```
/*****************************************************************************
* SPAOFE - Sonar Perception Avoiding Objects Fuzzy Engine - Version 1.0      *
* © Copyright 2012,2013,2014 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 V3.08 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 : 1.0
Date    : 21/01/2014
Author  : Filippo Pardini
Company : Filippo Pardini

Comments:

This is SPAOFE engine first version.

Functions that have been corrected since the preliminary version:
range_sonar_X
General_Perception_Vector
General_Perception_V_Change
Perception_Change_Membership
Perception_Angle_Membership
Perception_Value_Membership
```

```
43: fHR,fR,fC,fL,fHL,fEB,fB,fZ,fP
44:
45: The defuzzification functions are not yet tested.
46:
47: I am posting this version for interested people that
48: can help me to semantically verify and evaluate it.
49:
50: Chip type               : ATxmega128A1
51: Program type            : Application
52: AVR Core Clock frequency: 32,000000 MHz
53: Memory model            : Small
54: Data Stack size         : 2500
55: *****************************************************/
56:
57: #define _ATXMEGA_USART_USARTF0
58: #include <io.h>
59: #include <stdio.h>
60: #include <stdbool.h>
61: #include <math.h>
62: #include <delay.h>
63: #include <stdlib.h>
64:
65: //**********************SPAOFE**************************
66:
67: #define True 1
68: #define False 0
69:
70: #define zone 1.0                                    // Fuzzy one
71: #define zzero 0.0                                   // Fuzzy zero
72: #define zor(a,b) ((a) > (b) ? (a) : (b))            // Fuzzy or
73: #define zand(a,b) ((a) < (b) ? (a) : (b))           // Fuzzy and
74: #define znot(a) (zone+zzero-a)                      // Fuzzy not
75:
76: #define N_Sonar 8                                   // Number of ultrasonic sensors
77: #define infinito 1100                               // Infinity distance
78: #define eixo 34.0                                   // Distance (cm) between wheels
79:
80: #define reta_up(n,x) (x - x1[n])/(x2[n]-x1[n])      // Equation for stright line up
81: #define reta_down(n,x) 1-(x - x1[n])/(x2[n]-x1[n])  // Equation for stright line down
82:
83: #define ZE PC[0]                                    // ZE sector
84: #define LO PC[1]                                    // LO sector
```

```
 85: #define HI PC[2]                                    // HI sector
 86: #define RB PA[0]                                    // RB sector
 87: #define RF PA[1]                                    // RF sector
 88: #define LF PA[2]                                    // LF sector
 89: #define LB PA[3]                                    // LB sector
 90: #define VLP PV[0]                                   // VLP sector
 91: #define LP PV[1]                                    // LP sector
 92: #define MP PV[2]                                    // MP sector
 93: #define HP PV[3]                                    // HP sector
 94: #define VHP PV[4]                                   // VHP sector
 95:
 96: #define ad_aeternum while(1)
 97:
 98: struct sonar
 99: {
100:     // Number of sensors in the ring .
101:     unsigned char Sonar_Number;
102:     // Distance previous reading instant of time
103:     unsigned long int Ti_1[N_Sonar];
104:     // Time elapsed since previous distance reading.
105:     unsigned long int DTi[N_Sonar];
106:     // Sensors I2C addresses.
107:     unsigned char Sonar_Addr[N_Sonar];
108:     // 0 if > Minimum_Distance; 1 if <= Minimum_Distance).
109:     unsigned char Sonar_Condition[N_Sonar];
110:     // Sensor readed distance.
111:     unsigned int Sonar_Distance[N_Sonar];
112:     // Distance change
113:     signed int Sonar_Distance_Change[N_Sonar];
114:     // Sensor individual perception direction (degrees).
115:     float Sonar_Angle[N_Sonar];
116:     // Normalized individual Perception vector value.
117:     float Perception_Value[N_Sonar];
118:     // General Perception change in time.
119:     float General_Perception_Change;
120:     // General Perception direction. (-180 to 180) degrees to heading
121:     float General_Perception_Angle;
122:     // Normalized General Perception Value (0 to 1).
123:     float General_Perception_Value;
124:     // Maximum acceleration (cm/s^2).
125:     float Maximum_Acceleration;
126:     // Maximum velocity (cm/s).
```

```
127:     float Maximum_Velocity;
128:     // Minimum velocity (cm/s).
129:     float Minimum_Velocity;
130:     // Maximum distance (cm).
131:     unsigned int Maximum_Distance;
132:     // Minimum distance (cm).
133:     unsigned int Minimum_Distance;
134:     // Normal velocity (cm/s) corresponding to acceleration sector ZERO.
135:     float Velocity;
136: };
137:
138: struct sonar SPAOFE =   // Initializations
139: {
140:     .Sonar_Number = N_Sonar,
141:     .Sonar_Addr = {0xE0,0xE2,0xE4,0xE6,0xE8,0xEA,0xEC,0xEE},
142:     .Sonar_Angle = {90.0,45.0,0.0,315.0,270.0,225.0,180.0,135.0},
143:     .Maximum_Acceleration = 10.0,
144:     .Velocity = 15.0,
145:     .Maximum_Velocity = 40.0,
146:     .Minimum_Velocity = 4.0,
147:     .Maximum_Distance = 400,
148:     .Minimum_Distance = 30,
149:     .Sonar_Distance = {infinito,infinito,infinito,infinito,infinito,infinito,infinito,infinito},
150:     .Ti_1 = {0,0,0,0,0,0,0,0}
151: };
152:
153: //Time (49 days capacity)
154: unsigned long int relogio = 0;
155: // 9 - Stop abruptly; 10 - Stop smoothly.
156: unsigned char Brake;
157: // Sectors VLP,LP,MP,HP,VHP.
158: float PV[5];
159: // PV sector index.
160: unsigned char iPV[5];
161: // Number of iPVs.
162: unsigned char nPV;
163: // Sectors RB,RF,LF,LB.
164: float PA[4];
165: // PA sector index.
166: unsigned char iPA[4];
167: // Number of iPAs.
168: unsigned char nPA;
```

```
169: // Sectors ZE,LO,HI.
170: float PC[3];
171: // PC sector index.
172: unsigned char iPC[3];
173: // Number of iPCs.
174: unsigned char nPC;
175: // Steer sector index.
176: unsigned char iSteer[10];
177: // Corresponding cut values.
178: float CSteer[10];
179: // Number of iSteer.
180: unsigned char nSteer;
181: // Acceleration sector index.
182: unsigned char iAcc[10];
183: // Corresponding cut values.
184: float CAcc[10];
185: // Number of iAcc.
186: unsigned char nAcc;
187: // Steer crisp value.
188: float Steer_Crisp;
189: // Turn direction (-1 right; +1 left; 0 stright).
190: signed char Direcao;
191: // Turn radius (cm) to obtain Steer_Crisp (degrees/s).
192: float Rg;
193: // Acceleration crisp value.
194: float Acceleration_Crisp;
195:
196: char CharConv[15];          // Debug
197: char CSt0[5];               // Debug
198: char CSt1[5];               // Debug
199: char CSt2[5];               // Debug
200: char CSt3[5];               // Debug
201: char CSt4[5];               // Debug
202: char CSt5[5];               // Debug
203:
204: //Simulates the ranging of the sonar with address "ender".
205: //Returns the read distance
206: //sim => case simulated
207: //ender => sonar I2c address
208: unsigned int range_sonar_X(unsigned char sim, unsigned char ender);
209:
210: //Calculates the value and direction of the General Perception vector using the individual Perception vectors
```

```
211: //of the 8 sensors. The velocity vector is at 90º. Are calculated: General_Perception_Value (0.0 to 1.0 -
212: //normalized value); General_Perception_Angle (-180.0 to 180.0 - direction of the General Perception vector
213: //relative to the velocity vector. Positive to the right, negative to the left and 0 on heading)
214: void General_Perception_Vector(void);
215:
216: //Calculates the variation of the General Perception value. Only the positive variation is considered.
217: void General_Perception_V_Change(void);
218:
219: //Calculates the membership levels and sectors for the General Perception angle.
220: //RB (0) - membership level in the Right_Back sector
221: //RF (1) - membership level in the Right_Front sector
222: //LF (2) - membership level in the Left_Front sector
223: //LB (3) - membership level in the Left_Back sector
224: //Angle => direction of the General Perception vector
225: bool Perception_Angle_Membership(float Angle);
226:
227: //Calculates the membership level for the normalized General Perception value.
228: //VLP (0) - membership level for Very_Low Perception
229: //LP (1) - membership level for Low Perception
230: //MP (2) - membership level for Medium Perception
231: //HP (3) - membership level for High Perception
232: //VHP (4) - membership level for Very_High Perception
233: //Value => lenght of the General Perception vector
234: bool Perception_Value_Membership(float Value);
235:
236: //Calculates the membership level for the normalized General Perception Change value.
237: //ZE (0.0 a 1.0) - membership level for Perception Change zero
238: //LO (0.0 a 1.0) - membership level for low Perception Change
239: //HI (0.0 a 1.0) - membership level for high Perception Change
240: //Value => change in lenght of the General Perception vector
241: bool Perception_Change_Membership(float Value);
242:
243: //Evaluates the 20 Steer rules. As inputs uses the Perception Value and Perception Angle membership sectors.
244: //Calculates the Steer sectors to be aggregated and it's cutoff (or scaled) value "CSteer[]".
245: void Rules_Steer(void);
246:
247: //Evaluates the 11 Acceleration rules. As inputs uses the Perception Value and Perception Change membership sectors.
248: //Calculates the Acceleration sectors to be aggregated and it's cutoff (or scaled) value "CAcc[]".
249: void Rules_Acceleration(void);
250:
251: //Calculates the crisp value of the output value Steer. This is done aggregating the
252: //cutted or scaled membership sectors and using the centroid method. The aggregation is done
```

```
253: //by the use of the OR for fuzzy sets (union). The centroid method calculates the
254: //gravity center of the resulting area.
255: float Cutted_Defuzzy_Steer(float Xinicial,float Xfinal,float Step);
256:
257: //Calculates the crisp value of the output value Steer. This is done aggregating the
258: //scaled or scaled membership sectors and using the centroid method. The aggregation is done
259: //by the use of the OR for fuzzy sets (union). The centroid method calculates the
260: //gravity center of the resulting area.
261: float Scaled_Defuzzy_Steer(float Xinicial,float Xfinal,float Step);
262:
263: //Calculates the crisp value of the output value Acceleration. This is done aggregating the
264: //cutted membership sectors and using the centroid method. The aggregation is done
265: //by the use of the OR for fuzzy sets (union). The centroid method calculates the
266: //gravity center of the resulting area.
267: float Cutted_Defuzzy_Acceleration(float Xinicial,float Xfinal,float Step);
268:
269: //Calculates the crisp value of the output value Acceleration. This is done aggregating the
270: //scaled membership sectors and using the centroid method. The aggregation is done
271: //by the use of the OR for fuzzy sets (union). The centroid method calculates the
272: //gravity center of the resulting area.
273: float Scaled_Defuzzy_Acceleration(float Xinicial,float Xfinal,float Step);
274:
275: //Output function Hard_Right for Steer
276: //x => Steer in deg/sec
277: float fHR(float x);
278:
279: //Output function Right for Steer
280: //x => Steer in deg/sec
281: float fR(float x);
282:
283: //Output function Center for Steer
284: //x => Steer in deg/sec
285: float fC(float x);
286:
287: //Output function Left for Steer
288: //x => Steer in deg/sec
289: float fL(float x);
290:
291: //Output function Hard_Left for Steer
292: //x => Steer in deg/sec
293: float fHL(float x);
294:
```

```
295: //Output function Emergency_Brake for Acceleration
296: //x => Acceleration (m/s^2)
297: //acelmax => maximum acceleration (cm/sec^2)
298: float fEB(float x);
299:
300: //Output function Brake for Acceleration
301: //x => Acceleration (m/s^2)
302: //acelmax => maximum acceleration (cm/sec^2)
303: float fB(float x);
304:
305: //Output function Zero for Acceleration
306: //x => Acceleration (m/s^2)
307: float fZ(float x);
308:
309: //Output function Positive for Acceleration
310: //x => Acceleration (m/s^2)
311: float fP(float x);
312:
313: //Verifies the first "Comp" elements of the array "*Vetor" to verify which are non zero.
314: //The indices of the non zero elements are stored in the array "*Indices" and the indices
315: //quantity is stored in "*Num"
316: void Msectors_Nao_Zero(float *Vetor,unsigned char Comp,unsigned char *Indices,unsigned char *Num);
317:
318: //**********************SPAOFE*************************
319:
320: // System Clocks initialization
321: void system_clocks_init(void)
322: {
323:     unsigned char n,s;
324:
325:     // Optimize for speed
326:     #pragma optsize-
327:     // Save interrupts enabled/disabled state
328:     s=SREG;
329:     // Disable interrupts
330:     #asm("cli")
331:
332:     // Internal 32 kHz RC oscillator initialization
333:     // Enable the internal 32 kHz RC oscillator
334:     OSC.CTRL|=OSC_RC32KEN_bm;
335:     // Wait for the internal 32 kHz RC oscillator to stabilize
336:     while ((OSC.STATUS & OSC_RC32KRDY_bm)==0);
```

```
337:
338:     // Internal 32 MHz RC oscillator initialization
339:     // Enable the internal 32 MHz RC oscillator
340:     OSC.CTRL|=OSC_RC32MEN_bm;
341:
342:     // System Clock prescaler A division factor: 1
343:     // System Clock prescalers B & C division factors: B:1, C:1
344:     // ClkPer4: 32000,000 kHz
345:     // ClkPer2: 32000,000 kHz
346:     // ClkPer:  32000,000 kHz
347:     // ClkCPU:  32000,000 kHz
348:     n=(CLK.PSCTRL & (~(CLK_PSADIV_gm | CLK_PSBCDIV1_bm | CLK_PSBCDIV0_bm))) |
349:         CLK_PSADIV_1_gc | CLK_PSBCDIV_1_1_gc;
350:     CCP=CCP_IOREG_gc;
351:     CLK.PSCTRL=n;
352:
353:     // Internal 32 MHz RC osc. calibration reference clock source: 32.768 kHz Internal Osc.
354:     OSC.DFLLCTRL&= ~(OSC_RC32MCREF_bm | OSC_RC2MCREF_bm);
355:     // Enable the auto-calibration of the internal 32 MHz RC oscillator
356:     DFLLRC32M.CTRL|=DFLL_ENABLE_bm;
357:
358:     // Wait for the internal 32 MHz RC oscillator to stabilize
359:     while ((OSC.STATUS & OSC_RC32MRDY_bm)==0);
360:
361:     // Select the system clock source: 32 MHz Internal RC Osc.
362:     n=(CLK.CTRL & (~CLK_SCLKSEL_gm)) | CLK_SCLKSEL_RC32M_gc;
363:     CCP=CCP_IOREG_gc;
364:     CLK.CTRL=n;
365:
366:     // Disable the unused oscillators: 2 MHz, external clock/crystal oscillator, PLL
367:     OSC.CTRL&= ~(OSC_RC2MEN_bm | OSC_XOSCEN_bm | OSC_PLLEN_bm);
368:
369:     // ClkPer output disabled
370:     PORTCFG.CLKEVOUT&= ~PORTCFG_CLKOUT_gm;
371:     // Restore interrupts enabled/disabled state
372:     SREG=s;
373:     // Restore optimization for size if needed
374:     #pragma optsize_default
375: }
376:
377: // Ports initialization
378:     void ports_init(void)
```

```
379:     {
380:     // PORTA initialization
381:     // OUT register
382:     PORTA.OUT=0x00;
383:     // Pin0: Input
384:     // Pin1: Input
385:     // Pin2: Input
386:     // Pin3: Input
387:     // Pin4: Input
388:     // Pin5: Input
389:     // Pin6: Input
390:     // Pin7: Input
391:     PORTA.DIR=0x00;
392:     // Pin0 Output/Pull configuration: Totempole/No
393:     // Pin0 Input/Sense configuration: Sense both edges
394:     // Pin0 Inverted: Off
395:     // Pin0 Slew Rate Limitation: Off
396:     PORTA.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
397:     // Pin1 Output/Pull configuration: Totempole/No
398:     // Pin1 Input/Sense configuration: Sense both edges
399:     // Pin1 Inverted: Off
400:     // Pin1 Slew Rate Limitation: Off
401:     PORTA.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
402:     // Pin2 Output/Pull configuration: Totempole/No
403:     // Pin2 Input/Sense configuration: Sense both edges
404:     // Pin2 Inverted: Off
405:     // Pin2 Slew Rate Limitation: Off
406:     PORTA.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
407:     // Pin3 Output/Pull configuration: Totempole/No
408:     // Pin3 Input/Sense configuration: Sense both edges
409:     // Pin3 Inverted: Off
410:     // Pin3 Slew Rate Limitation: Off
411:     PORTA.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
412:     // Pin4 Output/Pull configuration: Totempole/No
413:     // Pin4 Input/Sense configuration: Sense both edges
414:     // Pin4 Inverted: Off
415:     // Pin4 Slew Rate Limitation: Off
416:     PORTA.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
417:     // Pin5 Output/Pull configuration: Totempole/No
418:     // Pin5 Input/Sense configuration: Sense both edges
419:     // Pin5 Inverted: Off
420:     // Pin5 Slew Rate Limitation: Off
```

```
421:     PORTA.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
422:     // Pin6 Output/Pull configuration: Totempole/No
423:     // Pin6 Input/Sense configuration: Sense both edges
424:     // Pin6 Inverted: Off
425:     // Pin6 Slew Rate Limitation: Off
426:     PORTA.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
427:     // Pin7 Output/Pull configuration: Totempole/No
428:     // Pin7 Input/Sense configuration: Sense both edges
429:     // Pin7 Inverted: Off
430:     // Pin7 Slew Rate Limitation: Off
431:     PORTA.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
432:     // Interrupt 0 level: Disabled
433:     // Interrupt 1 level: Disabled
434:     PORTA.INTCTRL=(PORTA.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
435:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
436:     // Pin0 Pin Change interrupt 0: Off
437:     // Pin1 Pin Change interrupt 0: Off
438:     // Pin2 Pin Change interrupt 0: Off
439:     // Pin3 Pin Change interrupt 0: Off
440:     // Pin4 Pin Change interrupt 0: Off
441:     // Pin5 Pin Change interrupt 0: Off
442:     // Pin6 Pin Change interrupt 0: Off
443:     // Pin7 Pin Change interrupt 0: Off
444:     PORTA.INT0MASK=0x00;
445:     // Pin0 Pin Change interrupt 1: Off
446:     // Pin1 Pin Change interrupt 1: Off
447:     // Pin2 Pin Change interrupt 1: Off
448:     // Pin3 Pin Change interrupt 1: Off
449:     // Pin4 Pin Change interrupt 1: Off
450:     // Pin5 Pin Change interrupt 1: Off
451:     // Pin6 Pin Change interrupt 1: Off
452:     // Pin7 Pin Change interrupt 1: Off
453:     PORTA.INT1MASK=0x00;
454:
455:     // PORTB initialization
456:     // OUT register
457:     PORTB.OUT=0x00;
458:     // Pin0: Input
459:     // Pin1: Input
460:     // Pin2: Input
461:     // Pin3: Input
462:     // Pin4: Input
```

```
463:     // Pin5: Input
464:     // Pin6: Input
465:     // Pin7: Input
466:     PORTB.DIR=0x00;
467:     // Pin0 Output/Pull configuration: Totempole/No
468:     // Pin0 Input/Sense configuration: Sense both edges
469:     // Pin0 Inverted: Off
470:     // Pin0 Slew Rate Limitation: Off
471:     PORTB.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
472:     // Pin1 Output/Pull configuration: Totempole/No
473:     // Pin1 Input/Sense configuration: Sense both edges
474:     // Pin1 Inverted: Off
475:     // Pin1 Slew Rate Limitation: Off
476:     PORTB.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
477:     // Pin2 Output/Pull configuration: Totempole/No
478:     // Pin2 Input/Sense configuration: Sense both edges
479:     // Pin2 Inverted: Off
480:     // Pin2 Slew Rate Limitation: Off
481:     PORTB.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
482:     // Pin3 Output/Pull configuration: Totempole/No
483:     // Pin3 Input/Sense configuration: Sense both edges
484:     // Pin3 Inverted: Off
485:     // Pin3 Slew Rate Limitation: Off
486:     PORTB.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
487:     // Pin4 Output/Pull configuration: Totempole/No
488:     // Pin4 Input/Sense configuration: Sense both edges
489:     // Pin4 Inverted: Off
490:     // Pin4 Slew Rate Limitation: Off
491:     PORTB.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
492:     // Pin5 Output/Pull configuration: Totempole/No
493:     // Pin5 Input/Sense configuration: Sense both edges
494:     // Pin5 Inverted: Off
495:     // Pin5 Slew Rate Limitation: Off
496:     PORTB.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
497:     // Pin6 Output/Pull configuration: Totempole/No
498:     // Pin6 Input/Sense configuration: Sense both edges
499:     // Pin6 Inverted: Off
500:     // Pin6 Slew Rate Limitation: Off
501:     PORTB.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
502:     // Pin7 Output/Pull configuration: Totempole/No
503:     // Pin7 Input/Sense configuration: Sense both edges
504:     // Pin7 Inverted: Off
```

```
505:     // Pin7 Slew Rate Limitation: Off
506:     PORTB.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
507:     // Interrupt 0 level: Disabled
508:     // Interrupt 1 level: Disabled
509:     PORTB.INTCTRL=(PORTB.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
510:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
511:     // Pin0 Pin Change interrupt 0: Off
512:     // Pin1 Pin Change interrupt 0: Off
513:     // Pin2 Pin Change interrupt 0: Off
514:     // Pin3 Pin Change interrupt 0: Off
515:     // Pin4 Pin Change interrupt 0: Off
516:     // Pin5 Pin Change interrupt 0: Off
517:     // Pin6 Pin Change interrupt 0: Off
518:     // Pin7 Pin Change interrupt 0: Off
519:     PORTB.INT0MASK=0x00;
520:     // Pin0 Pin Change interrupt 1: Off
521:     // Pin1 Pin Change interrupt 1: Off
522:     // Pin2 Pin Change interrupt 1: Off
523:     // Pin3 Pin Change interrupt 1: Off
524:     // Pin4 Pin Change interrupt 1: Off
525:     // Pin5 Pin Change interrupt 1: Off
526:     // Pin6 Pin Change interrupt 1: Off
527:     // Pin7 Pin Change interrupt 1: Off
528:     PORTB.INT1MASK=0x00;
529:
530:     // PORTC initialization
531:     // OUT register
532:     PORTC.OUT=0x00;
533:     // Pin0: Input
534:     // Pin1: Input
535:     // Pin2: Input
536:     // Pin3: Input
537:     // Pin4: Input
538:     // Pin5: Input
539:     // Pin6: Input
540:     // Pin7: Input
541:     PORTC.DIR=0x00;
542:     // Pin0 Output/Pull configuration: Totempole/No
543:     // Pin0 Input/Sense configuration: Sense both edges
544:     // Pin0 Inverted: Off
545:     // Pin0 Slew Rate Limitation: Off
546:     PORTC.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
```

```
547:     // Pin1 Output/Pull configuration: Totempole/No
548:     // Pin1 Input/Sense configuration: Sense both edges
549:     // Pin1 Inverted: Off
550:     // Pin1 Slew Rate Limitation: Off
551:     PORTC.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
552:     // Pin2 Output/Pull configuration: Totempole/No
553:     // Pin2 Input/Sense configuration: Sense both edges
554:     // Pin2 Inverted: Off
555:     // Pin2 Slew Rate Limitation: Off
556:     PORTC.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
557:     // Pin3 Output/Pull configuration: Totempole/No
558:     // Pin3 Input/Sense configuration: Sense both edges
559:     // Pin3 Inverted: Off
560:     // Pin3 Slew Rate Limitation: Off
561:     PORTC.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
562:     // Pin4 Output/Pull configuration: Totempole/No
563:     // Pin4 Input/Sense configuration: Sense both edges
564:     // Pin4 Inverted: Off
565:     // Pin4 Slew Rate Limitation: Off
566:     PORTC.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
567:     // Pin5 Output/Pull configuration: Totempole/No
568:     // Pin5 Input/Sense configuration: Sense both edges
569:     // Pin5 Inverted: Off
570:     // Pin5 Slew Rate Limitation: Off
571:     PORTC.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
572:     // Pin6 Output/Pull configuration: Totempole/No
573:     // Pin6 Input/Sense configuration: Sense both edges
574:     // Pin6 Inverted: Off
575:     // Pin6 Slew Rate Limitation: Off
576:     PORTC.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
577:     // Pin7 Output/Pull configuration: Totempole/No
578:     // Pin7 Input/Sense configuration: Sense both edges
579:     // Pin7 Inverted: Off
580:     // Pin7 Slew Rate Limitation: Off
581:     PORTC.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
582:     // Interrupt 0 level: Disabled
583:     // Interrupt 1 level: Disabled
584:     PORTC.INTCTRL=(PORTC.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
585:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
586:     // Pin0 Pin Change interrupt 0: Off
587:     // Pin1 Pin Change interrupt 0: Off
588:     // Pin2 Pin Change interrupt 0: Off
```

```
589:     // Pin3 Pin Change interrupt 0: Off
590:     // Pin4 Pin Change interrupt 0: Off
591:     // Pin5 Pin Change interrupt 0: Off
592:     // Pin6 Pin Change interrupt 0: Off
593:     // Pin7 Pin Change interrupt 0: Off
594:     PORTC.INT0MASK=0x00;
595:     // Pin0 Pin Change interrupt 1: Off
596:     // Pin1 Pin Change interrupt 1: Off
597:     // Pin2 Pin Change interrupt 1: Off
598:     // Pin3 Pin Change interrupt 1: Off
599:     // Pin4 Pin Change interrupt 1: Off
600:     // Pin5 Pin Change interrupt 1: Off
601:     // Pin6 Pin Change interrupt 1: Off
602:     // Pin7 Pin Change interrupt 1: Off
603:     PORTC.INT1MASK=0x00;
604:
605:     // PORTD initialization
606:     // OUT register
607:     PORTD.OUT=0x00;
608:     // Pin0: Input
609:     // Pin1: Input
610:     // Pin2: Input
611:     // Pin3: Input
612:     // Pin4: Input
613:     // Pin5: Input
614:     // Pin6: Input
615:     // Pin7: Input
616:     PORTD.DIR=0x00;
617:     // Pin0 Output/Pull configuration: Totempole/No
618:     // Pin0 Input/Sense configuration: Sense both edges
619:     // Pin0 Inverted: Off
620:     // Pin0 Slew Rate Limitation: Off
621:     PORTD.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
622:     // Pin1 Output/Pull configuration: Totempole/No
623:     // Pin1 Input/Sense configuration: Sense both edges
624:     // Pin1 Inverted: Off
625:     // Pin1 Slew Rate Limitation: Off
626:     PORTD.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
627:     // Pin2 Output/Pull configuration: Totempole/No
628:     // Pin2 Input/Sense configuration: Sense both edges
629:     // Pin2 Inverted: Off
630:     // Pin2 Slew Rate Limitation: Off
```

```
631:     PORTD.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
632:     // Pin3 Output/Pull configuration: Totempole/No
633:     // Pin3 Input/Sense configuration: Sense both edges
634:     // Pin3 Inverted: Off
635:     // Pin3 Slew Rate Limitation: Off
636:     PORTD.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
637:     // Pin4 Output/Pull configuration: Totempole/No
638:     // Pin4 Input/Sense configuration: Sense both edges
639:     // Pin4 Inverted: Off
640:     // Pin4 Slew Rate Limitation: Off
641:     PORTD.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
642:     // Pin5 Output/Pull configuration: Totempole/No
643:     // Pin5 Input/Sense configuration: Sense both edges
644:     // Pin5 Inverted: Off
645:     // Pin5 Slew Rate Limitation: Off
646:     PORTD.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
647:     // Pin6 Output/Pull configuration: Totempole/No
648:     // Pin6 Input/Sense configuration: Sense both edges
649:     // Pin6 Inverted: Off
650:     // Pin6 Slew Rate Limitation: Off
651:     PORTD.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
652:     // Pin7 Output/Pull configuration: Totempole/No
653:     // Pin7 Input/Sense configuration: Sense both edges
654:     // Pin7 Inverted: Off
655:     // Pin7 Slew Rate Limitation: Off
656:     PORTD.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
657:     // Interrupt 0 level: Disabled
658:     // Interrupt 1 level: Disabled
659:     PORTD.INTCTRL=(PORTD.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
660:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
661:     // Pin0 Pin Change interrupt 0: Off
662:     // Pin1 Pin Change interrupt 0: Off
663:     // Pin2 Pin Change interrupt 0: Off
664:     // Pin3 Pin Change interrupt 0: Off
665:     // Pin4 Pin Change interrupt 0: Off
666:     // Pin5 Pin Change interrupt 0: Off
667:     // Pin6 Pin Change interrupt 0: Off
668:     // Pin7 Pin Change interrupt 0: Off
669:     PORTD.INT0MASK=0x00;
670:     // Pin0 Pin Change interrupt 1: Off
671:     // Pin1 Pin Change interrupt 1: Off
672:     // Pin2 Pin Change interrupt 1: Off
```

```
673:     // Pin3 Pin Change interrupt 1: Off
674:     // Pin4 Pin Change interrupt 1: Off
675:     // Pin5 Pin Change interrupt 1: Off
676:     // Pin6 Pin Change interrupt 1: Off
677:     // Pin7 Pin Change interrupt 1: Off
678:     PORTD.INT1MASK=0x00;
679:
680:     // PORTE initialization
681:     // OUT register
682:     PORTE.OUT=0x00;
683:     // Pin0: Input
684:     // Pin1: Input
685:     // Pin2: Input
686:     // Pin3: Input
687:     // Pin4: Input
688:     // Pin5: Input
689:     // Pin6: Input
690:     // Pin7: Input
691:     PORTE.DIR=0x00;
692:     // Pin0 Output/Pull configuration: Totempole/No
693:     // Pin0 Input/Sense configuration: Sense both edges
694:     // Pin0 Inverted: Off
695:     // Pin0 Slew Rate Limitation: Off
696:     PORTE.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
697:     // Pin1 Output/Pull configuration: Totempole/No
698:     // Pin1 Input/Sense configuration: Sense both edges
699:     // Pin1 Inverted: Off
700:     // Pin1 Slew Rate Limitation: Off
701:     PORTE.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
702:     // Pin2 Output/Pull configuration: Totempole/No
703:     // Pin2 Input/Sense configuration: Sense both edges
704:     // Pin2 Inverted: Off
705:     // Pin2 Slew Rate Limitation: Off
706:     PORTE.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
707:     // Pin3 Output/Pull configuration: Totempole/No
708:     // Pin3 Input/Sense configuration: Sense both edges
709:     // Pin3 Inverted: Off
710:     // Pin3 Slew Rate Limitation: Off
711:     PORTE.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
712:     // Pin4 Output/Pull configuration: Totempole/No
713:     // Pin4 Input/Sense configuration: Sense both edges
714:     // Pin4 Inverted: Off
```

```
715:     // Pin4 Slew Rate Limitation: Off
716:     PORTE.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
717:     // Pin5 Output/Pull configuration: Totempole/No
718:     // Pin5 Input/Sense configuration: Sense both edges
719:     // Pin5 Inverted: Off
720:     // Pin5 Slew Rate Limitation: Off
721:     PORTE.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
722:     // Pin6 Output/Pull configuration: Totempole/No
723:     // Pin6 Input/Sense configuration: Sense both edges
724:     // Pin6 Inverted: Off
725:     // Pin6 Slew Rate Limitation: Off
726:     PORTE.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
727:     // Pin7 Output/Pull configuration: Totempole/No
728:     // Pin7 Input/Sense configuration: Sense both edges
729:     // Pin7 Inverted: Off
730:     // Pin7 Slew Rate Limitation: Off
731:     PORTE.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
732:     // Interrupt 0 level: Disabled
733:     // Interrupt 1 level: Disabled
734:     PORTE.INTCTRL=(PORTE.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
735:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
736:     // Pin0 Pin Change interrupt 0: Off
737:     // Pin1 Pin Change interrupt 0: Off
738:     // Pin2 Pin Change interrupt 0: Off
739:     // Pin3 Pin Change interrupt 0: Off
740:     // Pin4 Pin Change interrupt 0: Off
741:     // Pin5 Pin Change interrupt 0: Off
742:     // Pin6 Pin Change interrupt 0: Off
743:     // Pin7 Pin Change interrupt 0: Off
744:     PORTE.INT0MASK=0x00;
745:     // Pin0 Pin Change interrupt 1: Off
746:     // Pin1 Pin Change interrupt 1: Off
747:     // Pin2 Pin Change interrupt 1: Off
748:     // Pin3 Pin Change interrupt 1: Off
749:     // Pin4 Pin Change interrupt 1: Off
750:     // Pin5 Pin Change interrupt 1: Off
751:     // Pin6 Pin Change interrupt 1: Off
752:     // Pin7 Pin Change interrupt 1: Off
753:     PORTE.INT1MASK=0x00;
754:
755:     // PORTF initialization
756:     // OUT register
```

```
757:     PORTF.OUT=0x08;
758:     // Pin0: Input
759:     // Pin1: Input
760:     // Pin2: Input
761:     // Pin3: Output
762:     // Pin4: Input
763:     // Pin5: Input
764:     // Pin6: Input
765:     // Pin7: Input
766:     PORTF.DIR=0x08;
767:     // Pin0 Output/Pull configuration: Totempole/No
768:     // Pin0 Input/Sense configuration: Sense both edges
769:     // Pin0 Inverted: Off
770:     // Pin0 Slew Rate Limitation: Off
771:     PORTF.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
772:     // Pin1 Output/Pull configuration: Totempole/No
773:     // Pin1 Input/Sense configuration: Sense both edges
774:     // Pin1 Inverted: Off
775:     // Pin1 Slew Rate Limitation: Off
776:     PORTF.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
777:     // Pin2 Output/Pull configuration: Totempole/No
778:     // Pin2 Input/Sense configuration: Sense both edges
779:     // Pin2 Inverted: Off
780:     // Pin2 Slew Rate Limitation: Off
781:     PORTF.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
782:     // Pin3 Output/Pull configuration: Totempole/No
783:     // Pin3 Input/Sense configuration: Sense both edges
784:     // Pin3 Inverted: Off
785:     // Pin3 Slew Rate Limitation: Off
786:     PORTF.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
787:     // Pin4 Output/Pull configuration: Totempole/No
788:     // Pin4 Input/Sense configuration: Sense both edges
789:     // Pin4 Inverted: Off
790:     // Pin4 Slew Rate Limitation: Off
791:     PORTF.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
792:     // Pin5 Output/Pull configuration: Totempole/No
793:     // Pin5 Input/Sense configuration: Sense both edges
794:     // Pin5 Inverted: Off
795:     // Pin5 Slew Rate Limitation: Off
796:     PORTF.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
797:     // Pin6 Output/Pull configuration: Totempole/No
798:     // Pin6 Input/Sense configuration: Sense both edges
```

```
799:     // Pin6 Inverted: Off
800:     // Pin6 Slew Rate Limitation: Off
801:     PORTF.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
802:     // Pin7 Output/Pull configuration: Totempole/No
803:     // Pin7 Input/Sense configuration: Sense both edges
804:     // Pin7 Inverted: Off
805:     // Pin7 Slew Rate Limitation: Off
806:     PORTF.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
807:     // Interrupt 0 level: Disabled
808:     // Interrupt 1 level: Disabled
809:     PORTF.INTCTRL=(PORTF.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
810:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
811:     // Pin0 Pin Change interrupt 0: Off
812:     // Pin1 Pin Change interrupt 0: Off
813:     // Pin2 Pin Change interrupt 0: Off
814:     // Pin3 Pin Change interrupt 0: Off
815:     // Pin4 Pin Change interrupt 0: Off
816:     // Pin5 Pin Change interrupt 0: Off
817:     // Pin6 Pin Change interrupt 0: Off
818:     // Pin7 Pin Change interrupt 0: Off
819:     PORTF.INT0MASK=0x00;
820:     // Pin0 Pin Change interrupt 1: Off
821:     // Pin1 Pin Change interrupt 1: Off
822:     // Pin2 Pin Change interrupt 1: Off
823:     // Pin3 Pin Change interrupt 1: Off
824:     // Pin4 Pin Change interrupt 1: Off
825:     // Pin5 Pin Change interrupt 1: Off
826:     // Pin6 Pin Change interrupt 1: Off
827:     // Pin7 Pin Change interrupt 1: Off
828:     PORTF.INT1MASK=0x00;
829:
830:     // PORTH initialization
831:     // OUT register
832:     PORTH.OUT=0x00;
833:     // Pin0: Input
834:     // Pin1: Input
835:     // Pin2: Input
836:     // Pin3: Input
837:     // Pin4: Input
838:     // Pin5: Input
839:     // Pin6: Input
840:     // Pin7: Input
```

```
841:     PORTH.DIR=0x00;
842:     // Pin0 Output/Pull configuration: Totempole/No
843:     // Pin0 Input/Sense configuration: Sense both edges
844:     // Pin0 Inverted: Off
845:     // Pin0 Slew Rate Limitation: Off
846:     PORTH.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
847:     // Pin1 Output/Pull configuration: Totempole/No
848:     // Pin1 Input/Sense configuration: Sense both edges
849:     // Pin1 Inverted: Off
850:     // Pin1 Slew Rate Limitation: Off
851:     PORTH.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
852:     // Pin2 Output/Pull configuration: Totempole/No
853:     // Pin2 Input/Sense configuration: Sense both edges
854:     // Pin2 Inverted: Off
855:     // Pin2 Slew Rate Limitation: Off
856:     PORTH.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
857:     // Pin3 Output/Pull configuration: Totempole/No
858:     // Pin3 Input/Sense configuration: Sense both edges
859:     // Pin3 Inverted: Off
860:     // Pin3 Slew Rate Limitation: Off
861:     PORTH.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
862:     // Pin4 Output/Pull configuration: Totempole/No
863:     // Pin4 Input/Sense configuration: Sense both edges
864:     // Pin4 Inverted: Off
865:     // Pin4 Slew Rate Limitation: Off
866:     PORTH.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
867:     // Pin5 Output/Pull configuration: Totempole/No
868:     // Pin5 Input/Sense configuration: Sense both edges
869:     // Pin5 Inverted: Off
870:     // Pin5 Slew Rate Limitation: Off
871:     PORTH.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
872:     // Pin6 Output/Pull configuration: Totempole/No
873:     // Pin6 Input/Sense configuration: Sense both edges
874:     // Pin6 Inverted: Off
875:     // Pin6 Slew Rate Limitation: Off
876:     PORTH.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
877:     // Pin7 Output/Pull configuration: Totempole/No
878:     // Pin7 Input/Sense configuration: Sense both edges
879:     // Pin7 Inverted: Off
880:     // Pin7 Slew Rate Limitation: Off
881:     PORTH.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
882:     // Interrupt 0 level: Disabled
```

```
883:     // Interrupt 1 level: Disabled
884:     PORTH.INTCTRL=(PORTH.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
885:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
886:     // Pin0 Pin Change interrupt 0: Off
887:     // Pin1 Pin Change interrupt 0: Off
888:     // Pin2 Pin Change interrupt 0: Off
889:     // Pin3 Pin Change interrupt 0: Off
890:     // Pin4 Pin Change interrupt 0: Off
891:     // Pin5 Pin Change interrupt 0: Off
892:     // Pin6 Pin Change interrupt 0: Off
893:     // Pin7 Pin Change interrupt 0: Off
894:     PORTH.INT0MASK=0x00;
895:     // Pin0 Pin Change interrupt 1: Off
896:     // Pin1 Pin Change interrupt 1: Off
897:     // Pin2 Pin Change interrupt 1: Off
898:     // Pin3 Pin Change interrupt 1: Off
899:     // Pin4 Pin Change interrupt 1: Off
900:     // Pin5 Pin Change interrupt 1: Off
901:     // Pin6 Pin Change interrupt 1: Off
902:     // Pin7 Pin Change interrupt 1: Off
903:     PORTH.INT1MASK=0x00;
904:
905:     // PORTJ initialization
906:     // OUT register
907:     PORTJ.OUT=0x00;
908:     // Pin0: Input
909:     // Pin1: Input
910:     // Pin2: Input
911:     // Pin3: Input
912:     // Pin4: Input
913:     // Pin5: Input
914:     // Pin6: Input
915:     // Pin7: Input
916:     PORTJ.DIR=0x00;
917:     // Pin0 Output/Pull configuration: Totempole/No
918:     // Pin0 Input/Sense configuration: Sense both edges
919:     // Pin0 Inverted: Off
920:     // Pin0 Slew Rate Limitation: Off
921:     PORTJ.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
922:     // Pin1 Output/Pull configuration: Totempole/No
923:     // Pin1 Input/Sense configuration: Sense both edges
924:     // Pin1 Inverted: Off
```

```
925:     // Pin1 Slew Rate Limitation: Off
926:     PORTJ.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
927:     // Pin2 Output/Pull configuration: Totempole/No
928:     // Pin2 Input/Sense configuration: Sense both edges
929:     // Pin2 Inverted: Off
930:     // Pin2 Slew Rate Limitation: Off
931:     PORTJ.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
932:     // Pin3 Output/Pull configuration: Totempole/No
933:     // Pin3 Input/Sense configuration: Sense both edges
934:     // Pin3 Inverted: Off
935:     // Pin3 Slew Rate Limitation: Off
936:     PORTJ.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
937:     // Pin4 Output/Pull configuration: Totempole/No
938:     // Pin4 Input/Sense configuration: Sense both edges
939:     // Pin4 Inverted: Off
940:     // Pin4 Slew Rate Limitation: Off
941:     PORTJ.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
942:     // Pin5 Output/Pull configuration: Totempole/No
943:     // Pin5 Input/Sense configuration: Sense both edges
944:     // Pin5 Inverted: Off
945:     // Pin5 Slew Rate Limitation: Off
946:     PORTJ.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
947:     // Pin6 Output/Pull configuration: Totempole/No
948:     // Pin6 Input/Sense configuration: Sense both edges
949:     // Pin6 Inverted: Off
950:     // Pin6 Slew Rate Limitation: Off
951:     PORTJ.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
952:     // Pin7 Output/Pull configuration: Totempole/No
953:     // Pin7 Input/Sense configuration: Sense both edges
954:     // Pin7 Inverted: Off
955:     // Pin7 Slew Rate Limitation: Off
956:     PORTJ.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
957:     // Interrupt 0 level: Disabled
958:     // Interrupt 1 level: Disabled
959:     PORTJ.INTCTRL=(PORTJ.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
960:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
961:     // Pin0 Pin Change interrupt 0: Off
962:     // Pin1 Pin Change interrupt 0: Off
963:     // Pin2 Pin Change interrupt 0: Off
964:     // Pin3 Pin Change interrupt 0: Off
965:     // Pin4 Pin Change interrupt 0: Off
966:     // Pin5 Pin Change interrupt 0: Off
```

```
967:     // Pin6 Pin Change interrupt 0: Off
968:     // Pin7 Pin Change interrupt 0: Off
969:     PORTJ.INT0MASK=0x00;
970:     // Pin0 Pin Change interrupt 1: Off
971:     // Pin1 Pin Change interrupt 1: Off
972:     // Pin2 Pin Change interrupt 1: Off
973:     // Pin3 Pin Change interrupt 1: Off
974:     // Pin4 Pin Change interrupt 1: Off
975:     // Pin5 Pin Change interrupt 1: Off
976:     // Pin6 Pin Change interrupt 1: Off
977:     // Pin7 Pin Change interrupt 1: Off
978:     PORTJ.INT1MASK=0x00;
979:
980:     // PORTK initialization
981:     // OUT register
982:     PORTK.OUT=0x00;
983:     // Pin0: Input
984:     // Pin1: Input
985:     // Pin2: Input
986:     // Pin3: Input
987:     // Pin4: Input
988:     // Pin5: Input
989:     // Pin6: Input
990:     // Pin7: Input
991:     PORTK.DIR=0x00;
992:     // Pin0 Output/Pull configuration: Totempole/No
993:     // Pin0 Input/Sense configuration: Sense both edges
994:     // Pin0 Inverted: Off
995:     // Pin0 Slew Rate Limitation: Off
996:     PORTK.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
997:     // Pin1 Output/Pull configuration: Totempole/No
998:     // Pin1 Input/Sense configuration: Sense both edges
999:     // Pin1 Inverted: Off
1000:    // Pin1 Slew Rate Limitation: Off
1001:    PORTK.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1002:    // Pin2 Output/Pull configuration: Totempole/No
1003:    // Pin2 Input/Sense configuration: Sense both edges
1004:    // Pin2 Inverted: Off
1005:    // Pin2 Slew Rate Limitation: Off
1006:    PORTK.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1007:    // Pin3 Output/Pull configuration: Totempole/No
1008:    // Pin3 Input/Sense configuration: Sense both edges
```

```
1009:     // Pin3 Inverted: Off
1010:     // Pin3 Slew Rate Limitation: Off
1011:     PORTK.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1012:     // Pin4 Output/Pull configuration: Totempole/No
1013:     // Pin4 Input/Sense configuration: Sense both edges
1014:     // Pin4 Inverted: Off
1015:     // Pin4 Slew Rate Limitation: Off
1016:     PORTK.PIN4CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1017:     // Pin5 Output/Pull configuration: Totempole/No
1018:     // Pin5 Input/Sense configuration: Sense both edges
1019:     // Pin5 Inverted: Off
1020:     // Pin5 Slew Rate Limitation: Off
1021:     PORTK.PIN5CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1022:     // Pin6 Output/Pull configuration: Totempole/No
1023:     // Pin6 Input/Sense configuration: Sense both edges
1024:     // Pin6 Inverted: Off
1025:     // Pin6 Slew Rate Limitation: Off
1026:     PORTK.PIN6CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1027:     // Pin7 Output/Pull configuration: Totempole/No
1028:     // Pin7 Input/Sense configuration: Sense both edges
1029:     // Pin7 Inverted: Off
1030:     // Pin7 Slew Rate Limitation: Off
1031:     PORTK.PIN7CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1032:     // Interrupt 0 level: Disabled
1033:     // Interrupt 1 level: Disabled
1034:     PORTK.INTCTRL=(PORTK.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
1035:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
1036:     // Pin0 Pin Change interrupt 0: Off
1037:     // Pin1 Pin Change interrupt 0: Off
1038:     // Pin2 Pin Change interrupt 0: Off
1039:     // Pin3 Pin Change interrupt 0: Off
1040:     // Pin4 Pin Change interrupt 0: Off
1041:     // Pin5 Pin Change interrupt 0: Off
1042:     // Pin6 Pin Change interrupt 0: Off
1043:     // Pin7 Pin Change interrupt 0: Off
1044:     PORTK.INT0MASK=0x00;
1045:     // Pin0 Pin Change interrupt 1: Off
1046:     // Pin1 Pin Change interrupt 1: Off
1047:     // Pin2 Pin Change interrupt 1: Off
1048:     // Pin3 Pin Change interrupt 1: Off
1049:     // Pin4 Pin Change interrupt 1: Off
1050:     // Pin5 Pin Change interrupt 1: Off
```

```
1051:     // Pin6 Pin Change interrupt 1: Off
1052:     // Pin7 Pin Change interrupt 1: Off
1053:     PORTK.INT1MASK=0x00;
1054:
1055:     // PORTQ initialization
1056:     // OUT register
1057:     PORTQ.OUT=0x00;
1058:     // Pin0: Input
1059:     // Pin1: Input
1060:     // Pin2: Input
1061:     // Pin3: Input
1062:     PORTQ.DIR=0x00;
1063:     // Pin0 Output/Pull configuration: Totempole/No
1064:     // Pin0 Input/Sense configuration: Sense both edges
1065:     // Pin0 Inverted: Off
1066:     // Pin0 Slew Rate Limitation: Off
1067:     PORTQ.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1068:     // Pin1 Output/Pull configuration: Totempole/No
1069:     // Pin1 Input/Sense configuration: Sense both edges
1070:     // Pin1 Inverted: Off
1071:     // Pin1 Slew Rate Limitation: Off
1072:     PORTQ.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1073:     // Pin2 Output/Pull configuration: Totempole/No
1074:     // Pin2 Input/Sense configuration: Sense both edges
1075:     // Pin2 Inverted: Off
1076:     // Pin2 Slew Rate Limitation: Off
1077:     PORTQ.PIN2CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1078:     // Pin3 Output/Pull configuration: Totempole/No
1079:     // Pin3 Input/Sense configuration: Sense both edges
1080:     // Pin3 Inverted: Off
1081:     // Pin3 Slew Rate Limitation: Off
1082:     PORTQ.PIN3CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1083:     // Interrupt 0 level: Disabled
1084:     // Interrupt 1 level: Disabled
1085:     PORTQ.INTCTRL=(PORTQ.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
1086:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
1087:     // Pin0 Pin Change interrupt 0: Off
1088:     // Pin1 Pin Change interrupt 0: Off
1089:     // Pin2 Pin Change interrupt 0: Off
1090:     // Pin3 Pin Change interrupt 0: Off
1091:     PORTQ.INT0MASK=0x00;
1092:     // Pin0 Pin Change interrupt 1: Off
```

```
1093:     // Pin1 Pin Change interrupt 1: Off
1094:     // Pin2 Pin Change interrupt 1: Off
1095:     // Pin3 Pin Change interrupt 1: Off
1096:     PORTQ.INT1MASK=0x00;
1097:
1098:     // PORTR initialization
1099:     // OUT register
1100:     PORTR.OUT=0x00;
1101:     // Pin0: Input
1102:     // Pin1: Input
1103:     PORTR.DIR=0x00;
1104:     // Pin0 Output/Pull configuration: Totempole/No
1105:     // Pin0 Input/Sense configuration: Sense both edges
1106:     // Pin0 Inverted: Off
1107:     // Pin0 Slew Rate Limitation: Off
1108:     PORTR.PIN0CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1109:     // Pin1 Output/Pull configuration: Totempole/No
1110:     // Pin1 Input/Sense configuration: Sense both edges
1111:     // Pin1 Inverted: Off
1112:     // Pin1 Slew Rate Limitation: Off
1113:     PORTR.PIN1CTRL=PORT_OPC_TOTEM_gc | PORT_ISC_BOTHEDGES_gc;
1114:     // Interrupt 0 level: Disabled
1115:     // Interrupt 1 level: Disabled
1116:     PORTR.INTCTRL=(PORTR.INTCTRL & (~(PORT_INT1LVL_gm | PORT_INT0LVL_gm))) |
1117:         PORT_INT1LVL_OFF_gc | PORT_INT0LVL_OFF_gc;
1118:     // Pin0 Pin Change interrupt 0: Off
1119:     // Pin1 Pin Change interrupt 0: Off
1120:     PORTR.INT0MASK=0x00;
1121:     // Pin0 Pin Change interrupt 1: Off
1122:     // Pin1 Pin Change interrupt 1: Off
1123:     PORTR.INT1MASK=0x00;
1124: }
1125:
1126: // Virtual Ports initialization
1127: void vports_init(void)
1128: {
1129:     // PORTA mapped to VPORT0
1130:     // PORTB mapped to VPORT1
1131:     PORTCFG.VPCTRLA=PORTCFG_VP1MAP_PORTB_gc | PORTCFG_VP0MAP_PORTA_gc;
1132:     // PORTC mapped to VPORT2
1133:     // PORTD mapped to VPORT3
1134:     PORTCFG.VPCTRLB=PORTCFG_VP3MAP_PORTD_gc | PORTCFG_VP2MAP_PORTC_gc;
```

```
1135: }
1136:
1137: // Disable a Timer/Counter type 1
1138: void tc1_disable(TC1_t *ptc)
1139: {
1140:     // Timer/Counter off
1141:     ptc->CTRLA=(ptc->CTRLA & (~TC1_CLKSEL_gm)) | TC_CLKSEL_OFF_gc;
1142:     // Issue a reset command
1143:     ptc->CTRLFSET=TC_CMD_RESET_gc;
1144: }
1145:
1146: // Timer/Counter TCD1 initialization
1147: void tcd1_init(void)
1148: {
1149:     unsigned char s;
1150:
1151:     // Note: The correct PORTD direction for the Compare Channels
1152:     // outputs is configured in the ports_init function.
1153:
1154:     // Save interrupts enabled/disabled state
1155:     s=SREG;
1156:     // Disable interrupts
1157:     #asm("cli")
1158:
1159:     // Disable and reset the timer/counter just to be sure
1160:     tc1_disable(&TCD1);
1161:     // Clock source: ClkPer/1
1162:     TCD1.CTRLA=(TCD1.CTRLA & (~TC1_CLKSEL_gm)) | TC_CLKSEL_DIV1_gc;
1163:     // Mode: Normal Operation, Overflow Int./Event on TOP
1164:     // Compare/Capture on channel A: Off
1165:     // Compare/Capture on channel B: Off
1166:     TCD1.CTRLB=(TCD1.CTRLB & (~(TC1_CCAEN_bm | TC1_CCBEN_bm | TC1_WGMODE_gm))) |
1167:         TC_WGMODE_NORMAL_gc;
1168:
1169:     // Capture event source: None
1170:     // Capture event action: None
1171:     TCD1.CTRLD=(TCD1.CTRLD & (~(TC1_EVACT_gm | TC1_EVSEL_gm))) |
1172:         TC_EVACT_OFF_gc | TC_EVSEL_OFF_gc;
1173:
1174:     // Overflow interrupt: High Level
1175:     // Error interrupt: Disabled
1176:     TCD1.INTCTRLA=(TCD1.INTCTRLA & (~(TC1_ERRINTLVL_gm | TC1_OVFINTLVL_gm))) |
```

```
1177:         TC_ERRINTLVL_OFF_gc | TC_OVFINTLVL_HI_gc;
1178:
1179:     // Compare/Capture channel A interrupt: Disabled
1180:     // Compare/Capture channel B interrupt: Disabled
1181:     TCD1.INTCTRLB=(TCD1.INTCTRLB & (~(TC1_CCBINTLVL_gm | TC1_CCAINTLVL_gm))) |
1182:         TC_CCBINTLVL_OFF_gc | TC_CCAINTLVL_OFF_gc;
1183:
1184:     // High resolution extension: Off
1185:     HIRESD.CTRLA&= ~HIRES_HREN1_bm;
1186:
1187:     // Clear the interrupt flags
1188:     TCD1.INTFLAGS=TCD1.INTFLAGS;
1189:     // Set counter register
1190:     TCD1.CNT=0x0000;
1191:     // Set period register
1192:     TCD1.PER=0x7CFF;
1193:     // Set channel A Compare/Capture register
1194:     TCD1.CCA=0x0000;
1195:     // Set channel B Compare/Capture register
1196:     TCD1.CCB=0x0000;
1197:
1198:     // Restore interrupts enabled/disabled state
1199:     SREG=s;
1200: }
1201:
1202: // USARTF0 initialization
1203: void usartf0_init(void)
1204: {
1205:     // Note: The correct PORTF direction for the RxD, TxD and XCK signals
1206:     // is configured in the ports_init function.
1207:
1208:     // Transmitter is enabled
1209:     // Set TxD=1
1210:     PORTF.OUTSET=0x08;
1211:
1212:     // Communication mode: Asynchronous USART
1213:     // Data bits: 8
1214:     // Stop bits: 1
1215:     // Parity: Disabled
1216:     USARTF0.CTRLC=USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc | USART_CHSIZE_8BIT_gc;
1217:
1218:     // Receive complete interrupt: Disabled
```

```
1219:     // Transmit complete interrupt: Disabled
1220:     // Data register empty interrupt: Disabled
1221:     USARTF0.CTRLA=(USARTF0.CTRLA & (~(USART_RXCINTLVL_gm | USART_TXCINTLVL_gm | USART_DREINTLVL_gm))) |
1222:         USART_RXCINTLVL_OFF_gc | USART_TXCINTLVL_OFF_gc | USART_DREINTLVL_OFF_gc;
1223:
1224:     // Required Baud rate: 115200
1225:     // Real Baud Rate: 115211,5 (x1 Mode), Error: 0,0 %
1226:     USARTF0.BAUDCTRLA=0x2E;
1227:     USARTF0.BAUDCTRLB=((0x09 << USART_BSCALE_gp) & USART_BSCALE_gm) | 0x08;
1228:
1229:     // Receiver: On
1230:     // Transmitter: On
1231:     // Double transmission speed mode: Off
1232:     // Multi-processor communication mode: Off
1233:     USARTF0.CTRLB=(USARTF0.CTRLB & (~(USART_RXEN_bm | USART_TXEN_bm | USART_CLK2X_bm | USART_MPCM_bm | USART_TXB8_bm))) |
1234:         USART_RXEN_bm | USART_TXEN_bm;
1235: }
1236:
1237: // Receive a character from USARTF0
1238: // USARTF0 is used as the default input device by the 'getchar' function
1239: #define _ALTERNATE_GETCHAR_
1240:
1241: #pragma used+
1242: char getchar(void)
1243: {
1244:     char data;
1245:     unsigned char status;
1246:
1247:     while (1)
1248:     {
1249:       while (((status=USARTF0.STATUS) & USART_RXCIF_bm) == 0);
1250:       data=USARTF0.DATA;
1251:       if ((status & (USART_FERR_bm | USART_PERR_bm | USART_BUFOVF_bm)) == 0) return data;
1252:     }
1253: }
1254: #pragma used-
1255:
1256: // Write a character to the USARTF0 Transmitter
1257: // USARTF0 is used as the default output device by the 'putchar' function
1258: #define _ALTERNATE_PUTCHAR_
1259:
1260: #pragma used+
```

```
1261: void putchar(char c)
1262: {
1263:     while ((USARTF0.STATUS & USART_DREIF_bm) == 0);
1264:     USARTF0.DATA=c;
1265: }
1266: #pragma used-
1267:
1268: // Timer/Counter TCD1 Overflow/Underflow interrupt service routine
1269: interrupt [TCD1_OVF_vect] void tcd1_overflow_isr(void)
1270: {
1271:     relogio++;  //Counts milliseconds (49 days capacity)
1272: }
1273:
1274: //*****************************************************
1275:
1276:
1277: void main(void)
1278: {
1279:
1280: //**********************SPAOFE*************************
1281:
1282:     unsigned char i;
1283:     unsigned char sim;
1284:     unsigned int temp1;
1285:     unsigned long int temp2;
1286:
1287: //**********************SPAOFE*************************
1288:
1289:     unsigned char n;
1290:     // Interrupt system initialization
1291:     // Optimize for speed
1292:     #pragma optsize-
1293:     // Make sure the interrupts are disabled
1294:     #asm("cli")
1295:     // Low level interrupt: On
1296:     // Round-robin scheduling for low level interrupt: On
1297:     // Medium level interrupt: On
1298:     // High level interrupt: On
1299:     // The interrupt vectors will be placed at the start of the Application FLASH section
1300:     n=(PMIC.CTRL & (~(PMIC_RREN_bm | PMIC_IVSEL_bm | PMIC_HILVLEN_bm | PMIC_MEDLVLEN_bm | PMIC_LOLVLEN_bm))) |
1301:         PMIC_LOLVLEN_bm | PMIC_RREN_bm | PMIC_MEDLVLEN_bm | PMIC_HILVLEN_bm;
1302:     CCP=CCP_IOREG_gc;
```

```
1303:     PMIC.CTRL=n;
1304:     // Set the default priority for round-robin scheduling
1305:     PMIC.INTPRI=0x00;
1306:     // Restore optimization for size if needed
1307:     #pragma optsize_default
1308:
1309:     // System clocks initialization
1310:     system_clocks_init();
1311:
1312:     // Ports initialization
1313:     ports_init();
1314:
1315:     // Virtual Ports initialization
1316:     vports_init();
1317:
1318:     // Timer/Counter TCD1 initialization
1319:     tcd1_init();
1320:
1321:     // USARTF0 initialization
1322:     usartf0_init();
1323:
1324:     // Globally enable interrupts
1325:     #asm("sei")
1326:
1327: //**********************SPAOFE*************************
1328:
1329:     temp2 = relogio;
1330:     for(i=0; i<SPAOFE.Sonar_Number; i++)
1331:     {
1332:         SPAOFE.Ti_1[i] = temp2;
1333:     }
1334:
1335:     // Each "sim" simulates a ranging set of the sensors
1336:     delay_ms(3000);
1337:     sim = 0;
1338:     for(i=0; i<SPAOFE.Sonar_Number; i++)                                    // For each sensor
1339:     {
1340:         temp1 = range_sonar_X(sim,SPAOFE.Sonar_Addr[i]);
1341:         temp2 = relogio;                                                    // Instant time
1342:         SPAOFE.DTi[i] = temp2 - SPAOFE.Ti_1[i];                             // Elapsed time
1343:         SPAOFE.Ti_1[i] = temp2;                                             // New inicial instant time
1344:         SPAOFE.Sonar_Distance_Change[i] = temp1 - SPAOFE.Sonar_Distance[i]; // Distance change
```

```
1345:         SPAOFE.Sonar_Distance[i] = temp1;                                    // New inicial distance
1346:         printf("Range=%i DTi=%u SDC=%i\r\n",temp1,SPAOFE.DTi[i],SPAOFE.Sonar_Distance_Change[i]);
1347:     }
1348:
1349:     General_Perception_Vector();
1350:     General_Perception_V_Change();
1351:
1352:     printf("General_Perception_Value = %f\r\n",SPAOFE.General_Perception_Value);
1353:     printf("General_Perception_Angle = %f\r\n",SPAOFE.General_Perception_Angle);
1354:     printf("General_Perception_Change = %f\r\n",SPAOFE.General_Perception_Change);
1355:
1356:     Perception_Value_Membership(SPAOFE.General_Perception_Value);
1357:
1358:     printf("VLP=%f\r\n",VLP);
1359:     printf("LP=%f\r\n",LP);
1360:     printf("MP=%f\r\n",MP);
1361:     printf("HP=%f\r\n",HP);
1362:     printf("VHP=%f\r\n",VHP);
1363:
1364:     //Msectors_Nao_Zero(PV,5,iPV,&nPV);
1365:
1366:     printf("nPV=%d\r\n",nPV);
1367:     printf("%d %d %d %d %d\r\n",iPV[0],iPV[1],iPV[2],iPV[3],iPV[4]);
1368:
1369:     Perception_Angle_Membership(SPAOFE.General_Perception_Angle);
1370:
1371:     printf("RB=%f\r\n",RB);
1372:     printf("RF=%f\r\n",RF);
1373:     printf("LF=%f\r\n",LF);
1374:     printf("LB=%f\r\n",LB);
1375:
1376:     //Msectors_Nao_Zero(&PA[0],4,&iPA[0],&nPA);
1377:
1378:     printf("nPA=%d\r\n",nPA);
1379:     printf("%d %d %d %d\r\n",iPA[0],iPA[1],iPA[2],iPA[3]);
1380:
1381:     Perception_Change_Membership(SPAOFE.General_Perception_Change);
1382:
1383:     printf("ZE=%f\r\n",ZE);
1384:     printf("LO=%f\r\n",LO);
1385:     printf("HI=%f\r\n",HI);
1386:
```

```
1387:     //Msectors_Nao_Zero(&PC[0],3,&iPC[0],&nPC);
1388:
1389:     printf("nPC=%d\r\n",nPC);
1390:     printf("%d %d %d\r\n",iPC[0],iPC[1],iPC[2]);
1391:
1392:     ad_aeternum;   // ******** Tested untill this point ********
1393:
1394:     // The variables needed to use the XMEGA/LM629 library in velocity mode are:
1395:     // New Velocity => SPAOFE.Velocity
1396:     // Direction (left,right or stright) => Direcao
1397:     // Gyration radius (if = 0.0 => turn on vertical axis) => Rg
1398:     // Brake (abruptly or smoothly) => Brake
1399:
1400:     Rules_Steer();
1401:
1402:     printf("nSteer = %d\r\n", nSteer);
1403:     printf("%d  %d  %d  %d  %d  %d\r\n", iSteer[0],iSteer[1],iSteer[2],iSteer[3],iSteer[4],iSteer[5]);
1404:
1405:     ftoa(CSteer[0],2,CSt0);
1406:     ftoa(CSteer[1],2,CSt1);
1407:     ftoa(CSteer[2],2,CSt2);
1408:     ftoa(CSteer[3],2,CSt3);
1409:     ftoa(CSteer[4],2,CSt4);
1410:     ftoa(CSteer[5],2,CSt5);
1411:
1412:     printf("%s  %s  %s  %s  %s  %s\r\n", CSt0,CSt1,CSt2,CSt3,CSt4,CSt5);
1413:
1414:     Steer_Crisp = Scaled_Defuzzy_Steer(-100.0,100.0,1.0);
1415:
1416:     if (Steer_Crisp == 0.0) Direcao = 0; // Stright
1417:     else if (Steer_Crisp > 0.0) Direcao = -1;   // Right
1418:     else Direcao = +1;  // Left
1419:     Steer_Crisp = fabs(Steer_Crisp);
1420:
1421:     Rg = 57.3 * SPAOFE.Velocity / Steer_Crisp;
1422:     if (Rg < (1.5 * eixo) && Rg >= eixo) Rg = 1.5 * eixo;
1423:     else if (Rg < eixo) Rg = 0.0;  // This signifies that the robot must rotate on its vertical axis
1424:     else;
1425:
1426:     Rules_Acceleration();
1427:
1428:     printf("nAcc = %d\r\n", nAcc);
```

```
1429:     printf("%d  %d  %d  %d  %d  %d\r\n", iAcc[0],iAcc[1],iAcc[2],iAcc[3],iAcc[4],iAcc[5]);
1430:
1431:     ftoa(CAcc[0],2,CSt0);
1432:     ftoa(CAcc[1],2,CSt1);
1433:     ftoa(CAcc[2],2,CSt2);
1434:     ftoa(CAcc[3],2,CSt3);
1435:     ftoa(CAcc[4],2,CSt4);
1436:     ftoa(CAcc[5],2,CSt5);
1437:
1438:     printf("%s  %s  %s  %s  %s  %s\r\n", CSt0,CSt1,CSt2,CSt3,CSt4,CSt5);
1439:
1440:     // New acceleration
1441:     Acceleration_Crisp = Scaled_Defuzzy_Acceleration(-SPAOFE.Maximum_Acceleration,SPAOFE.Maximum_Acceleration,0.5);
1442:
1443:     // New velocity
1444:     SPAOFE.Velocity = SPAOFE.Velocity + Acceleration_Crisp;
1445:
1446:     if (SPAOFE.Velocity > SPAOFE.Maximum_Velocity) SPAOFE.Velocity = SPAOFE.Maximum_Velocity;
1447:     else if (SPAOFE.Velocity < SPAOFE.Minimum_Velocity)
1448:     {
1449:         SPAOFE.Velocity = 0.0;
1450:         Brake = 10; // Stop smoothly
1451:     }
1452:     if (Acceleration_Crisp < -(0.875 * SPAOFE.Maximum_Acceleration))
1453:     {
1454:         SPAOFE.Velocity = 0.0;
1455:         Brake = 9;  // Stop abruptly
1456:     }
1457:
1458:     ftoa(Steer_Crisp,2,CharConv);
1459:     printf("Steer_Crisp = %s\r\n",CharConv);
1460:         ftoa(Acceleration_Crisp,2,CharConv);
1461:     printf("Acceleration_Crisp = %s\r\n",CharConv);
1462:     printf("Direction = %d\r\n",Direcao);
1463:     ftoa(Rg,2,CharConv);
1464:     printf("Rg = %s\r\n",CharConv);
1465:     ftoa(SPAOFE.Velocity,2,CharConv);
1466:     printf("New velocity = %s\r\n",CharConv);
1467:     printf("Brake = %d\r\n",Brake);
1468:
1469: //**********************SPAOFE*************************
1470:
```

```
1471:     ad_aeternum;
1472: }
1473:
1474: //*****************SPAOFE FUNCTIONS*********************
1475:
1476: void Msectors_Nao_Zero(float *Vetor,unsigned char Comp,unsigned char *Indices,unsigned char *Num)
1477: {
1478:     //Verifies the first "Comp" elements of the array "*Vetor" to verify which are non zero.
1479:     //The indices of the non zero elements are stored in the array "*Indices" and the indices
1480:     //quantity is stored in "*Num"
1481:
1482:     unsigned char i;
1483:
1484:     *Num = 0;
1485:     for(i=0;i<Comp;i++) if (Vetor[i] > 0.0) Indices[*Num++] = i;
1486: }
1487:
1488: //____________________________________________________________________________________________________
1489:
1490: void Rules_Steer(void)
1491: {
1492:     //Evaluates the 20 Steer rules. As inputs uses the Perception Value and Perception Angle membership sectors.
1493:     //Calculates the Steer sectors to be aggregated and it's cutoff (or scaled) value "CSteer[]".
1494:
1495:     unsigned char i;
1496:     unsigned char j;
1497:     unsigned char Regra;
1498:
1499:     nSteer = 0;
1500:
1501:       for(i=0;i<nPA;i++)                          // For all PA[] > zero
1502:        {
1503:         for(j=0;j<nPV;j++)                       // For all PV[] > zero
1504:          {
1505:               Regra = 10 * iPA[i] + iPV[j];      // Rule
1506:            switch (Regra)
1507:            {
1508:                case 0:                           // if a is RB and p is VLP then s is HR
1509:                    iSteer[nSteer] = 0;           // iSteer[] = 0 (HR sector index)
1510:                    break;
1511:                case 1:                           // if a is RB and p is LP then s is HR
1512:                    iSteer[nSteer] = 0;           // iSteer[] = 0 (HR sector index)
```

```
1513:                 break;
1514:             case 2:                     // if a is RB and p is MP then s is R
1515:                 iSteer[nSteer] = 1;     // iSteer[] = 1 (R sector index)
1516:                 break;
1517:             case 3:                     // if a is RB and p is HP then s is R
1518:                 iSteer[nSteer] = 1;     // iSteer[] = 1 (R sector index)
1519:                 break;
1520:             case 4:                     // if a is RB and p is VHP then s is C
1521:                 iSteer[nSteer] = 2;     // iSteer[] = 2 (C sector index)
1522:                 break;
1523:             case 10:                    // if a is RF and p is VLP then s is C
1524:                 iSteer[nSteer] = 2;     // iSteer[] = 2 (C sector index)
1525:                 break;
1526:             case 11:                    // if a is RF and p is LP then s is L
1527:                 iSteer[nSteer] = 3;     // iSteer[] = 3 (L sector index)
1528:                 break;
1529:             case 12:                    // if a is RF and p is MP then s is L
1530:                 iSteer[nSteer] = 3;     // iSteer[] = 3 (L sector index)
1531:                 break;
1532:             case 13:                    // if a is RF and p is HP then s is HL
1533:                 iSteer[nSteer] = 4;     // iSteer[] = 4 (HL sector index)
1534:                 break;
1535:             case 14:                    // if a is RF and p is VHP then s is HL
1536:                 iSteer[nSteer] = 4;     // iSteer[] = 4 (HL sector index)
1537:                 break;
1538:             case 20:                    // if a is LF and p is VLP then s is C
1539:                 iSteer[nSteer] = 2;     // iSteer[] = 2 (C sector index)
1540:                 break;
1541:             case 21:                    // if a is LF and p is LP then s is R
1542:                 iSteer[nSteer] = 1;     // iSteer[] = 1 (R sector index)
1543:                 break;
1544:             case 22:                    // if a is LF and p is MP then s is R
1545:                 iSteer[nSteer] = 1;     // iSteer[] = 1 (R sector index)
1546:                 break;
1547:             case 23:                    // if a is LF and p is HP then s is HR
1548:                 iSteer[nSteer] = 0;     // iSteer[] = 0 (HR sector index)
1549:                 break;
1550:             case 24:                    // if a is LF and p is VHP then s is HR
1551:                 iSteer[nSteer] = 0;     // iSteer[] = 0 (HR sector index)
1552:                 break;
1553:             case 30:                    // if a is LB and p is VLP then s is HL
1554:                 iSteer[nSteer] = 4;     // iSteer[] = 4 (HL sector index)
```

```
1555:                     break;
1556:                 case 31:                          // if a is LB and p is LP then s is HL
1557:                     iSteer[nSteer] = 4;             // iSteer[] = 4 (HL sector index)
1558:                     break;
1559:                 case 32:                          // if a is LB and p is MP then s is L
1560:                     iSteer[nSteer] = 3;             // iSteer[] = 3 (L sector index)
1561:                     break;
1562:                 case 33:                          // if a is LB and p is HP then s is L
1563:                     iSteer[nSteer] = 3;           // iSteer[] = 3 (L sector index)
1564:                     break;
1565:                 case 34:                          // if a is LB and p is VHP then s is C
1566:                     iSteer[nSteer] = 2;           // iSteer[] = 2 (C sector index)
1567:                     break;
1568:                 default:
1569:                     iSteer[nSteer] = 100;          // There is no rule. Aggregation will not be applied
1570:             }
1571:             CSteer[nSteer++] = zand(PA[iPA[i]],PV[iPV[j]]);    // Lower value
1572:         }
1573:     }
1574: }
1575:
1576: //_____________________________________________________________________________________________________________
1577:
1578: void Rules_Acceleration(void)
1579: {
1580:     //Evaluates the 11 Acceleration rules. As inputs uses the Perception Value and Perception Change membership sectors.
1581:     //Calculates the Acceleration sectors to be aggregated and it's cutoff (or scaled) value "CAcc[]".
1582:
1583:     unsigned char i;
1584:     unsigned char j;
1585:     unsigned char Regra;
1586:
1587:     nAcc = 0;
1588:
1589:     for(i=0;i<nPC;i++)                              // For all PC[] > zero
1590:      {
1591:           for(j=0;j<nPV;j++)                          // For all PV[] > zero
1592:         {
1593:              Regra = 10 * iPC[i] + iPV[j];
1594:             switch (Regra)
1595:             {
1596:                 case 0:                              // if dp is ZE and p is VLP then ac is Z
```

```
1597:                     iAcc[nAcc] = 2;                 // iAcc[] = 2 (Z sector index)
1598:                     break;
1599:                 case 1:                             // if dp is ZE and p is LP then ac is P
1600:                     iAcc[nAcc] = 3;                 // iAcc[] = 3 (P sector index)
1601:                     break;
1602:                 case 2:                             // if dp is ZE and p is MP then ac is P
1603:                     iAcc[nAcc] = 3;                 // iAcc[] = 3 (P sector index)
1604:                     break;
1605:                 case 3:                             // if dp is ZE and p is HP then ac is P
1606:                     iAcc[nAcc] = 3;                 // iAcc[] = 3 (P sector index)
1607:                     break;
1608:                 case 4:                             // if dp is ZE and p is VHP then ac is Z
1609:                     iAcc[nAcc] = 2;                 // iAcc[] = 2 (Z sector index)
1610:                     break;
1611:                 case 10:                           // if dp is LO and p is VLP then ac is EB
1612:                     iAcc[nAcc] = 0;                 // iAcc[] = 0 (EB sector index)
1613:                     break;
1614:                 case 11:                           // if dp is LO and p is LP then ac is B
1615:                     iAcc[nAcc] = 1;                 // iAcc[] = 1 (B sector index)
1616:                     break;
1617:                 case 12:                           // if dp is LO and p is MP then ac is Z
1618:                     iAcc[nAcc] = 2;                 // iAcc[] = 2 (Z sector index)
1619:                     break;
1620:                 case 13:                           // if dp is LO and p is HP then ac is B
1621:                     iAcc[nAcc] = 1;                 // iAcc[] = 1 (B sector index)
1622:                     break;
1623:                 case 14:                           // if dp is LO and p is VHP then ac is EB
1624:                     iAcc[nAcc] = 0;                 // iAcc[] = 0 (EB sector index)
1625:                     break;
1626: //                case 20:                          // if dp is HI and p is VLP then ac is P
1627: //                    iAcc[nAcc] = 3;               // iAcc[] = 3 (P sector index)
1628: //                    break;
1629: //                case 21:                          // if dp is HI and p is LP then ac is P
1630: //                    iAcc[nAcc] = 3;               // iAcc[] = 3 (P sector index)
1631: //                    break;
1632: //                case 22:                          // if dp is HI and p is MP then ac is Z
1633: //                    iAcc[nAcc] = 2;               // iAcc[] = 2 (Z sector index)
1634: //                    break;
1635:                 case 23:                           // if dp is HI and p is HP then ac is EB
1636:                     iAcc[nAcc] = 0;                 // iAcc[] = 0 (EB sector index)
1637:                     break;
1638: //                case 24:                          // if dp is HI and p is VHP then ac is EB
```

```
1639: //                    iAcc[nAcc] = 0;                    // iAcc[] = 0 (EB sector index)
1640: //                    break;
1641:                 default:
1642:                     iAcc[nAcc] = 100;                // There is no rule. Aggregation will not be applied
1643:             }
1644:             CAcc[nAcc++] = zand(PC[iPC[i]],PV[iPV[j]]); // Lower value
1645:         }
1646:     }
1647: }
1648:
1649: //_____________________________________________________________________________________________________________
1650:
1651: float Cutted_Defuzzy_Steer(float Xinicial,float Xfinal,float Step)
1652: {
1653:     //Calculates the crisp value of the output value Steer. This is done aggregating the
1654:     //cutted or scaled membership sectors and using the centroid method. The aggregation is done
1655:     //by the use of the OR for fuzzy sets (union). The centroid method calculates the
1656:     //gravity center of the resulting area.
1657:
1658:     unsigned char i;
1659:     unsigned char j;
1660:     unsigned int Namos;       // Number of steps "Step" in the interval "Xinicial" to "Xfinal"
1661:     float Fagregado[210];     // Resulting aggregated function
1662:     float X[210];           // X value for each step
1663:     float Snumerador;       // Integral of the momentum approximation
1664:     float Sdenominador;     // Integral of the area approximation
1665:
1666:     Namos = (unsigned int)((Xfinal - Xinicial) / Step);
1667:     X[0] = Xinicial;
1668:     Fagregado[0] = 0.0;
1669:
1670:     for (i=1;i<Namos;i++)
1671:     {
1672:         X[i] = X[i-1] + Step;
1673:         Fagregado[i] = 0.0;
1674:     }
1675:
1676:     for (i=0;i<nSteer;i++)
1677:     {
1678:         switch (iSteer[i])
1679:         {
1680:             case 0:        // HR
```

```
1681:                       for (j=0;j<Namos;j++)
1682:                       {
1683:                           Fagregado[j] = zor(Fagregado[j],zand(CSteer[i],fHR(X[j])));
1684:                       }
1685:                       break;
1686:               case 1:         // R
1687:                       for (j=0;j<Namos;j++)
1688:                       {
1689:                           Fagregado[j] = zor(Fagregado[j],zand(CSteer[i],fR(X[j])));
1690:                       }
1691:                       break;
1692:               case 2:         // C
1693:                       for (j=0;j<Namos;j++)
1694:                       {
1695:                           Fagregado[j] = zor(Fagregado[j],zand(CSteer[i],fC(X[j])));
1696:                       }
1697:                       break;
1698:               case 3:         // L
1699:                       for (j=0;j<Namos;j++)
1700:                       {
1701:                           Fagregado[j] = zor(Fagregado[j],zand(CSteer[i],fL(X[j])));
1702:                       }
1703:                       break;
1704:               case 4:         // HL
1705:                       for (j=0;j<Namos;j++)
1706:                       {
1707:                           Fagregado[j] = zor(Fagregado[j],zand(CSteer[i],fHL(X[j])));
1708:                       }
1709:                       break;
1710:               default:
1711:
1712:           }
1713:       }
1714:       Snumerador = Sdenominador = 0.0;
1715:       for (i=0;i<Namos;i++)
1716:       {
1717:           Snumerador = Snumerador + (Fagregado[i] * X[i]);
1718:           Sdenominador = Sdenominador + Fagregado[i];
1719:       }
1720:       return (Snumerador / Sdenominador);
1721: }
1722:
```

```
1723: //_____________________________________________________________________________________________
1724:
1725: float Scaled_Defuzzy_Steer(float Xinicial,float Xfinal,float Step)
1726: {
1727:     //Calculates the crisp value of the output value Steer. This is done aggregating the
1728:     //scaled or scaled membership sectors and using the centroid method. The aggregation is done
1729:     //by the use of the OR for fuzzy sets (union). The centroid method calculates the
1730:     //gravity center of the resulting area.
1731:
1732:     unsigned char i;
1733:     unsigned char j;
1734:     unsigned int Namos;       // Number of steps "Step" in the interval "Xinicial" to "Xfinal"
1735:     float Fagregado[210];    // Resulting aggregated function
1736:     float X[210];            // X value for each step
1737:     float Snumerador;       // Integral of the momentum approximation
1738:     float Sdenominador;       // Integral of the area approximation
1739:
1740:     Namos = (unsigned int)((Xfinal - Xinicial) / Step);    // Número de steps para o X
1741:     X[0] = Xinicial;
1742:     Fagregado[0] = 0.0;
1743:
1744:     for (i=1;i<Namos;i++)
1745:     {
1746:         X[i] = X[i-1] + Step;
1747:         Fagregado[i] = 0.0;
1748:     }
1749:
1750:     for (i=0;i<nSteer;i++)
1751:     {
1752:         switch (iSteer[i])
1753:         {
1754:              case 0:        // HR
1755:                     for (j=0;j<Namos;j++)
1756:                     {
1757:                         Fagregado[j] = zor(Fagregado[j],CSteer[i] * fHR(X[j]));
1758:                     }
1759:                     break;
1760:              case 1:        // R
1761:                     for (j=0;j<Namos;j++)
1762:                     {
1763:                         Fagregado[j] = zor(Fagregado[j],CSteer[i] * fR(X[j]));
1764:                     }
```

```
1765:                     break;
1766:             case 2:         // C
1767:                     for (j=0;j<Namos;j++)
1768:                     {
1769:                         Fagregado[j] = zor(Fagregado[j],CSteer[i] * fC(X[j]));
1770:                     }
1771:                     break;
1772:             case 3:         // L
1773:                     for (j=0;j<Namos;j++)
1774:                     {
1775:                         Fagregado[j] = zor(Fagregado[j],CSteer[i] * fL(X[j]));
1776:                     }
1777:                     break;
1778:             case 4:         // HL
1779:                     for (j=0;j<Namos;j++)
1780:                     {
1781:                         Fagregado[j] = zor(Fagregado[j],CSteer[i] * fHL(X[j]));
1782:                     }
1783:                     break;
1784:             default:
1785:
1786:         }
1787:     }
1788:     Snumerador = Sdenominador = 0.0;
1789:     for (i=0;i<Namos;i++)
1790:     {
1791:         Snumerador = Snumerador + (Fagregado[i] * X[i]);
1792:         Sdenominador = Sdenominador + Fagregado[i];
1793:     }
1794:     return (Snumerador / Sdenominador); // Centroid - Crisp value for Steer
1795: }
1796:
1797: //_____________________________________________________________________________________________________________
1798:
1799: float Cutted_Defuzzy_Acceleration(float Xinicial,float Xfinal,float Step)
1800: {
1801:     //Calculates the crisp value of the output value Acceleration. This is done aggregating the
1802:     //cutted membership sectors and using the centroid method. The aggregation is done
1803:     //by the use of the OR for fuzzy sets (union). The centroid method calculates the
1804:     //gravity center of the resulting area.
1805:
1806:     unsigned char i;
```

```
1807:     unsigned char j;
1808:     unsigned int Namos;     // Number of steps "Step" in the interval "Xinicial" to "Xfinal"
1809:     float Fagregado[210];    // Resulting aggregated function
1810:     float X[210];              // X value for each step
1811:     float Snumerador;         // Integral of the momentum approximation
1812:     float Sdenominador;     // Integral of the area approximation
1813:
1814:     Namos = (unsigned int)((Xfinal - Xinicial) / Step);
1815:     X[0] = Xinicial;
1816:     Fagregado[0] = 0.0;
1817:
1818:     for (i=1;i<Namos;i++)
1819:     {
1820:         X[i] = X[i-1] + Step;
1821:         Fagregado[i] = 0.0;
1822:     }
1823:
1824:     for (i=0;i<nAcc;i++)
1825:     {
1826:         switch (iAcc[i])
1827:         {
1828:              case 0:        // EB
1829:                     for (j=0;j<Namos;j++)
1830:                     {
1831:                         Fagregado[j] = zor(Fagregado[j],zand(CAcc[i],fEB(X[j])));
1832:                     }
1833:                     break;
1834:              case 1:        // B
1835:                     for (j=0;j<Namos;j++)
1836:                     {
1837:                         Fagregado[j] = zor(Fagregado[j],zand(CAcc[i],fB(X[j])));
1838:                     }
1839:                     break;
1840:              case 2:        // Z
1841:                     for (j=0;j<Namos;j++)
1842:                     {
1843:                         Fagregado[j] = zor(Fagregado[j],zand(CAcc[i],fZ(X[j])));
1844:                     }
1845:                     break;
1846:              case 3:        // P
1847:                     for (j=0;j<Namos;j++)
1848:                     {
```

```
1849:                         Fagregado[j] = zor(Fagregado[j],zand(CAcc[i],fP(X[j])));
1850:                     }
1851:                     break;
1852:             default:
1853:
1854:         }
1855:     }
1856:     Snumerador = Sdenominador = 0.0;
1857:     for (i=0;i<Namos;i++)
1858:     {
1859:         Snumerador = Snumerador + (Fagregado[i] * X[i]);
1860:         Sdenominador = Sdenominador + Fagregado[i];
1861:     }
1862:     return (Snumerador / Sdenominador); // Centroid - Crisp value for acceleration
1863: }
1864:
1865: //_____________________________________________________________________________________________________
1866:
1867: float Scaled_Defuzzy_Acceleration(float Xinicial,float Xfinal,float Step)
1868: {
1869:     //Calculates the crisp value of the output value Acceleration. This is done aggregating the
1870:     //scaled membership sectors and using the centroid method. The aggregation is done
1871:     //by the use of the OR for fuzzy sets (union). The centroid method calculates the
1872:     //gravity center of the resulting area.
1873:
1874:     unsigned char i;
1875:     unsigned char j;
1876:     unsigned int Namos;        // Number of steps "Step" in the interval "Xinicial" to "Xfinal"
1877:     float Fagregado[210];    // Resulting aggregated function
1878:     float X[210];            // X value for each step
1879:     float Snumerador;          // Integral of the momentum approximation
1880:     float Sdenominador;      // Integral of the area approximation
1881:
1882:     Namos = (unsigned int)((Xfinal - Xinicial) / Step);
1883:     X[0] = Xinicial;
1884:     Fagregado[0] = 0.0;
1885:
1886:     for (i=1;i<Namos;i++)
1887:     {
1888:         X[i] = X[i-1] + Step;
1889:         Fagregado[i] = 0.0;
1890:     }
```

```
1891:
1892:     for (i=0;i<nAcc;i++)
1893:     {
1894:         switch (iAcc[i])
1895:         {
1896:              case 0:        // EB
1897:                     for (j=0;j<Namos;j++)
1898:                     {
1899:                         Fagregado[j] = zor(Fagregado[j],CAcc[i] * fEB(X[j]));
1900:                     }
1901:                     break;
1902:              case 1:        // B
1903:                     for (j=0;j<Namos;j++)
1904:                     {
1905:                         Fagregado[j] = zor(Fagregado[j],CAcc[i] * fB(X[j]));
1906:                     }
1907:                     break;
1908:              case 2:        // Z
1909:                     for (j=0;j<Namos;j++)
1910:                     {
1911:                         Fagregado[j] = zor(Fagregado[j],CAcc[i] * fZ(X[j]));
1912:                     }
1913:                     break;
1914:              case 3:        // P
1915:                     for (j=0;j<Namos;j++)
1916:                     {
1917:                         Fagregado[j] = zor(Fagregado[j],CAcc[i] * fP(X[j]));
1918:                     }
1919:                     break;
1920:              default:
1921:
1922:         }
1923:     }
1924:     Snumerador = Sdenominador = 0.0;
1925:     for (i=0;i<Namos;i++)
1926:     {
1927:         Snumerador = Snumerador + (Fagregado[i] * X[i]);
1928:         Sdenominador = Sdenominador + Fagregado[i];
1929:     }
1930:     return (Snumerador / Sdenominador);    // Centroid - Crisp value for acceleration
1931: }
1932:
```

```
1933: //_____________________________________________________________________________________________________________
1934:
1935: float fHR(float x)
1936: {
1937:     //Output function Hard_Right for Steer
1938:     //x => Steer in deg/sec
1939:
1940:     if ((x < -100.0) || (x >= -30.0)) return (0.0);
1941:     else if ((x >= -100.0) && (x <= -60.0)) return (1.0);
1942:     else return (1.0 - ((x + 60.0)/30.0));
1943: }
1944:
1945: //_____________________________________________________________________________________________________________
1946:
1947: float fR(float x)
1948: {
1949:     //Output function Right for Steer
1950:     //x => Steer in deg/sec
1951:
1952:     if ((x <= -60.0) || (x >= 0.0)) return (0.0);
1953:     else if ((x > -60.0) && (x <= -30.0)) return ((x + 60.0)/30.0);
1954:     else return (1.0 - ((x + 30.0)/30.0));
1955: }
1956:
1957: //_____________________________________________________________________________________________________________
1958:
1959: float fC(float x)
1960: {
1961:     //Output function Center for Steer
1962:     //x => Steer in deg/sec
1963:
1964:     if ((x <= -30.0) || (x >= 30.0)) return (0.0);
1965:     else if ((x > -30.0) && (x <= 0.0)) return ((x + 30.0)/30.0);
1966:     else return (1.0 - (x/30.0));
1967: }
1968:
1969: //_____________________________________________________________________________________________________________
1970: float fL(float x)
1971: {
1972:     //Output function Left for Steer
1973:     //x => Steer in deg/sec
1974:
```

```
1975:     if ((x <= 0.0) || (x >= 60.0)) return (0.0);
1976:     else if ((x > 0.0) && (x <= 30.0)) return (x/30.0);
1977:     else return (1.0 - ((x - 30.0)/30.0));
1978: }
1979:
1980: //_____________________________________________________________________________________________________________
1981:
1982: float fHL(float x)
1983: {
1984:     //Output function Hard_Left for Steer
1985:     //x => Steer in deg/sec
1986:
1987:     if ((x <= 30.0) || (x > 100.0)) return (0.0);
1988:     else if ((x > 30.0) && (x < 60.0)) return ((x - 30.0)/30.0);
1989:     else return (1.0);
1990: }
1991:
1992: //_____________________________________________________________________________________________________________
1993:
1994: float fEB(float x)
1995: {
1996:     //Output function Emergency_Brake for Acceleration
1997:     //x => Acceleration (m/s^2)
1998:     //acelmax => maximum acceleration (cm/sec^2)
1999:
2000:     if ((x < -SPAOFE.Maximum_Acceleration) || (x >= -(0,875 * SPAOFE.Maximum_Acceleration))) return 0.0;
2001:     else return (1.0 - ((x + SPAOFE.Maximum_Acceleration)/(SPAOFE.Maximum_Acceleration - 0,875 *
      SPAOFE.Maximum_Acceleration)));
2002: }
2003:
2004: //_____________________________________________________________________________________________________________
2005: float fB(float x)
2006: {
2007:     //Output function Brake for Acceleration
2008:     //x => Acceleration (m/s^2)
2009:     //acelmax => maximum acceleration (cm/sec^2)
2010:
2011:     if ((x <= -SPAOFE.Maximum_Acceleration) || (x >= 0.0)) return (0.0);
2012:     else if ((x > -SPAOFE.Maximum_Acceleration) && (x < -(0.625 * SPAOFE.Maximum_Acceleration))) return ((x -
      SPAOFE.Maximum_Acceleration)/(0.375 * SPAOFE.Maximum_Acceleration));
2013:     else if ((x >= -(0.625 * SPAOFE.Maximum_Acceleration)) && (x <= -(0.25 * SPAOFE.Maximum_Acceleration))) return 1.0;
2014:     else return (1.0 - (x + 0.25 * SPAOFE.Maximum_Acceleration)/(0.25 * SPAOFE.Maximum_Acceleration));
```

```
2015: }
2016:
2017: //_____________________________________________________________________________________________________________
2018:
2019: float fZ(float x)
2020: {
2021:     //Output function Zero for Acceleration
2022:     //x => Acceleration (m/s^2)
2023:
2024:     if ((x <= -(0.375 * SPAOFE.Maximum_Acceleration)) || (x >= (0.375 * SPAOFE.Maximum_Acceleration))) return (0.0);
2025:     else if ((x > -(0.375 * SPAOFE.Maximum_Acceleration)) && (x <= 0.0)) return ((x + 0.375 *
      SPAOFE.Maximum_Acceleration)/(0.375 * SPAOFE.Maximum_Acceleration));
2026:     else return (1.0 - (x/(0.375 * SPAOFE.Maximum_Acceleration)));
2027: }
2028:
2029: //_____________________________________________________________________________________________________________
2030: float fP(float x)
2031: {
2032:     //Output function Positive for Acceleration
2033:     //x => Acceleration (m/s^2)
2034:
2035:     if ((x <= 0.0) || (x > SPAOFE.Maximum_Acceleration)) return (0.0);
2036:     else if ((x > 0.0) && (x < (0.25 * SPAOFE.Maximum_Acceleration))) return (x/(0.25 * SPAOFE.Maximum_Acceleration));
2037:     else return (1.0);
2038: }
2039:
2040: //_____________________________________________________________________________________________________________
2041:
2042: unsigned int range_sonar_X(unsigned char sim, unsigned char ender)
2043: {
2044:     //Simulates the ranging of the sonar with address "ender".
2045:     //Returns the read distance
2046:     //sim => case simulated
2047:     //ender => sonar I2c address
2048:
2049:     unsigned int data;
2050:
2051:     switch (sim)
2052:     {
2053:         case 0: // Front Left obstacle
2054:             switch (ender)
2055:             {
```

```
2056:                 case 0xE0:
2057:                     data = 100;
2058:                     break;
2059:                 case 0xE2:
2060:                     data = 1000;
2061:                     break;
2062:                 case 0xE4:
2063:                     data = 1000;
2064:                     break;
2065:                 case 0xE6:
2066:                     data = 1000;
2067:                     break;
2068:                 case 0xE8:
2069:                     data = 1000;
2070:                     break;
2071:                 case 0xEA:
2072:                     data = 1000;
2073:                     break;
2074:                 case 0xEC:
2075:                     data = 1000;
2076:                     break;
2077:                 case 0xEE:
2078:                     data = 100;
2079:                     break;
2080:             }
2081:             break;
2082:         case 1: // Front Right obstacle
2083:             switch (ender)
2084:             {
2085:                 case 0xE0:
2086:                     data = 80;
2087:                     break;
2088:                 case 0xE2:
2089:                     data = 100;
2090:                     break;
2091:                 case 0xE4:
2092:                     data = 150;
2093:                     break;
2094:                 case 0xE6:
2095:                     data = 1000;
2096:                     break;
2097:                 case 0xE8:
```

```
2098:                     data = 1000;
2099:                     break;
2100:                 case 0xEA:
2101:                     data = 1000;
2102:                     break;
2103:                 case 0xEC:
2104:                     data = 1000;
2105:                     break;
2106:                 case 0xEE:
2107:                     data = 1000;
2108:                     break;
2109:             }
2110:             break;
2111:         case 2: // Front Left and Front Right obstacle
2112:             switch (ender)
2113:             {
2114:                 case 0xE0:
2115:                     data = 70;
2116:                     break;
2117:                 case 0xE2:
2118:                     data = 80;
2119:                     break;
2120:                 case 0xE4:
2121:                     data = 90;
2122:                     break;
2123:                 case 0xE6:
2124:                     data = 1000;
2125:                     break;
2126:                 case 0xE8:
2127:                     data = 1000;
2128:                     break;
2129:                 case 0xEA:
2130:                     data = 1000;
2131:                     break;
2132:                 case 0xEC:
2133:                     data = 1000;
2134:                     break;
2135:                 case 0xEE:
2136:                     data = 150;
2137:                     break;
2138:             }
2139:             break;
```

```
2140:         case 3: // Lateral Right obstacle
2141:             switch (ender)
2142:             {
2143:                 case 0xE0:
2144:                     data = 1000;
2145:                     break;
2146:                 case 0xE2:
2147:                     data = 150;
2148:                     break;
2149:                 case 0xE4:
2150:                     data = 100;
2151:                     break;
2152:                 case 0xE6:
2153:                     data = 150;
2154:                     break;
2155:                 case 0xE8:
2156:                     data = 1000;
2157:                     break;
2158:                 case 0xEA:
2159:                     data = 1000;
2160:                     break;
2161:                 case 0xEC:
2162:                     data = 1000;
2163:                     break;
2164:                 case 0xEE:
2165:                     data = 1000;
2166:                     break;
2167:             }
2168:             break;
2169:     }
2170:     if (data <= SPAOFE.Minimum_Distance) SPAOFE.Sonar_Condition[ender] = 1;
2171:     else SPAOFE.Sonar_Condition[ender] = 0;
2172:     delay_ms(65);
2173:     return data;
2174: }
2175:
2176: //_____________________________________________________________________________________________________________
2177:
2178: void General_Perception_Vector(void)
2179: {
2180:     //Calculates the value and direction of the General Perception vector using the individual Perception vectors
2181:     //of the 8 sensors. The velocity vector is at 90º. Are calculated: General_Perception_Value (0.0 to 1.0 -
```

```
2182:     //normalized value); General_Perception_Angle (-180.0 to 180.0 - direction of the General Perception vector
2183:     //relative to the velocity vector. Negative to the right, positive to the left and 0 on heading)
2184:
2185:     float Xp;
2186:     float Yp;
2187:     float Pt;
2188:     const float to_radians = 0.0174532925199;
2189:
2190:     unsigned char i;
2191:
2192:     SPAOFE.General_Perception_Value = 0.0;
2193:     Xp = 0.0;
2194:     Yp = 0.0;
2195:     //For each sensor
2196:     for (i=0; i<SPAOFE.Sonar_Number; i++)
2197:     {
2198:         if (SPAOFE.Sonar_Distance[i] > SPAOFE.Maximum_Distance) Pt = 0.0;
2199:         else if (SPAOFE.Sonar_Distance[i] <= SPAOFE.Minimum_Distance) Pt = 1.0;
2200:         else Pt = (float)(SPAOFE.Maximum_Distance - SPAOFE.Sonar_Distance[i])/(float)(SPAOFE.Maximum_Distance -
      SPAOFE.Minimum_Distance);
2201:         SPAOFE.Perception_Value[i]= Pt;
2202:         if (Pt > SPAOFE.General_Perception_Value) SPAOFE.General_Perception_Value = Pt;
2203:         if ((Pt > 0.0) && (Pt <= 1.0))
2204:         {
2205:             Xp = Xp + (Pt * cos(SPAOFE.Sonar_Angle[i] * to_radians));
2206:             Yp = Yp + (Pt * sin(SPAOFE.Sonar_Angle[i] * to_radians));
2207:         }
2208:         printf("Pt=%f Xp=%f Yp=%f GPV=%f\r\n",Pt,Xp,Yp,SPAOFE.General_Perception_Value);
2209:     }
2210:     //Heading
2211:     if ((Xp == 0.0) && (Yp > 0.0)) SPAOFE.General_Perception_Angle = 0.0;
2212:     //Back
2213:     else if ((Xp == 0.0) && (Yp < 0.0)) SPAOFE.General_Perception_Angle = 180.0;
2214:     //Perpendicular right
2215:     else if ((Yp == 0.0) && (Xp > 0.0)) SPAOFE.General_Perception_Angle = -90.0;
2216:     //Perpendicular left
2217:     else if ((Yp == 0.0) && (Xp < 0.0)) SPAOFE.General_Perception_Angle = 90.0;
2218:     //Other
2219:     else SPAOFE.General_Perception_Angle = (atan2(Yp,Xp) / to_radians);
2220:
2221:     printf("GPA=%f\r\n",SPAOFE.General_Perception_Angle);
2222:
```

```
2223:     if ((Xp < 0.0) && (Yp < 0.0)) SPAOFE.General_Perception_Angle = 270 + SPAOFE.General_Perception_Angle;
2224:     else SPAOFE.General_Perception_Angle = SPAOFE.General_Perception_Angle - 90.0;
2225:
2226:     printf("GPA=%f\r\n",SPAOFE.General_Perception_Angle);
2227: }
2228: //_____________________________________________________________________________________________________________
2229:
2230: void General_Perception_V_Change(void)
2231: {
2232:     //Calculates the variation of the General Perception value. Only the positive variation is considered.
2233:
2234:     unsigned char i;
2235:     float temp;
2236:
2237:     SPAOFE.General_Perception_Change = 0.0;
2238:     //For each sensor, verify the positive variation
2239:     for (i=0; i<SPAOFE.Sonar_Number; i++)
2240:     {
2241:         if (SPAOFE.Sonar_Distance_Change[i] < 0)
2242:         {
2243:             temp = - ((float)SPAOFE.Sonar_Distance_Change[i] / (((float)SPAOFE.DTi[i] / 1000.0) *
      SPAOFE.Maximum_Velocity));
2244:             temp = fmin(1.0,temp);
2245:         }
2246:         else temp = 0.0;
2247:         if (temp > SPAOFE.General_Perception_Change) SPAOFE.General_Perception_Change = temp;
2248:     }
2249: }
2250:
2251: //_____________________________________________________________________________________________________________
2252:
2253: bool Perception_Change_Membership(float Value)
2254: {
2255:     //Calculates the membership level for the normalized General Perception Change value.
2256:     //ZE (0) - membership level for Perception Change zero
2257:     //LO (1) - membership level for low Perception Change
2258:     //HI (2) - membership level for high Perception Change
2259:     //Value => change in lenght of the General Perception vector
2260:
2261:     float x1[4],x2[4],xp1[5],xp2[5];
2262:     unsigned char i,inter_i;
2263:
```

```
2264:     xp1[0] = 0.0;
2265:     xp2[0] = 0.2;
2266:     xp1[1] = 0.2;
2267:     xp2[1] = 0.3;
2268:     xp1[2] = 0.3;
2269:     xp2[2] = 0.45;
2270:     xp1[3] = 0.45;
2271:     xp2[3] = 0.75;
2272:     xp1[4] = 0.75;
2273:     xp2[4] = 1.0;
2274:
2275:     x1[0] = 0.0;
2276:     x2[0] = 0.3;
2277:     x1[1] = 0.0;
2278:     x2[1] = 0.2;
2279:     x1[2] = 0.2;
2280:     x2[2] = 0.45;
2281:     x1[3] = 0.2;
2282:     x2[3] = 0.75;
2283:
2284:     inter_i = 255;
2285:     for (i=0;i<5;i++)
2286:     {
2287:         if ((Value >= xp1[i]) && (Value <= xp2[i])) {inter_i = i; break;}
2288:     }
2289:      if (inter_i == 255) return False;
2290:     switch (inter_i)
2291:     {
2292:         case 0:
2293:             nPC = 2;
2294:             ZE = reta_down(0,Value);
2295:             LO = reta_up(1,Value);
2296:             iPC[0] = 1;
2297:             iPC[1] = 1;
2298:             iPC[2] = 0;
2299:             return True;
2300:         case 1:
2301:             nPC = 3;
2302:             ZE = reta_down(0,Value);
2303:             LO = reta_down(2,Value);
2304:             HI = reta_up(3,Value);
2305:             iPC[0] = 1;
```

```
2306:             iPC[1] = 1;
2307:             iPC[2] = 1;
2308:             return True;
2309:         case 2:
2310:             nPC = 2;
2311:             LO = reta_down(2,Value);
2312:             HI = reta_up(3,Value);
2313:             iPC[0] = 0;
2314:             iPC[1] = 1;
2315:             iPC[2] = 1;
2316:             return True;
2317:         case 3:
2318:             nPC = 1;
2319:             HI = reta_up(3,Value);
2320:             iPC[0] = 0;
2321:             iPC[1] = 0;
2322:             iPC[2] = 1;
2323:             return True;
2324:         case 4:
2325:             nPC = 1;
2326:             HI = 1.0;
2327:             iPC[0] = 0;
2328:             iPC[1] = 0;
2329:             iPC[2] = 1;
2330:             return True;
2331:     }
2332: }
2333:
2334: //_____________________________________________________________________________________________________________
2335:
2336: bool Perception_Angle_Membership(float Angle)
2337: {
2338:     //Calculates the membership levels and sectors for the General Perception angle.
2339:     //RB (0) - membership level in the Right_Back sector
2340:     //RF (1) - membership level in the Right_Front sector
2341:     //LF (2) - membership level in the Left_Front sector
2342:     //LB (3) - membership level in the Left_Back sector
2343:     //Angle => direction of the General Perception vector
2344:
2345:     float x1[7],x2[7],xp1[10],xp2[10];
2346:     unsigned char i,inter_i;
2347:
```

```
2348:     xp1[0] = -180.0;
2349:     xp2[0] = -135.0;
2350:     xp1[1] = -135.0;
2351:     xp2[1] = -120.0;
2352:     xp1[2] = -120.0;
2353:     xp2[2] = -60.0;
2354:     xp1[3] = -60.0;
2355:     xp2[3] = -45.0;
2356:     xp1[4] = -45.0;
2357:     xp2[4] = 0.0;
2358:     xp1[5] = 0.0;
2359:     xp2[5] = 45.0;
2360:     xp1[6] = 45.0;
2361:     xp2[6] = 60.0;
2362:     xp1[7] = 60.0;
2363:     xp2[7] = 120.0;
2364:     xp1[8] = 120.0;
2365:     xp2[8] = 135.0;
2366:     xp1[9] = 135.0;
2367:     xp2[9] = 180.0;
2368:
2369:     x1[0] = -225.0;
2370:     x2[0] = -135.0;
2371:     x1[1] = -135.0;
2372:     x2[1] = -60.0;
2373:     x1[2] = -120.0;
2374:     x2[2] = -45.0;
2375:     x1[3] = -45.0;
2376:     x2[3] = 45.0;
2377:     x1[4] = 45.0;
2378:     x2[4] = 120.0;
2379:     x1[5] = 60.0;
2380:     x2[5] = 135.0;
2381:     x1[6] = 135.0;
2382:     x2[6] = 225.0;
2383:
2384:     inter_i = 255;
2385:     for (i=0;i<10;i++)
2386:     {
2387:         if ((Angle >= xp1[i]) && (Angle <= xp2[i])) {inter_i = i; break;}
2388:     }
2389:      if (inter_i == 255) return False;
```

```
2390:     switch (inter_i)
2391:     {
2392:         case 0:
2393:             nPA = 1;
2394:             RB = reta_up(0,Angle);
2395:             iPA[0] = 1;
2396:             iPA[1] = 0;
2397:             iPA[2] = 0;
2398:             iPA[3] = 0;
2399:             return True;
2400:         case 1:
2401:             nPA = 1;
2402:             RB = reta_down(1,Angle);
2403:             iPA[0] = 1;
2404:             iPA[1] = 0;
2405:             iPA[2] = 0;
2406:             iPA[3] = 0;
2407:             return True;
2408:         case 2:
2409:             nPA = 2;
2410:             RB = reta_down(1,Angle);
2411:             RF = reta_up(2,Angle);
2412:             iPA[0] = 1;
2413:             iPA[1] = 1;
2414:             iPA[2] = 0;
2415:             iPA[3] = 0;
2416:             return True;
2417:         case 3:
2418:             nPA = 1;
2419:             RF = reta_up(2,Angle);
2420:             iPA[0] = 0;
2421:             iPA[1] = 1;
2422:             iPA[2] = 0;
2423:             iPA[3] = 0;
2424:             return True;
2425:         case 4:
2426:             nPA = 1;
2427:             RF = reta_down(3,Angle);
2428:             iPA[0] = 0;
2429:             iPA[1] = 1;
2430:             iPA[2] = 0;
2431:             iPA[3] = 0;
```

```
2432:             return True;
2433:         case 5:
2434:             nPA = 1;
2435:             LF = reta_up(3,Angle);
2436:             iPA[0] = 0;
2437:             iPA[1] = 0;
2438:             iPA[2] = 1;
2439:             iPA[3] = 0;
2440:             return True;
2441:         case 6:
2442:             nPA = 1;
2443:             LF = reta_down(4,Angle);
2444:             iPA[0] = 0;
2445:             iPA[1] = 0;
2446:             iPA[2] = 1;
2447:             iPA[3] = 0;
2448:             return True;
2449:         case 7:
2450:             nPA = 2;
2451:             LF = reta_down(4,Angle);
2452:             LB = reta_up(5,Angle);
2453:             iPA[0] = 0;
2454:             iPA[1] = 0;
2455:             iPA[2] = 1;
2456:             iPA[3] = 1;
2457:             return True;
2458:         case 8:
2459:             nPA = 1;
2460:             LB = reta_up(5,Angle);
2461:             iPA[0] = 0;
2462:             iPA[1] = 0;
2463:             iPA[2] = 0;
2464:             iPA[3] = 1;
2465:             return True;
2466:         case 9:
2467:             nPA = 1;
2468:             LB = reta_down(6,Angle);
2469:             iPA[0] = 0;
2470:             iPA[1] = 0;
2471:             iPA[2] = 0;
2472:             iPA[3] = 1;
2473:             return True;
```

```
2474:     }
2475: }
2476:
2477: //_____________________________________________________________________________________________
2478:
2479: bool Perception_Value_Membership(float Value)
2480: {
2481:     //Calculates the membership level for the normalized General Perception value.
2482:     //VLP (0) - membership level for Very_Low Perception
2483:     //LP (1) - membership level for Low Perception
2484:     //MP (2) - membership level for Medium Perception
2485:     //HP (3) - membership level for High Perception
2486:     //VHP (4) - membership level for Very_High Perception
2487:     //Value => lenght of the General Perception vector
2488:
2489:
2490:     float x1[5],x2[5],xp1[8],xp2[8];
2491:     unsigned char i,inter_i;
2492:
2493:     xp1[0] = 0.0;
2494:     xp2[0] = 0.2;
2495:     xp1[1] = 0.2;
2496:     xp2[1] = 0.3;
2497:     xp1[2] = 0.3;
2498:     xp2[2] = 0.4;
2499:     xp1[3] = 0.4;
2500:     xp2[3] = 0.5;
2501:     xp1[4] = 0.5;
2502:     xp2[4] = 0.6;
2503:     xp1[5] = 0.6;
2504:     xp2[5] = 0.7;
2505:     xp1[6] = 0.7;
2506:     xp2[6] = 0.8;
2507:     xp1[7] = 0.8;
2508:     xp2[7] = 1.0;
2509:
2510:     x1[0] = 0.2;
2511:     x2[0] = 0.4;
2512:     x1[1] = 0.3;
2513:     x2[1] = 0.5;
2514:     x1[2] = 0.4;
2515:     x2[2] = 0.6;
```

```
2516:     x1[3] = 0.5;
2517:     x2[3] = 0.7;
2518:     x1[4] = 0.6;
2519:     x2[4] = 0.8;
2520:
2521:     inter_i = 255;
2522:     for (i=0;i<8;i++)
2523:     {
2524:         if ((Value >= xp1[i]) && (Value <= xp2[i])) {inter_i = i; break;}
2525:     }
2526:     if (inter_i == 255) return False;
2527:     switch (inter_i)
2528:     {
2529:         case 0:
2530:             nPV = 1;
2531:             VLP = 1.0;
2532:             iPV[0] = 1;
2533:             iPV[1] = 0;
2534:             iPV[2] = 0;
2535:             iPV[3] = 0;
2536:             iPV[4] = 0;
2537:             return True;
2538:         case 1:
2539:             nPV = 2;
2540:             VLP = reta_down(0,Value);
2541:             LP = reta_up(0,Value);
2542:             iPV[0] = 1;
2543:             iPV[1] = 1;
2544:             iPV[2] = 0;
2545:             iPV[3] = 0;
2546:             iPV[4] = 0;
2547:             return True;
2548:         case 2:
2549:             nPV = 3;
2550:             VLP = reta_down(0,Value);
2551:             LP = reta_up(0,Value);
2552:             MP = reta_up(1,Value);
2553:             iPV[0] = 1;
2554:             iPV[1] = 1;
2555:             iPV[2] = 1;
2556:             iPV[3] = 0;
2557:             iPV[4] = 0;
```

```
2558:             return True;
2559:         case 3:
2560:             nPV = 3;
2561:             LP = reta_down(2,Value);
2562:             MP = reta_up(1,Value);
2563:             HP = reta_up(2,Value);
2564:             iPV[0] = 0;
2565:             iPV[1] = 1;
2566:             iPV[2] = 1;
2567:             iPV[3] = 1;
2568:             iPV[4] = 0;
2569:             return True;
2570:         case 4:
2571:             nPV = 3;
2572:             LP = reta_down(2,Value);
2573:             MP = reta_down(3,Value);
2574:             HP = reta_up(2,Value);
2575:             iPV[0] = 0;
2576:             iPV[1] = 1;
2577:             iPV[2] = 1;
2578:             iPV[3] = 1;
2579:             iPV[4] = 0;
2580:             return True;
2581:         case 5:
2582:             nPV = 3;
2583:             MP = reta_down(3,Value);
2584:             HP = reta_down(4,Value);
2585:             VHP = reta_up(4,Value);
2586:             iPV[0] = 0;
2587:             iPV[1] = 0;
2588:             iPV[2] = 1;
2589:             iPV[3] = 1;
2590:             iPV[4] = 1;
2591:             return True;
2592:         case 6:
2593:             nPV = 2;
2594:             HP = reta_down(4,Value);
2595:             VHP = reta_up(4,Value);
2596:             iPV[0] = 0;
2597:             iPV[1] = 0;
2598:             iPV[2] = 0;
2599:             iPV[3] = 1;
```

```
2600:             iPV[4] = 1;
2601:             return True;
2602:         case 7:
2603:             nPV = 1;
2604:             VHP = 1.0;
2605:             iPV[0] = 0;
2606:             iPV[1] = 0;
2607:             iPV[2] = 0;
2608:             iPV[3] = 0;
2609:             iPV[4] = 1;
2610:             return True;
2611:     }
2612: }
2613:
2614: //*****************SPAOFE FUNCTIONS*********************
2615:
2616:
2617:
```