****

# DESCRIÇÃO FUNCIONAL DO ALGORITMO DE CONTROLE E RASTREIO DO PC

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 PC, que funciona em conjunto com o algoritmo de controle e rastreio do arduino, o qual está descrito em [5]. O software de controle foi desenvolvido utilizando a linguagem C/C++ para rodar no sistema operacional Windows. A comunicação entre o PC e o arduino é feita via porta serial. O software atual ainda está em versão de testes, mas vem apresentando bons resultados em testes práticos.

# Sumário

[1- Estrutura geral do algoritmo de controle e rastreio 5](#_Toc479000032)

[2 - Biblioteca Efem 6](#_Toc479000033)

[a) Atributos 6](#_Toc479000034)

[b) Métodos 6](#_Toc479000035)

[3 - Inserção da quantidade de arquivos de passagens e seus nomes 8](#_Toc479000036)

[4 - Leitura e tratamento dos arquivos de passagens 9](#_Toc479000037)

[Exemplo: 11](#_Toc479000038)

[5 - Verificação de passagens úteis 13](#_Toc479000039)

[6 - Comunicação serial entre o pc e o arduino 14](#_Toc479000040)

[7 - Loop de rastreio 16](#_Toc479000041)

[a) Posicionamento inicial 17](#_Toc479000042)

[b) Rastreio 18](#_Toc479000043)

[c) Posicionamento final 19](#_Toc479000044)

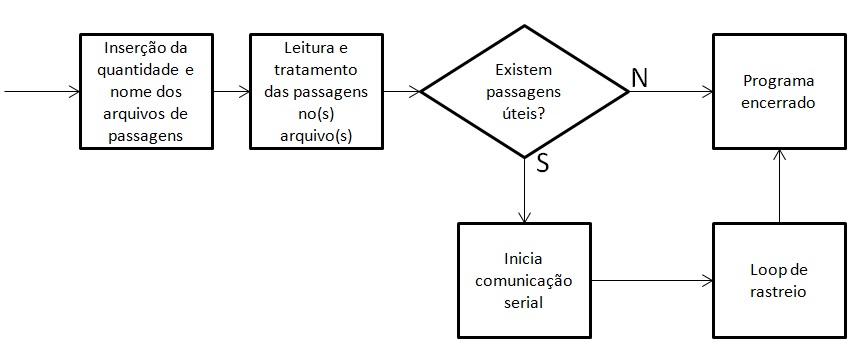
[8 – Considerações finais 20](#_Toc479000045)

[REFERÊNCIAS 21](#_Toc479000046)

# Estrutura geral do algoritmo de controle e rastreio

O programa foi desenvolvido de modo a executar todas as funcionalidades planejadas. Para isso foram utilizadas algumas bibliotecas: uma para comunicação serial e outra para armazenamento e tratamento das tabelas de passagens. Além disso, todo o tratamento feito em relação ao tempo das passagens foi feito com a utilização da biblioteca “time.h” da linguagem C. Foram elaboradas várias funções para executar alguns comandos necessários, que serão descritas mais a frente.

O Diagrama 1 ilustra a visão geral da sequência de funcionamento do software no PC.

**Diagrama 1 – Visão geral do software de controle e rastreio do PC.**

O programa inicia solicitando ao usuário a quantidade de arquivos e logo após os nomes desses arquivos a serem carregados. Após a correta inserção, o PC faz a leitura e tratamento das informações contidas nos arquivos. O passo seguinte é a verificação de passagens úteis no arquivo (será descrito mais a frente), e - caso existam – será executado o loop de rastreio.

Esta sequência será descrita de forma mais detalhada, nos tópicos a seguir.

# 2 - Biblioteca Efem

Essa biblioteca foi desenvolvida por um ex-bolsista do INPE e atual pesquisador, José Marcelo de Lima Duarte, e era utilizada no antigo software de controle e rastreio. Para evitar o retrabalho, foram feitas adaptações para o novo software de modo a deixa-la funcional às necessidades atuais. Ela consiste de uma classe com vários atributos e métodos para armazenar e fazer o tratamento de passagens. Abaixo será feita uma descrição dos atributos e métodos que ela contém.

## Atributos

Os atributos dessa classe foram definidos para armazenar todas as informações contidas na seção 2 do arquivo de passagens padrão [2], tais como: posição de azimute, elevação, distância entre a estação e o satélite (*range*), e a hora exata de cada efeméride. Existem também atributos para armazenar as informações contidas na seção 1 de um arquivo: Hora de início e fim das passagens; máxima elevação do satélite na passagem; e a duração da passagem. Além disso, existe um atributo para armazenar o nome do satélite da referida passagem, e alguns atributos para tratamentos das efemérides, como offsets de ângulos, número de linhas de uma tabela e posição atual na tabela (utilizado durante rastreio). Todos esses atributos são definidos como privados, ou seja, apenas os métodos da classe tem acesso para modificá-los.

## Métodos

Foram definidos alguns métodos para melhor execução das atividades necessárias do software. Os métodos podem ser definidos em três tipos: métodos de definição; métodos de retorno e métodos para tratamento das efemérides. Os métodos de definição atribuem ou atualizam valores dos atributos da classe. Alguns exemplos são métodos para atribuir as informações contidas na seção 1 e 2 do arquivo de efemérides a um objeto da classe. Os métodos de retorno retornam as informações contidas em determinados atributos, como as informações contidas nas seções 1 e 2 do arquivo de passagem. Os métodos de tratamento fazem alterações nos valores de alguns atributos de um objeto criado. Alguns exemplos de alterações são: alteração dos ângulos de referência das passagens da referência geográfica para a referência da antena; verificação de passagem pelo fim de curso da antena, e caso passe, alteração dos quadrantes de operação nessa passagem, entre outros. O Diagrama 1 ilustra os aspectos principais de como a classe Efem, como atributos e métodos.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Efem** | | |
| **A** | **Azimute** |  |  |
| **t** | **Elevação** |  |  |
| **r** | **Hora de inicio** | |  |
| **i** | **Hora de fim** | |  |
| **b** | **Máxima elevação** | |  |
| **u** | **Nome do satelite** | |  |
| **t** | **Tamanho da tabela** | |  |
| **o** | **Posição atual na tabela** | | |
| **M** | set\_data() | |  |
| **é** | set\_efem() | |  |
| **t** | get\_efem() | |  |
| **o** | get\_info() | |  |
| **d** | change\_efem() | |  |
| **o** | next() |  |  |

**Diagrama 1 - Classe Efem.**

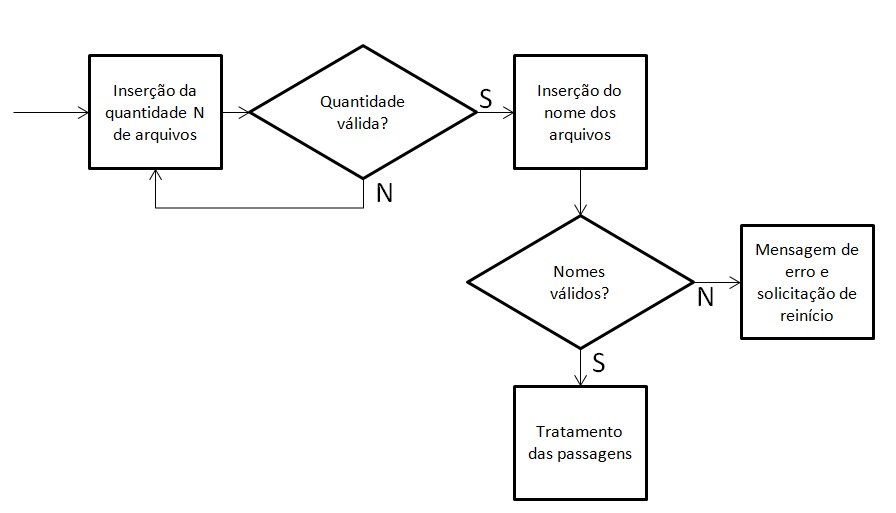
# 3 - Inserção da quantidade de arquivos de passagens e seus nomes

Ao ser executado o programa solicita ao usuário a quantidade de arquivos que serão carregados. O programa atual tem capacidade de carregar até 3 arquivos de passagens. Esses arquivos devem ser de texto (.txt), no formato gerado pelo software STK [4]. Essa quantidade de arquivos foi estabelecida devido a quantidade de satélites que o INPE-CRN tem interesse em rastrear, que são: SCD-1, SCD-2 e o CBERS-4. Desse modo, para qualquer valor inserido maior que 3 ou menor que 1 o programa apresentará erro, indicando que o valor inserido é inválido e solicitando a reinserção do valor.

Após a inserção correta do número de arquivos, o software irá solicitar ao usuário o nome dos arquivos a serem carregados. É de extrema importância que o nome do arquivo inserido contenha “.txt” ao final, que indica o formato do arquivo. Caso contrário o software apresentará erro e será encerrado.

**Exemplo de nome de arquivo a ser inserido:** efemerides-scd2.txt

O Diagrama 2 ilustra esses procedimentos iniciais do software.

**Diagrama 2 – Procedimentos iniciais do software.**

Após essa etapa, o programa fará o carregamento dos arquivos de passagens e o devido tratamento necessário.

# 4 - Leitura e tratamento dos arquivos de passagens

A leitura e o tratamento das efemérides são feito através algumas funções e alguns métodos da biblioteca Efem. A primeira coisa a ser feita é a leitura dos arquivos para contagem de quantas passagens contém em cada um deles. Para isso foi elaborada a função *cont\_passagem().*

* **cont\_passagem():** Esta função recebe o nome de um arquivo texto, abre-o e faz a contagem de quantas passagens o arquivo contém. O número de passagens é definido através da contagem de linhas da seção 1 dos arquivos de passagens.

Sabendo o número total de passagens em todos os arquivos, é feita alocação dinâmica de memória para objetos da classe Efem. Após a alocação o programa irá abrir os arquivos das passagens, ler todas as informações e armazena-las nos objetos das passagens na mesma ordem em que aparecem nos arquivos, que é a ordem cronológica das passagens. A função que carrega as informações dos arquivos textos e as armazena nos objetos das passagens é a *load\_efem();*

* **load\_efem():** Essa função carrega um arquivo de passagens e armazena as informações em objetos da classe Efem. Para isso ela recebe o nome de um arquivo texto a ser aberto, e um intervalo numérico no qual as informações devem ser armazenadas nos respectivos objetos passagens. Exemplo: se um primeiro arquivo carregado contém 5 passagens, e um segundo arquivo de passagens contém 10 passagens, o intervalo numérico passado pra função na segunda vez que ela for executada é 5 a 10. Caso não seja possível abrir o arquivo a função retorna um código de erro.

Caso o usuário carregue apenas um arquivo, as passagens já estarão em ordem cronológica, mas caso contrário, será necessário ordena-las em ordem cronológica. Com esse objetivo, a função *ordenar\_passagens()* foi elaborada.

* **ordenar\_passagens():** Esta função recebe o número total de passagens alocadas, e as ordena cronologicamente. Para isso foi feita sobrecarga dos operadores ‘>’ e ‘=’. O algoritmo utilizado foi o *bubble sort.*

O ultimo procedimento inicial que o programa executa é o tratamento das passagens. Esse tratamento é feito através de métodos da classe Efem, mas para realizar o tratamento de todas as passagens, foi criada uma função chamada *tratar\_efem()*.

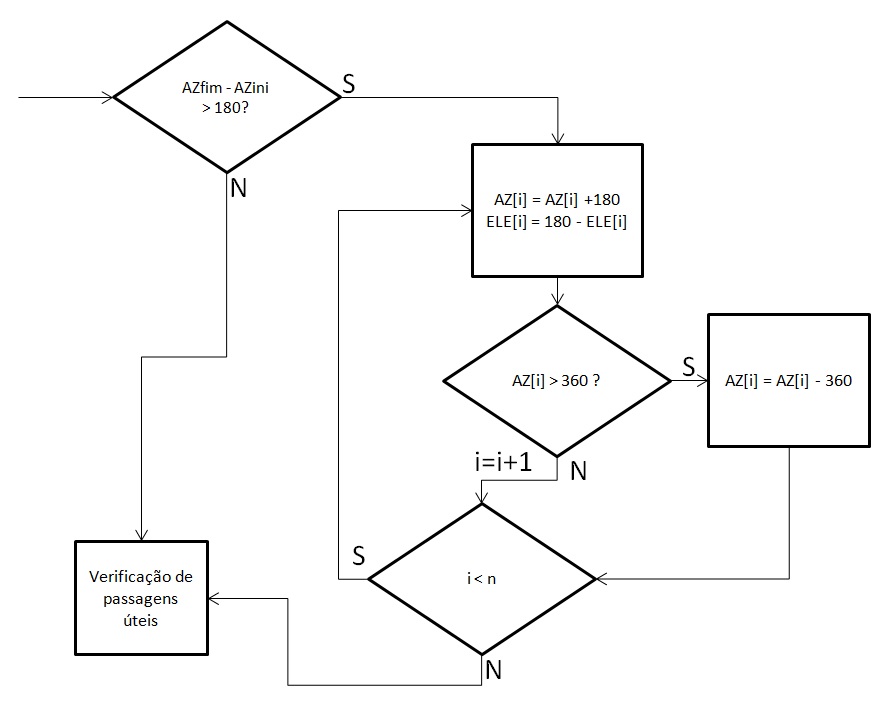
* **tratar\_efem():** Essa função recebe o número de passagens carregadas e executa um loop no qual executa o método Efem::*change\_efem()* para todos os objetos de passagens alocados dinamicamente.

Esse método faz o ajuste de todas as posições de referência armazenado em um objeto Efem, alterando essas posições da referência geográfica para a referência da antena. Essa conversão de referências é feita através das equações 1 e 2.

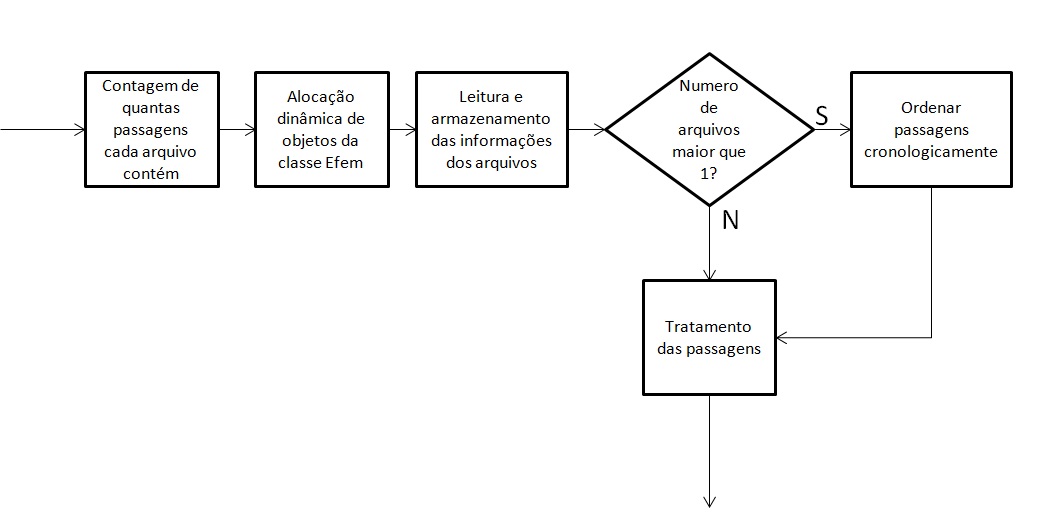
(1)

(2)

Em [1] é descrito o procedimento para encontrar a relação entre as referências. Em seguida o método verifica se haverá passagem pelo final de curso da antena. Caso ocorra, é feito a mudança dos quadrantes de operação da antena, de modo a evitar o final de curso. O diagrama 3 esquematiza o processo de verificação de passagem pelo final de curso, onde *n* é o número total de efemérides em uma passagem.

**Diagrama 3 – Verificação de passagem pelo fim de curso.**

O Diagrama 4 ilustra de forma esquemática como é feito esse processo de leitura e tratamento das passagens.

**Diagrama 4 – Leitura e tratamento dos arquivos de passagens.**

A seguir será feito um exemplo do processo de alocação de memória e ordenação das passagens em ordem cronológica. As passagens aqui citadas são apenas com fins didáticos.

## Exemplo:

Dois arquivos serão carregados. O arquivo 1 contém 3 passagens do satélite SCD-2 e o arquivo 2 contém 4 passagens do satélite CBERS-4. Sendo assim, o número total de passagens é 7. O PC irá alocar espaço na memória para 7 objetos da classe Efem, e os ordenará da seguinte forma. A Tabela 1 indica a ordem das passagens após a alocação.

|  |  |  |
| --- | --- | --- |
| **Dia e hora inicial** | **Satélite** | **Arquivo** |
| 03/02/2017 - 04:03:01 | SCD-2 | 1 |
| 03/02/2017 - 07:33:05 | SCD-2 | 1 |
| 04/02/2017 - 14:21:53 | SCD-2 | 1 |
| 03/02/2017 - 05:03:01 | CBERS-4 | 2 |
| 03/02/2017 - 06:21:20 | CBERS-4 | 2 |
| 04/02/2017 - 21:21:14 | CBERS-4 | 2 |
| 05/02/2017 - 12:30:45 | CBERS-4 | 2 |

**Tabela 1 – Ordem das passagens após alocação**

Após a alocação de memória, o PC irá ordenar as passagens em ordem cronológica. A tabela 2 mostra a sequência de passagens após a ordenação cronológica das passagens.

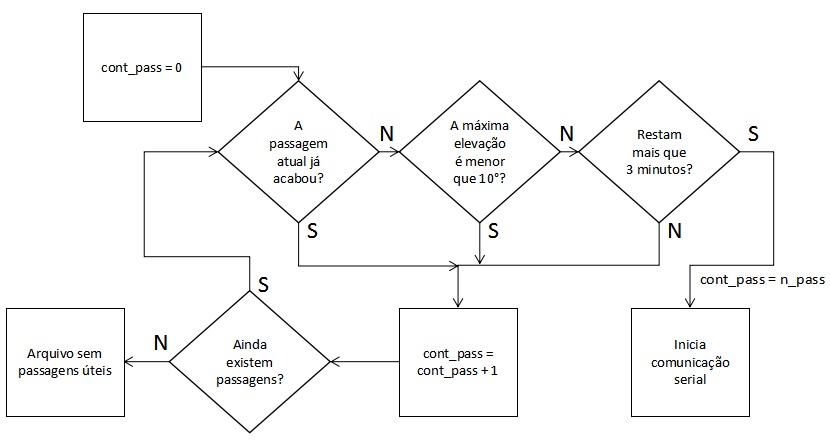
|  |  |  |
| --- | --- | --- |
| **Dia e hora inicial** | **Satélite** | **Arquivo** |
| 03/02/2017 – 04:03:01 | SCD-2 | 1 |
| 03/02/2017 – 05:03:01 | CBERS-4 | 2 |
| 03/02/2017 – 06:21:20 | CBERS-4 | 2 |
| 03/02/2017 – 07:33:05 | SCD-2 | 1 |
| 04/02/2017 – 14:21:53 | SCD-2 | 1 |
| 04/02/2017 – 21:21:14 | CBERS-4 | 2 |
| 05/02/2017 – 12:30:45 | CBERS-4 | 2 |

**Tabela 2 – Ordem das passagens após ordenação cronológica.**

Ao terminar de ordenar as passagens o PC irá realizar a verificação de passagens úteis, que é descrita a seguir.

# 5 - Verificação de passagens úteis

Ao fim do tratamento das passagens o programa irá procurar por passagens úteis. Os critérios que foram definidos para decidir se uma passagem é útil foram: a passagem não pode ter acabado; a passagem tem que ter elevação máxima de pelo menos 10°; e a passagem tem que ter pelo menos 3 minutos restantes. Tais critérios foram elaborados pensando também em passagens pegas depois do inicio (após um rastreio executado, por exemplo, quando o programa for para a próxima passagem e ela já estiver ocorrendo). Caso a passagem atual na contagem não atenda aos requisitos, o programa irá pular para a passagem seguinte. O Diagrama 5 mostra de forma sequencial esses procedimentos, onde *cont\_pass* representa o contador que indica a passagem atual, e *n\_pass* representa o valor que o contador chegou após pular as passagens inválidas.

**Diagrama 5 – Verificação de passagens úteis**

Caso o programa avance na procura e não ache nenhuma passagem válida, o programa será encerrado e exibirá uma mensagem dizendo que o arquivo não possui passagens úteis. Caso uma passagem válida seja encontrada, o programa irá iniciar a comunicação serial com o arduino.

# 6 - Comunicação serial entre o PC e o arduino

A comunicação entre o arduino e o PC foi elaborada de acordo com o protocolo desenvolvido. Para implementar essa comunicação foi utilizada uma biblioteca chamada “serial”, obtida na internet no site Code Project, a qual foi desenvolvida por Ramon de Klein. O objetivo aqui não é detalhar as funcionalidades da biblioteca, que são inúmeras, sendo apenas descrever os recursos que estão sendo utilizados. Vale salientar que para executar a comunicação, esta biblioteca utiliza a biblioteca “Windowns.h”. Portanto, o software atual funcionará apenas no sistema operacional Windows. Caso seja necessário a operação em outro Sistema Operacional será necessário substituir a biblioteca de comunicação. Maiores detalhes sobre a biblioteca estão disponíveis no site do projeto, em [3]. A biblioteca foi testada e funcionou perfeitamente para os objetivos do protocolo em questão.

A biblioteca permite a configuração de vários parâmetros como: taxa de transmissão, ter ou não bit de paridade, bit de parada, se o PC deve esperar determinada quantidade de bytes no buffer de entrada antes de fazer a leitura ou se ele deve simplesmente ler na hora que a função for chamada, entre outras.

A comunicação é iniciada criando um objeto para a porta serial, abrindo-a e configurando-a. O nome da porta serial a ser utilizada é inserido no código do programa no formato *const wchar\_t*, e é um argumento da função serial::open(). Neste ponto, é criado o objeto da porta serial e feito algumas configurações como: a taxa de transmissão para 9600 bps, tamanho do byte para 8 bits, sem bit de paridade e 1 bit de parada. Além disso, é configurada a função de leitura da porta para esperar até que chegue a quantidade de bytes esperada, já que é um valor bem determinado (essa quantidade de bytes é um parâmetro da função de leitura). Feito isso, verifica-se se a porta serial realmente foi aberta. Caso tenha sido, o programa continuará normalmente. Caso contrário, o programa será encerrado exibindo mensagem de erro.

Para realizar a comunicação serial foram elaboradas funções que implementam o protocolo de comunicação, as quais são utilizadas durante o “loop de rastreio”, o qual será descrito mais a frente.

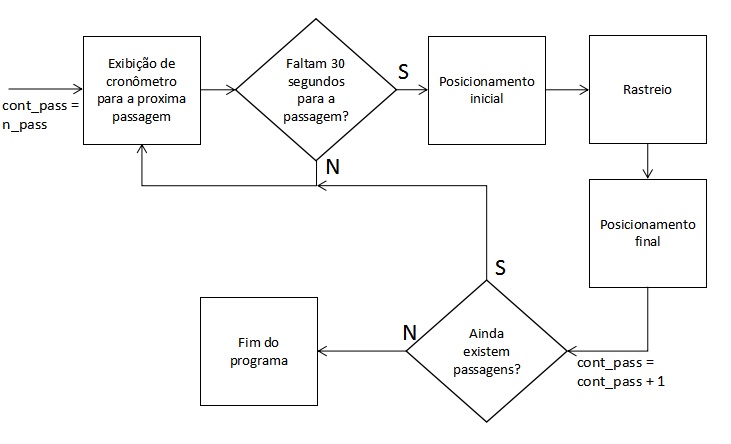
A seguir será feita uma descrição funcional das funções elaboradas para enviar os comandos do protocolo ao arduino.

* **Função send\_set():** Essa função recebe como parâmetros: os ultimos valores de azimute e elevação carregado na função Efem::get(), a string onde será formado o comando set a ser enviado, e a referência para um contador de erros de comunicação. Com isso, a função codifica os ângulos em 16 bits, os separa em dois bytes cada um, forma a string de mensagem incluindo o checksum e envia para o arduino. Após o envio ela esperará o a resposta do arduino, que pode ser um ACK ou NACK. Caso seja um NACK, ela irá incrementar o contador de erros, reenviar a string de comando para o arduino, e irá esperar outra resposta dele. Permanecerá nesse ciclo até que o arduino responda com um ACK (ou seja, até que a mensagem seja enviada corretamente). A escrita dos bytes na porta serial, é feita utilizando o método *serial::Write()*.
* **Função send\_state():** Essa função recebe como parâmetros: uma string onde será formada a string de state, referências para as variáveis que receberão a posição de azimute e elevação atual da antena, e a referência para o contador de erros. A função envia para o arduino o byte ‘S’ (comando STATE do protocolo), e espera para ler a string que ele enviar. Feita a leitura, o PC irá verificar se o checksum está correto. Caso não esteja, irá incrementar o contador de erros e enviar NACK para o arduino. Irá então esperar para ler novamente a string de STATE. Ficará nesse ciclo até que o checksum venha correto. Feito isso, a função decodificará os ângulos contidos nos bytes a1, a0, e1, e0, e decodificará o status do sistema contido no byte C. A leitura dos bytes na porta serial é feita utilizando o método *serial::Read()*.
* **Função send\_power():** Essa função não recebe parâmetros. Após ser chamada, ela escreve o byte ‘P’ (comando POWER do protocolo) na porta serial e aguarda uma resposta (ACK ou NACK). Caso seja um NACK, ela irá reenviar o byte para o arduino, e irá esperar outra resposta dele. Permanecerá nesse ciclo até que o arduino responda com um ACK. A escrita do byte na porta serial, é feita utilizando o método *serial::Write()*.
* **Função verifica\_checksum():** Essa função faz a verificação se o checksum na string de STATE está correto. Caso esteja, ela retornará 0, caso contrário retornará 1. É utilizada na função send\_state().
* **Função envia\_reconhecimento():** Essa função recebe como parâmetro o código de erro retornado na função verifica\_checksum(). Caso o valor seja zero, a função escreverá ACK na porta serial, caso contrário escreverá NACK. A escrita do byte na porta serial, é feita utilizando o método *serial::Write()*.
* **Função reconhecimento\_arduino():** Essa função recebe a resposta do arduino a algum comando enviado pelo PC, um ACK ou NACK. Caso a leitura seja um ACK, ela retornará 0, caso contrário retornará 1. A leitura do byte na porta serial é feita utilizando o método *serial::Read()*.

Após a correta inicialização e configuração da porta serial, o PC irá para o próximo estágio do programa, que é o inicio do loop de rastreio.

# 7 - Loop de rastreio

O loop de rastreio é onde o software e irá passar maior parte do tempo de execução, e consiste de: esperar até a hora de início da próxima passagem; executar o posicionamento inicial; executar o rastreio; executar o posicionamento final; e verificar se ainda existem passagens a ser feito o rastreio. Esse ciclo irá se repetir para o número de passagens úteis totais de todos os arquivos carregados. O Diagrama 6 ilustra graficamente a sequência do loop de rastreio, onde *cont\_pass* representa o contador que indica a passagem atual, e *n\_pass* representa o valor que o contador chegou após pular as passagens inválidas.

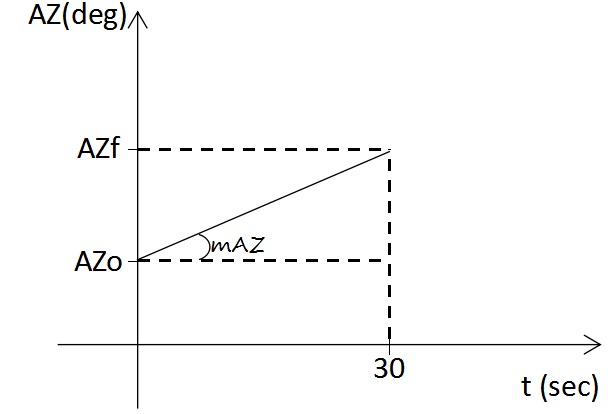
**Diagrama 6 - Loop de rastreio.**

O loop de rastreio inicia com o cronômetro, exibindo para o usuário o tempo restante para a próxima passagem. São exibidas também as informações contidas na seção 1 dos arquivos de passagens: Hora de início e fim da passagem, máxima elevação da antena no rastreio, a duração da passagem e o nome do satélite que será rastreado. Além disso, nesse estágio, o PC mandará constantemente o comando STATE para o arduino, exibindo para o usuário também a posição atual da antena, o status dos relés, do controle manual, da potência do sistema e o status dos *drivers* de azimute e elevação.

O cronômetro será exibido até faltar 30 segundos para a passagem iniciar. Quando isso acontecer, o PC irá gerar uma rampa de posicionamento da posição atual da antena até a posição onde o satélite surgirá no horizonte. Esse posicionamento irá durar 30 segundos, e logo em seguida a antena irá executar o rastreio. Isso é feito para evitar altos sinais de controle no inicio do rastreio devido aos altos erros de posição no inicio da passagem, que podem vir a diminuir a vida útil dos componentes do sistema.

## Posicionamento inicial

Para gerar a rampa de posicionamento foi elaborada uma função chamada *gerar\_rampa().* Essa função utiliza as variáveis globais das posições de referência atual e posição atual da antena (obtida através do comando STATE), calcula a inclinação de uma reta que em 30 segundos leve a antena para a posição inicial do rastreio e gera dois *arrays,* um para as posições de azimute e um para as posições de elevação. A Figura 1 ilustra graficamente o processo criação da rampa.

  
**Figura 1 - Criação da rampa de posicionamento.**

Neste gráfico, *AZf* representa a primeira posição de azimute contida na tabela de uma passagem, *AZo* representa a posição atual da antena, e *mAZ* representa a inclinação da reta, sendo dada por:



A partir daí, calcula-se as posições da rampa por:

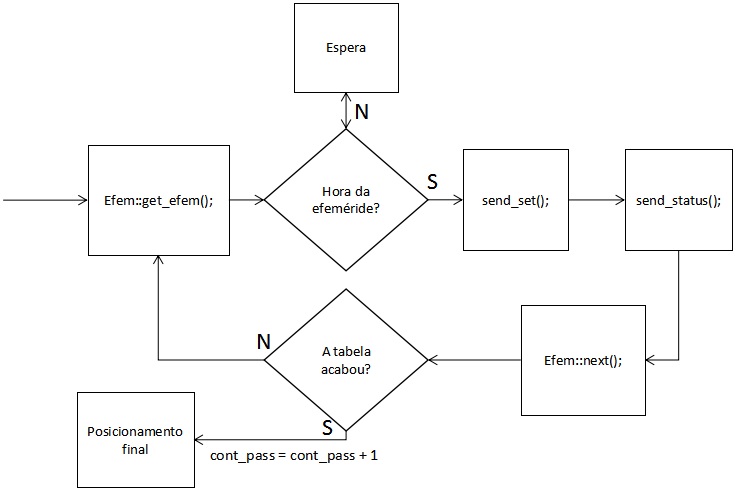


Apesar de o exemplo ter mostrado o calculo para as posições de azimute, o calculo para as posições de elevação é feito de forma idêntica.

Caso a passagem atual que o PC for rastrear já esteja ocorrendo, o PC irá buscar a posição que a antena deverá estar 30 segundos na frente e realizar esse mesmo procedimento para gerar a rampa de posicionamento. Com a rampa gerada, o PC envia o comando POWER ao arduino, que ligará a potência do sistema. A partir daí serão enviados comandos SET, onde serão enviadas as posições calculadas nas rampas de posicionamento, seguidos de comandos STATE. Durante esse processo a antena se moverá para a posição onde o satélite surgirá. Ao chegar lá, será iniciado o processo de rastreio.

## Rastreio

O rastreio é o período em que a antena de fato estará seguindo o satélite. O Diagrama 7 ilustra sequencialmente o processo de rastreio no software.

**Diagrama 7- Rastreio**

O rastreio inicia utilizando o método *Efem::get\_efem()*, a qual retorna as posições de azimute, elevação e a hora da efeméride, na linha atual da tabela. Após isso o PC espera até a hora da efeméride. Quando chega a hora, envia o comando SET e em seguida o comando STATE. Após isso o PC utiliza o método *Efem::next(),* que avança uma linha na tabela de efemérides. Caso a tabela tenha acabado, esse método retorna um código de erro.

Esse loop se repetirá até que a tabela de efemérides acabe, onde a partir daí o PC irá iniciar o posicionamento final.

## Posicionamento final

O posicionamento final é feito de forma similar ao posicionamento inicial com a função *gerar\_rampa()*, através do calculo de uma rampa que leva a antena de sua posição atual no final do rastreio para a posição que desejamos, que é azimute em 180° e elevação em 90°. Essa posição foi adotada por ser a de menor resistência do vento, chamada “posição de descanso”. Sendo assim, a inclinação da rampa de posicionamento final é obtida através de:



E a rampa de posicionamento é dada por:



Durante o posicionamento final o PC enviará ao arduino comandos SET com as posições da rampa calculada e comandos STATE. Ao final desse posicionamento o PC irá enviar um comando POWER, onde será desligada a potência do sistema. Em seguida será incrementado o contador de passagens e feita a verificação se ainda existem passagens a ser rastreadas. Essa verificação é feita comparando o valor atual do contador *cont\_pass* e o valor total de passagens carregadas. Caso ainda existam passagens a ser rastreadas, o PC irá mostrar um cronômetro com o tempo restante para a próxima passagem, além das informações da passagem, descritas anteriormente.

Caso o PC incremente o contador de passagens e não existam mais passagens a ser feito o rastreio, o PC irá terminar o programa e exibir uma mensagem ao usuário informando que as passagens do(s) arquivo(s) acabaram.

# 8 – Considerações finais

# 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] SARAIVA, A. B. D. Especificação do arquivo de efemérides a ser utilzado no novo programa de controle e rastreio, Instituto Nacional de Pesquisas Espaciais, Natal. 2017.

[3] <https://www.codeproject.com/Articles/992/Serial-library-for-C.> Acesso em: 03/04/2017

[4] SARAIVA, A. B. D. Descrição de como gerar um arquivo de passagens no software STK, Instituto Nacional de Pesquisas Espaciais, Natal. 2017.

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