UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

PEDRO HENRIQUE OLIVEIRA DE PAULA

CONSTRUÇÃO E CONTROLE DE UM ROBÔ HEXÁPODE USANDO O TLC5943

VITÓRIA – ES
DEZEMBRO/2014

PEDRO HENRIQUE OLIVEIRA DE PAULA

CONSTRUÇÃO E CONTROLE DE UM ROBÔ HEXÁPODE USANDO O TLC5943

Parte manuscrita do Projeto de Graduação do aluno **Pedro Henrique Oliveira de Paula**, submetida ao Colegiado de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para a obtenção do Grau de Engenheiro Eletricista.

VITÓRIA – ES
DEZEMBRO/2014

PEDRO HENRIQUE OLIVEIRA DE PAULA

CONSTRUÇÃO E CONTROLE DE UM ROBÔ HEXÁPODE USANDO O TLC5943

Comissão Examinadora:

_____
Profa. Dra. Raquel Frizera Vassallo
Orientadora

_____
Eng. Philipe Rangel Demuth
Co-orientador

_____
Prof. Dr. Evandro Ottoni Teatini Salles
Avaliador

_____
Prof. Dr. Hans Jorg Andreas Schneebeli
Avaliador

VITÓRIA – ES
DEZEMBRO/2014

AGRADECIMENTOS

Em primeiro lugar, quero agradecer à Deus: por esta existência neste planeta incrível, por todas as oportunidades a mim concedidas nesta vida e também pelas pessoas maravilhosas que cada vez mais Ele coloca no meu caminho. Muito obrigado. Também quero agradecer à minha família, em especial à minha mãe, Anelisa, pelos valores ensinados ao longo desses 23 anos e também por todo apoio, que sempre me foi dado incondicionalmente. Ainda quero agradecer aos meus companheiros nesta jornada: aos funcionários do CT-II, por sempre me tratarem com carinho e amizade. Aos mestres, em especial à Raquel Frizera Vassallo, Marcelo Eduardo Vieira Segatto e Moisés Renato Nunes Ribeiro, por todos os ensinamentos, principalmente aqueles dados fora da sala de aula. Aos meus colegas de classe, por fazerem as minhas manhãs serem um pouco menos desgastantes. À galera do PND, pelas discussões caóticas e a alegria diária. Aos amigos de sempre, Carlos Henrique Oliveira de Oliveira, Lucas Avanci Gaudio, Murilo Porto Amaral e Philipe Rangel Demuth, pelo companheirismo, pelas conversas e por toda a ajuda nesta difícil jornada, muito obrigado! Por último, e muito importante, quero agradecer à todos os amigos que esta vida maravilhosa já me deu. Muito obrigado pela amizade, pelos momentos inesquecíveis, pela compreensão e pela sabedoria dos pequenos detalhes. Eu sou, sinceramente, muito grato à todos vocês.

RESUMO

Este trabalho apresenta a construção de um robô hexápode, englobando a construção de sua estrutura, projeto e confecção da placa de circuito impresso, programação do firmware e testes de movimento. O protótipo foi o segundo a ser elaborado na UFES e deve ser encarado como a continuidade do trabalho iniciado pelo engenheiro Philipe Rangel Demuth. A construção das peças do robô foi realizada em impressão 3D. A nova placa de circuito impresso inclui um novo controlador de PWM, o TLC5943, e uma nova plataforma de desenvolvimento, a EK-LM4F120XL, que contém um microcontrolador ARM Cortex-M4, o LM4F120H5QR. O firmware programado utiliza um sistema operacional de tempo real como base, a fim de facilitar a implementação das tarefas necessárias para o controle do robô. Um teste com movimentos de caminhada e giro foi realizado para validar os resultados alcançados.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

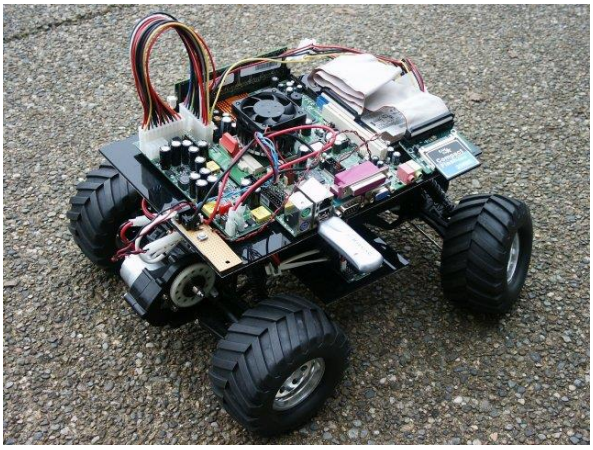
ARM	*Advanced RISC Machine*
CAN	*Controller Area Network*
CNC	Controle Numérico Computadorizado
I²C	*Inter-Integrated Circuit*
ICDI	*In-Circuit Debug Interface* (Interface de Debug Integrada)
LAI	Laboratório de Automação Inteligente
PWM	*Pulse-Width Modulation* (Modulação por Largura de Pulso)
PCI	Placa de Circuito Impresso
PLL	*Phase-Locked Loop* (Malha de Captura de Fase)
RTOS	*Real-Time Operating System* (Sistema Operacional de Tempo Real)
UART	*Universal Asynchronous Receiver/Transmitter* (Receptor/Transmissor Assíncrono Universal)
UFES	Universidade Federal do Espírito Santo
USB	*Universal Serial Bus* (Barramento Serial Universal)

SUMÁRIO

1 INTRODUÇÃO

A robótica é um campo de estudos vasto e está inserida em diversos contextos da vida humana. Suas aplicações podem ser domésticas, comerciais, industriais, militares, medicinais e até sociais. Muitas vezes, o uso de robôs remotos se faz necessário em atividades perigosas ou irrealizáveis por seres humanos, como desarmamento de bombas, detecção de minas, exploração de cavernas e naufrágios, entre outros. Nestes casos, existem diversos parâmetros que podem limitar a ação de um determinado robô, como o terreno em que ele irá se locomover.

Figura 1- Robô com rodas

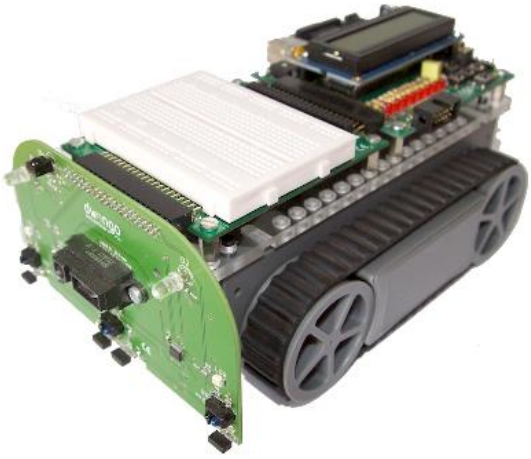


Fonte: TANTOS.

Sabe-se que os robôs que utilizam rodas para a locomoção (Figura 1) possuem um projeto mais simples em estrutura e controle do movimento pois, em superfícies planas, todas as suas rodas estão em contato com o chão, fato que permite a desconsideração do balanço do veículo em sua modelagem. Contudo, em superfícies irregulares as rodas podem não conseguir a aderência considerada no projeto fazendo com que a locomoção não ocorra como esperado.

Duas opções possíveis para operar neste tipo de terreno são robôs com patas ou com esteiras, representados na Figura 2. Apesar da utilização de esteiras simplificar o controle do movimento e viabilizar a locomoção em terrenos de grande desnível, ela não garante a transposição de obstáculos grandes como uma escada, raiz de árvore ou escombros de uma construção. Neste quesito, os robôs com patas apresentam melhor desempenho, além de serem importantes pela sua capacidade de regulação da estabilidade e de atingir alta eficiência energética (WETTERGREEN, 1996).

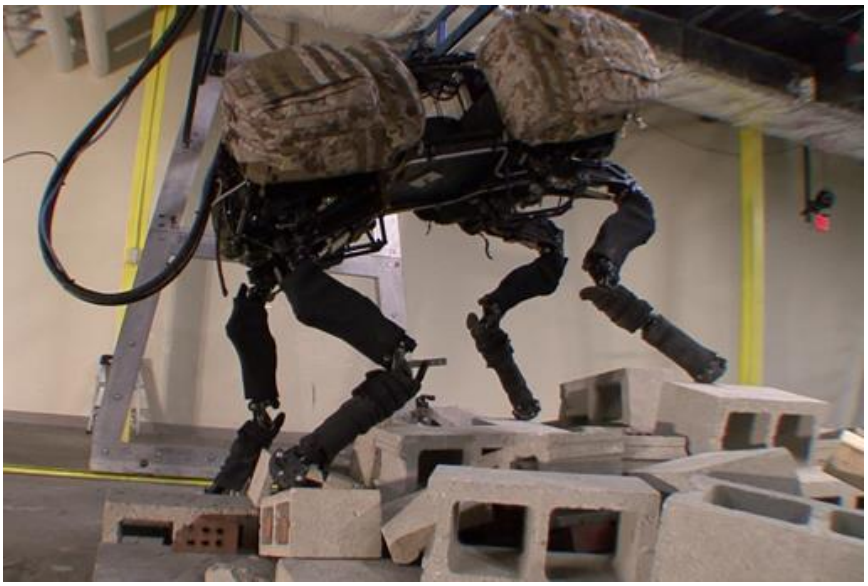
Figura 2 - Robô com patas e robô com esteiras



Fonte: ANDY e DWENGO.

Um robô com patas de sucesso é o BigDog, produzido pela empresa americana Boston Dynamics. Ilustrado na Figura 3, o BigDog é um quadrúpede que consegue correr até 6 quilômetros por hora, subir inclinações de até 35 graus e carregar cerca de 150 quilogramas (BOSTON DYNAMICS, 2013). Por possuírem menos motores, os robôs com até quatro patas possuem um sistema de acionamento menos complexo do que aqueles com mais patas, porém essa categoria de robôs não consegue se locomover com técnicas estáticas, isto é, técnicas que consideram o robô sempre em equilíbrio estático.

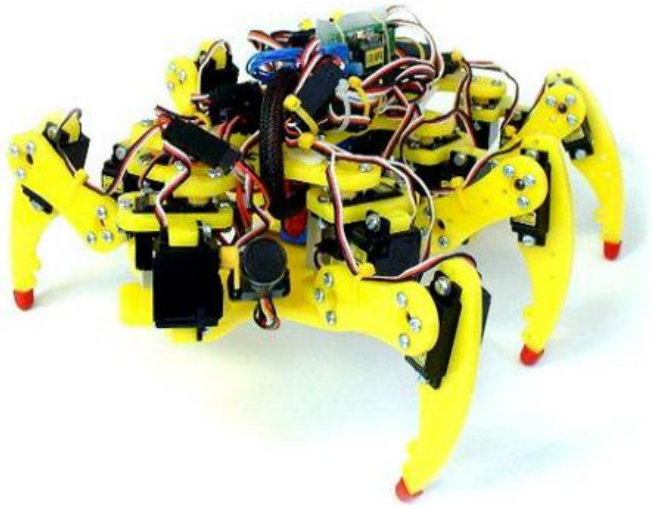
Figura 3 - BigDog



Fonte: BOSTON DYNAMICS.

No entanto, robôs com seis ou mais patas, possuem um número de apoios suficiente para que não haja grande variação do centro de massa durante o seu deslocamento, o que isenta a implementação de técnicas avançadas de controle. Entre eles estão os robôs hexápodes que possuem seis patas e são inspirados em insetos. Um exemplo desse tipo de robô pode ser visualizado na Figura 4.

Figura 4 - Robô hexápode



Fonte: MICROMAGIC SYSTEMS.

Na UFES, o estudo com robôs hexápodes mais recente foi o do Eng. Philipe Demuth, que desenvolveu o seu próprio protótipo. O trabalho deste aluno foi pioneiro na área dentro da universidade e ao seu fim foram sugeridas diversas melhorias para o protótipo.

Dentre as melhorias sugeridas, destacam-se a utilização de um sistema operacional de tempo real para facilitar a multiplexação dos comandos de acionamento dos servo-motores, a separação da fonte de alimentação do microcontrolador e dos servos e a modelagem cinemática do robô para auxiliar na execução do movimento de rotação (DEMUTH, 2013).

Essas sugestões foram estudadas e implementadas no desenvolvimento deste trabalho. Além disso, também foram avaliados outros meios de corrigir as deficiências do protótipo anterior, a fim de confeccionar um novo robô que seja capaz de se locomover com autonomia em terrenos irregulares.

1.1 Justificativa

A motivação para o desenvolvimento deste projeto é composta por dois motivos principais:

- A primeira é a possibilidade de continuação de um projeto trabalhoso e pioneiro nos nossos laboratórios. Este caráter de continuidade não só é importante pela otimização dos recursos existentes e pela valorização do conhecimento adquirido pelos antecessores como também pela oportunidade de agregar ainda mais valor ao projeto.

- A segunda influência na motivação deste projeto é o próprio campo de estudo: a robótica. Se trata de um trabalho importante, inovador e muito motivante.

1.2 Objetivo

O objetivo geral deste trabalho é construir um robô hexápode que seja capaz de se locomover em superfícies irregulares, nas quais robôs com rodas ou com esteiras não seriam capazes de navegar.

Neste trabalho espera-se alcançar os seguintes objetivos específicos:

- Definir um novo projeto de estrutura, que minimize a demanda por torque nos motores;
- Projetar os circuitos eletrônicos do robô;
- Construir o robô;
- Desenvolver *software* embarcado que descreva as funções básicas de movimento: caminhada e giro;
- Testar, ajustar e validar o *software* desenvolvido. Os testes finais incluirão caminhada em linha reta, giro em 90 graus e transposição de obstáculos, como uma raiz de árvore.

1.3 Metodologia

Este trabalho consistiu na construção de um protótipo. Para isto foram necessárias cinco etapas de desenvolvimento: pesquisa, projeto, construção, validação e documentação.

Pesquisas foram realizadas tanto para escolher uma nova estrutura para o robô, quanto para projetar o circuito eletrônico e programar a lógica envolvida no controle do robô. A etapa de

projeto incluiu a especificação da estrutura, bem como a criação de seu desenho no *software* adequado, e a especificação do circuito eletrônico que será embarcado no hexápode.

A fase de construção foi executada na parte final do projeto, quando todas as peças impressas foram montadas e o *firmware* foi programado. Além disso, a etapa de validação foi executada em paralelo com a construção, visto que montar o robô e programar rotinas envolvem uma série de testes em cada parte do todo.

Por fim, a etapa de documentação foi constituída principalmente pela redação deste relatório final. Contudo, durante toda a execução do trabalho foram tomadas notas e registros de informações e eventos relevantes ocorridos durante o desenvolvimento.

1.4 Estrutura do texto

O projeto do robô hexápode pode ser dividido nos projetos da estrutura, da eletrônica e do *firmware*. O texto deste trabalho segue o mesmo raciocínio: após uma breve introdução ao tema, no Capítulo 1, os próximos capítulos irão apresentar de forma detalhada essas três componentes do projeto.

O Capítulo 2 é dedicado ao projeto da estrutura: em primeiro lugar a estrutura desenvolvida por Demuth (2013) será apresentada e em seguida o novo projeto desenvolvido neste trabalho. No Capítulo 3, o circuito eletrônico será apresentado, destacando as diferenças entre o novo projeto e o antigo. O circuito esquemático e o projeto da placa estarão disponíveis ao final do capítulo.

Em seguida, o Capítulo 4 detalha o projeto do *firmware* desenvolvido. Inicia-se com uma breve explicação do sistema operacional de tempo real escolhido e, logo após, apresentam-se as funções programadas. O Capítulo 5, Resultados Experimentais, trata dos testes de validação do protótipo construído e o Capítulo 6, da conclusão do trabalho.

2 ESTRUTURA DO ROBÔ HEXÁPODE

A estrutura de um robô hexápode é inspirada no corpo e nas pernas dos insetos. Na natureza existe uma grande variedade de insetos e, consequentemente, diversas formas diferentes de corpos e pernas. De maneira geral, o corpo de um inseto pode ser dividido em três partes: cabeça, tórax e abdome. Suas pernas também podem ser divididas em três partes: coxa, fêmur e tíbia.

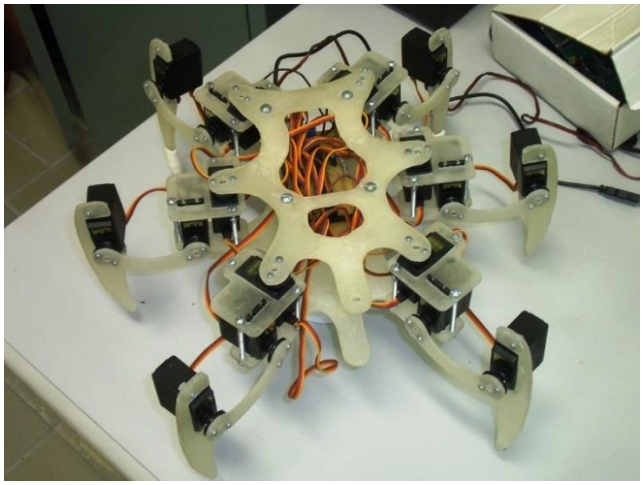
Os robôs construídos com base na morfologia dos insetos assumem as mais variadas combinações de segmentos do corpo e das pernas. Neste projeto, a estrutura adotada possui um corpo com apenas um segmento e seis pernas, cada uma composta pelos três segmentos apresentados anteriormente. Além disso, um encaixe para a cabeça foi disponibilizado, para que em projetos futuros esse segmento do corpo possa ser adicionado.

Neste capítulo, a primeira seção será destinada à apresentação da estrutura construída por Demuth (2013), o objeto de estudo inicial deste trabalho. Na segunda seção um novo projeto será apresentado junto com as justificativas de sua superioridade frente ao anterior.

2.1 Estrutura do projeto original

A estrutura original do robô hexápode possui um corpo composto de apenas um segmento e seis pernas compostas de três segmentos e três articulações, que tem o seu movimento controlado por um servo-motor cada.

Figura 5 - Robô desenvolvido por Demuth (2013).



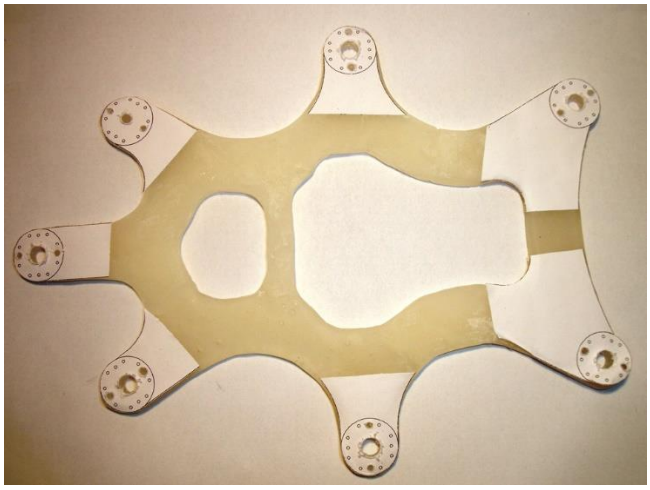
Fonte: DEMUTH, 2013.

A Figura 5 mostra o robô desenvolvido por Demuth (2013). Nela pode-se observar as peças que formam o robô: a parte do corpo à qual as seis pernas estão conectadas, a parte inferior do corpo que carrega a placa de circuito impresso e os três segmentos que compõem cada perna.

Uma característica deste projeto é que todas as peças foram confeccionadas artesanalmente com resina acrílica. Por isso, todas elas são superfícies planas o que limita bastante as possibilidades na construção como, por exemplo, encaixes em três dimensões. Neste projeto estrutural, foram confeccionadas vinte e seis peças, sendo uma parte superior e outra inferior do corpo, seis partes superiores da coxa e seis partes inferiores da mesma, seis fêmures e seis tíbias.

A peça superior do corpo do antigo robô pode ser visto na Figura 6. Esta peça possui sete encaixes para servos, sendo os seis divididos simetricamente para as pernas e o outro, na ponta do corpo, para a cabeça. Além disso a peça também possui duas aberturas em sua parte interior para facilitar o manuseio dos componentes que ficarão alojados dentro do corpo. A peça inferior do corpo possui a mesma silhueta da parte superior, porém, essa não passa apenas de uma peça sólida, sem furos ou aberturas.

Figura 6 - Corpo do robô desenvolvido por Demuth (2013).



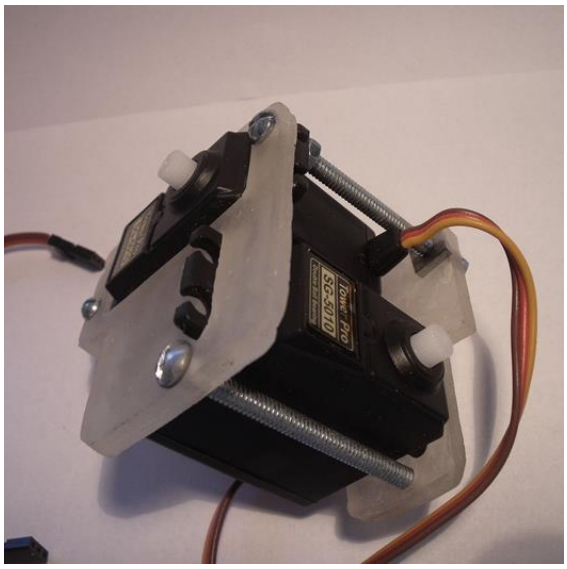
Fonte: DEMUTH, 2013.

A coxa também é dividida em duas partes, exibidas na Figura 7. Essas peças precisam unir e dar estabilidade ao conjunto formado por ambas e pelos dois servo-motores responsáveis por movimentar as articulações envolvidas. O conjunto completo pode ser visto na Figura 8.

Figura 7 - Peças que compõem a coxa do robô original.

Fonte: DEMUTH, 2013.

Figura 8 - Coxa do robô original montada.



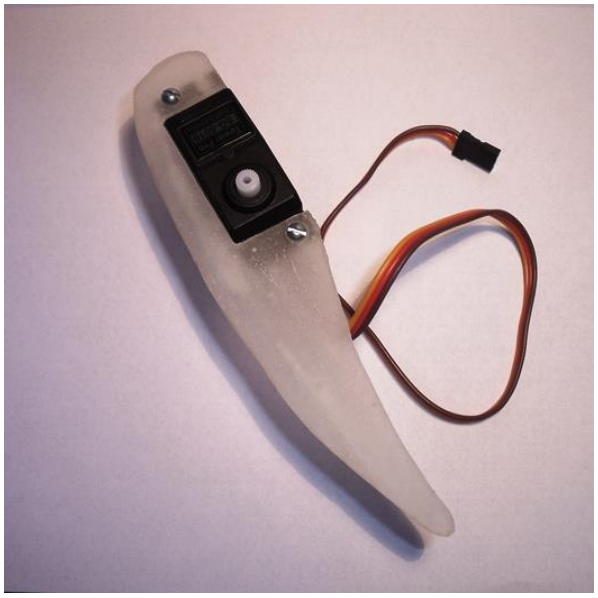
Fonte: DEMUTH, 2013.

Figura 9 - Fêmur do robô desenvolvido por Demuth (2013).

Fonte: DEMUTH, 2013.

O fêmur nada mais é que uma peça simples que possui apenas dois encaixes para servos em suas extremidades. O fêmur do projeto original pode ser visto na Figura 9. Nesta figura, o fêmur ainda não possui os furos para o encaixe dos servos.

Figura 10 - Conjunto tíbia e servo-motor.



Fonte: DEMUTH,2013.

A tíbia (Figura 10) também é uma peça simples que possui apenas um encaixe para fixação de um servo. A tíbia é a parte do robô que toca a superfície e, por isso, tem o desenho feito para essa função.

O projeto original da estrutura do hexápode apresentou fragilidades em alguns aspectos. Por exemplo, a utilização de apenas um fêmur para a conexão da tíbia com a coxa não estabelece uma conexão rígida o suficiente e, consequentemente, os servos conectados ao fêmur estavam sofrendo a ação de forças paralelas ao seu eixo.

Além disso, as articulações que conectam o corpo às coxas são apenas o encaixe do eixo dos servos com o corpo. Desta maneira, quando o robô está montado, a ação da gravidade faz com que estas articulações entortem, já que as pernas precisam sustentar o peso do corpo.

Por fim, a resina acrílica é um material de densidade maior que outras opções disponíveis como a placa de acrílico e os plásticos ABS e PLA. A substituição deste material poderia amenizar a demanda por torque nos servos, um dos problemas encontrados no trabalho anterior.

2.2 Novo projeto estrutural

O novo projeto estrutural do robô segue o mesmo modelo do original. Apesar de todas as peças terem sido redesenhadas, a aparência do robô não foi alterada de maneira significativa. As principais mudanças foram realizadas para amenizar ou eliminar as fragilidades da estrutura original, expostas anteriormente.

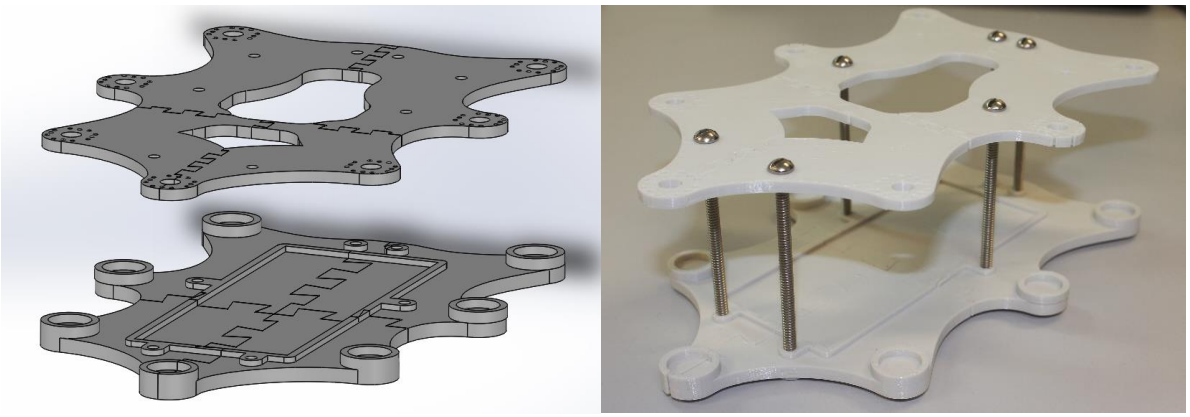
Ao todo foram confeccionadas quarenta e quatro peças: as partes superior e inferior do corpo, as partes superiores e inferiores das seis coxas, doze fêmures, seis tíbias e doze tampas para s servo-motores. Desta vez, a maioria das peças foram projetadas em três dimensões para viabilizar e facilitar encaixes. Todas as peças foram confeccionadas com uma impressora 3D, sendo as peças que compõem as pernas impressas em plástico ABS e as peças que compõem o corpo em plástico PLA.
Esta diferença nos materiais das peças das pernas e do corpo ocorreu pela disponibilidade dos materiais em questão. Entretanto, a estrutura não foi comprometida pois, apesar do plástico PLA ser menos flexível do que o ABS, o corpo do robô não corre o risco de quebra, já que ele não sofre à ação de forças de torção.

Utilizando software apropriado (*SolidWorks*), todas as peças foram desenhadas e, em seguida, combinadas em montagens. As peças desenhadas no *SolidWorks* podiam ser exportadas para arquivos com o formato aceito pela impressora 3D (.stl). As montagens foram feitas para a

visualização prévia do robô, o que permitiu a realização de correções no projeto antes mesmo da impressão das peças.

O corpo da nova estrutura foi projetado com os seguintes objetivos: aumentar a área interior do corpo para acomodação da nova placa de circuito impresso e disponibilizar encaixe para a coxa nas duas extremidades do eixo da articulação corpo-coxa. A montagem do corpo pode ser vista na Figura 11, sendo a imagem à esquerda uma vista obtida no *SolidWorks* e a imagem à direita uma foto das peças impressas.

Figura 11 - Montagem do corpo do robô. À esquerda, vista do SolidWorks e, à direta, foto das peças impressas.



Fonte: Produção do próprio autor.

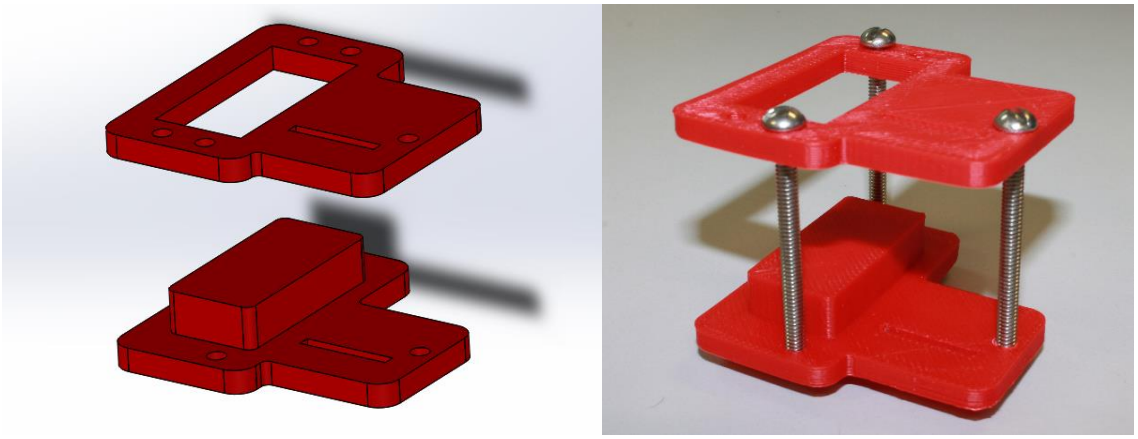
Observa-se na Figura 11 que a parte superior do corpo não sofreu grandes alterações. A posição dos parafusos foi modificada, as dimensões da placa foram estendidas e o seu formato foi levemente alterado. A parte inferior sofreu essas mesmas alterações e ainda recebeu calços para encaixe do fundo da coxa, em cada perna, e também uma cavidade para prevenir a placa de circuito impresso de deslizar enquanto o robô estiver em movimento.

Finalmente, nota-se que cada uma das partes do corpo é feita a partir do encaixe de quatro peças. Isto foi necessário por causa de uma limitação da impressora 3D utilizada. O volume de trabalho da impressora é um cubo de 12x12x12cm e cada uma dessas peças mede 22,2x15cm, por isso foi necessária a divisão em quatro partes.

O novo projeto da coxa foi feito visando um melhor encaixe de todo o conjunto e um encaixe consistente com a estrutura projetada para o corpo. No projeto antigo, a parte de trás do servo-motor da articulação corpo-coxa não está apoiada na parte inferior da coxa. Por isso, ao apertar

os parafusos e também durante o funcionamento do robô esta peça está sujeita à flexão, o que é um problema a longo prazo.

Figura 12 - Montagem da coxa do robô. À esquerda, vista do SolidWorks e, à direta, foto das peças impressas.

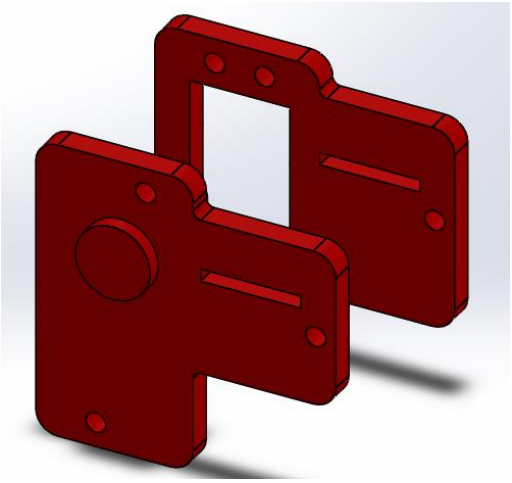


Fonte: Produção do próprio autor.

A Figura 12 mostra uma montagem da coxa sem os servos e uma foto dessa montagem feita com as peças impressas. Observa-se que na peça inferior um calço foi inserido, para que o servo da articulação corpo-coxa tenha um apoio e a montagem não fique comprometida.

A fim de garantir um encaixe consistente, um pino foi acrescentado no fundo da coxa (Figura 13). Desta maneira, a articulação corpo-coxa possui um encaixe tanto na parte superior do corpo, o próprio eixo dos servos, quanto na parte inferior do corpo, garantindo uma variação angular mínima do seu eixo.

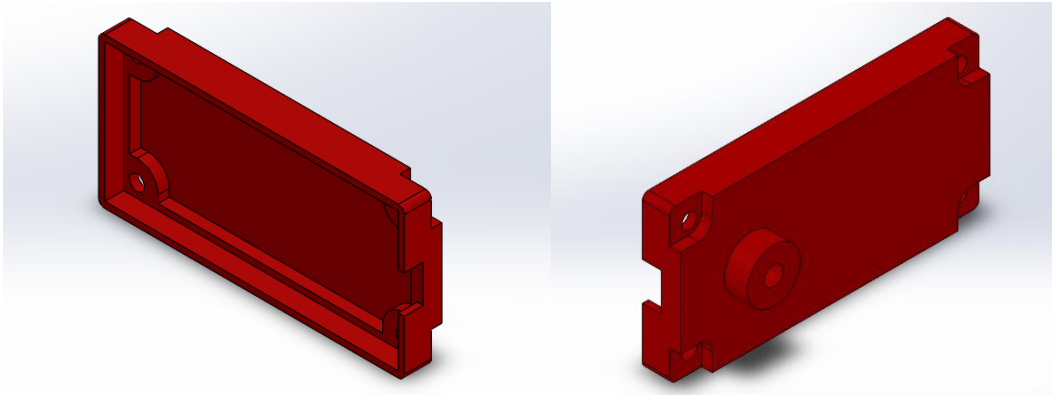
Figura 13 - Vista inferior da coxa, com destaque para o pino acrescentado.



Fonte: Produção do próprio autor.

O fêmur da nova estrutura do robô foi projetado para diminuir a flexão neste segmento da perna. Isto foi realizado com a duplicação do fêmur original, isto é, uma peça semelhante ao fêmur anterior foi inserida, conectando os mesmos dois eixos que o fêmur original porém, desta vez, pela parte de trás dos servos. Para aplicar este conceito foi necessária a inserção de um pino na parte de trás dos servos, onde o novo fêmur está encaixado.

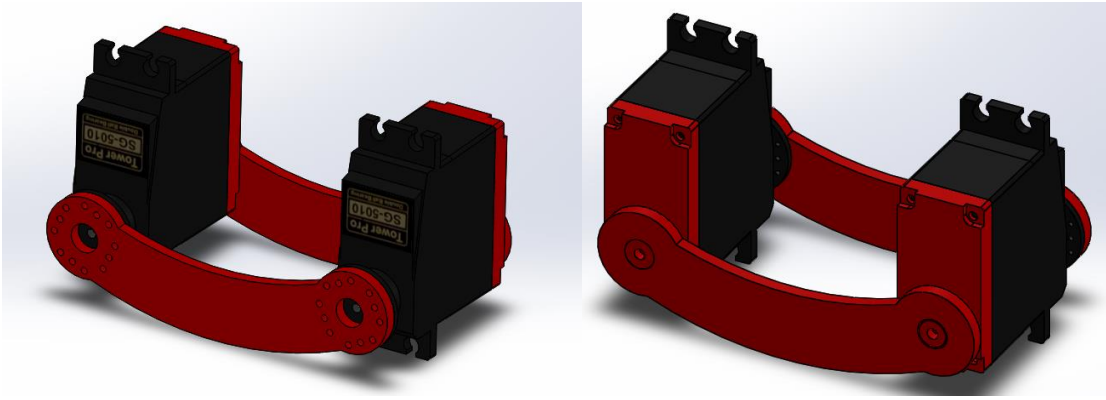
Figura 14 - Desenho da nova tampa para os servo-motores no *SolidWorks*.



Fonte: Produção do próprio autor.

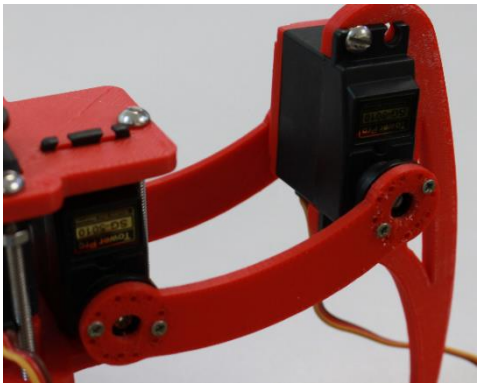
A fim de construir o pino de forma robusta, decidiu-se substituir as tampas traseiras dos servos por novas tampas impressas em 3D e já com o pino acoplado. A tampa foi medida com o paquímetro e o seu modelo desenhado no *SolidWorks*, onde sofreu alterações posteriormente. A Figura 14 mostra duas vistas da tampa projetada e a Figura 15 mostra a montagem do novo fêmur no software, em uma vista frontal e outra traseira. Uma foto do fêmur construído pode ser vista na Figura 16.

Figura 15 - Montagem do fêmur do robô. À esquerda, vista frontal do fêmur e, à direita, vista traseira.



Fonte: Produção do próprio autor.

Figura 16 – Fêmur do robô construído.



Fonte: Produção do próprio autor.

Por último, o projeto da nova tíbia foi bem simples e não envolveu características críticas ao projeto. As alterações foram feitas apenas para remodelar o formato do segmento e também garantir espaço para encaixe das roscas dos parafusos. A peça desenhada no *SolidWorks* e a peça impressa podem ser vistas na Figura 17.

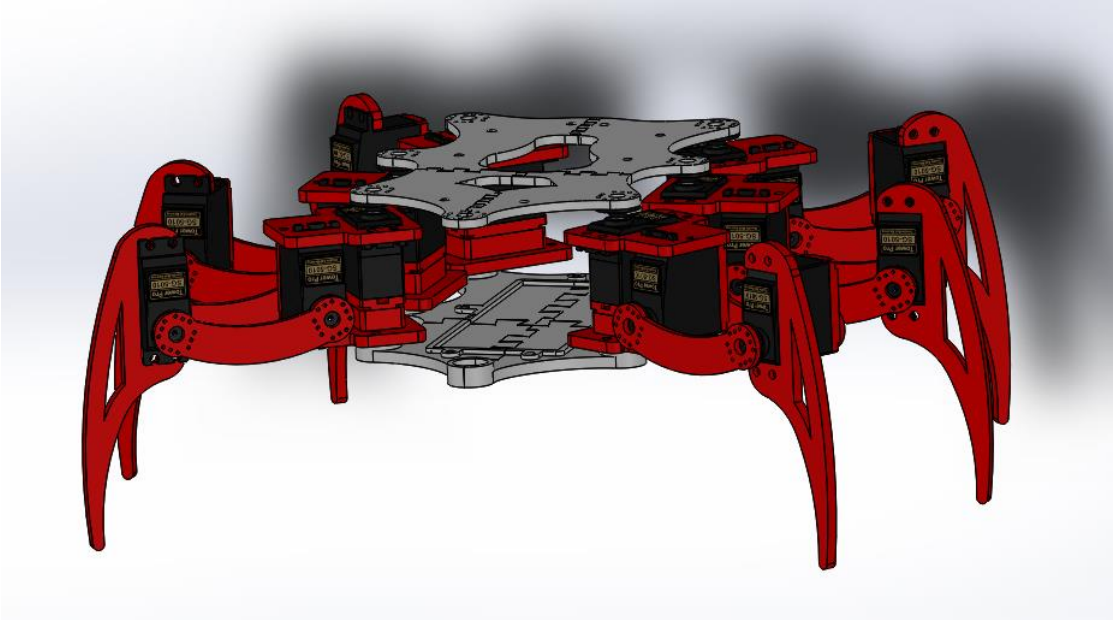
Figura 17 - Montagem da tíbia do robô. À esquerda, vista do SolidWorks e, à direta, foto das peças impressas.

Fonte: Produção do próprio autor.

A Figura 18 mostra a montagem completa da nova estrutura do robô no *SolidWorks* e a Figura 19 mostra o robô após sua construção completa. Este novo projeto elimina as fragilidades citadas anteriormente pois garante que as articulações não sofrerão variações em seus eixos e também garante que os servos não serão exigidos em direções que eles não foram projetados para atender.

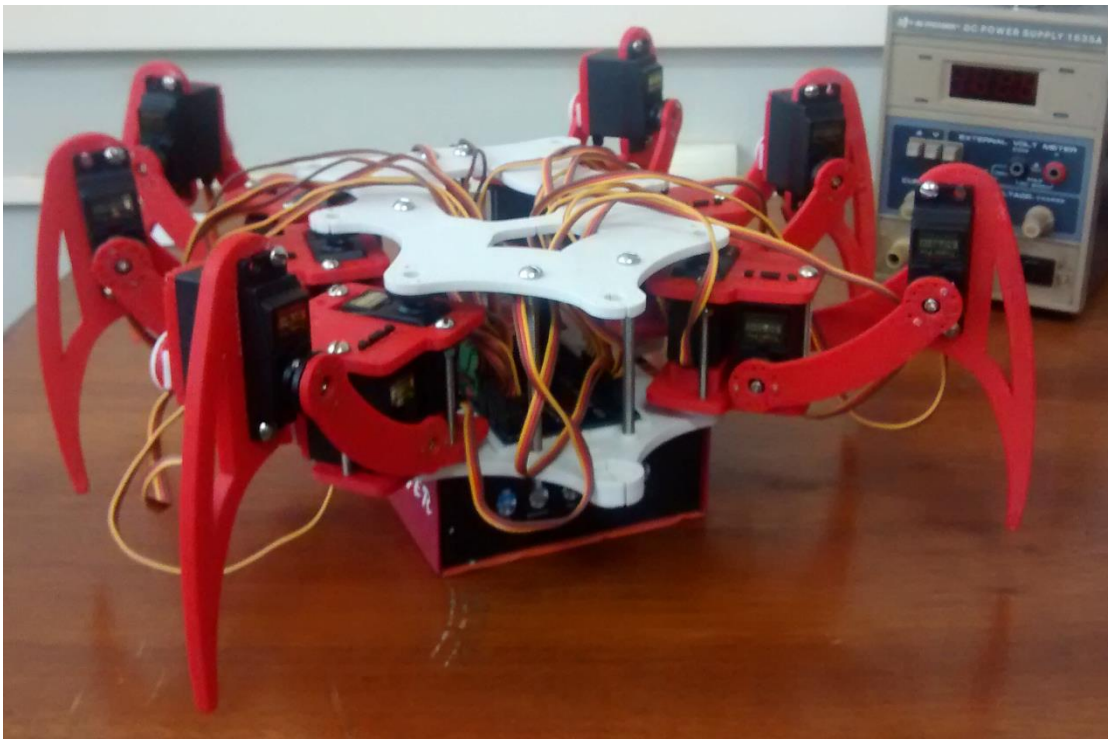
Ainda mais, a confecção das peças em impressora 3D diminuiu a massa da estrutura pois, utilizou-se material menos denso e também porque o interior das peças é preenchido com estruturas do tipo colmeia e não de maneira maciça.

Figura 18 - Montagem completa do robô no *SolidWorks*.



Fonte: Produção do próprio autor.

Figura 19 - Robô construído.



Fonte: Produção do próprio autor.

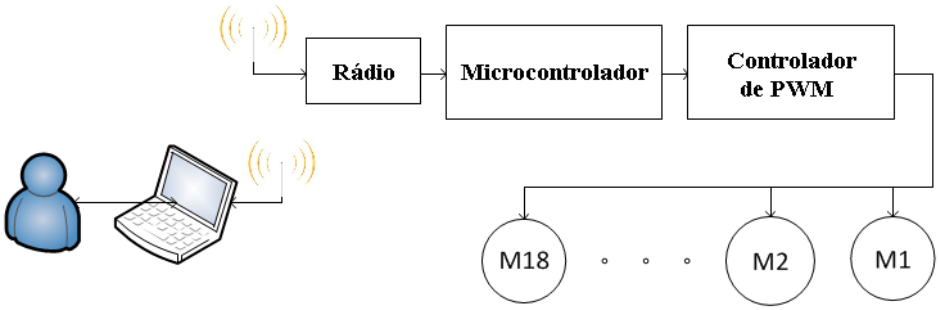
3 SISTEMA EMBARCADO

O sistema embarcado do robô hexápode foi desenvolvido segundo as seguintes necessidades: controlar os dezoito servo-motores responsáveis pelo movimento das pernas do robô, receber comandos de um usuário remotamente e preservar o fornecimento de energia ao microcontrolador em qualquer situação.

Essas necessidades foram atendidas pela arquitetura exibida na Figura 20, adaptada de Demuth (2013). O controle dos servos é realizado pelo conjunto Microcontrolador e Controlador de PWM, a recepção de comandos remotos é realizada pelo conjunto Rádio e Microcontrolador e a proteção da alimentação do microcontrolador é garantida pela utilização de duas baterias distintas para alimentação do sistema, uma para os servo-motores e a outra para toda a parte lógica.

Vale destacar que o escopo deste trabalho não inclui o recebimento de comandos remotamente e nem a utilização de baterias para a alimentação. Essas características foram consideradas no projeto do circuito eletrônico para que a placa produzida neste trabalho possa ser utilizada em futuros trabalhos e aplicações.

Figura 20 - Arquitetura do sistema embarcado.



Fonte: Adaptado de DEMUTH, 2013.

Neste trabalho foram usados os mesmos servos do robô original, da marca *TowerPro* e modelo SG5010. As especificações detalhadas dos servos são descritas em Demuth (2013). Os demais componentes utilizados foram o controlador de PWM, TLC5943, o microcontrolador, ARM Cortex-M4, e o módulo *bluetooth*, HC-05, que serão descritos na seções seguintes. Após a descrição dos componentes, o esquemático do circuito e o projeto da placa serão apresentados.

3.1 Controlador de PWM

O controlador de PWM é um circuito integrado que possui diversos canais de PWM para acionamento de diversas cargas simultaneamente. Este tipo de componente é bastante utilizado em painéis de LED, onde o número de lâmpadas a acionar é extremamente grande.

No caso do robô hexápode, é necessário o acionamento de dezoito servo-motores, um número acima do comum. Por isso, os microcontroladores que possuem esta quantidade de canais são muito caros ou até mesmo de difícil acesso no país. Por este motivo, o uso de um controlador dedicado se fez necessário.

A escolha do controlador deste projeto foi baseada na experiência adquirida no projeto anterior. O controlador utilizado no antigo protótipo, o TLC5940, foi, talvez, o principal problema do projeto. O TLC5940, em operação, não mantem um valor de PWM na saída por mais de um ciclo, isto é, para contínuo acionamento de um servo-motor em determinada posição se fazia necessário a atualização do valor do registrador correspondente antes do final de cada ciclo.

No caso do controlador não receber o novo valor a tempo, o servo-motor fica sem excitação e sujeito apenas ao torque estático, não suficiente para manter os servos na posição desejada. Quando o novo valor do PWM é atualizado, o servo-motor já saiu da posição e agora precisa retornar à ela, entrando assim em estado de partida. Por consequência, todos os servos estavam quase sempre em estado de partida, o que causa tremores em todas as articulações e um consumo de corrente acima de 2 ampères.

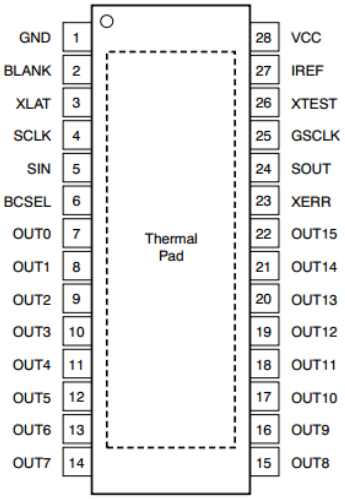
Para corrigir esse problema e tornar a utilização do robô viável, um outro controlador da mesma família, o TLC5943, foi sugerido por gaudio (2013). Este controlador possui a função de *auto repeat*, que dispensa a atualização contínua dos registradores do controlador de PWM.

3.1.1 Visão geral do controlador

O TLC5943 (*Texas Instruments*) é um driver de LED de 16 canais com corrente constante e cada canal tem como saída um PWM de 16 bits de resolução. Além disso, o controlador também possui controle de brilho global de 7 bits que permite variar a corrente dos canais de saída em 128 níveis entre zero e a corrente máxima. Outra funcionalidade deste componente é a detecção

de erros: é possível detectar se algum canal está aberto (LOD – *LED Open Detection*) e se o componente está sobreaquecido (TEF – *Thermal Error Flag*). Assim como em Demuth (2013), foram utilizados dois TLC5943 em cascata para atender aos dezoito servo-motores das pernas.

Figura 21 - Representação do TLC5943 com a seu diagrama de pinos.



Fonte: Adaptado de TEXAS INSTRUMENTS, 2007.

A programação do TLC5943 (Figura 21) envolve as seguintes etapas: inicialização das entradas do controlador, configuração do nível de brilho global, ativação da função de repetição automática e envio dos valores de PWM a serem enviados aos servos.

A etapa de inicialização é necessária por que o controlador não possui valores padrão para o registrador de escala de cinza (*Grayscale Shift Register*), então o primeiro passo é desabilitar as saídas do controlador a fim de evitar acionamento indesejado dos servos.

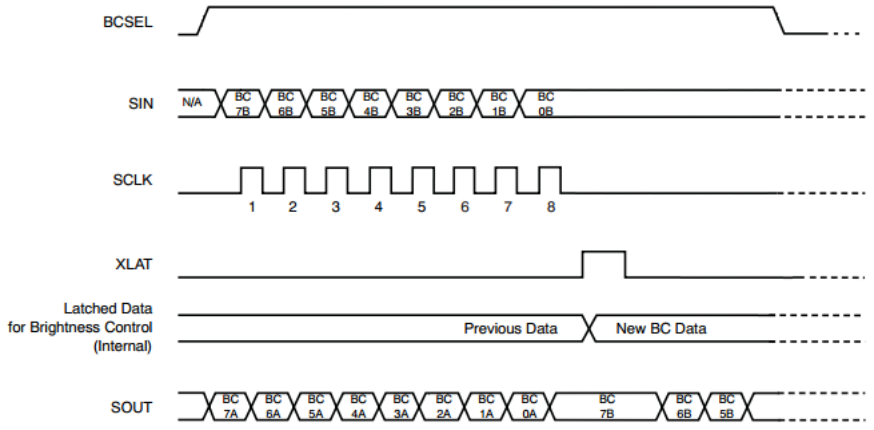
O registrador de escala de cinza é o responsável por receber os valores de PWM que serão enviados nas saídas do TLC5943. No caso do acionamento dos servo-motores o nome escala de cinza não ilustra muito bem este acionamento, já que ele é feito através de pulsos correspondentes às posições desejadas dos motores (DEMUTH, 2013).

Neste trabalho o termo escala de cinza será utilizado para facilitar o entendimento do funcionamento do TLC5943, porém, o objetivo do controlador é a geração de pulsos e não a variação da tensão média, como sugerido pelo nome.

A configuração do nível de brilho global é feita junto com a ativação da função de repetição automática do PWM, ao se escrever no registrador de controle de brilho (*Brightness Control/Auto Repeat Enable Shift Register*). Os sete bits menos significativos desse registrador definem o nível de corrente de saída e o bit mais significativo define se a função de repetição automática está ativada ou não.

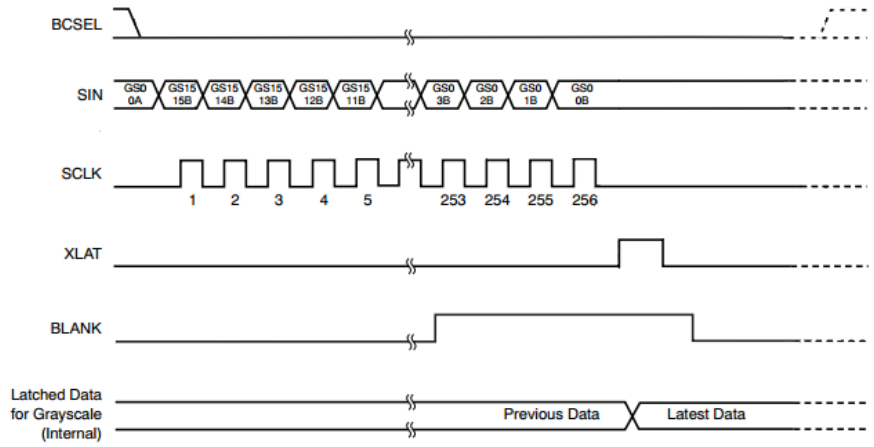
A temporização do controlador para a escrita no registrador de controle de brilho é exibida na Figura 22. O primeiro passo é fazer a entrada BCSEL receber um sinal alto. Em seguida, envia-se via SPI (*Serial Peripheral Interface*) o byte a ser escrito. Por fim, um pulso em XLAT faz com que o conteúdo enviado ao registrador seja carregado em um *latch* que envia aos demais componentes do circuito do TLC5943.

Figura 22 - Temporização dos sinais para a escrita no registrador de controle de brilho.



Fonte: Adaptado de TEXAS INSTRUMENTS, 2007.

Figura 23 - Temporização do sinais para a escrita no registrador de escala de cinza.



Fonte: Adaptado de TEXAS INSTRUMENTS, 2007.

A Figura 23 mostra a temporização dos sinais para a escrita no registrador de escala de cinza. Para escrever nesse registrador, é necessário que a entrada BCSEL receba um sinal baixo. Em seguida, os 16 bits de cada um dos 16 canais são enviados na ordem mostrada na figura, totalizando 256 bits. Após o envio dos bits, um pulso em XLAT grava os dados enviados no primeiro *latch* de escala de cinza. Finalmente, um pulso na entrada BLANK é responsável por zerar todas as saídas e, após a borda de descida, enviar os novos valores de PWM para as mesmas.

Ambas as temporizações apresentadam forma simplificadas para um melhor entendimento. Na prática, as transições e pulsos obedecem várias restrições de tempo, que devem ser consultadas antes de utilizar este componente.

Uma característica importante do TLC5943 é que a sua escala de cinza não funciona como um PWM comum, em que o pulso é mantido alto pelo número de ciclos estipulados e depois baixo até o fim do período do PWM. Neste caso, o período total do PWM é dividido em 128 segmentos. O período total do PWM (T_{PWM}) é o período de tempo entre a primeira entrada de GSCLK até a 65.536ª entrada após o sinal BLANK ter recebido a entrada baixa.

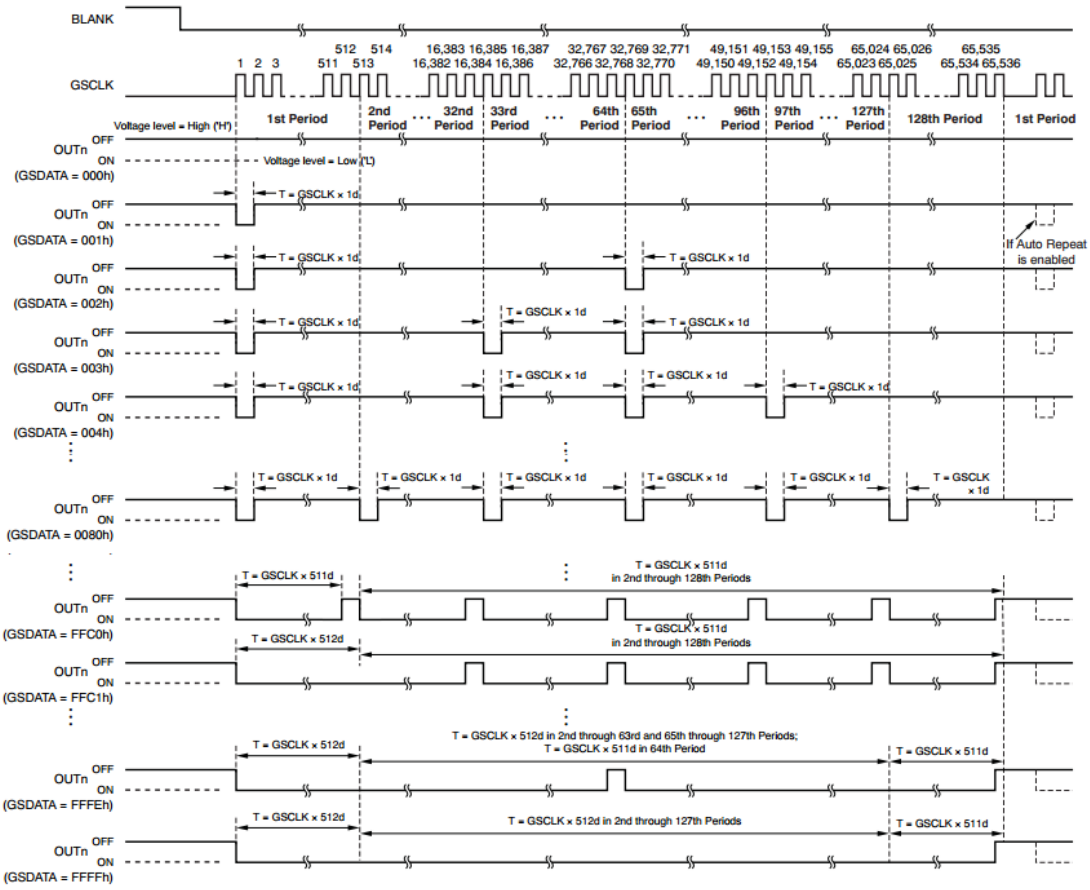
Cada um desses segmentos tem uma escala de cinza com um máximo de 512. Ao incrementar o valor da escala de cinza global, um nível de escala de cinza é adicionado a um dos 128 segmentos a cada incremento global. O problema é que os níveis não são adicionados em segmentos adjacentes e sim segundo uma lógica bem complexa, como pode ser visto na Figura 24.

Porém, para todos os valores de escala de cinza global múltiplos de 128 observa-se que o número de níveis altos em cada um dos 128 segmentos é igual. Desta forma, neste projeto, a saída PWM do controlador não tem como período os 65.536 níveis da escala de cinza do canal. O período do PWM de saída para os servos é composto pelos 512 níveis de escala de cinza dos segmentos mencionados e, seu acionamento é realizado carregando sempre um múltiplo de 128 no registrador de escala de cinza do canal.

O TLC5943 também possui uma função chamada *auto output off* que desliga automaticamente os canais de saída em que foram detectados LEDs em aberto ou em curto-circuito. A detecção

dessas falhas é feita a partir da medição da tensão de saída. Quando a tensão de saída é menor que 0,3V a *flag* de LOD do canal é atribuída.

Figura 24 – Escala de cinza do TLC5943.



Fonte: Adaptado de TEXAS INSTRUMENTS, 2007.

A função *auto output off* monitora as *flags* de todos os canais e no 33º ciclo de GSCLK todos os canais com detecção de erro atribuída são desligados automaticamente. As consequências dessa função e os ajustes realizados serão descritos no Capítulo 5.

3.1.2 Frequências de relógio

Os servo-motores utilizados nesse projeto são acionados periodicamente com pulsos de comprimento variável, entre 0,615ms e 2,491ms, em uma frequência de 50Hz (DEMUTH,2013). Portanto, a frequência do PWM enviado pelo controlador deve ser de 50 Hz. Para garantir essa temporização duas frequências devem ser calculadas: f_{GSCLK} e f_{SCLK}.

A frequência do sinal GSCLK determina o período dos níveis da escala de cinza global e a frequência do sinal SCLK é utilizada na comunicação do microcontrolador com o TLC5943. Considerando as características do TLC5943, f_{GSCLK} pode ser calculada a partir da seguinte expressão:

$$f_{GSCLK} = 512 * f_{PWM} \tag{1}$$

Portanto, para f_{PWM} = 50Hz, a frequência do sinal GSCLK tem que ser 25,6kHz. A frequência do sinal SCLK tem que ser grande o suficiente para que uma escrita nos dois registradores de escala de cinza demore menos tempo que um estouro dos 65536 níveis de GSCLK. A expressão seguinte calcula a frequência mínima do sinal SCLK:

$$f_{SCLK} = \frac{n}{128} * f_{PWM} \tag{2}$$

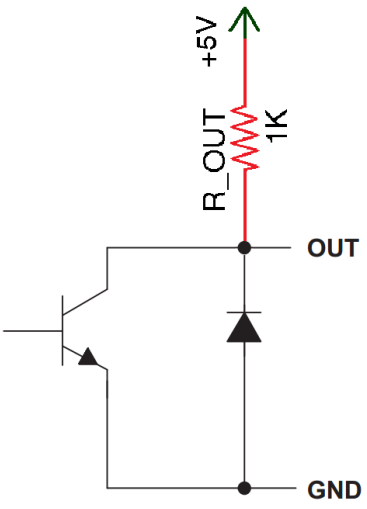
Onde n é o número de bits a serem escritos. Como neste casos deseja-se escrever em dois registradores de 256 bits, n é igual a 512 e, portanto, a frequência mínima do sinal SCLK é 200Hz. Esta frequência não necessita da mesma precisão da anterior. Na verdade, quando maior o seu valor, mais confiável será a temporização de escrita no controlador. Neste projeto a frequência utilizada para o sinal SCLK foi de 1MHz.

3.1.3 Componentes externos

Assim como apresentado em Demuth (2013) para o TLC5940, o TLC5943 também possui os estágios de saída de cada canal de PWM em coletor aberto. Desta forma, cada saída deste controlador precisa de um resistor de *pull-up* para que seja possível obter um nível lógico na saída.

O valor do resistor de *pull-up* utilizado foi arbitrado em 1kΩ. O estágio de saída do TLC5943 pode ser visualizado na Figura 25, que mostra o resistor introduzido.

Figura 25 - Estágio de saída do TLC5943.



Fonte: Adaptado de DEMUTH, 2013.

A corrente máxima no coletor do transistor do estágio de saída é definida por um resistor externo R_{IREF}, conectado entre os pinos IREF e GND. A corrente máxima de saída, é dada pela seguinte expressão, retirada do *datasheet*:

$$I_{max} = 41 * \frac{V_{IREF}}{R_{IREF}} \tag{3}$$

A tensão de referência V_{IREF} é interna ao controlador, tipicamente 1,2V. O valor de I_{max} foi calculado de forma a possibilitar um ajuste da tensão correspondente ao nível lógico baixo. Por isso, o valor do resistor escolhido foi 1,8kΩ o que faz a corrente máxima de saída ser aproximadamente 27mA.

3.2 Microcontrolador

Componente crítico de qualquer sistema embarcado, o microcontrolador a ser usado nesse projeto deveria atender às seguintes necessidades:

1. Ser barato;
2. UART, para a comunicação com o controle de alto nível (para desenvolvimento futuro);
3. *Timer* para o sinal GSCLK, responsável pelo PWM;
4. *Timer* para a interrupção de tempo do sistema (*SysTick*);
5. SPI, para a comunicação com o controlador de PWM;

6. Ferramenta de *debug* de fácil acesso;
7. Frequência de relógio do sistema elevada ou melhor arquitetura, para aumentar a capacidade de processamento e viabilizar aplicações futuras.

Os requisitos apresentados acima continuam, em sua maioria, iguais aos definidos por Demuth (2013). Porém, no trabalho citado, um dos *timers* era necessário para criar uma interrupção periódica para atualização dos servos mas, neste caso, o requisito é uma interrupção para a criação de um relógio para o sistema operacional de tempo real que foi utilizado (*FreeRTOS*).

Além disso, o microcontrolador utilizado anteriormente, o MSP430FR5739, estava no limite da sua capacidade de processamento, causado principalmente pela necessidade de gerenciar os dados enviados pelo controlador de alto nível via UART e a atualização periódica do controlador de PWM. Portanto, a adição do sétimo requisito foi necessária ao projeto.

Encontrar um componente que atenda à todos os requisitos acima é uma tarefa difícil, principalmente pelo conflito entre o baixo custo, a capacidade de processamento e uma ferramenta de *debug* de fácil acesso. Por este motivo, outro requisito mencionado por Demuth (2013) foi dispensado: a necessidade de uma arquitetura conhecida. Desta forma, foi possível utilizar um kit de desenvolvimento produzido pela *Texas Instruments* que surpreendentemente atende à todos os requisitos listados.

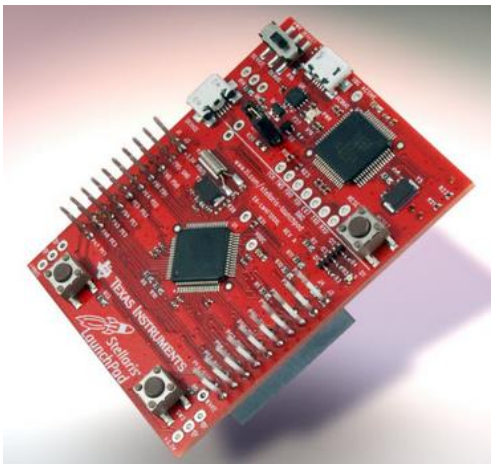
O microcontrolador escolhido para o integrar este projeto foi o ARM Cortex-M4, especificamente o LM4F120H5QR. A empresa americana *Texas Instruments* possui em seu catálogo um kit de desenvolvimento de baixíssimo custo com esse microcontrolador. A título de comparação, o kit utilizado, o *Stellaris Launchpad* foi adquirido por US$10,00 enquanto o kit de desenvolvimento do MSP430FR5739 custa US$35,00 no site do fabricante.

Apesar de serem recursos críticos, a quantidade de memórias ROM e RAM não foram listadas acima pois não seria possível avaliar de maneira precisa a real quantidade desses recursos para aplicações futuras. Além disso, a maioria dos microcontroladores com alto processamento disponíveis no mercado tem quantidades razoáveis desses recursos. O microcontrolador escolhido, por exemplo, possui 256kB de memória Flash e 32kB de memória RAM.

3.2.1 Stellaris LM4F120 *Launchpad Evaluation Board*

Em primeiro lugar, esta plataforma de desenvolvimento foi utilizada por já ter sido adquirida previamente à concepção deste projeto, quando ainda estava em pré-venda pela fabricante. É importante ressaltar que a mesma empresa já substituiu toda a linha *Stellaris* pela nova linha de desenvolvimento para processadores ARM, a *Tiva*.

Figura 26 - *Stellaris* LM4F120 *Launchpad Evaluation Board*.



Fonte: TEXAS INSTRUMENTS, 2014.

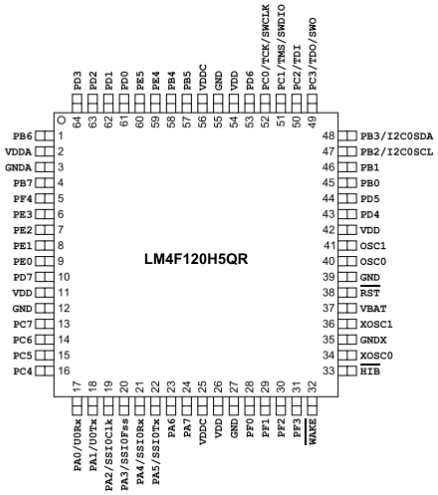
A *Stellaris* LM4F120 *Launchpad Evaluation Board* (Figura 26) é uma placa de desenvolvimento para ARMs Córtex-M4 da família LM4F120. Os recursos desta placa incluem:

- ARM LM4F120H5QR integrado:
  - 256kB de memória Flash;
  - 32kB de memória RAM;
  - Arquitetura RISC de 32 bits com até 80MHz;
  - Três comparadores analógicos, dois conversores analógico-digital de 12 bits, sensor de temperatura;
  - Oito interfaces UARTs, quatro SPIs, um USB *Host* de alta velocidade, duas CAN e seis I²C;
  - Duas entradas de *encoder* em quadratura e dezesseis saídas PWM;
  - *Timer SysTick* e doze *timers* (seis de 32 bits e seis de 64 bits);
- Um LED RGB;

- Dois botões de entrada para o usuário;
- ICDI integrado;
- Chave de *reset*;
- Conectores para placas de expansão.

Um diagrama com os pinos do microcontrolador pode ser visto na Figura 27.

Figura 27 - LM4F120H5QR.



Fonte: Adaptado de TEXAS INSTRUMENTS, 2014.

3.3 Rádio

O rádio necessário para este projeto não necessita de muitas funcionalidades, dentre elas, estão:

1. Baixo custo;
2. Conhecimento prévio da arquitetura;
3. Interface UART com o microcontrolador.

O Bluetooth foi a tecnologia escolhida para estabelecer a comunicação entre o robô hexápode e o usuário de alto nível, esta decisão teve dois motivos. Primeiro, existe um módulo transceptor *bluetooth* de baixo custo no mercado (Figura 28) e, segundo, este tem sido utilizado por diversos alunos do curso de Engenharia Elétrica da UFES. Ainda mais, a configuração de fábrica do rádio atende aos requisitos de integração do projeto, pois sua interface UART já vem ativada.

Figura 28 - Módulo transceptor *bluetooth*, HC-05.

Fonte: EMARTEE, 2014.

Além disso, a tecnologia *bluetooth* permite uma fácil comunicação com computadores pessoais portáteis (*laptops*), uma vez que a maioria desses computadores já tem o *hardware* para este tipo de comunicação. Desta forma, no futuro, o robô poderá ser controlado remotamente por um programa escrito em C++, por exemplo, que será portável para qualquer outro computador.

A tecnologia *bluetooth* oferece, de maneira geral, um curto alcance de comunicação, que varia de 1 à 100 metros. O módulo escolhido pertence à classe 2, com uma potência de emissão de até 4dBm, o que representa um alcance de até 10 metros. Esse alcance permite testes em salas, laboratórios e até em alguns ambientes abertos e, portanto, atende aos requisitos deste projeto.

3.4 Esquemático

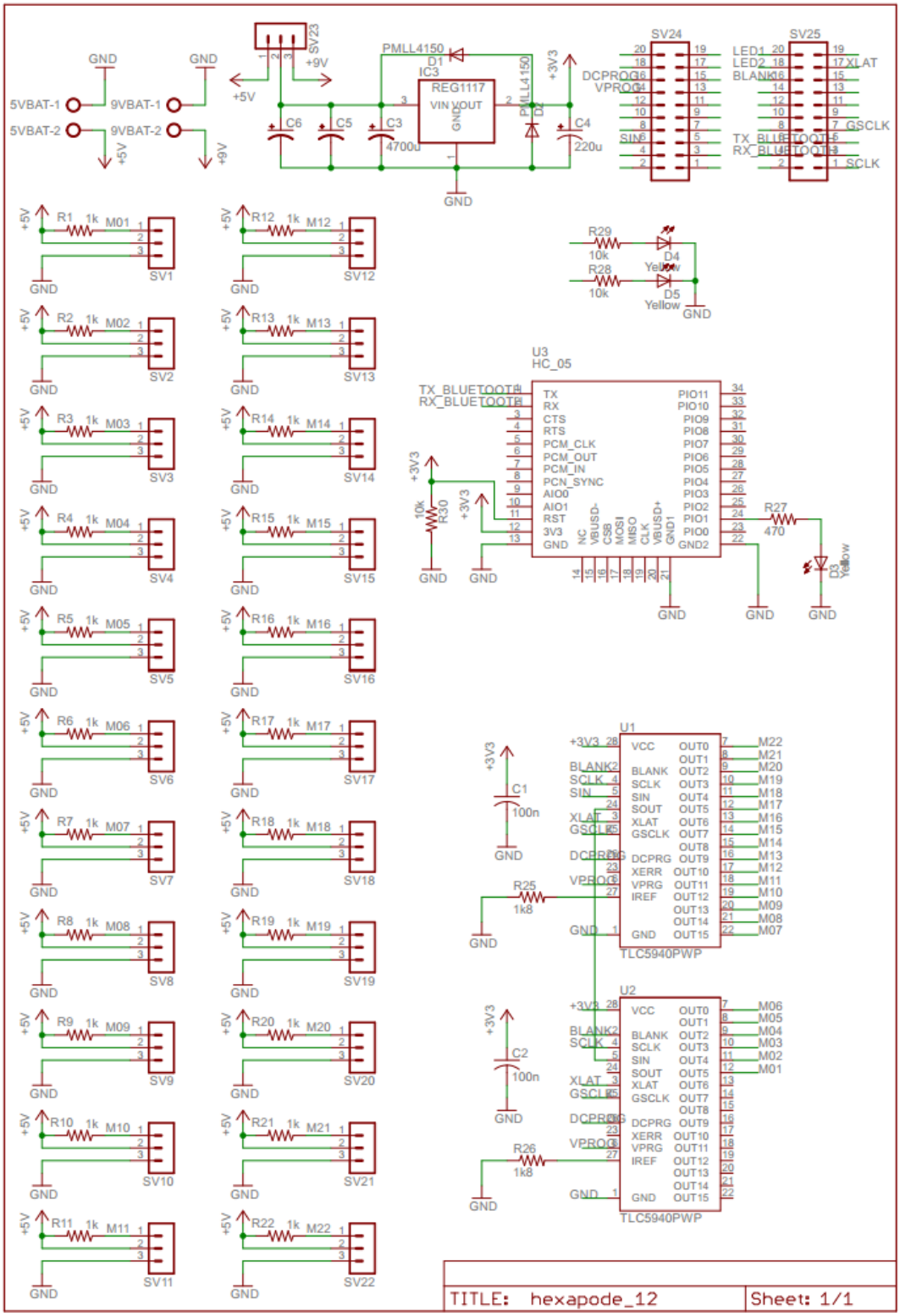
A fim de conectar os componentes do circuito eletrônico deste projeto, uma placa de circuito impresso foi confeccionada. O seu esquemático está exibido na Figura 29.

Na Figura 29 pode-se observar as partes que integram o circuito. No canto superior esquerdo está a entrada de alimentação da placa, com conectores para duas baterias (6V e 9V), um *jumper* para selecionar qual bateria irá alimentar o circuito de controle e, em seguida, o regulador de tensão de 3,3V. Como recomendado por Demuth (2013) foi feita a separação da alimentação dos servo-motores e da parte de controle.

A bateria de 9V está conectada somente ao *jumper* que seleciona a fonte de energia do circuito de controle. Porém, a bateria de 6V está conectada tanto aos servos quanto ao mesmo *jumper*, pois, caso seja necessário e possível, o robô pode operar somente com uma bateria. Pensando

nesta eventual situação, foram colocados dois capacitores extras na entrada do regulador de tensão caso seja necessário um aumento da capacitância neste ponto do circuito.

Figura 29 - Esquemático da placa de circuito impresso.



Fonte: Produção do próprio autor.

À direita do regulador de tensão estão os dois conectores que ligam o *Stellaris Launchpad* à placa desenvolvida. Os sinais que vem do microcontrolador foram nomeados para facilitar a visualização do esquemático. Observa-se que não chega alimentação nenhuma à esses conectores, o que foi um erro na fase de projeto da placa de circuito impresso. Este problema foi corrigido com a soldagem de dois *jumpers* externos na parte inferior da placa.

Abaixo dos conectores, temos dois LEDs para uso geral pelo usuário. Seu acionamento é feito diretamente pelo microcontrolador. Abaixo dos LEDs, está o módulo *bluetooth* e seus componentes externos necessários para a configuração básica.

No canto inferior direito, observa-se os dois controladores de PWM. Ambos recebem os mesmos sinais de controle do microcontrolador, sendo apenas o sinal SIN do segundo controlador proveniente da saída (SOUT) do primeiro, ou seja, os controladores estão em configuração cascata.

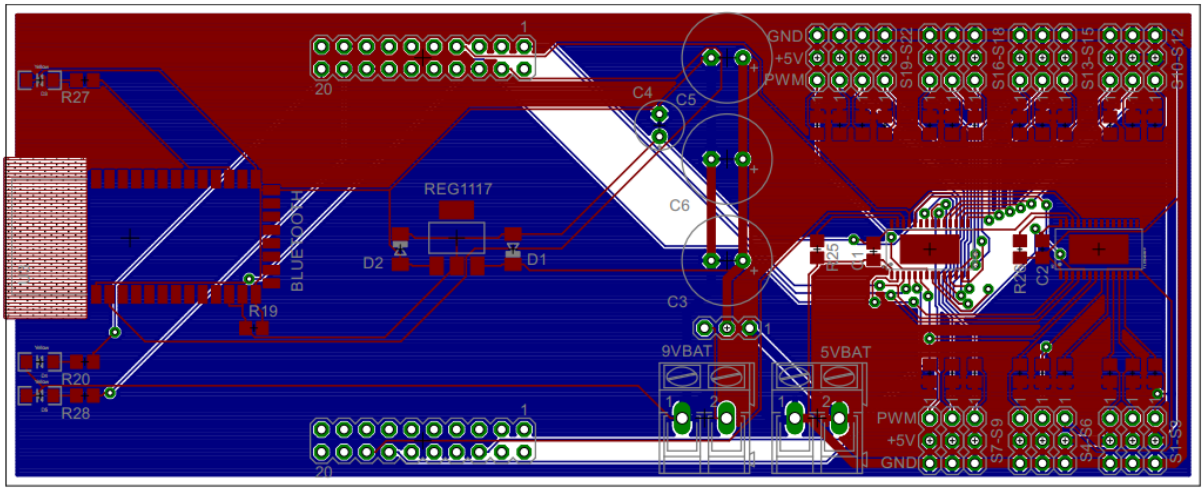
Na parte esquerda do esquemático observa-se conectores para 22 servo-motores, além dos resistores de *pull-up* necessários às saídas dos TLC5943s. Apesar dos movimentos do robô necessitarem apenas de 18 servos, 3 para cada perna, o circuito foi projetado para operar com até 22 servos, desta forma, futuros trabalhos podem envolver a construção de uma cabeça ou cauda para o robô.

3.5 Placa

A placa de circuito impresso (Figura 30) foi projetada utilizando duas camadas para roteamento. As medidas relevantes ao projeto são:

Distância entre trilhas:	0,203 mm
Distância entre trilhas e vias:	0,203 mm
Distância entre trilhas e *pads*:	0,203 mm
Distância entre vias e vias:	0,203 mm
Distância entre *pads* e *pads*:	0,203 mm
Tamanho das trilhas:	0,203 mm
Tamanho das trilhas que alimentam os servos:	1,016 mm
Furação mínima:	0,330 mm

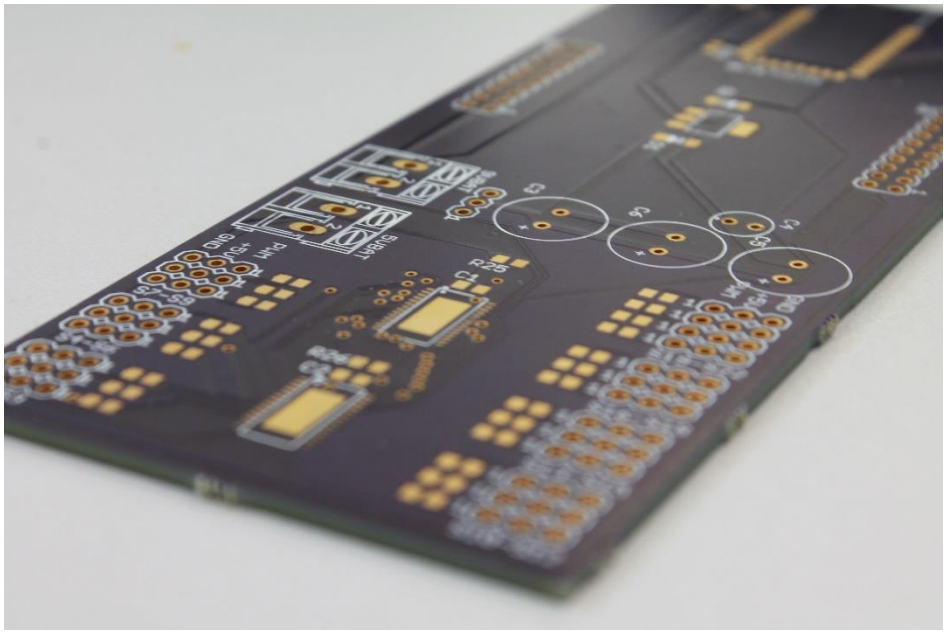
Figura 30 - *Layout* da placa de circuito impresso.



Fonte: Produção do próprio autor.

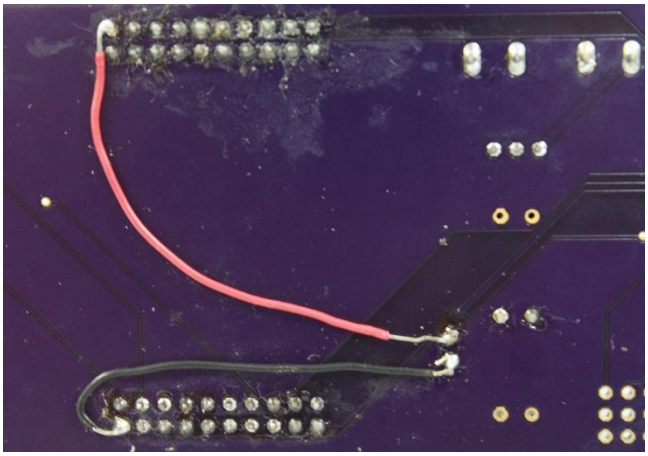
A placa foi confeccionada pela empresa americana *OSHPark*, que produz placas com furos metalizados, verniz de acabamento e camada para identificação dos componentes. A placa pode ser vista na Figura 31 e a adaptação feita com os *jumpers* para alimentação pode ser vista na Figura 32.

Figura 31 - Placa de circuito impresso confeccionada.



Fonte: Produção do próprio autor.

Figura 32 - Adaptação feita na parte inferior da placa.



Fonte: Produção do próprio autor.

4 SOFTWARE EMBARCADO

O programa para controlar o robô hexápode foi desenvolvido para executar um teste simples de caminhada e giro. Apesar deste objetivo simples, foi utilizado um sistema operacional de tempo real (RTOS) para viabilizar futuras implementações mais complexas. A primeira seção deste capítulo explica a escolha do sistema operacional. Em seguida, as estruturas de dados utilizadas são detalhadas e, depois, o fluxograma. Por fim, as funções utilizadas no código são apresentadas.

4.1 Sistema Operacional de Tempo Real

A utilização de um RTOS, que foi recomendada por Demuth (2013), visa facilitar a implementação de funcionalidades no robô. Um sistema operacional possibilita a programação de tarefas que serão executadas paralelamente, através de preempção. Por isso, é mais fácil descrever ações simultâneas usando um sistema operacional.

Por exemplo, a rotina de movimento pode ser dividida em tarefas diferentes responsáveis, cada uma, apenas pelo movimento de uma perna. Desta forma, cada tarefa atualizaria os servos da perna correspondente e, por meio da preempção, todas as tarefas seriam executadas simultaneamente e as pernas também se movimentariam ao mesmo tempo.

O sistema operacional utilizado neste projeto foi o *FreeRTOS*. Sua escolha foi baseada em um trabalho de iniciação científica realizado no Laboratório Cisne (GAUDIO, 2013). Em uma etapa da iniciação cientifica, foram avaliadas duas opções de sistemas operacionais de tempo real gratuitas a serem estudadas e implementadas, o *FreeRTOS* e o NUT/OS.

No trabalho, constatou-se que, dentre as duas opções, o *FreeRTOS* possui documentação muito superior em termos de explicação das funções disponíveis. Além disso o consumo de memória RAM e ROM dos programas desenvolvidos com *FreeRTOS* possuem estimativas bem definidas.

4.2 Estruturas de dados

O programa desenvolvido é também bastante simples nas estruturas de dados que utiliza. O valor do PWM de cada canal é armazenado em um vetor global de 44 *bytes*, dois *bytes* para

cada canal. A primeira posição do vetor contém o *byte* mais significativo do PWM do primeiro servo-motor (M01, segundo a Figura 29), a segunda posição contém o *byte* menos significativo deste mesmo PWM e as posições seguintes contém os valores referentes aos próximos servos em ordem crescente. Um esquema do vetor pode ser visto na Tabela 1.

Tabela 1 - Vetor que armazena os valores de PWM de cada canal.

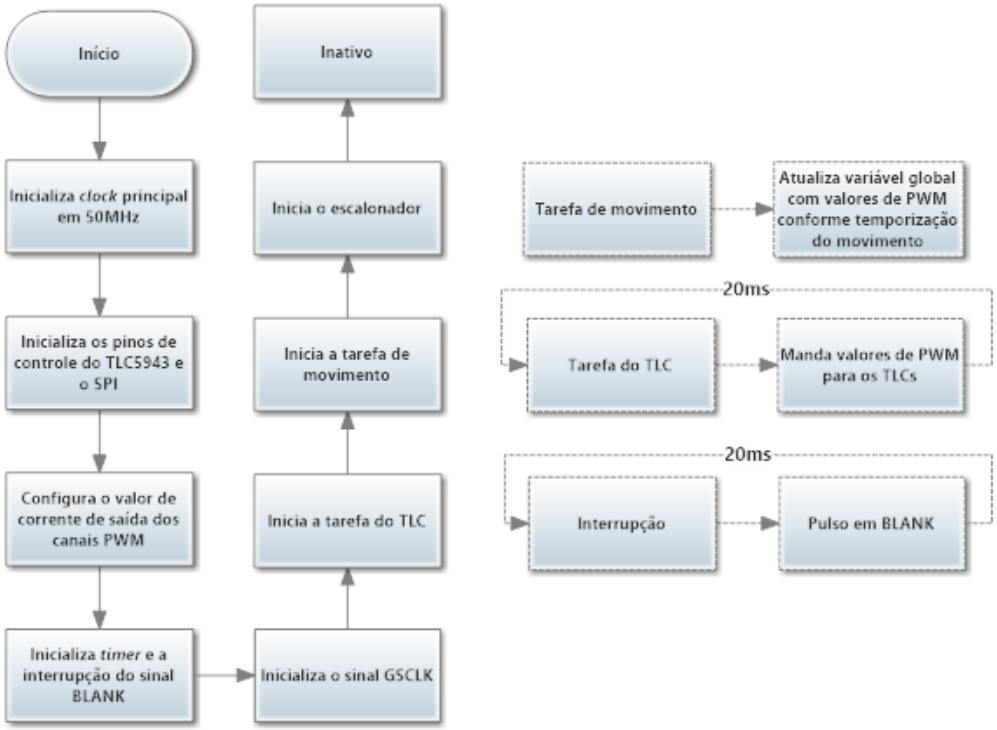
Canal	Canal 1		Canal 2		...	Canal 18	
Posição no Vetor	0	1	2	3	...	34	35

Este vetor é atualizado por uma tarefa que controla a temporização e posição de cada um dos servos. Outra tarefa é encarregada de ler o vetor e mandar os valores via SPI para o controlador TLC5943.

4.3 Fluxograma

Esta seção descreve a execução do programa desenvolvido para o controle do robô hexápode. O fluxograma da Figura 33 mostra de forma sintetizada todos os passos envolvidos nesse controle.

Figura 33 - Fluxograma do *software* embarcado.



Fonte: Produção do próprio autor.

O primeiro passo é iniciar o *clock* principal do microcontrolador. A referência de *clock* utilizada é um cristal interno de 16Mhz. O microcontrolador usa um circuito PLL para gerar uma frequência de 400Mhz que então é dividida para alcançar 50MHz.

Em seguida, os pinos de controle XLAT, BCSEL, BLANK e XTEST são configurados como saídas e tem os seus valores inicializados. O SPI também é habilitado, junto com os pinos SOUT e SCLK. A frequência do SPI é de 1MHz.

O próximo passo é a inicialização do *timer* e da interrupção do sinal de BLANK. O *timer* é configurado como periódico de 32 bits. É carregado um valor correspondente à um período de 20ms. A interrupção é habilitada por estouro.

Seguindo, o microcontrolador manda dois caracteres para a programação do controle de brilho dos controladores. Depois, o sinal GSCLK é inicializado em 25,6kHz. A partir de agora os controladores estão com saídas habilitadas.

As tarefas de atualização do TLC5943 e dos valores armazenados no programa são inicializadas e, em seguida, o escalonador é chamado. A partir deste momento o programa funciona periodicamente com as duas tarefas e a interrupção.

A tarefa do TLC é executada a cada 20ms. Ela é responsável por enviar os valores de PWM para acionamento dos servo-motores ao TLC5943. Os valores enviados são aqueles que estão armazenados no vetor global descrito na seção 4.2.

A tarefa de movimento é responsável por atualizar os valores de PWM no vetor global, que será lido pela tarefa do TLC. É importante ressaltar que a tarefa de movimento é a única função do programa que escreve nesse vetor, e que a tarefa do TLC é a única que o lê.

A interrupção por *timer* é executada a cada 20ms e é responsável pelo envio de um pulso em BLANK aos controladores. Para o seu bom funcionamento, os controladores necessitam que o pulso de BLANK seja recebido precisamente a cada 20ms. Uma alteração neste período causa movimentos indesejados nos servos, pois os atrasos são somados até que o PWM enviado para os servos seja acrescido de um ciclo de GSCLK.

A utilização desta interrupção foi necessária pois não foi possível garantir a temporização utilizando a função de *delay* do *FreeRTOS*. Além disso, a função *auto repeat* não foi utilizada pois, nos movimentos programados, os valores enviados aos servos são atualizados frequentemente, e a alta taxa de atualização não era garantida pela função *auto refresh*.

4.4 Funções

Abaixo estão descritas as funções implementadas neste trabalho.

4.4.1 void Init_TLC5943(void)

Inicializa os periféricos e portas utilizadas no funcionamento do TLC5943. Os pinos de saída BCSEL e XLAT são inicializados com nível lógico baixo e XTEST e BLANK com nível lógico alto.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.2 void ControlsInit(void)

Configura os pinos correspondentes às saídas BCSEL, XTEST, BLANK e XLAT como saída.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.3 void SPIInit(void)

Inicializa o módulo SSI do microcontrolador a partir do *clock* principal de 50MHz. O SPI é configurado no modo *master*, com protocolo motorola, pacote de 8 bits e frequência de transmissão de 1MHz.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.4 void Timer_TLC5943_Init(void)

Inicializa o *timer* 3A do microcontrolador e sua correspondente interrupção. Este *timer* é configurado em 32 bits e no modo periódico. O valor carregado corresponde a interrupções com intervalos de 20ms entre si.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.5 void Interrupt_TLC5943_Init(void)

Habilita a interrupção por estouro do *timer* 3A.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.6 void Set_TLC5943_BCdata(void)

Transmite 2 caracteres que controlam o valor da fonte de corrente na saída do TLC5943 e a função *auto repeat* do controlador. São enviados ao todo 16 bits, 8 para cada controlador. Os valores enviados são definido na variável bcData, inicializada em user_TLC5943.c.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.7 void Set_TLC5943_GSdata(void)

Transmite 44 caracteres que controlam o ciclo de trabalho das saídas do TLC5943. São enviados ao todo 352 bits, sendo 16 bits para cada um dos 22 servos conectados aos controladores. Os caracteres enviados são o conteúdo do variável global PWM, declarada em user_VariaveisGlobais.c.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.8 void GSClockInit(void)

Inicializa a saída GSCLK a partir de um *timer* de 16 bits. A saída é um sinal PWM com ciclo de trabalho de 50% e frequência 25600Hz.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.9 void Pulse_BlankPin(void)

Manda um pulso de aproximadamente 200ns na saída BLANK. Essa função é utilizada na interrupção do *timer* 3A que sincroniza a borda de descida de BLANK em um período exato de 20ms. Variações nesse período causam movimentos indesejados nos servos.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.10 void TlcTaskInit(void)

Cria a tarefa responsável por atualizar os valores de PWM nos registradores dos controladores TLC5943. Esta tarefa é inicializada com prioridade igual à prioridade da tarefa *Idle* mais quatro. A prioridade da tarefa *Idle* é um referência nos níveis de prioridade, pois esta tarefa só é executada quando não há nenhuma outra para executar. A escolha de quatro níveis acima foi arbitrária. Na verdade, o que importa para o programa desenvolvido é que a tarefa do TLC tenha prioridade maior que a tarefa de movimento.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.11 void MovementTaskInit(void)

Cria a tarefa responsável por atualizar os valores de PWM na variável global que a outra tarefa lê antes de mandar os valores para os controladores TLC5943. Esta tarefa é inicializada com prioridade igual à prioridade da tarefa *Idle* mais três.

Parâmetros: Nenhum

Retorno: Nenhum

4.4.12 void TlcTask(void)

Transmite os dados de PWM para os controladores utilizando a função Set_TLC5943_GSdata() descrita anteriormente. Esta tarefa é executada uma vez a cada 20ms.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.13 void MovementTask(void)

Atualiza os valores de PWM na variável global utilizando as funções descritas em user_Hexapode.c. Esta tarefa é executada uma vez a cada 20ms.

Parâmetros: Nenhum
Retorno: Nenhum

4.4.14 void SetAngle(unsigned char motor, int angle)

Atualiza os valores de PWM do servo determinado pela variável motor na variável global. O valor recebido no parâmetro angle é multiplicado por 128 e alocado em duas posições de 8 bits cada na variável global. A multiplicação por 128 se faz necessária pelas características do TLC5943, detalhadas em 3.1.1.

Parâmetros:	motor	Servo-motor que terá o ângulo alterado
	angle	Valor do PWM correspondente ao ângulo desejado
Retorno:	Nenhum	

4.4.15 portTickType StartUp(portTickType lastTime)

Comanda o robô hexápode em um movimento de inicialização de seus servo-motores para uma posição padrão.

Parâmetros:	lastTime	Tempo de execução do RTOS antes da função.

Retorno:	lastTime	Tempo de execução do RTOS após a função.

4.4.16 portTickType PreStep(portTickType lastTime)

Comanda o robô hexápode em um movimento de preparação para a caminhada.

Parâmetros:	lastTime	Tempo de execução do RTOS antes da função.
Retorno:	lastTime	Tempo de execução do RTOS após a função.

4.4.17 portTickType Step(portTickType lastTime)

Comanda o robô hexápode em um movimento de caminhada em dois passos.

Parâmetros:	lastTime	Tempo de execução do RTOS antes da função.
Retorno:	lastTime	Tempo de execução do RTOS após a função.

4.4.18 portTickType PreSpin(portTickType lastTime)

Comanda o robô hexápode em um movimento de preparação para o giro.

Parâmetros:	lastTime	Tempo de execução do RTOS antes da função.
Retorno:	lastTime	Tempo de execução do RTOS após a função.

4.4.19 portTickType Spin(portTickType lastTime)

Comanda o robô hexápode em um movimento de giro em dois passos.

Parâmetros:	lastTime	Tempo de execução do RTOS antes da função.
Retorno:	lastTime	Tempo de execução do RTOS após a função.

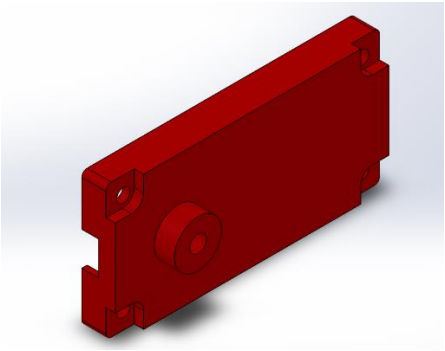
5 RESULTADOS EXPERIMENTAIS

Durante o desenvolvimento deste trabalho várias dificuldades foram encontradas. Algumas foram superadas com alterações no projeto original do robô enquanto outras inviabilizaram parte dos testes propostos. Neste capítulo, serão apresentados os principais problemas encontrados, suas soluções e situações que não puderam ser resolvidas completamente.

5.1 Encaixe do fêmur

O projeto original da estrutura sofreu alterações, dentre elas, a inserção de um segundo fêmur com encaixe na parte posterior dos servos da tíbia e da coxa. Para realizar os encaixes, foram impressas novas tampas para os servos, com a adição de um pino onde o fêmur seria encaixado. Entretanto, este pino (Figura 34) se mostrou muito frágil, devido ao processo de impressão da peça.

Figura 34 - Tampa dos servo-motores com pino de encaixe.

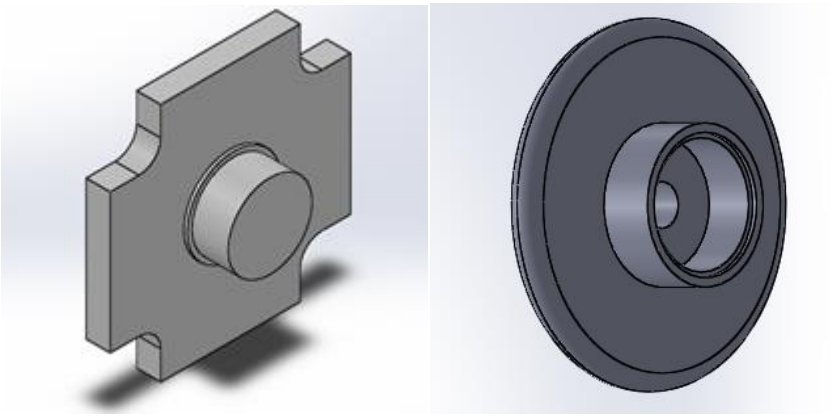


Fonte: Produção do próprio autor.

Por causa da posição das peças durante a impressão, as camadas do pino demoravam para ser impressas e então a ligação entre o pino e a parte plana da tampa ficou frágil. Durante a montagem percebeu-se essa fragilidade e então uma adaptação foi proposta e realizada.

A nova peça possui uma superfície plana para ser colada na tampa do servo-motor e um pino de mesmas dimensões do que estava na tampa anteriormente. Desta vez as características da peça permitiram uma impressão mais rápida, o que aumentou a resistência da peça. Além disso, o furo dentro do pino foi preenchido e o encaixe deixou de ser feito com parafuso e passou a ser feito com uma tampa também projetada e impressa em 3D. As novas peças podem ser visualizadas na Figura 35.

Figura 35 - Peças impressas para substituir o encaixe original do fêmur.



Fonte: Produção do próprio autor.

5.2 *Auto output off*

Como mencionado no Capítulo 3, o TLC5943 possui uma função chamada *auto output off*. Por causa desta função a saída dos canais não pode ter nível lógico baixo em zero volts. Neste caso, o controlador entende que há um erro no circuito, como um LED aberto ou em curto circuito, e automaticamente desabilita o canal em questão. Esta função pode ser muito desejada quando o controlador é empregado para o acionamento de LEDs mas, neste caso, ela interfere negativamente no projeto.

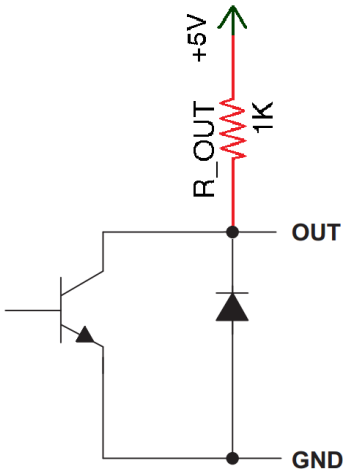
Para contornar este problema foi necessário elevar o nível lógico baixo para um valor acima do limiar de detecção de erro e abaixo do máximo valor de nível lógico baixo aceito pelos servos. Como o fabricante dos servos utilizados não fornece um *datasheet* com todas as características elétricas, entre elas o valor de tensão máximo para o nível lógico baixo, um sinal de controle foi simulado utilizando um gerador de funções.

Durante o teste, variou-se o nível lógico baixo entre 0V e 1,5V e o servo não apresentou mau funcionamento. Desta forma, o valor de 700mV foi considerado um bom nível de tensão para o nível lógico baixo pois, além de cobrir o mínimo de 0,3V em mais de 100%, ele não está no limite máximo aceito pelos servos.

Em seguida, calculou-se a corrente necessária no coletor dos transistores de saída dos controladores para se obter o nível lógico baixo de 700mV. Do circuito do estágio de saída dos

controladores (Figura 36) sabe-se que quando o transistor estiver ativado, a corrente no coletor será:

Figura 36 - Estágio de saída do TLC5943.



Fonte: Adaptado de DEMUTH, 2013.

$$V_{saída} = V_{cc} - R_{pull-up} I_{saída}$$

$$I_{saída} = \frac{V_{cc} - V_{saída}}{R_{pull-up}} = \frac{5 - 0{,}7}{1k} = 4{,}3mA \tag{4}$$

Portanto a corrente de saída deveria ser 4,3mA. Este valor foi alcançado com o controle de brilho fornecido pelo TLC5943. Como mostrado anteriormente, o TLC5943 permite um controle de brilho em 128 níveis que é ajustado pelo valor no registrador de controle de brilho. O nível de controle de brilho (BC) necessário foi calculado como mostrado a seguir.

$$BC = 128 * \frac{I_{saída}}{I_{max}} = 128 * \frac{4{,}3}{27{,}33} = 20{,}137 \cong 20 \tag{4}$$

Com este nível de controle de brilho a tensão mínima de saída de fato foi 700mV e o funcionamento normal do controlador foi alcançado.

5.3 Resolução do PWM

Outro obstáculo encontrado foi a baixa resolução obtida com o PWM dos controladores. O TLC5943 possui uma função distinta de distribuição do ciclo de trabalho do PWM, detalhada na Figura 24 – Escala de cinza do TLC5943. Por causa desse funcionamento, não se pode

utilizar todo o período de PWM para controlar o servo-motor, pois o ciclo é dividido em 128 segmentos que são incrementados independentemente. A única maneira de enviar o pulso requerido pelos servos é utilizar cada um dos segmentos como um ciclo de PWM de 20ms.

O problema disso é que cada segmento possui uma resolução de apenas 9 bits, ou 512 níveis. A implementação do sinal de controle do servo com esta resolução consegue um passo de 3,75 graus, um número extremamente alto que exclui qualquer possibilidade do robô realizar movimentos suaves na caminhada.

Por esse motivo, os movimentos de caminhada e giro do robô foram descritos de maneira bem simples, apenas para demonstração do funcionamento das outras partes.

5.4 Alimentação do circuito

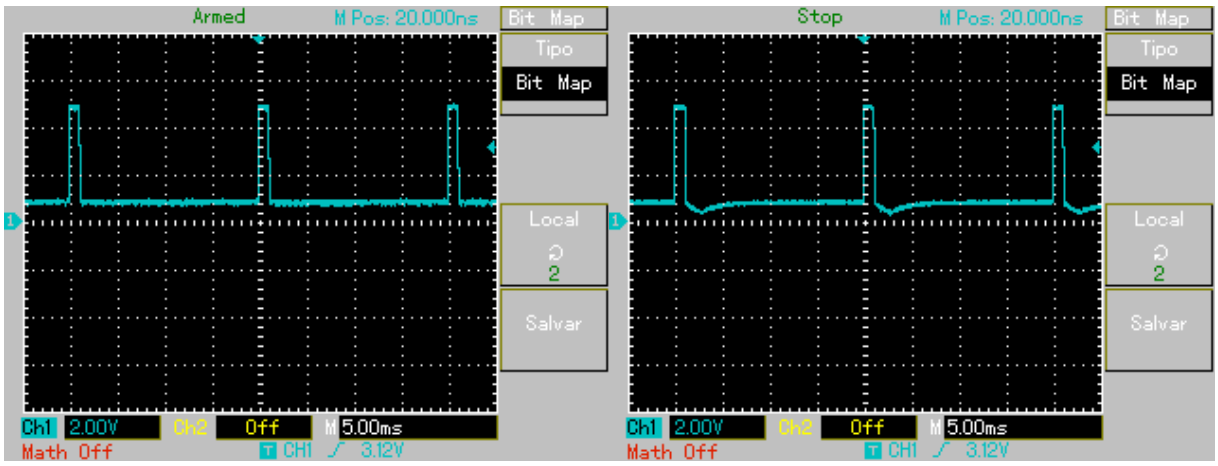
Outro problema encontrado no desenvolvimento desse projeto foi a queda da tensão de alimentação dos servo-motores. Como mostrado anteriormente, a tensão de saída dos canais do TLC5943 não pode ser inferior a 0,3V. Por isso, o estágio de saída foi projetado para fornecer 0,7V de tensão quando em nível lógico baixo.

Entretanto, este projeto depende incondicionalmente de uma tensão de alimentação constante para o seu funcionamento. Se a fonte de alimentação dos servos apresenta uma queda de tensão, mesmo que momentânea, o nível lógico baixo de todos os canais pode cair para menos de 0,3V o que causaria um erro de LED aberto em todos os canais e, por consequência, a desativação de todos eles.

Para se sustentar de pé no chão, o robô consome aproximadamente 1A com alguns picos momentâneos. Nesta condição a fonte de tensão utilizada, uma fonte de bancada comum, não consegue manter a tensão constante e então todos os canais são desabilitados, como explicado acima. Esta situação também se repete quando mais de um servo é acionado ao mesmo tempo ou quando dois acionamentos são feitos com um curto intervalo de tempo entre eles (menor que 60ms).

O sinal de controle dos servos enviado pelos controladores foi medido quando todas as pernas estava relaxadas e quando uma delas era pressionada contra o chão. As medições podem ser vistas na Figura 37.

Figura 37 – Sinal de controle dos servos com as pernas relaxadas (esquerda) e com uma das pernas pressionada (direita).



Fonte: Produção do próprio autor.

Como visualizado, a tensão de alimentação apresenta queda de aproximadamente 500mV quando uma perna é pressionada levemente. Para o caso de dois acionamentos simultâneos ou quando o robô tem que se sustentar em pé a queda de tensão é muito maior e a tensão de saída dos controladores atinge um nível menor que 0,3V.

Os testes também foram realizados com outra fonte de bancada, com 10 ampères de corrente máxima, e também com uma fonte chaveada industrial. Em ambos os casos as fontes de alimentação não foram capazes de manter a tensão constante e então os mesmos resultados foram obtidos.

Para conseguir visualizar os movimentos programados o robô foi apoiado pela parte inferior do corpo, de forma a deixar suas pernas sem apoio. Os movimentos foram gravados em vídeo e estão disponíveis em http://youtu.be/0UtLx4AwSwo.

6 CONCLUSÃO

A proposta deste trabalho foi de dar continuidade ao desenvolvimento de um robô hexápode, revendo seus projetos de estrutura, *hardware* e *firmware*. Os novos projetos foram feitos de forma a criar um robô mais robusto, que possa ser utilizado em aplicações futuras, e também de forma a atender as sugestões feitas nos trabalhos anteriores.

A nova estrutura do robô ficou mais leve e durável. Além disso as modificações no projeto estrutural tornaram as articulações mais rígidas nas direções em que ela não foi projetada para se movimentar.

A placa de circuito impresso foi confeccionada em um processo de alto nível que entregou um produto confiável e robusto. Além disso, as modificações no circuito eletrônico foram realizadas pra proteger o microcontrolador de quedas na tensão de alimentação, além de visar um melhor funcionamento dos servo-motores pela troca de controlador.

Infelizmente, o controlador de PWM sugerido por Gaudio (2013), o TLC5943, não se mostrou tão vantajoso pois seu principal atrativo, que era a função de a*uto repeat*, em que o sinal de BLANK não seria mais necessário para atualizar as saídas dos canais, acabou não sendo utilizada. O controlador possui diversas funções que são muito úteis para acionamento de LEDs, mas que complicam muito o acionamento de servo-motores.

O *firmware* foi revisto e um sistema operacional de tempo real foi utilizado. Este novo código permite o fácil desenvolvimento de trabalhos futuros sobre a plataforma desenvolvida, pois, permite ao usuário adicionar novas tarefas ao código sem grandes complicações.

As funções de movimento de caminhada e giro foram implementadas e testadas, ainda que de maneira simplificada. O movimento de fato não pode ser verificado por causa da limitação das fontes de tensão utilizadas e também pelas características do controlador escolhido.

Este trabalho obteve êxito em dar mais um passo para a construção de um robô hexápode robusto e utilizável. Entretanto, ainda existem algumas melhorias que precisam ser realizadas para atingir esse objetivo, como as descritas a seguir.

- A utilização de baterias para a alimentação dos servos. Utilizando uma bateria adequada ao circuito do robô a tensão se manterá constante enquanto ela está carregada, sem comprometer o acionamento dos servos. Uma sugestão é a bateria de 6V, Ni-MH e 2800mAh de carga.
- A substituição do TLC5943 pelo antigo TLC5940. O TLC5943 se mostrou atrativo pela sua função de a*uto repeat*, porém, suas funções *enhanced spectrum PWM operation* e *auto output off* são extremamente projetadas para LEDs e complicam bastante o acionamento de servo-motores. O principal motivo para a troca do TLC5940 foram os movimentos indesejados dos servos, causados pelo atraso do pulso do sinal BLANK. Porém, este problema foi corrigido neste trabalho utilizando uma interrupção de *timer* periódica que garantiu o período necessário de 20ms. Desta maneira, utilizar o TLC5940 novamente não acarretaria em movimentos indesejados e eliminaria alguns problemas encontrados no desenvolvimento deste novo projeto. Além disso, os dois controladores tem diagramas de pinos compatíveis o que permite a substituição direta do controlador na placa confeccionada neste trabalho.

Se as sugestões acima não forem suficientes para o funcionamento adequado do robô uma revisão do projeto deve ser feita, especialmente no *hardware* escolhido para acionamento dos servos. A utilização de outras arquiteturas devem ser consideradas, como o acionamento dos servos feito diretamente pelo microcontrolador.

Para conseguir acionar todos os servos simultaneamente pode-se utilizar uma ou mais saídas PWM para acionar diversos servos através de uma multiplexação no tempo. Neste caso, algumas saídas PWM do microcontrolador seriam configuradas para acionar um grupo de servo-motores cada.

Outra alternativa é a criação de saídas PWM via *software*, isto é, configurar alguns pinos de entrada e saída do microcontrolador para atuarem como PWM e criar funções que controlem este funcionamento. Neste caso, cada servo-motor seria acionado por uma saída distinta do microcontrolador, tenha ela *hardware* específico para PWM ou não.

Outra opção é manter o acionamento dos servos com um controlador separado do microcontrolador, porém, utilizando um dispositivo programável, como as FPGAs, para realizar este controle.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ANDY. **Forever Geek**. Disponivel em: <http://www.forevergeek.com/2009/08/cool_ four-legged_walking_robot_controlled_by_wii_nunchucks/>. Acesso em: 13 ago. 2013.

BOSTON DYNAMICS. **BigDog: The Most Advanced Rough-Terrain Robot on Earth**. Disponivel em: <http://www.bostondynamics.com/robot_bigdog.html>. Acesso em: 13 ago. 2013.

DEMUTH, P. R. **Desenvolvimento e Controle de um Robô Multiarticulado Hexápode**. 2013. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica) - Colegiado do Curso de Engenharia Elétrica, Universidade Federal do Espírito Santo. Vitória. 2013.

DWENGO. **Dwengo**. Disponivel em: <http://www.dwengo.org/tutorials/robot-starters-kit>. Acesso em: 13 ago. 2013.

EMARTEE. **HC-05 Serial Port Bluetooth Module - emartee.com**. Disponível em: < http://emartee.com/product/41982/>. Acesso em: 12 dez. 2014.

GAUDIO, L. A.; SALLES, E. O. T. **Desenvolvimento de Sistemas Embarcados Multifuncionais**. Trabalho de Iniciação Científica (Graduação em Engenharia Elétrica) - Laboratório Cisne, Universidade Federal do Espírito Santo. Vitória 2013.

MICROMAGIC SYSTEMS. **Micromagic Systems**. Disponivel em: <http://www.micro magicsystems.com/#/hexapod-v1/4516299667>. Acesso em: 13 ago. 2013.

TANTOS, A. **The Tumbleweed Robot**. Disponivel em: <http://www.tantosonline.com/ andras/robot.htm>. Acesso em: 13 ago. 2013.

TEXAS INSTRUMENTS. **16-Channel, 16-Bit PWM LED Driver with 7-Bit Global Brightness Control.** Austin, TX, EUA. 2007.

TEXAS INSTRUMENTS. **16 Channel LED Driver with Dot Correction and Grayscale PWM Control.** Austin, TX, EUA. 2007.

TEXAS INSTRUMENTS. **Stellaris Peripheral Driver Library.** Austin, TX, EUA. 2013.

TEXAS INSTRUMENTS. **Stellaris LM4F120 Launchpad Evaluation Board.** Austin, TX, EUA. 2013.

TEXAS INSTRUMENTS. **Stellaris LM4F120 Launchpad Evaluation Kit - EK-LM4F120XL - TI Tool Folder.** Disponível em: <http://www.ti.com/tool/ek-lm4f120xl>. Acesso em 12 dez. 2014.

TEXAS INSTRUMENTS. **Stellaris LM4F120H5QR Microcontroller.** Austin, TX, EUA. 2013.

SERVO DATABASE. **TowerPro SG-5010 Servo Specifications ad Reviews**. Disponível em: < http://www.servodatabase.com/servo/towerpro/sg-5010>. Acesso em: 12 dez. 2014.

WETTERGREEN, D. **NASA Space Telerobotics Program**. Disponivel em: <http://archive.is/34WNb>. Acesso em: 13 ago. 2013.

APÊNDICE A - Firmware

Este apêndice contém todos os códigos implementados como o firmware do robô.

- freertos_demo.c

```
//*****************************************************************************
//
// freertos_demo.c - Rotina principal de comando do robô hexápode.
//
// O código foi desenvolvido a partir de um exemplo de aplicação do FreeRTOS
// disponível na StellarisWare.
//
//*****************************************************************************

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_timer.h"
#include "inc/hw_ints.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/ssi.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"

#include "user_TLC5943.h"
#include "user_VariaveisGlobais.h"
#include "user_Timers.h"
#include "tlc_task.h"
#include "mov_task.h"

#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"

//*****************************************************************************
//
// The error routine that is called if the driver library encounters an error.
//
//*****************************************************************************
#ifdef DEBUG
void __error__(char *pcFilename, unsigned long ulLine)
{}
#endif

//*****************************************************************************
//
// This hook is called by FreeRTOS when an stack overflow error is detected.
//
//*****************************************************************************
```

```
void vApplicationStackOverflowHook(xTaskHandle *pxTask, signed char *pcTaskName)
{
    // This function can not return, so loop forever.  Interrupts are disabled
    // on entry to this function, so no processor interrupts will interrupt
    // this loop.
    while(1);
}

//*****************************************************************************
//
// Initialize FreeRTOS and start the initial set of tasks.
//
//*****************************************************************************
int main(void)
{
      // Set the clocking to run at 50 MHz from the PLL.
      ROM_SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN);

      Init_TLC5943();           // Inicializa os pinos de controle TLC5943 e o SPI
      Set_TLC5943_BCdata();     // Inicializa o limitador de corrente do TLC5943
      Timer_TLC5943_Init();     // Inicializa timer e interrupção
      GSClockInit();            // Inicializa GSCLK

      if(TlcTaskInit() != 0)
             while(1);

      if(MovementTaskInit() != 0)
            while(1);

      vTaskStartScheduler(); // Start the scheduler.  This should not return.

      while(1);
}
```

- startup_ccs.c

```
//*****************************************************************************
//
// startup_ccs.c - Startup code for use with TI's Code Composer Studio.
//
// Copyright (c) 2012 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 9453 of the EK-LM4F120XL Firmware Package.
```

```
//
//*****************************************************************************

//*****************************************************************************
//
// Forward declaration of the default fault handlers.
//
//*****************************************************************************
void ResetISR(void);
static void NmiSR(void);
static void FaultISR(void);
static void IntDefaultHandler(void);

//*****************************************************************************
//
// External declaration for the reset handler that is to be called when the
// processor is started
//
//*****************************************************************************
extern void _c_int00(void);

//*****************************************************************************
//
// Linker variable that marks the top of the stack.
//
//*****************************************************************************
extern unsigned long __STACK_TOP;

//*****************************************************************************
//
// External declarations for the interrupt handlers used by the application.
//
//*****************************************************************************
extern void xPortPendSVHandler(void);
extern void vPortSVCHandler(void);
extern void xPortSysTickHandler(void);
extern void TimerInterrupt3A_PulseBlankPin(void);

//*****************************************************************************
//
// The vector table.  Note that the proper constructs must be placed on this to
// ensure that it ends up at physical address 0x0000.0000 or at the start of
// the program if located at a start address other than 0.
//
//*****************************************************************************
#pragma DATA_SECTION(g_pfnVectors, ".intvecs")
void (* const g_pfnVectors[])(void) =
{
    (void (*)(void))((unsigned long)&__STACK_TOP),
                                            // The initial stack pointer
    ResetISR,                               // The reset handler
    NmiSR,                                  // The NMI handler
    FaultISR,                               // The hard fault handler
    IntDefaultHandler,                      // The MPU fault handler
    IntDefaultHandler,                      // The bus fault handler
    IntDefaultHandler,                      // The usage fault handler
    0,                                      // Reserved
    0,                                      // Reserved
```

```
0,                                      // Reserved
0,                                      // Reserved
vPortSVCHandler,                        // SVCall handler
IntDefaultHandler,                      // Debug monitor handler
0,                                      // Reserved
xPortPendSVHandler,                     // The PendSV handler
xPortSysTickHandler,                    // The SysTick handler
IntDefaultHandler,                      // GPIO Port A
IntDefaultHandler,                      // GPIO Port B
IntDefaultHandler,                      // GPIO Port C
IntDefaultHandler,                      // GPIO Port D
IntDefaultHandler,                      // GPIO Port E
IntDefaultHandler,                      // UART0 Rx and Tx
IntDefaultHandler,                      // UART1 Rx and Tx
IntDefaultHandler,                      // SSI0 Rx and Tx
IntDefaultHandler,                      // I2C0 Master and Slave
IntDefaultHandler,                      // PWM Fault
IntDefaultHandler,                      // PWM Generator 0
IntDefaultHandler,                      // PWM Generator 1
IntDefaultHandler,                      // PWM Generator 2
IntDefaultHandler,                      // Quadrature Encoder 0
IntDefaultHandler,                      // ADC Sequence 0
IntDefaultHandler,                      // ADC Sequence 1
IntDefaultHandler,                      // ADC Sequence 2
IntDefaultHandler,                      // ADC Sequence 3
IntDefaultHandler,                      // Watchdog timer
IntDefaultHandler,                      // Timer 0 subtimer A
IntDefaultHandler,                      // Timer 0 subtimer B
IntDefaultHandler,                      // Timer 1 subtimer A
IntDefaultHandler,                      // Timer 1 subtimer B
IntDefaultHandler,                      // Timer 2 subtimer A
IntDefaultHandler,                      // Timer 2 subtimer B
IntDefaultHandler,                      // Analog Comparator 0
IntDefaultHandler,                      // Analog Comparator 1
IntDefaultHandler,                      // Analog Comparator 2
IntDefaultHandler,                      // System Control (PLL, OSC, BO)
IntDefaultHandler,                      // FLASH Control
IntDefaultHandler,                      // GPIO Port F
IntDefaultHandler,                      // GPIO Port G
IntDefaultHandler,                      // GPIO Port H
IntDefaultHandler,                      // UART2 Rx and Tx
IntDefaultHandler,                      // SSI1 Rx and Tx
TimerInterrupt3A_PulseBlankPin,         // Timer 3 subtimer A
IntDefaultHandler,                      // Timer 3 subtimer B
IntDefaultHandler,                      // I2C1 Master and Slave
IntDefaultHandler,                      // Quadrature Encoder 1
IntDefaultHandler,                      // CAN0
IntDefaultHandler,                      // CAN1
IntDefaultHandler,                      // CAN2
IntDefaultHandler,                      // Ethernet
IntDefaultHandler,                      // Hibernate
IntDefaultHandler,                      // USB0
IntDefaultHandler,                      // PWM Generator 3
IntDefaultHandler,                      // uDMA Software Transfer
IntDefaultHandler,                      // uDMA Error
IntDefaultHandler,                      // ADC1 Sequence 0
IntDefaultHandler,                      // ADC1 Sequence 1
IntDefaultHandler,                      // ADC1 Sequence 2
```

```
IntDefaultHandler,                      // ADC1 Sequence 3
IntDefaultHandler,                      // I2S0
IntDefaultHandler,                      // External Bus Interface 0
IntDefaultHandler,                      // GPIO Port J
IntDefaultHandler,                      // GPIO Port K
IntDefaultHandler,                      // GPIO Port L
IntDefaultHandler,                      // SSI2 Rx and Tx
IntDefaultHandler,                      // SSI3 Rx and Tx
IntDefaultHandler,                      // UART3 Rx and Tx
IntDefaultHandler,                      // UART4 Rx and Tx
IntDefaultHandler,                      // UART5 Rx and Tx
IntDefaultHandler,                      // UART6 Rx and Tx
IntDefaultHandler,                      // UART7 Rx and Tx
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
IntDefaultHandler,                      // I2C2 Master and Slave
IntDefaultHandler,                      // I2C3 Master and Slave
IntDefaultHandler,                      // Timer 4 subtimer A
IntDefaultHandler,                      // Timer 4 subtimer B
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
IntDefaultHandler,                      // Timer 5 subtimer A
IntDefaultHandler,                      // Timer 5 subtimer B
IntDefaultHandler,                      // Wide Timer 0 subtimer A
IntDefaultHandler,                      // Wide Timer 0 subtimer B
IntDefaultHandler,                      // Wide Timer 1 subtimer A
IntDefaultHandler,                      // Wide Timer 1 subtimer B
IntDefaultHandler,                      // Wide Timer 2 subtimer A
IntDefaultHandler,                      // Wide Timer 2 subtimer B
IntDefaultHandler,                      // Wide Timer 3 subtimer A
IntDefaultHandler,                      // Wide Timer 3 subtimer B
IntDefaultHandler,                      // Wide Timer 4 subtimer A
IntDefaultHandler,                      // Wide Timer 4 subtimer B
IntDefaultHandler,                      // Wide Timer 5 subtimer A
IntDefaultHandler,                      // Wide Timer 5 subtimer B
IntDefaultHandler,                      // FPU
IntDefaultHandler,                      // PECI 0
IntDefaultHandler,                      // LPC 0
```

```
    IntDefaultHandler,                      // I2C4 Master and Slave
    IntDefaultHandler,                      // I2C5 Master and Slave
    IntDefaultHandler,                      // GPIO Port M
    IntDefaultHandler,                      // GPIO Port N
    IntDefaultHandler,                      // Quadrature Encoder 2
    IntDefaultHandler,                      // Fan 0
    0,                                      // Reserved
    IntDefaultHandler,                      // GPIO Port P (Summary or P0)
    IntDefaultHandler,                      // GPIO Port P1
    IntDefaultHandler,                      // GPIO Port P2
    IntDefaultHandler,                      // GPIO Port P3
    IntDefaultHandler,                      // GPIO Port P4
    IntDefaultHandler,                      // GPIO Port P5
    IntDefaultHandler,                      // GPIO Port P6
    IntDefaultHandler,                      // GPIO Port P7
    IntDefaultHandler,                      // GPIO Port Q (Summary or Q0)
    IntDefaultHandler,                      // GPIO Port Q1
    IntDefaultHandler,                      // GPIO Port Q2
    IntDefaultHandler,                      // GPIO Port Q3
    IntDefaultHandler,                      // GPIO Port Q4
    IntDefaultHandler,                      // GPIO Port Q5
    IntDefaultHandler,                      // GPIO Port Q6
    IntDefaultHandler,                      // GPIO Port Q7
    IntDefaultHandler,                      // GPIO Port R
    IntDefaultHandler,                      // GPIO Port S
    IntDefaultHandler,                      // PWM 1 Generator 0
    IntDefaultHandler,                      // PWM 1 Generator 1
    IntDefaultHandler,                      // PWM 1 Generator 2
    IntDefaultHandler,                      // PWM 1 Generator 3
    IntDefaultHandler                       // PWM 1 Fault
};

//*****************************************************************************
//
// This is the code that gets called when the processor first starts execution
// following a reset event.  Only the absolutely necessary set is performed,
// after which the application supplied entry() routine is called.  Any fancy
// actions (such as making decisions based on the reset cause register, and
// resetting the bits in that register) are left solely in the hands of the
// application.
//
//*****************************************************************************
void
ResetISR(void)
{
    //
    // Jump to the CCS C initialization routine.  This will enable the
    // floating-point unit as well, so that does not need to be done here.
    //
    __asm("    .global _c_int00\n"
          "    b.w     _c_int00");
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives a NMI.  This
// simply enters an infinite loop, preserving the system state for examination
// by a debugger.
```

```
//
//*****************************************************************************
static void
NmiSR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
    }
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives a fault
// interrupt.  This simply enters an infinite loop, preserving the system state
// for examination by a debugger.
//
//*****************************************************************************
static void
FaultISR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
    }
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives an unexpected
// interrupt.  This simply enters an infinite loop, preserving the system state
// for examination by a debugger.
//
//*****************************************************************************
static void
IntDefaultHandler(void)
{
    //
    // Go into an infinite loop.
    //
    while(1)
    {
    }
}
```

- FreeRTOSConfig.h

```
/*
    FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.


    ***************************************************************************
     *                                                                         *
     *    FreeRTOS tutorial books are available in pdf and paperback.          *
```

```
     *    Complete, revised, and edited pdf reference manuals are also       *
     *    available.                                                          *
     *                                                                        *
     *    Purchasing FreeRTOS documentation will not only help you, by        *
     *    ensuring you get running as quickly as possible and with an         *
     *    in-depth knowledge of how to use FreeRTOS, it will also help        *
     *    the FreeRTOS project to continue with its mission of providing      *
     *    professional grade, cross platform, de facto standard solutions     *
     *    for microcontrollers - completely free of charge!                   *
     *                                                                        *
     *    >>> See http://www.FreeRTOS.org/Documentation for details. <<<      *
     *                                                                        *
     *    Thank you for using FreeRTOS, and thank you for your support!       *
     *                                                                        *
    ***************************************************************************


    This file is part of the FreeRTOS distribution.

    FreeRTOS is free software; you can redistribute it and/or modify it under
    the terms of the GNU General Public License (version 2) as published by the
    Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
    >>>NOTE<<< The modification to the GPL is included to allow you to
    distribute a combined work that includes FreeRTOS without being obliged to
    provide the source code for proprietary components outside of the FreeRTOS
    kernel.  FreeRTOS is distributed in the hope that it will be useful, but
    WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
    or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
    more details. You should have received a copy of the GNU General Public
    License and the FreeRTOS license exception along with FreeRTOS; if not it
    can be viewed here: http://www.freertos.org/a00114.html and also obtained
    by writing to Richard Barry, contact details for whom are available on the
    FreeRTOS WEB site.

    1 tab == 4 spaces!

    http://www.FreeRTOS.org - Documentation, latest information, license and
    contact details.

    http://www.SafeRTOS.com - A version that is certified for use in safety
    critical systems.

    http://www.OpenRTOS.com - Commercial support, development, porting,
    licensing and training services.
*/

#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H

/*-----------------------------------------------------------
 * Application specific definitions.
 *
 * These definitions should be adjusted for your particular hardware and
 * application requirements.
 *
 * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF THE
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.
 *
```

```
 * See http://www.freertos.org/a00110.html.
 *----------------------------------------------------------*/

#define configUSE_PREEMPTION                    1
#define configUSE_IDLE_HOOK                     0
#define configUSE_TICK_HOOK                     0
#define configCPU_CLOCK_HZ                      ( ( unsigned long ) 50000000 )
#define configTICK_RATE_HZ                      ( ( portTickType ) 1000 )
#define configMINIMAL_STACK_SIZE                ( ( unsigned short ) 200 )
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) ( 30000 ) )
#define configMAX_TASK_NAME_LEN                 ( 12 )
#define configUSE_TRACE_FACILITY                1
#define configUSE_16_BIT_TICKS                  0
#define configIDLE_SHOULD_YIELD                 0
#define configUSE_CO_ROUTINES                   0
#define configUSE_MUTEXES                       1
#define configUSE_RECURSIVE_MUTEXES             1
#define configCHECK_FOR_STACK_OVERFLOW          2
#define configMAX_PRIORITIES                    ( ( unsigned portBASE_TYPE ) 16 )
#define configMAX_CO_ROUTINE_PRIORITIES         ( 2 )
#define configQUEUE_REGISTRY_SIZE               10

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */
#define INCLUDE_vTaskPrioritySet                1
#define INCLUDE_uxTaskPriorityGet               1
#define INCLUDE_vTaskDelete                     1
#define INCLUDE_vTaskCleanUpResources           0
#define INCLUDE_vTaskSuspend                    1
#define INCLUDE_vTaskDelayUntil                 1
#define INCLUDE_vTaskDelay                      1
#define INCLUDE_uxTaskGetStackHighWaterMark 1

/* Be ENORMOUSLY careful if you want to modify these two values and make sure
 * you read http://www.freertos.org/a00110.html#kernel_priority first! */

#define configKERNEL_INTERRUPT_PRIORITY         ( 7 << 5 )    /* Priority 7, or
0xE0 as only the top three bits are implemented.  This is the lowest priority. */
#define configMAX_SYSCALL_INTERRUPT_PRIORITY     ( 5 << 5 )  /* Priority 5, or
0xA0 as only the top three bits are implemented. */

#endif /* FREERTOS_CONFIG_H */
```

- tlc_task.h

```
/*
 * tlc_task.h
 *
 *  Created on: 05/12/14
 *      Author: Pedro H.
 */

#ifndef __TLC_TASK_H__
#define __TLC_TASK_H__

extern unsigned long TlcTaskInit(void);

#endif // __TLC_TASK_H__
```

- tlc_task.c

```c
/*
 * tlc_task.c
 *
 *  Created on: 05/12/14
 *      Author: Pedro H.
 */

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "drivers/buttons.h"
#include "utils/uartstdio.h"

#include "tlc_task.h"
#include "user_TLC5943.h"
#include "user_VariaveisGlobais.h"

#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"

#define TLCTASKSTACKSIZE        128         // Stack size in words

static void
TlcTask(void *pvParameters)
{
    portTickType ulLastTime;
    unsigned long ulSwitchDelay = 20;

    ulLastTime = xTaskGetTickCount(); // Get the current tick count.

    while(1)
    {
        Set_TLC5943_GSdata(PWM);

        vTaskDelayUntil(&ulLastTime, ulSwitchDelay / portTICK_RATE_MS);
    }
}

//*****************************************************************************
//
// Initializes the tlc task.
//
//*****************************************************************************
unsigned long
TlcTaskInit(void)
{
    if(xTaskCreate(TlcTask, (signed portCHAR *)"Switch",
                   TLCTASKSTACKSIZE, NULL, tskIDLE_PRIORITY +
                   4, NULL) != pdTRUE)
    {
        return(1);
```

```
    }

    return(0);
}
```

- mov_task.h

```
/*
 * mov_task.h
 *
 *  Created on: 05/12/14
 *      Author: Pedro H.
 */

#ifndef __MOV_TASK_H__
#define __MOV_TASK_H__

extern unsigned long MovementTaskInit(void);

#endif // __MOV_TASK_H__
```

- mov_task.c

```
/*
 * mov_task.c
 *
 *  Created on: 05/12/14
 *      Author: Pedro H.
 */

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "drivers/buttons.h"
#include "utils/uartstdio.h"

#include "mov_task.h"
#include "user_Hexapode.h"

#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"

#define MOVTASKSTACKSIZE        128         // Stack size in words

static void
MovementTask(void *pvParameters)
{
    portTickType ulLastTime;

    ulLastTime = xTaskGetTickCount();

    while(1)
```

```
    {
      ulLastTime = StartUp(ulLastTime);
      ulLastTime = PreStep(ulLastTime);
      ulLastTime = Step(ulLastTime);
      ulLastTime = Step(ulLastTime);
      ulLastTime = PreSpin(ulLastTime);
      ulLastTime = Spin(ulLastTime);
      ulLastTime = Spin(ulLastTime);
      ulLastTime = StartUp(ulLastTime);
      while(1);
    }
}

//*****************************************************************************
//
// Initializes the movement task.
//
//*****************************************************************************
unsigned long
MovementTaskInit(void)
{
    if(xTaskCreate(MovementTask, (signed portCHAR *)"Movement",
                   MOVTASKSTACKSIZE, NULL, tskIDLE_PRIORITY +
                   3, NULL) != pdTRUE)
    {
        return(1);
    }

    return(0);
}
```

- user_VariaveisGlobais.h

```
/*
 * user_VariaveisGlobais.h
 *
 *  Created on: 07/02/2013            Modified: 31/10/14
 *      Author: Hyori                   Author: Pedro H.
 */

#ifndef __VARIAVEISGLOBAIS_H__
#define __VARIAVEISGLOBAIS_H__

extern unsigned char PWM[];

#endif
```

- user_VariaveisGlobais.c

```
/*
 * user_VariaveisGlobais.c
 *
 *  Created on: 07/02/2013            Modified: 31/10/14
 *      Author: Hyori                   Author: Pedro H.
 */
```

```
unsigned char PWM[44] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0 };
```

- user_Interrupts.h

```
/*
 * user_Interrupts.h
 *
 *  Created on: 05/12/2014
 *      Author: Pedro H.
 */


#ifndef __USER_INTERRUPTS_H__
#define __USER_INTERRUPTS_H__


void Interrupt_TLC5943_Init(void);
void TimerInterrupt3A_PulseBlankPin(void);


#endif /* __USER_INTERRUPTS_H__ */
```

- user_Interrupts.c

```
/*
 * user_Interrupts.c
 *
 *  Created on: 05/12/2014
 *      Author: Pedro H.
 */

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_timer.h"
#include "inc/hw_ints.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "utils/uartstdio.h"

#include "user_TLC5943.h"

void Interrupt_TLC5943_Init(void)
{
	IntMasterEnable(); // Enable processor interrupts.
	TimerIntEnable(TIMER3_BASE, TIMER_TIMA_TIMEOUT);    // Timer3A interrupt
	IntEnable(INT_TIMER3A); // Enable the Timer3A interrupt on the processor
}


void TimerInterrupt3A_PulseBlankPin(void)
{
    TimerIntClear(TIMER3_BASE, TIMER_TIMA_TIMEOUT); // Clear the timer interrupt
flag.
```

```
	Pulse_BlankPin();	// Pulsa a
saída BLANK => Atualiza as saídas PWM do TLC5943
}
```

- user_Timers.h

```
/*
 * user_Timers.h
 *
 *  Created on: 05/12/2014
 *      Author: Pedro H.
 */

#ifndef __USER_TIMERS_H__
#define __USER_TIMERS_H__

void Timer_TLC5943_Init(void);

#endif /* __USER_TIMERS_H__ */
```

- user_Timers.c

```
/*
 * user_Timers.c
 *
 *  Created on: 05/12/2014
 *      Author: Pedro H.
 */

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_timer.h"
#include "inc/hw_ints.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "utils/uartstdio.h"

#include "user_Interrupts.h"

void Timer_TLC5943_Init(void)
{
	SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER3);
	TimerConfigure(TIMER3_BASE, TIMER_CFG_PERIODIC);
	TimerLoadSet(TIMER3_BASE, TIMER_A, SysCtlClockGet()/50 - 65);

	Interrupt_TLC5943_Init();

	TimerEnable(TIMER3_BASE, TIMER_A); // Enable Timer3A.
}
```

- user_TLC5943.h

```
/*
 * UART.h
```

```
 *
 *  Created on: 07/02/2013            Modified: 31/10/14
 *      Author: Hyori                   Author: Pedro H.
 */

#ifndef USER_TLC5943_H_
#define USER_TLC5943_H_

void SPIInit();     // Inicializa o canal SPI que se comunicará com o TLC 5940
void GSClockInit();// Inicializa o Clock da grey scale do TLC5940
void ControlsInit();// Inicializa os sinais de controle BLANK, XLAT e BCSEL
void Init_TLC5943(void);
void Set_TLC5943_BCdata(void);
void Set_TLC5943_GSdata(unsigned char gsData[]);
void Pulse_BlankPin(void);

#endif /* USER_TLC5943_H_ */
```

- user_TLC5943.c

```
/*
 * user_TLC5943.c
 *
 *  Created on: 07/02/2013            Modified: 31/10/14
 *      Author: Hyori                   Author: Pedro H.
 */

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/pin_map.h"
#include "driverlib/ssi.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"


#define      XLAT                    GPIO_PIN_2
#define      XLAT_PORT               GPIO_PORTB_BASE
#define      XLAT_PERIPH             SYSCTL_PERIPH_GPIOB

#define      BCSEL                   GPIO_PIN_1
#define      BCSEL_PORT              GPIO_PORTB_BASE
#define      BCSEL_PERIPH            SYSCTL_PERIPH_GPIOB

#define      XTEST                   GPIO_PIN_0
#define      XTEST_PORT              GPIO_PORTB_BASE
#define      XTEST_PERIPH            SYSCTL_PERIPH_GPIOB

#define      BLANK                   GPIO_PIN_3
#define      BLANK_PORT              GPIO_PORTB_BASE
#define      BLANK_PERIPH            SYSCTL_PERIPH_GPIOB

#define      GSCLK                   GPIO_PIN_6
#define      GSCLK_PORT              GPIO_PORTB_BASE
#define      GSCLK_PERIPH            SYSCTL_PERIPH_GPIOB
#define      GSCLK_TIMER_PERIPH      SYSCTL_PERIPH_TIMER0
```

```
#define      GSCLK_TIMER_BASE          TIMER0_BASE
#define      GSCLK_TIMER               TIMER_A
#define      GSCLK_PERIPH_PIN          GPIO_PB6_T0CCP0
#define      GSCLK_PWM_DIV             SYSCTL_PWMDIV_1
#define      GSCLK_FREQ                25600//25,6kHz
#define      DUTY_CYCLE_PERCENT              50

#define      SPI                             GPIO_PIN_6
#define      SPI_PORT_PERIPH                 SYSCTL_PERIPH_GPIOA
#define      SPI_PERIPH                      SYSCTL_PERIPH_SSI0
#define      SPI_PIN_BASE                    GPIO_PORTA_BASE
#define      SPI_TX                          GPIO_PIN_5
#define      SPI_RX                          GPIO_PIN_4
#define      SPI_CLK                         GPIO_PIN_2
#define      SPI_FSS                         GPIO_PIN_3
#define      SPI_TX_PINCONF                  GPIO_PA5_SSI0TX
#define      SPI_RX_PINCONF                  GPIO_PA4_SSI0RX
#define      SPI_CLK_PINCONF                 GPIO_PA2_SSI0CLK
#define      SPI_FSS_PINCONF                 GPIO_PA3_SSI0FSS
#define      SPI_BASE                        SSI0_BASE
#define      SPI_SOURCE_CLK                  SSI_CLOCK_SYSTEM
#define      SPI_CLK_FREQ                    1000000
#define      SPI_DATA_SIZE                   8


#define      SYSTEM_CLK                      50000000
#define      PIN_LOW                         0
#define      PIN_HIGH                        0xFF

//-------------------------------------------------------------------------------
//
//           Variáveis locais
//
//-------------------------------------------------------------------------------

 unsigned char bcData = 0x14;

//-------------------------------------------------------------------------------
//
//           Inicializa o SPI para comunicação com o TLC5940
//
//-------------------------------------------------------------------------------

void SPIInit(){

          ROM_SysCtlPeripheralEnable(SPI_PERIPH); // Habilita o periférico SSI
          SysCtlDelay(10); // Tempo de segurança
          SSIClockSourceSet(SPI_BASE, SPI_SOURCE_CLK);// Configura o source de
clock do SPI como sendo o main clock
          ROM_SysCtlPeripheralEnable(SPI_PORT_PERIPH);
          GPIOPinTypeSSI(SPI_PIN_BASE, SPI_FSS | SPI_CLK | SPI_RX | SPI_TX); //
registra no port desejado os pinos da comunicação SPI -- Não serve como
configuração
          GPIOPinConfigure(SPI_CLK_PINCONF); // Configura o MUX do GPIO ligando ao
pino de clock do CPI
          GPIOPinConfigure(SPI_TX_PINCONF);  // Configura o MUX do GPIO ligando ao
pino de TX do CPI
```

```
          GPIOPinConfigure(SPI_FSS_PINCONF); // Configura o MUX do GPIO ligando ao
pino de FSS do CPI
          GPIOPinConfigure(SPI_RX_PINCONF);  // Configura o MUX do GPIO ligando ao
pino de RX do CPI
          SSIConfigSetExpClk(SPI_BASE, SYSTEM_CLK, SSI_FRF_MOTO_MODE_0,
SSI_MODE_MASTER, SPI_CLK_FREQ, SPI_DATA_SIZE );           // Configura o clock,
modo de operação (master), protocolo motorola e "pacote" de 8 bits
          SSIEnable(SPI_BASE); // Habilita a operação do SPI

}

//-----------------------------------------------------------------------------
//
//           Inicializa o clock do Greyscale com um PWM, configurado para a
frequencia definida em GSCLK_FREQ
//
//-----------------------------------------------------------------------------

void GSClockInit(){
      ROM_SysCtlPeripheralEnable(GSCLK_PERIPH);
      SysCtlDelay(2);
      ROM_SysCtlPeripheralEnable(GSCLK_TIMER_PERIPH);
      SysCtlDelay(2);
      GPIOPinConfigure(GSCLK_PERIPH_PIN);
      GPIOPinTypeTimer(GSCLK_PORT, GSCLK);
      SysCtlPWMClockSet(GSCLK_PWM_DIV);
      TimerConfigure(GSCLK_TIMER_BASE, TIMER_CFG_SPLIT_PAIR|TIMER_CFG_A_PWM);
      TimerLoadSet(GSCLK_TIMER_BASE, GSCLK_TIMER, (int)(SYSTEM_CLK/GSCLK_FREQ)-1);
      TimerMatchSet(GSCLK_TIMER_BASE, GSCLK_TIMER,
(int)((((SYSTEM_CLK)/(GSCLK_FREQ*100)))*DUTY_CYCLE_PERCENT - 1));
      TimerEnable(GSCLK_TIMER_BASE, GSCLK_TIMER);
}


//-----------------------------------------------------------------------------
-----------------
//
//           Inicializa os sinais de controle  XLAT, VPROG, DCPROG e BLANK
//
//-----------------------------------------------------------------------------
-----------------


void ControlsInit(){

    ROM_SysCtlPeripheralEnable(XLAT_PERIPH); // Habilita a porta (periférico GPIO)
em que XLAT está
    SysCtlDelay(2);
    ROM_SysCtlPeripheralEnable(BCSEL_PERIPH);// Habilita a porta (periférico GPIO)
em que BCSEL está
    SysCtlDelay(2);
    ROM_SysCtlPeripheralEnable(XTEST_PERIPH);// Habilita a porta (periférico GPIO)
em que XTEST está
    SysCtlDelay(2);
    ROM_SysCtlPeripheralEnable(BLANK_PERIPH);// Habilita a porta (periférico GPIO)
em que BLANK está
    SysCtlDelay(2);
    GPIOPinTypeGPIOOutput(XLAT_PORT,  XLAT  );// Configura XLAT como saída
```

```
    GPIOPinTypeGPIOOutput(BCSEL_PORT, BCSEL );// Configura BCSEL como saída
    GPIOPinTypeGPIOOutput(XTEST_PORT, XTEST );// Configura XTEST como saída
    GPIOPinTypeGPIOOutput(BLANK_PORT,  BLANK );// Configura BLANK como saída
}

void Init_TLC5943(void){
      ControlsInit(); //XTEST, BCSEL, XLAT and BLANK as output
      GPIOPinWrite(XTEST_PORT, XTEST, PIN_HIGH); // Inicializa XTEST como baixo,
controle de corrente pela EEPROM
      GPIOPinWrite(BCSEL_PORT, BCSEL, PIN_LOW);  // Inicializa BCSEL como baixo,
programar DC register
      GPIOPinWrite(XLAT_PORT, XLAT, PIN_LOW);      // Inicializa XLAT como baixo,
não carrega nada para DC register por enquanto
      GPIOPinWrite(BLANK_PORT, BLANK, PIN_HIGH);   // Inicializa BLANK como baixo,
saídas habilitadas
      SPIInit();   // Inicializa o SPI,
}

void Set_TLC5943_BCdata(void)
{
      GPIOPinWrite(BCSEL_PORT, BCSEL, PIN_HIGH); // Levanta BCSEL, habilita a
escrita no BC Shift Register
      GPIOPinWrite(XLAT_PORT, XLAT, PIN_LOW);      // XLAT low, garantindo que não
haverá pulso para carregar os registradores
      GPIOPinWrite(BLANK_PORT, BLANK, PIN_LOW);    // BLANK low, sem necessidade
de apagar as saída (Auto Refresh Function)
      SysCtlDelay(2);

      SSIDataPut(SPI_BASE, bcData);   // Manda 1 byte para o BC Shift Register do
Segundo Controlador
      SSIDataPut(SPI_BASE, bcData);   // Manda 1 byte para o BC Shift Register do
Primeiro Controlador
      while(SSIBusy(SPI_BASE)) {}
      SysCtlDelay(2);

      GPIOPinWrite(XLAT_PORT, XLAT, PIN_LOW);      // XLAT low,
      SysCtlDelay(2);                              //
      GPIOPinWrite(XLAT_PORT, XLAT, PIN_HIGH);     // XLAT high, dá um pulso de
xlat, jogando o que foi recebido serialmente para os BC registers
      SysCtlDelay(4);                              //
      GPIOPinWrite(XLAT_PORT, XLAT, PIN_LOW);      // XLAT low,
      SysCtlDelay(10);
}

void Set_TLC5943_GSdata(unsigned char gsData[]){
      unsigned char counter = 0;

      GPIOPinWrite(BLANK_PORT, BLANK, PIN_LOW);        // BLANK low, sem
necessidade de apagar as saída (Auto Refresh Function)
      GPIOPinWrite(XLAT_PORT, XLAT, PIN_LOW);            // XLAT low, garantindo
que não há pulso de trigger
      GPIOPinWrite(BCSEL_PORT, BCSEL, PIN_LOW);          // Abaixa BCSEL,
habilita a escrita no GS Shift Register

      while(counter < 44)
      {
            SSIDataPut(SPI_BASE, gsData[counter]);
            counter++;
```

```
        }
        while(SSIBusy(SPI_BASE)) {}
        SysCtlDelay(2);

        GPIOPinWrite(XLAT_PORT, XLAT, PIN_LOW);      // XLAT low,
        SysCtlDelay(2);                              //
        GPIOPinWrite(XLAT_PORT, XLAT, PIN_HIGH);     // XLAT high, dá um pulso de
xlat, jogando o que foi recebido serialmente para os BC registers
        SysCtlDelay(4);                              //
        GPIOPinWrite(XLAT_PORT, XLAT, PIN_LOW);      // XLAT low,
        SysCtlDelay(10);
}



void Pulse_BlankPin(void)
{
        GPIOPinWrite(BLANK_PORT, BLANK, PIN_LOW);    // BLANK low,
        SysCtlDelay(2);                              //
        GPIOPinWrite(BLANK_PORT, BLANK, PIN_HIGH);   // BLANK high, dá um pulso em
blank. Transfere os dados do 1 para o 2 GS Latch
        SysCtlDelay(4);                              //
        GPIOPinWrite(BLANK_PORT, BLANK, PIN_LOW);    // BLANK low,
        SysCtlDelay(10);
}
```

- user_Hexapode.h

```
/*
 * hexapod.h
 *
 *  Created on: 06/02/2013            Modified: 05/12/2014
 *      Author: Hyori                   Author: Pedro H.
 */

#ifndef USER_HEXAPOD_H_
#define USER_HEXAPOD_H_

#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"

void SetAngle(unsigned char motor, int angle);
portTickType  StartUp(portTickType lastTime);
portTickType  PreStep(portTickType lastTime);
portTickType  Step(portTickType lastTime);
portTickType  PreSpin(portTickType lastTime);
portTickType  Spin(portTickType lastTime);

#endif /* USER_HEXAPOD_H_ */
```

- user_Hexapode.c

```
/*
 * user_Hexapode.c
 *
```

```
 *  Created on: 06/02/2013            Modified: 05/12/2014
 *      Author: Hyori                   Author: Pedro H.
 */

#include "user_Hexapode.h"
#include "user_VariaveisGlobais.h"

#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"


#define            motor_RightLeg1_FEMUR_TIBIA          0
#define            motor_RightLeg1_COXA_FEMUR           1
#define            motor_RightLeg1_BODY_COXA            2
#define            motor_RightLeg2_FEMUR_TIBIA          3
#define            motor_RightLeg2_COXA_FEMUR           4
#define            motor_RightLeg2_BODY_COXA            5
#define            motor_RightLeg3_FEMUR_TIBIA          6
#define            motor_RightLeg3_COXA_FEMUR           7
#define            motor_RightLeg3_BODY_COXA            8
#define            motor_LeftLeg1_FEMUR_TIBIA           9
#define            motor_LeftLeg1_COXA_FEMUR            10
#define            motor_LeftLeg1_BODY_COXA             11
#define            motor_LeftLeg2_FEMUR_TIBIA           12
#define            motor_LeftLeg2_COXA_FEMUR            13
#define            motor_LeftLeg2_BODY_COXA             14
#define            motor_LeftLeg3_FEMUR_TIBIA           15
#define            motor_LeftLeg3_COXA_FEMUR            16
#define            motor_LeftLeg3_BODY_COXA             17

#define            PLUS_86_25_DEGREES                   449
#define            PLUS_82_50_DEGREES                   450
#define            PLUS_78_75_DEGREES                   451
#define            PLUS_75_00_DEGREES                   452
#define            PLUS_71_25_DEGREES                   453
#define            PLUS_67_50_DEGREES                   454
#define            PLUS_63_75_DEGREES                   455
#define            PLUS_60_00_DEGREES                   456
#define            PLUS_56_25_DEGREES                   457
#define            PLUS_52_50_DEGREES                   458
#define            PLUS_48_75_DEGREES                   459
#define            PLUS_45_00_DEGREES                   460
#define            PLUS_41_25_DEGREES                   461
#define            PLUS_37_50_DEGREES                   462
#define            PLUS_33_75_DEGREES                   463
#define            PLUS_30_00_DEGREES                   464
#define            PLUS_26_25_DEGREES                   465
#define            PLUS_22_50_DEGREES                   466
#define            PLUS_18_75_DEGREES                   467
#define            PLUS_15_00_DEGREES                   468
#define            PLUS_11_25_DEGREES                   469
#define            PLUS_07_50_DEGREES                   470
#define            PLUS_03_75_DEGREES                   471
#define            ZERO_00_00_DEGREES                   472
#define            MINUS_03_75_DEGREES                  473
#define            MINUS_07_50_DEGREES                  474
```

```
#define            MINUS_11_25_DEGREES                    475
#define            MINUS_15_00_DEGREES                    476
#define            MINUS_18_75_DEGREES                    477
#define            MINUS_22_50_DEGREES                    478
#define            MINUS_26_25_DEGREES                    479
#define            MINUS_30_00_DEGREES                    480
#define            MINUS_33_75_DEGREES                    481
#define            MINUS_37_50_DEGREES                    482
#define            MINUS_41_25_DEGREES                    483
#define            MINUS_45_00_DEGREES                    484
#define            MINUS_48_75_DEGREES                    485
#define            MINUS_52_50_DEGREES                    486
#define            MINUS_56_25_DEGREES                    487
#define            MINUS_60_00_DEGREES                    488
#define            MINUS_63_75_DEGREES                    489
#define            MINUS_67_50_DEGREES                    490
#define            MINUS_71_25_DEGREES                    491
#define            MINUS_75_00_DEGREES                    492
#define            MINUS_78_75_DEGREES                    493
#define            MINUS_82_50_DEGREES                    494
#define            MINUS_86_25_DEGREES                    495

#define            DELAY                                  200

extern unsigned char PWM[];

void SetAngle(unsigned char motor, int angle)
{
	int _pwm = 0;

	//__bic_SR_register(GIE);
	if (motor < 9)
	{
		_pwm = 944-angle;
	}
	else
	{
		_pwm = angle;
	}


	PWM[2*motor] = (unsigned char) (int) (_pwm/2);
	PWM[2*motor+1] = (_pwm % 2)&0x80;

}

portTickType  StartUp(portTickType lastTime){

	SetAngle(motor_RightLeg1_FEMUR_TIBIA,PLUS_37_50_DEGREES);
	vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

	SetAngle(motor_RightLeg1_COXA_FEMUR,PLUS_37_50_DEGREES);
	vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

	SetAngle(motor_RightLeg1_BODY_COXA,ZERO_00_00_DEGREES);
	vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

	SetAngle(motor_RightLeg2_FEMUR_TIBIA,PLUS_37_50_DEGREES);
```

```
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg2_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg2_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg3_FEMUR_TIBIA,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg3_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg3_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_LeftLeg1_FEMUR_TIBIA,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_LeftLeg1_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_LeftLeg1_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_LeftLeg2_FEMUR_TIBIA,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_LeftLeg2_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_LeftLeg2_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_LeftLeg3_FEMUR_TIBIA,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_LeftLeg3_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_LeftLeg3_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        return lastTime;
}

portTickType  Step(portTickType lastTime){

        SetAngle(motor_RightLeg2_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg1_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg3_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg2_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
```

```
SetAngle(motor_LeftLeg1_BODY_COXA,ZERO_00_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg3_BODY_COXA,ZERO_00_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_RightLeg1_BODY_COXA,ZERO_00_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_RightLeg3_BODY_COXA,ZERO_00_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg2_BODY_COXA,ZERO_00_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

SetAngle(motor_RightLeg2_BODY_COXA,PLUS_22_50_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg1_BODY_COXA,PLUS_22_50_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg3_BODY_COXA,PLUS_22_50_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_RightLeg1_BODY_COXA,MINUS_22_50_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_RightLeg3_BODY_COXA,MINUS_22_50_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg2_BODY_COXA,MINUS_22_50_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

SetAngle(motor_RightLeg2_COXA_FEMUR,PLUS_37_50_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg1_COXA_FEMUR,PLUS_37_50_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg3_COXA_FEMUR,PLUS_37_50_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

vTaskDelayUntil(&lastTime, 10*DELAY / portTICK_RATE_MS);

SetAngle(motor_RightLeg1_COXA_FEMUR,PLUS_60_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_RightLeg3_COXA_FEMUR,PLUS_60_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg2_COXA_FEMUR,PLUS_60_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

SetAngle(motor_RightLeg1_BODY_COXA,ZERO_00_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_RightLeg3_BODY_COXA,ZERO_00_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg2_BODY_COXA,ZERO_00_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_RightLeg2_BODY_COXA,ZERO_00_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg1_BODY_COXA,ZERO_00_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg3_BODY_COXA,ZERO_00_00_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

SetAngle(motor_RightLeg1_BODY_COXA,PLUS_22_50_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_RightLeg3_BODY_COXA,PLUS_22_50_DEGREES);
vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
SetAngle(motor_LeftLeg2_BODY_COXA,PLUS_22_50_DEGREES);
```

```
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg2_BODY_COXA,MINUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg1_BODY_COXA,MINUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg3_BODY_COXA,MINUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg1_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg3_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg2_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        vTaskDelayUntil(&lastTime, 10*DELAY / portTICK_RATE_MS);

        return lastTime;
}

portTickType  PreSpin(portTickType lastTime){

        SetAngle(motor_RightLeg2_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg1_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg3_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg2_BODY_COXA,MINUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg1_BODY_COXA,PLUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg3_BODY_COXA,PLUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg2_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg1_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg3_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        vTaskDelayUntil(&lastTime, 10*DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg1_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg3_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg2_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg1_BODY_COXA,PLUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg3_BODY_COXA,PLUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg2_BODY_COXA,MINUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
```

```
        SetAngle(motor_RightLeg1_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg3_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg2_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        vTaskDelayUntil(&lastTime, 10*DELAY / portTICK_RATE_MS);

        return lastTime;
}

portTickType  Spin(portTickType lastTime){

        SetAngle(motor_RightLeg2_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg1_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg3_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg2_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg1_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg3_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg1_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg3_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg2_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg2_BODY_COXA,PLUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg1_BODY_COXA,MINUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg3_BODY_COXA,MINUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg1_BODY_COXA,MINUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg3_BODY_COXA,MINUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg2_BODY_COXA,PLUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg2_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg1_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg3_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        vTaskDelayUntil(&lastTime, 10*DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg1_COXA_FEMUR,PLUS_60_00_DEGREES);
```

```
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg3_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg2_COXA_FEMUR,PLUS_60_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg1_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg3_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg2_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg2_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg1_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg3_BODY_COXA,ZERO_00_00_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg2_BODY_COXA,MINUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg1_BODY_COXA,PLUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg3_BODY_COXA,PLUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg1_BODY_COXA,PLUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg3_BODY_COXA,PLUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg2_BODY_COXA,MINUS_22_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        SetAngle(motor_RightLeg1_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_RightLeg3_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);
        SetAngle(motor_LeftLeg2_COXA_FEMUR,PLUS_37_50_DEGREES);
        vTaskDelayUntil(&lastTime, DELAY / portTICK_RATE_MS);

        vTaskDelayUntil(&lastTime, 10*DELAY / portTICK_RATE_MS);

        return lastTime;
}
```