****

# DESCRIÇÃO FUNCIONAL DO ALGORITMO DE CONTROLE E RASTREIO DO ARDUINO

Arthur Bezerra Dantas Saraiva (UFRN, Bolsista PIBITI/CNPq)  
E-mail: arthur.saraiva@crn.inpe.br

Dr. José Marcelo Lima Duarte (INPE, Orientador)  
E-mail: jmarcelo@crn.inpe.br

Abril, 2017

# Agradecimentos

Agradeço a todas as pessoas que me ajudaram neste trabalho, direta ou indiretamente.

Ao Instituto Nacional de Pesquisas Espaciais (INPE) – Pela oportunidade de estudos e utilização de suas instalações.

Ao meu orientador Dr. José Marcelo Lima Duarte pela disponibilidade constante para tirar duvidas e pelo apoio durante o trabalho.

A minha família por sempre me dar apoio e incentivo ao estudo.

Aos professores do curso de Engenharia elétrica da Universidade Federal do Rio Grande do Norte, por todo o conhecimento passado.

Ao Engenheiro Lúcio dos Santos Jotha pela participação e intensa contribuição para o desenvolvimento do trabalho.

# Resumo

Este documento tem por objetivo descrever a estrutura e o funcionamento do algoritmo de controle e rastreio do arduino, que funciona em conjunto com o algoritmo de controle e rastreio do PC, o qual está descrito em [4]. A placa utilizada foi a do arduino Mega, principalmente pela quantidade de portas digitais, além da tensão de operação em 5V. Ao final desse documento será anexada uma tabela com a configuração de todos os pinos atualmente utilizados do arduino. Para maiores detalhes sobre a estrutura física dos módulos que compõe o sistema de controle e rastreio, ler [1].

# Sumário

[1 - Estrutura do algoritmo de controle 4](#_Toc479001427)

[2 - ALGORITMO DE COMUNICAÇÃO SERIAL 6](#_Toc479001428)

[a) Protocolo de comunicação entre o arduino e o PC 6](#_Toc479001429)

[I. Comando SET 6](#_Toc479001430)

[II. Comando STATE 7](#_Toc479001431)

[III. Byte C 7](#_Toc479001432)

[IV. VERIFICADOR DE ERRO DE BITS – CHECKSUM 8](#_Toc479001433)

[V. COMANDO POWER 9](#_Toc479001434)

[VI. COMANDO RESPOSTA – ACKNOWLEDGED E NOT ACKNOWLEDGED 9](#_Toc479001435)

[b) Descrição do algoritmo que implementa o protocolo de comunicação – lado do arduino 10](#_Toc479001436)

[2 - Teste de leds 12](#_Toc479001437)

[3 - Leitura da posição da antena 12](#_Toc479001438)

[4 - Algoritmo de segurança 14](#_Toc479001439)

[5 - Controle manual 14](#_Toc479001440)

[6 - Condição para rastreio 16](#_Toc479001441)

[7 - Calculo do sinal de controle 17](#_Toc479001442)

[8 – Considerações finais 19](#_Toc479001443)

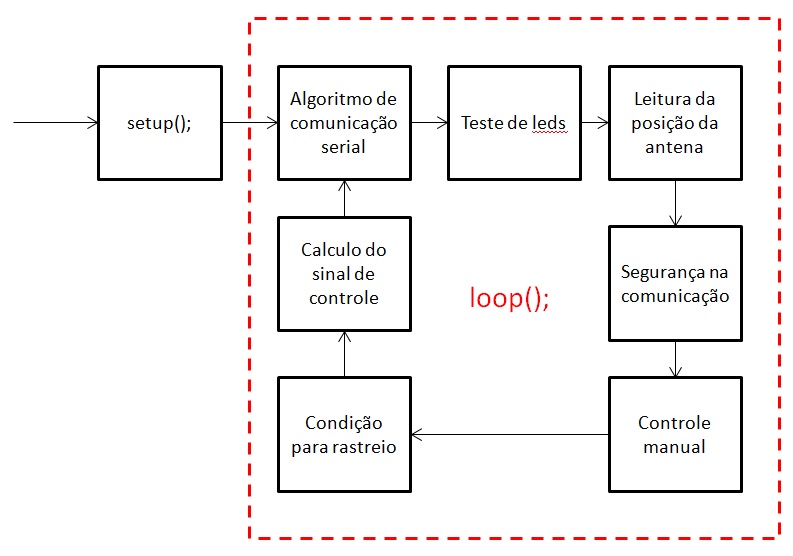
[REFERÊNCIAS 20](#_Toc479001444)

[ANEXO 1 – CONFIGURAÇÃO DOS PINOS UTILIZADOS DO ARDUINO 21](#_Toc479001445)

[ANEXO 2 – CÓDIGO DESENVOLVIDO PARA O ARDUINO 23](#_Toc479001446)

# 1 - Estrutura do algoritmo de controle

O algoritmo foi estruturado de modo a realizar as várias tarefas necessárias para o funcionamento correto do sistema. O Diagrama 1 ilustra de modo global como funciona esse programa a partir do instante que o arduino é ligado.

**Diagrama 1 – Visão macro do código do arduino.**

O código geral do arduino é dividido em duas funções: a função *setup* e a função *loop.* A declaração de todas as variáveis e funções a serem utilizadas neste projeto foi feita antes da função setup. Essa função só é executada uma vez, no momento em que o arduino é inicializado. Ela geralmente é utilizada para configurar quais recursos do arduino serão utilizados, como: portas digitais, analógicas, porta serial, entre outras coisas. Neste projeto, na função *setup* é feita configuração da porta serial para uma taxa de 9600 bps, configuradas todas as portas digitais que são utilizadas e definido o valor estado inicial de algumas delas, como: porta digital que controla o ligamento da potência do sistema em “desligado”; todos os leds e displays iniciam apagados; e sinal de controle para os *drivers* inicial é 0V. Uma tabela com a configuração de todos os pinos utilizados está em anexo a este arquivo.

Após a função setup ser executada, a função *loop* é executada. A principal característica dessa função é que ela é executada infinitamente enquanto o arduino estiver ligado. Caso ele venha a ser desligado e ligado, o ultimo código enviado a ele é executado, primeiro executando a função *setup* e o que estiver declarado antes dela, e após a função *loop*. No Diagrama 1, a função loop é representada pela linha vermelha tracejada, mostrando todas as tarefas que são executadas nela. O código desse projeto como um todo foi elaborado tendo em mente essa funcionalidade da função loop de se repetir infinitamente.

A seguir será detalhada cada subtarefa que é executada na função *loop*.

# 2 - ALGORITMO DE COMUNICAÇÃO SERIAL

Para realizar a comunicação serial entre o PC e o arduíno foi elaborado um protocolo de comunicação. Esse protocolo será detalhado a seguir, e após será explicado funcionalmente o algoritmo desenvolvido para realizar esse protocolo, no lado de comunicação do arduino.

# Protocolo de comunicação entre o arduino e o PC

A comunicação entre o PC e o Arduino é feita via porta USB, a uma taxa de 9600 bps, com 1 bit de parada e sem paridade. O sistema contem três comandos: SET, STATE, POWER. Além disso, existem três tipos de respostas: ACKNOWLEDGED e NOT ACKNOWLEDGED e a resposta ao comando STATE. As respostas ACKNOWLEDGED E NOT ACKNOWLEDGED serão abreviadas aqui por ACK e NACK, respectivamente. Esses comandos serão descritos nos tópicos seguintes. No comando SET, STATE e resposta ao STATE serão enviados um código para verificação de erro na mensagem, o *checksum*.

## Comando SET

Nesse comando, são enviadas para o arduino as posições de referência para as quais ele deve apontar no instante em que receber o comando. Os ângulos de referência são enviados codificados em 16 bits, ou seja, ocupam dois bytes na *string* de mensagem. A codificação é feita de modo que 0° corresponda a 0 em binário, e 360° corresponda a 65535 (2^16 -1). Segue abaixo o modelo da *string* de mensagem.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S | A1 | A0 | E1 | E0 | CS |
| 0x57 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 |

**Comando SET.**

O primeiro byte é o caractere ASCII 'W', que indica o início da mensagem. Os bytes A1 e A0 representam o ângulo de azimute, sendo A1 o byte com os 8 bits mais significativos (MSB) e A0 o byte com os 8 bits menos significativos (LSB). O mesmo vale para os bytes de elevação E1 e E0. O ultimo byte é o *Checksum*, o código para detecção de erro na mensagem. Segue abaixo exemplo da codificação dos ângulos.

**Ex: ângulo de azimute = 235,43°**

Ângulo codificado = 235,43 \* (65535/360) = 42858,069 ≈ 42858

Ângulo codificado em binário = 1010 0111 0110 1010

A1 = 1010 0111 = 167

A0 = 0110 1010 = 106

## Comando STATE

O comando STATE solicita ao arduino o status atual do sistema. Esse comando é enviado através de um único byte, formado pelo caractere ASCII 'S'. Esse comando não envia *checksum*, porém, a mensagem de resposta a esse comando, do arduino ao PC contém.

A resposta a esse comando é uma string com o status atual do sistema com: posição atual da antena; status dos relés; defeito dos inversores; status do controle manual; e status da potência. Além disso, é incluído o byte de *checksum*. Segue abaixo o modelo da string de resposta.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| W | A1 | A0 | E1 | E0 | C | CS |
| 0x57 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 |

**Resposta ao comando STATE.**

O primeiro byte representa o início de mensagem 'W'. Os ângulos da posição atual da antena são codificados da mesma forma que no comando SET. O byte C contém informações do status atual do sistema. E o ultimo byte será o *checksum*, para verificação de erros na mensagem.

## Byte C

No byte 'C', dos 8 bits que compõe seu byte, são utilizados os 5 menos significativos e o bit mais significativo (MSB). Para cada um dos eventos é enviado o bit 0 caso a situação esteja normalizada e 1 caso algum desses eventos tenha ocorrido. Segue abaixo um modelo da resposta ao comando STATE.

O byte C é codificado da seguinte maneira:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| P | 0 | 0 | KA1 | KA2 | DAZ | DEL | M |

**Formação do byte de status C.**

Dessa forma, o valor numérico do byte C será:

C= P\*2^7 + KA1\*2^4 + KA2\*2^3+ DAZ\*2^2 +DEL\*2^1 + M\*2^0

Onde:  
P --> Potência (0: desligada, 1: ligada);  
KA1 --> Relé KA1 (0: inativo, 1: ativo);  
KA2 --> Relé KA2 (0: inativo, 1: ativo);  
DAZ --> Defeito no Inversor de Azimute (0: inativo, 1: ativo)  
DEL --> Defeito no Inversor de Elevação (0: inativo, 1: ativo)  
M --> Controle Manual (0: inativo, 1: ativo)

## VERIFICADOR DE ERRO DE BITS – CHECKSUM

Um código *checksum* é utilizado para detecção de erro na transmissão. Esse código consiste em efetuar a operação XOR entre todos os bytes da mensagem e anexar o byte resultante no final da mensagem, antes de sua transmissão. Para verificar a integridade da mensagem no receptor, a mesma operação é feita nos bytes recebidos, mas incluindo o byte de *checksum*. Um resultado diferente de 0 nesta operação indica que a mensagem está corrompida, e um resultado igual a 0 indica que ou não há erro ou se ocorreu, o erro é indetectável pelo *checksum*.

**Exemplo de checksum no comando SET:**

Azimute: 115,76° 🡪 Ângulo codificado = 115,76\*65535/360 ≈ 21073

A1 = 0101 0010 = 82

A0 = 0101 0001 = 81

Elevação: 43,23° 🡪 Ângulo codificado = 43,23 \*65535/360 = 7875

E1 = 0001 1110 = 30

E0 = 1100 0011 = 195

Checksum = 87 xor 82 xor 81 xor 30 xor 195

Checksum = 137

Valor numérico da string de comando: {87, 82, 81, 30, 195, 137}

Caso o arduino perceba que o *checksum* da mensagem chegou errado, será enviado um comando NACK ao PC, que será descrito mais a frente.

## COMANDO POWER

O comando POWER será enviado pelo PC para ligar ou desligar a potência do sistema. Esse comando será enviado como um único byte, formado pelo caractere 'P'. A função desse comando será alternar o estado do sistema entre ligado e desligado. Esse comando será enviado a cada inicio e fim de rastreio. Não será enviado checksum.

## COMANDO RESPOSTA – ACKNOWLEDGED E NOT ACKNOWLEDGED

Os comandos ACK e NACK serão codificados por caracteres ASCII 'A'e 'N', respectivamente. Esses comandos serão enviados por ambos os lados, arduino e PC, assim que ambos receberem um comando ou resposta a um comando qualquer. Será enviado um ACK caso um comando recebido por algum dos lados tenha sido entendido corretamente, e um NACK caso não tenha sido entendido. Caso o receptor não tenha entendido a mensagem (ou seja, caso NACK), ele enviará NACK ao transmissor e o transmissor desse comando deverá reenviar a mensagem novamente imediatamente.

**Exemplo:** o PC envia ao arduino o comando STATE, o arduino responde ao PC com a string do comando STATE. O PC verifica que o checksum da mensagem recebida está errado e por isso envia ao arduino o comando NACK. O arduino ao receber o comando NACK refaz e reenvia a string de comando STATE ao PC. O PC verifica que a mensagem veio com o checksum correto, e por isso envia o comando ACK ao arduino. A partir daí a comunicação continua normalmente.

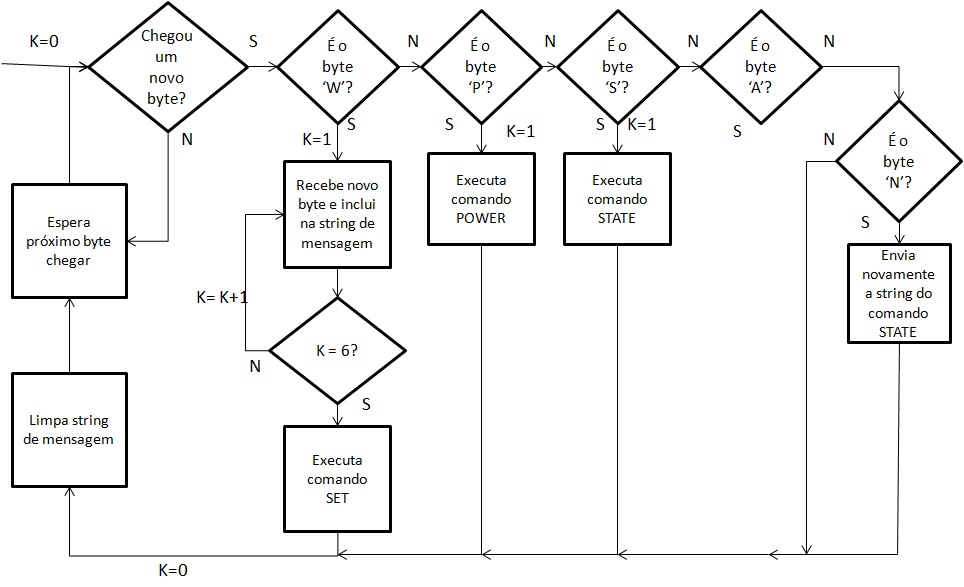
A Tabela1 mostra as possibilidades de envio de comandos para cada plataforma. Um V significa que a referida plataforma pode enviar o comando em questão, um F significa que essa plataforma não pode enviar esse comando, apenas receber.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **COMANDO** | | | | | |
| **PLATAFORMA** | **SET** | **STATE** | **Resposta ao STATE** | **POWER** | **ACK** | **NACK** |
| **PC** | V | V | F | V | V | V |
| **Arduino** | F | F | V | F | V | V |

**Tabela 1 – Possibilidades de comandos por plataforma.**

## Descrição do algoritmo que implementa o protocolo de comunicação – lado do arduino

Esse protocolo não possui byte de fim de mensagem, por isso o sistema de comunicação no arduino foi feito com base na contagem de quantos bytes chegaram e análise do primeiro byte para definição do comando enviado. Para isso, é utilizada uma variável como contadora. Essa variável é inicializada com valor zero, e sempre que um comando é executado, o valor zero é atribuído novamente a ela. O Fluxograma 1 representa o funcionamento do protocolo no arduino.

**Diagrama 2 – Algortimo que executa o protocolo de comunicação do arduino.**

O programa começa esperando chegar algum byte vindo do PC. Quando chega o primeiro byte da mensagem, é feita a verificação de qual é o byte, pois a partir do primeiro já é possível saber qual o comando está sendo solicitado, ou se não foi um comando (algum byte flutuante).

**Comando SET:** Caso o primeiro byte seja um ‘W’, o arduino irá esperar chegar os 5 bytes restantes do comando SET. Após isso, irá verificar se o checksum está correto, caso esteja correto, será enviado um comando ACK para o PC, e será executado o comando SET normalmente. Caso o checksum não esteja correto, o arduino enviará um NACK para o PC, irá esperar pelo reenvio da string de comando SET. Enquanto o PC mandar mensagens com erros o arduino permanecerá nesse loop de enviar NACK e receber a string até que o checksum da mensagem venha correto.

**Comando POWER:** Caso o primeiro byte da mensagem seja um ‘P’, o arduino irá enviar um ACK para o PC e inverter o status atual da potência do sistema: se estiver ligado irá desliga-la, e se estiver ligado irá desliga-la.

**Comando STATE:** Caso o primeiro byte da mensagem seja um ‘S’, o arduino irá formar a string de STATE, irá enviar para o PC e aguardar a resposta, um ACK ou NACK. Caso receba um NACK, ele ficará em um loop de reenvio da string de STATE, aguardo da resposta do PC enquanto a resposta for NACK. Caso o PC responda ACK, o arduino irá esperar o próximo comando.

**Comando ACK e NACK:** A única possibilidade na qual o arduino receberá esses comandos do PC será quando o arduino enviar a string de STATE. Esses comandos serão tratados como foi descrito no **Comando STATE**.

Quando o primeiro byte de alguma mensagem for um byte diferente dos anteriores, o arduino não executará nenhum comando, e irá desativar a *flag “*endMsg”, a qual quando é verdadeira juntamente com a flag de algum comando (SET, STATE ou POWER), permite a execução do respectivo comando. Isso é feito para garantir que nenhum comando aleatório seja executado no momento indevido.

# 2 - Teste de leds

Após o algoritmo de comunicação serial, é executado o algoritmo do teste de leds. Para realizar esse teste, existe um botão no painel frontal da gaveta de pilotage chamado “Test voyants”. Esse botão está conectado a um contato do relé KA4 da gaveta de pilotage, e este contato está conectado a uma porta digital do arduino. O algoritmo do teste de leds consiste em verificar se esse relé foi acionado. Caso seja, o arduino irá ligar todos os leds do painel frontal, até que o botão seja solto. Por questões de segurança, o algoritmo só permite o teste de leds caso o sistema não esteja em rastreio, ou seja, caso a potência do sistema esteja desligada.

# 3 - Leitura da posição da antena

Após o teste de leds, é executada a leitura da posição atual da antena, que é codificada em 16 bits. Para isso, foi elabora a função *readAngle()* que executa a leitura atual marcada pelo resolver da antena (sensor de posição angular). Essa função é descrita a baixo.

* **readAngle():** Esta função faz interface com o conversor analógico para digital dos resolvers. O objetivo principal é fazer a leitura correta da posição atual da antena de azimute e elevação. Como só é possível fazer a leitura de um dos ângulos por vez (primeiro leitura de azimute e depois elevação, por exemplo), essa função recebe como parâmetro um inteiro podendo ser 1 ou 0, o qual é utilizado para diferenciar qual leitura está sendo solicitada. Caso o inteiro recebido seja 0, a leitura feita será da posição atual de azimute da antena. Caso seja 1, a leitura será da posição de elevação. Feita a leitura e decodificação dos 16 bits do resolver, o ângulo resultante é retornado pela função. Desse modo, para realizar a leitura da posição de azimute e elevação, essa função tem que ser executada duas vezes.

É importante deixar claro que a leitura do resolver não está na mesma escala da posição real da antena. A Figura 1 faz o comparativo entre as escalas do resolver e a escala de posição da antena.

**Figura1 – Relação entre escalas do resolver e da antena**

A partir desse comparativo, foram elaboradas as equações 1 e 2, para conversão entre essas escalas.

AZant = ((360 - angResAZ) - 15) \* (96.0 / 88.0); (1)

ELEant = ((360 - angResELE) - 34.0) \* (80.0 / 130.0); (2)

Feita a leitura, decodificação e conversão entre escalas, o ângulo real da posição da antena estará pronto. Após isso esses ângulos são utilizados em várias partos do projeto, como no calculo do sinal de controle (que será descrito a seguir), na comunicação serial, para envio das posições angulares para o PC, e para escrita no display da gaveta de pilotage.

O processo de escrita da gaveta de pilotage é feito através de uma função chamada *display\_gaveta().* Essa função será descrita a baixo.

* **display\_gaveta():** Esta função faz interface com os displays de 7 segmentos da gaveta de pilotage, os quais mostram a posição atual da antena de azimute e elevação. Cada ângulo do display é constituído de 4 displays de 7 segmentos, sendo um para centenas, um para dezenas, um para unidades e um para frações. Cada digito que compõe os displays são escritos na forma de 4 bits. Com isso, a função recebe como parâmetros os ângulos da posição atual da antena e codifica-os em números binários. A partir daí é feito o processo de escrita em cada um dos dígitos de cada display. A função foi implementada de modo que caso um ângulo possuísse valor do dígito de centena igual a zero, esse display é apagado. E caso algum ângulo possua digito de centena e dezena igual a zero, ambos são apagados. Isso foi feito apenas por questões estéticas. Maiores detalhes sobre a configuração dos displays da gaveta de pilotage são descritas em [2].

Por questões estéticas, foi elaborado um trecho de código para permitir a atualização do display da gaveta de pilotage apenas a cada 200 mS. Isso foi feito devido a quando os ângulos variavam a uma taxa muito elevada, alguns displays ficavam apagando.

# 4 - Algoritmo de segurança

Feita a execução da leitura da antena e escrita dos displays de posição, é executado um código bem simples para evitar problemas. Sempre que o arduino entra em rastreio, ou seja, quando ele recebe o primeiro comando POWER, um contador de tempo é atualizado em uma variável. A partir daí, todos os comandos recebidos pelo arduino do PC atualizam esse contador. E constantemente é feita uma verificação: caso esse contador de tempo não tenha sido atualizado a mais que 3 segundos, a potência do sistema será desligada. Isso é feito como medida de segurança para o caso de a comunicação entre o PC e o arduino venha a ser perdida durante um rastreio (caso alguém puxe o cabo USB do PC, por exemplo). Se essa comunicação fosse perdida, a antena iria se mover com velocidade constante em uma direção, pois o arduino só atualiza o sinal de controle quando recebe um comando SET (será descrito mais a frente). Como durante um rastreio o arduino recebe dois comandos no período de 1 segundo (SET e STATE), esse tempo de 3 segundos como segurança foi considerado razoável.

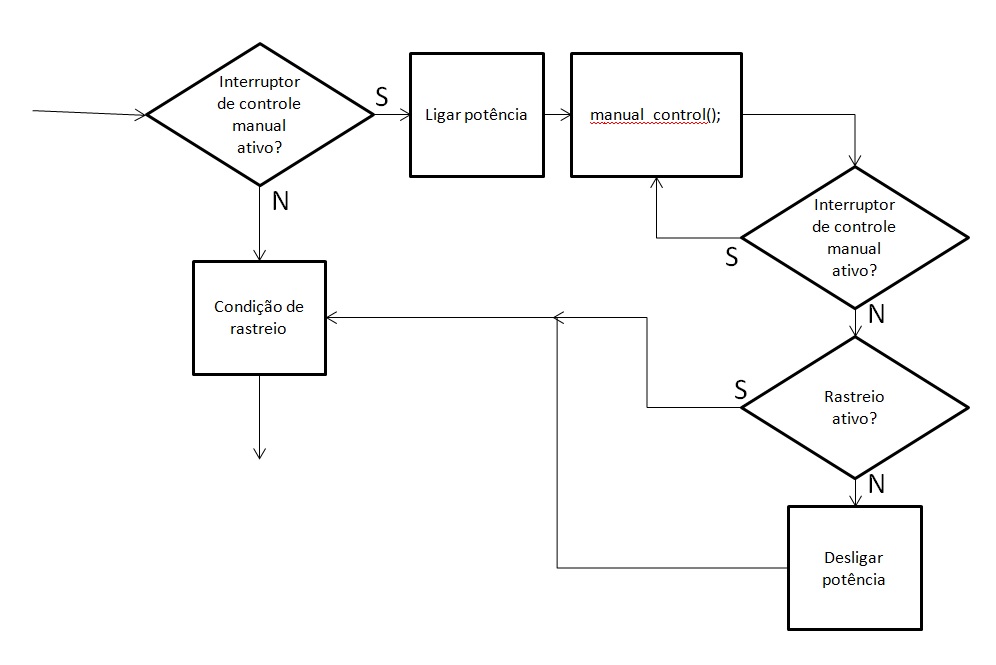
# 5 - Controle manual

Para realizar o controle manual da antena foi implementada uma função específica para isso: a função *manual\_control().*

* **manual\_control():** Esta função faz interface com o joystick da gaveta de pilotage. O objetivo dela é possibilitar o controle manual da antena. O joystick da gaveta de pilotage envia constantemente para entradas analógicas do arduino valores de tensão de acordo com uma escala, descrita em [3]. Duas entradas analógicas são utilizadas, uma para tensões de referência para a posição de azimute, e outra para tensões de referência para posição de elevação. Quando a função é executada, ela faz a leitura desses valores analógicos, decodifica-os e os escreve na saída PWM do arduino.

Essa função é executada ao acionar o interruptor de controle manual no painel frontal da gaveta de pilotage. Caso o interruptor seja acionado durante um rastreio, o sinal de controle para os inversores não será mais o calculado pelo algoritmo PI, e sim os valores lidos nas entradas analógicas do arduino após a decodificação. Ou seja, se o interruptor for acionado durante um rastreio, o controle da posição da antena será feito pelo joystick da gaveta de pilotage (o controle manual tem prioridade sobre o controle PI calculado). Caso o interruptor seja posto na posição desativada ainda dentro de um rastreio, o sinal de controle voltará a ser o calculado pelo algoritmo PI.

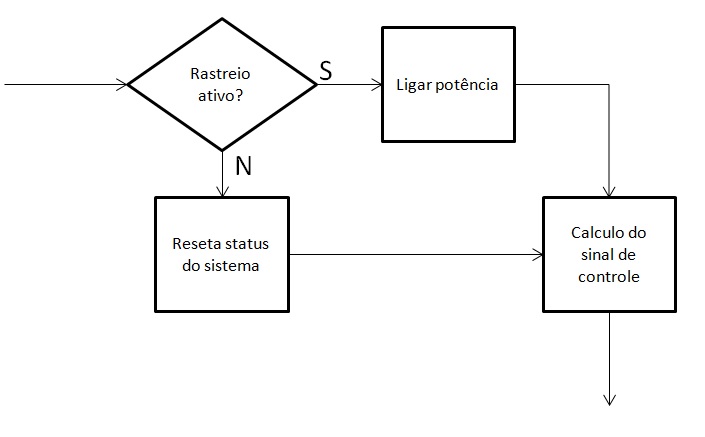
Se o interruptor for acionado fora de rastreio, a potência do sistema será ligada, e o sinal de controle para os inversores será o valor lido nas portas analógicas do arduino. Se o interruptor for solto fora de rastreio a potência do sistema será desligada e a antena permanecerá na posição que ficar. O Diagrama 3 mostra de forma resumida como funciona algoritmo de controle manual.

**Diagrama 3 – Algoritmo de controle manual**

Após a execução do algoritmo de controle, o próximo a ser executado é a verificação da condição de rastreio.

# 6 - Condição para rastreio

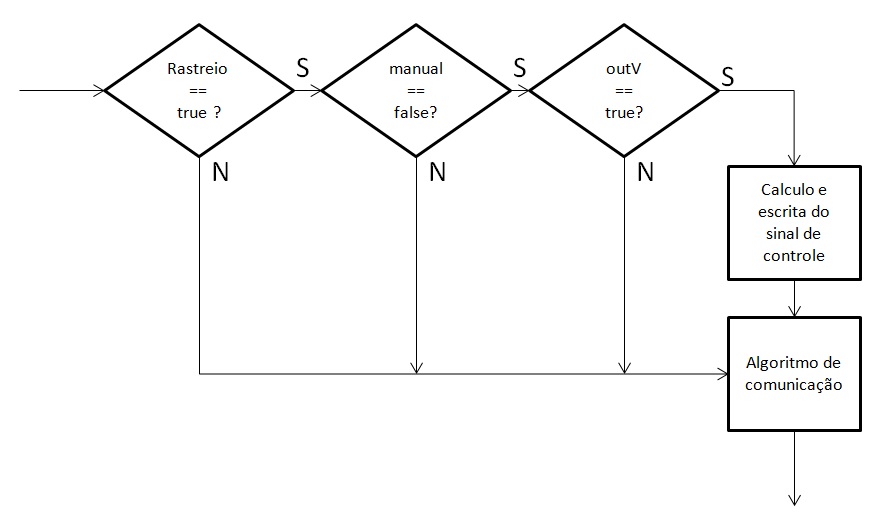
O critério para inicio de rastreio do sistema de controle e rastreio é o comando POWER. Quando o arduino receber um comando POWER, uma flag chamada *rastreio* terá seu valor invertido, como descrito anteriormente. Caso o sistema esteja desligado, a potência será ligada e permanecerá até que o arduino receba outro comando POWER. Quando o arduino receber o primeiro comando POWER do PC, a potência do sistema será ligada, e a cada comando SET que o arduino receber o sinal de controle será atualizado. Nos finais de rastreio, quando o arduino receber um comando POWER para desligar a potência, caso o controle manual não esteja ativo, a potência do sistema será desligada e alguns parâmetros serão resetados: o sinal de controle será zerado; os acumuladores do algoritmo de controle PI serão zerados; e as posições de referência do sistema será atualizada para 180° de azimute e 90° de elevação. Caso o arduino receba um comando POWER para desligar a potência e o controle manual esteja ativo, a potêcia não será desligada, permanecendo sob controle do joystick da gaveta de pilotage. No momento que o interruptor de controle manual for desativado, a potência será desligada e os parâmetros do sistema serão resetados.

 **Diagrama 4 – Condição para rastreio**

Após essa seção do código, é executada a seção de calculo do sinal de controle.

# 7 - Calculo do sinal de controle

Para o calculo do sinal de controle ser atualizado nas saídas PWM do arduino 3 condições são necessárias: a potência do sistema tem que estar ligada, ou seja, o arduino tem que ter recebido um comando POWER; o sistema não pode estar com controle manual ativo (sendo assim, o sinal de controle manual tem prioridade sobre o sinal de controle PI calculado); e o arduino deve ter recebido um comando SET. Sempre que o arduino recebe um comando SET uma *flag* chamada *outV* é atualizada para valor verdadeiro, permitindo a atualização do sinal de controle uma vez. Após o sinal de controle ser atualizado essa *flag* é atualizada para falso, e o sinal de controle só poderá ser atualizado novamente quando ela for verdadeira, ou seja, quando o arduino receber um comando SET (e as outras condições citadas forem satisfeitas). Isso é feito para garantir um sinal de controle a cada 1 segundo, que é o período de amostragem adotado para esse sistema e é o intervalo de tempo no qual o PC envia comandos SET ao arduino. O diagrama 5 ilustra esse funcionamento.

**Diagrama 5 – Calculo do sinal de controle**

O algoritmo de controle PI do sistema foi mantido similar ao sistema de controle anterior. É feito o calculo do erro atual de posição com base na posição de referência atual (recebida no comando SET) e a posição atual da antena lida. A partir desse erro é calculado o sinal de controle.

Está sendo utilizada uma limitação do sinal de controle de -2.5V a +2.5V em hardware e em software. Apesar de não utilizar toda a escala de tensão possível (que varia de -10V a +10V), esse modo de operação apresentou bons resultados durante alguns testes práticos realizados.

A saída de controle está sendo feita pelas saídas PWM do arduino. Como os valores possíveis gerados por essas saídas variam de 0 a 5V, foi necessário a elaboração de uma placa que convertesse 0V de saída do arduino em -2.5V e 5 V do arduino em +2.5V. O placa foi elaborado com a utilização de Ampops e um circuito para filtragem do sinal. A Figura 1 ilustra a comparação de tensões entre a saída do arduino (Voa) e a saída da placa (Vop).

**Figura 1 – Comparativo entre tensão de saída do arduino e da placa conversora**

O sinal de controle é calculado normalmente, e após isso o valor final de saída é alterado de acordo com a equação abaixo.

(1)

(2)

Após essa conversão, o sinal de controle é escrito nas saídas PWM do arduino, e o programa volta a executar a parte de comunicação do algoritmo.

# 8 – Considerações finais

O algoritmo desenvolvido foi testado e funcionou conforme desejado.

Caso alguma nova funcionalidade venha a ser adicionada a esse código, é recomendado não implementar trechos de códigos que possam impedir a execução do algoritmo de controle, pois, já que o arduino recebe comandos constantemente do PC, isso iria estourar o buffer de entrada do arduino, e poderá causar alguma execução indesejada.

# REFERÊNCIAS

[1] QUEIROZ, K. I. P. M. Descrição Funcional do Subsistema de Controle e Rastreio da Estação Multimissão de Natal (EMMN), Instituto Nacional de Pesquisas Espaciais, Natal. 2006.

[2] JOTHA, L. EMMN-DT-030-V01 - Displays de Ângulos de AZ e EL - Tabela de Cabeação

[3] JOTHA, L. EMMN-DT-032-V01 - Tensões do Joystick para o ARDUINO

[4] SARAIVA, A. B. D. Descrição funcional do algoritmo de controle e rastreio do PC, Instituto Nacional de Pesquisas Espaciais, Natal. 2017.

# ANEXO 1 – CONFIGURAÇÃO DOS PINOS UTILIZADOS DO ARDUINO

|  |  |  |
| --- | --- | --- |
| **PINO** | **DESCRIÇÃO** | **CONFIGURAÇÃO** |
| 52 | Bit de posição 00 (LSB) | Entrada Digital |
| 51 | Bit de posição 01 | Entrada Digital |
| 49 | Bit de posição 02 | Entrada Digital |
| 47 | Bit de posição 03 | Entrada Digital |
| 45 | Bit de posição 04 | Entrada Digital |
| 43 | Bit de posição 05 | Entrada Digital |
| 41 | Bit de posição 06 | Entrada Digital |
| 39 | Bit de posição 07 | Entrada Digital |
| 37 | Bit de posição 08 | Entrada Digital |
| 35 | Bit de posição 09 | Entrada Digital |
| 33 | Bit de posição 10 | Entrada Digital |
| 31 | Bit de posição 11 | Entrada Digital |
| 29 | Bit de posição 12 | Entrada Digital |
| 27 | Bit de posição 13 | Entrada Digital |
| 25 | Bit de posição 14 | Entrada Digital |
| 23 | Bit de posição 15 (MSB) | Entrada Digital |
| 12 | Defeito azimute | Entrada Digital (Pull-UP) |
| 11 | Defeito elevação | Entrada Digital (Pull-UP) |
| 10 | Status KA1 | Entrada Digital (Pull-UP) |
| 8 | Status KA2 | Entrada Digital (Pull-UP) |
| 9 | Status KA4 | Entrada Digital (Pull-UP) |
| 22 | Libera leitura azimute | Saída Digital |
| 24 | Libera leitura elevação | Saída Digital |
| 5 | Liga gaveta de potência (*Power*) | Saída Digital |
| 2 | Saída digital para placa conversora DAC | Saída Analógica |
| 3 | Saída digital para placa conversora DAC | Saída Analógica |
| 34 | Habilitação fração AZ display | Saída Digital |
| 38 | Habilitação unidade AZ display | Saída Digital |
| 44 | Habilitação dezena AZ display | Saída Digital |
| 52 | Habilitação centena AZ display | Saída Digital |
| 30 | Habilitação fração ELE display | Saída Digital |
| 36 | Habilitação unidade ELE display | Saída Digital |
| 40 | Habilitação dezena ELE display | Saída Digital |
| 48 | Habilitação centena ELE display | Saída Digital |
| 32 | Bit 00 display (LSB) | Saída Digital |
| 46 | Bit 01 display | Saída Digital |
| 50 | Bit 02 display | Saída Digital |
| 28 | Bit 03 display (MSB) | Saída Digital |
| 6 | LED parada de emergência | Saída Digital |
| 4 | LED Potência | Saída Digital |
| 14 | LED falha inversor | Saída Digital |
| 15 | LED rastreio | Saída Digital |
| 16 | LED final de curso | Saída Digital |
| 17 | LED Manual | Saída Digital |
| 7 | Controle Manual | Entrada Digital (Pull-UP) |
|  |  |  |

# ANEXO 2 – CÓDIGO DESENVOLVIDO PARA O ARDUINO