Universidade Estadual de Campinas

Faculdade de Engenharia Mecânica

ES770 - Laboratório de Sistemas Digitais

Relatório - Projeto Final

Veículo robótico seguidor de linha

Nome:	*RA*
Daniel Dello Russo Oliveira	101918
Vinícius Ragazi David	120258

29 de novembro de 2016

1 Objetivo

O objetivo do projeto é, de maneira incremental, planejar, montar e implementar um veículo robótico seguidor de linha 1. Inicialmente planejar todo o projeto através do Rational Rhapsody Modeler, em seguida, construir as placas eletrônicas e unificar os componentes mecânicos, por fim, implementar o controlador do sistema através do *target*.

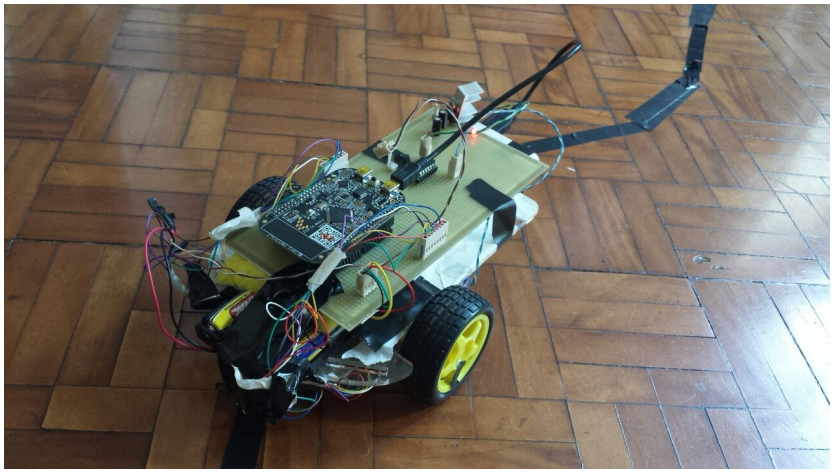

Figura 1: Veículo robótico seguidor de linha

2 Modelagem

Utilizando o Rational Rhapsody Modeler e tomando como base os requisitos propostos mostrados na figura 2, modelamos nossos sistema. Nessa etapa inicial nos preocupamos em detalhar como funcionaria cada componente do software e como estes interagem entre si, as figuras 3 e 4 nos permitem ter uma visão geral dos módulos do sistema.

Figura 2: Diagrama de requisitos

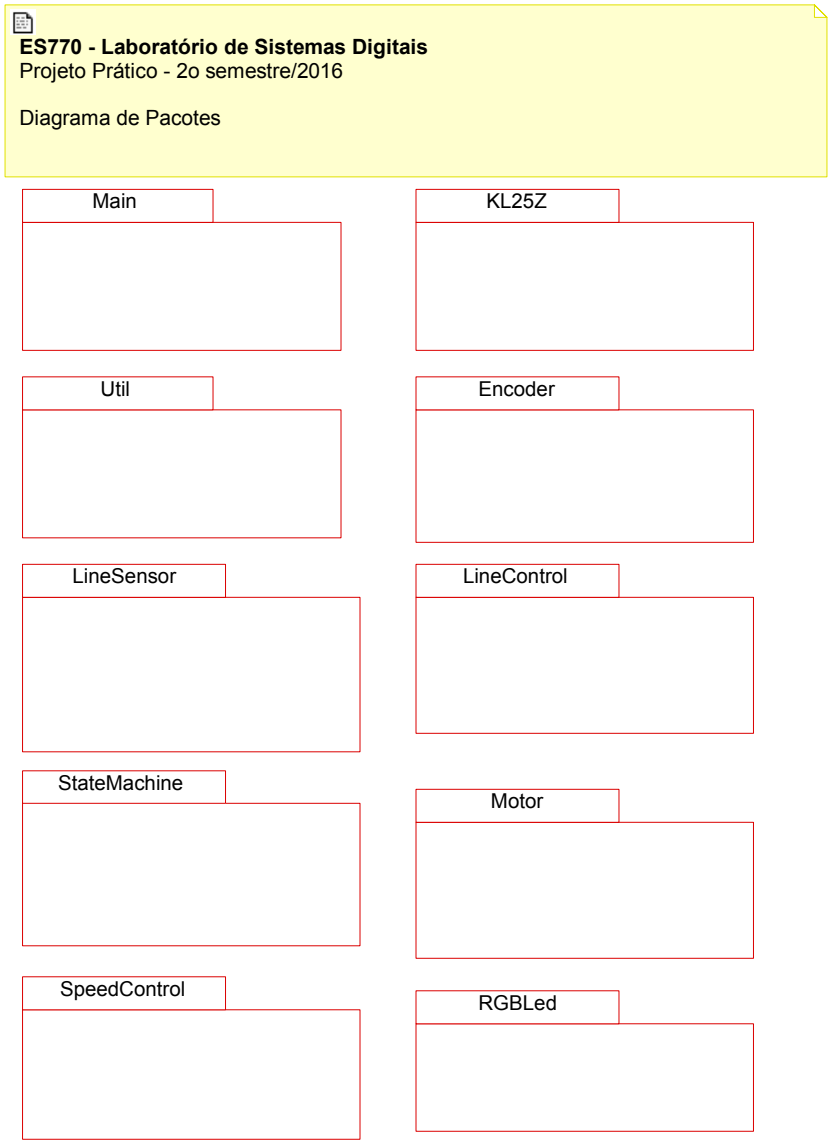

Figura 3: Diagrama de pacotes

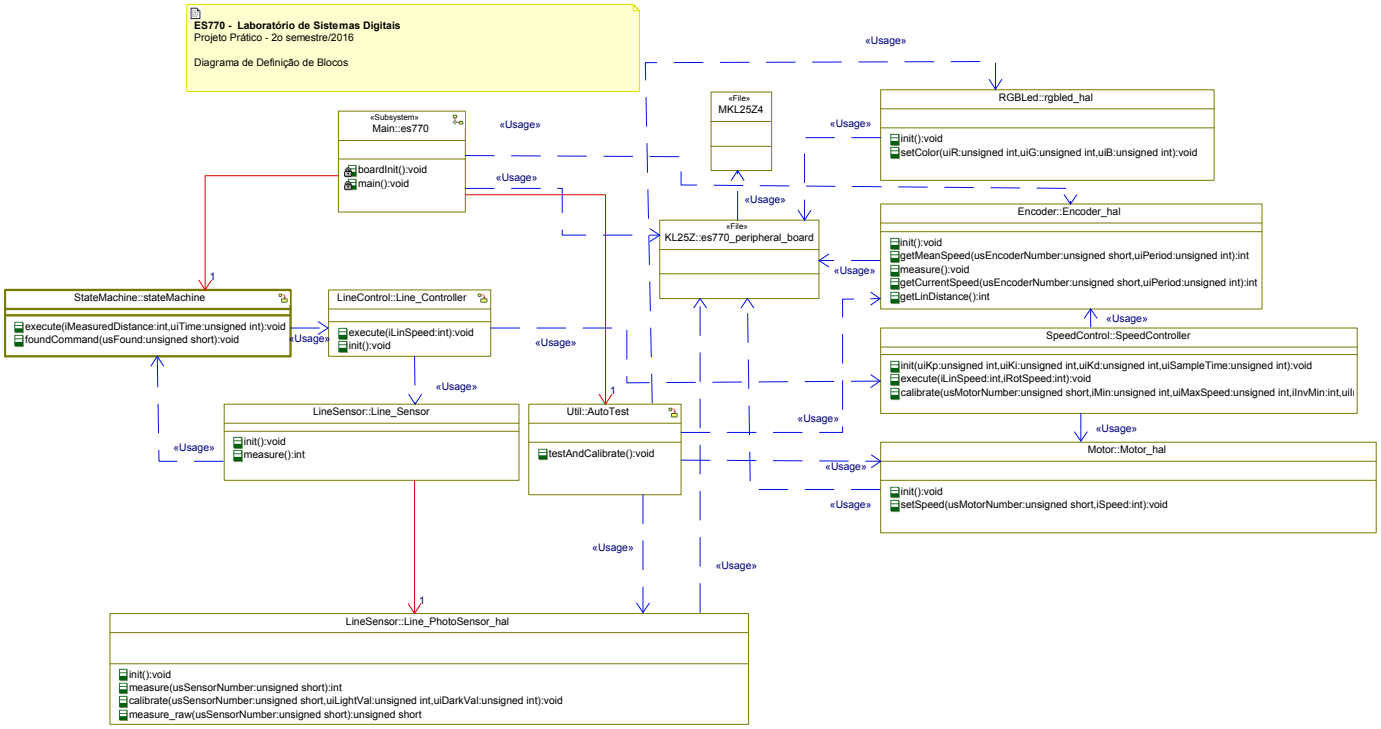

Figura 4: Diagrama de definição de blocos

Os blocos *Line_PhotoSensor_hal*, *encoder_hal*, *motor_hal* e *rgb_led_hal* fazem a interface entre o hardware e o software, possuindo funções para o controle dos atuadores ou a medição dos sensores. No bloco *encoder_hal* encontramos mais operações que nos outros devido à algumas particularidades do sistema. Escolhemos separar a requisição de leitura dos encoders da obtenção dos resultados para facilitar o reaproveitamento da mesma leitura em mais de um local do ciclo executivo cooperativo, fizemos issso através das funções *measure* e *getCurrentSpeed*/*getLinDistance*. Notamos que a precisão de nossos encoders não é alta o suficiente para obter uma medida de velocidade razoável dentro do período do nosso ciclo executivo padrão, para resolver essa dificuldade utilizamos a função *getMeanSpeed* que nos retorna uma média das últimas medidas (mantidas em um buffer circular), essa solução, embora introduza um atraso no sistema, nos permite manter o período do ciclo executivo relativamente baixo.

O bloco *Line_Sensor* é responsável por obter as medidas de luminosidade dos foto-sensores e converte-las na posição do centro da linha, isto é feito através do algoritmo spline, detalhes do seu funcionamento são apresentados nas figura

6. Ele também avisa nossa máquina de estados quando um sinal de comando de linha é encontrado. O bloco *SpeedController* é responsável pelo controle de velocidade do carrinho, ele recebe uma velocidade linear e angular de referência e executa o controle dos motores em malha fechada utilizando a medida dos encoders. Já o bloco *Line_ Controller* é responsável por seguir a linha, este requisita medições do centro de linha do *Line_ Sensor* e calcula as velocidades de referência que deverão ser passadas para o *SpeedController*, um esquema geral do seu funcionamento pode ser visto nas figuras 5, 6 e 7

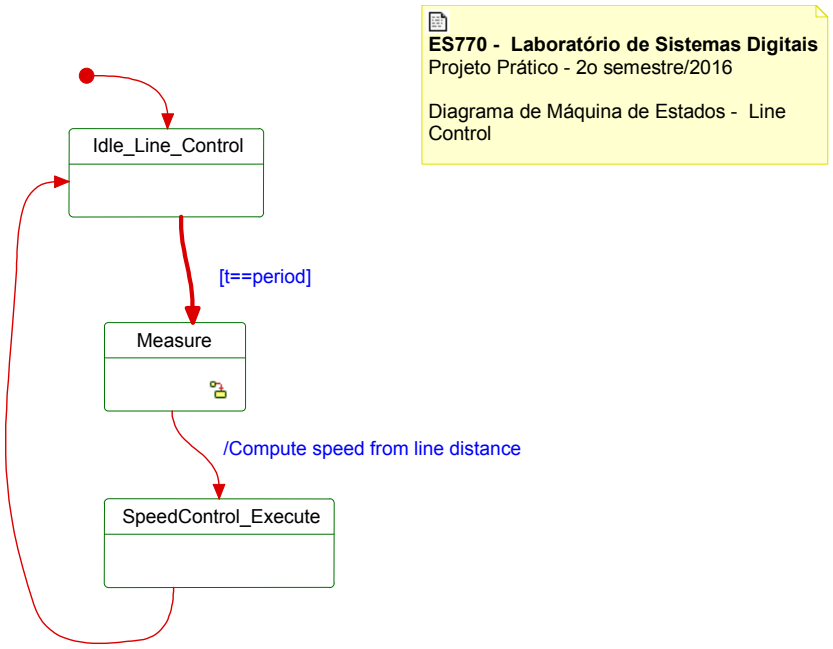

Figura 5: Diagrama de Máquina de Estados do controlador de linha

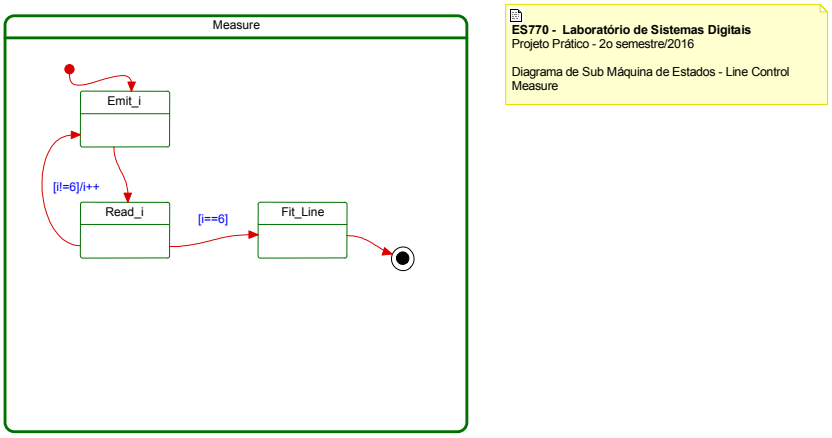

Figura 6: Diagrama de Sub Máquina de Estados - Sensor de linha

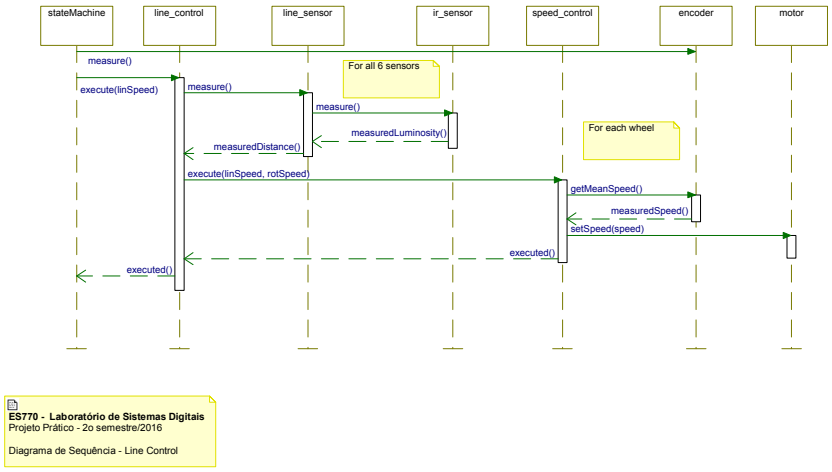

Figura 7: Diagrama de Sequência - Controlador de Linha

O bloco *AutoTest* tem como função o teste e calibração do sistema, testando primeiramente se os motores e foto-sensores estão funcionando para em seguida calibrá-los. Seu funcionamento encontra-se detalhado nas figuras 8, 9, 10 e 11.

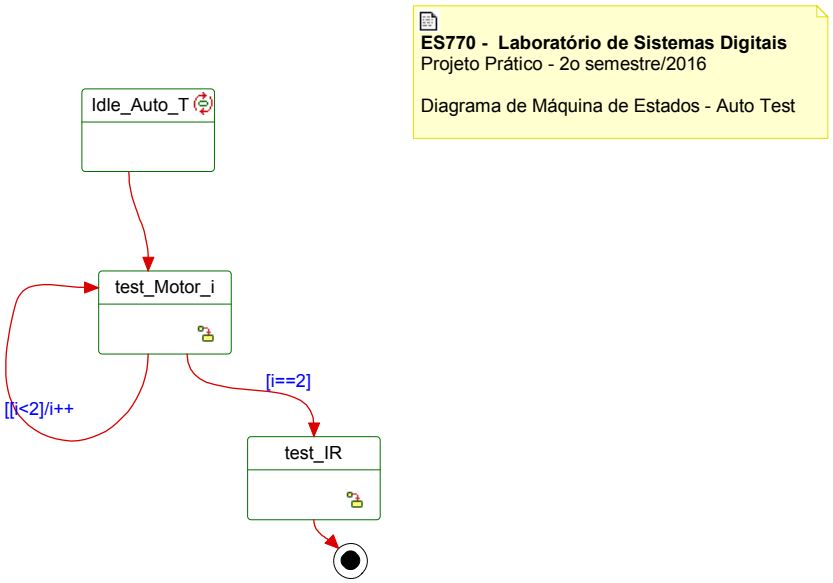

Figura 8: Diagrama de Máquina de Estados do módulo de Auto-Teste

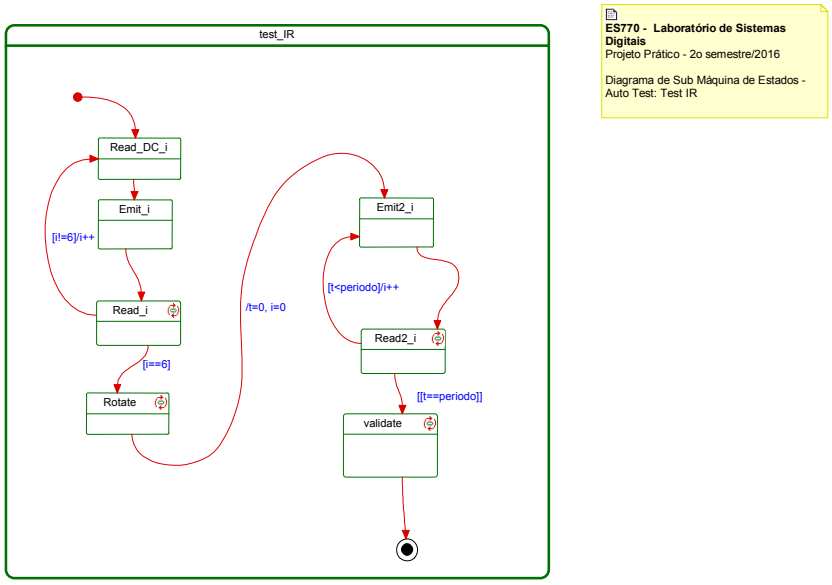

Figura 9: Diagrama de Sub Máquina de Estados - Teste dos Sensores infravermelhos

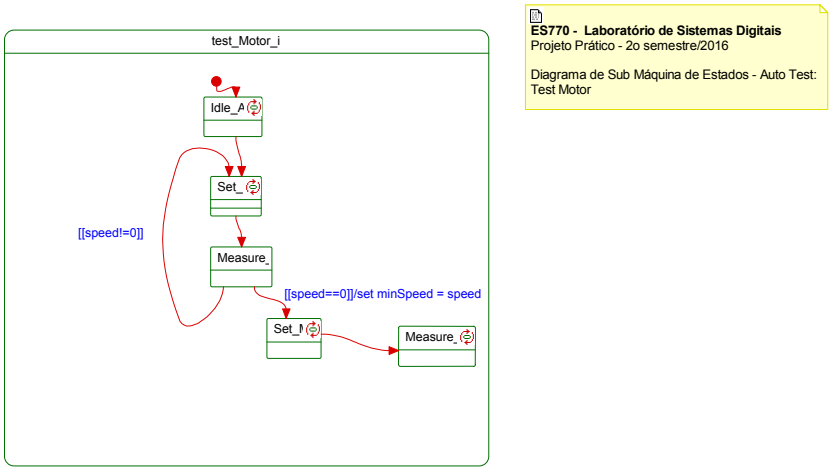

Figura 10: Diagrama de Sub Máquina de Estados - Teste dos Motores e Encoders

Figura 11: Diagrama de Sequência - Módulo de Auto-Teste e calibração

O bloco *stateMachine* lida com a interpretação dos comandos e ativação dos outros módulos do sistema (Acertando a velocidade linear de referência que será passada ao nosso controlador de lina). A máquina de estados do tratamento dos comandos pode ser vista na figura 12.

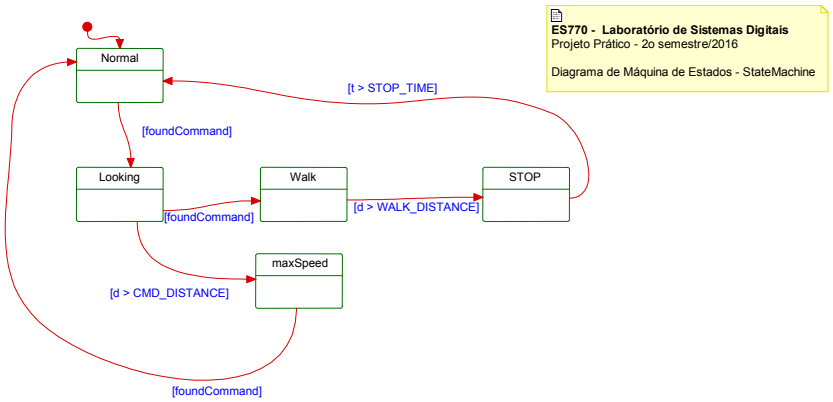

Figura 12: Diagrama de máquina de estados para interpretação dos comandos

Nos estados normal e walk o nosso controlador de linha receberá uma velocidade linear de referência relativamente baixa (em torno de 50% da velocidade máxima do sistema), quando estamos no estado maxSpeed essa referência será maior (95%) e em stop ela será nula.

O diagrama de atividades apresentado na figura 13 sintetiza o sistema como um todo, mostrando sua execução através do modelo cíclico executivo cooperativo.

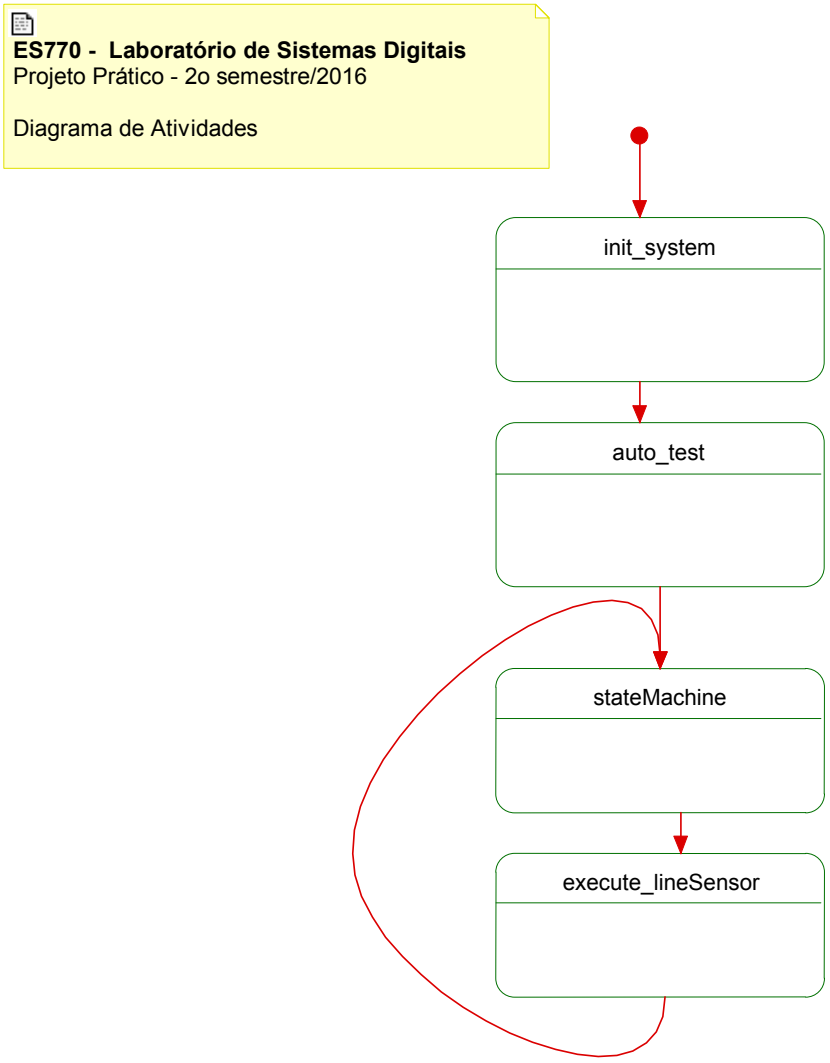

Figura 13: Diagrama de atividades do sistema

3 Matriz de Rastreabilidade

A matriz de rastreabilidade apresentada na tabela 1 relaciona cada um dos requisitos com a sua implementação.

Requisito	SubRequisito	Implementação
REQ1	REQ1.A	autotest.c - void autotest_testAndCalibrate(void) - void autotest_testPhotoSensors(void)
	REQ1.B	autotest.c - void autotest_testAndCalibrate(void) - void autotest_testPhotoSensors(void)
	REQ1.C	autotest.c - void autotest_testAndCalibrate(void) - void autotest_testPhotoSensors(void)
	REQ1.D	autotest.c - void autotest_testAndCalibrate(void) - void autotest_testPhotoSensors(void)
	REQ1.E	autotest.c - void autotest_testAndCalibrate(void) - void autotest_testPhotoSensors(void)
	REQ1.F	autotest.c - void autotest_testAndCalibrate(void) - void autotest_testPhotoSensors(void)
	REQ1.G	autotest.c - void autotest_testAndCalibrate(void) - void autotest_testMotors()
	REQ1.H	autotest.c - void autotest_testAndCalibrate(void) - void autotest_testMotors()
REQ2	REQ2.A	autotest.c - void autotest_testAndCalibrate(void) - void autotest_calibratePhotoSensors(void) photosensor_hal.c - void photoSensor_calibrate(unsigned short usSensorNumber, int iLightVal, int iDarkVal)
	REQ2.B	autotest.c - void autotest_testAndCalibrate(void) - void autotest_calibratePhotoSensors(void) photosensor_hal.c - void photoSensor_calibrate(unsigned short usSensorNumber, int iLightVal, int iDarkVal)
	REQ2.C	autotest.c - void autotest_testAndCalibrate(void)

		- void autotest_calibratePhotoSensors(void) photosensor_hal.c - void photoSensor_calibrate(unsigned short usSensorNumber, int iLightVal, int iDarkVal)
	REQ2.D	autotest.c - void autotest_testAndCalibrate(void) - void autotest_calibratePhotoSensors(void) photosensor_hal.c - void photoSensor_calibrate(unsigned short usSensorNumber, int iLightVal, int iDarkVal)
	REQ2.E	autotest.c - void autotest_testAndCalibrate(void) - void autotest_calibratePhotoSensors(void) photosensor_hal.c - void photoSensor_calibrate(unsigned short usSensorNumber, int iLightVal, int iDarkVal)
	REQ2.F	autotest.c - void autotest_testAndCalibrate(void) - void autotest_calibratePhotoSensors(void) photosensor_hal.c - void photoSensor_calibrate(unsigned short usSensorNumber, int iLightVal, int iDarkVal)
	REQ2.G	autotest.c - void autotest_testAndCalibrate(void) - void autotest_calibrateMotors() speedController.c - void speedControl_calibrate(unsigned short usMotorNumb, int iMin, unsigned int uiMax, int iInvMin, unsigned int uiInvMax)
	REQ2.H	autotest.c - void autotest_testAndCalibrate(void) - void autotest_calibrateMotors() speedController.c - void speedControl_calibrate(unsigned short usMotorNumb, int iMin, unsigned int uiMax, int iInvMin, unsigned int uiInvMax)
REQ3	REQ3.A	speedController.c - void speedControl_init(unsigned int uiKp[2], unsigned int uiKi[2], unsigned int uiKd[2], unsigned int sampleTime)

		- void speedControl_execute(int iLinSpeed, int iAngSpeed)
	REQ3.B	speedController.c - void speedControl_init(unsigned int uiKp[2], unsigned int uiKi[2], unsigned int uiKd[2], unsigned int sampleTime) - void speedControl_execute(int iLinSpeed, int iAngSpeed)
REQ4	REQ4.A	encoder_hal.c - void encoder_init(void) - void encoder_measure() - unsigned int encoder_getCurrentSpeed(unsigned short usEncoderNumber, unsigned int uiPeriod) - unsigned int encoder_getMeanSpeed(unsigned short usEncoderNumber, unsigned int uiPeriod) - unsigned int encoder_getLinDistance()
	REQ4.B	encoder_hal.c - void encoder_init(void) - void encoder_measure() - unsigned int encoder_getCurrentSpeed(unsigned short usEncoderNumber, unsigned int uiPeriod) - unsigned int encoder_getMeanSpeed(unsigned short usEncoderNumber, unsigned int uiPeriod) - unsigned int encoder_getLinDistance()
REQ5	REQ5.A	motor.c - void motor_init(void) - void motor_setSpeed(unsigned short usMotorNumber, int iSpeed)
	REQ5.B	motor.c - void motor_init(void) - void motor_setSpeed(unsigned short usMotorNumber, int iSpeed)
REQ6	-	stateMachine.c - void stateMachine_foundCommand(unsigned short usFoundCommand) lineSensor·c - int lineSensor_measure()
REQ7	-	stateMachine.c

		- void stateMachine_execute(int iMeasuredDistance, unsigned int uiTime))
REQ8	-	lineSensor·c - int lineSensor_measure() photosensor_hal.c - unsigned short photoSensor_measure(unsigned short usSensorNumber)
REQ9	-	lineControl·c - void lineControl_init(unsigned int uiKp, unsigned int uiKi, unsigned int uiKd, unsigned int sampleTime) - void lineControl_execute(int iLinSpeed)

Tabela 1: Matriz de Rastreabilidade

4 Montagem eletrônica e mecânica

Seguindo o modelo sugerido pela figura 14 montamos a placa de sensores 15. A placa de sensores é um componente elétrico que faz as medidas de posição do veículo, através de 6 pares de emissor/receptor de infra-vermelho. Cada par é separado fisicamente em 10mm. Essa distância é selecionada para que cada não haja grande interferência, a distância é interessante também para a computação da posição, já que, o tamanho da linha a ser seguida possui por volta deste valor. A placa de sensores de posição foi integrada à placa geral de controle, para isso através de conectores KK. A alimentação do circuito lógico foi realizada através de um Powerbank.

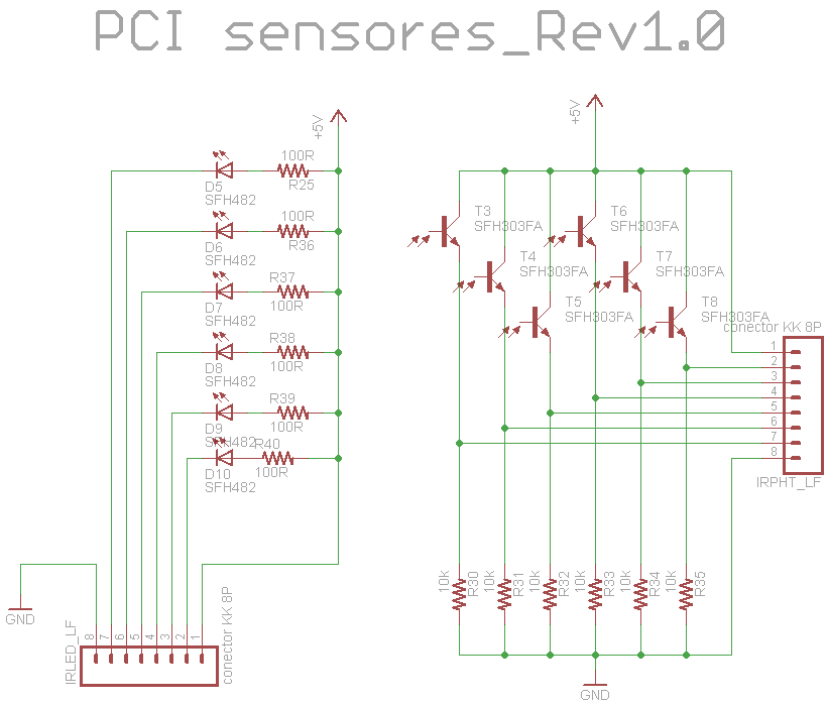

Figura 14: Diagrama esquemático da placa de sensores infra-vermelhos

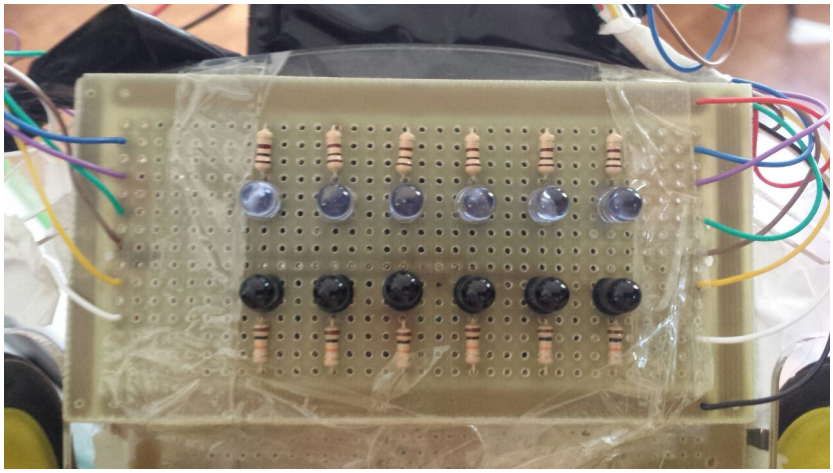

Figura 15: Placa de sensores infra-vermelhos

Como os receptores produzem um sinal analógico, for necessária a utilização de ADCs para a leitura dos mesmos, para isso utilizamos as portas as

portas PTB0, PTB1, PTB2, PTE20, PTE21 e PTE22 do target, configuradas,respectivamente, como ADC0_SE8, ADC0_SE9, ADC0_SE12, ADC0_SE0, ADC0_SE4a, ADC0_SE3. Já os emissores são acionados por portas GPIO, utilizamos as portas PTA16, PTA17, PTC13, PTC16, PTC17, e PTE31 configuradas como saídas GPIO. Para a aquisição dos sinais de entrada configuramos nosso módulo ADC para trabalhar com conversões de 16-bit, com tempo de amostragem curto e utilizando o recurso de média por hardware, configurada para 32 amostras por leitura. Calculamos o tempo de computação dessas medidas e vimos que estamos dentro do tempo aceitável para nosso ciclo executivo.

A montagem mecânica compreende a interligação dos encoders com o motor e a placa geral de controle 16. Para isso foi utilizada uma ponte H L293, 17, e uma alimentação própria, 4 pilhas AA em série.

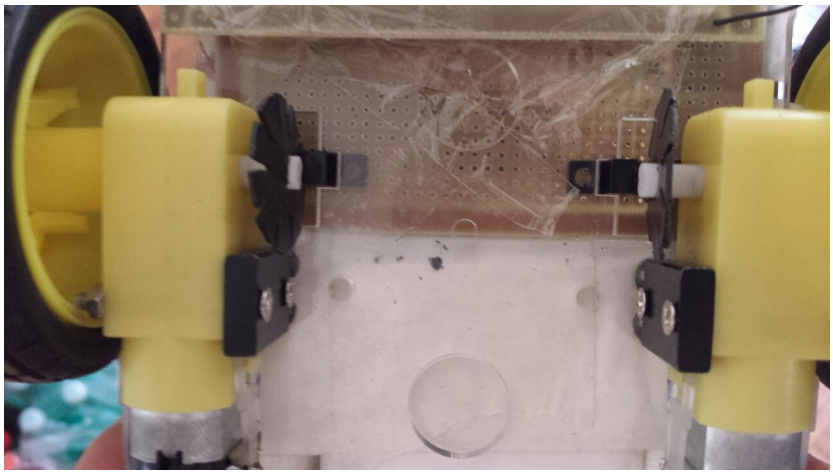

Figura 16: Motores e enconder

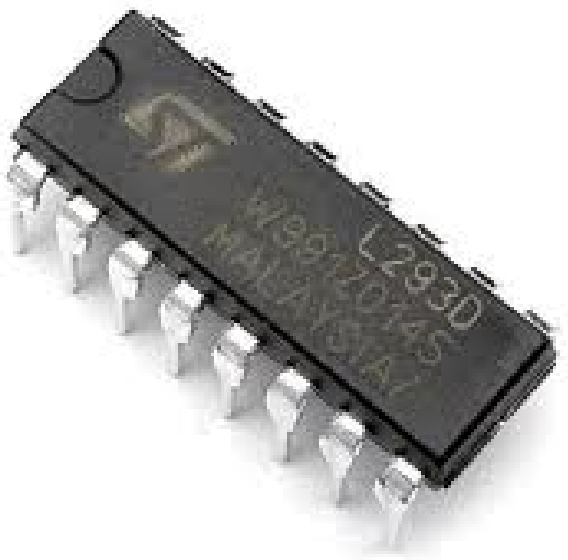

Figura 17: Ponte H L293

A ponte H permite o usuário controlar a velocidade e a direção de rotação dos motores. Esse controle é feito através de dois pinos GPIO e um sinal de PWM para cada motor. Através dos pinos GPIO, conectados esses aos pinos 1/3A e 2/4A da ponte H, selecionamos a direção de rotação e com o sinal PWM, conectado no pino enable, definimos a velocidade. Desta forma foi selecionado as seguintes portas para o motor 1: PTC4 e PTC7 como GPIO e PTA13 (configurado como FTM1_CH1) como PWM, já para o motor 2: PTC0 e PTC3 como GPIO e PTA12 (configurado como FTM1_CH0) como PWM. As saídas 1Y e 2Y da ponte H foram conectadas à alimentação do motor 1 e as saídas 3Y e 4Y à alimentação do motor 2. O diagrama lógico das entradas e saídas do IC pode ser visto na figura 18

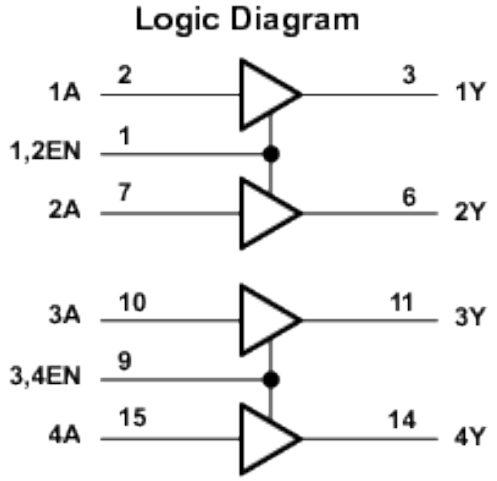

Figura 18: Diagrama Lógico da Ponte H L293 [1]

Para a montagem eletrônica dos encoders utilizamos o esquemático da figura 19 como referência. A tensão de alimentação foi dimínuída para $3.3V$ e as resistências foram dimensionadas de acordo com a datasheet [2], assumindo os mesmos valores utilizados na placa de sensores. A saída dos encoders foram conectadas nas portas PTE29 e PTE30, configuradas como FTM_CLKIN0 e FTM_CLKIN1 respectivamente, dessa maneira utilizamos essas duas portas como o clock de referência para os módulos TPM2 e TPM0.

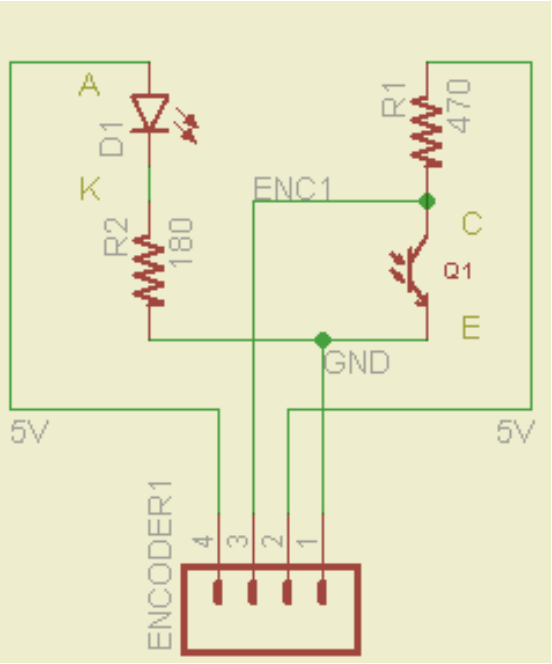

Figura 19: Diagrama esquemático para placa de encoder

5 Teste de funcionalidade

Uma parte importante do projeto é testar todos os componentes e ligações elétricas.

Ao iniciar a execução da rotina a primeira etapa do sistema é a análise da funcionalidade dos componentes, sensores e atuadores. Primeiramente é feito o teste dos infra-vermelhos, com o emissor desligado, capta-se alguns valores do receptor, gerando uma gama de dados de comparação com a etapa seguinte de emissor ligado, caso não haja real diferença entre os dois, este sistema está com defeito e deve ser averiguado. Para visualização do usuário, o LED do target se acende com uma cor determinada pelo infra-vermelho com falha.

Após este teste há o teste dos motores. O teste dos motores é baseado na medição dos encoders, o motor é ligado e os encoders inicializados, caso não haja medição, ou o encoder está com defeito, ou o motor é que está. Neste caso é mais fácil para o usuário verificar o defeito já que o robô se movimentará ou não. Caso haja falha do encoder, o LED do target se acenderá com uma cor definida pelo encoder com falha.

6 Calibração dos sensores

Feito os testes dos sensores e atuadores, é necessário realizar a calibração dos mesmos. A calibração é a definição dos valores máximos e mínimos que o sistema atuará, gerando dessa forma parâmetros que dependem de situações externas ao do robô, como por exemplo, baixa bateria, diferentes tipos de superfície e baixa ou alta luminosidade.

Como, no nosso caso, a calibração dos sensores dependerá da funcionalidade dos motores, a calibração começa pelo motor. A calibração do motor é realizado da seguinte forma: Coloca-se o valor máximo de PWM, aguarda alguns segundos e verifica-se o valor do encoder em pulsos por milissegundo. Feito isso reduz-se gradativamente o valor do PWM até a parada do robô, tal PWM é o mínimo necessário para movimentar o robô. Com tais valores podemos realizar o controle de velocidade compatível com o que o motor pode realizar. Esse procedimento é realizado nos dois sentidos para os dois motores.

A calibração dos infra-vermelhos é realizada pelas seguintes etapas: com o emissor desligado, mede-se o valor do receptor em cima da pista clara, esse valor é um offset do sistema e deve ser retirado. Com o robô ainda em cima da pista, liga-se um dos motores, fazendo o robô girar no próprio eixo, o robô passará tanto pela superfície clara, quanto pela escura, basta então armazenar os valores máximos e mínimos para definir o intervalo de operação. Faz-se o mesmo, só que rodando no sentido oposto. O intervalor é único para cada infra-vermelho.

7 Detecção de linha

Para seguir uma linha ou caminho é necessário inicialmente encontrar tal caminho e definir sua posição relativa à este. Pela largura do caminho e pelo distanciamento físico dos infra-vermelhos, haverá sempre 2 infra-vermelhos detectando a linha caso o robô esteja em cima da mesma (a resposta dos sensores varia conforme a sua posição relativa a linha, caso em cima, a resposta será baixa).

Possuindo os valores dos 6 sensores é necessário agora encontrar a real posição relativa do robô, para tal realizamos uma curva de aproximação. A opção de solução foi o algoritmo Catmull-Spline [3]. Repetimos a primeira e última medida para gerar os pontos inicial e final e, uma vez ajustado o polinômio, resolvemos a equação da derivada para a determinação do ponto mínimo em cada intervalo, encontrando por fim o mínimo global da curva. O algoritmo foi

implementado primeiramente em Matlab para testes e alguns de seus resultados podem ser vistos na figura 20.

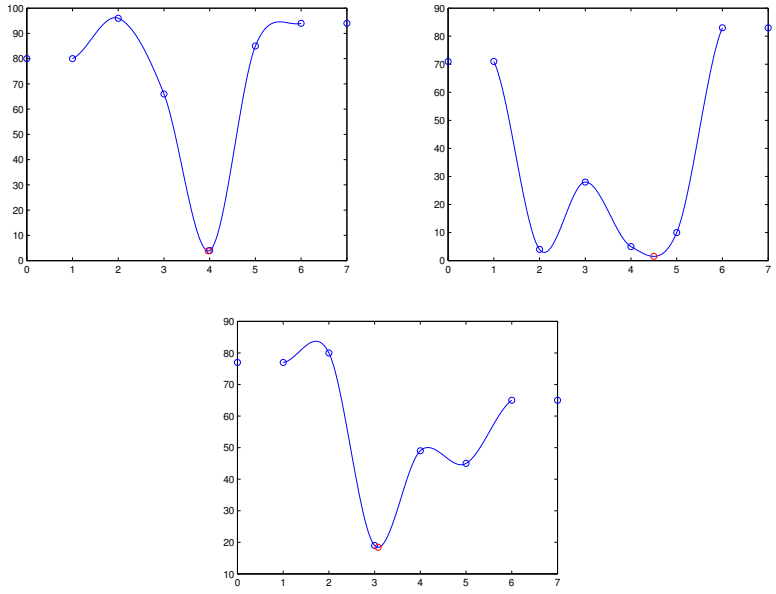

Figura 20: Resultados do algoritmo de spline simulado via Matlab

A distância entre o centro do robô até o sensor mais externo é 25mm, desta forma definimos que a posição relativa do robô varia entre -25 e 25, sendo 0 o centro. Caso o robô não esteja em cima da linha o valor de posição relativa permanece constante.

8 Detecção de comando

Para a detecção de comandos consideramos que quando mais do que 5 dos foto-sensores estão medindo preto é porque um sinal de comando foi encontrado, para eliminar ruídos, só levamos em consideração sinais de comando se eles foram medidos mais do que duas vezes consecutivas. Quando detectamos o primeiro sinal de comando o robô passa a procurar por um segundo, se ele percorrer uma distância pré definida sem encontrar este ele considera que recebeu o comando de velocidade máxima e aumenta sua velocidade linear de referência até que o próximo comando seja encontrado, caso contrário ele considera que recebeu o

comando de parada, nesse caso ele anda uma distância pré definida e depois para por um tempo também pré definido.

9 Controle de velocidade

Com a calibração dos motores e funcionalidade do encoder podemos realizar um controle de malha fechada para os dois motores separadamente afim de definir uma velocidade de cruzeiro.

A calibração nos serve para traçar uma curva de funcionalidade dos motores, verificamos qual é a velocidade máxima de cada motor e em cada direção. Como os motores são diferentes os limites máximos são diferentes, assim como o PWM mínimo para iniciar o movimento, o controlador deve atuar de forma a fornecer 2 PWMs, possivelmente diferentes, que geram a mesma velocidade linear, desta forma, movimentando o robô em linha reta.

O controlador realizado foi baseado em um PI. Os ganhos proporcionais e integrais foram encontrados com base em testes empíricos. Para definir o ganho proporcional foi verificado o tempo de resposta, via comunicação serial entre host-target, atingido um tempo satisfatório. A próxima etapa é encontrar o valor do ganho integral, para este foi analisado o erro estacionário.

Os ganhos do controlador de velocidade são iguais para os dois motores. Apesar da diferença dos motores, como o sistema não precisa de uma precisão grande, os controladores iguais são satisfatórios.

10 Controle de linha

Com a calibração dos sensores infra-vermelhos e com o Catmull-Spline funcionando corretamente, podemos realizar o controle de posição do robô em relação a linha.

Possuindo a posição relativa do centro do veículo ao centro da linha, nos basta incrementar ou decrementar um dos PWM das rodas, gerando um movimento angular. O controle de posição define o quanto deve ser essa rotação.

Para o controle de posição foi realizado um controlador P. O coeficiente foi encontrado de forma empírica com base na velocidade de resposta dos motores. Neste caso, o tempo de resposta é importante para que o veículo não perca o caminho, desta forma, o proporcional é de grande importância para o sistema.

Caso o veículo perca o caminho, o mesmo buscará o caminho rotacionando para a última posição armazenada.

11 Referências

[1] Texas Instruments - Datasheet: L293x Quadruple Half-H Drivers

[2] Photonic - Datasheet: CHAVES OPTOELETRÔNICAS TRANSMISSIVAS - PHCT102/3/4, PHCT202/3/4

[3] E. Catmull and R. Rom - A class of local interpolating splines