```
/*
   Cross-Correlation Speed Sensor Code.

   */

#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <memory.h>
long random(void);  /* This should be in the include file but isn't. */

#define WINDOW 100  /* Number of samples to compare. */
#define DIST   1.0  /* Distance between sensors, in meters. */
#define SIZE  1000  /* Number of samples to store. */
#define SAMPLE_INTERVAL 0.01 /* Time between reading sensors, in seconds. */

```

```
16 /*   Find delta that yields best match.  */
17
18 int best_match(int *samp_A, int *samp_B, int size)
19 {
20  int delta          =  0;
21  double lowest_err =  0; /* Initialize only to eliminate compiler warnings. */
22  int best_delta     = -1;
23
24  for( delta = 0; delta < size - WINDOW; delta++ ){
25   double this_err = 0;   /* Error for this delta. */
26   int i;
27
28   /* Compute error by summing absolute value of differences. */
29   for( i = delta; i < delta + WINDOW; i++ )
30    this_err += abs( samp_A[i] - samp_B[i-delta] );
31
32   /* See if error for this delta is smaller than all previous. */
33   if( best_delta == -1 || this_err < lowest_err ) {
34    lowest_err = this_err;
35    best_delta = delta;
36   }}
37
38  return best_delta;
39 }
40
```

```
40 /* Routines below are stubs for interface and timing functions.
41    (That is, they allow the program to be run without having to
42    read real sensors or to sleep.) */
43
44 double get_current_time()
45 {
46  struct timespec tp;
47  /* clock_gettime is part of posix library. Can substitute time or, since
48     the return value isn't really used, can comment out completely. */
49  clock_gettime(CLOCK_REALTIME, &tp);
50  return ((double)tp.tv_sec) + tp.tv_nsec * 1e9;
51  /* return 0.0; */
52 }
53
54 /* The sleep_until routine does nothing since we're not reading from
55    real sensors anyway.  If this were an actual RTS, we would need
56    to call a precision sleep routine. (The sleep routine in the C library
57    is too coarse.) */
58
59 void sleep_until(double wake_time){return;}
60
61
```

```
61 /*  Variables for simulating sensor output. */
62 static int ri_sample_history[SIZE]; /* Simulated sensor A output. */
63 static int ri_next_sample = -1;
64 static int ri_delta = SIZE / 17;
65 #define NOISE_MASK 0x3fff
66
67 /* Generate a random sample stream "for" sensor A.
68    Samples have a uniform distribution over [0,2147483647] */
69
70 int readInterfaceA()
71 {
72  if( ri_next_sample == -1 )
73    {
74      memset(ri_sample_history,0,sizeof(ri_sample_history));
75      ri_next_sample = 0;
76    }
77  else
78    {
79      if( ++ri_next_sample == SIZE ) ri_next_sample = 0;
80    }
81
82  return (ri_sample_history[ri_next_sample] = random() & 0x3FFF);
83 }
84
```

```
84 /* Sample stream "for" sensor B is A's stream delayed by constant ri_delta,
85    with some non-trivial noise added. */
86
87 int readInterfaceB() {
88  int index =  ri_next_sample - ri_delta;
89  if( index < 0 ) index += SIZE;
90  return ri_sample_history[ index ] +
91    ( random() & NOISE_MASK ) - ( NOISE_MASK >> 1 );
92 }
93
94 /* This just writes to stdout if the count is in the proper range.  The
95    count is used to only print out values of interest. (In a real system
96    we might display every value.) */
97
98 void display(double value){
99  static count=0;
100  count++;
101  if( count > 700 )printf("Speed %f meters / second.\n",value);
102  if( count == 720 )exit(0);
103 }
104
105
```

```
105 void cross_cor()   /*  Main Loop  */
106 {
107  int samp_A[SIZE];                        /* Hold values read from sensor A. */
108  int samp_B[SIZE];                        /* Hold values read from sensor B. */
109  double next_sample_time;                 /* Time at which sensors will be read.*/
110  int i;
111
112  for(i=0;i<SIZE;i++)samp_A[i]=samp_B[i]=0;   /* Inititalize arrays. */
113  next_sample_time = get_current_time();      /* Initialize to now. */
114
115  while(1){
116   int a = readInterfaceA();               /* Read sensor A. */
117   int b = readInterfaceB();               /* Read sensor B. */
118   double speed;
119   int delta;
120
121   /* Make room for new samples. (There are faster ways of doing this.) */
122   for(i=SIZE-1; i>0; i--){samp_A[i] = samp_A[i-1]; samp_B[i] = samp_B[i-1];}
123
124   samp_A[0] = a; samp_B[0] = b;           /* Store new samples. */
125
126   delta = best_match( samp_A, samp_B, SIZE );   /* Samples between sensors. */
127   speed = DIST / ( SAMPLE_INTERVAL * delta );   /* Compute speed. */
128
129   display( speed );                       /* Display flow speed. */
130
131   next_sample_time += SAMPLE_INTERVAL;  /* Compute when sensors next read. */
132   sleep_until( next_sample_time );      /* Returns when time to sample again. */
133
134  }
135 }
136
```

```
137
138 int main(int argv, char **argc)
139 {
140   printf("The correct speed is: %f meters / second.\n",
141          DIST / ( ri_delta * SAMPLE_INTERVAL ));
142   cross_cor(); /* Never returns. */
143   return 0;
144 }
end
```