外部控制 dll 模型 UDP 通信端口验证实验原理

1. 文件目录

例程目录：[安装目录]\RflySimAPIs\4.RflySimModel\0.ApiExps\14.inCopterData\

文件夹/文件名称	说明
1.ExtCtrlAPI-UDP20100\Readme.pdf	外部通信实验之读取状态估计值实验步骤
2.ExtCtrlAPI-UDP30100\Readme.pdf	外部通信实验之读取仿真真值数据实验步骤
3.ExtCtrlAPI-UDP40100\Readme.pdf	外部通信实验之获取平台 rfly_px4 uORB 消息实验步骤

2. 总体说明

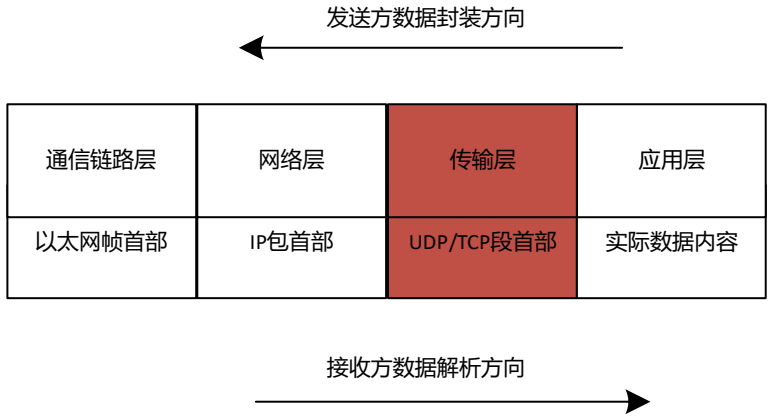
2.1. 外部控制通信需求

- 完整的外部控制及仿真和通信模式介绍参考[2]

外部控制程序是从在环仿真系统外部对载具模型进行的控制，包括路径规划、远程控制指令等。在平台例程中主要通过 MATLAB 和 Python 调用对应接口实现。这些接口是基于网络协议实现的。

2.2. 传输层协议 UDP 简介[7]

- **RflySim 平台的完整网络通信机制参考[4]**



2.2.1. UDP 报文结构

无论应用程序使用 TCP（传输控制协议）传数据，还是 UDP（User Datagram Protocol 用户数据报协议） 传数据，都要监听一个端口，这个端口用来区分应用程序，故端口不能冲突。所以，无论是 TCP 还是 UDP 包头里面都应该有端口号，根据端口号，将数据交给相应的应用程序。

源端口号（16位）	目的端口号（16位）
UDP长度（16位）	UDP校验和（16位）
实际数据内容	

每个 UDP 报文分为 UDP 报头和 UDP 数据区两部分。报头由 4 个 16 位长（2 字节*4=8 字节）字段组成，分别说明该报文的源端口、目的端口、报文长度和校验值。

- 源端口号(Source Port) :这个字段占据 UDP 报文头的前 16 位，通常包含发送数据报的应用程序所使用的 UDP 端口。接收端的应用程序利用这个字段的值作为发送响应的目的地址。这个字段是可选项，有时不会设置源端口号。没有源端口号就默认为 0，通常用于不需要返回消息的通信中。
- 目标端口号(Destination Port): 表示接收端端口，字段长为 16 位
- 长度(Length): 该字段占据 16 位，表示 UDP 数据报长度，包含 UDP 报文头和 UDP 数据长度。因为 UDP 报文头长度是 8 个字节，所以这个值最小为 8，最大长度为 65535 字节。
- 校验和(Checksum)：UDP 使用校验和来保证数据安全性，UDP 的校验和也提供了差错检测功能，差错检测用于校验报文段从源到目标主机的过程中，数据的完整性是否发生了改变。

2.2.2. UDP 传输的特点

- 速度快，采用 UDP 协议时，只要应用进程将数据传给 UDP，UDP 就会将此数据打包进 UDP 报文段并立刻传递给网络层，而 TCP 有拥塞控制的功能，它会在发送前判断互联网的拥堵情况，如果互联网极度阻塞，那么就会抑制 TCP 的发送方。使用 UDP 的目的就是希望实时性。
- 无须建立连接，TCP 在数据传输之前需要经过三次握手的操作，而 UDP 则无须任何准备即可进行数据传输。因此 UDP 没有建立连接的时延。
- 无连接状态，TCP 需要在端系统中维护连接状态，连接状态包括接收和发送缓存、拥塞控制参数以及序号和确认号的参数，在 UDP 中没有这些参数，也没有发送缓存和接受缓存。
- 分组首部开销小，每个 TCP 报文段都有 20 字节的首部开销，而 UDP 仅仅只有 8 字节的开销。

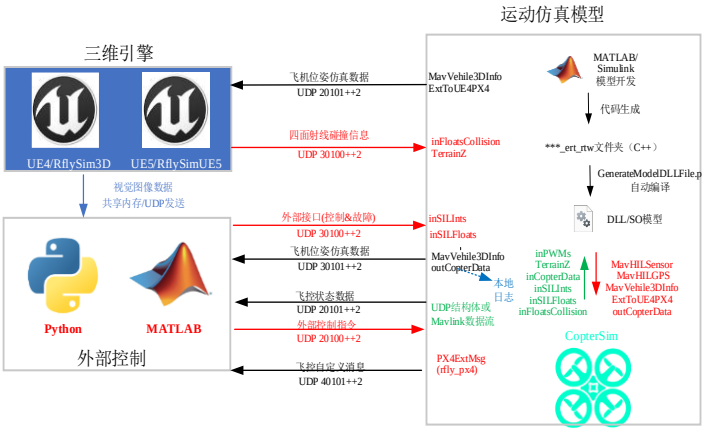
注意，应用程序可以通过增加确认和重传机制实现可靠的数据传输，所以使用 UDP 协议最大的特点就是速度快。

2.3. RflySim 平台外部控制 UDP 通信机制

- **RflySim 平台的详细控制协议参考[2]中的 RflySim 控制协议部分**

UDP 是一种无连接的网络协议，允许快速的数据传输，但不保证数据包的顺序或完整性。在仿真中，UDP 通信通常用于实时数据传输，如飞行器的状态信息（这里包括运动模

型解算出的位姿信息（真值）和底层控制器给出的状态估计值）。



外部控制程序可以通过 UDP 20100++2 系列端口发送外部控制指令给 CopterSim。CopterSim 中的 dll 模型可以通过 UDP 20101++2 系列端口将载具位姿仿真数据（运动仿真模型解算出的位姿数据）发送到三维引擎 RflySim3D；外部控制程序也可以通过监听 UDP 20101++2 系列端口获取飞控状态数据（底层控制器给出的状态估计值）。

三维引擎 RflySim3D 可以通过 UDP 30100++2 系列端口发送四面射线碰撞信息给 CopterSim。外部控制程序可以通过监听 UDP 30101++2 系列端口获取载具位姿仿真数据（运动仿真模型解算出的位姿数据）。

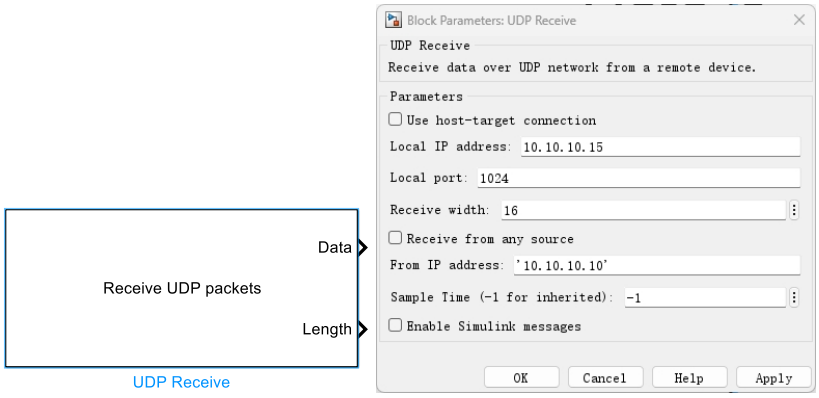
外部控制程序可以通过监听 UDP 40101++2 系列端口获取飞控自定义消息。

3. 关键功能的实现

- **RflySim 平台的完整 UDP 控制接口参考[2]中的控制接口部分**

3.1. UDP 解析模块[1]

3.1.1. 接收 UDP 包并提取长度（UDP Receive）



这个模块主要功能是从本机网络或远程设备接收 UDP 数据包。

所属模块库： 属于 Simulink Real-Time / IP 库。

描述： UDP 接收模块能够通过开发计算机和目标计算机之间的连接，或者使用专用以太网卡接收数据。如果使用专用以太网卡，需要使用 Speedgoat 配置工具来设置专用以太网

板卡。

当该模块在目标计算机上的实时应用程序中执行，或者在开发计算机上的模型模拟中运行时，参数“本地 IP 地址（Local IP address）”将被应用。如果您的模型在开发计算机上的 Simulink®中运行，您可以使用此模块接收来自远程设备的数据。在这种情况下，Windows®操作系统将决定网络连接。

输出

- 数据（Data）：接收的数据，是一个包含通过 UDP 网络接收的数据的 uint8 向量。如果没有接收到新的数据包，数据值将被保持。要确定是否收到新的数据包，可以使用长度（Length）输出端口。如果启用了“启用 Simulink 消息”参数，数据类型为 UDP_Packet。该数据类型包括：
    - IP_Address：数据类型为 uint8，实部，维度为[4 1]
    - IP_Port：数据类型为 uint16，实部，维度为 1
    - Length：数据类型为 uint16，实部，维度为 1
    - Data：数据类型为 uint8，实部，维度为[75 1]
- 长度（Length）：接收到的字节数，数据类型为 double。当禁用“启用 Simulink 消息”参数时，此端口可用。Length 是接收到的新包中的字节数，否则为 0。如果接收到的字节超过通过具有定义的数据宽度的接收端口可以输出的字节数，超出的字节将被丢弃。

参数详情

- **使用主机-目标连接**（Use host-target connection）
    - **选项**：'off'（默认）| 'on'
    - **依赖性**：启用此参数将停用“本地 IP 地址”参数，并且排除端口 1 至 1023 和 5500 至 5560 用于 UDP 通信。
    - **程序使用**：
        - **块参数**：useHostTargetConn
- **本地 IP 地址**（Local IP address）
    - **默认**：使用主机-目标连接
    - 当“本地 IP 地址”设置为使用主机-目标连接时，该模块使用开发和目标计算机之间的连接。使用 0.0.0.0 绑定到 INADDR_ANY，这使得套接字能够接收广播数据报。
    - **程序使用**：
        - **块参数**：ipAddress
- **本地端口**（Local port）
    - **范围**：1–65535
    - 指定 UDP 端口接收数据。端口 1 至 1023 和 5500 至 5560 被保留用于

Simulink Real-Time 通信。

- **程序使用：**
  - **块参数**：localPort

- **接收宽度**（Receive width）
  - **范围**：1–65504
  - 确定数据输出向量的宽度。判断此值是否小于接收包中的字节数，超出的字节将被丢弃。
  - **程序使用：**
    - **块参数**：rcvWidth

- **接收任意来源**（Receive from any source）
  - **选项**：on（默认）| off
  - 当“接收任意来源”启用时，该模块可以从任何可访问的 IP 地址接收数据。禁用时，模块只从您在“来自 IP 地址”中指定的地址接收数据。
  - 若要使“来自 IP 地址”参数可见，请取消选中“接收任意来源”复选框。
  - **程序使用：**
    - **块参数**：rcvFmAny

- **来自 IP 地址**（From IP address）
  - 输入有效的 IP 地址，格式为点分十进制字符向量，例如 192.168.7.2。您也可以使用 MATLAB®表达式返回有效的 IP 地址字符向量。
  - 地址 255.255.255.255 是无效的 IP 地址。
  - 若要使此参数可见，需要清除“接收任意来源”的选项。

- **来自 IP 地址**（fmAddress）
  - **用途**：此参数指定块应接受数据的源 IP 地址。
  - **可见性**：要使此参数可见，必须取消选中“从任何来源接收”复选框。
  - **编程使用：**
    - **块参数**：fmAddress

- **采样时间**（sampleTime）
  - **选项**：-1（默认继承）| 数值
  - **描述**：输入基本采样时间或基本采样时间的倍数。设置为 -1 表示采样时间从系统继承。
  - **编程使用：**
    - **块参数**：sampleTime

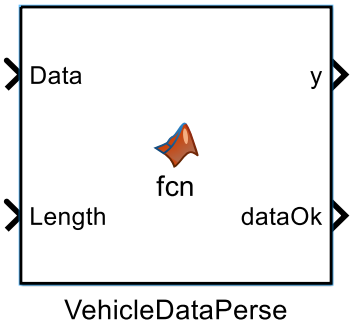
- **启用 Simulink 消息**（MessageOut）
  - **选项**：关闭（默认）| 开启
  - **描述**：启用时，块将传入数据视为消息而不是连续数据流。这种配

置会移除通常用于连续数据流的长度端口。

- **编程使用：**
  - **块参数：**MessageOut

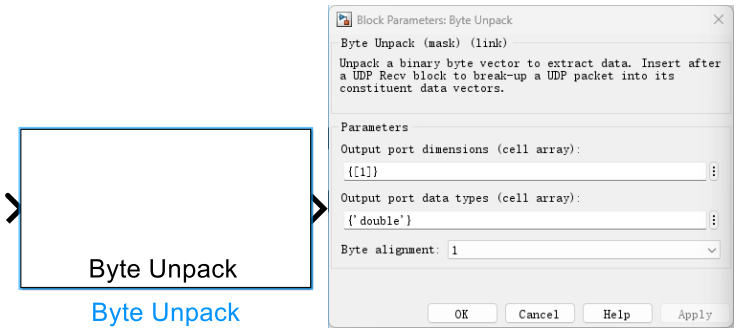
- **每个时间步的最大消息数**（maxPacket）
  - **选项：**1（默认）| 整数
  - **描述：**限制每个时间步可以处理的消息数量。如果超出此限制，块将生成错误。这对于管理数据流并确保消息处理不会使系统过载非常有用。
  - **编程使用：**
    - **块参数：**maxPacket

3.1.2. 校验数据（VehicleDataPerse）



函数接收两个参数：Data 和 Length，分别表示接收到的 UDP 数据内容和数据长度。函数返回两个值：y 和 dataOk，其中 y 用来存储处理后的数据，dataOk 是一个标志，用于指示数据是否有效。

3.1.3. 字节流解包（Byte Unpack）



主要功能是将 uint8 向量转换为不同数据类型的输出信号。

所属模块库

- **Embedded Coder / Embedded Targets / Host Communication（嵌入式编码器/嵌入式目标/主机通信）**

描述

- **Byte Unpack 模块接收一个 uint8 向量，并根据输入向量的内容将这个向量转换成不同 Simulink 数据类型的输出信号。**可以使用块参数来指定输出信号的维度和数据类型，以及块输出的各个向量中数据的对齐方式。由于 UDP 协议以 uint8 格

式传输数据，你可以使用这个模块将作为 UDP 数据包接收的数据重新格式化，以便在模型中使用，通过将这个块的输入与 UDP Send 块的输出相连。

输入

- **Port_1 — 要转换的信号**
  - **类型：向量**
  - **输入向量数据类型：uint8**

输出

- **Port_1 — 转换后的信号数据**
  - **信号类型：默认为双精度类型（double）| 信号数据的数组**
  - **输出信号包含一个或多个数据类型的输入信号数组。**

输出信号支持的数据类型

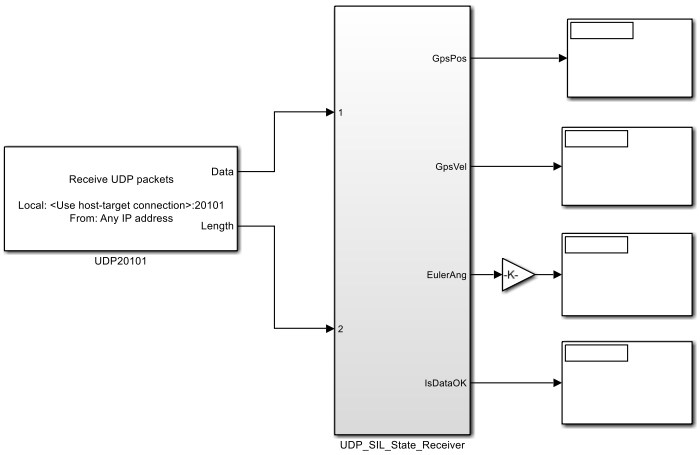
- **single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | Boolean | fixed point | enumerated | bus**

参数

- **Output port dimensions (cell array) — 块输出信号的维度**
  - 选项：默认值{[1]} | 数据维度规范的元胞数组
  - 指定模块输出信号的维度，使用一个元胞数组。数组中的每个元素指定对应信号的维度，这些维度是 MATLAB 的 size 函数返回的维度。指定的维度应与模型中相应的 Byte Pack 块转换的数据对齐。

3.2. 监听 20101UDP 端口读取飞控状态数据

3.2.1. 飞控状态数据结构体和 UDP 包



实际要传输的飞控状态数据的详细结构体如下

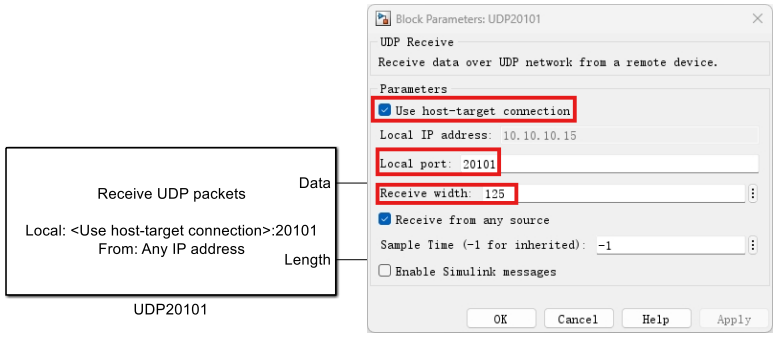
```
    struct outHILStateData{ // mavlink data forward from Pixhawk
        uint32_t time_boot_ms; //Timestamp of the message
        uint32_t copterID;     //Copter ID start from 1
        int32_t GpsPos[3];     //Estimated GPS position, lat&long: deg*1e7, alt: m*1e3 and up
is positive
```

```
        int32_t GpsVel[3];     //Estimated GPS velocity, NED, m/s*1e2->cm/s
        int32_t gpsHome[3];     //Home GPS position, lat&long: deg*1e7, alt: m*1e3 and up is p
ositive
        int32_t relative_alt;  //alt: m*1e3 and up is positive
        int32_t hdg;           //Course angle, NED,deg*1000, 0~360
        int32_t satellites_visible; //GPS Raw data, sum of satellite
        int32_t fix_type;     //GPS Raw data, Fixed type, 3 for fixed (good precision)
        int32_t resrveInit;       //Int, reserve for the future use
        float AngEular[3];    //Estimated Euler angle, unit: rad/s
        float localPos[3];    //Estimated locoal position, NED, unit: m
        float localVel[3];    //Estimated locoal velocity, NED, unit: m/s
        float pos_horiz_accuracy;   //GPS horizontal accuracy, unit: m
        float pos_vert_accuracy; //GPS vertical accuracy, unit: m
        float resrveFloat;      //float,reserve for the future use
    }
```

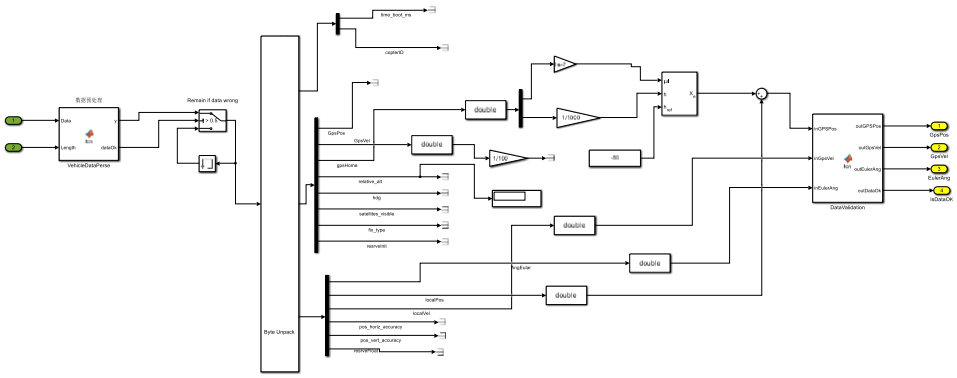
UDP 打包封装的结构体为

```
        typedef struct _netDataShortShort {
            TargetType tg; //目标端口  uint32
            int        len; //这个长度为传输结构体长度，目前是 120
            char       payload[112];//有效数据负载，存放了 outHILStateData
        }netDataShortShort;
```

UDP 包的总长度为 120

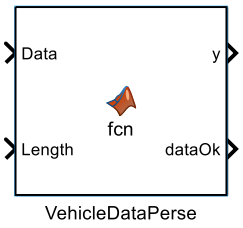


首先利用 UDP Receive 模块获取 20101 端口的 UDP 数据包，**接收宽度**（Receive width）要大于 UDP 包的总长度



这里 UDP_SIL_State_Receiver 最终会根据 UDP 包解析出位姿状态数据

3.2.2. 校验 UDP 包



其中 VehicleDataPerse 模块处理逻辑如下：

检查接收到的数据包长度

```
if Length ~= 120
    return;
end
```

- 初始检查 Length 是否等于 120，即 UDP 数据包的预期总长度。如果不是 120，则函数立即返回，dataOk 保持为 0，表示数据包不符合预期。

解析并验证有效负载长度

```
len = typecast(Data(5:8), 'int32');
```

- 通过 typecast 函数将 Data 数组中第 5 到第 8 字节转换为 int32 类型，获取数据包中声明的有效负载长度 len。

```
if len ~= 112
   return;
end
```

- 检查 len 是否等于 112，即预期的 outHILStateData 结构体的长度。如果不是 112，则函数同样立即返回，dataOk 保持为 0。

提取有效数据并设置成功标识

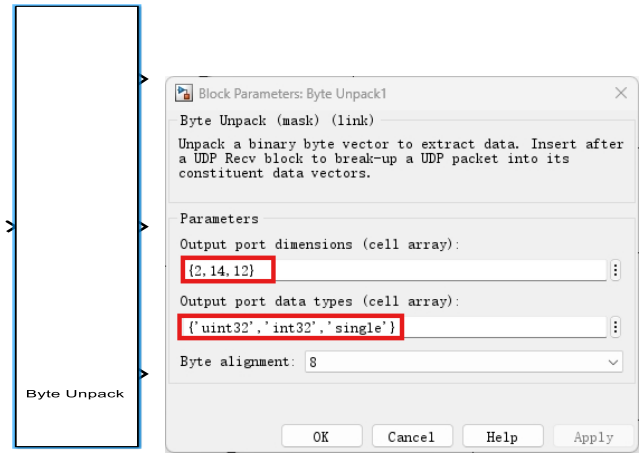
```
y = Data(9:120);
```

- 如果上述条件都满足，将 Data 中的第 9 到第 120 字节（共 112 字节）复制到 y 中。这部分数据对应于封装在 UDP 数据包中的 outHILStateData 结构体。

```
dataOk = 1;
```

- 设置 dataOk 为 1，表示数据包是有效的。

3.2.3. 字节流解包



字节流解包（Byte Unpack）处理如下：

outHILStateData 结构体中符合 uint32_t 数据类型的占有 2 字节分别是

```
    uint32_t time_boot_ms; //Timestamp of the message
    uint32_t copterID;     //Copter ID start from 1
```

outHILStateData 结构体中符合 int32_t 数据类型的占有 14 字节分别是

```
    int32_t GpsPos[3];     //Estimated GPS position, lat&long: deg*1e7, alt: m*1e3 and up is p
ositive
    int32_t GpsVel[3];     //Estimated GPS velocity, NED, m/s*1e2->cm/s
    int32_t gpsHome[3];     //Home GPS position, lat&long: deg*1e7, alt: m*1e3 and up is posit
ive
    int32_t relative_alt;  //alt: m*1e3 and up is positive
    int32_t hdg;           //Course angle, NED,deg*1000, 0~360
    int32_t satellites_visible; //GPS Raw data, sum of satellite
    int32_t fix_type;     //GPS Raw data, Fixed type, 3 for fixed (good precision)
    int32_t resrveInit;       //Int, reserve for the future use
```

outHILStateData 结构体中符合 single 数据类型的有 12 字节分别是

```
        float AngEular[3];    //Estimated Euler angle, unit: rad/s
        float localPos[3];    //Estimated locoal position, NED, unit: m
        float localVel[3];    //Estimated locoal velocity, NED, unit: m/s
        float pos_horiz_accuracy;   //GPS horizontal accuracy, unit: m
        float pos_vert_accuracy; //GPS vertical accuracy, unit: m
        float resrveFloat;      //float,reserve for the future use
```

3.2.4. 其余转换和校验

3.3. 监听 30101UDP 端口读取载具模型真值

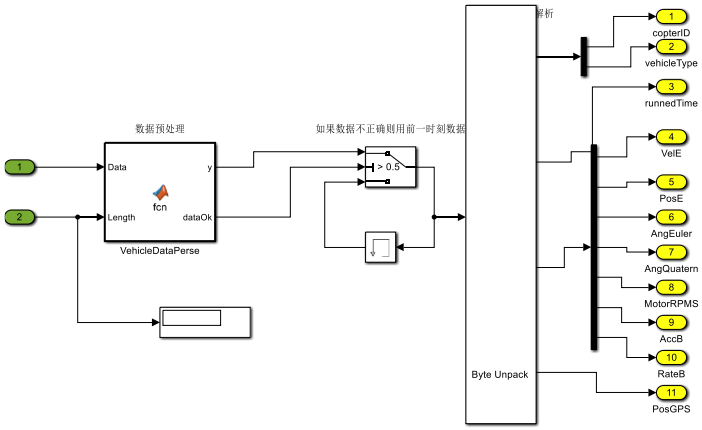
实际要传输的模型仿真真值结构体如下

```
    struct SOut2Simulator
    {
        int copterID;  //飞机 ID
        int vehicleType;  //飞机构型
        double runnedTime; //仿真时间
        float VelE[3];   //NED 地球系速度
        float PosE[3];   //NED 地球系位置
        float AngEuler[3];  //欧拉角
        float AngQuatern[4]; //四元数
        float MotorRPMS[8];  //电机转速 RPM
        float AccB[3];       //机体轴加速度
        float RateB[3];      //机体轴角速度
        double PosGPS[3];    //地球 GPS 经纬高
    }
```

UDP 打包封装的结构体为

```
    typedef struct _netDataShort {
        TargetType tg;   //目标端口  uint32
        int        len;  //这个长度为传输结构体长度，目前是 200
        char       payload[192];  //这里面前 152 位存放了 SOut2Simulator 结构体数据，后面的 40 位保留
    }netDataShort;
```

UDP 包的总长度为 200



校验和解析逻辑与 3.2 相似

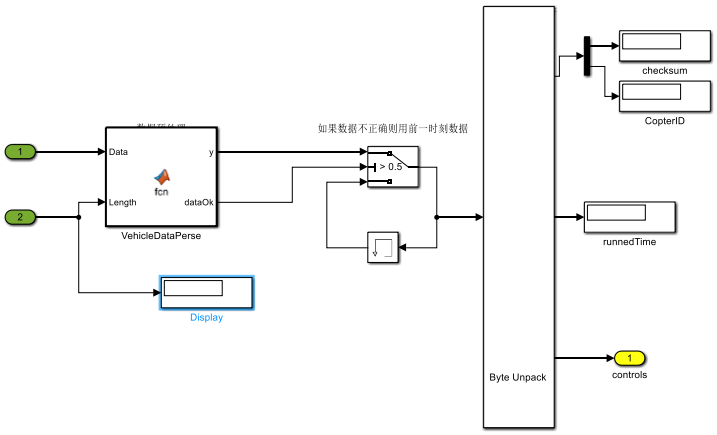
VehicleDataPerse 代码逻辑如下

1. **检查接收到的数据包长度**
   - 初始检查 Length 是否等于 200，即 UDP 数据包的预期总长度。如果不是 200，则函数立即返回，dataOk 保持为 0，表示数据包不符合预期。
2. **解析并验证有效负载长度**
   - 通过 typecast 函数将 Data 数组中第 5 到第 8 字节转换为 int32 类型，获取数据包中声明的有效负载长度 len。
   - 检查 len 是否等于 152，即预期的 SOut2Simulator 结构体的长度。如果不是 152，则函数同样立即返回，dataOk 保持为 0。
3. **提取有效数据并设置成功标识**
   - 如果上述条件都满足，将 Data 中的第 9 到第 160 字节（共 152 字节）复制到 y 中。这部分数据对应于封装在 UDP 数据包中的 SOut2Simulator 结构体。
   - 设置 dataOk 为 1，表示数据包是有效的。

3.4. 监听 40101UDP 端口读取自定义飞控消息

实际要传输的结构体如下：

```
struct PX4ExtMsg {
    int checksum;  //1234567898
    int CopterID;
    double runnedTime; //Current  stamp (s)
    float controls[8];
}
```



校验和解析逻辑与 3.2 相似

VehicleDataPerse 代码逻辑如下：

```
y=uint8(zeros(48,1));:
```

初始化 y 变量为一个 48x1 的列向量，每个元素都是 8 位无符号整型（uint8），并且初始值为 0。这意味着 y 准备存放最多 n 个字节的数据。

```
dataOk = 0;:
```

初始化 dataOk 为 0，表示默认情况下数据不是有效的。只有当数据满足特定条件时，这个值会被修改为 1。

```
if Length ~=48 :
```

检查传入的数据长度 Length 是否等于 n 。如果不等于 n，函数会直接返回，并且 data Ok 保持为 0，表示数据不符合预期长度。

```
return;:
```

如果长度检查失败，则提前结束函数执行，不再继续进行数据的其他检查或处理。

```
checksum=typecast(Data(1:4), 'int32');:
```

将 Data 数组中的 4 个字节解释为一个 32 位整数 (int32)。用于验证数据的完整性，这 4 个字节代表一个预定义的校验和。

```
if checksum ~= 1234567898:
```

检查计算出的校验和是否与预定的值（1234567898）相匹配。如果不匹配，函数同样提前返回，dataOk 保持为 0，表示数据校验失败。

```
y = Data(1:48);:
```

如果数据长度和校验和都符合要求，这行代码会将接收到的全部 n 字节数据赋值给 y。

```
dataOk = 1;:
```

数据长度正确且校验和也正确的情况下，将 dataOk 设置为 1，表示数据有效。

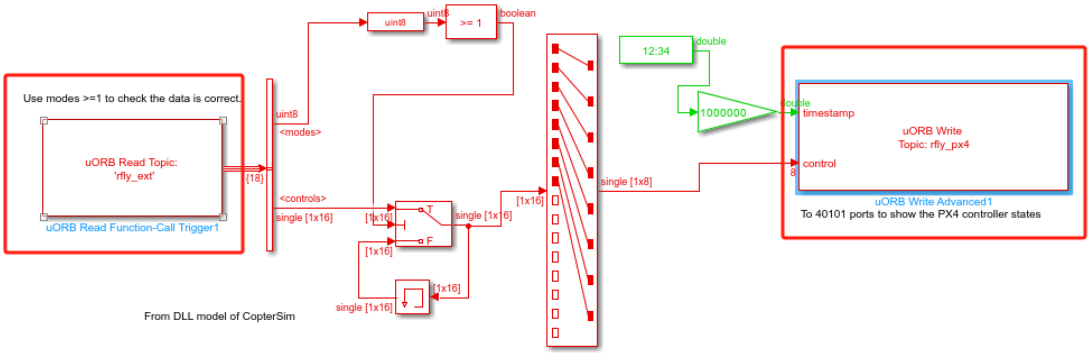
uORB 消息模块[5][6]

在 Simulink 中，uORB 是一个为了通信系统而设计的发布-订阅模型，提供了一种简单的方式去实现双方之间的消息传递。其中，uORB Read Function-Call Trigger 数据监听接口可以通过调用外部函数的形式响应接收到的新消息；uORB Write 数据发布接口允许用户向 uORB 主题发布指定的值或结构体；uORB Write Advanced 高级数据发布接口则可以允许用户对其发布的数据进行更灵活的控制。

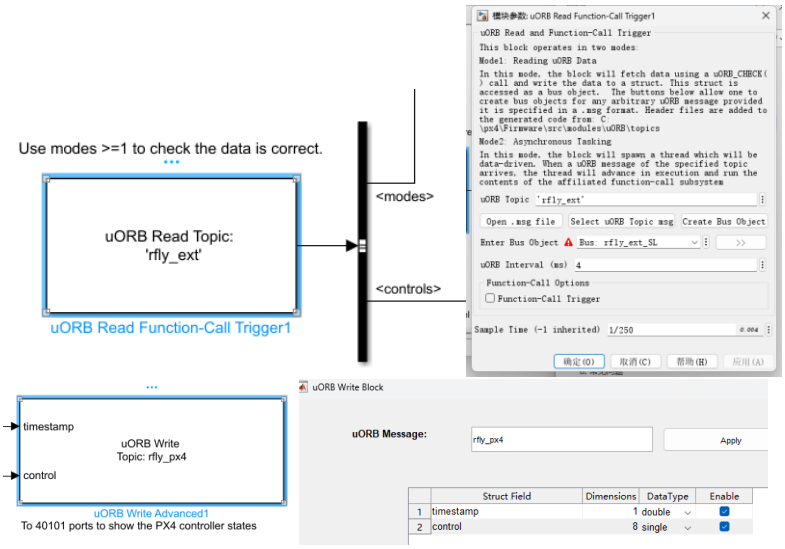
在 Simulink 中使用上述 uorb 收发接口，其参数配置包括消息类型、话题名称和消息数

据。消息类型是指要写入的消息的类型，可以在 PX4 源代码中的“msg”目录下找到相应的消息类型定义文件。话题名称（消息 ID）是指要写入的消息所对应的话题名称，可以在 PX4 源代码中的“topics”目录下找到相应的话题定义文件。消息数据是指要写入的实际数据，可以在 Simulink 模型中的其他模块中生成。RflySim 在运行一键安装脚本时，平台会对 Firmware 目录下的源码进行修改，在其中增加了 4 个 uORB 消息（rfly_ctrl.msg、rfly_ext.msg、rfly_px4.msg、rfly_insils.msg），并在*\PX4PSP\Firmware\msg\CMakeLists.txt 中已进行注册。

如下图，在 PX4ExtMsgReceiver.slx 中定义了对应的消息收发模块



这些模块只能在 PX4 飞控系统中使用，因此在使用之前需要确保已经正确地配置了 PX4 的开发环境，并且已经将 Simulink 模型成功地部署到 PX4 上；且在使用“uORB Read Function-Call Trigger”模块前，需要先在 PX4 中定义相应的 uORB 消息。在 PX4ExtMsgReceiver.slx 中，首先使用“uORB Read Function-Call Trigger1”模块订阅了 rfly_ext.msg 消息，最后将对应的消息数据在“uORB Write Advanced1”模块中通过 rfly_px4 消息发布。因此需要将 PX4ExtMsgReceiver.slx 编译生成的固件烧录到飞控中，且需要从 DLL 模型直接将消息数据传输进到 PX4 内部的 rfly_ext.msg



3.5. UDP 发送模块

3.5.1. 打包数据（Byte Pack）

参考将输入信号转换为 uint8 向量 - Simulink - MathWorks 中国

3.5.2. 发送 UDP 包（UDP Send）

参考 Send UDP message - Simulink - MathWorks 中国

4. 相关文献

[1]. Receive data over UDP network from specified remote machine - Simulink - MathWorks 中国

[2]. ..\..\..\6.RflySimExtCtrl\API.pdf

[3]. ..\..\..\3.RflySim3DUE\API.pdf

[4]. ..\..\..\9.RflySimComm\API.pdf

[5]. 自动代码生成外部通信接口../../../PX4PSP\RflySimAPIs\5.RflySimFlyCtrl/API.pdf

[6]. uORB Read and Write—uORB 消息读取和写入库../../../PX4PSP\RflySimAPIs\5.RflySimFlyCtrl/API.pdf

[7]. 40 张图带你搞懂 TCP 和 UDP (qq.com)

[8]. 平台多旋翼模型控制仿真实验原理..\..\2.AdvExps\e2_MultiModelCtrl\Intro.pdf

附加资源

官方文档：RflySim 官方文档：https://rflysim.com/doc/zh/

社区交流：加入 RflySim 技术交流群：951534390