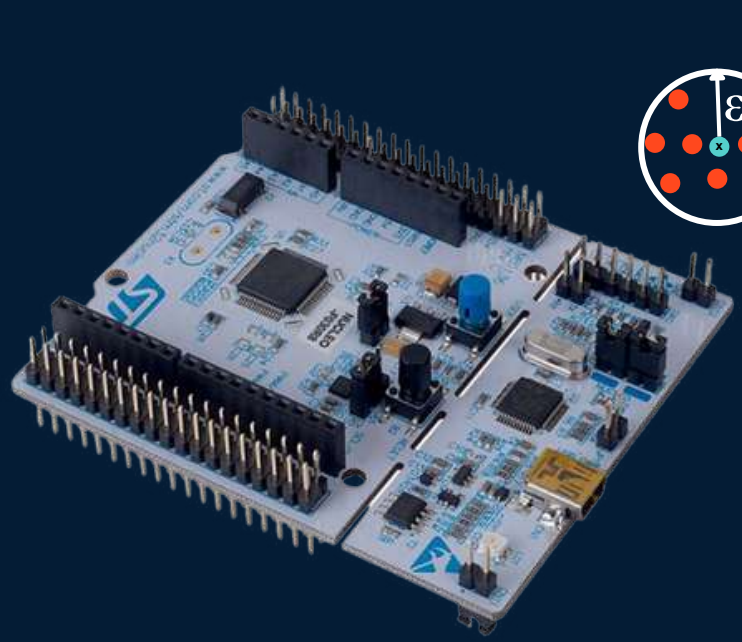
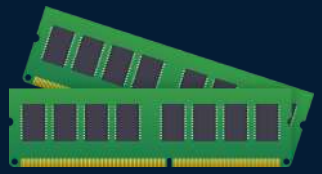
Algoritmo DBSCAN

Medições e melhorias





Vinícius Menezes Monte

Paulo Diego De Meneses

Otimização Proposta

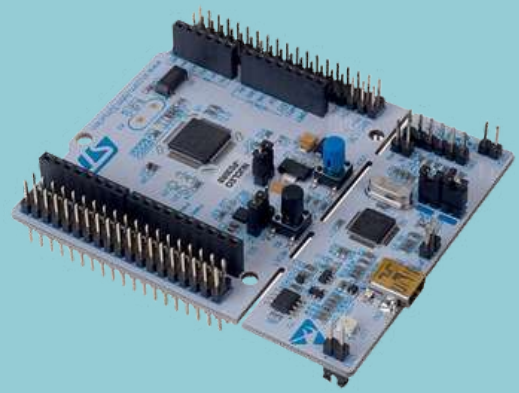
Tempo

Nosso algoritmo tem complexidade temporal $O(n^2)$

Em sua versão pioneira, esse algoritmo é lento, porém há espaço para melhorias tanto no tempo assintótico, quanto no absoluto.

Memória

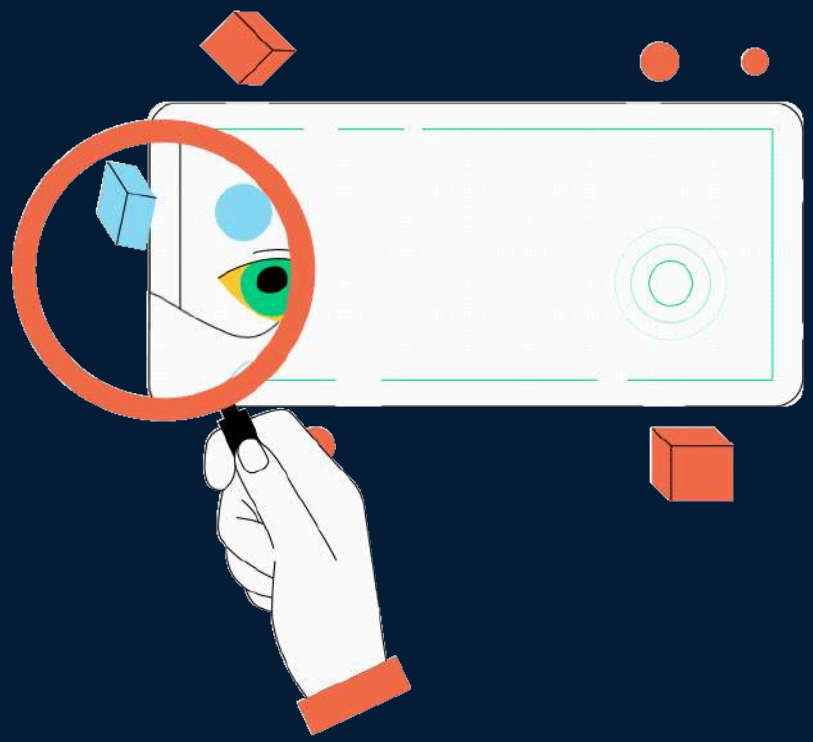
Nosso algoritmo usava muita memória RAM do embarcado.



Então melhoramos a tipagem de dados para que ele fosse embarcado e apresentado no T2.

Busca pela parte critica

O Algoritmo possui uma parte executada sempre. Após identificarmos essa parte, podemos começar a testar uma melhoria.



O Cálculo da Distância

Chegamos a uma conclusão que o cálculo da distância entre os pontos, estava fazendo computação desnecessária.

Usávamos, à risca, a fórmula da distância euclidiana:

$$\sqrt{(x_{i1} - x_{i2})^2 + (y_{i1} - y_{i2})^2} \leqslant eps$$

Da seguinte forma no código:

```
coord_t sum_squared = dx * dx + dy * dy;
if (sqrtf(sum_squared) <= eps) {
    neighbors[count++] = i;
}
```

Melhoria I

O cálculo da raiz quadrada usando o método “sqrtf” da biblioteca math.h tem complexidade temporal O(log(n))

Podemos fazer a verificação comparando apenas os quadrados:

$$(x_{i1} - x_{i2})^2 + (y_{i1} - y_{i2})^2 \leqslant eps^2$$

Fizemos o código da seguinte forma:

```
coord_t eps_squared = eps * eps;
coord_t dist_squared = dx * dx + dy * dy;
if (dist_squared <= eps_squared) {
    neighbors[count++] = i;
}
```

Melhoria II

A essa altura, o nosso algoritmo teve uma redução no tempo de 50%. Mas ainda assim, pensamos em usar uma estratégia de bounding box. Dessa forma fazemos checagens unidimensionais.
Nesse caso, um ponto é válido se atender a 3 condições:

$$|x_{i1} - x_{i2}| \leqslant eps$$

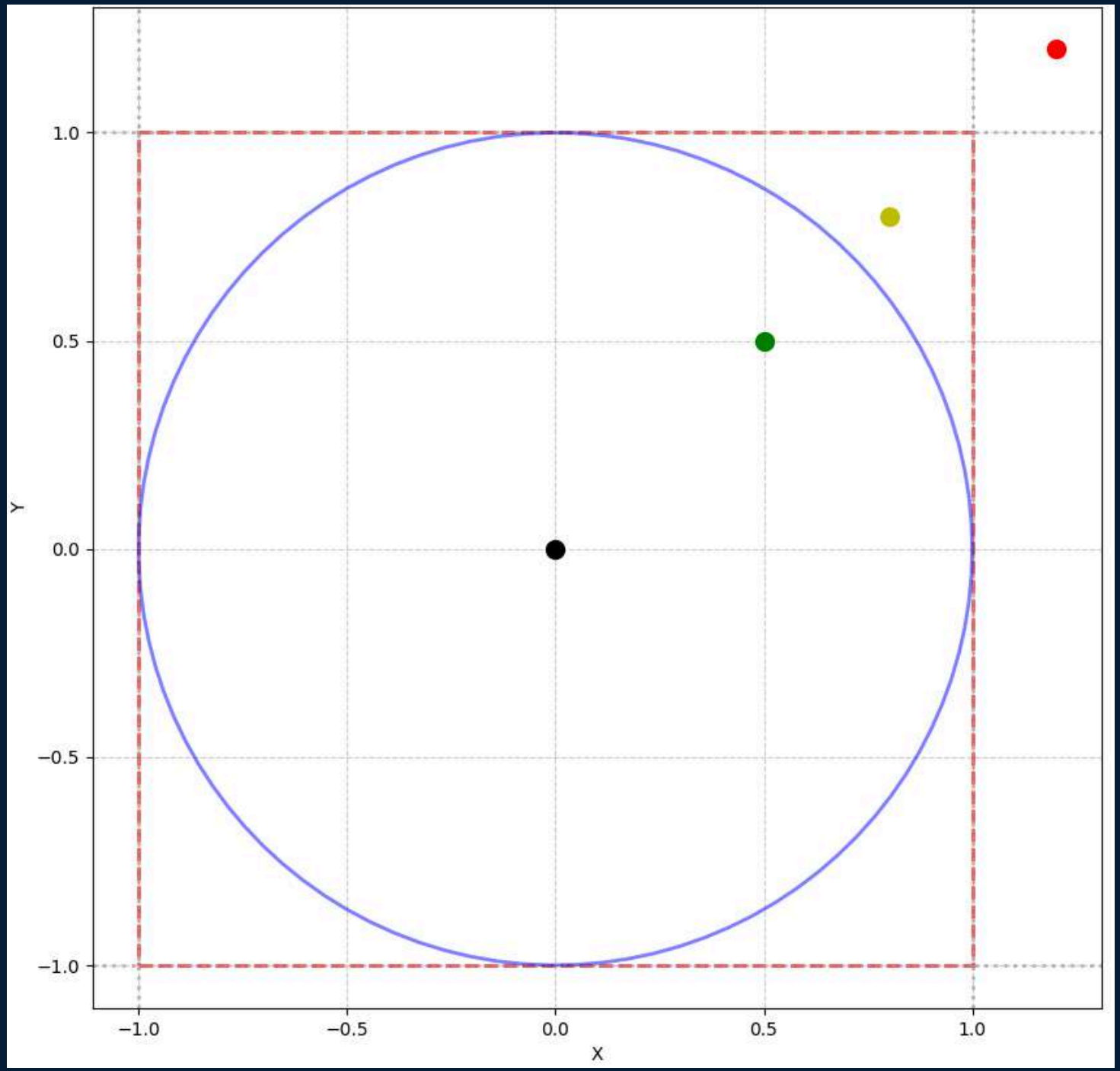
$$|y_{i1} - y_{i2}| \leqslant eps$$

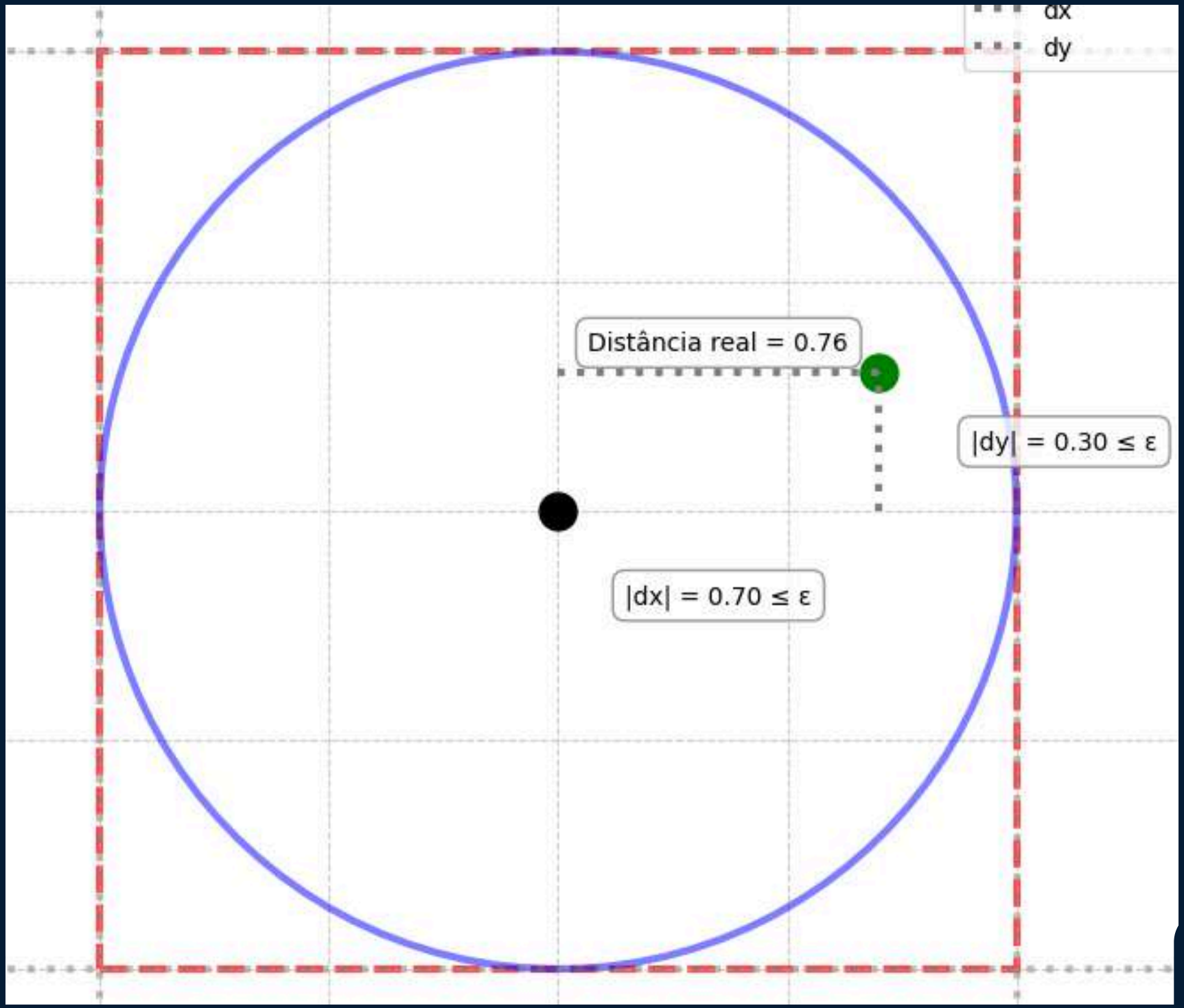
$$(x_{i1} - x_{i2})^2 + (y_{i1} - y_{i2})^2 \leqslant eps^2$$

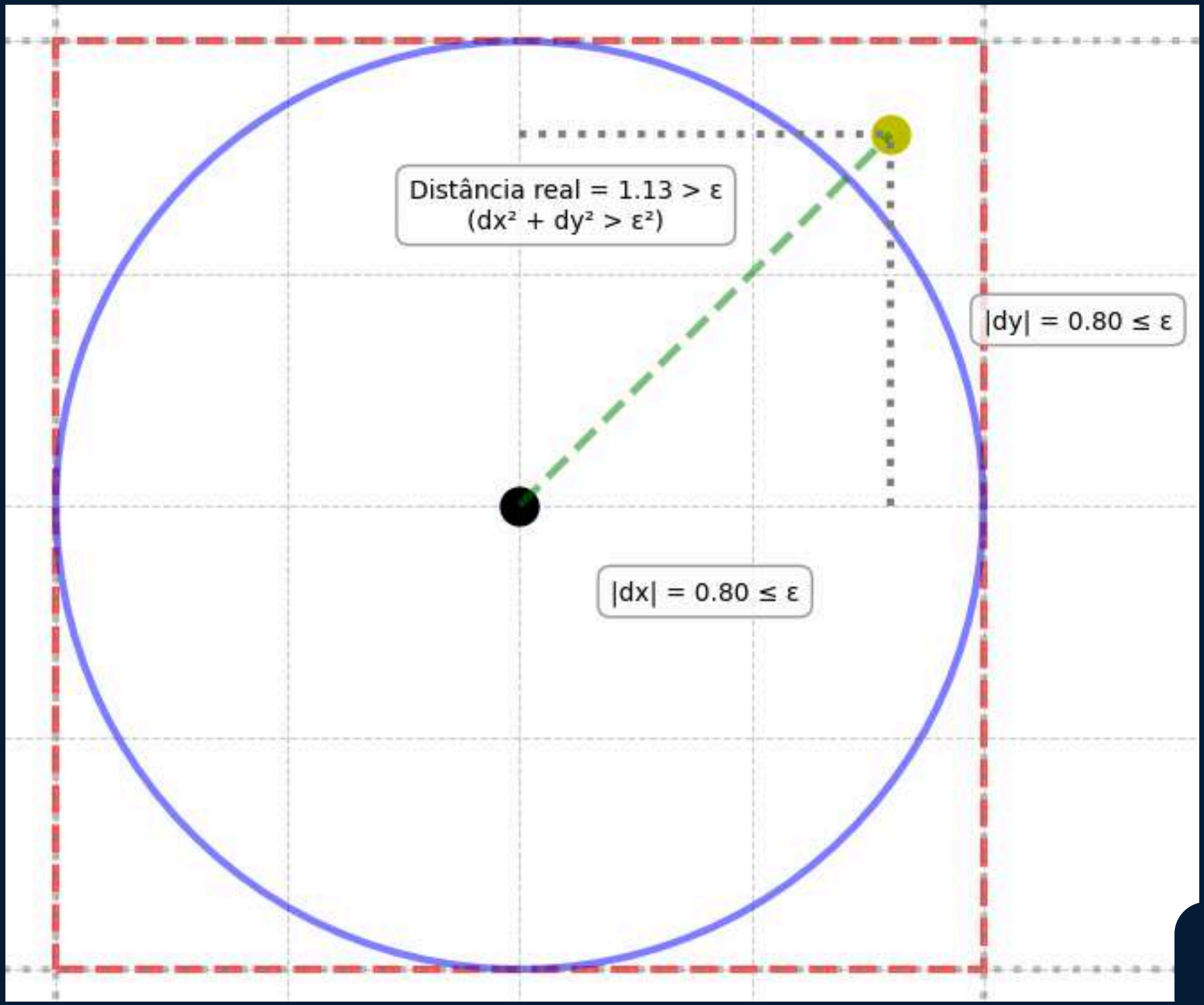
Melhoria II

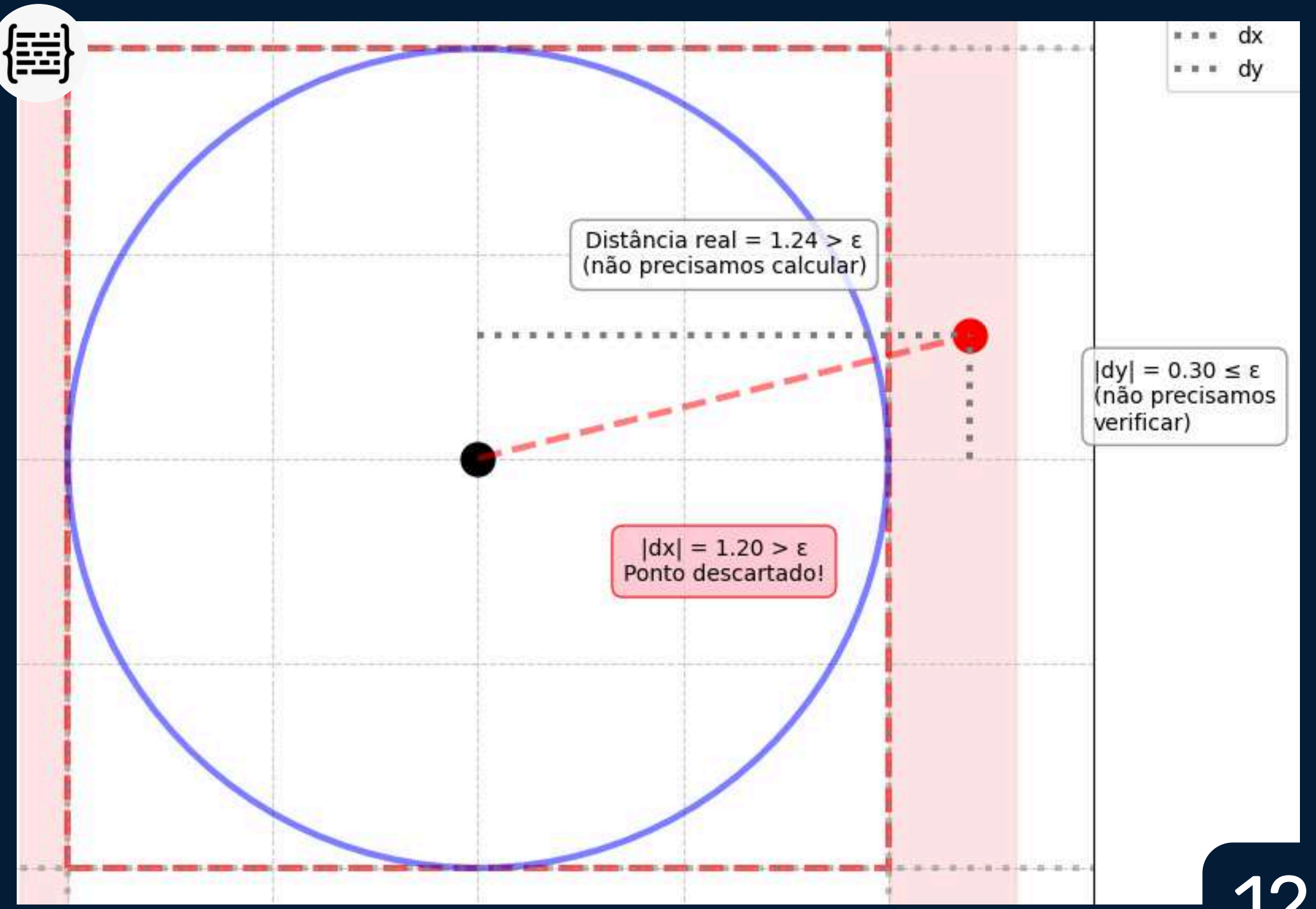
No código:

```
coord_t dx = point[0] - points[i][0];
if (fabs(dx) > eps) {
    continue;
}
coord_t dy = point[1] - points[i][1];
if (fabs(dy) > eps) {
    continue;
}
coord_t dist_squared = dx * dx + dy * dy;
if (dist_squared <= eps_squared) {
    neighbors[count++] = i;
}
```


1.0
0.5
0.0
−0.5
−1.0
Y
−1.0
−0.5
0.0
0.5
1.0
X


dx
dy
Distância real = 0.76
|dy| = 0.30 ≤ ε
|dx| = 0.70 ≤ ε


Distância real = 1.13 > ε
(dx² + dy² > ε²)
|dy| = 0.80 ≤ ε
|dx| = 0.80 ≤ ε


dx
dy
Distância real = 1.24 > ε
(não precisamos calcular)
|dy| = 0.30 ≤ ε
(não precisamos
verificar)
|dx| = 1.20 > ε
Ponto descartado!

Medições

Plataforma Desktop

Notebook Avell A70

- **Familia:** 12th Gen Intel(R) Core(TM) i7
- **RAM:** 32 GB
- **Armazenamento:** 512 GB SSD
- **Frequência:** até 2.30 GHz

Espaço Utilizado

Usamos o comando “size” do Windows para medir o tamanho do algoritmo. OBS: Sem os dados de entrada.

```
 text    data     bss     dec     hex filename
16908    1640     112   18660    48e4 .\a.exe
```

```
 text    data     bss     dec     hex filename
16492    1680     112   18284    476c .\a.exe
```

- Text -> Armazenamento de código.
- Data -> Dados já inicializadas no código.
  - Array de pontos
- BSS -> Dados não inicializados.

Tempo no Avell

Considerando 10 amostras de tempo.

	Tempo Médio	Desvio Padrão
Antes Das Melhorias	0,116200 Segundos	0,00677
Melhoria I	0,012700 segundos	0,000658
Melhoria II	0,003200 Segundos	0,0006

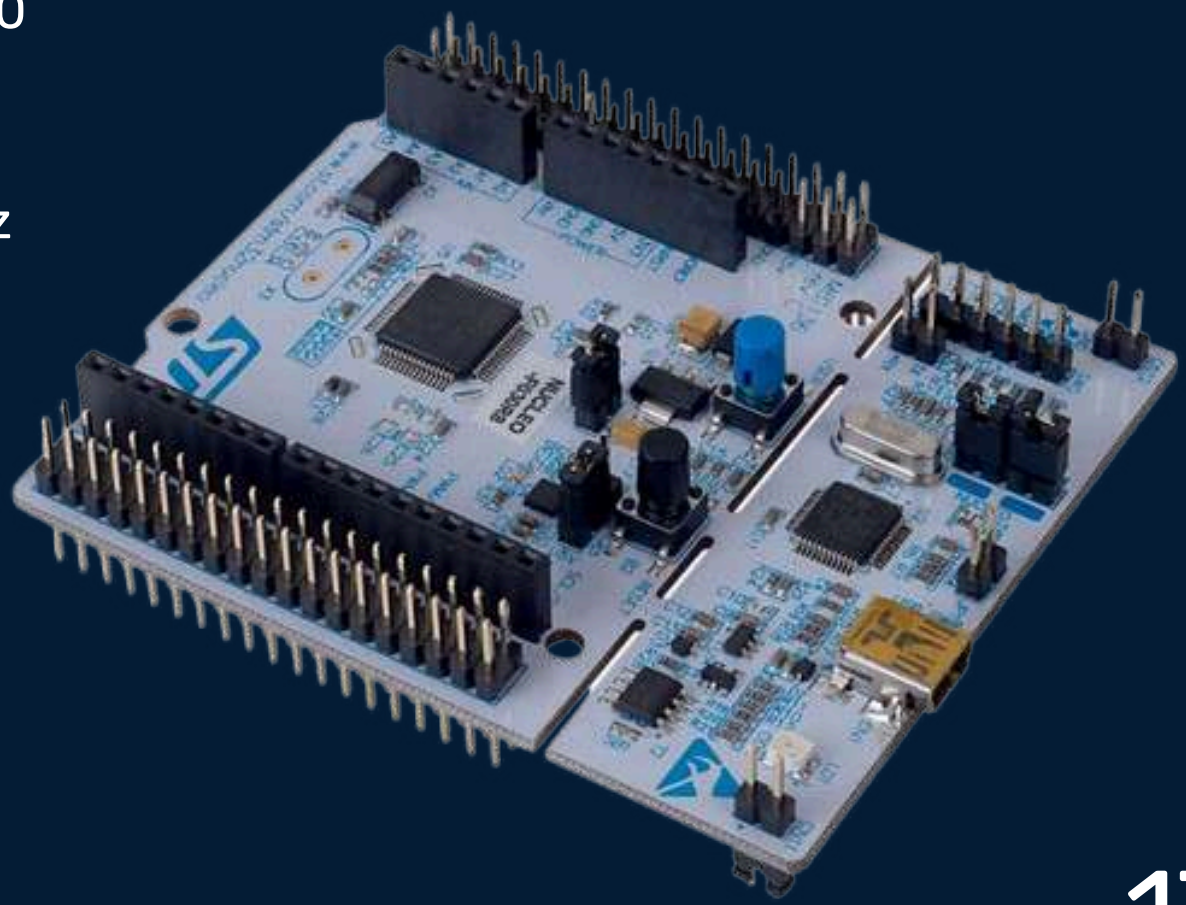
Redução de 97% no tempo.

$$Predu = \frac{t_{ant} - t_{novo}}{t_{antigo}} \times 100\%$$

Plataforma Embarcada

STM32F030R8

- **Familia:** ARM Cortex-M0
- **RAM:** 8 KiB
- **Flash:** 64 KiB
- **Frequência:** até 48MHz



Espaço Utilizado

Usamos a saída do compilador do Keil Studio.

	Antes das melhorias	Depois das melhorias
Flash/ROM Com Dados de entrada	31 KB	31 KB
RAM com dados de entrada (Mil pontos)	4 KB	4 KB
Flash/ROM Sem Dados de entrada	22 KB	22 KB
RAM sem dados de entrada	980 B	980 B

Tempo no STM32

Considerando 10 amostras de tempo e precisão de microsegundos.

	Tempo Médio	Desvio Padrão
Antes Das Melhorias	18,240990 Segundos	0,00000590
Melhoria I	9,038972 segundos	0,00000358
Melhoria II	3,437743 Segundos	0,00000689

Redução de 81% no tempo.

$$Predu = \frac{t_{ant} - t_{novo}}{t_{antigo}} \times 100\%$$

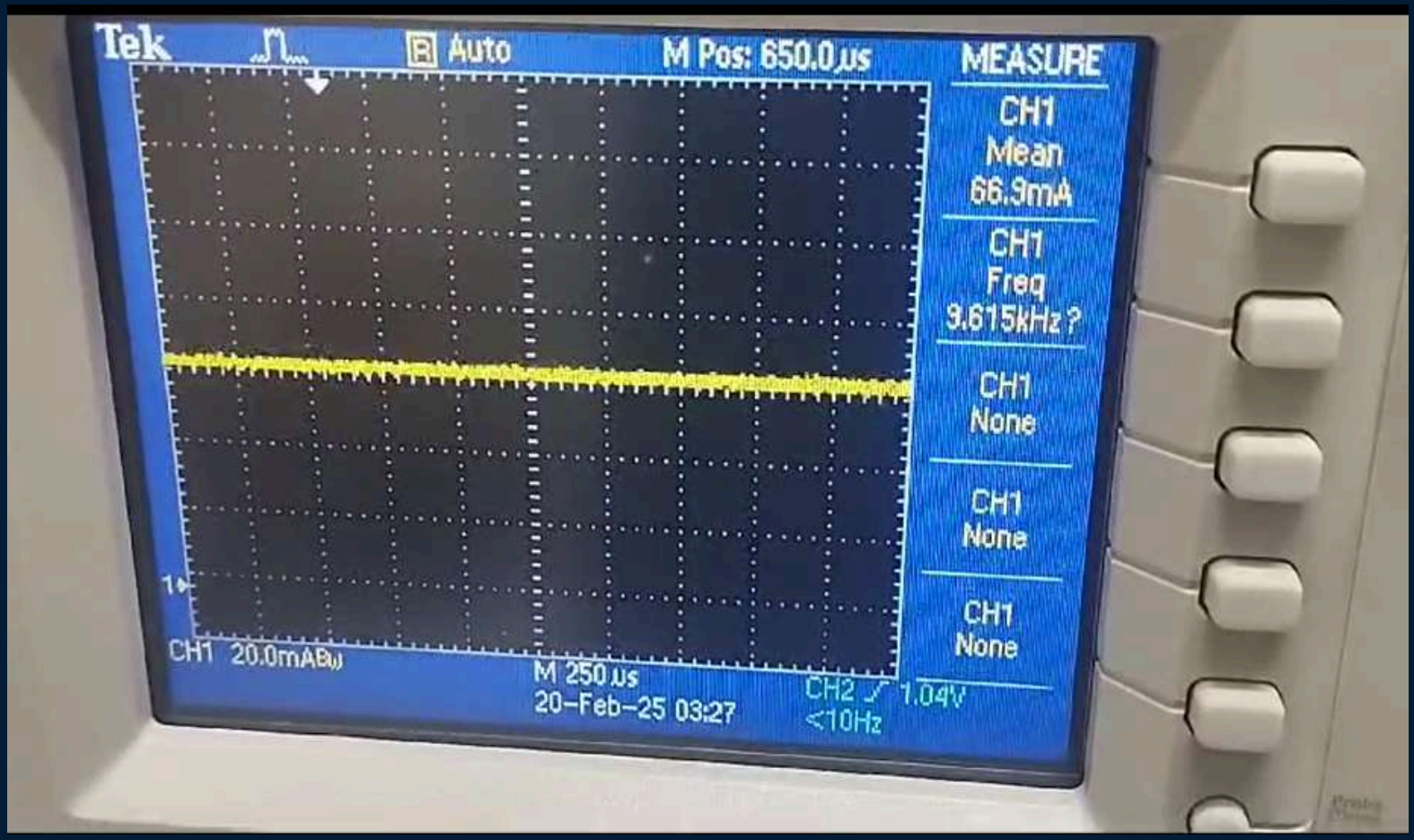
Corrente no STM32

Corrente antes e depois das melhorias.

	Corrente Médio	Desvio Padrão
Antes Das Melhorias	66,48 mA	0,736
Melhoria II	66,66 mA	0,973

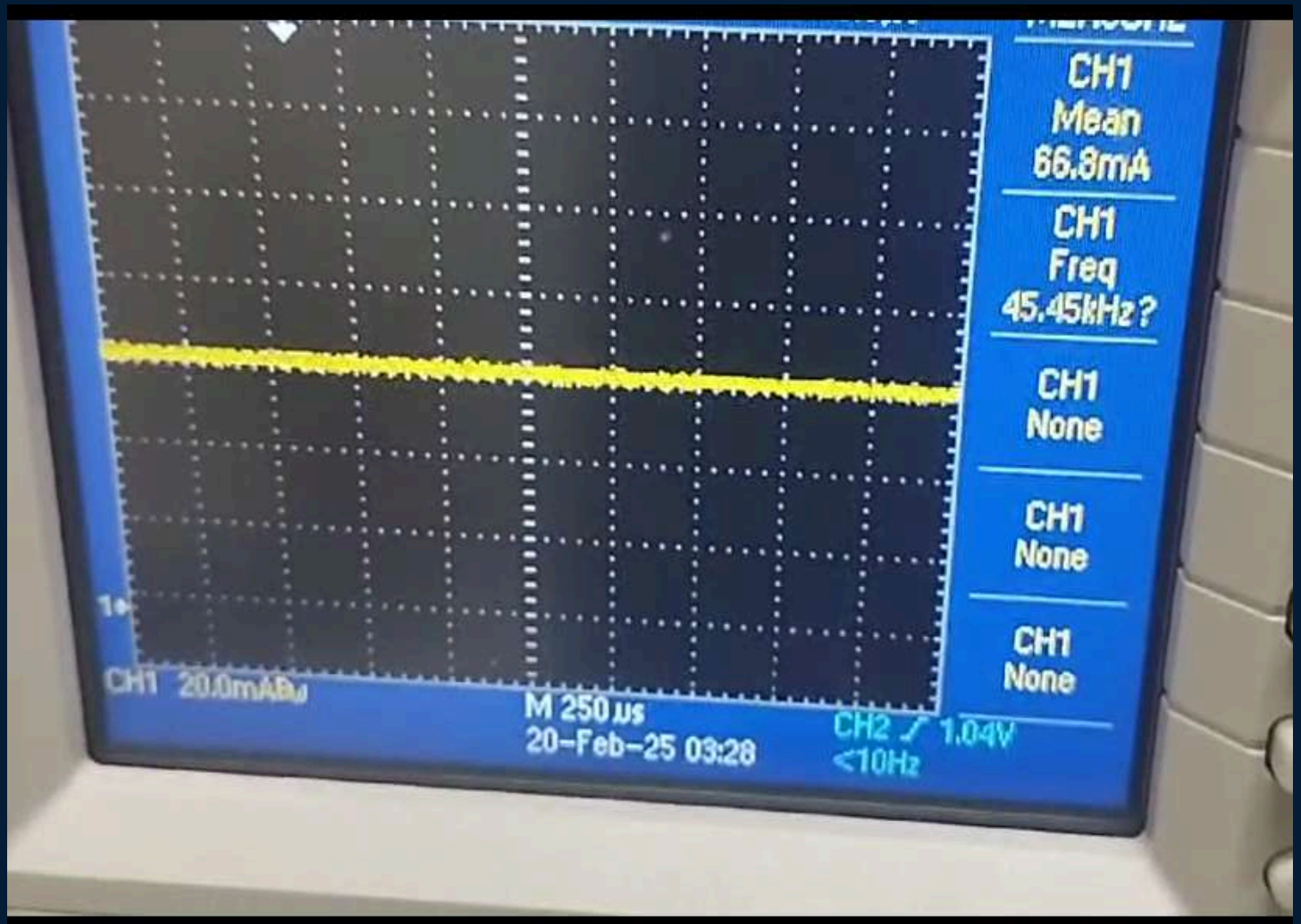
Corrente no STM32

Antes das melhorias.



Corrente no STM32

Depois das melhorias.



Desafios Técnicos

Como medir tempo?

Não sabíamos nem por onde começar, até achar na documentação do MBED OS, e no stack overflow.
Para o desktop usamos a biblioteca time.h, e o método clock:

```
clock_t start = clock();
dbscan(points, &set, set.size, eps, minPts);
clock_t end = clock();
double execution_times = (double)(end - start) / CLOCKS_PER_SEC;
```

Para o stm32 usamos a biblioteca mbed_stats.h, e a classe Timer:

```
Timer timer;
timer.start();
dbscan(points, &set, set.size, eps, minPts);
timer.stop();
uint32_t tempo_us = timer.read_us();
```

Como medir espaço?

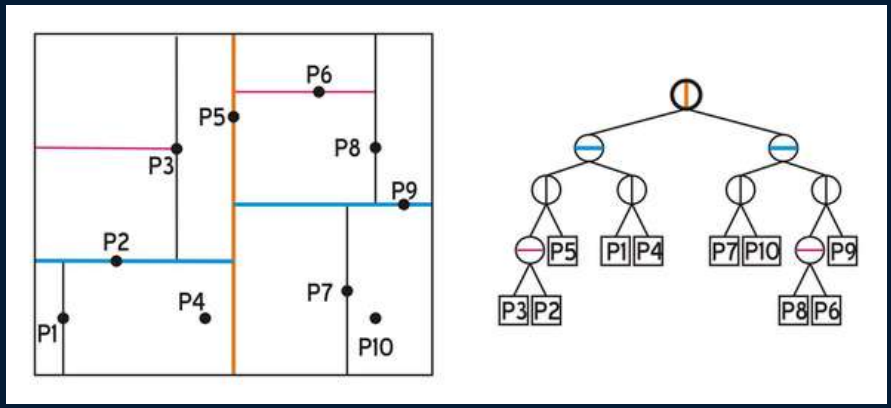
Para sistemas diferentes usamos métodos diferentes.
Para o windows, usamos o comando “size”, que já vem nativo. Mas para o sistema embarcado, tivemos que usar outras ferramentas, como o comando:

- arm-none-eabi-size

Só então descobrimos que o tamanho do binário é o que já estava sendo mostrado na IDE como Flash.

Embarcar KD-Trees

O algoritmo de Árvores de K-Dimensionais era perfeito para o problema, pela sua natureza espacial.



$$n^2 \rightarrow nlog\,(n)$$

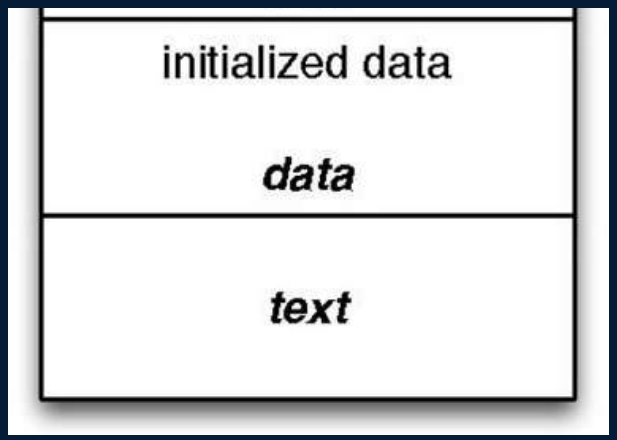
A operação crucial do algoritmo é encontrar vizinhos próximos. A distância euclidiana leva tempo quadrático, enquanto o uso de KD-Trees poderia reduzir para tempo linearítmico, uma melhoria importante. No entanto, a memória disponível não foi suficiente para essa implementação. **Isso nos levou a refinar a distância euclidiana.**

Aprendizados

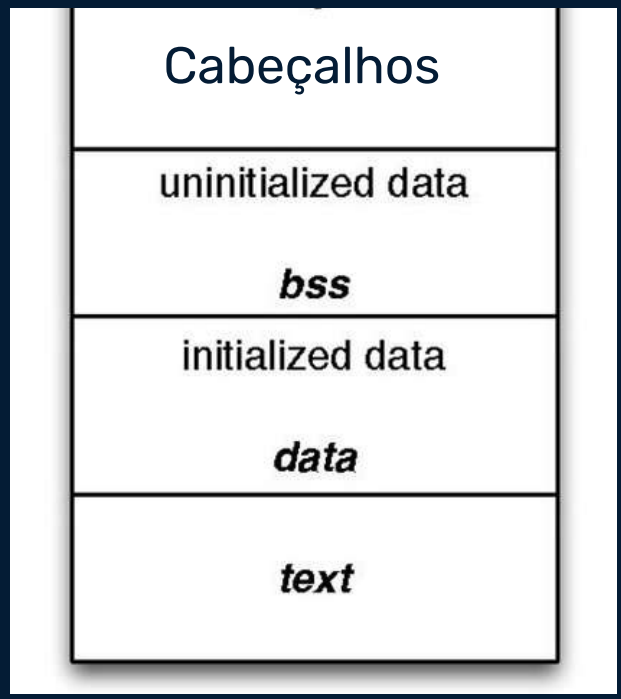
Bin X Elf

Elf é como um executável com cabeçalhos, já o .bin é o programa compilado puro, com apenas o necessário. O .elf também permite debug de uma aplicação embarcada.

bin



elf



A estrutura de dados certa pode ser errada

Ou então, “Otimizar o tempo pode prejudicar a memória”.
Certamente não foi o resultado que esperei quando fui otimizar com uma estrutura de dados mais adequada ao problema.

Nem sempre se pode ter tudo.

Bibliografia

ARM Keil Studio:

- https://studio.keil.arm.com/

Mbed OS

- https://os.mbed.com/docs/mbed-os/v6.16/reference/index.html
- https://os.mbed.com/handbook/Timer

STM32F030R8

- https://www.st.com/en/microcontrollers-microprocessors/stm32f030r8.html

Outros

- https://manpages.debian.org/testing/binutils-arm-none-eabi/arm-none-eabi-size.1.en.html
- https://learn.microsoft.com/pt-br/windows-hardware/customize/desktop/unattend/microsoft-windows-setup-diskconfiguration-disk-createpartitions-createpartition-size
- https://petbcc.ufscar.br/timefuncoes/
- https://stackoverflow.com/questions/3557221/how-do-i-measure-time-in-c

PERGUNTAS?

OBRIGADO!