1 Sonar

1.1 Sonar.h

```
/* ************** */
/* Fitxer sonar.h */
/* ************** */

// Funcions utilitzades per controlar els sonars

#ifndef _SONAR_H
#define _SONAR_H

// Funcions s_actualitzar

void s_actualitzar (void);
void s_actualitzar_d (void);
void s_actualitzar_e (void);
void s_actualitzar_f (void);

// Funcions s_maxim_x

int s_maxim (void);
int s_maxim_d (void);
int s_maxim_e (void);
int s_maxim_f (void);

// Funcions s_posar_maxim_x

void s_posar_maxim_l (int maxim);
void s_posar_maxim_f (int maxim);

// Funcions s_elements_x

int s_elements (void);
int s_elements_d (void);
int s_elements_e (void);
int s_elements_f (void);

// Funcions s_x_x, s_y_x

float s_x (int i);
float s_y (int i);

float s_x_d (int i);
float s_y_d (int i);

float s_x_e (int i);
float s_y_e (int i);

float s_x_f (int i);
float s_y_f (int i);

// Funcions s_dist_x

float s_dist (int i);
float s_dist_d (int i);
float s_dist_e (int i);
float s_dist_f (int i);
```

```
// Funcions s_angle_x

float s_angle (int i);
float s_angle_d (int i);
float s_angle_e (int i);
float s_angle_f (int i);

// Funcions s_total_quadre_x

int s_total_quadre (float x0, float y0, float x1, float y1);
int s_total_quadre_d (float x0, float y0, float x1, float y1);
int s_total_quadre_e (float x0, float y0, float x1, float y1);
int s_total_quadre_f (float x0, float y0, float x1, float y1);

// Funcions s_dist_min_x

int s_dist_min_d (void);
int s_dist_min_e (void);
int s_dist_min_f (void);
int s_dist_min (void);

// Funcions s_dist_max_x

int s_dist_max (void);
int s_dist_max_d (void);
int s_dist_max_e (void);
int s_dist_max_f (void);

// Funcions s_elements_cercle_x

int s_total_cercle (float radi);
int s_total_cercle_d (float radi);
int s_total_cercle_e (float radi);
int s_total_cercle_f (float radi);

#endif
```

1.2 Sonar.c

```
/* *************** */
/* Fitxer sonar.c  */
/* *************** */

#include "saphira.h"

// Constants

#define MIN_D 10    // Es defineix com a buffer mínim 10 (dret)
#define MIN_E 10    // Es defineix com a buffer mínim 10 (esquerra)
#define MIN_F 5           // Es defineix com a buffer mínim 10 (frontal)

#define TOL_MIN 0.001     // Es defineix el 0.0 com qualsevol n<TOL_MIN
#define PI 3.1416

// Definició Funcions Internes

float c_dist (float x, float y); // Retorna el mòdul del vector (x,y)
float c_angle (float x, float y);      // Retorna l'angle (deg.) del vector (x,y)
int dintre_q (float x, float y, float x0, float y0, float x1, float y1);
void Normalitzar (float *x0, float *y0, float *x1, float *y1);

// Variables globals

float b_sonar_x[CBUF_LEN];
float b_sonar_y[CBUF_LEN];
int b_sonar;
```

```
float b_sonar_x_d[CBUF_LEN];
float b_sonar_y_d[CBUF_LEN];
int b_sonar_d;

float b_sonar_x_e[CBUF_LEN];
float b_sonar_y_e[CBUF_LEN];
int b_sonar_e;

float b_sonar_x_f[CBUF_LEN];
float b_sonar_y_f[CBUF_LEN];
int b_sonar_f;

// Funcions Internes

float c_dist (float x, float y)
{
	return (float)sqrt(pow(x,2) + pow(y,2));
}

float c_angle (float x, float y)
{
	float angle = 0.0;

	if (x<TOL_MIN)
	{
		if (y>0.0) angle = 90.0;
		if (y<0.0) angle = -90.0;
	}

	if (y<TOL_MIN)
	{
		if (x>0.0) angle = 0.0;
		if (x<0.0) angle = 180.0;
	}

	if ((x>TOL_MIN)&&(y>TOL_MIN))
	{
		angle = (float)((180/PI)*(atan(y/x)));
	}

	if ((x>TOL_MIN)&&(y<TOL_MIN))
	{
		angle = (float)((180/PI)*(-atan(-y/x)));
	}

	if ((x<TOL_MIN)&&(y>TOL_MIN))
	{
		angle = (float)(180 - ((180/PI)*atan(y/-x)));
	}

	if ((x<TOL_MIN)&&(y<TOL_MIN))
	{
		angle = (float)(-180 + ((180/PI)*atan(y/x)));
	}

	return angle;
}

int dintre_q (float x, float y, float x0, float y0, float x1, float y1)
{
	return ((x>x0)&&(x<x1)&&(y>y0)&&(y<y1));
}

void Normalitzar (float *x0, float *y0, float *x1, float *y1)
{
	float aux;
```

```
	if (*x0>*x1)
	{
		aux = *x1;
		*x1 = *x0;
		*x0 = aux;
	}

	if (*y0>*y1)
	{
		aux = *y1;
		*y1 = *y0;
		*y0 = aux;
	}

	if (((*x0)==(*x1))||((*y0)==(*y1)))
		sfSMessage("Quadre buit. No poden haver elements interiors");
}

// Funcions s_actualitzar

EXPORT void s_actualitzar (void)
{
	int limit, i;

	b_sonar = 0;

	limit = sr_buf->limit;
	i = 0;

	while (i<limit)
	{
		if (sr_buf->valid[i])
		{
			b_sonar_x[b_sonar] = sr_buf->xbuf[i];
			b_sonar_y[b_sonar] = sr_buf->ybuf[i];
			b_sonar++;
		}

		i++;
	}

	limit = sl_buf->limit;
	i = 0;

	while (i<limit)
	{
		if (sl_buf->valid[i])
		{
			b_sonar_x[b_sonar] = sl_buf->xbuf[i];
			b_sonar_y[b_sonar] = sl_buf->ybuf[i];
			b_sonar++;
		}

		i++;
	}

	limit = sraw_buf->limit;
	i = 0;

	while (i<limit)
	{
		if (sraw_buf->valid[i])
		{
			b_sonar_x[b_sonar] = sraw_buf->xbuf[i];
			b_sonar_y[b_sonar] = sraw_buf->ybuf[i];
			b_sonar++;
		}
```

```
		i++;
	}
}

EXPORT void s_actualitzar_d (void)
{
	int limit, i;

	b_sonar_d = 0;
	limit = sr_buf->limit;
	i = 0;

	while (i<limit)
	{
		if (sr_buf->valid[i])
		{
			b_sonar_x_d[b_sonar_d] = sr_buf->xbuf[i];
			b_sonar_y_d[b_sonar_d] = sr_buf->ybuf[i];
			b_sonar_d++;
		}

		i++;
	}
}

EXPORT void s_actualitzar_e (void)
{
	int limit, i;

	b_sonar_e = 0;
	limit = sl_buf->limit;
	i = 0;

	while (i<limit)
	{
		if (sl_buf->valid[i])
		{
			b_sonar_x_e[b_sonar_e] = sl_buf->xbuf[i];
			b_sonar_y_e[b_sonar_e] = sl_buf->ybuf[i];
			b_sonar_e++;
		}

		i++;
	}
}

EXPORT void s_actualitzar_f (void)
{
	int limit, i;

	b_sonar_f = 0;
	limit = sraw_buf->limit;
	i = 0;

	while (i<limit)
	{
		if (sraw_buf->valid[i])
		{
			b_sonar_x_f[b_sonar_f] = sraw_buf->xbuf[i];
			b_sonar_y_f[b_sonar_f] = sraw_buf->ybuf[i];
			b_sonar_f++;
		}

		i++;
	}
}

// Funcions s_maxim_x
```

```
EXPORT int s_maxim (void)
{
	return sr_buf->limit + sl_buf->limit + sraw_buf->limit;
}

EXPORT int s_maxim_d (void)
{
	return sr_buf->limit;
}

EXPORT int s_maxim_e (void)
{
	return sl_buf->limit;
}

EXPORT int s_maxim_f (void)
{
	return sraw_buf->limit;
}

// Funcions s_posar_maxim_x

EXPORT void s_posar_maxim_l (int maxim)
{
	if ((maxim<=CBUF_LEN)&&(maxim>=MIN_D)) sfSetSideBuffer(maxim);
	if (maxim<MIN_D) sfSetSideBuffer(MIN_D);
	if (maxim>CBUF_LEN) sfSetSideBuffer(CBUF_LEN);
}

EXPORT void s_posar_maxim_f (int maxim)
{
	if ((maxim<=CBUF_LEN)&&(maxim>=MIN_F)) sfSetFrontBuffer(maxim);
	if (maxim<MIN_F) sfSetFrontBuffer(MIN_F);
	if (maxim>CBUF_LEN) sfSetFrontBuffer(CBUF_LEN);
}

//Funcions s_elements_x

EXPORT int s_elements (void)
{
	return b_sonar;
}

EXPORT int s_elements_d (void)
{
	return b_sonar_d;
}

EXPORT int s_elements_e (void)
{
	return b_sonar_e;
}

EXPORT int s_elements_f (void)
{
	return b_sonar_f;
}

// Funcions s_x_x, s_y_x

EXPORT float s_x (int i)
{
	if ((i<=b_sonar)&&(i>0)) return b_sonar_x[i-1];
	else
	{
		sfSMessage("Error d'accés al buffer. Índex incorrecte");
		return 0.0;
```

```
	}
}

EXPORT float s_y (int i)
{
	if ((i<=b_sonar)&&(i>0)) return b_sonar_y[i-1];
	else
	{
		sfSMessage("Error d'accés al buffer. Índex incorrecte");
		return 0.0;
	}
}

EXPORT float s_x_d (int i)
{
	if ((i<=b_sonar_d)&&(i>0)) return b_sonar_x_d[i-1];
	else
	{
		sfSMessage("Error d'accés al buffer. Índex incorrecte");
		return 0.0;
	}
}

EXPORT float s_y_d (int i)
{
	if ((i<=b_sonar_d)&&(i>0)) return b_sonar_y_d[i-1];
	else
	{
		sfSMessage("Error d'accés al buffer. Índex incorrecte");
		return 0.0;
	}
}

EXPORT float s_x_e (int i)
{
	if ((i<=b_sonar_e)&&(i>0)) return b_sonar_x_e[i-1];
	else
	{
		sfSMessage("Error d'accés al buffer. Índex incorrecte");
		return 0.0;
	}
}

EXPORT float s_y_e (int i)
{
	if ((i<=b_sonar_e)&&(i>0)) return b_sonar_y_e[i-1];
	else
	{
		sfSMessage("Error d'accés al buffer. Índex incorrecte");
		return 0.0;
	}
}

EXPORT float s_x_f (int i)
{
	if ((i<=b_sonar_f)&&(i>0)) return b_sonar_x_f[i-1];
	else
	{
		sfSMessage("Error d'accés al buffer. Índex incorrecte");
		return 0.0;
	}
}

EXPORT float s_y_f (int i)
{
	if ((i<=b_sonar_f)&&(i>0)) return b_sonar_y_f[i-1];
	else
	{
```

```
		sfSMessage("Error d'accés al buffer. Índex incorrecte");
		return 0.0;
	}
}

// Funcions s_dist_x

EXPORT float s_dist (int i)
{
	double aux = 0.0;

	if ((i<=b_sonar)&&(i>0))
	{
		aux = c_dist(b_sonar_x[i-1],b_sonar_y[i-1]);
	}
	else sfSMessage("Error d'accés al buffer. Índex incorrecte");

	return (float)aux;
}

EXPORT float s_dist_d (int i)
{
	double aux = 0.0;

	if ((i<=b_sonar_d)&&(i>0))
	{
		aux = c_dist(b_sonar_x_d[i-1],b_sonar_y_d[i-1]);
	}
	else sfSMessage("Error d'accés al buffer. Índex incorrecte");

	return (float)aux;
}

EXPORT float s_dist_e (int i)
{
	double aux = 0.0;

	if ((i<=b_sonar_e)&&(i>0))
	{
		aux = c_dist(b_sonar_x_e[i-1],b_sonar_y_e[i-1]);
	}
	else sfSMessage("Error d'accés al buffer. Índex incorrecte");

	return (float)aux;
}

EXPORT float s_dist_f (int i)
{
	double aux = 0.0;

	if ((i<=b_sonar_f)&&(i>0))
	{
		aux = c_dist(b_sonar_x_f[i-1],b_sonar_y_f[i-1]);
	}
	else sfSMessage("Error d'accés al buffer. Índex incorrecte");

	return (float)aux;
}

// Funcions s_angle_x

EXPORT float s_angle (int i)
{
	double aux = 0.0;

	if ((i<=b_sonar)&&(i>0))
	{
		aux = c_angle(b_sonar_x[i-1],b_sonar_y[i-1]);
```

```
	}
	else sfSMessage("Error d'accés al buffer. Índex incorrecte");

	return (float)aux;
}

EXPORT float s_angle_d (int i)
{
	double aux = 0.0;

	if ((i<=b_sonar_d)&&(i>0))
	{
		aux = c_angle(b_sonar_x_d[i-1],b_sonar_y_d[i-1]);
	}
	else sfSMessage("Error d'accés al buffer. Índex incorrecte");

	return (float)aux;
}

EXPORT float s_angle_e (int i)
{
	double aux = 0.0;

	if ((i<=b_sonar_e)&&(i>0))
	{
		aux = c_angle(b_sonar_x_e[i-1],b_sonar_y_e[i-1]);
	}
	else sfSMessage("Error d'accés al buffer. Índex incorrecte");

	return (float)aux;
}

EXPORT float s_angle_f (int i)
{
	double aux = 0.0;

	if ((i<=b_sonar_f)&&(i>0))
	{
		aux = c_angle(b_sonar_x_f[i-1],b_sonar_y_f[i-1]);
	}
	else sfSMessage("Error d'accés al buffer. Índex incorrecte");

	return (float)aux;
}

// Funcions s_total_quadre_x

EXPORT int s_total_quadre (float x0, float y0, float x1, float y1)
{
	int num,i;

	// Normalitzem

	Normalitzar (&x0,&y0,&x1,&y1);

	// Comprovem cada element

	i = 0;
	num = 0;

	while (i<b_sonar)
	{
		if (dintre_q(b_sonar_x[i],b_sonar_y[i],x0,y0,x1,y1)) num++;

		i++;
	}

	return num;
```

```
}

EXPORT int s_total_quadre_d (float x0, float y0, float x1, float y1)
{
	int num,i;

	// Normalitzem

	Normalitzar (&x0,&y0,&x1,&y1);

	// Comprovem cada element

	i = 0;
	num = 0;

	while (i<b_sonar_d)
	{
		if (dintre_q(b_sonar_x_d[i],b_sonar_y_d[i],x0,y0,x1,y1)) num++;

		i++;
	}

	return num;
}

EXPORT int s_total_quadre_e (float x0, float y0, float x1, float y1)
{
	int num,i;

	// Normalitzem

	Normalitzar (&x0,&y0,&x1,&y1);

	// Comprovem cada element

	i = 0;
	num = 0;

	while (i<b_sonar_e)
	{
		if (dintre_q(b_sonar_x_e[i],b_sonar_y_e[i],x0,y0,x1,y1)) num++;

		i++;
	}

	return num;
}

EXPORT int s_total_quadre_f (float x0, float y0, float x1, float y1)
{
	int num,i;

	// Normalitzem

	Normalitzar (&x0,&y0,&x1,&y1);

	// Comprovem cada element

	i = 0;
	num = 0;

	while (i<b_sonar_f)
	{
		if (dintre_q(b_sonar_x_f[i],b_sonar_y_f[i],x0,y0,x1,y1)) num++;

		i++;
	}
```

```
	return num;
}

// Funcions s_dist_min_x

EXPORT int s_dist_min (void)
{
	int i,num;

	num = 0;

	if (b_sonar == 0) sfSMessage ("Buffer Buit");
	else num = 1;

	i = 1;

	while (i<b_sonar)
	{
		if (c_dist(b_sonar_x[i],b_sonar_y[i])<c_dist(b_sonar_x[num-
1],b_sonar_y[num-1]))
			num = i + 1;

		i++;
	}

	return num;
}

EXPORT int s_dist_min_d (void)
{
	int i,num;

	num = 0;

	if (b_sonar_d == 0) sfSMessage ("Buffer Buit");
	else num = 1;

	i = 1;

	while (i<b_sonar_d)
	{
		if (c_dist(b_sonar_x_d[i],b_sonar_y_d[i])<c_dist(b_sonar_x_d[num-
1],b_sonar_y_d[num-1]))
			num = i + 1;

		i++;
	}

	return num;
}

EXPORT int s_dist_min_e (void)
{
	int i,num;

	num = 0;

	if (b_sonar_e == 0) sfSMessage ("Buffer Buit");
	else num = 1;

	i = 1;

	while (i<b_sonar_e)
	{
		if (c_dist(b_sonar_x_e[i],b_sonar_y_e[i])<c_dist(b_sonar_x_e[num-
1],b_sonar_y_e[num-1]))
			num = i + 1;
```

```
		i++;
	}

	return num;
}

EXPORT int s_dist_min_f (void)
{
	int i,num;

	num = 0;

	if (b_sonar_f == 0) sfSMessage ("Buffer Buit");
	else num = 1;

	i = 1;

	while (i<b_sonar_f)
	{
		if (c_dist(b_sonar_x_f[i],b_sonar_y_f[i])<c_dist(b_sonar_x_f[num-
1],b_sonar_y_f[num-1]))
			num = i + 1;

		i++;
	}

	return num;
}

// Funcions s_dist_max_x

EXPORT int s_dist_max (void)
{
	int i,num;

	num = 0;

	if (b_sonar == 0) sfSMessage ("Buffer Buit");
	else num = 1;

	i = 1;

	while (i<b_sonar)
	{
		if (c_dist(b_sonar_x[i],b_sonar_y[i])>c_dist(b_sonar_x[num-
1],b_sonar_y[num-1]))
			num = i + 1;

		i++;
	}

	return num;
}

EXPORT int s_dist_max_d (void)
{
	int i,num;

	num = 0;

	if (b_sonar_d == 0) sfSMessage ("Buffer Buit");
	else num = 1;

	i = 1;

	while (i<b_sonar_d)
	{
```

```
			if (c_dist(b_sonar_x_d[i],b_sonar_y_d[i])>c_dist(b_sonar_x_d[num-
1],b_sonar_y_d[num-1]))
				num = i + 1;

			i++;
		}

		return num;
}

EXPORT int s_dist_max_e (void)
{
		int i,num;

		num = 0;

		if (b_sonar_e == 0) sfSMessage ("Buffer Buit");
		else num = 1;

		i = 1;

		while (i<b_sonar_e)
		{
			if (c_dist(b_sonar_x_e[i],b_sonar_y_e[i])>c_dist(b_sonar_x_e[num-
1],b_sonar_y_e[num-1]))
				num = i + 1;

			i++;
		}

		return num;
}

EXPORT int s_dist_max_f (void)
{
		int i,num;

		num = 0;

		if (b_sonar_f == 0) sfSMessage ("Buffer Buit");
		else num = 1;

		i = 1;

		while (i<b_sonar_f)
		{
			if (c_dist(b_sonar_x_f[i],b_sonar_y_f[i])>c_dist(b_sonar_x_f[num-
1],b_sonar_y_f[num-1]))
				num = i + 1;

			i++;
		}

		return num;
}

// Funcions s_elements_bola_x

EXPORT int s_total_cercle (float radi)
{
		int num,i;

		// Comprovem cada element

		i = 0;
		num = 0;

		while (i<b_sonar)
```

```
	{
		if (c_dist(b_sonar_x[i],b_sonar_y[i])<radi) num++;

		i++;
	}

	return num;
}

EXPORT int s_total_cercle_d (float radi)
{
	int num,i;

	// Comprovem cada element

	i = 0;
	num = 0;

	while (i<b_sonar_d)
	{
		if (c_dist(b_sonar_x_d[i],b_sonar_y_d[i])<radi) num++;

		i++;
	}

	return num;
}

EXPORT int s_total_cercle_e (float radi)
{
	int num,i;

	// Comprovem cada element

	i = 0;
	num = 0;

	while (i<b_sonar_e)
	{
		if (c_dist(b_sonar_x_e[i],b_sonar_y_e[i])<radi) num++;

		i++;
	}

	return num;
}

EXPORT int s_total_cercle_f (float radi)
{
	int num,i;

	// Comprovem cada element

	i = 0;
	num = 0;

	while (i<b_sonar_f)
	{
		if (c_dist(b_sonar_x_f[i],b_sonar_y_f[i])<radi) num++;

		i++;
	}

	return num;
}
```

1.3 Sonar_Colbert.c

```
/* ********************** */
/* Fitxer sonar_colbert.c  */
/* ********************** */

#include "saphira.h"
#include "sonar.h"

/* ************************************** */
/* Fitxers de càrrega i descàrrega de la dll */
/* ************************************** */

EXPORT void
sfLoadInit(void)    // Fitxer que s'avalua al carregar-se la dll
{
       // Missatge Inici

       sfSMessage("DLL del Sonar iniciant càrrega...");

       // Inicialització dels valors del buffer

       sfSMessage("Inicialitzant buffers...");

       sfSetSideBuffer(40);
       sfSetFrontBuffer(20);

       // Càrrega de funcions

       sfSMessage("Inicialitzant funcions...");

       sfSMessage("Inicialitzant s_actualitzar_x...");

       sfAddEvalFn("s_actualitzar",s_actualitzar,sfVOID,0,sfVOID);
       sfAddEvalFn("s_actualitzar_d",s_actualitzar_d,sfVOID,0,sfVOID);
       sfAddEvalFn("s_actualitzar_e",s_actualitzar_e,sfVOID,0,sfVOID);
       sfAddEvalFn("s_actualitzar_f",s_actualitzar_f,sfVOID,0,sfVOID);

       sfSMessage("Inicialitzant s_maxim_x...");

       sfAddEvalFn("s_maxim",s_maxim,sfINT,0,sfVOID);
       sfAddEvalFn("s_maxim_d",s_maxim_d,sfINT,0,sfVOID);
       sfAddEvalFn("s_maxim_e",s_maxim_e,sfINT,0,sfVOID);
       sfAddEvalFn("s_maxim_f",s_maxim_f,sfINT,0,sfVOID);

       sfSMessage("Inicialitzant s_posar_maxim_x...");

       sfAddEvalFn("s_posar_maxim_l",s_posar_maxim_l,sfVOID,1,sfINT);
       sfAddEvalFn("s_posar_maxim_f",s_posar_maxim_f,sfVOID,1,sfINT);

       sfSMessage("Inicialitzant s_elements_x...");

       sfAddEvalFn("s_elements",s_elements,sfINT,0,sfVOID);
       sfAddEvalFn("s_elements_d",s_elements_d,sfINT,0,sfVOID);
       sfAddEvalFn("s_elements_e",s_elements_e,sfINT,0,sfVOID);
       sfAddEvalFn("s_elements_f",s_elements_f,sfINT,0,sfVOID);

       sfSMessage("Inicialitzant s_x_x, s_y_x...");

       sfAddEvalFn("s_x",s_x,sfFLOAT,1,sfINT);
       sfAddEvalFn("s_y",s_y,sfFLOAT,1,sfINT);

       sfAddEvalFn("s_x_d",s_x_d,sfFLOAT,1,sfINT);
       sfAddEvalFn("s_y_d",s_y_d,sfFLOAT,1,sfINT);

       sfAddEvalFn("s_x_e",s_x_e,sfFLOAT,1,sfINT);
       sfAddEvalFn("s_y_e",s_y_e,sfFLOAT,1,sfINT);
```

```
        sfAddEvalFn("s_x_f",s_x_f,sfFLOAT,1,sfINT);
        sfAddEvalFn("s_y_f",s_y_f,sfFLOAT,1,sfINT);

        sfSMessage("Inicialitzant s_dist_x...");

        sfAddEvalFn("s_dist",s_dist,sfFLOAT,1,sfINT);
        sfAddEvalFn("s_dist_d",s_dist_d,sfFLOAT,1,sfINT);
        sfAddEvalFn("s_dist_e",s_dist_e,sfFLOAT,1,sfINT);
        sfAddEvalFn("s_dist_f",s_dist_f,sfFLOAT,1,sfINT);

        sfSMessage("Inicialitzant angle_x...");

        sfAddEvalFn("s_angle",s_angle,sfFLOAT,1,sfINT);
        sfAddEvalFn("s_angle_d",s_angle_d,sfFLOAT,1,sfINT);
        sfAddEvalFn("s_angle_e",s_angle_e,sfFLOAT,1,sfINT);
        sfAddEvalFn("s_angle_f",s_angle_f,sfFLOAT,1,sfINT);

        sfSMessage("Inicialitzant s_total_quadre_x...");

        sfAddEvalFn("s_total_quadre",s_total_quadre,sfINT,
                    4,sfFLOAT,sfFLOAT,sfFLOAT,sfFLOAT);
        sfAddEvalFn("s_total_quadre_d",s_total_quadre_d,sfINT,
                    4,sfFLOAT,sfFLOAT,sfFLOAT,sfFLOAT);
        sfAddEvalFn("s_total_quadre_e",s_total_quadre_e,sfINT,
                    4,sfFLOAT,sfFLOAT,sfFLOAT,sfFLOAT);
        sfAddEvalFn("s_total_quadre_f",s_total_quadre_f,sfINT,
                    4,sfFLOAT,sfFLOAT,sfFLOAT,sfFLOAT);

        sfSMessage("Inicialitzant s_dist_min_x...");

        sfAddEvalFn("s_dist_min",s_dist_min,sfINT,0,sfVOID);
        sfAddEvalFn("s_dist_min_d",s_dist_min_d,sfINT,0,sfVOID);
        sfAddEvalFn("s_dist_min_e",s_dist_min_e,sfINT,0,sfVOID);
        sfAddEvalFn("s_dist_min_f",s_dist_min_f,sfINT,0,sfVOID);

        sfSMessage("Inicialitzant s_dist_max_x...");

        sfAddEvalFn("s_dist_max",s_dist_max,sfINT,0,sfVOID);
        sfAddEvalFn("s_dist_max_d",s_dist_max_d,sfINT,0,sfVOID);
        sfAddEvalFn("s_dist_max_e",s_dist_max_e,sfINT,0,sfVOID);
        sfAddEvalFn("s_dist_max_f",s_dist_max_f,sfINT,0,sfVOID);

        sfSMessage("Inicialitzant s_total_cercle_x...");

        sfAddEvalFn("s_total_cercle",s_total_cercle,sfINT,1,sfFLOAT);
        sfAddEvalFn("s_total_cercle_d",s_total_cercle_d,sfINT,1,sfFLOAT);
        sfAddEvalFn("s_total_cercle_e",s_total_cercle_e,sfINT,1,sfFLOAT);
        sfAddEvalFn("s_total_cercle_f",s_total_cercle_f,sfINT,1,sfFLOAT);

        // Missatge Final

        sfSMessage("DLL del Sonar carregada.");
}

EXPORT void
sfLoadExit(void)    // Fitxer que s'avalua al descarregar-se la dll
{
        sfSMessage("DLL del Sonar Descarregada.");
}
```

2 sfUtils

2.1 sfUtils.h

```
#ifndef _SFUTILS_H
#define _SFUTILS_H

#include "saphira.h"

// --------------------------
// Definició del tipus vector
// --------------------------

typedef struct
{
    float *telem;               /* Taula d'elements */
    int dim;                    /* Dimensio */
} Vector;

// --------------------------
// Definició del tipus llista
// --------------------------

typedef struct datos
{
  point *punt;
  struct datos *siguiente;
} telemento;

typedef struct
{
      telemento **punts;
      int *numpunts;
}LlistaPunts;

// -------------------------------
// Definició de funcions de Vector
// -------------------------------

EXPORT Vector CrearVector(int n);
EXPORT void DestruirVector(Vector v);
EXPORT void InicialitzarVector(Vector v);

EXPORT int DimVector(Vector v);
EXPORT float ConsVector(Vector v, int i);
EXPORT void AssigVector(Vector v, int i, float f);
EXPORT void EscriureVector(Vector v);

EXPORT LlistaPunts ObtenirLlistaVectors(Vector x, Vector y, Vector angle);
EXPORT LlistaPunts ObtenirLlistaVectorsGlobals(Vector x, Vector y, Vector angle);

// -------------------------------
// Definició de funcions de Llista
// -------------------------------

EXPORT LlistaPunts llCrearLlistaPunts(void);
EXPORT void llDestruirLlistaPunts(LlistaPunts l);

EXPORT int llNumPunts(LlistaPunts l);
```

```
EXPORT void llAfegirPunt(LlistaPunts l, float x, float y, float angle);
EXPORT void llAfegirPuntGlobal(LlistaPunts l, float x, float y, float angle);

EXPORT void llInsertarPunt(LlistaPunts l, float x, float y, float angle, int
pos);
EXPORT void llInsertarPuntGlobal(LlistaPunts l, float x, float y, float angle,
int pos);

EXPORT void llEliminarPunt(LlistaPunts l, int pos);
EXPORT void llEliminarPrimerPunt(LlistaPunts l);
EXPORT void llEliminarUltimPunt(LlistaPunts l);
EXPORT void llEliminarTotsPunts(LlistaPunts l);

EXPORT void llCambiarPunt(LlistaPunts l, int pos, float x, float y, float angle);
EXPORT void llCambiarPuntGlobal(LlistaPunts l, int pos, float x, float y, float
angle);

EXPORT float llObtenirXPunt(LlistaPunts l, int pos);
EXPORT float llObtenirYPunt(LlistaPunts l, int pos);
EXPORT float llObtenirAnglePunt(LlistaPunts l, int pos);

EXPORT float llObtenirXGlobalPunt(LlistaPunts l, int pos);
EXPORT float llObtenirYGlobalPunt(LlistaPunts l, int pos);
EXPORT float llObtenirAngleGlobalPunt(LlistaPunts l, int pos);

EXPORT Vector llObtenirVectorXLlista(LlistaPunts l);
EXPORT Vector llObtenirVectorYLlista(LlistaPunts l);
EXPORT Vector llObtenirVectorAngleLlista(LlistaPunts l);

EXPORT Vector llObtenirVectorXGlobalLlista(LlistaPunts l);
EXPORT Vector llObtenirVectorYGlobalLlista(LlistaPunts l);
EXPORT Vector llObtenirVectorAngleGlobalLlista(LlistaPunts l);

EXPORT void llEscriureLlistaPunts(LlistaPunts l);
EXPORT void llEscriureLlistaPuntsGlobals(LlistaPunts l);

// ---------------------
// Funcions geomètriques
// ---------------------

EXPORT LlistaPunts CalcularSpline(LlistaPunts l, Vector knots, int k, int
num_punts);
EXPORT LlistaPunts CalcularStandardSpline(LlistaPunts l, int k, int num_punts);

EXPORT LlistaPunts CalcularBezier(LlistaPunts l, int k, int num_punts);

EXPORT int MinimaDistanciaLlistaPunts(LlistaPunts l);
EXPORT int MaximaDistanciaLlistaPunts(LlistaPunts l);

EXPORT float llDistanciaPunt(LlistaPunts l, int pos);
EXPORT float llAnglePunt(LlistaPunts l, int pos);

EXPORT float llLongitudLlista(LlistaPunts l);
EXPORT float llLongitudLlistaInterval(LlistaPunts l, int inici, int fi);
// Si es vol inici 0 es pot posar -1, si es vol final últim posar -1)

#endif
```

2.2 sfUtils.c

```
// -------------------
// Inclusió de fitxers
// -------------------

#include "sfUtils.h"
#include <assert.h>
```

```
// ------------------------------
// Definició de funcions internes
// ------------------------------

float CalcularNik(float u, Vector knots, int i, int k);
float CalcularBkn(int k, int n, float u);
float CalcularCnk(int n, int k);
float factorial(int n);
float potencia(float u, int k);

// ------------------------------
// Implementació del tipus Vector
// ------------------------------

EXPORT Vector CrearVector(int n)
{
	Vector v;

	assert(n > 0);
	v.dim = n;
	v.telem = (float *)malloc(n * sizeof (float));

	assert(v.telem != NULL);
	return v;
}

EXPORT void InicialitzarVector(Vector v)
{
	int i;

	i = 0;
	while (i<DimVector(v))
	{
		AssigVector(v,i,0.0);

		i++;
	}
}

EXPORT int DimVector(Vector v)
{
	return v.dim;
}

EXPORT void AssigVector(Vector v, int i, float f)
{
	assert(0 <= i && i < v.dim);
	v.telem[i] = f;
}

EXPORT float ConsVector(Vector v, int i)
{
	assert(0 <= i && i < v.dim);
	return v.telem[i];
}

EXPORT void DestruirVector(Vector v)
{
	free(v.telem);
}

EXPORT void EscriureVector(Vector v)
{
	int i;

	i = 0;
	while (i<DimVector(v))
```

```
	{
		sfSMessage("  v[%d] = %f", i, ConsVector(v,i));

		i++;
	}
}

EXPORT LlistaPunts ObtenirLlistaVectors(Vector x, Vector y, Vector angle)
{
	LlistaPunts llista;
	int i;

	llista = llCrearLlistaPunts();

	if (DimVector(x)!=DimVector(y) || DimVector(y)!=DimVector(angle))
	{
		sfSMessage("Els vectors han de ser del mateix tamany");
		sfSMessage("Es retorna una llista buida");
		return llista;
	}

	i = 0;
	while (i<DimVector(x))
	{

	llAfegirPunt(llista,ConsVector(x,i),ConsVector(y,i),ConsVector(angle,i));

		i++;
	}

	return llista;
}

EXPORT LlistaPunts ObtenirLlistaVectorsGlobals(Vector x, Vector y, Vector angle)
{
	LlistaPunts llista;
	int i;

	llista = llCrearLlistaPunts();

	if (DimVector(x)!=DimVector(y) || DimVector(y)!=DimVector(angle))
	{
		sfSMessage("Els vectors han de ser del mateix tamany");
		sfSMessage("Es retorna una llista buida");
		return llista;
	}

	i = 0;
	while (i<DimVector(x))
	{

	llAfegirPuntGlobal(llista,ConsVector(x,i),ConsVector(y,i),ConsVector(angle,
i));

		i++;
	}

	return llista;
}

// -----------------------------------
// Implementació del tipus LlistaPunts
// -----------------------------------

telemento *NuevoElemento(void)
{
	telemento *q = ((telemento *)malloc(sizeof(telemento)));
	if (!q)
```

```
	{
		perror("Insuficiente memoria");
		exit(1);
	}
	return q;
}


EXPORT LlistaPunts llCrearLlistaPunts(void)
{
	LlistaPunts l;

	l.punts = (telemento **)malloc(sizeof(telemento *));
	l.punts[0] = NuevoElemento();
	l.punts[0]->siguiente = NULL;

	l.numpunts = (int *)malloc(sizeof(int));
	l.numpunts[0] = 0;

	return l;
}

EXPORT void llDestruirLlistaPunts(LlistaPunts l)
{
	llEliminarTotsPunts(l);
	free(l.numpunts);
	free(l.punts);
}

EXPORT int llNumPunts(LlistaPunts l)
{
	return l.numpunts[0];
}

EXPORT void llAfegirPunt(LlistaPunts l, float x, float y, float angle)
{
	point *aux;
	telemento *q;

	aux = sfCreateLocalPoint(x,y,angle);

	q = NuevoElemento();
	q->punt = aux;
	sfAddPoint(q->punt);

	q->siguiente = l.punts[0];
	l.punts[0] = q;

	l.numpunts[0]++;
}

EXPORT void llAfegirPuntGlobal(LlistaPunts l, float x, float y, float angle)
{
	point *aux;
	telemento *q;

	aux = sfCreateGlobalPoint(x,y,angle);

	q = NuevoElemento();
	q->punt = aux;
	sfAddPoint(q->punt);

	q->siguiente = l.punts[0];
	l.punts[0] = q;

	l.numpunts[0]++;
}
```

```
EXPORT void llInsertarPunt(LlistaPunts l, float x, float y, float angle, int pos)
{
	int i;
	telemento *q, *r;
	point *aux;

	// Es pot insertar en posició de llNumPunts(l)

	if (pos<0 || pos>llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte. No es realitza cap operació");
		return;
	}

	if (pos==llNumPunts(l))
	{
		llAfegirPunt(l,x,y,angle);
		return;
	}

	aux = sfCreateLocalPoint(x,y,angle);
	q = NuevoElemento();
	q->punt = aux;
	sfAddPoint(q->punt);

	i = llNumPunts(l) - 1;
	r = l.punts[0];
	while (i>pos)
	{
		r = r->siguiente;
		i--;
	}

	q->siguiente = r->siguiente;
	r->siguiente = q;

	l.numpunts[0]++;
}

EXPORT void llInsertarPuntGlobal(LlistaPunts l, float x, float y, float angle,
int pos)
{
	int i;
	telemento *q, *r;
	point *aux;

	// Es pot insertar en posició de llNumPunts(l)

	if (pos<0 || pos>llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte. No es realitza cap operació");
		return;
	}

	if (pos==llNumPunts(l))
	{
		llAfegirPuntGlobal(l,x,y,angle);
		return;
	}

	aux = sfCreateGlobalPoint(x,y,angle);
	q = NuevoElemento();
	q->punt = aux;
	sfAddPoint(q->punt);

	i = llNumPunts(l) - 1;
	r = l.punts[0];
	while (i>pos)
```

```
	{
		r = r->siguiente;
		i--;
	}

	q->siguiente = r->siguiente;
	r->siguiente = q;

	l.numpunts[0]++;
}

EXPORT void llEliminarPunt(LlistaPunts l, int pos)
{
	telemento *q, *aux;
	int i;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es realitza cap operació");
		return;
	}

	if (pos<0 || pos>=llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte. No es realitza cap operació");
		return;
	}

	if(pos==llNumPunts(l)-1)
	{
		llEliminarUltimPunt(l);
		return;
	}

	q = l.punts[0];
	i = llNumPunts(l)-1;

	while (i>pos+1)
	{
		q = q->siguiente;
		i--;
	}

	aux = q->siguiente;
	q->siguiente = aux->siguiente;
	sfRemPoint(aux->punt);
	free(aux);
	l.numpunts[0]--;
}

EXPORT void llEliminarPrimerPunt(LlistaPunts l)
{
	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es realitza cap operació");
		return;
	}

	llEliminarPunt(l,0);
}

EXPORT void llEliminarUltimPunt(LlistaPunts l)
{
	telemento *q;

	if (llNumPunts(l)==0) return;

	q = l.punts[0];
```

```
	l.punts[0] = l.punts[0]->siguiente;

	sfRemPoint(q->punt);
	free(q);

	l.numpunts[0]--;
}

EXPORT void llEliminarTotsPunts(LlistaPunts l)
{
	while(llNumPunts(l)>0)
	{
		llEliminarUltimPunt(l);
	}
}

EXPORT void llCambiarPunt(LlistaPunts l, int pos, float x, float y, float angle)
{
	telemento *q;
	int i;
	point *p;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es pot cambiar el punt");
		return;
	}

	if (pos<0 || pos>=llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte");
		return;
	}

	q = l.punts[0];
	i = llNumPunts(l)-1;

	while (i>pos)
	{
		q = q->siguiente;
		i--;
	}

	p = q->punt;
	p->x = x;
	p->y = y;
	p->th = angle;
	sfSetGlobalCoords(p);
}

EXPORT void llCambiarPuntGlobal(LlistaPunts l, int pos, float x, float y, float
angle)
{
	telemento *q;
	int i;
	point *p;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es pot cambiar el punt");
		return;
	}

	if (pos<0 || pos>=llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte");
		return;
	}
```

```
	q = l.punts[0];
	i = llNumPunts(l)-1;

	while (i>pos)
	{
		q = q->siguiente;
		i--;
	}

	p = q->punt;
	p->ax = x;
	p->ay = y;
	p->ath = angle;
	sfSetLocalCoords(p);
}

EXPORT float llObtenirXPunt(LlistaPunts l, int pos)
{
	telemento *q;
	int i;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es pot obtenir punt");
		return 0.0;
	}

	if (pos<0 || pos>=llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte");
		return 0.0;
	}

	q = l.punts[0];
	i = llNumPunts(l)-1;

	while (i>pos)
	{
		q = q->siguiente;
		i--;
	}

	return q->punt->x;
}

EXPORT float llObtenirYPunt(LlistaPunts l, int pos)
{
	telemento *q;
	int i;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es pot obtenir punt");
		return 0.0;
	}

	if (pos<0 || pos>=llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte");
		return 0.0;
	}

	q = l.punts[0];
	i = llNumPunts(l)-1;

	while (i>pos)
	{
```

```
		q = q->siguiente;
		i--;
	}

	return q->punt->y;
}

EXPORT float llObtenirAnglePunt(LlistaPunts l, int pos)
{
	telemento *q;
	int i;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es pot obtenir punt");
		return 0.0;
	}

	if (pos<0 || pos>=llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte");
		return 0.0;
	}

	q = l.punts[0];
	i = llNumPunts(l)-1;

	while (i>pos)
	{
		q = q->siguiente;
		i--;
	}

	return q->punt->th;
}

EXPORT float llObtenirXGlobalPunt(LlistaPunts l, int pos)
{
	telemento *q;
	int i;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es pot obtenir punt");
		return 0.0;
	}

	if (pos<0 || pos>=llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte");
		return 0.0;
	}

	q = l.punts[0];
	i = llNumPunts(l)-1;

	while (i>pos)
	{
		q = q->siguiente;
		i--;
	}

	return q->punt->ax;
}

EXPORT float llObtenirYGlobalPunt(LlistaPunts l, int pos)
{
	telemento *q;
```

```
	int i;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es pot obtenir punt");
		return 0.0;
	}

	if (pos<0 || pos>=llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte");
		return 0.0;
	}

	q = l.punts[0];
	i = llNumPunts(l)-1;

	while (i>pos)
	{
		q = q->siguiente;
		i--;
	}

	return q->punt->ay;
}

EXPORT float llObtenirAngleGlobalPunt(LlistaPunts l, int pos)
{
	telemento *q;
	int i;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es pot obtenir punt");
		return 0.0;
	}

	if (pos<0 || pos>=llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte");
		return 0.0;
	}

	q = l.punts[0];
	i = llNumPunts(l)-1;

	while (i>pos)
	{
		q = q->siguiente;
		i--;
	}

	return q->punt->ath;
}

EXPORT Vector llObtenirVectorXLlista(LlistaPunts l)
{
	Vector v;
	int i;

	if (llNumPunts(l)==0)
	{
		v = CrearVector(1);
		InicialitzarVector(v);
		sfSMessage("Llista buida. Es crea un vector de 1 element. Valor 0");

		return v;
	}
```

```
	v = CrearVector(llNumPunts(l));

	i = 0;
	while (i<llNumPunts(l))
	{
		AssigVector(v,i,llObtenirXPunt(l,i));

		i++;
	}

	return v;
}

EXPORT Vector llObtenirVectorYLlista(LlistaPunts l)
{
	Vector v;
	int i;

	if (llNumPunts(l)==0)
	{
		v = CrearVector(1);
		InicialitzarVector(v);
		sfSMessage("Llista buida. Es crea un vector de 1 element. Valor 0");

		return v;
	}

	v = CrearVector(llNumPunts(l));

	i = 0;
	while (i<llNumPunts(l))
	{
		AssigVector(v,i,llObtenirYPunt(l,i));

		i++;
	}

	return v;
}

EXPORT Vector llObtenirVectorAngleLlista(LlistaPunts l)
{
	Vector v;
	int i;

	if (llNumPunts(l)==0)
	{
		v = CrearVector(1);
		InicialitzarVector(v);
		sfSMessage("Llista buida. Es crea un vector de 1 element. Valor 0");

		return v;
	}

	v = CrearVector(llNumPunts(l));

	i = 0;
	while (i<llNumPunts(l))
	{
		AssigVector(v,i,llObtenirAnglePunt(l,i));

		i++;
	}

	return v;
}
```

```
EXPORT Vector llObtenirVectorXGlobalLlista(LlistaPunts l)
{
	Vector v;
	int i;

	if (llNumPunts(l)==0)
	{
		v = CrearVector(1);
		InicialitzarVector(v);
		sfSMessage("Llista buida. Es crea un vector de 1 element. Valor 0");

		return v;
	}

	v = CrearVector(llNumPunts(l));

	i = 0;
	while (i<llNumPunts(l))
	{
		AssigVector(v,i,llObtenirXGlobalPunt(l,i));

		i++;
	}

	return v;
}

EXPORT Vector llObtenirVectorYGlobalLlista(LlistaPunts l)
{
	Vector v;
	int i;

	if (llNumPunts(l)==0)
	{
		v = CrearVector(1);
		InicialitzarVector(v);
		sfSMessage("Llista buida. Es crea un vector de 1 element. Valor 0");

		return v;
	}

	v = CrearVector(llNumPunts(l));

	i = 0;
	while (i<llNumPunts(l))
	{
		AssigVector(v,i,llObtenirYGlobalPunt(l,i));

		i++;
	}

	return v;
}

EXPORT Vector llObtenirVectorAngleGlobalLlista(LlistaPunts l)
{
	Vector v;
	int i;

	if (llNumPunts(l)==0)
	{
		v = CrearVector(1);
		InicialitzarVector(v);
		sfSMessage("Llista buida. Es crea un vector de 1 element. Valor 0");

		return v;
	}
```

```
	v = CrearVector(llNumPunts(l));

	i = 0;
	while (i<llNumPunts(l))
	{
		AssigVector(v,i,llObtenirAngleGlobalPunt(l,i));

		i++;
	}

	return v;
}

EXPORT void llEscriureLlistaPunts(LlistaPunts l)
{
	int i;

	i = 0;
	while (i<llNumPunts(l))
	{
		sfSMessage("Punt %d: ( %f , %f , <%f )", i,
			llObtenirXPunt(l,i), llObtenirYPunt(l,i),
			llObtenirAnglePunt(l,i));

		i++;
	}
}

EXPORT void llEscriureLlistaPuntsGlobals(LlistaPunts l)
{
	int i;

	i = 0;
	while (i<llNumPunts(l))
	{
		sfSMessage("Punt %d: ( %f , %f , <%f )", i,
			llObtenirXGlobalPunt(l,i), llObtenirYGlobalPunt(l,i),
			llObtenirAngleGlobalPunt(l,i));

		i++;
	}
}

// ----------------------------------
// Implementació Funcions geomètriques
// ----------------------------------

EXPORT LlistaPunts CalcularBezier(LlistaPunts l, int k, int num_punts)
{
	LlistaPunts llista;
	Vector x,y;
	int j,n;
	float interval;

	llista = llCrearLlistaPunts();
	x = llObtenirVectorXLlista(l);
	y = llObtenirVectorYLlista(l);

	n = DimVector(x) - 1;
	if (n<1)
	{
			sfSMessage("Llista de punts incorrecte");
			sfSMessage("Es retorna llista buida");
			return llista;
	}

	num_punts--;
	if (num_punts<1)
```

```
	{
		sfSMessage("Nombre de punts incorrecte");
		sfSMessage("Es retorna llista buida");
		return llista;
	}

	interval = (float)(1.0/num_punts);

	j = 0;

	while (j<=num_punts)
	{
		float u;
		float x_bezier, y_bezier, bkn;

		u = j*interval;

		// Calcul de valors de x i de y de bezier

		x_bezier = 0.0;
		y_bezier = 0.0;

		k = 0;
		while (k<=n)
		{
			bkn = CalcularBkn(k,n,u);
			x_bezier += ConsVector(x,k)*bkn;
			y_bezier += ConsVector(y,k)*bkn;

			k++;
		}

		// Fi calcul
		llAfegirPunt(llista,x_bezier,y_bezier,0.0);

		j++;
	}

	DestruirVector(x);
	DestruirVector(y);

	return llista;
}

EXPORT LlistaPunts CalcularSpline(LlistaPunts l, Vector knots, int k, int
num_punts)
{
	LlistaPunts llista;
	Vector x,y;
	int n,i,j;
	float uinici, ufi, interval;


	x = llObtenirVectorXLlista(l);
	y = llObtenirVectorYLlista(l);

	llista = llCrearLlistaPunts();

	n = DimVector(x) - 1;
	if (n<1)
	{
			sfSMessage("Llista de punts incorrecte");
			sfSMessage("Es retorna llista buida");
			return llista;
	}

	if (DimVector(knots)!=n+k+1)
	{
```

```
                sfSMessage("Vector de nusos incorrecte");
                sfSMessage("Es retorna llista buida");
                return llista;
        }

        if (k<1)
        {
                sfSMessage("Vector de k incorrecte");
                sfSMessage("Es retorna llista buida");
                return llista;
        }

        uinici = ConsVector(knots,k-1);
        ufi = ConsVector(knots,n+1);

        // Falta comprovar vector de knots correcte

        num_punts--;
        if (num_punts<1)
        {
                sfSMessage("Nombre de punts incorrecte");
                sfSMessage("Es retorna llista buida");
                return llista;
        }

        interval = (ufi-uinici)/num_punts;

        j = 0;

        while (j<=num_punts)
        {
                float u;
                float x_spline, y_spline, nik;

                u = uinici+(j*interval);

                // Calcul de valors de x i de y de spline

                x_spline = 0.0;
                y_spline = 0.0;

                i = 0;
                while (i<=n)
                {
                        nik = CalcularNik(u,knots,i,k);
                        x_spline += ConsVector(x,i)*nik;
                        y_spline += ConsVector(y,i)*nik;

                        i++;
                }

                // Fi calcul

                llAfegirPunt(llista,x_spline,y_spline,0.0);

                j++;
        }

        DestruirVector(x);
        DestruirVector(y);

        return llista;
}

EXPORT LlistaPunts CalcularStandardSpline(LlistaPunts l, int k, int num_punts)
{
        LlistaPunts llista;
        Vector knots;
```

```
	int i,n;

	llista = llCrearLlistaPunts();
	n = llNumPunts(l)-1;
	knots = CrearVector(n+k+1);

	i = 0;
	while (i<DimVector(knots))
	{
		AssigVector(knots,i,(float)i);

		i++;
	}

	llista = CalcularSpline(l,knots,k,num_punts);
	DestruirVector(knots);

	return llista;
}

EXPORT int MinimaDistanciaLlistaPunts(LlistaPunts l)
{
	//Es fan càlculs amb distàncies^2 per augmentar la rapidesa de càlculs

	int i, min;
	float x, y;
	double distmin;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista buida. Retorn negatiu");
		return -1;
	}

	x = llObtenirXPunt(l,0);
	y = llObtenirYPunt(l,0);
	distmin = pow(x,2)+pow(y,2);

	min = 0;
	i = 1;
	while (i<llNumPunts(l))
	{
		x = llObtenirXPunt(l,i);
		y = llObtenirYPunt(l,i);
		if ((pow(x,2)+pow(y,2))<distmin)
		{
			distmin = pow(x,2)+pow(y,2);
			min = i;
		}

		i++;
	}

	return min;
}

EXPORT int MaximaDistanciaLlistaPunts(LlistaPunts l)
{
	//Es fan càlculs amb distàncies^2 per augmentar la rapidesa de càlculs

	int i, max;
	float x, y;
	double distmax;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista buida. Retorn negatiu");
		return -1;
```

```
	}

	x = llObtenirXPunt(l,0);
	y = llObtenirYPunt(l,0);
	distmax = pow(x,2)+pow(y,2);

	max = 0;
	i = 1;
	while (i<llNumPunts(l))
	{
		x = llObtenirXPunt(l,i);
		y = llObtenirYPunt(l,i);
		if ((pow(x,2)+pow(y,2))>distmax)
		{
			distmax = pow(x,2)+pow(y,2);
			max = i;
		}

		i++;
	}

	return max;
}

EXPORT float llDistanciaPunt(LlistaPunts l, int pos)
{
	telemento *q;
	int i;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es pot obtenir punt");
		return 0.0;
	}

	if (pos<0 || pos>=llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte");
		return -1.0;
	}

	q = l.punts[0];
	i = llNumPunts(l)-1;

	while (i>pos)
	{
		q = q->siguiente;
		i--;
	}

	return sfPointDist(q->punt);
}

EXPORT float llAnglePunt(LlistaPunts l, int pos)
{
	telemento *q;
	int i;

	if (llNumPunts(l)==0)
	{
		sfSMessage("Llista Buida. No es pot obtenir punt");
		return 0.0;
	}

	if (pos<0 || pos>=llNumPunts(l))
	{
		sfSMessage("Index pos incorrecte");
		return -1.0;
```

```
	}

	q = l.punts[0];
	i = llNumPunts(l)-1;

	while (i>pos)
	{
		q = q->siguiente;
		i--;
	}

	return sfPointPhi(q->punt);
}

EXPORT float llLongitudLlista(LlistaPunts l)
{
	return llLongitudLlistaInterval(l,0,llNumPunts(l)-1);
}

EXPORT float llLongitudLlistaInterval(LlistaPunts l, int inici, int fi)
{
	int i;
	float longitud, x, y;

	if (inici<=0)
	{
		inici = 0;
	}
	if (inici>=llNumPunts(l)-1) return 0.0;
	if (fi<0) fi = llNumPunts(l);
	if (fi<=inici) return 0.0;
	if (fi>=llNumPunts(l)) fi = llNumPunts(l)-1;

	longitud = 0.0;

	i = inici;
	while (i<=(fi-1))
	{
		x = llObtenirXPunt(l,i) - llObtenirXPunt(l,i+1);
		y = llObtenirYPunt(l,i) - llObtenirYPunt(l,i+1);

		longitud += (float)sqrt(pow(x,2)+pow(y,2));

		i++;
	}

	return longitud;
}

// ---------------------------------
// Implementació de Funcions internes
// ---------------------------------

float CalcularNik(float u, Vector knots, int i, int k)
{
	float num1, num2, den1, den2, valorfinal;

	valorfinal = 0.0;

	if (k==1)
	{
		if ((ConsVector(knots,i)<=u)&&(u<ConsVector(knots,i+1)))
			valorfinal = 1.0;
		else
			valorfinal = 0.0;
	}
	else
	{
```

```
		num1 = u - ConsVector(knots,i);
		den1 = ConsVector(knots,i+k-1) - ConsVector(knots,i);

		num2 = ConsVector(knots, i+k) -u;
		den2 = ConsVector(knots,i+k) - ConsVector(knots,i+1);

		if (den1!=0.0)
		{
			valorfinal += (num1/den1)*CalcularNik(u,knots,i,k-1);
		}

		if (den2!=0.0)
		{
			valorfinal += (num2/den2)*CalcularNik(u,knots,i+1,k-1);
		}
	}

	return valorfinal;
}

float CalcularBkn(int k, int n, float u)
{
	float aux;

	aux = CalcularCnk(n,k)*potencia(u,k)*potencia(1-u,n-k);
	return aux;
}

float CalcularCnk(int n, int k)
{
	float aux;
	aux = factorial(n);
	aux /= factorial(k);
	aux /= factorial(n-k);
	return aux;
}

float factorial(int n)
{
	float aux;

	assert(n>=0);
	aux = 1.0;
	while (n>0)
	{
		aux *= n;
		n--;
	}

	return aux;
}

float potencia(float u, int k)
{
	return (float)pow((double)u,(double)k);
}


// Funcio de proves
EXPORT void
sfLoadInit(void)    // Fitxer que s'avalua al carregar-se la dll
{
	// Proves

	LlistaPunts llista;
	float aux;

	llista = llCrearLlistaPunts();
```

```
        llAfegirPunt(llista,0.0,0.0,0.0);
        llAfegirPunt(llista,100.0,0.0,0.0);
        llAfegirPunt(llista,400.0,400.0,0.0);
        llAfegirPunt(llista,600.0,400.0,0.0);

        aux = llLongitudLlista(llista);
        sfSMessage("Longitud total: %f",aux);
        sfSMessage("");

        aux = llLongitudLlistaInterval(llista,0,1);
        sfSMessage("Interval 0,1: %f",aux);

        aux = llLongitudLlistaInterval(llista,0,2);
        sfSMessage("Interval 0,2: %f",aux);

        aux = llLongitudLlistaInterval(llista,0,3);
        sfSMessage("Interval 0,3: %f",aux);

        aux = llLongitudLlistaInterval(llista,1,2);
        sfSMessage("Interval 1,2: %f",aux);

        aux = llLongitudLlistaInterval(llista,1,3);
        sfSMessage("Interval 1,3: %f",aux);

        aux = llLongitudLlistaInterval(llista,2,3);
        sfSMessage("Interval 2,3: %f",aux);

        sfSMessage("");

        aux = llLongitudLlistaInterval(llista,0,0);
        sfSMessage("Interval 0,0: %f",aux);

        aux = llLongitudLlistaInterval(llista,1,1);
        sfSMessage("Interval 1,1: %f",aux);

        aux = llLongitudLlistaInterval(llista,2,2);
        sfSMessage("Interval 2,2: %f",aux);

        aux = llLongitudLlistaInterval(llista,3,3);
        sfSMessage("Interval 3,3: %f",aux);

        llDestruirLlistaPunts(llista);

        // Missatge inici i final
        sfSMessage("Iniciant càrrega sfUtils.dll");
        sfSMessage("Càrrega de sfUtils.dll finalitzada");
}
```

3 sfReachPosition

3.1 behavior.beh

```
float angle_entrada;

BeginBehavior sfTurnToPosition
	Params
	Rules
		If aen Then Turn Right
		If aeng Then Turn Right Sharply
		If aep Then Turn Left
		If aepg Then Turn Left Sharply
	Update
		angle_entrada = llAnglePunt(punts,1);
		dist_zero = f_eq(llDistanciaPunt(punts,0),0.0,p_dist_tol*3);

		aeng = straight_down(angle_entrada,-90.0,-60.0);
		aen = straight_down(angle_entrada,-15.0,-5.0);
		aez = f_eq(angle_entrada,0.0,5.0);
		aep = up_straight(angle_entrada,5.0,15.0);
		aepg = up_straight(angle_entrada,60.0,90.0);
	Activity
		Turn dist_zero
		Goal aez And dist_zero
EndBehavior


static float at_max_speed = 400.0;
#define GP_TURN_FACTOR 6.0

BeginBehavior sfAnarAPos

Params
  sfFLOAT speed
  sfFLOAT radius

Rules
  Approach_speed  If at_too_slow Or at_too_fast Then Speed at_speed
  Pos_on_left     If at_pos_on_left  Then Turn Left  at_turn_amount
  Pos_on_right    If at_pos_on_right Then Turn Right at_turn_amount
  Pos_here     If at_pos_achieved Then Speed 0.0
  Angle_way_off   If at_angle_way_off Then Speed 0.0

Update
  float x, y, phi, delta;
  int lhs_clear, rhs_clear;
  float ach, left, right, bleft, bright, wayoff;
  float too_slow, too_fast;

  at_speed = MIN(speed,at_max_speed);
  too_fast = f_greater(sfTargetVel(),speed,(float)(speed*.2));
  too_slow = f_smaller(sfTargetVel(),at_speed,(float)(at_speed*.2));

  x = llObtenirXPunt(punts,0);
  y = llObtenirYPunt(punts,0);
  phi = llAnglePunt(punts,0);

  delta = sfTargetHead();
```

```
  if (phi < 0.0)
    {
      if (delta > phi) delta = delta - phi;
      else delta = 0.0;
    }
  else
    {
      if (delta < phi) delta = phi - delta;
      else delta = 0.0;
    }
  at_turn_amount = (float)(GP_TURN_FACTOR) * f_not(f_near(delta,6.0));
  rhs_clear = occupied(300,-600,400,600,1);
  lhs_clear = occupied(300,600,400,600,1);

  ach = straight_down(MAX(ABS(x),ABS(y)),radius,(float)(radius * 0.5));
  left = up_straight(phi,10.0,30.0);
  right = up_straight(-phi,10.0,30.0);
  bleft = straight_down((float)lhs_clear,100.0,300.0);
  bright = straight_down((float)rhs_clear,100.0,300.0);
  wayoff = up_straight(ABS(phi),50.0,70.0);

  at_pos_on_left = f_and(left, f_or(wayoff,f_not(bleft)));
  at_pos_on_right = f_and(right, f_or(wayoff,f_not(bright)));
  at_pos_achieved = f_very(f_very(ach));
  at_angle_way_off = wayoff;

  at_too_slow = f_and(too_slow, f_and(f_not(ach), f_not(wayoff)));
  at_too_fast = f_and(too_fast, f_not(wayoff));

Activity
  Speed 1.0
  Turn  1.0
  Progress 1.0
  Goal  ach

EndBehavior
```

3.2 sfReachposition.h

```
/* ********************** */
/* FITXER sfReachPosition.h     */
/* ********************** */

// --------------------
// Inclusió de funcions
// --------------------

#include "sfUtils.h"

// ---------------------
// Definició de funcions
// ---------------------

void sfrStop(void);
void sfrStart(void);

void sfrInactive(void);

void sfrSetPosition(float x, float y, float angle);
void sfrSetGlobalPosition(float x, float y, float angle);
void sfrSetVelocity(float v);

float sfrGetXPosition(void);
float sfrGetYPosition(void);
float sfrGetAnglePosition(void);
```

```
float sfrGetXGlobalPosition(void);
float sfrGetYGlobalPosition(void);
float sfrGetGlobalAnglePosition(void);

float sfrGetVelocity(void);

int sfrIsPositionReach(void);
int sfrIsPointReach(void);
float sfrGetDistancePosition(void);

// ------------------------------
// Definició de variables globals
// ------------------------------

LlistaPunts punts;

// goal = punts[0]              => punt destí
// orientacio = punts[1]  => punt que marca la orientació

float velocitat;

int en_funcionament = 0;
int inactive = 1;

#define PI 3.1416
#define L_VECTOR 500

// ---------------------------------------
// Definició de paràmetres de funcionament
// ---------------------------------------

// Declaració dels paràmetres

int p_priority = 1;
float p_dist_tol = 50.0;
float p_angle_tol = 6.0;

// Funcions d'accés

int sfrGetPriority(void);
float sfrGetDistTol(void);
float sfrGetAngleTol(void);

void sfrSetPriority(int priority);
void sfrSetDistTol(float tol);
void sfrSetAngleTol(float tol);
```

3.3 sfReachposition.c

```
/* ********************** */
/* FITXER sfReachPosition.c      */
/* ********************** */

#include "saphira.h"
#include "sfReachPosition.h"
#include "beh.c"

// ------------------------------
// Definició de funcions internes
// ------------------------------

void ActivarComportaments(void)
{
       if (!inactive)
       {
              if (en_funcionament) sfrStart();
```

```
            else sfrStop();
      }
}

// --------------------
// Codi de les funcions
// --------------------

void sfrInactive(void)
{
      sfRemoveTask("assolir_pos");
      sfRemoveTask("assolir_angle");
      sfRemoveTask("sfrStop");

      inactive = 1;
}

void sfrStart(void)
{
      sfStartBehavior(sfAnarAPos,"assolir_pos", 0, p_priority, 0,
            velocitat, p_dist_tol);
      sfStartBehavior(sfTurnToPosition,"assolir_angle", 0, p_priority, 0);
      sfStartBehavior(sfStop,"sfrStop", 0, p_priority, 1);

      en_funcionament = 1;
      inactive = 0;
}

void sfrStop(void)
{
      sfStartBehavior(sfAnarAPos,"assolir_pos", 0, p_priority, 1,
            velocitat, p_dist_tol);
      sfStartBehavior(sfTurnToPosition,"assolir_angle", 0, p_priority, 1);
      sfStartBehavior(sfStop,"sfrStop", 0, p_priority, 0);

      en_funcionament = 0;
      inactive = 0;
}

void sfrSetPosition(float x, float y, float angle)
{
      llCambiarPunt(punts,0,x,y,angle);

      x = x+L_VECTOR*(float)cos(angle*2*PI/360);
      y = y+L_VECTOR*(float)sin(angle*2*PI/360);

      llCambiarPunt(punts,1,x,y,angle);

      ActivarComportaments();
}

void sfrSetGlobalPosition(float x, float y, float angle)
{
      llCambiarPuntGlobal(punts,0,x,y,angle);

      x = x+L_VECTOR*(float)cos(angle*2*PI/360);
      y = y+L_VECTOR*(float)sin(angle*2*PI/360);

      llCambiarPuntGlobal(punts,1,x,y,angle);

      ActivarComportaments();
}

void sfrSetVelocity(float v)
{
      velocitat = v;

      // Per tal de fer el canvi de velocitat inmediatament
```

```
	// es tornen a iniciar els comportaments

	ActivarComportaments();
}

float sfrGetXPosition(void)
{
	return llObtenirXPunt(punts,0);
}

float sfrGetYPosition(void)
{
	return llObtenirYPunt(punts,0);
}

float sfrGetAnglePosition(void)
{
	return llObtenirAnglePunt(punts,0);
}

float sfrGetXGlobalPosition(void)
{
	return llObtenirXGlobalPunt(punts,0);
}

float sfrGetYGlobalPosition(void)
{
	return llObtenirYGlobalPunt(punts,0);
}

float sfrGetGlobalAnglePosition(void)
{
	return llObtenirAngleGlobalPunt(punts,0);
}

float sfrGetVelocity(void)
{
	return velocitat;
}

int sfrIsPositionReach(void)
{
	return ((sfGetTaskState("assolir_pos")==sfSUCCESS)&&
		(abs(llAnglePunt(punts,1))<p_angle_tol));
}

int sfrIsPointReach(void)
{
	return ((sfGetTaskState("assolir_pos")==sfSUCCESS));
}

float sfrGetDistancePosition(void)
{
	return llDistanciaPunt(punts,0);
}

// -----------------------------------
// Codi dels paràmetres de funcionament
// -----------------------------------

int sfrGetPriority(void)
{
	return p_priority;
}

float sfrGetDistTol(void)
{
	return p_dist_tol;
```

```
}

float sfrGetAngleTol(void)
{
	return p_angle_tol;
}

void sfrSetPriority(int priority)
{
	p_priority = priority;

	ActivarComportaments();
}

void sfrSetDistTol(float tol)
{
	p_dist_tol = tol;

	ActivarComportaments();
}

void sfrSetAngleTol(float tol)
{
	p_angle_tol = tol;

	ActivarComportaments();
}

// ----------------------------------------
// Fitxers de càrrega i descàrrega de la dll
// ----------------------------------------

EXPORT void
sfLoadInit(void)    // Fitxer que s'avalua al carregar-se la dll
{
	// Missatge d'inici

	sfSMessage("Iniciant càrrega sfReachPosition.dll");

	// Càrrega de les funcions a Colbert

	sfAddEvalFn("sfrSetPosition", sfrSetPosition, sfVOID, 3, sfFLOAT, sfFLOAT,
sfFLOAT );
	sfAddEvalFn("sfrSetGlobalPosition", sfrSetGlobalPosition, sfVOID, 3,
sfFLOAT, sfFLOAT, sfFLOAT );
	sfAddEvalFn("sfrSetVelocity", sfrSetVelocity, sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfrGetXPosition", sfrGetXPosition, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfrGetYPosition", sfrGetYPosition, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfrGetAnglePosition", sfrGetAnglePosition, sfFLOAT, 0,
sfVOID);
	sfAddEvalFn("sfrGetXGlobalPosition", sfrGetXGlobalPosition, sfFLOAT, 0,
sfVOID);
	sfAddEvalFn("sfrGetYGlobalPosition", sfrGetYGlobalPosition, sfFLOAT, 0,
sfVOID);
	sfAddEvalFn("sfrGetGlobalAnglePosition", sfrGetGlobalAnglePosition,
sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfrGetVelocity", sfrGetVelocity, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfrIsPositionReach", sfrIsPositionReach, sfINT, 0, sfVOID);
	sfAddEvalFn("sfrIsPointReach", sfrIsPointReach, sfINT, 0, sfVOID);
	sfAddEvalFn("sfrGetDistancePosition", sfrGetDistancePosition, sfFLOAT, 0,
sfVOID);

	sfAddEvalFn("sfrGetPriority", sfrGetPriority, sfINT, 0, sfVOID);
	sfAddEvalFn("sfrSetPriority", sfrSetPriority, sfVOID, 1, sfINT);
	sfAddEvalFn("sfrGetDistTol", sfrGetDistTol, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfrSetDistTol", sfrSetDistTol, sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfrGetAngleTol", sfrGetAngleTol, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfrSetAngleTol", sfrSetAngleTol, sfVOID, 1, sfFLOAT);
```

```
	sfAddEvalFn("sfrStart", sfrStart, sfVOID, 0, sfVOID);
	sfAddEvalFn("sfrStop", sfrStop, sfVOID, 0, sfVOID);
	sfAddEvalFn("sfrInactive", sfrInactive, sfVOID, 0, sfVOID);

	// sfRegisterSchema("sfTurnToPosition",sfBEHAVIOR,sfTurnToPosition);

	// Inicialitzar variables per defecte

	punts = llCrearLlistaPunts();

	llAfegirPunt(punts,0,0,0);
	llAfegirPunt(punts,L_VECTOR,0,0);

	velocitat = 100.0;

	// Missatge final

	sfSMessage("Càrrega de sfReachPosition.dll finalitzada");
}

EXPORT void
sfLoadExit(void)    // Fitxer que s'avalua al descarregar-se la dll
{
	sfrInactive();

	// Treiem els punts de la llista

	llDestruirLlistaPunts(punts);

	// Missatge de descàrrega

	sfSMessage("Descàrrega de reachPosition.dll efectuada");
}
```

3.4 beh.c

```
//Starting Behavior/Act Schema Parser
float angle_entrada;

//Defining behavior sfTurnToPosition


 /* for reference pointers */

static int _sfleft = TURN_LEFT;
static int _sfright = TURN_RIGHT;
static int _sfbackward = DECEL;
static int _sfforward = ACCEL;
static int _sfspeed = ASPEED;




/* Starting code for behavior sfTurnToPosition  */


void sfTurnToPosition_update();
void sfTurnToPosition_setup();

/* global vars for update fn */
static float aen;
static float mod__0;
static float aeng;
static float mod__1;
static float aep;
```

```
static float mod__2;
static float aepg;
static float mod__3;

behavior
_beh_sfTurnToPosition = {"sfTurnToPosition",
    sfTurnToPosition_setup, /* setup function */
    sfTurnToPosition_update, /* body function */
    0, /* number of params */
    {sfINT},
    4, /* number of rules */
    {{"rule__0", &aen, &_sfright, &mod__0},
     {"rule__1", &aeng, &_sfright, &mod__1},
     {"rule__2", &aep, &_sfleft, &mod__2},
     {"rule__3", &aepg, &_sfleft, &mod__3}
    }
};
behavior *sfTurnToPosition = &_beh_sfTurnToPosition;



void
sfTurnToPosition_setup (beh_params _params) {
}

/* global vars for update fn */
static float dist_zero;
static float aez;



void
sfTurnToPosition_update(beh_params _params)
{

             angle_entrada = llAnglePunt(punts,1);
             dist_zero = f_eq(llDistanciaPunt(punts,0),0.0,p_dist_tol*3);

             aeng = straight_down(angle_entrada,-90.0,-60.0);
             aen = straight_down(angle_entrada,-15.0,-5.0);
             aez = f_eq(angle_entrada,0.0,5.0);
             aep = up_straight(angle_entrada,5.0,15.0);
             aepg = up_straight(angle_entrada,60.0,90.0);
       /* Begin activity section */
   c_act_v = c_act_t = 0.0;
    c_act_t = dist_zero;
    c_goal = f_and(aez,dist_zero);
    mod__0 = sfMODERATELY;
    mod__1 = sfSHARPLY;
    mod__2 = sfMODERATELY;
    mod__3 = sfSHARPLY;
}


/* Ending code for behavior sfTurnToPosition  */



static float at_max_speed = 400.0;
#define GP_TURN_FACTOR 6.0

//Defining behavior sfAnarAPos


/* Starting code for behavior sfAnarAPos  */
```

```
void sfAnarAPos_update();
void sfAnarAPos_setup();

/* global vars for update fn */
static float at_too_slow;
static float at_too_fast;
static float at_speed;
static float ant__4;
static float at_pos_on_left;
static float at_turn_amount;
static float ant__5;
static float at_pos_on_right;
static float ant__6;
static float at_pos_achieved;
static float speed__7 = 0.0;
static float ant__8;
static float at_angle_way_off;
static float speed__9 = 0.0;
static float ant__10;

behavior
_beh_sfAnarAPos = {"sfAnarAPos",
    sfAnarAPos_setup, /* setup function */
    sfAnarAPos_update, /* body function */
    2, /* number of params */
    {sfFLOAT, sfFLOAT},
    5, /* number of rules */
    {{"Approach_speed", &ant__4, &_sfspeed, &at_speed},
     {"Pos_on_left", &ant__5, &_sfleft, &at_turn_amount},
     {"Pos_on_right", &ant__6, &_sfright, &at_turn_amount},
     {"Pos_here", &ant__8, &_sfspeed, &speed__7},
     {"Angle_way_off", &ant__10, &_sfspeed, &speed__9}
    }
};
behavior *sfAnarAPos = &_beh_sfAnarAPos;



void
sfAnarAPos_setup (beh_params _params) {
   float speed = _params[0].f;
   float radius = _params[1].f;
}

/* global vars for update fn */
static float ach;



void
sfAnarAPos_update(beh_params _params)
{
   float speed = _params[0].f;
   float radius = _params[1].f;

  float x, y, phi, delta;
  int lhs_clear, rhs_clear;
  float ach, left, right, bleft, bright, wayoff;
  float too_slow, too_fast;

  at_speed = MIN(speed,at_max_speed);
  too_fast = f_greater(sfTargetVel(),speed,(float)(speed*.2));
  too_slow = f_smaller(sfTargetVel(),at_speed,(float)(at_speed*.2));

  x = llObtenirXPunt(punts,0);
  y = llObtenirYPunt(punts,0);
  phi = llAnglePunt(punts,0);
```

```
  delta = sfTargetHead();
  if (phi < 0.0)
    {
      if (delta > phi) delta = delta - phi;
      else delta = 0.0;
    }
  else
    {
      if (delta < phi) delta = phi - delta;
      else delta = 0.0;
    }
  at_turn_amount = (float)(GP_TURN_FACTOR) * f_not(f_near(delta,6.0));
  rhs_clear = occupied(300,-600,400,600,1);
  lhs_clear = occupied(300,600,400,600,1);

  ach = straight_down(MAX(ABS(x),ABS(y)),radius,(float)(radius * 0.5));
  left = up_straight(phi,10.0,30.0);
  right = up_straight(-phi,10.0,30.0);
  bleft = straight_down((float)lhs_clear,100.0,300.0);
  bright = straight_down((float)rhs_clear,100.0,300.0);
  wayoff = up_straight(ABS(phi),50.0,70.0);

  at_pos_on_left = f_and(left, f_or(wayoff,f_not(bleft)));
  at_pos_on_right = f_and(right, f_or(wayoff,f_not(bright)));
  at_pos_achieved = f_very(f_very(ach));
  at_angle_way_off = wayoff;

  at_too_slow = f_and(too_slow, f_and(f_not(ach), f_not(wayoff)));
  at_too_fast = f_and(too_fast, f_not(wayoff));

 /* Begin activity section */
   c_act_v = c_act_t = 0.0;
    c_act_v = 1.0;
    c_act_t = 1.0;
    c_progress = 1.0;
    c_goal = ach;
    ant__4 = f_or(at_too_slow,at_too_fast);
    ant__5 = at_pos_on_left;
    ant__6 = at_pos_on_right;
    ant__8 = at_pos_achieved;
    ant__10 = at_angle_way_off;
}
/* Ending code for behavior sfAnarAPos  */
//Ending Behavior/Act Schema Parser
```

4 sfDoTrajectory

4.1 behavior.beh

```
BeginBehavior sfDTFollowTrajectory
	Params
	Rules
		If esquerra Then Turn Left Very Slowly
		If molt_esquerra Then Turn Left
		If dreta Then Turn Right Very Slowly
		If molt_dreta Then Turn Right
	Update
		punt_seguiment = BuscarPuntSeguiment();
		final = (float)(punt_seguiment==llNumPunts(trajectoria)-1);

		if (final<0.5)
		{
			float angle_aux;

			angle_aux = llAnglePunt(trajectoria,punt_seguiment);
			if (sentit==-1)
			{
				angle_aux = sfAdd2Angle(angle_aux,180.0);
			}

			esquerra = up_straight(angle_aux,0.0,+min_angle);
			molt_esquerra = up_straight(angle_aux,90.0,90+min_angle);
			dreta = straight_down(angle_aux,-min_angle,0.0);
			molt_dreta = straight_down(angle_aux,-90-min_angle,-90.0);
		}
		else
		{
			PararVelocitat();

			esquerra = 0.0;
			dreta = 0.0;
			molt_esquerra = 0.0;
			molt_dreta = 0.0;
		}
	Activity
		Turn 1.0
		Goal final
EndBehavior


BeginBehavior sfDTParar
	Params
	Rules
		If 1.0 Then Speed 0.0
	Update
		vel_zero = (float)(sfTargetVel()==0.0);
		final = (float)(punt_seguiment==llNumPunts(trajectoria)-1);
		if (final>0.5) punt_seguiment = 0;

	Activity
		Speed final
		Goal vel_zero And final
EndBehavior
```

```
BeginBehavior sfDTVelocitatConstant
	Params
	Rules
		If desviat Then Speed vel_parada
		If molt_desviat Then Speed 0.0
		If Not(desviat Or molt_desviat) Then Speed velocitat
	Update
		{
			float angle_aux;

			angle_aux = llAnglePunt(trajectoria,punt_seguiment);
			if (sentit==-1)
			{
				angle_aux = sfAdd2Angle(angle_aux,180.0);
			}

			velocitat = velocity*sentit;
			vel_parada = velocity * v_slow;
			desviat = f_or(up_straight(angle_aux,0.0,+min_angle),
				straight_down(angle_aux,-min_angle,0.0));
			molt_desviat = f_or(straight_down(angle_aux,-min_angle,0.0),
				straight_down(angle_aux,-90-min_angle,-90.0));
		}

	Activity
		Speed 1.0
EndBehavior
```

4.2 sfDoTrajectory.h

```
/* ********************** */
/* FITXER sfDoTrajectory.h */
/* ********************** */

// ------------------------
// Inclusió de definicions
// ------------------------

#include "sfUtils.h"
#include "saphira.h"

// ----------------------
// Definició de constants
// ----------------------

#define PI 3.1416

// ---------------------
// Definició de funcions
// ---------------------

void sfDTStart(void);
void sfDTStop(void);

void sfDTSetInactive(void);

int sfDTIsTrajectoryReach(void);
float sfDTTrajectoryDistance(void);

void sfDTAddPoint(float x, float y);
void sfDTAddGlobalPoint(float x, float y);
void sfDTRemoveAllPoints(void);
int sfDTGetNumberOfPoints(void);
void sfDTWritePoints(void);
void sfDTWriteGlobalPoints(void);
```

```
void sfDTSetForward(void);
void sfDTSetBackward(void);
int sfDTIsForward(void);
int sfDTIsBackward(void);

void sfDTSetVelocity(float v);
float sfDTGetVelocity(void);
void sfDTSetPriority(int p);
int sfDTGetPriority(void);

void sfDTSetSpline(void);
void sfDTSetBezier(void);
int sfDTIsSpline(void);
int sfDTIsBezier(void);

void sfDTSetKnots(int pos, float value);
void sfDTSetKTrajectory(int k);
int sfDTGetKTrajectory(void);
void sfDTSetResolution(float r);
float sfDTGetResolution(void);
void sfDTSetVSlow(float vs);
float sfDTGetVSlow(void);
void sfDTSetMinDist(float md);
float sfDTGetMinDist(void);
void sfDTSetMinAngle(float ma);
float sfDTGetMinAngle(void);

// Per un bon funcionament s'ha de cridar després de posar els punts
// Si no es fa així no té efecte (cada cop que s'afegeix un punt es reinicia)

// -------------------------------------------
// Definició de funcions Simplificades Tracking
// -------------------------------------------

void sfDTSTSetPosition(float x, float y, float angle);
void sfDTSTStart(void);
void sfDTSTStop(void);
int sfDTSTIsPositionReach(void);
void sfDTSTSetVelocity(float v);
void sfDTSTSetPriority(int p);

// ------------------------------
// Definició de variables globals
// ------------------------------

LlistaPunts llista;
LlistaPunts trajectoria;

// --------------------------------------
// Definició de paràmetres de funcionament
// --------------------------------------

int es_spline;
float velocity;
int priority;
Vector knots;
int ktrajectory;
float resolucio;
float v_slow;
float min_dist;
float min_angle;

int sentit;  // 1 Endavant, -1 Enrera

int en_funcionament = 0;
int inactive = 1;

// ------------------------------
```

```
// Definició de paràmetres d'estat
// -------------------------------

int punt_seguiment;
```

4.3 sfDoTrajectory.c

```
/* ********************** */
/* FITXER sfDoTrajectory.c*/
/* ********************** */

#include "sfDoTrajectory.h"

// -------------------------------
// Declaració de funcions internes
// -------------------------------

int BuscarPuntSeguiment(void);
int ComprovarKnots(void);
void IniciarComportaments(void);
void PararComportaments(void);
void PararVelocitat(void);

void ActivarComportaments(void)
{
	if (!inactive)
	{
		if (en_funcionament) sfDTStart();
		else sfDTStop();
	}
}

void DrawTrajectory(void)
{
	int numpunts;
	int knotsok;

	// Comprovar que tot és correcte (si no sortir)

	if (llNumPunts(llista)<=1) return;
	knotsok = ComprovarKnots();

	// Es destrueix la trajectoria actual

	llDestruirLlistaPunts(trajectoria);

	// Es calcula el nombre de punts estimats necessaris

	numpunts = (int)(llLongitudLlista(llista)/resolucio);
	if (numpunts<=(3*llNumPunts(llista)))
	{
		numpunts = 3*llNumPunts(llista);
	}

	// Calculem la trajectoria

	if (es_spline)
	{
		if (knotsok)
		{
			trajectoria =
CalcularSpline(llista,knots,ktrajectory,numpunts);
		}
		else
		{
```

```
                    trajectoria =
CalcularStandardSpline(llista,ktrajectory,numpunts);
             }
       }
       else
       {
             trajectoria = CalcularBezier(llista,ktrajectory,numpunts);
       }
}

// -------------------------
// Inclusió de comportaments
// -------------------------

#include "beh.c"

// --------------------
// Codi de les funcions
// --------------------

void sfDTSetInactive(void)
{
       sfRemoveTask("sfDTStop");
       sfRemoveTask("sfDTParar");
       sfRemoveTask("sfDTmantenir_velocitat");
       sfRemoveTask("sfDTFollowTrajectory");

       inactive = 1;
}

void sfDTSetForward(void)
{
       sentit = 1;

       ActivarComportaments();
}

void sfDTSetBackward(void)
{
       sentit = -1;

       ActivarComportaments();
}

int sfDTIsForward(void)
{
       return (sentit==1);
}

int sfDTIsBackward(void)
{
       return (sentit==-1);
}

void sfDTStart(void)
{
       DrawTrajectory();
       IniciarComportaments();
}

void sfDTStop(void)
{
       DrawTrajectory();
       PararComportaments();
}

int sfDTIsTrajectoryReach(void)
{
```

```
	return (punt_seguiment==llNumPunts(trajectoria)-1);
}

float sfDTTrajectoryDistance(void)
{
	return llLongitudLlistaInterval(trajectoria, punt_seguiment, -1);
}

void sfDTAddPoint(float x, float y)
{
	llAfegirPunt(llista, x, y, 0.0);
	DestruirVector(knots);
	knots = CrearVector(llNumPunts(llista)+ktrajectory);
	InicialitzarVector(knots);
	DrawTrajectory();

	ActivarComportaments();
}

void sfDTAddGlobalPoint(float x, float y)
{
	llAfegirPuntGlobal(llista, x, y, 0.0);
	DestruirVector(knots);
	knots = CrearVector(llNumPunts(llista)+ktrajectory);
	InicialitzarVector(knots);
	DrawTrajectory();

	ActivarComportaments();
}

void sfDTRemoveAllPoints(void)
{
	punt_seguiment = 0;

	llEliminarTotsPunts(llista);
	llEliminarTotsPunts(trajectoria);

	ActivarComportaments();
}

int sfDTGetNumberOfPoints(void)
{
	return llNumPunts(llista);
}

void sfDTWritePoints(void)
{
	llEscriureLlistaPunts(llista);
}

void sfDTWriteGlobalPoints(void)
{
	llEscriureLlistaPuntsGlobals(llista);
}

void sfDTSetVelocity(float v)
{
	if (v<=10.0) v = 10.0;
	if (v>=300.0) v = 300.0;

	velocity = v;

	ActivarComportaments();
}

float sfDTGetVelocity(void)
{
	return velocity;
```

```
}

void sfDTSetSpline(void)
{
	es_spline = 1;
	DrawTrajectory();

	ActivarComportaments();
}

void sfDTSetBezier(void)
{
	es_spline = 0;
	DrawTrajectory();

	ActivarComportaments();
}

int sfDTIsSpline(void)
{
	return es_spline;
}

int sfDTIsBezier(void)
{
	return (!es_spline);
}

void sfDTSetKnots(int pos, float value)
{
	if (pos<0 || pos>=DimVector(knots))
	{
		sfSMessage("Index incorrecte. No s'efectua cap acció");
		return;
	}

	AssigVector(knots,pos,value);

	if (ComprovarKnots())
	{
		DrawTrajectory();
		ActivarComportaments();
	}
}

void sfDTSetKTrajectory(int k)
{
	if (k<2) k = 2;

	ktrajectory = k;
	DrawTrajectory();

	ActivarComportaments();
}

int sfDTGetKTrajectory(void)
{
	return ktrajectory;
}

void sfDTSetPriority(int p)
{
	if (p<0) p = 0;

	priority = p;

	ActivarComportaments();
}
```

```
int sfDTGetPriority(void)
{
	return priority;
}

void sfDTSetResolution(float r)
{
	if (r<10.0) r = 10.0;

	resolucio = r;
	DrawTrajectory();

	ActivarComportaments();
}

float sfDTGetResolution(void)
{
	return resolucio;
}

void sfDTSetVSlow(float vs)
{
	if (vs<0.0) vs = 0.0;
	if (vs>1.0) vs = 1.0;

	v_slow = vs;

	ActivarComportaments();
}

float sfDTGetVSlow(void)
{
	return v_slow;
}

void sfDTSetMinDist(float md)
{
	if (md<0.0) md = 0.0;

	min_dist = md;

	ActivarComportaments();
}

float sfDTGetMinDist(void)
{
	return min_dist;
}

void sfDTSetMinAngle(float ma)
{
	if (ma<0.0) ma = 0.0;

	min_angle = ma;

	ActivarComportaments();
}

float sfDTGetMinAngle(void)
{
	return min_angle;
}

// -----------------------------
// Codi de les funcions internes
// -----------------------------
```

```
int BuscarPuntSeguiment(void)
{
	int i, punt;
	float dist;

	punt = llNumPunts(trajectoria)-1;
	dist = llDistanciaPunt(trajectoria,punt);
	if (dist<min_dist)
	{
		return punt;
	}

	i = MinimaDistanciaLlistaPunts(trajectoria);
	if (i<punt_seguiment) i = punt_seguiment;
	while (i<llNumPunts(trajectoria))
	{
		float aux;

		aux = llDistanciaPunt(trajectoria,i);
		if (aux>=min_dist && aux<dist)
		{
			dist = aux;
			punt = i;
		}

		i++;
	}

	return punt;
}

int ComprovarKnots(void)
{
	int i, correcte;

	if (DimVector(knots)<=2) return 0;

	correcte = 1;
	i = 0;
	while (i<(DimVector(knots)-2) && correcte)
	{
		correcte = (ConsVector(knots,i)<=ConsVector(knots,i+1));

		i++;
	}

	correcte = correcte && (ConsVector(knots,0)<
		ConsVector(knots,DimVector(knots)-1));

	return correcte;
}

void IniciarComportaments(void)
{
	sfStartBehavior(sfStop, "sfDTStop",
		0,priority,1);

	sfStartBehavior(sfDTParar, "sfDTParar",
		0,priority,0);

	sfStartBehavior(sfDTVelocitatConstant, "sfDTmantenir_velocitat",
		0,priority,0);

	sfStartBehavior(sfDTFollowTrajectory, "sfDTFollowTrajectory",
		0,priority,0);

	en_funcionament = 1;
	inactive = 0;
```

```
}

void ParарComportaments(void)
{
	sfStartBehavior(sfStop, "sfDTStop",
		0,priority,0);

	sfStartBehavior(sfDTParar, "sfDTParar",
		0,priority,1);

	sfStartBehavior(sfDTVelocitatConstant, "sfDTmantenir_velocitat",
		0,priority,1);

	sfStartBehavior(sfDTFollowTrajectory, "sfDTFollowTrajectory",
		0,priority,1);

	en_funcionament = 0;
	inactive = 0;
}

void PararVelocitat(void)
{
	sfStartBehavior(sfDTVelocitatConstant, "sfDTmantenir_velocitat",
		0,priority,1);
}

// ----------------------------------------
// Codi de les funcions simplificades tracking
// ----------------------------------------

void sfDTSTSetPosition(float x, float y, float angle)
{
	float vx, vy, aux;

	aux = (float)(sqrt(x*x+y*y)/3);
	if (aux>1200.0) aux = 1200.0;

	vx = aux*(float)(cos(angle*2*PI/360));
	vy = aux*(float)(sin(angle*2*PI/360));

	sfDTRemoveAllPoints();

	if (x>=0.0)
	{
		sfDTSetForward();

		sfDTAddPoint(0,0);
		sfDTAddPoint(aux/2,0);
		sfDTAddPoint(aux,0);
		sfDTAddPoint(x-vx,y-vy);
		sfDTAddPoint(x-(vx/2),y-(vy/2));
		sfDTAddPoint(x,y);
	}
	else
	{
		sfDTSetBackward();

		sfDTAddPoint(0,0);
		sfDTAddPoint(-aux/2,0);
		sfDTAddPoint(-aux,0);
		sfDTAddPoint(x+vx,y+vy);
		sfDTAddPoint(x+(vx/2),y+(vy/2));
		sfDTAddPoint(x,y);
	}

	sfDTSetBezier();
}
```

```
void sfDTSTStart(void)
{
	sfDTStart();
}

void sfDTSTStop(void)
{
	sfDTStop();
}

int sfDTSTIsPositionReach(void)
{
	return sfDTIsTrajectoryReach();
}

void sfDTSTSetVelocity(float v)
{
	sfDTSetVelocity(v);
}

void sfDTSTSetPriority(int p)
{
	sfDTSetPriority(p);
}

// ----------------------------------------
// Fitxers de càrrega i descàrrega de la dll
// ----------------------------------------

EXPORT void
sfLoadInit(void)    // Fitxer que s'avalua al carregar-se la dll
{
	// Missatge d'inici

	sfSMessage("Iniciant càrrega sfDoTrajectory.dll");

	// Càrrega de les funcions a Colbert

	sfAddEvalFn("sfDTStart", sfDTStart, sfVOID, 0, sfVOID);
	sfAddEvalFn("sfDTStop", sfDTStop, sfVOID, 0, sfVOID);

	sfAddEvalFn("sfDTSetInactive", sfDTSetInactive, sfVOID, 0, sfVOID);

	sfAddEvalFn("sfDTIsTrajectoryReach", sfDTIsTrajectoryReach, sfINT, 0,
sfVOID);
	sfAddEvalFn("sfDTTrajectoryDistance", sfDTTrajectoryDistance, sfFLOAT, 0,
sfVOID);

	sfAddEvalFn("sfDTAddPoint", sfDTAddPoint, sfVOID, 2, sfFLOAT, sfFLOAT);
	sfAddEvalFn("sfDTAddGlobalPoint", sfDTAddGlobalPoint, sfVOID, 2, sfFLOAT,
sfFLOAT);
	sfAddEvalFn("sfDTRemoveAllPoints", sfDTRemoveAllPoints, sfVOID, 0, sfVOID);
	sfAddEvalFn("sfDTGetNumberOfPoints", sfDTGetNumberOfPoints, sfINT, 0,
sfVOID);
	sfAddEvalFn("sfDTWritePoints", sfDTWritePoints, sfVOID, 0, sfVOID);
	sfAddEvalFn("sfDTWriteGlobalPoints", sfDTWriteGlobalPoints, sfVOID, 0,
sfVOID);

	sfAddEvalFn("sfDTSetVelocity", sfDTSetVelocity, sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfDTGetVelocity", sfDTGetVelocity, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfDTSetPriority", sfDTSetPriority, sfVOID, 1, sfINT);
	sfAddEvalFn("sfDTGetPriority", sfDTGetPriority, sfINT, 0, sfVOID);

	sfAddEvalFn("sfDTSetSpline", sfDTSetSpline, sfVOID, 0, sfVOID);
	sfAddEvalFn("sfDTSetBezier", sfDTSetBezier, sfVOID, 0, sfVOID);
	sfAddEvalFn("sfDTIsSpline", sfDTIsSpline, sfINT, 0, sfVOID);
	sfAddEvalFn("sfDTIsBezier", sfDTIsBezier, sfINT, 0, sfVOID);
	sfAddEvalFn("sfDTSetKnots", sfDTSetKnots, sfVOID, 2, sfINT, sfFLOAT);
```

```
	sfAddEvalFn("sfDTSetKTrajectory", sfDTSetKTrajectory, sfVOID, 1, sfINT);
	sfAddEvalFn("sfDTGetKTrajectory", sfDTGetKTrajectory, sfINT, 0, sfVOID);
	sfAddEvalFn("sfDTSetResolution", sfDTSetResolution, sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfDTGetResolution", sfDTGetResolution, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfDTSetVSlow", sfDTSetVSlow, sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfDTGetVSlow", sfDTGetVSlow, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfDTSetMinDist", sfDTSetMinDist, sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfDTGetMinDist", sfDTGetMinDist, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfDTSetMinAngle", sfDTSetMinAngle, sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfDTGetMinAngle", sfDTGetMinAngle, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfDTSetForward",sfDTSetForward, sfVOID, 0, sfVOID);
	sfAddEvalFn("sfDTSetBackward", sfDTSetBackward, sfVOID, 0, sfVOID);
	sfAddEvalFn("sfDTIsForward", sfDTIsForward, sfINT, 0, sfVOID);
	sfAddEvalFn("sfDTIsBackward", sfDTIsBackward, sfINT, 0, sfVOID);

	sfAddEvalFn("sfDTSTSetPosition", sfDTSTSetPosition, sfVOID, 3, sfFLOAT,
sfFLOAT, sfFLOAT);
	sfAddEvalFn("sfDTSTStart", sfDTSTStart, sfVOID, 0, sfVOID);
	sfAddEvalFn("sfDTSTStop", sfDTSTStop, sfVOID, 0, sfVOID);
	sfAddEvalFn("sfDTSTIsPositionReach", sfDTSTIsPositionReach, sfINT, 0,
sfVOID);
	sfAddEvalFn("sfDTSTSetVelocity", sfDTSTSetVelocity, sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfDTSTSetPriority", sfDTSTSetPriority, sfVOID, 1, sfINT);

	// Inicialitzar variables per defecte

	llista = llCrearLlistaPunts();
	trajectoria = llCrearLlistaPunts();
	knots = CrearVector(1);

	es_spline = 1;
	velocity = 100.0;
	priority = 1;
	ktrajectory = 4;
	resolucio = 50.0;
	v_slow = 0.5;
	min_dist = 100.0;
	min_angle = 10.0;
	sentit = 1;

	// Missatge final

	sfSMessage("Càrrega de sfDoTrajectory.dll finalitzada");
}

EXPORT void
sfLoadExit(void)    // Fitxer que s'avalua al descarregar-se la dll
{
	// Descarregar memòria de variables

	llDestruirLlistaPunts(llista);
	llDestruirLlistaPunts(trajectoria);
	DestruirVector(knots);

	// Missatge de descàrrega

	sfSMessage("Descàrrega de sfDoTrajectory.dll efectuada");
}
```

4.4 beh.c

```
//Starting Behavior/Act Schema Parser
//Defining behavior sfDTFollowTrajectory


 /* for reference pointers */
```

```
static int _sfleft = TURN_LEFT;
static int _sfright = TURN_RIGHT;
static int _sfbackward = DECEL;
static int _sfforward = ACCEL;
static int _sfspeed = ASPEED;



/* Starting code for behavior sfDTFollowTrajectory  */


void sfDTFollowTrajectory_update();
void sfDTFollowTrajectory_setup();

/* global vars for update fn */
static float esquerra;
static float mod__0;
static float molt_esquerra;
static float mod__1;
static float dreta;
static float mod__2;
static float molt_dreta;
static float mod__3;

behavior
_beh_sfDTFollowTrajectory = {"sfDTFollowTrajectory",
    sfDTFollowTrajectory_setup, /* setup function */
    sfDTFollowTrajectory_update, /* body function */
    0, /* number of params */
    {sfINT},
    4, /* number of rules */
    {{"rule__0", &esquerra, &_sfleft, &mod__0},
     {"rule__1", &molt_esquerra, &_sfleft, &mod__1},
     {"rule__2", &dreta, &_sfright, &mod__2},
     {"rule__3", &molt_dreta, &_sfright, &mod__3}
    }
};
behavior *sfDTFollowTrajectory = &_beh_sfDTFollowTrajectory;



void
sfDTFollowTrajectory_setup (beh_params _params) {
}

/* global vars for update fn */
static float final;



void
sfDTFollowTrajectory_update(beh_params _params)
{

              punt_seguiment = BuscarPuntSeguiment();
              final = (float)(punt_seguiment==llNumPunts(trajectoria)-1);

              if (final<0.5)
              {
                     float angle_aux;

                     angle_aux = llAnglePunt(trajectoria,punt_seguiment);
                     if (sentit==-1)
                     {
                            angle_aux = sfAdd2Angle(angle_aux,180.0);
                     }
```

```
                        esquerra = up_straight(angle_aux,0.0,+min_angle);
                        molt_esquerra = up_straight(angle_aux,90.0,90+min_angle);
                        dreta = straight_down(angle_aux,-min_angle,0.0);
                        molt_dreta = straight_down(angle_aux,-90-min_angle,-90.0);
                }
                else
                {
                        PararVelocitat();

                        esquerra = 0.0;
                        dreta = 0.0;
                        molt_esquerra = 0.0;
                        molt_dreta = 0.0;
                }
        /* Begin activity section */
   c_act_v = c_act_t = 0.0;
    c_act_t = 1.0;
    c_goal = final;
    mod__0 = sfVSLOWLY;
    mod__1 = sfMODERATELY;
    mod__2 = sfVSLOWLY;
    mod__3 = sfMODERATELY;
}


/* Ending code for behavior sfDTFollowTrajectory  */



//Defining behavior sfDTParar


/* Starting code for behavior sfDTParar  */


void sfDTParar_update();
void sfDTParar_setup();

/* global vars for update fn */
static float speed__4 = 0.0;
static float ant__5;

behavior
_beh_sfDTParar = {"sfDTParar",
    sfDTParar_setup, /* setup function */
    sfDTParar_update, /* body function */
    0, /* number of params */
    {sfINT},
    1, /* number of rules */
    {{"rule__0", &ant__5, &_sfspeed, &speed__4}
    }
};
behavior *sfDTParar = &_beh_sfDTParar;



void
sfDTParar_setup (beh_params _params) {
}

/* global vars for update fn */
static float vel_zero;



void
```

```
sfDTParar_update(beh_params _params)
{

             vel_zero = (float)(sfTargetVel()==0.0);
             final = (float)(punt_seguiment==llNumPunts(trajectoria)-1);
             if (final>0.5) punt_seguiment = 0;

        /* Begin activity section */
   c_act_v = c_act_t = 0.0;
    c_act_v = final;
    c_goal = f_and(vel_zero,final);
    ant__5 = 1.0;
}


/* Ending code for behavior sfDTParar  */



//Defining behavior sfDTVelocitatConstant


/* Starting code for behavior sfDTVelocitatConstant  */


void sfDTVelocitatConstant_update();
void sfDTVelocitatConstant_setup();

/* global vars for update fn */
static float desviat;
static float vel_parada;
static float molt_desviat;
static float speed__6 = 0.0;
static float velocitat;
static float ant__7;

behavior
_beh_sfDTVelocitatConstant = {"sfDTVelocitatConstant",
    sfDTVelocitatConstant_setup, /* setup function */
    sfDTVelocitatConstant_update, /* body function */
    0, /* number of params */
    {sfINT},
    3, /* number of rules */
    {{"rule__0", &desviat, &_sfspeed, &vel_parada},
     {"rule__1", &molt_desviat, &_sfspeed, &speed__6},
     {"rule__2", &ant__7, &_sfspeed, &velocitat}
    }
};
behavior *sfDTVelocitatConstant = &_beh_sfDTVelocitatConstant;



void
sfDTVelocitatConstant_setup (beh_params _params) {
}

/* global vars for update fn */



void
sfDTVelocitatConstant_update(beh_params _params)
{

             {
                    float angle_aux;

                    angle_aux = llAnglePunt(trajectoria,punt_seguiment);
```

```
                    if (sentit==-1)
                    {
                          angle_aux = sfAdd2Angle(angle_aux,180.0);
                    }

                    velocitat = velocity*sentit;
                    vel_parada = velocity * v_slow;
                    desviat = f_or(up_straight(angle_aux,0.0,+min_angle),
                          straight_down(angle_aux,-min_angle,0.0));
                    molt_desviat = f_or(straight_down(angle_aux,-min_angle,0.0),
                          straight_down(angle_aux,-90-min_angle,-90.0));
              }

        /* Begin activity section */
   c_act_v = c_act_t = 0.0;
    c_act_v = 1.0;
    ant__7 = f_not(f_or(desviat,molt_desviat));
}


/* Ending code for behavior sfDTVelocitatConstant  */



//Ending Behavior/Act Schema Parser
```

5 sfTracking

5.1 behavior.beh

```
float dist_act;
float marge_dist;
float marge_angle;

BeginBehavior sfTracking
	Params
		sfFLOAT c_dist
		sfFLOAT c_velocitat
	Rules
		If (Not emergencia) And ((too_fast Or too_slow) And (Not
massa_angle)) Then Speed c_velocitat
		If (Not emergencia) And ((massa_angle)) Then Speed marge_parada
		If (Not emergencia) And (esquerra) Then Turn Left Very Slowly
		If (Not emergencia) And (dreta) Then Turn Right Very Slowly
		If emergencia And ((too_fast Or too_slow) And (Not e_massa_angle))
Then Speed c_velocitat
		If emergencia And e_massa_angle Then Speed marge_parada
		If emergencia And e_esquerra Then Turn Left Very Slowly
		If emergencia And e_dreta Then Turn Right Very Slowly
		If emergencia And ((e_massa_prop) And (Not e_massa_angle)) Then Turn
Left Very Slowly
		If emergencia And ((e_massa_lluny) And (Not e_massa_angle)) Then
Turn Right Very Slowly
	Init
		distancia_a_mantenir = c_dist;
		velocitat_a_mantenir = c_velocitat;
	Update
		// Inicialització de parèmetres

		marge_dist = dist_sens*c_dist;
		marge_angle = angle_sens*360;
		marge_parada = parada_sens*c_velocitat;

		// Cálcul del contorn

		c_calcul_contorn();

		// Parametres de lògica difusa

		too_fast = up_straight(sfTargetVel(), c_velocitat, c_velocitat+50);
		too_slow = straight_down(sfTargetVel(), c_velocitat-50,
c_velocitat);

		massa_angle = straight_down(angle_cami_control,-60.0,-20.0);
		massa_angle =
f_or(massa_angle,up_straight(angle_cami_control,20.0,60.0));

		esquerra = up_straight(angle_cami_control,0.0,marge_angle);
		dreta = straight_down(angle_cami_control,0.0,-marge_angle);

		// Parametres emergència de lògica difusa

		dist_act = distancia_control;
```

```
            emergencia =
(float)((dist_act>(c_dist*const_dist_emergencia))||(modul(angle_control)>angle_co
ntrol_emergencia)||(llNumPunts(llista)==0));
            emergencia =
f_or(emergencia,straight_down(r_reg,r_min_emergencia,r_max_emergencia));
            //emergencia = 0.0;

            // if (emergencia==1.0) sfSMessage("Emergencia!");

            e_massa_prop = up_straight(c_dist-dist_act,0.0,marge_dist);
            e_massa_lluny = straight_down(c_dist-dist_act,-marge_dist,0.0);

            if (!seguir_dreta)
            {
                  e_massa_lluny = up_straight(c_dist-dist_act,0.0,marge_dist);
                  e_massa_prop = straight_down(c_dist-dist_act,-marge_dist,0.0);
            }

            e_massa_angle = straight_down(angle_control,-60.0,-20.0);
            e_massa_angle =
f_or(e_massa_angle,up_straight(angle_control,20.0,60.0));

            e_esquerra = up_straight(angle_control,0.0,marge_angle);
            e_dreta = straight_down(angle_control,0.0,-marge_angle);

            // Monitoritzar variables

            c_monitoritzar(monitoritzacio);
            c_visualitzar();
      Activity
            Turn 1.0
            Speed 1.0
EndBehavior
```

5.2 sfTracking.h

```
/* ****************** */
/* Fitxer sfTracking.h */
/* ****************** */

// -------------------
// Inclusió de fitxers
// -------------------

#include "sfUtils.h"

// ------------------------------------------------
// Funció principal (ha d'estar en un bloc update)
// ------------------------------------------------

void c_calcul_contorn(void);

// ----------------------------
// Funcions que agafen les dades
// ----------------------------

void c_agafar_dades(void);

// ------------------------------------------------
// Funcions que realitzen un filtrat de les dades
// ------------------------------------------------

// Crida als filtres

void c_aplicar_filtres(void);
```

```
void c_filtre_angle(LlistaPunts l, float alfa_1, float alfa_2, float
max_correcte);
void c_filtre_focus(LlistaPunts l, float dist_focus);
void c_filtre_recta(LlistaPunts l);

// Funcions filtres

int correcte_filtre_angle(float x, float y, float alfa_1, float alfa_2, float
max_correcte);

// ---------------------------------------------------------------
// Funcions que calculen el control (x_reg, y_reg, angle_control)
// ---------------------------------------------------------------

void calcular_cami(void);
void calcular_angle_cami(void);

void c_calcul_control(float x, float y, float *angle, float *dist);

void c_angle_control_y(float b0, float b1, float *angle);
void c_angle_control_x(float b0, float b1, float *angle);
void c_distancia_control(float x, float y, float *dist);

float c_calcularXReg(void);
float c_calcularYReg(void);

// ---------------------------------------------------------------
// Funció de monitorització (veure vars internes des de Colbert)
// ---------------------------------------------------------------

void c_monitoritzar(int opcions);
void c_visualitzar();

// ------------------------------
// Definició de funcions internes
// ------------------------------

// Funcions geomètriques: mòdul (valor absolut) i distancia

float modul(float x);
float distancia(float x, float y);

// Funcions que realitzen una regresió (i càlculs associats)

void c_calcul_mitjanes(LlistaPunts l, float *x_mitja, float *y_mitja);
void c_calcul_variances(LlistaPunts l, float *cov_xy, float *sx_2, float *sy_2);
void c_calcul_recta_regressio(LlistaPunts l, float *x0, float *y0, float *vx,
float *vy, float *r);
void c_calcul_r(LlistaPunts l, float x0, float y0, float vx, float vy, float *r);
void c_calcul_projeccio(float x1, float y1, float x0, float y0, float vx, float
vy, float *xp, float *yp);

// ----------------------
// Definició de constants
// ----------------------

// Constants numèriques

#define PI 3.1416         // Nombre PI
#define TOL_MIN 0.001     // Qualsevol valor menor serà tractat com 0.0

// ------------------------
// Definició de paràmetres
// ------------------------

// Paràmetres del contorn

float distancia_a_mantenir;            // Var que conté la distancia a mantenir
```

```
float velocitat_a_mantenir;            // Var que conté la velocitat a mantenir

int seguir_dreta = 1;                  // Seguiment d'objectes a la dreta = 1, a
l'esquerra = 0
int monitoritzacio = 0;                // Monitorització a aplicar
int visualitzacio = 7;                 // Visualització dels punts a aplicar

// Parametres de control

float angle_sens = (float)0.1;         // Sensibilitat a l'angle
float dist_sens = (float)0.1;          // Sensibilitat a la distancia
float parada_sens = 0.0;               // Maxim parada possible quan angle o
distancia incorrecte
float p_dist_radi_cami = 1000.0; // Distancia d'intersecció amb recta (Calcul
cami)
float p_focus_control = 100.0;         // Distancia del focus control (seguiment
de contorn)

// Paràmetres dels filtres

int filtre_angle_activat = 1;          // Filtre angle activat = 1, desactivat =
0
int filtre_focus_activat = 1;          // Filtre focus activat = 1, desactivat =
0
int filtre_recta_activat = 1;          // Filtre recta activat = 1, desactivat =
0

float p_alfa_1 = 15.0;                       // Angle recta 1 per ser correctes
(Filtre recta)
float p_alfa_2 = 45.0;                       // Angle recta 2 per ser correctes
(Filtre recta)
float p_max_correcte = 3.0;                  // Factor multiplicatiu per
distancia (Filtre recta)
float p_dist_focus = 500.0;                  // Distancia màxima del focus
(Filtre focus)

// Parametres emergència

float angle_control_emergencia = 25.0;
float r_min_emergencia = (float)0.2;
float r_max_emergencia = (float)0.2;
int metode_calcul_r = 1;
float const_dist_emergencia = 1.0;

// -------------------------------------------------------
// Definició de funcions per fixar i veure els parametres
// -------------------------------------------------------

float sfTGetDistanciaAMantenir(void);
void sfTSetDistanciaAMantenir(float d);

float sfTGetVelocitatAMantenir(void);
void sfTSetVelocitatAMantenir(float v);

int sfTIsSeguimentDreta(void);
void sfTSetSeguimentDreta(void);
int sfTIsSeguimentEsquerra(void);
void sfTSetSeguimentEsquerra(void);

int sfTGetMonitoritzacio(void);
void sfTSetMonitoritzacio(int m);
int sfTGetVisualitzacio(void);
void sfTSetVisualitzacio(int v);

float sfTGetAngleSensibilitat(void);
void sfTSetAngleSensibilitat(float s);
float sfTGetDistanciaSensibilitat(void);
void sfTSetDistanciaSensibilitat(float s);
```

```
float sfTGetParadaSensibilitat(void);
void sfTSetParadaSensibilitat(float s);
float sfTGetDistanciaRadiControl(void);
void sfTSetDistanciaRadiControl(float s);
float sfTGetDistanciaFocusControl(void);
void sfTSetDistanciaFocusControl(float d);

int sfTIsFiltreAngleActivat(void);
int sfTIsFiltreFocusActivat(void);
int sfTIsFiltreRectaActivat(void);

void sfTSetFiltreAngleActivat(void);
void sfTSetFiltreFocusActivat(void);
void sfTSetFiltreRectaActivat(void);

void sfTSetFiltreAngleDesactivat(void);
void sfTSetFiltreFocusDesactivat(void);
void sfTSetFiltreRectaDesactivat(void);

float sfTGetPAlfa1(void);
void sfTSetPAlfa1(float p);
float sfTGetPAlfa2(void);
void sfTSetPAlfa2(float p);
float sfTGetPMaxCorrecte(void);
void sfTSetPMaxCorrecte(float p);
float sfTGetPDistFocus(void);
void sfTSetPDistFocus(float p);
float sfTGetPDistPunts(void);
void sfTSetPDistPunts(float p);

float sfTGetAngleControlEmergencia(void);
float sfTGetRMinEmergencia(void);
float sfTGetRMaxEmergencia(void);
void sfTSetAngleControlEmergencia(float v);
void sfTSetREmergencia(float min, float max);
int sfTGetMetodeCalculR(void);
void sfTSetMetodeCalculR(int mr);
float sfTGetConstDistEmergencia();
void sfTSetConstDistEmergencia(float c);

// ----------------------
// Definició de variables
// ----------------------

// Llista de dades (x,y)

LlistaPunts cami;                       // Llista de punts que marca el camí a
seguir
LlistaPunts llista;                     // Llista de punts amb les dades originals
(i filtrades)
LlistaPunts llista_emergencia;   // Llista de punts per seguiment d'objectes o per
emergències

// Variables de control

float angle_control;             // Variable per controlar l'angle des de lògica
difusa
float distancia_control;  // Variable per controlar la distancia des de lògica
difusa

float angle_cami_control; // Variable que determina l'angle del punt del camí a
seguir

float r_reg;
```

5.3 sfTracking.c

```
/* ****************** */
/* Fitxer sfTracking.c */
/* ****************** */

#include "saphira.h"
#include "sonar.h"
#include "sfTracking.h"
#include <assert.h>

// ----------------
// Funció principal
// ----------------

void c_calcul_contorn(void)
{
	c_agafar_dades();
	c_aplicar_filtres();
	calcular_angle_cami();
	c_calcul_control(c_calcularXReg(),c_calcularYReg(),&angle_control,&distanci
a_control);
}

// -----------------------------
// Funcions que agafen les dades
// -----------------------------

void c_agafar_dades(void)
{
	int i;

	s_actualitzar();                                        //S'actualitza
el sonar

	llEliminarTotsPunts(cami);                          // Esborrem la llista
cami
	llEliminarTotsPunts(llista);                        // Esborrem la llista
llista
	llEliminarTotsPunts(llista_emergencia);       // Esborrem la llista
llista_emergencia

	// Insertem els punts del sonar

	i = 1;

	while (i<=s_elements())
	{
		llAfegirPunt(llista,s_x(i),s_y(i),0.0);
		llAfegirPunt(llista_emergencia,s_x(i),s_y(i),0.0);

		i++;
	}
}

// -------------------------------------------
// Funcions que realitzen un filtrat de les dades
// -------------------------------------------

void c_aplicar_filtres(void)
{
	if (filtre_angle_activat)
	{
		c_filtre_angle(llista,p_alfa_1,p_alfa_2,p_max_correcte);
		if (llNumPunts(llista)>0)
c_filtre_angle(llista_emergencia,p_alfa_1,p_alfa_2,p_max_correcte);
	}
```

```
	if (filtre_focus_activat)
	{
		c_filtre_focus(llista,p_dist_focus);
		if (llNumPunts(llista)>0)
c_filtre_focus(llista_emergencia,p_focus_control);
	}

	if (filtre_recta_activat)
	{
		c_filtre_recta(llista);
	}
}

void c_filtre_angle(LlistaPunts l, float alfa_1, float alfa_2, float
max_correcte)
{
	int i;

	i = 0;
	while (i<llNumPunts(l))
	{
		if
(!correcte_filtre_angle(llObtenirXPunt(l,i),llObtenirYPunt(l,i),alfa_1,alfa_2,max
_correcte))
		{
			llEliminarPunt(l,i);
			continue;
		}

		i++;
	}
}

int correcte_filtre_angle(float x, float y, float alfa_1, float alfa_2, float
max_correcte)
{
	int resultat;
	float dist_max;

	dist_max = distancia_a_mantenir*max_correcte;

	resultat = 1;

	if (!seguir_dreta)
	{
		alfa_1 = -alfa_1;
		alfa_2 = -alfa_2;

		resultat = resultat&&((y-x*tan(alfa_1*2*PI/360))>=0);
		resultat = resultat&&((y-x*tan(alfa_2*2*PI/360))>=0);
	}
	else
	{
		alfa_1 = alfa_1;
		alfa_2 = alfa_2;

		resultat = resultat&&((y-x*tan(alfa_1*2*PI/360))<=0);
		resultat = resultat&&((y-x*tan(alfa_2*2*PI/360))<=0);
	}

	resultat = resultat&&(distancia(x,y)<=dist_max);

	return resultat;
}

void c_filtre_focus(LlistaPunts l, float dist_focus)
{
```

```
	int i, i_min;
	float x_min, y_min;

	if (llNumPunts(l)<1) return;

	i_min = MinimaDistanciaLlistaPunts(l);
	x_min = llObtenirXPunt(l,i_min);
	y_min = llObtenirYPunt(l,i_min);

	i = 0;
	while (i<llNumPunts(l))
	{
		if (!(distancia(x_min-llObtenirXPunt(l,i),y_min-
llObtenirYPunt(l,i))<=dist_focus))
		{
			llEliminarPunt(l,i);
			continue;
		}

		i++;
	}
}

void c_filtre_recta(LlistaPunts l)
{
	// Posa es punts sobre una recta

	float x0,y0,vx,vy;
	float x_final,y_final;
	int i;

	if (llNumPunts(l)<2) return;

	c_calcul_recta_regressio(l,&x0,&y0,&vx,&vy,&r_reg);
	c_calcul_r(l,x0,y0,vx,vy,&r_reg);

	i = 0;
	while (i<llNumPunts(l))
	{

	c_calcul_projeccio(llObtenirXPunt(l,i),llObtenirYPunt(l,i),x0,y0,vx,vy,&x_f
inal,&y_final);
		llCambiarPunt(l,i,x_final,y_final,0.0);

		i++;
	}
}

// ---------------------------------------------------------------
// Funcions que calculen el cami control (distancia, angle_control)
// ---------------------------------------------------------------

void calcular_angle_cami(void)
{
	float x_vect, y_vect, x0, y0, x0antic, y0antic, r;
	double a,b,c, discriminant;
	double z1,z2,z;
	float xf,yf;

	angle_cami_control = 0.0; // Es posa a 0 per si no es pot calcular
	if (llNumPunts(llista)<2) return;

	// Calculem els vectors

	if (filtre_recta_activat)
c_calcul_recta_regressio(llista,&x0,&y0,&x_vect,&y_vect,&r);
	else
	{
```

```
	c_calcul_recta_regressio(llista,&x0,&y0,&x_vect,&y_vect,&r_reg);
	c_calcul_r(llista,x0,y0,x_vect,y_vect,&r_reg);
}

x0antic = x0;
y0antic = y0;

// Calculem la recta a distància adequada

{
	float x_perp, y_perp;

	x_perp = y_vect;
	y_perp = -x_vect;

	if ((x0*x_perp+y0*y_perp)>0)
	{
		x_perp = -y_vect;
		y_perp = x_vect;
	}

	x0 = x0 + x_perp*distancia_a_mantenir;
	y0 = y0 + y_perp*distancia_a_mantenir;
}

// Calculem els paràmetres de la equació

a = x_vect*x_vect + y_vect*y_vect;
b = 2 * (x_vect*x0 + y_vect*y0);
c = x0*x0 + y0*y0 - p_dist_radi_cami*p_dist_radi_cami;
discriminant = b*b - 4*a*c;

if (a<TOL_MIN) return;

// Eliminem tots els punts del cami

llEliminarTotsPunts(cami);

// Comprovem tipus d'equació

if (discriminant>TOL_MIN)
{
	z1 = (-b+sqrt(discriminant))/(2*a);
	z2 = (-b-sqrt(discriminant))/(2*a);

	xf = (float)(x0+z1*x_vect);
	yf = (float)(y0+z1*y_vect);
	llAfegirPunt(cami,xf,yf,0.0);

	xf = (float)(x0+z2*x_vect);
	yf = (float)(y0+z2*y_vect);
	llAfegirPunt(cami,xf,yf,0.0);
}
else if (discriminant<TOL_MIN)
{
	z1 = (y_vect*x0 - x_vect*y0)/a;

	xf = (float)(z1*y_vect);
	yf = (float)(-z1*x_vect);

	llAfegirPunt(cami,xf,yf,0.0);
}
else
{
	z = -b/(2*a);

	xf = (float)(x0+z*x_vect);
	yf = (float)(y0+z*y_vect);
```

```
                llAfegirPunt(cami,xf,yf,0.0);
        }

        // S'ha de calcular quin és el punt a seguir

        if (llNumPunts(cami)==1)
        {
                angle_cami_control = llAnglePunt(cami,0);
        }
        else if (llNumPunts(cami)==2)
        {
                float x1,y1,x2,y2,xr,yr,z;

                x1 = llObtenirXPunt(cami,0);
                y1 = llObtenirYPunt(cami,0);

                x2 = llObtenirXPunt(cami,1);
                y2 = llObtenirYPunt(cami,1);

                c_calcul_projeccio(0.0,0.0,x0antic,y0antic,x_vect,y_vect,&xr,&yr);

                z = (xr*y1)-(x1*yr);

                //sfSMessage("z = %f",z);

                if (seguir_dreta)
                {
                        if (z>=0) angle_cami_control = llAnglePunt(cami,0);
                        else angle_cami_control = llAnglePunt(cami,1);
                }
                else
                {
                        if (z>=0) angle_cami_control = llAnglePunt(cami,1);
                        else angle_cami_control = llAnglePunt(cami,0);
                }
        }
}

// ------------------------------------------------------------
// Funcions que calculen el control (distancia, angle_control)
// ------------------------------------------------------------

void c_calcul_control(float x, float y, float *angle, float *dist)
{
        float x_reg, y_reg, b0, b1;

        x_reg = x;
        y_reg = y;

        // Càlcul de les variables de control

        if (modul(y_reg)>=modul(x_reg))
        {
                b1 = x_reg/(-y_reg);
                b0 = y_reg + (x_reg*x_reg/y_reg);
                c_angle_control_y(b0, b1, angle);
                c_distancia_control(x_reg,y_reg, dist);
        }
        else
        {
                b1 = -y_reg/x_reg;
                b0 = x_reg + (y_reg*y_reg/x_reg);
                c_angle_control_x(b0, b1,angle);
                c_distancia_control(x_reg,y_reg,dist);
        }
}
```

```
void c_angle_control_y(float b0, float b1, float *angle)
{
	if (seguir_dreta)
	{
		if ((b0<=0.0)&&(b1>=0.0)) (*angle) = (float)atan(b1);
		if ((b0<=0.0)&&(b1<=0.0)) (*angle) = (float)(-atan(-b1));
		if ((b0>=0.0)&&(b1<=0.0)) (*angle) = (float)(PI-atan(-b1));
		if ((b0>=0.0)&&(b1>=0.0)) (*angle) = (float)(-PI+atan(b1));
	}
	else
	{
		if ((b0>=0.0)&&(b1>=0.0)) (*angle) = (float)atan(b1);
		if ((b0>=0.0)&&(b1<=0.0)) (*angle) = (float)(-atan(-b1));
		if ((b0<=0.0)&&(b1<=0.0)) (*angle) = (float)(PI-atan(-b1));
		if ((b0<=0.0)&&(b1>=0.0)) (*angle) = (float)(-PI+atan(b1));
	}


	(*angle) = (float)((*angle) * 180.0 / PI);
}

void c_angle_control_x(float b0, float b1, float *angle)
{
	if (seguir_dreta)
	{
		if ((b0>=0.0)&&(b1>=0.0)) (*angle) = (float)(PI/2-atan(b1));
		if ((b0<=0.0)&&(b1<=0.0)) (*angle) = (float)(-PI/2+atan(-b1));
		if ((b0>=0.0)&&(b1<=0.0)) (*angle) = (float)(PI/2 + atan(-b1));
		if ((b0<=0.0)&&(b1>=0.0)) (*angle) = (float)(-PI/2 - atan(b1));
	}
	else
	{
		if ((b0<=0.0)&&(b1>=0.0)) (*angle) = (float)(PI/2-atan(b1));
		if ((b0>=0.0)&&(b1<=0.0)) (*angle) = (float)(-PI/2+atan(-b1));
		if ((b0<=0.0)&&(b1<=0.0)) (*angle) = (float)(PI/2 + atan(-b1));
		if ((b0>=0.0)&&(b1>=0.0)) (*angle) = (float)(-PI/2 - atan(b1));
	}

	(*angle) = (float)((*angle) * 180.0 / PI);
}

void c_distancia_control(float x, float y, float *dist)
{
	(*dist) = distancia(x,y);
}

float c_calcularXReg(void)
{
	if (llNumPunts(llista_emergencia)==0) return 0.0;
	else
	{
		float x,y;

		c_calcul_mitjanes(llista_emergencia,&x,&y);
		return x;
	}
}

float c_calcularYReg(void)
{
	if (llNumPunts(llista_emergencia)==0)
	{
		if (!seguir_dreta) return distancia_a_mantenir;
		else return -distancia_a_mantenir;
	}
	else
	{
		float x,y;
```

```
            c_calcul_mitjanes(llista_emergencia,&x,&y);
            return y;
      }
}

// --------------------
// Funcions del Colbert
// --------------------

void c_monitoritzar(int opcions)
{
      int i;
      static int interval = 0;

      interval++;
      if (interval!=10) return; // Monitoritzem cada 10*100ms
      else interval = 0;

      if (opcions)
      {
            sfSMessage("");
            sfSMessage("MONITORITZACIÓ");
            sfSMessage("--------------");
      }

      if (opcions&1)
      {
            sfSMessage("");
            sfSMessage("Variables de control:");
            sfSMessage("  Angle control: %f Distancia control: %f"
                  , angle_control, distancia_control);
      }

      if (opcions&2)
      {
            sfSMessage("");
            sfSMessage("Buffer correcte de dades:");
            llEscriureLlistaPunts(llista);
      }

      if (opcions&4)
      {
            sfSMessage("");
            sfSMessage("Dades del sonar:");
            i = 1;
            while (i<=s_elements())
            {
                  sfSMessage("  Punt %d: (%f , %f), dist: %f",i,s_x(i),s_y(i),
                        distancia(s_x(i),s_y(i)));

                  i++;
            }
      }

      if (opcions&8)
      {
            sfSMessage("");
            sfSMessage("Valor r_reg: %f",r_reg);
      }
}

void c_visualitzar()
{
      if (visualitzacio&1) llEliminarTotsPunts(llista_emergencia);
      if (visualitzacio&2) llEliminarTotsPunts(llista);
      if (visualitzacio&4) llEliminarTotsPunts(cami);
}
```

```
// -----------------
// Funcions internes
// -----------------

// Funcions geomètriques: mòdul (valor absolut) i distancia

float modul(float x)
{
	if (x<0.0) return -x;
	else return x;
}

float distancia(float x, float y)
{
	return (float)(sqrt(x*x+y*y));
}

// Funcions que realitzen una regresió (i càlculs associats)

void c_calcul_mitjanes(LlistaPunts l, float *x_mitja, float *y_mitja)
{
	int i;

	(*x_mitja) = 0.0;
	(*y_mitja) = 0.0;

	if (llNumPunts(l)<1) return; // Comprovem que es pot fer una mitjana

	i = 0;
	while (i<llNumPunts(l))
	{
		(*x_mitja) += llObtenirXPunt(l,i);
		(*y_mitja) += llObtenirYPunt(l,i);

		i++;
	}

	(*x_mitja) /= llNumPunts(l);
	(*y_mitja) /= llNumPunts(l);
}

void c_calcul_variances(LlistaPunts l, float *cov_xy, float *sx_2, float *sy_2)
{
	float x_mitja, y_mitja;
	float resta_x, resta_y;
	int i;

	(*cov_xy) = 0.0;
	(*sx_2) = 0.0;
	(*sy_2) = 0.0;

	if(llNumPunts(l)<2) return; // Comprovem que es poden calcular variances

	c_calcul_mitjanes(l,&x_mitja,&y_mitja);

	i = 0;
	while (i<llNumPunts(l))
	{
		resta_x = llObtenirXPunt(l,i) - x_mitja;
		resta_y = llObtenirYPunt(l,i) - y_mitja;

		(*cov_xy) += resta_x * resta_y;
		(*sx_2) += resta_x * resta_x;
		(*sy_2) += resta_y * resta_y;

		i++;
	}
```

```
	(*cov_xy) /= llNumPunts(l)-1;
	(*sx_2) /= llNumPunts(l)-1;
	(*sy_2) /= llNumPunts(l)-1;
}

void c_calcul_recta_regressio(LlistaPunts l, float *x0, float *y0, float *vx,
float *vy, float *r)
{
	float cov_xy, sx_2, sy_2;
	float x_mitja, y_mitja;
	float vap_gran;
	float modul;

	assert(llNumPunts(l)>=2);

	c_calcul_variances(l,&cov_xy,&sx_2,&sy_2);
	c_calcul_mitjanes(l,&x_mitja,&y_mitja);

	vap_gran = (sx_2 + sy_2 + (float)sqrt(((sx_2-sy_2)*(sx_2-
sy_2))+(4*(cov_xy*cov_xy))))/2;

	if (sx_2>sy_2)
	{
		if ((sx_2-vap_gran)>cov_xy)
		{
			(*vx) = -cov_xy/(sx_2-vap_gran);
			(*vy) = 1.0;
		}
		else
		{
			(*vx) = 1.0;
			(*vy) = -(sx_2-vap_gran)/cov_xy;
		}
	}
	else
	{
		if ((sy_2-vap_gran)>cov_xy)
		{
			(*vx) = 1.0;
			(*vy) = -cov_xy/(sy_2-vap_gran);
		}
		else
		{
			(*vx) = -(sy_2-vap_gran)/cov_xy;
			(*vy) = 1.0;
		}
	}

	modul =  distancia((*vx),(*vy));

	(*vx) = (*vx)/modul;
	(*vy) = (*vy)/modul;

	(*x0) = x_mitja;
	(*y0) = y_mitja;
	(*r) = (float)sqrt((cov_xy*cov_xy)/(sx_2*sy_2));
}

void c_calcul_projeccio(float x1, float y1, float x0, float y0, float vx, float
vy, float *xp, float *yp)
{
	float t;

	// Es suposa que |(vx,vy)|=1

	t = (vx*(x1-x0)+vy*(y1-y0))/(vx*vx+vy*vy);
	(*xp) = x0 + t*vx;
```

```
	(*yp) = y0 + t*vy;
}

void c_calcul_r(LlistaPunts l, float x0, float y0, float vx, float vy, float *r)
{
	int i,j;
	float mitja,xp,yp,x,y,max;

	if (llNumPunts(l)<2)
	{
		(*r) = 0.0;
		return;
	}

	mitja = 0;

	i = 0;
	while (i<llNumPunts(l))
	{
		x = llObtenirXPunt(l,i);
		y = llObtenirYPunt(l,i);

		c_calcul_projeccio(x,y,x0,y0,vx,vy,&xp,&yp);
		mitja += distancia(xp-x,yp-y);

		i++;
	}

	mitja /= llNumPunts(l);

	max = 1.0; // Equivalent a max = 0.0;

	i = 0;
	while (i<llNumPunts(l))
	{
		j = 0;
		while (j<llNumPunts(l))
		{
			float dist;

			dist = distancia(llObtenirXPunt(l,i)-llObtenirXPunt(l,j),
				llObtenirYPunt(l,i)-llObtenirYPunt(l,j));

			if (dist>max) max = dist;

			j++;
		}

		i++;
	}

	// 5 Mètodes diferents de calcul r (per defecte 1)

	if (metode_calcul_r==1)
	{
		(*r) = 1 - (mitja*llNumPunts(l)/max);
	}
	else if (metode_calcul_r==2)
	{
		(*r) = 1 - (mitja/max);
	}
	else if (metode_calcul_r==3)
	{
		(*r) = 1 - (mitja*llNumPunts(l)/distancia_a_mantenir);
	}
	else if (metode_calcul_r==4)
	{
		(*r) = 1 - (mitja/distancia_a_mantenir);
```

```
	}
	else if (metode_calcul_r==5)
	{
		float x0,y0,vx,vy;
		c_calcul_recta_regressio(l,&x0,&y0,&vx,&vy,r);
	}
	else
	{
		(*r) = 1 - (mitja*llNumPunts(l)/max);
	}
}

// ---------------------------------------------
// Funcions de lectura i escriptura de paràmetres
// ---------------------------------------------

float sfTGetDistanciaAMantenir(void)
{
	return distancia_a_mantenir;
}

void sfTSetDistanciaAMantenir(float d)
{
	if ((d>0.0)&&(d<4000.0)) distancia_a_mantenir = d;
	else sfSMessage("No es pot seguir una distancia de %f",d);
}

float sfTGetVelocitatAMantenir(void)
{
	return velocitat_a_mantenir;
}

void sfTSetVelocitatAMantenir(float v)
{
	if ((v<=400.0)&&(v>0.0)) velocitat_a_mantenir = v;
	else sfSMessage("No es pot seguir una velocitat de %f",v);
}

int sfTIsSeguimentDreta(void)
{
	return seguir_dreta;
}

void sfTSetSeguimentDreta(void)
{
	seguir_dreta = 1;
}

int sfTIsSeguimentEsquerra(void)
{
	return !seguir_dreta;
}

void sfTSetSeguimentEsquerra(void)
{
	seguir_dreta = 0;
}

int sfTGetMonitoritzacio(void)
{
	return monitoritzacio;
}

void sfTSetMonitoritzacio(int m)
{
	monitoritzacio = m;
}
```

```
int sfTGetVisualitzacio(void)
{
	return visualitzacio;
}

void sfTSetVisualitzacio(int v)
{
	visualitzacio = v;
}

float sfTGetAngleSensibilitat(void)
{
	return angle_sens;
}

void sfTSetAngleSensibilitat(float s)
{
	if ((s>=0)&&(s<=1)) angle_sens = s;
	else sfSMessage("La sensibilitat a l'angle ha d'estar entre 0 i 1");
}

float sfTGetDistanciaSensibilitat(void)
{
	return dist_sens;
}

void sfTSetDistanciaSensibilitat(float s)
{
	if ((s>=0)&&(s<=1)) dist_sens = s;
	else sfSMessage("La sensibilitat a la distancia ha d'estar entre 0 i 1");
}

float sfTGetParadaSensibilitat(void)
{
	return parada_sens;
}

void sfTSetParadaSensibilitat(float s)
{
	if ((s>=0)&&(s<=1)) parada_sens = s;
	else sfSMessage("La sensibilitat a la parada ha d'estar entre 0 i 1");
}

float sfTGetDistanciaRadiControl(void)
{
	return p_dist_radi_cami;
}

void sfTSetDistanciaRadiControl(float s)
{
	if ((s>0)&&(s<=10000.0)) p_dist_radi_cami = s;
	else sfSMessage("La Distancia del radi de control ha d'estar entre 0 i
10000.0");

}

float sfTGetDistanciaFocusControl(void)
{
	return p_focus_control;
}

void sfTSetDistanciaFocusControl(float d)
{
	if (d<=0.0)
	{
		sfSMessage("Distància del focus de control ha de ser positiva");
		return;
	}
```

```
	p_focus_control = d;
}

int sfTIsFiltreAngleActivat(void)
{
	return filtre_angle_activat;
}

int sfTIsFiltreFocusActivat(void)
{
	return filtre_focus_activat;
}

int sfTIsFiltreRectaActivat(void)
{
	return filtre_recta_activat;
}

void sfTSetFiltreAngleActivat(void)
{
	filtre_angle_activat = 1;
}

void sfTSetFiltreFocusActivat(void)
{
	filtre_focus_activat = 1;
}

void sfTSetFiltreRectaActivat(void)
{
	filtre_recta_activat = 1;
}

void sfTSetFiltreAngleDesactivat(void)
{
	filtre_angle_activat = 0;
}

void sfTSetFiltreFocusDesactivat(void)
{
	filtre_focus_activat = 0;
}

void sfTSetFiltreRectaDesactivat(void)
{
	filtre_recta_activat = 0;
}

float sfTGetPAlfa1(void)
{
	return p_alfa_1;
}

void sfTSetPAlfa1(float p)
{
	p_alfa_1 = p;
}

float sfTGetPAlfa2(void)
{
	return p_alfa_2;
}

void sfTSetPAlfa2(float p)
{
	p_alfa_2 = p;
}
```

```
float sfTGetPMaxCorrecte(void)
{
	return p_max_correcte;
}

void sfTSetPMaxCorrecte(float p)
{
	if (p>1.0) p_max_correcte = p;
	else sfSMessage("PMaxCorrecte ha de ser major que 1");
}

float sfTGetPDistFocus(void)
{
	return p_dist_focus;
}

void sfTSetPDistFocus(float p)
{
	if (p>0.0) p_dist_focus = p;
	else sfSMessage("PDistFocus ha de tenir un valor positiu");
}

float sfTGetAngleControlEmergencia(void)
{
	return angle_control_emergencia;
}

float sfTGetRMinEmergencia(void)
{
	return r_min_emergencia;
}

float sfTGetRMaxEmergencia(void)
{
	return r_max_emergencia;
}

void sfTSetAngleControlEmergencia(float v)
{
	if (v<0.0)
	{
		sfSMessage("Angle emergencia ha de ser major que 0");
		return;
	}

	angle_control_emergencia = v;
}

void sfTSetREmergencia(float min, float max)
{
	if ((min>max)||(max>1.0))
	{
		sfSMessage("Restrccions de R Emergencia:");
		sfSMessage("  min<=max, max<=1.0");
		return;
	}

	r_min_emergencia = min;
	r_max_emergencia = max;
}

int sfTGetMetodeCalculR(void)
{
	return metode_calcul_r;
}

void sfTSetMetodeCalculR(int mr)
```

```
{
	if ((mr<1)||(mr>5))
	{
		sfSMessage("Métode de calcul R incorrecte. Ha d'estar entre 1 i 5");
	}

	metode_calcul_r = mr;
}

float sfTGetConstDistEmergencia()
{
	return const_dist_emergencia;
}

void sfTSetConstDistEmergencia(float c)
{
	if (c<0)
	{
		sfSMessage("const_dist_emergencia ha de ser un valor positiu");
		return;
	}

	const_dist_emergencia = c;
}

// -------------
// Comportaments
// -------------

#include "beh.c"

// ----------------------------------------
// Fitxers de càrrega i descàrrega de la dll
// ----------------------------------------

EXPORT void
sfLoadInit(void)    // Fitxer que s'avalua al carregar-se la dll
{
	// Missatge d'inici

	sfSMessage("Iniciant càrrega trackingObject.dll");

	// Càrrega d'esquemes

	sfRegisterSchema("sfTracking",sfBEHAVIOR,sfTracking);

	// Càrrega de funcions

	sfAddEvalFn("sfTGetDistanciaAMantenir", sfTGetDistanciaAMantenir, sfFLOAT,
0, sfVOID);
	sfAddEvalFn("sfTSetDistanciaAMantenir", sfTSetDistanciaAMantenir, sfVOID,
1, sfFLOAT);
	sfAddEvalFn("sfTGetVelocitatAMantenir", sfTGetVelocitatAMantenir, sfFLOAT,
0, sfVOID);
	sfAddEvalFn("sfTSetVelocitatAMantenir", sfTSetVelocitatAMantenir, sfVOID,
1, sfFLOAT);

	sfAddEvalFn("sfTIsSeguimentDreta", sfTIsSeguimentDreta, sfINT, 0, sfVOID);
	sfAddEvalFn("sfTSetSeguimentDreta", sfTSetSeguimentDreta, sfVOID, 0,
sfVOID);
	sfAddEvalFn("sfTIsSeguimentEsquerra", sfTIsSeguimentEsquerra, sfINT, 0,
sfVOID);
	sfAddEvalFn("sfTSetSeguimentEsquerra", sfTSetSeguimentEsquerra, sfVOID, 0,
sfVOID);

	sfAddEvalFn("sfTGetMonitoritzacio", sfTGetMonitoritzacio, sfINT, 0,
sfVOID);
```

```
	sfAddEvalFn("sfTSetMonitoritzacio", sfTSetMonitoritzacio, sfVOID, 1,
sfINT);
	sfAddEvalFn("sfTGetVisualitzacio", sfTGetVisualitzacio, sfINT, 0, sfVOID);
	sfAddEvalFn("sfTSetVisualitzacio", sfTSetVisualitzacio, sfVOID, 1, sfINT);

	sfAddEvalFn("sfTGetAngleSensibilitat", sfTGetAngleSensibilitat, sfFLOAT, 0,
sfVOID);
	sfAddEvalFn("sfTSetAngleSensibilitat", sfTSetAngleSensibilitat, sfVOID, 1,
sfFLOAT);
	sfAddEvalFn("sfTGetDistanciaSensibilitat", sfTGetDistanciaSensibilitat,
sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfTSetDistanciaSensibilitat", sfTSetDistanciaSensibilitat,
sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfTGetParadaSensibilitat", sfTGetParadaSensibilitat, sfFLOAT,
0, sfVOID);
	sfAddEvalFn("sfTSetParadaSensibilitat", sfTSetParadaSensibilitat, sfVOID,
1, sfFLOAT);

	sfAddEvalFn("sfTGetDistanciaRadiControl", sfTGetDistanciaRadiControl,
sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfTSetDistanciaRadiControl", sfTSetDistanciaRadiControl,
sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfTGetDistanciaFocusControl", sfTGetDistanciaFocusControl,
sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfTSetDistanciaFocusControl", sfTSetDistanciaFocusControl,
sfVOID, 1, sfFLOAT);

	sfAddEvalFn("sfTIsFiltreAngleActivat", sfTIsFiltreAngleActivat, sfINT, 0,
sfVOID);
	sfAddEvalFn("sfTIsFiltreFocusActivat", sfTIsFiltreFocusActivat, sfINT, 0,
sfVOID);
	sfAddEvalFn("sfTIsFiltreRectaActivat", sfTIsFiltreRectaActivat, sfINT, 0,
sfVOID);

	sfAddEvalFn("sfTSetFiltreAngleActivat", sfTSetFiltreAngleActivat, sfVOID,
0, sfVOID);
	sfAddEvalFn("sfTSetFiltreFocusActivat", sfTSetFiltreFocusActivat, sfVOID,
0, sfVOID);
	sfAddEvalFn("sfTSetFiltreRectaActivat", sfTSetFiltreRectaActivat, sfVOID,
0, sfVOID);

	sfAddEvalFn("sfTSetFiltreAngleDesactivat", sfTSetFiltreAngleDesactivat,
sfVOID, 0, sfVOID);
	sfAddEvalFn("sfTSetFiltreFocusDesactivat", sfTSetFiltreFocusDesactivat,
sfVOID, 0, sfVOID);
	sfAddEvalFn("sfTSetFiltreRectaDesactivat", sfTSetFiltreRectaDesactivat,
sfVOID, 0, sfVOID);

	sfAddEvalFn("sfTGetPAlfa1", sfTGetPAlfa1, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfTSetPAlfa1", sfTSetPAlfa1, sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfTGetPAlfa2", sfTGetPAlfa2, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfTSetPAlfa2", sfTSetPAlfa2, sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfTGetPMaxCorrecte", sfTGetPMaxCorrecte, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfTSetPMaxCorrecte", sfTSetPMaxCorrecte, sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfTGetPDistFocus", sfTGetPDistFocus, sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfTSetPDistFocus", sfTSetPDistFocus, sfVOID, 1, sfFLOAT);

	sfAddEvalFn("sfTGetAngleControlEmergencia", sfTGetAngleControlEmergencia,
sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfTGetRMinEmergencia", sfTGetRMinEmergencia, sfFLOAT, 0,
sfVOID);
	sfAddEvalFn("sfTGetRMaxEmergencia", sfTGetRMaxEmergencia, sfFLOAT, 0,
sfVOID);
	sfAddEvalFn("sfTSetAngleControlEmergencia", sfTSetAngleControlEmergencia,
sfVOID, 1, sfFLOAT);
	sfAddEvalFn("sfTSetREmergencia", sfTSetREmergencia, sfVOID, 2, sfFLOAT,
sfFLOAT);
```

```
	sfAddEvalFn("sfTGetMetodeCalculR", sfTGetMetodeCalculR, sfINT, 0, sfVOID);
	sfAddEvalFn("sfTSetMetodeCalculR", sfTSetMetodeCalculR, sfVOID, 1, sfINT);

	sfAddEvalFn("sfTGetConstDistEmergencia", sfTGetConstDistEmergencia,
sfFLOAT, 0, sfVOID);
	sfAddEvalFn("sfTSetConstDistEmergencia", sfTSetConstDistEmergencia, sfVOID,
1, sfFLOAT);

	// Inicialització de variables

	llista = llCrearLlistaPunts();
	cami = llCrearLlistaPunts();
	llista_emergencia = llCrearLlistaPunts();

	// Missatge final

	sfSMessage("Càrrega de trackingObject.dll finalitzada");
}

EXPORT void
sfLoadExit(void)    // Fitxer que s'avalua al descarregar-se la dll
{
	// Descàrrega de variables

	llDestruirLlistaPunts(llista);
	llDestruirLlistaPunts(cami);
	llDestruirLlistaPunts(llista_emergencia);

	// Missatge de descàrrega

	sfSMessage("Descàrrega de trackingObject.dll efectuada");
}
```

5.4 beh.c

```
//Starting Behavior/Act Schema Parser

float dist_act;
float marge_dist;
float marge_angle;

//Defining behavior sfTracking


 /* for reference pointers */

static int _sfleft = TURN_LEFT;
static int _sfright = TURN_RIGHT;
static int _sfbackward = DECEL;
static int _sfforward = ACCEL;
static int _sfspeed = ASPEED;




/* Starting code for behavior sfTracking  */


void sfTracking_update();
void sfTracking_setup();

/* global vars for update fn */
static float emergencia;
static float too_fast;
static float too_slow;
static float massa_angle;
```

```
static float g__0;
static float ant__1;
static float marge_parada;
static float ant__2;
static float esquerra;
static float mod__3;
static float ant__4;
static float dreta;
static float mod__5;
static float ant__6;
static float e_massa_angle;
static float g__7;
static float ant__8;
static float ant__9;
static float e_esquerra;
static float mod__10;
static float ant__11;
static float e_dreta;
static float mod__12;
static float ant__13;
static float e_massa_prop;
static float mod__14;
static float ant__15;
static float e_massa_lluny;
static float mod__16;
static float ant__17;

behavior
_beh_sfTracking = {"sfTracking",
    sfTracking_setup, /* setup function */
    sfTracking_update, /* body function */
    2, /* number of params */
    {sfFLOAT, sfFLOAT},
    10, /* number of rules */
    {{"rule__0", &ant__1, &_sfspeed, &g__0},
     {"rule__1", &ant__2, &_sfspeed, &marge_parada},
     {"rule__2", &ant__4, &_sfleft, &mod__3},
     {"rule__3", &ant__6, &_sfright, &mod__5},
     {"rule__4", &ant__8, &_sfspeed, &g__7},
     {"rule__5", &ant__9, &_sfspeed, &marge_parada},
     {"rule__6", &ant__11, &_sfleft, &mod__10},
     {"rule__7", &ant__13, &_sfright, &mod__12},
     {"rule__8", &ant__15, &_sfleft, &mod__14},
     {"rule__9", &ant__17, &_sfright, &mod__16}
    }
};
behavior *sfTracking = &_beh_sfTracking;



void
sfTracking_setup (beh_params _params) {
   float c_dist = _params[0].f;
   float c_velocitat = _params[1].f;

            distancia_a_mantenir = c_dist;
            velocitat_a_mantenir = c_velocitat;
      }

/* global vars for update fn */



void
sfTracking_update(beh_params _params)
{
   float c_dist = _params[0].f;
   float c_velocitat = _params[1].f;
```

```
              // Inicialització de parèmetres

              marge_dist = dist_sens*c_dist;
              marge_angle = angle_sens*360;
              marge_parada = parada_sens*c_velocitat;

              // Cálcul del contorn

              c_calcul_contorn();

              // Parametres de lògica difusa

              too_fast = up_straight(sfTargetVel(), c_velocitat, c_velocitat+50);
              too_slow = straight_down(sfTargetVel(), c_velocitat-50,
c_velocitat);

              massa_angle = straight_down(angle_cami_control,-60.0,-20.0);
              massa_angle =
f_or(massa_angle,up_straight(angle_cami_control,20.0,60.0));

              esquerra = up_straight(angle_cami_control,0.0,marge_angle);
              dreta = straight_down(angle_cami_control,0.0,-marge_angle);

              // Parametres emergència de lògica difusa

              dist_act = distancia_control;

              emergencia =
(float)((dist_act>(c_dist*const_dist_emergencia))||(modul(angle_control)>angle_co
ntrol_emergencia)||(llNumPunts(llista)==0));
              emergencia =
f_or(emergencia,straight_down(r_reg,r_min_emergencia,r_max_emergencia));
              //emergencia = 0.0;

              // if (emergencia==1.0) sfSMessage("Emergencia!");

              e_massa_prop = up_straight(c_dist-dist_act,0.0,marge_dist);
              e_massa_lluny = straight_down(c_dist-dist_act,-marge_dist,0.0);

              if (!seguir_dreta)
              {
                     e_massa_lluny = up_straight(c_dist-dist_act,0.0,marge_dist);
                     e_massa_prop = straight_down(c_dist-dist_act,-marge_dist,0.0);
              }

              e_massa_angle = straight_down(angle_control,-60.0,-20.0);
              e_massa_angle =
f_or(e_massa_angle,up_straight(angle_control,20.0,60.0));

              e_esquerra = up_straight(angle_control,0.0,marge_angle);
              e_dreta = straight_down(angle_control,0.0,-marge_angle);

              // Monitoritzar variables

              c_monitoritzar(monitoritzacio);
              c_visualitzar();
       /* Begin activity section */
   c_act_v = c_act_t = 0.0;
    c_act_t = 1.0;
    c_act_v = 1.0;
    g__0 = c_velocitat;
    ant__1 =
f_and(f_not(emergencia),f_and(f_or(too_fast,too_slow),f_not(massa_angle)));
    ant__2 = f_and(f_not(emergencia),massa_angle);
    mod__3 = sfVSLOWLY;
    ant__4 = f_and(f_not(emergencia),esquerra);
    mod__5 = sfVSLOWLY;
```

```
    ant__6 = f_and(f_not(emergencia),dreta);
    g__7 = c_velocitat;
    ant__8 =
f_and(emergencia,f_and(f_or(too_fast,too_slow),f_not(e_massa_angle)));
    ant__9 = f_and(emergencia,e_massa_angle);
    mod__10 = sfVSLOWLY;
    ant__11 = f_and(emergencia,e_esquerra);
    mod__12 = sfVSLOWLY;
    ant__13 = f_and(emergencia,e_dreta);
    mod__14 = sfVSLOWLY;
    ant__15 = f_and(emergencia,f_and(e_massa_prop,f_not(e_massa_angle)));
    mod__16 = sfVSLOWLY;
    ant__17 = f_and(emergencia,f_and(e_massa_lluny,f_not(e_massa_angle)));
}


/* Ending code for behavior sfTracking  */


//Ending Behavior/Act Schema Parser
```

6 CalcularTrajectoria

6.1 CalcularTrajectoria.act

```
/* Iniciem alguns processos (han d'estar activats avans de començar) */

sfInitControlProcs();                   /* Per control de comportaments */
sfInitRegistrationProcs();      /* Registre del robot fent servir 'sensed
artifacts' */
sfInitInterpretationProcs();            /* Iniciem processos d'interpretació */
sfInitAwareProcs();             /* Procés per coneixer la situació */
connect local;

/* Carreguem les llibreries */

load e:\saphira\sfReachPosition\debug\sfReachPosition.dll;
load e:\saphira\CalcularTrajectoria\debug\CalcularTrajectoria.dll;

/* Definicio de la funsió Seguir Trajectoria */

act SeguirTrajectoria (int num)
{
       int final;
       float x;
       float y;
       float angle;
       float v;
       int global;

       sfrStart();

       InicialitzarTrajectoria(num);
       final = 0;

       while (!final)
       {
              x = CalcularXTrajectoria (num);
              y = CalcularYTrajectoria (num);
              angle = CalcularAngleTrajectoria (num);
              v = CalcularVelocitatTrajectoria (num);
              final = EsFinalTrajectoria (num);
              global = EsGlobalTrajectoria (num);

              if (global)
              {
                     sfrSetGlobalPosition(x,y,angle);
              }
              else
              {
                     sfrSetPosition(x,y,angle);
              }

              sfrSetVelocity(v);

              while (!sfrIsPositionReach())
              {
                     /* Esperar */
              }
       }
```

```
	sfrStop();
}
```

6.2 CalcularTrajectoria.h

```
/* ************************** */
/* FITXER CalcularTrajectoria.h  */
/* ************************** */

// ----------------------
// Definició de constants
// ----------------------

#define PI 3.1416

// ---------------------
// Definició de funcions
// ---------------------

void InicialitzarTrajectoria (int trj);
float CalcularXTrajectoria (int trj);
float CalcularYTrajectoria (int trj);
float CalcularAngleTrajectoria (int trj);
float CalcularVelocitatTrajectoria (int trj);
int EsFinalTrajectoria (int trj);
int  EsGlobalTrajectoria (int trj);

// ------------------------------
// Definició de variables globals
// ------------------------------
```

6.3 CalcularTrajectoria.c

```
/* ************************** */
/* FITXER CalcularTrajectoria.c  */
/* ************************** */

#include "CalcularTrajectoria.h"
#include "saphira.h"

// ----------------------------------
// Inclusió de fitxers de trajectories
// ----------------------------------

#include "Trajectoria_1.c"
#include "Trajectoria_2.c"
#include "Trajectoria_3.c"
#include "Trajectoria_4.c"
#include "Trajectoria_5.c"

// --------------------
// Codi de les funcions
// --------------------

void InicialitzarTrajectoria (int trj)
{
	if (trj==1) InicialitzarTrajectoria_1();
	if (trj==2) InicialitzarTrajectoria_2();
	if (trj==3) InicialitzarTrajectoria_3();
	if (trj==4) InicialitzarTrajectoria_4();
	if (trj==5) InicialitzarTrajectoria_5();
}
```

```
float CalcularXTrajectoria (int trj)
{
	if (trj==1) return CalcularXTrajectoria_1();
	if (trj==2) return CalcularXTrajectoria_2();
	if (trj==3) return CalcularXTrajectoria_3();
	if (trj==4) return CalcularXTrajectoria_4();
	if (trj==5) return CalcularXTrajectoria_5();
	return 0.0;
}

float CalcularYTrajectoria (int trj)
{
	if (trj==1) return CalcularYTrajectoria_1();
	if (trj==2) return CalcularYTrajectoria_2();
	if (trj==3) return CalcularYTrajectoria_3();
	if (trj==4) return CalcularYTrajectoria_4();
	if (trj==5) return CalcularYTrajectoria_5();
	return 0.0;
}

float CalcularAngleTrajectoria (int trj)
{
	if (trj==1) return CalcularAngleTrajectoria_1();
	if (trj==2) return CalcularAngleTrajectoria_2();
	if (trj==3) return CalcularAngleTrajectoria_3();
	if (trj==4) return CalcularAngleTrajectoria_4();
	if (trj==5) return CalcularAngleTrajectoria_5();
	return 0.0;
}

float CalcularVelocitatTrajectoria (int trj)
{
	if (trj==1) return CalcularVelocitatTrajectoria_1();
	if (trj==2) return CalcularVelocitatTrajectoria_2();
	if (trj==3) return CalcularVelocitatTrajectoria_3();
	if (trj==4) return CalcularVelocitatTrajectoria_4();
	if (trj==5) return CalcularVelocitatTrajectoria_5();
	return 100.0;
}

int EsFinalTrajectoria (int trj)
{
	if (trj==1) return EsFinalTrajectoria_1();
	if (trj==2) return EsFinalTrajectoria_2();
	if (trj==3) return EsFinalTrajectoria_3();
	if (trj==4) return EsFinalTrajectoria_4();
	if (trj==5) return EsFinalTrajectoria_5();
	return 1;
}

int EsGlobalTrajectoria (int trj)
{
	if (trj==1) return EsGlobalTrajectoria_1();
	if (trj==2) return EsGlobalTrajectoria_2();
	if (trj==3) return EsGlobalTrajectoria_3();
	if (trj==4) return EsGlobalTrajectoria_4();
	if (trj==5) return EsGlobalTrajectoria_5();
	return 0;
}

// ----------------------------------------
// Fitxers de càrrega i descàrrega de la dll
// ----------------------------------------

EXPORT void
sfLoadInit(void)    // Fitxer que s'avalua al carregar-se la dll
{
	// Missatge d'inici
```

```
	sfSMessage("Iniciant càrrega CalcularTrajectoria.dll");

	// Càrrega de les funcions a Colbert

	sfAddEvalFn("InicialitzarTrajectoria", InicialitzarTrajectoria, sfVOID, 1,
sfINT);

	sfAddEvalFn("CalcularXTrajectoria", CalcularXTrajectoria, sfFLOAT, 1,
sfINT);
	sfAddEvalFn("CalcularYTrajectoria", CalcularYTrajectoria, sfFLOAT, 1,
sfINT);
	sfAddEvalFn("CalcularAngleTrajectoria", CalcularAngleTrajectoria, sfFLOAT,
1, sfINT);
	sfAddEvalFn("CalcularVelocitatTrajectoria", CalcularVelocitatTrajectoria,
sfFLOAT, 1, sfINT);

	sfAddEvalFn("EsFinalTrajectoria", EsFinalTrajectoria, sfINT, 1, sfINT);
	sfAddEvalFn("EsGlobalTrajectoria", EsGlobalTrajectoria, sfINT, 1, sfINT);

	// Missatge final

	sfSMessage("Càrrega de CalcularTrajectoria.dll finalitzada");
}

EXPORT void
sfLoadExit(void)    // Fitxer que s'avalua al descarregar-se la dll
{
	// Missatge de descàrrega

	sfSMessage("Descàrrega de CalcularTrajectroia.dll efectuada");
}
```

6.4 Trajectoria_1.c

```
/* ***************************** */
/* FITXER CalcularTrajectoria_1.c */
/* ***************************** */

int punts_1;
int maxim_punts_1;

void InicialitzarTrajectoria_1 ()
{
	punts_1 = 0;
	maxim_punts_1 = 4;
}

float CalcularXTrajectoria_1 ()
{
	return 2000.0;
}

float CalcularYTrajectoria_1 ()
{
	return 0.0;
}

float CalcularAngleTrajectoria_1 ()
{
	return -90.0;
}

float CalcularVelocitatTrajectoria_1 ()
{
	return 200.0;
```

```
}

int EsFinalTrajectoria_1 ()
{
	punts_1++;
	return (punts_1 == maxim_punts_1);
}

int EsGlobalTrajectoria_1 ()
{
	return 0;
}
```

6.5 Trajectoria_2.c

```
/* ***************************** */
/* FITXER CalcularTrajectoria_2.c */
/* ***************************** */

#include <time.h>

// ------------------------------
// Definició de funcions internes
// ------------------------------

float aleatori(float x1, float x2);

int punts_2;
int maxim_punts_2;

// --------------------
// Codi de les funcions
// --------------------

void InicialitzarTrajectoria_2 ()
{
	long ltime;
	int stime;

	punts_2 = 0;
	maxim_punts_2 = 10;

	ltime = time(NULL);
	stime = (unsigned) ltime/2;
	srand(stime);
}

float CalcularXTrajectoria_2 ()
{
	// Generem un nombre aleatori entre -2000 i 2000

	return aleatori(-2000.0,2000.0);
}

float CalcularYTrajectoria_2 ()
{
	// Generem un nombre aleatori entre -2000 i 2000

	return aleatori(-2000.0,2000.0);
}

float CalcularAngleTrajectoria_2 ()
{
	// Generem un nombre aleatori entre 0 i 360

	return aleatori(0.0,360.0);
```

```
}

float CalcularVelocitatTrajectoria_2 ()
{
	return aleatori(50.0,250.0);
}

int EsFinalTrajectoria_2 ()
{
	punts_2++;
	return (punts_2 == maxim_punts_2);
}

int EsGlobalTrajectoria_2 ()
{
	return 1;
}

float aleatori(float x1, float x2)
{
	int num;
	float aux;

	if (x2<x1)
	{
		aux = x1;
		x1 = x2;
		x2 = aux;
	}

	num = rand();
	aux = ((float)num)/RAND_MAX;
	aux *= (x2-x1);
	aux += x1;

	return aux;
}
```

6.6 Trajectoria_3.c

```
/* ***************************** */
/* FITXER CalcularTrajectoria_3.c */
/* ***************************** */

int punts_3;
int maxim_punts_3;
int angle_3;

void InicialitzarTrajectoria_3 ()
{
	punts_3 = 0;
	maxim_punts_3 = 36;
	angle_3 = 0;
}

float CalcularXTrajectoria_3 ()
{
	return 3000*(float)cos(angle_3*2*PI/360);
}

float CalcularYTrajectoria_3 ()
{
	return 3000*(float)sin(angle_3*2*PI/360);
}

float CalcularAngleTrajectoria_3 ()
```

```
{
	return (float)(90 + angle_3);
}

float CalcularVelocitatTrajectoria_3 ()
{
	return 100.0;
}

int EsFinalTrajectoria_3 ()
{
	punts_3++;
	angle_3+=10;

	return (punts_3 == maxim_punts_3);
}

int EsGlobalTrajectoria_3 ()
{
	return 1;
}
```

6.7 Trajectoria_4.c

```
/* ***************************** */
/* FITXER CalcularTrajectoria_4.c */
/* ***************************** */

int punts_4;
int maxim_punts_4;

void InicialitzarTrajectoria_4 ()
{
	punts_4 = 0;
	maxim_punts_4 = 6;
}

float CalcularXTrajectoria_4 ()
{
	float aux = 0;

	if (punts_4 == 0) aux = 0.0;
	if (punts_4 == 1) aux = 1500.0;
	if (punts_4 == 2) aux = 1500.0;
	if (punts_4 == 3) aux = 3000.0;
	if (punts_4 == 4) aux = 3000.0;
	if (punts_4 == 5) aux = 0.0;

	return aux;
}

float CalcularYTrajectoria_4 ()
{
	float aux = 0;

	if (punts_4 == 0) aux = 1500.0;
	if (punts_4 == 1) aux = 1500.0;
	if (punts_4 == 2) aux = 750.0;
	if (punts_4 == 3) aux = 750.0;
	if (punts_4 == 4) aux = 0.0;
	if (punts_4 == 5) aux = 0.0;

	return aux;
}

float CalcularAngleTrajectoria_4 ()
```

```
{
	float aux = 0;

	if (punts_4 == 0) aux = 0.0;
	if (punts_4 == 1) aux = -90.0;
	if (punts_4 == 2) aux = 0.0;
	if (punts_4 == 3) aux = -90.0;
	if (punts_4 == 4) aux = 180.0;
	if (punts_4 == 5) aux = 90.0;

	return aux;
}

float CalcularVelocitatTrajectoria_4 ()
{
	float aux = 0;

	if (punts_4 == 0) aux = 150.0;
	if (punts_4 == 1) aux = 100.0;
	if (punts_4 == 2) aux = 100.0;
	if (punts_4 == 3) aux = 150.0;
	if (punts_4 == 4) aux = 150.0;
	if (punts_4 == 5) aux = 200.0;

	return aux;
}

int EsFinalTrajectoria_4 ()
{
	punts_4++;
	return (punts_4 == maxim_punts_4);
}

int EsGlobalTrajectoria_4 ()
{
	return 1;
}
```

6.8 Trajectoria_5.c

```
/* ***************************** */
/* FITXER CalcularTrajectoria_5.c */
/* ***************************** */

int punts_5;
int maxim_punts_5;
int angle_5;

void InicialitzarTrajectoria_5 ()
{
	punts_5 = 0;
	maxim_punts_5 = 150;
	angle_5 = 0;
}

float CalcularXTrajectoria_5 ()
{
	return 1000*(((float)angle_5*4/1000)+1)*(float)cos(angle_5*2*PI/360);
}

float CalcularYTrajectoria_5 ()
{
	return 1000*(((float)angle_5*4/1000)+1)*(float)sin(angle_5*2*PI/360);
}

float CalcularAngleTrajectoria_5 ()
```

```
{
	return (float)(90 + angle_5);
}

float CalcularVelocitatTrajectoria_5 ()
{
	return 100.0;
}

int EsFinalTrajectoria_5 ()
{
	punts_5++;
	angle_5 += 10;
	return (punts_5 == maxim_punts_5);
}

int EsGlobalTrajectoria_5 ()
{
	return 1;
}
```

6.9 Trajectoria_x.c

```
/* ***************************** */
/* FITXER CalcularTrajectoria_x.c */
/* ***************************** */

void InicialitzarTrajectoria_x ()
{
}

float CalcularXTrajectoria_x ()
{
	return 0.0;
}

float CalcularYTrajectoria_x ()
{
	return 0.0;
}

float CalcularAngleTrajectoria_x ()
{
	return 0.0;
}

float CalcularVelocitatTrajectoria_x ()
{
	return 100.0;
}

int EsFinalTrajectoria_x ()
{
	return 1;
}

int EsGlobalTrajectoria_x ()
{
	return 0;
}
```

7 CrearTrajectoria

7.1 CrearTrajectoria.act

```
/* Iniciem alguns processos (han d'estar activats avans de començar) */

sfInitControlProcs();                   /* Per control de comportaments */
sfInitRegistrationProcs();      /* Registre del robot fent servir 'sensed
artifacts' */
sfInitInterpretationProcs();            /* Iniciem processos d'interpretació */
sfInitAwareProcs();             /* Procés per coneixer la situació */
connect local;

/* Carreguem les llibreries */

load e:\saphira\sfDoTrajectory\debug\sfDoTrajectory.dll;
load e:\saphira\CrearTrajectoria\debug\CrearTrajectoria.dll;

/* Definicio de la funsió Seguir Trajectoria */

act SeguirTrajectoria (int num)
{
       int final;
       int punts;
       float x;
       float y;
       float v;
       int global;
       int spline;
       int k;
       float vslow;
       float distmin;
       float anglemin;

       int i;

       InicialitzarTrajectoria(num);
       final = EsFinalTrajectoria (num);

       while (!final)
       {
              v = VelocitatTrajectoria(num);
              spline = EsSplineTrajectoria(num);
              k = KTrajectoria(num);
              vslow = VSlowTrajectoria(num);
              distmin = MinDistTrajectoria(num);
              anglemin = MinAngleTrajectoria(num);

              sfDTRemoveAllPoints();
              sfDTSetVelocity(v);
              sfDTSetResolution(150);

              if (spline)
              {
                     sfDTSetSpline();
              }
              else
              {
                     sfDTSetBezier();
              }
```

```
                sfDTSetKTrajectory(k);
                sfDTSetVSlow(vslow);
                sfDTSetMinDist(distmin);
                sfDTSetMinAngle(anglemin);

                punts = NumPuntsTrajectoria(num);
                i = 0;
                while (i<punts)
                {
                        x = CrearXTrajectoria(num,i);
                        y = CrearYTrajectoria(num,i);
                        global = EsGlobalTrajectoria(num);

                        if (global)
                        {
                                sfDTAddGlobalPoint(x,y);
                        }
                        else
                        {
                                sfDTAddPoint(x,y);
                        }

                        i = i+1;
                }

                sfDTStart();

                final = EsFinalTrajectoria(num);

                while (!sfDTIsTrajectoryReach())
                {
                        /* Esperar */
                }

                sfDTStop();
        }
}
```

7.2 CrearTrajectoria.h

```
/* ************************* */
/* FITXER CrearTrajectoria.h     */
/* ************************* */

// ----------------------
// Definició de constants
// ----------------------

#define PI 3.1416

// ---------------------
// Definició de funcions
// ---------------------

void InicialitzarTrajectoria (int trj);
int NumPuntsTrajectoria (int trj);
float CrearXTrajectoria (int trj, int num);
float CrearYTrajectoria (int trj, int num);
float VelocitatTrajectoria (int trj);
int EsFinalTrajectoria (int trj);
int  EsGlobalTrajectoria (int trj);
int EsSplineTrajectoria (int trj);
int KTrajectoria (int trj);
float VSlowTrajectoria (int trj);
float MinDistTrajectoria (int trj);
```

```
float MinAngleTrajectoria (int trj);

// ------------------------------
// Definició de variables globals
// ------------------------------
```

7.3 CrearTrajectoria.c

```
/* ************************* */
/* FITXER CrearTrajectoria.c     */
/* ************************* */

#include "CrearTrajectoria.h"
#include "saphira.h"

// ----------------------------------
// Inclusió de fitxers de trajectories
// ----------------------------------

#include "Trajectoria_1.c"
#include "Trajectoria_2.c"
#include "Trajectoria_3.c"
#include "Trajectoria_4.c"
#include "Trajectoria_5.c"
#include "Trajectoria_x.c"

// --------------------
// Codi de les funcions
// --------------------

void InicialitzarTrajectoria (int trj)
{
       if (trj==1) InicialitzarTrajectoria_1();
       if (trj==2) InicialitzarTrajectoria_2();
       if (trj==3) InicialitzarTrajectoria_3();
       if (trj==4) InicialitzarTrajectoria_4();
       if (trj==5) InicialitzarTrajectoria_5();
       else InicialitzarTrajectoria_x();
}

int NumPuntsTrajectoria (int trj)
{
       if (trj==1) return NumPuntsTrajectoria_1();
       if (trj==2) return NumPuntsTrajectoria_2();
       if (trj==3) return NumPuntsTrajectoria_3();
       if (trj==4) return NumPuntsTrajectoria_4();
       if (trj==5) return NumPuntsTrajectoria_5();
       else return NumPuntsTrajectoria_x();
}

float CrearXTrajectoria (int trj, int num)
{
       if (trj==1) return CrearXTrajectoria_1(num);
       if (trj==2) return CrearXTrajectoria_2(num);
       if (trj==3) return CrearXTrajectoria_3(num);
       if (trj==4) return CrearXTrajectoria_4(num);
       if (trj==5) return CrearXTrajectoria_5(num);
       else return CrearXTrajectoria_x(num);
}

float CrearYTrajectoria (int trj, int num)
{
       if (trj==1) return CrearYTrajectoria_1(num);
       if (trj==2) return CrearYTrajectoria_2(num);
       if (trj==3) return CrearYTrajectoria_3(num);
```

```
	if (trj==4) return CrearYTrajectoria_4(num);
	if (trj==5) return CrearYTrajectoria_5(num);
	else return CrearYTrajectoria_x(num);
}

float VelocitatTrajectoria (int trj)
{
	if (trj==1) return VelocitatTrajectoria_1();
	if (trj==2) return VelocitatTrajectoria_2();
	if (trj==3) return VelocitatTrajectoria_3();
	if (trj==4) return VelocitatTrajectoria_4();
	if (trj==5) return VelocitatTrajectoria_5();
	else return VelocitatTrajectoria_x();
}

int EsFinalTrajectoria (int trj)
{
	if (trj==1) return EsFinalTrajectoria_1();
	if (trj==2) return EsFinalTrajectoria_2();
	if (trj==3) return EsFinalTrajectoria_3();
	if (trj==4) return EsFinalTrajectoria_4();
	if (trj==5) return EsFinalTrajectoria_5();
	else return EsFinalTrajectoria_x();
}

int  EsGlobalTrajectoria (int trj)
{
	if (trj==1) return EsGlobalTrajectoria_1();
	if (trj==2) return EsGlobalTrajectoria_2();
	if (trj==3) return EsGlobalTrajectoria_3();
	if (trj==4) return EsGlobalTrajectoria_4();
	if (trj==5) return EsGlobalTrajectoria_5();
	else return EsGlobalTrajectoria_x();
}

int EsSplineTrajectoria (int trj)
{
	if (trj==1) return EsSplineTrajectoria_1();
	if (trj==2) return EsSplineTrajectoria_2();
	if (trj==3) return EsSplineTrajectoria_3();
	if (trj==4) return EsSplineTrajectoria_4();
	if (trj==5) return EsSplineTrajectoria_5();
	else return EsSplineTrajectoria_x();
}

int KTrajectoria (int trj)
{
	if (trj==1) return KTrajectoria_1();
	if (trj==2) return KTrajectoria_2();
	if (trj==3) return KTrajectoria_3();
	if (trj==4) return KTrajectoria_4();
	if (trj==5) return KTrajectoria_5();
	else return KTrajectoria_x();
}

float VSlowTrajectoria (int trj)
{
	if (trj==1) return VSlowTrajectoria_1();
	if (trj==2) return VSlowTrajectoria_2();
	if (trj==3) return VSlowTrajectoria_3();
	if (trj==4) return VSlowTrajectoria_4();
	if (trj==5) return VSlowTrajectoria_5();
	else return VSlowTrajectoria_x();
}

float MinDistTrajectoria (int trj)
{
	if (trj==1) return MinDistTrajectoria_1();
```

```
	if (trj==2) return MinDistTrajectoria_2();
	if (trj==3) return MinDistTrajectoria_3();
	if (trj==4) return MinDistTrajectoria_4();
	if (trj==5) return MinDistTrajectoria_5();
	else return MinDistTrajectoria_x();
}

float MinAngleTrajectoria (int trj)
{
	if (trj==1) return MinAngleTrajectoria_1();
	if (trj==2) return MinAngleTrajectoria_2();
	if (trj==3) return MinAngleTrajectoria_3();
	if (trj==4) return MinAngleTrajectoria_4();
	if (trj==5) return MinAngleTrajectoria_5();
	else return MinAngleTrajectoria_x();
}

// ----------------------------------------
// Fitxers de càrrega i descàrrega de la dll
// ----------------------------------------

EXPORT void
sfLoadInit(void)    // Fitxer que s'avalua al carregar-se la dll
{
	// Missatge d'inici

	sfSMessage("Iniciant càrrega CrearTrajectoria.dll");

	// Càrrega de les funcions a Colbert

	sfAddEvalFn("InicialitzarTrajectoria", InicialitzarTrajectoria, sfVOID, 1,
sfINT);
	sfAddEvalFn("NumPuntsTrajectoria", NumPuntsTrajectoria, sfINT, 1, sfINT);
	sfAddEvalFn("CrearXTrajectoria", CrearXTrajectoria, sfFLOAT, 2, sfINT,
sfINT);
	sfAddEvalFn("CrearYTrajectoria", CrearYTrajectoria, sfFLOAT, 2, sfINT,
sfINT);
	sfAddEvalFn("VelocitatTrajectoria", VelocitatTrajectoria, sfFLOAT, 1,
sfINT);
	sfAddEvalFn("EsFinalTrajectoria", EsFinalTrajectoria, sfINT, 1, sfINT);
	sfAddEvalFn("EsGlobalTrajectoria", EsGlobalTrajectoria, sfINT, 1, sfINT);
	sfAddEvalFn("EsSplineTrajectoria", EsSplineTrajectoria, sfINT, 1, sfINT);
	sfAddEvalFn("KTrajectoria", KTrajectoria, sfINT, 1, sfINT);
	sfAddEvalFn("VSlowTrajectoria", VSlowTrajectoria, sfFLOAT, 1, sfINT);
	sfAddEvalFn("MinDistTrajectoria", MinDistTrajectoria, sfFLOAT, 1, sfINT);
	sfAddEvalFn("MinAngleTrajectoria", MinAngleTrajectoria, sfFLOAT, 1, sfINT);

	// Missatge final

	sfSMessage("Càrrega de CrearTrajectoria.dll finalitzada");
}

EXPORT void
sfLoadExit(void)    // Fitxer que s'avalua al descarregar-se la dll
{
	// Missatge de descàrrega

	sfSMessage("Descàrrega de CrearTrajectroia.dll efectuada");
}
```

7.4 Trajectoria_1.c

```
/* ********************** */
/* FITXER Trajectoria_1.c */
/* ********************** */
```

```
int punts_1;
int maxim_punts_1;

void InicialitzarTrajectoria_1 (void)
{
	punts_1 = 0;
	maxim_punts_1 = 1;
}

int NumPuntsTrajectoria_1 (void)
{
	return 10;
}

float CrearXTrajectoria_1 (int num)
{
	if (num == 0) return 0.0;
	if (num == 1) return 0.0;
	if (num == 2) return 0.0;
	if (num == 3) return 200.0;
	if (num == 4) return 2200.0;
	if (num == 5) return 2400.0;
	if (num == 6) return 2400.0;
	if (num == 7) return 2200.0;
	if (num == 8) return 200.0;
	if (num == 9) return 200.0;
	else return 0.0;
}

float CrearYTrajectoria_1 (int num)
{
	if (num == 0) return 0.0;
	if (num == 1) return 0.0;
	if (num == 2) return 2000.0;
	if (num == 3) return 2200.0;
	if (num == 4) return 2200.0;
	if (num == 5) return 2000.0;
	if (num == 6) return 0.0;
	if (num == 7) return -200.0;
	if (num == 8) return -200.0;
	if (num == 9) return -200.0;
	else return 0.0;
}

float VelocitatTrajectoria_1 (void)
{
	return 100.0;
}

int EsFinalTrajectoria_1 (void)
{
	punts_1++;
	return (punts_1==maxim_punts_1+1);
}

int  EsGlobalTrajectoria_1 (void)
{
	return 1;
}

int EsSplineTrajectoria_1 (void)
{
	return 1;
}

int KTrajectoria_1 (void)
{
	return 4;
```

```
}

float VSlowTrajectoria_1 (void)
{
	return 0.0;
}

float MinDistTrajectoria_1 (void)
{
	return 100.0;
}

float MinAngleTrajectoria_1 (void)
{
	return 10.0;
}
```

7.5 Trajectoria_2.c

```
/* ********************* */
/* FITXER Trajectoria_2.c */
/* ********************* */

#include <time.h>

float aleatori(float x1, float x2);

int punts_2;
int maxim_punts_2;

float x_aleat;
float y_aleat;
float angle_aleat;
int global;

void InicialitzarTrajectoria_2 (void)
{
	punts_2 = 0;
	maxim_punts_2 = 10;

	x_aleat = 0.0;
	y_aleat = 0.0;
	angle_aleat = 0.0;
	global = 0;
}

int NumPuntsTrajectoria_2 (void)
{
	return 6;
}

float CrearXTrajectoria_2 (int num)
{
	if (num==0)
	{
		global = 0;
		return 0.0;
	}
	if (num==1)
	{
		global = 0;
		return 600.0;
	}
	if (num==2)
	{
		global = 0;
```

```
		return 1200.0;
	}
	if (num==3)
	{
		global = 1;
		return x_aleat + 1200*(float)(cos(angle_aleat*2*PI/360));
	}
	if (num==4)
	{
		global = 1;
		return x_aleat + 600*(float)(cos(angle_aleat*2*PI/360));
	}
	if (num==5)
	{
		global = 1;
		return x_aleat;
	}
	else return 0.0;
}

float CrearYTrajectoria_2 (int num)
{
	if (num==0)
	{
		global = 0;
		return 0.0;
	}
	if (num==1)
	{
		global = 0;
		return 0.0;
	}
	if (num==2)
	{
		global = 0;
		return 0.0;
	}
	if (num==3)
	{
		global = 1;
		return y_aleat + 1200*(float)(sin(angle_aleat*2*PI/360));
	}
	if (num==4)
	{
		global = 1;
		return y_aleat + 600*(float)(sin(angle_aleat*2*PI/360));
	}
	if (num==5)
	{
		global = 1;
		return y_aleat;
	}
	else return 0.0;
}

float VelocitatTrajectoria_2 (void)
{
	return aleatori(50.0,300.0);
}

int EsFinalTrajectoria_2 (void)
{
	x_aleat = aleatori(-2000.0,2000.0);
	y_aleat = aleatori(-2000.0,2000.0);
	angle_aleat = aleatori(0.0,360.0);

	punts_2++;
	return (punts_2==maxim_punts_2+1);
```

```
}

int  EsGlobalTrajectoria_2 (void)
{
	return global;
}

int EsSplineTrajectoria_2 (void)
{
	return 0;
}

int KTrajectoria_2 (void)
{
	return 4;
}

float VSlowTrajectoria_2 (void)
{
	return -1.0;
}

float MinDistTrajectoria_2 (void)
{
	return 100.0;
}

float MinAngleTrajectoria_2 (void)
{
	return 10.0;
}

float aleatori(float x1, float x2)
{
	int num;
	float aux;

	if (x2<x1)
	{
		aux = x1;
		x1 = x2;
		x2 = aux;
	}

	num = rand();
	aux = ((float)num)/RAND_MAX;
	aux *= (x2-x1);
	aux += x1;

	return aux;
}
```

7.6 Trajectoria_3.c

```
/* ********************* */
/* FITXER Trajectoria_3.c */
/* ********************* */

int punts_3;
int maxim_punts_3;
int angle_3;

void InicialitzarTrajectoria_3 (void)
{
	punts_3 = 0;
	maxim_punts_3 = 1;
```

```
	angle_3 = 0;
}

int NumPuntsTrajectoria_3 (void)
{
	return 34;
}

float CrearXTrajectoria_3 (int num)
{
	angle_3 += 10;
	return 1500*(float)cos(angle_3*2*PI/360)-1500;
}

float CrearYTrajectoria_3 (int num)
{
	return 1500*(float)sin(angle_3*2*PI/360);
}

float VelocitatTrajectoria_3 (void)
{
	return 100.0;
}

int EsFinalTrajectoria_3 (void)
{
	punts_3++;
	return (punts_3 == maxim_punts_3+1);
}

int  EsGlobalTrajectoria_3 (void)
{
	return 1;
}

int EsSplineTrajectoria_3 (void)
{
	return 1;
}

int KTrajectoria_3 (void)
{
	return 4;
}

float VSlowTrajectoria_3 (void)
{
	return 0.0;
}

float MinDistTrajectoria_3 (void)
{
	return 100.0;
}

float MinAngleTrajectoria_3 (void)
{
	return 10.0;
}
```

7.7 Trajectoria_4.c

```
/* ********************** */
/* FITXER Trajectoria_4.c */
/* ********************** */
```

```
int punts_4;
int maxim_punts_4;

void InicialitzarTrajectoria_4 (void)
{
	punts_4 = 0;
	maxim_punts_4 = 1;
}

int NumPuntsTrajectoria_4 (void)
{
	return 9;
}

float CrearXTrajectoria_4 (int num)
{
	if (num==0) return 0.0;
	if (num==1) return 0.0;
	if (num==2) return 0.0;
	if (num==3) return 1500.0;
	if (num==4) return 1500.0;
	if (num==5) return 3000.0;
	if (num==6) return 3000.0;
	if (num==7) return 400.0;
	if (num==8) return 400.0;
	else return 0.0;
}

float CrearYTrajectoria_4 (int num)
{
	if (num==0) return 0.0;
	if (num==1) return 0.0;
	if (num==2) return 1500.0;
	if (num==3) return 1500.0;
	if (num==4) return 750.0;
	if (num==5) return 750.0;
	if (num==6) return 0.0;
	if (num==7) return 0.0;
	if (num==8) return 0.0;
	else return 0.0;
}

float VelocitatTrajectoria_4 (void)
{
	return 100.0;
}

int EsFinalTrajectoria_4 (void)
{
	punts_4++;
	return (punts_4==maxim_punts_4+1);
}

int  EsGlobalTrajectoria_4 (void)
{
	return 1;
}

int EsSplineTrajectoria_4 (void)
{
	return 1;
}

int KTrajectoria_4 (void)
{
	return 4;
}
```

```
float VSlowTrajectoria_4 (void)
{
	return 0.0;
}

float MinDistTrajectoria_4 (void)
{
	return 100.0;
}

float MinAngleTrajectoria_4 (void)
{
	return 10.0;
}
```

7.8 Trajectoria_5.c

```
/* ********************* */
/* FITXER Trajectoria_5.c */
/* ********************* */

int punts_5;
int maxim_punts_5;
int angle_5;

void InicialitzarTrajectoria_5 (void)
{
	punts_5 = 0;
	maxim_punts_5 = 1;
	angle_5 = 0;
}

int NumPuntsTrajectoria_5 (void)
{
	return 35;
}

float CrearXTrajectoria_5 (int num)
{
	angle_5 += 20;
	return 1000*(((float)angle_5*4/1000)+1)*(float)cos(angle_5*2*PI/360)-1000;
}

float CrearYTrajectoria_5 (int num)
{
	return 1000*(((float)angle_5*4/1000)+1)*(float)sin(angle_5*2*PI/360);
}

float VelocitatTrajectoria_5 (void)
{
	return 100.0;
}

int EsFinalTrajectoria_5 (void)
{
	punts_5++;
	return (punts_5 == maxim_punts_5+1);
}

int  EsGlobalTrajectoria_5 (void)
{
	return 1;
}

int EsSplineTrajectoria_5 (void)
{
```

```
	return 1;
}

int KTrajectoria_5 (void)
{
	return 4;
}

float VSlowTrajectoria_5 (void)
{
	return 0.0;
}

float MinDistTrajectoria_5 (void)
{
	return 100.0;
}

float MinAngleTrajectoria_5 (void)
{
	return 10.0;
}
```

7.9 Trajectoria_x.c

```
/* ********************** */
/* FITXER Trajectoria_x.c */
/* ********************** */

void InicialitzarTrajectoria_x (void)
{
}

int NumPuntsTrajectoria_x (void)
{
	return 0;
}

float CrearXTrajectoria_x (int num)
{
	return 0.0;
}

float CrearYTrajectoria_x (int num)
{
	return 0.0;
}

float VelocitatTrajectoria_x (void)
{
	return 100.0;
}

int EsFinalTrajectoria_x (void)
{
	return 1;
}

int  EsGlobalTrajectoria_x (void)
{
	return 0;
}

int EsSplineTrajectoria_x (void)
{
	return 1;
```

```
}

int KTrajectoria_x (void)
{
	return 4;
}

float VSlowTrajectoria_x (void)
{
	return 0.0;
}

float MinDistTrajectoria_x (void)
{
	return 100.0;
}

float MinAngleTrajectoria_x (void)
{
	return 10.0;
}
```