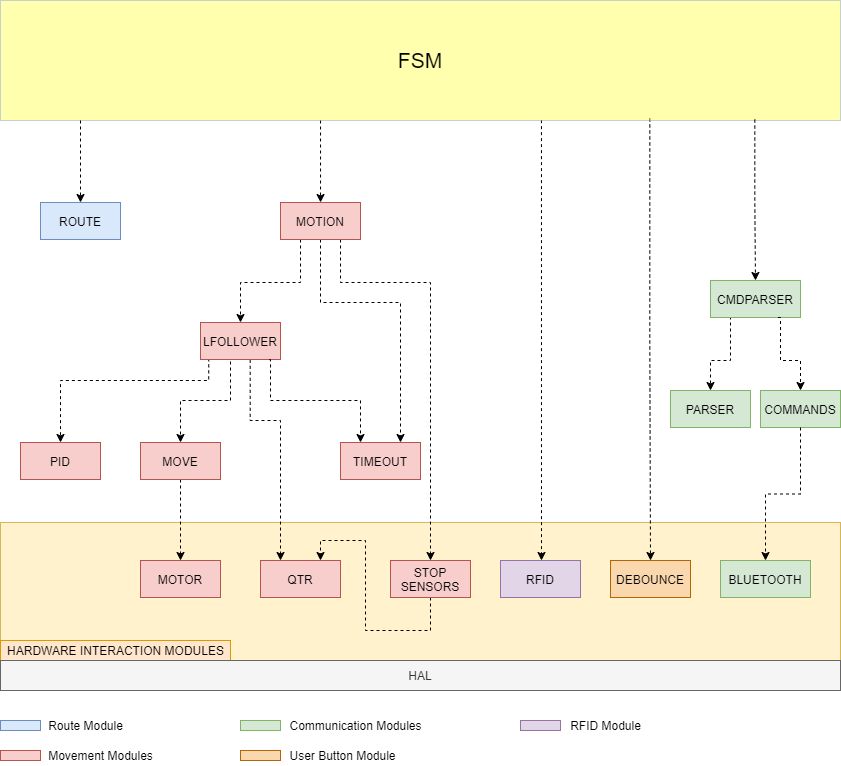
**Implementação em Software - 1.1 Descrição de Software e Módulos Criados**

Na criação de *software* torna-se indispensável a utilização de métodos de programação modular que 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, tal como apresentado na legenda na figura, os módulos estão agrupados por cores e as setas são usadas para representar as interações entre os módulos indicando a relação de dependência. 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 controlam 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

Legenda: Estrutura que define um motor.

O módulo QTR inicializa o *array* de sensores do seguidor de linha [ref], possibilitando a obtenção do valor analógico e do valor 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 usados.

Uma imagem com texto

Descrição gerada automaticamente

Legenda: Enumerado que representa os sensores do QTR utilizados.

O módulo Stop Sensors permite detetar as marcas de paragem (cruzes) e eventuais obstáculos que apareçam no percurso do DWR. Este módulo utiliza dois sensores do módulo QTR para fazer a deteção das marcas de paragem, SENSOR1 e SENSOR8, e um sensor de obstáculos. Tal como no módulo 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, correspondendo a um valor digital de 0x2000. Este módulo 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 módulo Move controla o movimento dos motores, definindo as suas velocidades e sentidos de rotação. Além disso, este módulo inicializa os dois motores a serem usados: motor direito e motor esquerdo.

O módulo 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 para realizar o algoritmo PID.

Uma imagem com texto

Descrição gerada automaticamente

Legenda: Estrutura que agrupa as variáveis de cálculo do algoritmo PID.

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

|  |  |
| --- | --- |
| Uma imagem com texto  Descrição gerada automaticamente | |
|  |  |

Legenda: Duração dos vários *timeouts*, em segundos, e definição das *flags* respetivas.

O módulo Lfollower implementa o seguidor de linha, através dos sensores do QTR já mencionados, aplicando o algoritmo PID, utilizando o módulo Move para provocar uma alteração na velocidade de rotação dos motores. Além disso, este módulo 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 módulo 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 módulo 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

Legenda: Enumerado com os possíveis estados de movimento.

Este módulo utiliza um *timer* para provocar uma interrupção a cada 10 milissegundos, 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 *timeout* acaba, o estado do movimento passa para MOT\_TIMEOUT. Quando não é detetada nenhuma linha durante o movimento, o estado do movimento passa para MOT\_ERR.

O módulo RFID permite ler um cartão RFID, obtendo-se um CardID, 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 bem-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)** |

Legenda: Módulo RFID: a) Estrutura que define um cartão RFID; b) Estado do leitor RFID.

Relativamente aos módulos da comunicação, o módulo 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 módulo 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

Legenda: Estado do módulo Bluetooth.

O módulo Commands define a lista de comandos válidos para esta aplicação. 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 o executa (*callback*).

|  |  |
| --- | --- |
|  |  |
| Uma imagem com texto  Descrição gerada automaticamente  Legenda: Definição de um *callback* de um comando e da estrutura que define um comando. | |

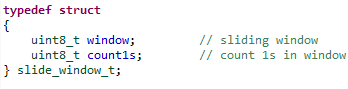
O módulo Parser permite analisar uma trama, dividindo-a em diferentes *strings* a partir de um delimitador. Além disso, se a trama for um comando válido, este módulo executa a função relativa a este comando. O módulo Cmd Parser permite interpretar um comando, definido no módulo Commands, através do módulo Parser, e executá-lo.

O módulo Route permite criar uma rota com vários pontos de paragem (*checkpoints*). Assim, na fig x, está representada a estrutura que define um ponto de paragem numa rota, contendo uma *string* com o identificador do *checkpoint* (RFID) e a ação a executar nesse ponto de paragem (*action*)*.*

|  |  |
| --- | --- |
|  |  |
|  | |

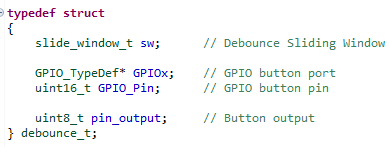
Legenda: Ações possíveis a realizar num *checkpoint,* e, definição de um *checkpoint*.

O módulo Debounce efetua o *debounce* de um botão de pressão associado a um pino GPIO, por *software*, através de um algoritmo janela deslizante, tal como mostra a figura x.



Legenda: Definição de uma janela deslizante.

Uma janela deslizante é um algoritmo que permite armazenar um conjunto de valores, neste caso, um conjunto de *bits* numajanela de 8 *bits* (*window*), e também, saber quantos *bits* a nível lógico alto existem na janela. Como definido na estrutura apresentada na fig x, o módulo Debounce utiliza uma janela deslizante (*sw)* para que, quando o número de *bits* a nível lógico alto for superior a um certo valor, menor do que o tamanho da janela, coloque o *pin*\_*output,* querepresenta o sinal resultante da aplicação do *debounce* ao pino GPIO, a nível lógico alto. Para isso, este módulo utiliza um *timer* que gera uma interrupção a cada 50 milissegundos.



Legenda: Definição da estrutura *debounce*.