UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Vítor Mello de Araujo Lima

Proposta e implementação de novo projeto eletrônico para o robô UARM-E

São Carlos

2018

Vítor Mello de Araujo Lima

Proposta e implementação de novo projeto eletrônico para o robô UARM-E

Monografia apresentada ao Curso de Engenharia Elétrica com Ênfase em Eletrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Valdir Grassi Junior

São Carlos

2018

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Lima, Vítor Mello de Araujo
L732p Proposta e implementação de novo projeto eletrônico para o robô UARM-E / Vítor Mello de Araujo Lima; orientador Valdir Grassi Junior. São Carlos, 2018.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2018.

1. Manipulador Espacial. 2. Arduino. 3. XBee. 4. Robótica. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Vítor Mello de Araujo Lima

Título: "Proposta e implementação de novo projeto eletrônico para o robô UARM-E"

Trabalho de Conclusão de Curso defendido e aprovado em 19 / 06 / 2018 ,

com NOTA 9,5 (nove , meio), **pela Comissão Julgadora:**

Prof. Dr. Valdir Grassi Junior - Orientador - SEL/EESC/USP

Prof. Titular Marco Henrique Terra - SEL/EESC/USP

Prof. Associado Adriano Almeida Gonçalves Siqueira - SEM/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

Dedico a meus pais e meus avós.

AGRADECIMENTOS

Agradeço em primeiro lugar aos meus pais, Lauro e Silvana, que me dão suporte e apoio em todos os momentos.

Agradeço a meu orientador, Professor Valdir Grassi, pelo tempo e conhecimento partilhados comigo. Agradeço também a seu orientando José Nuno, pela grande ajuda em várias etapas deste projeto.

Agradeço ao LASI, pela estrutura disponibilizada para que o projeto pudesse ser executado.

Agradeço à Fundação para o Desenvolvimento Tecnológico da Engenharia, pelos recursos cedidos ao projeto.

Agradeço aos meus amigos, pela paciência, ajuda, e pelos bons momentos.

"Sempre busque ser o melhor! Mas não melhor
que os outros, apenas o melhor de si!"
Marcílio Flávio Rangel de Farias

RESUMO

LIMA, V. **Proposta e implementação de novo projeto eletrônico para o robô UARM-E**. 2018. 78p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

Neste trabalho foi proposto e implementado um novo sistema eletrônico e *software* embarcado para o robô UARM-E, uma plataforma para experimentação de algoritmos de controle em manipuladores espaciais de base livre flutuante. O trabalho utiliza a plataforma Arduino para controle do sistema, e o módulo XBee para comunicação. O sistema busca comandar os motores elétricos nas juntas do robô e fazer o processamento e envio de dados de posição para um computador remoto, onde o algoritmo de controle deve estar sendo executado. Para este fim foi projetada e implementada uma placa de circuito impresso, que faz a interface entre os componentes e deve ir embarcada na plataforma.

Palavras-chave: Manipulador Espacial. Arduino. XBee. Robótica.

ABSTRACT

LIMA, V. **Propositon and implementation of a new electronic system for the robot UARM-E**. 2018. 78p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

In this work, a new electronic system and embedded software for the robot UARM-E was proposed and implemented. This robot is an experimental platform for test and validation of control algorithms that target free flying base spatial manipulators. The system was implemented using an Arduino board and uses a XBee module for wireless communication. The goal is to command the electrical motors in the joints, process the encoder feeedback and send data position to a external computer, where a control algorithm must be running. A printed circuit board was designed and implemented in order to interface the components.

Keywords: Spatial Manipulator. Arduino. XBee. Robotics.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O estudo e desenvolvimento de robótica é de grande interesse para o campo espacial. Nessa área, robôs desempenham tarefas como serviços de resgate e operações de manutenção em órbita. Esses sistemas são usados desde atividades onde a habilidade humana é limitada, até a atuação em áreas inacessíveis ou com grande risco à vida humana. Nesse contexto, robôs manipuladores possuem grande importância.

Robôs manipuladores espaciais são projetados como estruturas leves, mas que possuem longos braços. Esse design permite a economia de combustível e largo alcance para trabalho. Outra característica desses sistemas é o acoplamento dinâmico entre base e braço robótico. Os sistemas podem ser de base livre controlada, onde a orientação da base espacial é ativamente controlada por jatos propulsores e/ou rodas da reação, ou de base livre flutuante, onde a base se move livremente em resposta aos movimentos do braço.

Para o estudo de controladores e validação de técnicas de controle para robôs espaciais de base livre flutuante, é necessário a montagem de plataformas experimentais próprias para este fim. Uma característica importante dessas plataformas é a necessidade de que nenhuma conexão física, cabos ou fios, exista entre robô e ambiente. Assim, é possível que o sistema flutue sem interferências mecânicas. Em um sistema assim, toda a eletrônica de controle e alimentação de componentes deve ser embarcada na base flutuante, e a comunicação do sistema com um computador remoto deve ser feita utilizando um padrão sem fio.

Nesse contexto que o presente trabalho se insere. Tendo a disposição a estrutura mecânica e do robô UARM-E no LASI (Laboratório de Sistemas Inteligentes) e havendo a necessidade de um novo sistema eletrônico para utilização do robô. O projeto visa implementar esse sistema, que deve permitir o envio de comandos a partir de um computador remoto para a plataforma, atuação nos motores que geram o movimento do robô, e leitura e processamento dos sinais elétricos advindos do sensoriamento, de forma a enviar as informações para o computador externo.

Para controle do sistema eletrônico, será utilizado a plataforma Arduino. Trata-se uma plataforma de prototipagem eletrônica de software livre, que possui como vantagens sua acessibilidade, baixo custo e flexibilidade. Além disso, essa plataforma possui uma gama de acessórios, que podem ser combinados em um sistema mais completo.

Entre esses acessórios encontra-se o Xbee Shield, que permite a comunicação sem fio do Arduino utilizando padrão ZigBee. Ele é baseado no módulo Xbee da Digi, e permitirá a troca de dados entre o sistema embarcado e o computador remoto. Consistindo no módulo de comunicação.

1.1 Justificativa

O projeto se justifica pois se tem no LASI linhas de pesquisa na área de controle de manipuladores espaciais, e se dispõe de uma plataforma para teste e validação dessas estratégias, cuja eletrônica pode ser modernizada e sua complexidade reduzida, dessa forma havia grande motivação para implementação de novo *hardware*.

1.2 Objetivos

O presente trabalho de conclusão de curso tem como objetivo a implementação um sistema eletrônico moderno e de baixa complexidade em uma plataforma experimental de manipulador espacial de base flutuante, utilizando a plataforma Arduíno Mega, e comunicação sem fio usando padrão *Zigbee*, bem como o teste e validação da estrutura implementada.

2 ROBÔS MANIPULADORES

O termo robô foi utilizado pela primeira vez pelo dramaturgo tcheco Karel Capek, em sua peça *Rossumovi Univerzální Roboti* (Robôs Universais de Rossum), sendo "*robota*"a palavra tcheca para trabalhador (SPONG, 2006). Na peça, os *roboti* são humanos artificiais construídos para trabalhar no lugar dos humanos reais. Apesar dos robôs atuais não serem construídos a partir de matéria orgânica sintética, como Capek imaginava, o objetivo de utilizá-los para realização das mais diversas tarefas se mantém.

Segundo uma definição mais atual da palavra, da *Robot Institution of America* (RIA): "*Robô é um manipulador multifuncional, reprogramável projetado para mover materiais, ferramentas, ou dispositivos especializados através de movimentos variáveis programados para desempenhar uma variedade de tarefas*". A definição chama a atenção para as características "reprogramável"e "multifuncional", que evoluíram muito com o passar do tempo.

Inicialmente robôs eram utilizados para substituição de mão de obra humana em tarefas repetitivas, como moldagem por injeção de termoplástico ou estampagem (Figura 1a), onde a programação se limitava a seguir sequências de movimentos predefinidos, sem variação ou uso de sensores (SPONG, 2006). A evolução dos sistemas computacionais tornou possível movimentos mais complexos e o uso de sensoriamento externo (como visão, sensores de força e velocidade). Com isso os sistemas robóticos passaram a ser mais conscientes de seu entorno, e a ser capazes de atividades como cirurgias, exploração submarina e de outros planetas (Figura 1b).

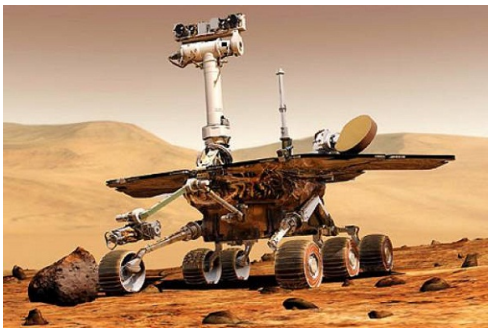
Compõe o corpo dos robôs manipuladores corpos rígidos denominados *elos*, que

Figura 1 – Exemplos de sistemas robóticos para atividades com diferentes graus de complexidade.

(a) Robô utilizado para moldagem por injeção de termoplástico.

Fonte: Fanuc Robotics.

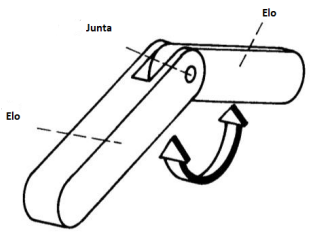
(b) Spirit, um dos *rovers* utilizados pela NASA para exploração de Marte.



Fonte: NASA.

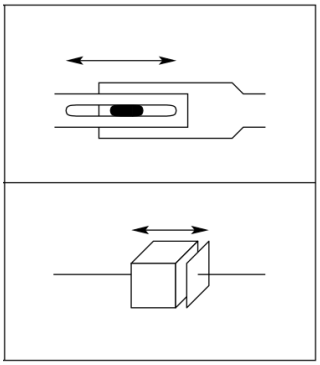
Figura 2 – Tipos mais usuais de juntas de manipuladores.

(a) Estrutura de elos e juntas com junta de rotação.



Fonte: Teodoro e Augusto (2012).

(b) Junta de translação (3D e representação simbólica).



Fonte: Spong (2006).

são conectados por meio de *juntas*, que possibilitam a movimentação entre elos adjacentes (LYNCH; PARK, 2017). Essa estrutura pode ser visualizada na Figura 2a. Existem vários tipos de juntas, mas as mais comuns são as de rotação e prismáticas (Figura 2b). O primeiro tipo permite o movimento de revolução, enquanto que o segundo tipo executa o movimento de translação.

Ao final da cadeia de elos e juntas econtra-se o *efetuador*, que na prática pode ser uma garra, uma ponta de solda, ou mesmo um eletroimã, dependendo da função do manipulador (CRAIG, 2005). Problemas comuns envolvendo manipuladores são o cálculo de velocidade ou posição do efetuador. Na outra extremidade encontra-se a *base*, geralmente fixa, e tida como referência para o sistema.

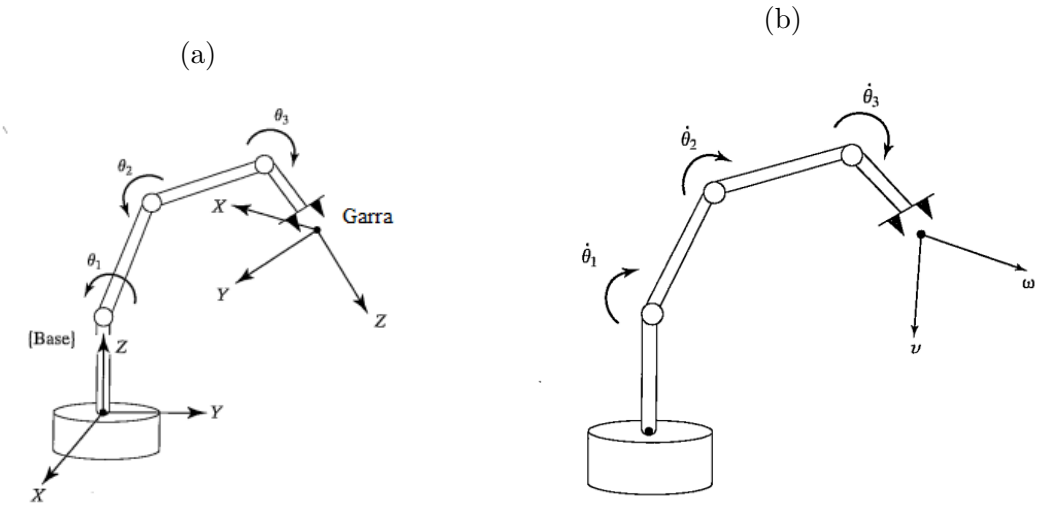
Outros elementos que podem ser citados são *atuadores* e *sensores*. Os atuadores são responsáveis por transformar energia em movimento (BENSON, 2010). É comum que a energia em robótica seja elétrica, e podemos citar os motores elétricos como tipos comuns de atuadores, que transformam a energia elétrica no movimento de rotação. Os sensores são responsáveis por traduzir os eventos do ambiente para o programa que controla o robô. Podem ser dispositivos simples, como sensores de força ou luminosidade, ou componentes mais complexos, como câmeras digitais que fornecem vídeo e imagem para o robô.

2.1 Cinemática

A cinemática é a ciência que estuda o movimento sem se preocupar com as forças que geram esse movimento (CRAIG, 2005). A aplicação de cinemática em manipuladores visa o estudo de grandezas como posição, velocidade e aceleração, relacionado à suas juntas ou ao efetuador. De forma geral, são estudadas as grandezas geométricas e as propriedades do movimento que variam com o tempo.

Isso pode ser feito, por exemplo, colocando sensores de posição nas juntas do

Figura 3 – Em (a) é exibido a representação simbólica de um manipulador de base fixa com três juntas de rotação. Tem-se em (b) a representação das velocidades no espaço de juntas e da velocidade do efetuador no espaço cartesiano.



Fonte: Craig (2005)

manipulador. No caso das juntas de rotação, cada sensor proporcionaria o ângulo de rotação. Já para juntas prismáticas, o dado obtido seria o deslocamento (*offset*). Apresentadas essas variáveis, é possível introduzir o conceito de *graus de liberdade*, termo típico no estudo de mecanismos. Este termo designa o número de variáveis de posição independentes que devem ser especificadas para encontrar as partes do mecanismo (CRAIG, 2005). Nos manipuladores, de forma geral, este número costuma ser igual ao de juntas.

Uma vez conhecidos os ângulos de rotação, os *offsets* das juntas prismáticas, e as dimensões do manipulador, pode-se obter a posição do efetuador. Esse processo, onde se utiliza as variáveis de posição das juntas para determinar a posição do efetuador, é chamada de *cinemática direta*. A Figura 3a ilustra um sistema onde esse problema se aplica. O conhecimento de θ_1, θ_2 e θ_3, além das dimensões do sistema, permite determinar a posição da garra em relação à base, utilizando relações geométricas.

Esse problema, todavia, não é o que se costuma enfrentar na prática. Em situações reais, deseja-se mover o efetuador do manipulador para uma posição específica com determinada orientação, para pegar um objeto ou manusear um material, por exemplo. Nesse caso, o desafio é: Dado a posição e orientação de destino, que ângulo ou deslocamento se deve impor nos atuadores do robô para levar o efetuador até lá.

Este problema se chama *cinemática inversa*, e sua resolução envolve maior complexidade que o caso direto. Isso se deve ao fato de que os cálculos do caso inverso envolvem equações não-lineares, e podem inclusive não possuir solução. A resolução do problema de cinemática inversa permite definir o *espaço de trabalho* do manipulador, que é composto pelos pontos onde existe solução para o problema. As posições onde não existe solução são

definidas como fora do alcance do espaço de trabalho.

Além dos problemas de posicionamento estático, também se pode fazer a análise do movimento do manipulador. O objetivo pode requerer que o efetuador, em seu deslocamento, mantenha certa velocidade angular e linear. Uma vez que os atuadores estão localizados nas juntas, é necessário que se consiga, a partir da velocidade desejada no efetuador, determinar as velocidades a serem aplicadas nas juntas (Figura 3b). Isso é feito por uma matriz chamada *Jacobiano* do Manipulador, que especifica o mapeamento das velocidades no espaço das juntas para o espaço cartesiano (CRAIG, 2005). Esse mapeamento é ilustrado pela Equação 2.1, onde v e w são as velocidades linear e angular, respectivamente, J é o Jacobiano para uma configuração q das juntas e $\dot{q}$ é o vetor com as velocidades nas juntas.

$$\begin{bmatrix} v \\ w \end{bmatrix} = J(q)\dot{q} \tag{2.1}$$

Com relação à matriz Jacobiano, ela possui grau (número de colunas com variáveis independentes) máximo igual ao número de juntas no manipulador. O termo "máximo"é utilizado pois, em algumas configurações, o grau da matriz pode assumir valores menores. As configurações em que isso ocorre são chamadas de *singularidades*, e na prática, essas são configurações onde o manipulador perde a capacidade de se movimentar em certas direções. Geralmente os pontos de singularidade pertencem à fronteira do espaço de trabalho (SPONG, 2006). Este é um conceito bastante importante para projetistas e usuários de manipuladores.

Outra aplicação do Jacobiano é na aplicação de forças estáticas em determinado corpo. Além de movimentação, pode ser necessário ao manipulador que toque um objeto ou superfície com uma força constante. Dado esse objetivo, é necessário traduzir isso em torques que devem ser aplicados nas juntas. Nesse caso, obtemos a Equação 2.2, onde utilizamos a matriz transposta do Jacobiano, o vetor de forças no efetuador F para obter τ, que é o vetor de torques a ser aplicado nas juntas.

$$\tau = J(q)^T F \tag{2.2}$$

2.2 Dinâmica

Em situações reais, para retirar o efetuador da posição de repouso, acelerá-lo até determinada velocidade, e posteriormente freá-lo para que pare na posição desejada, é preciso levar em consideração as *forças que causam o movimento*, que não são observadas na análise cinemática. Nesse caso, é necessário recorrer à *Dinâmica*, que descreve as relações entre força e movimento. Além de necessárias para o projeto de manipuladores e dos

algoritmos de controle aplicados a eles, as equações de movimento também são utilizadas para simulações de comportamento e animações.

Elas levam em consideração não só os aspectos geométricos do sistema, mas propriedades físicas também, como a massa e inércia dos componentes. O problema agora passa a ser: Dado a movimentação desejada no efetuador, quais valores de torque devem ser aplicados em cada uma das juntas? Há mais de uma maneira de resolver esse problema.

Um dos métodos que pode ser utilizado é através das *Equações de Euler-Lagrange*. Primeiro é necessário conhecer o operador *Lagrangiano*, exibido na Equação 2.3, que corresponde à diferença entre a energia cinética ($\mathcal{K}$) e a energia potencial devido a gravidade ($\mathcal{P}$). Então, para um sistema com coordenadas $(q_1, ..., q_n)$, onde n é o número de graus de liberdade, a força (τ) para cada coordenada será encontrada aplicando a relação descrita na Equação 2.4 (SPONG, 2006).

$$\mathcal{L} = \mathcal{K} - \mathcal{P} \tag{2.3}$$

$$\frac{d}{dt}\frac{\delta \mathcal{L}}{\delta \dot{q}_k} - \frac{\delta \mathcal{L}}{\delta q_k} = \tau_k; k = 1, ..., n \tag{2.4}$$

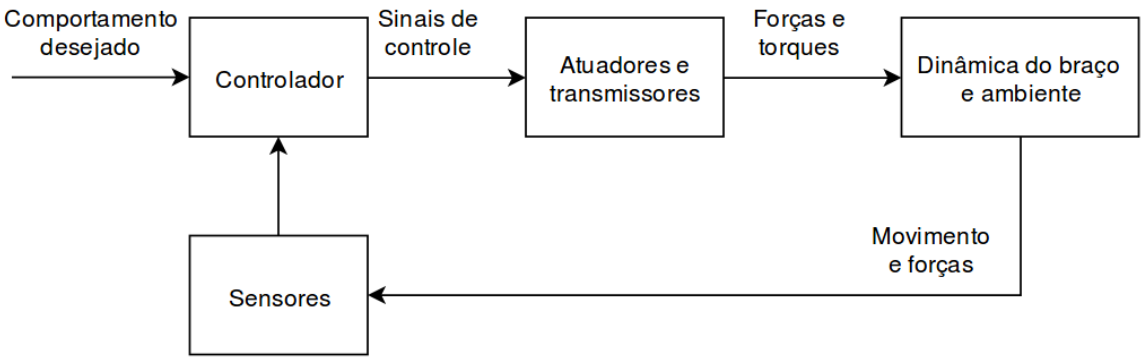
2.3 Controle

Quando em atividade, manipuladores podem se comportar como fontes de força, movimento, ou mesmo ambos. Se o manipulador está realizando o polimento de um objeto, ele deve responder com a força necessária para se manter estático durante o trabalho. Se ele está realizando a pintura de uma superfície, deve operar em uma trajetória adequada, que faça com que pinte a totalidade da supérficie. Se sua tarefa é realizar um desenho em um quadro, o movimento é importante, mas a força direcionada à superfície do quadro também é.

Em todos esses casos, o *controlador* deve traduzir o objetivo da tarefa em forças e torques que devem ser aplicados aos atuadores. Deve-se escolher uma estratégia de controle que seja compatível com o objetivo. Essa estratégia pode ser: controle de movimento, de força, híbrido ou de impedância. No caso do controle híbrido, é importante ressaltar que, por restrições inerentes à mecânica do sistema, não se pode controlar força e movimento em uma mesma direção, ou seja, se o robô impõe uma força nessa direção, o ambiente determinará o movimento, e se o controle impõe um movimento, caberá ao ambiente definir a força (LYNCH; PARK, 2017).

Definida a estratégia, podemos implementá-la utilizando *controle realimentado*. Esse método, utilizado em quase todos os sistemas robóticos, consiste em utilizar sensores de posição, velocidade e força para obter o comportamento real do robô. Estes dados

Figura 4 – Diagrama de controle realimentado para um sistema robótico.



Fonte: Adaptado de Lynch e Park (2017)

são comparados com a *referência* do sistema, que é o comportamento desejado para o manipulador. Com base no resultado dessa comparação, é decidido o valor dos sinais de controle, que são aplicados nos atuadores. Um diagrama de blocos ilustrando esse sistema pode ser visto na Figura 4.

2.4 Manipulador Espacial

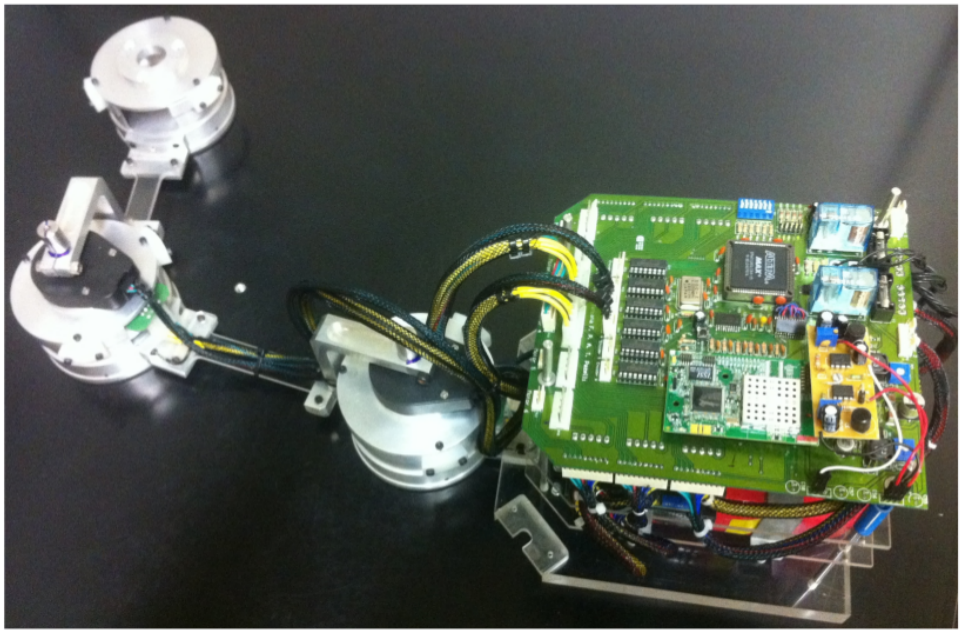
Manipuladores robóticos são utilizados no ambiente espacial, realizando atividades como inspeção de naves, posicionamento, acoplamento e transporte de materiais. Esses manipuladores, por atuarem em órbita, possuem uma dinâmica diferenciada daqueles que atuam na Terra. Existe neles um acoplamento dinâmico entre a base e o braço. Isso significa que a movimentação do braço provocará movimentos na base.

Pode-se classificar estes manipuladores em duas categorias: Manipuladores de base livre controlada e manipuladores de base livre flutuante (PAZELLI et al., 2011). No primeiro caso, o uso de jatos propulsores e rodas de reação permite neutralizar os movimentos da base, porém isso acarreta gastos com energia e combústivel ao sistema, o que reduz sua longevidade. No segundo caso, esses gastos não ocorrem, se permite que a base se mova livremente. Por essa razão, se justificam os estudos na área de manipuladores de base livre flutuante.

Desenvolver plataformas de teste para estes manipuladores, na Terra, é um desafio. Essas plataformas devem ser capazes de simular os efeitos da microgravidade, e permitir que braço e base se movam livremente. Algumas possibilidades são realizar experimentos em vôos parabólicos ou com estruturas submersas, porém uma maneira mais simples é utilizar mesas com sistema de sustentação por colchão de ar. Utilizando robôs planares, Esse método simula de maneira eficiente os efeitos da microgravidade em 2D (MENON; BUSOLO, 2007).

O LASI possui uma plataforma para testes que utiliza o sistema de sustentação

Figura 5 – Plataforma para testes utilizando colchão de ar disponível no LASI.



Fonte: Pazelli et al. (2011).

por colchão de ar. O robô UARM-E (Figura 5), desenvolvido pela Profa. Dra. Tatiana Pazzelli durante seu doutorado, visa permitir o teste e validação de algoritmos de controle para manipuladores espaciais de base livre flutuante.

O robô é reconfigurável, podendo ser montado em diferentes configurações, com um ou dois braços, mas com um limite de 6 juntas atuadas. O braço mecânico se conecta a base, onde se localiza o sistema eletrônico e as baterias. O algoritmo de controle é executado em um computador remoto, e a comunicação com a plataforma de testes é sem fio.

O sistema eletrônico é baseado no microcontrolador Rabbit 4000. Foi observado que, utilizando componentes mais novos, seria possível a implementação de um sistema menos complexo e mais moderno. Sendo assim, foi proposto e implementado um novo sistema eletrônico para o robô, que é detalhado no capítulo seguinte.

3 METODOLOGIA E DESENVOLVIMENTO

Esta seção trata do novo projeto eletrônico e de *software* embarcado proposto para o robô UARM-E. Na Figura 6 pode-se ver a topologia do sistema proposto, que será detalhada no decorrer deste capítulo.

Os blocos em verde fazem parte do projeto eletrônico e são discutidos na seção 3.1, enquanto que os blocos em cinza referenciam os programas implementados, e são discutidos na seção 3.2. A linha tracejada se diferencia das demais para indicar que a conexão do módulo de comunicação com a interface com o *software* de controle, que é executado em uma máquina externa, é sem fio. Isso é feito pois a plataforma não pode ter conexões físicas que insiram interferências mecânicas na flutuação (PAZELLI et al., 2011). O bloco amarelo (*software* de controle) aparece em cor diferente para indicar que não será discutido neste trabalho.

3.1 Projeto eletrônico

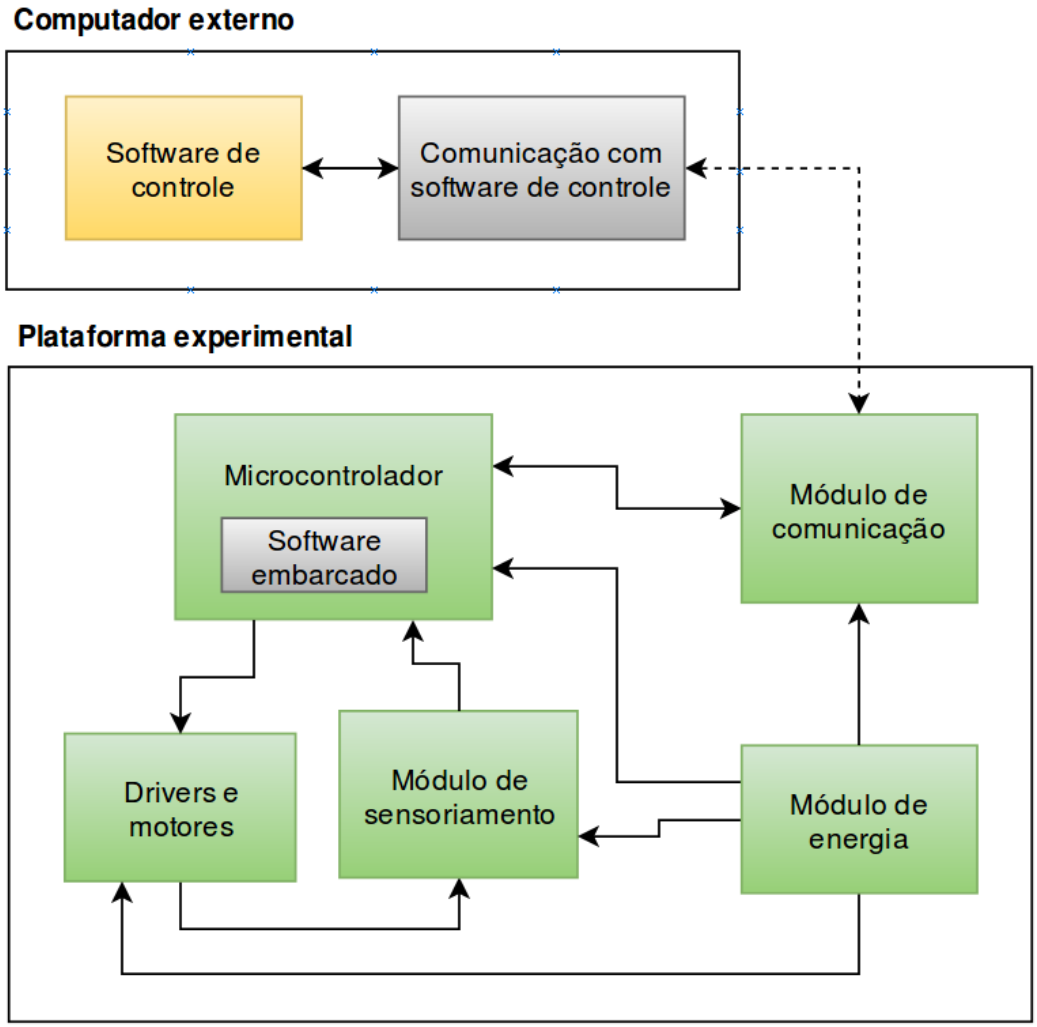
3.1.1 Microcontrolador

O Arduino é uma plataforma eletrônica *open-source* que busca prover um conjunto de *hardware* e *software* fácil de se trabalhar, possibilitando que ele seja usado em vários campos, como robótica, educação e arte (Arduino LLC, 2018).

Por possuir essa vasta gama de aplicações, a plataforma possui extensa documentação, exemplos de projetos e fóruns de disscussão, o que facilita o trabalho e a troca de informações, especialmente para aqueles que estão começando a trabalhar com microcontroladores. Além disso, a plataforma possui uma linha com mais de vinte placas, que possuem foco em diversos nichos, como educação, projetos em *IoT*, robótica e *wearables* (Arduino LLC, 2018). Outro diferencial é a grande variedade de *shields* disponíveis, que são elementos que podem ser encaixados nas placas Arduino e realizam a interface com vários componentes, como cartões SD, módulos *WiFi*, *bluetooth* e *Xbee*. Este último foi utilizado no projeto, sua funcionalidade é descrita na subseção 3.1.2.

Para o projeto, foi escolhido o Arduino Mega 2560 (Figura 7), que é uma placa baseada no microcontrolador ATmega2560. Entre as particularidades que justificaram a escolha dessa placa, está o grande número de pinos destinados à entrada e saída que essa placa possui, qualidade necessária para este projeto, que exige do arduino a leitura de dezenas de sinais. Outras vantagens são o grande número de saídas PWM, necessárias para controle de motores, e a boa quantidade de memória disponível, que permite gravar programas mais extensos. As especificações da placa podem ser vistas na tabela 15, Apêndice A.

Figura 6 – Esquema geral do sistema.



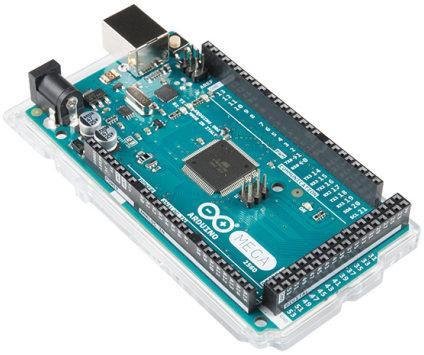
Fonte: Autor.

3.1.2 Módulo de Comunicação

O XBee S1 (Figura 8) é um módulo RF desenvolvido pela Digi, que atende à especificação de comunicação IEEE 802.15.4, também usada no padrão ZigBee, que busca entregar módulos de baixo custo e consumo para redes de sensoriamento sem fio (Digi International, 2018a). Conforme análise feita em Pazelli et al. (2011), a taxa de bits necessária para comunicação no sistema é de 32 kbps, sendo a taxa de 250 kbps do XBee S1 mais que suficiente para a tarefa. Esse dispositivo também apresenta baixo consumo de energia, o que é importante já que o sistema é alimentado por baterias, e possui alcance apropriado para projeto. As especificações do módulo podem ser visualizadas na Tabela 16, Apêndice A.

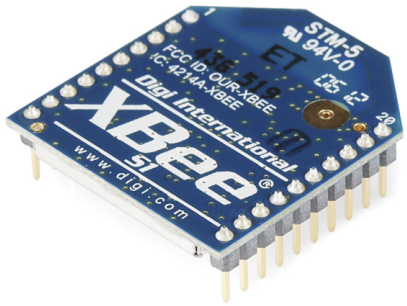
Para utilizar o XBee, deve-se conectá-lo na interface UART do dispositivo que irá se comunicar com ele, ou em um conversor, caso o dispositivo não possua a interface.

Figura 7 – Arduino Mega 2560, placa com microcontrolador utilizada no projeto.



Fonte: www.sparkfun.com.

Figura 8 – Módulo RF XBee S1.



Fonte: www.sparkfun.com.

Ambos devem possuir configurações compatíves de *baud rate* (velocidade da comunicação), paridade (se a soma de bits 1 do pacote é par ou ímpar), número de bits de parada, e número de bits que compõe o dado propriamente transmitido. No projeto, foram utilizados os valores mais usuais para esses campos, de forma a simplificar o protocolo de comunicação, e não causar atrasos desnecessários. Os valores são exibidos na Tabela 1.

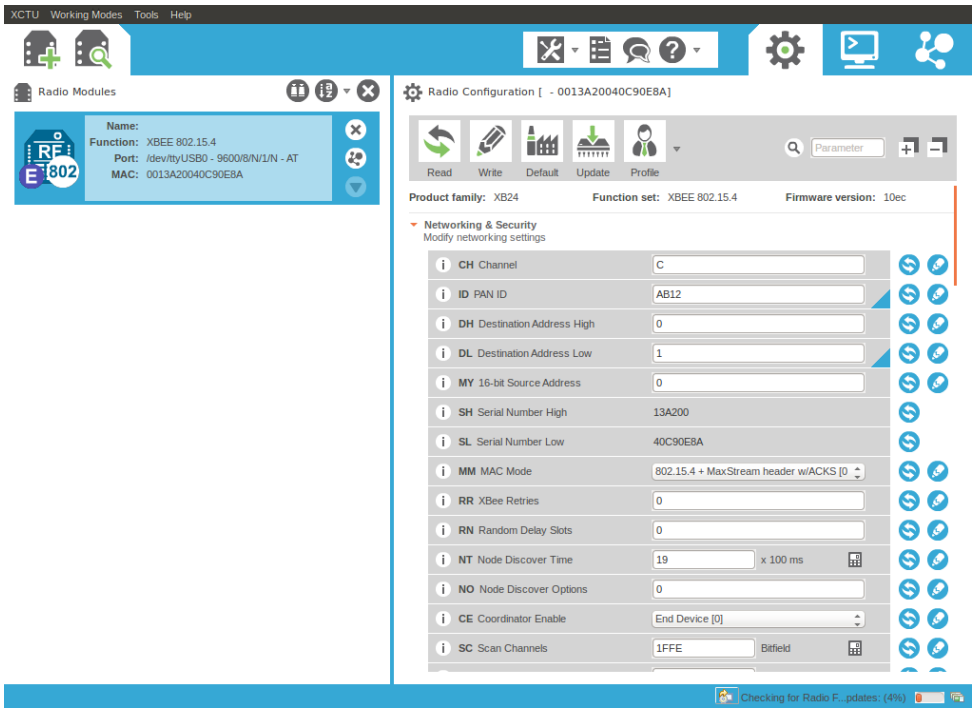
Tabela 1 – Configurações de comunicação utilizadas no projeto.

Configuração	**Valor**
Baud Rate	9600
Bits de dado	8
Bits de paridade	Nenhum
Bits de parada	1

No microcontrolador, essas configurações são definidas no código que é compilado e enviado à placa. No XBee, elas podem ser enviadas via comandos AT, mas a forma mais simples é através do programa XCTU (Figura 9). Ele é uma aplicação disponibilizada pela Digi que possui interface gráfica e permite a configuração de diversos módulos XBee de maneira muito fácil (Digi International, 2018b). Pode-se com ele gerenciar vários dispositivos, programá-los, enviar pacotes e visualizar pacotes recebidos, atualizar firmware, entre outras funcionalidades.

Para fazer a interface entre o módulo e o computador no qual está instalado o programa, deve-se utilizar um conversor USB para serial. No projeto, foi utilizado a placa XBee Explorer USB (Figura 10). Além das configurações que devem ser feitas para que os XBee se comuniquem com o microcontrolador e computador, é preciso configurá-los também para que se comuniquem entre si.

Figura 9 – Interface do programa XCTU utilizado para configurar os módulos XBee, aqui é exibida a configuração do módulo que fica ligado ao computador.



Fonte: Autor.

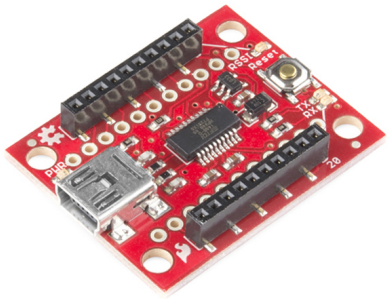
Para que os XBee se encontrem na rede, é necessário que ambos estejam programados para utilizar o mesmo *canal*, que o *Personal Area Network ID* (PAN ID) de ambos seja o mesmo, e que cada um tenha como endereço de destino o outro. As configurações feitas, via XCTU, em ambos estão disponíveis na tabela Tabela 2. Os valores escolhidos são aleatórios, mas diferentes dos valores padrões, de forma a dificultar interferência de outros dispositivos XBees que possam vir a funcionar nas proximidades do sistema.

Tabela 2 – Configuração dos XBees utilizados no projeto.

Configuração	**XBee no UARM-E**	**Xbee no computador remoto**
Canal	C	C
PAN ID	AB12	AB12
Endereço de Destino	0x01	0x00
Endereço de Fonte	0x00	0x01

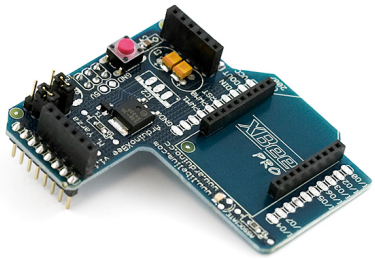
Como comentado na subseção 3.1.1, existe um *shield* (Figura 11) para a plataforma Arduino que faz sua interface com o XBee. Esse elemento se faz necessário porque Arduino e XBee usam níveis de tensão diferentes, o primeiro é alimentado e trabalha com sinais em 5V, enquanto que o último funciona com tensões de 2,8 a 3,4V. Logo, é necessário um circuito que abaixe a tensão de 5V do microcontrolador para alimentar o dispositivo RF e converta os níveis lógicos dos sinais de comunicação para que o XBee não seja danificado por altas tensões. O *shield*, que pode ser simplesmente encaixado no Arduino, torna esse

Figura 10 – Placa XBee Explorer USB, utilizada no projeto para conversão de USB para serial.



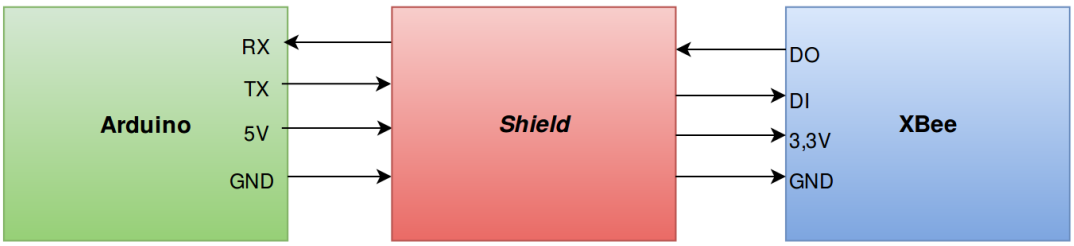
Fonte: www.sparkfun.com.

Figura 11 – *Shield* que realiza a interface entre Arduino e XBee.



Fonte: www.filipeflop.com

Figura 12 – Interface entre Arduino e XBee utilizando o *shield*.



Fonte: Autor.

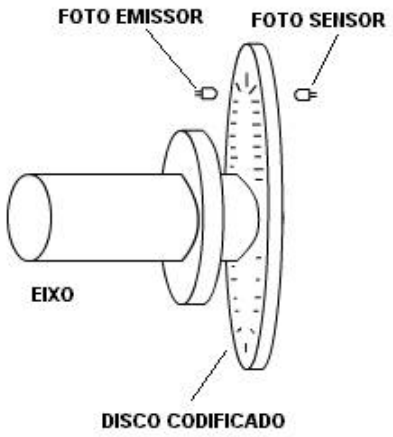
trabalho transparente para o usuário.

A interface entre Arduino e XBee, utilizando o *shield*, é ilustrada na Figura 12. O pino de transmissão de dados do Arduino (TX) tem sua tensão rebaixada antes de ser percebida pelo pino *Data In* (DI) do XBee. Os processo inverso ocorre com o pino de saída de dados do XBee (DO) antes de ser percebido pelo pino de recepção (RX) do Arduino.

3.1.3 Módulo de Sensoriamento

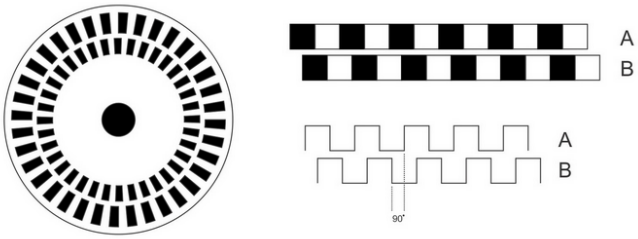
Na placa, o sensoriamento dos motores é feito através de *encoders* de rotação, que monitoram o giro do eixo do motor e possibilitam extrair informações de posição e movimento. Mais especificamente, os *encoders* utilizados são ópticos e incrementais. Eles são denominados incrementais porque sua saída consiste em duas ondas quadradas em quadratura (defasadas de 90º), que correspondem à incrementos no movimento de rotação. Essas ondas são geradas por um sistema óptico: Dois feixes de luz, defasados de 90°, passam por um disco que possui segmentos opacos e transparentes. A passagem dos feixes por estes segmentos cria padrões claro-escuro, que são capturados por sensores ópticos

Figura 13 – Sistema óptico de um *encoder* incremental.



Fonte: Adaptado de Eitel (2014).

Figura 14 – Códigos Gray gerados pelos canais A e B.



Fonte: Adaptado de Creative Robotics Ltd. (2018).

(em geral fotodiodos), que traduzem esses padrões em duas ondas quadradas, comumente denominadas canais A e B (EITEL, 2014). A Figura 13 ilustra o funcionamento desse sistema.

Os canais A e B geram uma sequência do tipo código Gray, onde de um dígito para o outro apenas um bit varia. Analisando essa sequência, consegue-se identificar o sentido em que o motor está girando, como pode ser visto nas Figura 14. Essa sequência também descreve quatro posições bem definidas que se consegue perceber no período de um pulso. Sabendo que a resolução do encoder é de 2 mil pulsos por volta, com a decodificação das ondas em quadratura a resolução é expandida para 8 mil pulsos por volta.

Na placa, o trabalho de traduzir as ondas quadradas para direção e posição angular fica a cargo do componente eletrônico HCTL-2017. Trata-se de um decodificador de quadratura e contador de pulsos. Este circuito tem como entrada as ondas quadradas referentes aos canais A e B, e, após a decodificação, fornece um número de 16 bits correspondente à posição angular relativa do eixo. A Figura 15 exibe um diagrama de blocos com as conexões entre Arduino, decodificador e encoder. Apesar de exibido apenas

para um encoder, o circuito se replica para os 4 decodificadores presentes na placa.

Sabendo que a resolução é de 8 mil pulsos por volta, para converter o número lido em rad/s basta aplicar a equação 3.1, onde θ representa a posição angular em rad/s e N representa o número de 16 bits retornado pelo decodificador.

$$\theta = \frac{N}{8000} \times 2\pi \tag{3.1}$$

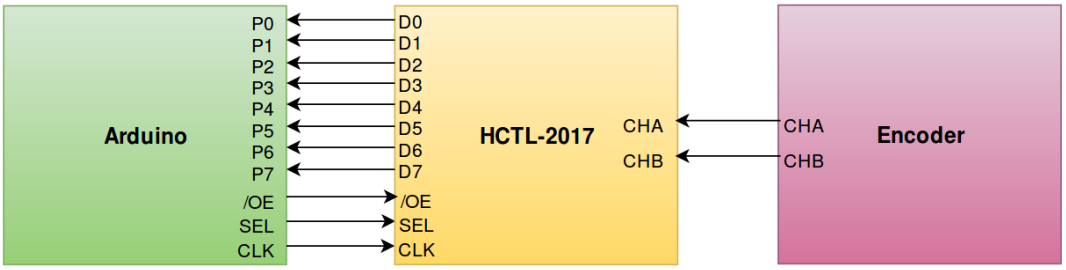
Apesar de retornar um número de 16 bits, pode-se observar na Figura 15 que o CI possui apenas 8 pinos para saída de dados (D0 à D7). Ocorre que é preciso fazer duas leituras para obter o dado completo. Na primeira são lidos os 8 bits mais significativos (MSB) e na segunda os 8 bits menos significativos (LSB). Os pinos de controle OE e SEL são utilizados para fazer este controle, como mostrado na tabela 3, onde a saída Z indica alta impedância. Estes sinais são compartilhados pelos 4 decodificadores, uma vez que os dados dos 4 são lidos simultaneamente.

Outro detalhe que pode ser observado na Figura 15 é que o CI necessita de um sinal de sincronização (CLK). Este sinal pode ser de até 14 MHz, conforme a Tabela 17. Segundo Pazelli et al. (2011), a frequência mínima de relógio que deve ser fornecida, para que a leitura funcione mesmo em velocidade máxima do motor, é de 600 kHz. Para o projeto, utilizou-se 1 MHz de frequência.

Tabela 3 – Controle da saída de dados do CI HCTL-2017.

$\overline{OE}$	SEL	Saída
1	X	Z
0	0	MSB
0	1	LSB

Figura 15 – Diagram de blocos com conexões entre Arduino, decodificador de quadratura e os sinais de saída dos *encoders*.

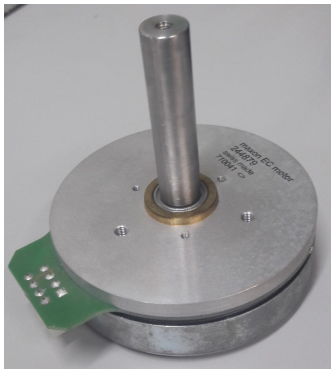


Fonte: Autor.

3.1.4 *Drivers* e Motores

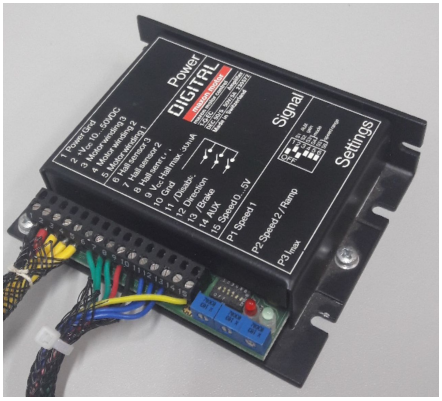
O projeto utilizou os *drivers* e *motores* disponíveis no LASI que fazem parte do UARM-E. O motor utilizado é o *Maxon EC 90 flat* (Figura 16), cujas especificações estão disponíveis na Tabela 21, Apêndice A. Esse motor compõe as juntas rotativas do manipulador, tendo como características interessantes sua base larga e distribuição uniforme de massa, que facilita a flutuação.

Figura 16 – Motor Maxon EC 90 flat.



Fonte: Autor.

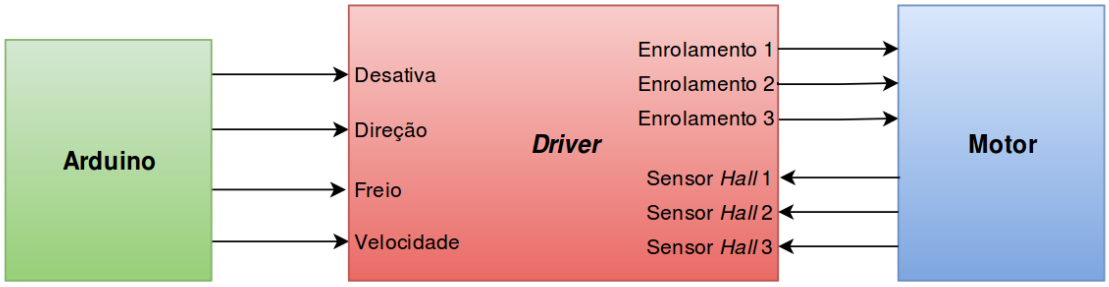
Figura 17 – DEC 50/5.



Fonte: Autor.

O *driver* utilizado é o dispositivo Maxon DEC 50/5 (Figura 17). Este componente consiste em um amplificador para controle digital de motores CC. Ele realiza a interface entre os sinais que partem do Arduino e os motores. A Figura 18 demonstra de que forma essa interface é feita. Este esquema é repetido quatro vezes, já que a placa foi projetada para controle de até 4 juntas atuantes.

Figura 18 – Interface do *driver* com Arduino e motores.



Fonte: Autor.

O sinal *Desativar* bloqueia o estágio de potência, encerrando o controle sobre o motor. O sinal *Direção* controla o sentido (horário ou anti-horário) de giro do motor. O sinal *Freio* faz com que o motor desacelere, e trava o eixo do motor, impedindo o giro até que seja desabilitado. O sinal *Velocidade* é ligado a uma saída *PWM* do arduino. A tensão

média na saída PWM varia de 0 a 5V em 256 passos. A partir disso se pode definir o passo de torque mínimo $M_{B_{min}}$ através da Equação 3.2, onde $\Delta n/\Delta m$ representa o gradiente de velocidade/torque do motor, K_n a constante de velocidade do motor e V_{CC} o valor da tensão de alimentação da placa. Para um dado torque M_B, o valor de tensão médio V_{in} que deve ser gerado pelo PWM do Arduino é dado pela Equação 3.3.

$$M_{B_{min}}[mNm] = \frac{V_{CC}K_n}{256(\Delta n/\Delta M)} \tag{3.2}$$

$$V_{in}[V] = \frac{5(\Delta n/\Delta M)}{V_{CC}K_n}M_B \tag{3.3}$$

3.1.5 Módulo de Energia

Como mostrado na Figura 6, o módulo de energia deve fornecer alimentação para os demais módulos. O sistema possui cerca de 4 níveis de tensão de alimentação diferentes, que devem ser suportados pelo módulo de energia, são eles:

- 18 a 22,2V: Alimentação dos *drivers*.
- 7 a 12V: Alimentação do Arduino Mega.
- 3,3V: Alimentação do XBee.
- 5V: Demais componentes.

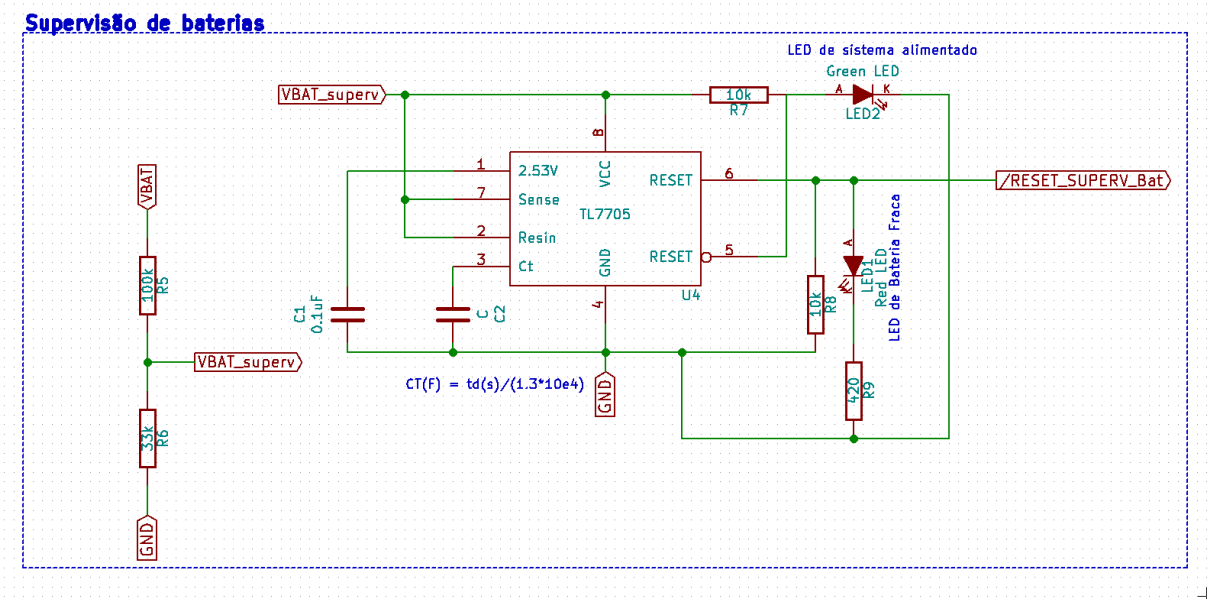
A estratégia utilizada foi alimentar o circuito com a maior tensão e inserir reguladores na placa para abaixar essa tensão nas regiões que demandam menor alimentação. A maior tensão, de 22,2V, é obtida por meio de uma bateria recarregável de Lítio-íon. Essa é a mesma bateria utilizada no projeto eletrônico original. Ela deve ser recarregável, pois não se conhece o intervalo de tempo entre os experimentos que serão feitos. Além disso, baterias de Lítio-ion se destacam pelo seu baixo peso, alta tensão e segurança (em relação à explosões) por utilizarem eletrólito sólido em vez de líquido (PAZELLI et al., 2011).

Ao contrário do projeto original, no qual foram utilizados duas baterias, usou-se apenas uma para alimentação do circuito descrito nesta tese. Com a redução do número de motores, a demanda energética da placa caiu em cerca de um terço, já que o consumo dos componentes eletrônicos é muito baixo e pode ser desprezado. Somado ao fato de que os testes com essa plataforma costumam ser rápidos e envolver baixas cargas, optou-se por essa simplificação no circuito, que também possibilitou descartar a lógica necessária para chaveamento entre baterias.

Outro detalhe com relação à bateria é que a tensão por célula, normalmente de 3,7V, não pode ficar abaixo de 3V (PAZELLI et al., 2011). Para uma bateria de 6 células

isso implica que a tensão mínima de funcionamento é 18V, abaixo disso o sistema deve ser desligado. Para isso, foi utilizado um supervisor de bateria, o CI TL7705. Como pode ser visto na Figura 19, utiliza-se um divisor de tensão para obter o valor de um quarto da tensão nos pinos *Resin* e *Sense*. Se este valor estiver abaixo da tensão de limiar (4,55V), o sinal de *RESET* vai para alto. Este sinal está ligado ao microcontrolador, que desabilita os motores para que se pare de drenar corrente.

Figura 19 – Lógica implementada para supervisão da bateria utilizando o CI TL7705, *VBAT* é a tensão da bateria.

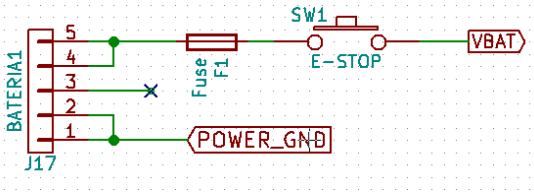


Fonte: Autor.

Na Figura 19 também pode-se observar a lógica com LEDs implementada: Quando a tensão está acima do limiar acende-se o verde, e caso a tensão caia abaixo desse valor, além de desabilitar os motores, o LED vermelho é acesso, para informar o usuário que a bateria deve ser retirada para carregar. Outras proteções que o circuito tem são: Um botão de emergência, caso o usuário detecte algum mal funcionamento e queira encerrar de imediato o experimento (Figura 21) e um fusível, que corta a alimentação caso uma corrente muito alta seja detectada (Figura 20).

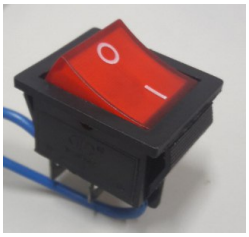
Como dito anteriormente, a tensão da bateria de 22,2V é abaixada para os demais componentes do circuito. Para alimentar o Arduino Mega, utiliza-se o regulador LM7810CT (Tabela 19), que converte a tensão de entrada, de até 35V, para 10V. Já para o XBee e demais componentes, utilizou-se dois reguladores LM2596 (Tabela 20), que possuem saída ajustável. Um foi configurado para fornecer 5V e o outro 3,3V. Observa-se que com o uso de reguladores, flutuações da tensão da bateria entre 22,2V e 18V não afetam a alimentação dos componentes do sistema.

Figura 20 – Lógica implementada com fusível e botão de emergência.



Fonte: Autor.

Figura 21 – Modelo de botão de emergência utilizado na placa.



Fonte: Autor.

3.2 Projeto de *Software*

3.2.1 *Software* embarcado

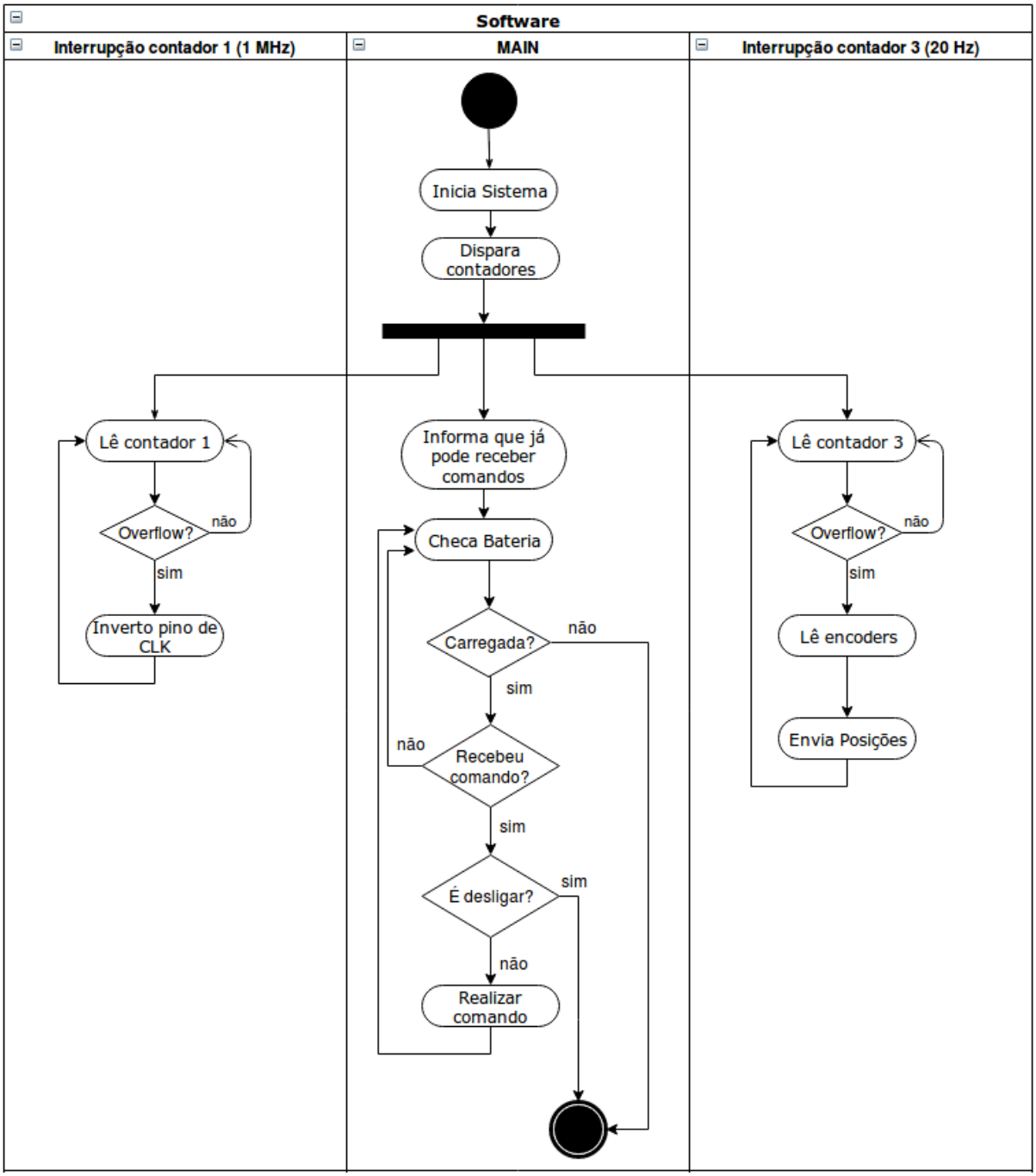
O *software* do módulo de comando foi desenvolvido para atender uma série de funções que o microcontrolador deve realizar na plataforma, que são:

- Inicialização do sistema.
- Leitura dos encoders.
- Geração do *clock* do decodificador de quadratura.
- Desligar o sistema caso a tensão de bateria caia abaixo do limiar.
- Receber do computador externo a velocidade desejada para os motores e configurá-los dessa forma.
- Envio da velocidade atual dos motores para o computador externo.
- Desativar o sistema.

Algumas das tarefas listadas acima devem ser feitas em paralelo. O programa deve ficar na espera dos comandos que virão da máquina externa, mas, ao mesmo tempo, deve estar fornecendo o sinal de *clock* para os decodificadores, e também deve realizar a leitura dos *encoders*, fornecendo esses dados com frequência constante. O sistema foi implementado de forma a paralelizar essas atividades, como mostrado na Figura 22, e para isso utiliza interrupções do microcontrolador.

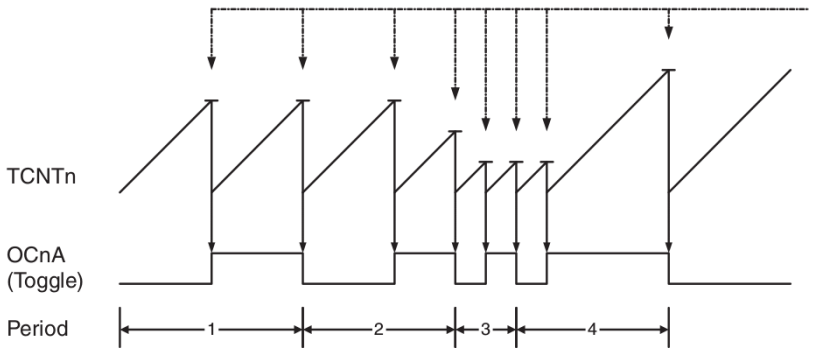
Interrupções são eventos que param o fluxo do programa principal e levam o microcontrolador à executar outra tarefa. Geralmente, esta outra tarefa está definida em um ISR (*Interrupt Service Routine*), que é um trecho de código, salvo em uma posição de memória diferente, que o programa executa e então retorna para o fluxo principal

Figura 22 – Fluxograma UML do programa executado pelo microcontrolador.



Fonte: Autoria própria.

Figura 23 – Modo CTC com inversão do pino de OCnA associada a *overflow* do contador TCNTn. A variação de período é causada pela mudança do valor escrito em OCR.



Fonte: (Atmel, 2014)

(GRIDLING, 2007). No caso das interrupções utilizadas no programa, o evento que as dispara é o *overflow* dos Contadores 1 e 3 do Arduino.

3.2.1.1 Geração do sinal de sincronia para os decodificadores de quadratura

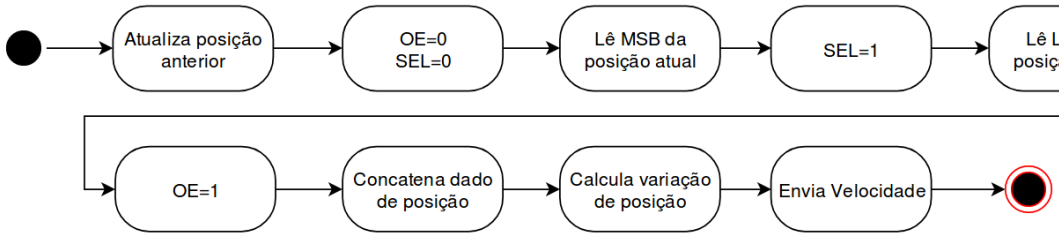
Para geração do *clock* de 1MHz não foi preciso programar um ISR. O ATmega2560 possui as funcionalidades *Output Compare* (OC) e *Clear Timer on Compare (Auto Reload)* (CTC). A primeira significa que se pode associar o *overflow* de seus contadores a eventos em pinos específicos, como colocar nível de tensão alto (5V), baixo (0V) ou inverter sua tensão (GRIDLING, 2007). A segunda permite que, quando o contador atinja o valor programado como máximo, ele seja zerado automaticamente, sem necessidade codificar uma ISR para este fim.

A frequência é controlada com base no valor carregado no registrador OCR (*Output Compare Register*), a relação entre o valor desse registrador e a frequência observada no pino de OC é descrita pela Equação 3.4, onde f_{OC} designa a frequência no pino de OC, f_{clk} representa o sinal de relógio e N representa o valor do *prescale factor*, que pode ser configurado caso se queira diminuir a frequência de *clock* utilizada. A Figura 23 ilustra o processo, sendo que TCNT designa o registrador onde é armazenado o valor do contador, que é comparado com o conteúdo de OCR.

$$f_{OC} = \frac{f_{clk}}{2 \times N \times (1 + OCR)} \tag{3.4}$$

Para gerar a onda de 1 MHz foi usado o contador 1, em modo CTC, com pino de OC habilitado e inversão de tensão no pino quando TCNT equivale a OCR. Sabendo que o sinal relógio do ATmega2560 tem frequência de 16 MHz e usando o fator de *prescale* como 1, o valor que é obtido para OCR, a partir da Equação 3.4, é 7. Assim é gerado o sinal

Figura 24 – Fluxograma da ISR implementada para leitura de posição e envio das velocidades.



Fonte: Autoria própria.

de sincronia necessário para os decodificadores de quadratura. É interessante observar que, após a programação incial dos registradores associados ao processo, ele é resolvido inteiramente em *hardware*. Não há a necessidade de implementação de nenhuma subrotina pelo programador, assim não há concorrência com o programa principal.

3.2.1.2 Leitura dos *encoders* e envio das velocidades

Para a leitura dos *encoders* foi escolhido um método que garantisse um tempo de amostragem constante, e que não interferisse com as demais tarefas do módulo de comando. A abordagem escolhida foi associar esta tarefa à interrupção do Contador 3. A frequência com que a leitura é feita é definida em uma variável do programa denominada *sample_rate*, onde o usuário dever inserir o valor em Hz desejado para o experimento. O valor programado no registrador OCR do contador 3 é obtido pela Equação 3.5.

Observe que foi utilizado o valor máximo de *prescale* (N=1024). Isso foi feito por duas razões. A primeira é que existe um limite para o número salvo no registrador OCR. Para o contador 3 esse limite é de 16 bits. O aumento do fator, como observado na Equação 3.5 diminui o valor salvo no registrador. A outra razão é que esse fator já nos fornece uma resolução de tempo satisfatória. Cada incremento no contador é feito em 0,64 μs, enquanto que os períodos de amostragem usados são da ordem de milissegundos.

$$OCR3 = \frac{f_{clk}}{1024 \times sample\_rate} - 1 \tag{3.5}$$

Assim como feito com o contador 1, o modo CTC foi utilizado, para automatizar a limpeza do contador quando o valor máximo for atingido. Nesta etapa foi necessário a implementação de uma ISR, já que a tarefa a ser realizada é mais complexa que a inversão de tensão em um pino. O fluxo da ISR implementada pode ser observado na Figura 24. A lógica de que orienta os valores dos pinos OE e SEL é exibida na Tabela 3.

Quando se entra na subrotina, a primeira ação é atribuir o valor da variável *posição_atual* para a variável *posição_anterior*, pois o valor que se tem foi obtido na

última iteração. Isso é feito para cada junta. Após isso, altera-se os valores dos pinos de controle do HCTL-2017 de forma a conseguir ler o *byte* mais significativo (MSB) e o menos significativo (LSB) da posição de cada motor. Como já foi discutido, os dados de posição possui 16 bits, e são necessárias duas leituras pois o decodificador fornece apenas 8 pinos para leitura de dados.

Como mostrado na subseção 3.1.3, os pinos de dados dos decodificadores são roteados de forma que cada um está ligado a uma porta específica do Arduino. Isso foi feito para deixar a leitura dos pinos mais rápida e o código mais enxuto. Dessa forma, para leitura de um *byte* de posição, em vez da leitura sequencial de cada pino, utilizando a função *digitalRead* fornecida pela biblioteca arduino, utilizou-se um método mais baixo nível.

Cada porta do arduino é controlada por três registradores, que também são variáveis definidas na linguagem Arduino(Arduino LLC, 2018). Eles são DDR, PORT e PIN. O primeiro define o sentido dos pinos da porta. O segundo possibilita a escrita e leitura deles e o terceiro é um registro apenas de leitura. No código, faz-se uma leitura direta do registrador PIN conectado a cada decodificador para obtenção de um byte de posição.

O Código 3.1 mostra como seria a leitura de um byte utilizando a função da biblioteca Arduino, enquanto que o Código 3.2 mostra a leitura utilizando manipulação de portas. Para o código implementado, que lê 8 *bytes* a cada instante de amostragem, em vez de codificar 64 leituras, reduziu-se a operação a 4 leituras do registrador PIN. Ainda que cada operação se dê em microssegundos, Haveria uma diferença de 64 vezes esse intervalo entre a primeira e a última leitura.O método mais baixo nível garante uma leitura praticamente instantânea dos decodificadores.

Código 3.1 – Leitura utilizando função *digitalRead()* da biblioteca Arduino.

```
position  = digitalRead(DA0);
position |= digitalRead(DA1)<<1;
position |= digitalRead(DA2)<<2;
position |= digitalRead(DA3)<<3;
position |= digitalRead(DA4)<<4;
position |= digitalRead(DA5)<<5;
position |= digitalRead(DA6)<<6;
position |= digitalRead(DA7)<<7;
```

Código 3.2 – Leitura utilizando Registrador PIN.

```
position = PINA;
```

3.2.2 Protocolo de comandos

Os comandos implementados que a plataforma consegue realizar são a imposição de velocidade nas juntas rotativas, freio do sistema e desativação dos motores. Esses comandos são feitos através de escritas nos *drivers* que interfaceiam microcontrolador e motores.

Os comandos de freio e desativação utilizam pinos de entrada e saída convencionais, enquanto que a escrita de velocidade utiliza pinos de PWM do Arduino. Para escrever neles se utiliza a função *analogWrite* da biblioteca arduino, onde se insere um valor de até 1 byte. A saída no pino de PWM é uma onda quadrada a qual se consegue controlar o valor médio. O controle é feito com base no valor escrito na função. O argumento máximo, de 255, implica uma onda de saída de 5V, enquanto que 127 gera na saída uma onda quadrada com 2,5V de valor médio.

A Tabela 4 exibe como o protocolo de comandos enviados à placa deve funcionar. No caso do comando de envio de velocidades, o loop entra em modo de espera para receber os *bytes* referentes às velocidades. Os *headers* para os comandos foram escolhidos arbitrariamente, mas com a atenção de diferirem em alguns bits, de forma a dificultar erros.

Tabela 4 – Protocolo de comandos enviados ao sistema.

Byte	**comando**
0x1F	Envio de velocidade
0x4C	Freio do sistema
0x7A	Desligar sistema

Tabela 5 – Protocolo de comandos enviados do sistema.

Byte	**Comando**
0xA9	Comando realizado
0xD6	Envio de posições

4 RESULTADOS E DISCUSSÕES

4.1 Implementação da placa

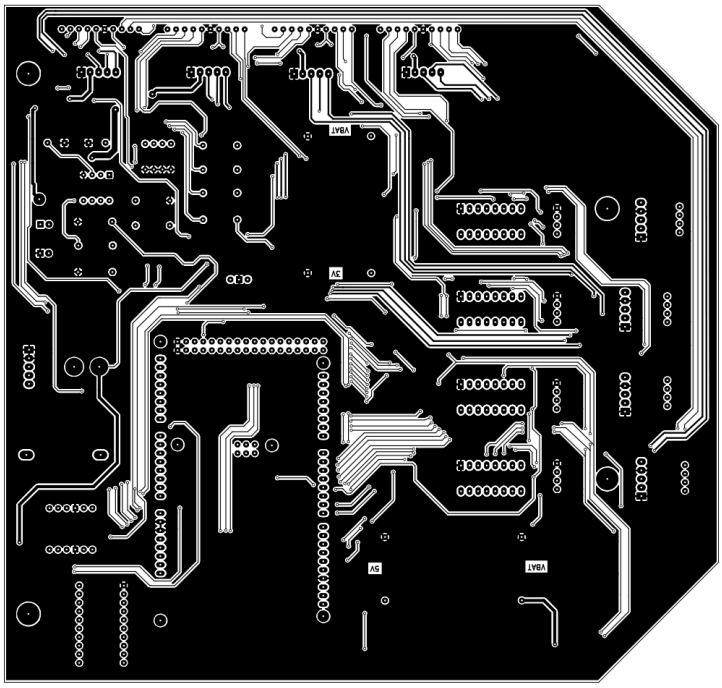
Nesta seção é mostrado o *layout* projetado para o sistema eletrônico, bem como a placa fabricada e a versão final, já com os componentes soldados, que foi utilizada para os testes.

Para o projeto de *layout* foi utilizado a ferramenta de *Electronic Design Automation* (EDA) KiCad, que foi escolhida por ser gratuita, código aberto e também por ser uma ferramenta com a qual já se tinha experiência. As Figuras 25 e 26 exibem as partes superior e inferior, respectivamente, da placa projetada. A área negra na parte superior da placa corresponde ao *plano terra*. Trata-se de uma estratégia utilizada no *design* de placas de circuito impresso, onde se utiliza uma superfície no mesmo potencial que os nós de Terra para diminuir ruído e facilitar o roteamento.

Após o projeto de *layout*, encaminhou-se os arquivos no formato *Gerber* gerados pelo programa para uma empresa especializada, para que a placa fosse fabricada. As figuras 27 e 28 mostram a placa de circuito impresso resultante do *layout*.

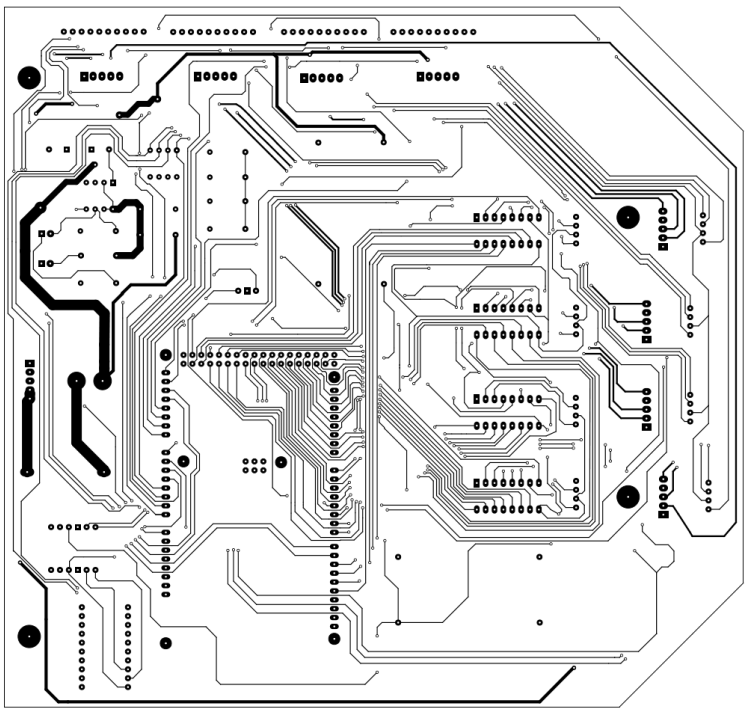
Com a placa em mãos, começou-se a soldadem dos componentes, para que os testes com o *hardware* pudessem ser feitos. As Figuras 29 e 30 mostram a placa já com os componentes soldados. É interessante observar que ela foi projetada de forma que alguns componentes pudessem continuar a ser utilizados em outros projetos, quando não se estiver utilizando a placa. O Arduino não fica soldado na placa, ela foi projetada para funcionar como um *shield*, onde o arduino pode ser encaixado, e retirado depois, quando não for ser usado. Os decodificadores de quadratura também são encaixados em *sockets*, que ficam soldados na placa. O botão de emergência não é soldado diretamente na placa, mas através de um cabo. Dessa forma o usuário pode deixá-lo na placa ou manuseá-lo com as mãos, caso o experimento permita.

Figura 25 – Layout da parte superior da placa.



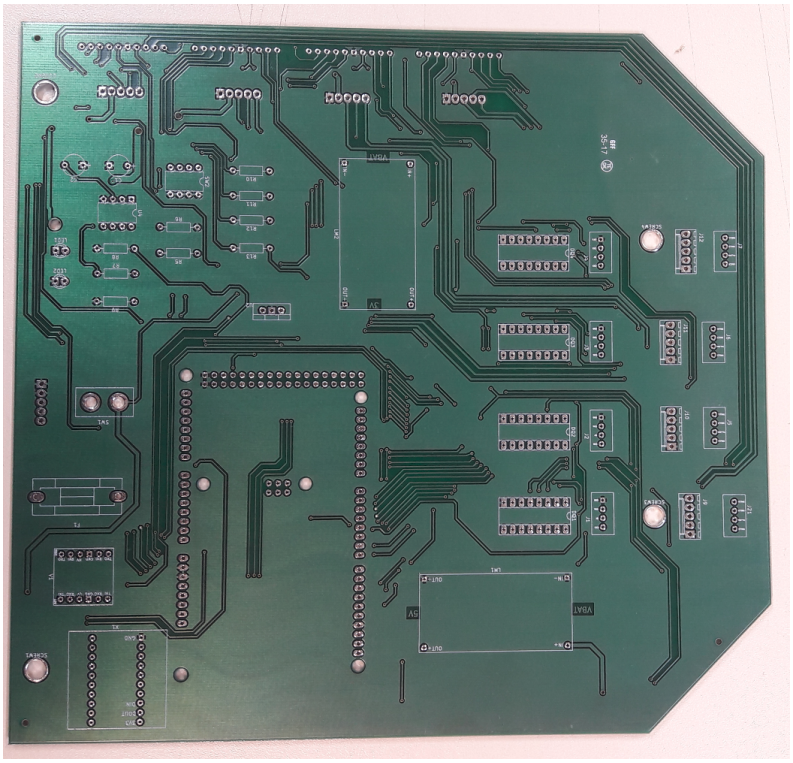
Fonte: Autor.

Figura 26 – Layout da parte inferior da placa.



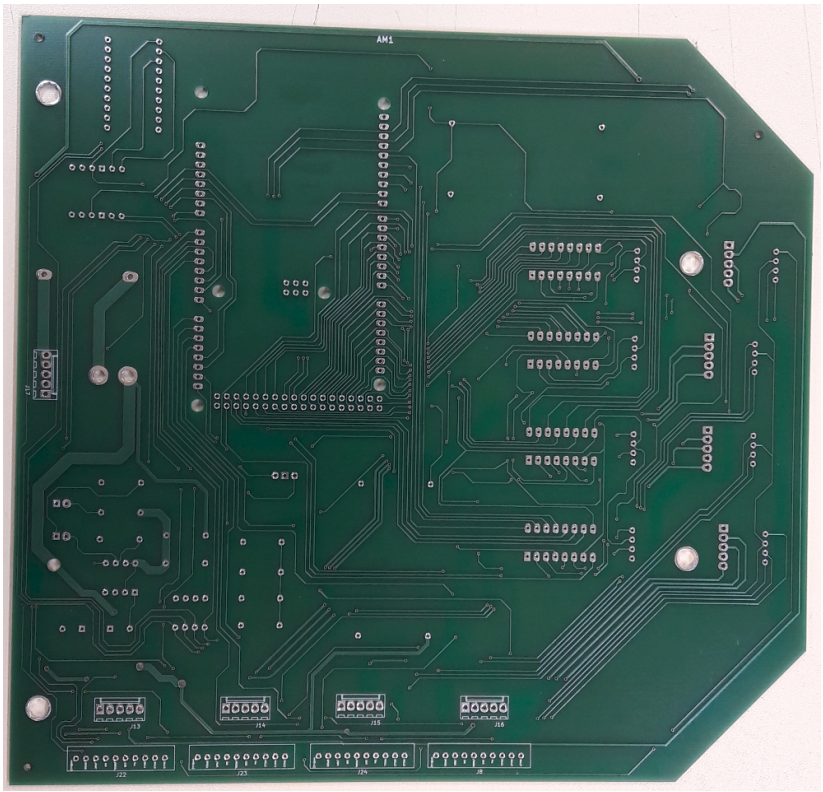
Fonte: Autor.

Figura 27 – Parte superior da placa fabricada.



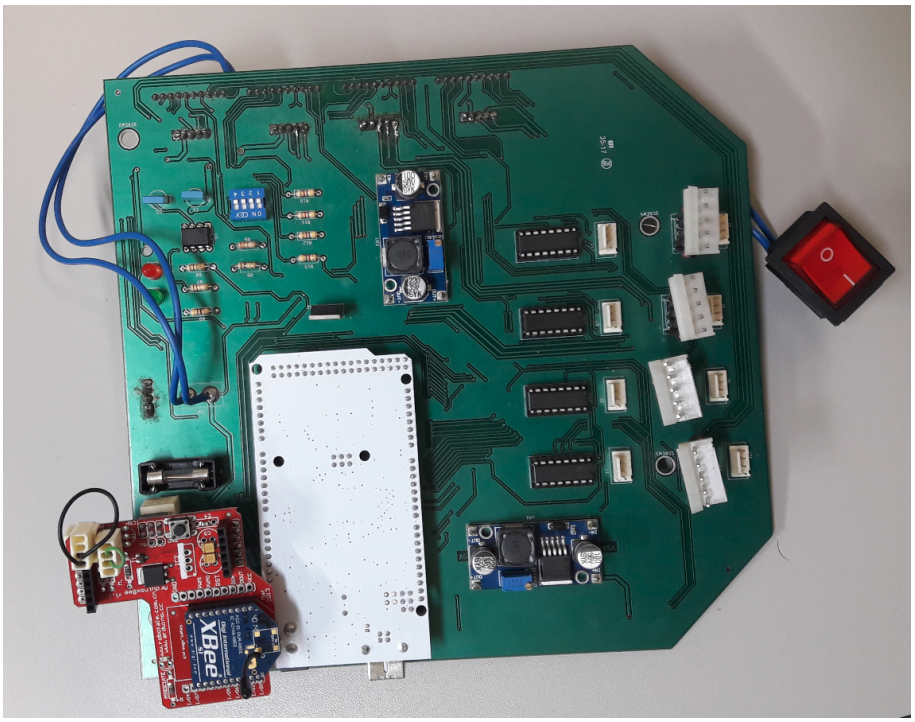
Fonte: Autor.

Figura 28 – Parte inferior da placa fabricada.



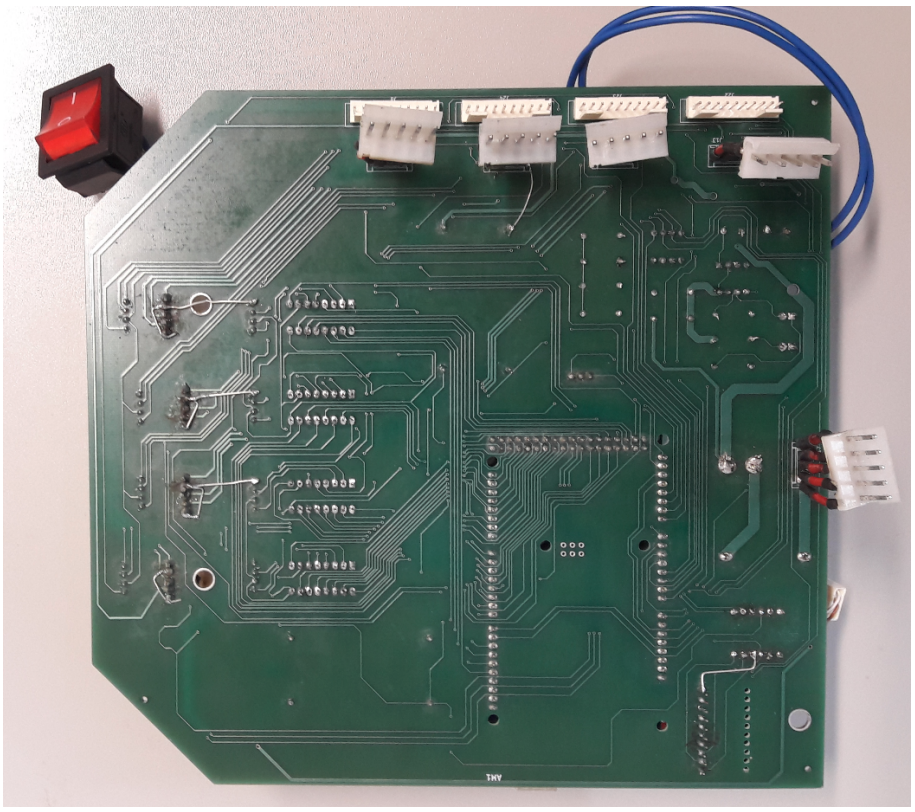
Fonte: Autor.

Figura 29 – Parte superior da placa já com os componentes soldados.



Fonte: Autor.

Figura 30 – Parte inferior da placa já com os componentes soldados.

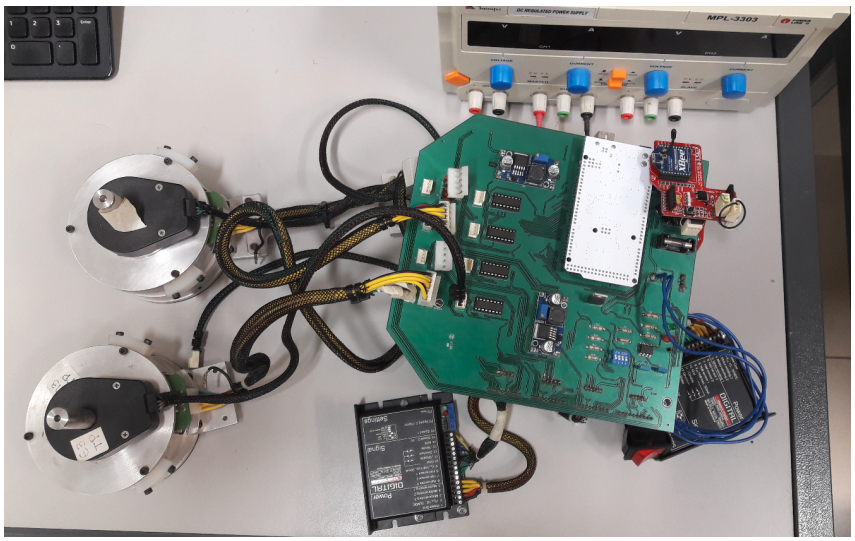


Fonte: Autor.

4.2 Testes dos módulos

Os testes foram realizados em bancada, utilizando dois motores que já estavam com *encoders* acoplados e cabos montados. A Figura 31 exibe a montagem do sistema para testes. O modelo da fonte utilizada para alimentação da placa é Ininipa MPL-3303. O modelo do multímetro utilizado para medir tensões é Ininipa ET-2070.

Figura 31 – Montagem do sistema com 2 motores, 2 *drivers* e fonte de bancada.



Fonte: Autor.

4.2.1 Motores e drivers

Os motores e *drivers* do robô foram testados de forma a verificar o bom funcionamento das interfaces implementadas, bem como do *software* que comandaria as ações. Neste teste o procedimento foi:

- Inicializar o sistema.
- Ligar os motores com rotação igual a 20% da velocidade máxima.
- Aumentar para 60% da velocidade máxima.
- Reduzir para 40% da velocidade máxima.
- Frear o motor.
- Desativar o motor.

A primeira etapa do teste visava confirmar o bom funcionamento das interfaces e conexões elétricas do sistema, bem como correta inicialização de pinos e atuação do

software controlando a velocidade do sistema. Além disso, pode-se observar a dinâmica do motor acelerando e reduzindo de velocidade quase instantâneamente.

No momento do freio o sistema apresentou parada brusca, e pode-se verificar que o eixo manteve-se rígido quando se tentou forçar movimento. Quando ele foi desativado o eixo perdeu a rigidez, podendo ser facilmente girado. Este procedimento foi repetido para cada um dos 4 terminais, revezando-se os 2 motores e 2 *drivers* disponíveis. O experimento foi satisfatório pois se pode verificar o comportamento esperado.

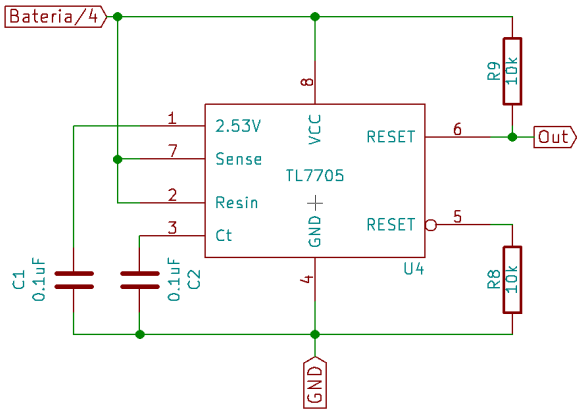
4.2.2 Módulo de energia

Neste teste o objetivo era verificar o correto funcionamento do módulo de energia com variações na corrente demandada pelo sistema e com a queda da tensão de alimentação. Primeiramente se manteve fixa a tensão de alimentação do sistema em 21,86V, e então aumentou-se a velocidade dos dois motores em giro, medindo a corrente demandada pelo sistema e as tensões rebaixadas pelo módulo de energia. Os dados colhidos encontram-se na Tabela 6. Pode-se observar que as tensões não foram afetadas pelo aumento de corrente. Porém, como os motores giravam em vazio, os valores de corrente exigidos foram pequenos.

Tabela 6 – Medição das tensões rebaixadas com o aumento da corrente exigida.

Velocidade	**Corrente**	**V_{in} do Arduino**	**Potencial de 5V**	**Potencial de 3,3V**
$0,2V_{MAX}$	0,17A	9,89V	5,06V	3,37V
$0,4V_{MAX}$	0,20A	9,89V	5,06V	3,37V
$0,6V_{MAX}$	0,24A	9,89V	5,07V	3,37V
$0,8V_{MAX}$	0,28A	9,89V	5,06V	3,36V

Figura 32 – Sugestão de novo circuito para o supervisor que não envolve a lógica de LEDs.



Fonte: Autor.

A segunda parte do teste consistiu em reduzir a tensão abaixo dos 18V para checar se o sistema de resposta desligaria o motor. O sistema falhou nesse teste. Utilizando o multímetro, verificou-se que a tensão supervisionada estava abaixo do previsto, com valor

de 1,15V. Notou-se que lógica implementada com o LED verde reduz o valor do divisor resistivo.

Como resultado, observou-se que este circuito de LEDs deve ser substituído de forma a não interferir com o divisor resistivo. Uma sugestão de circuito que pode ser implementada e não utiliza a lógica de LEDs é exibida na Figura 32. Essa falha não causou prejuízos pois se utilizou uma fonte de bancada nos testes, ao invés da bateria.

4.2.3 Módulo de sensoriamento

Neste teste, foi verificado a leitura de posição obtida pelo sistema. Para cada um dos 4 decodificadores, foi fixado um ponto no eixo do motor para marcar uma volta, e 8 voltas foram dadas, cada volta com valor esperado de 8 mil pulsos. Os valores colhidos em cada volta seguem abaixo, bem como média do experimento e desvio padrão.

Tabela 7 – Valores de teste obtidos para decodificador 1.

Volta	**Nº de pulsos**
1ª	8005
2ª	16001
3ª	23999
4ª	31998
5ª	40001
6ª	48000
7ª	55999
8ª	64001
Média	8000,125
Desvio padrão	19

Fonte: Autor.

Tabela 8 – Valores de teste obtidos para decodificador 2.

Volta	**Nº de Pulsos**
1ª	8003
2ª	15999
3ª	24000
4ª	31899
5ª	39901
6ª	47997
7ª	55998
8ª	63898
Média	7987
Desvio padrão	308

Fonte: Autor.

Tabela 9 – Valores de teste obtidos para decodificador 3.

Volta	**Nº de Pulsos**
1ª	7996
2ª	15993
3ª	23995
4ª	31994
5ª	39996
6ª	47997
7ª	55997
8ª	63995
Média	7999,4
Desvio padrão	15

Fonte: Autor.

Tabela 10 – Valores de teste obtidos para decodificador 3.

Volta	**Nº de pulsos**
1ª	7999
2ª	16003
3ª	24001
4ª	32002
5ª	40000
6ª	47997
7ª	56000
8ª	63998
Média	7999,8
Desvio padrão	18

Fonte: Autor.

Como pode ser visualizado nas tabelas, em média o valor de pulsos em uma volta é muito próximo de 8 mil, a média que mais se afastou só divergiu em 0,16%. O desvio padrão chegou a 3,85% ta Tabela 8, porém observa-se o valor mais alto apenas nela, e devido a 3 das 8 medidas.

O objetivo do teste não era medir a precisão do sistema, já que não se utilizou instrumentos de precisão, mas observar se os dados lidos fariam sentido, se não apresentariam grandes discrepâncias ou tendências. Com base nisso, os resultados foram satisfatórios.

4.2.4 Módulo de comunicação

O teste de comunicação consistiu em avaliar diferentes taxas de *bits* com relação à possíveis perdas de dados, erros de ordenamento, e tempo em que os dados seriam processados pelo Matlab. Para cada taxa, o teste se divide em 5 etapas. Cada etapa consiste no envio de 100 bytes para o Matlab, o *script* que processa os dados salva o tempo em que cada *byte* é percebido. Este tempo é obtido utilizando as instruções *tic* e *toc* do Matlab. O programa está disponível no Apêndice C.

Os dados eram enviados de forma ordenada, com valores variando entre 1 e 100, de forma que se algum fosse perdido, ou a ordem trocada, seria percebido pelo teste. Os resultados obtidos seguem nas tabelas 11, 12, 13 e 14. Para o teste, "processar"uma amostra implica recebê-la e salvá-la em uma variável local. As instruções *tic* e *toc* são usadas de forma a marcar o intervalo de tempo entre o processamento de duas amostras consecutivas.

Tabela 11 – Teste de comunicação com 9600 bps.

Etapa	**Bytes perdidos**	**Bytes desordenados**	**Tempo de processamento**
1ª	0	0	15,56 ms
2ª	0	0	16,82 ms
3ª	0	0	18,31 ms
4ª	0	0	16,28 ms
5ª	0	0	15,16 ms

Tabela 12 – Teste de comunicação com 19200 bps.

Etapa	**Bytes perdidos**	**Bytes desordenados**	**Tempo de processamento**
1ª	0	1	15,48 ms
2ª	0	1	15,96 ms
3ª	0	1	16,03 ms
4ª	0	1	16,08 ms
5ª	0	1	16,01 ms

Tabela 13 – Teste de comunicação com 32400 bps.

Etapa	Bytes perdidos	Bytes desordenados	Tempo de processamento
1ª	0	1	19,01 ms
2ª	0	1	19,11 ms
3ª	0	1	18,23 ms
4ª	0	1	16,26 ms
5ª	0	1	15,47 ms

Tabela 14 – Teste de comunicação com 57600 bps.

Etapa	Bytes perdidos	Bytes desordenados	Tempo de processamento
1ª	0	1	17,40 ms
2ª	0	1	16,32 ms
3ª	0	1	15,27 ms
4ª	63	0	344,33 ms
5ª	0	1	15,57 ms

A partir dos dados, pode-se concluir primeiro que, para a abordagem utilizada, aumentar a taxa de transmissão do XBee não surtiu efeito. O gargalo no processo não é a taxa com que o módulo de rádio envia dados, mas pode estar na forma como o Matlab processa os dados recebidos via porta serial no computador externo.

Com relação aos valores de tempo obtidos, considerando que se demora cerca de 15ms para o processamento de um *byte*, e que o *loop* de realimentação envolve a transmissão de 2 *bytes* por junta, seriam necessários pelo menos 120ms para transmissão de todos os dados de um laço de realimentação, o que restringiria a frequência de amostragem para menos de 9Hz.

Outra conclusão que se pode tirar é que a comunicação é bastante confiável, praticamente não apresentando perdas. Quanto ao byte desordenado que as taxas acima de 9600 bps apresentaram (com exceção da 4ª Etapa da máxima velocidade), ele sempre veio na primeira posição lida, mostrando-se um problema previsível.

5 CONCLUSÃO

O trabalho se propôs ao projeto e implementação de nova eletrônica para o robô UARM-E, de forma que a plataforma experimental pudesse voltar a ser utilizada em experimentos. Além de retomar o normal funcinamento, o projeto propôs uma arquitetura mais simples, que pudesse ser implementada mais rapidamente e com mais facilidade. Nesse quesito, as mudanças feitas permitiram que se descartasse alguns componentes, como o CPLD, o circuito de chaveamento de baterias e a segunda bateria. O uso de menos juntas também tornou o roteamento da placa menos complexo, contribuindo para que o projeto pudesse avançar mais rapidamente.

O uso da plataforma Arduino, uma das mudanças com relação ao projeto antigo, também rendeu bons frutos. Apesar de ser baseado em um microcontrolador mais limitado que o Rabbit 4000, utilizado no projeto original, em nenhum dos testes realizados ele se mostrou aquém das tarefas solicitadas. Além disso, o código desenvolvido para a plataforma é bem mais compreensível, facilitando que futuros alunos possam usar e modificar conforme as suas necessidades. Quando a interface com módulo XBee se mostrava mais complexa do que o esperado, a disponibilidade de um *shield* específico para a placa permitiu que o projeto continuasse sem maiores dificuldades na área de comunicação.

O módulo XBee, uma vez resolvido o problema da interface, se mostrou de fácil configuração graças às ferramentas disponibilizadas pela Digi, como o XCTU. Além disso, o modo de operação transparente, utilizado no projeto, faz com que a presença do módulo se torne invisível para o *software*, de forma que pode-se utilizar o mesmo programa com comunicação serial ou via XBee, o que contribuiu para um desenvolvimento mais rápido do *software* embarcado. A comunicação também se mostrou confiável. Os testes não mostraram haver necessidade de grande *overhead* no *software* para tratamento de erros na comunicação. Porém a interface com Matlab merece maior investigação, uma vez que os testes mostraram que o aumento na taxa de bits não implicava em maior velocidade.

De forma geral, o trabalho trouxe grandes desafios e requereu grande envolvimento. Por incluir uma implementação real do sistema idealizado, foi necessário lidar com vários problemas não previstos, como investigação de problemas elétricos na placa, adaptação de conectores, cabos e componentes. Ao mesmo tempo que provocaram dificuldades e atrasos, esses imprevistos agregaram ao aprendizado obtido no projeto.

5.1 Trabalhos futuros

Para futuras implementações de sistemas de mesmo tipo, é possível aproveitar melhor a capacidade do Arduino Mega. Ainda que a implementação descrita no projeto

tenha utilizado grande parte de sua pinagem, contadores e PWMs, ele tem mais a oferecer. O microcontrolador possui, por exemplo, cerca de 4 interfaces seriais, das quais apenas uma foi utilizada. Implementações que busquem um uso mais pesado da comunicação podem se aproveitar dessa arquitetura para diminuir a concorrência por recursos do controlador.

Outro aspecto que pode ser melhor explorado em outros projetos são as capacidades do XBee. Além dos 4 pinos usados no projeto, o componente apresenta mais 16, entre os quais se tem conversores AD e pinos para entrada e saída. Uma arquitetura que utilize mais a capacidade de processamento deste componente, aliviando o microcontrolador, pode render bons frutos.

Com relação ao módulo de energia, um novo circuito para o supervisor de tensão foi proposto. Seria interessante que futuros trabalhos validassem a sugestão feita ou propusessem seus próprios circuitos. Já em relação ao módulo de sensoriamento, ele se limita à sensores de posição (*encoders* rotativos), porém seria possível um projeto envolvendo sensores de força, para teste e validação de estratégias de controle mistas, envolvendo força e movimento. Caso a adição de funcionalidades torne o roteamento da placa muito complicado, pode-se utilizar placas de 4 camadas para viabilizar o projeto ou torná-lo mais rápido.

REFERÊNCIAS

BENSON, C. **Making Sense of Actuators**. 2010. Disponível em: <https://www.robotshop.com/blog/en/how-to-make-a-robot-lesson-3-actuators-2-3703>.

CRAIG, J. J. **Introduction to robotics: mechanics and control**. [S.l.]: Pearson/Prentice Hall Upper Saddle River, NJ, USA:, 2005. v. 3.

EITEL, E. Basics of rotary encoders: Overview and new technologies. **Machine Design Magazine**, 2014. Disponível em: <http://www.machinedesign.com/sensors/basics-rotary-encoders-overview-and-new-technologies-0>. Acesso em: 23 maio 2018.

GRIDLING, G. Introduction to microcontrollers. 2007.

LYNCH, K. M.; PARK, F. C. **Modern Robotics: Mechanics, Planning, and Control**. [S.l.]: Cambridge University Press, 2017.

MENON, C.; BUSOLO, S. Issues and solutions for testing free-flying robots. **Acta Astronautica**, Elsevier, v. 60, n. 12, p. 957–965, 2007.

PAZELLI, T. d. F. P. A. et al. **Montagem e controle H Infinito não linear de manipuladores espaciais com base flutuante**. 2011. Tese (Doutorado) — Universidade de São Paulo, 2011.

Arduino LLC. **What is Arduino?** 2018. Disponível em: <https://www.arduino.cc/en/guide/introduction>.

Atmel. **8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash**. 2549q. ed. [S.l.], 2014.

Creative Robotics Ltd. **What are Quadrature Encoders**. 2018. Disponível em: <http://www.creative-robotics.com/quadrature-intro>.

Digi International. **XBee/XBee-PRO S1 802.15.4 (Legacy) User Guide**. 90000982. ed. 11001 Bren Road East, Minnetonka, MN 55343, 2018.

_____. **XCTU: Next Generation Configuration Platform for XBee/RF Solutions**. 2018. Disponível em: <https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu>.

SPONG, M. W. **Robot modeling and control**. [S.l.]: Wiley New York, 2006. v. 3.

TEODORO, D.; AUGUSTO, C. Reformulação do hardware do braço robótico ma2000 com emrprego de arquitetura arm para controle. 07 2012.

Apêndices

APÊNDICE A – ESPECIFICAÇÕES DOS COMPONENTES

Tabela 15 – Especificações da placa Arduino Mega 2560.

Microcontrolador	Arduino Mega 2560
Frequência de Clock	16 MHz
Pinos de Entrada e Saída	54
Saídas PWM	16
Memória Flash	256 kB
Entradas Analógicas	16
Tensão de Alimentação	7-12V
Tensão de Operação	5V
Corrente por pino de E/S	20 mA
Peso	37g

Fonte: Arduino LLC (2018)

Tabela 16 – Especificações do módulo XBee.

Especificações	XBee
Área Urbana	30 m
Área sem obstáculos	90m
Taxa de dados RF	250.000 bps
Potência transmitida	1 mW
Tensão de Alimentação	2,8V-3,4V
Frequência de Operação	7-12V
Tensão de Operação	5V
Corrente por pino de E/S	20 mA
Peso	37g

Fonte: Digi International (2018a)

Tabela 17 – Especificações do decodificador de quadratura.

Decodificador de quadratura	**HCTL2017**
Tensão de alimentação	5 V
Nível alto de tensão de entrada	> 3.5 V
Nível baixo de tensão de entrada	< 1.5 V
Nível alto de tensão de saída	4.5 V
Nível baixo de tensão de saída	0.4 V
Resolução do contador	16 bits
Clock de operação	≤ 14 MHz
Interface de saída	Tristate

Fonte: Datasheet do componente.

Tabela 18 – Especificações do supervisor de tensão.

Supervisor	**TL7705**
Tensão de alimentação (VCC)	3,5V-10V
Tensão de threshold	4,55V

Fonte: Datasheet do componente.

Tabela 19 – Especificações do regulador LM7810CT.

Regulador	**LM7810C**
Tensão de Entrada	Até 35V
Tensão de saída	9,6-10,4

Fonte: Datasheet do componene.

Tabela 20 – Especificações do regulador LM2596

Regulador	**LM2596**
Tensão de Entrada	3,2-40V
Tensão de saída	1,5-35V

Fonte: Datasheet do componene.

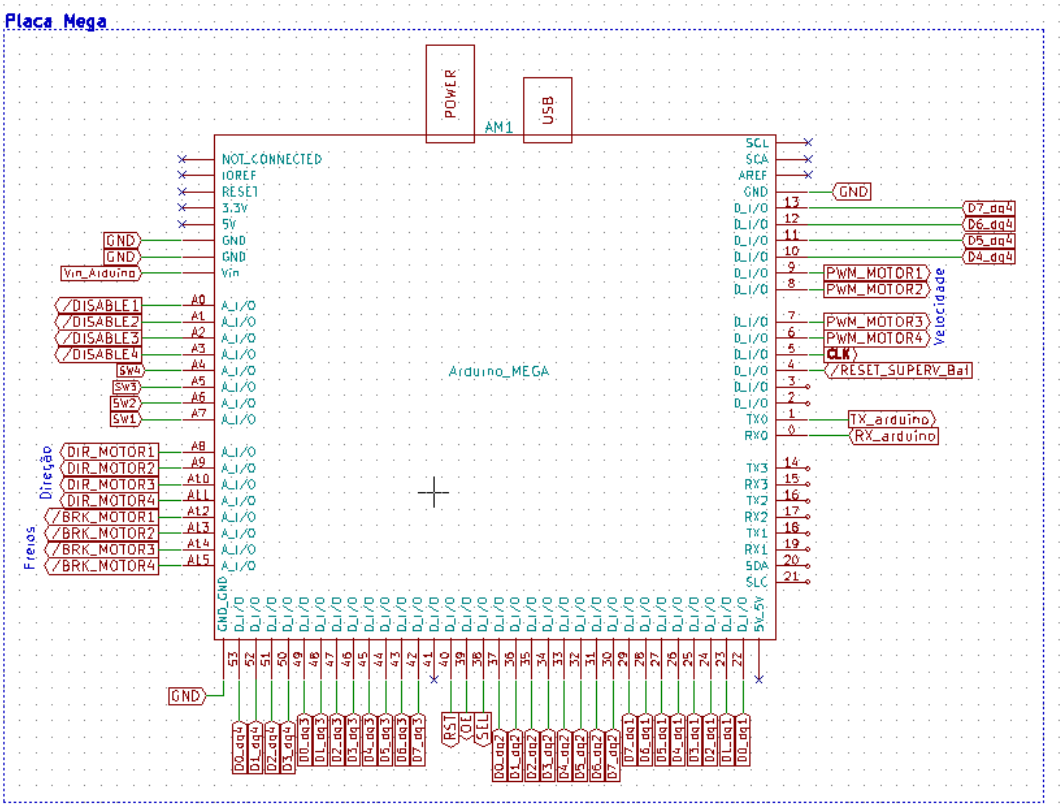
Tabela 21 – Especificação dos motores.

Motor	**EC 90 flat**
Tipo	Motor elétrico brushless
Fabricante	Maxon
Potência	90 W
Tensão nominal	48 VCC
Corrente nominal (sem carga)	130 mA
Máxima corrente contínua (1640rpm)	2.12 A
Velocidade nominal (sem carga)	2080 rpm
Máxima velocidade permitida	5000 rpm
Máximo torque contínuo @1640rpm	0.494 Nm
Torque contínuo travado	4530 mNm
Constante de torque, K_M	217 mNm/A
Constante de velocidade, K_n	44 rpm/V
Gradiente velocidade/torque, $\Delta n/\Delta M$	0.466 rpm/mNm
Resistência de enrolamento	2.30 Ohm (entre fases)
Inercia do rotor	3060 gcm 2
Massa do motor	648 g

Fonte: Maxon.

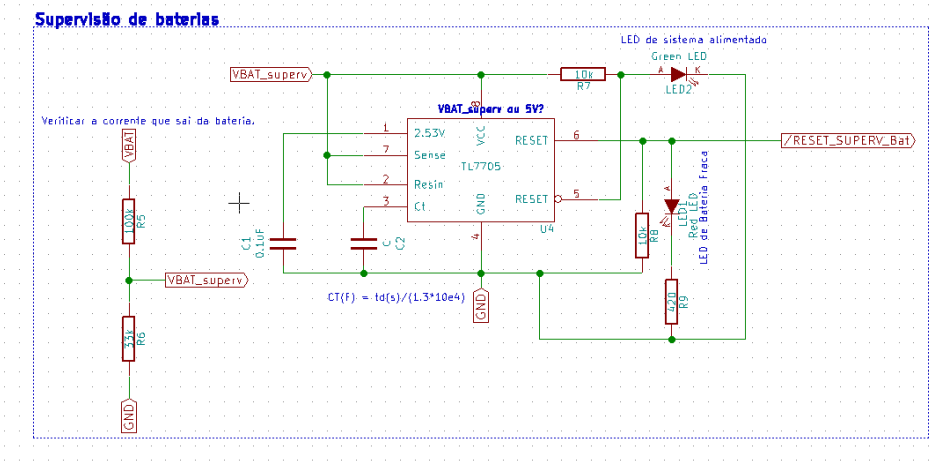
APÊNDICE B – ESQUEMÁTICOS

Figura 33 – Conexões Arduino Mega.



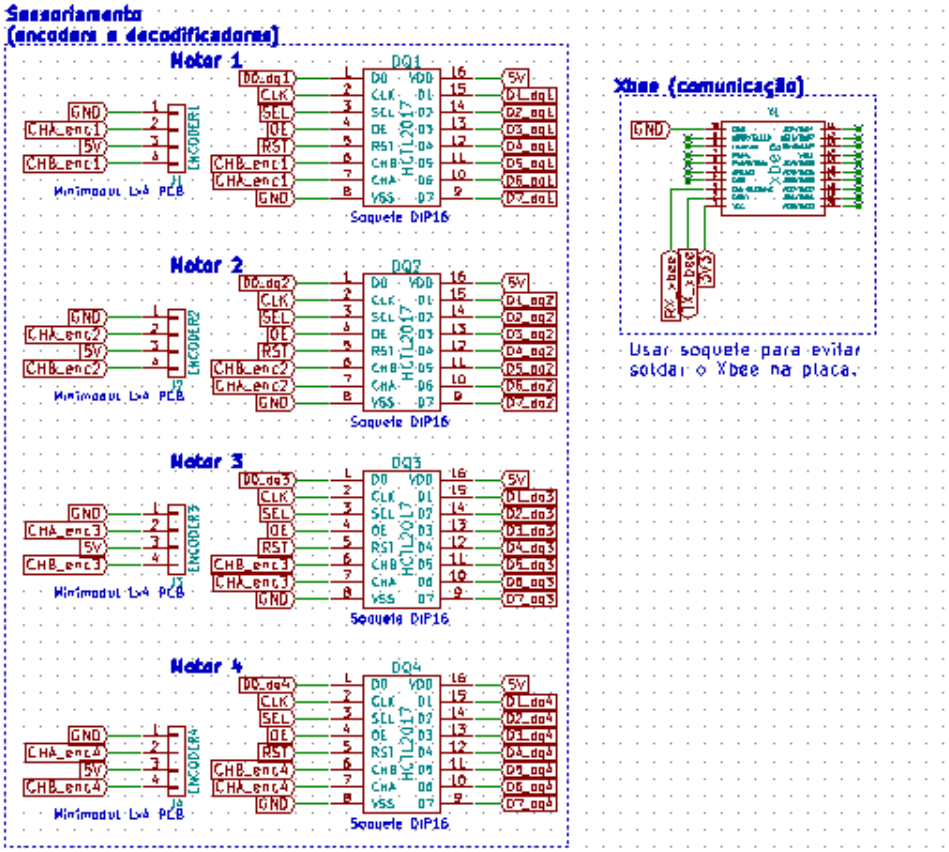
Fonte: Autor

Figura 34 – Conexões do supervisor de tensão.



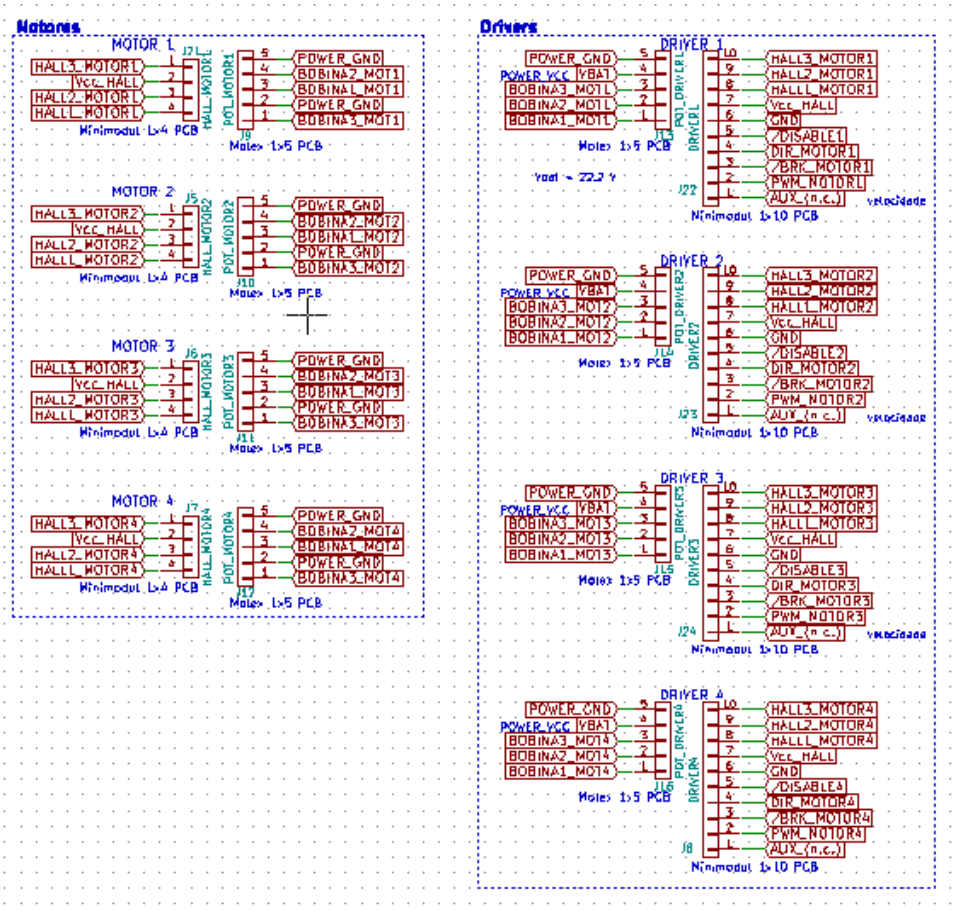
Fonte: Autor

Figura 35 – Conexões módulo de sensoriamento e XBee.



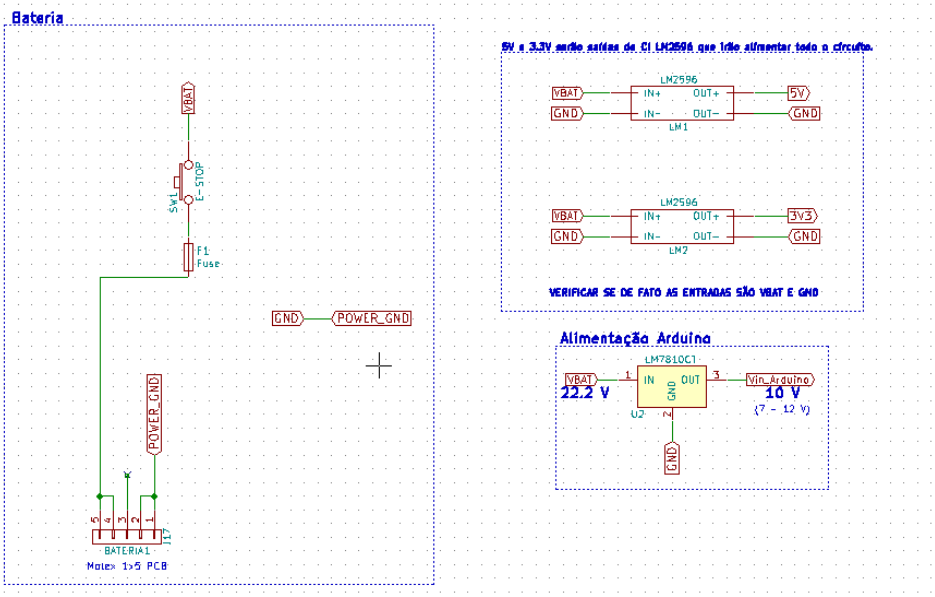
Fonte: Autor

Figura 36 – Conexões conectores de drivers e motores.



Fonte: Autor

Figura 37 – Conexões da bateria e reguladores.



Fonte: Autor

APÊNDICE C – CÓDIGOS IMPLEMENTADOS

Código C.1 – Teste de comunicação

```
% Close all serial ports
instrreset
% Set display format
format short
% Clear all local variables
clear
% Define serial communication settings
s = serial('/dev/CURRENT_PORT', 'BaudRate', CURRENT_BAUD);
fopen(s)
% Initialize variables
i = 1;
time_ = zeros(100,1);
Data = uint8(zeros(100,1));
% Start timer
tic
while (i<101)
    % Check serial port
    if (s.BytesAvailable>0)
        % Save this sample value
        Data(i) = fread(s,1);
        % Save processing time for this sample
        time_(i) = toc;
        i=i+1;
    end
end

% Display samples value and time
Data
time_
% Close serial port
fclose(s)
```

Código C.2 – Software embarcado

```
#include <Arduino.h>
/*
Explicacao do codigo
*/
//Program Defines
//If it becomes too big, create a define file for this
#define SAMPLE_FREQUENCY 20
#define COM_CHECK 75
#define SEL 38
#define OE  39
#define RST 40
#define PWM_MOT4 6
#define PWM_MOT3 7
#define PWM_MOT2 8
#define PWM_MOT1 9
#define DISABLE1 A0
#define DISABLE2 A1
#define DISABLE3 A2
#define DISABLE4 A3
#define SW4      A4
#define SW3      A5
#define SW2      A6
#define SW1      A7
#define DIR_MOT1 A8
#define DIR_MOT2 A9
#define DIR_MOT3 A10
#define DIR_MOT4 A11
//Attention, break pins are active in LOW
#define BRK_MOT1 A12
#define BRK_MOT2 A13
#define BRK_MOT3 A14
#define BRK_MOT4 A15
//      CLK  5  (Apenas para documentacao,
//essa definicao eh feita de maneira automatica.
#define pi   3.14159

int V1 = 0;
int V2 = 0;
```

```
int V3 = 0;
int V4 = 0;
int last_position1 = 0;
int last_position2 = 0;
int last_position3 = 0;
int last_position4 = 0;
int current_position1 = 0;
int current_position2 = 0;
int current_position3 = 0;
int current_position4 = 0;
float current_speed1 = 0;
float current_speed2 = 0;
float current_speed3 = 0;
float current_speed4 = 0;
int inByte = 70;
int command=0;
//int i;
byte speed1[] ={0,0};
signed int test = 34000;
float pos4_graus = 0;

unsigned char i;
int j;

unsigned char speeds [5] = {9 , 3, 12, 9, 7};//, 20, 13, 2, 5};

// the setup function runs once when you press reset
//or power the board
void setup() {
  analogWrite(PWM_MOT1, 0);
  analogWrite(PWM_MOT2, 0);
  analogWrite(PWM_MOT3, 0);
  analogWrite(PWM_MOT4, 0);
  // start serial port at 9600 bps
  Serial.begin(57600);
  delay(100);
//Coloca os bits do port D (2 a 7) como entradas, sem alterar
// o valor dos bits 0 e 1, que sao RX e TX
//  DDRD = DDRD & B00000011;
```

```
// Coloca o port A como entrada
   DDRA = 0b00000000;
 //Seta pinos de controle do decodificador como saida
   pinMode(RST, OUTPUT);
   pinMode(SEL, OUTPUT);
   pinMode(OE, OUTPUT);
   pinMode(41, OUTPUT);
   //Set drivers pins modes
   pinMode(BRK_MOT1, OUTPUT);
   pinMode(BRK_MOT2, OUTPUT);
   pinMode(BRK_MOT3, OUTPUT);
   pinMode(BRK_MOT4, OUTPUT);

   pinMode(DISABLE1, OUTPUT);
   pinMode(DISABLE2, OUTPUT);
   pinMode(DISABLE3, OUTPUT);
   pinMode(DISABLE4, OUTPUT);

   pinMode(DIR_MOT1, OUTPUT);
   pinMode(DIR_MOT2, OUTPUT);
   pinMode(DIR_MOT3, OUTPUT);
   pinMode(DIR_MOT4, OUTPUT);

 //Da um pulso no pino reset para zerar o decodificador
   digitalWrite(RST, LOW);
   delay(400);
   digitalWrite(RST, HIGH);

   //set direction in motors
   digitalWrite(DIR_MOT1,HIGH);
   digitalWrite(DIR_MOT2,HIGH);
   digitalWrite(DIR_MOT3,HIGH);
   digitalWrite(DIR_MOT4,HIGH);
   //Turn off break (this signal is active in LOW)
   digitalWrite(BRK_MOT1,HIGH);
   digitalWrite(BRK_MOT2,HIGH);
   digitalWrite(BRK_MOT3,HIGH);
   digitalWrite(BRK_MOT4,HIGH);
   //Chose which motors are disabled
```

```
    //(this signal is active in LOW)
    digitalWrite(DISABLE1,LOW);
    digitalWrite(DISABLE2,LOW);
    digitalWrite(DISABLE3,LOW);
    digitalWrite(DISABLE4,LOW);
    //Set the motors speed
    //analogWrite(PWM_MOT1, 50);
    //analogWrite(PWM_MOT2, 50);
    //analogWrite(PWM_MOT3, 50);
    //analogWrite(PWM_MOT4, 50);

//Configuracoes do Timer 1,
//usado para fazer a leitura constante do Encoder
// initialize timer1
  noInterrupts();           // disable all interrupts
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0;

  OCR1A = (int)(16000000/1024/SAMPLE_FREQUENCY);
  // compare match register 16MHz/256/2Hz
  TCCR1B |= (1 << WGM12);
  // CTC mode (zera o timer quando atinge o valor definido)
  TCCR1B |= (1 << CS12); // 1024 prescaler
  TCCR1B |= (1 << CS10); // 1024 prescaler
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
  interrupts();  // enable all interrupts

 //CLOCK DE 1MHZ
 //Utiliza-se o timer 3 para gerar um clock de 1MHz
 //no pino OC3A (No arduino mega, pino 5 ou PE3) do arduino.
 DDRE = _BV(DDE3);      //set OC3A
 TCCR3A = _BV(COM3A0); //toggle OC3A on compare match
 OCR3A = 7;        //top value for counter
 TCCR3B = _BV(WGM12) | _BV(CS30);
 //CTC mode, prescaler clock/1

  //
  delay(1000);
```

```
}

//***********FUNCTIONS*******************//

//Rotina de Interrupcao do timer 1
ISR(TIMER1_COMPA_vect)
// timer compare interrupt service routine
{
    // Update last position
    last_position1 = current_position1;
    last_position2 = current_position2;
    last_position3 = current_position3;
    last_position4 = current_position4;
    // Clear position variable
    V1 = 0;
    V2 = 0;
    V3 = 0;
    V4 = 0;
//******LEITURA DOS ENCODERS*******
    //LEITURA DOS PINOS HIGH
    //Controle do Decodificador
    digitalWrite(OE, HIGH);
    digitalWrite(SEL, LOW);
    digitalWrite(OE, LOW);

    //Uliza-se shift de 8 bits para
    //leitura do byte mais significativo
    //Using PORTA cause we're using decoder 1
    V1 = (PINA << 8);
    V2 = (PINC << 8);
    V3 = (PINL << 8);
    V4 = (PINB << 8);
    //LEITURA DOS PINOS LOW
    //Comando no pino SEL, agora os bits
    //menos significativos serao lidos
    digitalWrite(SEL, HIGH);
    //Em caso de problemas, incluir delay
    V1 = V1 | PINA;
```

```
    V2 = V2 | PINC;
    V3 = V3 | PINL;
    V4 = V4 | PINB;
    //Update current positions
    current_position1 = V1;
    current_position2 = V2;
    current_position3 = V3;
    current_position4 = V4;
    Serial.write(214);
    Serial.write(current_position1);
    Serial.write(current_position2);
    Serial.write(current_position3);
    Serial.write(current_position4);
}

void commandDone();{
    Serial.write(169)
}
void stopEngine(){
    digitalWrite(BRK_MOT1,LOW);
    digitalWrite(BRK_MOT2,LOW);
    digitalWrite(BRK_MOT3,LOW);
    digitalWrite(BRK_MOT4,LOW);
    commandDone();
};

void setSpeeds(){
    int value={0,0,0,0};
    for (int i=0;i<4;i++){
        value[i]=Serial.read();
    }
    analogWrite(PWM_MOT1, value[0]);
    analogWrite(PWM_MOT2, value[1]);
    analogWrite(PWM_MOT3, value[2]);
    analogWrite(PWM_MOT4, value[3]);
    commandDone();
}

void offSystem(){
```

```
    digitalWrite(DISABLE1,LOW);
    digitalWrite(DISABLE2,LOW);
    digitalWrite(DISABLE3,LOW);
    digitalWrite(DISABLE4,LOW);
    commandDone();
};
void battery_check(){
    if (digitalRead(6)==HIGH);
    offSystem();
}
unsigned char command;
void loop() {

     if(Serial.available()) {
        //get incoming byte
        command = Serial.read();

       switch (command) {
            Command 1 returns communication ok
            case 31:
                setSpeeds();
                break;
            case 76:
                stopEngine();
                break;
            case 122:
                offSystem();
                break;
        }
        battery_check();
}
```