

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
INSTITUTO DE INFORMÁTICA  
ESCOLA DE ENGENHARIA

*ENG10032 - Microcontroladores - 2019/2*

*Prof. Dr. Walter Fetter Lages*

**Projeto Final**

***Shield para Galileo Gen 2 - Controle do Robô Quanser 2DSFJE***

*Gustavo Madeira Santana - 00252853*

*Lúcio Pereira Franco - 00252867*

*Rodrigo Franzoi Scroferneker - 00252849*

*Porto Alegre, Rio Grande do Sul - 2019*

## Sumário

|          |                                      |           |
|----------|--------------------------------------|-----------|
| <b>1</b> | <b>Introdução</b>                    | <b>3</b>  |
| 1.1      | Intel Galileo Gen 2 . . . . .        | 3         |
| 1.2      | Quanser 2DSFJE . . . . .             | 3         |
| <b>2</b> | <b>Software API</b>                  | <b>4</b>  |
| 2.1      | Controle do Motor . . . . .          | 4         |
| 2.2      | Leitura dos Encoders . . . . .       | 5         |
| 2.3      | Sensores de Final de Curso . . . . . | 5         |
| <b>3</b> | <b>Hardware (<i>Shield</i>)</b>      | <b>5</b>  |
| 3.1      | Ponte H . . . . .                    | 6         |
| 3.2      | Decoder . . . . .                    | 6         |
| 3.3      | Sensores de Final de Curso . . . . . | 7         |
| 3.4      | <i>Shield</i> . . . . .              | 7         |
| <b>4</b> | <b>Aplicação</b>                     | <b>9</b>  |
| 4.1      | Controle Manual do Motor . . . . .   | 9         |
| 4.2      | PID no Quanser 2DSFJE . . . . .      | 10        |
| <b>5</b> | <b>Conclusão</b>                     | <b>11</b> |
| <b>6</b> | <b>Apêndice</b>                      | <b>12</b> |

### Enunciado

O objetivo do projeto é desenvolver um *shield* para a Galileo Gen 2 capaz de acionar pelo menos uma das juntas do robô Quanser 2DSFJE. O hardware deve ser capaz de acionar o motor (27V×3A)

e ler os sensores (encoder e 2 sensores de fim de curso magnético) de pelo menos uma das juntas do robô.

## 1 Introdução

O projeto final da cadeira de microcontroladores tem como objetivo desenvolver uma *shield* que se encaixa na placa Intel Galileo Gen 2 e faz a interface entre a Galileo e o robô Quanser 2DSFJE, permitindo controlar o robô através da Galileo. Para isso, o projeto é dividido em duas partes: projeto do hardware da *shield* e conectores entre a *shield* e o robô; e o desenvolvimento do software que permite controlar o robô no espaço de usuário.

As próximas subseções apresentam uma breve introdução sobre a Intel Galileo Gen 2 e o robô Quanser 2DSFJE. Na Seção 2 apresentamos a biblioteca desenvolvida, permitindo ao usuário utilizar rotinas para controlar o robô. Em seguida, apresentamos o hardware da *shield* na Seção 3. Por fim, mostramos um caso de uso do hardware e software, implementando o algoritmo de controle PID na Seção 4.

### 1.1 Intel Galileo Gen 2

A placa Intel Galileo Gen 2 é uma placa baseada no *system-on-chip* (SoC) Quark X1000, de 32-bits, fabricado pela Intel. O SoC é capaz de operar em frequências de até 400MHz. O processador Quark tem suporte a distribuição Poky Linux Yocto 1.4. A placa conta com conexões ethernet, USB, e um slot para cartões micro-SD. Dispõe de 20 pinos digitais de entrada/saída, na qual 6 podem ser usados como PWM, e 6 como entrada analógica. A placa foi criada pensando em ser compatível com o ambiente de desenvolvimento para placas Arduino, além disso possui leiaute dos pinos compatível com certos padrões de *shields* criadas para placas Arduino.

### 1.2 Quanser 2DSFJE

O robô Quanser 2DSFJE é um robô com juntas de 2-graus-de-liberdade. O robô conta com dois motores DC ( $27V \times 3A$ ) em seu braço mecânico, um para movimentar o ombro (*shoulder*), e outro para movimentar o cotovelo (*elbow*). Cada junta conta com dois sensores de final de curso que indicam o limite físico de movimentação das juntas. Os motores também contam com encoders de quadratura que permitem derivar a posição relativa de cada parte do braço em cada uma das juntas.

## 2 Software API

Para controlar o robô Quanser e ler os sensores, através da placa Galileo, uma biblioteca de funções foi desenvolvida. Isto facilita futuras implementações de rotinas e novas funcionalidades que pretendem utilizar a *shield* para controlar o robô. Nesta seção apresentamos as principais funcionalidades disponíveis em nossa biblioteca. A biblioteca inclui um *script* de inicialização a ser utilizado na Galileo que configura automaticamente os pinos de GPIO, PWM e os para a comunicação SPI. As configurações são feitas através dos pseudo-arquivos disponíveis no sistema linux utilizado na Galileo. Este *script* também altera as permissões de acesso à esses pseudo-arquivos, de forma com que a biblioteca possa ser utilizada no espaço de usuário, sem necessidade de utilizar o super usuário.

Três componentes essenciais para utilização correta e segura do robô quanser são: o controle do motor DC, leitura dos encoders, e leitura dos sensores de final de curso. O controle do motor irá definir sua direção e velocidade. Os encoders permitem ler a posição relativa do motor. Os sensores de final de curso indicam se o braço do robô Quanser atingiu uma de suas extremidades, indicando que o motor deve ser desligado ou ter sua direção alterada. Nas próximas seções explicamos como cada componente pode ser utilizando através de nossa biblioteca.

### 2.1 Controle do Motor

O controle do motor é essencialmente controlado através da tensão aplicada. A velocidade do motor é proporcional a tensão aplicada, podendo variar de  $-27V$  à  $27V$ . A escala de tensão é linear, ou seja,  $-27V$  é velocidade máxima em uma direção,  $0V$  o motor fica estacionário, e  $27V$  a velocidade máxima na direção oposta.

Para controlar o motor escolhendo a tensão aplicada, nossa biblioteca fornece a função *set\_motor\_voltage(float voltage)*. A função recebe um número em ponto flutuante, que indica a tensão a ser aplicada no motor. A função retorna 0 em caso de sucesso, e  $-1$  em caso de erro.

Outra forma de controlar a tensão aplicada no motor é através do sinal PWM que é aplicado na ponte H, que regula a tensão aplicada no motor. Nossa biblioteca disponibiliza a função *set\_pwm\_duty\_cycle\_percentage(float percentage)*. Essa função recebe um número em ponto flutuante equivalente a porcentagem desejada para o tamanho do ciclo de trabalho do sinal PWM. Para este projeto, o sinal PWM possui uma escala linear onde 0% de ciclo de trabalho implica em velocidade máxima em uma direção, 100% velocidade máxima na direção oposta, e 50% o motor fica estacionário.

## 2.2 Leitura dos Encoders

Para saber a posição do braço do robô Quanser, são disponíveis encoders que podem ser lidos na Galileo através da conexão da *shield* com o robô. Em nossa biblioteca disponibilizamos duas formas de leitura dos encoders: contagem relativa; e contagem em radianos. Para ambos tipos de leitura, utilizamos o circuito integrado LS7366, que permite leitura em quadratura de até 32 bits em hardware, sem perdas.

A contagem relativa se inicia logo que o LS7366 é inicializado. Movimentos do braço no sentido horário incrementam a contagem, enquanto movimentos no sentido anti-horário decrementam a contagem. A função que permite fazer a leitura dos encoders é `counter_read(COUNTER counter)`, onde COUNTER é uma estrutura que contém informações sobre quais pinos da Galileo estão ligados ao LS7366.

A leitura em radianos é feita usando como base a contagem relativa e convertida para radianos de acordo com as especificações do *datasheet* do robô Quanser, onde é apresentado que o encoder terá 4096 pulsos por revolução do motor, ou 1024 linhas por revolução. A função que permite a leitura em radianos é `counter_read_rad(COUNTER counter)`, onde novamente COUNTER é uma estrutura com referência aos pinos conectados ao LS7366. Dada a estrutura do robô, a leitura em radianos será referente a posição do robô na qual o decoder foi inicializado. Nossa biblioteca disponibiliza uma rotina de inicialização onde a contagem pode começar a partir do centro (1.57 rad), ou em uma das extremidades (0 rad ou 3.14 rad).

## 2.3 Sensores de Final de Curso

Os sensores de final de curso permitem saber se o braço está em um dos limites físicos do robô, indicando que o motor deve mudar de direção ou ficar estático. O robô possui dois sensores no ombro (*shoulder*), um para cada extremidade. A nossa biblioteca disponibiliza dois tipos de leitura dos sensores: leitura direta; e leitura por interrupção.

A leitura direta é feita através da função `limit_read(int switch_limit)`, onde `switch_limit` pode ser 1 ou 2, indicando qual sensor de final de curso se deseja ler. A leitura direta imediatamente retorna o valor lido do sensor. A leitura por interrupção irá fazer um *polling* (*POLLPRI*) no sensor desejado, aguardando uma interrupção que indica que o braço atingiu o limite. Isto é feito através da função `limit_poll(int switch_limit)`.

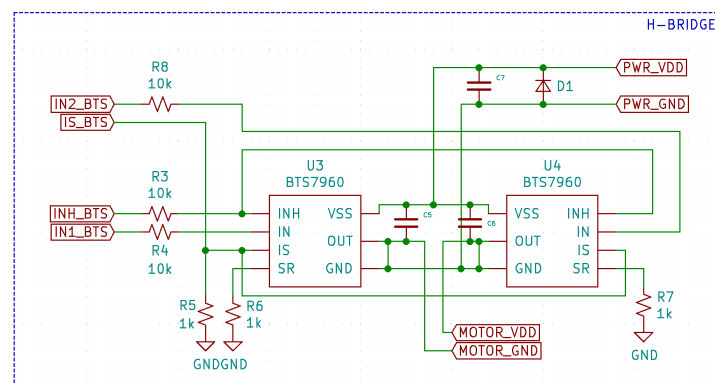
## 3 Hardware (*Shield*)

Para fazer a interface da Intel Galileo Gen 2 com o robô Quanser 2DSFJE, desenvolvemos uma *shield* que encaixa diretamente na Galileo e expõe pinos e conectores para fácil montagem com o robô Quanser.

Os dois principais componentes da *shield* são a Ponte H para controlar a tensão aplicada no motor DC do robô, e o decoder para fazer a leitura do encoder em hardware, sem perdas. Um terceiro componente é referente a leitura dos sensores de final de curso, trivialmente implementado através de dois pinos de GPIO da Galileo.

### 3.1 Ponte H

A Ponte H utiliza o circuito integrado (CI) BTS7960. Este CI é uma *half-bridge*, com suporte a correntes de até 43A. Combinando dois BTS7960, é possível construir uma ponte H completa. O esquemático da ponte H completa utilizado é o esquemático referência fornecido no *datasheet* do BTS7960 (Figura 7). Três características interessantes do BTS7960 é que ele possui proteção contra: *overvoltage*, *undervoltage*, e *overtemperature*.



**Figura 1:** Esquemático da Ponte H utilizando dois BTS7960.

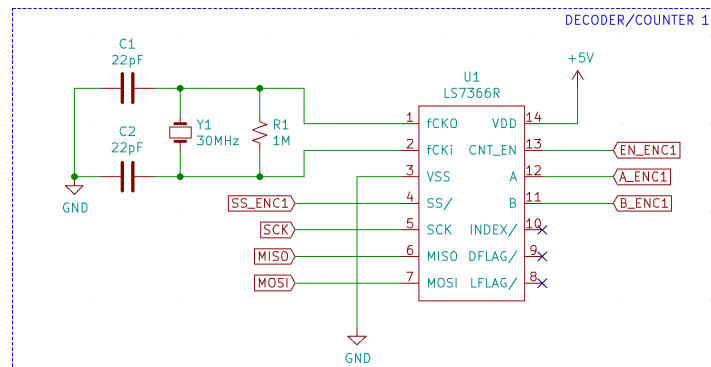
Aplicando um sinal PWM através da Galileo, a ponte H fornecerá uma tensão de saída relativa ao ciclo de trabalho utilizado.

### 3.2 Decoder

A *shield* também conta com um decoder para leitura do encoder de quadratura do robô, através do CI LS7366. Através deste CI a leitura dos encoders é feita em quadratura e em hardware, sem perdas. O LS7366 permite realizar a contagem utilizando 8, 16, 24, ou 32 bits. Inicialmente utilizamos 16 bits, mas para garantir que não ocorreria estouro da contagem, decidimos por utilizar a capacidade máxima do CI, que é 32 bits.

A comunicação da Galileo com o LS7366 é feita utilizando comunicação serial síncrona SPI, implementado utilizando portas de GPIO da Galileo. O esquemático detalhado do circuito do decoder é apresentado na Figura 2. Os valores dos capacitores foram derivados utilizando a equação fornecida no

*datasheet* do LS7366.



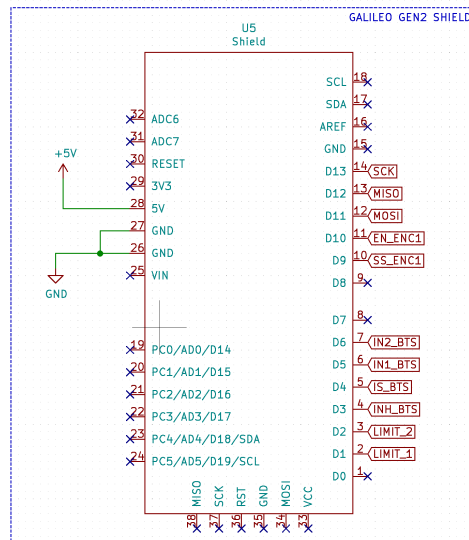
**Figura 2:** Esquemático para utilização do LS7366.

### 3.3 Sensores de Final de Curso

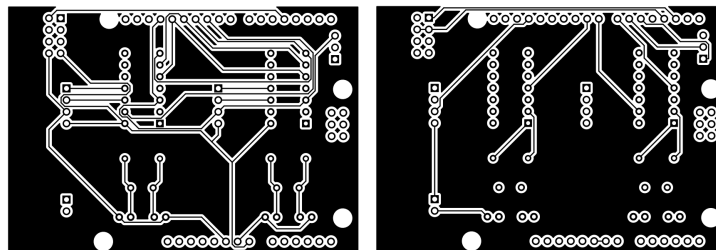
Os sensores magnéticos de final de curso são alimentados por uma fonte externa de 15V. O painel do robô Quanser fornece pinos de acesso direto aos sensores, que são conectados diretamente a dois conectores da *shield*, que por fim estão conectados a duas entradas GPIO da Galileo. Os sensores de final de curso são *low-active*, ou seja, quando o braço chega em um dos limites, o sinal é puxado para 0V.

### 3.4 *Shield*

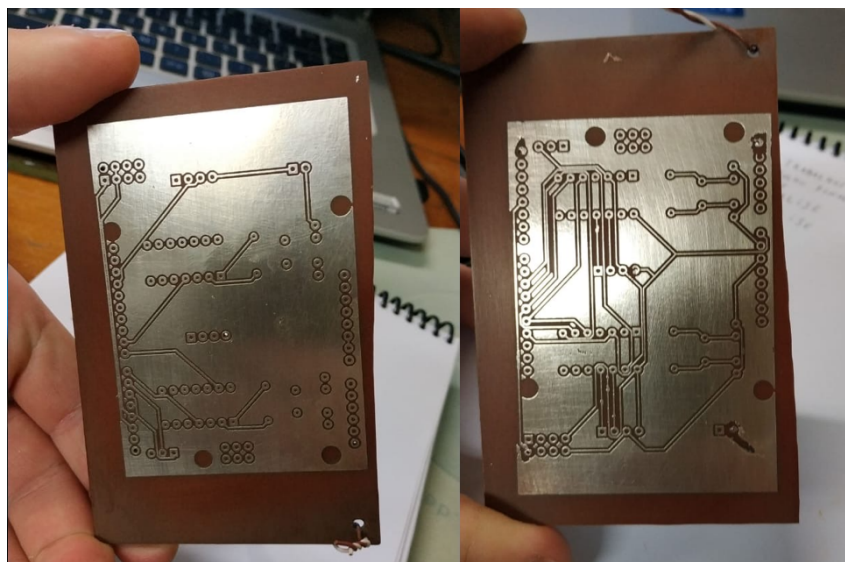
Juntando os três componentes explicados acima, desenvolvemos então uma *shield* que encaixa diretamente na Galileo. Além das conexões entre a fonte externa de 27V e do motor com a ponte H, a *shield* também fornece 5V para alimentar os encoders do robô, e um pino de terra referência para os sensores de final de curso. Apresentamos nessa seção fotos do esquemático, da placa fabricada, e da placa conectada ao robô durante a apresentação do trabalho.



**Figura 3:** Esquemático da *shield* e pinos utilizados na Galileo.

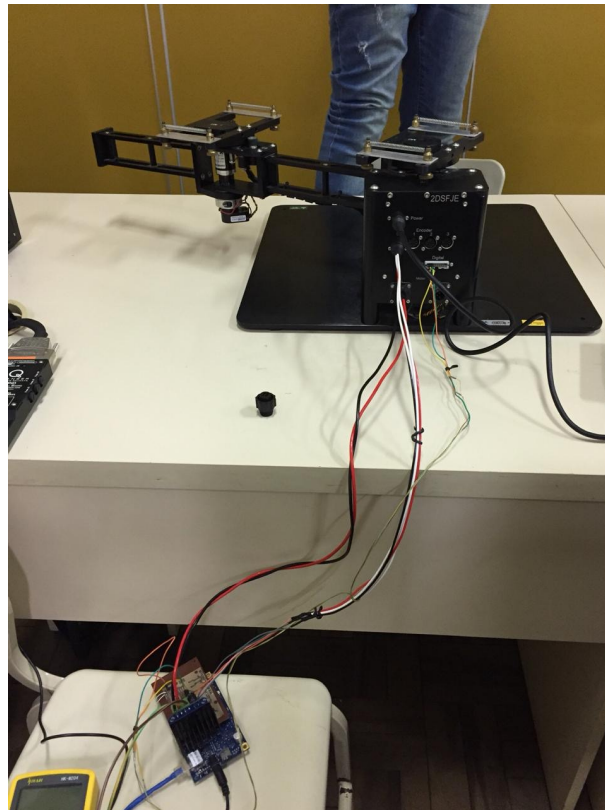


**Figura 4:** Templates utilizados para fabricar a *shield*.



**Figura 5:** Frente e verso da *shield* fabricada.





**Figura 6:** Ligações entre a *shield* e o robô Quanser utilizando cabos customizados para facilitar a conexão.

## 4 Aplicação

Nesta seção apresentamos duas rotinas básicas que demonstram a utilização da biblioteca para controle do robô Quanser 2DSFJE através da Galileo Gen 2 utilizando a *shield* desenvolvida.

### 4.1 Controle Manual do Motor

Primeiramente desenvolvemos uma rotina manual para testar as funcionalidades da biblioteca junto a *shield*, e verificar o funcionamento correto das funções desenvolvidas. A rotina consiste em movimentar o braço no sentido anti-horário até atingir o limite físico do robô, em seguida movimentar o braço até o limite no sentido horário, e depois posicionar o braço em uma série de posições em radianos, em velocidades pré-definidas. Para isso, usamos as funções disponíveis na biblioteca para movimentar o braço no sentido anti-horário usando uma tensão aplicada de  $-8V$ . Logo que colocamos a tensão no motor, iniciamos um *polling* no sensor de final de curso esquerdo. Uma vez que o braço chega no limite, a tensão aplicada é invertida para  $8V$  e se inicia um *polling* no sensor de final de curso da direita. Com o braço no limite, uma tensão de  $-10V$  é aplicada até o braço chegar na posição central de 1.5 radianos. Essa rotina pode facilmente ser estendida para que o motor gire em diferentes velocidades e para diferentes posições.

## 4.2 PID no Quanser 2DSFJE

Uma dos algoritmos de controle mais utilizados é o PID (*Proportional, Integral, Derivative*). O objetivo do PID é minimizar uma métrica de erro definida em algum sistema. Para isso, é necessário alguma informação de feedback do sistema sob qual o erro será computado. A saída do PID é a entrada para um atuador no sistema onde se deseja minimizar o erro. Cada componente do PID é multiplicado por uma constante, e portanto afeta o sistema de diferentes formas.

**Proportional** ( $k_p$ ): este termo escala a saída de acordo com o erro computado no sistema. Um valor pequeno pode causar o sistema a nunca minimizar o erro. Um valor grande pode fazer com que o sistema oscile muito e fique instável.

**Integral** ( $k_i$ ): o termo da integral funciona no sentido de acumular o erro ao longo do tempo de execução. Um valor grande pode fazer com que o algoritmo priorize corrigir erros passados, interferindo com correções ao erro atual, podendo fazer com que o sistema fique instável.

**Derivative** ( $k_d$ ): a derivada irá computar como o sistema está se comportando a cada iteração do algoritmo ao longo do tempo. Age no sentido de amortecer a resposta do sistema.

No caso do robô Quanser 2DSFJE, o erro será computado em relação a posição do braço, portanto temos como feedback a leitura do encoder. Através do encoder definimos a posição do braço e a cada iteração computamos o erro como sendo  $radianos_{objetivo} - radianos_{atual}$ . O atuador no robô é o motor, onde podemos mudar sua direção e velocidade de acordo com a tensão aplicada. A saída do PID então é a tensão a ser aplicada no motor para que o erro seja minimizado.

Aqui utilizamos como base a rotina anterior de controle manual do motor, onde o braço primeiro se move para ambos os limites, testando os sinais de final de curso. Em seguida é inicializado o algoritmo PID para a posição em radianos indicada pelo usuário. O código desta rotina está anexado junto a este relatório.

Sucintamente, a rotina é composta pelas seguintes funções e segue o roteiro aqui apresentado:

- Inicializar PWM, configurando o período e ciclo de trabalho, e aplicar  $-15V$  no motor (sentido anti-horário). Funções utilizadas:
  - `init_pwm()`;
  - `set_motor_voltage(-15)`;
- Aguardar o braço atingir o limite esquerdo, fazendo polling no sensor do final de curso. Em seguida, aplicar  $15V$  no motor e fazer polling no sensor de final de curso oposto. Funções utilizadas:
  - `limit_poll(1)`;

```
– set_motor_voltage(15);
```

```
– limit_poll(2);
```

- Inicializar o algoritmo do PID na posição indicada pelo usuário. As constantes do PID podem ser passadas via linha de comando. A saída do PID – valor de tensão – é aplicada ao motor. Funções utilizadas:

```
– pid_control(desired_value, counter_value_rad, kp, ki, kd, error_prior, update_period);
```

```
– set_motor_voltage(pid_output);
```

- Ao encerrar a execução do PID – o tempo de execução é passado via linha de comando – a rotina aplica 0V no motor, e desabilita o sinal PWM. O programa é então encerrado.

```
– set_motor_stop();
```

```
– pwm_stop();
```

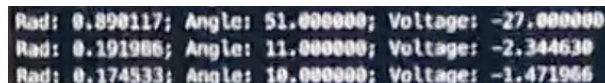
Internamente, o PID é calculado da seguinte forma:

```
error = desired_value – actual_value;
```

```
integral = integral + (error * update_period);
```

```
derivative = (error – error_prior)/update_period;
```

```
output = kp * error + ki * integral + kd * derivative;
```



```
Rad: 0.898117; Angle: 51.000000; Voltage: -27.000000
Rad: 0.191986; Angle: 11.000000; Voltage: -2.344630
Rad: 0.174533; Angle: 10.000000; Voltage: -1.471966
```

**Figura 7:** Foto da rotina do PID sendo executada na Galileo durante apresentação do projeto tendo como posição desejada 0.15 radianos. Três iterações do PID como exemplo. Os prints mostram a posição atual do braço em radianos (e em graus, para facilitar visualização), e a tensão aplicada de acordo com o algoritmo do PID. Inicialmente o PID tem uma saída alta (-27V), ou seja, velocidade máxima na direção que minimiza o erro. Conforme o braço chega próximo da posição destino, a tensão aplicada no motor diminui. O PID realiza uma iteração a cada 10ms, os prints mostram apenas uma iteração por segundo.

## 5 Conclusão

O projeto de desenvolvimento da *shield* e da biblioteca para a Galileo permitiu utilizarmos grande parte do conhecimento adquirido nos laboratórios ao longo do semestre, de forma a integrar partes de cada laboratório em uma aplicação prática. Parte valiosa do conteúdo foi aprender a configurar *hardware* através de pseudo-arquivos expostos no sistema Linux. O conceito de se controlar um atuador (motor) e ter um feedback (encoder) utilizando o algoritmo PID é algo aplicável a diversos sistemas, que vão além do robô utilizado na disciplina. Entender o funcionamento correto do PWM e da comunicação SPI também foram partes importantes do trabalho.

## 6 Apêndice

Acompanham este relatório:

- arquivo de netlist e Gerber do layout do PCB gerados pelo KiCad.
- arquivo de BOM (bill of materials) gerado pelo KiCad.
- código fonte da biblioteca desenvolvida, script de inicialização para a Galileo, e documentação gerada utilizando Doxygen.