```
1  /* ============================================================*
2   * Project: 2                                                 *
3   * Title: Non-Sequential Behaviors                            *
4   * Team: 10                                                   *
5   *     Eskridge, Brent                                        *
6   *     Lopez, Tony                                            *
7   *     Maole, Amit                                            *
8   *     Utley, Klo                                             *
9   *                                                            *
10  *============================================================*/
11
12
13  /* Boolean values */
14  persistent int TRUE = 1;
15  persistent int FALSE = 0;
16
17  /* The structure containing all sensor info */
18  #define REGION_COUNT 7
19  struct regionSensorData {
20      int light[REGION_COUNT];
21  };
22
23  /* Last sensed data */
24  struct regionSensorData *LAST_SENSED;
25
26  /* ID's & values used to create default fieldForce variables */
27  int FIELDFORCE_000_ID = 0;
28  int FIELDFORCE_030_ID = 1;
29  int FIELDFORCE_060_ID = 2;
30  int FIELDFORCE_090_ID = 3;
31  int FIELDFORCE_120_ID = 4;
32  int FIELDFORCE_150_ID = 5;
33  int FIELDFORCE_180_ID = 6;
34  int FIELDFORCE_NEG_030_ID = 7;
35  int FIELDFORCE_NEG_060_ID = 8;
36  int FIELDFORCE_NEG_090_ID = 9;
37  int FIELDFORCE_NEG_120_ID = 10;
38  int FIELDFORCE_NEG_150_ID = 11;
39
40  /* Map a region to it's fieldForce ID */
41  int REGION_TO_FIELDFORCE_ID[REGION_COUNT] = { 3, 2, 1, 0, 7, 8, 9 };
42
43  /* FieldForces for behavior levels */
44  int STRATEGIC_ID = FIELDFORCE_000_ID;
45  int TACTICAL_ID = FIELDFORCE_000_ID;
46
47  /* Behavior triggers */
48  int SENSOR_SWEEP_FLAG = TRUE;
49  int BEH_TROPOTAXIS_FLAG = TRUE;
50  int BEH_LIGHTSEARCH_FLAG = FALSE;
51
52  /* Used to keep track of times for finding a light */
53  long LIGHTSEARCH_LASTTURN = 0L;
54  long LIGHTSEARCH_TIME_BETWEEN_TURNS = 5000L;
55  long LIGHTSEARCH_START_TIME = 0L;
56  long LIGHTSEARCH_TURN_TIME = 2000L;
57
58  /* Bump variables */
```

```
 59  long LAST_BUMP_TIME;
 60  long BUMP_BACKUP_TIME = 600L;
 61  long BUMP_TURN_TIME = 500L;
 62  long BUMP_TURN_TIME_ALT1 = 550L;
 63  long BUMP_TURN_TIME_ALT2 = 450L;
 64  int LAST_BUMP_RIGHT_FLAG = FALSE;
 65  int LAST_BUMP_LEFT_FLAG = FALSE;
 66  int BUMP_MODE_IDLE = 0;
 67  int BUMP_MODE_BACKUP = 1;
 68  int BUMP_MODE_TURN_RIGHT = 2;
 69  int BUMP_MODE_TURN_LEFT = 3;
 70  int CURRENT_BUMP_MODE = BUMP_MODE_IDLE;
 71
 72  /* Range variables */
 73  int LAST_RANGE_FF_FELT_ID = FIELDFORCE_NEG_090_ID;
 74  long LAST_RANGE_TIME;
 75  long RANGE_TIME_EFFECT = 1000L;
 76
 77  /* Motor commands */
 78  int MOTOR_REVERSE = -1;
 79  int MOTOR_FULL_PWR = 40;
 80  int MOTOR_HALF_PWR = 20;
 81
 82  /* Sensor ports */
 83  int LIGHT_SENSOR_PORT = 6;
 84  int LEFT_TOUCH_SENSOR_PORT = 7;
 85  int RIGHT_TOUCH_SENSOR_PORT = 11;
 86  int RANGE_RIGHT_SENSOR_PORT = 16;
 87  int RANGE_LEFT_SENSOR_PORT = 19;
 88
 89  /* Motor IDs */
 90  int LEFT_MOTOR_ID = 2;
 91  int RIGHT_MOTOR_ID = 0;
 92
 93  /* Motor variables */
 94  int LAST_MOTOR_FFID = -1;
 95
 96  /* Servo settings corresponding to the appropriate
 97     sensor turret direction */
 98  persistent int SERVO_VAL_90 = 4000;
 99  persistent int SERVO_VAL_0 = 2500;
100  persistent int SERVO_VAL_NEG90 = 1000;
101
102  /* Servo settings for region scans */
103  persistent int SERVO_VAL_REGIONS[REGION_COUNT];
104  long REGION_WAIT_TIME = 100L;
105
106  /* Light sensor calibration variables */
107  persistent int LIGHT_SENSOR_AVG_AMBIENT = 200;
108  persistent int LIGHT_SENSOR_TARGET_VAL = 10;
109  persistent int LIGHT_SENSOR_THRESHOLD = 30;
110
111  /* Rangefinder variables */
112  int RANGE_THRESHOLD_INIT = 40;
113  int RANGE_THRESHOLD = RANGE_THRESHOLD_INIT;
114  int RANGE_DIFFERENCE_THRESHOLD = 15;
115
116  /* Sleep times for processes */
```

```
117 long STRATEGIC_BEHAVIORS_SLEEP = 150L;
118 long TACTICAL_BEHAVIORS_SLEEP = 60L;
119 long MOTOR_CONTROL_SLEEP = 200L;
120
121 /* Pids of various processes */
122 int SENSORSWEEP_PID = -1;
123 int STRATEGIC_BEHAVIORS_PID = -1;
124 int TACTICAL_BEHAVIORS_PID = -1;
125
126
127 /* ------------------------------------------------------------ *
128  | Initialization and cleanup                                   |
129  * ------------------------------------------------------------ */
130
131 /* Initialize the robot and all its pieces */
132 void initialize()
133 {
134
135     // Start the servos
136     init_expbd_servos(1);
137
138     // Calculate the sensor regions
139     SERVO_VAL_REGIONS[0] = SERVO_VAL_90;
140     SERVO_VAL_REGIONS[1] = SERVO_VAL_0
141       + (int) (SERVO_VAL_90 - SERVO_VAL_0) * 2 / 3;
142     SERVO_VAL_REGIONS[2] = SERVO_VAL_0
143       + (int) (SERVO_VAL_90 - SERVO_VAL_0) / 3;
144     SERVO_VAL_REGIONS[3] = SERVO_VAL_0;
145     SERVO_VAL_REGIONS[4] = SERVO_VAL_NEG90
146       + (int) (SERVO_VAL_0 - SERVO_VAL_NEG90) * 2 / 3;
147     SERVO_VAL_REGIONS[5] = SERVO_VAL_NEG90
148       + (int) (SERVO_VAL_0 - SERVO_VAL_NEG90) / 3;
149     SERVO_VAL_REGIONS[6] = SERVO_VAL_NEG90;
150 }
151
152 /* Clean up after the robot */
153 void cleanup()
154 {
155     // Stop the tacticalBehavior process
156     if( TACTICAL_BEHAVIORS_PID > 0 )
157     {
158         kill_process( TACTICAL_BEHAVIORS_PID );
159     }
160
161     // Stop the strategicBehavior process
162     if( STRATEGIC_BEHAVIORS_PID > 0 )
163     {
164         kill_process( STRATEGIC_BEHAVIORS_PID );
165     }
166
167     // Stop sensorSweep process
168     if( SENSORSWEEP_PID > 0 )
169     {
170         kill_process( SENSORSWEEP_PID );
171     }
172
173     // Shut down the servos
174     init_expbd_servos(0);
```

```
175 }
176
177
178 /* ------------------------------------------------------------ *
179  | Sensor data gathering                                        |
180  * ------------------------------------------------------------ */
181
182 /* Perform a sensor sweep */
183 void sensorSweep()
184 {
185     // Start sweeping
186     while( !stop_button() )
187     {
188         // Which way do we rotate?
189         if( (SERVO_VAL_REGIONS[0] - servo0) > (servo0 - SERVO_VAL_REGIONS[6]) )
190         {
191             // Servo is closer to region 6.  Go clockwise
192             sensorSweepClockwise();
193         }
194         else
195         {
196             // Servo is closer to region 0.  Go counter-clockwise
197             sensorSweepCounterClockwise();
198         }
199     }
200
201     // Reset our pid
202     SENSORSWEEP_PID = -1;
203 }
204
205 /* Perform a sensor sweep in the clockwise direction */
206 void sensorSweepClockwise()
207 {
208     //struct regionSensorData current;
209     int i = 0;
210
211     // Get the sensor data for each region
212     for(i = REGION_COUNT - 1; i >= 0; i-- )
213     {
214         // Turn to the specified region
215         servo0 = SERVO_VAL_REGIONS[i];
216
217         // Sleep for a second
218         msleep(REGION_WAIT_TIME);
219
220         // Get the sensor value
221         LAST_SENSED->light[i] = analog(LIGHT_SENSOR_PORT);
222     }
223 }
224
225 /* Perform a sensor sweep in the counter-clockwise direction */
226 void sensorSweepCounterClockwise()
227 {
228     //struct regionSensorData current;
229     int i = 0;
230
231     // Get the sensor data for each region
232     for(i = 0; i < REGION_COUNT; i++ )
```

```
233     {
234         // Turn to the specified region
235         servo0 = SERVO_VAL_REGIONS[i];
236 
237         // Sleep for a second
238         msleep(REGION_WAIT_TIME);
239 
240         // Get the sensor value
241         LAST_SENSED->light[i] = analog(LIGHT_SENSOR_PORT);
242     }
243 }
244 
245 
246 /* ------------------------------------------------------------ *
247  | Strategic Behaviors                                          |
248  * ------------------------------------------------------------ */
249 void strategicBehaviors()
250 {
251     // Loop until we are told to stop
252     while( !stop_button() )
253     {
254         // As a last resort, move straight ahead
255         moveForwardBehavior();
256 
257         // Try to move towards the light first
258         if( BEH_TROPOTAXIS_FLAG == TRUE )
259           tropotaxisBehavior();
260 
261         // Try to find the light
262         if( BEH_LIGHTSEARCH_FLAG == TRUE )
263           lightSearchBehavior();
264 
265         // Sleep for a bit
266         msleep(STRATEGIC_BEHAVIORS_SLEEP);
267     }
268 
269     STRATEGIC_BEHAVIORS_PID = -1;
270 }
271 
272 /* Move forward behavior */
273 void moveForwardBehavior()
274 {
275     // Go straight ahead
276     STRATEGIC_ID = FIELDFORCE_000_ID;
277 }
278 
279 /* Tropotaxis behavior (Follow the light) */
280 void tropotaxisBehavior()
281 {
282     int i = 0;
283     int strongestRegion = -1;
284     int strongestValue = 300;
285 
286     /* Go through all the regions and find the strongest
287        light source */
288     for( i = 0; i < REGION_COUNT; i++ )
289     {
290         // Is this stronger?
```

```
291         if( LAST_SENSED->light[i] < strongestValue )
292         {
293             // Save the specifics
294             strongestValue = LAST_SENSED->light[i];
295             strongestRegion = i;
296         }
297     }
298 
299     // Did we find anything?
300     if( (strongestValue < LIGHT_SENSOR_THRESHOLD)
301        && (strongestValue >= 0) )
302      {
303         // Yup.  Let's go in that direction
304         STRATEGIC_ID = REGION_TO_FIELDFORCE_ID[strongestRegion];
305         beep();
306 
307         // Signal that we are tracking a light
308         BEH_LIGHTSEARCH_FLAG = FALSE;
309     }
310     else
311       {
312         // Signal that we are not tracking a light
313         BEH_LIGHTSEARCH_FLAG = TRUE;
314     }
315 }
316 
317 /* Light search behavior */
318 void lightSearchBehavior()
319 {
320     // After set amount of timemake a right turn
321     if( mseconds() > (LIGHTSEARCH_TIME_BETWEEN_TURNS + LIGHTSEARCH_LASTTURN) )
322     {
323         // Send the command
324         STRATEGIC_ID = FIELDFORCE_090_ID;
325 
326         // Save the time
327         if( LIGHTSEARCH_START_TIME < LIGHTSEARCH_LASTTURN )
328         {
329             LIGHTSEARCH_START_TIME = mseconds();
330         }
331 
332         // We have turned.  Do we stop?
333         if( mseconds() > LIGHTSEARCH_START_TIME + LIGHTSEARCH_TURN_TIME )
334         {
335             // Yup
336             LIGHTSEARCH_LASTTURN = mseconds();
337         }
338     }
339 }
340 
341 
342 
343 /* ------------------------------------------------------------ *
344  | Tactical Behaviors                                           |
345  * ------------------------------------------------------------ */
346 void tacticalBehaviors()
347 {
348     // Loop until we are told to stop
```

```
349     while( !stop_button() )
350     {
351
352         // Avoid objects we bump into
353         avoidLowBehavior();
354
355         // Avoid tall objects
356         avoidHighBehavior();
357
358         /* This isn't a behavior, but putting it here guarantees that the
359            tactical behaviors get a chance to do their thing BEFORE
360            we actually move */
361         motorControl();
362
363         // Sleep for a bit
364         msleep(TACTICAL_BEHAVIORS_SLEEP);
365     }
366
367     TACTICAL_BEHAVIORS_PID = -1;
368 }
369
370 /* Avoid low obstacles behavior
371    Note that if we are reacting to a bump when we get a new event
372    we switch our reaction to the new one */
373 void avoidLowBehavior()
374 {
375     // Get the current settings
376     int currentBumpLeft = digital(LEFT_TOUCH_SENSOR_PORT);
377     int currentBumpRight = digital(RIGHT_TOUCH_SENSOR_PORT);
378
379     // Have we bumped?
380     if( currentBumpLeft == TRUE || currentBumpRight == TRUE )
381     {
382         // Yup, avoid it
383         LAST_BUMP_RIGHT_FLAG = currentBumpRight;
384         LAST_BUMP_LEFT_FLAG = currentBumpLeft;
385         LAST_BUMP_TIME = mseconds();
386         CURRENT_BUMP_MODE = BUMP_MODE_BACKUP;
387         //printf("\nBUMPED");
388     }
389
390     // Are we supposed to be doing anything?
391     if( CURRENT_BUMP_MODE != BUMP_MODE_IDLE )
392     {
393         if( CURRENT_BUMP_MODE == BUMP_MODE_BACKUP )
394         {
395             // Have we backed up enough?
396             if( LAST_BUMP_TIME + BUMP_BACKUP_TIME < mseconds() )
397             {
398                 // Yes, start turning
399                 if( LAST_BUMP_LEFT_FLAG == TRUE )
400                 {
401                     // Turn to the right
402                     CURRENT_BUMP_MODE = BUMP_MODE_TURN_RIGHT;
403                 }
404                 else
405                 {
406                     // Turn to the left
```

```
407                     CURRENT_BUMP_MODE = BUMP_MODE_TURN_LEFT;
408                 }
409
410                 // Add some variation to avoid cycles
411                 if( servo0 > 2000 )
412                 {
413                     BUMP_TURN_TIME = BUMP_TURN_TIME_ALT1;
414                 }
415                 else
416                 {
417                     BUMP_TURN_TIME = BUMP_TURN_TIME_ALT2;
418                 }
419
420                 // Save the time
421                 LAST_BUMP_TIME = mseconds();
422             }
423             else
424             {
425                 // Nope, keep on backing up ontinue
426                 TACTICAL_ID = FIELDFORCE_180_ID;
427                 //printf("\nBACKING");
428             }
429         }
430
431         // Are supposed to be turning?
432         if( (CURRENT_BUMP_MODE == BUMP_MODE_TURN_RIGHT)
433            || (CURRENT_BUMP_MODE == BUMP_MODE_TURN_LEFT) )
434         {
435             // Have we turned enough?
436             if( (LAST_BUMP_TIME + BUMP_TURN_TIME) < mseconds() )
437             {
438                 // Yup, stop it
439                 CURRENT_BUMP_MODE = BUMP_MODE_IDLE;
440
441                 // Do whatever the strategic wants us to do
442                 TACTICAL_ID = STRATEGIC_ID;
443                 //printf("\nEXPIRE IDLE");
444             }
445             else
446             {
447                 // Nope, keep on turning
448                 if( CURRENT_BUMP_MODE == BUMP_MODE_TURN_RIGHT )
449                 {
450                     // Turn to the right
451                     TACTICAL_ID = FIELDFORCE_090_ID;
452                     //printf("\nTURN RIGHT");
453                 }
454                 else
455                 {
456                     // Turn to the left
457                     TACTICAL_ID = FIELDFORCE_NEG_090_ID;
458                     //printf("\nTURN LEFT");
459                 }
460             }
461         }
462     }
463     else
464     {
```

```
465         // We have no input, do what the strategic wants to do
466         TACTICAL_ID = STRATEGIC_ID;
467     }
468 }
469
470 /* Avoid high obstacles behavior */
471 void avoidHighBehavior()
472 {
473     // Is there anything to the left?
474     int leftObstacleValue = analog(RANGE_LEFT_SENSOR_PORT);
475     int leftObstacleFlag = leftObstacleValue > RANGE_THRESHOLD;
476
477     // Is there anything to the right?
478     int rightObstacleValue = analog(RANGE_RIGHT_SENSOR_PORT);
479     int rightObstacleFlag = rightObstacleValue > RANGE_THRESHOLD;
480
481     // Is there anything to left & right
482     if( leftObstacleFlag && rightObstacleFlag )
483     {
484         // Is one stronger than the other?
485         if( leftObstacleValue - rightObstacleValue < RANGE_DIFFERENCE_THRESHOLD )
486         {
487             /* Nope, turn in the same direction as last time
488                (minimizes cycles) */
489             TACTICAL_ID = LAST_RANGE_FF_FELT_ID;
490         }
491         else
492         {
493             // Which is stronger
494             if( leftObstacleValue > rightObstacleValue )
495             {
496                 // Left is stronger, go right
497                 TACTICAL_ID = FIELDFORCE_090_ID;
498             }
499             else
500             {
501                 // Right is stronger, go left
502                 TACTICAL_ID = FIELDFORCE_NEG_090_ID;
503             }
504         }
505
506
507         // Save the time
508         LAST_RANGE_TIME = mseconds();
509         printf("\nBOTH");
510     }
511     else
512     {
513         // Is there anything to the right?
514         if( rightObstacleFlag )
515         {
516             // Yup
517             TACTICAL_ID = FIELDFORCE_NEG_090_ID;
518             LAST_RANGE_FF_FELT_ID = TACTICAL_ID;
519
520             // Save the
521             LAST_RANGE_TIME = mseconds();
522             printf("\nRIGHT");
```

```
523         }
524
525         // Is there anything to the left?
526         if( leftObstacleFlag )
527         {
528             // Yup
529             TACTICAL_ID = FIELDFORCE_090_ID;
530             LAST_RANGE_FF_FELT_ID = TACTICAL_ID;
531
532             // Save the time
533             LAST_RANGE_TIME = mseconds();
534             printf("\nLEFT");
535         }
536
537         /* Are there any lasting effects?
538            (i.e., are we still reacting?) */
539         if( !leftObstacleFlag && !rightObstacleFlag
540            && (LAST_RANGE_TIME + RANGE_TIME_EFFECT > mseconds()) )
541         {
542             // Yup, keep going the same direction
543             TACTICAL_ID = LAST_RANGE_FF_FELT_ID;
544         }
545     }
546 }
547
548 /* ------------------------------------------------------------ *
549  | Motor Control                                                |
550  * ------------------------------------------------------------ */
551 void motorControl()
552 {
553     int currentID = TACTICAL_ID;
554     int leftMotorPwr = 0;
555     int rightMotorPwr = 0;
556
557     // Do we need to make any changes?
558     if( LAST_MOTOR_FFID != currentID )
559     {
560         // Yup, find out what power we send to the motors
561         if( (currentID == FIELDFORCE_000_ID)
562            || (currentID == FIELDFORCE_180_ID))
563         {
564             // Go in a straight line
565             leftMotorPwr = MOTOR_FULL_PWR;
566             rightMotorPwr = MOTOR_FULL_PWR;
567
568             // Is it backwards
569             if( currentID == FIELDFORCE_180_ID )
570             {
571                 // Yup
572                 leftMotorPwr *= MOTOR_REVERSE;
573                 rightMotorPwr *= MOTOR_REVERSE;
574             }
575         }
576         else
577         {
578             // Will we still move forward?
579             if( (currentID == FIELDFORCE_030_ID)
580                || (currentID == FIELDFORCE_NEG_030_ID) )
```

```
581             {
582                 // Which way do we arc?
583                 if( currentID == FIELDFORCE_030_ID )
584                 {
585                     // To the right
586                     leftMotorPwr = MOTOR_HALF_PWR;
587                     rightMotorPwr = MOTOR_FULL_PWR;
588                 }
589                 else
590                 {
591                     // To the left
592                     leftMotorPwr = MOTOR_FULL_PWR;
593                     rightMotorPwr = MOTOR_HALF_PWR;
594                 }
595             }
596             else
597             {
598                 // Turn to the right or left?
599                 if( (currentID == FIELDFORCE_060_ID)
600                    || (currentID == FIELDFORCE_090_ID)
601                    || (currentID == FIELDFORCE_120_ID)
602                    || (currentID == FIELDFORCE_150_ID)
603                    || (currentID == FIELDFORCE_180_ID) )
604                 {
605                     // To the right
606                     leftMotorPwr = MOTOR_HALF_PWR;
607                     rightMotorPwr = MOTOR_HALF_PWR * MOTOR_REVERSE;
608                 }
609                 else
610                 {
611                     // to the left
612                     leftMotorPwr = MOTOR_HALF_PWR * MOTOR_REVERSE;
613                     rightMotorPwr = MOTOR_HALF_PWR;
614                 }
615             }
616         }
617 
618         // Send the command
619         motor( LEFT_MOTOR_ID, leftMotorPwr );
620         motor( RIGHT_MOTOR_ID, rightMotorPwr );
621 
622         // Save the new id
623         LAST_MOTOR_FFID = currentID;
624     }
625 }
626 
627 
628 
629 /* ------------------------------------------------------------ *
630  | Main                                                         |
631  * ------------------------------------------------------------ */
632 void main()
633 {
634     // Make the pointers point to something
635     struct regionSensorData initSD;
636     LAST_SENSED = &initSD;
637 
638     // Prompt the user to start us up
```

```
639       printf("\nReady...");
640       start_press();
641
642       // Initialize the robot
643       initialize();
644
645       // Kick off sensor sweeps
646       SENSORSWEEP_PID = start_process( sensorSweep() );
647
648       // Kick off the tactical behaviors
649       TACTICAL_BEHAVIORS_PID = start_process( tacticalBehaviors() );
650
651       // Kick off the strategic behaviors
652       STRATEGIC_BEHAVIORS_PID = start_process( strategicBehaviors() );
653
654       // Wait until we are told to quit
655       while( !stop_button() ) {}
656
657       // Turn off motors
658       ao();
659
660       // Cleanup
661       cleanup();
662
663       printf("\nDone..");
664   }
665
```