ANEXO 2. DESCRIPCIÓN DE LAS FUNCIONES Y LOS MÉTODOS IMPLEMENTADOS PARA EL GENERADOR DE TRAYECTORIAS.

A continuación se hace una breve descripción de las clases y las funciones que han sido implementadas en esta parte del proyecto. Más información sobre las mismas puede encontrarse en el propio código.

A2.1. Métodos de la clase de comunicaciones (CGenCommunication).

CGenCommunication::CGenCommunication()

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Constructor por defecto de la clase CParser. Reserva memoria.

CGenCommunication::~CGenCommunication()

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Destructor de la clase. Libera la memoria reservada.

bool CGenCommunication::Init(void)

Argumentos de entrada:

Ninguno.

Argumento de salida:

Variable booleana que indica si la operación ha sido o no correcta.

Descripción:

Inicializa las comunicaciones y abre los slots.

bool CGenCommunication::ReceiveWayPointsList(GenCriticalSection *GenCS)

Argumentos de entrada:

*GenCriticalSection *GenCS:*
Puntero a clase GenCriticalSection.

Argumento de salida:

Variable booleana que indica si la operación ha sido o no correcta.

Descripción:

El método recoge una lista de waypoints del BBCS y los guarda en la sección crítica.

bool CGenCommunication::SendTrajectory(GenCriticalSection *GenCS)

Argumentos de entrada:

*GenCriticalSection *GenCS:*
Puntero a clase GenCriticalSection.

Argumento de salida:

Variable booleana que indica si la operación ha sido o no correcta.

Descripción:

Al contrario que el anterior, este método saca un buffer de trayectorias de la sección crítica, y lo envía al slot que le corresponda.

bool CGenCommunication::InitCommunications(void)

Argumentos de entrada:

Ninguno.

Argumento de salida:

Variable booleana que indica si la operación ha sido o no correcta.

Descripción:

Método que inicializa los slots emisores que el proceso requiera.

A2.2. Métodos de la clase de la sección crítica (GenCriticalSection).

GenCriticalSection::GenCriticalSection()

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Constructor de la clase. Reserva memoria e inicializa variables.

GenCriticalSection::~GenCriticalSection()

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Destructor de la clase. Libera memoria.

void GenCriticalSection::SetDatoGenTraj(DATO_GEN_TRAJ *WPlist)

Argumentos de entrada:

*DATO_GEN_TRAJ *WPlist:*
Estructura del tipo DATO_GEN_TRAJ con una lista de waypoints.

Argumento de salida:

Ninguno.

Descripción:

Este método guarda la estructura que recibe en la sección crítica.

bool GenCriticalSection::GetDatoGenTraj(DATO_GEN_TRAJ *WPlist)

Argumentos de entrada:

*DATO_GEN_TRAJ *WPlist:*
Estructura del tipo DATO_GEN_TRAJ.

Argumento de salida:

Variable booleana.

Descripción:

Si hay nuevos datos, éstos se recogerán de la sección crítica, y se guardarán en la estructura a la que apunta la variable de entrada. En caso contrario no lo guarda. Devolverá *'true'* si hay nuevos datos, o *'false'* si no los hay.

void GenCriticalSection::SetBufferTraj(char *buffer_traj, int size)

Argumentos de entrada:

*char *buffer_traj:*
Puntero a un buffer que contiene una trayectoria.

int size:
Entero con el tamaño del buffer.

Argumento de salida:

Ninguno.

Descripción:

Este método guarda en la sección crítica el buffer con la trayectoria que se ha calculado.

char *GenCriticalSection::GetBufferTraj(char *buffer_traj, int *size, bool *flag)

Argumentos de entrada:

*char *buffer_traj:*
Buffer donde guardar la trayectoria.

*int *buffer_traj_size:*
Puntero a entero que dice el tamaño del buffer.

*bool *flag:*
Puntero a variable booleana que dice si hay nueva información o no.

Argumento de salida:

Puntero a cadena con la dirección del buffer.

Descripción:

Este método recoge la trayectoria de la sección crítica cuando la haya. Como el buffer puede ser realojado, el método ha de devolver un puntero al mismo.

A2.3. Funciones del hilo de comunicación.

void *CommunicationThread(void *)

Argumentos de entrada:

*void *:*
Ninguno.

Argumento de salida:

Puntero obligatorio en las rutinas de los hilos.

Descripción:

Rutina del hilo de comunicaciones.

A2.4. Funciones del hilo de cálculo de la trayectoria.

void *AlgorithmThread(void *)

Argumentos de entrada:

*void *:*
Ninguno.

Argumento de salida:

Puntero obligatorio en las rutinas de los hilos.

Descripción:

Rutina del hilo de cálculo de la trayectoria.

char *GenerateSimpleTrajectory(DATO_GEN_TRAJ WPlist, char *buffer, int *p_size)

Argumentos de entrada:

DATO_GEN_TRAJ WPlist:
Lista de waypoints a partir de los cuales se debe generar la trayectoria.

*char *buffer:*
Puntero a un buffer donde almacenar la trayectoria.

*int *p_size:*
Puntero a un entero donde se guarda el tamaño del buffer.

Argumento de salida:

Puntero a char que llevará la dirección del buffer.

Descripción:

Se trata de una función que calcula una trayectoria simple. Pasa directamente los waypoints que le llegan como argumento a formato de trayectoria, guardando la misma en el buffer. Escribe en la dirección *'p_size'* el tamaño del buffer, y devuelve su dirección, ya que éste ha podido ser realojado.

char *GenerateSplineTrajectory(DATO_GEN_TRAJ WPlist, char *buffer, int *p_size)

Argumentos de entrada:

DATO_GEN_TRAJ WPlist:
Lista de waypoints a partir de los cuales se debe generar la trayectoria.

*char *buffer:*
Puntero a un buffer donde almacenar la trayectoria.

*int *p_size:*
Puntero a un entero donde se guarda el tamaño del buffer.

Argumento de salida:

Puntero a char que llevará la dirección del buffer.

Descripción:

Se trata de una función que calcula una trayectoria mediante un algoritmo basado en splines cúbicos, guardando dicha trayectoria misma en el buffer. Escribe en la dirección *'p_size'* el tamaño del buffer, y devuelve su dirección, ya que éste ha podido ser realojado.

int CalculateDistances(double *p_dist, DATO_PUNTO_SEG_CAM *p_resol_list, int length)

Argumentos de entrada:

*double *p_dist:*
Tabla de flotantes de doble precisión donde guardar los resultados de los cálculos.

*DATO_PUNTO_SEG_CAM *p_resol_list:*
Puntero a lista de waypoints.

int length:
Número de waypoints.

Argumento de salida:

Entero que valdrá 0 si todo ha ido bien.

Descripción:

Esta función calcula las distancias entre cada punto y el primero, pasando en línea recta por todos los anteriores, y sin contar la altura de los mismos.

int CalculateParams(SPLINE_PARAMETERS *p_lat_params, SPLINE_PARAMETERS *p_long_params, SPLINE_PARAMETERS *p_z_params, double *p_dist, DATO_PUNTO_SEG_CAM *p_resol_list, int length)

Argumentos de entrada:

*SPLINE_PARAMETERS *p_lat_params:*
Puntero a estructura del tipo SPLINE_PARAMETERS, para los parámetros del spline de la latitud.

*SPLINE_PARAMETERS *p_long_params:*
Idem longitud.

*SPLINE_PARAMETERS *p_z_params:*
Idem altura, aunque las curva de la altura no será un spline estrictamente hablando.

*double *p_dist:*
Tabla de dobles con las distancias.

*DATO_PUNTO_SEG_CAM *p_resol_list:*
Puntero a lista de waypoints.

int length:
Número de waypoints.

Argumento de salida:

Entero que valdrá 0 si todo ha ido bien.

Descripción:

La función calcula los parámetros de los splines para la latitud y la longitud, así como los parámetros para las funciones cúbicas que seguirá la altura.

void CalculateNumPoints(SPLINE_PARAMETERS lat_params, SPLINE_PARAMETERS long_params, double t0, double tf, unsigned int *number_of_points)

Argumentos de entrada:

SPLINE_PARAMETERS lat_params:
Estructura del tipo SPLINE_PARAMETERS, con los parámetros del spline de la latitud.

SPLINE_PARAMETERS long_params:
Idem longitud.

double t0:
Doble que dice el punto inicial del intervalo sobre el que se van a hacer los cálculos.

double tf:
Idem punto final del intervalo.

*unsigned int *number_of_points:*
Puntero a entero sin signo donde se le sumarán los puntos que se hallen para el intervalo en cuestión.

Argumento de salida:

Ninguno.

Descripción:

Esta función calcula el número de puntos intermedios (incluido el inicial) que tiene que haber entre cada dos waypoints en una trayectoria tipo spline.

void CalculateIntermPoints(SPLINE_PARAMETERS lat_params, SPLINE_PARAMETERS long_params, double t0, double tf, double *p_t)

Argumentos de entrada:

SPLINE_PARAMETERS lat_params:
Estructura del tipo SPLINE_PARAMETERS, con los parámetros del spline de la latitud.

SPLINE_PARAMETERS long_params:
Idem longitud.

double t0:
Doble que dice el punto inicial del intervalo sobre el que se van a hacer los cálculos.

double tf:
Idem punto final del intervalo.

*double *p_t:*
Tabla de flotantes de precisión doble donde se guardarán los puntos intermedios del intervalo.

Argumento de salida:

Ninguno.

Descripción:

La función calcula la ubicación de los puntos intermedios del intervalo.

double SegGrad(double A, double B, double C, double pend, double t0, double tf);

Argumentos de entrada:

double A, double B, double C:
Flotantes de doble precisión con los tres primeros parámetros de una ecuación de tercer grado.

double pend:
Flotante de doble precisión que dice la pendiente en un punto .

double t0:
Doble que dice el punto inicial del intervalo sobre el que se van a hacer los cálculos.

double tf:
Idem punto final del intervalo.

Argumento de salida:

Doble con el resultado de los cálculos.

Descripción:

La función resuelve la ecuación de segundo grado: $At^2+Bt+C=arctg(pend)$, eligiendo la solución que esté en el intervalo dado. Se entiende que los datos se dan de manera que en tal intervalo se encuentre siempre una solución, y sólo una.

void CalculateCoordTraj(DATO_PUNTO_SEG_CAM *p_dpsc, DATO_PUNTO_SEG_CAM *p_resol_list, SPLINE_PARAMETERS *p_lat_params, SPLINE_PARAMETERS *p_long_params, SPLINE_PARAMETERS *p_z_params, double *p_dist, double *p_t, int length)

Argumentos de entrada:

*DATO_PUNTO_SEG_CAM *p_dpsc:*
Puntero a estructura de tipo DATO_PUNTO_SEG_CAM, donde se guardará la latitud, la longitud y la altura del punto intermedio.

*DATO_PUNTO_SEG_CAM *p_resol_list:*
Puntero a lista de waypoints.

*SPLINE_PARAMETERS *p_lat_params:*
Puntero a estructura del tipo SPLINE_PARAMETERS, para los parámetros del spline de la latitud.

*SPLINE_PARAMETERS *p_long_params:*
Idem longitud.

*SPLINE_PARAMETERS *p_z_params:*

Idem altura, aunque las curva de la altura no será un spline estrictamente hablando.

*double *p_t:*

Tabla de flotantes de precisión doble donde se guardarán los puntos intermedios del intervalo.

int length:

Número de waypoints.

Argumento de salida:

Ninguno.

Descripción:

La función calcula la latitud, longitud y altura en un punto intermedio.