```
/*****************************************************************************
* SPAOFE - Sonar Perception Avoiding Objects Fuzzy Engine V0.0                *
* © Copyright 2012,2013 Filippo Pardini - filippo@robotica.eng.br            *
*                                                                            *
* This program is free software: you can redistribute it and/or modify it    *
* under the terms of the GNU Lesser General Public License as published by   *
* the Free Software Foundation, either version 3 of the License, or any      *
* later version.                                                             *
*                                                                            *
* This program is distributed in the hope that it will be useful,            *
* but WITHOUT ANY WARRANTY; without even the implied warranty of             *
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the               *
* GNU Lesser General Public License for more details.                        *
*                                                                            *
* You should have received a copy of the GNU Lesser General Public License   *
* along with this program. If not, see <http://www.gnu.org/licenses/>.       *
*****************************************************************************/

/*****************************************************************************
This program was created by the
CodeWizardAVR V3.06 Standard
Automatic Program Generator
© Copyright 1998-2013 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project : XR-1 The First
          SPAOFE-Sonar Perception Avoiding Objects Fuzzy Engine
Version : Preliminary
Date    : 18/09/2013
Author  : Filippo Pardini
Company : Filippo Pardini
Comments: This is the first attempt to SPAOFE engine.
It is syntactically correct but it is not tested yet.
I am posting this version for interested people that
can help me to semantically verify and evaluate it.


Chip type               : ATxmega128A1
Program type            : Application
AVR Core Clock frequency: 32,000000 MHz
Memory model            : Small
Data Stack size         : 2500
```

```
43: ********************************************************************************/
44:
45: #include <io.h>
46: #include <stdio.h>
47: #include <twix.h>
48: #include <stdbool.h>
49: #include <math.h>
50: #include <delay.h>
51: #include <stdlib.h>
52:
53: //**********************SPAOFE*************************
54:
55: #define zone 1.0
56: #define zzero 0.0
57: #define zor(a,b) ((a) > (b) ? (a) : (b))
58: #define zand(a,b) ((a) < (b) ? (a) : (b))
59: #define znot(a) (zone+zzero-a)
60:
61: #define N_Sonar 8       // Number of ultrasonic sensor
62: #define infinito 65535  // Infinity distance
63: #define eixo 33.5       // Distance (cm) between wheels
64:
65: #define reta_up(n,x) (x - x1[n])/(x2[n]-x1[n])
66: #define reta_down(n,x) 1-(x - x1[n])/(x2[n]-x1[n])
67:
68: #define ZE PC[0]
69: #define LO PC[1]
70: #define HI PC[2]
71: #define RB PA[0]
72: #define RF PA[1]
73: #define LF PA[2]
74: #define LB PA[3]
75: #define VLP PV[0]
76: #define LP PV[1]
77: #define MP PV[2]
78: #define HP PV[3]
79: #define VHP PV[4]
80:
81: #define ad_aeternum while(1)
82:
83: struct sonar
84: {
```

```
85:     // Number of sensors in the ring .
86:     unsigned char Sonar_Number;
87:     // Distance previous reading instant of time
88:     unsigned long int Ti_1[N_Sonar];
89:     // Time elapsed since previous distance reading.
90:     unsigned long int DTi[N_Sonar];
91:     // Sensors I2C addresses.
92:     unsigned char Sonar_Addr[N_Sonar];
93:     // 0 if > Minimum_Distance; 1 if <= Minimum_Distance).
94:     unsigned char Sonar_Condition[N_Sonar];
95:     // Sensor readed distance.
96:     unsigned int Sonar_Distance[N_Sonar];
97:     // Distance change
98:     int Sonar_Distance_Change[N_Sonar];
99:     // Sensor individual perception direction (degrees).
100:    float Sonar_Angle[N_Sonar];
101:    // Normalized individual Perception vector value.
102:    float Perception_Value[N_Sonar];
103:    // General Perception change in time.
104:    float General_Perception_Change;
105:    // General Perception direction. (-180 to 180) degrees to heading
106:    float General_Perception_Angle;
107:    // Normalized General Perception Value (0 to 1).
108:    float General_Perception_Value;
109:    // Maximum acceleration (cm/s^2).
110:    float Maximum_Acceleration;
111:    // Maximum velocity (cm/s).
112:    float Maximum_Velocity;
113:    // Minimum velocity (cm/s).
114:    float Minimum_Velocity;
115:    // Maximum distance (cm).
116:    float Maximum_Distance;
117:    // Minimum distance (cm).
118:    float Minimum_Distance;
119:    // Normal velocity (cm/s) corresponding to acceleration sector ZERO.
120:    float Velocity;
121: };
122:
123: //Time (49 days capacity)
124: unsigned long int relogio = 0;
125: // 9 - Stop abruptly; 10 - Stop smoothly.
126: unsigned char Brake;
```

```
127: // Sectors VLP,LP,MP,HP,VHP.
128: float PV[5];
129: // PV sector index.
130: unsigned char iPV[5];
131: // Number of iPVs.
132: unsigned char nPV;
133: // Sectors RB,RF,LF,LB.
134: float PA[4];
135: // PA sector index.
136: unsigned char iPA[4];
137: // Number of iPAs.
138: unsigned char nPA;
139: // Sectors ZE,LO,HI.
140: float PC[3];
141: // PC sector index.
142: unsigned char iPC[3];
143: // Number of iPCs.
144: unsigned char nPC;
145: // Steer sector index.
146: unsigned char iSteer[10];
147: // Corresponding cut values.
148: float CSteer[10];
149: // Number of iSteer.
150: unsigned char nSteer;
151: // Acceleration sector index.
152: unsigned char iAcc[10];
153: // Corresponding cut values.
154: float CAcc[10];
155: // Number of iAcc.
156: unsigned char nAcc;
157: // Steer crisp value.
158: float Steer_Crisp;
159: // Turn direction (-1 right; +1 left; 0 stright).
160: signed char Direcao;
161: // Turn radius (cm) to obtain Steer_Crisp (degrees/s).
162: float Rg;
163: // Acceleration crisp value.
164: float Acceleration_Crisp;
165:
166: char CharConv[15];          // Debug
167: char CSt0[5];               // Debug
168: char CSt1[5];               // Debug
```

```
169: char CSt2[5];              // Debug
170: char CSt3[5];              // Debug
171: char CSt4[5];              // Debug
172: char CSt5[5];              // Debug
173:
174: unsigned int range_sonar_X(unsigned char sim, unsigned char ender, struct sonar sensor);
175: void General_Perception_Vector(struct sonar sensor);
176: void General_Perception_V_Change(struct sonar sensor);
177: bool Perception_Angle_Membership(float Angle);
178: bool Perception_Value_Membership(float Value);
179: bool Perception_Change_Membership(float Value);
180: void Rules_Steer(void);
181: void Rules_Acceleration(void);
182: float Cutted_Defuzzy_Steer(float Xinicial,float Xfinal,float Step);
183: float Scaled_Defuzzy_Steer(float Xinicial,float Xfinal,float Step);
184: float Cutted_Defuzzy_Acceleration(float Xinicial,float Xfinal,float Step,struct sonar sensor);
185: float Scaled_Defuzzy_Acceleration(float Xinicial,float Xfinal,float Step,struct sonar sensor);
186: float fHR(float x);
187: float fR(float x);
188: float fC(float x);
189: float fL(float x);
190: float fHL(float x);
191: float fEB(float x,struct sonar sensor);
192: float fB(float x,struct sonar sensor);
193: float fZ(float x,struct sonar sensor);
194: float fP(float x,struct sonar sensor);
195: void Msectors_Nao_Zero(float *Vetor,unsigned char Comp,unsigned char *Indices,unsigned char *Num);
196:
197: //**********************SPAOFE*************************
198:
199: // System Clocks initialization
200: void system_clocks_init(void)
201: {
202:     unsigned char n,s;
203:
204:     // Optimize for speed
205:     #pragma optsize-
206:     // Save interrupts enabled/disabled state
207:     s=SREG;
208:     // Disable interrupts
209:     #asm("cli")
210:
```

```
211:     // Internal 32 kHz RC oscillator initialization
212:     // Enable the internal 32 kHz RC oscillator
213:     OSC.CTRL|=OSC_RC32KEN_bm;
214:     // Wait for the internal 32 kHz RC oscillator to stabilize
215:     while ((OSC.STATUS & OSC_RC32KRDY_bm)==0);
216:
217:     // Internal 32 MHz RC oscillator initialization
218:     // Enable the internal 32 MHz RC oscillator
219:     OSC.CTRL|=OSC_RC32MEN_bm;
220:
221:     // System Clock prescaler A division factor: 1
222:     // System Clock prescalers B & C division factors: B:1, C:1
223:     // ClkPer4: 32000,000 kHz
224:     // ClkPer2: 32000,000 kHz
225:     // ClkPer:  32000,000 kHz
226:     // ClkCPU:  32000,000 kHz
227:     n=(CLK.PSCTRL & (~(CLK_PSADIV_gm | CLK_PSBCDIV1_bm | CLK_PSBCDIV0_bm))) |
228:         CLK_PSADIV_1_gc | CLK_PSBCDIV_1_1_gc;
229:     CCP=CCP_IOREG_gc;
230:     CLK.PSCTRL=n;
231:
232:     // Internal 32 MHz RC osc. calibration reference clock source: 32.768 kHz Internal Osc.
233:     OSC.DFLLCTRL&= ~(OSC_RC32MCREF_bm | OSC_RC2MCREF_bm);
234:     // Enable the auto-calibration of the internal 32 MHz RC oscillator
235:     DFLLRC32M.CTRL|=DFLL_ENABLE_bm;
236:
237:     // Wait for the internal 32 MHz RC oscillator to stabilize
238:     while ((OSC.STATUS & OSC_RC32MRDY_bm)==0);
239:
240:     // Select the system clock source: 32 MHz Internal RC Osc.
241:     n=(CLK.CTRL & (~CLK_SCLKSEL_gm)) | CLK_SCLKSEL_RC32M_gc;
242:     CCP=CCP_IOREG_gc;
243:     CLK.CTRL=n;
244:
245:     // Disable the unused oscillators: 2 MHz, external clock/crystal oscillator, PLL
246:     OSC.CTRL&= ~(OSC_RC2MEN_bm | OSC_XOSCEN_bm | OSC_PLLEN_bm);
247:
248:     // ClkPer output disabled
249:     PORTCFG.CLKEVOUT&= ~PORTCFG_CLKOUT_gm;
250:     // Restore interrupts enabled/disabled state
251:     SREG=s;
252:     // Restore optimization for size if needed
```

```
253:     #pragma optsize_default
254: }
255:
256: // Ports initialization
257:     void ports_init(void)
258:     {
259:     // PORTA initialization
260:     // OUT register
261:     PORTA.OUT=0x00;
262:     // Pin0: Input
263:     // Pin1: Input
264:     // Pin2: Input
265:     // Pin3: Input
266:     // Pin4: Input
267:     // Pin5: Input
268:     // Pin6: Input
269:     // Pin7: Input
270:     PORTA.DIR=0x00;
271:     // Pin0 Output/Pull configuration: Totempole/No
272:     // Pin0 Input/Sense configuration: Sense both edges
273:     // Pin0 Inverted: Off
274:     // Pin0 Slew Rate Limitation: Off
275:     PORTA.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
276:     // Pin1 Output/Pull configuration: Totempole/No
277:     // Pin1 Input/Sense configuration: Sense both edges
278:     // Pin1 Inverted: Off
279:     // Pin1 Slew Rate Limitation: Off
280:     PORTA.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
281:     // Pin2 Output/Pull configuration: Totempole/No
282:     // Pin2 Input/Sense configuration: Sense both edges
283:     // Pin2 Inverted: Off
284:     // Pin2 Slew Rate Limitation: Off
285:     PORTA.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
286:     // Pin3 Output/Pull configuration: Totempole/No
287:     // Pin3 Input/Sense configuration: Sense both edges
288:     // Pin3 Inverted: Off
289:     // Pin3 Slew Rate Limitation: Off
290:     PORTA.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
291:     // Pin4 Output/Pull configuration: Totempole/No
292:     // Pin4 Input/Sense configuration: Sense both edges
293:     // Pin4 Inverted: Off
294:     // Pin4 Slew Rate Limitation: Off
```

```
295:     PORTA.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
296:     // Pin5 Output/Pull configuration: Totempole/No
297:     // Pin5 Input/Sense configuration: Sense both edges
298:     // Pin5 Inverted: Off
299:     // Pin5 Slew Rate Limitation: Off
300:     PORTA.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
301:     // Pin6 Output/Pull configuration: Totempole/No
302:     // Pin6 Input/Sense configuration: Sense both edges
303:     // Pin6 Inverted: Off
304:     // Pin6 Slew Rate Limitation: Off
305:     PORTA.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
306:     // Pin7 Output/Pull configuration: Totempole/No
307:     // Pin7 Input/Sense configuration: Sense both edges
308:     // Pin7 Inverted: Off
309:     // Pin7 Slew Rate Limitation: Off
310:     PORTA.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
311:     // Interrupt 0 level: Disabled
312:     // Interrupt 1 level: Disabled
313:     PORTA.INTCTRL=(PORTA.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
314:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
315:     // Pin0 Pin Change interrupt 0: Off
316:     // Pin1 Pin Change interrupt 0: Off
317:     // Pin2 Pin Change interrupt 0: Off
318:     // Pin3 Pin Change interrupt 0: Off
319:     // Pin4 Pin Change interrupt 0: Off
320:     // Pin5 Pin Change interrupt 0: Off
321:     // Pin6 Pin Change interrupt 0: Off
322:     // Pin7 Pin Change interrupt 0: Off
323:     PORTA.INT0MASK=0x00;
324:     // Pin0 Pin Change interrupt 1: Off
325:     // Pin1 Pin Change interrupt 1: Off
326:     // Pin2 Pin Change interrupt 1: Off
327:     // Pin3 Pin Change interrupt 1: Off
328:     // Pin4 Pin Change interrupt 1: Off
329:     // Pin5 Pin Change interrupt 1: Off
330:     // Pin6 Pin Change interrupt 1: Off
331:     // Pin7 Pin Change interrupt 1: Off
332:     PORTA.INT1MASK=0x00;
333:
334:     // PORTB initialization
335:     // OUT register
336:     PORTB.OUT=0x00;
```

```
337:     // Pin0: Input
338:     // Pin1: Input
339:     // Pin2: Input
340:     // Pin3: Input
341:     // Pin4: Input
342:     // Pin5: Input
343:     // Pin6: Input
344:     // Pin7: Input
345:     PORTB.DIR=0x00;
346:     // Pin0 Output/Pull configuration: Totempole/No
347:     // Pin0 Input/Sense configuration: Sense both edges
348:     // Pin0 Inverted: Off
349:     // Pin0 Slew Rate Limitation: Off
350:     PORTB.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
351:     // Pin1 Output/Pull configuration: Totempole/No
352:     // Pin1 Input/Sense configuration: Sense both edges
353:     // Pin1 Inverted: Off
354:     // Pin1 Slew Rate Limitation: Off
355:     PORTB.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
356:     // Pin2 Output/Pull configuration: Totempole/No
357:     // Pin2 Input/Sense configuration: Sense both edges
358:     // Pin2 Inverted: Off
359:     // Pin2 Slew Rate Limitation: Off
360:     PORTB.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
361:     // Pin3 Output/Pull configuration: Totempole/No
362:     // Pin3 Input/Sense configuration: Sense both edges
363:     // Pin3 Inverted: Off
364:     // Pin3 Slew Rate Limitation: Off
365:     PORTB.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
366:     // Pin4 Output/Pull configuration: Totempole/No
367:     // Pin4 Input/Sense configuration: Sense both edges
368:     // Pin4 Inverted: Off
369:     // Pin4 Slew Rate Limitation: Off
370:     PORTB.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
371:     // Pin5 Output/Pull configuration: Totempole/No
372:     // Pin5 Input/Sense configuration: Sense both edges
373:     // Pin5 Inverted: Off
374:     // Pin5 Slew Rate Limitation: Off
375:     PORTB.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
376:     // Pin6 Output/Pull configuration: Totempole/No
377:     // Pin6 Input/Sense configuration: Sense both edges
378:     // Pin6 Inverted: Off
```

```
379:     // Pin6 Slew Rate Limitation: Off
380:     PORTB.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
381:     // Pin7 Output/Pull configuration: Totempole/No
382:     // Pin7 Input/Sense configuration: Sense both edges
383:     // Pin7 Inverted: Off
384:     // Pin7 Slew Rate Limitation: Off
385:     PORTB.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
386:     // Interrupt 0 level: Disabled
387:     // Interrupt 1 level: Disabled
388:     PORTB.INTCTRL=(PORTB.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
389:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
390:     // Pin0 Pin Change interrupt 0: Off
391:     // Pin1 Pin Change interrupt 0: Off
392:     // Pin2 Pin Change interrupt 0: Off
393:     // Pin3 Pin Change interrupt 0: Off
394:     // Pin4 Pin Change interrupt 0: Off
395:     // Pin5 Pin Change interrupt 0: Off
396:     // Pin6 Pin Change interrupt 0: Off
397:     // Pin7 Pin Change interrupt 0: Off
398:     PORTB.INT0MASK=0x00;
399:     // Pin0 Pin Change interrupt 1: Off
400:     // Pin1 Pin Change interrupt 1: Off
401:     // Pin2 Pin Change interrupt 1: Off
402:     // Pin3 Pin Change interrupt 1: Off
403:     // Pin4 Pin Change interrupt 1: Off
404:     // Pin5 Pin Change interrupt 1: Off
405:     // Pin6 Pin Change interrupt 1: Off
406:     // Pin7 Pin Change interrupt 1: Off
407:     PORTB.INT1MASK=0x00;
408:
409:     // PORTC initialization
410:     // OUT register
411:     PORTC.OUT=0x00;
412:     // Pin0: Input
413:     // Pin1: Input
414:     // Pin2: Input
415:     // Pin3: Input
416:     // Pin4: Input
417:     // Pin5: Input
418:     // Pin6: Input
419:     // Pin7: Input
420:     PORTC.DIR=0x00;
```

```
421:     // Pin0 Output/Pull configuration: Totempole/No
422:     // Pin0 Input/Sense configuration: Sense both edges
423:     // Pin0 Inverted: Off
424:     // Pin0 Slew Rate Limitation: Off
425:     PORTC.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
426:     // Pin1 Output/Pull configuration: Totempole/No
427:     // Pin1 Input/Sense configuration: Sense both edges
428:     // Pin1 Inverted: Off
429:     // Pin1 Slew Rate Limitation: Off
430:     PORTC.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
431:     // Pin2 Output/Pull configuration: Totempole/No
432:     // Pin2 Input/Sense configuration: Sense both edges
433:     // Pin2 Inverted: Off
434:     // Pin2 Slew Rate Limitation: Off
435:     PORTC.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
436:     // Pin3 Output/Pull configuration: Totempole/No
437:     // Pin3 Input/Sense configuration: Sense both edges
438:     // Pin3 Inverted: Off
439:     // Pin3 Slew Rate Limitation: Off
440:     PORTC.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
441:     // Pin4 Output/Pull configuration: Totempole/No
442:     // Pin4 Input/Sense configuration: Sense both edges
443:     // Pin4 Inverted: Off
444:     // Pin4 Slew Rate Limitation: Off
445:     PORTC.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
446:     // Pin5 Output/Pull configuration: Totempole/No
447:     // Pin5 Input/Sense configuration: Sense both edges
448:     // Pin5 Inverted: Off
449:     // Pin5 Slew Rate Limitation: Off
450:     PORTC.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
451:     // Pin6 Output/Pull configuration: Totempole/No
452:     // Pin6 Input/Sense configuration: Sense both edges
453:     // Pin6 Inverted: Off
454:     // Pin6 Slew Rate Limitation: Off
455:     PORTC.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
456:     // Pin7 Output/Pull configuration: Totempole/No
457:     // Pin7 Input/Sense configuration: Sense both edges
458:     // Pin7 Inverted: Off
459:     // Pin7 Slew Rate Limitation: Off
460:     PORTC.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
461:     // Interrupt 0 level: Disabled
462:     // Interrupt 1 level: Disabled
```

```
463:     PORTC.INTCTRL=(PORTC.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
464:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
465:     // Pin0 Pin Change interrupt 0: Off
466:     // Pin1 Pin Change interrupt 0: Off
467:     // Pin2 Pin Change interrupt 0: Off
468:     // Pin3 Pin Change interrupt 0: Off
469:     // Pin4 Pin Change interrupt 0: Off
470:     // Pin5 Pin Change interrupt 0: Off
471:     // Pin6 Pin Change interrupt 0: Off
472:     // Pin7 Pin Change interrupt 0: Off
473:     PORTC.INT0MASK=0x00;
474:     // Pin0 Pin Change interrupt 1: Off
475:     // Pin1 Pin Change interrupt 1: Off
476:     // Pin2 Pin Change interrupt 1: Off
477:     // Pin3 Pin Change interrupt 1: Off
478:     // Pin4 Pin Change interrupt 1: Off
479:     // Pin5 Pin Change interrupt 1: Off
480:     // Pin6 Pin Change interrupt 1: Off
481:     // Pin7 Pin Change interrupt 1: Off
482:     PORTC.INT1MASK=0x00;
483:
484:     // PORTD initialization
485:     // OUT register
486:     PORTD.OUT=0x00;
487:     // Pin0: Input
488:     // Pin1: Input
489:     // Pin2: Input
490:     // Pin3: Input
491:     // Pin4: Input
492:     // Pin5: Input
493:     // Pin6: Input
494:     // Pin7: Input
495:     PORTD.DIR=0x00;
496:     // Pin0 Output/Pull configuration: Totempole/No
497:     // Pin0 Input/Sense configuration: Sense both edges
498:     // Pin0 Inverted: Off
499:     // Pin0 Slew Rate Limitation: Off
500:     PORTD.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
501:     // Pin1 Output/Pull configuration: Totempole/No
502:     // Pin1 Input/Sense configuration: Sense both edges
503:     // Pin1 Inverted: Off
504:     // Pin1 Slew Rate Limitation: Off
```

```
505:     PORTD.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
506:     // Pin2 Output/Pull configuration: Totempole/No
507:     // Pin2 Input/Sense configuration: Sense both edges
508:     // Pin2 Inverted: Off
509:     // Pin2 Slew Rate Limitation: Off
510:     PORTD.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
511:     // Pin3 Output/Pull configuration: Totempole/No
512:     // Pin3 Input/Sense configuration: Sense both edges
513:     // Pin3 Inverted: Off
514:     // Pin3 Slew Rate Limitation: Off
515:     PORTD.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
516:     // Pin4 Output/Pull configuration: Totempole/No
517:     // Pin4 Input/Sense configuration: Sense both edges
518:     // Pin4 Inverted: Off
519:     // Pin4 Slew Rate Limitation: Off
520:     PORTD.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
521:     // Pin5 Output/Pull configuration: Totempole/No
522:     // Pin5 Input/Sense configuration: Sense both edges
523:     // Pin5 Inverted: Off
524:     // Pin5 Slew Rate Limitation: Off
525:     PORTD.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
526:     // Pin6 Output/Pull configuration: Totempole/No
527:     // Pin6 Input/Sense configuration: Sense both edges
528:     // Pin6 Inverted: Off
529:     // Pin6 Slew Rate Limitation: Off
530:     PORTD.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
531:     // Pin7 Output/Pull configuration: Totempole/No
532:     // Pin7 Input/Sense configuration: Sense both edges
533:     // Pin7 Inverted: Off
534:     // Pin7 Slew Rate Limitation: Off
535:     PORTD.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
536:     // Interrupt 0 level: Disabled
537:     // Interrupt 1 level: Disabled
538:     PORTD.INTCTRL=(PORTD.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
539:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
540:     // Pin0 Pin Change interrupt 0: Off
541:     // Pin1 Pin Change interrupt 0: Off
542:     // Pin2 Pin Change interrupt 0: Off
543:     // Pin3 Pin Change interrupt 0: Off
544:     // Pin4 Pin Change interrupt 0: Off
545:     // Pin5 Pin Change interrupt 0: Off
546:     // Pin6 Pin Change interrupt 0: Off
```

```
547:     // Pin7 Pin Change interrupt 0: Off
548:     PORTD.INT0MASK=0x00;
549:     // Pin0 Pin Change interrupt 1: Off
550:     // Pin1 Pin Change interrupt 1: Off
551:     // Pin2 Pin Change interrupt 1: Off
552:     // Pin3 Pin Change interrupt 1: Off
553:     // Pin4 Pin Change interrupt 1: Off
554:     // Pin5 Pin Change interrupt 1: Off
555:     // Pin6 Pin Change interrupt 1: Off
556:     // Pin7 Pin Change interrupt 1: Off
557:     PORTD.INT1MASK=0x00;
558:
559:     // PORTE initialization
560:     // OUT register
561:     PORTE.OUT=0x00;
562:     // Pin0: Input
563:     // Pin1: Input
564:     // Pin2: Input
565:     // Pin3: Input
566:     // Pin4: Input
567:     // Pin5: Input
568:     // Pin6: Input
569:     // Pin7: Input
570:     PORTE.DIR=0x00;
571:     // Pin0 Output/Pull configuration: Totempole/No
572:     // Pin0 Input/Sense configuration: Sense both edges
573:     // Pin0 Inverted: Off
574:     // Pin0 Slew Rate Limitation: Off
575:     PORTE.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
576:     // Pin1 Output/Pull configuration: Totempole/No
577:     // Pin1 Input/Sense configuration: Sense both edges
578:     // Pin1 Inverted: Off
579:     // Pin1 Slew Rate Limitation: Off
580:     PORTE.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
581:     // Pin2 Output/Pull configuration: Totempole/No
582:     // Pin2 Input/Sense configuration: Sense both edges
583:     // Pin2 Inverted: Off
584:     // Pin2 Slew Rate Limitation: Off
585:     PORTE.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
586:     // Pin3 Output/Pull configuration: Totempole/No
587:     // Pin3 Input/Sense configuration: Sense both edges
588:     // Pin3 Inverted: Off
```

```
589:     // Pin3 Slew Rate Limitation: Off
590:     PORTE.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
591:     // Pin4 Output/Pull configuration: Totempole/No
592:     // Pin4 Input/Sense configuration: Sense both edges
593:     // Pin4 Inverted: Off
594:     // Pin4 Slew Rate Limitation: Off
595:     PORTE.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
596:     // Pin5 Output/Pull configuration: Totempole/No
597:     // Pin5 Input/Sense configuration: Sense both edges
598:     // Pin5 Inverted: Off
599:     // Pin5 Slew Rate Limitation: Off
600:     PORTE.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
601:     // Pin6 Output/Pull configuration: Totempole/No
602:     // Pin6 Input/Sense configuration: Sense both edges
603:     // Pin6 Inverted: Off
604:     // Pin6 Slew Rate Limitation: Off
605:     PORTE.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
606:     // Pin7 Output/Pull configuration: Totempole/No
607:     // Pin7 Input/Sense configuration: Sense both edges
608:     // Pin7 Inverted: Off
609:     // Pin7 Slew Rate Limitation: Off
610:     PORTE.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
611:     // Interrupt 0 level: Disabled
612:     // Interrupt 1 level: Disabled
613:     PORTE.INTCTRL=(PORTE.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
614:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
615:     // Pin0 Pin Change interrupt 0: Off
616:     // Pin1 Pin Change interrupt 0: Off
617:     // Pin2 Pin Change interrupt 0: Off
618:     // Pin3 Pin Change interrupt 0: Off
619:     // Pin4 Pin Change interrupt 0: Off
620:     // Pin5 Pin Change interrupt 0: Off
621:     // Pin6 Pin Change interrupt 0: Off
622:     // Pin7 Pin Change interrupt 0: Off
623:     PORTE.INT0MASK=0x00;
624:     // Pin0 Pin Change interrupt 1: Off
625:     // Pin1 Pin Change interrupt 1: Off
626:     // Pin2 Pin Change interrupt 1: Off
627:     // Pin3 Pin Change interrupt 1: Off
628:     // Pin4 Pin Change interrupt 1: Off
629:     // Pin5 Pin Change interrupt 1: Off
630:     // Pin6 Pin Change interrupt 1: Off
```

```
631:     // Pin7 Pin Change interrupt 1: Off
632:     PORTE.INT1MASK=0x00;
633:
634:     // PORTF initialization
635:     // OUT register
636:     PORTF.OUT=0x08;
637:     // Pin0: Input
638:     // Pin1: Input
639:     // Pin2: Input
640:     // Pin3: Output
641:     // Pin4: Input
642:     // Pin5: Input
643:     // Pin6: Input
644:     // Pin7: Input
645:     PORTF.DIR=0x08;
646:     // Pin0 Output/Pull configuration: Totempole/No
647:     // Pin0 Input/Sense configuration: Sense both edges
648:     // Pin0 Inverted: Off
649:     // Pin0 Slew Rate Limitation: Off
650:     PORTF.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
651:     // Pin1 Output/Pull configuration: Totempole/No
652:     // Pin1 Input/Sense configuration: Sense both edges
653:     // Pin1 Inverted: Off
654:     // Pin1 Slew Rate Limitation: Off
655:     PORTF.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
656:     // Pin2 Output/Pull configuration: Totempole/No
657:     // Pin2 Input/Sense configuration: Sense both edges
658:     // Pin2 Inverted: Off
659:     // Pin2 Slew Rate Limitation: Off
660:     PORTF.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
661:     // Pin3 Output/Pull configuration: Totempole/No
662:     // Pin3 Input/Sense configuration: Sense both edges
663:     // Pin3 Inverted: Off
664:     // Pin3 Slew Rate Limitation: Off
665:     PORTF.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
666:     // Pin4 Output/Pull configuration: Totempole/No
667:     // Pin4 Input/Sense configuration: Sense both edges
668:     // Pin4 Inverted: Off
669:     // Pin4 Slew Rate Limitation: Off
670:     PORTF.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
671:     // Pin5 Output/Pull configuration: Totempole/No
672:     // Pin5 Input/Sense configuration: Sense both edges
```

```
673:     // Pin5 Inverted: Off
674:     // Pin5 Slew Rate Limitation: Off
675:     PORTF.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
676:     // Pin6 Output/Pull configuration: Totempole/No
677:     // Pin6 Input/Sense configuration: Sense both edges
678:     // Pin6 Inverted: Off
679:     // Pin6 Slew Rate Limitation: Off
680:     PORTF.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
681:     // Pin7 Output/Pull configuration: Totempole/No
682:     // Pin7 Input/Sense configuration: Sense both edges
683:     // Pin7 Inverted: Off
684:     // Pin7 Slew Rate Limitation: Off
685:     PORTF.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
686:     // Interrupt 0 level: Disabled
687:     // Interrupt 1 level: Disabled
688:     PORTF.INTCTRL=(PORTF.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
689:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
690:     // Pin0 Pin Change interrupt 0: Off
691:     // Pin1 Pin Change interrupt 0: Off
692:     // Pin2 Pin Change interrupt 0: Off
693:     // Pin3 Pin Change interrupt 0: Off
694:     // Pin4 Pin Change interrupt 0: Off
695:     // Pin5 Pin Change interrupt 0: Off
696:     // Pin6 Pin Change interrupt 0: Off
697:     // Pin7 Pin Change interrupt 0: Off
698:     PORTF.INT0MASK=0x00;
699:     // Pin0 Pin Change interrupt 1: Off
700:     // Pin1 Pin Change interrupt 1: Off
701:     // Pin2 Pin Change interrupt 1: Off
702:     // Pin3 Pin Change interrupt 1: Off
703:     // Pin4 Pin Change interrupt 1: Off
704:     // Pin5 Pin Change interrupt 1: Off
705:     // Pin6 Pin Change interrupt 1: Off
706:     // Pin7 Pin Change interrupt 1: Off
707:     PORTF.INT1MASK=0x00;
708:
709:     // PORTH initialization
710:     // OUT register
711:     PORTH.OUT=0x00;
712:     // Pin0: Input
713:     // Pin1: Input
714:     // Pin2: Input
```

```
715:     // Pin3: Input
716:     // Pin4: Input
717:     // Pin5: Input
718:     // Pin6: Input
719:     // Pin7: Input
720:     PORTH.DIR=0x00;
721:     // Pin0 Output/Pull configuration: Totempole/No
722:     // Pin0 Input/Sense configuration: Sense both edges
723:     // Pin0 Inverted: Off
724:     // Pin0 Slew Rate Limitation: Off
725:     PORTH.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
726:     // Pin1 Output/Pull configuration: Totempole/No
727:     // Pin1 Input/Sense configuration: Sense both edges
728:     // Pin1 Inverted: Off
729:     // Pin1 Slew Rate Limitation: Off
730:     PORTH.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
731:     // Pin2 Output/Pull configuration: Totempole/No
732:     // Pin2 Input/Sense configuration: Sense both edges
733:     // Pin2 Inverted: Off
734:     // Pin2 Slew Rate Limitation: Off
735:     PORTH.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
736:     // Pin3 Output/Pull configuration: Totempole/No
737:     // Pin3 Input/Sense configuration: Sense both edges
738:     // Pin3 Inverted: Off
739:     // Pin3 Slew Rate Limitation: Off
740:     PORTH.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
741:     // Pin4 Output/Pull configuration: Totempole/No
742:     // Pin4 Input/Sense configuration: Sense both edges
743:     // Pin4 Inverted: Off
744:     // Pin4 Slew Rate Limitation: Off
745:     PORTH.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
746:     // Pin5 Output/Pull configuration: Totempole/No
747:     // Pin5 Input/Sense configuration: Sense both edges
748:     // Pin5 Inverted: Off
749:     // Pin5 Slew Rate Limitation: Off
750:     PORTH.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
751:     // Pin6 Output/Pull configuration: Totempole/No
752:     // Pin6 Input/Sense configuration: Sense both edges
753:     // Pin6 Inverted: Off
754:     // Pin6 Slew Rate Limitation: Off
755:     PORTH.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
756:     // Pin7 Output/Pull configuration: Totempole/No
```

```
757:     // Pin7 Input/Sense configuration: Sense both edges
758:     // Pin7 Inverted: Off
759:     // Pin7 Slew Rate Limitation: Off
760:     PORTH.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
761:     // Interrupt 0 level: Disabled
762:     // Interrupt 1 level: Disabled
763:     PORTH.INTCTRL=(PORTH.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
764:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
765:     // Pin0 Pin Change interrupt 0: Off
766:     // Pin1 Pin Change interrupt 0: Off
767:     // Pin2 Pin Change interrupt 0: Off
768:     // Pin3 Pin Change interrupt 0: Off
769:     // Pin4 Pin Change interrupt 0: Off
770:     // Pin5 Pin Change interrupt 0: Off
771:     // Pin6 Pin Change interrupt 0: Off
772:     // Pin7 Pin Change interrupt 0: Off
773:     PORTH.INT0MASK=0x00;
774:     // Pin0 Pin Change interrupt 1: Off
775:     // Pin1 Pin Change interrupt 1: Off
776:     // Pin2 Pin Change interrupt 1: Off
777:     // Pin3 Pin Change interrupt 1: Off
778:     // Pin4 Pin Change interrupt 1: Off
779:     // Pin5 Pin Change interrupt 1: Off
780:     // Pin6 Pin Change interrupt 1: Off
781:     // Pin7 Pin Change interrupt 1: Off
782:     PORTH.INT1MASK=0x00;
783:
784:     // PORTJ initialization
785:     // OUT register
786:     PORTJ.OUT=0x00;
787:     // Pin0: Input
788:     // Pin1: Input
789:     // Pin2: Input
790:     // Pin3: Input
791:     // Pin4: Input
792:     // Pin5: Input
793:     // Pin6: Input
794:     // Pin7: Input
795:     PORTJ.DIR=0x00;
796:     // Pin0 Output/Pull configuration: Totempole/No
797:     // Pin0 Input/Sense configuration: Sense both edges
798:     // Pin0 Inverted: Off
```

```
799:     // Pin0 Slew Rate Limitation: Off
800:     PORTJ.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
801:     // Pin1 Output/Pull configuration: Totempole/No
802:     // Pin1 Input/Sense configuration: Sense both edges
803:     // Pin1 Inverted: Off
804:     // Pin1 Slew Rate Limitation: Off
805:     PORTJ.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
806:     // Pin2 Output/Pull configuration: Totempole/No
807:     // Pin2 Input/Sense configuration: Sense both edges
808:     // Pin2 Inverted: Off
809:     // Pin2 Slew Rate Limitation: Off
810:     PORTJ.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
811:     // Pin3 Output/Pull configuration: Totempole/No
812:     // Pin3 Input/Sense configuration: Sense both edges
813:     // Pin3 Inverted: Off
814:     // Pin3 Slew Rate Limitation: Off
815:     PORTJ.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
816:     // Pin4 Output/Pull configuration: Totempole/No
817:     // Pin4 Input/Sense configuration: Sense both edges
818:     // Pin4 Inverted: Off
819:     // Pin4 Slew Rate Limitation: Off
820:     PORTJ.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
821:     // Pin5 Output/Pull configuration: Totempole/No
822:     // Pin5 Input/Sense configuration: Sense both edges
823:     // Pin5 Inverted: Off
824:     // Pin5 Slew Rate Limitation: Off
825:     PORTJ.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
826:     // Pin6 Output/Pull configuration: Totempole/No
827:     // Pin6 Input/Sense configuration: Sense both edges
828:     // Pin6 Inverted: Off
829:     // Pin6 Slew Rate Limitation: Off
830:     PORTJ.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
831:     // Pin7 Output/Pull configuration: Totempole/No
832:     // Pin7 Input/Sense configuration: Sense both edges
833:     // Pin7 Inverted: Off
834:     // Pin7 Slew Rate Limitation: Off
835:     PORTJ.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
836:     // Interrupt 0 level: Disabled
837:     // Interrupt 1 level: Disabled
838:     PORTJ.INTCTRL=(PORTJ.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
839:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
840:     // Pin0 Pin Change interrupt 0: Off
```

```
841:     // Pin1 Pin Change interrupt 0: Off
842:     // Pin2 Pin Change interrupt 0: Off
843:     // Pin3 Pin Change interrupt 0: Off
844:     // Pin4 Pin Change interrupt 0: Off
845:     // Pin5 Pin Change interrupt 0: Off
846:     // Pin6 Pin Change interrupt 0: Off
847:     // Pin7 Pin Change interrupt 0: Off
848:     PORTJ.INT0MASK=0x00;
849:     // Pin0 Pin Change interrupt 1: Off
850:     // Pin1 Pin Change interrupt 1: Off
851:     // Pin2 Pin Change interrupt 1: Off
852:     // Pin3 Pin Change interrupt 1: Off
853:     // Pin4 Pin Change interrupt 1: Off
854:     // Pin5 Pin Change interrupt 1: Off
855:     // Pin6 Pin Change interrupt 1: Off
856:     // Pin7 Pin Change interrupt 1: Off
857:     PORTJ.INT1MASK=0x00;
858:
859:     // PORTK initialization
860:     // OUT register
861:     PORTK.OUT=0x00;
862:     // Pin0: Input
863:     // Pin1: Input
864:     // Pin2: Input
865:     // Pin3: Input
866:     // Pin4: Input
867:     // Pin5: Input
868:     // Pin6: Input
869:     // Pin7: Input
870:     PORTK.DIR=0x00;
871:     // Pin0 Output/Pull configuration: Totempole/No
872:     // Pin0 Input/Sense configuration: Sense both edges
873:     // Pin0 Inverted: Off
874:     // Pin0 Slew Rate Limitation: Off
875:     PORTK.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
876:     // Pin1 Output/Pull configuration: Totempole/No
877:     // Pin1 Input/Sense configuration: Sense both edges
878:     // Pin1 Inverted: Off
879:     // Pin1 Slew Rate Limitation: Off
880:     PORTK.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
881:     // Pin2 Output/Pull configuration: Totempole/No
882:     // Pin2 Input/Sense configuration: Sense both edges
```

```
883:     // Pin2 Inverted: Off
884:     // Pin2 Slew Rate Limitation: Off
885:     PORTK.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
886:     // Pin3 Output/Pull configuration: Totempole/No
887:     // Pin3 Input/Sense configuration: Sense both edges
888:     // Pin3 Inverted: Off
889:     // Pin3 Slew Rate Limitation: Off
890:     PORTK.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
891:     // Pin4 Output/Pull configuration: Totempole/No
892:     // Pin4 Input/Sense configuration: Sense both edges
893:     // Pin4 Inverted: Off
894:     // Pin4 Slew Rate Limitation: Off
895:     PORTK.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
896:     // Pin5 Output/Pull configuration: Totempole/No
897:     // Pin5 Input/Sense configuration: Sense both edges
898:     // Pin5 Inverted: Off
899:     // Pin5 Slew Rate Limitation: Off
900:     PORTK.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
901:     // Pin6 Output/Pull configuration: Totempole/No
902:     // Pin6 Input/Sense configuration: Sense both edges
903:     // Pin6 Inverted: Off
904:     // Pin6 Slew Rate Limitation: Off
905:     PORTK.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
906:     // Pin7 Output/Pull configuration: Totempole/No
907:     // Pin7 Input/Sense configuration: Sense both edges
908:     // Pin7 Inverted: Off
909:     // Pin7 Slew Rate Limitation: Off
910:     PORTK.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
911:     // Interrupt 0 level: Disabled
912:     // Interrupt 1 level: Disabled
913:     PORTK.INTCTRL=(PORTK.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
914:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
915:     // Pin0 Pin Change interrupt 0: Off
916:     // Pin1 Pin Change interrupt 0: Off
917:     // Pin2 Pin Change interrupt 0: Off
918:     // Pin3 Pin Change interrupt 0: Off
919:     // Pin4 Pin Change interrupt 0: Off
920:     // Pin5 Pin Change interrupt 0: Off
921:     // Pin6 Pin Change interrupt 0: Off
922:     // Pin7 Pin Change interrupt 0: Off
923:     PORTK.INT0MASK=0x00;
924:     // Pin0 Pin Change interrupt 1: Off
```

```
925:     // Pin1 Pin Change interrupt 1: Off
926:     // Pin2 Pin Change interrupt 1: Off
927:     // Pin3 Pin Change interrupt 1: Off
928:     // Pin4 Pin Change interrupt 1: Off
929:     // Pin5 Pin Change interrupt 1: Off
930:     // Pin6 Pin Change interrupt 1: Off
931:     // Pin7 Pin Change interrupt 1: Off
932:     PORTK.INT1MASK=0x00;
933:
934:     // PORTQ initialization
935:     // OUT register
936:     PORTQ.OUT=0x00;
937:     // Pin0: Input
938:     // Pin1: Input
939:     // Pin2: Input
940:     // Pin3: Input
941:     PORTQ.DIR=0x00;
942:     // Pin0 Output/Pull configuration: Totempole/No
943:     // Pin0 Input/Sense configuration: Sense both edges
944:     // Pin0 Inverted: Off
945:     // Pin0 Slew Rate Limitation: Off
946:     PORTQ.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
947:     // Pin1 Output/Pull configuration: Totempole/No
948:     // Pin1 Input/Sense configuration: Sense both edges
949:     // Pin1 Inverted: Off
950:     // Pin1 Slew Rate Limitation: Off
951:     PORTQ.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
952:     // Pin2 Output/Pull configuration: Totempole/No
953:     // Pin2 Input/Sense configuration: Sense both edges
954:     // Pin2 Inverted: Off
955:     // Pin2 Slew Rate Limitation: Off
956:     PORTQ.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
957:     // Pin3 Output/Pull configuration: Totempole/No
958:     // Pin3 Input/Sense configuration: Sense both edges
959:     // Pin3 Inverted: Off
960:     // Pin3 Slew Rate Limitation: Off
961:     PORTQ.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
962:     // Interrupt 0 level: Disabled
963:     // Interrupt 1 level: Disabled
964:     PORTQ.INTCTRL=(PORTQ.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
965:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
966:     // Pin0 Pin Change interrupt 0: Off
```

```
967:     // Pin1 Pin Change interrupt 0: Off
968:     // Pin2 Pin Change interrupt 0: Off
969:     // Pin3 Pin Change interrupt 0: Off
970:     PORTQ.INT0MASK=0x00;
971:     // Pin0 Pin Change interrupt 1: Off
972:     // Pin1 Pin Change interrupt 1: Off
973:     // Pin2 Pin Change interrupt 1: Off
974:     // Pin3 Pin Change interrupt 1: Off
975:     PORTQ.INT1MASK=0x00;
976:
977:     // PORTR initialization
978:     // OUT register
979:     PORTR.OUT=0x00;
980:     // Pin0: Input
981:     // Pin1: Input
982:     PORTR.DIR=0x00;
983:     // Pin0 Output/Pull configuration: Totempole/No
984:     // Pin0 Input/Sense configuration: Sense both edges
985:     // Pin0 Inverted: Off
986:     // Pin0 Slew Rate Limitation: Off
987:     PORTR.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
988:     // Pin1 Output/Pull configuration: Totempole/No
989:     // Pin1 Input/Sense configuration: Sense both edges
990:     // Pin1 Inverted: Off
991:     // Pin1 Slew Rate Limitation: Off
992:     PORTR.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
993:     // Interrupt 0 level: Disabled
994:     // Interrupt 1 level: Disabled
995:     PORTR.INTCTRL=(PORTR.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
996:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
997:     // Pin0 Pin Change interrupt 0: Off
998:     // Pin1 Pin Change interrupt 0: Off
999:     PORTR.INT0MASK=0x00;
1000:     // Pin0 Pin Change interrupt 1: Off
1001:     // Pin1 Pin Change interrupt 1: Off
1002:     PORTR.INT1MASK=0x00;
1003: }
1004:
1005: // Virtual Ports initialization
1006: void vports_init(void)
1007: {
1008:     // PORTA mapped to VPORT0
```

```
1009:     // PORTB mapped to VPORT1
1010:     PORTCFG.VPCTRLA=PORTCFG_VP1MAP_PORTB_gc | PORTCFG_VP0MAP_PORTA_gc;
1011:     // PORTC mapped to VPORT2
1012:     // PORTD mapped to VPORT3
1013:     PORTCFG.VPCTRLB=PORTCFG_VP3MAP_PORTD_gc | PORTCFG_VP2MAP_PORTC_gc;
1014: }
1015:
1016: // Disable a Timer/Counter type 1
1017: void tc1_disable(TC1_t *ptc)
1018: {
1019:     // Timer/Counter off
1020:     ptc->CTRLA=(ptc->CTRLA & (~TC1_CLKSEL_gm)) | TC_CLKSEL_OFF_gc;
1021:     // Issue a reset command
1022:     ptc->CTRLFSET=TC_CMD_RESET_gc;
1023: }
1024:
1025: // Timer/Counter TCD1 initialization
1026: void tcd1_init(void)
1027: {
1028:     unsigned char s;
1029:
1030:     // Note: The correct PORTD direction for the Compare Channels
1031:     // outputs is configured in the ports_init function.
1032:
1033:     // Save interrupts enabled/disabled state
1034:     s=SREG;
1035:     // Disable interrupts
1036:     #asm("cli")
1037:
1038:     // Disable and reset the timer/counter just to be sure
1039:     tc1_disable(&TCD1);
1040:     // Clock source: ClkPer/1
1041:     TCD1.CTRLA=(TCD1.CTRLA & (~TC1_CLKSEL_gm)) | TC_CLKSEL_DIV1_gc;
1042:     // Mode: Normal Operation, Overflow Int./Event on TOP
1043:     // Compare/Capture on channel A: Off
1044:     // Compare/Capture on channel B: Off
1045:     TCD1.CTRLB=(TCD1.CTRLB & (~(TC1_CCAEN_bm | TC1_CCBEN_bm | TC1_WGMODE_gm))) |
1046:         TC_WGMODE_NORMAL_gc;
1047:
1048:     // Capture event source: None
1049:     // Capture event action: None
1050:     TCD1.CTRLD=(TCD1.CTRLD & (~(TC1_EVACT_gm | TC1_EVSEL_gm))) |
```

```
1051:         TC_EVACT_OFF_gc | TC_EVSEL_OFF_gc;
1052:
1053:     // Overflow interrupt: High Level
1054:     // Error interrupt: Disabled
1055:     TCD1.INTCTRLA=(TCD1.INTCTRLA & (~(TC1_ERRINTLVL_gm | TC1_OVFINTLVL_gm))) |
1056:         TC_ERRINTLVL_OFF_gc | TC_OVFINTLVL_HI_gc;
1057:
1058:     // Compare/Capture channel A interrupt: Disabled
1059:     // Compare/Capture channel B interrupt: Disabled
1060:     TCD1.INTCTRLB=(TCD1.INTCTRLB & (~(TC1_CCBINTLVL_gm | TC1_CCAINTLVL_gm))) |
1061:         TC_CCBINTLVL_OFF_gc | TC_CCAINTLVL_OFF_gc;
1062:
1063:     // High resolution extension: Off
1064:     HIRESD.CTRLA&= ~HIRES_HREN1_bm;
1065:
1066:     // Clear the interrupt flags
1067:     TCD1.INTFLAGS=TCD1.INTFLAGS;
1068:     // Set counter register
1069:     TCD1.CNT=0x0000;
1070:     // Set period register
1071:     TCD1.PER=0x7CFF;
1072:     // Set channel A Compare/Capture register
1073:     TCD1.CCA=0x0000;
1074:     // Set channel B Compare/Capture register
1075:     TCD1.CCB=0x0000;
1076:
1077:     // Restore interrupts enabled/disabled state
1078:     SREG=s;
1079: }
1080:
1081: // USARTF0 initialization
1082: void usartf0_init(void)
1083: {
1084:     // Note: The correct PORTF direction for the RxD, TxD and XCK signals
1085:     // is configured in the ports_init function.
1086:
1087:     // Transmitter is enabled
1088:     // Set TxD=1
1089:     PORTF.OUTSET=0x08;
1090:
1091:     // Communication mode: Asynchronous USART
1092:     // Data bits: 8
```

```
1093:     // Stop bits: 1
1094:     // Parity: Disabled
1095:     USARTF0.CTRLC=USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc | USART_CHSIZE_8BIT_gc;
1096:
1097:     // Receive complete interrupt: Disabled
1098:     // Transmit complete interrupt: Disabled
1099:     // Data register empty interrupt: Disabled
1100:     USARTF0.CTRLA=(USARTF0.CTRLA & (~(USART_RXCINTLVL_gm | USART_TXCINTLVL_gm | USART_DREINTLVL_gm))) |
1101:         USART_RXCINTLVL_OFF_gc | USART_TXCINTLVL_OFF_gc | USART_DREINTLVL_OFF_gc;
1102:
1103:     // Required Baud rate: 115200
1104:     // Real Baud Rate: 115211,5 (x1 Mode), Error: 0,0 %
1105:     USARTF0.BAUDCTRLA=0x2E;
1106:     USARTF0.BAUDCTRLB=((0x09 << USART_BSCALE_gp) & USART_BSCALE_gm) | 0x08;
1107:
1108:     // Receiver: On
1109:     // Transmitter: On
1110:     // Double transmission speed mode: Off
1111:     // Multi-processor communication mode: Off
1112:     USARTF0.CTRLB=(USARTF0.CTRLB & (~(USART_RXEN_bm | USART_TXEN_bm | USART_CLK2X_bm | USART_MPCM_bm | USART_TXB8_bm))) |
1113:         USART_RXEN_bm | USART_TXEN_bm;
1114: }
1115:
1116: // Receive a character from USARTF0
1117: // USARTF0 is used as the default input device by the 'getchar' function
1118: #define _ALTERNATE_GETCHAR_
1119:
1120: #pragma used+
1121: char getchar(void)
1122: {
1123:     char data;
1124:     unsigned char status;
1125:
1126:     while (1)
1127:     {
1128:       while (((status=USARTF0.STATUS) & USART_RXCIF_bm) == 0);
1129:       data=USARTF0.DATA;
1130:       if ((status & (USART_FERR_bm | USART_PERR_bm | USART_BUFOVF_bm)) == 0) return data;
1131:     }
1132: }
1133: #pragma used-
1134:
```

```
1135: // Write a character to the USARTF0 Transmitter
1136: // USARTF0 is used as the default output device by the 'putchar' function
1137: #define _ALTERNATE_PUTCHAR_
1138:
1139: #pragma used+
1140: void putchar(char c)
1141: {
1142:     while ((USARTF0.STATUS & USART_DREIF_bm) == 0);
1143:     USARTF0.DATA=c;
1144: }
1145: #pragma used-
1146:
1147: // TWIC initialization
1148: // structure that holds information used by the TWIC Master
1149: // for performing a TWI bus transaction
1150: TWI_MASTER_INFO_t twic_master;
1151:
1152: void twic_init(void)
1153: {
1154:     // General TWIC initialization
1155:     // External Driver Interface: Off
1156:     // SDA Hold: Off
1157:     twi_init(&TWIC,false,0);
1158:
1159:     // TWIC Master initialization
1160:     // Master interrupt: Low Level
1161:     // Peripheral Clock frequency: 32000000 Hz
1162:     // SCL Rate: 100000 bps
1163:     // Real SCL Rate: 100000 bps, Error: 0,0 %
1164:     twi_master_init(&twic_master,&TWIC,TWI_MASTER_INTLVL_LO_gc,
1165:         TWI_BAUD_REG(32000000,100000));
1166:
1167:     // TWIC Slave is disabled
1168:     TWIC.SLAVE.CTRLA=0;
1169: }
1170:
1171: // TWIC Master interrupt service routine
1172: #pragma optsize- // optimize for speed
1173: interrupt [TWIC_TWIM_vect] void twic_master_isr(void)
1174: {
1175:     twi_master_int_handler(&twic_master);
1176: }
```

```
1177: #pragma optsize_default
1178:
1179: // Timer/Counter TCD1 Overflow/Underflow interrupt service routine
1180: interrupt [TCD1_OVF_vect] void tcd1_overflow_isr(void)
1181: {
1182:     relogio++;  //Counts milliseconds (49 days capacity)
1183: }
1184:
1185: //*****************************************************
1186:
1187:
1188: void main(void)
1189: {
1190:
1191: //**********************SPAOFE*************************
1192:
1193:     struct sonar SPAOFE =
1194:     {
1195:         .Sonar_Number = N_Sonar,
1196:         .Sonar_Addr = {0xE0,0xE2,0xE4,0xE6,0xE8,0xEA,0xEC,0xEE},
1197:         .Sonar_Angle = {90.0,45.0,0.0,315.0,270.0,225.0,180.0,135.0},
1198:         .Maximum_Acceleration = 10.0,
1199:         .Velocity = 15,
1200:         .Maximum_Velocity = 40.0,
1201:         .Minimum_Velocity = 4.0,
1202:         .Maximum_Distance = 400.0,
1203:         .Minimum_Distance = 30.0,
1204:         .Sonar_Distance = {infinito,infinito,infinito,infinito,infinito,infinito,infinito,infinito},
1205:         .Ti_1 = {0,0,0,0,0,0,0,0}
1206:     };
1207:
1208:     unsigned char i;
1209:     unsigned char sim;
1210:     unsigned int temp1;
1211:     unsigned long int temp2;
1212:
1213: //**********************SPAOFE*************************
1214:
1215:     unsigned char n;
1216:     // Interrupt system initialization
1217:     // Optimize for speed
1218:     #pragma optsize-
```

```
1219:     // Make sure the interrupts are disabled
1220:     #asm("cli")
1221:     // Low level interrupt: On
1222:     // Round-robin scheduling for low level interrupt: On
1223:     // Medium level interrupt: On
1224:     // High level interrupt: On
1225:     // The interrupt vectors will be placed at the start of the Application FLASH section
1226:     n=(PMIC.CTRL & (~(PMIC_RREN_bm | PMIC_IVSEL_bm | PMIC_HILVLEN_bm | PMIC_MEDLVLEN_bm | PMIC_LOLVLEN_bm))) |
1227:         PMIC_LOLVLEN_bm | PMIC_RREN_bm | PMIC_MEDLVLEN_bm | PMIC_HILVLEN_bm;
1228:     CCP=CCP_IOREG_gc;
1229:     PMIC.CTRL=n;
1230:     // Set the default priority for round-robin scheduling
1231:     PMIC.INTPRI=0x00;
1232:     // Restore optimization for size if needed
1233:     #pragma optsize_default
1234:
1235:     // System clocks initialization
1236:     system_clocks_init();
1237:
1238:     // Ports initialization
1239:     ports_init();
1240:
1241:     // Virtual Ports initialization
1242:     vports_init();
1243:
1244:     // Timer/Counter TCD1 initialization
1245:     tcd1_init();
1246:
1247:     // USARTF0 initialization
1248:     usartf0_init();
1249:
1250:     // TWIC initialization
1251:     twic_init();
1252:
1253:     // Globally enable interrupts
1254:     #asm("sei")
1255:
1256: //**********************SPAOFE*************************
1257:
1258:     temp2 = relogio;
1259:     for(i=0; i<SPAOFE.Sonar_Number; i++)
1260:     {
```

```
1261:         SPAOFE.Ti_1[i] = temp2;
1262:     }
1263:
1264:     // Each "sim" simulates a ranging set of the sensors
1265:     for(sim=0; sim<4; sim++)
1266:     {
1267:         for(i=0; i<SPAOFE.Sonar_Number; i++)        // For each sensor
1268:         {
1269:             temp1 = range_sonar_X(sim,i,SPAOFE);
1270:             temp2 = relogio;    // Instant time
1271:             SPAOFE.DTi[i] = temp2 - SPAOFE.Ti_1[i];    // Elapsed time
1272:             SPAOFE.Ti_1[i] = temp2;    // New inicial instant time
1273:             SPAOFE.Sonar_Distance_Change[i] = temp1 - SPAOFE.Sonar_Distance[i];    // Distance change
1274:             SPAOFE.Sonar_Distance[i] = temp1;    // New inicial distance
1275:         }
1276:         General_Perception_Vector(SPAOFE);
1277:         General_Perception_V_Change(SPAOFE);
1278:
1279:             ftoa(SPAOFE.General_Perception_Value,4,CharConv);
1280:             printf("General_Perception_Value = %s\r\n", CharConv);
1281:             ftoa(SPAOFE.General_Perception_Angle,4,CharConv);
1282:             printf("General_Perception_Angle = %s\r\n", CharConv);
1283:             ftoa(SPAOFE.General_Perception_Change,4,CharConv);
1284:             printf("General_Perception_Change = %s\r\n",CharConv);
1285:
1286:         Perception_Value_Membership(SPAOFE.General_Perception_Value);
1287:
1288:             ftoa(VLP,4,CharConv);
1289:             printf("VLP = %s\r\n", CharConv);
1290:             ftoa(LP,4,CharConv);
1291:             printf("LP = %s\r\n", CharConv);
1292:             ftoa(MP,4,CharConv);
1293:             printf("MP = %s\r\n", CharConv);
1294:             ftoa(HP,4,CharConv);
1295:             printf("HP = %s\r\n", CharConv);
1296:             ftoa(VHP,4,CharConv);
1297:             printf("VHP = %s\r\n", CharConv);
1298:
1299:         Msectors_Nao_Zero(&PV[0],5,&iPV[0],&nPV);
1300:
1301:             printf("nPV = %d\r\n", nPV);
1302:             printf("%d %d %d %d %d\r\n",iPV[0],iPV[1],iPV[2],iPV[3],iPV[4]);
```

```
1303:
1304:         Perception_Angle_Membership(SPAOFE.General_Perception_Angle);
1305:
1306:             ftoa(RB,4,CharConv);
1307:             printf("RB = %s\r\n", CharConv);
1308:             ftoa(RF,4,CharConv);
1309:             printf("RF = %s\r\n", CharConv);
1310:             ftoa(LF,4,CharConv);
1311:             printf("LF = %s\r\n", CharConv);
1312:             ftoa(LB,4,CharConv);
1313:             printf("LB = %s\r\n", CharConv);
1314:
1315:         Msectors_Nao_Zero(&PA[0],4,&iPA[0],&nPA);
1316:
1317:             printf("nPA = %d\r\n", nPA);
1318:             printf("%d %d %d %d\r\n",iPA[0],iPA[1],iPA[2],iPA[3]);
1319:
1320:         Perception_Change_Membership(SPAOFE.General_Perception_Change);
1321:
1322:             ftoa(ZE,4,CharConv);
1323:             printf("ZE = %s\r\n", CharConv);
1324:             ftoa(LO,4,CharConv);
1325:             printf("LO = %s\r\n", CharConv);
1326:             ftoa(HI,4,CharConv);
1327:             printf("HI = %s\r\n", CharConv);
1328:
1329:         Msectors_Nao_Zero(&PC[0],3,&iPC[0],&nPC);
1330:
1331:             printf("nPC = %d\r\n", nPC);
1332:             printf("%d %d %d\r\n",iPC[0],iPC[1],iPC[2]);
1333:
1334:         // The variables needed to use the XMEGA/LM629 library in velocity mode are:
1335:         // New Velocity => SPAOFE.Velocity
1336:         // Direction (left,right or stright) => Direcao
1337:         // Gyration radius (if = 0.0 => turn on vertical axis) => Rg
1338:         // Brake (abruptly or smoothly) => Brake
1339:
1340:         Rules_Steer();
1341:
1342:             printf("nSteer = %d\r\n", nSteer);
1343:             printf("%d  %d  %d  %d  %d  %d\r\n", iSteer[0],iSteer[1],iSteer[2],iSteer[3],iSteer[4],iSteer[5]);
1344:
```

```
1345:             ftoa(CSteer[0],2,CSt0);
1346:             ftoa(CSteer[1],2,CSt1);
1347:             ftoa(CSteer[2],2,CSt2);
1348:             ftoa(CSteer[3],2,CSt3);
1349:             ftoa(CSteer[4],2,CSt4);
1350:             ftoa(CSteer[5],2,CSt5);
1351:
1352:             printf("%s  %s  %s  %s  %s  %s\r\n", CSt0,CSt1,CSt2,CSt3,CSt4,CSt5);
1353:
1354:         Steer_Crisp = Scaled_Defuzzy_Steer(-100.0,100.0,1.0);
1355:
1356:         if (Steer_Crisp == 0.0) Direcao = 0; // Stright
1357:         else if (Steer_Crisp > 0.0) Direcao = -1;   // Right
1358:         else Direcao = +1;  // Left
1359:         Steer_Crisp = fabs(Steer_Crisp);
1360:
1361:         Rg = 57.3 * SPAOFE.Velocity / Steer_Crisp;
1362:         if (Rg < (1.5 * eixo) && Rg >= eixo) Rg = 1.5 * eixo;
1363:         else if (Rg < eixo) Rg = 0.0;  // This signifies that the robot must rotate on its vertical axis
1364:         else;
1365:
1366:         Rules_Acceleration();
1367:
1368:             printf("nAcc = %d\r\n", nAcc);
1369:             printf("%d  %d  %d  %d  %d  %d\r\n", iAcc[0],iAcc[1],iAcc[2],iAcc[3],iAcc[4],iAcc[5]);
1370:
1371:             ftoa(CAcc[0],2,CSt0);
1372:             ftoa(CAcc[1],2,CSt1);
1373:             ftoa(CAcc[2],2,CSt2);
1374:             ftoa(CAcc[3],2,CSt3);
1375:             ftoa(CAcc[4],2,CSt4);
1376:             ftoa(CAcc[5],2,CSt5);
1377:
1378:             printf("%s  %s  %s  %s  %s  %s\r\n", CSt0,CSt1,CSt2,CSt3,CSt4,CSt5);
1379:
1380:         // New acceleration
1381:         Acceleration_Crisp = Scaled_Defuzzy_Acceleration(-SPAOFE.Maximum_Acceleration,SPAOFE.Maximum_Acceleration,0.5,
      SPAOFE);
1382:
1383:         // New velocity
1384:         SPAOFE.Velocity = SPAOFE.Velocity + Acceleration_Crisp;
1385:
```

```
1386:         if (SPAOFE.Velocity > SPAOFE.Maximum_Velocity) SPAOFE.Velocity = SPAOFE.Maximum_Velocity;
1387:         else if (SPAOFE.Velocity < SPAOFE.Minimum_Velocity)
1388:         {
1389:             SPAOFE.Velocity = 0.0;
1390:             Brake = 10; // Stop smoothly
1391:         }
1392:         if (Acceleration_Crisp < -(0.875 * SPAOFE.Maximum_Acceleration))
1393:         {
1394:             SPAOFE.Velocity = 0.0;
1395:             Brake = 9;  // Stop abruptly
1396:         }
1397:
1398:             ftoa(Steer_Crisp,2,CharConv);
1399:             printf("Steer_Crisp = %s\r\n",CharConv);
1400:             ftoa(Acceleration_Crisp,2,CharConv);
1401:             printf("Acceleration_Crisp = %s\r\n",CharConv);
1402:             printf("Direction = %d\r\n",Direcao);
1403:             ftoa(Rg,2,CharConv);
1404:             printf("Rg = %s\r\n",CharConv);
1405:             ftoa(SPAOFE.Velocity,2,CharConv);
1406:             printf("New velocity = %s\r\n",CharConv);
1407:             printf("Brake = %d\r\n",Brake);
1408:
1409:         delay_ms(2000);
1410:     }
1411:
1412: //**********************SPAOFE*************************
1413:
1414:     ad_aeternum;
1415: }
1416:
1417: //*****************SPAOFE FUNCTIONS*********************
1418:
1419: void Msectors_Nao_Zero(float *Vetor,unsigned char Comp,unsigned char *Indices,unsigned char *Num)
1420: {
1421:     //Verifies the first "Comp" elements of the array "*Vetor" to verify which are non zero.
1422:     //The indices of the non zero elements are stored in the array "*Indices" and the indices
1423:     //quantity is stored in "*Num"
1424:
1425:     unsigned char i;
1426:
1427:     *Num = 0;
```

```
1428:     for(i=0;i<Comp;i++) if (Vetor[i] > 0.0) Indices[*Num++] = i;
1429: }
1430:
1431: //_____________________________________________________________________________________________________
1432:
1433: void Rules_Steer(void)
1434: {
1435:     //Evaluates the 20 Steer rules. As inputs uses the Perception Value and Perception Angle membership sectors.
1436:     //Calculates the Steer sectors to be aggregated and it's cutoff (or scaled) value "CSteer[]".
1437:
1438:     unsigned char i;
1439:     unsigned char j;
1440:     unsigned char Regra;
1441:
1442:     nSteer = 0;
1443:
1444:       for(i=0;i<nPA;i++)                          // For all PA[] > zero
1445:        {
1446:         for(j=0;j<nPV;j++)                       // For all PV[] > zero
1447:          {
1448:               Regra = 10 * iPA[i] + iPV[j];      // Rule
1449:             switch (Regra)
1450:             {
1451:                 case 0:                          // if a is RB and p is VLP then s is HR
1452:                     iSteer[nSteer] = 0;          // iSteer[] = 0 (HR sector index)
1453:                     break;
1454:                 case 1:                          // if a is RB and p is LP then s is HR
1455:                     iSteer[nSteer] = 0;          // iSteer[] = 0 (HR sector index)
1456:                     break;
1457:                 case 2:                          // if a is RB and p is MP then s is R
1458:                     iSteer[nSteer] = 1;          // iSteer[] = 1 (R sector index)
1459:                     break;
1460:                 case 3:                          // if a is RB and p is HP then s is R
1461:                     iSteer[nSteer] = 1;        // iSteer[] = 1 (R sector index)
1462:                     break;
1463:                 case 4:                          // if a is RB and p is VHP then s is C
1464:                     iSteer[nSteer] = 2;        // iSteer[] = 2 (C sector index)
1465:                     break;
1466:                 case 10:                       // if a is RF and p is VLP then s is C
1467:                     iSteer[nSteer] = 2;          // iSteer[] = 2 (C sector index)
1468:                     break;
1469:                 case 11:                       // if a is RF and p is LP then s is L
```

```
1470:                 iSteer[nSteer] = 3;           // iSteer[] = 3 (L sector index)
1471:                 break;
1472:             case 12:                          // if a is RF and p is MP then s is L
1473:                 iSteer[nSteer] = 3;           // iSteer[] = 3 (L sector index)
1474:                 break;
1475:             case 13:                          // if a is RF and p is HP then s is HL
1476:                 iSteer[nSteer] = 4;           // iSteer[] = 4 (HL sector index)
1477:                 break;
1478:             case 14:                          // if a is RF and p is VHP then s is HL
1479:                 iSteer[nSteer] = 4;           // iSteer[] = 4 (HL sector index)
1480:                 break;
1481:             case 20:                          // if a is LF and p is VLP then s is C
1482:                 iSteer[nSteer] = 2;           // iSteer[] = 2 (C sector index)
1483:                 break;
1484:             case 21:                          // if a is LF and p is LP then s is R
1485:                 iSteer[nSteer] = 1;           // iSteer[] = 1 (R sector index)
1486:                 break;
1487:             case 22:                          // if a is LF and p is MP then s is R
1488:                 iSteer[nSteer] = 1;           // iSteer[] = 1 (R sector index)
1489:                 break;
1490:             case 23:                          // if a is LF and p is HP then s is HR
1491:                 iSteer[nSteer] = 0;           // iSteer[] = 0 (HR sector index)
1492:                 break;
1493:             case 24:                          // if a is LF and p is VHP then s is HR
1494:                 iSteer[nSteer] = 0;           // iSteer[] = 0 (HR sector index)
1495:                 break;
1496:             case 30:                          // if a is LB and p is VLP then s is HL
1497:                 iSteer[nSteer] = 4;           // iSteer[] = 4 (HL sector index)
1498:                 break;
1499:             case 31:                          // if a is LB and p is LP then s is HL
1500:                 iSteer[nSteer] = 4;           // iSteer[] = 4 (HL sector index)
1501:                 break;
1502:             case 32:                          // if a is LB and p is MP then s is L
1503:                 iSteer[nSteer] = 3;           // iSteer[] = 3 (L sector index)
1504:                 break;
1505:             case 33:                          // if a is LB and p is HP then s is L
1506:                 iSteer[nSteer] = 3;           // iSteer[] = 3 (L sector index)
1507:                 break;
1508:             case 34:                          // if a is LB and p is VHP then s is C
1509:                 iSteer[nSteer] = 2;           // iSteer[] = 2 (C sector index)
1510:                 break;
1511:             default:
```

```
1512:                     iSteer[nSteer] = 100;        // There is no rule. Aggregation will not be applied
1513:             }
1514:             CSteer[nSteer++] = zand(PA[iPA[i]],PV[iPV[j]]);    // Lower value
1515:         }
1516:     }
1517: }
1518:
1519: //_____________________________________________________________________________________________________
1520:
1521: void Rules_Acceleration(void)
1522: {
1523:     //Evaluates the 11 Acceleration rules. As inputs uses the Perception Value and Perception Change membership sectors.
1524:     //Calculates the Acceleration sectors to be aggregated and it's cutoff (or scaled) value "CAcc[]".
1525:
1526:     unsigned char i;
1527:     unsigned char j;
1528:     unsigned char Regra;
1529:
1530:     nAcc = 0;
1531:
1532:     for(i=0;i<nPC;i++)                                  // For all PC[] > zero
1533:      {
1534:           for(j=0;j<nPV;j++)                             // For all PV[] > zero
1535:         {
1536:              Regra = 10 * iPC[i] + iPV[j];
1537:             switch (Regra)
1538:             {
1539:                 case 0:                                 // if dp is ZE and p is VLP then ac is Z
1540:                     iAcc[nAcc] = 2;                     // iAcc[] = 2 (Z sector index)
1541:                     break;
1542:                 case 1:                                 // if dp is ZE and p is LP then ac is P
1543:                     iAcc[nAcc] = 3;                     // iAcc[] = 3 (P sector index)
1544:                     break;
1545:                 case 2:                                 // if dp is ZE and p is MP then ac is P
1546:                     iAcc[nAcc] = 3;                     // iAcc[] = 3 (P sector index)
1547:                     break;
1548:                 case 3:                                 // if dp is ZE and p is HP then ac is P
1549:                     iAcc[nAcc] = 3;                     // iAcc[] = 3 (P sector index)
1550:                     break;
1551:                 case 4:                                 // if dp is ZE and p is VHP then ac is Z
1552:                     iAcc[nAcc] = 2;                     // iAcc[] = 2 (Z sector index)
1553:                     break;
```

```
1554:                 case 10:                              // if dp is LO and p is VLP then ac is EB
1555:                     iAcc[nAcc] = 0;                      // iAcc[] = 0 (EB sector index)
1556:                     break;
1557:                 case 11:                              // if dp is LO and p is LP then ac is B
1558:                     iAcc[nAcc] = 1;                      // iAcc[] = 1 (B sector index)
1559:                     break;
1560:                 case 12:                              // if dp is LO and p is MP then ac is Z
1561:                     iAcc[nAcc] = 2;                      // iAcc[] = 2 (Z sector index)
1562:                     break;
1563:                 case 13:                              // if dp is LO and p is HP then ac is B
1564:                     iAcc[nAcc] = 1;                      // iAcc[] = 1 (B sector index)
1565:                     break;
1566:                 case 14:                              // if dp is LO and p is VHP then ac is EB
1567:                     iAcc[nAcc] = 0;                      // iAcc[] = 0 (EB sector index)
1568:                     break;
1569: //                case 20:                               // if dp is HI and p is VLP then ac is P
1570: //                    iAcc[nAcc] = 3;                      // iAcc[] = 3 (P sector index)
1571: //                    break;
1572: //                case 21:                               // if dp is HI and p is LP then ac is P
1573: //                    iAcc[nAcc] = 3;                      // iAcc[] = 3 (P sector index)
1574: //                    break;
1575: //                case 22:                               // if dp is HI and p is MP then ac is Z
1576: //                    iAcc[nAcc] = 2;                      // iAcc[] = 2 (Z sector index)
1577: //                    break;
1578:                 case 23:                              // if dp is HI and p is HP then ac is EB
1579:                     iAcc[nAcc] = 0;                      // iAcc[] = 0 (EB sector index)
1580:                     break;
1581: //                case 24:                               // if dp is HI and p is VHP then ac is EB
1582: //                    iAcc[nAcc] = 0;                      // iAcc[] = 0 (EB sector index)
1583: //                    break;
1584:                 default:
1585:                     iAcc[nAcc] = 100;                    // There is no rule. Aggregation will not be applied
1586:             }
1587:             CAcc[nAcc++] = zand(PC[iPC[i]],PV[iPV[j]]); // Lower value
1588:         }
1589:     }
1590: }
1591:
1592: //_____________________________________________________________________________________________________________
1593:
1594: float Cutted_Defuzzy_Steer(float Xinicial,float Xfinal,float Step)
1595: {
```

```
1596:     //Calculates the crisp value of the output value Steer. This is done aggregating the
1597:     //cutted or scaled membership sectors and using the centroid method. The aggregation is done
1598:     //by the use of the OR for fuzzy sets (union). The centroid method calculates the
1599:     //gravity center of the resulting area.
1600:
1601:     unsigned char i;
1602:     unsigned char j;
1603:     unsigned int Namos;       // Number of steps "Step" in the interval "Xinicial" to "Xfinal"
1604:     float Fagregado[210];     // Resulting aggregated function
1605:     float X[210];           // X value for each step
1606:     float Snumerador;       // Integral of the momentum approximation
1607:     float Sdenominador;     // Integral of the area approximation
1608:
1609:     Namos = (unsigned int)((Xfinal - Xinicial) / Step);
1610:     X[0] = Xinicial;
1611:     Fagregado[0] = 0.0;
1612:
1613:     for (i=1;i<Namos;i++)
1614:     {
1615:         X[i] = X[i-1] + Step;
1616:         Fagregado[i] = 0.0;
1617:     }
1618:
1619:     for (i=0;i<nSteer;i++)
1620:     {
1621:         switch (iSteer[i])
1622:         {
1623:              case 0:        // HR
1624:                     for (j=0;j<Namos;j++)
1625:                     {
1626:                         Fagregado[j] = zor(Fagregado[j],zand(CSteer[i],fHR(X[j])));
1627:                     }
1628:                     break;
1629:              case 1:        // R
1630:                     for (j=0;j<Namos;j++)
1631:                     {
1632:                         Fagregado[j] = zor(Fagregado[j],zand(CSteer[i],fR(X[j])));
1633:                     }
1634:                     break;
1635:              case 2:        // C
1636:                     for (j=0;j<Namos;j++)
1637:                     {
```

```
1638:                         Fagregado[j] = zor(Fagregado[j],zand(CSteer[i],fC(X[j])));
1639:                     }
1640:                     break;
1641:             case 3:         // L
1642:                     for (j=0;j<Namos;j++)
1643:                     {
1644:                         Fagregado[j] = zor(Fagregado[j],zand(CSteer[i],fL(X[j])));
1645:                     }
1646:                     break;
1647:             case 4:         // HL
1648:                     for (j=0;j<Namos;j++)
1649:                     {
1650:                         Fagregado[j] = zor(Fagregado[j],zand(CSteer[i],fHL(X[j])));
1651:                     }
1652:                     break;
1653:             default:
1654:
1655:         }
1656:     }
1657:     Snumerador = Sdenominador = 0.0;
1658:     for (i=0;i<Namos;i++)
1659:     {
1660:         Snumerador = Snumerador + (Fagregado[i] * X[i]);
1661:         Sdenominador = Sdenominador + Fagregado[i];
1662:     }
1663:     return (Snumerador / Sdenominador);
1664: }
1665:
1666: //_____________________________________________________________________________________________________________
1667:
1668: float Scaled_Defuzzy_Steer(float Xinicial,float Xfinal,float Step)
1669: {
1670:     //Calculates the crisp value of the output value Steer. This is done aggregating the
1671:     //scaled or scaled membership sectors and using the centroid method. The aggregation is done
1672:     //by the use of the OR for fuzzy sets (union). The centroid method calculates the
1673:     //gravity center of the resulting area.
1674:
1675:     unsigned char i;
1676:     unsigned char j;
1677:     unsigned int Namos;       // Number of steps "Step" in the interval "Xinicial" to "Xfinal"
1678:     float Fagregado[210];    // Resulting aggregated function
1679:     float X[210];            // X value for each step
```

```
1680:     float Snumerador;       // Integral of the momentum approximation
1681:     float Sdenominador;       // Integral of the area approximation
1682:
1683:     Namos = (unsigned int)((Xfinal - Xinicial) / Step);    // Número de steps para o X
1684:     X[0] = Xinicial;
1685:     Fagregado[0] = 0.0;
1686:
1687:     for (i=1;i<Namos;i++)
1688:     {
1689:         X[i] = X[i-1] + Step;
1690:         Fagregado[i] = 0.0;
1691:     }
1692:
1693:     for (i=0;i<nSteer;i++)
1694:     {
1695:         switch (iSteer[i])
1696:         {
1697:              case 0:        // HR
1698:                     for (j=0;j<Namos;j++)
1699:                     {
1700:                         Fagregado[j] = zor(Fagregado[j],CSteer[i] * fHR(X[j]));
1701:                     }
1702:                     break;
1703:              case 1:        // R
1704:                     for (j=0;j<Namos;j++)
1705:                     {
1706:                         Fagregado[j] = zor(Fagregado[j],CSteer[i] * fR(X[j]));
1707:                     }
1708:                     break;
1709:              case 2:        // C
1710:                     for (j=0;j<Namos;j++)
1711:                     {
1712:                         Fagregado[j] = zor(Fagregado[j],CSteer[i] * fC(X[j]));
1713:                     }
1714:                     break;
1715:              case 3:        // L
1716:                     for (j=0;j<Namos;j++)
1717:                     {
1718:                         Fagregado[j] = zor(Fagregado[j],CSteer[i] * fL(X[j]));
1719:                     }
1720:                     break;
1721:              case 4:        // HL
```

```
1722:                     for (j=0;j<Namos;j++)
1723:                     {
1724:                         Fagregado[j] = zor(Fagregado[j],CSteer[i] * fHL(X[j]));
1725:                     }
1726:                     break;
1727:                 default:
1728:
1729:             }
1730:         }
1731:         Snumerador = Sdenominador = 0.0;
1732:         for (i=0;i<Namos;i++)
1733:         {
1734:             Snumerador = Snumerador + (Fagregado[i] * X[i]);
1735:             Sdenominador = Sdenominador + Fagregado[i];
1736:         }
1737:         return (Snumerador / Sdenominador); // Centroid - Crisp value for Steer
1738: }
1739:
1740: //_____________________________________________________________________________________________________________
1741:
1742: float Cutted_Defuzzy_Acceleration(float Xinicial,float Xfinal,float Step,struct sonar sensor)
1743: {
1744:     //Calculates the crisp value of the output value Acceleration. This is done aggregating the
1745:     //cutted membership sectors and using the centroid method. The aggregation is done
1746:     //by the use of the OR for fuzzy sets (union). The centroid method calculates the
1747:     //gravity center of the resulting area.
1748:
1749:     unsigned char i;
1750:     unsigned char j;
1751:     unsigned int Namos;     // Number of steps "Step" in the interval "Xinicial" to "Xfinal"
1752:     float Fagregado[210];    // Resulting aggregated function
1753:     float X[210];              // X value for each step
1754:     float Snumerador;         // Integral of the momentum approximation
1755:     float Sdenominador;     // Integral of the area approximation
1756:
1757:     Namos = (unsigned int)((Xfinal - Xinicial) / Step);
1758:     X[0] = Xinicial;
1759:     Fagregado[0] = 0.0;
1760:
1761:     for (i=1;i<Namos;i++)
1762:     {
1763:         X[i] = X[i-1] + Step;
```

```
1764:         Fagregado[i] = 0.0;
1765:     }
1766:
1767:     for (i=0;i<nAcc;i++)
1768:     {
1769:         switch (iAcc[i])
1770:         {
1771:              case 0:        // EB
1772:                     for (j=0;j<Namos;j++)
1773:                     {
1774:                         Fagregado[j] = zor(Fagregado[j],zand(CAcc[i],fEB(X[j],sensor)));
1775:                     }
1776:                     break;
1777:              case 1:        // B
1778:                     for (j=0;j<Namos;j++)
1779:                     {
1780:                         Fagregado[j] = zor(Fagregado[j],zand(CAcc[i],fB(X[j],sensor)));
1781:                     }
1782:                     break;
1783:              case 2:        // Z
1784:                     for (j=0;j<Namos;j++)
1785:                     {
1786:                         Fagregado[j] = zor(Fagregado[j],zand(CAcc[i],fZ(X[j],sensor)));
1787:                     }
1788:                     break;
1789:              case 3:        // P
1790:                     for (j=0;j<Namos;j++)
1791:                     {
1792:                         Fagregado[j] = zor(Fagregado[j],zand(CAcc[i],fP(X[j],sensor)));
1793:                     }
1794:                     break;
1795:              default:
1796:
1797:         }
1798:     }
1799:     Snumerador = Sdenominador = 0.0;
1800:     for (i=0;i<Namos;i++)
1801:     {
1802:         Snumerador = Snumerador + (Fagregado[i] * X[i]);
1803:         Sdenominador = Sdenominador + Fagregado[i];
1804:     }
1805:     return (Snumerador / Sdenominador); // Centroid - Crisp value for acceleration
```

```
1806: }
1807:
1808: //_____________________________________________________________________________________________________
1809:
1810: float Scaled_Defuzzy_Acceleration(float Xinicial,float Xfinal,float Step,struct sonar sensor)
1811: {
1812:     //Calculates the crisp value of the output value Acceleration. This is done aggregating the
1813:     //scaled membership sectors and using the centroid method. The aggregation is done
1814:     //by the use of the OR for fuzzy sets (union). The centroid method calculates the
1815:     //gravity center of the resulting area.
1816:
1817:     unsigned char i;
1818:     unsigned char j;
1819:     unsigned int Namos;        // Number of steps "Step" in the interval "Xinicial" to "Xfinal"
1820:     float Fagregado[210];    // Resulting aggregated function
1821:     float X[210];            // X value for each step
1822:     float Snumerador;          // Integral of the momentum approximation
1823:     float Sdenominador;      // Integral of the area approximation
1824:
1825:     Namos = (unsigned int)((Xfinal - Xinicial) / Step);
1826:     X[0] = Xinicial;
1827:     Fagregado[0] = 0.0;
1828:
1829:     for (i=1;i<Namos;i++)
1830:     {
1831:         X[i] = X[i-1] + Step;
1832:         Fagregado[i] = 0.0;
1833:     }
1834:
1835:     for (i=0;i<nAcc;i++)
1836:     {
1837:         switch (iAcc[i])
1838:         {
1839:              case 0:        // EB
1840:                     for (j=0;j<Namos;j++)
1841:                     {
1842:                         Fagregado[j] = zor(Fagregado[j],CAcc[i] * fEB(X[j],sensor));
1843:                     }
1844:                     break;
1845:              case 1:        // B
1846:                     for (j=0;j<Namos;j++)
1847:                     {
```

```
1848:                         Fagregado[j] = zor(Fagregado[j],CAcc[i] * fB(X[j],sensor));
1849:                     }
1850:                     break;
1851:             case 2:         // Z
1852:                     for (j=0;j<Namos;j++)
1853:                     {
1854:                         Fagregado[j] = zor(Fagregado[j],CAcc[i] * fZ(X[j],sensor));
1855:                     }
1856:                     break;
1857:             case 3:         // P
1858:                     for (j=0;j<Namos;j++)
1859:                     {
1860:                         Fagregado[j] = zor(Fagregado[j],CAcc[i] * fP(X[j],sensor));
1861:                     }
1862:                     break;
1863:             default:
1864:
1865:         }
1866:     }
1867:     Snumerador = Sdenominador = 0.0;
1868:     for (i=0;i<Namos;i++)
1869:     {
1870:         Snumerador = Snumerador + (Fagregado[i] * X[i]);
1871:         Sdenominador = Sdenominador + Fagregado[i];
1872:     }
1873:     return (Snumerador / Sdenominador);    // Centroid - Crisp value for acceleration
1874: }
1875:
1876: //_____________________________________________________________________________________________________________
1877:
1878: float fHR(float x)
1879: {
1880:     //Output function Hard_Right for Steer
1881:     //x => Steer in deg/sec
1882:
1883:     if ((x < -100.0) || (x >= -30.0)) return (0.0);
1884:     else if ((x >= -100.0) && (x <= -60.0)) return (1.0);
1885:     else return (1.0 - ((x + 60.0)/30.0));
1886: }
1887:
1888: //_____________________________________________________________________________________________________________
1889:
```

```
1890: float fR(float x)
1891: {
1892:     //Output function Right for Steer
1893:     //x => Steer in deg/sec
1894:
1895:     if ((x <= -60.0) || (x >= 0.0)) return (0.0);
1896:     else if ((x > -60.0) && (x <= -30.0)) return ((x + 60.0)/30.0);
1897:     else return (1.0 - ((x + 30.0)/30.0));
1898: }
1899:
1900: //_____________________________________________________________________________________________________________
1901:
1902: float fC(float x)
1903: {
1904:     //Output function Center for Steer
1905:     //x => Steer in deg/sec
1906:
1907:     if ((x <= -30.0) || (x >= 30.0)) return (0.0);
1908:     else if ((x > -30.0) && (x <= 0.0)) return ((x + 30.0)/30.0);
1909:     else return (1.0 - (x/30.0));
1910: }
1911:
1912: //_____________________________________________________________________________________________________________
1913: float fL(float x)
1914: {
1915:     //Output function Left for Steer
1916:     //x => Steer in deg/sec
1917:
1918:     if ((x <= 0.0) || (x >= 60.0)) return (0.0);
1919:     else if ((x > 0.0) && (x <= 30.0)) return (x/30.0);
1920:     else return (1.0 - ((x - 30.0)/30.0));
1921: }
1922:
1923: //_____________________________________________________________________________________________________________
1924:
1925: float fHL(float x)
1926: {
1927:     //Output function Hard_Left for Steer
1928:     //x => Steer in deg/sec
1929:
1930:     if ((x <= 30.0) || (x > 100.0)) return (0.0);
1931:     else if ((x > 30.0) && (x < 60.0)) return ((x - 30.0)/30.0);
```

```
1932:     else return (1.0);
1933: }
1934:
1935: //_____________________________________________________________________________________________
1936:
1937: float fEB(float x,struct sonar sensor)
1938: {
1939:     //Output function Emergency_Brake for Acceleration
1940:     //x => Acceleration (m/s^2)
1941:     //acelmax => maximum acceleration (cm/sec^2)
1942:
1943:     if ((x < -sensor.Maximum_Acceleration) || (x >= -(0,875 * sensor.Maximum_Acceleration))) return 0.0;
1944:     else return (1.0 - ((x + sensor.Maximum_Acceleration)/(sensor.Maximum_Acceleration - 0,875 *
      sensor.Maximum_Acceleration)));
1945: }
1946:
1947: //_____________________________________________________________________________________________
1948: float fB(float x,struct sonar sensor)
1949: {
1950:     //Output function Brake for Acceleration
1951:     //x => Acceleration (m/s^2)
1952:     //acelmax => maximum acceleration (cm/sec^2)
1953:
1954:     if ((x <= -sensor.Maximum_Acceleration) || (x >= 0.0)) return (0.0);
1955:     else if ((x > -sensor.Maximum_Acceleration) && (x < -(0.625 * sensor.Maximum_Acceleration))) return ((x -
      sensor.Maximum_Acceleration)/(0.375 * sensor.Maximum_Acceleration));
1956:     else if ((x >= -(0.625 * sensor.Maximum_Acceleration)) && (x <= -(0.25 * sensor.Maximum_Acceleration))) return 1.0;
1957:     else return (1.0 - (x + 0.25 * sensor.Maximum_Acceleration)/(0.25 * sensor.Maximum_Acceleration));
1958: }
1959:
1960: //_____________________________________________________________________________________________
1961:
1962: float fZ(float x,struct sonar sensor)
1963: {
1964:     //Output function Zero for Acceleration
1965:     //x => Acceleration (m/s^2)
1966:
1967:     if ((x <= -(0.375 * sensor.Maximum_Acceleration)) || (x >= (0.375 * sensor.Maximum_Acceleration))) return (0.0);
1968:     else if ((x > -(0.375 * sensor.Maximum_Acceleration)) && (x <= 0.0)) return ((x + 0.375 *
      sensor.Maximum_Acceleration)/(0.375 * sensor.Maximum_Acceleration));
1969:     else return (1.0 - (x/(0.375 * sensor.Maximum_Acceleration)));
1970: }
```

```
1971:
1972: //_____________________________________________________________________________________________________
1973: float fP(float x,struct sonar sensor)
1974: {
1975:     //Output function Positive for Acceleration
1976:     //x => Acceleration (m/s^2)
1977:
1978:     if ((x <= 0.0) || (x > sensor.Maximum_Acceleration)) return (0.0);
1979:     else if ((x > 0.0) && (x < (0.25 * sensor.Maximum_Acceleration))) return (x/(0.25 * sensor.Maximum_Acceleration));
1980:     else return (1.0);
1981: }
1982:
1983: //_____________________________________________________________________________________________________
1984:
1985: unsigned int range_sonar_X(unsigned char sim, unsigned char ender, struct sonar sensor)
1986:
1987: {
1988:     //Simulates the ranging of the sonar with address "ender" and compares the distance (cm) to the
1989:     //proximity limit "limite". If less or equal sets Sonar_Condition=1, on the contrary Sonar_Condition=0
1990:     //Returns the read distance
1991:     //sim => case simulated
1992:     //ender => sonar I2c address
1993:     //sensor => sonar data structure
1994:
1995:     unsigned int data;
1996:
1997:     switch (sim)
1998:     {
1999:         case 0: // Front Left obstacle
2000:             switch (ender)
2001:             {
2002:                 case 0xE0:
2003:                     data = 100;
2004:                     break;
2005:                 case 0xE2:
2006:                     data = 1000;
2007:                     break;
2008:                 case 0xE4:
2009:                     data = 1000;
2010:                     break;
2011:                 case 0xE6:
2012:                     data = 1000;
```

```
2013:                   break;
2014:               case 0xE8:
2015:                   data = 1000;
2016:                   break;
2017:               case 0xEA:
2018:                   data = 1000;
2019:                   break;
2020:               case 0xEC:
2021:                   data = 1000;
2022:                   break;
2023:               case 0xEE:
2024:                   data = 100;
2025:                   break;
2026:           }
2027:           break;
2028:       case 1: // Front Right obstacle
2029:           switch (ender)
2030:           {
2031:               case 0xE0:
2032:                   data = 80;
2033:                   break;
2034:               case 0xE2:
2035:                   data = 100;
2036:                   break;
2037:               case 0xE4:
2038:                   data = 150;
2039:                   break;
2040:               case 0xE6:
2041:                   data = 1000;
2042:                   break;
2043:               case 0xE8:
2044:                   data = 1000;
2045:                   break;
2046:               case 0xEA:
2047:                   data = 1000;
2048:                   break;
2049:               case 0xEC:
2050:                   data = 1000;
2051:                   break;
2052:               case 0xEE:
2053:                   data = 1000;
2054:                   break;
```

```
2055:             }
2056:             break;
2057:         case 2: // Front Left and Front Right obstacle
2058:             switch (ender)
2059:             {
2060:                 case 0xE0:
2061:                     data = 70;
2062:                     break;
2063:                 case 0xE2:
2064:                     data = 80;
2065:                     break;
2066:                 case 0xE4:
2067:                     data = 90;
2068:                     break;
2069:                 case 0xE6:
2070:                     data = 1000;
2071:                     break;
2072:                 case 0xE8:
2073:                     data = 1000;
2074:                     break;
2075:                 case 0xEA:
2076:                     data = 1000;
2077:                     break;
2078:                 case 0xEC:
2079:                     data = 1000;
2080:                     break;
2081:                 case 0xEE:
2082:                     data = 150;
2083:                     break;
2084:             }
2085:             break;
2086:         case 3: // Lateral Right obstacle
2087:             switch (ender)
2088:             {
2089:                 case 0xE0:
2090:                     data = 1000;
2091:                     break;
2092:                 case 0xE2:
2093:                     data = 150;
2094:                     break;
2095:                 case 0xE4:
2096:                     data = 100;
```

```
2097:                     break;
2098:                 case 0xE6:
2099:                     data = 150;
2100:                     break;
2101:                 case 0xE8:
2102:                     data = 1000;
2103:                     break;
2104:                 case 0xEA:
2105:                     data = 1000;
2106:                     break;
2107:                 case 0xEC:
2108:                     data = 1000;
2109:                     break;
2110:                 case 0xEE:
2111:                     data = 1000;
2112:                     break;
2113:             }
2114:             break;
2115:     }
2116:     if (data <= sensor.Minimum_Distance) sensor.Sonar_Condition[ender] = 1;
2117:     else sensor.Sonar_Condition[ender] = 0;
2118:     return data;
2119: }
2120:
2121: //_____________________________________________________________________________________________________________
2122:
2123: void General_Perception_Vector(struct sonar sensor)
2124: {
2125:     //Calculates the value and direction of the General Perception vector using the individual Perception vectors
2126:     //of the 8 sensors. The velocity vector is at 90º. Are calculated: General_Perception_Value (0.0 to 1.0 -
2127:     //normalized value); General_Perception_Angle (-180.0 to 180.0 - direction of the General Perception vector
2128:     //relative to the velocity vector. Positive to the right, negative to the left and 0 on heading)
2129:     //sensor => sonar data structure
2130:
2131:     float Xp;
2132:     float Yp;
2133:     float Pt;
2134:     const float to_radians = 0.0174532925199;
2135:
2136:     unsigned char i;
2137:
2138:     sensor.General_Perception_Value = 0.0;
```

```
2139:     Xp = 0.0;
2140:     Yp = 0.0;
2141:     //For each sensor
2142:     for (i=0; i<sensor.Sonar_Number; i++)
2143:     {
2144:         if (sensor.Sonar_Distance[i] > sensor.Maximum_Distance) Pt = 0.0;
2145:         else if (sensor.Sonar_Distance[i] <= sensor.Minimum_Distance) Pt = 1.0;
2146:         else Pt = (sensor.Maximum_Distance - sensor.Sonar_Distance[i])/(sensor.Maximum_Distance -
      sensor.Minimum_Distance);
2147:         sensor.Perception_Value[i]= Pt;
2148:         if (Pt > sensor.General_Perception_Value) sensor.General_Perception_Value = Pt;
2149:         if ((Pt > 0.0) && (Pt <= 1.0))
2150:         {
2151:             Xp = Xp + (sensor.Sonar_Distance[i] * cos(sensor.Sonar_Angle[i] * to_radians));
2152:             Yp = Yp + (sensor.Sonar_Distance[i] * sin(sensor.Sonar_Angle[i] * to_radians));
2153:         }
2154:     }
2155:     //Heading
2156:     if ((Xp == 0.0) && (Yp >= 0.0)) sensor.General_Perception_Angle = 0.0;
2157:     //Back
2158:     else if ((Xp == 0.0) && (Yp < 0.0)) sensor.General_Perception_Angle = 180.0;
2159:     else sensor.General_Perception_Angle = atan(Yp/Xp) / to_radians;
2160:     //Fourth quadrant
2161:     if ((Xp > 0.0) && (Yp < 0.0)) sensor.General_Perception_Angle = 450 - sensor.General_Perception_Angle;
2162:     //First, Second or Third quadrant
2163:     else sensor.General_Perception_Angle = 90.0 - sensor.General_Perception_Angle;
2164: }
2165:
2166: //_____________________________________________________________________________________________________
2167:
2168: void General_Perception_V_Change(struct sonar sensor)
2169: {
2170:     //Calculates the variation of the General Perception value. Only the positive variation is considered.
2171:     //sensor => sonar data structure
2172:
2173:     unsigned char i;
2174:     float temp;
2175:
2176:     sensor.General_Perception_Change = 0.0;
2177:     //For each sensor, verify the positive variation
2178:     for (i=0; i<sensor.Sonar_Number; i++)
2179:     {
```

```
2180:         if (sensor.Sonar_Distance_Change[i] < 0)
2181:         {
2182:             temp = - (sensor.Sonar_Distance_Change[i] / (sensor.DTi[i] / 1000.0) / sensor.Maximum_Velocity);
2183:             temp = fmin(1.0,temp);
2184:         }
2185:         else temp = 0.0;
2186:         if (temp > sensor.General_Perception_Change) sensor.General_Perception_Change = temp;
2187:     }
2188: }
2189:
2190: //_____________________________________________________________________________________________________________
2191:
2192: bool Perception_Change_Membership(float Value)
2193: {
2194:     //Calculates the membership level for the normalized General Perception Change value.
2195:     //ZE (0.0 a 1.0) - membership level for Perception Change zero
2196:     //LO (0.0 a 1.0) - membership level for low Perception Change
2197:     //HI (0.0 a 1.0) - membership level for high Perception Change
2198:     //Value => change in lenght of the General Perception vector
2199:
2200:     #define inter1 5
2201:     #define retas1 4
2202:     float x1[retas1],x2[retas1],xp1[inter1],xp2[inter1];
2203:     unsigned char i,inter_i;
2204:
2205:     xp1[0] = 0.0;
2206:     xp2[0] = 0.2;
2207:     xp1[1] = 0.2;
2208:     xp2[1] = 0.3;
2209:     xp1[2] = 0.3;
2210:     xp2[2] = 0.45;
2211:     xp1[3] = 0.45;
2212:     xp2[3] = 0.75;
2213:     xp1[4] = 0.75;
2214:     xp2[4] = 1.0;
2215:
2216:     x1[0] = 0.0;
2217:     x2[0] = 0.3;
2218:     x1[1] = 0.0;
2219:     x2[1] = 0.2;
2220:     x1[2] = 0.2;
2221:     x2[2] = 0.45;
```

```
2222:     x1[3] = 0.2;
2223:     x2[3] = 0.75;
2224:
2225:     inter_i = 255;
2226:     for (i=0;i<inter1;i++)
2227:     {
2228:         if ((Value >= xp1[i]) && (Value <= xp2[i])) {inter_i = i; break;}
2229:     }
2230:      if (inter_i == 255) return 0.0;
2231:     switch (inter_i)
2232:     {
2233:         case 0:
2234:                return zor(reta_down(0,Value),reta_up(1,Value));
2235:         case 1:
2236:                return zor(zor(reta_down(2,Value),reta_down(0,Value)),reta_up(3,Value));
2237:         case 2:
2238:                return zor(reta_down(2,Value),reta_up(3,Value));
2239:         case 3:
2240:             return reta_up(3,Value);
2241:         case 4:
2242:             return 1.0;
2243:     }
2244: }
2245:
2246: //_____________________________________________________________________________________________________________
2247:
2248: bool Perception_Angle_Membership(float Angle)
2249: {
2250:     //Calculates the membership levels and sectors for the General Perception angle.
2251:     //RB (0) - membership level in the Right_Back sector
2252:     //RF (1) - membership level in the Right_Front sector
2253:     //LF (2) - membership level in the Left_Front sector
2254:     //LB (3) - membership level in the Left_Back sector
2255:     //Angle => direction of the General Perception vector
2256:
2257:     #define inter2 10
2258:     #define retas2 8
2259:     float x1[retas2],x2[retas2],xp1[inter2],xp2[inter2];
2260:     unsigned char i,inter_i;
2261:
2262:     xp1[0] = -180.0;
2263:     xp2[0] = -135.0;
```

```
2264:     xp1[1] = -135.0;
2265:     xp2[1] = -120.0;
2266:     xp1[2] = -120.0;
2267:     xp2[2] = -60.0;
2268:     xp1[3] = -60.0;
2269:     xp2[3] = -45.0;
2270:     xp1[4] = -45.0;
2271:     xp2[4] = 0.0;
2272:     xp1[5] = 0.0;
2273:     xp2[5] = 45.0;
2274:     xp1[6] = 45.0;
2275:     xp2[6] = 60.0;
2276:     xp1[7] = 60.0;
2277:     xp2[7] = 120.0;
2278:     xp1[8] = 120.0;
2279:     xp2[8] = 135.0;
2280:     xp1[9] = 135.0;
2281:     xp2[9] = 180.0;
2282:
2283:     x1[0] = -225.0;
2284:     x2[0] = -135.0;
2285:     x1[1] = -135.0;
2286:     x2[1] = -60.0;
2287:     x1[2] = -120.0;
2288:     x2[2] = -45.0;
2289:     x1[3] = -45.0;
2290:     x2[3] = 45.0;
2291:     x1[4] = -45.0;
2292:     x2[4] = 45.0;
2293:     x1[5] = 45.0;
2294:     x2[5] = 120.0;
2295:     x1[6] = 60.0;
2296:     x2[6] = 135.0;
2297:     x1[7] = 135.0;
2298:     x2[7] = 225.0;
2299:
2300:     inter_i = 255;
2301:     for (i=0;i<inter2;i++)
2302:     {
2303:         if ((Angle >= xp1[i]) && (Angle <= xp2[i])) {inter_i = i; break;}
2304:     }
2305:      if (inter_i == 255) return 0;
```

```
2306:     switch (inter_i)
2307:     {
2308:         case 0:
2309:             nPA = 1;
2310:             PA[0] = reta_up(0,Angle);
2311:             iPA[0] = 0;
2312:             return 1;
2313:         case 1:
2314:             nPA = 1;
2315:             PA[0] = reta_down(1,Angle);
2316:             iPA[0] = 0;
2317:             return 1;
2318:         case 2:
2319:             nPA = 2;
2320:             PA[0] = reta_down(1,Angle);
2321:             PA[1] = reta_up(2,Angle);
2322:             iPA[0] = 0;
2323:             iPA[1] = 1;
2324:             return 1;
2325:         case 3:
2326:             nPA = 1;
2327:             PA[0] = reta_up(2,Angle);
2328:             iPA[0] = 1;
2329:             return 1;
2330:         case 4:
2331:             nPA = 1;
2332:             PA[0] = reta_down(3,Angle);
2333:             iPA[0] = 1;
2334:             return 1;
2335:         case 5:
2336:             nPA = 1;
2337:             PA[0] = reta_up(4,Angle);
2338:             iPA[0] = 2;
2339:             return 1;
2340:         case 6:
2341:             nPA = 1;
2342:             PA[0] = reta_down(5,Angle);
2343:             iPA[0] = 2;
2344:             return 1;
2345:         case 7:
2346:             nPA = 2;
2347:             PA[0] = reta_down(5,Angle);
```

```
2348:             PA[1] = reta_up(6,Angle);
2349:             iPA[0] = 2;
2350:             iPA[1] = 3;
2351:             return 1;
2352:         case 8:
2353:             nPA = 1;
2354:             PA[0] = reta_up(6,Angle);
2355:             iPA[0] = 3;
2356:             return 1;
2357:         case 9:
2358:             nPA = 1;
2359:             PA[0] = reta_down(7,Angle);
2360:             iPA[0] = 3;
2361:             return 1;
2362:     }
2363: }
2364:
2365: //_____________________________________________________________________________________________________________
2366:
2367: bool Perception_Value_Membership(float Value)
2368: {
2369:     //Calculates the membership level for the normalized General Perception value.
2370:     //VLP (0) - membership level for Very_Low Perception
2371:     //LP (1) - membership level for Low Perception
2372:     //MP (2) - membership level for Medium Perception
2373:     //HP (3) - membership level for High Perception
2374:     //VHP (4) - membership level for Very_High Perception
2375:     //Value => lenght of the General Perception vector
2376:
2377:
2378:     #define inter3 8
2379:     #define retas3 8
2380:     float x1[retas3],x2[retas3],xp1[inter3],xp2[inter3];
2381:     unsigned char i,inter_i;
2382:
2383:     xp1[0] = 0.0;
2384:     xp2[0] = 0.2;
2385:     xp1[1] = 0.2;
2386:     xp2[1] = 0.3;
2387:     xp1[2] = 0.3;
2388:     xp2[2] = 0.4;
2389:     xp1[3] = 0.4;
```

```
2390:     xp2[3] = 0.5;
2391:     xp1[4] = 0.5;
2392:     xp2[4] = 0.6;
2393:     xp1[5] = 0.6;
2394:     xp2[5] = 0.7;
2395:     xp1[6] = 0.7;
2396:     xp2[6] = 0.8;
2397:     xp1[7] = 0.8;
2398:     xp2[7] = 1.0;
2399:
2400:     x1[0] = 0.2;
2401:     x2[0] = 0.4;
2402:     x1[1] = 0.2;
2403:     x2[1] = 0.4;
2404:     x1[2] = 0.4;
2405:     x2[2] = 0.6;
2406:     x1[3] = 0.3;
2407:     x2[3] = 0.5;
2408:     x1[4] = 0.5;
2409:     x2[4] = 0.7;
2410:     x1[5] = 0.4;
2411:     x2[5] = 0.6;
2412:     x1[6] = 0.6;
2413:     x2[6] = 0.8;
2414:     x1[7] = 0.6;
2415:     x2[7] = 0.8;
2416:
2417:     inter_i = 255;
2418:     for (i=0;i<inter3;i++)
2419:     {
2420:         if ((Value >= xp1[i]) && (Value <= xp2[i])) {inter_i = i; break;}
2421:     }
2422:      if (inter_i == 255) return 0;
2423:     switch (inter_i)
2424:     {
2425:         case 0:
2426:             nPV = 1;
2427:             PV[0] = 1.0;
2428:             iPV[0] = 0;
2429:             return 1;
2430:         case 1:
2431:             nPV = 2;
```

```
2432:             PV[0] = reta_down(0,Value);
2433:             PV[1] = reta_up(1,Value);
2434:             iPV[0] = 0;
2435:             iPV[1] = 1;
2436:             return 1;
2437:         case 2:
2438:             nPV = 3;
2439:             PV[0] = reta_down(0,Value);
2440:             PV[1] = reta_up(1,Value);
2441:             PV[2] = reta_up(3,Value);
2442:             iPV[0] = 0;
2443:             iPV[1] = 1;
2444:             iPV[2] = 2;
2445:             return 1;
2446:         case 3:
2447:             nPV = 3;
2448:             PV[0] = reta_down(2,Value);
2449:             PV[1] = reta_up(3,Value);
2450:             PV[2] = reta_up(5,Value);
2451:             iPV[0] = 1;
2452:             iPV[1] = 2;
2453:             iPV[2] = 3;
2454:             return 1;
2455:         case 4:
2456:             nPV = 3;
2457:             PV[0] = reta_down(4,Value);
2458:             PV[1] = reta_up(5,Value);
2459:             PV[2] = reta_down(2,Value);
2460:             iPV[0] = 2;
2461:             iPV[1] = 3;
2462:             iPV[2] = 1;
2463:             return 1;
2464:         case 5:
2465:             nPV = 3;
2466:             PV[0] = reta_down(4,Value);
2467:             PV[1] = reta_up(7,Value);
2468:             PV[2] = reta_down(6,Value);
2469:             iPV[0] = 2;
2470:             iPV[1] = 4;
2471:             iPV[2] = 3;
2472:             return 1;
2473:         case 6:
```

```
2474:             nPV = 2;
2475:             PV[0] = reta_down(6,Value);
2476:             PV[1] = reta_up(7,Value);
2477:             iPV[0] = 3;
2478:             iPV[1] = 4;
2479:             return 1;
2480:         case 7:
2481:             nPV = 1;
2482:             PV[0] = 1.0;
2483:             iPV[0] = 4;
2484:             return 1.0;
2485:     }
2486: }
2487:
2488: //*****************SPAOFE FUNCTIONS*********************
2489:
2490:
```