Второй отборочный этап

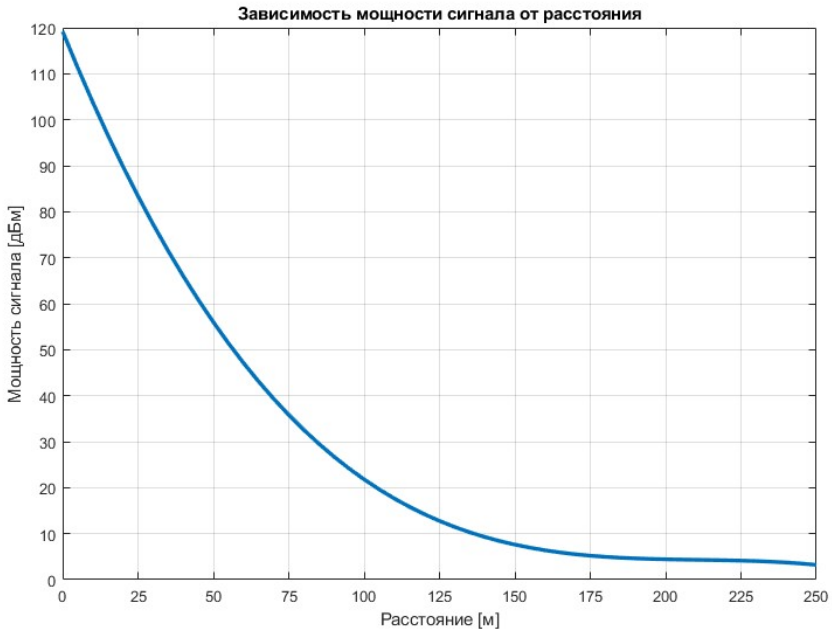
Индивидуальная часть

Задача II.1.1. Определение местоположения БПЛА методом трилатерации (15 баллов)

В процессе полета БПЛА получает информационный сигнал от трех вышек связи и измеряет мощность этого сигнала. Известна экспериментальная зависимость мощности полученного сигнала от расстояния до источника (вышки связи).

Мощность сигнала (дБм)	Дальность до вышки (м)
120	0
80	25
60	50
20	100
5	200
3	250

Аппроксимация этой зависимости полиномом 3 степени имеет вид:



Напишите программу на языке C++, определяющую местоположение БПЛА по известным координатам вышек связи в трехмерном декартовом пространстве и по заданным измерениям мощности сигнала до каждой из вышек. При пересчете мощности сигнала в расстояние до вышек связи следует использовать аппроксимацию

полиномом 3 степени (функция на языке C++ для аппроксимации сигнала прилагается к условию задачи). Гарантируется, что расстояние от БПЛА до вышек связи не превышает 250 метров, и что задача имеет решение. Кривизну поверхности земли при расчетах не учитывать.

Примечание: на графике представлен примерный вид зависимости расстояния от мощности сигнала. Пожалуйста, используйте данные таблицы для получения правильных коэффициентов пересчета мощности сигнала в расстояние. Для предотвращения ошибок перевода используйте следующий код для получения коэффициентов с помощью предоставленой функции polyfit (порялок коэффициентов x^0, x^1, x^2, x^3):

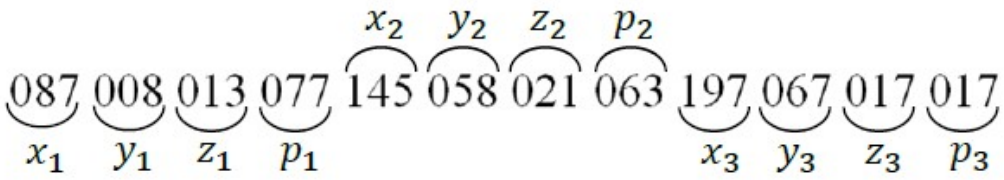
polyfit(Distance, Power, 6, 3, coefs);

Формат входных данных

Массив координат (x, y, z) вышек связи в метрах и измеренной мощности (P) сигнала в дБм, заданных последовательно, через пробелы. Гарантируется, что все значения координат и мощностей — положительные и целые величины (без дробной части).

XXX YYY ZZZ PPP XXX YYY ZZZ PPP XXX YYY ZZZ PPP

Пример входных данных:



Формат выходных данных

Значения трех координат (x, y, z) БПЛА в декартовом пространстве (в метрах) с точностью до 2 знака после запятой, записанные через пробел. Если задача имеет несколько решений, правильным ответом считается среднее арифметическое всех решений (например, $\frac{x_1+x_2}{2}$, $\frac{y_1+y_2}{2}$, $\frac{z_1+z_2}{2}$). Запись значений должна соответствовать формату: 3 целых числа, точка и сотые доли значения (например, значение 2.1 записывается как 002.10).

XXX.XX YYY.YY ZZZ.ZZ

Пример результата программы:

110.72 031.06 017.09

Список литературы

1. Трилатерация [электронный ресурс], `https://ru.wikipedia.org/wiki/Трилатерация`.
2. Функция для аппроксимации данных на языке C++ [электронный ресурс], https://github.com/pec-orange/cpp-functions/blob/main/cpp_polyfit.cpp.

3. Аппроксимация [электронный ресурс], https://ru.wikipedia.org/wiki/Аппроксимация.

Примеры

Пример №1

Стандартный ввод
087 008 013 077 145 058 021 063 197 067 017 017
Стандартный вывод
110.72 031.06 017.09

Пример №2

Стандартный ввод
070 112 016 057 157 158 034 055 144 157 030 070
Стандартный вывод
112.35 135.44 024.53

Решение

```
#include <iostream>
#include <math.h>
#include <vector>
// Libraries for .clue generation
#include <fstream>
#include <sstream>
#include <iomanip>
using namespace std;
struct Tower {
        double x;
        double y;
        double z;
        double distance;
};
// Function prototypes
Tower solve(Tower tow1, Tower tow2, Tower tow3);
int polyfit(const double* const dependentValues,
        const double* const independentValues,
        unsigned int countOfElements,
        unsigned int order,
        double* coefficients);
// Constant values
const double Power[6] = { 120.0, 80.0, 60.0, 20.0, 5.0, 3.0 };
const double Distance[6] = { 0.0, 25.0, 50.0, 100.0, 200.0, 250.0 };
// Main function
int main()
{
        cout << "Welcome to the triangulation program.\n\n";
        // Read data from the input file
        string ifilename;
        ifstream ifile;
        stringstream ss; ss << 0;
        Tower towers[3];
```

```
34          ifilename = "input_2";
35          cout << "Input file: " << ifilename << endl;
36          string sd;
37          ifile.open(ifilename.c_str(), ios::out);
38          int i = 0;
39          while (ifile >> sd)
40          {
41                  towers[i].x = stod(sd);
42                  ifile >> sd;
43                  towers[i].y = stod(sd);
44                  ifile >> sd;
45                  towers[i].z = stod(sd);
46                  ifile >> sd;
47                  towers[i].distance = stod(sd);
48                  i++;
49          }
50          ifile.close();
51          double coefs[4];
52          polyfit(Distance, Power, 6, 3, coefs);
53          cout << "coefs: " << coefs[0] << "\t" << coefs[1] << "\t" << coefs[2] <<
54                  "\t" << coefs[3] << endl;
55          for (int i = 0; i < 3; i++)
56          {
57                  towers[i].distance = coefs[0] + towers[i].distance * coefs[1] +
58                          pow(towers[i].distance, 2) * coefs[2] +
                            ↪  pow(towers[i].distance, 3) * coefs[3];
59          }
60          cout << "Tower1: " << towers[0].x << "\t" << towers[0].y << "\t" <<
61                  towers[0].z << "\t" << towers[0].distance << endl;
62          cout << "Tower2: " << towers[1].x << "\t" << towers[1].y << "\t" <<
63                  towers[1].z << "\t" << towers[1].distance << endl;
64          cout << "Tower3: " << towers[2].x << "\t" << towers[2].y << "\t" <<
65                  towers[2].z << "\t" << towers[2].distance << endl;
66          Tower sol = solve(towers[0], towers[1], towers[2]);
67          cout << setprecision(4);
68          cout << "\nFinal solution: " << sol.x << " " << sol.y << " " <<
69                  sol.z << endl;
70          return 0;
71  }
72  Tower solve(Tower tow1, Tower tow2, Tower tow3)
73  {
74          Tower solution;
75          // Coefs of the tow1-tow2 intersection plane equation
76          double a1 = 2 * (tow2.x - tow1.x);
77          double b1 = 2 * (tow2.y - tow1.y);
78          double c1 = 2 * (tow2.z - tow1.z);
79          double k1 = tow2.x * tow2.x - tow1.x * tow1.x + tow2.y * tow2.y -
80                  tow1.y * tow1.y + tow2.z * tow2.z - tow1.z * tow1.z -
81                  tow2.distance * tow2.distance + tow1.distance * tow1.distance;
82          // Coefs of the tow2-tow3 intersection plane equation
83          double a3 = 2 * (tow2.x - tow3.x);
84          double b3 = 2 * (tow2.y - tow3.y);
85          double c3 = 2 * (tow2.z - tow3.z);
86          double k3 = tow2.x * tow2.x - tow3.x * tow3.x + tow2.y * tow2.y -
87                  tow3.y * tow3.y + tow2.z * tow2.z - tow3.z * tow3.z -
88                  tow2.distance * tow2.distance + tow3.distance * tow3.distance;
89          // Find y as a linear expression of z
90          // y = e*z + f
91          double e, f;
92          if (a1 == 0)
```

```
93          {
94                  e = -c1 / b1;
95                  f = k1 / b1;
96          }
97          else if (a3 == 0)
98          {
99                  e = -c3 / b3;
100                 f = k3 / b3;
101         }
102         else
103         {
104                 double a31 = a3 / a1;
105                 e = -(a31 * c1 - c3) / (a31 * b1 - b3);
106                 f = (a31 * k1 - k3) / (a31 * b1 - b3);
107         }
108         // Find x as a linear expression of z
109         // x = g*z + h
110         double g, h;
111         if (b1 == 0)
112         {
113                 g = -c1 / a1;
114                 h = k1 / a1;
115         }
116         else if (b3 == 0)
117         {
118                 g = -c3 / a3;
119                 h = k3 / a3;
120         }
121         else
122         {
123                 double b31 = b3 / b1;
124                 g = -(b31 * c1 - c3) / (b31 * a1 - a3);
125                 h = (b31 * k1 - k3) / (b31 * a1 - a3);
126         }
127         // Solve the quadratic equation
128         double A = g * g + e * e + 1;
129         double B = -tow1.x * g - tow1.y * e - 2 * tow1.z - tow1.x * g - tow1.y * e + 2
            ↪  * g * h + 2 * e * f;
130         double C = tow1.x * tow1.x + tow1.y * tow1.y + tow1.z * tow1.z - 2 * tow1.x *
            ↪  h - 2 * tow1.y * f + h * h + f * f - tow1.distance * tow1.distance;
131         double rootD = sqrt(B * B - 4 * A * C);
132         double z = (-B + rootD) / (2 * A);
133         double z_ = (-B - rootD) / (2 * A);
134         // Find fittinf x and y for the z solutions
135         double x = g * z + h;
136         double x_ = g * z_ + h;
137         double y = e * z + f;
138         double y_ = e * z_ + f;
139         cout << "raw solution1: " << x << "\t" << y << "\t" << z << endl;
140         cout << "raw solution2: " << x_ << "\t" << y_ << "\t" << z_ << endl;
141         solution.x = (x + x_) / 2.0;
142         solution.y = (y + y_) / 2.0;
143         solution.z = (z + z_) / 2.0;
144         solution.distance = 0.0;
145         /*if (z > 0)
146         {
147         solution.x = x;
148         solution.y = y;
149         solution.z = z;
150         solution.distance = 0.0;
```

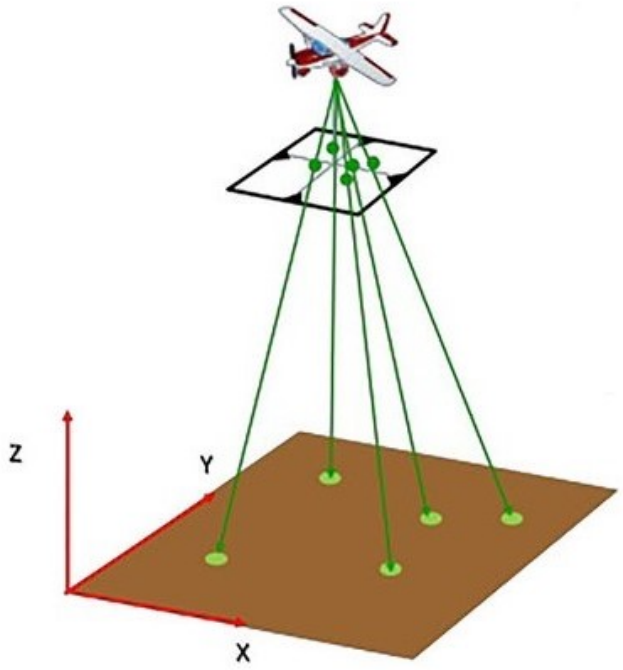
```
151         }
152         else
153         {
154         solution.x = x_;
155         solution.y = y_;
156         solution.z = z_;
157         solution.distance = 0.0;
158         }*/
159         return solution;
160 }
161 int polyfit(const double* const dependentValues,
162         const double* const independentValues,
163         unsigned int countOfElements,
164         unsigned int order,
165         double* coefficients)
166 {
167         // Declarations...
168         // ----------------------------------
169         enum { maxOrder = 5 };
170         double B[maxOrder + 1] = { 0.0f };
171         double P[((maxOrder + 1) * 2) + 1] = { 0.0f };
172         double A[(maxOrder + 1) * 2 * (maxOrder + 1)] = { 0.0f };
173         double x, y, powx;
174         unsigned int ii, jj, kk;
175         // Verify initial conditions....
176         // ----------------------------------
177         // This method requires that the countOfElements >
178         // (order+1)
179         if (countOfElements <= order)
180                 return -1;
181         // This method has imposed an arbitrary bound of
182         // order <= maxOrder. Increase maxOrder if necessary.
183         if (order > maxOrder)
184                 return -1;
185         // Begin Code...
186
187         // ----------------------------------
188         // Identify the column vector
189         for (ii = 0; ii < countOfElements; ii++)
190         {
191                 x = dependentValues[ii];
192                 y = independentValues[ii];
193                 powx = 1;
194                 for (jj = 0; jj < (order + 1); jj++)
195                 {
196                         B[jj] = B[jj] + (y * powx);
197                         powx = powx * x;
198                 }
199         }
200         // Initialize the PowX array
201         P[0] = countOfElements;
202         // Compute the sum of the Powers of X
203         for (ii = 0; ii < countOfElements; ii++)
204         {
205                 x = dependentValues[ii];
206                 powx = dependentValues[ii];
207                 for (jj = 1; jj < ((2 * (order + 1)) + 1); jj++)
208                 {
209                         P[jj] = P[jj] + powx;
210                         powx = powx * x;
```

```
211                 }
212         }
213         // Initialize the reduction matrix
214         //
215         for (ii = 0; ii < (order + 1); ii++)
216         {
217                 for (jj = 0; jj < (order + 1); jj++)
218                 {
219                         A[(ii * (2 * (order + 1))) + jj] = P[ii + jj];
220                 }
221                 A[(ii * (2 * (order + 1))) + (ii + (order + 1))] = 1;
222         }
223         // Move the Identity matrix portion of the redux matrix
224         // to the left side (find the inverse of the left side
225         // of the redux matrix
226         for (ii = 0; ii < (order + 1); ii++)
227         {
228                 x = A[(ii * (2 * (order + 1))) + ii];
229                 if (x != 0)
230                 {
231                         for (kk = 0; kk < (2 * (order + 1)); kk++)
232                         {
233                                 A[(ii * (2 * (order + 1))) + kk] =
234                                         A[(ii * (2 * (order + 1))) + kk] / x;
235                         }
236                         for (jj = 0; jj < (order + 1); jj++)
237                         {
238                                 if ((jj - ii) != 0)
239                                 {
240                                         y = A[(jj * (2 * (order + 1))) + ii];
241                                         for (kk = 0; kk < (2 * (order + 1)); kk++)
242                                         {
243                                                 A[(jj * (2 * (order + 1))) + kk] =
244                                                         A[(jj * (2 * (order + 1))) +
                                                         ↪  kk] -
245                                                         y * A[(ii * (2 * (order + 1)))
                                                         ↪  + kk];
246                                         }
247                                 }
248                         }
249                 }
250                 else
251                 {
252                         // Cannot work with singular matrices
253                         return -1;
254                 }
255         }
256         // Calculate and Identify the coefficients
257         for (ii = 0; ii < (order + 1); ii++)
258         {
259                 for (jj = 0; jj < (order + 1); jj++)
260                 {
261                         x = 0;
262                         for (kk = 0; kk < (order + 1); kk++)
263                         {
264                                 x = x + (A[(ii * (2 * (order + 1))) + (kk + (order +
                                 ↪  1))]
265                                         *
266                                         B[kk]);
267                         }
```

```
268                         coefficients[ii] = x;
269                 }
270         }
271         return 0;
272 }
```

Задача II.1.2. Определение площади зоны чрезвычайной ситуации (15 баллов)

БПЛА совершает фотосъемку подстилающей поверхности в автоматическом режиме. Система технического зрения обнаруживает ориентиры, обозначающие границу зоны чрезвычайной ситуации (ЧС). Для каждого из ориентиров рассчитываются его географические координаты (широта и долгота).



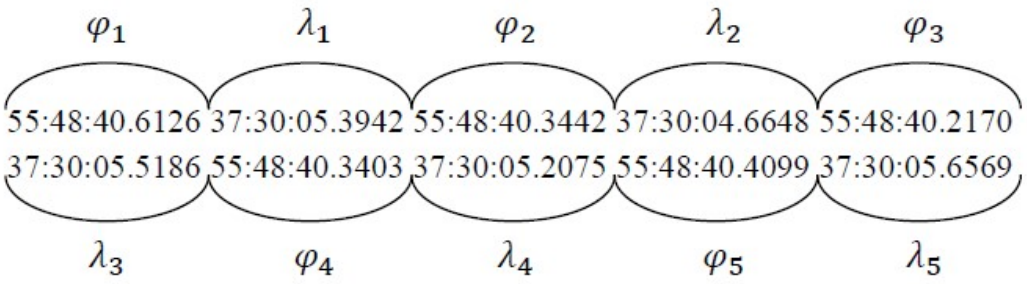
Напишите программу на языке C++, определяющую площадь зоны ЧС (в квадратных метрах, с точностью до 1 знака после запятой), ограниченной произвольным числом ориентиров с заданными географическими координатами. Гарантируется, что заданный набор ориентиров описывает простой многоугольник (стороны многоугольника не пересекаются между собой). При пересчете долготы и широты в метрические координаты использовать коэффициенты пересчета, найденные для геометрического центра (среднее арифметическое значение географических координат) заданного многоугольника. При нахождении этих коэффициентов считать Землю шаром с радиусом 6371 км. Локальная кривизна поверхности Земли в окрестности геометрического центра многоугольника не учитывается.

Формат входных данных

Массив координат (широта φ и долгота λ) ориентиров в формате градусы:минуты:секунды, заданных последовательно, через пробелы. Например, для пяти ориентиров:

DD:MM:SS.SSSS DD:MM:SS.SSSS DD:MM:SS.SSSS DD:MM:SS.SSSS
DD:MM:SS.SSSS DD:MM:SS.SSSS DD:MM:SS.SSSS DD:MM:SS.SSSS
DD:MM:SS.SSSS DD:MM:SS.SSSS

Пример входных данных:



Последовательность соединения ориентиров задается входными данными. Так, ориентир 1 соединяется с ориентиром 2 и с последним ориентиром (в данном примере — ориентир 5). Ориентир 2, соответственно, соединен с ориентирами 1 и 3 и т. д.

Формат выходных данных

Значение площади зоны ЧС в м^2 с точностью до 1 знака после запятой. Ответ должен соответствовать формату: 4 числа, точка, десятые доли значения (например, значение 9 метров записывается как 0009.0).

MMMM.M

Пример результата программы:

0086.7

Список литературы

1. Географические координаты [электронный ресурс], https://v-ipc.ru/guides/coord.

Примеры

Пример №1

Стандартный ввод
55:48:40.6126 37:30:05.3942 55:48:40.3442 37:30:04.6648 55:48:40.2170 37:30:05.5186 55:48:40.3403 37:30:05.2075 55:48:40.4099 37:30:05.6569
Стандартный вывод
0086.7

Пример №2

Стандартный ввод
55:48:40.6999 37:30:06.9613 55:48:39.9945 37:30:06.4939 55:48:39.3496 37:30:05.5508 55:48:39.7418 37:30:04.4591 55:48:40.2756 37:30:04.0208 55:48:40.9803 37:30:03.7630 55:48:41.1855 37:30:04.9548 55:48:40.8626 37:30:06.1035
Стандартный вывод
1937.6

Пример №3

Стандартный ввод
55:48:39.8204 37:30:03.6616 55:48:40.3549 37:30:03.4972 55:48:41.0368 37:30:05.1746 55:48:40.7682 37:30:06.6083 55:48:39.9436 37:30:07.5003
Стандартный вывод
1724.7

Решение

```
#include <iostream>
#include <math.h>
#include <vector>
// Libraries for .clue generation
#include <fstream>
#include <sstream>
#include <iomanip>
using namespace std;
// Sturcture for storing geo coordinates (input format)
struct GEO {
        int degrees;
        int minutes;
        double seconds;
};
// Function prototypes
double solve(double lat[], double lon[], int n);
double mean(double input_array[], int len);
double deg2rad(double degrees);
double bound(double x, double min, double max);
double geo2decimal(GEO coord);
GEO decimal2geo(double degrees);
GEO string2GEO(string input);
// Constant values
const double pi = 3.1415926;
const int R = 6371; // km
// Main function
int main()
{
        cout << "Welcome to emergency zone program.\n\n";
        // Read data from the input file
        string ifilename;
        ifstream ifile;
        stringstream ss; ss << 0;
        vector<string> Lats;
        vector<string> Lons;
```

```
36          ifilename = "input_" + ss.str();
37          cout << "Input file: " << ifilename << endl;
38          string sd;
39          ifile.open(ifilename.c_str(), ios::out);
40          while (ifile >> sd)
41          {
42                  Lats.push_back(sd);
43                  ifile >> sd;
44                  Lons.push_back(sd);
45          }
46          ifile.close();
47          // Parse data to coordinate arrays
48          int num = Lats.size();
49          GEO holder;
50          double lat[num], lon[num];
51          cout << fixed << setprecision(6);
52          for (int i = 0; i < num; i++)
53          {
54                  holder = string2GEO(Lats[i]);
55                  lat[i] = geo2decimal(holder);
56                  //cout << holder.degrees << ":" << holder.minutes << ":" <<
57                  holder.seconds << " = " << lat[i] << endl;
58                  holder = string2GEO(Lons[i]);
59                  lon[i] = geo2decimal(holder);
60                  //cout << holder.degrees << ":" << holder.minutes << ":" <<
61                  holder.seconds << " = " << lon[i] << endl;
62          }
63          // Find the solution
64          double S = solve(lat, lon, num);
65          cout << fixed << setprecision(1);
66          cout << "Actual polygon area equals to " << S << endl << endl;
67          return 0;
68  }
69  // Find the solution for given input
70  double solve(double lat[], double lon[], int n)
71  {
72          // Find mean value of the geo coordinates
73          double lat_avg = mean(lat, n);
74          double lon_avg = mean(lon, n);
75          // Find geo to meters transition coefs
76          double lat_deg_len = 2 * pi * R / 360 * 1000;
77          double lon_deg_len = 2 * pi * R * cos(deg2rad(lat_avg)) / 360 * 1000;
78          double x[n], y[n];
79          // Transform geo coordinates to meters
80          for (int i = 0; i < n; i++)
81          {
82                  x[i] = (lat[i] - lat_avg) * lat_deg_len;
83                  y[i] = (lon[i] - lon_avg) * lon_deg_len;
84          }
85          // Find the polygon area with Gauss area equation
86          double S = 0.0;
87          for (int i = 0; i < (n - 1); i++)
88          {
89                  S += x[i] * y[i + 1] - x[i + 1] * y[i];
90          }
91          S += x[n - 1] * y[0] - x[0] * y[n - 1];
92          S /= 2;
93          S = abs(S);
94          return S;
95  }
```

```
96  // Find average value of the array
97  double mean(double input_array[], int len)
98  {
99          double sum = 0.0;
100         for (int i = 0; i < len; i++)
101                 sum += input_array[i];
102         return (sum / len);
103 }
104 // Transform degrees to radians
105 double deg2rad(double degrees)
106 {
107         return degrees * pi / 180.0;
108 }
109 // Bound the value with given min and max values
110 double bound(double x, double min, double max)
111 {
112         if (min > max)
113                 return x;
114         else if (x < min)
115                 return min;
116         else if (x > max)
117                 return max;
118         else
119                 return x;
120 }
121 // Transform GEO coordinates to decimal equal
122 double geo2decimal(GEO coord)
123 {
124         return (coord.degrees + coord.minutes / 60.0 + coord.seconds / 3600.0);
125 }
126 // Transform decimal coordinate to GEO equal
127 GEO decimal2geo(double degrees)
128 {
129         int deg = (int)degrees;
130         double all = (degrees - deg) * 60.0;
131         int minutes = (int)(all);
132         double seconds = ((all - minutes) * 60.0);
133         GEO coord;
134         coord.degrees = deg;
135         coord.minutes = minutes;
136         coord.seconds = seconds;
137         return coord;
138 }
139 // Parse geo coordinate from a string
140 GEO string2GEO(string input)
141 {
142         vector<string> parts;
143         size_t pos = 0;
144         string delimiter = ":";
145         string token;
146         GEO output;
147         while ((pos = input.find(delimiter)) != std::string::npos) {
148                 token = input.substr(0, pos);
149                 //std::cout << token << std::endl;
150                 parts.push_back(token);
151                 input.erase(0, pos + delimiter.length());
152         }
153         //std::cout << input << std::endl;
154         parts.push_back(input);
155         output.degrees = stoi(parts[0]);
```

```
156         output.minutes = stoi(parts[1]);
157         output.seconds = stod(parts[2]);
158         return output;
159 }
```

Задача II.1.3. Разбор данных GPS приемника (15 баллов)

В процессе полета БПЛА в вычислитель навигационного комплекса поступают данные с GPS приемника в виде потока байтов, содержащих два вида сообщений: GNGGA протокола NMEA 0183 и NAV-PVT протокола UBX (приемника u-blox 7). Напишите программу на языке C++, которая будет определять количество сообщений каждого вида в массиве байт и разбирать находящиеся в массиве байт сообщения. В массиве байт гарантированно находится по крайней мере одно сообщение каждого вида, все сообщения в массиве являются цельными и расположены в неизвестном порядке.

Сообщения имеют следующую структуру: сообщение GNGGA — строковое сообщение, начинающееся с символа «$» и заканчивающееся невидимыми символами окончания строки и возврата каретки «\n\r».

Структура сообщения

1	$GNGGA	Наименование сообщения
2	hhmmss.ss	Гринвичское время на момент определения местоположения
3	xxxx.xx	Географическая широта местоположения
4	a	Север/Юг
5	yyyy.yy	Географическая долгота местоположения
6	a	Запад/Восток
7	x	Статус GPS приемника
8	xx	Количество используемых спутников
9	x.xx	Горизонтальный геометрический фактор ухудшения точности (HDOP)
10	xx.x	Высота над уровнем моря
11	m	Единицы измерения высоты
12	xx.x	Разница между эллипсоидом Земли и уровнем моря
13	m	Единицы измерения разницы
14	xx	Количество секунд, прошедших с момента получения последней DGPS поправки
15	*4F	ID базовой станции предоставляющей DPGS поправки
16	«\n\r»	Окончание сообщения

Пример сообщений:

"$GNGGA,112904.00,5548.67749,N,03730.05644,E,1,12,0.62,167.0,M,13.4,M,,*4F

$GNGGA,112904.00,5548.67749,N,03730.05644,E,1,12,0.62,167.0,M,13.4,M,,*4F"

Сообщение NAV-PVT — байтовое сообщение, использующее следующие типы данных:

Сокр.	Тип	Байт	Комментарии	Min..Max	Точность	Сокр.
U1	Unsigned Char	1		0..255	1	U1
I1	Signed Char	1	Формат с дополнением до 2	-128..127	1	I1

X1	Bitfield	1		нет	нет	X1
U2	Unsigned Short	2		0..65535	1	U2
I2	Signed Short	2	Формат с дополнением до 2	-32768..32767	1	I2
X2	Bitfield	2		нет	нет	X2
U4	Unsigned Long	4		0..4'294'967'295	1	U4
I4	Signed Long	4	Формат с дополнением до 2	-2'147'483'648 .. 2'147'483'647	1	I4
X4	Bitfield	4		нет		X4
R4	IEEE 754 Single Precision	4		$-2^{+127}..2^{+127}$	2^{-24}	R4
R8	IEEE 754 Double Precision	8		$-2^{+1023}..2^{+1023}$	2^{-53}	R8
CH	ASCII / ISO 8859.1 Encoding	1				CH

Структура сообщения

Заголовок (2 байта)	ID (2 байта)	Количество байт в полезной нагрузке (1 байт)	Полезная нагрузка	Контр. сумма (2 байта)
0xB5 0x62	0x01 0x07	84	См. ниже	CK_A CK_B

Содержимое полезной нагрузки:

Смещ. байт	Формат числа	Масштаб	Имя	Ед.	Описание
0	U4	-	iTOW	мс	Время прошедшее с момента начала новой недели GPS
4	U2	-	year	-	Год (UTC)
6	U1	-	month	-	Месяц, диапазон 1..12 (UTC)
7	U1	-	day	-	День месяца, диапазон 1..31 (UTC)
8	U1	-	hour	-	Час дня, диапазон 0..23 (UTC)
9	U1	-	min	-	Минута часа, диапазон 0..59 (UTC)
10	U1	-	sec	-	Секунда в минуте, диапазон 0..60 (UTC)
11	X1	-	valid	-	Флаги достоверности (см. рисунок ниже)
12	U4	-	tAcc	нс	Оценка точности времени (UTC)
16	L4	-	nano	нс	Дробная часть секунды, диапазон -1e9 .. 1e9 (UTC)
20	U1	-	fixType	-	Тип фиксации позиции GNSS, диапазон 0..5: 0x00 No Fix (нет фиксации) 0x01 только Dead Reckoning (точный расчет) 0x02 2D-Fix 0x03 3D-Fix 0x04 комбинация GNSS + dead reckoning 0x05 только фиксация времени 0x06..0xff: зарезервировано
21	X1	-	flags	-	
22	U1	-	reserved1	-	Зарезервировано

Смещ. байт	Формат числа	Масштаб	Имя	Ед.	Описание
23	U1	-	numSV	-	Количество спутников, использовавшихся в определении навигации.
24	L4	1e-7	lon	град	Долгота
28	L4	1e-7	lat	град	Широта
32	L4	-	height	мм	Высота над эллипсоидом
36	L4	-	hMSL	мм	Высота над уровнем моря
40	U4	-	hAcc	мм	Оценка горизонтальной точности
44	U4	-	vAcc	мм	Оценка вертикальной точности
48	L4	-	velN	мм/сек	Скорость в северном направлении (NED)
52	L4	-	velE	мм/сек	Скорость в восточном направлении (NED)
56	L4	-	velD	мм/сек	Скорость в вертикальном направлении (NED)
60	L4	-	gSpeed	мм/сек	Скорость по земле (2-D)
64	L4	1e-5	heading	град	Направление движения (2-D)
68	U4	-	sAcc	мм/сек	Оценка точности скорости
72	U4	1e-5	headingAcc	град	Оценка точности направления
76	U2	0.01	pDOP	-	DOP позиции
78	X2	-	reserved2	-	Зарезервировано
80	U4	-	reserved3	-	Зарезервировано

Для решения задачи нет необходимости разбирать все поля имеющихся сообщений. Достаточно получить необходимые данные и вывести их в правильном виде.

Формат входных данных

Строка, содержащая шестнадцатеричное представление массива байт, содержащего сообщения GNGGA и NAV-PVT.

Примечание: в шаблоне решения есть функция позволяющая преобразовать строку в бинарные данные.

Формат выходных данных

Первая строка — количество сообщений GNGGA и NAV-PVT через запятую.

Вторая строка — время полученное из последнего сообщения GNGGA в формате HH:MM:SS.

Третья строка — долгота и широта полученные из последнего сообщения NAV-PVT, через запятую, с точностью до 6го знака после запятой.

Пример результата программы:

6, 4 11:46:13 40.858486, 60.846586

Примечание: результаты разбора бинарных пакетов могут не значительно отличаться от приведенных, ответ в этом случае будет так же засчитан как верный

Список литературы:

1. Описание протокола NMEA https://wiki.iarduino.ru/page/NMEA-0183/ (Дата обращения 15.10.2020).
2. Описание протокола UBX http://microsin.net/adminstuff/others/u-blox-7-ubx-protocol-part-3.html (Дата обращения 15.10.2020).
3. Полезная информация по C++ https://docs.microsoft.com/ru-ru/cpp/cpp/unions?view=vs-2019 (Дата обращения 15.10.2020).

Примеры

Пример №1

Стандартный ввод

```
b5 62 01 07 54 00 00 00 4e 00 00 00 e2 07 02 16 0d 23 12 07 1b 00 00 00 91 41
00 00 02 01 00 04 cf 66 4b 14 75 9d 0a 24 57 5d 00 00 97 0b fe ff 18 0d 00 00
9d 02 00 00 7a f6 ff ff 36 0b 00 00 f7 03 00 00 b6 0e 00 00 11 fe c6 00 a6 00
00 00 e2 4c 06 00 2b 00 00 00 00 00 00 00 24 47 4e 47 47 41 2c 31 30 35 33 31 34
2e 30 30 2c 36 30 37 34 2e 31 37 35 32 39 2c 4e 2c 34 33 30 36 2e 31 39 31 38
39 2c 45 2c 38 2c 31 30 2c 32 2e 32 38 2c 34 35 32 2e 38 33 2c 4d 2c 33 30 30
2e 38 33 2c 4d 2c 2a 34 46 0a 0d 24 47 4e 47 47 41 2c 31 31 34 34 31 36 2e 30
30 2c 35 31 31 32 2e 37 35 30 34 39 2c 4e 2c 33 31 33 39 2e 36 35 32 35 39 2c
45 2c 38 2c 37 2c 32 2e 38 37 2c 33 38 37 2e 38 39 2c 4d 2c 32 33 35 2e 38 39
2c 4d 2c 2a 34 46 0a 0d b5 62 01 07 54 00 00 00 46 00 00 00 df 07 04 0f 12 1f
2c 07 7e 00 00 00 a7 18 00 00 02 01 00 04 c3 0f 9e 19 fe a1 2c 24 02 32 00 00
42 e0 fd ff 57 21 00 00 b0 07 00 00 1b 05 00 00 1c 00 00 00 4f 04 00 00 1c 05
00 00 f5 e4 01 00 68 02 00 00 b1 2c 09 00 90 00 00 00 00 00 00 00 b5 62 01 07 54
00 00 00 23 00 00 00 e3 07 05 06 08 22 3b 07 0d 00 00 00 c5 37 00 00 02 01 00
04 1e ef 7c 1a c2 ca 6e 26 c6 49 00 00 06 f8 fd ff 54 16 00 00 ff 07 00 00 4f
03 00 00 5d 04 00 00 a5 01 00 00 7b 05 00 00 e9 a2 50 00 1e 05 00 00 76 66 07
00 4f 01 00 00 00 00 00 b5 62 01 07 54 00 00 00 2a 00 00 00 df 07 02 06 15 32
2c 07 1f 00 00 00 24 0c 00 00 02 01 00 11 9d 82 5a 18 b3 73 44 24 65 1a 00 00
a5 c8 fd ff ea 1c 00 00 37 01 00 00 32 fb ff ff 2b 0c 00 00 17 01 00 00 16 0d
00 00 bc 4a aa 00 f0 00 00 00 b1 fa 03 00 b4 01 00 00 00 00 00 24 47 4e 47 47
41 2c 31 38 32 36 33 30 2e 30 30 2c 36 31 39 30 2e 34 30 34 37 39 2c 4e 2c 34
34 39 31 2e 35 36 31 35 32 2c 45 2c 38 2c 38 2c 35 2e 38 36 2c 31 35 39 38 2e
33 32 2c 4d 2c 31 34 34 36 2e 33 32 2c 4d 2c 2a 34 46 0a 0d 24 47 4e 47 47 41
2c 30 35 34 30 32 36 2e 30 30 2c 35 34 32 38 2e 35 37 31 32 39 2c 4e 2c 34 30
32 37 2e 35 35 38 31 31 2c 45 2c 38 2c 34 2c 32 2e 31 34 2c 31 39 35 37 2e 38
38 2c 4d 2c 31 38 30 35 2e 38 38 2c 4d 2c 2a 34 46 0a 0d 24 47 4e 47 47 41 2c
31 38 33 36 34 31 2e 30 30 2c 36 33 33 35 2e 34 32 39 32 30 2c 4e 2c 34 35 37
34 2e 37 35 34 38 38 2c 45 2c 38 2c 39 2c 33 2e 39 39 2c 31 36 31 37 2e 38 35
2c 4d 2c 31 34 36 35 2e 38 35 2c 4d 2c 2a 34 46 0a 0d 24 47 4e 47 47 41 2c 31
31 34 36 31 33 2e 30 30 2c 35 39 38 31 2e 33 33 37 38 39 2c 4e 2c 33 37 33 36
2e 36 35 35 37 36 2c 45 2c 38 2c 31 37 2c 32 2e 34 31 2c 31 33 38 37 2e 31 39
2c 4d 2c 31 32 33 35 2e 31 39 2c 4d 2c 2a 34 46 0a 0d
```

Стандартный вывод

```
6, 4
11:46:13
40.858486, 60.846586
```

Решение

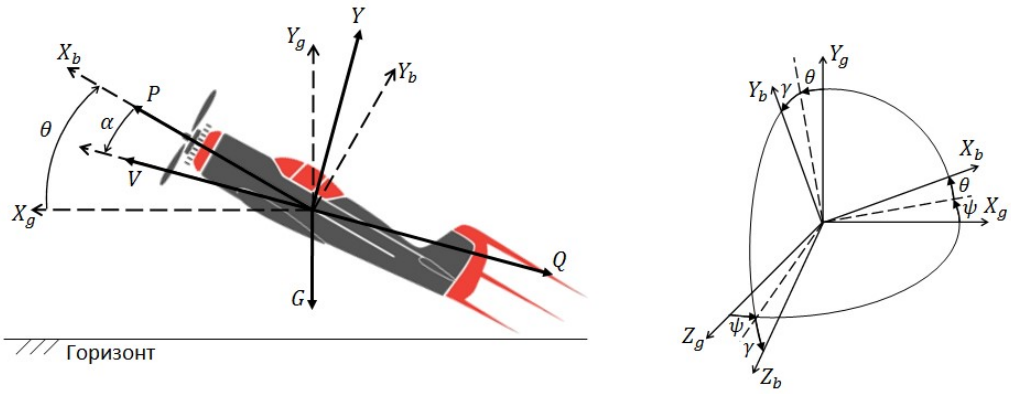
```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <cstdint>
#include <string>
#include <sstream>
#include <iomanip>
#include <algorithm>
using namespace std;
#pragma pack()
typedef struct {
        uint8_t h1;
        uint8_t h2;
        uint8_t id1;
        uint8_t id2;
        uint8_t len;
        uint32_t iTOW;
        uint16_t year;
        uint8_t month;
        uint8_t day;
        uint8_t hour;
        uint8_t min;
        uint8_t sec;
        uint8_t valid1;
        uint32_t tAcc;
        int32_t nano;
        uint8_t fixType;
        uint8_t flags;
        uint8_t reserved1;
        uint8_t numSV;
        int32_t lon;
        int32_t lat;
        int32_t height;
        int32_t hMSL;
        uint32_t hAcc;
        uint32_t vAcc;
        int32_t velN;
        int32_t velE;
        int32_t velD;
        int32_t gSpeed;
        int32_t heading;
        uint32_t sAcc;
        uint32_t headingAcc;
        uint16_t pDOP;
        uint16_t reserved2;
        uint32_t reserved3;
        uint8_t ckA;
        uint8_t ckB;
} _pvt_nav_pack;
typedef union {
        _pvt_nav_pack buffer;
        uint8_t bytes[sizeof(_pvt_nav_pack)];
} pvt_nav_union;
string convert_to_binary(const string& s)
{
```

```
59          char delimiter = ' ';
60          string token;
61          istringstream tokenStream(s);
62          int i = count(s.begin(), s.end(), ' ');
63          stringstream ss;
64          while (getline(tokenStream, token, delimiter))
65          {
66                  uint8_t byte = stoi(token, 0, 16);
67                  ss << byte;
68          }
69          return ss.str();
70  }
71  string parse_GGA_string(string GGA_string) {
72          string delimiter = ",";
73          size_t pos = 0;
74          string time;
75          stringstream ss;
76          for (int i = 0; i < 2; i++) {
77                  pos = GGA_string.find(delimiter);
78                  time = GGA_string.substr(0, pos);
79                  GGA_string.erase(0, pos + delimiter.length());
80          }
81          int hours = stoi(time.substr(0, 2));
82          int mins = stoi(time.substr(2, 2));
83          int seconds = stoi(time.substr(4, 2));
84
85          ss << setfill('0') << setw(2) << hours << ":" << setfill('0') << setw(2) <<
            ↪  mins << ":" << s
86                  etfill('0') << setw(2) << seconds << endl;
87          return ss.str();
88  }
89  string parse_UBX_string(string UBX_string) {
90          pvt_nav_union ubx_pack;
91          stringstream ss;
92          memcpy(ubx_pack.bytes, UBX_string.c_str(), sizeof(_pvt_nav_pack));
93          ss << fixed << setprecision(6) << float(ubx_pack.buffer.lon) / 10000000.0 <<
            ↪  ", "<< float(ubx_pack.buffer.lat)/10000000.0<<"\n";
94                  return ss.str();
95  }
96  int main(int argc, char** argv)
97  {
98          int gga_n = 0;
99          int ubx_n = 0;
100         char* UBX_pack = new char[91];
101         char* GGA_pack = new char[76];
102         ifstream task_file("task1.txt");
103         filebuf* pbuf = task_file.rdbuf();
104         size_t size = pbuf->pubseekoff(0, task_file.end, task_file.in);
105         string third_string, second_string, first_string, file_string;
106         pbuf->pubseekpos(0, task_file.in);
107         char* buffer = new char[size];
108         pbuf->sgetn(buffer, size);
109         task_file.close();
110         file_string = string(buffer, sizeof(char) * size);
111         string data = convert_to_binary(file_string);
112         const char* binary_data = data.c_str();
113         for (int i = 0; i < size; i++) {
114                 uint8_t newbyte = binary_data[i];
115                 if (newbyte == 0xB5) {
116                         uint8_t secondbyte = binary_data[i + 1];
```

```
                        if (secondbyte == 0x62) {
                                memcpy(UBX_pack, binary_data + i,
                                ↪  sizeof(_pvt_nav_pack));
                                third_string =
                                        parse_UBX_string(string(UBX_pack,
                                        ↪  sizeof(_pvt_nav_pack)));
                                ubx_n++;
                        }
                }
                if (binary_data[i] == '$' && binary_data[i + 1] == 'G' &&
                        binary_data[i + 1] == 'G') {
                        memcpy(GGA_pack, binary_data + i, sizeof(char) * 76);
                        second_string = parse_GGA_string(string(GGA_pack, sizeof(char)
                        ↪  * 76));
                        gga_n++;
                }
        }
        cout << gga_n << ", " << ubx_n << endl;
        cout << second_string;
        cout << third_string;
        return 0;
}
```

Задача II.1.4. Определение путевой скорости движения ЛА (15 баллов)

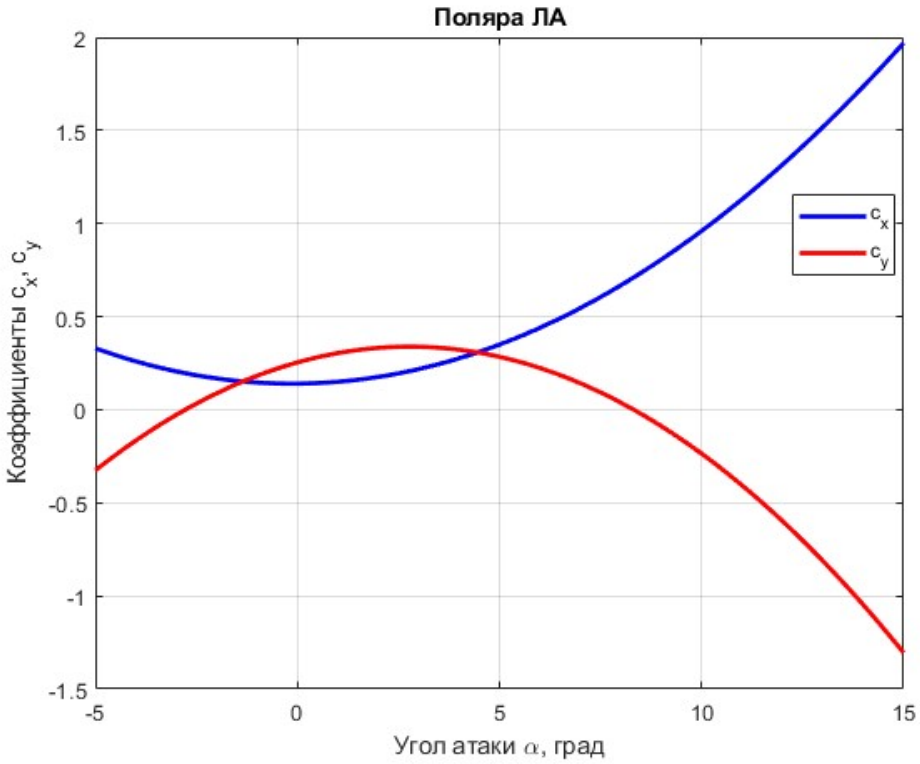
Летательный аппарат (ЛА) массой 8 кг совершает координированный разворот (угол скольжения равен нулю) с постоянной скоростью.



Сила лобового сопротивления $Q = c_x(\alpha)\frac{\rho V^2}{2}S$ направлена противоположно путевой скорости V ЛА, подъемная сила $Y = c_y(\alpha)\frac{\rho V^2}{2}S$— перпендикулярно путевой скорости ЛА, тяга P направлена по продольной оси ЛА, а сила тяжести G — вертикально вниз. Оси связанной системы координат (СК) $OX_bY_bZ_b$ повернуты относительно осей географической СК $OX_gY_gZ_g$ (ось OX_g направлена на север, ось OZ_g — на восток) на углы курса ψ, тангажа θ и крена γ. Угол атаки α — угол между вектором путевой скорости V и продольной осью ЛА OX_b. Плотность воздуха $\rho = 1,225$кг/м^3, площадь крыльев ЛА $S = 0,56$ м^2. Поляра ЛА аппроксимирована следующими полиномами (указанные коэффициенты подразумевают задание угла атаки в градусах):

$$c_x(\alpha) = 0,008 \cdot \alpha^2 + 0,002 \cdot \alpha + 0,14$$

$$c_y(\alpha) = -0,011 \cdot \alpha^2 + 0,061 \cdot \alpha + 0,254$$



Напишите программу на языке C++, определяющую величины проекций вектора путевой скорости V на оси географической $OX_gY_gZ_g$ при заданных углах ориентации ψ,θ,γ и угле атаки α. Считать, что ускорение свободного падения $g = 9,81$ м/с^2. Гарантируется, что задача имеет решение. Обратите внимание на расположение осей — ось OY_g направлена вертикально вверх.

Формат входных данных

Угол курса (ψ, YY), угол тангажа (θ, PP), угол крена (γ, RR) и угол атаки (α, AA), заданные в градусах, через пробел.

YY PP RR AA

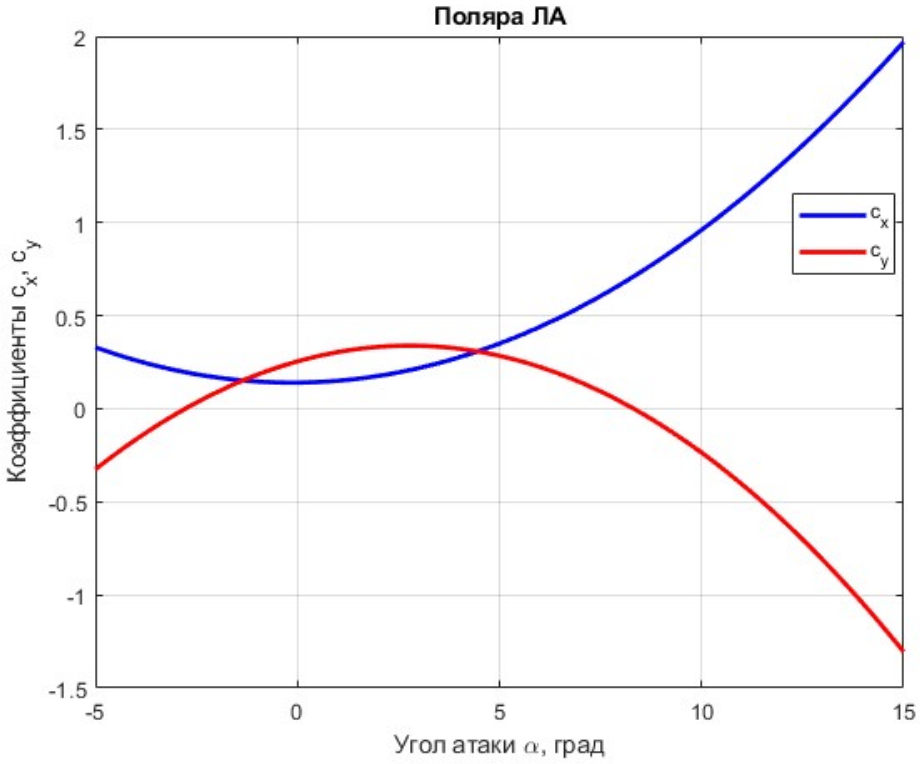
Пример входной строки:

Формат выходных данных

Значения проекций вектора путевой скорости V на оси географической СК: $V_{x_g}, V_{y_g}, V_{z_g}$ в м/с, записанные через пробел с точностью до 2 знака после запятой. Результат программы должен соответствовать формату: три символа, точка, два числа (например, значение 4.2 записывается как 004.20, а значение -10 записывается как -10.00).

XXX.XX YYY.YY ZZZ.ZZ

Пример результата программы:



020.90 002.69 -14.75

Список литературы

1. Матрица поворота [электронный ресурс], википедия https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%D0%BF%D0%BE%D0%B2%D0%BE%D1%80%D0%BE%D1%82%D0%B0.
2. Остославский И.В. Аэродинамика самолета. — М.: Государственное Издательство Оборонной Промышленности, 1957.

Примеры

Пример №1

Стандартный ввод
`35 10 03 04`

Стандартный вывод
`020.90 002.69 -14.75`

Пример №2

Стандартный ввод
52 11 07 08

Стандартный вывод
024.94 002.21 -33.06

Решение

```
#include <iostream>
#include <math.h>
// Libraries for .clue generation
#include <fstream>
#include <sstream>
#include <iomanip>
using namespace std;
// Function prototypes
void solve(double yaw, double pitch, double roll, double aoa, double* arr);
double deg2rad(double degrees);
void rotateVector(double* vector, double matrix[3][3]);
// Constant values
const double ro = 1.225;
const double S = 0.56;
const double m = 8.0;
const double g = 9.81;
const double pi = 3.1415926;
// Main function
int main()
{
        cout << "Welcome to the velocity calculation program.\n\n";
        // Read data from the input file
        string ifilename;
        ifstream ifile;
        ifilename = "input_4";
        double yaw, pitch, roll, aoa;
        string sd;
        ifile.open(ifilename.c_str(), ios::out);
        ifile >> sd;
        yaw = stod(sd);
        ifile >> sd;
        pitch = stod(sd);
        ifile >> sd;
        roll = stod(sd);
        ifile >> sd;
        aoa = stod(sd);
        ifile.close();
        // Find the solution
        double sol[3];
        solve(yaw, pitch, roll, aoa, sol);
        cout << fixed << setprecision(2) << setfill('0');
        cout << "Solution: " << setw(6) << sol[0] << " ";
        cout << setw(6) << sol[1] << " ";
        cout << setw(6) << sol[2] << endl;

        return 0;
}
```

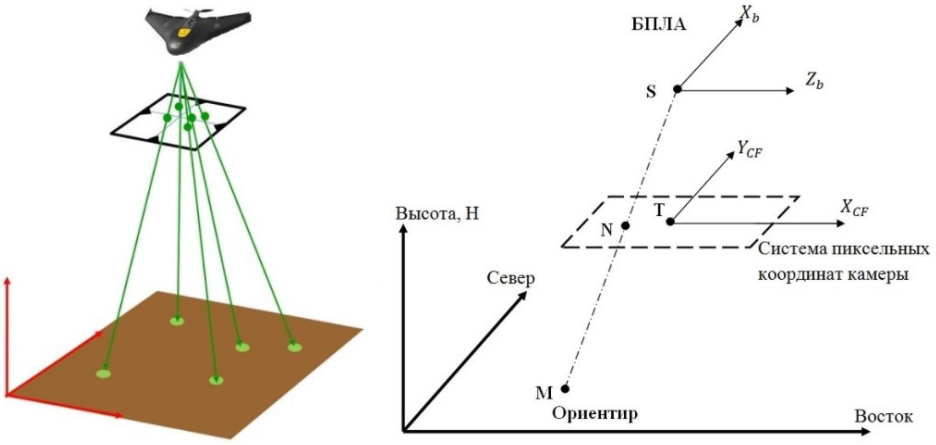
```
48  // Find the solution for given input
49  void solve(double yaw, double pitch, double roll, double aoa, double* arr)
50  {
51          // Find aerodynamic coefs
52          double cx = 0.008 * aoa * aoa + 0.002 * aoa + 0.14;
53          double cy = -0.011 * aoa * aoa + 0.061 * aoa + 0.254;
54          aoa = deg2rad(aoa);
55          yaw = deg2rad(yaw);
56          pitch = deg2rad(pitch);
57          roll = deg2rad(roll);
58          // Find the total thrust value, by solving two equations
59          double P = m * g * (sin(pitch - aoa) + cx / cy * cos(pitch - aoa));
60          P = P / (cos(aoa) + cx / cy * sin(aoa));
61          // Solve quadratic equation to find total velocity value
62          double A = -cx * ro * S / 2.0;
63          double C = P * cos(aoa) - m * g * sin(pitch - aoa);
64          double sqD = sqrt(-4.0 * A * C);
65          double V = abs(sqD / 2.0 / A);
66          //cout << "V: " << V << endl;
67          // Base frame to geo frame transition matrix
68          double Rbg[3][3];
69          Rbg[0][0] = cos(yaw) * cos(pitch);
70          Rbg[0][1] = -cos(yaw) * sin(pitch) * cos(roll) + sin(yaw) * sin(roll);
71          Rbg[0][2] = cos(yaw) * sin(pitch) * sin(roll) + sin(yaw) * cos(roll);
72          Rbg[1][0] = sin(pitch);
73          Rbg[1][1] = cos(pitch) * cos(roll);
74          Rbg[1][2] = -cos(pitch) * sin(roll);
75          Rbg[2][0] = -sin(yaw) * cos(pitch);
76          Rbg[2][1] = cos(yaw) * sin(roll) + sin(yaw) * sin(pitch) * cos(roll);
77          Rbg[2][2] = cos(yaw) * cos(roll) - sin(yaw) * sin(pitch) * sin(roll);
78          // Speed frame to base frame transition matrix
79          double Rsb[3][3];
80          Rsb[0][0] = cos(aoa);
81          Rsb[0][1] = sin(aoa);
82          Rsb[0][2] = 0;
83          Rsb[1][0] = -sin(aoa);
84          Rsb[1][1] = cos(aoa);
85          Rsb[1][2] = 0;
86          Rsb[2][0] = 0;
87          Rsb[2][1] = 0;
88          Rsb[2][2] = 1.0;
89          double res[3];
90          res[0] = V; res[1] = 0.0; res[2] = 0.0;
91          rotateVector(res, Rsb);
92          //cout << "in base: " << res[0] << " " << res[1] << " " << res[2] <<
93          endl;
94          rotateVector(res, Rbg);
95          //cout << "in geo: " << res[0] << " " << res[1] << " " << res[2] <<
96          endl;
97          arr[0] = res[0];
98          arr[1] = res[1];
99          arr[2] = res[2];
100         return;
101 }
102 // Transform degrees to radians
103 double deg2rad(double degrees)
104 {
105         return degrees * pi / 180.0;
106 }
107 void rotateVector(double* vector, double matrix[3][3])
```

```
{
        double result[3];
        result[0] = matrix[0][0] * vector[0] + matrix[0][1] * vector[1] + matrix[0][2]
        ↪  * vector[2];
        result[1] = matrix[1][0] * vector[0] + matrix[1][1] * vector[1] + matrix[1][2]
        ↪  * vector[2];
        result[2] = matrix[2][0] * vector[0] + matrix[2][1] * vector[1] + matrix[2][2]
        ↪  * vector[2];
        vector[0] = result[0];
        vector[1] = result[1];
        vector[2] = result[2];
        return;
}
```

Командная часть

Задача II.2.1. Поиск зоны чрезвычайной ситуации (20 баллов)

БПЛА совершает полет в автоматическом режиме и осуществляет фотосъемку подстилающей поверхности. Задачей системы технического зрения является обнаружение ориентиров, обозначающих границу зоны чрезвычайной ситуации (ЧС).



Камера жестко закреплена на корпусе БПЛА и ее оптическая ось направлена вертикально вниз. Камера делает фотоснимки подстилающей поверхности в формате .pgm (Portable Grey Map) с разрешением 100×100 пикселей. Напишите программу на языке C++ или Python, которая определит географические координаты местоположения контрастных ориентиров (точка M, число ориентиров может быть произвольным), присутствующих на снимке. При этом известны:

- текущие географические координаты БПЛА (координаты точки S): широта $\varphi = 56,095618$, долгота $\lambda = 35,880042$ град, высота полета $H = 120$ метров;
- текущие углы ориентации БПЛА (курс, тангаж, крен) $\psi = \theta = \gamma = 0$ град. БПЛА летит строго на север;
- размер одного пикселя на изображении, полученном с фотокамеры $s_x = s_y = 22$ мкм;
- фокусное расстояние камеры $F = 3,8$ мм (расстояние от точки S до центра изображения T в системе пиксельных координат камеры).

При расчетах считать, что подстилающая поверхность ровная, и ее высота равна 0 м. При пересчете метрических координат в долготу и широту использовать коэффициенты пересчета, найденные для текущего местоположения БПЛА. При нахождении этих коэффициентов считать Землю шаром с радиусом $R = 6371$ км.

Примечание: текущие координаты БПЛА (точки S) соответствуют координатам точке в центре кадра (пиксельные координаты 50, 50). Обратите внимание, что пиксельные координаты отсчитываются от левого верхнего угла.

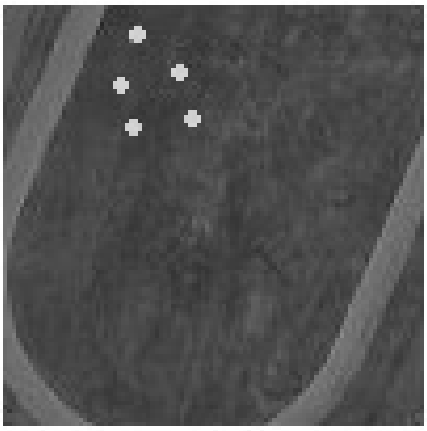
Примечание: пороговое значения для фильтрации пикселей изображения одинаковое для всех тестов. Проведите нормализацию яркости изображения, после этого вы сможете отфильтровать ориентиры оставив только те пиксели, яркость которых превышает 200 (или 195). При этом у вас все еще могут остаться «лишние» пиксели — для того что убрать их тоже — введите ограничение по количеству пикселей в ориентире, например если количество соседних пикселей в ориентире меньше 10 (или 8) — это шум изображения и его можно не учитывать.

Формат входных данных

Исходными данными является снимок подстилающей поверхности в формате .pgm размером 100×100 пикселей. Гарантируется, что на каждом снимке есть как минимум один контрастный ориентир. Контрастность (отношение яркости ориентира к яркости фона изображения) каждого снимка составляет как минимум 5:1. На платформе stepik содержимое .pgm файла задается в виде строки через консоль (формат строки повторяет формат .pgm файла).

Дополнительные примеры исходных файлов: https://drive.google.com/drive/folders/1b9b9ATEvLXTUtAaTMcwQ2LofyoYVMsdg.

Пример исходного файла:



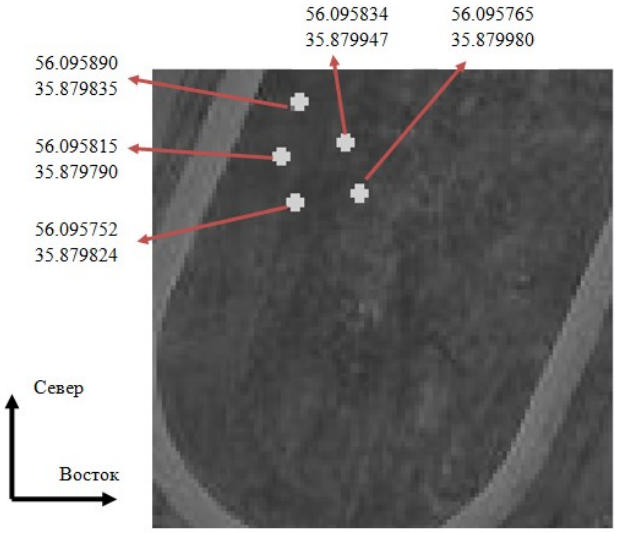
Результатом выполнения программы должны быть значения географических координат (широты и долготы) всех найденных ориентиров. Так как каждому ориентиру соответствует несколько пикселей (не меньше 5) на исходном изображении, при расчете координат местоположения ориентиров используется среднее арифметическое значение. Широта и долгота каждого ориентира записываются в градусах через пробел, в виде десятичной дроби с точностью до 6 знака после запятой. Координаты каждого ориентира записываются отдельной строкой. Порядок записи ориентиров —

в порядке возрастания широты. Гарантируется, что широта местоположения каждого из ориентиров на снимке различается.

Пример результата программы:

56.095752 35.879824 56.095765 35.879980 56.095815 35.879790 56.095834 35.879947 56.095890 35.879835

Графическое отображение результатов (не требуется от участников) имеет вид:



Список литературы

1. Описание формата файлов PGM [электронный ресурс], http://netpbm.sourceforge.net/doc/pgm.html.
2. Дополнительные примеры исходных файлов [электронный ресурс], https://drive.google.com/drive/folders/1b9b9ATEvLXTUtAaTMcwQ2LofyoYVMsdg.
3. Географические координаты [электронный ресурс], https://ru.wikipedia.org/wiki/Географические_координаты.

Решение

```
#include <iostream>
#include <math.h>
#include <vector>
#include <string>
// Libraries for .clue generation
#include <fstream>
#include <sstream>
#include <iomanip>
#include <bits/stdc++.h>
using namespace std;
struct Pixel {
  int row;
  int col;
};
// Function prototypes
double deg2rad(double degrees);
// Constant values
const double pi = 3.1415926;
```

```
const int R = 6371; // km
// Main function
int main()
{
  // Read data from the input file
  string sd, ifilename = "camera_shot_0.pgm";
  ifstream ifile;
  ifile.open(ifilename.c_str(), ios::out);
  ifile >> sd;
  while (sd.compare("100") != 0)
  {
    ifile >> sd;
  }
  int width = stoi(sd);
  ifile >> sd;
  int height = stoi(sd);
  ifile >> sd;
  int image[height][width];
  int max_value = 0;
  for (int i = 0; i < height; i++)
  {
    for (int j = 0; j < width; j++)
    {
      ifile >> sd;
      image[i][j] = stoi(sd);
      if (image[i][j] > max_value)
        max_value = image[i][j];
    }
  }
  ifile.close();
  // Normalise the contrast of the image
  for (int i = 0; i < height; i++)
  {
    for (int j = 0; j < width; j++)
    {
      image[i][j] = (int)((double)image[i][j] / max_value * 255.0);
    }
  }
  // Create two-dim dynamic array
  vector<vector<Pixel>> orientiers;
  bool pointFound = false;
  bool horConnect = false;
  bool verConnect = false;
  // Sort the image into orientiers array
  for (int i = 0; i < height; i++)
  {
    for (int j = 0; j < width; j++)
    {
      if (image[i][j] > 200)
      {
        pointFound = false;
        horConnect = false;
        verConnect = false;
        if (orientiers.size() > 0)
        {
          for (int k = 0; k < orientiers.size(); k++)
          {
            horConnect = false;
            verConnect = false;
            for (int p = 0; p < orientiers[k].size(); p++)
```

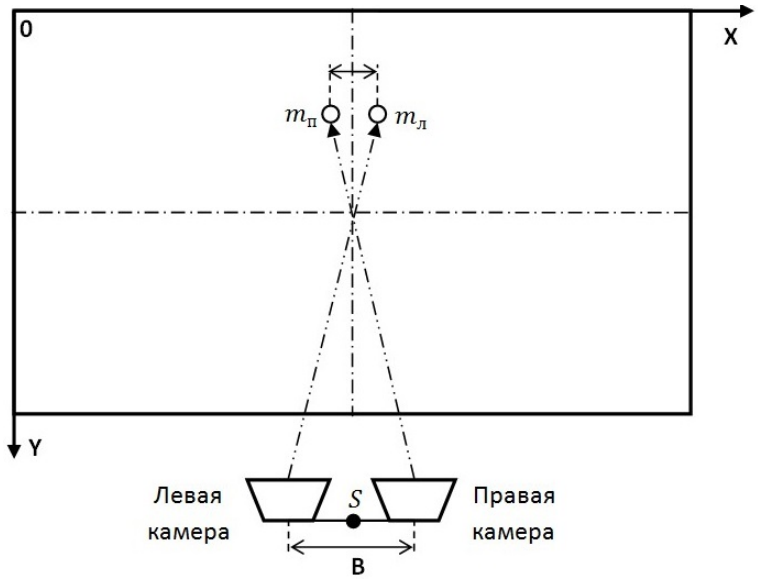
```
79              {
80                if ((abs(i - orientiers[k][p].row) <= 1) &&
81                  (abs(j - orientiers[k][p].col) <= 1))
82                {
83                  pointFound = true;
84                  Pixel dot; dot.row = i; dot.col = j;
85                  orientiers[k].push_back(dot);
86                  //cout << "Point " << i << " " << j << " attached to " << k << endl;
87                  break;
88                }
89              }
90            }
91            if (!pointFound)
92            {
93              Pixel dot; dot.row = i; dot.col = j;
94              vector<Pixel> vect;
95              vect.push_back(dot);
96              orientiers.push_back(vect);
97              pointFound = true;
98            }
99          }
100         else
101         {
102           Pixel dot; dot.row = i; dot.col = j;
103           vector<Pixel> vect;
104           vect.push_back(dot);
105           orientiers.push_back(vect);
106           //cout << "Created: " << 0 << " with a point " << i << " " << j
107           << endl;
108         }
109       }
110     }
111   }
112   //cout << "\nResults:\n";
113   //cout << "Found " << orientiers.size() << " orientiers:" << endl;
114   //for (int i = 0; i < orientiers.size(); i++) {
115   // cout << "Orientier " << i << " contains " << orientiers[i].size() << " points" <<
      endl;
116   //}
117   // Transform generated polygon to geo coordinates
118   // Center of the poligon will be bound to these coordinates
119   double mai_lat = 56.095618, mai_lon = 35.880042;
120   // Calculate degrees->meter transition coefs
121   double lat_deg_len = 2 * pi * R / 360 * 1000;
122   double lon_deg_len = 2 * pi * R * cos(deg2rad(mai_lat)) / 360 * 1000;
123   double min_lat = 99;
124   vector<double> Lats;
125   vector<double> Lons;
126   vector<double> sorter;
127   double F = 3.8 * 1e-3;
128   double H = 120.0 - F;
129   double P = 22 * 1e-6;
130   int count = 0;
131   //cout << P * 50 * H / F << endl;
132   //cout << "\nLets leave only the valid ones:" << endl;
133   // Calculate the coordinates for valid orientiers
134   for (int i = 0; i < orientiers.size(); i++)
135   {
136     if (orientiers[i].size() >= 10)
137     {
```

```
138         count++;
139         int num = orientiers[i].size();
140         //cout << "Orientier " << i << " contains " << num << " points" <<
141         endl;
142         double avgx = 0.0;
143         double avgy = 0.0;
144         for (int j = 0; j < num; j++)
145         {
146           avgx += (double)orientiers[i][j].col / num;
147           avgy += (double)orientiers[i][j].row / num;
148         }
149         //cout << "Average X: " << avgx << " and average Y: " << avgy << endl;
150         //cout << "Coordinates: " << (avgx - 50.0) << " " << (avgy - 50.0) << endl;
151         double mx = (avgx - 50.0) * P * H / F;
152         double my = (50.0 - avgy) * P * H / F;
153         double lat = my / lat_deg_len + mai_lat;
154         double lon = mx / lon_deg_len + mai_lon;
155         //cout << "Metric coordinates: " << mx << " " << my << endl;
156         //cout << fixed << setprecision(6);
157         //cout << "Geo coordinates: " << setw(9) << lat << " " << setw(9) << lon <<
          ↪  endl;
158         //cout << endl;
159         Lats.push_back(lat);
160         Lons.push_back(lon);
161         sorter.push_back(lat);
162         if (lat < min_lat)
163           min_lat = lat;
164       }
165     }
166     //cout << "\n\nFound " << count << " orientiers!!!" << endl;
167     sort(sorter.begin(), sorter.end(), greater<double>());
168     cout << fixed << setprecision(6);
169     int printed = 0;
170     for (int i = sorter.size() - 1; i >= 0; i--)
171     {
172       double value = sorter[i];
173       //cout << endl << value << endl;
174       for (int j = 0; j < Lats.size(); j++)
175       {
176         if (Lats[j] == value)
177         {
178           cout << setw(9) << Lats[j] << " " << setw(9) << Lons[j] << endl;
179         }
180       }
181     }
182     return 0;
183 }
184 // Transform degrees to radians
185 double deg2rad(double degrees)
186 {
187     return degrees * pi / 180.0;
188 }
```

Задача II.2.2. Определение высоты полета летательного аппарата (20 баллов)

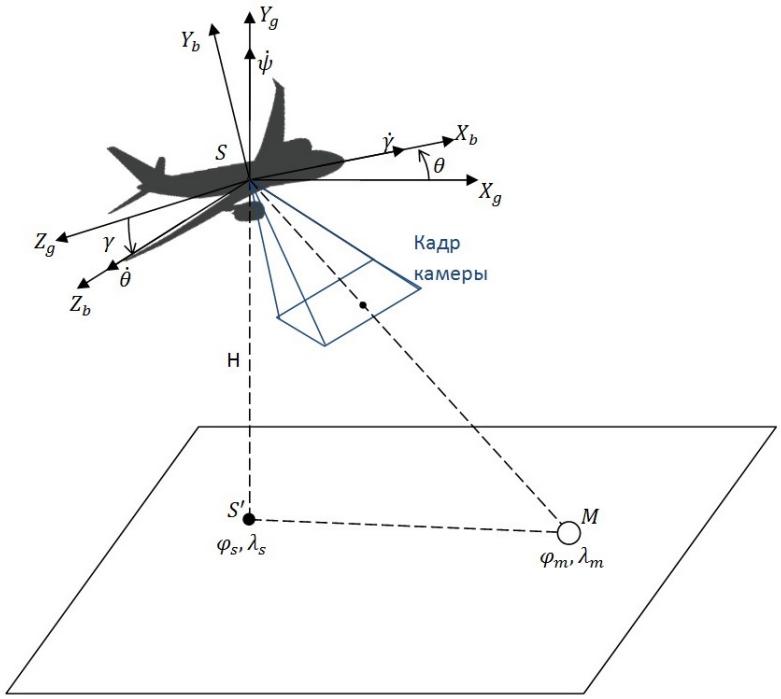
Летательный аппарат (ЛА) совершает полет в автоматическом режиме и осуществляет фотосъемку подстилающей поверхности. Две камеры, жестко закреплен-

ные на ЛА, составляют стереопару с длиной базы B.



Точка S соответствует текущему местоположению ЛА. Широта и долгота этой точки составляют $\varphi_s = 55,748252$, $\lambda_s = 48,733528$ град соответственно. В кадре каждой из камер присутствует ориентир (ориентир находится в пересечении поля зрения камер). В кадре левой камеры ориентир расположен в точке m_l, в кадре правой камеры — в точке m_p. Алгоритмы технического зрения (стереозрения) позволяют измерить расстояние от плоскости расположения камер (стереопары) до ориентира при известных длине базы B и смещении координат ориентира в кадре.

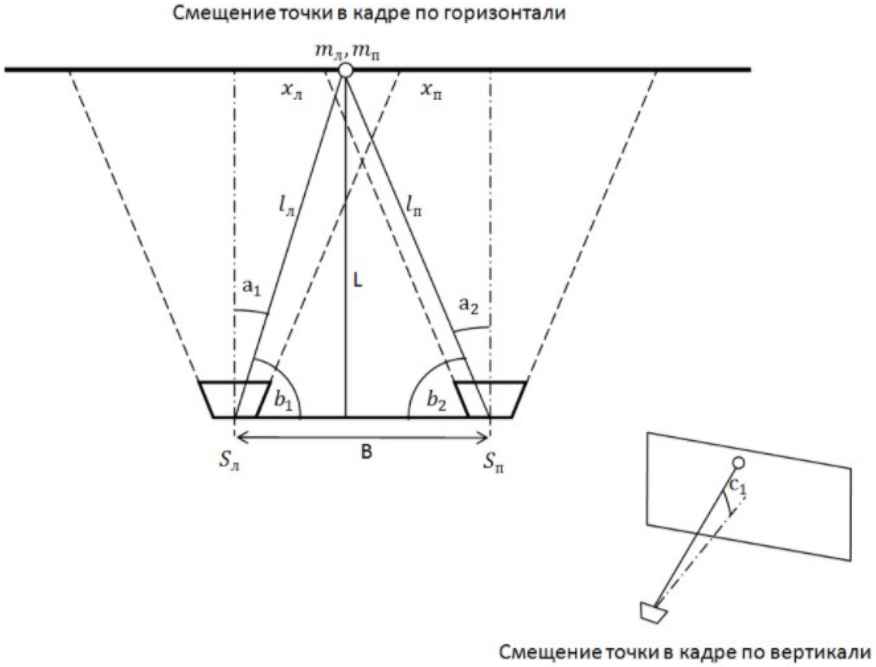
Таким образом, система технического зрения может быть использована для коррекции навигационной системы при наличии в кадре стереопары ориентиров с известными географическими координатами.



В процессе полета ЛА обнаруживает ориентир M с координатами φ_m, λ_m. На-

пишите программу на языке C++ или Python, которая определит текущую высоту полета ЛА H при заданных углах ориентации ЛА (курс ψ, тангаж θ, крен γ), текущем местоположении ЛА φ_s, λ_s, координатах ориентира (точки М) φ_m, λ_m, и пиксельных координатах точек m_l, m_p, в кадрах левой и правой камер соответственно, а также при заданной длине базы $B = 0,3$ метра. Считается, что камеры направлены вертикально вниз (ось Y кадра камер направлена противоположно оси X_b ЛА, направление оси X кадра совпадает с направлением оси Z_b ЛА).

Дополнительная иллюстрация с вебинара:



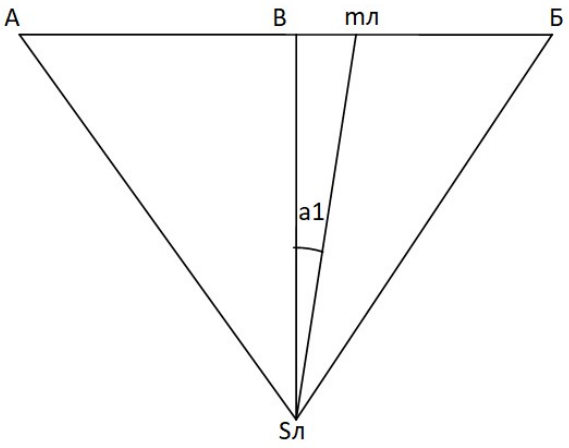
Характеристики камер:

- разрешение RESX × RESY = 2592 × 1944;
- горизонтальный угол обзора камеры (ось X) — 120 градусов;
- вертикальный угол обзора камеры (ось Y) — 90 градусов.

При пересчете долготы и широты в метрические координаты использовать коэффициенты пересчета, найденные для точки S. При нахождении этих коэффициентов считать Землю шаром с радиусом 6371 км. При расчетах считать, что подстилающая поверхность ровная и ее высота равна 0 м.

Примечание: физический размер самих камер не учитывается. Считается, что камеры расположены на расстоянии $B/2$ от точки S. Обратите внимание, что широта и долгота точки S постоянны и заданы в условии задачи.

Примечание: для расчета углов смещения ориентира в кадрах камер используйте следующие соотношения (на примере левой камеры):



АБ = 2592 пикселей

ВБ = 1296 пикселей

Вмл = хл пикселей

$\angle ASлБ = 120$ градусов

$\angle BSлБ = 60$ градусов

$$\frac{\angle BSлmл}{хл} = \frac{\angle BSлБ}{ВБ}$$

Формат входных данных

Входными данными являются следующие параметры:

Первая строка: широта φ_m и долгота λ_m точки в град (в виде десятичной дроби с точностью до 6 знака после запятой).

Вторая строка углы ориентации ЛА (курс Y, тангаж P, крен R) в градусах (угол курса ψ отсчитывается от направления на север против часовой стрелки).

Третья строка: координаты ориентира в кадре левой камеры (точка m_l) в пикселях (x_l,y_l).

Четвертая стока: координаты ориентира в кадре правой камеры (точка m_p) в пикселях (x_p,y_p). Пиксельные координаты точек отсчитываются от левого верхнего угла кадра камер (ось Y направлена вниз).

Внутри каждой строки заданные параметры разделены пробелами.

FF.FFFFFF LL.LLLLLL

YYY PPP RRR

XXXX YYYY

XXXX YYYY

Пример входных данных:

55.748500 48.733497 004 000 000 1298 0375 1291 0375

Формат выходных данных

Результатом выполнения программы должно быть значение высоты полета ЛА H, округленное до целых метров по правилам математического округления. Запись значения высоты должна быть трехзначной (например, значение 2 м. записывается как 002).

Пример результата программы:

053

Список литературы

1. Основы стереозрения [электронный ресурс], Хабр https://habr.com/ru/post/130300/.
2. Углы Эйлера [электронный ресурс], https://ru.wikipedia.org/wiki/Углы_Эйлера.
3. Географические координаты [электронный ресурс], https://ru.wikipedia.org/wiki/Географические_координаты.
4. Правила создания стереоизображения (учтите, что в рассматриваемом в задаче случае плоскость проекции стереоизображения и плоскость расположения камер совпадают – фокусное расстояние равно нулю) [электронный ресурс], https://render.ru/ru/m.titov/post/11837.

Примеры

Пример №1

Стандартный ввод
55.748500 48.733497 004 000 000 1298 0375 1291 0375

Стандартный вывод
053

Пример №2

Стандартный ввод
55.748553 48.733472 006 000 000 1298 0244 1291 0244

Стандартный вывод
054

Решение

```
#include <iostream>
#include <math.h>
#include <vector>
#include <string>
// Libraries for .clue generation
#include <fstream>
#include <sstream>
#include <iomanip>
#include <bits/stdc++.h>
```

```
using namespace std;
// Function prototypes
double deg2rad(double degrees);
// Constant values
const double pi = 3.1415926;
const int R = 6371; // km
const int FOV = 120; // degrees
const int FOVv = 90; // degrees
const int width = 2592;
const int height = 1944;
const double base = 0.3; // meters
const double lat_s = 55.748252;
const double lon_s = 48.733528;
// Main function
int main()
{
        // Read data from the input file
        string ifilename;
        ifstream ifile;
        ifilename = "input_6";
        double yaw, pitch, roll, lat, lon;
        int cam1x, cam1y, cam2x, cam2y;
        string sd;
        ifile.open(ifilename.c_str(), ios::out);
        ifile >> sd;
        lat = stod(sd);
        ifile >> sd;
        lon = stod(sd);
        ifile >> sd;
        yaw = stod(sd);
        ifile >> sd;
        pitch = stod(sd);
        ifile >> sd;
        roll = stod(sd);
        ifile >> sd;
        cam1x = stoi(sd);
        ifile >> sd;
        cam1y = stoi(sd);
        ifile >> sd;
        cam2x = stoi(sd);
        ifile >> sd;
        cam2y = stoi(sd);
        ifile.close();
        double a1 = abs(cam1x - width / 2) * deg2rad(FOV) / (width)+
                deg2rad(roll);
        double a2 = abs(cam2x - width / 2) * deg2rad(FOV) / (width)-
                deg2rad(roll);
        double b1 = pi / 2 - a1;
        double b2 = pi / 2 - a2;
        double angle = pi - b1 - b2;
        double len1 = sin(b2) * base / sin(angle);
        double Lh = len1 * sin(b1);
        double c = (height / 2 - cam1y) * deg2rad(FOVv) / (height)+
                deg2rad(pitch);
        double L = Lh / cos(c);
        cout << "L: " << L << endl;
        double deg_lat = (2 * pi * R / 360) * 1000;
        double deg_lon = (2 * pi * R / 360) * cos(deg2rad(lat_s)) * 1000;
        double xm = deg_lat * (lat - lat_s);
        double ym = deg_lon * (lon - lon_s);
```

```
70          double Lxy = sqrt(xm * xm + ym * ym);
71          cout << "Lxy: " << Lxy << endl;
72          double H = sqrt(L * L - Lxy * Lxy);
73          cout << fixed << setprecision(0) << setfill('0');
74          cout << setw(3) << round(H) << endl;
75          return 0;
76  }
77  // Transform degrees to radians
78  double deg2rad(double degrees)
79  {
80          return degrees * pi / 180.0;
81  }
```