UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROGRAMA DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

SAULO GUILLERMO D'EDUARD ARARIPE SOBREIRA MUNIZ

CONTROLE DE MÚLTIPLOS SERVOMOTORES: ANIMAÇÃO EM ROBÓTICA

FORTALEZA
2013

SAULO GUILLERMO D'EDUARD ARARIPE SOBREIRA MUNIZ

CONTROLE DE MÚLTIPLOS SERVOMOTORES: ANIMAÇÃO EM ROBÓTICA

Trabalho de Conclusão de Curso apresentada ao Departamento de Engenharia Elétrica da Universidade Federal do Ceará, como requisito parcial à obtenção do título de graduado em Engenharia Elétrica. Área de concentração: Automação e Robótica.

Orientador: Prof. Dr. Paulo Praça

FORTALEZA

2013

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M936c Muniz, Saulo Guillermo D'Eduard Araripe Sobreira.
Controle de múltiplus servomotores : animação em robótica / Saulo Guillermo D'Eduard Araripe Sobreira Muniz. – 2013.
63 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2013.
Orientação: Prof. Dr. Paulo Peixoto Praça.

1. Automação. 2. Servo motor. 3. Controle. 4. Sistemas embarcados. 5. CAD. I. Título.

CDD 621.3

SAULO GUILLERMO D'EDUARD ARARIPE SOBREIRA MUNIZ

CONTROLE DE MÚLTIPLOS SERVOMOTORES: ANIMAÇÃO EM ROBÓTICA

Trabalho de Conclusão de Curso apresentada ao Departamento de Engenharia Elétrica da Universidade Federal do Ceará, como requisito parcial à obtenção do título de graduado em Engenharia Elétrica. Área de concentração: Automação e Robótica.

Aprovada em: ___/___/_____.

BANCA EXAMINADORA

_____

Prof. Dr. Paulo Praça (Orientador)

Universidade Federal do Ceará (UFC)

_____

Profª. Dr. Luiz Henrique Silva Colado Barreto

Universidade Federal do Ceará (UFC)

_____

Me. Antônio Barbosa de Souza Júnior

Universidade Federal do Ceará (UFC)

Dedico este trabalho primeiramente à Deus e a minha família.

Aos demais que motivam e inspiram a minha evolução, em especial aos meus amigos e professores.

AGRADECIMENTO

Agradeço em primeiro minha família, Sandra e Eduardo, meus pais, cuja integridade foi meu exemplo durante constituição de minha moral. Aos meus irmãos: Igor, que sempre me apoiou, e Filipe, que me proporcionou continuo desenvolvimento intelectual e criativo.

Aos meus professores e pesquisadores do curso de Engenheira Elétrica, que foram sempre solícitos e me ajudaram ao longo de cinco longos anos, em especial, ao professor Dr. Paulo Peixoto Praça, que me orientou e desde o início do curso me motivou a estudar e trabalhar com primor. A professora Dra. Laurinda Lúcia Nogueira Reis, que de maneira ímpar me motivou a ir mais longe e a aprofundar meus conhecimentos. A professora Dra. Gabriela Baub Shiguemoto e ao Gleidson da Rocha que sempre me receberam prontamente e me ampararam no curso sempre que possível.

Aos meus colegas de graduação pelo companheirismo e momentos de descontração nos intervalos da Universidade Federal. Aos meus amigos, particularmente ao Michel Ney e ao Vicente Queiroz, que constituíram junto comigo importantes projetos de engenharia. A minha belíssima amiga, companheira, psicóloga e namorada Gabrielle Arruda que me apoia irracionalmente e me tolerou ao longo deste conturbado ano.

Aos demais que me influenciaram durante o período da faculdade que por ventura eu não tenha mencionado, desde já minhas sinceras desculpas.

“Se eu vi mais longe, foi por estar de pé sobre ombros de gigantes.”

Isaac Newton

RESUMO

Com objetivo de acionar a cabeça e os membros superiores de um protótipo de robô humanoide, desenvolve-se um algoritmo para automação de múltiplos servos motores, no qual, estes atuam como articulações da estrutura do robô. Como se trata de um protótipo, o algoritmo apresentado é genérico e foi adaptado para permitir futuras alterações do acionamento do robô. Além disso, uma planilha de cálculos foi gerada como interface entre o código e o usuário, para que este possa alterar ou criar novas animações para o robô. Para simular a lógica do algoritmo, projeta-se em CAD um modelo da estrutura utilizando doze servos motores, no qual, dois são utilizados na cabeça e cinco são utilizados em cada braço, sendo assim, o algoritmo separa cada membro em um módulo distinto, para que estes atuem de maneira independente. O projeto em CAD também permite gerenciar os ângulos limites de atuação dos servos. Programa-se o algoritmo numa plataforma de desenvolvimento, que atuará como controlador. Ao ser implementado junto a estrutura articulada, o controlador funcionará como um sistema embarcado capaz de acionar a estrutura. O sistema permitiu, ainda, a implantação de sensores capazes de gerar interrupções, animações ou, até mesmo, buscar a estabilidade mecânica através do controle digital dos servos motores. O processo, apesar de sequencial, foi capaz de articular os doze servos distintos sem que houvesse um atraso relevante nos movimentos. Em suma, o controlador torna-se uma boa opção para uso em sistemas embarcados, funcionando para o controle e automação da estrutura.

Palavras-chave: Automação. Servo Motor. Controle. Sistemas Embarcados. CAD.

ABSTRACT

In order to move the head and upper limbs of a prototype humanoid robot, developed an algorithm to automate multiple servo motors, in which these act as joints of the robot structure. As this is a prototype, the presented algorithm is generic and has been adapted to allow future changes in the robot. In addition, a spreadsheet calculation was generated as an interface between the code and the user, so that it can change or create new animations for the robot. To simulate the logic of the algorithm, a CAD model is projected. The structure uses twelve servomotors, in which two are used in the head and five are used for each arm, thus, the algorithm divides each member in a separate module, so that they operate independently. The CAD design also allows the user to manage the angles limits of the servomotors. The algorithm is used in a development platform, which will act as a controller. When implemented along the articulated structure, the controller will operate as an embedded system capable of moving the structure. The system also allows the deployment of sensors capable of generating interrupts, animations or even seek mechanical stability through control of the digital servomotors. The process despite sequence was able to articulate twelve different servomotors without any significant delay in movement. In short, the controller becomes a good choice for use in embedded systems, running for automation and control structure.

Keywords: Automation. Servomotor. Control. Embedded Systems. Cad.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CV	Constant Voltage
CC	Corrente Contínua
VCC	Tensão de Alimentação
GND	Neutro
PWM	Pulse Width Modulation
PIC	Peripheral Interface Controller
SRAM	Static Random Access Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
VBA	Visual Basic for Applications
RARM	Right Arm Module, Módulo do braço direito
LARM	Left Arm Module, Módulo do braço esquerdo
HEAD	Head Module, Módulo da cabeça
T_{Total}	Período Total do Loop de Processo
T_{Grau}	Período para deslocar fisicamente 1º grau do servo motor.
T_{servo}	Período máximo para comunicação com servo HS-311
$T_{processamento}$	Período estipulado para processamento
V_{max}	Velocidade Máxima do servo HS-311

LISTA DE SÍMBOLOS

©	Copyright
™	Trademark
Hz	Hertz
s	Segundos
V	Tensão
A	Amperagem
KB	Kilobytes
MHz	Mega Hertz

SUMÁRIO

1 INTRODUÇÃO

Com a evolução dos sistemas de automação, tornou-se imprescindível o uso de atuadores em processos com cargas variantes e de movimentos precisos. Como os motores convencionais são puramente a conversão de potência elétrica em mecânica, a primeira forma disponível para controle direto do motor seria o controle da fonte de alimentação, gerenciando assim a potência e o tempo de fornecimento. Porém, para controle da posição, observou-se, nestas situações, que, devido à presença de cargas variantes, não se pode assumi-las como inercias e deduzir o torque necessário de maneira proporcional à aceleração, pois, para cargas diferentes, o mesmo torque aplicado, durante o mesmo tempo, obter-se-ia uma posição final diferente. Para solucionar esse problema, utiliza-se um atuador controlado, o servo motor retro alimentado com a posição.

Na área da robótica, os servos motores ganharam muito espaço devido a sua capacidade de orientar-se com precisão. No caso estudado neste projeto, os servos motores utilizados foram os digitais HS-311 da HITEC© que são, fundamentalmente, a combinação simples de um motor CC (corrente contínua), um microprocessador retroalimentado com a posição para o controle do circuito, um potenciômetro e um conjunto de engrenagens. Para ativar o servo motor há três canais, onde dois canais são destinados à alimentação e o terceiro é responsável pelo sinal de controle. Através de um sinal PWM[1] (Pulse Width Modulation), inserido no terceiro canal, pode-se alterar a posição angular do eixo do motor. Sendo assim, os servos diferem dos motores convencionais, pois não são construídos com intuito de girar livremente, mas para serem aplicados onde é necessário saber precisamente a posição do eixo.

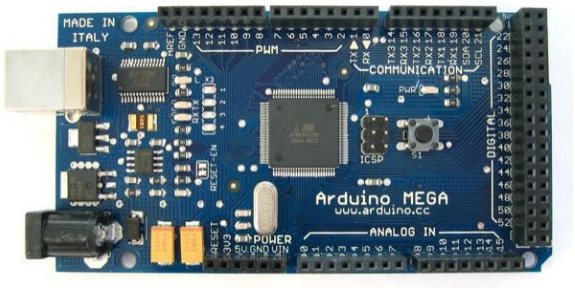
A aplicação mais comum de servo motores encontra-se no controle e na automação em sistemas embarcados. No aeromodelismo, por exemplo, servo motores são utilizados para posicionar os *ailerons*[2], profundores, entre outras estruturas, em que a posição precisa é essencial. Na robótica, servo motores são responsáveis pela movimentação e, em alguns casos, pela locomoção do robô.

Contudo, para controlar múltiplos servos e movimentá-los ao mesmo tempo, é necessário emitir múltiplos sinais, ou, ao menos, emitir um único sinal e comutar a saída entre os servos. Para um processador com processamento sequencial, ou seja, executando

[1]PWM: Modulação por Largura de Pulso, em inglês Pulse Width Modulation.
[2]Ailerons: São as partes móveis presentes nas asas de uma aeronave usadas para o movimento de rolamento.

uma linha de código por vez, não é possível emitir múltiplos sinais PWM simultaneamente, a menos que esse processador tenha osciladores dedicados. No caso da placa de controle utilizada, o *Arduino*™ Mega 2560 (ver Figura 1), há apenas um cristal oscilador operando na frequência de 16MHz. Apesar de possuir quatorze pinos que podem ser dedicados à PWM, cada canal exige processamento da placa, reduzindo a frequência de atuação individual do sinal PWM. Sendo assim, pretende-se gerar um algoritmo que otimize o processo de movimentação de múltiplos servos e permita a inserção de funções utilizando sensores. No caso estudado, o algoritmo deve processar a movimentação dos braços e da cabeça de um robô e, ainda, garantir que o usuário tenha informações sobre a estabilidade do robô.

Figura 1 Arduino Mega 2560

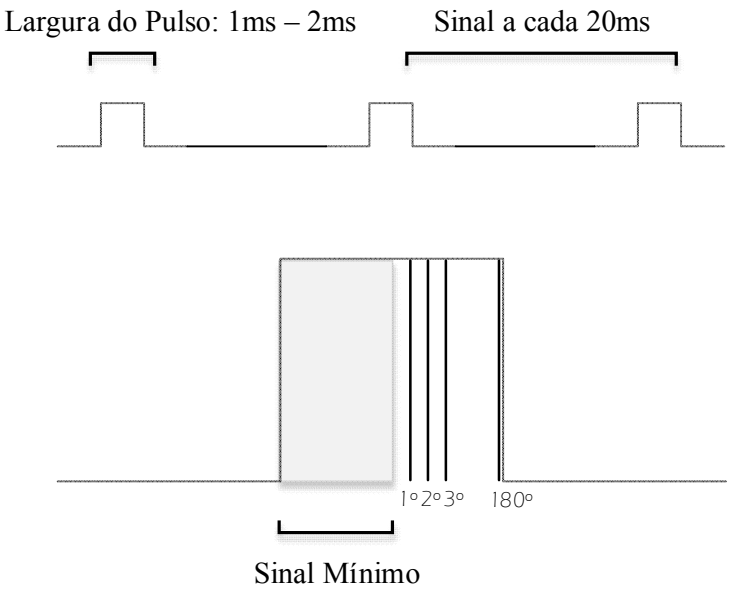


Fonte: www.arduino.cc

Para o modelo de servo motor utilizado, segundo as referências presentes no manual do fabricante, sabe-se que o canal de comando do servo motor funciona na frequência de 50Hz com razão cíclica variando entre 5% a 10%. Em outras palavras, como visto na Figura 2, a largura dos pulsos deve estar entre 1ms e 2ms. Para garantir que o microprocessador no servo mantenha a posição desejada, o pulso deve ser enviado novamente a cada 20ms.

Como visto no manual, sabe-se que o pulso de 1ms corresponde ao ângulo 0º do motor e que o pulso de 2ms corresponde ao ângulo 180º do motor, temos, então, que a variação de 1ms corresponde à variação de 180º no motor. Sendo assim, pode-se assumir que, para a precisão de um grau no eixo do motor, o processador deve operar, no mínimo, com um frequência de 180KHz.

Figura 2 PWM para Servo Motor



Fonte: Própria

Entretanto, o processador não pode se dedicar exclusivamente à emitir um, ou quatorze sinais PWMs para os servos motores. Como deseja-se torná-lo automático, é necessário estabelecer, também, funções para comunicação com meio externo (no caso, via serial[3]), interpretar as funções recebidas, analisar sensores de controle, comutar saídas, entre outras funções que consumirão tempo de processamento. O algoritmo utilizado ajusta essas funções no código, de modo a não comprometer o funcionamento dos servos motores e, ainda, garante que qualquer alteração futura (que busque inserir novos comandos, gere resposta a sensores ou que acrescente outro dispositivo para armazenar dados) seja facilmente implementada.

Em suma, o projeto pretende gerar um algoritmo para controlar doze, já que dois canais entre os quatorze serão utilizados para comunicação, servo motores (com o intuito de movimentar a estrutura de formato humanoide), inserir respostas automáticas à sensores (como acelerômetro e de presença) e possibilitar ao usuário a fácil modificação dessas funções através de uma interface simples.

[3] Serial: Porta de comunicação utilizada para transferência em série.

2 ESPECIFICAÇÕES DE PROJETO

Para gerar um algoritmo de controle, deve-se definir, primeiramente, com que tipo de servo mecanismo irá se trabalhar, assim como qual modelo de processador se utilizará. Neste capítulo, justificam-se as escolhas dos principais *hardwares* utilizados.

2.1 Servos Motores

Para movimentar qualquer estrutura, veículo ou corpo de maneira mecânica, é necessário aplicar um motor a essa estrutura, de modo a converter energia elétrica em energia mecânica. Em certos casos, é essencial o controle da posição com precisão. Nestes condições, deve-se aplicar um motor controlado retroalimentado com a posição, como é o caso do HS-311 da HITEC© utilizado neste projeto.

2.1.1 Como funciona o HS-311

Projetado para posicionar-se com precisão em volta de seu eixo, o servo motor utilizado é a combinação de um motor CC comum, conectado à um potenciômetro e controlado por um microprocessador.

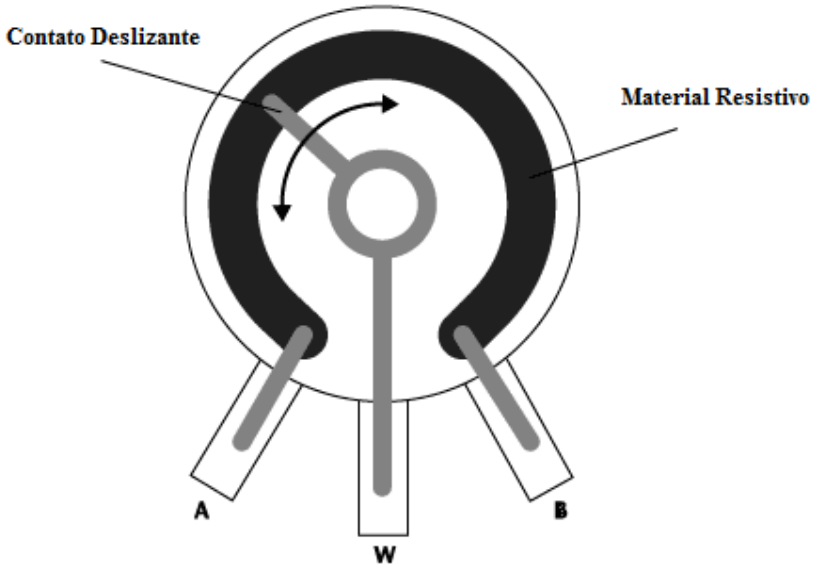
Figura 3 Componentes do Servo

Fonte: Eng Leandro Alves

O microprocessador do sistema de um servo motor é o cerne de sua funcionalidade, pois este é responsável por interpretar o sinal de comando, o sinal PWM, ler o sinal analógico do potenciômetro, que indicará a posição, e controlar o motor CC, pois, mesmo após atingir a posição aspirada, o motor deve manter a sua posição, gerando resistência à movimentos radiais no eixo.

O potenciômetro é um dispositivo de resistência variável, formado por três terminais com um contato deslizante, gerando um divisor de tensão. Sendo assim, ao usar dois, dos três terminais, obter-se-á um resistor variável. Para alterar a posição do contato deslizante, há um eixo de rotação que varia, normalmente, num intervalo de 270º. O eixo do potenciômetro é conectado ao eixo do motor, sendo assim, quando o motor modificar sua posição, o potenciômetro irá o acompanhar, alterando, assim, a tensão observada pelo microprocessador. Importante observar que, por esse motivo, devido ao acoplamento dos eixos do motor e do potenciômetro, há um limite no ângulo de atuação do servo.

Figura 4 Elementos de um Potenciômetro



Fonte: FDDRSN

Por fim, há o motor CC, o conversor de energia elétrica em energia mecânica, que realiza o trabalho de todo dispositivo servo motor. Ao escolher um servo, há dois critérios importantes que definirão a escolha: o torque e a velocidade de rotação. Ambos

critérios são fornecidos pelo motor, que está diretamente vinculado ao tamanho do componente.

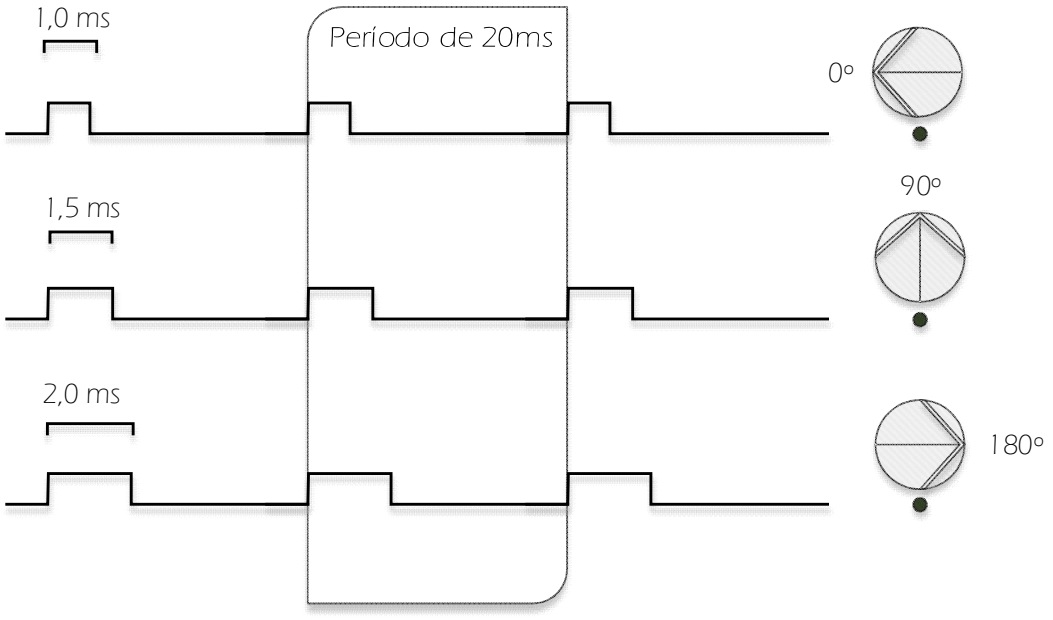
Em suma, este servo motor é uma tríade de processamento, motor e potenciômetro, restrito à um espaço volumétrico pequeno, pois agrega os três componentes no mesmo encapsulamento, tornando-se viável para aplicações precisas.

2.1.2 Acionamento do HS-311

Após compreender o funcionamento do servo motor, a sua aplicação se torna simples. Para comandá-lo, basta aplicar um pulso com largura modulada, em se tratando de servos motores da HITEC©, os valores previstos para este PWM é de: pulsos na frequência de 50Hz com larguras entre um e dois milissegundos, ou seja, razão cíclica entre 5% a 10%.

Como visto na Figura 5, a orientação do eixo do servo motor é função da largura do pulso no canal de comando. Para uma largura de 1ms, seu ângulo no eixo do motor é de 0º, para uma largura de 1,0055ms seu ângulo é de 1º no eixo do motor, e assim por diante. Quando o pulso tiver largura de 1,5ms o eixo estará em 90º e em 2ms o eixo do motor se encontrará em 180º.

Figura 5 Orientação da PWM para Servo Motor



Fonte: Própria

Observa-se que a largura do pulso dita a posição almejada pelo servo, sendo assim, o controlador deve operar com um ciclo a cada:

$$T_{servo} = {}^{1(ms)}\!/_{180(intervalos)} = 5{,}56\mu s \qquad (1)$$

Onde T_{servo} representa o tempo que o sinal deve ficar em alta para variar 1º grau do servo motor. Ou seja, para controladores dedicados, a frequência mínima para operar este servo, ou F_{servo}, será:

$$F_{servo} = {}^{1}\!/_{T_{servo}} = 180\ kHz \qquad (2)$$

Na situação estudada neste documento, no qual o processo não é dedicado, o período de cada orientação PWM vai ser importante para calcular o *delay*[4] do processo (Ver Item: "3.4 Estimativa do tempo de processo", pág.:38). Após gerar o sinal PWM, torna-se simples visualizar como o processador irá controlar os servos motores.

2.2 Arduino ™

Como apresentado no site do processador, o Arduino foi criado por um professor chamado *Massimo Banzi*, com a finalidade de ensinar eletrônica e programação de computadores. Neste intuito, Massimo, David Cuartielles e David Mellis desenvolveram as placas conhecidas como Arduino. Devido a simplicidade do uso e a proximidade com a linguagem C para programação, a aceitação dos novos processadores foi muito grande (FACOM, 2012).

2.2.1 Conhecendo o Arduino ™

Essencialmente, o Arduino é uma placa de desenvolvimento que disponibiliza entradas analógicas e digitais, leds, pinos, conectores, plugue para alimentação própria e outros dispositivos que facilitam seu uso em sistemas embarcados. No momento da programação a porta USB fornecerá alimentação necessária para simular e testar o software em desenvolvimento.

[4] Delay: Atraso temporal.

No Arduino, informações ou ordens são transmitidas de um computador para a placa através de Bluetooth, wireless, USB, infravermelho, etc. Essas informações devem ser traduzidas utilizando a linguagem *Wiring* baseada em C/C++.

2.2.2 Vantagens do Arduino ™

Ideal para qualquer estudante, o Arduino além de possuir um software livre para programação, baseado em *java*, que é compatível com as atuais plataformas; conta, ainda, com uma enciclopédia de códigos que vão do básico ao complexo, para facilitar a aprendizado do estudante.

O Arduino possui um preço acessível para estudante, seus módulos podem ser adquiridos entre 40 e 110 reais. A linguagem simples do Arduino, baseada em C/C++, atrai muitos iniciantes o que gerou uma grande comunidade ativa de usuários.

2.2.3 Modelos e especificações

Há, entretanto, diversos modelos de *Arduinos*™ que podem ser utilizados a fim de realizar a função descrita. Uma visão melhor da capacidade de cada *Arduino*™ pode ser obtida ao comparar os modelos existente.

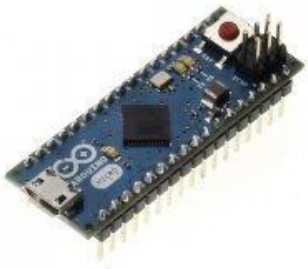
Figura 6 Arduino Uno

Fonte: http://store.arduino.cc/

O *Arduino*™ Uno, modelo mais utilizado atualmente, é bastante compacto e utiliza um microprocessador com encapsulamento do tipo *Dual in-Line Package*, o ATmega328, com 16MHz como frequência de operação. Apesar disso ele apresenta apenas 6 saídas para sinal PWM e uma memória FLASH de 32KB. Porém, com uma

velocidade de 16Mhz, 6 servos motores não serão o suficiente para sobrecarregar a placa. Sendo assim, opta-se por outro modelo.

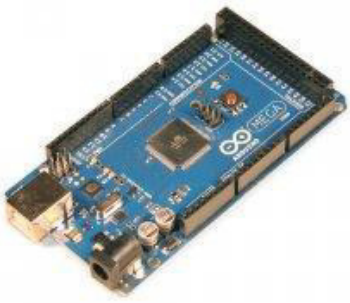
Figura 7 Arduino Micro



Fonte: http://store.arduino.cc/

O *Arduino*™ Micro é um modelo mais moderno e tende a substituir a plataforma Uno de aprendizado. Este modelo é muito similar ao Uno, porém é bem mais compacto e, além disso, apresenta 7 saídas para sinal PWMs. Mesmo assim, ainda não é o modelo mais apropriado para este projeto (ARDUINO, 2013).

Figura 8 Arduino Mega



Fonte: http://store.arduino.cc/

O *Arduino*™ Mega é o modelo mais adequado para o projeto, pois além de apresentar 16MHz como frequência de operação, ele dispõe de 256KB de *flash*, o que permitirá a inserção de várias bibliotecas para sensores e possui quatorze saídas para o

sinal PWM. Constam nele outros quarenta pinos digitais e dezesseis pinos analógicos que serão utilizados para implementação de sensores e outras funções (ARDUINO, 2013).

Como visto, a maioria dos processadores das placas ofertadas pela Arduino trabalham na frequência próxima de 16MHz, o que supera, em muito, a necessidade estimada (Ver Item: "3.4 Estimativa do tempo de processo", pág.: 38). Sendo assim, o critério para a seleção da placa foi a quantidade de pinos dedicados para PWM, e a relação de Memória FLASH e SRAM utilizadas para inserir o código da programação.

Tabela 1 Comparativo entre Arduino

	Arduino Uno	**Arduino Micro**	**Arduino Mega**
Microcontrolador	ATmega328	ATmega32u4	ATmega2560
Pinos I/O Digitais	14 (6 para PWM)	20 (7 para PWM)	54 (14 para PWM)
Entrada Analógica	6	12	16
Memória Flash	32 KB	32 KB	256 KB
SRAM	2 KB (ATmega328)	2.5 KB (ATmega32u4)	8 KB (ATmega2560)
EEPROM	1 KB (ATmega328)	1 KB (ATmega32u4)	4 KB (ATmega2560)
Frequência de Operação	16 MHz	16 MHz	16 MHz

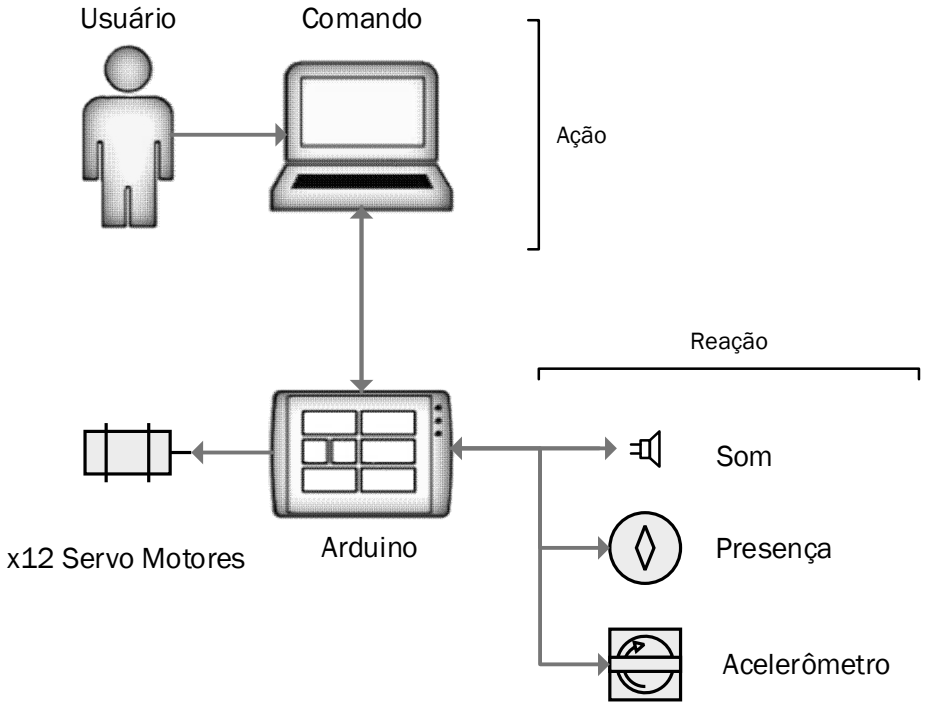
Fonte: www.arduino.cc

Como visto na Tabela 1, o *Arduino*™ Mega 2560 Rev3 supera a quantidade de pinos dedicados para sinal PWM, o que permitirá testar o algoritmo com um quantidade considerável de servo motores, ou seja, será possível alocar ao menos 12 servo motores (duas portas dedicadas à PWM são utilizadas para a comunicação serial), utilizar sensores que consomem grande tempo de processamento (como acelerômetro, ultrassom) e enviar diversos comandos via serial USB. Em resumo, o Arduino Mega permitirá analisar melhor o limite de servos implementados na operação, o que acaba resultando em um custo por servo menor, já que um único processador controlaria uma quantidade superior de servos motores.

3 ALGORITMO

Objetivando o comando de doze servo motores para mobilizar uma estrutura robótica, pode-se utilizar informações oriunda do usuários (as ações propriamente ditas) e as nativas na programação dos sensores (as reações do sistema). Observando a Figura 9, tem-se que as ações são comandos que podem ser pré-programados pelo usuário e necessitam de uma interface simples, para facilitar a adição de novas funções. Porém, as reações são funções que buscam estabilizar tanto fisicamente, o robô, quanto processualmente.

Figura 9 Lógica de Atuação



Fonte: Própria

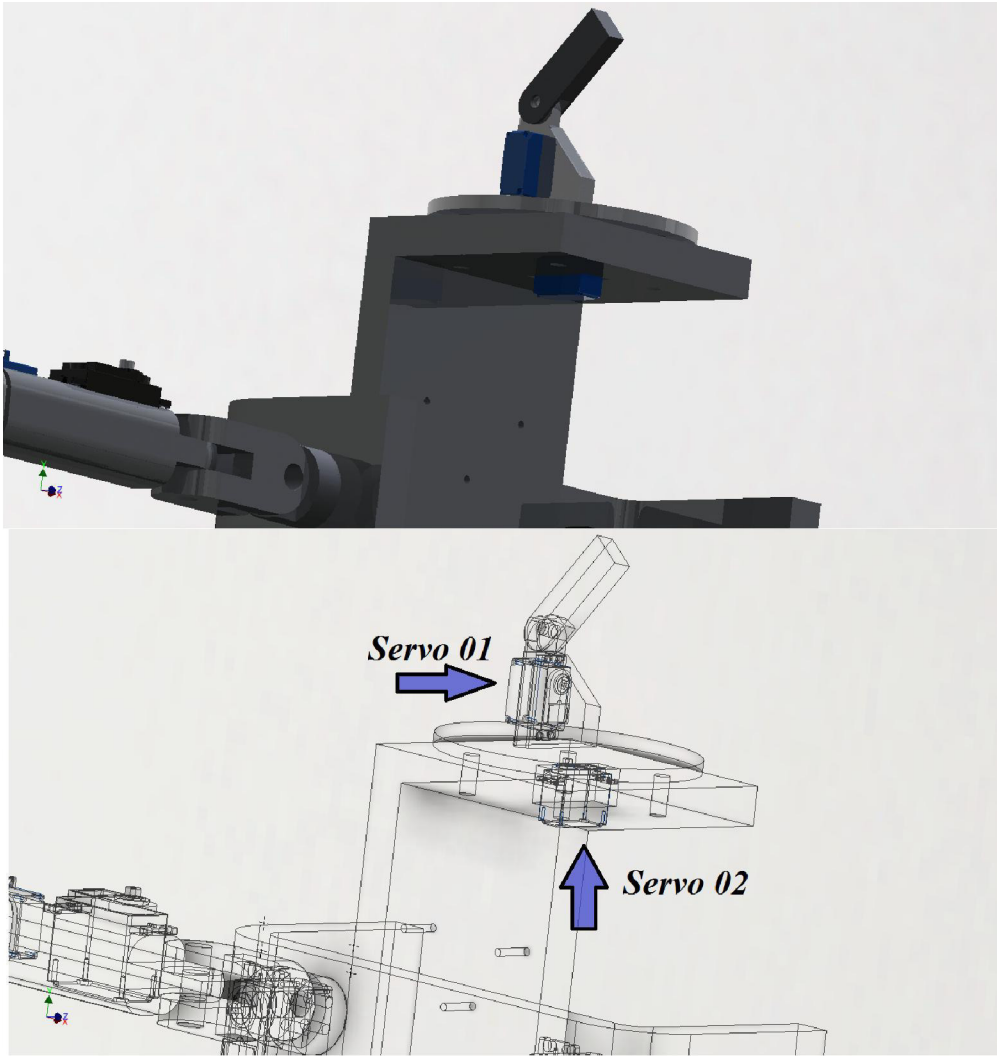
3.1 Módulos de Operação

Antes de compreender o processo de ação e reação do sistema, deve-se entender que certos processos devem ocorrer em paralelo, ou seja, a atuação da mão esquerda deve ocorrer em paralelo com a direita, as ações pertinentes à cabeça não devem interromper as ações das mãos, entre outros. Sendo assim, separa-se os doze servos em três módulos de atuação:

3.1.1 Cabeça

O módulo que representa a cabeça possui apenas dois servo, um atua em movimentos radiais (Ver Figura 10: Servo se encontra em azul, abaixo da cabeça.) e é responsável pela animação de movimentos como o de negação, o segundo se encontra sobre a plataforma orientada do primeiro (Ver Figura 10: Servo se encontra em azul, sobre a plataforma circular. A imagem foi gerada no *Autodesk® Inventor® 2014*) e é responsável pela animação de movimentos como o de afirmação. A combinação dos dois servo motores define o módulo Cabeça do sistema e trabalha de maneira independente dos demais.

Figura 10 Projeto em CAD da Cabeça

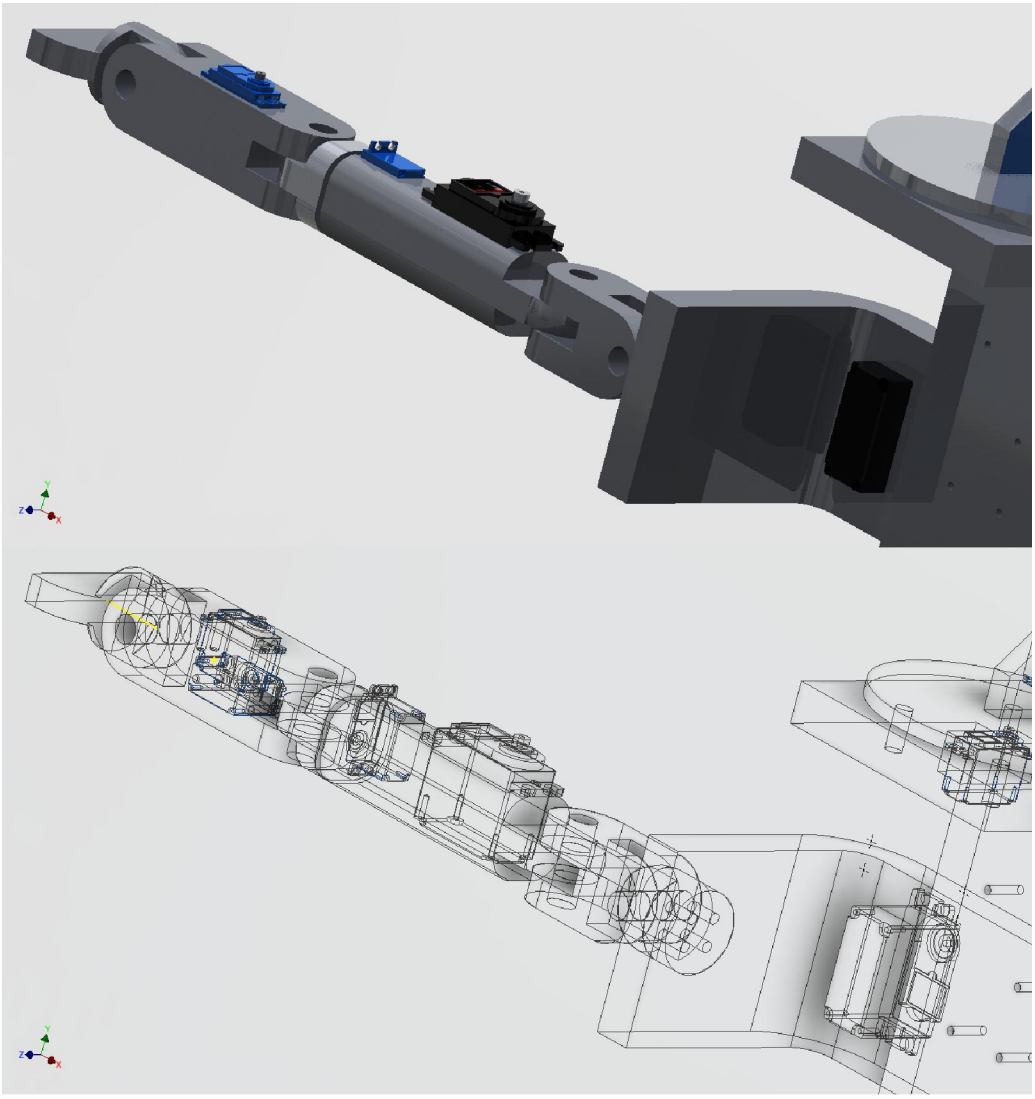


Fonte: Própria

3.1.2 Braço Direito

O módulo do braço direito possui no total cinco servo. Dois atuam nos eixo do ombro, um terceiro atua no eixo do cotovelo, outro é radial ao cotovelo e um quinto atua no eixo do pulso. Como mostrado na Figura 11, os cinco servo são dispostos ao longo da estrutura e são inseridos dentro dos moldes, a fim de economizar espaço. A transmissão do eixo do motor, até o eixo de atuação é feita através de polias (não representadas nas figuras) ou, no caso do servo atuando de maneira radial, há engrenagens os conectando à estrutura a ser movimentada.

Figura 11 Projeto em CAD do Braço Direito

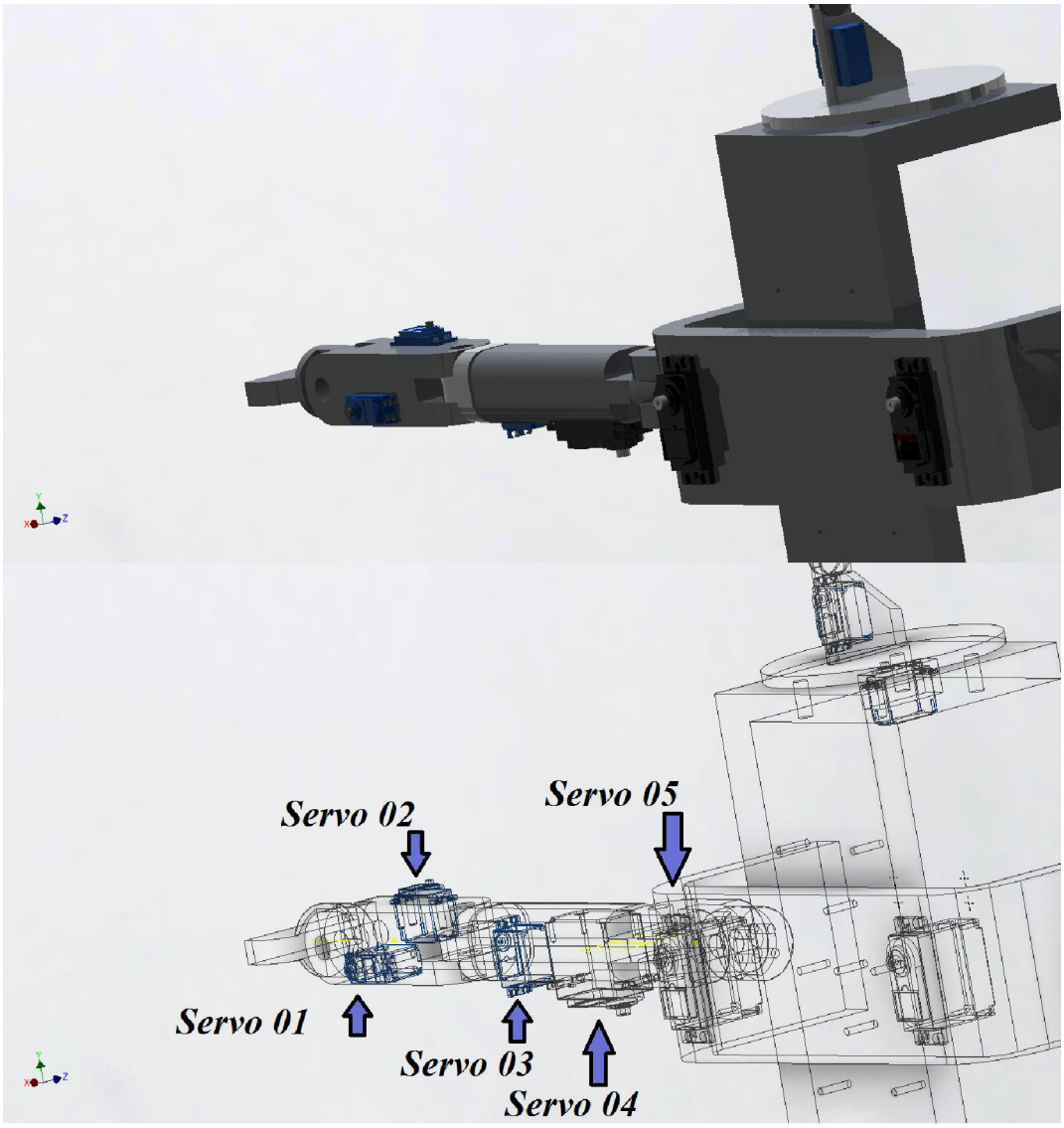


Fonte: Própria

3.1.3 Braço Esquerdo

O módulo do braço esquerdo também possui cinco servo, pois a estrutura funcionará de maneira análoga ao braço direito. Os dois módulos dos braços trabalham de maneiras distintas, ou seja, as ações pertinentes ao braço esquerdo não interferem nas ações em execução do braço direito.

Figura 12 Projeto em CAD do Braço Esquerdo

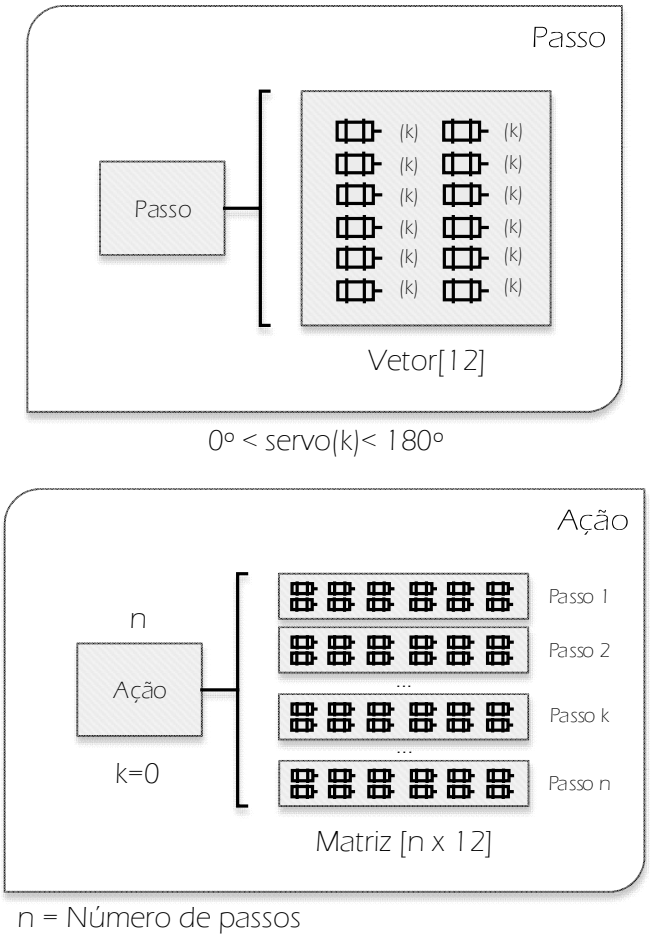


Fonte: Própria

3.2 Gerando Ações

Como visto na Figura 9, as ações são enviadas pelo usuário, sendo assim, elas devem ser previamente cadastradas no *Arduino™*. Uma ação pode ser resumida em um conjunto de posições que a estrutura deve assumir, neste caso, define-se então que cada posição será um passo a ser realizado pelo processador e cada passo deve indicar a posição alvo de cada servo. Ou seja, uma ação é definida ao se determinar o número de passos da ação e a posição alvo de cada passo:

Figura 13 Algoritmo da Ação

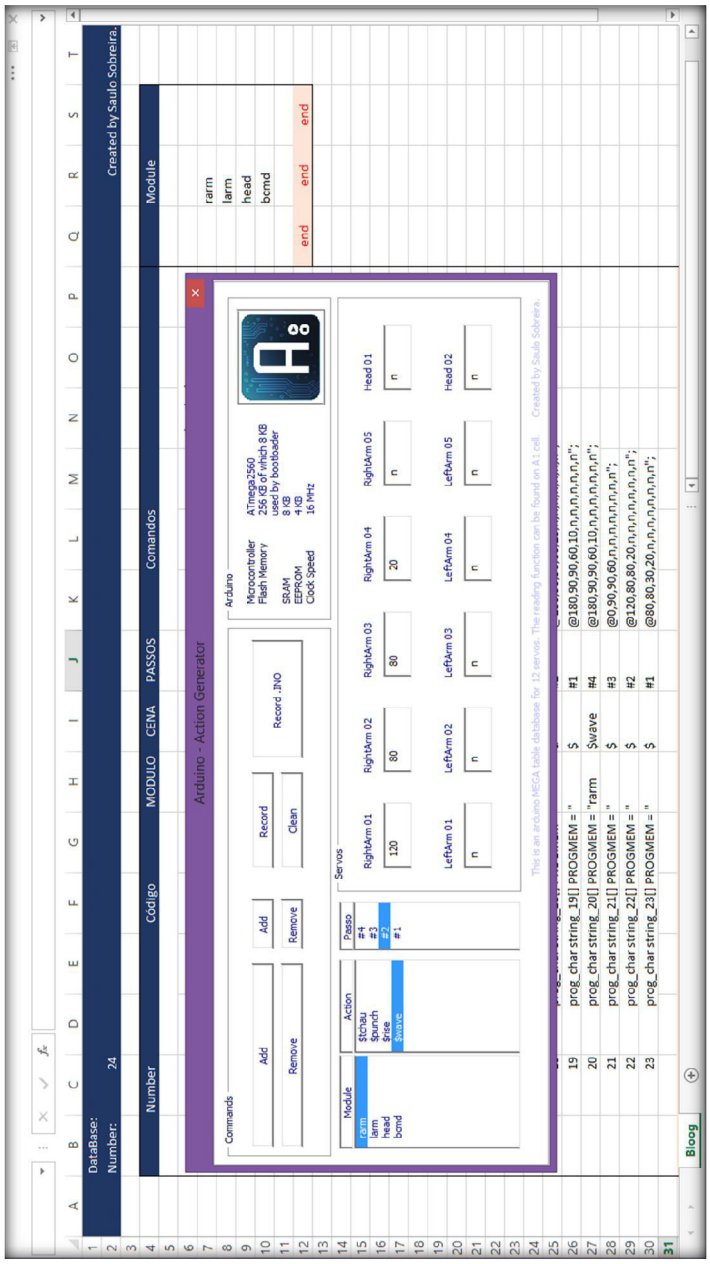


Fonte: Própria

Pode-se observar pela Figura 13 que uma ação será uma matriz de *n* linhas, onde *n* é o número de passos da ação (número de posições que a estrutura deve assumir ao longo da animação), por 12 colunas, onde cada coluna representa um dos servos e cada elemento da matriz é um valor entre 0 a 180, que indicará o ângulo alvo do servo.

Após definir o que será uma ação, criou-se uma planilha no *Microsoft® Excel* capaz de gerar a "matriz ação" que o controlador interpretará. Essa planilha é uma interface entre o usuário e o processo que facilitará a criação de novos movimentos e a alteração dos existentes.

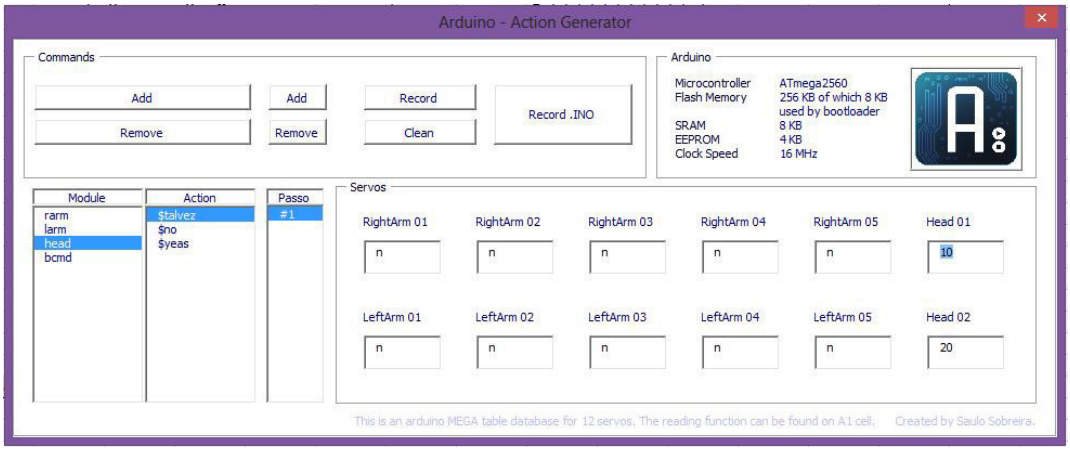
Figura 14 Planilha de Geração de Código



Fonte: Própria

Na Figura 14, observa-se que a planilha permite ao usuário a composição das ações através de uma interface gerada em *Visual Basic,* linguagem de programação gerada pela empresa *Microsoft®*. Os comandos do software consentem a capacidade de modificar cada passo e gerar novas ações e permite salvar a "matriz ação" no formato lido pelo processador Arduino.

Figura 15 Interface da Planilha



Fonte: Própria

Pela Figura 15, nota-se que o usuário tem acesso aos módulos (representados na primeira coluna), e às ações pertinentes ao modulo selecionado (representados na segunda coluna), sobre essas colunas há os comandos de adição e remoção de ações. Ao lado, na terceira coluna, há os passo relativos à ação selecionada, sobre esta coluna há os botões que permitem o usuário adicionar ou remover passos da ação selecionada. A interface ao lado indica qual a posição alvo dos servos para aquele respectivo passo, ou seja, é o vetor que indica a posição em que o robô ficara quando a ação "talvez" do módulo HEAD for executada.

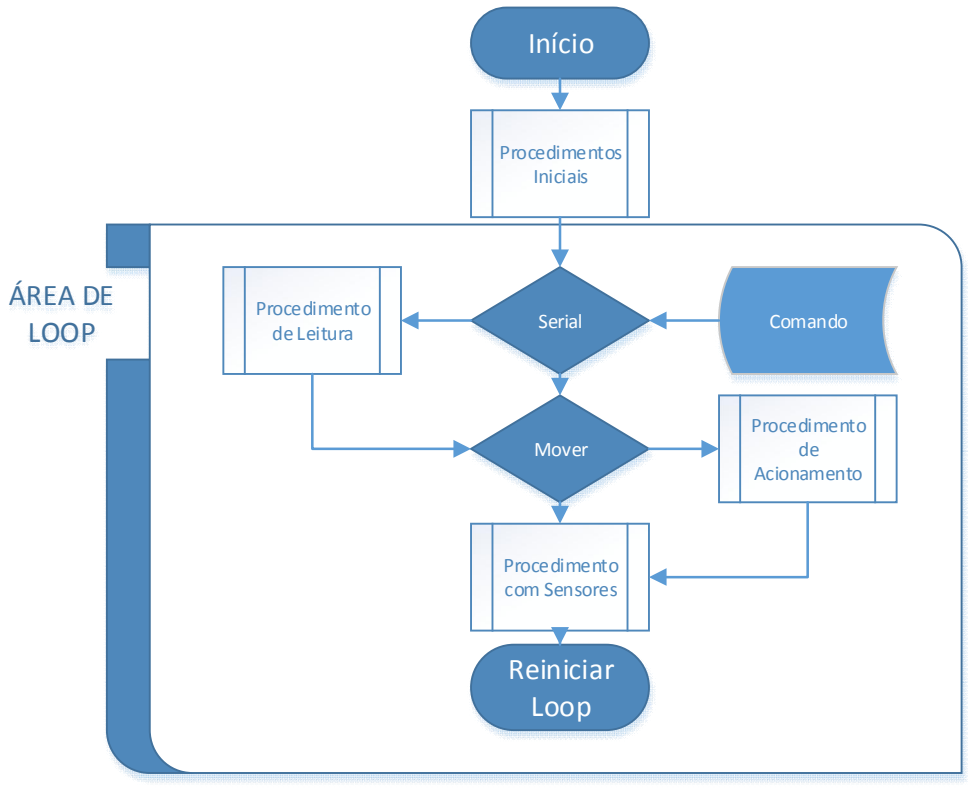
Esta interface permite ao usuário gerar facilmente sua "matriz ação". Quando finalizada o usuário pode utilizar o comando "Record .INO" e gerar o arquivo que é utilizado pelo *Arduino™*.

3.3 Lógica de Operação

A ordem de execução, os cálculos a serem realizados, a comunicação serial e os testes lógicos de sensores, podem ser realizadas de maneiras distintas, gerando, assim, uma infinidade de possíveis *softwares* capazes de executar a mesma tarefa. Obviamente, haverá opções em que o software funcionará de maneira ótima, ou seja, realizará os procedimentos em tempo hábil sem afetar a dinâmica dos movimentos.

Sendo assim, utilizar-se-á, inicialmente, uma lógica empírica, com subprocessos a serem desenvolvidos com o fim de achar um modelo ótimo de atuação. No organograma da Figura 16, pode-se observar os procedimentos de comando (enviado pelo usuário), leitura, acionamentos de servos, avaliação de sensores e, por fim, o reinicio do *loop*. Então, esses são os procedimentos que podem ser otimizados com propósito de minimizar o tempo de processamento.

Figura 16 Organograma Resumido

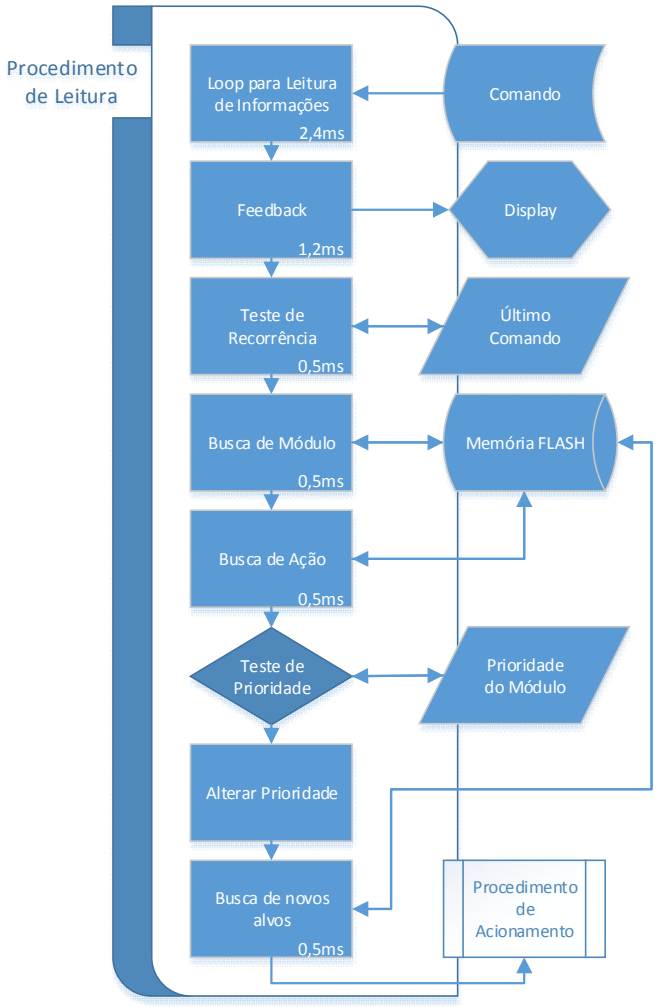


Fonte: Própria

3.3.1 Procedimento de Leitura

O procedimento se resume em: obter o comando enviado do usuário, retornar o valor, para assegurar comunicação adequada; haverá, em seguida, o teste de recorrência para identificar se o comando atual é o mesmo do comando anterior, caso verdadeiro, o procedimento segue para a seção de acionamento. Observe que, como fomentado no item "3.1 Módulos de Operação" deste mesmo capítulo, o próximo passo assegura que o comando seja endereçado a um dos módulo. Sendo assim, após o teste de recorrência, o teste de prioridade será relativo à atual ação sendo executada pelo módulo a que se destina o novo comando. Por fim, altera-se o alvo de cada servo, dando início ao procedimento de acionamento.

Figura 17 Organograma para Leitura



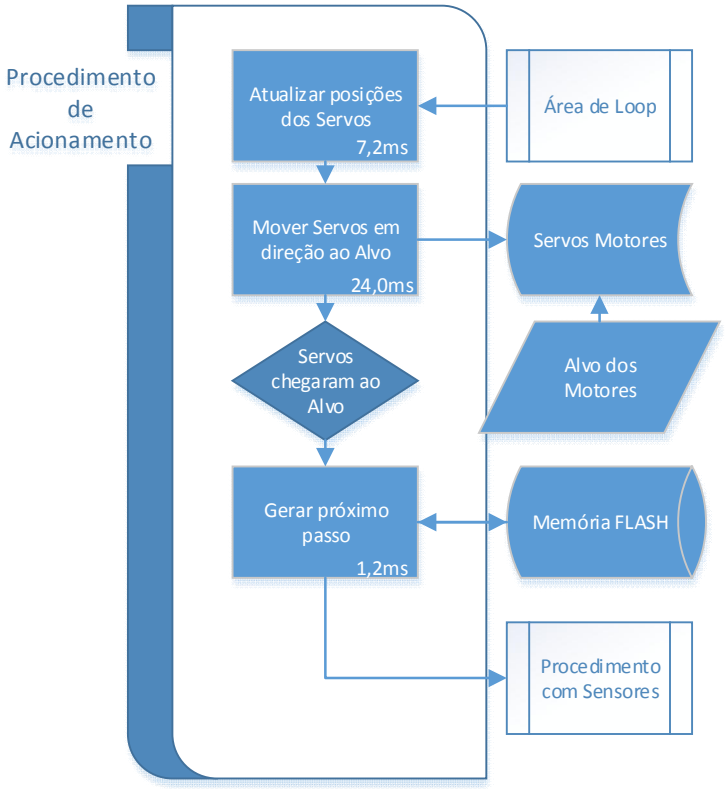
Fonte: Própria

O procedimento de Leitura é simples. Este objetiva, como visto na Figura 17, receber e interpretar as informações oriundas do usuário. O código pode ser obtido no anexo “7.1 Funções de Comunicação”, na página 52.

3.3.2 Procedimento de Acionamento

O procedimento de Acionamento é ainda mais simples. Como a Figura 18 demonstra, as ações provenientes do procedimento anterior, o procedimento de Leitura, são as entradas da nova expressão. Antes de executar qualquer deslocamento, deve-se identificar qual a posição atual dos servos motores (apesar das informações pertinentes aos servos estarem sempre presentes na memória do programa, uma ação externa poderia alterar a posição do servo sem o consentimento do software). Após atualizar as posições, deve-se deslocar os servos com um passo, estipulado no item “3.4 Estimativa do tempo de processo”, variante. Por fim, o procedimento termina e segue-se para os demais procedimentos com sensores.

Figura 18 Organograma para Acionamento

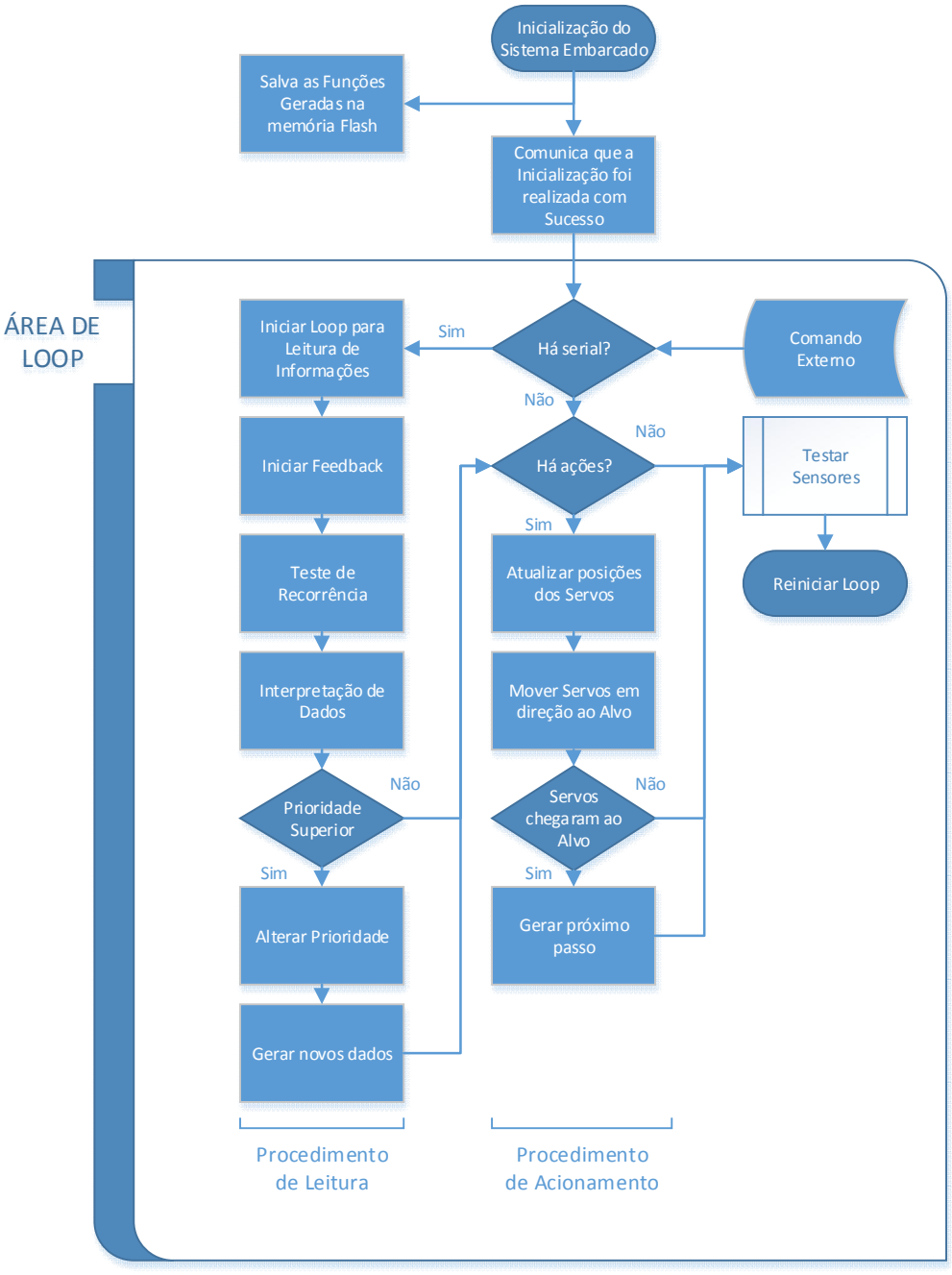


Fonte: Própria

3.3.3 Organograma

Sabendo que o funcionamento do procedimento com sensores se altera em função da finalidade do projeto, conclui-se que, para finalizar o algoritmo basta concatenar os dois procedimentos vistos nos itens anteriores.

Figura 19 Organograma Completo



Fonte: Própria

Analisando a Figura 19, pode-se compreender o ciclo de operação:

- **Inicialização do Sistema Embarcado:** Neste momento o processador define as variáveis padrão (Alvo Padrão dos Servos, Atraso Padrão dos Servos, Nível de Prioridade dos Módulos igual a zero, entre outros). Define, também, o pino relativo à serial, aos sensores e aos servos.
- **Salva na memória FLASH:** O Código gerado pela planilha do item "3.2 Gerando Ações" é salvo na memória FLASH do processador, para aumentar o espaço da memória SRAM e, assim, aumentar a capacidade de alocação de memória para o processo.
- **Comunica que a Inicialização foi realizada com sucesso:** Essa informação garante que a inicialização ocorreu com sucesso, que, a partir desse momento, o processo entrará em loop e que o tempo de cada atuação precisa ser estimado para não afetar o processo.
- **Há serial:** Nesse momento, verifica-se o pino TX e RX da porta serial para saber se o usuário ou o controlador deseja se comunicar com o processador. Se não, o processador segue para o próximo passo, que é o acionamento Porém, se houver uma serial, ele irá seguir para a região de leitura.
    - **Loop para leitura de informações:** Este loop recolhe cada caractere da porta serial até a presença do byte de parada ou se a comunicação ultrapassar 0,5 s (critério do programador). No caso, está programada para suportar 2600 ciclos, aproximadamente 0,2 ms. Em seguida, concatenam-se os caracteres e forma a *string*[5] de comando. (Ver anexo: "7.1.1 Função de Leitura" pág.:52)
    - **Iniciar *Feedback*[6]:** O processador agora reenvia o comando recebido pela serial, assim, o usuário pode determinar se a serial está funcionando corretamente.
    - **Teste de Recorrência:** É um loop que verifica se a ação atual não é a mesma enviada na última vez, a fim de não reiniciar um processo que, por ventura, tenha sido enviado duas vezes.

[5] String: Vetor de Caracteres
[6] Feedback: Retroalimentação, retorno de informações.

- **Interpretação:** Busca-se, agora, o módulo, ou o conjunto de servos, que deve ser operado na ação desejada, assim como a função almejada, o número de passos da ação e a posição alvo dos servos. (Ver anexo: "7.1.2 Função de Interpretação" pág.:53)
- **Teste de Prioridade:** Teste se a prioridade da nova ação é superior à prioridade da ação anterior. Para que, assim, ações utilizadas para estabilizar a estrutura não possam ser interrompidas.
- **Alterando prioridades:** O processo define que a prioridade inerente de cada servo é a prioridade da respectiva função a ser executada.
- **Gerar novos alvos:** O ângulo alvo de cada servo é alterado para os valores exigidos pela função. Em seguida, a função retorna para a linha de código principal, e executa o trecho de acionamento.

- **Há Ação:** Através do contador de passos fornecido pelo procedimento anterior, quando uma função é alterada, e pelo procedimento seguinte, quando uma ação chega ao fim, é possível determinar se o processo deve seguir ou não para a área de acionamento. Se não houver mais passos a serem realizados eles seguem para o procedimento com sensores, caso contrário o processo seguirá para a área de atuação.
  - **Atualizar posição dos servos:** O processador faz a leitura do ângulo de cada servos para saber se estes se encontram nos seus respectivos ângulos almejados. (Ver anexo: "7.2.1 Função de Acionamento" pág.:56)
  - **Mover os servos:** Em seguida, o processador deslocará os servos em direção ao ângulo almejado. O passo de variação do ângulo pode ser pré-programado.
  - **Testar Servos em Alvo:** Testa-se se os servos atingiram seus alvos. A função só irá seguir para o próximo passo da animação caso todos os servos de um mesmo módulo tenham chegado ao seu destino. (Ver anexo: "7.2.2 Função de Acompanhamento" pág.:57)
  - **Gerar próximo passo:** Busca-se na memória *flash* o próximo passo da função em execução do modulo que tiver todos os seus servos nos respectivos alvos.
- **Testar sensores:** Executam-se as funções pré-programadas nos sensores.

3.4 Estimativa do tempo de processo

Importante observar que a atuação dos doze servos ocorre em sequência e como o processamento da placa não é dedicado, há um *gap* temporal até os servos motores serem atuados novamente. Outra consideração importante para estimar o tempo de processo é que, após o servo receber a posição desejada, ele levará um tempo para atingir essa posição, restrito pelas suas propriedades físicas. Calcula-se, então, a velocidade máxima angular do servo a fim de calcular o período de cada ciclo. Utilizando as informações presentes no *datasheet* (HITEC, 2002) do modelo HS-311:

$$V_{Max} = {0{,}19\ (\text{sec})}\Big/{60\ (graus)} \quad (3)$$

Pode-se estimar, assim, o período para cada grau variado:

$$T_{grau} = 3{,}17\ ms \quad (4)$$

Importante ressaltar que os dados acima são para situações em que o servo não apresente carga na ponta do eixo e esteja sendo alimentado por uma tensão de 4,8V, exatamente a tensão de saída do *Arduino*™.

Assumindo o uso de doze servos motores, o período de processamento seria o equivalente à somatória de doze sinais PWMs e o tempo de processamento do *Arduino*™ para os demais processos, que inclui leitura dos sensores, comunicação serial, entre outros processos.

Como visto nas referências do fabricante (olhar item: HITEC, *Datasheet* HS-311, página 49) e na Figura 5, o sinal de comando possui uma largura de no máximo dois milissegundos, sendo assim, podemos estimar:

$$T_{Total} = (12 * T_{servo}) + T_{Processamento} \quad (5)$$

$$T_{Total} = (12 * 2) + 33 = 57\ ms \quad (6)$$

O valor de 33 ms relativo ao tempo de processamento do *Arduino*™ foi retirado a partir de testes experimentais do algoritmo (ver: Figura 17, Figura 18, Figura 20), apesar desse valor alterar-se constantemente, sua parcela majoritária são as comunicações em serial com uma *baud rate*[7] de 115200, o máximo permitido pelo processador, gerando um atraso de aproximadamente 1,2 ms por comunicação de 128 caracteres, como visto na equação seguinte.

[7] Baud Rate: Velocidade de Sinalização, determinar a taxa de transmissão em bits por segundo

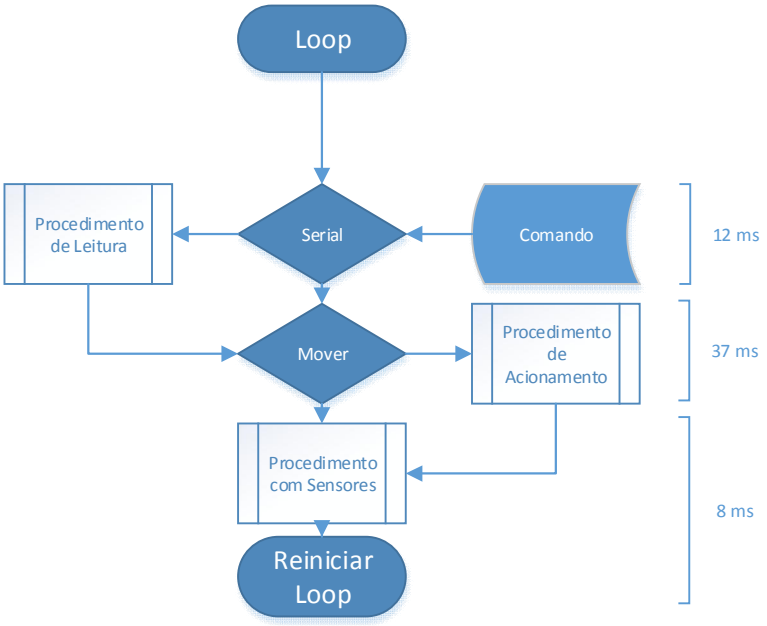
$$T_{com} = {}^{1}/_{BAUD} * nº\ caracteres = 1{,}21ms \qquad (7)$$

Com os valores obtidos nas equações anteriores, pode-se estimar o passo mínimo para a operação padrão, sem sensores digitais, com servos trabalhando a vazio.

$$Passo = {}^{T_{Total}}/_{T_{grau}} = 17{,}8° \qquad [8]$$

Obtém-se, assim, que o passo mínimo é de 18º graus, ou seja, enquanto o servo percorre os 18º graus impostos pelo algoritmo, este é capaz de executar um loop completo e comandar o servo motor novamente. Este ângulo também garante que a ação seja executada na velocidade máxima permitida pelo servo, caso seja do interesse do usuário um movimento mais lento, este pode deslocar os motores com passos inferiores a 18º, gerando assim um atraso de 3ms por grau reduzido. Ou seja, o giro completo de 180º graus com passos de 1º grau demoraria 0,5s a mais que o mesmo movimento com passos de 18º graus quando o eixo do servo se apresentar sem carga. Como o passo de cada servo pode ser programado pelo usuário, pode-se, assim, programar a velocidade de maneira indireta, utilizando o atraso inerente ao algoritmo como o atraso do movimento do servo.

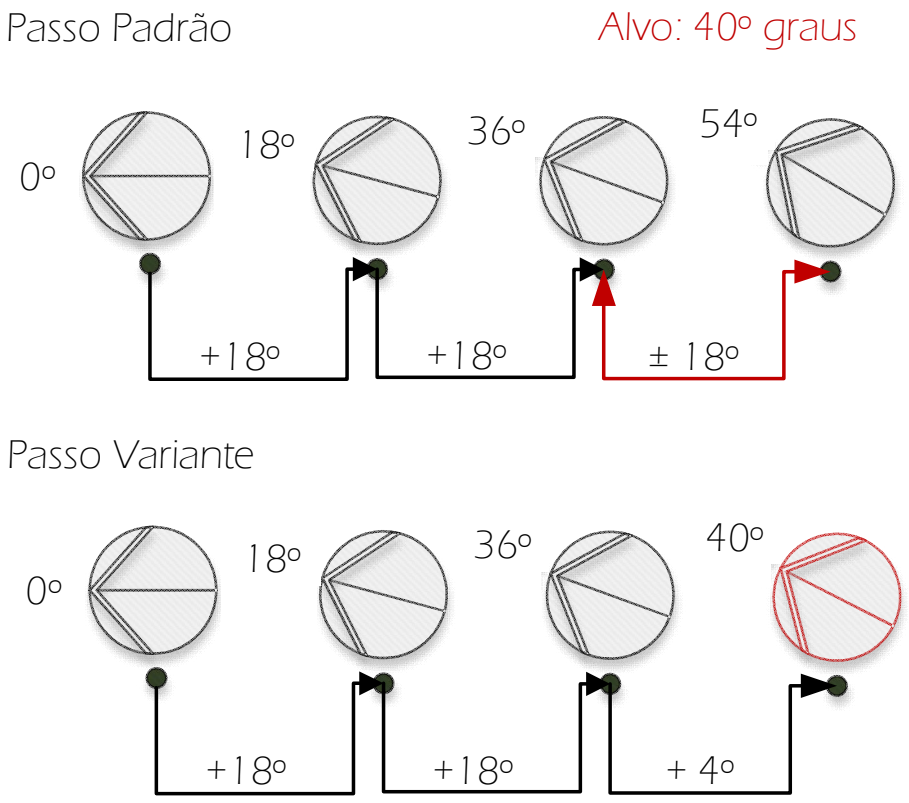
Figura 20 Tempo Estimado



Fonte: Própria

Pode-se observar, também, que esta lógica não irá permitir o servo atingir ângulos que não sejam múltiplos do passo (ver Figura 21), ou seja, o ângulo de 40º graus não seria atingido por passos de 18º graus que permutariam entre o ângulo de 36º graus e o de 54º graus. Por isso é implementado o passo variante, quando módulo da diferença entre a posição alvo e a posição atual é menor que o passo e, ainda assim, diferente de zero, o passo é o módulo da diferença.

Figura 21 Passo Variante



Fonte: Própria

Em suma, com o tempo de processo bem estipulado, pode-se orientar devidamente os servos, otimizando a precisão, e, ainda assim, controlar a velocidade dentro dos limites físicos permitido pela máquina. Na prática, sempre haverá presença de carga no eixo do motor o que reduz a velocidade de rotação do mesmo, garantindo um passo de iteração ainda menor. O algoritmo, apesar de não encontrar o passo ótimo para realização do movimento em velocidade máxima, permite ao usuário alterar os passos do movimento a vontade, garantindo atrasados na ordem de alguns milésimos de segundo não perceptíveis.

3.5 Definindo ângulo limite de rotação

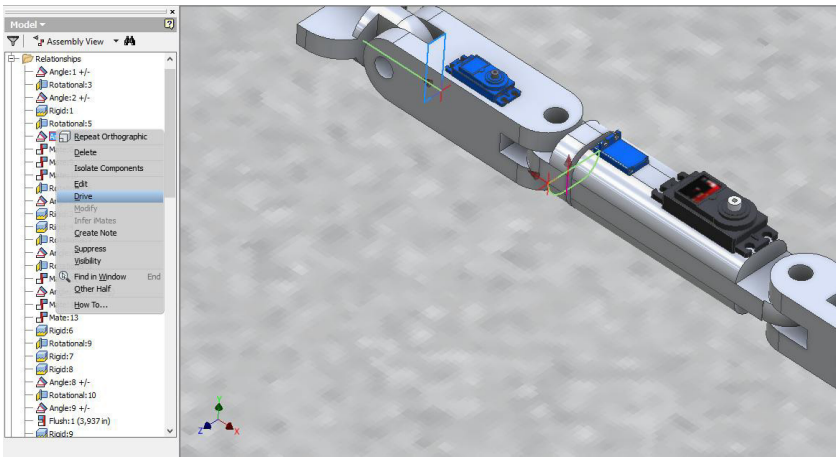
A planilha permite ao usuário definir, também, o ângulo limite de cada servo. Isto garante que o movimento inerente à um servo não irá colidir com outra parte física, gerando, assim, uma situação indesejada. Para determinar o valor deste ângulo, utiliza-se no projeto um protótipo gerado no *Autodesk® Inventor®* 2014, ver Figura 22.

Figura 22 Protótipo Ambientado

Fonte: Própria

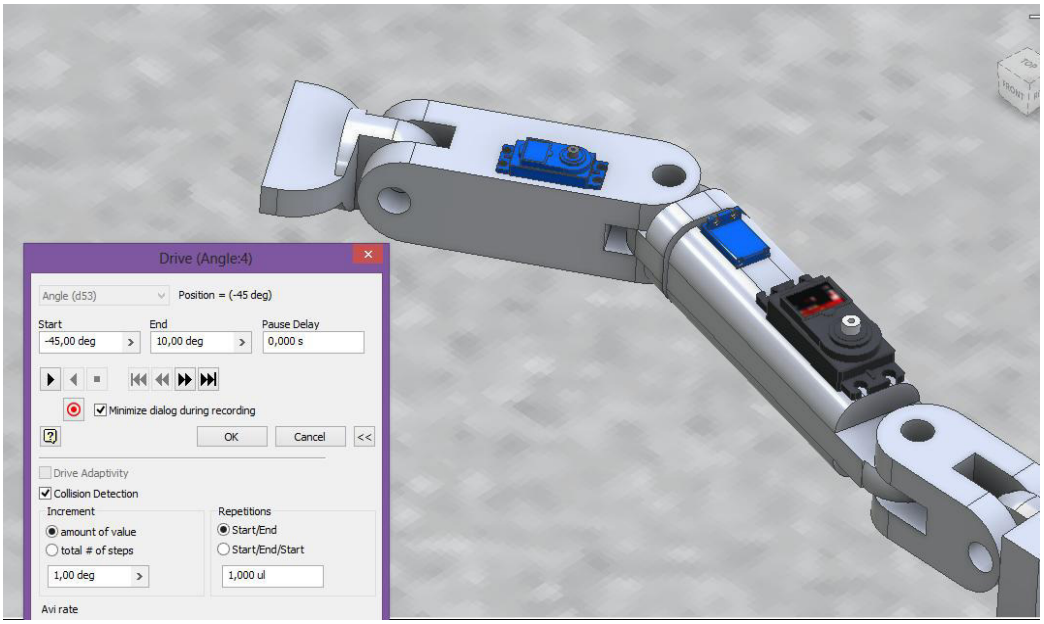
Após gerar o protótipo, o usuário deve determinar as restrições de movimento. Isso ativará a função "*drive*" do *Autodesk® Inventor®* 2014 (ver Figura 23), nela consta a orientação do servo, assim como os ângulos da animação. Pode-se, então, ativar a detecção de colisão, ferramenta útil para estipular os ângulos limite, ver Figura 24.

Figura 23 Ativando Função Drive



Fonte: Própria

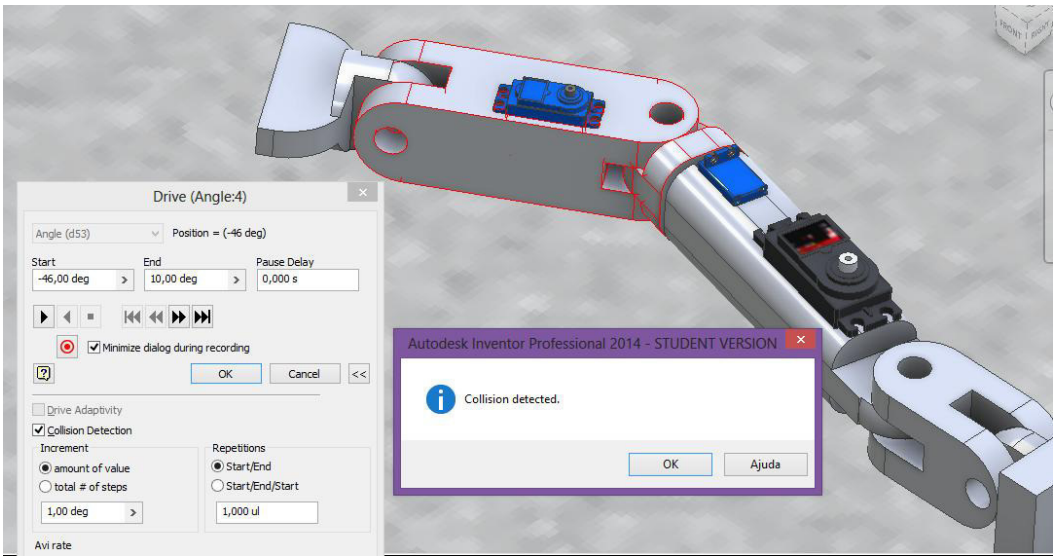
Figura 24 Animação Máximo



Fonte: Própria

Com a opção "*Collission Detection"* ativada, toda vez que a animação entrar em conflito o software será capaz de determinar e avisará ao usuário, ver Figura 25. Sendo assim, com os valores obtidos pelo *software*, pode-se agora finalizar o projeto determinando os valores máximos e mínimos dos servos motores.

Figura 25 Animação em Conflito



Fonte: Própria

4 RESULTADOS

Pretende-se demonstrar o funcionamento do algoritmo operando conforme o apresentado nos itens anteriores. Para isto, este deve apresentar a capacidade de: interagir com dois módulos de maneira distinta, ou seja, um novo comando não deve cancelar o comando anterior, a menos que ambos sejam direcionados para mesmo módulo; distinguir a prioridade dos comandos e não cancelar comandos com prioridade superior; executar demais funções enquanto aciona os servos motores; realizar o passo variável quando houver a necessidade.

Para demonstrar que o algoritmo desenvolvido é capaz de satisfazer esses critérios, utilizar-se-á o código presente na Figura 26, nela consta a matriz ação, gerada de acordo com o item "3.2 Gerando Ações". Nesta há quatro comandos, um do módulo HEAD, uma do módulo LARM, e duas do módulo RARM. Observa-se que ao lado do módulo há o nome da ação, seguida do número de passos e o vetor contendo a posição alvo de cada servo (quando preenchido com '*n*'o algoritmo não irá alterar a posição desse servo).

4.1 Módulos Independentes

O primeiro critério é que dois módulos possam operar de maneira distinta, observe na Figura 26, a matriz ação, ou, o código relativo à quatro comandos utilizados para teste. Para satisfazer esse critério executa-se o comando "*Rise*" do módulo RARM e o comando "*Rise*" do módulo LARM, ambos devem atuar em paralelo.

Figura 26 Comandos Gerados para Simulação

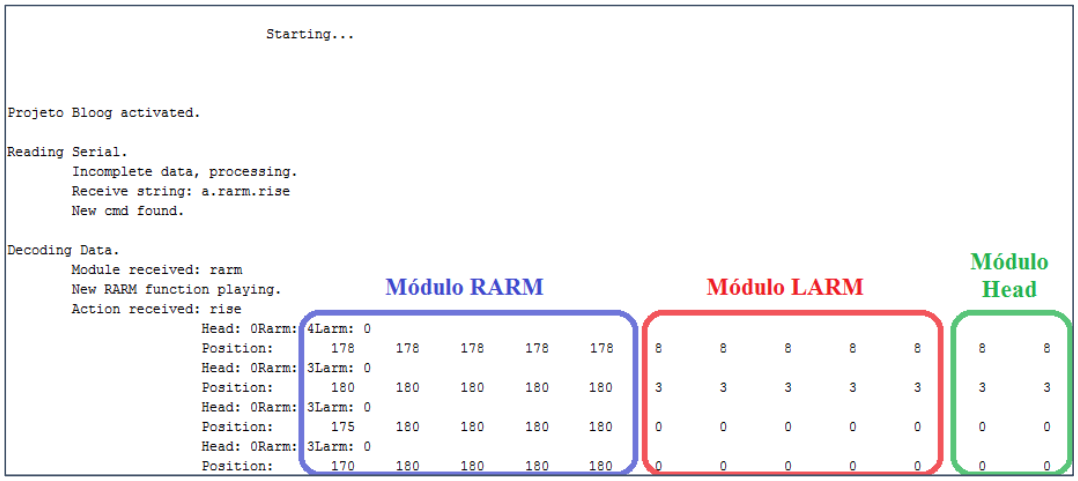
Auto_Actual | Auto_LookUpTable | Auto_MainFunction | Auto_Sensor | Auto_TestFunctions

```
prog_char string_4[] PROGMEM =     "head   $yeas   #3     @n,n,n,n,n,n,n,n,n,n,n";
prog_char string_5[] PROGMEM =     "       $       #2     @n,n,n,n,n,n,n,n,n,n,n";
prog_char string_6[] PROGMEM =     "       $       #1     @n,n,n,n,n,n,n,n,n,n,n";
prog_char string_7[] PROGMEM =     "larm   $rise   #3     @n,n,n,n,n,180,180,180,180,180,n,n";
prog_char string_8[] PROGMEM =     "       $       #2     @n,n,n,n,n,0,0,180,0,0,n,n";
prog_char string_9[] PROGMEM =     "       $       #1     @n,n,n,n,n,30,30,30,30,30,n,n";
prog_char string_10[] PROGMEM =    "rarm   $rise   #4     @180,180,180,180,180,n,n,n,n,n,n,n";
prog_char string_11[] PROGMEM =    "       $       #3     @0,180,180,180,180,n,n,n,n,n,n,n";
prog_char string_12[] PROGMEM =    "       $       #2     @0,0,180,180,180,n,n,n,n,n,n,n";
prog_char string_13[] PROGMEM =    "       $       #1     @0,0,0,180,180,n,n,n,n,n,n,n";
prog_char string_14[] PROGMEM =    "rarm   $wave   #4     @180,90,60,45,30,n,n,n,n,n,n,n";
prog_char string_15[] PROGMEM =    "       $       #3     @0,90,60,45,30,n,n,n,n,n,n,n";
prog_char string_16[] PROGMEM =    "       $       #2     @0,0,60,45,30,n,n,n,n,n,n,n";
prog_char string_17[] PROGMEM =    "       $       #1     @0,0,0,90,90,n,n,n,n,n,n,n";
```

Fonte: Própria

A Figura 27, e a Figura 28 são trechos retirados da resposta do algoritmo ao comando "*a.rarm.rise*" e ao comando "*a.larm.rise*". Pode-se ver na Figura 27, que o primeiro comando a ser executado é o "*a.rarm.rise*", ou seja, o comando "*Rise"* do módulo RARM com a prioridade "a" (a classificação das prioridades será demonstrado no item seguinte), que é máxima. Conforme o previsto na Figura 26, o algoritmo força o primeiro servo do módulo RARM ao ângulo de 0º, enquanto os demais se mantêm no ângulo de 180º. Deve-se ressaltar que o módulo LARM não está sendo executado, logo todos os seus servos mantêm a posição padrão, em outras palavras, eles não se alterarão ainda.

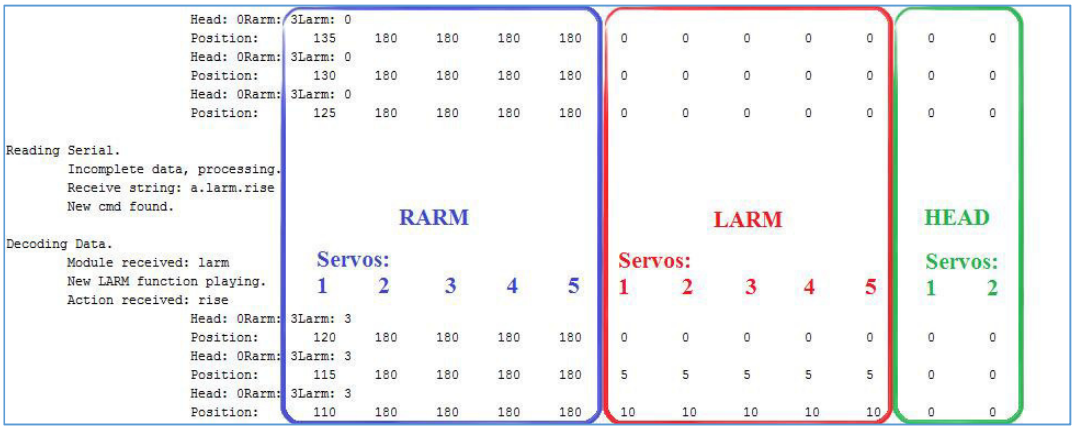
Figura 27 Ínicio do Feedback



Fonte: Própria

Em seguida, envia-se o comando "*a.larm.rise*" que, como visto na Figura 28, não altera o procedimento no módulo RARM. Pode-se observar na parte superior da imagem que o primeiro servo do módulo RARM está decrescendo em 5º graus por iteração e que os servos referentes ao módulo LARM não se alteram, pois ainda não foram acionados. Em seguida, há o *feedback* do algoritmo indicando que um novo comando foi recebido, sendo que, este está direcionado ao módulo LARM (o comando é o "*a.larm.rise*", em outras palavras, o comando "*Rise"* do módulo LARM com a prioridade "a"). Por fim, na parte inferior da figura, observa-se que o modulo RARM continua seu procedimento normalmente, enquanto o módulo LARM inicia seu primeiro passo, como sugere a Figura 26.

Figura 28 Acionamento com dois módulo distintos



Fonte: Própria

Como os dois módulos conseguem interagir com o processador sem afetar a atuação individual, pode-se concluir que este critério foi alcançado com sucesso. Neste mesmo exemplos pode-se observar que os módulos tem prioridade independentes, pois a ação com prioridade máxima do módulo LARM não sobrescreve a ação do módulo RARM apresentada.

4.2 Prioridade de Comandos

O segundo critério está relacionado à determinar a prioridade de comandos. Ao ser programada, uma ação não possui uma prioridade intrínseca, somente ao ser executada o usuário definirá a prioridade desta ação, sendo assim, cabe ao usuário não emitir ações com prioridade superior às prioridades das ações que busquem a estabilidade do robô. Para exemplificar o conceito de prioridade da ação, emite-se o comando "*a.rarm.rise*" e em seguida o mesmo comando com uma prioridade inferior "*c.rarm.rise*" (lembrando que o primeiro caractere representa a prioridade do comando, onde '*a*' é a mais alta e '*z*' a menor). Espera-se que a segunda ação não seja executado, uma vez que ela não possui uma prioridade superior à anterior.

Nota-se, pela Figura 29, que o processador recebe o novo comando, porém o identifica como um comando de prioridade inferior, sendo assim, não o executa. Pode-se observar, também, que após realizar esta verificação, o processador continua o acionamento dos servos.

Figura 29 Prioridade de Comandos

```
                            Starting...



Projeto Bloog activated.

Reading Serial.
        Incomplete data, processing.
        Receive string: a.rarm.rise
        New cmd found.

Decoding Data.
        Module received: rarm
        New RARM function playing.
        Action received: rise
                        Head: 0Rarm: 4Larm: 0
                        Position:       178     178     178     178     178     8       8       8       8       8       8       8
                        Head: 0Rarm: 3Larm: 0
                        Position:       180     180     180     180     180     3       3       3       3       3       3       3
                        Head: 0Rarm: 3Larm: 0
                        Position:       175     180     180     180     180     0       0       0       0       0       0       0

Reading Serial.
        Incomplete data, processing.
        Receive string: c.rarm.rise
        New cmd found.

Decoding Data.
        Module received: rarm
        There is a high priority function playing.
                        Head: 0Rarm: 3Larm: 0
                        Position:       170     180     180     180     180     0       0       0       0       0       0       0
                        Head: 0Rarm: 3Larm: 0
                        Position:       165     180     180     180     180     0       0       0       0       0       0       0
```

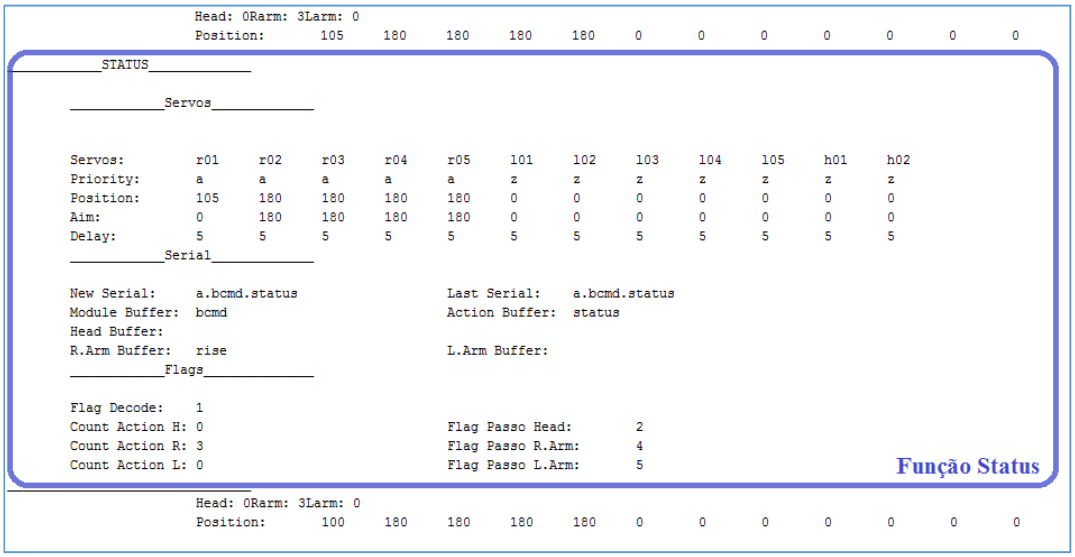
Fonte: Própria

Esta função é importante, pois garante que comandos de ação emitidos pelo usuário não sobrescrevam os comandos de reação do controle que busquem a estabilidade. A ação emitida com prioridade inferior é cancelada, ela não será armazenada afim de ser executada posteriormente.

4.3 Acionamento e demais funções em paralelo

O próximo critério faz referência a capacidade do processador interagir com outras funções (sensores, estabilidade, entre outros) sem comprometer o acionamento. Como visto na Figura 30, opta-se por chamar a função "*Status*" em meio ao acionamento. A função de acionamento a ser executada é a "*a.rarm.rise*" (a mesma apresentada nos itens anteriores), chama-se então a função que atuará em paralelo, no caso, a função "*Status*" (qualquer outra função pode ser chamada a qualquer momento, seja ela de sensores, lógica ou estabilidade).

Figura 30 Chamando outras funções



Fonte: Própria

Visivelmente, a função "*a.rarm.rise*" não é alterada, pela chamada da função "*Status*", e segue decrescendo o ângulo do primeiro servo até 0º graus. A função "*Status*" permite visualizar o acionamento por completo, pois apresenta a posição alvo dos servos, assim como a posição atual, o *delay* inerente a cada passo e a prioridade individual do movimento dos servos.

Conclui-se que o algoritmo pode executar outras funções sem comprometer o acionamento, porém, deve-se observar que durante o processo desta nova função, o acionamento não ocorre, deve-se então ter os cuidados presentes no item "3.4 Estimativa do tempo de processo" para que não haja um atraso perceptível ao executar uma função em paralelo com o acionamento dos servos motores.

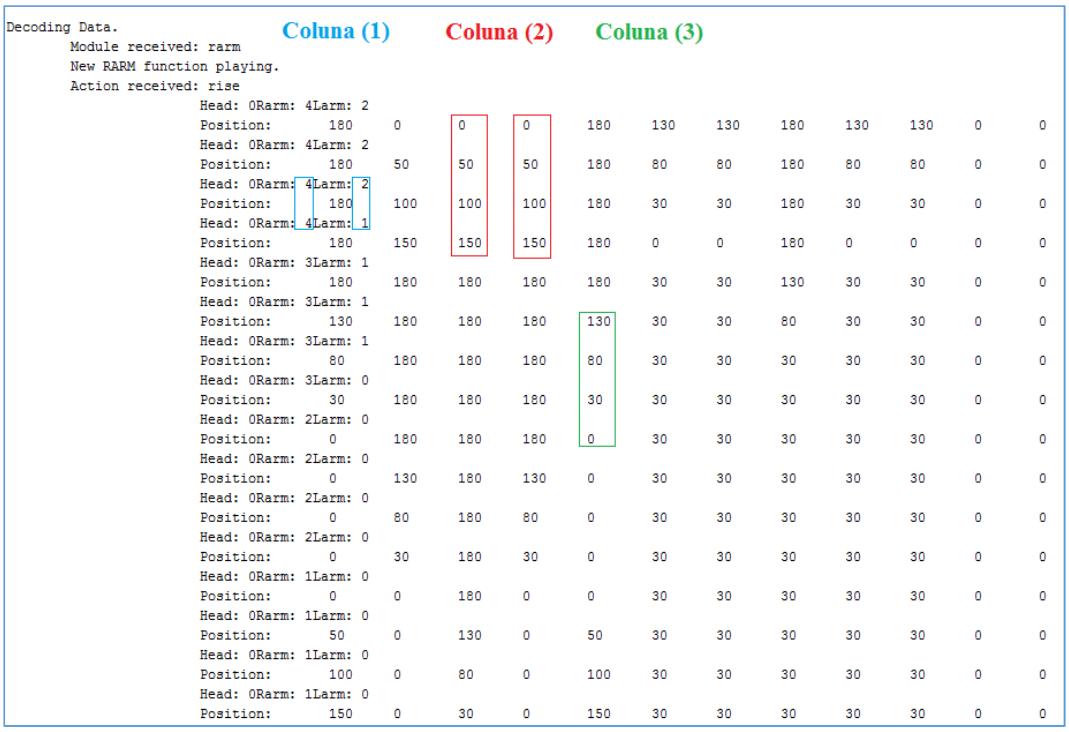
4.4 Passo Variável

Pretende-se, por fim, demonstrar o funcionamento do algoritmo apresentado no capítulo anterior. Sendo assim, na Figura 31, observa-se o *feedback* do algoritmo para um passo de 50º graus. Nela estão sendo executada duas funções, nos módulos RARM (braço direito) e o LARM (braço esquerdo).

A seleção na coluna (1) demonstra que os módulos trabalham de maneira distintas, as alterações do passo no LARM não alteram as ações que ocorrem no RARM.

Na seleção da coluna (2) é visível o passo de 50º graus por iteração, e, na seleção da coluna (3), pode-se notar que o último passo é de 30º graus. Isso ocorre pois o passo padrão de 50º não seria capaz de atingir o ângulo alvo do movimento que é 0º graus.

Figura 31 Feedback do Algoritmo




```
Decoding Data.
        Module received: rarm
        New RARM function playing.
        Action received: rise
                        Head: 0Rarm: 4Larm: 2
                        Position:       180     0       0       0       180     130     130     180     130     130     0       0
                        Head: 0Rarm: 4Larm: 2
                        Position:       180     50      50      50      180     80      80      180     80      80      0       0
                        Head: 0Rarm: 4Larm: 2
                        Position:       180     100     100     100     180     30      30      180     30      30      0       0
                        Head: 0Rarm: 4Larm: 1
                        Position:       180     150     150     150     180     0       0       180     0       0       0       0
                        Head: 0Rarm: 3Larm: 1
                        Position:       180     180     180     180     180     30      30      130     30      30      0       0
                        Head: 0Rarm: 3Larm: 1
                        Position:       130     180     180     180     130     30      30      80      30      30      0       0
                        Head: 0Rarm: 3Larm: 1
                        Position:       80      180     180     180     80      30      30      30      30      30      0       0
                        Head: 0Rarm: 3Larm: 0
                        Position:       30      180     180     180     30      30      30      30      30      30      0       0
                        Head: 0Rarm: 2Larm: 0
                        Position:       0       180     180     180     0       30      30      30      30      30      0       0
                        Head: 0Rarm: 2Larm: 0
                        Position:       0       130     180     130     0       30      30      30      30      30      0       0
                        Head: 0Rarm: 2Larm: 0
                        Position:       0       80      180     80      0       30      30      30      30      30      0       0
                        Head: 0Rarm: 2Larm: 0
                        Position:       0       30      180     30      0       30      30      30      30      30      0       0
                        Head: 0Rarm: 1Larm: 0
                        Position:       0       0       180     0       0       30      30      30      30      30      0       0
                        Head: 0Rarm: 1Larm: 0
                        Position:       50      0       130     0       50      30      30      30      30      30      0       0
                        Head: 0Rarm: 1Larm: 0
                        Position:       100     0       80      0       100     30      30      30      30      30      0       0
                        Head: 0Rarm: 1Larm: 0
                        Position:       150     0       30      0       150     30      30      30      30      30      0       0
```

Fonte: Própria

Em suma, os resultados foram esperados e, apesar do passo mínimo ser grande e de 18º graus, a utilização do passo variável garante o funcionamento correto do algoritmo. Na prática, os atrasos só serão perceptíveis quando o algoritmo estiver repleto de procedimentos com sensores, pois se utilizado unicamente para controle dos servos motores, o algoritmo é completamente viável e eficaz.

5 CONCLUSÃO

Tendo em vista que a proposta era criar um algoritmo capaz de controlar doze servo motores e ser capaz de realizar procedimentos com sensores relacionados aos movimentos dos servos motores, pode-se concluir que o algoritmo é capacitado para solucionar o problema, pois traz uma solução básica, e, por isto, rápida dentre as possíveis soluções.

O algoritmo foi capaz de controlar a posição com a devida precisão e controlar relativamente a velocidade, já que os atrasos que determinam a velocidade são dependentes do processamento e este oscila em cada iteração. Apesar do processamento dedicado possuir um controle de velocidade consideravelmente melhor, o algoritmo para controle de vários servos apresentado reduziu consideravelmente o número de *hardwares* do sistema embarcado, o que, na situação apresentada, reduziu o custo por servo. Havendo a possibilidade, ao se utilizar um processador de alta capacidade, do acionamento possuir uma precisão de 1º grau (a máxima permitida pelo servo apresentado) para cada iteração, permitindo um controle mais aprimorado da velocidade.

Outra objetivo era que o algoritmo fosse genérico e permitisse a alteração, incremento à vontade do usuário, o que é obtido pois o algoritmo permite que a planilha altere facilmente as ações definidas pelo usuário.

Em resumo, o algoritmo pode ser usado não só em robôs mas em qualquer outro sistema embarcado que utilize múltiplos servos motores, garantindo movimentos com precisão e permitindo ao usuário acrescentar qualquer lógica com sensores. O robô utilizado no projeto funcionou normalmente, acrescentando unicamente drivers para os servos motores, já que o a fonte que alimenta o Arduino é limitada. O projeto é viável e pode ser facilmente reproduzido para desempenhar outras atividades.

6 REFERÊNCIAS

VAN DE STRAETE, H.J.; DEGEZELLE, P.; DE SCHUTTER, J.; Belmans, R. J M. **Servo motor selection criterion for mechatronic applications,** Mechatronics, IEEE/ASME Transactions on , vol.3, no.1, Mar 1998. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=662867&isnumber=14510>

BORTHOLOTTO, Julio C.; BORGES, Marcos Augusto F. **Integração de dispositivos robóticos a sistemas de apoio ao aprendizado utilizando a plataforma Arduino.** Artigo, 2009, UNICAMP, São Paulo. Disponível em: < http://www.ft.unicamp.br/liag/robotica/downloads/a5.pdf > Acesso em: 24 maio 2013.

MCROBERTS, Michael; **Arduino Básico**; (tradução Rafael Zanolii). 1ª ed. São Paulo: Novatec Editora, 2011.

GRUPO DE ROBOTICA da Faculdade de Computação – FACOM, Mato Grosso do Sul. **Apostila**: UFMS, 2012. Disponível em: <http://destacom.ufms.br/mediawiki/images/9/9f/Arduino_Destacom.pdf >. Acesso em: 22 maio. 2013.

Arduino. Disponível em: < http://www.arduino.cc/ >. Acesso em: 24 maio 2013.

Arduino Store. Disponível em: < http://store.arduino.cc/ >. Acesso em: 21 maio 2013.

Datasheet ATMega, ATMEL. Disponível em: < http://www.atmel.com/Images/doc2549.pdf >. Acesso em: 24 maio 2013.

Eng Leandro Alves. Disponível em: < http://engleandroalves.wordpress.com/ >. Acesso em: 24 maio 2013.

FDDRSN. Disponível em: < http://fddrsn.net/pcomp/examples/potentiometers.html >. Acesso em: 24 maio 2013.

TecMundo. Disponível em < http://www.tecmundo.com.br/ >. Acesso em: 24 maio 2013.

FACOM. Disponível em: < http://destacom.ufms.br/mediawiki/images/9/9f/Arduino_Destacom.pdf >. Acesso em: 24 maio 2013.

Manual HITEC RCD: Disponível em: < http://www.robotshop.com/PDF/Servomanual.pdf >. Acesso em: 24 maio 2013.

Datasheet Servo HS-311, HITEC. Disponível em: < http://www.robotshop.com/PDF/HS311.pdf >. Acesso em: 24 maio 2013.

7 ANEXOS

Segue abaixo os principais códigos utilizados para programar a placa de controle, sendo eles separados pelas respectivas funções exercidas no sistema embarcado.

7.1 Funções de Comunicação

As funções de comunicação são responsáveis por receber as informações do usuário quando o sistema embarcado já estiver instalado, também são responsáveis pela interpretação de dados e chamada das funções de Memória.

7.1.1 Função de Leitura

A função de leitura irá receber as *strings* de comando através da serial configurada no início do código na função Setup. Em seguida irá retornar os valores recebidos e comparar se o comando não foi enviado em duplicidade.

```
/*
_____________________________________________________________________________

                2.1.2 Função Leitura
_____________________________________________________________________________
*/
        void sread() {

         /*_____________________Recebendo String_____________________________________*/
         Serial.print("\n\nReading Serial.");
         aux_read = 0;
         aux_count = 0;
         while(true){
          if (Serial.available() > 0) {
            aux_count = 0;
            dado_recebido = Serial.read();
            if (dado_recebido==stopByte || dado_recebido=='\0'){
                                                    // Testa se recebeu stopbyte pela serial.Caso sim parar.
              buf[aux_read]='\0';
              break; }
            buf[aux_read]=dado_recebido;
            aux_read++;
          }  else aux_count++;
          if (aux_count > 2256){                    // Testa se estrapolou a paciência do Processador.
            Serial.print("\n\tIncomplete data, processing.");
            buf[aux_read]='\0';
```

```
            break;  }
          }
          /*_____________________FeedBack da Função_____________________________*/


            Serial.print("\n\tReceive string: ");  // Aqui ele mostra todos os valores armazenados no vetor buf.
            Serial.print(buf);
            aux_read = 0;
            do{
             aux_read++;
             if( buf[aux_read] != save[aux_read] ){
               flag_alteration = true;
               Serial.print("\n\tNew cmd found.");
               return;
             }
            } while (aux_read < 64);
            flag_alteration = false;
            return;
        }
//    _____________________________________________FIM DE FUNÇÃO LEITURA________
```

7.1.2 Função de Interpretação

A função Interpretação definirá o módulo, a função e as prioridades das ações.

```
/*
_____________________________________________________________________________

          2.1.3 Função Interpretação
_____________________________________________________________________________
*/
     */
        void decode (){
          int k;
          char priot;
          Serial.print("\n\nDecoding Data.");

          /*__________________Testa Protocolo de comunicação_____________________*/

          if (buf[1] != '.' || buf[6] != '.'){
            Serial.print("\n\tData wasn't processed.");
            flag_alteration = false;
            return;                          // Interrompe a função caso os dados não sejam os previstos.
          }

          /*__________________Armazena Serial de Comunicação_____________________*/
          k = 0;
```

```
for(k=0;k<=64;k++){
  save[k] = buf[k];
}
//Serial.println(save);
/*________Prioridade da Comunicação_______________*/

priot = buf[0];

/*____Novo Alvo da Comunicação____________________*/

save_module[0]=buf[2];save_module[1]=buf[3];save_module[2]=buf[4];
  save_module[3]=buf[5];save_module[4]='\0';
Serial.print("\n\tModule received: ");Serial.print(save_module);
if (strcmp(save_module,"bcmd") == 0){
  if (prio[0] < priot && prio[1] < priot && prio[2] < priot && prio[3] < priot && prio[4] < priot
  && prio[5] < priot && prio[6] < priot && prio[7] < priot && prio[8] < priot && prio[9] < priot
  && prio[10] < priot && prio[11] < priot && prio[12] < priot){
    Serial.print("\n\tThere is a high priority function playing.");
    flag_alteration = false;return;
  }else {
    Serial.print("\n\tNew Bloog Command function playing.");
  }
} else if (strcmp(save_module,"rarm") == 0){
  if (prio[0]  < priot && prio[1]  < priot && prio[2]  < priot && prio[3]  < priot){
    Serial.print("\n\tThere is a high priority function playing.");
    flag_alteration = false;return;
  }else {
    prio[0] = priot;prio[1] =priot;prio[2]=priot;prio[3]=priot;prio[4]=priot;
    Serial.print("\n\tNew RARM function playing.");
  }
} else if (strcmp(save_module,"larm") == 0){
  if (prio[4]  < priot && prio[5]  < priot && prio[6]  < priot && prio[7]  < priot){
    Serial.print("\n\tThere is a high priority function playing.");
    flag_alteration = false;return;
  }else {
    prio[5] = priot; prio[6] = priot; prio[7] = priot; prio[8] = priot; prio[9] = priot;
    Serial.print("\n\tNew LARM function playing.");
  }
} else if (strcmp(save_module,"head") == 0){
  if (prio[8]  < priot && prio[9]  < priot && prio[10]  < priot ){
    Serial.print("\n\tThere is a high priority function playing.");
    flag_alteration = false;return;
  }else {
    prio[10] = priot; prio[11] = priot;
    Serial.print("\n\tNew HEAD function playing.");
  }
} else {
  Serial.print("\n\tModule not found.");flag_alteration = false;return;
}
//  Serial.print("\n\tMoteste.");
```

```
    /*_____________________Ações Pré-definidas_____________________*/

    k=7;
    while (buf[k] != '\0'){
     save_action[k-7] =  buf[k]; save_action[k-6] = '\0'; k++;
    }
    Serial.print("\n\tAction received: ");Serial.print(save_action);

    // Serial.print("\n\tModule Found: ");Serial.print(save_module);//Serial.print(save_module);
    // Substituir essas configurações pela tabela.
    if (strcmp(save_module,"bcmd") == 0){
                if (strcmp(save_action,"clear") == 0){
                   clean();
                   flag_alteration = false;return;
                }else if (strcmp(save_action,"passo") == 0){
                   Serial.print(search_passo(search_action("rarm","rise")));
                   flag_alteration = false;return;
                }else if (strcmp(save_action,"status") == 0){
                   Status();
                   flag_alteration = false;return;
                }else Serial.print("\n\tAction not found.");flag_alteration = false;return;

    }else {
     if (search_action(save_module,save_action) != -1){
      search_aim(search_action(save_module,save_action));
      if (strcmp(save_module,"head") == 0) {
       strcpy(save_actionh,save_action); passo_actionh =
                      search_passo(search_action(save_module,save_action));}
      if (strcmp(save_module,"rarm") == 0) {
       strcpy(save_actionr,save_action); passo_actionr =
                      search_passo(search_action(save_module,save_action));}
      if (strcmp(save_module,"larm") == 0) {
       strcpy(save_actionl,save_action); passo_actionl =
                      search_passo(search_action(save_module,save_action));}
      flag_alteration = false;return;
     }else Serial.print("\n\tAction not found");flag_alteration = false;return;
    }
    return;
   }
//  _____________________________FIM DE FUNÇÃO INTERPRETAÇÃO________
```

7.2 Funções de Execução

As funções de execução são responsáveis por comandar o sistema embarcado uma vez que ele já está instalado, também verificam se a ação foi concluída para chamar o próximo passo através das funções de Memória.

7.2.1 Função de Acionamento

A função de acionamento recolhe os dados interpretados e busca na memória a posição almejada de cada servo. A cada iteração do processador ele posiciona os servos mais próximos do alvo de acordo o *delay* pré-programado.

```
/*
_____________________________________________________________________________

        2.1.4 Função Acionamento
_____________________________________________________________________________
*/

    void action(){
     flag_parir = 0;flag_paril = 0;flag_parih = 0;

     /*__________Atualizando Posições_____________________________*/

     atual[0] = servo_r01.read();atual[1] = servo_r02.read();atual[2] = servo_r03.read();
         atual[3] = servo_r04.read();atual[4] = servo_r05.read();atual[5] = servo_l01.read();
     atual[6] = servo_l02.read();atual[7] = servo_l03.read();atual[8] = servo_l04.read();
         atual[9] = servo_l05.read();atual[10] = servo_h01.read();atual[11] = servo_h02.read();

     /*_____________________Alterando Posições_____________________*/

     if (subaction(servo_r01, 0)== false) flag_parir++;  //subaction(servo_r01, 0);
     if (subaction(servo_r02, 1)== false) flag_parir++;  //subaction(servo_r02, 1);
     if (subaction(servo_r03, 2)== false) flag_parir++;  //subaction(servo_r03, 2);
     if (subaction(servo_r04, 3)== false) flag_parir++;  //subaction(servo_r04, 3);
     if (subaction(servo_r05, 4)== false) flag_parir++;  //subaction(servo_r05, 4);
     if (subaction(servo_l01, 5)== false) flag_paril++;  //subaction(servo_l01, 5);
     if (subaction(servo_l02, 6)== false) flag_paril++;  //subaction(servo_l02, 6);
     if (subaction(servo_l03, 7)== false) flag_paril++;  //subaction(servo_l03, 7);
     if (subaction(servo_l04, 8)== false) flag_paril++;  //subaction(servo_l04, 8);
     if (subaction(servo_l05, 9)== false) flag_paril++;  //subaction(servo_l05, 9);
     if (subaction(servo_h01, 10)== false) flag_parih++;  //subaction(servo_h01, 10);
```

```
 if (subaction(servo_h02, 11)== false) flag_parih++;  //subaction(servo_h02, 11);
 return;
}

/*_____________________Função Sub.Acionamento_____________________*/

boolean subaction (Servo servo, int ray){
 if (abs(aim[ray] - atual[ray]) >= sdel[ray] ){
  if (aim[ray] > atual[ray]){  servo.write(atual[ray]+sdel[ray]); return true;}
  if (aim[ray] < atual[ray]){ servo.write(atual[ray]-sdel[ray]); return true;}
 }
 return false;
}

//  ____________________________________FIM DE FUNÇÃO ACIONAMENTO___
```

7.2.2 Função de Acompanhamento

A função de acompanhamento verifica se o servos atingiram a posição almejado, caso sim, ele busca da memória os próximos passos inerente a função.

```
/*
_____________________________________________________________________________

      2.1.4 Função PariPassu
_____________________________________________________________________________
*/

void pari(){
 int k = 0;
 if (flag_parir > 4){
   if (search_action("rarm",save_actionr) != -1){
    if (passo_actionr <= 1){
     prio[0] = 'z';prio[1] = 'z';prio[2] = 'z';prio[3] = 'z';prio[4] = 'z';passo_actionr = 0;
    }else{
     passo_actionr--;
     search_aim(search_action("rarm",save_actionr)+
                search_passo(search_action("rarm",save_actionr)) - passo_actionr);}
   } else Serial.print("Action cannot be found anymore.");}
 if (flag_paril > 4){
   if (search_action("larm",save_actionl) != -1){
    if (passo_actionl <= 1) {
     prio[5] = 'z';prio[6] = 'z';prio[7] = 'z';prio[8] = 'z';prio[9] = 'z';passo_actionl = 0;
    }else{
```

```
            passo_actionl--;
            search_aim(search_action("larm",save_actionl)+
                        search_passo(search_action("larm",save_actionl)) - passo_actionl);}
        } else Serial.print("Action cannot be found anymore.");}
    if (flag_parih > 1){
        if (search_action("head",save_actionh) != -1){
          if (passo_actionh <= 1) {
            prio[10] = 'z';prio[11] = 'z';passo_actionh = 0;return;
          }else{
            passo_actionr--;
            search_aim(search_action("head",save_actionh)+
            search_passo(search_action("head",save_actionh)) - passo_actionh);}
        } else Serial.print("Action cannot be found anymore.");}
        return;
    }


//_____________________________________________________FIM DE FUNÇÃO PARIPASSU___
```

7.3 Funções Básicas

Algumas funções básicas são necessárias para interpretar melhor o que ocorre dentro do processador, assim como iniciar resposta automaticamente dos estímulos gerados por sensores, ou interrupções.

7.3.1 Função Status

A função Status apresenta todas as variáveis utilizadas no processo, de modo que o operador saiba o que ocorre no processador em tempo real. Através dela é possível verificar as *flags* de operação dos módulos, contadores de passos, valores presentes na serial, posição dos servos, posição alvo dos servos, *delay* da operação, entre outros.

```
/*
_____________________________________________________________________________

        2.1.1 Função Status
_____________________________________________________________________________
*/

    void Status(){
      Serial.print("\n\n_____________STATUS_____________\n");
      Serial.print("\n\t_____________Servos_____________\n");
```

```
Serial.print("\n\n\tServos:\t\tr01\tr02\tr03\tr04\tr05\tl01\tl02\tl03\tl04\tl05\th01\th02");
Serial.print("\n\tPriority:");Serial.print("\t");Serial.print(prio[0]);Serial.print("\t");
 Serial.print(prio[1]);Serial.print("\t");Serial.print(prio[2]);
 Serial.print("\t");Serial.print(prio[3]);Serial.print("\t");Serial.print(prio[4]);Serial.print("\t");
 Serial.print(prio[5]);Serial.print("\t");Serial.print(prio[6]);
 Serial.print("\t");Serial.print(prio[7]);Serial.print("\t");Serial.print(prio[8]);Serial.print("\t");
 Serial.print(prio[9]);Serial.print("\t");Serial.print(prio[10]);Serial.print("\t");Serial.print(prio[11]);
 Serial.print("\n\tPosition:");Serial.print("\t");Serial.print(atual[0]);Serial.print("\t");
 Serial.print(atual[1]);Serial.print("\t");Serial.print(atual[2]);
 Serial.print("\t");Serial.print(atual[3]);Serial.print("\t");Serial.print(atual[4]);Serial.print("\t");
 Serial.print(atual[5]);Serial.print("\t");Serial.print(atual[6]);
 Serial.print("\t");Serial.print(atual[7]);Serial.print("\t");Serial.print(atual[8]);Serial.print("\t");
 Serial.print(atual[9]);Serial.print("\t");Serial.print(atual[10]);Serial.print("\t");
 Serial.print(atual[11]);
 Serial.print("\n\tAim:\t");Serial.print("\t");Serial.print(aim[0]);Serial.print("\t");Serial.print(aim[1]);Seri
 al.print("\t");Serial.print(aim[2]);
 Serial.print("\t");Serial.print(aim[3]);Serial.print("\t");Serial.print(aim[4]);Serial.print("\t");
 Serial.print(aim[5]);Serial.print("\t");Serial.print(aim[6]);
 Serial.print("\t");Serial.print(aim[7]);Serial.print("\t");Serial.print(aim[8]);Serial.print("\t");
 Serial.print(aim[9]);Serial.print("\t");Serial.print(aim[10]);Serial.print("\t");Serial.print(aim[11]);
 Serial.print("\n\tDelay:\t");Serial.print("\t");Serial.print(sdel[0]);Serial.print("\t");
 Serial.print(sdel[1]);Serial.print("\t");Serial.print(sdel[2]);
 Serial.print("\t");Serial.print(sdel[3]);Serial.print("\t");Serial.print(sdel[4]);Serial.print("\t");
 Serial.print(sdel[5]);Serial.print("\t");Serial.print(sdel[6]);
 Serial.print("\t");Serial.print(sdel[7]);Serial.print("\t");Serial.print(sdel[8]);Serial.print("\t");Serial.print(
 sdel[9]);Serial.print("\t");Serial.print(sdel[10]);Serial.print("\t");Serial.print(sdel[11]);
Serial.print("\n\t_____Serial_____\n");
Serial.print("\n\tNew Serial:\t");Serial.print(buf);
Serial.print("\n\tLast Serial:\t");Serial.print(save);
Serial.print("\n\tModule Buffer:\t");Serial.print(save_module);
Serial.print("\n\tAction Buffer:\t");Serial.print(save_action);
Serial.print("\n\tHead Buffer:\t");Serial.print(save_actionh);
Serial.print("\n\tR.Arm Buffer:\t");Serial.print(save_actionr);
Serial.print("\n\tL.Arm Buffer:\t");Serial.print(save_actionl);
Serial.print("\n\t_____Flags_____\n");
Serial.print("\n\tFlag Decode:\t");Serial.print(flag_alteration);
//Serial.print("\n\tCount Action:\t");Serial.print(passo_actionh+passo_actionl+passo_actionr);
Serial.print("\n\tCount Action H:\t");Serial.print(passo_actionh);
Serial.print("\n\tCount Action R:\t");Serial.print(passo_actionr);
Serial.print("\n\tCount Action L:\t");Serial.print(passo_actionl);
Serial.print("\n\tFlag Passo Head:\t");Serial.print(flag_parih);
Serial.print("\n\tFlag Passo R.Arm:\t");Serial.print(flag_parir);
Serial.print("\n\tFlag Passo L.Arm:\t");Serial.print(flag_paril);
Serial.print("\n_____");
return; }

// _____FIM DE FUNÇÃO STATUS_____
```

7.3.2 Função de Limpeza

A função de limpeza irá apagar os dados gravados nas variáveis, nas *flags* dos sistema, no contadores. Ela também é responsável por direcionar os motores para a posição inicial.

```
/*
_____________________________________________________________________________

        2.1.6 Função Clear All

_____________________________________________________________________________
*/

    void clean(){
      int k=0;
      memset(atual,0, sizeof(atual));
                        // Posição atual dos servos [r01,r02,r03,r04,l01,l02,l03,l04,h01,h02,h03]
      memset(aim,0, sizeof(aim));
                        // Posição alvo dos servos [r01,r02,r03,r04,l01,l02,l03,l04,h01,h02,h03]
      for (k=0;k==sizeof(prio);k++){
        prio[k] = 'z';}
            // Prioridade dos movimentos dos servos [r01,r02,r03,r04,l01,l02,l03,l04,h01,h02,h03]
      memset(sdel,5, sizeof(sdel));
          // Delay inerente de cada servo [r01,r02,r03,r04,l01,l02,l03,l04,h01,h02,h03]
      flag_alteration = false;
          // Determina se a animação está sendo executada pela primeira vez.
      passo_actionr = 0;
          // Determina se a animação está sendo animada.
      passo_actionl = 0;
           // Determina se a animação está sendo animada.
      passo_actionh = 0;
           // Determina se a animação está sendo animada.
      do {
        save[aux_read] = ' ';
        aux_read++;
      } while (aux_read <64);
      aux_count = 0;
                        // Auxiliar [ para FOR ] para saída de loops.
      aux_read = 0;
                       // Auxiliar [ para FOR ] de contagem na função leitura.
      servo_r01.write(0);              servo_r02.write(0);
       servo_r03.write(0);              servo_r04.write(0);
      servo_r05.write(0);              servo_l01.write(0);
       servo_l02.write(0);              servo_l03.write(0);
      servo_l04.write(0);              servo_l05.write(0);
       servo_h01.write(0);              servo_h02.write(0);
      do {
        Serial.read();
```

```
    } while (Serial.available() > 0);
    Serial.flush();
    Serial.print("\nCleared!\n\n\n\n");
    Status();
    return;
  }
// _____FIM DE FUNÇÃO CLEAR ALL_____
```

7.4 Funções de Memória

As funções de memória são responsáveis por recolher dados da memória FLASH do processador. São as funções mais comuns e utilizados pelos demais processos.

7.4.1 Função Busca de Módulo

Busca se o módulo inserido pela serial existe no Banco de Dados do processador.

```
/*
_____

    2.2.0 Função Busca de Módulo - Retorna se o Módulo existe ou não na database.
_____
*/

  boolean search_module(char mod[4]){
    for (int i = 0; i < sizeof(string_table)-11; i++){
     strcpy_P(buffer, (char*)pgm_read_word(&(string_table[i])));
              // Necessary casts and dereferencing, just copy.
     String membuf( buffer );
     if (membuf.startsWith(mod)) return true;
    }
    return false;
  }


// _____FIM DE FUNÇÃO BUSCA MÓDULO__
```

7.4.2 Função Busca de Ação

Identifica na memória a posição da ação a ser executada pelo processador.

```
/*
_____________________________________________________________________________

        2.2.1 Função Busca de Ação - Retorna a posição da Função na DataBase.
_____________________________________________________________________________
*/
        int search_action(char mod[4],char act[32]){
            for (int i = 0; i < sizeof(string_table)-11; i++){
              strcpy_P(buffer, (char*)pgm_read_word(&(string_table[i]))); // Necessary casts and dereferencing,
just copy.
              String membuf( buffer );
              if (membuf.startsWith(mod) && membuf.startsWith(act,6)) return i;
            }
            return -1;
        }
//  _____________________________________________FIM DE FUNÇÃO BUSCA ACTION__
```

7.4.3 Função Busca de Passo

A função Busca de Passo é utilizada para que o programa saiba a quantidade de passos restantes de alguma função, ela é útil para minimizar as buscas por ações.

```
/*
_____________________________________________________________________________

    2.2.2 Função Busca de Passo - Retorna o número de passos restantes da Função na DataBase.
_____________________________________________________________________________
*/          int search_passo(int i){
            char aux[2];
            strcpy_P(buffer, (char*)pgm_read_word(&(string_table[i])));
                                    // Necessary casts and dereferencing, just copy.
            String membuf( buffer );
            for(int k=0;k<=sizeof(buffer);k++){    // Passo Confirmation
              if (buffer[k] == '#') {
                aux[0] = buffer[k+1]; aux[1] = '\0';
                return atoi(aux);
                //Serial.print(atoi(aux)); } }
            return 0;       }
  //  _____________________________________________FIM DE FUNÇÃO BUSCA PASSO___
```

7.4.4 Função Alterar Alvo

Busca na memória pela ação, e o passo, a fim de retorna os novos valores almejados para o novo movimento dos servos.

```
/*
_____________________________________________________________________________

        2.2.3 Função Alterar Alvo
_____________________________________________________________________________
*/
        void search_aim(int i){
          int y = 0;
          int k;
          char nun[4];
          k = 0;
          strcpy_P(buffer, (char*)pgm_read_word(&(string_table[i])));
          String membuf( buffer );
          do{
            //if (buffer[k] == '@') Serial.print(" ACHOU O @!!! ");
            if (buffer[k] == '@') break;
            if(k == sizeof(buffer)) return;
            k++;
          }while(k <= 64);
          //Serial.print("\nAlvo dos servos: \n");
          //Serial.print(sizeof(buffer));
          k++;
          int ser = 0;
          do {
            if (buffer[k] == 'n'){
              y = 0;k++; k++; ser++;
            }else if (buffer[k] == ','){
              nun[y] = '\0';
              aim[ser] = atoi(nun);
              ser++;
              y = 0; k++;
            }else {
              nun[y] = buffer[k];
              y++;
              k++;
            }
          }while( k < sizeof(buffer)); //64);//
          return;
        }
//  _____________________________________FIM DE FUNÇÃO BUSCA MÓDULO__
```

7.5 Tabelas dos Arduinos

Tabela 2 Especificações Arduino Uno

Microcontrolador	ATmega328
Tensão de Operação	5V
Tensão de Entrada	7-12V
Tensão Limite de Entrada	6-20V
Pinos I/O Digitais	14 (sendo 6 para PWM)
Pinos de Entrada Analógica	6
Corrente CC por I/O	40 mA
Corrente CC para 3.3V	50 mA
Memória Flash	32 KB (ATmega328) 0.5 KB usados para bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Frequência de Operação	16 MHz

Tabela 3 Especificações Arduino Micro

Microcontrolador	ATmega32u4
Tensão de Operação	5V
Tensão de Entrada	7-12V
Tensão Limite de Entrada	6-20V
Pinos I/O Digitais	20 (sendo 7 para PWM)
Pinos de Entrada Analógica	12
Corrente CC por I/O	40 mA
Corrente CC para 3.3V	50 mA
Memória Flash	32 KB (ATmega32u4) 4 KB usados para bootloader
SRAM	2.5 KB (ATmega32u4)
EEPROM	1 KB (ATmega32u4)
Frequência de Operação	16 MHz

Tabela 4 Especificações Arduino Mega

Microcontrolador	ATmega2560
Tensão de Operação	5V
Tensão de Entrada	7-12V
Tensão Limite de Entrada	6-20V
Pinos I/O Digitais	54 (sendo 14 para PWM)
Pinos de Entrada Analógica	16
Corrente CC por I/O	40 mA
Corrente CC para 3.3V	50 mA
Memória Flash	256 KB (ATmega2560) 8 KB usados para bootloader
SRAM	8 KB (ATmega2560)
EEPROM	4 KB (ATmega2560)
Frequência de Operação	16 MHz