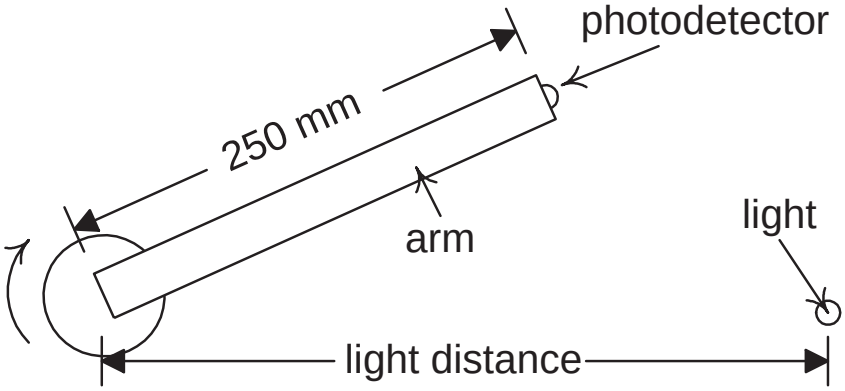
EE 4770 **Homework 3** Solution **Due: 8 March 1999**

Problem 1: The distance from a light source of unknown but steady brightness is to be determined using a photodetector mounted on one end of a 250 mm rotating arm (with the rotation around the other end) as illustrated below.



The light source radiates uniformly in all directions; the photodetector is sensitive to irradiance and is equally sensitive to light in all directions and is sensitive to no other light sources. The light source will be outside the circle traced by the rotating photodetector. The photodetector is connected to a conditioning circuit (already designed) which feeds an 18-bit ADC. The ADC output is interfaced to a computer. The photodetector and conditioning circuit are linear; the ADC saturates at $10\frac{\text{mW}}{\text{cm}^2}$. The ADC is noisy, so its output randomly varies by ± 1.

The arm makes one complete rotation in exactly 10 seconds. A proximity sensor detects when the arm is at its 0° angle, at which point a 16-bit counter is reset. The counter is clocked at 6.5536 kHz and is interfaced to the computer.

An interface routine is to sample the ADC and counter values and determine the direction of the light source in degrees (with respect to the arm's zero position), the radiant flux of the light source in milliwatts, and its distance from the axis of rotation in centimeters. To do this the interface routine will have to sample the sensors multiple times (and so will not be able to produce values the first time called).

The light source can move, the distance and radiant flux values need not be correct while it is moving or for a short time after it stops.

A shell of the interface routine appears below. Complete the shell.

Given the photodetector and conditioning circuit it is a simple matter to determine irradiance. Let E denote irradiance, r_e denote the ADC output, and $E_{\max} = 10\frac{\text{mW}}{\text{cm}^2}$ denote the maximum detectable irradiance. Since the photodetector and all parts of the conditioning circuit are linear, $E = \frac{r_e}{2^{18}-1}E_{\max}$.

A similar method is used to find the arm angle. Let θ denote the arm angle and c denote the counter output. Then $\theta = \frac{c}{2^{16}-1}360^\circ$.

The direction of the light is the angle of the arm at which irradiance is at a maximum.

It would be easy to determine the distance of the light if its radiant flux were known, using $E = \frac{\Phi}{4\pi L^2}$, where L is the distance and Φ is the radiant flux. Using the same equation it would be easy to determine the radiant flux if the distance were known. Neither distance nor radiant flux are known, we need to work with what we have. One advantage of a homework assignment over a real-world engineering problem is that one can be sure it has a solution and that the assignment provides important information. One piece of information in the problem is the arm length. The photodetector positions at minimum and maximum detected irradiance (ignoring precision errors) will be two arm lengths apart, $d = 500\,\text{mm}$. Use this information in a pair of equations based on minimum and maximum measured irradiance:

$$E_1 = \frac{\Phi}{4\pi L^2} \qquad \text{and} \qquad E_2 = \frac{\Phi}{4\pi(L+d)^2},$$

where E_1 and E_2 are the maximum and minimum irradiances, respectively. Solving simultaneously,

$$L = d\frac{1+\sqrt{E_1/E_2}}{E_1/E_2 - 1}.$$

The distance is defined to be from the object to the axis of rotation, $L + 250\,\text{mm}$.

The interface routine code is included at the end of the solution.

```
extern double light_direction, light_distance, light_radiant_flux;
void interface_routine()
{
  double next_sample_time = get_current_time(); /* Time in seconds since 1970 */
  /* Insert additional declarations and initializations below. */

  /* Main Loop */
  while(1){
    int photo_raw = readInterace(PHOTODETECTOR);
    int angle_raw = readInterace(BEAMCOUNTER);

    /* Insert code below. */




    light_direction = /* Insert code */ ;
    light_distance = /* Insert code */ ;
    light_radiant_flux = /* Insert code */ ;
    next_sample_time = /* Insert code */ ;
    sleep_until(next_sample_time);
  }
}
```

Problem 2: Write equations that can be solved to determine the minimum and maximum distances that the system above can determine in terms of the radiant flux of the light. The equations do not have to be solved.

Recall:

$$E_1 = \frac{\Phi}{4\pi L^2} \qquad \text{and} \qquad E_2 = \frac{\Phi}{4\pi(L+d)^2},$$

The minimum distance is the distance at which the photodetector will saturate (at $E_{\max}$) when it is closest. That distance is

$$L_{\min} = \frac{d}{2} + \sqrt{\frac{\Phi}{4\pi E_{\max}}}$$

(The $\frac{d}{2}$ is for the distance from the photodetector to the center of rotation, from which distance is measured.)

The maximum distance is the distance at which the difference in irradiances E_1 and E_2 produce a change of 1 in ADC output. That can be found by solving

$$\left(\frac{\Phi}{4\pi(L_{\max} - d/2)^2} - \frac{\Phi}{4\pi(L_{\max} + d/2)^2}\right)\frac{2^{18}-1}{E_{\max}} = 1$$

for $L_{\max}$.

Problem 3: Add code to the program to write the precision of the distance computed for a particular measurement to variable `light_distance_precision`. The precision is the magnitude of the maximum possible difference between the value of `light_distance` and the actual distance assuming everything works perfectly except the photodetector ADC, which may be off ± 1.

Consider again:

$$L = d\frac{1 + \sqrt{E_1/E_2}}{E_1/E_2 - 1}.$$

When E_2 is small and E_1 is large the higher error occurs when the maximum ADC reading is incremented and the minimum is decremented. In other cases the maximum would be decremented and the minimum incremented. It would be easiest to have the program check both cases.

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

#define MAX_IRRAD   10      /* milliwatts / cm^2                              */
#define ARM_LEN     25      /* cm                                             */
#define REV_TIME    10      /* seconds. Time for arm to make a revolution.    */
#define PHOTO_PREC (1<<18)  /* Precision of photodetector ADC.                */
#define ANGLE_PREC (1<<16)  /* Precision of angle counter.                    */

#define PHOTODETECTOR 1
#define ARMCOUNTER 2

/* Declarations for sensor simulation.                                        */
double sim_x         = 63;    /* cm                                         */
double sim_y         = 156;   /* cm                                         */
#define SIM_RAD_FLUX   10000  /* Light milliWatts                           */
int revs             = 5;     /* Number of revolutions to run simulation for. */
double sim_arm_angle = 0;
double sim_distance;          /* Set by read interface.                     */


int readInterface(int port)
{
  if( port == PHOTODETECTOR ) {
    double delta_x = sim_x - ARM_LEN * sin(sim_arm_angle);
    double delta_y = sim_y - ARM_LEN * cos(sim_arm_angle);
    double distance_sq = delta_x * delta_x + delta_y * delta_y;
    double irrad =
      SIM_RAD_FLUX / ( 4.0 * M_PI * distance_sq );
    double adc_out = irrad / MAX_IRRAD * PHOTO_PREC + ( random() % 3 ) - 1;
    sim_distance = pow( distance_sq, 0.5 );
    return adc_out >= PHOTO_PREC ? PHOTO_PREC-1 : adc_out;
  } else {
    return sim_arm_angle / ( 2.0 * M_PI ) * ANGLE_PREC;
  }
  return -1; /* Avoid compiler warnings. */
}

void sleep_until(double time)
{
  static double last_time = 0;

  if( last_time == 0 ) {  /* Inititalize */
    sim_arm_angle = random() * 2.0 * M_PI / RAND_MAX;
  } else {
    sim_arm_angle += 2.0 * M_PI * (time-last_time) / REV_TIME;
```

```
    while( sim_arm_angle >= 2.0 * M_PI ) sim_arm_angle -= 2.0 * M_PI;
  }
  last_time = time;
}

double get_current_time()
{
  struct timespec tp;
  /* clock_gettime is part of posix library. Can substitute time or, since
     the return value isn't really used, can comment out completely. */
  clock_gettime(CLOCK_REALTIME, &tp);
  return ((double)tp.tv_sec) + tp.tv_nsec / 1e9;
  /*  return 0.0;  */
}

double light_direction, light_distance, light_radiant_flux;
void see_the_light()
{
  double next_sample_time = get_current_time(); /* Time in seconds since 1970 */
  int min_photo_raw = 1<<17;
  int max_photo_raw = -1;
  double light_direction = 0;
  double next_direction = 0;

  /* Main Loop */
  while(revs){
    int photo_raw = readInterface(PHOTODETECTOR);
    double arm_angle = readInterface(ARMCOUNTER) * 360.0 / ANGLE_PREC;
    double phase = light_direction - arm_angle;
    if( phase < 0 ) phase += 360;

    if( phase < 90 or phase > 270 ) {
      if( min_photo_raw < PHOTO_PREC ) {
        double ratio = ((double)max_photo_raw) / min_photo_raw;
        double ratio_maybe = ((double)max_photo_raw-1) / (min_photo_raw+1);
        double ratio_or_maybe = ((double)max_photo_raw+1) / (min_photo_raw-1);
        double max_irrad = ((double)max_photo_raw) * MAX_IRRAD / PHOTO_PREC;
        double light_distance_maybe = ARM_LEN +
          2.0 * ARM_LEN * ( 1.0 + pow(ratio_maybe,0.5) )
          / ( ratio_maybe - 1.0 );
        double light_distance_or_maybe = ARM_LEN +
          2.0 * ARM_LEN * ( 1.0 + pow(ratio_or_maybe,0.5) )
          / ( ratio_or_maybe - 1.0 );
        double error_maybe = light_distance_maybe - light_distance;
        double error_or_maybe = light_distance - light_distance_or_maybe;
        double the_error =
          error_maybe > error_or_maybe ? error_maybe : error_or_maybe;
          light_direction = next_direction;
        light_distance = ARM_LEN +
```

```
          2.0 * ARM_LEN * ( 1.0 + pow(ratio,0.5) ) / ( ratio - 1.0 );
        light_radiant_flux =
          4.0 * M_PI * light_distance * light_distance * max_irrad;
        printf("(%3d,%3d) Dist, %f; angle, %6.2f deg; radiant flux, %f mW\n",
               min_photo_raw, max_photo_raw,
               light_distance, light_direction, light_radiant_flux);
        printf("Precision %f\n", the_error);
        min_photo_raw = PHOTO_PREC;
        max_photo_raw = 0;
        revs--;
      }
      if( photo_raw > max_photo_raw ) {
        max_photo_raw = photo_raw;
        next_direction = arm_angle;
      }
    } else {
      if( photo_raw < min_photo_raw ) {
        min_photo_raw = photo_raw;
      }
    }

    next_sample_time += 10.0 / ANGLE_PREC;
    sleep_until(next_sample_time);
  }
}

int main(int argv, char **argc)
{
  double angle = atan2(sim_x,sim_y) / ( 2.0 * M_PI ) * 360.0;
  if( angle < 0 ) angle += 180;
  printf("Initial position %5.1f, %5.1f\n",sim_x,sim_y);
  printf("Simulated distance, %f cm; angle %6.2f deg; rf %d mW\n",
         pow( sim_x * sim_x + sim_y * sim_y, 0.5), angle, SIM_RAD_FLUX);

  see_the_light();
  return 0;
}
```