Robot LineBot

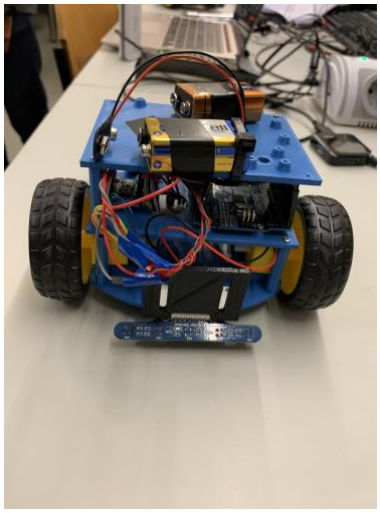


Figura 3 - Robot visto de frente

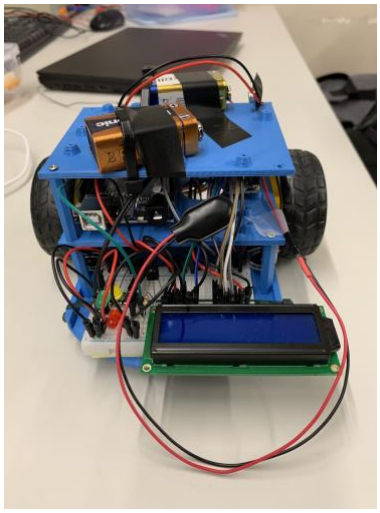


Figura 4 - Robot visto de trás

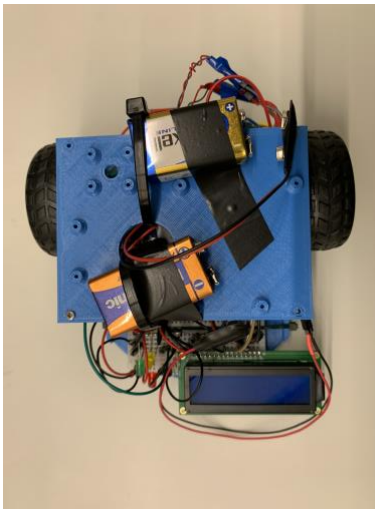


Figura 5 - Robot visto de cima

Procedimento

O procedimento para colocar o LineBot a andar em qualquer pista será:

1. Ligar as baterias;
2. Assim que ligar, realizar a calibração (indicada pelo LED Amarelo), mexendo o robô sobre a linha. Isto é andar de um lado para o outro, até aparecer uma mensagem no LCD;
3. Clicar na tecla 1 do comando;
4. Fazer o que quiser com o comando e divertir-se ao máximo!

Conclusão

Em suma, este trabalho foi na sua totalidade um desafio excecional. Aprendemos bastante, não só sobre conteúdos lecionados na unidade curricular, como também conhecimentos adicionais que tivemos de aplicar no projeto. Mesmo tendo outros projetos nas outras unidades curriculares, penso que conseguimos não só criar um robot com uma excelente performance como também um robot com algumas funcionalidades que achamos importantes num projeto deste tipo.

Referências

[1] - https://www.alldatasheet.com/datasheet-pdf/pdf/837644/TI1/SN74LS139AN.html , *datasheet* do desmultiplexador SN74LS139 NA.

[2] - https://www.waveshare.com/wiki/Tracker_Sensor

[3] – Slides ATmega328p EEPROM, disponíveis no moodle.

[4] - https://github.com/pinkeen/avr-rc5?files=1

[5] – Slides ATmega328p Interrupt System, disponíveis no moodle.