Universidad Complutense de Madrid

Manual envío de mensajes PprzGCS

Armando David González García

4 de diciembre de 2024

1. Introducción

En este documento se detallará como cambiar los waypoints desde la estación de tierra. Se ha realizado a partir de un botón creado en la interfaz de gráfica de QT. Este recoge de un txt coordenadas latitud y longitud y mueve los diferentes waypoints del flight plan a esas coordenadas. El código creado para la planificación se encuentra en /PprzGCS/src/widgets/planificacionwindow.cpp.

2. Construcción del mensaje

Lo primero que hay que hacer es construir el mensaje que queremos mandar. Los diferentes mensajes que están definidos los podemos encontrar en el siguiente enlace con el diccionario de mensajes a fecha de 4/12/2024 (en un futuro puede que estos hayan cambiado o se hayan añadido otros).

Para mover waypoints el mensaje se tiene que definir de la siguiente manera.

```
auto messages = appConfig()->value("MESSAGES").toString();
dict = new pprzlink::MessageDictionary(messages);
pprzlink::Message msg(dict->getDefinition("MOVE_WAYPOINT"));
```

El mensaje correspondiente de mover los waypoints se llama MOVE_WAYPOINT. Una vez tenemos el mensaje hay que añadir los campos. Si vamos al xml donde están definidos los mensajes se puede ver que este mensaje tiene los siguientes campos:

- ac_id: Esto sirve para identificar con que vehículo estamos trabajando. En el documento /paparazzi/conf/airdrames/UCM/conf.xml se encuentran todos los vehículos con su ac_id correspondiente.
- wp_id: Este parámetro hace referencia al id del waypoint que queremos mover. El primer waypoint de la lista de waypoints del xml del flight plan tiene un wp_id = 1, por lo que habrá que configurar adecuadamente el xml del flight plan para poder controlar bien los puntos que queremos mover.
- lat, lon, alt: Son las coordenadas a las que queremos mover el waypoint.

Para añadir los campos hay que hacerlo de la siguiente forma:

```
pprzlink::Message msg(dict->getDefinition("MOVE_WAYPOINT"));
msg.setSenderId(pprzlink_id);
msg.addField("ac_id", ac_id);
msg.addField("wp_id", wp_id);
msg.addField("lat", lat);
msg.addField("long", lon);
msg.addField("alt", alt);
```

La linea msg.setSenderId(pprzlink_id) lo que hace es definir el id del mensaje. Cada mensaje de la lista del xml tiene su propio identificador.

3. Envío del mensaje.

Para realizar el envío del mensaje hay que tener en cuenta el código /PprzGCS/src/tools/pprz_pprzdispatcher.cpp. Es el que se encarga de emitir señales y enviar los mensajes. En concreto la función que se utiliza de este código es la siguiente:

```
    void PprzDispatcher::sendMessage(pprzlink::Message msg) {
    assert(started);
    if(!silent_mode) {
        msg.setSenderId(pprzlink_id);
        link->sendMessage(msg);
    }
}
```

Esta función lo primero que hace es comprobar si el valor de started es igual a true, en caso de que true sea false no mandará el mensaje. Por tanto también hemos tenido que crear una función en el archivo pprz_dispatcher.h para cambiar el valor a true.

```
    void setStart(bool value){
    started=value;
    }
```

4. Código para mover el waypoint desde el botón de planificación.

El código correspondiente al botón para mover los waypoints se encuentra en /PprzGCS/src/-widgets/planificacionwindow.cpp.

```
    void PlanificacionWindow::on_button_move_wp_clicked()
{
    disconnect(ui->button_move_wp, &QPushButton::clicked, this,

    &PlanificacionWindow::on_button_move_wp_clicked);
    // Obtener la ruta del archivo usando QString
    const QString filename = homeDir + "/PprzGCS/Planificacion/

    Resources/waypoints.txt"; // Cambia esta ruta al archivo real

    double latitudes[100];
    double longitudes[100];
    int max_puntos = 100; // Numero maximo de puntos

    // Llamada a la funcion para leer los puntos del archivo
    int puntos_leidos =

    leerArchivo(filename.toStdString().c_str(), latitudes,
     longitudes, max_puntos);

    // Configurar el temporizador para enviar los puntos uno por uno
    currentIndex = 0;  // Reiniciar el indice de los puntos
    timer = new QTimer(this);

    // Conectar el timeout al envio de puntos
    connect(timer, &QTimer::timeout, [=]() {
        if (currentIndex < puntos_leidos) {
            double latitud = latitudes[currentIndex];
            double longitud = longitudes[currentIndex];
```

```
            sendwp(latitud, longitud);  // envio de wp
                    currentIndex++;
        } else {
            timer->stop();  // Detener el temporizador
            timer->deleteLater();
            qDebug() << "Todos los puntos han sido enviados.";
        }
    });

    // Iniciar el temporizador con un intervalo de 1 segundo (1000 ms)
    timer->start(1000);

    //}
}
```

En primer lugar, añadimos un disconnect para que en caso de que esta función se esté ejecutando en un segundo plano por haber hecho un doble click o por cualquier otro motivo, se paren las demás y lo ejecute solo una vez. A continuación, definimos los vectores donde guardaremos los valores de latitud y longitud y el número máximo de puntos. Una vez tenemos definidas las variables llamamos a la función encargada de leer el txt y almacenar los valores de las coordenadas en los vectores. Esta función realiza lo siguiente:

```
int PlanificacionWindow::leerArchivo
(const char *filename, double lat[], double lon[], int max_puntos)
{
FILE *file = fopen(filename, "r");
if (file == NULL) {
    perror("Error al abrir el archivo");
    return -1;
}

int count = 0;
char linea[256]; // Para leer lineas completas
char nombre[50];
char slat[50];
char slon[50];
// Ignorar la primera linea (encabezado)
fgets(linea, sizeof(linea), file);

// Leer cada linea y parsear los valores
while (count < max_puntos && fgets(linea, sizeof(linea), file)) {
//        qDebug() << "Leyendo linea: " << linea; // Depuracion

    sscanf(linea, "%s\t%s\t%s\n", nombre, slat, slon);

    sscanf(slat, "%lf", &lat[count]);
    sscanf(slon, "%lf", &lon[count]);
;
```

```
        count++;
    }

    fclose(file);
    return count; // Numero de puntos leidos
}
```

Resumidamente, la función leerArchivo ignora el encabezado y va guardando los valores de lat y lon en los diferentes vectores y te devuelve el número puntos que ha leido.

Una vez se ha leido el txt y tenemos las coordenadas en los vectores configuramos un temporizador. Este temporizador se encagará de enviar una señal cada cierto tiempo. Con el connect conectamos esta señal enviada por el timer con una función lambda. La función lambda comprueba que el número de puntos enviados sea menor que el total de puntos que hay que enviar. Por un lado, si esto es cierto llamamos a la función sendwp.

```
void PlanificacionWindow::sendwp(double latitud, double longitud){

// Recorrer los puntos leidos y enviar un mensaje
auto messages = appConfig()->value("MESSAGES").toString();
dict = new pprzlink::MessageDictionary(messages);


double lat;
double lon;
float alt;
PprzDispatcher::get()->setStart(true);

// Recorrer cada par de coordenadas
QString ac_id = "4";
quint8 wp_id = i+8;
lat = latitud; // Convertir a formato de latitud/longitud
lon = longitud; // Convertir a formato de latitud/longitud
alt = 660.7;

pprzlink::Message msg(dict->getDefinition("MOVE_WAYPOINT"));
msg.setSenderId(pprzlink_id);
msg.addField("ac_id", ac_id);
msg.addField("wp_id", wp_id);
msg.addField("lat", lat);
msg.addField("long", lon);
msg.addField("alt", alt);


// Enviar el mensaje para este waypoint

PprzDispatcher::get()->sendMessage(msg);

//printf("Latitud enviada %11.8lf\n", lat);
i++;
```

}

Esta función empieza llamando a setStart para poner started en true y a continuación construye el mensaje como se ha comentado en la primera sección. Una vez está el mensaje construido llama a la función sendMessage de PprzDispathcer y envía el mensaje.

Por otro lado, si cuando se envía la señal del timer el número de puntos enviados es mayor o igual que el total de puntos que tenemos significa que ya se han enviado todos los puntos por tanto para el timer y lo elimina. La última linea del código timer→start(1000) sirve para definir la frecuencia del envío del mensaje en milisegundos. En este caso está definido para que se envíe uno cada segundo pero se puede cambiar para que envíe con una mayor o menor frecuencia.