Mavlink communication

In this document, we will explain how mavlink communication works in an UDP program.

Initialization

Firts of all you need to know how to initialize UDP connection for listen and send packets.

-Initialize listening This first step is the more simple, you only have to initialize a listening socket on the port 14550 (port where the controller send informations) and bind it.

```
struct sockaddr_in locAddr;
int sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);

locAddr.sin_family = AF_INET;
locAddr.sin_addr.s_addr = INADDR_ANY;
locAddr.sin_port = htons(14550);
memset (&locAddr.sin_zero, 0, sizeof(locAddr.sin_zero));

if (-1 == bind(sock,(struct sockaddr *)&locAddr, sizeof(struct sockaddr))){
        perror("error bind failed");
        close(sock);
        return -1;
}
```

-Initialize sending This step is more complicated because the target port always changes. The solution that we have found is to listen, with the listening socket initialized previously, all incoming messages and stop when we receive one from the target IP : 10.1.1.1 (adress of the controller). Thus we get the target port and we can initialize the sending socket.

```
struct sockaddr_in targetAddr;
targetAddr.sin_family = AF_INET;
targetAddr.sin_addr.s_addr = inet_addr("10.1.1.1");
targetAddr.sin_port = possibleTarget.sin_port;
memset (&targetAddr.sin_zero, 0, sizeof(targetAddr.sin_zero));
```

Receive mavlink by UDP

-Buffer First of all you need intialize a buffer that you fill block of memory at each use with "memset" :

```
#define BUFFER_LENGTH 2041
...
uint8_t buf[BUFFER_LENGTH];
...
memset(buf, 0, BUFFER_LENGTH);
```

-Receiving After that, you receive bytes with de function "recvfrom".

```
recsize = recvfrom(sock, (void *)buf, BUFFER_LENGTH, 0, (struct sockaddr *)&targetAddr, &
    fromlen);
```

-Parsing Then you parse it in order to convert it in a mavlink message, thus you can interpret this message.

```
if (mavlink_parse_char(chan, buf[i], &msg, &status)){
        printf("\nReceived packet: SYS: %d, COMP: %d, LEN: %d, MSG ID: %d\n\n", msg.
            sysid, msg.compid, msg.len, msg.msgid);
}
```

Send mavlink by UDP

-Buffer First of all you need intialize a buffer that you fill block of memory at each use with "memset" like before :

```
#define BUFFER_LENGTH 2041
...
uint8_t buf[BUFFER_LENGTH];
...
memset(buf, 0, BUFFER_LENGTH);
```

-Packing Second step is to "pack" the mavlink message with the "pack function". There are numerous type of "pack function" and for various uses, we will dwell on these different types later, for the moment we will use the basic message "heartbeat" which is a frequent message to give "sign of life" of a device.

```
mavlink_msg_heartbeat_pack(255,0,&msg,MAV_TYPE_GCS,MAV_AUTOPILOT_ARDUPILOTMEGA,0xc0,0x0,
    MAV_STATE_ACTIVE);
```

-Converting After being pack, we put the mavlink message in the buffer with the function "mavlink_msg_to_send_buffer".

```
len = mavlink_msg_to_send_buffer(buf, &msg);
```

-Sending To finish, send the buffer with the function "sendto".

```
bytes_sent = sendto(sock, buf, len, 0, (struct sockaddr*)&targetAddr, sizeof(struct
    sockaddr_in));
```

Decode mavlink message

In the part we have seen how to get a mavlink message. Now we are going see how to decode it and how to diplay his informations.

-Identified First of all you have to know that each mavlink message have a message id that indicate the type of message encapsulated. Thus we only have to compare the field "msgid" of the message with the macro constant of a type. To have more informations about tram's construction and about the different types, we recommend you to see the documents mavlink-devguide.pdf and mavlink_protocol.pdf.

```
if(msg.msgid == MAVLINK_MSG_ID_SYS_STATUS) //If the message is of type SYS_STATUS
```

-Decoding When you have identified the type, you need to create a variable of this type in order to call the right decode function to put information into the variable. To find the type and the fonction's name you have to go in the corresponding file of the library.

```
mavlink_sys_status_t sys_status;
mavlink_msg_sys_status_decode(&msg, &sys_status);
```

-Displaying To finish, if you want to see the data contents into the variable, you simply have to print fields of the structure. In the same way, to know their name, you have to go in the file of the corresponding type.

```
printf("SYS_STATUS :\nOnboard_control_sensors_present : %d,
    Onboard_control_sensors_enabled : %d, Onboard_control_sensors_health : %d, Load : %d,
    Voltage_battery : %d, Current_battery : %d, Drop_rate_comm : %d, Errors_comm : %d,
    Errors_count1 : %d, Errors_count2 : %d, Errors_count3 : %d, Errors_count4 : %d,
    Battery_remaining : %d\n\n", sys_status.onboard_control_sensors_present, sys_status.
    onboard_control_sensors_enabled, sys_status.onboard_control_sensors_health,
    sys_status.load, sys_status.voltage_battery, sys_status.current_battery, sys_status.
    drop_rate_comm, sys_status.errors_comm, sys_status.errors_count1, sys_status.
    errors_count2, sys_status.errors_count3, sys_status.errors_count4, sys_status.
    battery_remaining);
```

Send mavlink order

As we have seen in the part we have to call a "pack function" in order to send mavlink message. Now we are going see what packs can be used to send an order to drone.

-Set mode To request to change the flight mode you need to call the function "mavlink_msg_set_mode_pack". The first parameter is the system id of the source (255 because we simulate a QGC), the second one is the component id of the source, the third is the mavlink message that we want to pack, the fourth is system id of the target (1 for all vehicle) and the others the type of flight mode choice.

```
mavlink_msg_set_mode_pack(255, 0, &msg, 1, MAV_MODE_MANUAL_ARMED+
    MAV_MODE_FLAG_CUSTOM_MODE_ENABLED, COPTER_MODE_ALT_HOLD);
```

-Command long For many orders the "pack function" which is called is "mavlink_msg_command_long_pack". His four first parameters are the same than before, after that we have the component id of the target and then the type of command. The rest of the parameters depends of the command. Thus with this kind of function we can send the order to arm the drone like below :

```
mavlink_msg_command_long_pack(255, 0, &msg, 1, 0, MAV_CMD_COMPONENT_ARM_DISARM, 0, 1, 0, 0, 0,
     0, 0, 0);
```

Send mavlink mission

In this section we will describe how a mission is made and how to prepare and send one to the drone.

-Mission count First of all, you have to send a MAVLINK_MSG_ID_MISSION_COUNT. The contents of the different fields of the message are :

— count = (2 or 4 or ...), represents the number of mission items in the sequence

— arget_system = 1 (0x01), the ID of the target system.
— target_component = 190 (0xbe), the ID of the target component

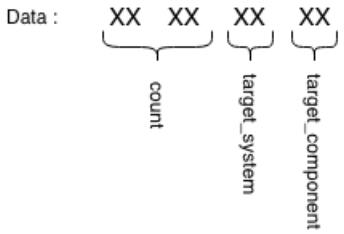

FIGURE 1 – Show how a mission count frame is built

Example of function call for packing a mission count message :

```
mavlink_msg_mission_count_pack(255, 0, &msg, 1, 190, 2); //MISSION_COUNT
```

-Mission item Next, you send some MAVLINK_MSG_ID_MISSION_ITEM depending on the mission you want to perform. The contents of the different fields of the message are :

— param1 = depends on the command, see in MAV_CMD enum.
— param2 = depends on the command, see in MAV_CMD enum.
— param3 = depends on the command, see in MAV_CMD enum.
— param4 = depends on the command, see in MAV_CMD enum.
— x = depends on the command, local : X coordinate, global : latitude.
— y = depends on the command, local : Y coordinate, global : longitude.
— z = depends on the command, local : Z coordinate, global : altitude (relative or absolute, depending on frame).
— seq = sequence (0, 1, 2, 3, ...).
— command = MAV_CMD_NAV_WAYPOINT (0x0010 = 16) or MAV_CMD_NAV_TAKEOFF (0x0016 = 22) or MAV_CMD_DO_CHANGE_SPEED (0x00b2 = 178) or ...
— target_system = 1 (0x01), the ID of the target system.
— target_component = 190 (0xbe), the ID of the target component.
— frame = MAV_FRAME_GLOBAL (0x0000 = 0) or MAV_FRAME_GLOBAL_RELATIVE_ALT (0x0003 = 3) or MAV_FRAME_MISSION (0x0002 = 2) or ..., see more in MAV_FRAME enum. The coordinate system of the waypoint.
— current = 1 for the first item, 0 for the following items.
— autocontinue = 1, autocontinue to next waypoint.

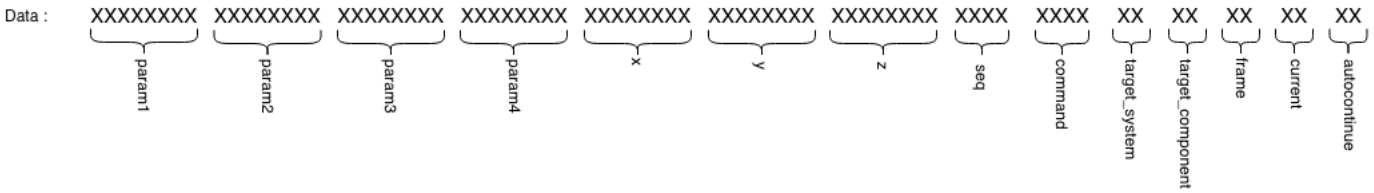

FIGURE 2 – Show how a mission item frame is built

Examples of function call for packing a mission item message :

```
mavlink_msg_mission_item_pack(255, 0, &msg, 1, 190, 0, MAV_FRAME_GLOBAL,
    MAV_CMD_NAV_WAYPOINT, 1, 1, 0, 0, 0, 0, gps_raw_int.lat, gps_raw_int.lon, gps_raw_int.alt);
    //MISSION ITEM WAYPOINT
...
```

```
mavlink_msg_mission_item_pack(255, 0, &msg, 1, 190, 1, MAV_FRAME_GLOBAL_RELATIVE_ALT,
    MAV_CMD_NAV_TAKEOFF, 0, 1, 15, 0, 0, 0, 0, 0, 1); //MISSION ITEM TAKEOFF 1m
```

-Set mode Then, you send a MAVLINK_MSG_ID_SET_MODE in order to put the drone in guided mode. The contents of the different fields of the message are :

- target_system = 1, the ID of the target system.
- custom_mode = COPTER_MODE_GUIDED (0x0004 = 4) or COPTER_MODE_LAND (0x0009 = 9), see more in COPTER_MODE enum.
- base_mode = MAV_MODE_GUIDED_DISARMED+MAV_MODE_FLAG_CUSTOM_MODE_ENAI (0x59 = 89) or MAV_MODE_GUIDED_ARMED+MAV_MODE_FLAG_CUSTOM_MODE_ENABLE (OxD9 = 217), see more in MAV_MODE and MODE_FLAG enum.

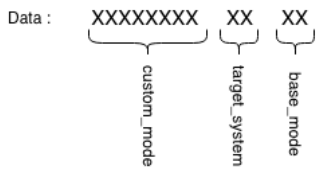

FIGURE 3 – Show how a set mode frame is built

Example of function call for packing a set mode message :

```
mavlink_msg_set_mode_pack(255, 0, &msg, 1, MAV_MODE_GUIDED_DISARMED+
    MAV_MODE_FLAG_CUSTOM_MODE_ENABLED, COPTER_MODE_GUIDED); // Request change flight mode
```

-Command long Then, you send 2 MAVLINK_MSG_ID_COMMAND_LONG in order to arm and to start the mission. The contents of the different fields of the message are :

- param1 = 1 or 0, depends on the command, see in MAV_CMD enum
- param2 = depends on the command, see in MAV_CMD enum
- param3 = depends on the command, see in MAV_CMD enum
- param4 = depends on the command, see in MAV_CMD enum
- param5 = depends on the command, see in MAV_CMD enum
- param6 = depends on the command, see in MAV_CMD enum
- param7 = depends on the command, see in MAV_CMD enum
- command = MAV_CMD_COMPONENT_ARM_DISARM (0x0190 = 400) or MAV_CMD_MISSION_S (0x012C = 300)
- target_system = 1, the ID of the target system.
- target_component = 1, the ID of the target component.
- confirmation = 0 : first transmission of this command, 1-255 : confirmation transmissions.

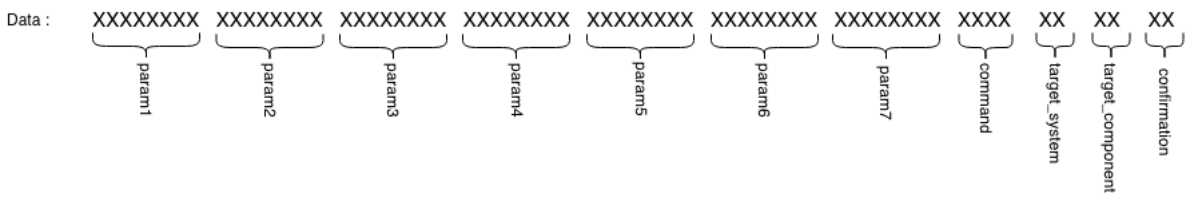

FIGURE 4 – Show how a command long frame is built

Example of function call for packing a command long message :

```
mavlink_msg_command_long_pack(255, 0, &msg, 1, 1, MAV_CMD_COMPONENT_ARM_DISARM, 0, 1, 0, 0, 0,
    0, 0, 0); //Request arm motors
```

...
```
mavlink_msg_command_long_pack(255, 0, &msg, 1, 1, MAV_CMD_MISSION_START, 0, 0, 0, 0, 0, 0, 0, 0);
    //Request start mission
```

Sources

- Mavlink library version 1 in c : https ://github.com/mavlink/c_library_v1
- Small example using the library in c : https ://mavlink.io/en/examples/c_udp.html
- IP adress communication : https ://unix.stackexchange.com/questions/320640/3dr-solo-drone-wifi-communication
- Port communication : https ://github.com/PX4/Devguide/issues/357
- Message ID macro and values : https ://groups.google.com/forum/# !topic/mavlink/1zgHUM67E-A