3.1 Considerações Iniciais

Nesse Capítulo encontram-se as implementações presentes no projeto tanto de *hardware* quanto de *software*. Na parte de *software*, é explicado o funcionamento dos módulos como um todo e por fim, o funcionamento detalhado de cada função utilizada. HARDWARE MISSING Também estão presentes os resultados obtidos, análises dos resultados e uma análise do projeto como um todo, levando em conta as dificuldades e limitações que surgiram durante o processo.

3.2 Projeto

O objetivo do projeto é desenvolver um sistema de irrigação autônomo que, através de imagens da área a ser irrigada e dados coletados da internet e de sensores, decida quais regiões precisam ser irrigadas e o quanto de água cada uma dessas regiões necessita.

O sistema de irrigação autônomo possui duas frentes igualmente importantes: O *software* que é o responsável por controlar o sistema e tomar a decisão de regar ou não a área e o *hardware*, que é onde o *software* irá ficar hospedado e também responsável pelas ações do sistema.

O *software* é dividido em duas partes, a parte de tomada de decisão, que foi desenvolvida na linguagem C++ e a parte responsável por ler os sensores e transmitir os valores obtidos, que foi desenvolvida na linguagem do Arduino, que é baseada na linguagem C.

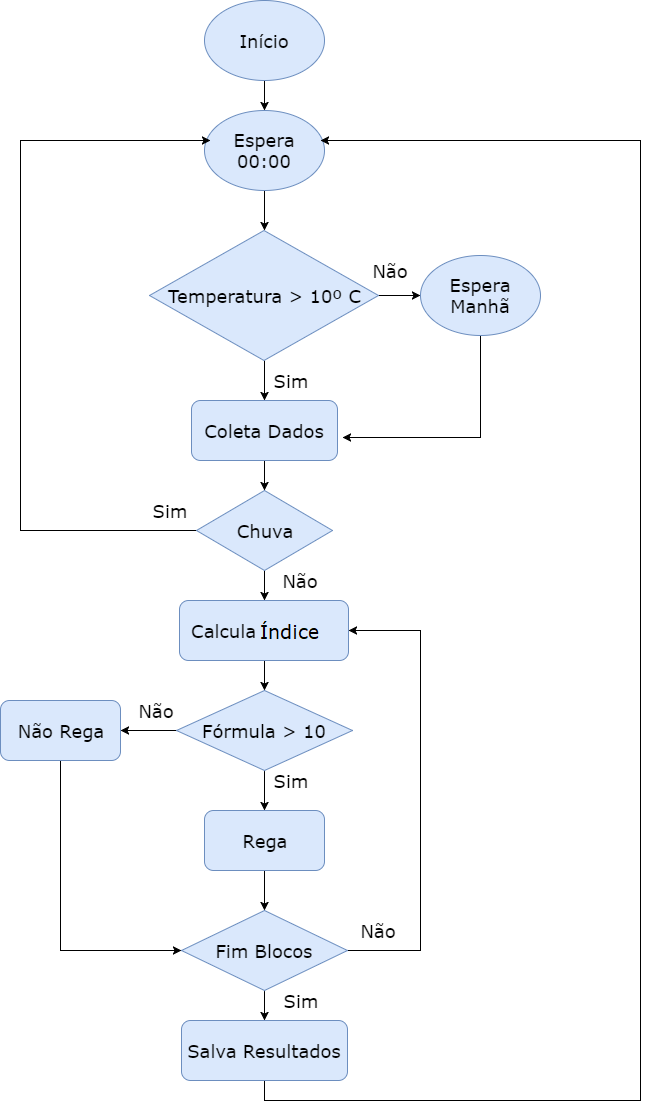
O *hardware* consiste na Raspbery Pi, hospedeira do programa responsável pela tomada de decisão, um Arduino, responsável por ler os valores dos sensores e outro Arduino responsável por controlar o motor que regula a quantidade de água usada para regar uma área.

3.2.1 Software

O programa principal, o que recebe os dados e a imagem e gera a decisão para cada bloco a ser irrigado foi dividido em quatro módulos: Núcleo, onde se encontra a função *main*, que comanda todo o programa; Processamento de Imagem, que é onde a imagem recebida pelo programa é processada a fim de extrair as informações necessárias; Tomada de Decisão, onde se encontram as funções responsáveis por obter os dados da *internet* e dos sensores, assim como a função que salva os resultados em um arquivo e Aspersor, módulo que se encarrega de moldar o jato que vai irrigar o bloco selecionado.

3.2.1.1 Núcleo

No núcleo, todos os outros módulos são chamados através de suas funções. Seu fluxo principal está representado no fluxograma abaixo:



O programa foi projetado como um laço que espera sempre o horário de regar as plantas, que nesse projeto foi definido como meia noite (00:00). Esse horário foi escolhido pois se o jardim for irrigado já com sol, a água seca mais rápido, diminuindo seu aproveitamento. Seguindo esse pensamento, o horário escolhido poderia ter sido logo após ao pôr do sol mas o início da noite foi escolhido pois a incidência de pessoas andando no jardim é menor.

Caso a temperatura esteja muito baixa (abaixo dos 10º C), opta-se por regar as plantas de manhã, evitando que a água congele.

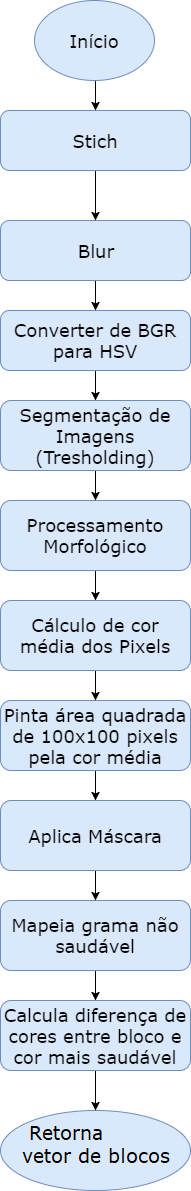
Definido o horário em que as plantas serão regadas, chama-se, então, as funções responsáveis por coletar os dados. Desses valores coletados, o primeiro a ser utilizado é o coeficiente de chuva. Ele representa a probabilidade de chuva contabilizando a previsão do tempo para o dia atual e os próximos dois (ensolarado, nublado, chuvoso, tempo claro e nevando) e quanto maior o seu valor, menor a necessidade de se regar as plantas. Se o número for muito baixo, significa que não há necessidade e então o laço volta para o começo, esperando pelo horário da próxima irrigação.

Após a verificação da possibilidade de chuva, vem o procedimento principal do programa, que é outro laço onde se faz o cálculo da fórmula que diz se o bloco (a imagem com a área total a ser irrigada é dividida em blocos, sendo que para cada um é calculada a fórmula para decidir se eles serão regados ou não) deve ser irrigado ou não. A fórmula leva em conta os valores coletados anteriormente, que são: Insolação, Velocidade do Vento, Umidade do Solo, Coeficiente de Chuva, Cor da Área e o Resultado do dia Anterior. Assim como o coeficiente da chuva, esse cálculo resulta num valor numérico entre 0 e 100 onde 100 é a necessidade máxima de água e de 10 até 0 significa que não é necessário regar aquele bloco.

Feito o cálculo da fórmula e decidido se o bloco deve ser irrigado ou não, faz-se o mesmo para os próximos até o fim dos blocos. Ao terminar o laço interior, os resultados são salvos num arquivo para serem utilizados no dia seguinte e então o programa volta a esperar pela meia noite.

3.2.1.2 Processamento de Imagens

Para a criação desse módulo, foi utilizado como referência o código criado por (Tapparo, 2016). O módulo do programa responsável pelo processamento de imagens é extremamente dependente da biblioteca OpenCV, portanto algumas etapas de seu fluxo são funções implementadas pela biblioteca. O fluxo completo deste módulo encontra-se no fluxograma abaixo:



O processamento de imagens nesse projeto é uma sucessão de ações em cima das imagens iniciais, resultando em um vetor de blocos que representa as áreas que serão avaliadas para irrigação. Seu fluxo é simples e não possui nenhum laço de repetição ou rota alternativa, essa sequência de ações é chamada no início do programa e só é executada uma vez.

As imagens que chegam como parâmetro para o programa são juntadas através do método de *stitching* e a partir desse resultado são feitas as operações necessárias para extrair a informação desejada da imagem. O *blur*, conversão do espaço BGR (RGB na notação do OpenCV) para HSV, segmentação da imagem e processamento morfológico são todos métodos implementados pela biblioteca do OpenCV e são necessários para manter na imagem somente o que é desejado, que no caso desse projeto é a grama. A partir da imagem gerada por essas operações, são calculados a cor média dos pixels de bloco em bloco, sendo cada bloco um quadrado de lado de 100 *pixels*. Após esse cálculo, todos os pixels dentro da área onde foi calculada a média são “pintados” na cor média. Após essa transformação, é aplicada uma máscara para determinar quais blocos são considerados saudáveis e quais não são. Os blocos considerados não saudáveis, ou blocos que necessitam ser regados, são salvos num vetor que será mandado ao núcleo do programa onde será feito o processo de tomada de decisão.

3.2.1.3 Tomada de Decisão

Assim como no processamento de imagens, o fluxo de tomada de decisão é uma sequência de operações com a diferença que no final existem duas possibilidades: regar ou não regar o bloco. O fluxograma da tomada de decisão é:

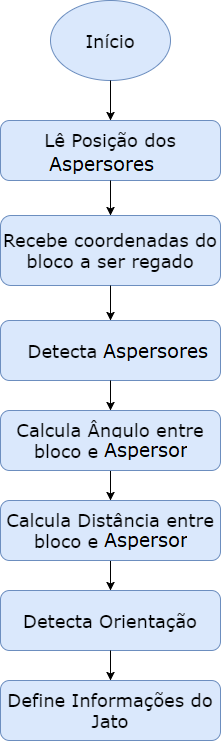


As informações do clima e do vento são retiradas da internet através de uma linha de comando, já as informações da umidade do solo e insolação são obtidas através dos respectivos sensores. O resultado anterior é lido de um arquivo onde todos os resultados são salvos, como mostrado no fluxograma do núcleo do programa. Por fim, o valor da cor é o módulo do vetor diferença entre a cor do bloco e a cor que é o valor inferior no intervalo que define se a grama precisa ser regada, ou seja, é o valor da grama menos saudável.

Após a obtenção de todos os valores necessários, faz-se o cálculo da fórmula para decidir se o bloco vai ser regado ou não. A partir da leitura do resultado anterior até o final, o fluxo é o mesmo que o fluxo do *loop* interior encontrado no Núcleo. Isso acontece pois para a tomada de decisão, não existe uma função que englobe todas as outras do mesmo módulo como acontece com o processamento de imagens, portanto, as funções são chamadas individualmente pelo núcleo.

3.2.1.4 Aspersores

O módulo responsável pelos aspersores é o mais simples, sendo dois métodos os mais importantes: O método responsável por ler os aspersores de um arquivo e o método que define os jatos. Esse último detecta o aspersor adequado para o bloco que foi passado como parâmetro e chama todas as funções restantes do módulo: a do cálculo da distância entre aspersor e módulo e a do cálculo do ângulo entre aspersor e módulo. A orientação é determinada na detecção de qual aspersor será utilizado.



3.3 Descrição das Atividades Realizadas

Como foi dito na seção anterior, o projeto possui duas partes: o *software* e o *hardware*. Nesta sessão encontra-se a explicação para as implementações de cada módulo e HARDWARE.

3.3.1 Software

O software produzido para esse projeto é *open source* e encontra-se em <https://github.com/pedrosmv/TCC/> . O código referente à tomada de decisão, feito na linguagem C++, foi dividido em quatros arquivos, cada um relativo a um módulo e três headers, um para cada módulo menos o núcleo, que não possui função além da *main*.

3.3.1.1 Processamento de Imagem

3.3.1.1.1 Stitch

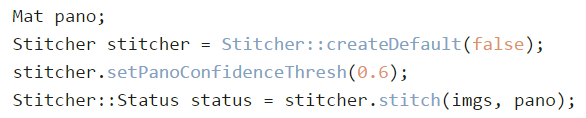
A primeira função do módulo de processamento de imagens é stitch:



Os seus argumentos, *argc* e *argv* são os mesmos recebidos pela *main* passados via linha de comando. O inteiro *argc* é um valor que indica a quantidade de argumentos que foram passados na chamada do programa. Já *argv* é um vetor do tipo *char* que contém os argumentos. É necessário atentar-se para o fato que *argv[0]* é o nome do programa, portanto os argumentos encontram-se nas posições posteriores.

O tipo de retorno da função, Mat, é um tipo definido pela biblioteca OpenCV, que representa um *array* de canal simples ou multicanal que pode armazenar imagens. No caso da função aqui citada o retorno é uma imagem, união das imagens passadas por argumento.

As imagens presentes no vetor *argv* são lidas e colocadas num vetor de imagens do tipo Mat chamado “imgs” e esse vetor será utilizado pelo método stitch do objeto sitcher, como é possível observar na imagem abaixo:



A imagem resultado, presente em pano, é o retorno da função.

Nesse projeto, durante o desenvolvimento, as imagens de entrada foram:



Após a chamada da função, o resultado foi:

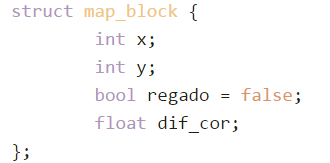


3.3.1.1.2 Image Processing

Essa função é o método principal do módulo de processamento de imagens. É ela que vai chamar todas as outras funções, exceto a stitch(), e extrair o vetor de blocos que será utilizado pelo núcleo do programa.



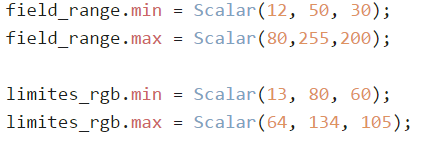
Esse vetor de blocos, que é o tipo do retorno da função é um vetor da *struct* map\_block, determinada no *header* do módulo, imageprocessing.h:



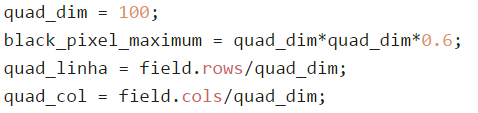
Nessa struct estão agrupados os valores: x, posição X do bloco no mapa de pixels; y, posição Y do bloco no mapa de pixels; regado, uma variável booleana que diz se o bloco foi regado ou não e dif\_cor, o valor da diferença entre a cor do bloco e a cor que é o valor inferior no intervalo que define se a grama precisa ser regada, ou seja, é o valor da grama menos saudável.

Como argumentos, image\_processing recebe a imagem gerada pela função stitch() e o endereço de dois ponteiros, max\_col representando o número máximo de colunas e max\_linha representando o número máximo de linhas. Essas variáveis são utilizadas no módulo do aspersor e, portanto, seus valores são atribuídos no módulo de processamento de imagens, único local onde esse valor é acessível.

A primeira parte dessa função é definir o intervalo de cores que define a área de interesse da imagem, no nosso caso, o gramado (field\_range) e o intervalo em que o gramado necessita de água (limites\_rgb). Os valores usados nesse projeto foram definidos em (Tapparo, 2016).



Após a definição dos intervalos, são definidas algumas variáveis que são utilizadas entre todas as funções:



Essas variáveis estão relacionadas com a divisão da imagem em blocos, quad\_dim é a quantidade de pixels que formam um lado do bloco, black\_pixel\_maximum define a quantidade máxima de pixels que podem ser pretos dentro de um bloco e quad\_linha e quad\_col representam o número de blocos no eixo Y e X da imagem respectivamente.

O próximo passo da função é aplicar as operações necessárias para poder extrair da imagem as informações desejadas. A primeira parte utiliza apenas funções implementadas pelo OpenCV, resultando em uma imagem em preto e branco onde o branco é gramado e o preto é o que não deve ser considerado nas próximas operações.

A primeira dessas funções é a blur, usada para suavizar a imagem:



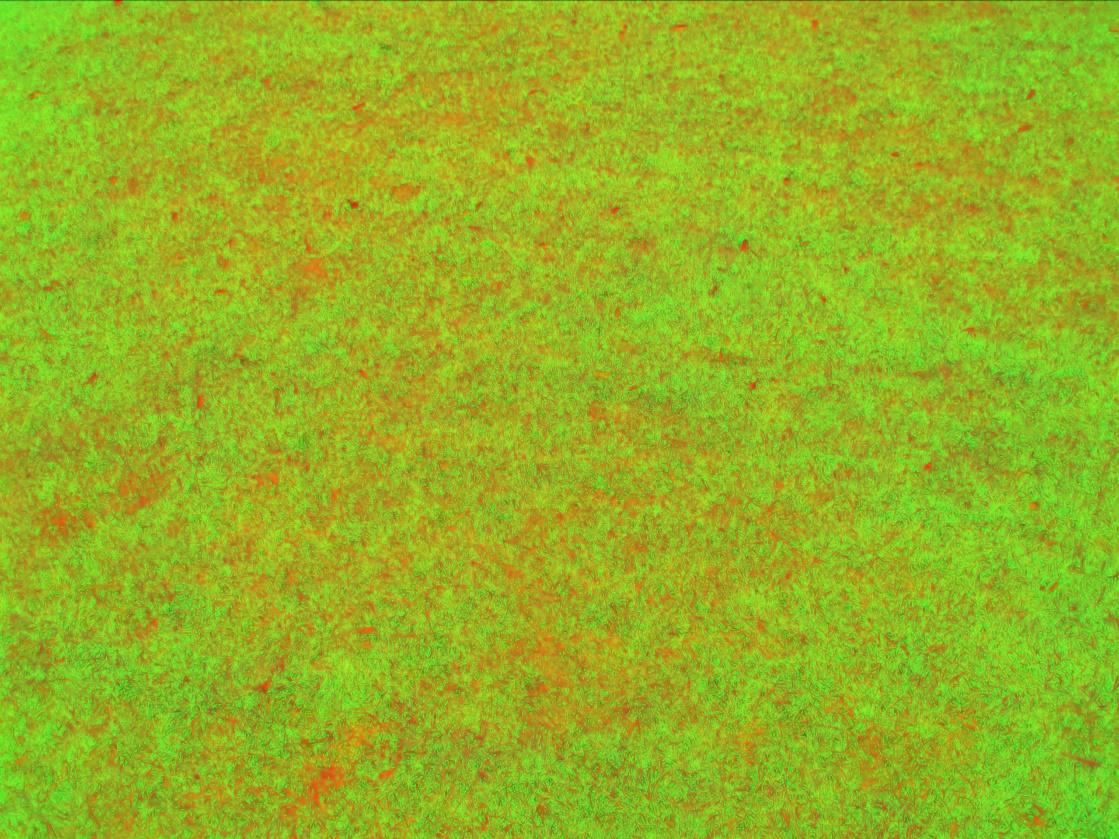
O resultado pode ser visto na figura abaixo:



Após suavizar a imagem, é feita a conversão do espaço de cores BGR (RGB na notação do OpenCV) para HSV:



O resultado pode ser visto na figura abaixo:



Com a imagem no espaço de cor HSV, é feita a segmentação na imagem, usando como intervalo de *treshold* o intervalo field\_range, que vai delimitar o que é gramado e o que não é:



Como na imagem utilizada para teste não há nada além de gramado, o resultado, que pode ser visto na figura X, não apresenta uma região em preto, apenas pequenos pontos, que serão corrigidos pelo processo morfológico:



O processo morfológico encarrega-se de eliminar as imperfeições do *tresholding*, resultando na figura Y.



(Segmentação)



(Processamento morfológico)

Após o processamento morfológico, a imagem está pronta para ser processada a fim de fornecer as informações desejadas. A partir da figura Y, é chamada a função:



Essa função vai calcular a média das cores dos pixels numa área de 100x100 pixels e pintar todos os pixels dentro dessa área da cor média. O resultado é uma imagem quadriculada, como na figura abaixo V:

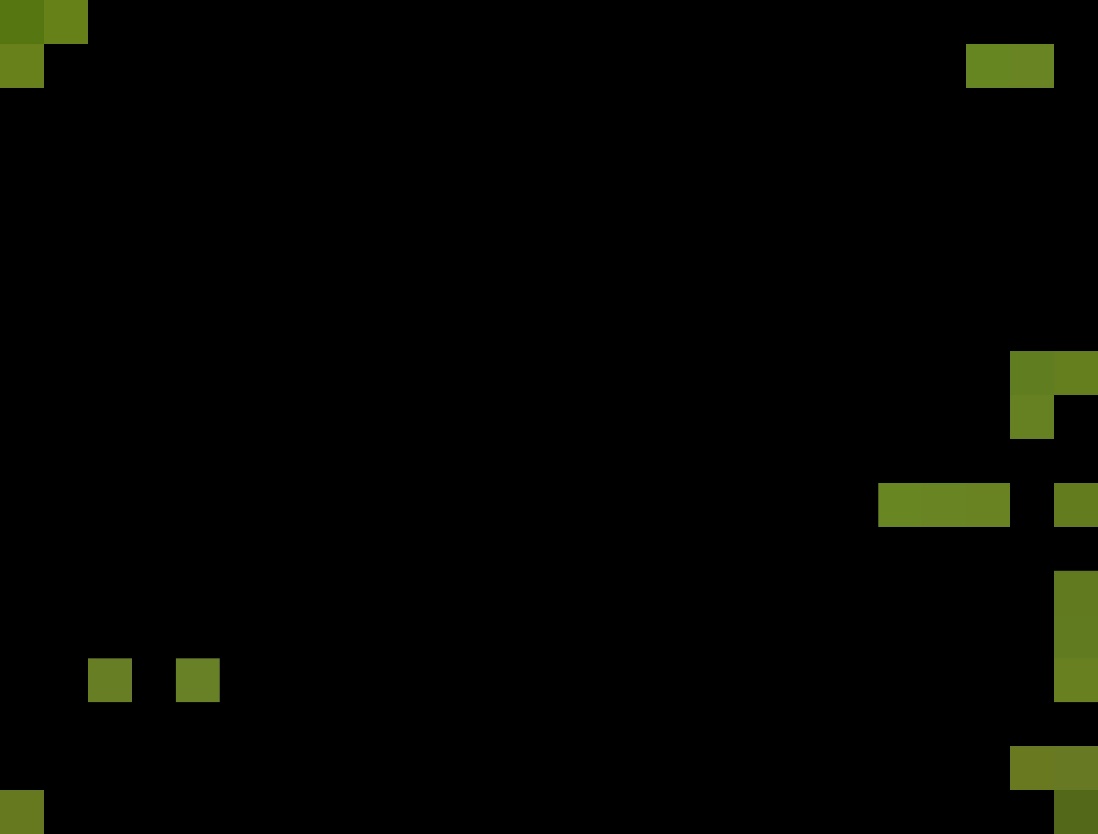


Nessa imagem, utilizando o intervalo determinado em limites\_rgb, será feita outra segmentação, a fim de gerar uma máscara que separará os blocos que devem ser regados daqueles que não devem ser regados.

A máscara é gerada pela função:



O resultado é uma imagem onde os blocos que devem ser regados estão em preto e os blocos considerados saudáveis estão em branco. Após copiar os pixels brancos para uma outra imagem, o resultado é a imagem abaixo Z:



Por fim, a função que gera o vetor de blocos que será retornado pela image\_processing é 

Essa função utiliza as figuras V e Z para dizer se o bloco está saudável ou não. Se o bloco é considerado não saudável, é criado um bloco do tipo map\_block que é salvo no vetor de blocos. Terminada a execução dessa função, termina a execução da image\_processing e o vetor de blocos é retornado.

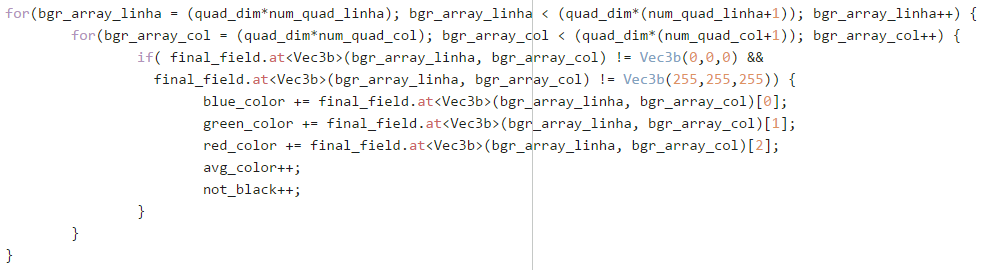
3.3.1.1.3 Cálculo da Média de Cores do Pixel

A função calculateAvgPxlColor é responsável por transformar a imagem que passou pelo processamento morfológico na imagem quadriculada vista na figura V.

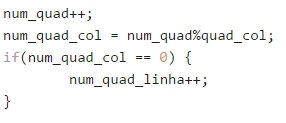


Seus argumentos são: final\_field, imagem que passou pelo processamento morfológico; quad\_dim, dimensão do lado do quadrado que vai ser pintado com a cor média dos pixels dentro de sua área; quad\_linha representa o número de blocos no eixo Y da imagem; quad\_col representa o número de blocos no eixo X da imagem e black\_pixel\_maximum é o máximo de pixels que podem ser pretos dentro de um bloco. O retorno da função é um objeto do tipo Mat, classe do OpenCV usada para representar uma imagem.

O núcleo dessa função é um laço que itera por todos os blocos da imagem. Dentro desse laço principal, para percorrer a imagem passada por parâmetro, são usados dois laços *for* encadeados, como pode-se ver na figura abaixo:



Nesses dois laços, as variáveis de iteração bgr\_array\_linha e bgr\_array\_col vão iterar sobre os pixels de um bloco. Começando em 0, num\_quad\_linha e num\_quad\_col são alteradas respectivamente fora desses laços interiores, mas dentro do laço principal, após a execução da função presente na figura C, como podemos ver na figura abaixo B:



Com isso, a cada iteração do laço principal, os laços interiores percorrem 100 pixels no eixo x e 100 pixels no eixo y.

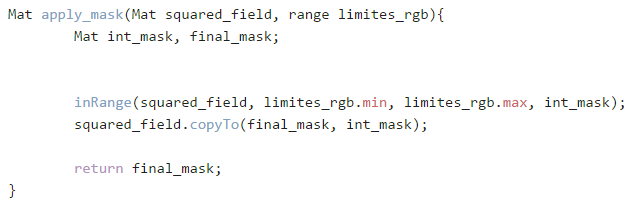
Dentro dos dois laços interiores, é verificado se o pixel não é branco nem preto, e então soma-se o valor de cada cor (verde, vermelho e azul) em sua respectiva variável. As variáveis avg\_color e not\_black servem para contar quantos pixels foram contabilizados e se eles não eram pretos, respectivamente.

Saindo dos laços interiores, são feitas duas verificações antes de pintar o bloco com a cor média: se o número de pixels não pretos é menor ou igual que o máximo de pixels pretos permitidos ou se ele é zero e se o número de pixels contabilizados foi 0. Essas verificações são feitas para evitar erros na hora de calcular a cor média.

Por fim, a modificação na imagem é feita através da função rectangle da biblioteca OpenCV, como podemos ver na figura abaixo C:



3.3.1.1.4 Aplicação da Máscara

A função responsável por aplicar a máscara que irá separar os blocos saudáveis dos não saudáveis é um encapsulamento da função inRange da biblioteca OpenCV. Como pode-se ver na figura abaixo: 

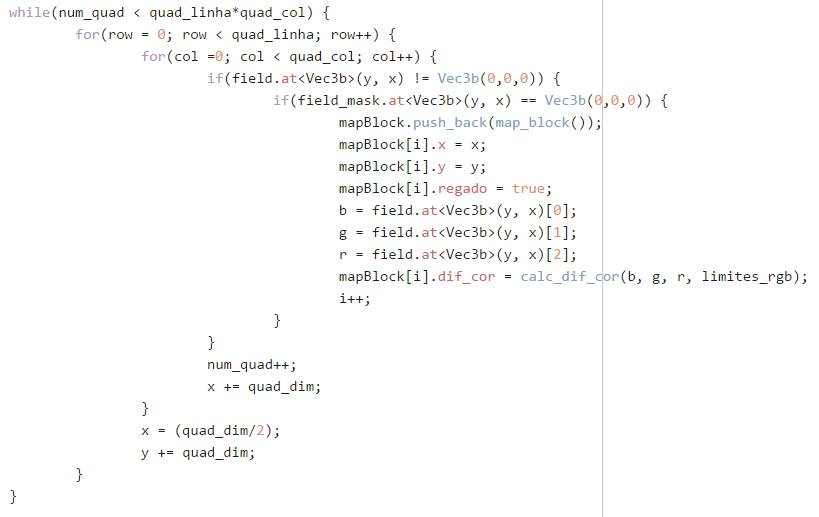
A função apply\_mask recebe como parâmetro a imagem quadriculada e o intervalo que delimita as cores que representam um bloco saudável e retorna uma imagem segmentada onde os blocos que precisam ser irrigados estão em preto, como pode ser visto na figura Z.

3.3.1.1.5 Mapeamento de Gramado Não Saudável

A função responsável por mapear o gramado não saudável recebe como parâmetro a imagem do jardim já transformada em uma imagem quadriculada, a imagem da máscara, o tamanho do lado de cada bloco, o número de linhas e de colunas da imagem, o endereço do vetor de blocos que será retornado pela função image\_processing e o intervalo que representa uma cor saudável. Sua chamada pode ser vista na figura abaixo:



Assim como a função que calcula a cor média dos pixels, mapUnhealthyGrass é composta por um laço principal e dois laços interiores encadeados. O laço principal percorre cada bloco da imagem, já os dois laços interiores percorrem os eixos X e Y da imagem. Para cada bloco, é verificado na imagem quadriculada e na máscara se o bloco precisa ser irrigado ou não. Caso seja necessário, salva-se as informações do bloco em um objeto do vetor de blocos, como pode ser visto na figura abaixo T (as variáveis x e y da função representam as coordenadas do centro de cada bloco):

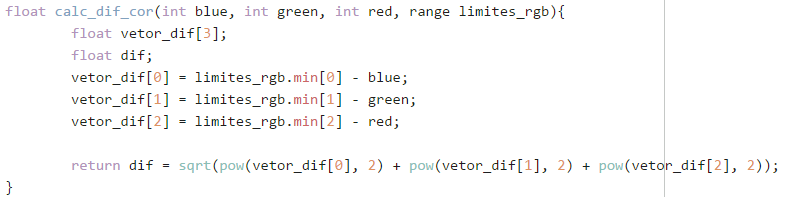


Como o argumento da função é o endereço do vetor de blocos e não uma cópia dele, não é necessário retornar nada, portanto, a função é do tipo *void*. Ao fim do laço principal, o vetor estará preenchido e o módulo de processamento de imagem se encerra.

3.3.1.1.6 Cálculo da Diferença de Cor

A função calc\_dif\_cor, vista na figura T, é responsável por calcular a diferença entre a cor de cada bloco e o ponto mínimo do intervalo que define as cores de um gramado saudável. O ponto mínimo foi escolhido pois para essa função ser chamada, a cor do bloco está dentro do intervalo e ela está abaixo da cor que é considerada saudável, portanto, optou-se por usar a cor menos saudável como referência.

O cálculo é uma conta de diferença de vetores, sendo o resultado o módulo do vetor diferença, como visto na figura abaixo M:



3.3.1.2 Tomada de Decisão

3.3.1.2.1 Captação da Previsão do Tempo

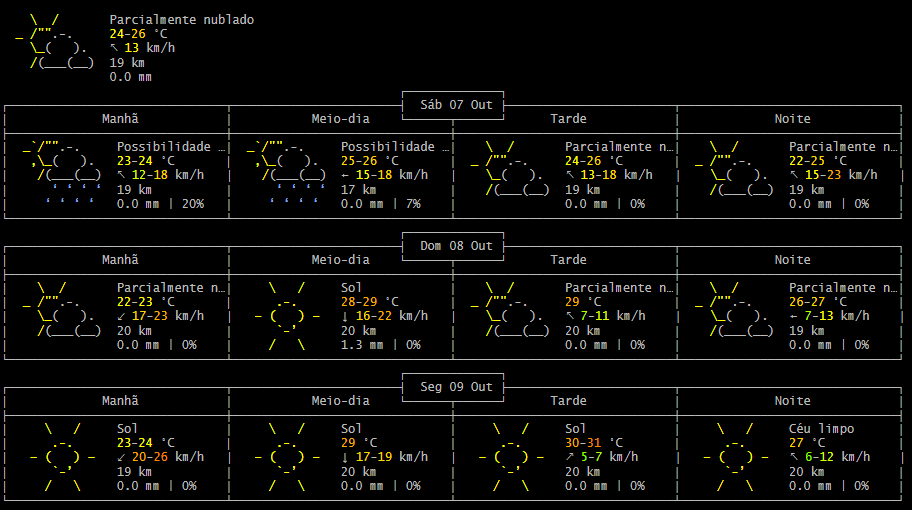
Um dos valores utilizado para calcular a quantidade de água necessária para um bloco é o coeficiente de chuva. Esse coeficiente, que tem valor entre 0 e 1, é calculado usando valores captados do site wttr.in referentes à previsão do tempo e probabilidade de chuva do dia atual e os dois seguintes. A chamada da função pode ser vista na figura abaixo:



Esses valores são obtidos através de uma chamada do sistema que faz uma requisição ao site via linha de comando, como pode ser visto na figura abaixo:



O resultado, salvo no arquivo, segue o modelo da figura abaixo:



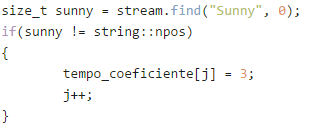
Para gerar os valores que serão utilizados na função do cálculo do coeficiente de chuva, é necessário analisar o arquivo de texto onde o resultado está salvo utilizando o método find da classe string, própria da linguagem C++.

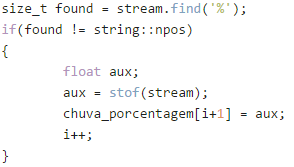
Essa análise é feita num laço que itera pelo arquivo doze veze, procurando pelas palavras Sunny, Clear, Cloudy, Overcast, Rain e Snow. No programa as palavras estão em inglês em decorrência da linguagem padrão do sistema operacional onde o projeto foi desenvolvido. Cada uma dessas palavras possui um valor que é inserido no vetor de coeficientes do clima. Esses valores são:

* Sunny (Ensolarado) = 3
* Clear (Tempo Claro) = 2
* Cloudy/Overcast (Nublado) = 1
* Rain (Chuva) = 0
* Snow (Neve) = 0

Esses valores irão formar o vetor de coeficientes do clima que serão utilizados no cálculo do coeficiente de chuva. Por fim, para cada parte do dia é salvo em um vetor a probabilidade de chuva, que também será utilizada no cálculo do coeficiente de chuva.

Nas figuras O e P abaixo, encontra-se um exemplo de como é atribuído o valor do coeficiente do clima ao vetor e como é formado o vetor de probabilidade de chuva:





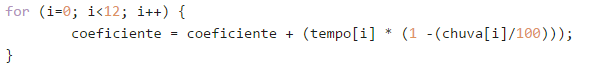
3.3.1.2.2 Cálculo do Coeficiente de Chuva

O cálculo do coeficiente de chuva é feito utilizando os valores referentes à previsão do tempo e probabilidade de chuva. Como podemos ver na figura a baixo:



A função recebe como parâmetros dois vetores, um referente à previsão do tempo e outro à probabilidade de chuva. Utilizando os valores presentes nesses vetores, calcula-se um coeficiente com valor entre 0 e 1 que mede a probabilidade de ser necessário regar as plantas. Quanto maior o valor do coeficiente, menos chuva está previsto.

A fórmula para o cálculo do coeficiente está presente na figura abaixo:



3.3.1.2.3 Captação da Velocidade do Vento

A função responsável por captar a velocidade do vento é uma função sem parâmetros que retorna um valor do tipo *float,* como podemos ver na figura abaixo:



Assim como as informações do clima, a velocidade do vento é obtida através de uma chamada do sistema que executa o programa ansiweather, como pode-se ver na figura abaixo:



O resultado é salvo num arquivo que vai ser lido pelo programa e atribuído a uma string. Nessa string será feita uma busca utilizando expressão regular para achar a velocidade do vento. A expressão regular utilizada pode ser vista na figura abaixo:



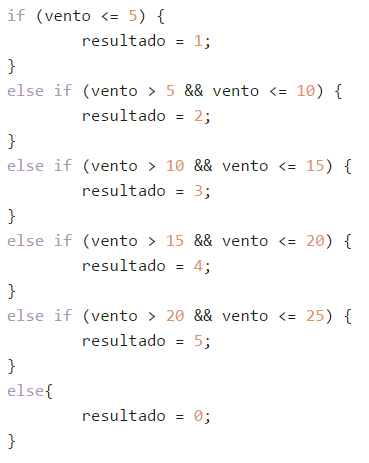
Essa expressão regular busca por números decimais ou inteiros que sejam seguidos por m/s. O valor captado pela busca é então convertido para um número do tipo *float*, multiplicado por 3.6 para obter o valor em km/h e então é retornado pela função.

3.3.1.2.4 Atribuição do Valor do Vento

Para cada elemento utilizado no cálculo do índice de necessidade de água, existe uma função responsável por captar e atribuir os seus valores. No caso do vento, essa função se chama get\_vento, como é visto na figura abaixo:



A função get\_vento chama a função find\_wind e de acordo com a velocidade do vento retornada pela função atribui-se um valor a resultado que varia entre 0 e 5, sendo que 0 significa um vento muito forte, situação em que não compensa regar o jardim, e 5 representa a velocidade máxima aceitável para se regar o jardim. Esse valor presente em resultado será o valor utilizado no cálculo do índice de necessidade de água. A divisão é feita seguindo o modelo da figura abaixo:



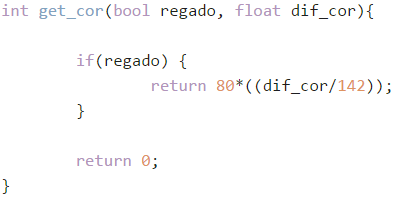
Essa divisão foi feita baseada no que diz (Kobiyama; Chaffe): “O vento modifica a camada de ar vizinho a superfície, substituindo uma camada muitas vezes saturada por uma com menor teor de vapor da água. Portanto, quanto maior a intensidade do vento, maior a intensidade de evaporação. ”. Portanto, quanto maior a velocidade do vento, maior é a necessidade de água. Essa relação pode ser vista na figura U.

3.3.1.2.5 Atribuição do Valor da Umidade

3.3.1.2.6 Atribuição do Valor da Insolação

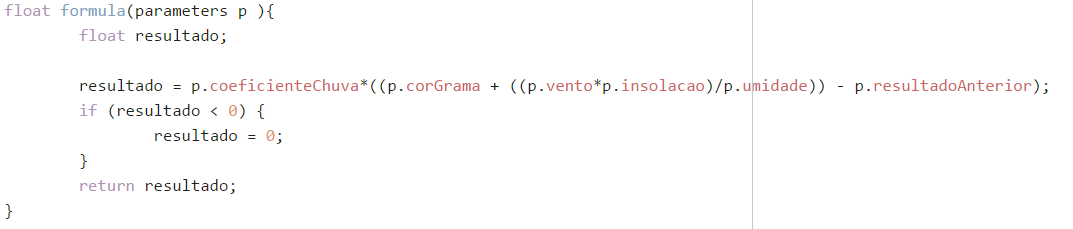
3.3.1.2.7 Atribuição do Valor da Cor

O valor numérico atribuído à cor do bloco é um cálculo feito utilizando o resultado da função calc\_dif\_cor, vista na figura M. Esse resultado é dividido pela diferença máxima do intervalo determinado na função image\_processing (resultado da diferença entre o ponto mínimo do intervalo e o ponto máximo), gerando um coeficiente que é multiplicado pelo valor 80. Esse valor foi escolhido para complementar o resultado da equação entre a umidade do solo, insolação e velocidade do vento, que tem como valor máximo 20. Caso o bloco não tenha necessidade de ser regado, não é feito o cálculo e o retorno é 0. A implementação da função pode ser vista na figura abaixo:



3.3.1.2.8 Cálculo do Índice de Necessidade de Água

Para o cálculo do índice de necessidade de água, a função formula recebe os parâmetros do dia num objeto da *struct* parameters e aplica os valores na fórmula, como pode ser visto na figura abaixo U:

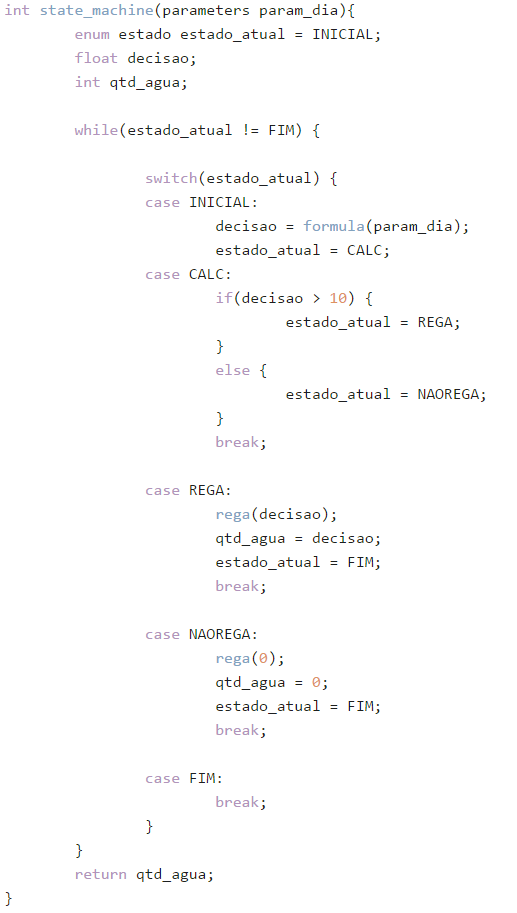


Essa fórmula foi elaborada considerando que a velocidade do vento e a insolação, segundo (Kobiyama; Chaffe), aumentam a taxa de evaporação da água no solo, portanto, seus valores são diretamente proporcionais a necessidade de água. Já a umidade do solo, quanto maior for, menos água é necessário, por isso sua contribuição é inversamente proporcional à necessidade de água. A cor da grama, calculada na função get\_cor, representa a saúde do gramado e, portanto, seu valor tem grande influência no resultado da fórmula. Por fim, o resultado anterior é utilizado para evitar que caso um bloco seja regado com muita água em um dia, no próximo seja utilizado mais água do que o necessário. Caso o resultado anterior seja maior que o resultado da primeira equação, o resultado final é considerado como 0.

3.3.1.2.9 Tomada de Decisão sobre Bloco

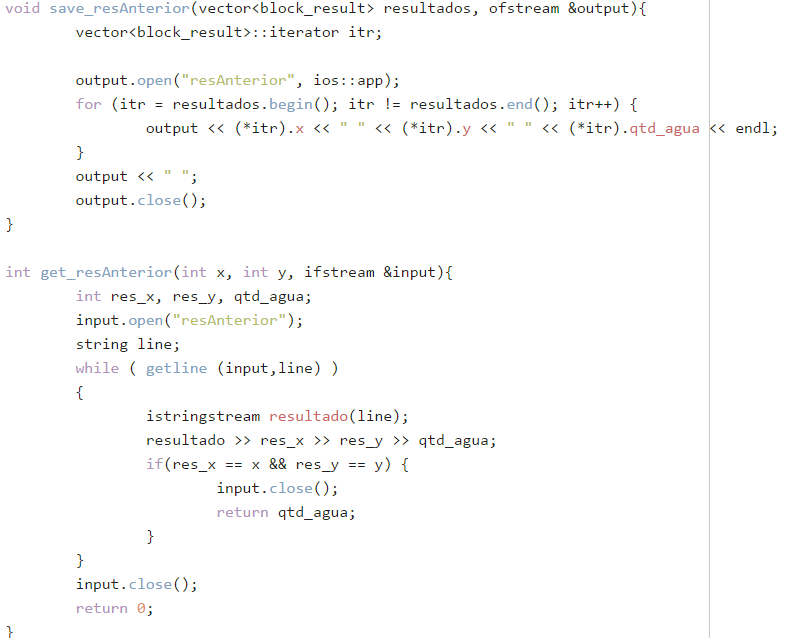
A tomada de decisão sobre um bloco ser irrigado ou não é feita através de uma máquina de estados simples, que possui quatro estados: INICIAL, estado onde é feito o cálculo do índice da necessidade de água; NAOREGA, estado de decisão que diz para o bloco não ser regado; CALC, estado onde é tomada a decisão. Caso o resultado do índice seja maior do que 10, o bloco será irrigado, caso contrário, não; REGA, estado de decisão que diz para o bloco ser regado e define a quantidade de água a ser utilizada; FIM, estado final.

A máquina de estados pode ser vista na figura abaixo:



3.3.1.2.10 Leitura e Armazenamento do Resultado Anterior em Arquivos

As últimas funções referentes ao módulo de tomada de decisão são as funções responsáveis por ler e salvar os resultados em um arquivo. O arquivo, chamado de resAnterior possui em cada linha as coordenadas X e Y do bloco e a quantidade de água utilizada para regá-lo. O processo de leitura e armazenamento dos dados é feito linha a linha, como pode-se ver na imagem abaixo:



3.3.1.3 Aspersores

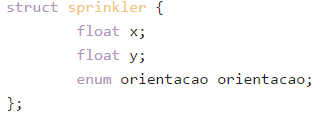
3.3.1.3.1 Leitura de Posição dos Aspersores

A posição dos aspersores no jardim é definida manualmente, portanto, para o programa saber onde estão localizados os aspersores, é necessário inserir manualmente em um arquivo suas posições, na ordem coordenada X coordenada Y. Cada linha representa um aspersor.

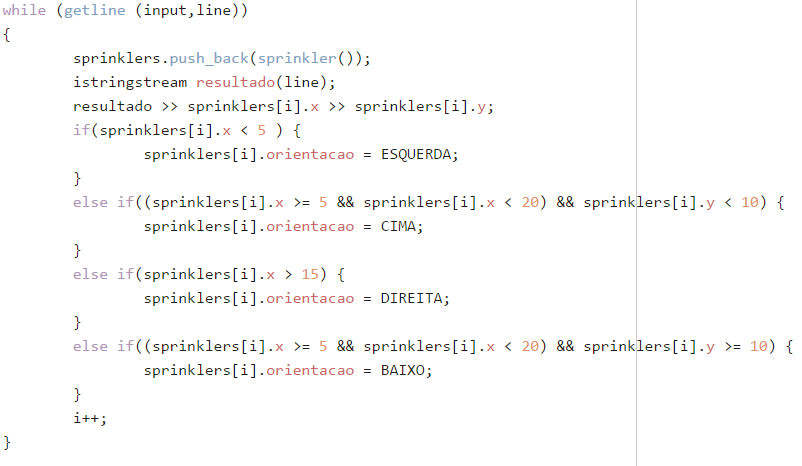
Como pode-se ver na figura abaixo, a função responsável pela leitura da posição dos aspersores retorna um vetor com todos os aspersores e suas coordenadas além da orientação:



A *struct* sprinkler (aspersor em inglês), possui três campos: coordenada x, coordenada y e orientação, como pode ser visto na figura abaixo:



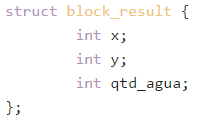
Existem quatro orientações possíveis: CIMA, BAIXO, ESQUERDA e DIREITA. A orientação do aspersor é definida na hora da leitura do arquivo, de acordo com sua posição no jardim. A figura L abaixo ilustra a decisão:



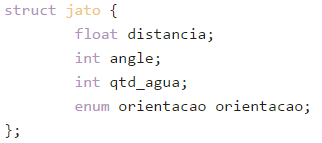
3.3.1.3.2 Modelagem do Jato D’água

Para molhar cada bloco do jardim, o microcontrolador responsável por controlar o aspersor precisa saber as coordenadas do bloco, a distância do aspersor até ele, o ângulo entre os dois e a quantidade de água a ser utilizada. A função responsável por agrupar as informações é a get\_jato, como na figura abaixo:

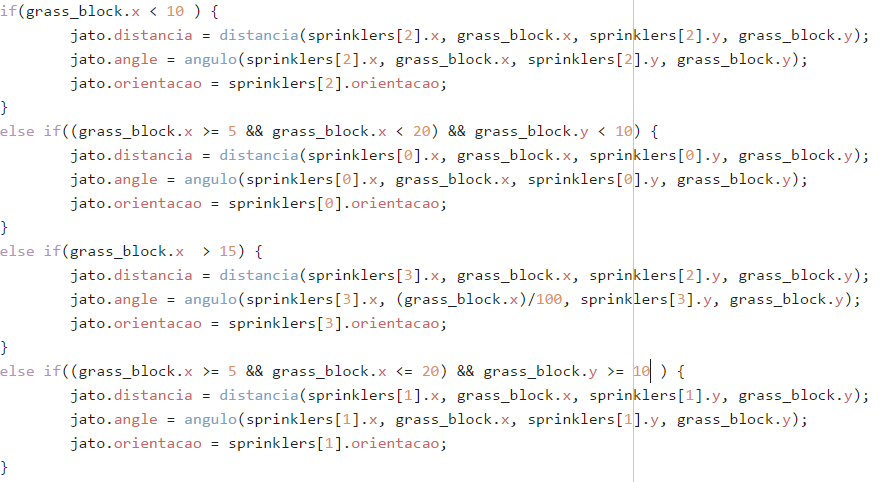
 As coordenadas do bloco são passadas por parâmetro através de um objeto da *struct* block\_result, que pode ser vista na figura abaixo:



E o resultado, um objeto da *struct* jato, contendo todas as informações, tem sua estrutura representada na figura abaixo:



A detecção de qual aspersor deverá ser utilizado é feita através das coordenadas do bloco, como pode ser visto na figura abaixo:



A lógica para selecionar o aspersor é a mesma utilizada para determinar a orientação do aspersor lido do arquivo, como pode ser visto na figura L.

3.3.1.3.3 Cálculo da Distância entre Aspersor e Bloco

A distância entre o aspersor e o bloco é calculada através de uma equação de distância entre dois pontos em uma reta. A função, que recebe as coordenadas como parâmetro, como pode ser visto na figura abaixo:



A implementação da função é uma transcrição da equação abaixo:

3.3.1.3.4 Cálculo do Ângulo entre Aspersor e Bloco

Para o cálculo do ângulo entre o aspersor e o bloco, a função ângulo recebe as coordenadas assim como a função para calcular a distância entre os dois, como está ilustrado na figura abaixo:



A implementação da função é baseada na equação abaixo:

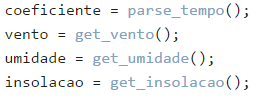
Onde:

Para obter o valor em graus ao invés de radianos é feita uma conversão antes de retornar o resultado, como pode ser visto na figura abaixo:



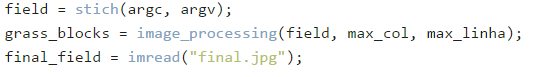
3.3.1.4 Núcleo (Esperar meia noite faltando)

A função principal do programa é a *main* presente no arquivo core.cpp. Ela é responsável por chamar todos os módulos e interliga-los. A primeira etapa é coletar os dados do dia, como na figura abaixo:



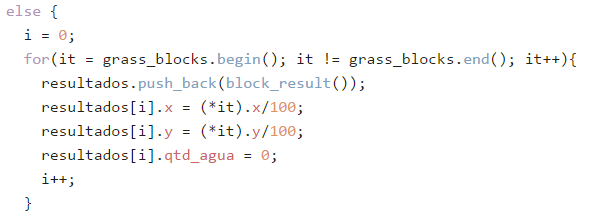
Essas operações são feitas fora do laço principal pois elas só precisam ser feitas uma vez ao dia, já que o valor é o mesmo para o jardim todo.

Após a coleção de dados, é chamado o fluxo do processamento de imagens, responsável por formar a panorâmica do jardim e retornar um vetor contendo as informações de todos os blocos do jardim. Essa chamada é feita através das funções stitch e image\_processing, como pode ser visto na figura abaixo:

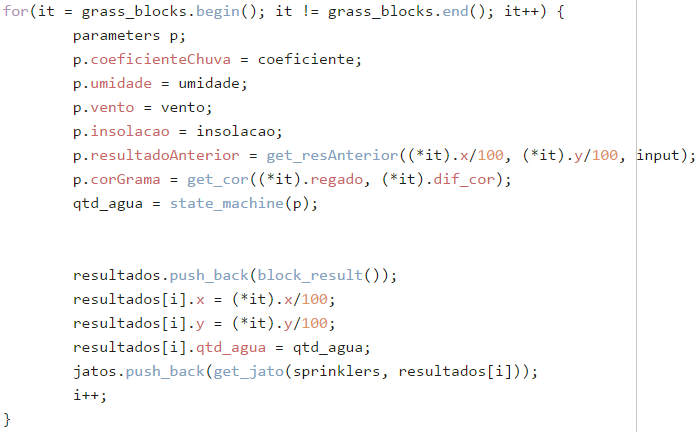


Por fim são lidos os aspersores do arquivo sprinklerpos.txt antes do laço principal.

Antes do laço principal, é feita uma verificação, caso o coeficiente de chuva seja menor que 10%, significa que irá chover e, portanto, não é necessário regar o jardim. É feito então uma iteração sobre os blocos presentes em grass\_blocks para obter suas coordenadas e então é definido o vetor de resultados, como na figura abaixo:



Caso o coeficiente de chuva seja maior que 10%, o laço principal itera sobre o vetor grass\_blocks, criando um objeto do tipo parameters temporário, que é usado para a tomada de decisão. A tomada de decisão retorna a quantidade de água a ser utilizada e então essa quantidade de água é salva no vetor de resultados que será utilizado para definir a direção do jato d’água. A figura abaixo mostra o fluxo do laço principal:



Ao final da execução do laço, remove-se o arquivo de resultados anteriores para salvar o novo, como na figura abaixo:



E então termina a execução do programa, que aguarda até o próximo dia para ser executado novamente.