



Instituto Tecnológico de Aeronáutica – ITA  
CES-35

## Lab 2:

# Construindo uma aplicação em Rede

### Membros da equipe:

Bryan Diniz Borck  
Rodrigo Macedo Rios

Prof. Dra. Cecília de Azevedo

03/09/2023

# Sumário

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>A solução</b>	<b>3</b>
2.1	O Protocolo . . . . .	3
2.2	Implementação . . . . .	6
2.3	Uso . . . . .	11
<b>3</b>	<b>Testes</b>	<b>12</b>
<b>4</b>	<b>Possíveis Melhorias e Comentários Gerais</b>	<b>15</b>
4.1	Melhorias . . . . .	15
4.2	Comentários . . . . .	15

## Lab 2: Construindo uma aplicação em Rede

### 1 Introdução

A aplicação selecionada envolve a criação de um protocolo de comunicação entre servidor e um drone (podendo ser adicionado múltipla conexão no futuro), visando coordenar movimentos e evitar colisões. Essa escolha é motivada pela necessidade de apresentar um projeto claro para o exame final, contribuindo para a compreensão da segmentação do projeto final.

A relevância dessa aplicação reside na criação de um protocolo eficiente para operações com um drone tático, coordenando missões específicas dado um alvo final como objetivo.

No cenário específico, busca-se criar um protocolo que permita que um servidor identifique um drone, receba dados de posição e velocidade e emita mensagens de reposicionamento do alvo final do drone. Objetivo é que dado um alvo, o servidor coordene o drone para chegar no alvo estipulado

Os resultados esperados incluem o servidor obtendo identificação do drone e realizando autorização dele, seguido da solicitação contínua de informações de posição do drone no espaço e ajustando movimentos conforme necessário para que alvo seja atingido, confirmado por mensagens de retorno.

## 2 A solução

### 2.1 O Protocolo

O Protocolo escolhido foi o **Protocolo de Controle de Transmissão (TCP)**, devido à confiabilidade da conexão, uma vez que para controle dos drones é necessário que o client siga rigorosamente o que é transmitido pelo server, que realiza o controle da aplicação. Ademais, a ordem das mensagens não parece importar rigorosamente, uma vez que o controle pode ser realizado a partir do recebimento da mensagem apenas, e não da sua ordem.

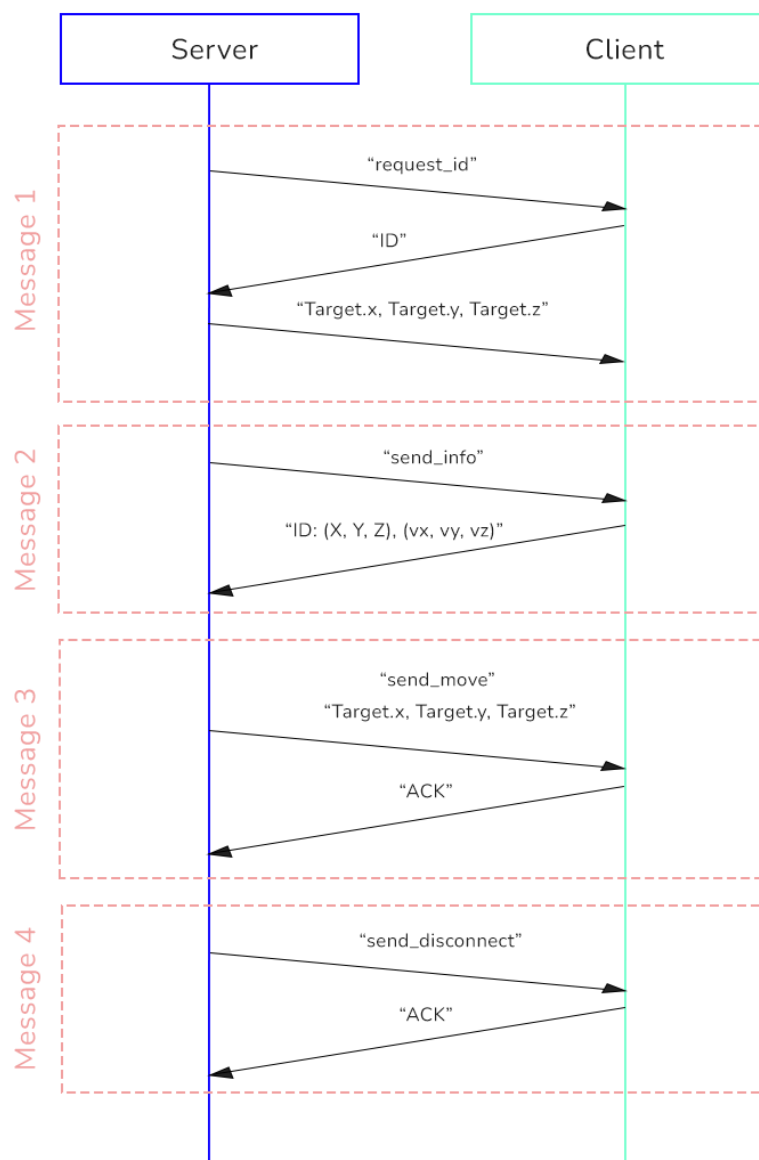


Figura 1: Conexão TCP para a aplicação

Primeiramente, para melhor entendimento da troca de mensagens, seguem structs utilizadas no código:

- Do lado do Client:
  - `drone.Drone_ID` - %d
  - `drone.Drone_position.x_coordinate` - %f
  - `drone.Drone_position.y_coordinate` - %f
  - `drone.Drone_position.z_coordinate` - %f
  - `drone.Drone_speed.x_coordinate` - %f
  - `drone.Drone_speed.y_coordinate` - %f
  - `drone.Drone_speed.z_coordinate` - %f
  - `drone.Drone_target.x_coordinate` - %f
  - `drone.Drone_target.y_coordinate` - %f
  - `drone.Drone_target.z_coordinate` - %f
- Do lado do server (além de algumas informações do drone):
  - `target.target.x_coordinate` - %f
  - `target.target.y_coordinate` - %f
  - `target.target.z_coordinate` - %f

Para cada troca de mensagens explicitada na figura acima segue o objetivo e resultado esperado da aplicação:

- **Mensagem 1:** Server envia uma requisição de ID (`request_id`) para o drone ao qual se deseja conectar (client). O client recebe a requisição e retorna apenas o número correspondente ao seu ID - %d `drone.Drone_ID`. Quando o server recebe o ID retornado, faz uma comparação com as entradas de uma lista de IDs autorizados e verifica a presença ou não do ID retornado na lista. Caso o ID esteja presente na lista, o server envia ao client uma posição de referência (target). O formato da troca de mensagens está explicitado abaixo:

1. **Server:** envia:

`"request_id"`

2. **Client:** envia:

`"%d", drone.Drone_ID`

e printa *"Server Request: request\_id"*

3. **Server:** (Caso autorizado) envia:

```
%.0f, %.0f, %.0f", target.target.x_coordinate,  
target.target.y_coordinate, target.target.z_coordinate
```

e *printa "Authorized drone connected: ID = %s"*, com o ID do drone.

4. **Client:** *printa "Target: (%.0f, %.0f, %.0f)"*, com as coordenadas do target

- **Mensagem 2:** Após a conexão, o server envia uma requisição de status do cliente (**send\_info**) e este retorna os dados relativos a sua posição e sua velocidade nos eixos do espaço tridimensional a cada 5 segundos. O formato da troca de mensagens está explicitado abaixo:

1. **Server:** envia:

```
"send_info"
```

e *printa "Sending info instruction to drone"*.

2. **Client:** envia:

```
"%d: (%.0f, %.0f, %.0f), (%.0f, %.0f, %.0f)", drone.Drone_ID,  
drone.Drone_position.x_coordinate, drone.Drone_position.y_coordinate,  
drone.Drone_position.z_coordinate, drone.Drone_speed.x_coordinate,  
drone.Drone_speed.y_coordinate, drone.Drone_speed.z_coordinate
```

e *printa "Server Request: send\_info"*.

3. **Server:** *printa "Drone response: %d: (%.0f, %.0f, %.0f), (%.0f, %.0f, %.0f)"*, com os dados do drone

- **Mensagem 3:** Server envia informações a respeito da nova posição espacial a ser assumida pelo client (**send\_move**) e este atualiza a sua posição a ser atingida (target) e retorna com uma resposta de reconhecimento (ACK). O formato da troca de mensagens está explicitado abaixo:

1. **Server:** envia:

```
"send_move"
```

e *printa "Sending target change instruction to drone"*.

2. **Server:** envia:

```
%.0f, %.0f, %.0f", target.target.x_coordinate,  
target.target.y_coordinate, target.target.z_coordinate
```

3. **Client:** (Caso seja processado) envia:

"ACK"

e printa *"NEW Target: (%.0f, %.0f, %.0f)"*, com as novas coordenadas do target

4. **Server:** printa *"Drone response: ACK"*, caso drone processe a mudança

- **Mensagem 4:** O server envia uma requisição de desconexão (`send_disconnect`) caso o ID retornado pelo client na requisição feita pela troca de mensagens 1 não esteja presente na lista de IDs autorizados ou caso o client tenha atingido a posição final(target) estabelecida pelo server e o client retorna uma mensagem de reconhecimento (ACK).

1. **Server:** envia:

"send\_disconnect"

2. **Client:** printa: *"Server Request: send\_disconnect"e "Closing connection"*

3. **Server:** (Caso target tenha sido atingido) printa: *"Target Achieved"*

## 2.2 Implementação

Primeiramente, os códigos `client.cpp` e `server.cpp` foram implementados e colocados no github abaixo:

[Clique aqui para ver github com códigos](#)

Agora que a troca de mensagens já está explicitada, será explicado a implementação de cada código:

1. `server.cpp`:

Primeiramente, assim como código do template fornecido, as variáveis `SERVER_PORT` e `BUFSIZE` são inicializadas de modo a serem compatíveis com o problema e o `client.cpp`.

São definidas as seguintes structs do lado do `server.cpp`:

(a) **Target:**

- **Target\_position:** Uma estrutura aninhada que contém três membros de tipo float - `x_coordinate`, `y_coordinate` e `z_coordinate`, que representam as coordenadas tridimensionais da posição final do alvo a ser atingido pelo drone.

(b) **Drone:**

- **Drone\_ID:** Um inteiro que representa o identificador único do drone.
- **Drone\_position:** Uma estrutura aninhada que contém três membros de tipo float - `x_coordinate`, `y_coordinate` e `z_coordinate`, que representam as coordenadas tridimensionais da posição atual do drone.
- **Drone\_speed:** Uma estrutura aninhada similar à `Drone_position`, que representa a velocidade atual do drone nas direções x, y e z.

Após isso, existe o tratamento dos argumentos passados para o arquivo, de modo a capturar a posição final do alvo (target) a ser atingida.

```

47     if (argc != 4) {
48         fprintf(stderr, "Usage: %s x_coordinate y_coordinate z_coordinate\n", argv[0]);
49         exit(EXIT_FAILURE);
50     }
51
52     float x = atof(argv[1]);
53     float y = atof(argv[2]);
54     float z = atof(argv[3]);
55
56     struct Target target = { { x, y, z } };
57     struct Drone drone = {0, {0, 0, 0}, {0, 0, 0}};

```

Figura 2: Tratamento dos argumentos

Depois, há o uso das funções `socket`, `bind` e `listen` para manusear o servidor (bem como visto no template)

Quando o servidor estiver no ar, ele printa "Server is ready" e entra em um `while` de modo a conectar-se com o client.

Uma vez conectado com o client, o server segue uma rotina meio que estabelecida conforme itens abaixo:

(a) **Autorização do Drone e Envio de Target:**

Envia uma requisição de "request\_id" pedindo o ID do drone. Após recebimento, o servidor confere se o ID recebido está presente no vetor `authorized_ids`:

- Caso esteja: Printa uma mensagem autorizando o drone com sua ID e envia a posição do target para o drone.
- Caso não esteja: Fecha a conexão com o drone, printa uma mensagem não autorizando a conexão e envia um "send\_disconnect" para o drone.

Segue imagem abaixo:

```

85     strcpy(buf, "request_id");
86     send(sa, buf, strlen(buf) + 1, 0);
87     bytes = recv(sa, buf, BUF_SIZE, 0);
88     if (bytes <= 0) { close(sa); continue; }
89
90     bool found = false;
91     for (const std::string& str : authorized_ids) {
92         if (strcmp(str.c_str(), buf) == 0) {
93             found = true;
94             break;
95         }
96     }
97     if (bytes <= 0 || found == false) {
98         printf("Unauthorized connection or connection error. Disconnecting client.\n");
99         strcpy(buf, "send_disconnect");
100        send(sa, buf, strlen(buf) + 1, 0);
101        close(sa);
102        continue;
103    }
104    printf("Authorized drone connected: ID = %s\n", buf);
105    sprintf(buf, "%.0f, %.0f, %.0f", target.target.x_coordinate, target.target.y_coordinate, target.target.z_coordinate);
106    send(sa, buf, strlen(buf) + 1, 0);
107

```

Figura 3: Código da Autorização



**(b) Requisição de informação:**

Envia uma requisição de "send\_info" pedindo o ID do drone, posição e velocidade a cada 5 segundos. Implementação foi realizada usando a biblioteca chrono com um current\_time contabilizando o tempo total e last\_info\_request\_time o tempo da última requisição. Segue imagem abaixo:

```

112     while (true) {
113         auto current_time = std::chrono::steady_clock::now();
114         if (std::chrono::duration_cast<std::chrono::seconds>(current_time - last_info_request_time) >= 5s) {
115             printf("Sending info instruction to drone\n");
116             strcpy(buf, "send_info");
117             send(sa, buf, strlen(buf) + 1, 0);
118
119             bytes = recv(sa, buf, BUF_SIZE, 0);
120             if (bytes <= 0) { close(sa); continue; }
121             printf("Drone Response: %s\n", buf);
122
123             sscanf(buf, "%d: (%f, %f, %f), (%f, %f, %f)", &drone.Drone_ID,
124                 &drone.Drone_position.x_coordinate, &drone.Drone_position.y_coordinate, &drone.Drone_position.z_
125                 &drone.Drone_speed.x_coordinate, &drone.Drone_speed.y_coordinate, &drone.Drone_speed.z_coordinat
126
127             last_info_request_time = current_time;
128         }

```

Figura 4: Código da requisição de informação

Assim que recebe a informação do drone, realiza o scan para conseguir variáveis do drone.

**(c) Requisição de mudança de target:**

Envia uma requisição de "send\_move" depois 18 segundos com os novos dados de posicionamento do target. Implementação foi realizada usando a biblioteca chrono como visto anteriormente.

```

130     if (std::chrono::duration_cast<std::chrono::seconds>(current_time - last_sent_time) >= 18s && done)
131         printf("Sending target change instruction to drone\n");
132         strcpy(buf, "send_move");
133         send(sa, buf, strlen(buf) + 1, 0);
134
135         target = {300, 100, 60};
136         sprintf(buf, "%.0f, %.0f, %.0f", target.target.x_coordinate, target.target.y_coordinate, target.
137         send(sa, buf, strlen(buf) + 1, 0);
138
139         bytes = recv(sa, buf, BUF_SIZE, 0);
140         if (bytes <= 0) { close(sa); continue; }
141         printf("Drone Response: %s\n", buf);
142
143         last_sent_time = current_time;
144         done = false;
145     }

```

Figura 5: Código da requisição de mudança de target

**(d) Requisição de desconexão:**

Pode ser realizada no tratamento da autorização, mas caso principal é na conclusão da missão. É inicializado um erro para evitar caso teste com velocidade 0 não atingindo o alvo por erro de float. Com o erro, calcula-se uma possível conclusão da missão comparando cada coordenada do target com o intervalo da coordenada do drone corrigida pela velocidade (multiplicada por 5, pois o check se dá a cada 5 segundos) acrescida do erro.

```

147     float error = 0.1;
148
149     if ((target.target.x_coordinate - 5 * abs(drone.Drone_speed.x_coordinate) - error) < drone.Drone_position.x_coordinate &&
150         drone.Drone_position.x_coordinate < (target.target.x_coordinate + 5 * abs(drone.Drone_speed.x_coordinate) + error) &&
151         (target.target.y_coordinate - 5 * abs(drone.Drone_speed.y_coordinate) - error) < drone.Drone_position.y_coordinate &&
152         drone.Drone_position.y_coordinate < (target.target.y_coordinate + 5 * abs(drone.Drone_speed.y_coordinate) + error) &&
153         (target.target.z_coordinate - 5 * abs(drone.Drone_speed.z_coordinate) - error) < drone.Drone_position.z_coordinate &&
154         drone.Drone_position.z_coordinate < (target.target.z_coordinate + 5 * abs(drone.Drone_speed.z_coordinate) + error))
155     {
156         printf("Target achieved.\n");
157         strcpy(buf, "send_disconnect");
158         send(sa, buf, strlen(buf) + 1, 0);
159         close(sa);
160         close(s);
161         break;
162     }

```

Figura 6: Código ao atingir o alvo e requisição de desconexão

Após atingir alvo, é enviado requisição de "send\_disconnect" e printado "Target Achieved". Posteriormente fecham-se todas conexões e missão acaba.

## 2. client.cpp:

Assim como visto no server, as variáveis `SERVER_PORT` e `BUFSIZE` são inicializadas de modo a serem compatíveis com o problema e o `server.cpp`.

É definida a struct relativa ao drone com o target embutido do lado do `client.cpp`:

### Drone:

- **Drone\_ID:** Um inteiro que representa o identificador único do drone.
- **Drone\_position:** Uma estrutura aninhada que contém três membros de tipo float - `x_coordinate`, `y_coordinate` e `z_coordinate`, que representam as coordenadas tridimensionais da posição atual do drone.
- **Drone\_speed:** Uma estrutura aninhada similar à `Drone_position`, que representa a velocidade atual do drone nas direções x, y e z.
- **Drone\_target:** Uma estrutura aninhada similar à `Drone_position`, que representa o alvo final a ser atingido pelo drone nas direções x, y e z.

Após isso, existe o tratamento dos argumentos passados para o arquivo, de modo a capturar as informações de posição e id do drone, bem como visto no server.

Uma vez conectado com o server, o client segue uma rotina meio que estabelecida conforme itens abaixo dentro de um while:

- (a) **Controle de posição:** Realiza correção de sua posição por meio de sua velocidade multiplicada pelo tempo de acordo com lógica abaixo:

```

71     auto start_time = std::chrono::steady_clock::now();
72
73     while (1) {
74
75         auto current_time = std::chrono::steady_clock::now();
76
77         std::chrono::duration<double> elapsed_seconds = current_time - start_time;
78
79         drone.Drone_position.x_coordinate += drone.Drone_speed.x_coordinate * elapsed_seconds.count();
80         drone.Drone_position.y_coordinate += drone.Drone_speed.y_coordinate * elapsed_seconds.count();
81         drone.Drone_position.z_coordinate += drone.Drone_speed.z_coordinate * elapsed_seconds.count();
82
83         bytes = recv(s, buf, BUFSIZE, 0);
84         if (bytes < 0) {
85             printf("recv call failed");
86             exit(-1);
87         }
88         printf("Server Request: %s\n", buf);

```

Figura 7: Código de correção

Sempre depois de realizado correção, verifica a requisição recebida pelo servidor para posterior tratamento da requisição.

(b) **Envio de ID:**

Ao receber a requisição "request\_id", drone envia seu ID ao servidor e com posterior autorização recebe o target, como será visto depois.

(c) **Recebimento de Mudança de Target:**

Ao receber a requisição "send\_move", drone envia um "ACK" ao servidor e recebe o novo target, como será visto depois.

(d) **Setando Target:**

De modo geral, ao receber as requisições de "request\_id" e "send\_move", o drone irá imediatamente depois receber um target e irá scanear essa informação para ser repassado para as variáveis do Drone\_target e após isso recalcula a velocidade.

```

104     printf("Target: (%s)\n", buf);
105     sscanf(buf, "%f, %f, %f", &drone.Drone_target.x_coordinate, &drone.Drone_target.y_coordinate, &drone.Drone_target.z_coordinate);
106
107     drone.Drone_speed.x_coordinate = (drone.Drone_target.x_coordinate - drone.Drone_position.x_coordinate);
108     drone.Drone_speed.y_coordinate = (drone.Drone_target.y_coordinate - drone.Drone_position.y_coordinate);
109     drone.Drone_speed.z_coordinate = (drone.Drone_target.z_coordinate - drone.Drone_position.z_coordinate);
110
111     float module = sqrt(pow(drone.Drone_speed.x_coordinate, 2) + pow(drone.Drone_speed.y_coordinate, 2) + pow(drone.Drone_speed.z_coordinate, 2));
112
113     drone.Drone_speed.x_coordinate = 5 * drone.Drone_speed.x_coordinate / module;
114     drone.Drone_speed.y_coordinate = 5 * drone.Drone_speed.y_coordinate / module;
115     drone.Drone_speed.z_coordinate = 5 * drone.Drone_speed.z_coordinate / module;
116 }

```

Figura 8: Código ao setar target

A velocidade sempre terá um módulo 5 fixado, como visto no código, mas isso pode ser ajustado e futuramente passado como variável.

(e) **Envio de informações:**

A cada 5 segundos, recebe requisição de "send\_info", e então envia ao servidor informações de seu ID, posição e velocidade

```
118     if (strcmp(buf, "send_info") == 0) {
119         sprintf(buf, "%d: (%.0f, %.0f, %.0f), (%.1f, %.1f, %.1f)", drone.Drone_ID,
120             drone.Drone_position.x_coordinate, drone.Drone_position.y_coordinate, drone.Drone_position.z_coordinate,
121             drone.Drone_speed.x_coordinate, drone.Drone_speed.y_coordinate, drone.Drone_speed.z_coordinate);
122         send(s, buf, strlen(buf) + 1, 0);
123     }
```

Figura 9: Código ao enviar informações ao servidor

(f) **Ao atingir o alvo**

Por fim, ao atingir o alvo e receber "send\_disconnect", drone printa "Target Achieved".

## 2.3 Uso

Após compilação usando g++ (gcc não compila a biblioteca chrono), gera-se os arquivos `server` e `client`. Abre-se então dois terminais e em cada um é passado o seguinte comando:

- `server:`

```
./server target.x_coordinate target.y_coordinate target.z_coordinate
```

Em que cada coordenada é um float e representa o alvo final a ser atingido.

- `server:`

```
./client localhost drone_ID drone.x_coord drone.y_coord drone.z_coord
```

Em que id do drone é um inteiro e cada coordenada de **posição** é um float (abreviei `coordinate` to `coord` para melhor visualização) representando a posição inicial do drone. Ademais, pode ser usada outra conexão além de `localhost`, mas foi essa que foi testada durante o projeto.

### 3 Testes

Três testes foram realizados para verificar o funcionamento correto do protocolo de conexão proposto, baseando-se na implementação de mensagens sugerida. Os detalhes de cada teste estão descritos abaixo:

- **Teste 1 - ID não autorizado:** Neste primeiro caso de teste, inicialmente foi estabelecida uma lista de IDs que possuem a autorização necessária para conectar-se ao servidor. A situação sintetizada aqui foi de um cliente possuindo um ID que não constava nessa lista. Dessa maneira, a mensagem de `request_id` foi enviada e o retorno indicou um ID não presente na lista de IDs autorizados. Diante dessa ocorrência, o servidor enviou a mensagem de desconexão (`send_disconnect`) e a conexão foi interrompida. Os resultados desta simulação podem ser observados abaixo:

```

bry@Notebook-Bry: /mnt/c/...$ ./server_v5 250 300 50
Server is ready
Unauthorized connection or connection error. Disconnecting client.

bry@Notebook-Bry: /mnt/c/...$ ./client_v5 localh
ost 4 100 200 50
Server Request: request_id
Server Request: send_disconnect
Closing connection

```

Figura 10: Teste com drone de id = 4 não autorizado

- **Teste 2 - ID autorizado sem atualização de target:** Para este segundo caso, novamente foi consultada a lista de IDs que têm a autorização necessária para a conexão com o servidor. A situação criada envolve um cliente com um ID que está presente na lista de autorizados.

Desta forma, a mensagem de `request_id` foi enviada e o retorno confirmou um ID que estava presente na lista de IDs autorizados. Com essa verificação, o servidor autorizou a conexão com o cliente e enviou uma posição de referência para se tornar o alvo deste. A partir desse momento, o cliente passou a enviar automaticamente suas posições e velocidades a cada 5 segundos, demonstrando mudanças consistentes com a tentativa de alcançar a posição alvo definida pelo servidor. Os resultados desta simulação podem ser observados abaixo:

```

bry@Notebook-Bry: /mnt/c/...$ ./server_v5 250 300 50
Server is ready
Authorized drone connected: ID = 1
Sending info instruction to drone
Drone Response: 1: (100, 200, 50), (4.2, 2.8, 0.0)
Sending info instruction to drone
Drone Response: 1: (121, 214, 50), (4.2, 2.8, 0.0)
Sending info instruction to drone
Drone Response: 1: (142, 228, 50), (4.2, 2.8, 0.0)
Sending info instruction to drone
Drone Response: 1: (162, 242, 50), (4.2, 2.8, 0.0)
Sending info instruction to drone
Drone Response: 1: (183, 255, 50), (4.2, 2.8, 0.0)
Sending info instruction to drone
Drone Response: 1: (204, 269, 50), (4.2, 2.8, 0.0)
Sending info instruction to drone
Drone Response: 1: (225, 283, 50), (4.2, 2.8, 0.0)
Sending info instruction to drone
Drone Response: 1: (246, 297, 50), (4.2, 2.8, 0.0)
Target achieved.
accept failed

bry@Notebook-Bry: /mnt/c/...$ ./client_v5 localh
ost 1 100 200 50
Server Request: request_id
Target: (250, 300, 50)
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_disconnect
Target achieved.
Closing connection

```

Figura 11: Teste com mudança de target não setada

- **Teste 3 - ID autorizado com atualização de target:** No terceiro e último teste, procedeu-se de maneira similar aos testes anteriores, com a consulta da lista de IDs

autorizados para a conexão com o servidor. Neste cenário, o cliente detinha um ID que estava devidamente autorizado na lista.

Após a confirmação do ID através da mensagem de `request_id`, o servidor permitiu a conexão e enviou uma posição de referência para ser o alvo do cliente. Consequentemente, o cliente começou a enviar automaticamente suas posições e velocidades a cada 5 segundos. Estas mostravam-se alteradas, reflexo da tentativa do cliente de igualar sua posição à do alvo enviado pelo servidor.

No entanto, após 18 segundos de simulação, o servidor enviou uma nova mensagem (`send_move`), instruindo o cliente a atualizar seu alvo e mover-se para alcançar a nova posição estabelecida. A porção de código com essa mudança não é presente como variável (portanto tanto o posicionamento quanto o tempo de mudança estão "hardcodados" no `server.cpp` dentro do tratamento da instrução `send_move`, como pode-se ver na figura auxiliar)

Em resposta, através do envio automático da mensagem `send_info`, foi possível perceber a mudança na posição do cliente até atingir o novo objetivo. Uma vez alcançada a posição final, o servidor encerrou a conexão, enviando uma mensagem de desconexão (`send_disconnect`). Os detalhes desta simulação podem ser conferidos abaixo:

```

bry@Notebook-Bry: /mnt/c/.../ces35$ ./server_v5 250 300 50
Server is Ready
Authorized drone connected: ID = 1
Sending info instruction to drone
Drone Response: 1: (100, 200, 50), (4.2, 2.8, 0.0)
Sending info instruction to drone
Drone Response: 1: (121, 214, 50), (4.2, 2.8, 0.0)
Sending info instruction to drone
Drone Response: 1: (142, 228, 50), (4.2, 2.8, 0.0)
Sending target change instruction to drone
Drone Response: ACK
Sending info instruction to drone
Drone Response: 1: (173, 231, 51), (3.5, -3.6, 0.3)
Sending info instruction to drone
Drone Response: 1: (180, 224, 51), (3.5, -3.6, 0.3)
Sending info instruction to drone
Drone Response: 1: (197, 206, 53), (3.5, -3.6, 0.3)
Sending info instruction to drone
Drone Response: 1: (215, 188, 54), (3.5, -3.6, 0.3)
Sending info instruction to drone
Drone Response: 1: (232, 170, 55), (3.5, -3.6, 0.3)
Sending info instruction to drone
Drone Response: 1: (250, 152, 56), (3.5, -3.6, 0.3)
Sending info instruction to drone
Drone Response: 1: (267, 134, 58), (3.5, -3.6, 0.3)
Sending info instruction to drone
Drone Response: 1: (285, 116, 59), (3.5, -3.6, 0.3)
Target achieved.
accept failed bry@Notebook-Bry: /mnt/c/.../ces35$ |

bry@Notebook-Bry: /mnt/c/.../ces35$ ./client_v5 localh
ost 1 100 200 50
Server Request: request_id
Target: (250, 300, 50)
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_move
NEW Target: (300, 100, 60)
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_info
Server Request: send_disconnect
Target achieved.
Closing connection
bry@Notebook-Bry: /mnt/c/.../ces35$ |
  
```

Figura 12: Teste com mudança de target setada para 18s após início da operação

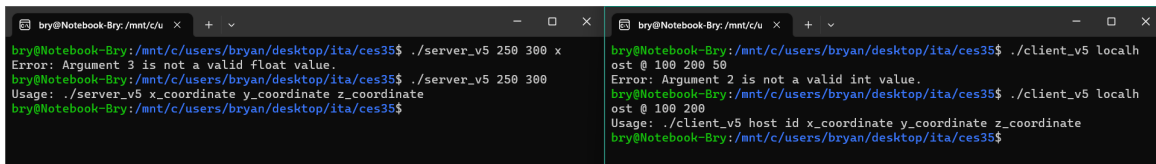
```

130         if (std::chrono::duration_cast<std::chrono::seconds>(current_time - last_sent_time) >= 18s && done)
131             printf("Sending target change instruction to drone\n");
132             strcpy(buf, "send_move");
133             send(sa, buf, strlen(buf) + 1, 0);
  
```

Figura 13: Porção "hardcoded" em que é setado tempo de mudança do target

#### • Teste 4 - Erro ao inicializar input:

Ao inicializar o server e o client, usuário acaba por esquecer algum input ou passar uma variável inadequada para o propósito da aplicação, como um char no lugar de um inteiro ou um float. O código acusa os erros e aponta para que usuário consiga acessar corretamente os arquivos.



The image shows two terminal windows side-by-side. The left window shows the execution of a server program with an error message: 'Error: Argument 3 is not a valid float value.' The right window shows the execution of a client program with an error message: 'Error: Argument 2 is not a valid int value.' Both windows show the usage instructions for their respective programs.

```
bry@Notebook-Bry:/mnt/c/users/bryan/desktop/ita/ces35$ ./server_v5 250 300 x
Error: Argument 3 is not a valid float value.
bry@Notebook-Bry:/mnt/c/users/bryan/desktop/ita/ces35$ ./server_v5 250 300
Usage: ./server_v5 x_coordinate y_coordinate z_coordinate
bry@Notebook-Bry:/mnt/c/users/bryan/desktop/ita/ces35$

bry@Notebook-Bry:/mnt/c/users/bryan/desktop/ita/ces35$ ./client_v5 localhost @ 100 200 50
Error: Argument 2 is not a valid int value.
bry@Notebook-Bry:/mnt/c/users/bryan/desktop/ita/ces35$ ./client_v5 localhost @ 100 200
Usage: ./client_v5 host id x_coordinate y_coordinate z_coordinate
bry@Notebook-Bry:/mnt/c/users/bryan/desktop/ita/ces35$
```

Figura 14: Erros apontados ao usuário não inicializar corretamente

## 4 Possíveis Melhorias e Comentários Gerais

### 4.1 Melhorias

- **Segurança:** Verificou-se a existência de uma fragilidade de segurança no sentido do quesito escolhido para o estabelecimento ou não da conexão abseado apenas no uso do ID e uma lista de IDs autorizados, uma vez que o client, não possuindo um ID presente na lista, poderia alterá-lo até possuir um ID compatível com os presentes na lista. Dessa forma, um sistema de segurança baseado em ID e senha correspondente seria mais eficiente em termos de segurança.
- **Múltipla Conexão:** Poderia haver uma melhoria em relação ao número de drones que possam se conectar com o server ao mesmo tempo. O protocolo utilizado faz a conexão de um drone por vez, sendo que, em termos de aplicação real, necessitaria de um protocolo que abrangesse a conexão de mais elementos por vez.
- **Inputs de Mudança de Target:** A respeito da mudança de target, da maneira implementada, as coordenadas e o tempo de mudança são definidos em termos do código implementado, de modo que poderia haver uma implementação em que as coordenadas entrariam como input do server ou input de um arquivo que possa ser lido. Ademais, lista de drones autorizados também poderia ser passado como argumento, ao invés de estar presente apenas no código.
- **Desconexão:** Outra melhoria que poderia ser feita no código se refere a desconexão feita com o drone a partir do momento em que este atinge a posição definida pelo target. Essa desconexão poderia ser feita quando a missão estabelecida fosse terminada e o drone retornasse a posição original de início, de modo a evitar sua perda.

### 4.2 Comentários

Sobre os comentários gerais do código, as trocas de mensagens "printadas" no prompt command foram feitas de modo a ficar mais fácil de ler todos os server requests do lado do client e o envio e recebimento de informações do lado do server, porém o buffer literal enviado encontra-se na seção 2.1 explicado, e o que aparece no terminal é adicionado de outras strings para maior compreensão.

Além disso, o tratamento de erros foi pensado de modo a evitar que o usuário cometa algum erro, como usar uma string ao invés de float ou int para passar argumentos para o client/-server, além de contabilizar os argumentos para verificação que tudo foi enviado corretamente para inicialização.

Por fim, o propósito foi realizar uma implementação simples da comunicação de servidor e drone tático, portanto vários envios e recebimentos de mensagem acabam por ter uma estrutura de condicionamento mais exata (como em `send_move`) bem dependente da resposta correta do client, caso contrário haveria algum erro. Portanto não foi realizado mecanismo para tentativa contínua de request, de modo que uma pequena falha (que não foi simulada) poderia fechar a conexão imediatamente entre drone e servidor, deixando o drone em uma situação ruim.