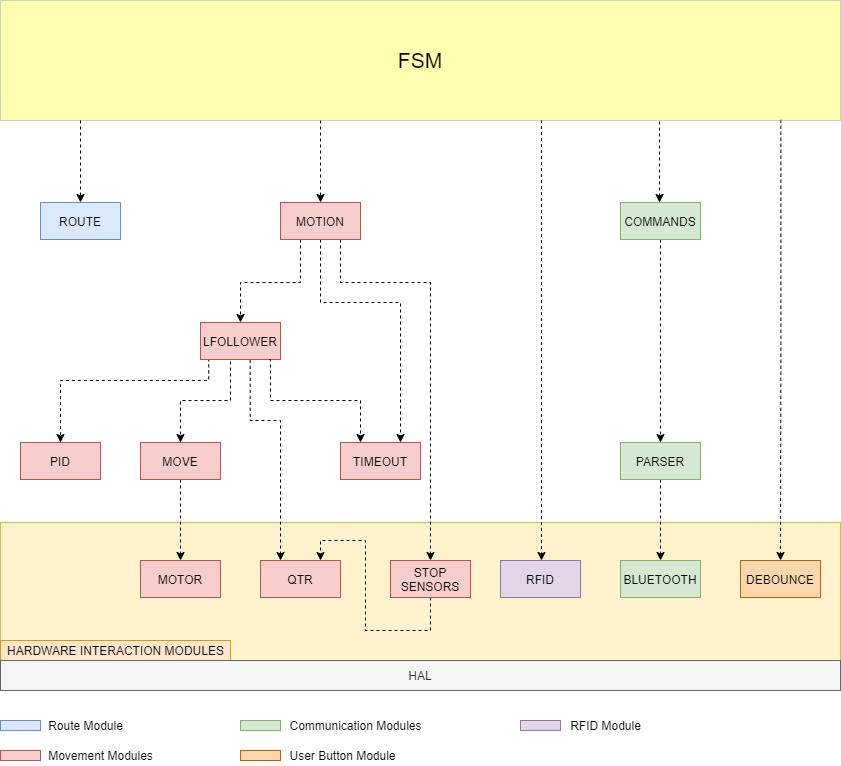
# Implementação em Software

## Descrição de *Software* e Módulos Criados

Na criação de *software* com um grau de complexidade relativamente elevado, torna-se indispensável a utilização de métodos de programação modular. Estes consistem na divisão do código em diversos ficheiros, denominados de módulos. Esta abordagem permite estruturar um programa de forma mais clara, agrupando funções e variáveis relacionadas num mesmo ficheiro. Além disso, facilita a reutilização de funções e a manutenção de código.

A fig x representa a divisão do *software* criado nos vários módulos e as suas interações, em que, FSM significa *Finite State Machine* (Máquina de estados) e HAL significa *Hardware Abstraction Layer.* De notar que as setas usadas para representar as interações entre os módulos indicam a relação de dependência, e os módulos estão agrupados por cores, tal como apresentado na legenda na figura. Por exemplo, o módulo FSM depende do módulo Motion, mas Motion não depende do módulo FSM, em que Motion pertence ao grupo de módulos que controla o movimento (*Movement Modules*).



Legenda: Divisão do *software* criado nos vários módulos.

Começando pela descrição dos módulos que controlam o movimento (representados a cor avermelhada, na figura x), na camada de interação com o *hardware* existemtrês módulos: Motor, QTR e Stop Sensors. O módulo Motor, permite o controlo de um motor através de um *timer* com canal PWM, que efetua a variação da tensão de alimentação média do motor, por modulação de largura de impulso (PWM). O sentido de rotação do motor é definido por dois pinos GPIO, IN1 e IN2, tal como apresentado na tabela y (tabela do driver) do capítulo anterior. Na fig x (fig abaixo struct) é apresentada a estrutura que define um motor.

Uma imagem com texto

Descrição gerada automaticamente

O modulo QTR inicializa o *array* de sensores do seguidor de linha [ref], possibilitando obter o valor analógico e lógico de cada um dos sensores, sendo o valor lógico alto definido a partir de 2,45 V. A leitura dos valores dos sensores do QTR é feita através do DMA. Na fig x, encontram-se representados, em forma de um enumerado, todos os sensores do *array* de sensores usadosneste projeto.

Uma imagem com texto

Descrição gerada automaticamente

O modulo Stop Sensors permite detetar as marcas de paragem (cruzes) e eventuais obstáculos que apareçam no percurso do DWR. Este modulo utiliza dois sensores do modulo QTR, SENSOR1 e SENSOR8, para fazer a deteção das marcas de paragem, e um sensor de obstáculos. Tal como no modulo QTR, a leitura dos valores do sensor de obstáculos é feita através do DMA. Definiu-se que o sensor de obstáculos sinaliza a presença de um objeto na trajetória do robô quando este se encontra a, aproximadamente, 15 cm de distância. Este modulo tem dois códigos de erro associados, E\_ST\_CROSS\_FOUND, usado quando deteta uma marca de paragem, e E\_ST\_OBS\_FOUND, usado quando deteta um obstáculo.

O modulo Move controla o movimento dos motores definindo as suas velocidades e sentidos de rotação. Além disso, este modulo inicializa os dois motores a serem usados, motor direito e motor esquerdo.

O modulo PID implementa o algoritmo do controlador PID, tal como apresentado no cap x. Na fig x, é apresentada a estrutura que define as variáveis utilizadas no algoritmo PID.

Uma imagem com texto

Descrição gerada automaticamente

O modulo Timeout permite gerar quatro *timeouts,* com duração, em segundos, representada na fig x. Quando um *timeout* termina, a *flag* respetiva será ativa de forma a sinalizar o sucedido. Para isso, este modulo utiliza um *timer* que gera uma interrupção a cada 1 segundo.

Uma imagem com texto

Descrição gerada automaticamente

O modulo Lfollower implementa o seguidor de linha, através dos sensores do QTR já mencionados, aplicando o algoritmo PID, utilizando o modulo Move para provocar uma alteração na velocidade de rotação dos motores. Além disso, este modulo implementa uma função que permite rodar o robô numa direção, direita ou esquerda, até que o sensor do QTR no lado correspondente à direção de rotação, SENSOR1 e SENSOR8, respetivamente, detetem novamente a linha. Caso nenhum sensor detete a linha durante o movimento de rotação, este será parado ao fim de um *timeout* predefinido, ROTATE\_TIMEOUT apresentado na fig x. Este modulo tem dois códigos de erro associados, E\_LF\_OFF, usado quando se tenta utilizar o seguidor de linha antes de o inicializar, e E\_LF\_NO\_LINE, usado quando o seguidor de linha não encontra uma linha para seguir. Quando ROTATE\_TIMEOUT termina, é utilizado o código de erro E\_TIMEOUT.

O modulo Motion controla o movimento do robô utilizando o seguidor de linha, os sensores de paragem e o sensor de obstáculos. Na fig x, está representado um enumerado com os possíveis estados de movimento. Sempre que algum dos erros apresentados anteriormente acontece, é efetuada a mudança do estado do movimento para o estado respetivo.

Uma imagem com texto

Descrição gerada automaticamente

Este modulo utiliza um *timer* para provocar uma interrupção a cada 10 ms, de forma a aplicar o seguidor de linha e verificar os sensores de paragem. Quando um obstáculo é detetado, o estado do movimento passa a ser MOT\_HOLD, iniciando-se a contagem de um *timeout* com duração HOLD\_TIMEOUT, tal como apresentado na fig x. Quando este timout acaba, o estado do movimento passa para MOT\_TIMEOUT. Quando não é detetada nenhuma linha, o estado do movimento passa para MOT\_ERR.

O modulo RFID permite ler um cartão RFID, obtendo-se um CardID unívoco, a sua representação em *string* e o seu tipo, tal como apresentado na fig x. O enumerado da fig x representa o estado do leitor RFID. Quando a leitura é sucedida, o seu estado será MI\_OK. Se houver um erro na leitura ou passar demasiado tempo após o início da leitura, o estado do leitor RFID será MI\_ERR ou MI\_TIMEOUT, respetivamente.

|  |  |
| --- | --- |
| Uma imagem com texto  Descrição gerada automaticamente | Uma imagem com texto  Descrição gerada automaticamente |
| **(a)** | **(b)** |

Relativamente aos módulos da comunicação, o modulo Bluetooth é responsável por receber e executar uma trama via UART, que está conectada a um dispositivo Bluetooth. Na fig x, está representado, em enumerado, o estado do Bluetooth. Quando uma trama for recebida com sucesso o estado será BLUET\_OK. Se uma trama estiver a ser recebida, o estado será BLUET\_RECEIVING, enquanto quando o estiver pronto para receber, o estado será BLUET\_READY.

Uma imagem com texto

Descrição gerada automaticamente

O modulo Parser permite analisar uma trama recebida por comunicação, dividindo-a em diferentes *arrays* a partir de um delimitador. Além disso, se a trama for um comando válido, este modulo executa a função relativa a este comando. Na fig x está representada a estrutura que define um comando, sendo composta pela *string* que define o comando, uma *string* com um texto de ajuda para o comando e a função que executa o comando.

Uma imagem com texto

Descrição gerada automaticamente

O modulo Commands define a lista de comandos válidos para esta aplicação