ANEXO 1. DESCRIPCIÓN DE LAS FUNCIONES Y LOS MÉTODOS IMPLEMENTADOS PARA EL PROCESO UAV.

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.

A1.1. Métodos de la clase de lectura del archivo de configuración (CParser).

CParser::CParser(void)

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Constructor por defecto de la clase CParser. Inicializa a cero o NULL algunas variables.

CParser::CParser(char *Filename)

Argumentos de entrada:

*char *Filename:*
Cadena de caracteres con el nombre de un fichero de configuración.

Argumento de salida:

Ninguno.

Descripción:

Constructor alternativo de la clase CParser. Abre el fichero de configuración que se le pase como argumento, reservando memoria y guardando en ella la lista de comandos que lea del fichero.

CParser::~CParser(void)

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Destructor de la clase CParser. Libera la memoria reservada.

bool CParser::SetFile(char *File)

Argumentos de entrada:

*char *File:*
Cadena de caracteres con el nombre de un archivo de configuración.

Argumento de salida:

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

Descripción:

Abre el archivo de configuración, reserva memoria y guarda el contenido del fichero en el buffer.

bool CParser::CreateCommandsList(void)

Argumentos de entrada:

Ninguno.

Argumento de salida:

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

Descripción:

A partir del buffer crea una lista de comandos. Ésta es una lista de cadenas de caracteres, en cuyos elementos se encuentran los comandos del fichero de configuración.

void CParser::GetMinuscule(char *string)

Argumentos de entrada:

*char *string:*
Cadena de caracteres.

Argumento de salida:

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

Descripción:

El método recoge una cadena de caracteres, y la modifica, pasando a minúsculas todas las letras mayúsculas que contuviera.

int CParser::ParseFile(CONFIG *datos)

Argumentos de entrada:

*CONFIG *datos:*
Puntero a estructura de tipo CONFIG, que será rellenada con los datos de configuración que vengan en el archivo.

Argumento de salida:

Entero que indicará si la operación ha ido bien (SUCCESS), o si a habido algún error (ERROR).

Descripción:

Se trata del método principal de la clase. Es el que lee la lista de comandos y la interpreta, rellenando los campos de la estructura datos.

int CParser::GetCtrlParam(PID *param_pid, int Index)

Argumentos de entrada:

*PID *param_pid:*
Puntero a estructura de tipo PID, donde se guardan los parámetros de un controlador tipo PID.

int Index:
Entero que recoge el índice de lectura de la lista de comandos.

Argumento de salida:

Entero que indicará si la operación ha ido bien (SUCCESS), o si ha habido algún error (ERROR).

Descripción:

Ésta es una subfunción de *'ParseFile'* que almacena los parámetros de los controladores en sus respectivas subestructuras, dentro de la estructura CONFIG.

A1.2. Métodos de la clase comunicacion.

void comunicacion::comunicacion(void)

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Constructor de la clase configuracion. Declara los mutex y las variables de condición, inicializa variables, listas y reserva memoria.

void comunicacion::~comunicacion(void)

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Destructor de la clase configuracion. Libera memoria y limpia las listas.

void comunicacion::meter_telemetria(UAV_TELEMETRY UavTelemetry_ext)

Argumentos de entrada:

UAV_TELEMETRY UavTelemetry_ext:
Estructura del tipo UAV_TELEMETRY con la telemetría que se ha recogido del DSP.

Argumento de salida:

Ninguno.

Descripción:

Este método guarda la estructura que recibe en la sección crítica, que estará protegida por mutex.

void comunicacion::sacar_telemetria(UAV_TELEMETRY *p_UavTelemetry_ext, char *flag_telemetria_ext)

Argumentos de entrada:

*UAV_TELEMETRY *p_UavTelemetry_ext:*
Puntero a estructura del tipo UAV_TELEMETRY.

*char *flag_telemetria_ext:*
Puntero que apunta a un char utilizado como flag.

Argumento de salida:

Ninguno.

Descripción:

Si hay nueva telemetría, este método recoge la estructura de la sección crítica, guardándola en la estructura a la que apunta el primer argumento, poniendo el flag a uno. En caso de no existir datos nuevos no hace nada.

void comunicacion::meter_info_pan_tilt(E_S_PAN_TILT ESPanTilt_ext, int recepcion_ext)

Argumentos de entrada:

E_S_PAN_TILT ESPanTilt_ext:
Estructura del tipo E_S_PAN_TILT, donde está la información referente al pan & tilt recibida por BBCS.

int recepcion_ext:
Entero con el que se señala qué variables se han recibido en el último muestreo.

Argumento de salida:

Ninguno.

Descripción:

Este método guarda en la sección crítica los campos de la estructura que haya recibido. Hay protección por mutex.

void comunicacion::sacar_info_pan_tilt(E_S_PAN_TILT *p_ESPanTilt_ext, int *recepcion_ext)

Argumentos de entrada:

*E_S_PAN_TILT *p_ESPanTilt_ext:*
Puntero a estructura del tipo E_S_PAN_TILT.

*int *recepcion_ext:*
Entero que señala qué datos nuevos se han recibido.

Argumento de salida:

Ninguno.

Descripción:

Este método recoge los nuevos datos de la sección crítica, guardándolos en la estructura a la que apunta el primer argumento. Se modifica el entero al que apunta el segundo argumento de manera que señale qué nuevos datos se han recibido.

void comunicacion::meter_error_pan_tilt(int ErrorPanTilt_ext)

Argumentos de entrada:

int ErrorPanTilt_ext:
Entero con el número de error que haya surgido.

Argumento de salida:

Ninguno.

Descripción:

En el momento en el que haya un error se llama a este método para encolarlo en la sección crítica.

void comunicacion::sacar_error_pan_tilt(E_S_PAN_TILT *p_ESPanTilt_ext, char * flag_ext)

Argumentos de entrada:

*E_S_PAN_TILT *p_ESPanTilt_ext:*
Puntero a estructura tipo E_S_PAN_TILT donde se guardará el error que se saque de la sección crítica.

*char * flag_ext:*
Puntero a char que actúa como un flag que indicará si hay o no errores en la lista de errores.

Argumento de salida:

Ninguno.

Descripción:

El método guardará el error que haya en la lista de errores en la estructura a la que apunte p_ESPanTilt_ext, si es que hay alguno. La existencia o no se le indicará al hilo invocador en el segundo argumento.

void comunicacion::meter_info_seg_cam(E_S_SEG_CAM *p_ESSegCam_ext, int recepcion_ext, int tam_actual, char *buffer_datos_ext)

Argumentos de entrada:

*E_S_SEG_CAM *p_ESSegCam_ext:*
Estructura del tipo E_S_SEG_CAM, donde está la información referente al pan & tilt recibida por BBCS.

int recepcion_ext:
Entero con el que se señala qué variables se han recibido en el último muestreo.

int tam_actual:
Entero que dice el tamaño del buffer en ese momento.

*char *buffer_datos_ext:*
Buffer necesario para recoger la información del slot DATOS.

Argumento de salida:

Ninguno.

Descripción:

Este método guarda en la sección crítica los campos de la estructura que haya recibido, así como el buffer de datos. Realojará el buffer de la sección crítica en caso de ser necesario. Hay protección por mutex.

char * comunicacion::sacar_info_seg_cam(E_S_SEG_CAM *p_ESSegCam_ext, char *buffer_datos_ext, int *tam_actual_ext, int *recepcion_ext)

Argumentos de entrada:

*E_S_SEG_CAM *p_ESSegCam_ext:*
Puntero a estructura del tipo E_S_SEG_CAM.

*char *buffer_datos_ext:*
Buffer donde se soltrá la información del slot DATOS.

*E_S_PAN_TILT *p_ESPanTilt_ext:*
Puntero a estructura del tipo E_S_PAN_TILT.

*int *tam_actual_ext:*
Puntero a entero donde se escribirá el tamaño del buffer.

*int *recepcion_ext:*
Puntero a entero que dirá qué datos hay nuevos.

Argumento de salida:

Puntero a char, que contendrá la dirección del buffer, ya que éste ha podido ser realojado.

Descripción:

Este método recoge los nuevos datos de la sección crítica, guardándolos en la estructura a la que apunta el primer argumento, y en el buffer para el caso del slot de DATOS. Se modifica el entero al que apunta el último argumento de manera que señale qué nuevos datos se han recibido. El buffer de entrada puede ser demasiado pequeño, en cuyo caso se realojaría, se variaría el entero al que apunta *'tam_actual_ext'*, y se devuelve su dirección como argumento de salida.

void comunicacion::meter_error_seg_cam(int ErrorSegCam_ext)

Argumentos de entrada:

int ErrorSegCam_ext:
Entero con el número de error que haya surgido.

Argumento de salida:

Ninguno.

Descripción:

En el momento en el que haya un error se llama a este método para encolarlo en la sección crítica.

void comunicacion :: sacar_error_seg_cam (E_S_SEG_CAM *p_ESSegCam_ext, char *flag_ext)

Argumentos de entrada:

*E_S_SEG_CAM *p_ESSegCam_ext:*
Puntero a estructura tipo E_S_SEG_CAM donde se guardará el error que se saque de la sección crítica.

*char * flag_ext:*
Puntero a char que actúa como un flag que indicará si hay o no errores en la lista de errores.

Argumento de salida:

Ninguno.

Descripción:

El método guardará el error que haya en la lista de errores en la estructura a la que apunte p_ESSegCam_ext, si es que hay alguno. La existencia o no se le indicará al hilo invocador en el segundo argumento.

A1.3. Funciones para el hilo de comunicaciones (hilo_coms.cpp).

void iniciar_hilo_coms(POINTERS *p_pointers)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura tipo POINTERS, que apunto a una estructura con clases y subestructuras de uso general.

Argumento de salida:

Ninguno.

Descripción:

Esta función lanza el hilo de comunicaciones del proceso UAV.

void * funcion_hilo_coms(void *args)

Argumentos de entrada:

*void *args:*
Puntero a void, obligatorio en las rutinas de los hilos. En este caso el puntero lo será a la estructura POINTERS que se le pasara a *'iniciar_hilo_coms'*.

Argumento de salida:

Puntero a void obligatorio para este tipo de rutinas.

Descripción:

Se trata de la rutina principal del hilo de comunicaciones.

void destruir_hilo_coms(void)

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Cancela la ejecución del hilo de comunicaciones.

void anhadir_slots (POINTERS *p_pointers, UAV_TELEMETRY *p_UavTelemetry, E_S_PAN_TILT *p_ESPanTilt, E_S_SEG_CAM *p_ESSegCam)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a la estructura POINTERS, descrita anteriormente.

*UAV_TELEMETRY *p_UavTelemetry, E_S_PAN_TILT *p_ESPanTilt, E_S_SEG_CAM *p_ESSegCam:*
Punteros a todos los tipos de variables que vayan a entrar o salir por BBCS.

Argumento de salida:

Ninguno.

Descripción:

Subprograma que añade todos los slots que el hilo pueda necesitar.

int comprobar_slots_pt (E_S_PAN_TILT *p_ESPanTilt)

Argumentos de entrada:

*E_S_PAN_TILT *p_ESPanTilt:*
Puntero a estructura de datos de entrada/salida del pan & tilt.

Argumento de salida:

Entero que indica los slots cuyo número es correcto.

Descripción:

El hilo de comunicaciones recibe por un slot una estructura con los números de varios slots. Esta función comprueba que esos números son correctos.

int comprobar_slots_seg_cam (E_S_SEG_CAM *p_ESSegCam)

Argumentos de entrada:

*E_S_SEG_CAM *p_ESSegCam:*
Puntero a estructura de datos de entrada/salida del seguidor.

Argumento de salida:

Entero que indica los slots cuyo número es correcto.

Descripción:

El hilo de comunicaciones recibe por un slot una estructura con los números de varios slots. Esta función comprueba que esos números son correctos.

A1.4. Funciones para el hilo de telemetría (telemetria.cpp).

void iniciar_hilo_telemetria(POINTERS *p_pointers)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura tipo POINTERS, que apunto a una estructura con clases y subestructuras de uso general.

Argumento de salida:

Ninguno.

Descripción:

Esta función lanza el hilo de telemetría del proceso UAV.

void * funcion_hilo_telemetria(void *args)

Argumentos de entrada:

*void *args:*
Puntero a void, obligatorio en las rutinas de los hilos. En este caso el puntero lo será a la estructura POINTERS que se le pasara a *'iniciar_hilo_telemetria'*.

Argumento de salida:

Puntero a void obligatorio para este tipo de rutinas.

Descripción:

Se trata de la rutina principal del hilo de telemetría.

void destruir_hilo_telemetria(void)

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Cancela la ejecución del hilo de telemetría.

A1.5. Funciones para el hilo para el pan & tilt(pan_tilt.cpp).

void iniciar_hilo_pan_tilt(POINTERS *p_pointers)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura tipo POINTERS, que apunto a una estructura con clases y subestructuras de uso general.

Argumento de salida:

Ninguno.

Descripción:

Esta función lanza el hilo controlador de pan & tilt del proceso UAV.

void * funcion_hilo_pan_tilt(void *args)

Argumentos de entrada:

*void *args:*
Puntero a void, obligatorio en las rutinas de los hilos. En este caso el puntero lo será a la estructura POINTERS que se le pasara a *'iniciar_hilo_pan_tilt'*.

Argumento de salida:

Puntero a void obligatorio para este tipo de rutinas.

Descripción:

Se trata de la rutina principal del hilo del pan & tilt.

void destruir_hilo_pan_tilt(void)

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Cancela la ejecución del hilo del pan & tilt.

int comprobar_datos_pan_tilt(POINTERS *p_pointers, E_S_PAN_TILT *p_ESPanTilt)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura POINTERS.

*E_S_PAN_TILT *p_ESPanTilt:*
Puntero a estructura E_S_PAN_TILT.

Argumento de salida:

Entero con el código del error, o 0 si no se ha producido ninguno.

Descripción:

Comprueba la información recibida por todos los slots del pan & tilt antes de que sea enviada al DSP. Si hay algún dato erróneo encola el error. Devolverá el código del último error detectado, o un cero en caso de no detectar ninguno.

void enviar_modo_dsp_pan_tilt(POINTERS *p_pointers, E_S_PAN_TILT *p_ESPanTilt)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura POINTERS.

*E_S_PAN_TILT *p_ESPanTilt:*
Puntero a estructura E_S_PAN_TILT.

Argumento de salida:

Ninguno.

Descripción:

Envía al DSP la información recibida por el slot CONTROL MODO del pan & tilt.

void enviar_datos_dsp_pan_tilt(POINTERS *p_pointers, E_S_PAN_TILT *p_ESPanTilt)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura POINTERS.

*E_S_PAN_TILT *p_ESPanTilt:*
Puntero a estructura E_S_PAN_TILT.

Argumento de salida:

Ninguno.

Descripción:

Envía al DSP la información recibida por el slot DATOS del pan & tilt.

A1.6. Funciones para el hilo del seguidor (seguidor.cpp).

void iniciar_hilo_seguidor(POINTERS *p_pointers)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura tipo POINTERS, que apunto a una estructura con clases y subestructuras de uso general.

Argumento de salida:

Ninguno.

Descripción:

Esta función lanza el hilo seguidor/puente del proceso UAV.

void * funcion_hilo_seguidor(void *args)

Argumentos de entrada:

*void *args:*
Puntero a void, obligatorio en las rutinas de los hilos. En este caso el puntero lo será a la estructura POINTERS que se le pasara a *'iniciar_hilo_seguidor'*.

Argumento de salida:

Puntero a void obligatorio para este tipo de rutinas.

Descripción:

Se trata de la rutina principal del hilo del seguidor.

void destruir_hilo_seguidor(void)

Argumentos de entrada:

Ninguno.

Argumento de salida:

Ninguno.

Descripción:

Cancela la ejecución del hilo del seguidor.

int comprobar_datos_seg_cam(POINTERS *p_pointers, E_S_SEG_CAM *p_ESSegCam, char *buffer_datos, unsigned int tamanho, int recepcion)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura POINTERS.

*E_S_SEG_CAM *p_ESSegCam:*
Puntero a estructura E_S_SEG_CAM.

*char *buffer_datos:*
Puntero a un buffer con datos sobre una trayectoria.

unsigned int tamanho:
Entero sin signo con el tamaño del buffer anterior.

int recepcion:
Entero que indica los datos que se han recibido en el último muestreo.

Argumento de salida:

Entero con el código del error, ó 0 si no se ha producido ninguno.

Descripción:

Comprueba la información recibida por todos los slots del seguidor antes de que sea enviada al DSP. Si hay algún dato erróneo encola el error. Devolverá el código del último error detectado, o un cero en caso de no detectar ninguno.

void enviar_modo_dsp_seg_cam(POINTERS *p_pointers, E_S_SEG_CAM *p_ESSegCam)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura POINTERS.

*E_S_SEG_CAM *p_ESSegCam:*
Puntero a estructura E_S_SEG_CAM.

Argumento de salida:

Ninguno.

Descripción:

Envía al DSP la información recibida por el slot CONTROL MODO del seguidor.

void enviar_heading_dsp_seg_cam(POINTERS *p_pointers, E_S_SEG_CAM *p_ESSegCam)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura POINTERS.

*E_S_SEG_CAM *p_ESSegCam:*
Puntero a estructura E_S_SEG_CAM.

Argumento de salida:

Ninguno.

Descripción:

Envía al DSP la información recibida por el slot HEADING del seguidor.

void enviar_velocidad_dsp_seg_cam(POINTERS *p_pointers, E_S_SEG_CAM *p_ESSegCam)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura POINTERS.

*E_S_SEG_CAM *p_ESSegCam:*
Puntero a estructura E_S_SEG_CAM.

Argumento de salida:

Ninguno.

Descripción:

Envía al DSP la información recibida por el slot VELOCIDAD del seguidor.

void enviar_datos_dsp_seg_cam(POINTERS *p_pointers, char *buffer_datos, E_S_SEG_CAM *p_ESSegCam)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura POINTERS.

*char *buffer_datos:*
Puntero a un buffer con datos sobre waypoints.

*E_S_SEG_CAM *p_ESSegCam:*
Puntero a estructura E_S_SEG_CAM.

Argumento de salida:

Ninguno.

Descripción:

Decodifica y envía al DSP la información recibida por el slot DATOS del seguidor.

void enviar_refs_dsp_seg_cam(POINTERS *p_pointers, E_S_SEG_CAM *p_ESSegCam)

Argumentos de entrada:

*POINTERS *p_pointers:*
Puntero a estructura POINTERS.

*E_S_SEG_CAM *p_ESSegCam:*
Puntero a estructura E_S_SEG_CAM.

Argumento de salida:

Ninguno.

Descripción:

Envía al DSP la información recibida por el slot REFS del seguidor.