Tài liệu ACS

```
STX = 0x02
ETX = 0x03


def create_frame(index_type, ap, msg, agv_id, command):
    if len(command) < 12:
        emp_buf = [0 for _ in range(12 - len(command))]
        command = command + ''.join(map(str, emp_buf))
    elif len(command) > 12:
        logging.info("Command Error")
    else:
        command = command
    if int(agv_id) < 10:
        raw_array = [index_type, ap, msg, 0, agv_id, command]
    else:
        raw_array = [index_type, ap, msg, agv_id, command]

    list_to_str = ''.join(map(str, raw_array))

    raw_msg = bytearray(list_to_str, "ascii")

    raw_msg.insert(0, STX)
    check_sum = 0x00
    for item in raw_msg:
        check_sum += int(item)
    check_sum = check_sum - 2
    check_sum_arr = ''.join(map(str, hex(check_sum)))
    check_sum = bytes(check_sum_arr[len(check_sum_arr) - 1].upper(), 'utf-8')
    raw_msg += check_sum
    raw_msg.append(ETX)
    return raw_msg
```

```python
def listen_rev(server: socket.socket):
    ready = select.select([server], [], [], 1)
    if ready[0]:
        data_all = server.recv(20)
        if (data_all[0] != STX) or (data_all[19] != ETX):
            logging.warning("Pack head error")
        else:
            check_sum = 0x00
            for item in range(18):
                check_sum += int(data_all[item])
            check_sum = check_sum - 2
            check_sum_arr = ''.join(map(str, hex(check_sum)))
            check_sum = bytes(check_sum_arr[len(check_sum_arr) - 1].upper(), 'utf-8')
            if int.from_bytes(check_sum, "little", signed=False) != data_all[18]:
                logging.info("Checksum Error")
            else:
                return {"index": chr(data_all[1]), "ap": int(chr(data_all[2])), "msg": int(chr(data_all[3])),
                        "agv_id": int(data_all[4:6].decode('utf-8')), "command": data_all[6:18].decode('utf-8')}
    return False
```

```python
def collision_in_tag_read(self, agv_id, area):
    msg = random.randint(1, 9)
    comm = "I" + str(area)
    self._server.send(create_frame('E', 1, msg, agv_id, comm))
    logging.info("Collision In Tag: " + str({"agv_id": agv_id, "target": area}))
```

Tổng quan:

AGV sẽ kết nối với qua ACS thông qua giao thức socket tcp/ip. Nó sẽ gửi cũng như nhận về theo dạng frame với size là 20 byte.

Ví dụ gửi frame dạng: \x02E1772I801000000008\x03

STX	index_type	ap	msg	agv_id	command	checksum	ETX
\x02	E	1	7	72	I80100000000	8	\x03

Giải thích

- **STX (\x02):** Bắt đầu frame
- **index_type (E):** Tùy vào trường hợp sẽ có các ký tự khác nhau (ví dụ collision là **E**, thang máy là **T**, …)
- **ap (1):** Thường là 1 và 0. Do nhà máy thêm vào để kiểm soát frame.

- **msg (7):** có kích thước **1 < msg < 9**
- **agv_id (72):** có kích thước **1 < msg < 99.** Nếu nhỏ hơn 10 thì thêm số 0 đằng trước (01, 02, …, 09)
- **command (I80100000000):**
    - **I** Tùy vào trường hợp sẽ có các ký tự khác nhau (ví dụ collision là **I**, thang máy là **CALL**, …)
    - **801:** Vùng collision (tùy vào trường hợp có hoặc không)
    - **00000000:** Pađing command để cho command luôn có size là 12
- **checksum (8):** Tổng kiểm tra tính toàn vẹn của dữ liệu
- **ETX (\x03):** Kết thúc khung**.**

1. create_frame

def create_frame(index_type, ap, msg, agv_id, command)

Mô tả:

Hàm này tạo ra một khung truyền thông bằng cách sử dụng các tham số đã cung cấp. Nó đảm bảo khung theo định dạng cụ thể với một ký tự bắt đầu (STX), thông điệp chính, một mã kiểm tra (checksum) để phát hiện lỗi và một ký tự kết thúc (ETX).

Tham số:

- **index_type**: Loại hoặc chỉ số cho thông điệp (ví dụ: một tín hiệu điều khiển).
- **ap**: ID ứng dụng hoặc một trường đại diện cho các tham số cụ thể.
- **msg**: Một mã hoặc số nhận dạng thông điệp.
- **agv_id**: ID của xe AGV (Automated Guided Vehicle), có thể là một chữ số hoặc hai chữ số.
- **command**: Lệnh mà AGV cần thực thi. Chuỗi lệnh sẽ được thêm bù cho đủ 12 ký tự nếu ngắn hơn, hoặc sẽ ghi nhận lỗi nếu dài hơn.

Chức năng: Hàm create_frame có chức năng tạo ra một khung truyền thông (frame) theo định dạng cụ thể để gửi dữ liệu giữa các thiết bị, ví dụ như xe tự hành AGV. Khung này bao gồm các thông tin như loại thông điệp (index_type), các tham số cụ thể (ap, msg, agv_id), lệnh (command), cùng với mã kiểm tra (checksum) để đảm bảo tính toàn vẹn của dữ liệu trong quá trình truyền.

- Xử lý chuỗi lệnh (command)
    - Nếu chuỗi lệnh ngắn hơn 12 ký tự, hàm sẽ thêm số 0 cho đến khi đủ 12 ký tự.

    - Nếu chuỗi lệnh dài hơn 12 ký tự, nó sẽ ghi lại thông tin lỗi trong nhật ký (logging.info).
- Tạo mảng dữ liệu thô (raw_array):
    - Dữ liệu thô sẽ bao gồm các tham số: loại thông điệp (index_type), ứng dụng (ap), thông điệp (msg), ID AGV (agv_id), và lệnh (command).
    - Nếu ID của AGV nhỏ hơn 10, hàm thêm số 0 trước ID AGV để đảm bảo định dạng đúng.
- Chuyển mảng thành chuỗi ASCII:
    - Sau khi tạo mảng dữ liệu thô, hàm chuyển mảng này thành chuỗi ASCII và sau đó thành bytearray.
- Chèn ký tự bắt đầu (STX) và tính mã kiểm tra (checksum):
    - Ký tự bắt đầu (STX) được thêm vào đầu khung.
    - Hàm tính toán mã kiểm tra bằng cách cộng tất cả các giá trị trong thông điệp thô, sau đó trừ đi 2 (để loại trừ ký tự STX và ETX khỏi tính toán).
- Chèn mã kiểm tra và ký tự kết thúc (ETX):
    - Sau khi tính toán mã kiểm tra, hàm chèn mã kiểm tra vào cuối thông điệp.
    - Cuối cùng, ký tự kết thúc (ETX) được thêm vào để hoàn tất khung.
- Trả về khung dữ liệu
    - Hàm trả về khung dữ liệu hoàn chỉnh dưới dạng một bytearray sẵn sàng để được gửi qua mạng.

2. listen_rev

def listen_rev(server: socket.socket)

Mô tả:

Hàm này lắng nghe dữ liệu đến từ một socket và xử lý khung nhận được. Nó kiểm tra ký tự bắt đầu (STX) và kết thúc (ETX) và kiểm tra mã kiểm tra (checksum) để đảm bảo tính toàn vẹn của dữ liệu.

Chức năng:

- Sử dụng select.select để kiểm tra có dữ liệu đến trên socket server hay không.
- Nếu nhận được dữ liệu, nó đảm bảo rằng thông điệp có các ký tự bắt đầu và kết thúc đúng (STX, ETX).
- Tính toán và kiểm tra mã kiểm tra (checksum) để đảm bảo tính toàn vẹn của dữ liệu.

- Trích xuất và giải mã các phần liên quan của thông điệp, như chỉ số, ap, thông điệp, ID AGV và lệnh.
- Trả về dữ liệu đã phân tích dưới dạng dictionary nếu thông điệp hợp lệ; nếu không, ghi nhận các lỗi như sai mã kiểm tra hoặc khung lỗi.

3. collision_in_tag_read

def collision_in_tag_read(agv_id, area)

Mô tả:
Hàm này mô phỏng việc gửi một cảnh báo va chạm từ AGV. Nó tạo ra một thông điệp với area cụ thể và gửi thông điệp bằng cách sử dụng hàm create_frame.
Tham số:

- agv_id: ID của AGV báo cáo va chạm
- area: Khu vực mục tiêu nơi xảy ra va chạm hoặc sự kiện.

Chức năng:

- Tạo một mã thông điệp ngẫu nhiên (msg).
- Tạo lệnh với tiền tố I và tiếp theo là area.
- Gửi lệnh bằng cách sử dụng hàm create_frame để định dạng thông điệp, và ghi lại sự kiện va chạm.

Cách sử dụng chung

1. **Gửi một thông điệp (lệnh AGV)**:
   - Sử dụng create_frame để tạo khung thông điệp.
   - Gửi khung qua kết nối socket đến máy nhận.
2. **Nhận và xử lý thông điệp**:
   - Sử dụng listen_rev để lắng nghe thông điệp đến trên socket server.
   - Phân tích thông điệp và xử lý dựa trên nội dung của nó.
3. **Xử lý va chạm**:
   - collision_in_tag_read được sử dụng để báo cáo va chạm trong một thẻ của AGV. Dữ liệu về va chạm được đóng gói và gửi qua mạng