Командный тур

5.1. Задачи

Задачи командного финального тура профиля составлены таким образом, чтобы они, с одной стороны, были достаточно независимыми, а с другой стороны, образовывали взаимосвязанную цепочку, необходимую для решения финальной задачи. Количество одновременно решаемых задач определяет слаженность командной работы. Всего командам предлагается 5 задач, в каждой команде по 3 -5 человек, таким образом, каждый участник может решать свою задачу. На рисунке 2, представлено дерево задач с разбивкой на подзадачи.

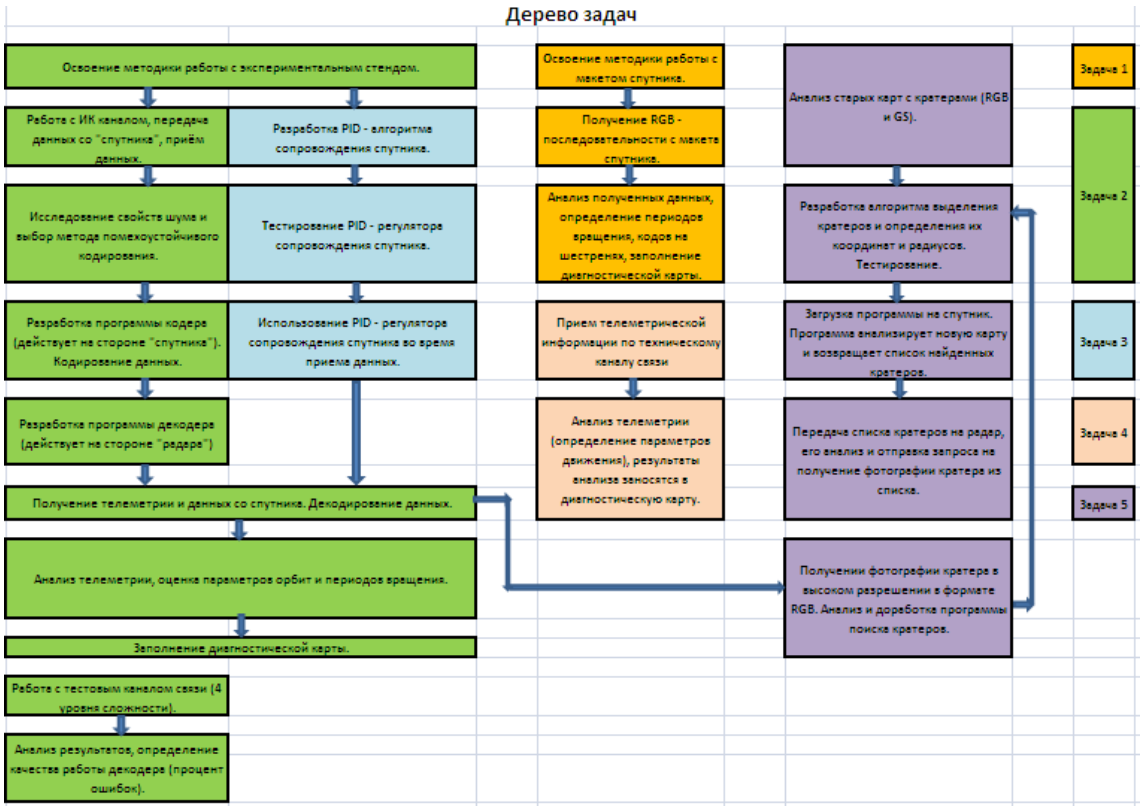


Рисунок 1. Дерево задач трека ТБС

Задача 5.1.1. 10 очков

Цель:

1. Получить навыки работы с помехоустойчивым кодом Хемминга, исправляющим однократную ошибку, освоить алгоритмы кодирования и декодирования сообщения, работа с матрицей преобразования, определить синдром, освоить методику исправления однократной ошибки.
2. Получить навыки работы со статистическим анализом данных: определение среднего и среднеквадратичного отклонения, обработка длинных рядов данных.

Результаты решения задачи могут быть использованы при решении задач 2 и 5.

Постановка задачи

Определение кода Хэмминга. Прорези и стенки на шестернях представляют код Хемминга различной длины. По рисунку прорезей на каждой шестеренке необходимо определить код Хемминга: элемент с наименьшим угловым размером – прорезь или стенка имеют смысл ‘0’ или ‘1’. Последовательность прорезей может читаться как по часовой, так и против часовой, признаком того что кодовое слово определено верно является отсутствие ошибок в закодированном слове. Закодированные и декодированные последовательности заносятся в диагностическую карту в двоичном и десятичном представлении.

Определение периода вращения. Получить последовательность сигналов с трех каналов макета (R - красный, G - зеленый, B - синий, далее RGB последовательность, по легенде - телеметрия), динамика сигнала в каждом канале модулируется своей вращающейся шестеренкой. По полученной последовательности сигналов на макете спутника определить средний период вращения каждой шестеренки, оценить стабильность вращения (среднеквадратичное отклонение от среднего периода). Результаты исследования занести в диагностическую карту.

Краткое описание подзадач:

1. Научиться работать с макетом спутника: включение, снятие данных.
2. Получить последовательность сигналов с трех каналов макета (R - красный, G - зеленый, B -синий, далее RGB-последовательность, по легенде - телеметрия), динамика сигнала в каждом канале модулируется своей вращающейся шестеренкой.
3. По полученной последовательности сигналов на макете спутника команды определяют период вращения каждой шестеренки, оценивают стабильность вращения (среднеквадратичное отклонение от среднего периода). Результаты исследования заносятся в диагностическую карту.
4. Прорези и стенки на шестернях представляют код Хемминга различной длины (на втором отборочном этапе командам предоставлялись ссылки на методические материалы по работе с кодом Хемминга). Команды по рисунку прорезей на каждой шестеренке определяют код Хемминга: элемент с наименьшим угловым размером – прорезь или стенка имеют смысл ‘0’ или ‘1’. Последовательность может читаться как по часовой, так и против часовой, однако признаком того что кодовое слово определено верно является отсутствие ошибок в закодированном слове (командам об этом сообщается). Закодированные (последовательность прорезей на шестерне) и декодированные последовательности заносятся в диагностическую карту.

Макет спутника, возвращающий эталонную RGB последовательность, представлен на рисунке 2. На шестернях начало кодовой последовательности отмечено ради-

альной линией. ИК – передатчик излучает сигнал постоянной амплитуды, на пути приемника находятся вращающиеся шестерни, модулирующие ИК сигнал по уровням: ‘0’, ‘1’. Для наглядности каждая шестерня подсвечена своим светодиодом (малая – красным, средняя – зеленым, большая – синим).

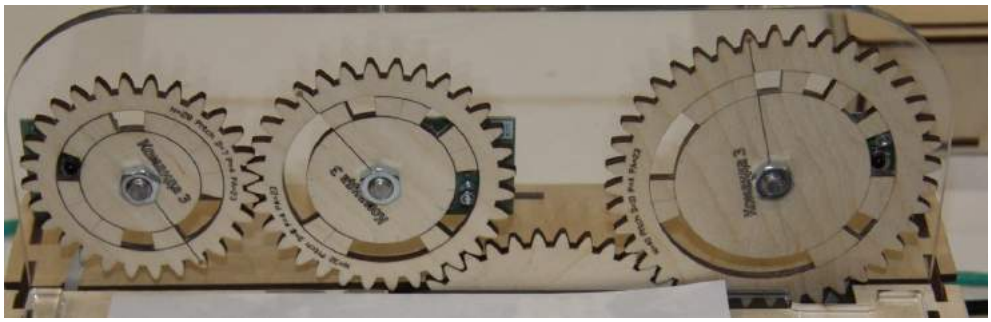


Рисунок 2. Макет спутника.

На рисунке 3 показана динамика амплитуды ИК – сигнала, прошедшего через вращающиеся шестерни. По полученной последовательности сигналов на макете спутника команды определяют период вращения каждой шестеренки, оценивают стабильность вращения (находят среднеквадратичное отклонение от среднего периода).

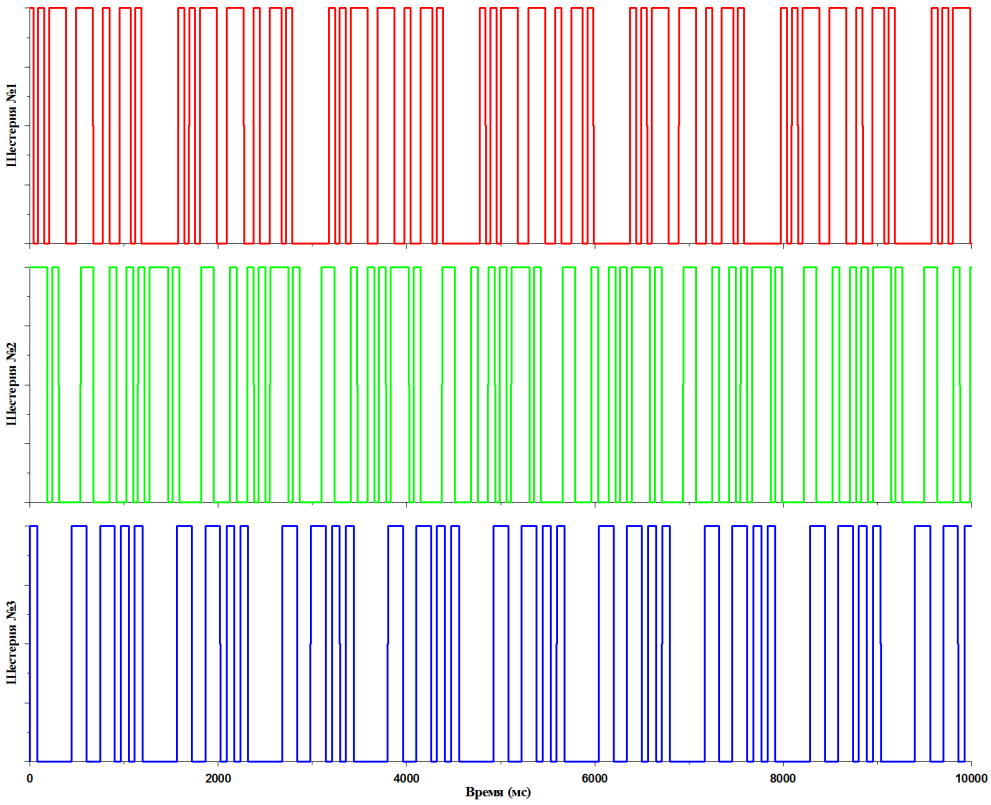


Рисунок 3. Динамика амплитуды ИК-сигнала, модулированного вращающимися шестернями.

Расчет очков (6 + 4)

Очки за шестерню:

Правильно определен период и СКО – 1 очко;

Последовательность правильно декодирована – 1 очко.

За каждую неудачную попытку снимается 5% от максимальных 6 очков, т.е. если команда приносит правильный ответ с первой попытки, то получает 100% (6 очков), если первая попытка неудачная, но ответ верно найден со второй попытки команда получает 95% от 6 очков, т.е. 5,7, и так далее.

Решение

Вариант программы декодирования кода Хемминга. На вход программы поступает произвольная последовательность из ‘0’ и ‘1’, на выходе программа возвращает декодированную последовательность из ‘0’ и ‘1’, причем если в исходной последовательности была допущена однократная ошибка, прогрмма ее исправляет и сообщает номер битого символа во входной последовательности.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
unsigned int maskf[] = {0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80, 0x100, 0x200, 0x400,
    0x800, 0x1000, 0x2000, 0x4000, 0x8000, 0x10000, 0x20000, 0x40000, 0x80000, 0x100000,
    0x200000, 0x400000, 0x800000, 0x1000000};
void CharToBinaryArray(int &a, int l, char arr[]);
void AddControlBit(char arr[], int l, char arrc[], int lc);
int order(int a); //кол-во разрядов в двоичном представлении
void matrpreobr(char **preobr, int lrow, int lcol);
void decoder_hamming(char **preobr, int lrow, int lcol, char earr[], char darr[]);
void cutcontrolbit(int lcol, char darr[], int l, char arr[]);
void HammingDecoderArr(char *esignal, int l_esignal, char *&signal, int &l_signal);
void main(int argc, char *argv[])
{
    //---------------Decoder--------------------------------------//
    int l_esignal = strlen(argv[1]); //Длина входной последовательности
    int l_signal;
    char *esignal;
    char *signal = NULL;
    esignal = (char *)calloc(l_esignal + 1, sizeof(char));
    strcpy(esignal, argv[1]);
    printf("esignal = %s length = %d\n", esignal, l_esignal);
    HammingDecoderArr(esignal, l_esignal, signal, l_signal);
    printf(" signal = %s length = %d\n", signal, l_signal);
    //------------------------------------------------------//
}
void HammingDecoderArr(char *esignal, int l_esignal, char *&signal, int &l_signal)
{
    int lrow = order(l_esignal);
    l_signal = l_esignal - lrow;
    signal = (char *)calloc(l_signal, sizeof(char));
    char *darr = new char[l_esignal];
    char **preobr = new char *[lrow];
    for (int i = 0; i < lrow; i++)
        preobr[i] = new char[l_esignal];
    matrpreobr(preobr, lrow, l_esignal); //Формируем матрицу преобразования
    decoder_hamming(preobr, lrow, l_esignal, esignal,
                    darr); //Декодированный сигнал (исправленыошибки)
    cutcontrolbit(l_esignal, darr, l_signal, signal);
```

```
42      delete[] darr;
43      for (int i = 0; i < lrow; i++)
44          delete[] preobr[i];
45      delete[] preobr;
46  }
47  void decoder_hamming(char **preobr, int lrow, int lcol, char earr[], char darr[])
48  {
49      int k = 0;
50      strcpy(darr, earr);
51      for (int i = 0; i < lrow; i++)
52      {
53          int sum = 0;
54          for (int j = 0; j < lcol; j++)
55              sum += (preobr[i][j] - '0') * (earr[j] - '0');
56          sum = sum % 2;
57          k += sum * (1 << i);
58      }
59      darr[k - 1] = (darr[k - 1] == '0') ? '1' : '0';
60      printf("k=%d\n", k);
61  }
62  void matrpreobr(char **preobr, int lrow, int lcol)
63  {
64      int i2 = 0, v;
65      char arr[100];
66      for (int i = 1; i <= lcol; i++)
67      { //цикл по столбцам
68          if ((i & (i - 1)) == 0)
69          { //степени двойки
70              i2++;
71              v = 1 << (i2 - 1);
72              CharToBinaryArray(v, lrow, arr);
73              for (int j = 0; j < lrow; j++) //цикл по строкам
74                  preobr[j][i - 1] = arr[j];
75          }
76          else
77          { //номера информационных битов
78              CharToBinaryArray(i, lrow, arr);
79              for (int j = 0; j < lrow; j++) //цикл по строкам
80                  preobr[j][i - 1] = arr[j];
81          }
82      }
83      for (int j = 0; j < lrow; j++)
84          preobr[j][lcol] = 0;
85  }
86  void cutcontrolbit(int lcol, char darr[], int l, char arr[])
87  {
88      int j = 0, i = 1;
89      while (i <= lcol)
90      {
91          if (i & (i - 1))
92              arr[j++] = darr[i - 1];
93          i++;
94      }
95      arr[j] = 0;
96  }
97  void AddControlBit(char arr[], int l, char arrc[], int lc)
98  {
99      int j = 0, i = 1;
100     while (j < l)
101     {
```

```
        if ((i & (i - 1)) == 0)
            arrc[i - 1] = '0';
        else
            arrc[i - 1] = arr[j++];
        i++;
    }
    arrc[i - 1] = 0;
}
int order(int a)
{
    double x = (double)a;
    x = log(x) / log(2.0);
    return (int)x + 1;
}
void CharToBinaryArray(int &a, int l, char *arr)
{
    for (int i = 0; i < l; i++)
        arr[i] = a & maskf[i] ? '1' : '0';
    arr[l] = 0;
}
```

Пример программы, определяющий период во входной последовательности и среднеквадратичное отклонение периода от среднего, дополнительно решаются следующие задачи: определяется корреляционная функция стабильного сигнала (с постоянным периодом) и измеренным на макете, определяется отношение квадрата невязки между модельным и измеренным сигналом к энергии модельного сигнала (т.е. определяется относительная энергия невязки между сигналами). Входные данные для программы содержатся в конфигурационном файле, имеющего следующий формат: кол-во последовательностей сигналов, длина кодовой последовательности (известна из предыдущей задачи), форма кодовой последовательности (последовательность из ‘0’ и ’1’), длительности переходных процессов (длительность переднего/заднего фронта сигнала, длительность смены фазы – переход от ‘0’ к ‘1’ и наоборот) – вводится для решения задачи в более общем случае, когда время переключения фазы не бесконечно быстрое. На выходе программа возвращает для каждого канала средний период и СКО отклонения от среднего периода.

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include "Gears.h"
#define PI (3.14159265358979323846264338327950)
//Инициализация массивов
void GearsSignal::InitArray()
{
    TP =(double *)calloc(ngear, sizeof(double));     //Период сигнала для каждой шестерни
    direction = (char *)calloc(ngear, sizeof(char)); //Направление вращения
    DT = (double *)calloc(ngear, sizeof(double));    //Сдвиг повремени
    Amp = (double *)calloc(ngear, sizeof(double));   //Амплитуда кор-функции
    Err = (double *)calloc(ngear, sizeof(double));   //Ошибка
}
GearsSignal::GearsSignal()
{
    ngear = 3;
    InitArray();
}
//Чтение конфигурационного файла
GearsSignal::GearsSignal(char *nameconfig)
```

```
22  {
23      FILE *conf;
24      conf = fopen(nameconfig, "r"); //Работаем с конфигурационным файлом
25      fscanf(conf, "%d", &ngear);    //Строка №1 - Длина кодовой последовательности
26      kd = (int *)calloc(ngear, sizeof(int));
27      kod = (char **)calloc(ngear, sizeof(char *));
28      InitArray();
29      namefile = (char *)calloc(1000, sizeof(char));
30      fscanf(conf, "%s", namefile); //Строка №2 имя файла с измерениями
31      for (int i = 0; i < ngear; i++)
32      {
33          fscanf(conf, "%d", &kd[i]); //Строка №3 - Длина кодовой
34                                      //последовательности
35          kod[i] = (char *)calloc(kd[i] + 1, sizeof(char));
36          fscanf(conf, "%s", kod[i]); //Строка №4 - форма кодовой
37                                      //последовательности
38          printf("kd[%d]=%d\n", i, kd[i]);
39          printf("kod[%d]=%s\n", i, kod[i]);
40      }
41      fscanf(conf, "%lf", &tau1); //Строка №5 - длительность переднего фронта
42                                  //(милисек)
43          fscanf(conf, "%lf", &tau2); //Строка №6 - длительность смены фазы (милисек)
44      fscanf(conf, "%lf", &tau3);     //Строка №7 - длительность заднего фронта
45                                      //(милисек)
46          fscanf(conf, "%lf", &dt); //Строка №8 - время между отсчетами (милисек)
47      fclose(conf);
48  }
49  //Анализ каналов - опредление энергии невязки между модельным и измеренным
50  //сигналами
51  void GearsSignal::GearsAnaliz()
52  {
53      if (ReadFile(namefile))
54      {
55          CalcPeriod();
56          SignalModel();
57          CorAnaliz2('w');
58          PrintArray();
59          EnergyError();
60      }
61      else
62          printf("analiz disable!\n");
63  }
64  void GearsSignal::EnergyError()
65  {
66      double dE, E, E1, E2;
67      for (int k = 0; k < ngear; k++)
68      {
69          int i = (int)DT[k];   //сдвиг
70          int n = direction[k]; //направление вращения
71          dE = 0.0;             //интеграл ошибок
72          E1 = E2 = E = 0.0;
73          for (int j = 0; j < I0[k]; j++) //Интегреируем
74              if (i + j >= 0 && i + j < I)
75              {
76                  E1 += A0[n][j] * A0[n][j];
77                  E2 += A[k + 1][i + j] * A[k + 1][i + j];
78                  E += A0[n][j] * A[k + 1][i + j];
79                  dE += (A[k + 1][i + j] - A0[n][j]) * (A[k + 1][i + j] -
80                                                        A0[n][j]);
81              }
```

```
82          if (IO[k] > 0)
83          {
84              Err[k] = 100.0 * dE / E1; //Енергия невязки
85          }
86          else
87              Err[k] = -100.0;
88      }
89      for (int i = 0; i < ngear; i++)
90          printf("Err[%d]=%lf\n", i, Err[i]);
91  }
92  void GearsSignal::CorAnaliz2(char rw)
93  {
94      double dtime[2], amp[2];
95      double **akf;
96      akf = (double **)calloc(ngear, sizeof(double *));
97      // for(int i = 0; i<ngear; i++)
98      // akf[i] = (double*)calloc(I,sizeof(double));
99      for (int i = 0; i < ngear; i++)
100     {
101         if (IO[i] > 0)
102         {
103             akf[i] = CorAnaliz(AO[2 * i], IO[i], A[i + 1], I, dtime[0],
104                                amp[0], 'r');
105             free(akf[i]);
106             akf[i] = CorAnaliz(AO[2 * i + 1], IO[i], A[i + 1], I, dtime[1],
107                                amp[1], 'r');
108             free(akf[i]);
109             if (amp[0] > amp[1])
110             {
111                 printf("direction[%d]=forvard!\n", i);
112                 DT[i] = dtime[0];
113                 Amp[i] = amp[0];
114                 direction[i] = 2 * i;
115                 akf[i] = CorAnaliz(AO[2 * i], IO[i], A[i + 1], I,
116                                    dtime[0], amp[0], rw);
117             }
118             else
119             {
120                 printf("direction[%d]=back!\n", i);
121                 DT[i] = dtime[1];
122                 Amp[i] = amp[1];
123                 direction[i] = 2 * i + 1;
124                 akf[i] = CorAnaliz(AO[2 * i + 1], IO[i], A[i + 1], I,
125                                    dtime[1], amp[1], rw);
126             }
127         }
128     }
129     FILE *rez;
130     rez = fopen("coranaliz_signal.dat", "w");
131     for (int j = 0; j < I; j++)
132     {
133         fprintf(rez, "%lf ", dt * (double)j);
134         for (int i = 0; i < ngear; i++)
135         {
136             fprintf(rez, "%lf ", akf[i][j]);
137         }
138         fprintf(rez, "\n");
139     }
140     fclose(rez);
141 }
```

```
142 double *GearsSignal::CorAnaliz(double *&A1, int I1, double *&A2,
143     int I2, double &DT, double &AmpAkf, char rw)
144 {
145     FILE *rez;
146     if (rw == 'w')
147     {
148         rez = fopen("coranaliz_test_signal.dat", "w");
149         fprintf(rez, "dI E\n");
150     }
151     int i, imax;
152     double E, E1, E2;
153     double Emax = 0.0;
154     double *akf;
155     akf = (double *)calloc(I2, sizeof(double));
156     for (i = 0; i < I2; i++)
157     { //Сдвиг первого сигнала относительно второго
158         E1 = E2 = E = 0.0;
159         for (int j = 0; j < I1; j++) //Интегреируем по развертке второго
160             //сигнала
161         if (i + j >= 0 && i + j < I2)
162         {
163             double v1 = A1[j];
164             double v2 = A2[i + j];
165             E1 += v1 * v1;
166             E2 += v2 * v2;
167             E += v1 * v2;
168         }
169         akf[i] = E;
170         if (rw == 'w')
171             fprintf(rez, "%d %lf\n", i, E);
172         if (E > Emax)
173         {
174             Emax = E;
175             imax = i;
176         }
177     }
178     if (rw == 'w')
179         fclose(rez);
180     DT = (double)imax;
181     AmpAkf = Emax;
182     return akf;
183 }
184 void GearsSignal::control_period()
185 {
186     if (ReadFile(namefile))
187         CalcPeriod(); //Средний период для всех шестерней
188     int k = 1;
189     int i = 1, i2 = 0;
190     char **shortkod;
191     shortkod = (char **)calloc(ngear, sizeof(char *));
192     for (int k = 0; k < ngear; k++)
193     {
194         shortkod[k] = (char *)calloc(kd[k] + 1, sizeof(char));
195     }
196     for (int k = 0; k < ngear; k++)
197     {
198         int l = 0;
199         shortkod[k][l++] = kod[k][0];
200         for (i = 1; i < kd[k]; i++)
201         {
```

```
202             if (kod[k][i] != kod[k][i - 1])
203                 shortkod[k][l++] = kod[k][i];
204         }
205         shortkod[k][l] = 0;
206     }
207     for (k = 1; k <= 3; k++)
208     {
209         i = 1, i2 = 0;
210         while (A[k][i] * A[k][i - 1] > 0.0)
211             i++;
212         i2 = i;
213         i++;
214         int n = 0, N = 0;
215         double per[1000] = {0.0};
216         double Period, cko;
217         while (i < I)
218         {
219             while (A[k][i] * A[k][i - 1] > 0.0 && i < I)
220                 i++;
221             if (i >= I)
222                 break;
223             n++;
224             if (n == strlen(shortkod[k - 1]))
225             {
226                 per[N] = (double)(i - i2);
227                 i2 = i;
228                 n = 0;
229                 N++;
230             }
231             i++;
232         }
233         Period = 0.0;
234         for (int i = 0; i < N; i++)
235             Period += per[i];
236         Period = Period / ((double)N);
237         cko = 0.0;
238         for (int i = 0; i < N; i++)
239             cko += (per[i] - Period) * (per[i] - Period);
240         cko = sqrt(cko / ((double)N));
241 printf("Kanal=%d Numder period=%d Period=%1.1lf
242 CKO=%1.1lf\n",k,N,Period,cko);
243     }
244 }
245 void GearsSignal::SignalModel()
246 { //Модель сигнала - один период
247     int Jmax = 0;
248     double Tadd = 0.0; //Уширение/сужение импульса
249     I0 = (int *)calloc(ngear, sizeof(int));
250     A0 = (double **)calloc(2 * ngear, sizeof(double *));
251     for (int j = 0; j < ngear; j++)
252     {
253         if (TP[j] > 1.0)
254         {
255             IzlImpAdd2(kd[j], kod[j], dt, Tadd, tau1, tau2, tau3, TP[j], A0[2 * j], I0[j]);
256             A0[2 * j + 1] = (double *)calloc(I0[j], sizeof(double));
257             for (int i = 0; i < I0[j]; i++)
258                 A0[2 * j + 1][i] = A0[2 * j][I0[j] - i - 1];
259             if (I0[j] > Jmax)
260                 Jmax = I0[j];
261         }
```

```
262         else
263             I0[j] = 0;
264     }
265     FILE *rez;
266     rez = fopen("model_signal.dat", "w");
267     for (int i = 0; i < Jmax; i++)
268     {
269         fprintf(rez, "%lf ", dt * (float)i);
270         for (int j = 0; j < ngear; j++)
271         {
272             if (i < I0[j])
273 fprintf(rez,"%lf %lf
274 ",A0[2*j][i],A0[2*j+1][i]);
275 else
276 fprintf(rez,"%lf %lf ",0.0,0.0);
277         }
278         fprintf(rez, "\n");
279     }
280     fclose(rez);
281 }
282 void GearsSignal::IzlImpAdd2(int kd, char *kod, double dt, double Tadd,
283     double tau1, double tau2, double tau3, double T, double *&A0, int &J0)
284 {
285     int i, j, J, K3, k0, j0, jb; //Длительность сигнала в отсчетах
286     int Np;                      //Длительность реализации в отсчетах - степень двойки
287     int ll, l, m, m1, m2, m3, ml, k;
288     int km[2][300];
289     double amp, max, freq, t, tau, td, tl, Tl, x, y, ct, cT, T0;
290     double Amp, st, sT, tc, E0;
291     double *Xt;
292     char c;
293     ll = 32000;
294     Xt = (double *)calloc(ll, sizeof(double));
295     k0 = (int)((double)(T) / dt); //Длительность плоской вершины изл. имп.
296     T0 = T + 0.0;
297     T += Tadd; //
298     //-------------------------------------------------------------//
299     ll = 0;
300     m = 0;
301     m1 = 0;
302     for (i = 0; i < kd; i++)
303     { //Разбераем строку на отдельные чипы
304         c = kod[i];
305         k = atoi(&c);
306         if (k == 0)
307             k = -1;
308         if ((i > 0) && (m1 != k))
309         {
310             km[1][ll] = i - m; //Число элементарных элементов в чипе
311             km[0][ll] = m1;    //Фаза чипа
312             m = i;
313             ll++;
314         }
315         m1 = k;
316     }
317     if (kd == 1)
318     {
319         km[0][ll] = 1;
320         km[1][ll] = 1;
321     }
```

```
322     else
323     {
324         km[0][ll] = k;
325         km[1][ll] = kd - m;
326     }
327     //-------------------------------------------------------------//
328     if (tau1 <= tau3)
329         tau = tau1;
330     else
331         tau = tau3;
332     if (tau > tau2 / 2)
333         tau = tau2 / 2; //Ищем самый короткий фронт
334     //---------------Формируем излученный импульс----------------//
335     td = 0.5;
336     if (td > dt)
337         td = dt;
338     tl = T0 / (double)(kd);              //Длительность чипа в in mks
339     tl = (double)((int)(tl / td)) * td; //Длительность чипа Д/Б кратна шагу
340     T0 = (double)(kd)*tl;                //Новая длительность импульса
341     k = (int)(T0 / td);
342     j = 0;
343     //----Создаем базу для фазаманипулированного импульса--------//
344     //----Т.е. огибающую на основе плоского импульса-------------//
345     for (t = 0; t <= T; t += td)
346     { //Текущая длительность в mks
347         j++;
348         //-------------------------------------------------------------//
349         if (tau1 + tau3 <= T)
350         { //Импульс не фазаманипулированный и длиньше
351             //длительности фронтов if (t <= tau1)
352             { //Строим передний фронт когда все нормально
353                 Xt[j] = (1.0 - cos(PI * t / tau1)) / 2.0;
354             }
355             if ((t > tau1) && (t <= T - tau3))
356             {
357                 Xt[j] = 1.0;
358             }
359             if (t > T - tau3)
360             {
361                 Xt[j] = (1.0 + cos(PI * (t - (T - tau3)) / tau3)) / 2.0;
362             }
363         }
364         //-------------------------------------------------------------//
365         else
366         { //Импульс не фазаманипулированный и короче длительности
367             //фронтов
368             tau = T * tau1 / (tau1 + tau3); //Место встречи фронтов
369             if (t <= tau)
370             {
371                 Xt[j] = (1.0 - cos(PI * t / tau1)) / 2.0;
372             }
373             else
374             {
375                 Xt[j] = (1.0 + cos(PI * (t - (T - tau3)) / tau3)) / 2.0;
376             }
377         }
378         //-------------------------------------------------------------//
379     }
380     jb = j;
381     //-------------------------------------------------------------//
```

```
382     //----Теперь наполняем импульс фазавой манипуляцией----------//
383     if (kd > 1)
384     {
385         j = 0;
386         j = (int)(tau1 / 2 / td);
387         for (i = 0; i <= ll; i++)
388         { //Цикл по чипам
389             Tl = tl * (double)(km[1][i]);
390             amp = (double)(km[0][i]);
391             //-------------------------------------------------------------//
392             for (t = 0; t <= Tl - td + 0.000000001; t += td)
393             { //Цикл внутри чипа
394                 j++;
395                 if (j <= k)
396                 {
397                     x = Xt[j];
398                     //-------------------------------------------------------------//
399                     if (Tl >= tau2)
400                     { //Если длительность чипа
401                         //больше фронта if (t <= tau2 / 2)
402                         { //Передний фронт
403                             if (i > 0)
404                             {
405                                 Xt[j] *= amp * sin(PI * t / tau2);
406                                 //Форма фронта в момент переброса фазы в А-
407                                 //квадратуре
408                             }
409                         }
410                         if ((t > tau2 / 2) && (t <= Tl - tau2 / 2))
411                         {
412                             Xt[j] *= amp;
413                             //Yt[j]=0;
414                         }
415                         if (t > Tl - tau2 / 2)
416                         { //Задний фронт
417                             if (i < ll)
418                             {
419                                 Xt[j] *= amp * sin(PI * (t - (Tl - tau2)) / tau2);
420                                 //Форма фронта в момент переброса
421                                 //фазы в А - квадратуре
422                             }
423                             if (i == ll)
424                             {
425                                 Xt[j] *= amp;
426                             }
427                         }
428                     }
429                     //-------------------------------------------------------------//
430                     else
431                     {
432                         tau = Tl / 2.0; //Место встречи фронтов
433                         if (t <= tau)
434                         {
435                             if (i > 0)
436                             {
437                                 Xt[j] *= amp * sin(PI * t / tau2);
438                                 //Форма фронта в момент переброса фазы в А-
439                                 //квадратуре
440                             }
441                         }
```

```
442                         else
443                         {
444                             if (i < ll)
445                             {
446                                 Xt[j] *= amp * sin(PI * (t - (Tl - tau2)) / tau2);
447                                 //Форма фронта в момент переброса
448                                 //фазы в А - квадратуре
449                             }
450                             if (i == ll)
451                             {
452                                 Xt[j] *= amp;
453                             }
454                         }
455                     }
456                     //-------------------------------------------------------------//
457                 } //if(j<=k){
458                 else
459                 {
460                     Xt[j] *= amp;
461                 }
462             } //for(t=0;t<=Tl;t+=td){//Цикл внутри чипа
463         }     //for(i=0;i<=ll;i++){//Цикл по чипам
464         //-------------------------------------------------------------//
465         for (i = j + 1; i <= jb; i++)
466         { //Цикл внутри чипа
467             Xt[i] *= amp;
468         }
469         //-------------------------------------------------------------//
470     } //if(kd>1){
471     //-------------------------------------------------------------//
472     //----Теперь сплайнируем сигнал на отсчеты через dt----------//
473     l = 0;
474     for (t = 0; t <= T; t += dt)
475         l++;
476     J0 = l; //Длительность импульса в отсчетах
477     A0 = (double *)calloc(J0, sizeof(double));
478     l = 0;
479     for (t = 0; t <= T; t += dt)
480     {
481         l++;
482         m = (int)(t / td) + 1;
483         m1 = m + 1;
484         if (m1 <= jb)
485         {
486             A0[l] = Xt[m] + (Xt[m1] - Xt[m]) * (t / td - (double)(m - 1));
487             if (fabs(A0[l]) > 1)
488                 A0[l] = A0[l] / fabs(A0[l]);
489         }
490         else
491         {
492             A0[l] = 0;
493         }
494     }
495     //-------------------------------------------------------------//
496     free(Xt);
497 }
498 void GearsSignal::CalcPeriod()
499 {
500     int i, k, imax1, imax2;
501     double **Emax, **EMAX;
```

```
502     double E1, E2, E, Emax1, Emax2;
503     Emax = (double **)calloc(ngear, sizeof(double *));
504     EMAX = (double **)calloc(ngear, sizeof(double *));
505     for (k = 0; k < ngear; k++)
506     { //Цикл по шестерням
507         Emax[k] = (double *)calloc(I, sizeof(double));
508         EMAX[k] = (double *)calloc(I / 2, sizeof(double));
509     }
510     for (k = 1; k <= ngear; k++)
511     { //Цикл по шестерням
512         for (i = 0; i < I; i++)
513         { //Сдвиг
514             E1 = E2 = E = 0.0;
515             for (int j = 0; j < I; j++) //Интегрируем
516                 if (i + j >= 0 && i + j < I)
517                 {
518                     E1 += A[k][j] * A[k][j];
519                     E2 += A[k][i + j] * A[k][i + j];
520                     E += A[k][j] * A[k][i + j];
521                 }
522             Emax[k - 1][i] = E;
523         }
524         Emax1 = Emax[k - 1][0];
525         imax1 = imax2 = 0;
526         Emax2 = 0.0;
527         for (i = 1; i < I - 1; i++)
528             if (Emax[k - 1][i] > Emax[k - 1][i - 1] &&
529                 Emax[k - 1][i] > Emax[k - 1][i + 1])
530             {
531                 if (Emax2 < Emax[k - 1][i])
532                 {
533                     imax2 = i;
534                     Emax2 = Emax[k - 1][i];
535                 }
536             }
537         TP[k - 1] = (double)(imax2 - imax1);
538     }
539     for (int i = 0; i < ngear; i++)
540         printf("TP[%d]=%lf\n", i, TP[i]);
541     FILE *rez;
542     rez = fopen("period_analiz.dat", "w");
543     for (int j = 0; j < I; j++)
544     {
545         fprintf(rez, "%lf ", dt * (double)j);
546         for (int i = 0; i < ngear; i++)
547         {
548             fprintf(rez, "%lf ", Emax[i][j]);
549         }
550         fprintf(rez, "\n");
551     }
552     fclose(rez);
553 }
554 int GearsSignal::ReadFile(char *_namefile)
555 {
556     FILE *file, *file2;
557     int i = 0;
558     char str[200], ch;
559     file = fopen(_namefile, "r");
560     if (!file)
561         return 0;
```

```
562     fgets(str, '/n', file);
563     while (!feof(file))
564     {
565         fgets(str, '/n', file);
566         if (strlen(str) < 5)
567             break;
568         i++;
569     }
570     fclose(file);
571     I = i;
572     A = (double **)calloc(ngear + 1, sizeof(double *));
573     for (i = 0; i < ngear + 1; i++)
574         A[i] = (double *)calloc(I, sizeof(double));
575     file2 = fopen("gear_data.dat", "w");
576     file = fopen(_namefile, "r");
577     fgets(str, '/n', file);
578     i = 0;
579     while (!feof(file))
580     {
581         fgets(str, '/n', file);
582         if (strlen(str) < 5)
583             break;
584 sscanf(str,"%lf %lf %lf
585 %lf",&A[0][i],&A[1][i],&A[2][i],&A[3][i]);
586 for(int j=0; j<=ngear; j++){
587             if (j > 0)
588                 A[j][i] = 1.0 - 2.0 * A[j][i];
589 }
590 fprintf(file2,"%lf %lf %lf
591 %lf\n",A[0][i],A[1][i],A[2][i],A[3][i]);
592 i++;
593     }
594     fclose(file);
595     fclose(file2);
596     return 1;
597 }
598 void GearsSignal::PrintArray()
599 {
600     FILE *rez;
601     int Jmax = 0, k;
602     int *DI;
603     DI = (int *)calloc(ngear, sizeof(int));
604     for (int i = 0; i < ngear; i++)
605         DI[i] = (int)DT[i];
606     rez = fopen("new_signal.dat", "w");
607     fprintf(rez, "T A\n");
608     for (int i = 0; i < I; i++)
609     {
610         fprintf(rez, "%d ", i);
611         for (int j = 0; j < ngear; j++)
612         {
613             int di = i - DI[j];
614             k = direction[j];
615             if (di >= 0 && di < I0[j])
616                 fprintf(rez, "%lf ", A0[k][di]);
617             else
618                 fprintf(rez, "%lf ", 0.0);
619         }
620         fprintf(rez, "\n");
621     }
```

```
622     free(DI);
623     fclose(rez);
624 }
```

Класс, в котором объеденены методы анализа анализа измеренного сигнала(получен на макете спутника)

```
1  #ifndef _GEARS_
2  #define _GEARS_
3  struct Gears
4  { //для формы шестерни
5      int kd;
6      char *kod;
7  };
8  class GearsSignal
9  {
10   private:
11     char *namefile;
12     int *kd;                  //Длина кодовой последовательности
13     char **kod;               //Форма кодовой последовательности
14     double tau1, tau2, tau3; //Длины фронтов модельного импульса
15     double dt;                //Расстояние между отсчетами
16     int ngear;
17     int I;            //Кол-во отсчетов в измеренной сигнале
18     double **A;       //Измеренный сигнал (0-я колонка - время)
19     double *TP;       //Период сигнала для каждой шестерни
20     int *I0;          //Кол-во отсчетов в модельном импульсе
21     double **A0;      //Форма модельного сигнала
22     char *direction; //Направление вращения
23     double *DT;       //Сдвиг по времени
24     double *Amp;      //Амплитуда кор-функции
25     double *Err;      //Невязка между измерениями и моделью
26     void InitArray();
27     int ReadFile(char *_namefile); //Чтение данных из файла
28     void CalcPeriod();              //Определяем период сигнала для каждой шестерни
29     void sort(double *A, int n);
30     void SignalModel();
31     void IzlImpAdd(int kd, double dt, double Tadd, double tau1, double tau2,
32         double tau3, double T, double *&A0, int &J0); //Модель сигнала
33     void IzlImpAdd2(int kd, char *kod, double dt, double Tadd, double tau1,
34         double tau2, double tau3, double T, double *&A0, int &J0); //Модель сигнала
35     void CorAnaliz2(char rw);
36     double *CorAnaliz(double *&A1, int I1, double *&A2, int I2, double &DT,
37                       double &AmpAkf, char rw);
38     void PrintArray();
39     void EnergyError();
40
41   public:
42     void calc_signal();
43     void control_period();
44     GearsSignal();
45     GearsSignal(char *nameconfig);
46     void GearsAnaliz();
47 };
48 #endif
```

Главная программа, из которой вызывается метод для анализа периодичности измеренного сигнала

```
#include <iostream>
#include <vector>
#include <math.h>
#include "Gears.h"
using namespace std;
void main()
{
    GearsSignal gears("gears_config.dat");
    gears.control_period();
}
```

Задача 5.1.2. 27 очков

Цель:

1. Получить навыки работы с помехоустойчивым кодированием, научиться использовать алгоритмы кодирования и декодирования сообщения, работа с байтами и битами (работа с бинарными файлами), освоить методику пакетной передачи данных.
2. Получить навыки работы со статистическим анализом данных.

Результаты решения задачи могут быть использованы при решении задачи 5

Постановка задачи. Кодирование/Декодирование

По ИК - каналу (со спутника на радар) передается телеметрическая информация и данные. В канале телеметрии передается координатная информация: время в секундах с момента начала движения и (X, Y) координаты спутника в системе координат стенда. Канал телеметрии очень надежный в нем отсутствуют шумы и помехи, информация передается без искажений. По каналу данных передается фотографии спутника большой планеты в высоком разрешении. В канале данных присутствует шум и помехи, требующие организации помехозащитного кодирования передаваемых данных. Шум сильнее если радар не сопровождает спутник.

Задачи команд:

Исследовать характер шума в тестовом канале и передать информацию без потерь. Написать программы: кодер загружается на передатчик (спутник) и декодер загружается на приемник (радар), позволяющее восстановить передаваемое сообщение и исправить возможные ошибки.

Краткое описание подзадач:

1. Научиться работать со стендом спутник – радар. Запустить передачу данных со спутника посредством ИК канала и начать приём данных на «радаре».
2. Исследовать свойства шума, на основе данного исследования выбрать метод помехоустойчивого кодирования.
3. Разработать программу кодер (действует на стороне «спутника»).
4. Разработать программу декодер (действует на стороне «радара»).
5. Провести тестирование программ кодера и декодера в модельных каналах с различным уровнем сложности распределения ошибок. (бонусные очки, максимум 10)
6. Используя разработанные и протестированные программы кодера/декодера

передать без ошибок тестовое сообщение со спутника (основные очки, максимум 17).

Расчет очков (17 + 10)

Расчет очков задачи 2а (оценка эффективности кодера и декодера на этапе тестирования)

Для подсчета верно переданных данных была написана программа, сравнивающая файлы и считающая количествво неисправленных ошибок в переданном файле. Таким образом, параметром V, оценивающим точность передачи данных являлось отношение количества байт (n), в которых была обнаружена не исправленная ошибка к общему количеству ошибок (N) в принятом файле прошедшему модельный канал связи:

$$V = \frac{n}{N}$$

Формула, по которой рассчитывается количество очков (I) по точности передачи V, имеет вид:

$$I = (1 - V) \cdot \cdot 0.37,$$

где — максимальное число основных очков за задачу 2а. Коэффициент 0.37 связан с долей основных очков, отведенных на решение задачи 2а.

Расчет очков задачи 2б (оценка эффективности кодера и декодера)

Для подсчета верно переданных данных была написана программа, сравнивающая файлы и считающая количество неисправленных ошибок в переданном файле. Таким образом, параметром V, оценивающим точность передачи данных являлось отношение количества байт (n), в которых была обнаружена не исправленная ошибка к общему количеству ошибок (N) в принятом файле прошедшему реальный канал связи:

$$V = \frac{n}{N}$$

Формула, по которой рассчитывается количество очков (I) по точности передачи V, имеет вид:

$$I = (1 - V) \cdot M \cdot 0.63,$$

где M — максимальное число основных очков за задачу 2б. Коэффициент 0.63 связан с долей основных очков, отведенных на решение задачи 2б.

Решение

Код решения задачи (блочное кодирование):

Программа осуществляетблочное кодирование - входной файл разбивается на блоки с четко заданной структурой. В принятом файле блоки оказываются перемешанными из-за периодического попадания спутника в «слепые» зоны радара. Программа декодер – находит целые блоки и далее выстраивает из них исходное сообщение, выставляя блоки согласно их номерам заданном на этапе кодирования.

Программа блочного кодерования:

```
1 #include <string.h>
2 #include <unistd.h>
```

```
#include <fcntl.h>
#ifndef __CODE_DECODE_H__
#define __CODE_DECODE_H__
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define DATA_BLOCK_LEN 400
#define CODE_BLOCK_LEN (sizeof(unsigned long))
#define FULL_BLOCK_LEN (DATA_BLOCK_LEN + CODE_BLOCK_LEN)
#define MAX_FILE_SIZE 20000000
#define HEADER_MARK "ABCDEFG"
#define FOOTER_MARK "HIJKLMN"
#define HEADER_MARK_LEN 7
#define FOOTER_MARK_LEN 7
typedef struct
{
    char head[HEADER_MARK_LEN + 1];
    unsigned int code_block;
    long block_len;
} header_code_type;
typedef struct
{
    char foot[FOOTER_MARK_LEN + 1];
    unsigned int code_block;
    // long block_len;
} footer_code_type;
#define HEADER_BLOCK_LEN (sizeof(header_code_type))
#define FOOTER_BLOCK_LEN (sizeof(footer_code_type))
#endif
int main(int argn, char *argv[])
{
    // FILE* tmpfile;
    int mode;
    mode = 1;
    unsigned long code_block = 0;
    long i, j, length;
    char *block;
    unsigned char *file_content;
    unsigned char *new_file_content;
    long file_len;
    long block_len;
    char srch_block[255];
    FILE *in;
    FILE *out;
    in = fopen(argv[1], "rb");
    out = fopen(argv[2], "wb");
    file_content = calloc(MAX_FILE_SIZE + 1, sizeof(char));
    if (!file_content)
    {
        fprintf(stderr, "no memory\n");
        exit(1);
    }
    new_file_content = calloc(MAX_FILE_SIZE + 1, sizeof(char));
    if (!new_file_content)
    {
        fprintf(stderr, "no memory\n");
        exit(1);
    }
    block = calloc(100000, /*DATA_BLOCK_LEN*2,*/ sizeof(char));
    if (!block)
```

```
63      {
64          fprintf(stderr, "no memory\n");
65          exit(1);
66      }
67      length = fread(file_content, sizeof(char), MAX_FILE_SIZE + 1, in);
68      if (length > MAX_FILE_SIZE)
69      {
70          fprintf(stderr, "file too long to process\n");
71          exit(1);
72      }
73      long actual_len;
74      header_code_type header_block;
75      footer_code_type footer_block;
76      file_len = 0;
77      // for(i=0;i<length;)
78      for (i = 0; i < length; i++)
79      {
80          memcpy((void *)(&header_block), (void *)(file_content + i),
81              HEADER_BLOCK_LEN);
82          memcpy(&footer_block, file_content + i + HEADER_BLOCK_LEN +
83              DATA_BLOCK_LEN, FOOTER_BLOCK _LEN);
84          if (memcmp((void *)(header_block.head), HEADER_MARK, HEADER_MARK_LEN) == 0 &&
85              memcmp((void *)(footer_block.foot), FOOTER_MARK, FOOTER_MARK_LEN) == 0 &&
86              footer_block.code_block == header_block.code_block &&
87              i + HEADER_BLOCK_LEN + DATA_BLOCK_LEN + FOOTER_BLOCK_LEN < length)
88          {
89              memcpy((void *)(block), file_content + i + HEADER_BLOCK_LEN, DATA_BLOCK_LEN);
90              memcpy(new_file_content + header_block.code_block *
91                  DATA_BLOCK_LEN, block, DATA_BLOCK_ LEN);
92              block[DATA_BLOCK_LEN] = 0;
93              i += HEADER_BLOCK_LEN;
94              i += DATA_BLOCK_LEN;
95              i += FOOTER_BLOCK_LEN;
96              i--;
97              file_len += DATA_BLOCK_LEN;
98              continue;
99          }
100     }
101     fprintf(stderr, "flen:%ld\n", file_len);
102     char *tmp1;
103     long new_len = 20000000;
104     for (i = file_len; i >= 0; i--)
105         if (new_file_content[i] != 0)
106             break;
107     // fwrite(new_file_content,sizeof(char),file_len,out);
108     fwrite(new_file_content, sizeof(char), i + 1, out);
109     fflush(out);
110 }
```

Программа декодирования – поиск блоков и сборка из них исходного сообщения

```
1  #ifndef __CODE_DECODE_H__
2  #define __CODE_DECODE_H__
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <math.h>
6  #define DATA_BLOCK_LEN 400
7  #define CODE_BLOCK_LEN (sizeof(unsigned long))
8  #define FULL_BLOCK_LEN (DATA_BLOCK_LEN + CODE_BLOCK_LEN)
9  #define MAX_FILE_SIZE 20000000
```

```
10  #define HEADER_MARK "ABCDEFG"
11  #define FOOTER_MARK "HIJKLMN"
12  #define HEADER_MARK_LEN 7
13  #define FOOTER_MARK_LEN 7
14  typedef struct
15  {
16      char head[HEADER_MARK_LEN + 1];
17      unsigned int code_block;
18      long block_len;
19  } header_code_type;
20  typedef struct
21  {
22      char foot[FOOTER_MARK_LEN + 1];
23      unsigned int code_block;
24      // long block_len;
25  } footer_code_type;
26  #define HEADER_BLOCK_LEN (sizeof(header_code_type))
27  #define FOOTER_BLOCK_LEN (sizeof(footer_code_type))
28  #endif
29  #include <string.h>
30  #include <unistd.h>
31  #include <fcntl.h>
32  void init_header_code(header_code_type *header_code, footer_code_type *
33                                                          footer_code)
34  {
35      sprintf(header_code->head, HEADER_MARK);
36      sprintf(footer_code->foot, FOOTER_MARK);
37      header_code->code_block = 0;
38      header_code->block_len = 0;
39  }
40  int main(int argn, char *argv[])
41  {
42      char *tmp_file1;
43      char *tmp_file2;
44      long compr_len;
45      unsigned long code_block = 0;
46      header_code_type header_code;
47      footer_code_type footer_code;
48      long i, j;
49      char *block;
50      int mode;
51      if (argn > 1)
52          mode = atol(argv[1]);
53      else
54          mode = 1;
55      FILE *in;
56      FILE *out;
57      tmp_file1 = calloc(sizeof(char), 40000000);
58      tmp_file2 = calloc(sizeof(char), 40000000);
59      long file_len, block_len;
60      block = calloc(DATA_BLOCK_LEN * 2, sizeof(char));
61      code_block = 0;
62      file_len = 0;
63      in = fopen(argv[1], "rb");
64      out = fopen(argv[2], "wb");
65      for (i = 0; !feof(in);)
66      {
67          block_len = fread(block, sizeof(char), DATA_BLOCK_LEN, in);
68          memcpy(tmp_file1 + i, block, block_len);
69          i += block_len;
```

```
    }
    compr_len = 20000000;
    memcpy(tmp_file2, tmp_file1, i);
    compr_len = i;
    file_len = 0;
    init_header_code(&header_code, &footer_code);
    for (j = 0; j < compr_len; j += DATA_BLOCK_LEN)
    {
        block_len = DATA_BLOCK_LEN;
        if (j + DATA_BLOCK_LEN <= compr_len)
            memcpy(block, tmp_file2 + j, DATA_BLOCK_LEN);
        else
        {
            block_len = compr_len - j;
            memcpy(block, tmp_file2 + j, block_len);
        }
        file_len += block_len;
        for (i = block_len; i < DATA_BLOCK_LEN; i++)
            block[i] = 0;
        header_code.block_len = DATA_BLOCK_LEN;
        header_code.code_block = code_block;
        footer_code.code_block = code_block;
        code_block++;
        fwrite(&(header_code), sizeof(char), sizeof(header_code), out);
        fwrite(block, sizeof(char), DATA_BLOCK_LEN, out);
        fwrite(&(footer_code), sizeof(char), sizeof(footer_code), out);
        for (i = 0; i < DATA_BLOCK_LEN; i++)
            block[i] = 0;
    }
    code_block = -1;
    fprintf(stderr, "file len: %ld\n", file_len);
    memcpy(block, &file_len, sizeof(long));
    fwrite(block, sizeof(char), DATA_BLOCK_LEN, out);
    fwrite(&code_block, sizeof(char), CODE_BLOCK_LEN, out);
    fflush(out);
    fclose(out);
    //copy result twice to compensate dead regions of satellite
    long len = 0;
    out = fopen(argv[2], "rb");
    len = fread(tmp_file1, sizeof(char), 40000000, out);
    fclose(out);
    out = fopen(argv[2], "wb");
    fwrite(tmp_file1, sizeof(char), len, out);
    fwrite(tmp_file1, sizeof(char), len, out);
    fclose(out);
}
```

Программа компилируется с -lm option

Вариант программы кодирования сигнала помехоустойчивым кодом Хемминга (8,4).

Используется (8,4) так как плотность помех изменяют не большего одного бит в байте

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
```

```
6  unsigned char mask[] = {128, 64, 32, 16, 8, 4, 2, 1};
7  void read_file(const char name[], char *&buff, long &size);
8  void write_file(const char name[], char *&buff, long &size);
9  class Hamming
10 {
11   private:
12     int order(int a);
13     void CharToBinaryArray(char &c, char
14         arr[]); //Разложение битов кода символа в массив
15     char EncodedPartArray(char arr[], int
16         begin); //Кодирование части битов символа (сейчас это 4 бита)
17     char BinaryArrayToChar(char arr[]); //Массив битов преобразуем в символ
18     char DecoderChar(char &e); //Декодируем отдельный символ
19     char UnionChar(char e[]); //Объединение двух отдельных символов при
20     //декодировании(так используется расширенный код Хэмминга)
21     void Encoder(char &c, char e[]); //Кодирование расширенным кодом Хэмминга
22     //(4, 8) из одного символа получаем 2
23     char Decoder(char e[]); //Декодирование пары символов и объединение в один
24   public:
25     unsigned int lencod; //Длина в битах закодированного сообщения
26     unsigned int lenbit; //Длина в битах полезного сообщения
27     unsigned int lenctr; //Кол-во контрольных бит
28     Hamming();
29     Hamming(unsigned int _lencod);
30     void ENCODER(char *&buff, const long size, char *&buff_encod, long &size_encod);
31     void DECODER(char *&buff, long &size, char *&buff_encod, const long size_encod);
32 };
33 void read_file(const char name[], char *&buff, long &size)
34 {
35     FILE *inp;
36     inp = fopen(name, "rb");
37     if (!inp)
38     {
39         printf("File %s not open!\n", name);
40         return;
41     }
42     else
43     {
44         fseek(inp, 0, SEEK_END); //переместить внутренний указатель в конец файла
45             size = ftell(inp);
46         rewind(inp); //Устанавливаем указатель в конец файла
47         buff = (char *)calloc(size, sizeof(char));
48         if (buff == NULL)
49         {
50             printf("Error of memory\n");
51             return;
52         }
53         printf("size = %ld\n", size);
54         long result = fread(buff, 1, size, inp);
55         printf("result = %d\n", result);
56         if (result != size)
57         {
58             printf("Error of read\n");
59             return;
60         }
61         fclose(inp);
62     }
63 }
64 void write_file(const char name[], char *&buff, long &size)
65 {
```

```
66      FILE *out;
67      out = fopen(name, "wb");
68      if (!out)
69      {
70          printf("File %s not open!\n", name);
71          return;
72      }
73      else
74      {
75          long result = fwrite(buff, 1, size, out);
76          if (result != size)
77          {
78              printf("Error of write!\n");
79              return;
80          }
81          fclose(out);
82      }
83  }
84  class CodFile
85  {
86    private:
87      int num; // кол-во байт в блоке
88      char *buff, *buff_encod;
89      long size, size_encod;
90      Hamming hamming;
91      void ReadFile(const char name[],
92          char *&_buff, long &_size);
93          //Чтение данных из файла и сброс содержимого в буфер buff
94      void WriteFile(char name[], char *&_buff, long &_size);
95          //Запись содержимого buff вфайл
96    public:
97      CodFile() {}
98      CodFile(const char *filename); //Заполнение буфера данными из файла
99      void ENCODER(char *name_inp_file, char *name_out_file);
100     void DECODER(char *name_inp_file, char *name_out_file);
101     void ADDNOISE(char *name_inp_file, char *name_out_file);
102     void WriteFile(char simbol, const char *name); //Запись содержимого buff в файл
103     ~CodFile();
104 };
105 void main(int argc, char *argv[])
106 {
107     if (argc < 4)
108     {
109         printf("Run file.exe E/D/N inpfile outfile\n");
110         return;
111     }
112     CodFile codfile;
113     if (argv[1][0] == 'E' || argv[1][0] == 'e')
114     {
115         codfile.ENCODER(argv[2], argv[3]);
116     }
117     else if (argv[1][0] == 'D' || argv[1][0] == 'd')
118     {
119         codfile.DECODER(argv[2], argv[3]);
120     }
121     else if (argv[1][0] == 'N' || argv[1][0] == 'n')
122     {
123         codfile.ADDNOISE(argv[2], argv[3]);
124     }
125     else
```

```
126     {
127         printf("Second param error (E|D)\n");
128     }
129 }
130 CodFile::CodFile(const char *namefile)
131 {
132     read_file(namefile, buff, size);
133     puts(buff);
134     printf("n=%d size = %d\n", strlen(buff), size);
135 }
136 void CodFile::DECODER(char *name_inp_file, char *name_out_file)
137 {
138     read_file(name_inp_file, buff_encod, size_encod);
139     hamming.DECODER(buff, size, buff_encod, size_encod);
140     write_file(name_out_file, buff, size);
141 }
142 void CodFile::ENCODER(char *name_inp_file, char *name_out_file)
143 {
144     read_file(name_inp_file, buff, size);
145     hamming.ENCODER(buff, size, buff_encod, size_encod);
146     write_file(name_out_file, buff_encod, size_encod);
147 }
148 void CodFile::ADDNOISE(char *name_inp_file, char *name_out_file)
149 {
150     read_file(name_inp_file, buff_encod, size_encod);
151     long i;
152     for (i = 0; i < size_encod; i++)
153         buff_encod[i] = buff_encod[i] ^ mask[rand() % 8];
154     write_file(name_out_file, buff_encod, size_encod);
155 }
156 void CodFile::WriteFile(char simbol, const char *name)
157 {
158     FILE *out;
159     if (simbol == 'D' || simbol == 'd')
160     {
161         write_file(name, buff, size);
162     }
163     else if (simbol == 'E' || simbol == 'e')
164     {
165         write_file(name, buff_encod, size_encod);
166     }
167     else
168     {
169         printf("Parametr for write: D|d|E|e\n");
170         return;
171     }
172 }
173 CodFile::~CodFile()
174 {
175     delete[] buff;
176     delete[] buff_encod;
177 }
178 Hamming::Hamming()
179 {
180     lencod = 7; //Общая длина
181     lenbit = 4; //Полезная информация
182 }
183 void Hamming::DECODER(char *&buff, long &size, char *&buff_encod, const long size_encod)
184 {
185     long i = 0, j = 0;
```

```
186     char e[2];
187     size = size_encod / 2;
188     buff = new char[size];
189     while (size_encod > i)
190     {
191         e[0] = buff_encod[i++];
192         e[1] = buff_encod[i++];
193         buff[j++] = Decoder(e);
194     }
195 }
196 void Hamming::ENCODER(char *&buff, const long size, char *&buff_encod, long &size_encod)
197 {
198     long i = 0, j = 0;
199     char e[2];
200     size_encod = 2 * size;
201     buff_encod = new char[size_encod];
202     for (i = 0; i < size; i++)
203     {
204         Encoder(buff[i], e);
205         buff_encod[j++] = e[0];
206         buff_encod[j++] = e[1];
207     }
208 }
209 char Hamming::Decoder(char e[])
210 {
211     char D, d[2];
212     d[0] = DecoderChar(e[0]); //Декодируем
213     d[1] = DecoderChar(e[1]); //Декодируем
214     D = UnionChar(d);         //Объединяем
215     return D;
216 }
217 void Hamming::Encoder(char &c, char e[])
218 {
219     char arr[8];
220     CharToBinaryArray(c, arr);
221     e[0] = EncodedPartArray(arr, 0); //Кодируем Хэмингом(4,7)
222     e[1] = EncodedPartArray(arr, 4); //Кодируем Хэмингом(4,7)
223 }
224 char Hamming::UnionChar(char d[])
225 {
226     char Data[8] = {0};
227     char data[2][8] = {0};
228     CharToBinaryArray(d[0], data[0]);
229     CharToBinaryArray(d[1], data[1]);
230     Data[0] = data[0][0];
231     Data[1] = data[0][1];
232     Data[2] = data[0][2];
233     Data[3] = data[0][4];
234     Data[4] = data[1][0];
235     Data[5] = data[1][1];
236     Data[6] = data[1][2];
237     Data[7] = data[1][4];
238     return BinaryArrayToChar(Data);
239 }
240 char Hamming::DecoderChar(char &e)
241 {
242     char c, c1, c2, c3;
243     char data[8] = {0};
244     CharToBinaryArray(e, data);
245     c1 = data[6] ^ data[4] ^ data[2] ^ data[0];
```

```
246     c2 = data[5] ^ data[4] ^ data[1] ^ data[0];
247     c3 = data[3] ^ data[2] ^ data[1] ^ data[0];
248     c = c3 * 4 + c2 * 2 + c1;
249     if (c)
250         if (data[7 - c] == '0')
251             data[7 - c] = '1';
252         else
253             data[7 - c] = '0';
254     return BinaryArrayToChar(data);
255 }
256 char Hamming::EncodedPartArray(char arr[], int begin)
257 {
258     char data[8] = {0};
259     data[0] = arr[0 + begin];
260     data[1] = arr[1 + begin];
261     data[2] = arr[2 + begin];
262     data[4] = arr[3 + begin];
263     data[6] = data[0] ^ data[2] ^ data[4];
264     data[5] = data[0] ^ data[1] ^ data[4];
265     data[3] = data[0] ^ data[1] ^ data[2];
266     char c = BinaryArrayToChar(data);
267     return c;
268 }
269 void Hamming::CharToBinaryArray(char &c, char arr[])
270 {
271     for (int i = 0; i < 8; i++)
272         arr[i] = c & mask[i] ? '1' : '0';
273 }
274 char Hamming::BinaryArrayToChar(char arr[])
275 {
276     char c = 0;
277     for (int i = 0; i < 8; i++)
278         c |= (arr[i] - '0') ? mask[i] : 0;
279     return c;
280 }
```

Задача 5.1.3. 15 очков

Цель:

1. Научиться работать с программным интерфейсом управления «радаром» (API).
2. Получить навыки работы по проектированию предсказательных алгоритмов, в частности алгоритмов сопровождения движущихся объектов.

Результаты решения задачи могут быть использованы при решении задач 2 и 5.

Постановка задачи

Необходимо разработать алгоритм сопровождения спутника радаром. Имитатор канала связи по легенде трека назван системой «радар-спутник» и состоит из подвижной каретки «спутник» и вращающейся вокруг вертикальной оси каретки «радар», рисунок 4. Каретка «спутник» может передвигаться вдоль рельсы (ось X) и имеет инфракрасный (ИК) передатчик. В зависимости от относительного отклонения спутника от оси наблюдения радара уровень шума в принимаемом канале меняется. Таким образом, моделируется отношение сигнал/шум и диаграмма направленности радара, плотность битового шума зависит от величины углового смещения ΔX, но нигде не превышает одного бита на байт. Для обеспечения наименьших уров-

ней помех в принимаемом сигнале необходимо разработать алгоритм сопровождения спутника радаром.

Управление спутником осуществляется с помощью компьютера RaspberryPi, который получает необходимые для передачи данные от компьютера управления и передает их во время движения. Блок «Радар» оборудован компьютером RaspberryPi и цифровой видеокамерой, с помощью библиотеки OpenCV он распознает положение графического маркера «спутника» в поле зрения камеры и передает угловое положение маркера ΔX относительно центра камеры вдоль оси X на управляющий компьютер.

Одновременно с этим, ИК-приемник «радара» принимает поток данных, передаваемый «спутником».

Краткое описание подзадач:

1. Необходимо разработать алгоритм сопровождения спутника радаром. Входными данными для программы является горизонтальное смещение спутника относительно оси радара ΔX. Для управления радаром используются команды, позволяющие регулировать скорость и направление вращения камеры.
2. Провести тестирование разработанной программы.
3. Использовать разработанную программу сопровождения спутника во время приема данных при решении задач 2 и 5, так как это позволяет значительно снизить уровень помех.

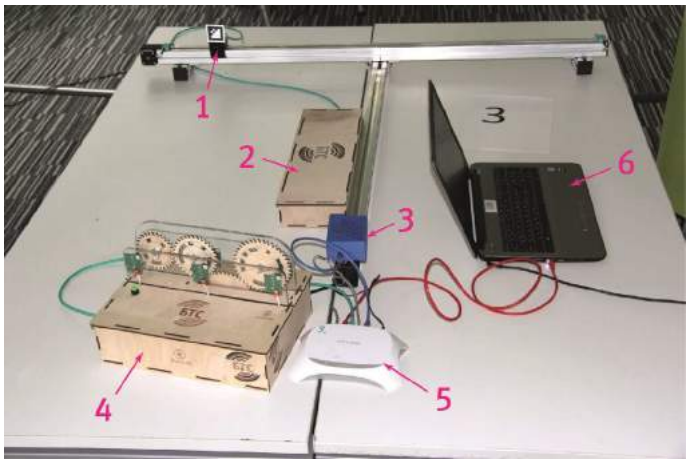


Рисунок 4. Общий вид стенда: 1 – подвижная передающая каретка «спутник» с графическим маркером, 2 – блок управления, 3 – подвижная приемная каретка «радар» с видеокамерой, 4 – макет с шестеренками, 5 – роутер, 6 – компьютер управления стендом.

Расчет очков (10 + 5)

Поскольку для выполнения работы наиболее важным является уровень зашумленности данных, для оценки точности сопровождения используется метод оценки среднего уровня шума.

Во время приема данных по инфракрасному каналу определяется положение спутника (dx) в секторе обзора радара. В зависимости от этого положения изменяется степень принимаемых зашумленности данных - чем дальше от центра - тем

более зашумлены данные. Уровень зашумленности (N) представляет собой величину в пределах от 0 (отсутствие шума) до 1 (самый сильный шум).

В файле сопровождения сохраняется уровень шума на каждые 32 байта принятых данных.

По каждому блоку данных (условное обозначение - точка) рассчитывается уровень невязки сопровождения:

если $|dx| < 100$, то уровень шума $N = |dx|/100$

если $|dx| > 100$, то уровень шума $N = 1$.

Таким образом, при невязке dx больше 100 единиц (пикселов видеокамеры) этот уровень равен единице, при невязке dx меньше 100 единиц этот уровень уменьшается до нуля пропорционально невязке.

При оценке точности сопровождения рассчитывается среднее значение и дисперсия шума. Для оценки используются только случаи, когда было принято достаточно много данных (более 2000 точек или более 64 кБайт данных), это обеспечивает точность оценки порядка 2%.

Нами рассматривались только результаты участников, в которых количество принятых точек составляло порядка 2000 и более, и рассматривалась средняя невязка сопровождения V, как критерий точности сопровождения.

Полученное среднее значение шума V является мерой точности сопровождения, чем эта величина меньше, тем сопровождение точнее.

Расчет основных очков (I) по средней невязке (V):

$$I = M \cdot \left(\frac{V_{min}}{V} \right) /100,$$

где M — максимальное число основных очков за задачу (10). V_{min} - наилучшая точность, достигнутая всеми командами. Бонусные очки добавлялись за более глубокую адаптацию стандартного PID –алгоритма под решаемую задачу.

Решение

Вариант решения задачи сопровождения радаром перемещающегося спутника.

```
1  import java.net.*;
2  import java.io.*;
3  import java.math.*;
4  class Tracking {
5      public static void main(String args[])
6      throws Exception{
7      Client client = new Client();
8      int speed = 1;
9      int dir = 1;
10     int dx;
11     int abs_dx;
12     client.start();
13     client.left(1);
14     Thread.sleep(1000);
15     client.right(speed);
16     for (int i = 0; i < 10; i = 0) // in fact - infinite loop, to fix smart java compiler
```

```
17      {
18          while (!client.readstatus()) {
19              System.out.println("not ready");
20              Thread.sleep(100);
21              client.readstatus();
22          }
23          dx = (int)client.getDx();
24          abs_dx = Math.abs(dx);
25          // System.out.print("dx: "); System.out.println(dx);
26          if (dx < 500 && dx > -500) {
27              speed = 1;
28              if (dx < 30 && dx > -30) {
29                  client.stop();
30                  // System.out.println("stop");
31              }
32              else {
33                  speed = abs_dx / 20;
34                  // System.out.print("move: ");
35                  // System.out.print(speed);
36                  if (dx < 0) {
37                      client.right(speed);
38                      // System.out.println("right ");
39                  }
40                  if (dx > 0) {
41                      client.left(speed);
42                      // System.out.println("left ");
43                  }
44              }
45          }
46          else {
47              speed = 0;
48              // System.out.println("DO NOTHING - CAN NOT SEE");
49          }
50          //search if lost
51          if (client.ifPositionRight()) {
52              speed = 10;
53              client.left(speed);
54              // System.out.println("LOST, SEARCH LEFT");
55          }
56          if (client.ifPositionLeft()) {
57              speed = 10;
58              client.right(speed);
59              // System.out.println("LOST, SEARCH RIGHT");
60          }
61          Thread.sleep(10);
62      }
63      client.quit();
64      client.stop();
65  }
66  }
```

Задача 5.1.4. 18 очков

Цель:

1. Получить навыки работы с функциями, заданными дискретным набором точек.
2. Получить навыки в анализе параметрических функций.

3. Получить модель движения спутника и использовать эти знания в задаче 3 – сопровождение спутника

Результаты решения задачи используются при решении задачи 2 и 3.

Постановка задачи. Вычисление параметров траектории

Проанализируйте телеметирическую информацию, полученную по техническому каналу. Определите параметры движения: период вращения искусственного спутника вокруг естественного спутника большой планеты и период обращения естественного спутника вокруг большой планеты. Известно, что искусственный спутник вращается вокруг естественного спутника по круговой орбите, определите радиус орбиты. Естественный спутник вокруг планеты вращается по эллиптической орбите, определите большую и малую полуось эллипса. Результаты анализа занесите в диагностическую карту.

Краткое описание подзадач:

1. Получите как можно больший объем телеметрической информацию по техническому каналу связи.
2. Постройте и проанализируйте зависимости: $x = x(t), y = y(t), y = y(x)$.
3. Определите параметры движения спутника: периоды обращения и параметры орбит.
4. Подставляя найденные параметры в модель движения и сравнивая результаты моделирования с полученными по техническому каналу данными убедитесь, что полученные зависимости соответствуют друг другу и значит параметры движения определены верно.

Расчет очков (14 + 4)

Команды на проверку сдают диагностические карты. Если команда приносит правильный ответ с первой попытки, то получает 100% (14 очков), если при этом команда решает задачу первой, то добавляются бонусные 4 очка, если команда решает второй, то добавляются бонусные 3 очка и т.д. Если с первой попытки команде не удалось правильно определить все параметры траектории, то бонусные очки не добавляются. Очки рассчитываются исходя из количества правильно определенных параметров, каждый параметр оценивается в 2.8 очка.

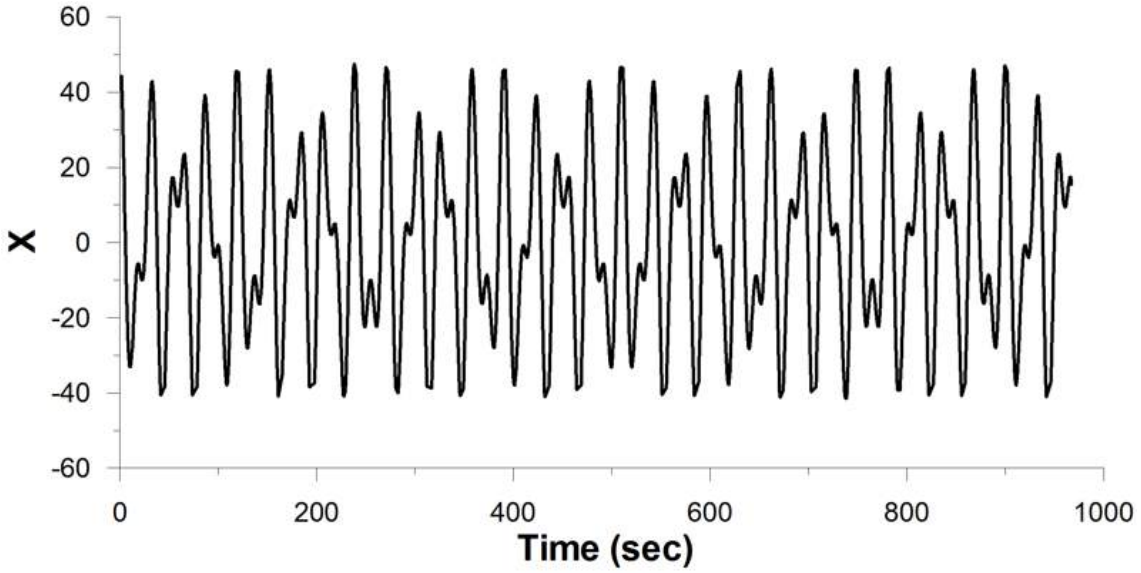
Основные очки рассчитываются следующим образом:

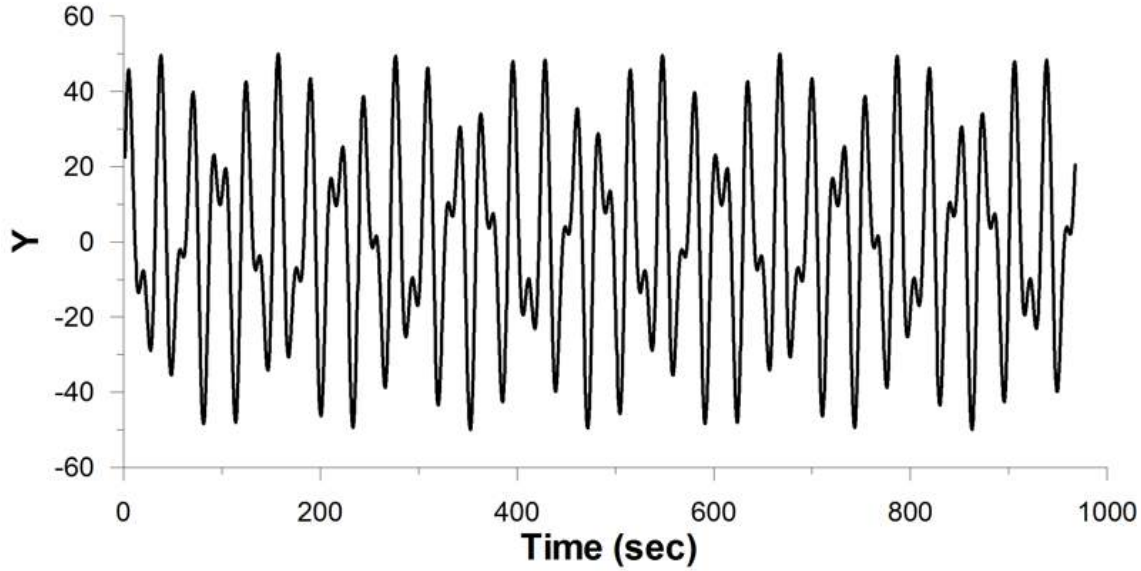
$$\text{Очки} = n \cdot 2.8,$$

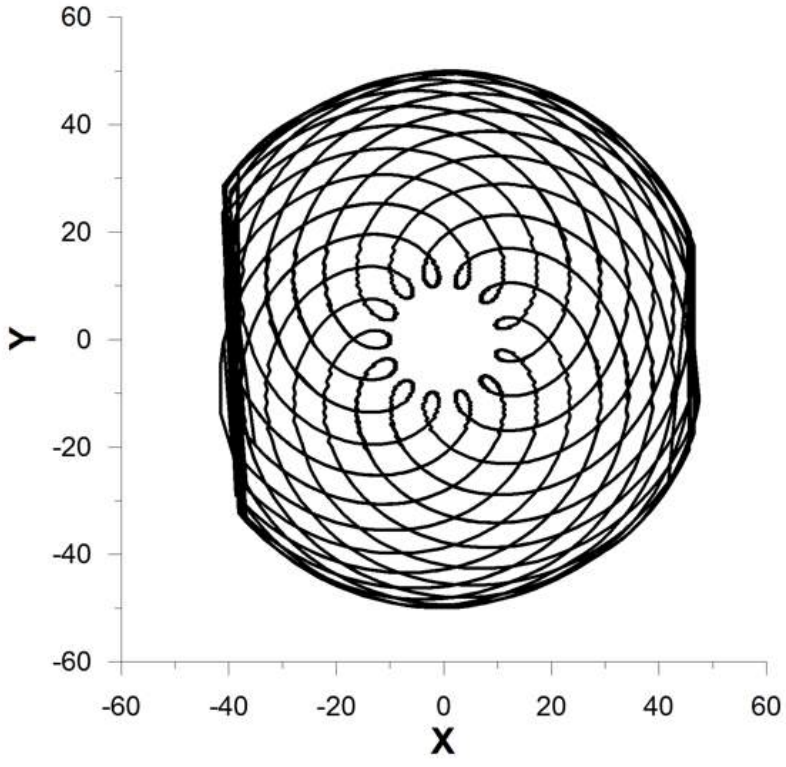
где n – количество правильно определенных параметров. Периоды обращения должны быть оценены с точностью 0.1 секунда, параметры орбит с точность 0.5 см.

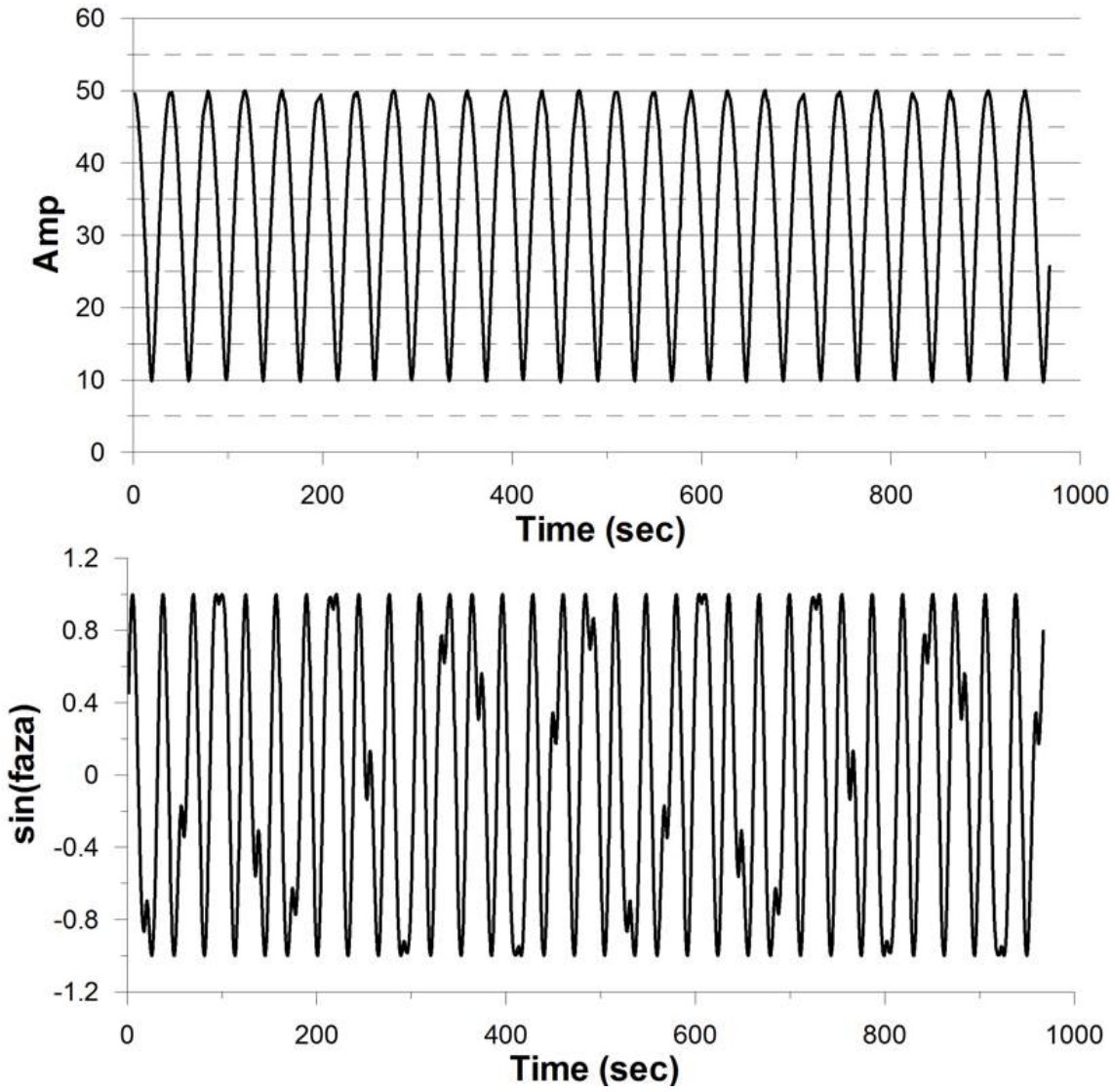
Решение

Для траектории построим на графике входные данные.


60
40
20
0
-20
-40
-60
X
0
200
400
600
800
1000
Time (sec)


60
40
20
0
-20
-40
-60
Y
0
200
400
600
800
1000
Time (sec)


60
40
20
0
-20
-40
-60
Y
-60
-40
-20
0
20
40
60
X



Из рисунка видно, что обе орбиты близки к круговым, тогда проекции модели движения на x и y можно записать:

$$\begin{cases} x = a_1 \cdot \cos(\omega_1 \cdot t + \varphi_1) + a_2 \cdot \cos(\omega_2 \cdot t + \varphi_2), \\ y = a_1 \cdot \sin(\omega_1 \cdot t + \varphi_1) + a_2 \cdot \sin(\omega_2 \cdot t + \varphi_2). \end{cases}$$

где a_1 — радиус орбиты естественного спутника вокруг планеты, a_2 — радиус орбиты искусственного спутника вокруг естественного, ω_1 и ω_2 — соответствующие им круговые частоты. Если перейти в полярную СК и рассмотреть динамику изменения длины радиус-вектора, тогда получим:

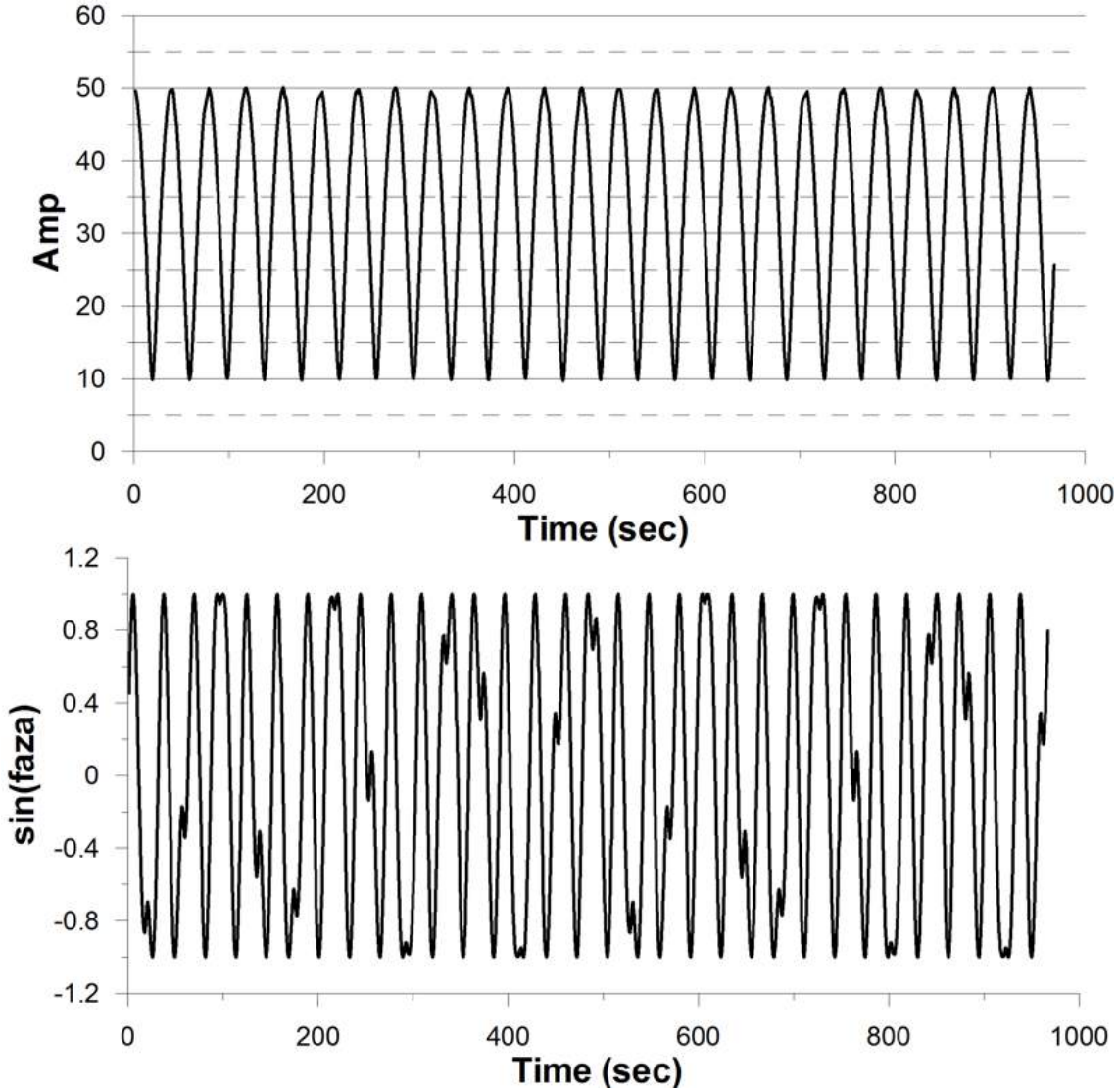
$$r^2 = a_1^2 + a_2^2 + 2a_1a_2\cos(t \cdot (\omega_1 - \omega_2) + \varphi_1 + \varphi_2).$$

Тогда минимальное и максимальное расстояние:

$$\begin{cases} r_{min} = a_1 - a_2, \\ r_{max} = a_1 + a_2. \end{cases}$$

Откуда легко найти a_1 и a_2. Рассмотрим (на верхний график рисунка) динамику

изменения радиус-вектора. В динамике амплитуды видны четкие биения, соответствующие разностной частоте. Период этих биений можно определить, анализируя интервал времени между соседними максимумами.



Найденный период связан с искомыми периодами следующим образом:

$$T_{\text{Биений}} = \frac{T_1 T_2}{T_1 - T_2}.$$

Анализируя таким же образом функцию

$$\sin\left(\operatorname{atan}\left(\frac{x}{y}\right)\right)$$

(нижний график на рисунке), можно определить период «несущей» частоты:

$$\frac{T_{\text{Несущей}}}{2} = \frac{T_1 T_2}{T_1 + T_2}.$$

Решая совместно систему уравнения для $T_{\text{Биений}}$ и $T_{\text{Несущей}}$ определяем соответствующие периоды обращения.

Задача 5.1.5. 30 очков

Цель:

1. Получить навыки работы с беспотерьной передачей данных в условиях канала с потерями.
2. Получить навыки работы с компрессией данных.

Задача является финальным испытанием, для успешного решения необходимо решить четыре предыдущих задачи.

Постановка задачи. Поиск кратеров и определение их размеров и координат

Наша задача состоит в том, чтобы отследить появление новых кратеров на естественном спутнике некоторой удаленной планеты. Область, в которой появление новых кратеров минимально, считается относительно безопасной для оборудования исследовательской базы. Для решения задачи на круговую орбиту вокруг естественного спутника планеты выведен исследовательский спутник с хорошей оптической системой на борту. Исследовательский спутник фотографирует поверхность планеты с очень высоким разрешением, однако, так как пропускная способность канала передачи данных низкая, то передать весь объем данных невозможно. На борту предусмотрена возможность запуска программы, преобразующая снимки, полученные в высоком разрешения, в черно-белый формат (‘0’ и ‘1’) для оперативного анализа. Таким образом, на борту спутника необходимо:

a) проанализировать новый снимок,

b) найти отличия,

c) сформировать список, содержащий: центры найденных областей и их размер (центр прямоугольника и длины его сторон).

По данному списку штатное бортовое ПО из большой карты вырезает выбранные фрагменты и передает их модулю, обеспечивающему помехоустойчивое кодирование передаваемых данных.

Краткое описание подзадач:

1. Получить старую карту в черно-белом формате (‘0’ и ‘1’).
2. Написать и протестировать программу поиска кратеров на основе данных старой карты.
3. Загрузить программу на борт исследовательского спутника.
4. Запустить программу поиска кратеров на борту спутника. Программа должна определить центры и радиусы кратеров на новой карте и данные результаты записать в отдельный файл craters.dat, в отдельных строках которого содержаться координаты центра кратера и его радиус в формате (XYR).
5. Получить через интерфейс общения со спутником список кратеров, выбрать нужный, что дает команду бортовому ПО – выделить с карты в высоком разрешении нужную область и запустить процесс передачи изображения кратера в высоком разрешении. При этом используется программа, обеспечивающая помехоустойчивое кодирование, разработанная на этапе решения 2-й задачи и программа, обеспечивающая эффективное сопровождение спутника радаром для минимизации шумов, разработанная на этапе решения 3-й задачи.

Расчет очков (25 + 5)

При расчете очков за задачу 5 оценивается общая площадь верно определенных и переданных изображений кратеров.

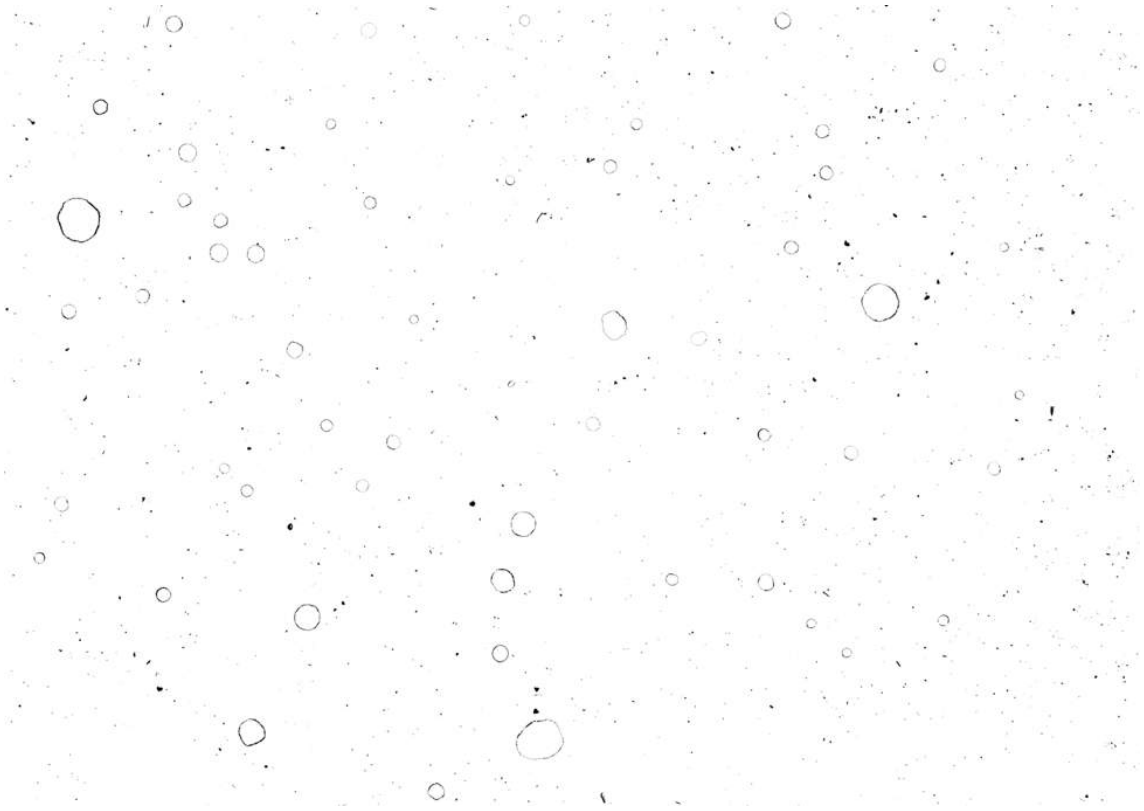
Команда, передавшая кратеры на наибольшую общую площадь, получает 25 очков, очки остальных команд находятся по пропорции относительно максимальных 25 очков. Бонусные 5 очков даются за качество работы программы поиска кратеров и определения их размеров. При этом сравниваются список параметров кратеров, полученный командой и список с заданным расположением кратеров. Параметром V, оценивающим точность определения кратеров является количество совпадений (n) в сравниваемых списках:

$$V = \frac{n}{N},$$

где N – кол-во кратеров в заданном списке.

Решение

На рисунке представлен вариант расположения кратеров, параметры которых нужно определить.



Так как кратеры не пересекаются, то решение можно разбить на три этапа:

1. Поиск связанных точек.
2. Вписывание окружности в найденные точки, так чтобы СКО между найденными точками и найденной окружностью была минимально.
3. Проверка того на сколько хорошо окружность вписана. Если СКО слишком велико или радиус кратера слишком мал, то найденные точки относятся к шуму и их в список не вносим.

Ниже представлен вариант решения задачи на C++:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>

using namespace std;

int rows;
int columns;
char** field;

struct Circle{
        double X0,Y0,R;
        double CK0;
        Circle(){
                Y0 = X0 = 0.0;
                R = 0.0;
                CK0 = 0.0;
        }
};

struct param_find{
        int n;
        double X, Y;
        double X2, Y2;
        double XY;
        double X3, Y3;
        double X2Y;
        double XY2;
        vector<int> pntX;
        vector<int> pntY;
        void instal(){
                n = 0;
                X=0.0;
                Y=0.0;
                X2=0.0;
                Y2=0.0;
                X3=0.0;
                Y3=0.0;
                XY = 0.0;
                X2Y = 0.0;
                XY2 = 0.0;
        }
        param_find(){
                instal();
        }
};

//Поиск связанной группы точек
int CountOne(int i, int j, param_find* param);
Circle* FindParamCircle(param_find* param);

//главная функция которой передаем имя файла с данными
void FoundCratersOnMap(const char namefile[]){

        ifstream in(namefile);

        if(!in.is_open()){
```

```
59                  cout<<"FILE "<<namefile<<" is not open!\n";
60          }
61
62          in>>rows>>columns;
63          cout<<"rows = "<<rows<<", columns="<<columns<<endl;
64
65          field = new char*[rows];
66          for(int i=0; i<rows; i++)
67                  field[i] = new char[columns + 2];
68
69
70          in.getline(field[0],columns + 2,'\n');
71          for(int i=0; i<rows; i++){
72                  in.getline(field[i],columns + 2,'\n');
73          }
74          in.close();
75
76          ofstream out("list_craters.dat");
77          out<<"X0        Y0        R\n";
78
79          vector<param_find*> param;
80          int k = 0;
81          int kc = 0;
82          for(int i = 0; i<rows; i++)
83                  for(int j = 0; j<columns; j++){
84                          param_find* temp = new param_find;
85                          int n = CountOne(i,j,temp);
86                          if(n>50){
87                                  param.push_back(temp);
88                                  Circle* cir = FindParamCircle(param[k]);
89                                  if(cir->R>20.0 && cir->CKO<2.5){
90                                          out<<cir->X0<<'\t'<<cir->Y0<<'\t'<<cir->R<<'\n';
91                                          kc++;
92                                  }
93                                  delete cir;
94                                  k++;
95                          }
96                          else
97                                  delete temp;
98                  }
99
100         out.close();
101 }
102
103 //Поиск параметров окружности
104 Circle* FindParamCircle(param_find* param){
105         Circle* circle = new Circle;
106         double A,B,D,A1,D1;
107         int N = param->n;
108         double X = param->X;
109         double Y = param->Y;
110         double X2 = param->X2;
111         double Y2 = param->Y2;
112         double XY = param->XY;
113         double X3 = param->X3;
114         double Y3 = param->Y3;
115         double XY2 = param->XY2;
116         double X2Y = param->X2Y;
117         double X0,Y0,R,CKO;
118
```

```
119         A = -X*X/(double)(N) + X2;
120         B = -X*Y/(double)(N) + XY;
121         D =  X*X2/(double)(N) + X*Y2/(double)(N) - X3 - XY2;
122         A1 =-Y*Y/(double)(N) + Y2;
123         D1 = X2*Y/(double)(N) + Y*Y2/(double)(N) - Y3 - X2Y;
124
125         Y0 = (-D*B + D1*A)/(B*B - A*A1)/2.0;
126         X0 = (-D*A1 + D1*B)/(A*A1 - B*B)/2.0;
127         R = sqrt((X2-2.0*X0*X+X0*X0*(double)(N)+Y2-2.0*Y0*Y+
128                 Y0*Y0*(double)(N))/(double)(N));
129
130         CKO = 0.0;
131         for(int i = 0; i<N; i++){
132                 double dx = (double)(param->pntX[i]) - X0;
133                 double dy = (double)(param->pntY[i]) - Y0;
134                 double dr = sqrt(dx*dx + dy*dy);
135                 CKO += (R-dr)*(R-dr);
136         }
137         CKO = sqrt(CKO/(double)(N));
138
139         circle->X0 = X0;
140         circle->Y0 = Y0;
141         circle->R = R;
142         circle->CKO = CKO;
143
144         return circle;
145 }
146
147 //Поиск связанной группы точек
148 int CountOne(int i, int j, param_find *param){
149         if(i<0 || i>=rows || j<0 || j>=columns)
150                 return 0;
151         if(field[i][j] != '1')
152                 return 0;
153         else{
154                 field[i][j] = '2';
155                 double x = (double)j;
156                 double y = (double)i;
157                 param->n++;
158                 param->X += x;
159                 param->Y += y;
160                 param->X2 += x*x;
161                 param->Y2 += y*y;
162                 param->XY += x*y;
163                 param->X3 += x*x*x;
164                 param->Y3 += y*y*y;
165                 param->X2Y += x*x*y;
166                 param->XY2 += x*y*y;
167                 param->pntX.push_back(j);
168                 param->pntY.push_back(i);
169                 return 1 + CountOne(i,j-1,param) + CountOne(i-1,j-1,param)
170                         + CountOne(i-1,j,param) + CountOne(i-1,j+1,param)
171                         + CountOne(i,j+1,param) + CountOne(i+1,j+1,param)
172                         + CountOne(i+1,j,param) + CountOne(i+1,j-1,param);
173         }
174 }
```

Система оценки задач командного тура

Команды получали за командный тур от 0 до 100 баллов: команда, набравшая максимальное число внутриигровых очков, получала 100 баллов. И становилась командой-победителем. Остальные команды, получали баллы, нормированные на этот результ по формуле S = 100×x/MAX, где x — число внутриигровых очков, набранных командой, MAX — число очков, набранное в финале командой-победителем. Таким образом, победитель финала получает всегда 100 баллов, остальные — от 0 до 100, пропорционально показанным ими результатам.

В таблице 1 приведена таблица внутриигровых очков по задачам командного тура, в дополнение к основным очкам за полноценное решение задачи предполагаются бонусные очки, за скорость и точность решения. Бонусные очки введены, для определения победителя при прочих равных условиях, а также, чтобы поощрить командную работу. В правой колонке таблицы 1 приведена формула расчета максимального числа очков за задачу $M + N$, где M – основные очки, N – бонусные очки. Бонусные очки даются за скорость и точность при решении задачи. Количество неудачных попыток сдать задачу снижает количество бонусных очков. Бонусные очки даются за оригинальную идею при решении задачи, более устойчивый алгоритм и т.д.

Таблица 5.1: Таблица 1. Разбалловка задач по профилю «Технологии беспроводной связи»

Задачи командного тура	Очки (внутренняя шкала)
1. Работа с узором шестеренки - определение кода Хемминга и декодирование. Определение среднего периода и среднеквадратичного отклонения от среднего – результаты вносятся в диагностическую карту.	6+4
2. Передача тестовой картинки со спутника (кодирование/декодирование, сжатие, разбиение на блоки). Бонус. Исследовать характер шума в тестовом канале и передать информацию без потерь.	17+10
3. Разработка алгоритма слежения за спутником.	10+5
4. Определение параметров траектории	14+4
5. Восстановление новой карты. Передача наибольшего количества кратеров.	25+5

В таблице 2 представлен вид диагностической карты, карта заполняется при решении первой и четвертой задач.

Таблица 5.2: Таблица 2. Формат диагностической карты

Диагностическая карта

Название команды: _____

Стенд с шестернями	Шестеренка 1	Шестеренка 2	Шестеренка 3	Время
Код Хэмминга (n,k)	1. 2. 3.	1. 2. 3.	1. 2. 3.	1. 2. 3.
Закодированное двоичное слово	1. 2. 3.	1. 2. 3.	1. 2. 3.	1. 2. 3.
Декодированное Двоичное слово	1. 2. 3.	1. 2. 3.	1. 2. 3.	1. 2. 3.
Декодированное слово в десятичной системе	1. 2. 3.	1. 2. 3.	1. 2. 3.	1. 2. 3.
Средний период вращения (мс)	1. 2. 3.	1. 2. 3.	1. 2. 3.	1. 2. 3.
Среднеквадратичное отклонение (мс)	1. 2. 3.	1. 2. 3.	1. 2. 3.	1. 2. 3.

Оценка параметров траектории

1. Период обращения естественного спутника вокруг планеты (секунды):

1.1. _____,

1.2. _____,

1.3. _____.

2. Размер большой полуоси (см):

2.1. _____,

2.2. _____,

2.3. _____.

3. Размер малой полуоси (см):

3.1. _____,

3.2. _____,

3.3. _____.

4. Период обращения искусственного спутника вокруг естественно спутника (секунды):

4.1. _____,

4.2. _____,

4.3. _____.

5. Радиус круговой орбиты искусственного спутника вокруг естественно спутника (см):

5.1. _____,

5.2. _____,

5.3. _____.

5.2. Приложения к задачам

Приложение А - Описание стенда трека

Стенд трека состоит из двух основных частей:

- имитатор некоторого механизма (макет с шестеренками), расположенного на удаленном объекте,
- имитатор канала связи с удаленным объектом (макет системы «спутник – радар»).

В качестве механизма, расположенного на некотором удаленном объекте, был создан макет с системой шестеренок, приводимых в движение двигателем и оборудованный тремя оптопарами. Этот макет позволяет, при включении, регистрировать оптический сигнал, проходящий через прорези в шестеренках и записывать его в файл. Оптический сигнал считывается системой регистрации макета (на базе RaspberryPi и STM32L432) через равные промежутки времени (1 мс). Сформированный таким образом файл представляет собой последовательность из ‘0’ и ‘1’ (двоичный код) с каждой шестерни макета и в дальнейшем передается на компьютер управления, для последующего анализа участниками трека.

Имитатор канала связи по легенде трека назван системой «радар-спутник» и состоит из подвижной каретки «спутник» и вращающейся вокруг вертикальной оси каретки «радар». Каретка «спутник» может передвигаться вдоль рельсы (ось X) и имеет инфракрасный (ИК) передатчик. Управление осуществляется с помощью компьютера RaspberryPi, который получает необходимые для передачи данные от компьютера управления и передает их во время движения. Блок «Радар» оборудован компьютером RaspberryPi и цифровой видеокамерой, с помощью библиотеки OpenCV он распознает положение графического маркера «спутника» в поле зрения камеры и передает положение маркера на управляющий компьютер. Одновременно с этим ИК-приемник «радара» принимает поток данных, передаваемый «спутником». Скорость передачи данных ограничена на аппаратном уровне и составляет 115.2 кбит/с.

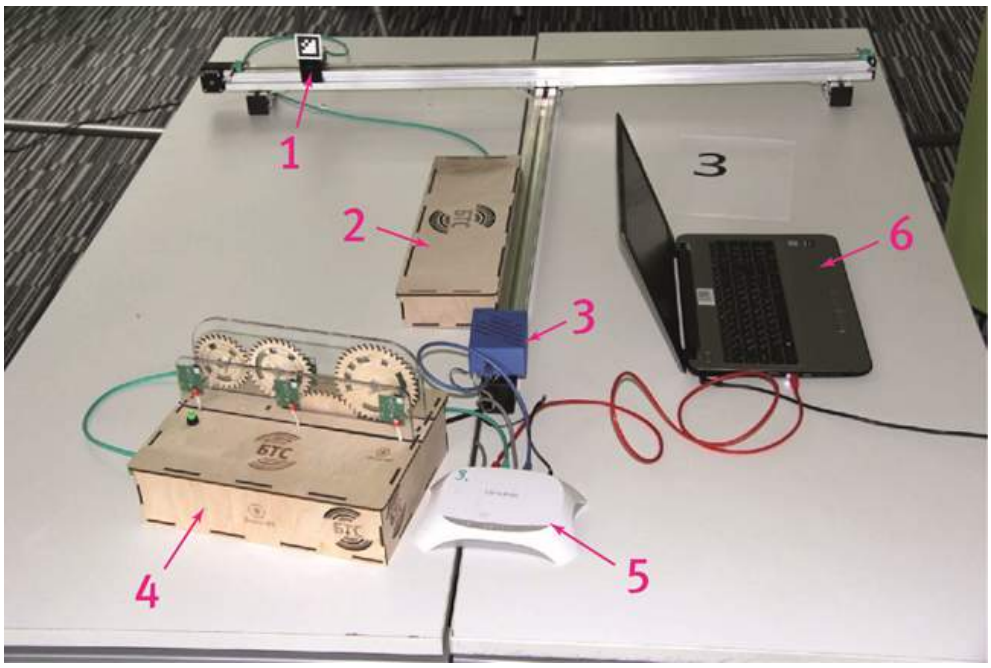


Рисунок 1. Общий вид стенда:

1 – подвижная передающая каретка «спутник» с графическим маркером, 2 – блок управления, 3 – подвижная приемная каретка «радар» с видеокамерой, 4 – макет с шестеренками, 5 – роутер, 6 – компьютер управления стендом.

На рисунке 1 представлены основные части стенда:

1 – подвижная передающая каретка («спутник»), с помощью двигателя и приводного ремня может перемещаться вдоль оси X влево-вправо по программно-задаваемому закону движения. Имеет на борту ИК-передатчик, передающий, во время движения, данные телеметрии бортового механизма.

2 – блок управления, содержит управляющий компьютер «спутника» на базе RaspberryPi, блоки питания и логику управления моторами.

3 – подвижная приемная каретка («радар») имеет ИК-приемник, для приема данных со «спутника», видеокамеру для отслеживания «спутника» и двигатель для вращения каретки вокруг вертикальной оси.

4 – макет с шестеренками, позволяет регистрировать в цифровом виде сигналы с оптических пар, установленных на каждой шестеренке и записывать их в файлы, для управления данной системой используется компьютер RaspberryPi и вспомогательный микропроцессор STM32L432.

5 – роутер, объединяет все компьютеры стенда (3 штуки RaspberryPi и управляющий ноутбук) в локальную сеть, а также обеспечивает дальнейшее соединение с локальной сетью организаторов для удаленного управления и настройки стенда.

6 – компьютер управления стендом (ноутбук), предоставляет участникам трека web-интерфейс к задачам, управляет всеми основными блоками стенда, а также позволяет создавать и загружать программное обеспечение для решения задач на языках C, C++, Python, Java.

На рисунке 2 приведена схема взаимодействия всех блоков стенда между собой, а также адреса в локальной сети. Каждый компьютер стенда имеет фиксированный IP-

адрес TCP/IP протокола, через который происходит управление, прием и передача данных для выполнения задач.

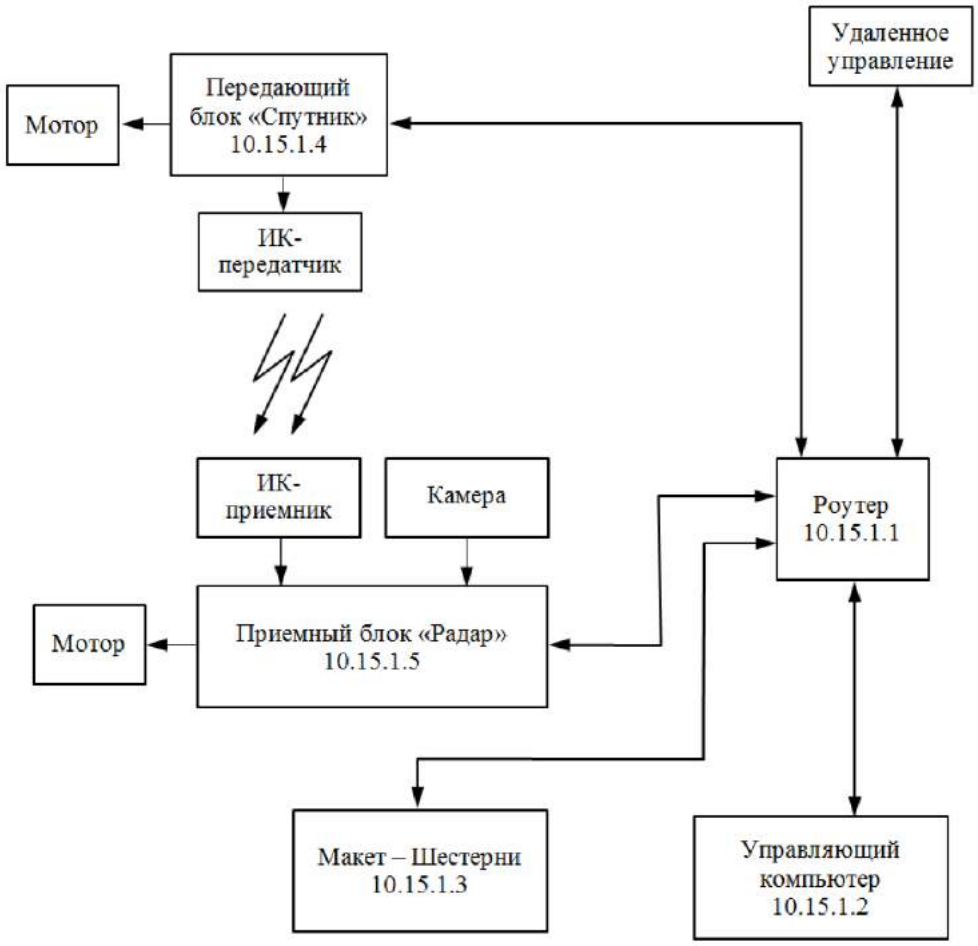


Рисунок 2. Схема взаимодействия блоков стенда и организация локальной сети.

Со стороны организаторов стенда имеется возможность в любой момент времени оперативно подключиться к любому компьютеру и произвести контроль за выполнением задач, а также диагностику неисправностей.