

Laboratório de Sistemas Digitais - ES770

Relatório de Engenharia do Spark

Alunos:

Bruno de Souza Ferreira, RA: 135164
Guilherme Kairalla Kolotelo, RA: 135964

25 de novembro de 2016

1 Objetivo

O objetivo deste relatório é descrever o funcionamento do Spark, um robô seguidor de linha capaz de interpretar comandos dispostos na pista. Segue abaixo as vistas superior e inferior do Spark.

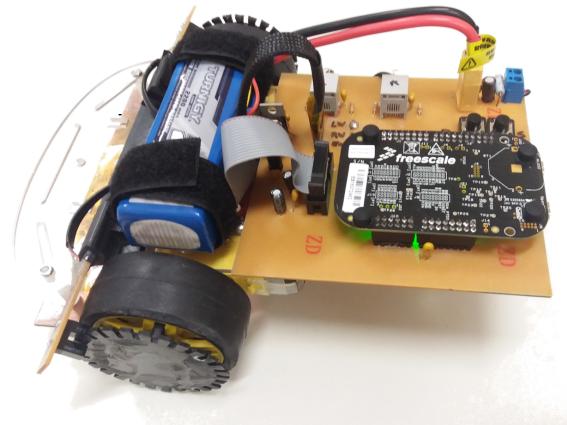


Figura 1: Vista superior do Spark

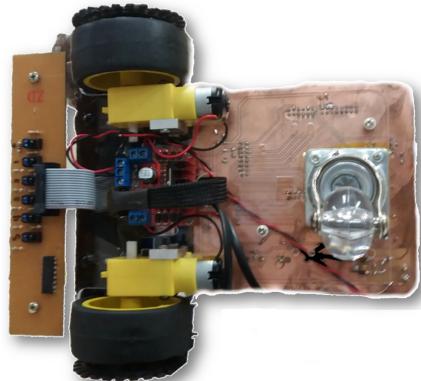


Figura 2: Vista Inferior do Spark

2 Diagramas representativos do Spark

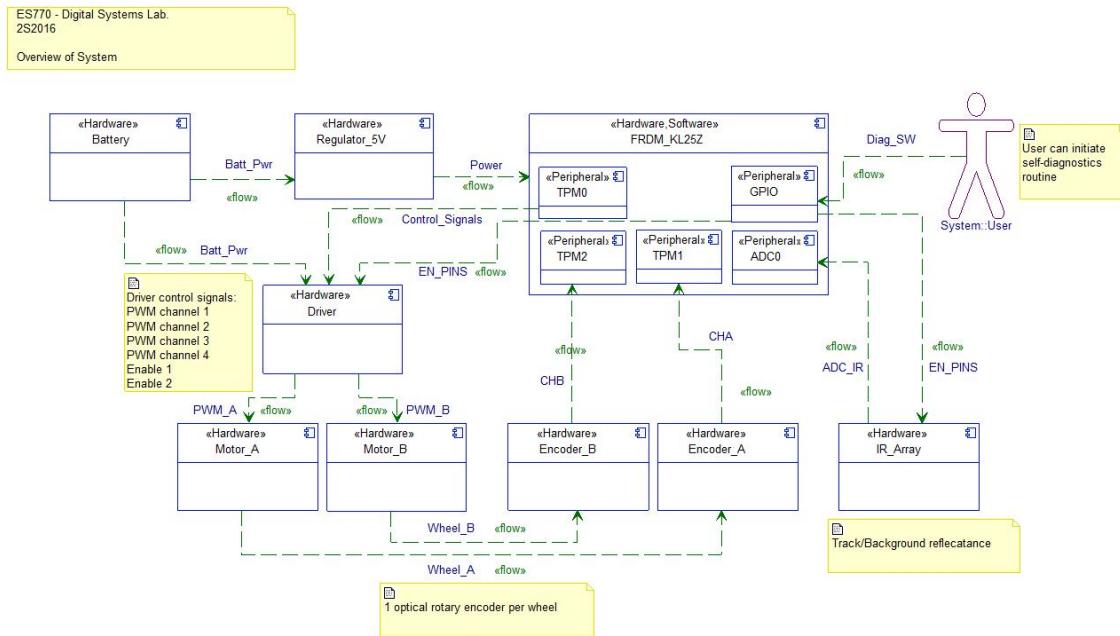


Figura 3: Diagrama de blocos do Spark

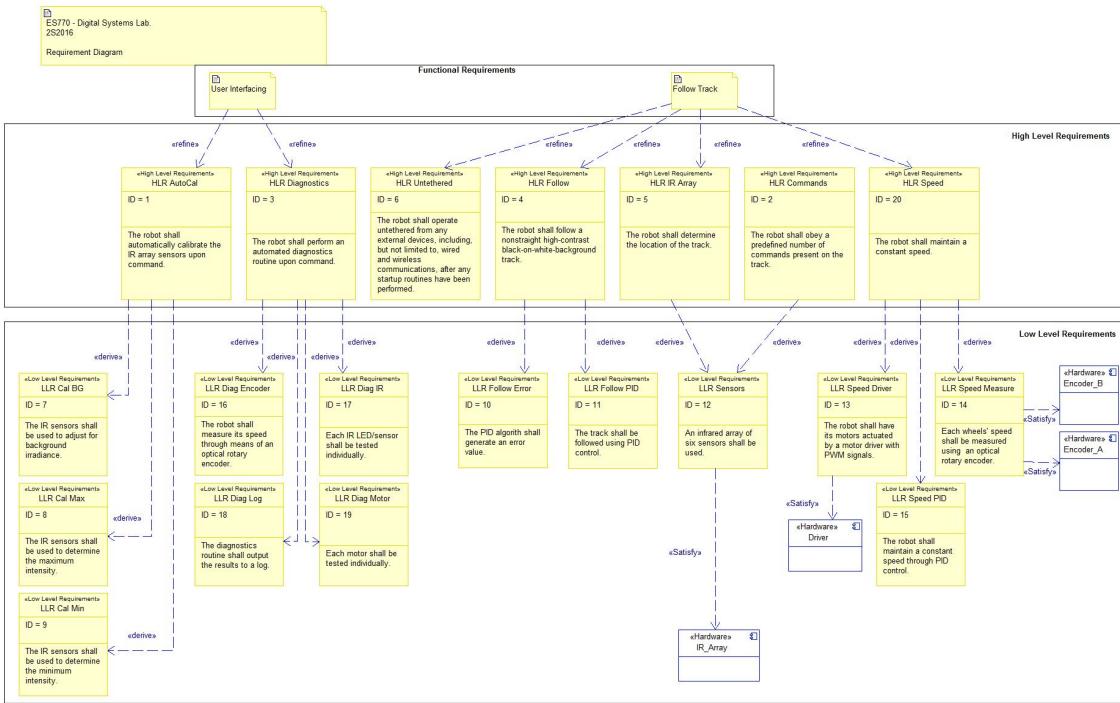


Figura 4: Diagrama de requisitos do Spark

 ES770 - Digital Systems Lab.
 2S2016
 Package Diagram

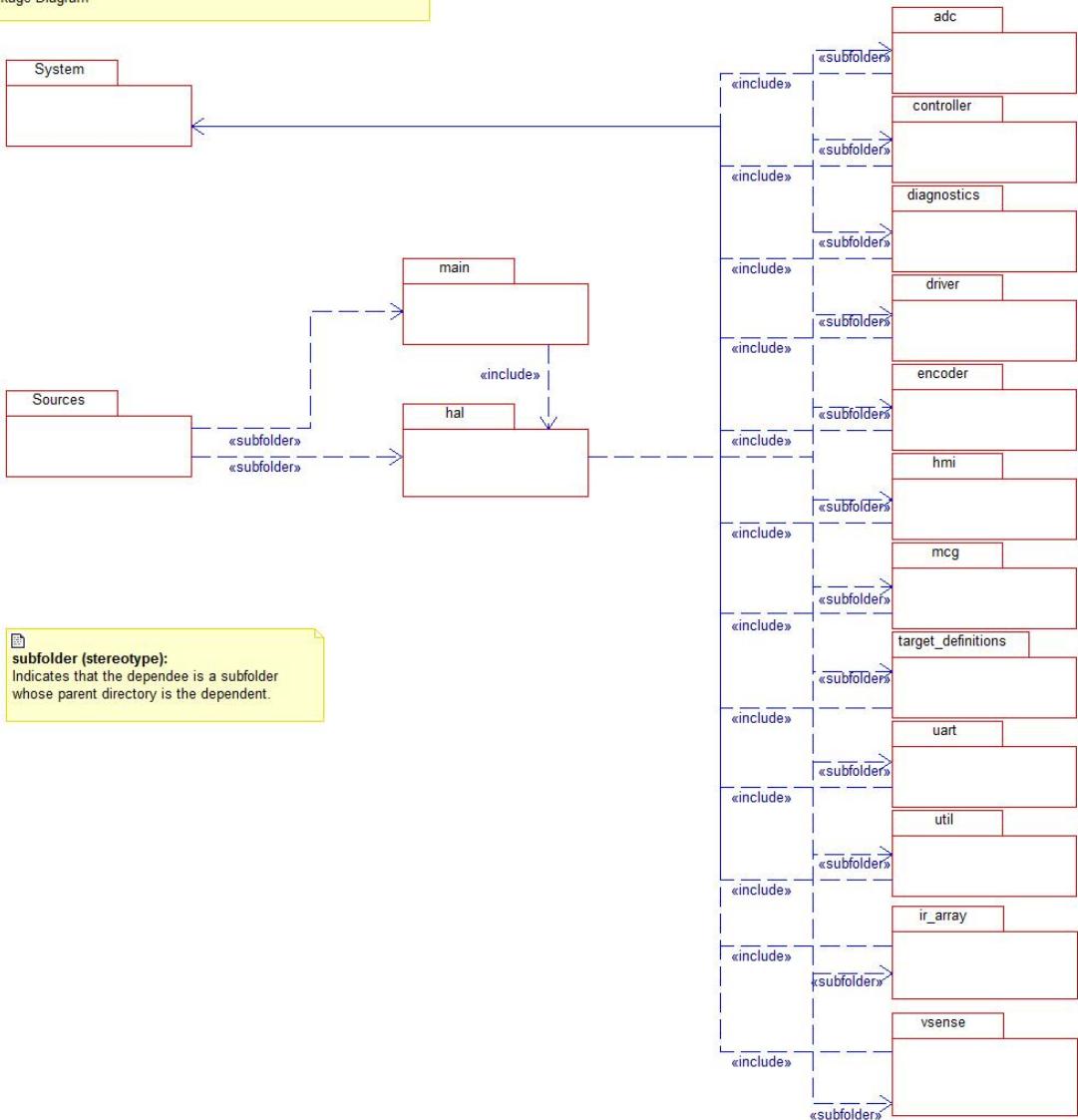


Figura 5: Diagrama de pacotes do Spark

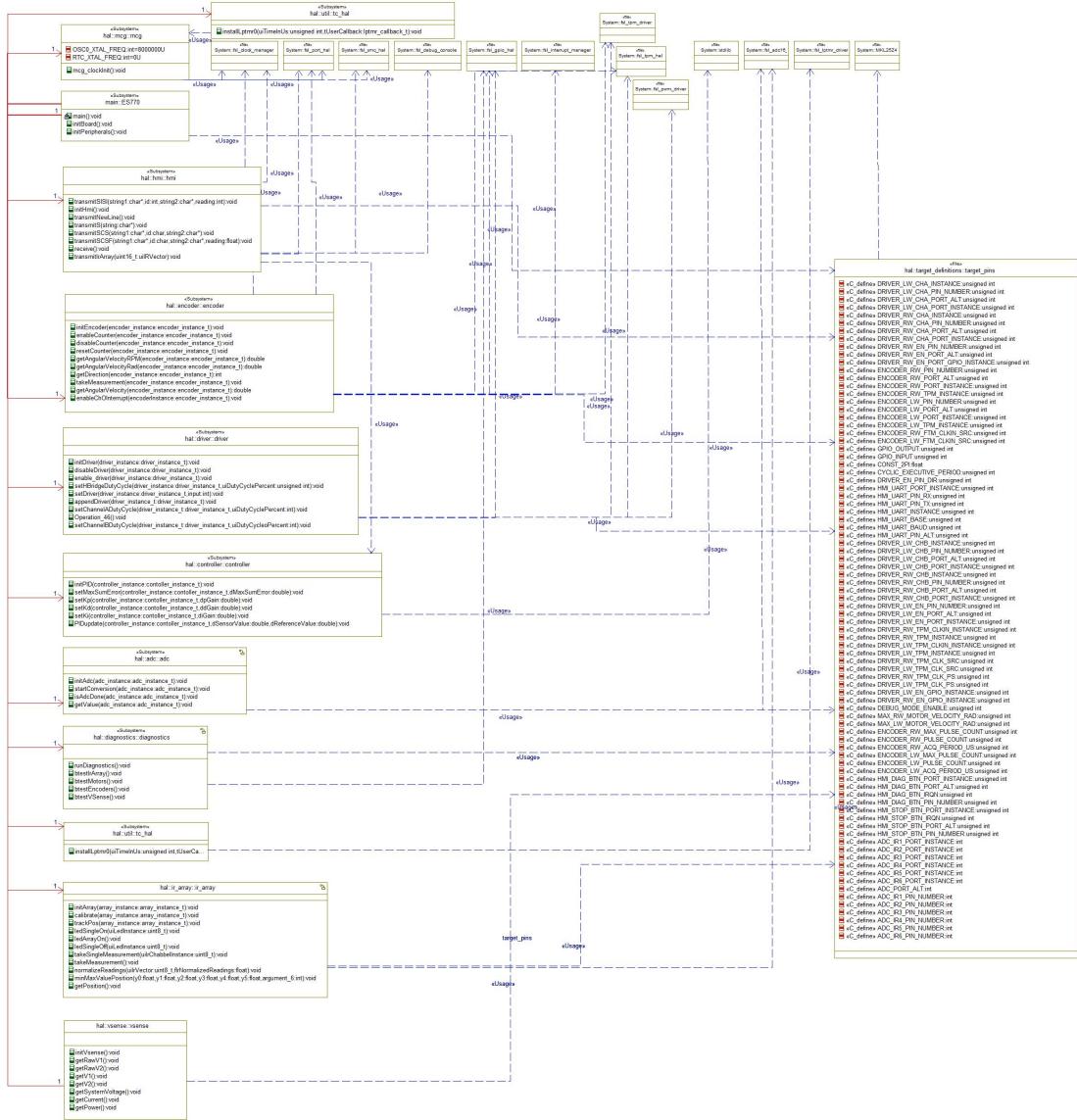


Figura 6: Diagrama de definição do Spark

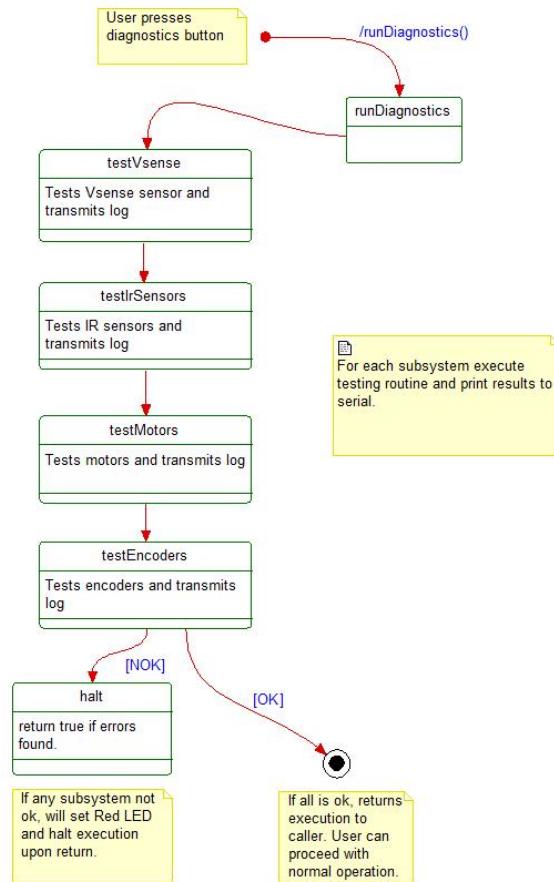


Figura 7: Diagrama de máquina de estados do auto-diagnóstico

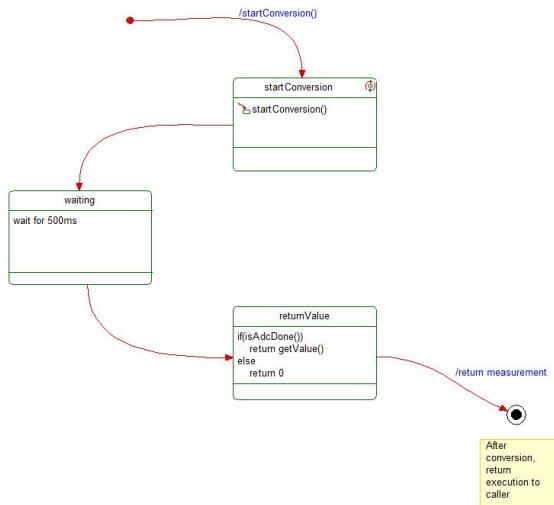


Figura 8: Diagrama de máquina de estados do ADC

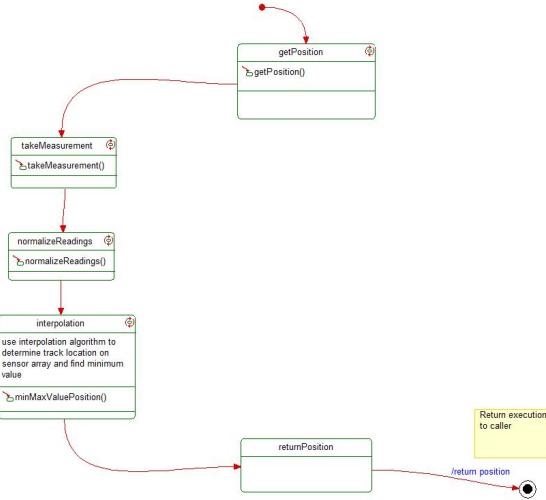


Figura 9: Diagrama de máquina de estados da leitura dos sensores

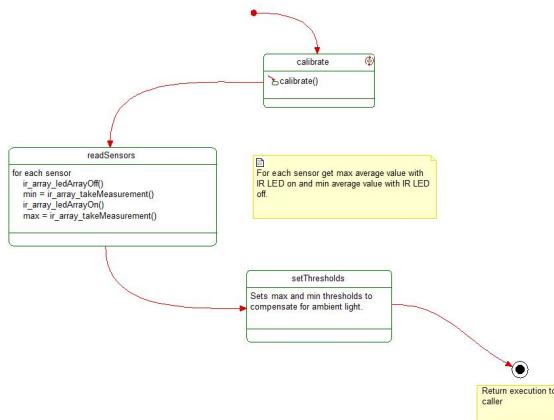


Figura 10: Diagrama de máquina de estados da calibração do sistema

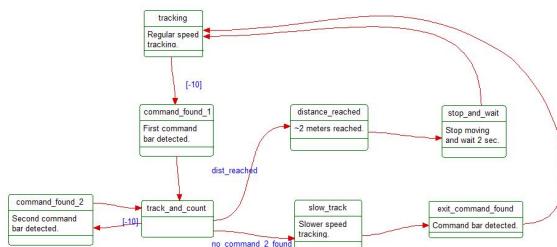


Figura 11: Diagrama de máquina de estados da interpretação de comandos

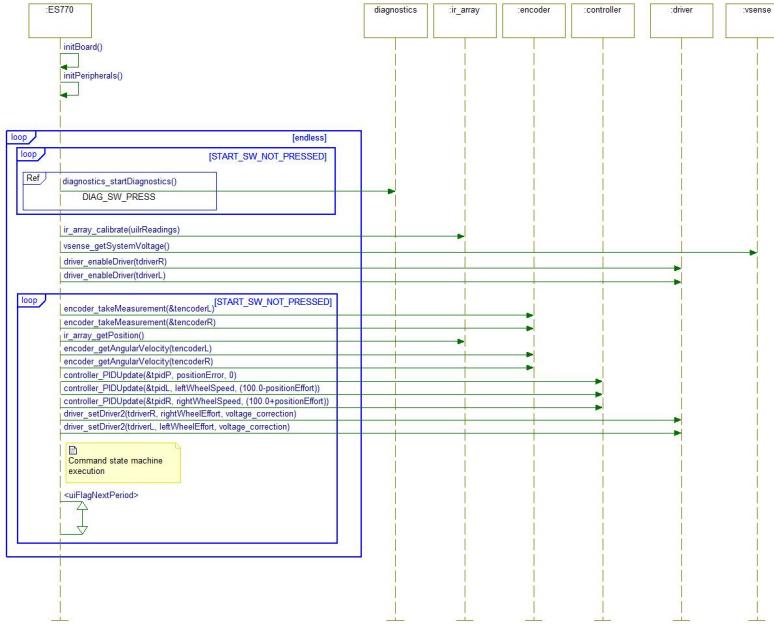


Figura 12: Diagrama de sequência da função *main*

3 Step by step da implementação do Spark

3.1 Overview do sistema

Como é possível observar na Figura 3. O sistema embarcado é constituído de uma placa de desenvolvimento Kinetis FRDM-KL25Z, responsável por executar o algoritmo de controle PID, tendo como *input* o sinal advindo dos *encoders* e sensores da pista, de forma que este gere o sinal de controle para os motor DC de cada roda. Todas as partes que compõe o *software* do *target* são executadas em uma janela fixa de tempo determinístico, conhecida como cíclico executivo, na qual todas as aquisições de dados e atuações no sistema devem ocorrer. Para o Spark foi utilizado um cíclico executivo com período de $50ms$, dos quais aproximadamente $28ms$ são utilizados para processamento dos dados, como pode ser observado na Figura 13.

3.2 Lógica de implementação do Spark

- Escolha do Microcontrolador

O NXP Kinetis KL25Z128 é o microcontrolador escolhido para o Spark, utilizado através da placa de desenvolvimento Kinetis FRDM-KL25Z. Este microcontrolador é adequado para a tarefa, uma vez que os sinais de controle do motor podem ser implementados usando seus módulos de temporizador de *hardware* fornecendo uma precisão aceitável e facilidade de implementação. A disponibilidade e baixo custo desta linha de microcontroladores também o torna um candidato adequado para esta aplicação.

- Escolha dos periféricos

Foram escolhidos os sensores para detecção da pista e encoders de forma a tornar a implementação simples e robusta. No caso foram utilizados 6 sensores igualmente espaçados para detecção da pista e encoders especialmente desenvolvidos para o Spark de forma que se encaixem facilmente nas rodas. Uma bateria de LiFePO₄ também foi utilizada para alimentar o Spark de forma que haja longo tempo de operação e seja facilmente recarregada. O chassis assim como os motores das rodas foram adquiridos comercialmente.

- Integração do sistema

Na Figura 3. pode-se observar como cada componente do sistema é interconectado, assim como suas funções. No diagrama de requisitos da figura 4. pode-se observar as tarefas que o sistema é capaz de realizar e através da matriz de rastreabilidade é possível identificar como o código fonte e decisões de implementação tornaram possível satisfazer tais requerimentos.

O diagrama de pacotes da Figura 5. indica a estrutura organizacional utilizada na implementação do *software*. Finalmente o diagrama de definições mostra as rotinas desenvolvidas e suas dependências. Podemos observar também os diagramas de máquina de estado (Fig. 7, 8, 9, 10, 11) que mostram o comportamento de diversos algoritmos implementados durante sua execução.

A sequência de execução do código principal é mostrada na Figura 12. onde é possível observar as rotinas executadas durante a operação do Spark.

- Auto diagnóstico

O Spark possui um sistema integrado de auto-diagnóstico que executa as rotinas utilizadas durante operação normal de forma a avaliar se o comportamento obtido está dentro do esperado. Os resultados da rotina de auto-diagnóstico, que pode ser iniciada pressionando-se o botão de diagnóstico, são mostrados através de interface serial, tornando-se fácil identificar os pontos de falha do sistema, como um sensor inoperante ou encoder defeituoso. Pode-se observar o comportamento da rotina de auto-diagnóstico pelo diagrama de máquina de estados na Figura 7.

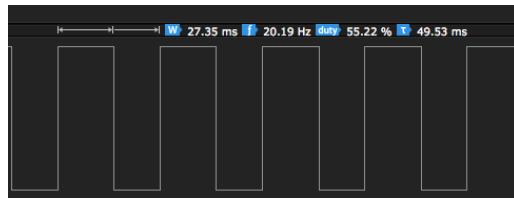


Figura 13: Ciclico executivo e tempo de execução.

3.3 Desenvolvimento das placas de circuito

O microcontrolador utilizado possui os módulos de *hardware* necessários para que satisfaçamos as necessidades e requisitos do Spark, resultando em uma melhor performance e maior facilidade de implementação quando comparado com um algoritmo implementado inteiramente em *software*.



Figura 14: *Zoom* na placa base e placa de desenvolvimento.

No caso foram utilizados 3 módulos de Timer/PWM. 1 módulo para operação do PWM de ambos os motores, através de 4 canais acionados por topologia *push-pull* (Fig.15) operando à uma frequência de 25kHz. Os 2 módulos restantes foram utilizados para a contagem dos pulsos de cada encoder, operando no modo de clock externo possibilitado pelo microcontrolador.

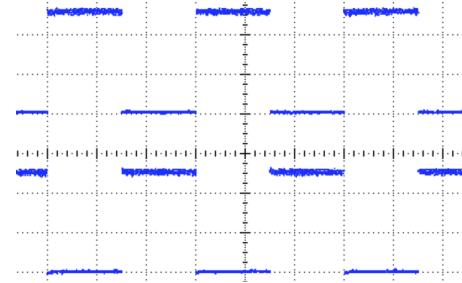


Figura 15: Topologia *Push-pull* PWM para operação da ponte H.

Foi desenvolvida uma placa base (Fig.16, 17) responsável pela interligação e alimentação do Spark com uma bateria de 11.1V LiFePO₄, botões de diagnóstico e início/parada, e resistor para medição de corrente sendo consumida. Também foi desenvolvidas uma placa de sensores da pista (Fig.18, 19) que contém 6 sensores infravermelho são utilizados para detectar a intensidade de luz refletida da pista.

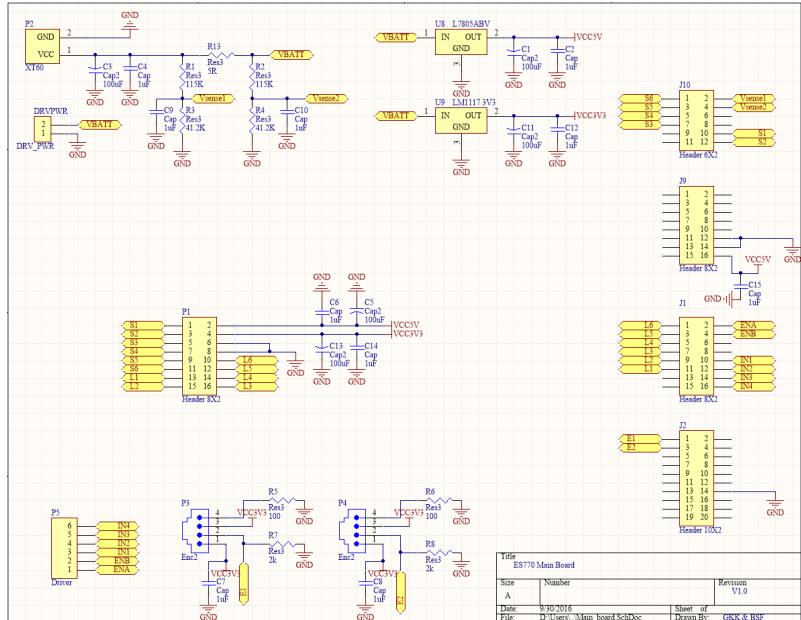


Figura 16: Diagrama esquemático da placa base do Spark.

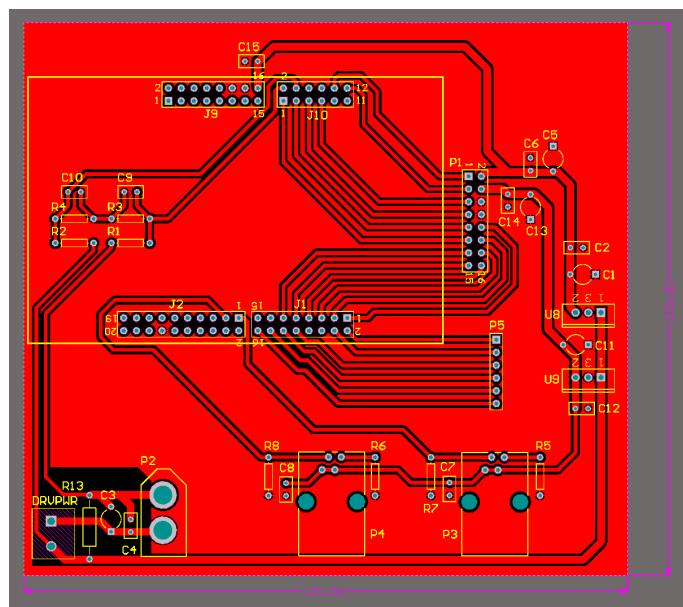


Figura 17: PCB da placa base do Spark.

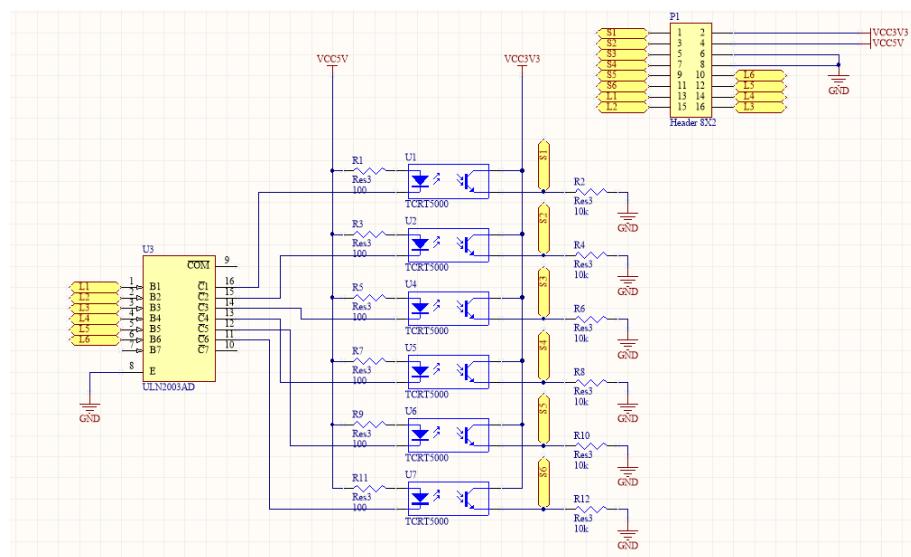


Figura 18: Diagrama esquemático da placa de sensores infra-vermelho do Spark.

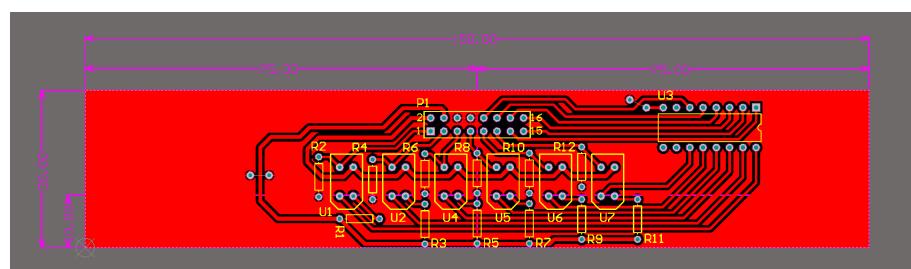


Figura 19: PCB da placa de sensores infra-vermelho do Spark.

O sensor infravermelho utilizado (Vishay TCRT5000, Fig.20) já possui os LED infravermelhos, operados a 5V e fototransistor com filtro infravermelho operado a 3.3V integrados em uma estrutura plástica. Sua leitura é feita através dos múltiplos canais *single-ended* do módulo de ADC de 16bits integrado ao microcontrolador.

A rotina de captura dos valores dos sensores pode ser observada nos diagramas de máquina de estados das Figuras 8 e 9.



Figura 20: Sensor infravermelho TCRT5000

3.4 Montagem mecânica

3.4.1 Ponte H

Para o funcionamento do motor DC em cada roda será utilizada uma ponte H dupla operada com PWM bipolar. Esta implementação utiliza um circuito comercial que faz uso do CI L298 *Dual H-Bridge Driver* com os 4 canais atuados pelo microcontrolador usando dois pares canais PWM com topologia *push-pull* para cada motor DC.

3.4.2 Encoder

Foi desenvolvido um *encoder* óptico com 27 pulsos por rotação completa que foi especialmente desenhado e impresso por processo de fusão por deposição de ABS para a utilização no Spark. Este encoder é adequado para um controlador de velocidade de motor DC de um seguidor de linha operando a baixas velocidades. O sensor utilizado para detecção das interrupções causadas pelos dentes do encoder é o PHCT203, operado com tensão de 3.3V para garantir compatibilidade com o microcontrolador utilizado. O eixo do motor é conectado diretamente ao encoder.

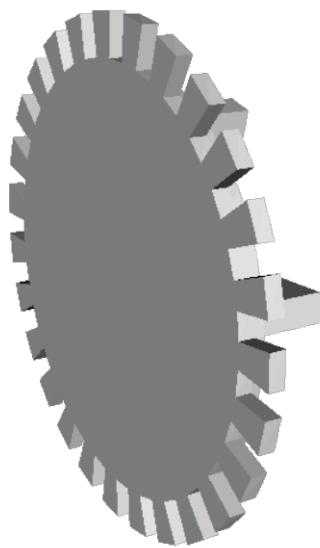


Figura 21: Desenho 3D do encoder.

A velocidade angular é determinada calculando a quantidade de pulsos que cada canal gera durante o período do executivo cíclico, e dividindo-a pela quantidade de pulsos que a roda do encoder produz por revolução como tal:

$$\omega = \frac{\text{pulse count}}{\text{time window} \times \text{pulses per revolution}}$$

3.5 Detecção da trajetória

O algoritmo de interpolação utilizado para geração da curva de intensidade lida pelos sensores é o *Uniform Catmull-Rom Spline Algorithm*. Este é um dos algoritmos mais simples de serem implementados, além disso a implementação utilizada, com espaçamento uniforme, foi suficiente e reduz o tempo de computação devido à uma grande redução da necessidade de operações intensivas como multiplicação de pontos flutuantes e raiz quadrada utilizadas no método centrípeto, que fornece resultados um pouco melhores.

Os pontos que definem uma *spline* são conhecidos como "Pontos de Controle". Uma das características do método *Catmull-Rom* é que a curva especificada passará por todos os pontos de controle. Para calcular um ponto na curva, são necessários 4 pontos (P_1, \dots, P_4), para encontrar a interpolação entre os pontos P_2 e P_3 , como mostrado na figura abaixo.

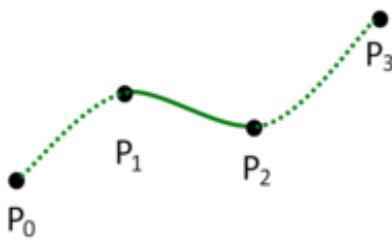


Figura 22: Funcionamento do algoritmo de interpolação

Antes de iniciar a operação do algoritmo de interpolação os sensores da pista são calibrados,

segundo-se a sequencia definida pelo diagrama de máquina de estados da Figura 10. Isto garante que a sensibilidade de cada sensor, ou variações nos valores dos componentes utilizados não influam adversamente nas leitura e valores confiáveis estejam disponíveis. A calibração é realizada sempre que o Spark inicia a operação.

O ponto é especificado por um valor t que representa a porção da distância entre os dois pontos de controle P_2 e P_3 .

A Figura 23 representa um *waterfall plot* das *splines* geradas utilizando-se $t = 46$ pontos (eixo horizontal) geradas ao longo do tempo (eixo vertical) ao mover o Spark lateralmente na pista. Cores mais escuras representam valores mais baixos. A Figura 24 indica uma única spline capturada, mostrando, à esquerda uma leitura válida, onde a pista é claramente discernida do entorno e à direita uma leitura inválida onde não pode ser tirada conclusões.

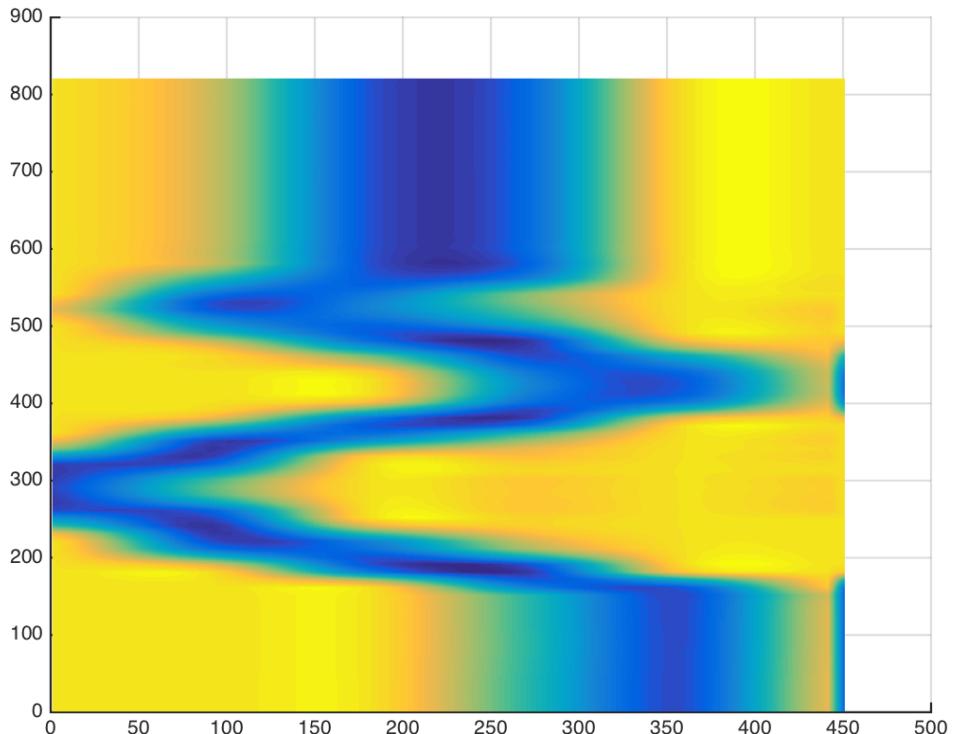
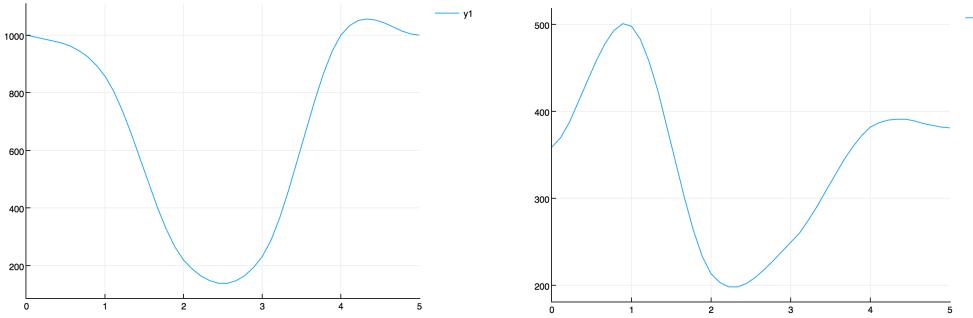


Figura 23: Trajetória calculada via algoritmo de *spline*



(a) Curva normal, posição pode ser encontrada entre sensores 2 e 3 (c) Curva anormal, não há diferença adequada entre máximo e mínimo, será ignorada.

Figura 24: Curvas interpoladas de intensidade dos sensores.

Após a execução do algoritmo de interpolação, o mínimo valor da spline é encontrado e caso seja significantemente menor que o valor máximo encontrado este ponto então é tratado como válido e será utilizado para o Spark poder seguir a pista. Caso os valores mínimo e máximo estiverem próximos isto indica a passagem de uma barra de comando que será tratada pela máquina de estados de comandos (Fig. 11).

3.5.1 Interpretação dos Comandos

Os comandos presentes na pista são interpretados através de uma máquina de estados (Fig. 11). Se o algoritmo de detecção da pista indicar que foi encontrada uma barra de comando, a máquina de estados presente no código principal fica responsável em interpretar a sequência de barras de comando presentes, executando assim comandos variados.

3.6 Projeto de controle do Spark e Acionamento dos Motores

O controlador PID pode ser representado no tempo pela seguinte relação matemática:

$$y(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{dE}{dt} \quad (1)$$

Em que:

- $y(t)$ é o valor sendo mensurado
- K_p é o ganho proporcional
- K_i é o ganho interativo
- K_d é o ganho derivativo
- $e(t)$ é o erro entre o valor de referência e o valor mensurado
- $u(t)$ é o esforço de controle do sistema

Para implementar este controlador via *software* embarcado no microcontrolador, sem a necessidade de uso de qualquer transformação para o domínio da frequência, o seguinte algoritmo é utilizado. O algoritmo pode ser dividido em três seções: Proporcional, Integrativo e Derivativo, tais como:

- Proporcional
 $\text{valueProportional} = K_p * \text{measuredError}$
- Derivativo
 $\text{valueDerivative} = K_d * (\text{ValueMeasured} - \text{lastValueMeasured})$
- Integrativo
 $\text{valueIntegrative} = K_i * \text{sumOfMeasuredValues}$
- $y(t)$
 $\text{controllerOutput} = \text{valueProportional} + \text{valueDerivative} + \text{valueIntegrative}$

Para o Spark foram implementados 3 controladores PID:

Um controlador PID para o erro de posição que possui uma referência fixa em $ref = 0$ ou seja o seguidor sempre procura centralizar-se com a pista. O erro obtido pelo algoritmo de interpolação e deteção de mínimo varia entre $-2.5 \leq e \leq 2.5$ que representa a posição relativa ao centro da placa de sensores. Após a execução do algoritmo e aplicação das constantes P, I, D do controlador é gerado um esforço de controle para corrigir a posição do seguidor.

Este esforço de controle é então alimentado na velocidade de referência somada de uma velocidade média constante, nos 2 controladores PID, um para cada roda, da seguinte maneira:

$$\begin{aligned} \text{Erro do controlador da roda esquerda} &= \text{velocidade média} - \text{esforço de posição} \\ \text{Erro do controlador da roda direita} &= \text{velocidade média} + \text{esforço de posição} \end{aligned} \quad (2)$$

De forma com que o seguidor mantenha uma velocidade média e se move em linha reta caso o erro for nulo, e caso o erro não for nulo o seguidor irá mudar de direção de forma a recuperar a trajetória.

As constantes P, I, D do controlador de posição foram encontradas de maneira que o esforço de posição pode ser alimentado diretamente como referência nos controladores de velocidade. As constantes P, I, D dos controladores dos motores foram encontradas de forma com que o seguidor se move em linha reta para uma referência constante.

O esforço de controle produzido pelos controladores das rodas no entanto não é limitado entre $-100 \leq \text{set_driver} \leq 100$ (o *range* de valores aceitos na rotina *set_driver()* para acionamento da ponte H, mais detalhes na documentação da função) para o controle da ponte H. Apesar das constantes P, I, D dos controladores das rodas terem sido escolhidas de forma a possibilitar uma alimentação direta do esforço de controle ao PWM das rodas, o sinal teve que ser condicionado antes de ser enviado para os motores. Foi assim limitado o valor mínimo do esforço para -10 evitando que o motor gire na direção oposta ao desejado e o esforço também é saturado em $+100$ pela rotina *set_driver()*.

Devido a operação com bateria LiFePO4 com tensão de operação de $10V \leq V_{in} \leq 11.8V$ o PWM dos motores teve que ser limitado para suprir os motores de uma tensão máxima dada a tensão atual da bateria, portanto a tensão da bateria é avaliada e um fator de correção gerado antes do inicio da operação e o valor do *duty cycle* desejado é então multiplicado pelo fator de correção de forma com que o motor sempre seja operado com a tensão máxima constante, sem necessidade de calibração do mesmo.

4 Matriz de rastreabilidade

ID do requisito	implementação
Req. 1	hal\ir_array calibrate(array_instance:array_instance_t):void
Req. 2	hal\ir_array initArray(array_instance:array_instance_t):void
Req. 3	hal\diagnostics hal\hmi hal\ir_array hal\vsense hal\encoder hal\driver runDiagnostics():void
Req. 4	hal\ir_array trackPos(array_instance:array_instance_t):void hal\controller initPID(controller_instance:controller_instance_t):void PIDupdate(controller_instance:controller_instance_t, dSensorValue:double,dReferenceValue:double):void
Req. 5	hal\ir_array trackPos(array_instance:array_instance_t):void
Req. 6	Spark opera com bateria e software embarcado.
Req. 7	hal\ir_array trackPos(array_instance:array_instance_t):void
Req. 8	hal\ir_array calibrate(array_instance:array_instance_t):void
Req. 9	hal\ir_array calibrate(array_instance:array_instance_t):void
Req. 10	hal\controller PIDupdate(controller_instance:controller_instance_t, dSensorValue:double,dReferenceValue:double):void
Req. 11	hal\controller PIDupdate(controller_instance:controller_instance_t, dSensorValue:double,dReferenceValue:double):void
Req. 12	Spark opera com uma placa de sensores infra-vermelho.

ID do requisito	implementação
Req. 13	<pre>hal\driver hal\vsense initDriver(driver_instance:driver_instance_t):void disableDriver(driver_instance:driver_instance_t):void enable_driver(driver_instance:driver_instance_t):void setHBridgeDutyCycle(driver_instance:driver_instance_t,uiDutyCyclePercent:int):void setDriver(driver_instance:driver_instance_t,input:int):void</pre>
Req. 14	<pre>hal\encoder initEncoder(encoder_instance:encoder_instance_t):void enableCounter(encoder_instance:encoder_instance_t):void disableCounter(encoder_instance:encoder_instance_t):void resetCounter(encoder_instance:encoder_instance_t):void getAngularVelocityRPM(encoder_instance:encoder_instance_t):double getAngularVelocityRad(encoder_instance:encoder_instance_t):double getDirection(encoder_instance:encoder_instance_t):int takeMeasurement(encoder_instance:encoder_instance_t):void getAngularVelocity(encoder_instance:encoder_instance_t):double</pre>
Req. 15	<pre>hal\controller initPID(controller_instance:contoller_instance_t):void setMaxSumError(controller_instance:contoller_instance_t,dMaxSumError:double):void setKp(controller_instance:contoller_instance_t,dpGain:double):void setKd(controller_instance:contoller_instance_t,ddGain:double):void setKi(controller_instance:contoller_instance_t,diGain:double):void PIDupdate(controller_instance:contoller_instance_t, dSensorValue:double,dReferenceValue:ind):void</pre>
Req. 16	<pre>hal\encoder initEncoder(encoder_instance:encoder_instance_t):void enableCounter(encoder_instance:encoder_instance_t):void disableCounter(encoder_instance:encoder_instance_t):void resetCounter(encoder_instance:encoder_instance_t):void getAngularVelocityRPM(encoder_instance:encoder_instance.t):double getAngularVelocityRad(encoder_instance:encoder_instance.t):double getDirection(encoder_instance:encoder_instance.t):int takeMeasurement(encoder_instance:encoder_instance.t):void getAngularVelocity(encoder_instance:encoder_instance.t):double</pre>
Req. 17	<pre>hal\diagnostics hal\hmi hal\ir_array hal\vsense hal\encoder hal\driver runDiagnostics():void</pre>

ID do requisito	implementação
Req. 18	<pre>hal\diagnostics hal\hmi hal\ir_array hal\vsense hal\encoder hal\driver runDiagnostics():void</pre>
Req. 19	<pre>hal\diagnostics hal\hmi hal\ir_array hal\vsense hal\encoder hal\driver runDiagnostics():void</pre>
Req. 20	<pre>hal\controller initPID(controller_instance:contoller_instance_t):void PIDupdate(controller_instance:contoller_instance_t, dSensorValue:double,dReferenceValue:ind):void hal\encoder initEncoder(encoder_instance:encoder_instance_t):void getAngularVelocity(encoder_instance:encoder_instance_t):double</pre>

5 Notas

As dificuldades encontradas no desenvolvimento desta primeira parte do experimento.

- A placa KL25Z acabou por queimar por duas vezes, a primeira devido a uma conexão errada que causou a queima do conversor ADC e a segunda ocasionada por um curto circuito em um componente interno da placa;
- Acabamos por optar por um projeto alternativo de *encoder*, deixando-o na parte externa da roda. Por causa disto, a roda por muitas vezes gerava atrito com o *encoder* sendo que ao final tivemos que cortar um pedaço da roda;
- Existem muitas poucas placas KL25Z de reposição disponíveis no mercado brasileiro, e as poucas que existem tem um preço consideravelmente alto;
- A fixação dos sensores na placa foi um pequeno desafio, dado que a altura do pino que veio com o carro fazia com que a placa tocasse o chão, nos obrigando a confeccionar pinos personalizados para a placa;
- Para a realização do *encoder*, foi necessário imprimir via impressora 3D um par de engrenagens dentadas para o projeto do *encoder* externo;
- O botão de auto-diagnóstico estava com a mesma pinagem do botão de *reset*, o que fez com que tivesse-mos que re-soldar a placa base;
- Algoritmo de *Centripetal Catmull-Rom spline* demasiadamente intensivo computacionalmente;

- Não há um método pragmático para encontrar as constantes k_p , k_i e k_d , sendo necessário tentativa e erro para que se possa estimar o valor das mesmas.

6 Apêndice

O código fonte está disponível no *github* assim como o modelo em SysML, figuras e tabela de alocação dos pinos e a documentação gerada através do gerador de documentação Doxygen: <https://github.com/gkolotelo/770>.