2. ВТОРОЙ ЭТАП

Задачи второго этапа

3.1. Задачи

Задача 3.1.1. Высота и скорость полёта (15 баллов)

Приемник воздушного давления (ПВД) выдает информацию о полном $P_{\text{полн}}$ и статическом $P_{\text{стат}}$ давлении. Напишите программу, определяющую высоту H, скорость полета V и число Маха M согласно модели стандартной атмосферы, считая, что полет проходит не выше тропосферы.

Формат входных данных

Два вещественных числа через пробел $P_{\text{полн}}$ и $P_{\text{стат}}$ — значения полного и статического давления в Паскалях.

Формат выходных данных

Три вещественных числа через пробел H, V, M — высота в метрах, скорость в м/с, и число Маха с точностью не ниже 10^{-3}.

Пример №1

Стандартный ввод
`137842.5939207129 40814.5390179144`
Стандартный вывод
`7039.2966692834 575.1731540867 1.8430291164`

Решение

В теоретической справке к задаче даны следующие исходные соотношения:

$$T = T_0 + \beta H,$$
$$\frac{P}{P_0} = \left(\frac{T}{T_0}\right)^{-\frac{g}{\beta R}},$$
$$\frac{\rho}{\rho_0} = \left(\frac{T}{T_0}\right)^{-\frac{g}{\beta R}-1},$$

где H – высота [м], T – температура воздуха на высоте H [K], P – статическое давление воздуха на высоте H [Па], ρ – плотность воздуха на высоте H [кг/м^3], $T_0 = 288.15$, $P_0 = 101325$,$\rho_0 = 1.225$ – значения соответствующих величин на уровне моря ($H = 0$), $g = 9.81$ – ускорение свободного падения [м/с^2], $\beta = \frac{\partial T}{\partial H} = -0.0065$ – температурный градиент [K/м] (значение указано для стратосферы), $R = 287$ – газовая постоянная воздуха [Дж/кг/K].

Подставим зависимость температур от высоты в формулу для давлений и проведем вывод формулы для нахождения высоты по статическому давлению:

$$\frac{P}{P_0} = \left(\frac{T_0 + \beta H}{T_0}\right)^{-\frac{g}{\beta R}},$$

$$\frac{P}{P_0} = \left(1 + \frac{\beta H}{T_0}\right)^{-\frac{g}{\beta R}},$$

$$1 + \frac{\beta H}{T_0} = \left(\frac{P}{P_0}\right)^{-\frac{\beta R}{g}},$$

$$H = \frac{T_0}{\beta}\left[\left(\frac{P}{P_0}\right)^{-\frac{\beta R}{g}} - 1\right].$$

Вычислив значение высоты, по исходным формулам можно вычислить температуру и плотность.

Скорость звука рассчитывается на основе температуры с использованием следующего соотношения:

$$a = \sqrt{\gamma R T},$$

где $\gamma = 1.4$ – показатель адиабаты воздуха.

Для определения скорости полета воспользуемся законом Бернулли:

$$P_{\text{полн}} = P + \frac{\rho V^2}{2},$$

откуда выражая V получим:

$$V = \sqrt{\frac{2P_{\text{дин}}}{\rho}},$$

где $P_{\text{дин}} = P_{\text{полн}} - P$.

Число Маха есть отношение скорости полета к местной скорости звука:

$$M = \frac{V}{a}.$$

Таким образом были получены формулы для нахождения высоты, скорости полета и числа Маха на основе модели МСА. Программная реализация в образце решения основана на объектно-ориентированном подходе. Созданы два класса: Atmosisa (модель стандартной атмосферы), PitotTube (модель приемника воздушного давления). Первый класс проводит вычисления параметров атмосферы на основе статического давления, второй – скорости и числа Маха на основе параметров атмосферы и динамического давления.

Пример программы-решения

Ниже представлено решение на языке C++

```cpp
#include <iostream>
#include <math.h>

using namespace std;

///============================================================================
class PitotTube
{
public:
    PitotTube();
    ~PitotTube();

    void compute(double q, double rho, double a);

    double getAirspeed() { return V; }
    double getMachNumber() { return M; }

private:
    double V;
    double M;
};

PitotTube::PitotTube()
{
    V = 0.0;    M = 0.0;
}

PitotTube::~PitotTube()
{

}

void PitotTube::compute(double q, double rho, double a)
{
    V = sqrt(2.0*q/rho);
    M = V/a;
}

///============================================================================
class Atmosisa
{
public:
    Atmosisa();
    ~Atmosisa();

    void compute(double P);

    double getHeight() { return H; }
    double getDensity() { return rho; }
    double getTemperature() { return T; }
    double getSpeedOfSound() { return a; }

private:
    const double beta = -0.0065;
    const double g = 9.81;
    const double R = 287.0;
```

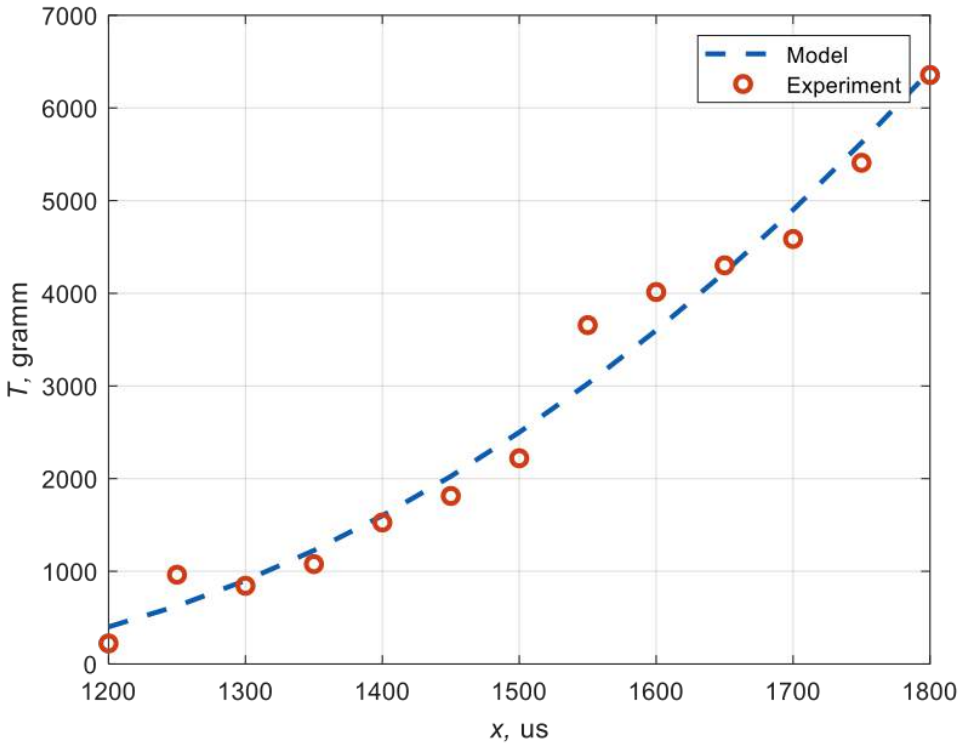
```
57      const double RB = 8.31;
58      const double gamma = 1.4;
59
60      const double P0 = 101325.0;
61      const double rho0 = 1.225;
62      const double T0 = 288.15;
63
64      double H;
65      double rho;
66      double T;
67      double a;
68  };
69
70  Atmosisa::Atmosisa()
71  {
72      H = 0.0;    rho = rho0;  T = T0; a = 340.27;
73  }
74
75  Atmosisa::~Atmosisa()
76  {
77
78  }
79
80  void Atmosisa::compute(double P)
81  {
82      H = T0/beta*(pow(P/P0, -beta*R/g) - 1.0);
83      T = T0 + beta*H;
84      rho = rho0*pow(T/T0, -g/beta/R - 1.0);
85      a = sqrt(gamma*R*T);
86  }
87
88  ///=============================================================================
89  Atmosisa atmosisa;
90  PitotTube pitotTube;
91
92  /// inputs
93  double Pfull = 3.5600e+04;
94  double Pstat = 3.5600e+04;
95
96  /// outputs
97  double H = 0.0;
98  double V = 0.0;
99  double M = 0.0;
100
101 void solve()
102 {
103     atmosisa.compute(Pstat);
104
105     pitotTube.compute(Pfull - Pstat, atmosisa.getDensity(), atmosisa.getSpeedOfSound());
106
107     H = atmosisa.getHeight();
108     V = pitotTube.getAirspeed();
109     M = pitotTube.getMachNumber();
110 }
111
112 int main()
113 {
114         /// get input
115         std::cin >> Pfull >> Pstat;
116
```

```
117         /// solve problem
118         solve();
119 
120         /// set output
121         std::cout << std::setprecision(20) << H << " " << V << " " << M << std::endl;
122     return 0;
123 }
```

Задача 3.1.2. Оценка параметров функции ШИМ – тяга винтомоторной группы БПЛА (25 баллов)

В результате испытаний винтомоторной группы БПЛА самолетного типа получена экспериментальная зависимость тяги $T = T_1, T_2, \ldots, T_N$ от длительности импульса ШИМ-сигнала, поступающего на вход регулятора оборотов $x = x_1, x_2, \ldots, x_N$, где N — объем выборки. Модельная зависимость имеет вид:

$$T = ax^2 + bx + c. (1)$$



Пример экспериментальной зависимости и
ее аппроксимации с помощью уравнения (1).

Напишите программу, определяющую по имеющимся экспериментальным замерам коэффициенты a, b, c модельной зависимости (1) такие, что их значения минимизируют суммарный квадрат отклонения модельной зависимости от экспериментальной:

$$J = \frac{1}{2}\sum_{i=1}^{N}(T_i - (ax_i^2 + bx_i + c))^2 \to min_{a,b,c}. (2)$$

В качестве ответа указать значение критерия (2) с точностью до 10^{-6} для найденных коэффициентов a, b, c.

Формат входных данных

В первой строке целое число $N(3 \leqslant N \leqslant 20)$ — количество замеров.

В следующей строке через пробел N вещественных чисел $x_i(1200 \leqslant x_i \leqslant 1800, i = 1...N)$.

В следующей строке через пробел N вещественных чисел $T_i(500 \leqslant T_i \leqslant 25000, i = 1...N)$.

Формат выходных данных

Единственное вещественное число — значение J с точностью не ниже 10^{-6}.

Пример №1

Стандартный ввод
13 1200.0000000000 1250.0000000000 1300.0000000000 1350.0000000000 1400.0000000000 1450.0000000000 1500.0000000000 1550.0000000000 1600.0000000000 1650.0000000000 1700.0000000000 1750.0000000000 1800.0000000000 6.9595774289 46.0021893921 122.2574184567 236.7533960937 387.2145491474 576.6941427344 803.2261672539 1068.1703676494 1368.2455215637 1705.5615035039 2082.3674051067 2493.9881033156 2944.9290232887
Стандартный вывод
2.3959750295

Решение

Для решения задачи применим метод наименьших квадратов (МНК). Запишем необходимые условия экстремума заданного функционала: $\frac{\partial J}{\partial a} = 0$, $\frac{\partial J}{\partial b} = 0$, $\frac{\partial J}{\partial c} = 0$. Здесь $\frac{\partial}{\partial k}$, $k = \{a, b, c\}$ – оператор частной производной. При взятии частных производных переменной считается только та величина, по которой берется производная, остальные величины дифференцируются как константы. Для заданного критерия получим:

$$\frac{\partial J}{\partial a} = \frac{\partial}{\partial a}\left\{\frac{1}{2}\sum_{i=1}^{N}[T_i - (ax_i^2 + bx_i + c)]^2\right\} =$$

$$= \sum_{i=1}^{N}[(T_i - ax_i^2 - bx_i - c)(-x_i^2)] = \sum_{i=1}^{N} x_i^4 a + \sum_{i=1}^{N} x_i^3 b + \sum_{i=1}^{N} x_i^2 c - \sum_{i=1}^{N} x_i^2 T_i = 0,$$

$$\frac{\partial J}{\partial b} = \frac{\partial}{\partial b}\left\{\frac{1}{2}\sum_{i=1}^{N}[T_i - (ax_i^2 + bx_i + c)]^2\right\} =$$

$$= \sum_{i=1}^{N}[(T_i - ax_i^2 - bx_i - c)(-x_i)] = \sum_{i=1}^{N} x_i^3 a + \sum_{i=1}^{N} x_i^2 b + \sum_{i=1}^{N} x_i c - \sum_{i=1}^{N} x_i T_i = 0,$$

$$\frac{\partial J}{\partial c} = \frac{\partial}{\partial c}\left\{\frac{1}{2}\sum_{i=1}^{N}[T_i - (ax_i^2 + bx_i + c)]^2\right\} =$$

$$= \sum_{i=1}^{N}[(T_i - ax_i^2 - bx_i - c)(-1)] = \sum_{i=1}^{N} x_i^2 a + \sum_{i=1}^{N} x_i b + Nc - \sum_{i=1}^{N} T_i = 0.$$

Полученные три уравнения представляют собой систему линейных алгебраических уравнений (СЛАУ) относительно параметров a,b,c. Введем следующие замены переменных:

$$a_{11} = \sum_{i=1}^{N} x_i^4, a_{12} = \sum_{i=1}^{N} x_i^3, a_{13} = \sum_{i=1}^{N} x_i^2, b_1 = \sum_{i=1}^{N} x_i^2 T_i,$$

$$a_{21} = \sum_{i=1}^{N} x_i^3, a_{22} = \sum_{i=1}^{N} x_i^2, a_{23} = \sum_{i=1}^{N} x_i, b_2 = \sum_{i=1}^{N} x_i T_i,$$

$$a_{31} = \sum_{i=1}^{N} x_i^2, a_{32} = \sum_{i=1}^{N} x_i, a_{33} = N, b_3 = \sum_{i=1}^{N} T_i.$$

В новых обозначениях СЛАУ примет вид:

$$a_{11}a + a_{12}b + a_{13}c = b_1,$$

$$a_{21}a + a_{22}b + a_{23}c = b_2,$$

$$a_{31}a + a_{32}b + a_{33}c = b_3.$$

Оптимальные в смысле заданного критерия параметры находятся как решение полученной системы уравнений. При использовании метода Крамера решение можно представить в следующем виде:

$$a^* = \frac{\Delta_1}{\Delta}, b^* = \frac{\Delta_2}{\Delta}, c^* = \frac{\Delta_3}{\Delta},$$

где

$$\Delta = a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{12}(a_{21}a_{33} - a_{23}a_{31}) + a_{13}(a_{21}a_{32} - a_{22}a_{31}),$$

$$\Delta_1 = b_1(a_{22}a_{33} - a_{23}a_{32}) - b_2(a_{12}a_{33} - a_{13}a_{32}) + b_3(a_{12}a_{23} - a_{13}a_{22}).$$

$$\Delta_2 = -b_1(a_{21}a_{33} - a_{23}a_{31}) + b_2(a_{11}a_{33} - a_{13}a_{31}) - b_3(a_{11}a_{23} - a_{13}a_{21}),$$

$$\Delta_3 = b_1(a_{21}a_{32} - a_{22}a_{31}) - b_2(a_{11}a_{32} - a_{12}a_{31}) + b_3(a_{11}a_{22} - a_{12}a_{21}).$$

Значение критерия для найденных коэффициентов вычисляется как

$$J = \frac{1}{2}\sum_{i=1}^{N}(T_i - a^*x_i^2 - b^*x_i - c^*)^2.$$

Программная реализация сводится к последовательному выполнению описанных шагов.

Пример программы-решения

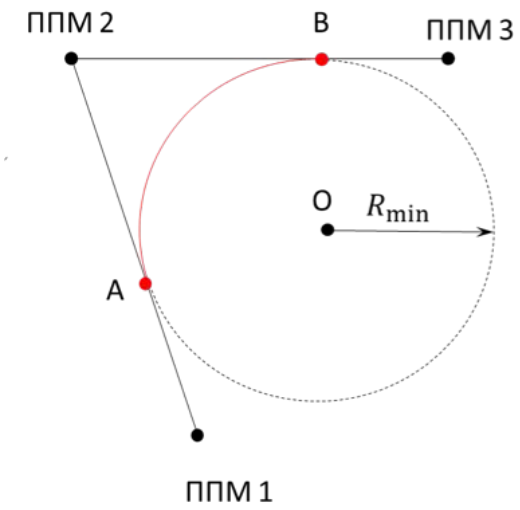
Ниже представлено решение на языке C++

```cpp
#include <iostream>
#include <math.h>

using namespace std;

///================================================================================
/// Problem solving
double *T;
double *x;
int N;
double J = 0.0;
extern void solve();

int main()
{
        cin >> N;
        x = new double[N];
        T = new double[N];
        for (int i = 0; i < N; i++)
        {
            cin >> x[i];
        }
        for (int i = 0; i < N; i++)
        {
            cin >> T[i];
        }

        /// solve problem
        solve();

        /// set output
            cout.setf(std::ios_base::fixed, std::ios_base::floatfield);
        cout.precision(10);
        cout << J << endl << endl;

        delete x;
        delete T;

    return 0;
}

///================================================================================
void solve()
{
    double sumTx2 = 0.0;
    double sumTx1 = 0.0;
    double sumTx0 = 0.0;

    double sumx4 = 0.0;
    double sumx3 = 0.0;
    double sumx2 = 0.0;
    double sumx1 = 0.0;
    double sumx0 = 0.0;

    for (int i = 0; i < N; i++)
    {
```

```
57          sumTx2 += T[i]*x[i]*x[i];
58          sumTx1 += T[i]*x[i];
59          sumTx0 += T[i];
60
61          sumx4 += x[i]*x[i]*x[i]*x[i];
62          sumx3 += x[i]*x[i]*x[i];
63          sumx2 += x[i]*x[i];
64          sumx1 += x[i];
65          sumx0 += 1.0;
66      }
67
68      double a11 = sumx4; double a12 = sumx3; double a13 = sumx2;
69      double a21 = sumx3; double a22 = sumx2; double a23 = sumx1;
70      double a31 = sumx2; double a32 = sumx1; double a33 = sumx0;
71
72      double b1 = sumTx2;  double b2 = sumTx1;  double b3 = sumTx0;
73
74      double delta = a11*(a22*a33 - a23*a32) - a12*(a21*a33 - a23*a31) +
75          a13*(a21*a32 - a22*a31);
76
77      double delta1 =  b1*(a22*a33 - a23*a32) - b2*(a12*a33 - a13*a32) +
78          b3*(a12*a23 - a13*a22);
79      double delta2 = -b1*(a21*a33 - a23*a31) + b2*(a11*a33 - a13*a31) -
80          b3*(a11*a23 - a13*a21);
81      double delta3 =  b1*(a21*a32 - a22*a31) - b2*(a11*a32 - a12*a31) +
82          b3*(a11*a22 - a12*a21);
83
84      double a = delta1/delta;
85      double b = delta2/delta;
86      double c = delta3/delta;
87
88      J = 0.0;
89      for (int i = 0; i < 13; i++)
90      {
91          J += 0.5*pow(T[i] - a*x[i]*x[i] - b*x[i] - c, 2.0);
92      }
93  }
```

Задача 3.1.3. Упрежденный разворот БПЛА самолетного типа (25 баллов)

БПЛА совершает горизонтальный полет. Программа полета БПЛА задается при помощи массива промежуточных пунктов маршрута (ППМ), где каждый i-й ППМ представляет собой точку в формате $\{\lambda_i, \varphi_i\}$, где , λ_i — долгота ППМ, φ_i — широта ППМ, измеряемые в формате: градусы, минуты, секунды. Массив ППМ кодирует кусочно-линейную траекторию, которой БПЛА, ввиду ограничений на минимальный радиус разворота, не может следовать идеально точно в точках поворота. Поэтому переход с одной линии пути на следующую происходит по линии упрежденного разворота, представляющую собой окружность, вписанную в угол между последовательными линиями пути (красная линия).



Линия упрежденного разворота при смене линий пути.

Напишите программу, определяющую координаты точек A, B, O по известным координатам трех ППМ и радиусу разворота БПЛА.

Гарантируется возможность выполнения описанного маневра. Также гарантируется, что точка A находится на линии между ППМ 1 и ППМ 2, а точка B — между ППМ 2 и ППМ 3. При пересчете долготы и широты в метрические координаты, использовать коэффициенты пересчета, найденные для точки ППМ 2. При нахождении этих коэффициентов считать Землю шаром с радиусом 6371 км.

Координаты точек даны в географической системе координат. Для решения задачи перейти в нормальную земную, прикреплённую к точке ППМ 2. Коэффициенты пересчёта географических координат в метрические вычислить для точки ППМ 2. Задачу решить в полученной плоской (нормальной земной) системе координат. Для обратного перехода в географические координаты использовать те же найденные коэффициенты.

(Литература: ГОСТ 20058-80 динамика ЛА в атмосфере)

Формат входных данных

В первой строке подается радиус разворота R [м].

Затем в следующих 3 строках по 3 числа: DD, MM, SS — координаты (часы, минуты, секунды) соответствующих ППМ.

Координаты даны в формате (Долгота, Широта).

Формат выходных данных

Три строки: соотвествующие координаты точек A, B, O в формате $DD\ MM\ SS$.

Выводить координаты следует в том же формате (Долгота, Широта).

Градусы и минуты выводить как целые числа, секунды с точностью до 5 знаков после запятой.

Пример №1

Стандартный ввод
9956.2972006643 51 6 19.4687440812 44 25 37.9940626228 51 7 17.0494705038 44 28 11.6165654469 51 9 19.9157763723 44 29 43.5638163330
Стандартный вывод
51 6 47.2291330212 44 26 52.0573941105 51 8 36.6927585101 44 29 11.2177826557 51 14 3.5882547879 44 25 28.7662660863

Решение

Для решения задачи требуется привести все координаты к единому метрическому масштабу, что можно сделать в 2 этапа. Сначала проведем преобразование координат из формата $DD.MM.SS$ к градусам и их долям dd:

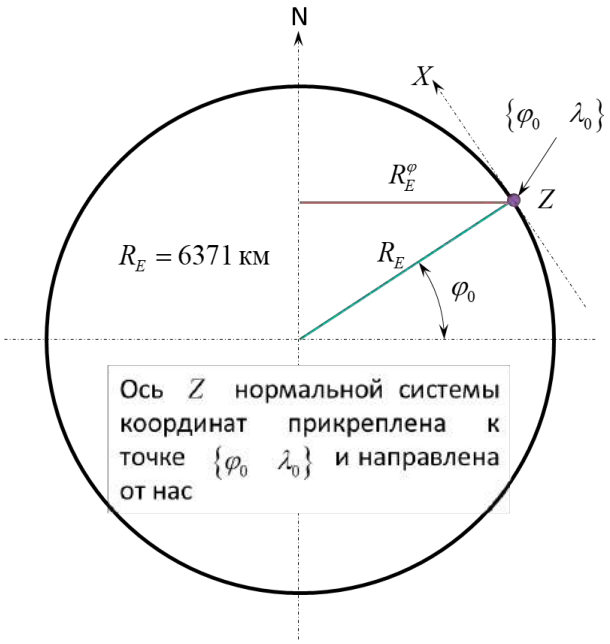
$$dd = DD + \frac{1}{60}MM + \frac{1}{3600}SS,$$

где $dd = \{\phi_i, \lambda_i\}, i = 1..3$, DD,MM,SS – градусы, минуты и секунды координат соответствующей i-ой точки.

Для пересчета угловых координат в метрические принимаются два допущения:

1. Земля имеет форму идеального шара,
2. В окрестности опорной точки (относительно которой рассчитываются коэффициенты преобразования) кривизна поверхности бесконечно мала.

За опорную точку, согласно условию задачи, принимается точка ППМ2. Тогда из геометрических соображений



Связь географической и нормальной систем координат.

получим следующие соотношения:

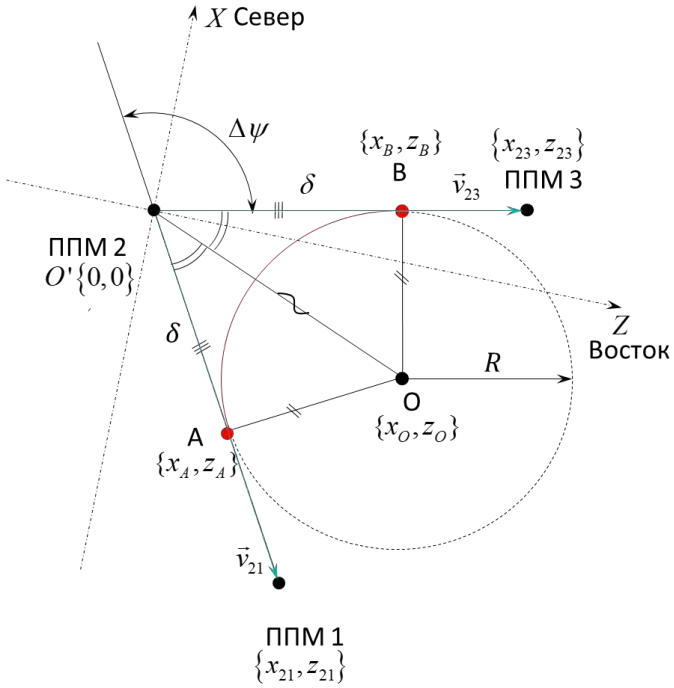
$$K_\phi = \frac{2\pi R_E}{360} = 111194.9\text{м/град}$$

$$K_\lambda = \frac{2\pi R_E^\phi}{360} = \frac{2\pi R_E cos\phi_0}{360}$$

$$X_i = K_\phi(\phi_i - \phi_0)$$

$$Z_i = K_\lambda(\lambda_i - \lambda_0)$$

с помощью которых координаты исходных ППМ переводятся к метрическому формату, и геометрия задачи сводится к геометрии на плоскости.



Геометрия упрежденного разворота в горизонтной плоскости.

Очевидно, что точка ППМ2, являющаяся началом метрической системы координат, будет иметь координаты $\{0, 0\}$

Из равенства треугольников {А ППМ2 О} и {В ППМ2 О} следует, что $[A$ ППМ2] = [В ППМ2] = δ – линейное упреждение разворота, которое можно выразить через изменение курса $\Delta\Psi$ и радиус разворота R:

$$\delta = Rtan\frac{|\Delta\Psi|}{2}.$$

Введем систему координат $O'XZ$ с центром в точке ППМ2. Пусть $\overrightarrow{v_{21}} = [x_{21}z_{21}]^T$, $\overrightarrow{v_{23}} = [x_{23}z_{23}]^T$ – векторы от точки ППМ2 до ППМ1 и ППМ3 соответственно. T – операция транспонирования. Для этих векторов справедливы соотношения скалярного и векторного произведения:

$$\overrightarrow{v_{23}} \cdot \overrightarrow{v_{21}} = \|\overrightarrow{v_{23}}\|\|\overrightarrow{v_{21}}\|cos(\pi - \Delta\Psi),$$

$$\overrightarrow{v_{23}} \times \overrightarrow{v_{21}} = \|\overrightarrow{v_{23}}\|\|\overrightarrow{v_{21}}\|sin(\pi - \Delta\Psi),$$

где $\|\cdot\|$ - норма (длина) вектора. Поделив второе выражение на первое и выразив $\Delta\Psi$ получим:

$$\Delta\Psi = \pi - atan2\frac{\overrightarrow{v_{23}} \times \overrightarrow{v_{21}}}{\overrightarrow{v_{23}} \cdot \overrightarrow{v_{21}}},$$

где для плоского случая:

$$\overrightarrow{v_{23}} \cdot \overrightarrow{v_{21}} = x_{23}x_{21} + z_{23}z_{21},$$
$$\overrightarrow{v_{23}} \times \overrightarrow{v_{21}} = x_{23}z_{21} - x_{21}z_{23}.$$

Для корректного машинного расчета необходимо, чтобы угол находился в пределах $-\pi..\pi$. Гарантировать это можно следующей простой операцией:

$$\Delta\Psi := atan2\frac{sin\Delta\Psi}{cos\Delta\Psi'},$$

где «:=» обозначает присвоение.

Метрические координаты точек A, B находятся масштабированием векторов:

$$\overrightarrow{O'A} = [x_A z_A]^T = \delta\frac{\overrightarrow{v_{21}}}{\|\overrightarrow{v_{21}}\|},$$
$$\overrightarrow{O'B} = [x_B z_B]^T = \delta\frac{\overrightarrow{v_{23}}}{\|\overrightarrow{v_{23}}\|}.$$

Точку O предлагается найти как пересечение линий AO и BO. Запишем их уравнения в нормальной земной системе координат:

$$\begin{cases} a_{11}X + a_{12}Z = b_1 \\ a_{21}X + a_{22}Z = b_2 \end{cases}$$

где $a_{11} = x_{21}$, $a_{12} = z_{21}$,$a_{21} = x_{23}$,$a_{22} = z_{23}$,$b_1 = x_{21}x_A + z_21z_A$,$b_2 = x_{23}x_B + z_{23}z_B$, $X = x_O$,$Z = z_O$. Две прямые согласно условию задачи имеют единственную точку пересечения – O, которую можно найти как решение полученной системы уравнений:

$$x_O = \frac{b_1a_{22} - b_2a_{12}}{a_{11}a_{22} - a_{12}a_{21}},$$
$$z_O = \frac{b_2a_{11} - b_1a_{21}}{a_{11}a_{22} - a_{12}a_{21}}.$$

Для осуществления преобразований к требуемому формату ответа воспользуемся формулами. которые получаются на основе прямых преобразований. Для точек $k = \{A, B, O\}$ переход в угловые координаты:

$$\phi_k = \phi_0 + \frac{x_k}{K_\phi},$$
$$\lambda_k = \lambda_0 + \frac{z_k}{K_\lambda}.$$

Переход к формату $DD.MM.SS$:

$$DD_k = \lfloor dd_k \rfloor,$$
$$MM_k = \lfloor 60(dd_k - DD_k) \rfloor,$$
$$SS_k = 3600(dd_k - DD_k - \frac{1}{60}MM_k).$$

где $\lfloor\cdot\rfloor$ - округление в меньшую сторону.

Программная реализация сводится к последовательному выполнению описанных шагов.

Пример программы-решения

Ниже представлено решение на языке C++

```cpp
#include <iostream>
#include <math.h>

using namespace std;

int WPT1_DD_x, WPT1_MM_x; double WPT1_SS_x;
int WPT1_DD_z, WPT1_MM_z; double WPT1_SS_z;
int WPT2_DD_x, WPT2_MM_x; double WPT2_SS_x;
int WPT2_DD_z, WPT2_MM_z; double WPT2_SS_z;
int WPT3_DD_x, WPT3_MM_x; double WPT3_SS_x;
int WPT3_DD_z, WPT3_MM_z; double WPT3_SS_z;

double R;
int A_DD_x, A_MM_x; double A_SS_x;
int A_DD_z, A_MM_z; double A_SS_z;
int B_DD_x, B_MM_x; double B_SS_x;
int B_DD_z, B_MM_z; double B_SS_z;
int O_DD_x, O_MM_x; double O_SS_x;
int O_DD_z, O_MM_z; double O_SS_z;

extern void solve();
extern double deg2rad(double deg);
extern double rad2deg(double rad);

int main()
{
            /// get input
        cin >> R;
        cin >> WPT1_DD_z >> WPT1_MM_z >> WPT1_SS_z;
        cin >> WPT1_DD_x >> WPT1_MM_x >> WPT1_SS_x;
        cin >> WPT2_DD_z >> WPT2_MM_z >> WPT2_SS_z;
        cin >> WPT2_DD_x >> WPT2_MM_x >> WPT2_SS_x;
        cin >> WPT3_DD_z >> WPT3_MM_z >> WPT3_SS_z;
        cin >> WPT3_DD_x >> WPT3_MM_x >> WPT3_SS_x;

        /// solve problem
        solve();

        /// set output
        cout.setf(std::ios_base::fixed, std::ios_base::floatfield);
        cout.precision(10);
        cout <<  A_DD_z << " " << A_MM_z << " " << A_SS_z << " ";
        cout <<  A_DD_x << " " << A_MM_x << " " << A_SS_x << "\n";
        cout <<  B_DD_z << " " << B_MM_z << " " << B_SS_z << " ";
        cout <<  B_DD_x << " " << B_MM_x << " " << B_SS_x << "\n";
        cout <<  O_DD_z << " " << O_MM_z << " " << O_SS_z << " ";
        cout <<  O_DD_x << " " << O_MM_x << " " << O_SS_x << "\n";
        cout << "\n";

    return 0;
}

void solve()
{
    /// constants
    const double RE = 6371000.0;   // Earth radius, m
```

```
57
58      /// waypoints
59
60      // GF
61      double WPT1_GF_x = double(WPT1_DD_x) + double(WPT1_MM_x)/60.0 + WPT1_SS_x/3600.0;
62      double WPT1_GF_z = double(WPT1_DD_z) + double(WPT1_MM_z)/60.0 + WPT1_SS_z/3600.0;
63
64      double WPT2_GF_x = double(WPT2_DD_x) + double(WPT2_MM_x)/60.0 + WPT2_SS_x/3600.0;
65      double WPT2_GF_z = double(WPT2_DD_z) + double(WPT2_MM_z)/60.0 + WPT2_SS_z/3600.0;
66
67      double WPT3_GF_x = double(WPT3_DD_x) + double(WPT3_MM_x)/60.0 + WPT3_SS_x/3600.0;
68      double WPT3_GF_z = double(WPT3_DD_z) + double(WPT3_MM_z)/60.0 + WPT3_SS_z/3600.0;
69
70      // coefs GFOC - cartesian FOC
71      double LAT2M = 2.0*M_PI*RE/360.0;
72      double LNG2M = 2.0*M_PI*RE*cos(deg2rad(WPT2_GF_x))/360.0;
73
74      // Cartesian frame
75      double vec21_GF_x = WPT1_GF_x - WPT2_GF_x;
76      double vec21_GF_z = WPT1_GF_z - WPT2_GF_z;
77
78      double vec23_GF_x = WPT3_GF_x - WPT2_GF_x;
79      double vec23_GF_z = WPT3_GF_z - WPT2_GF_z;
80
81      double WPT2_CF_x = 0;
82      double WPT2_CF_z = 0;
83
84      double WPT1_CF_x = WPT2_CF_x + vec21_GF_x*LAT2M;
85      double WPT1_CF_z = WPT2_CF_z + vec21_GF_z*LNG2M;
86
87      double WPT3_CF_x = WPT2_CF_x + vec23_GF_x*LAT2M;
88      double WPT3_CF_z = WPT2_CF_z + vec23_GF_z*LNG2M;
89
90      /// turn geometry
91      double vec21_CF_x = WPT1_CF_x - WPT2_CF_x;
92      double vec21_CF_z = WPT1_CF_z - WPT2_CF_z;
93
94      double vec23_CF_x = WPT3_CF_x - WPT2_CF_x;
95      double vec23_CF_z = WPT3_CF_z - WPT2_CF_z;
96
97      double crossProd = (vec21_CF_z*vec23_CF_x - vec21_CF_x*vec23_CF_z);
98      double dotProd   = (vec21_CF_x*vec23_CF_x + vec21_CF_z*vec23_CF_z);
99
100     double Turn = M_PI - atan2(crossProd, dotProd);
101
102         Turn = atan2(sin(Turn), cos(Turn));
103
104     // turn lead distance, m
105     double TLD = R*tan(fabs(Turn)/2.0);
106
107     /// points A, B
108     double dir21_x = vec21_CF_x/sqrt(vec21_CF_x*vec21_CF_x + vec21_CF_z*vec21_CF_z);
109     double dir21_z = vec21_CF_z/sqrt(vec21_CF_x*vec21_CF_x + vec21_CF_z*vec21_CF_z);
110
111     double dir23_x = vec23_CF_x/sqrt(vec23_CF_x*vec23_CF_x + vec23_CF_z*vec23_CF_z);
112     double dir23_z = vec23_CF_z/sqrt(vec23_CF_x*vec23_CF_x + vec23_CF_z*vec23_CF_z);
113
114     double A_CF_x = WPT2_CF_x + dir21_x*TLD;
115     double A_CF_z = WPT2_CF_z + dir21_z*TLD;
116
```

```
117     double B_CF_x = WPT2_CF_x + dir23_x*TLD;
118     double B_CF_z = WPT2_CF_z + dir23_z*TLD;
119
120     /// point O
121     double A1 = dir21_x;  double B1 = dir21_z;
122     double A2 = dir23_x;  double B2 = dir23_z;
123     double xa = A_CF_x;   double ya = A_CF_z;
124     double xb = B_CF_x;   double yb = B_CF_z;
125     double C1 = A1*xa + B1*ya;
126     double C2 = A2*xb + B2*yb;
127     double delta  = A1*B2 - A2*B1;
128     double delta1 = C1*B2 - C2*B1;
129     double delta2 = A1*C2 - A2*C1;
130     double O_CF_x = delta1/delta;
131     double O_CF_z = delta2/delta;
132
133     /// Answer
134     double A_GF_x = WPT2_GF_x + A_CF_x/LAT2M;
135     double A_GF_z = WPT2_GF_z + A_CF_z/LNG2M;
136
137     double B_GF_x = WPT2_GF_x + B_CF_x/LAT2M;
138     double B_GF_z = WPT2_GF_z + B_CF_z/LNG2M;
139
140     double O_GF_x = WPT2_GF_x + O_CF_x/LAT2M;
141     double O_GF_z = WPT2_GF_z + O_CF_z/LNG2M;
142
143     A_DD_x = floor(A_GF_x);
144     A_MM_x = floor(60.0*(A_GF_x - double(A_DD_x)));
145     A_SS_x = 3600.0*(A_GF_x - double(A_DD_x) - double(A_MM_x)/60.0);
146
147     A_DD_z = floor(A_GF_z);
148     A_MM_z = floor(60.0*(A_GF_z - double(A_DD_z)));
149     A_SS_z = 3600.0*(A_GF_z - double(A_DD_z) - double(A_MM_z)/60.0);
150
151     B_DD_x = floor(B_GF_x);
152     B_MM_x = floor(60.0*(B_GF_x - double(B_DD_x)));
153     B_SS_x = 3600.0*(B_GF_x - double(B_DD_x) - double(B_MM_x)/60.0);
154
155     B_DD_z = floor(B_GF_z);
156     B_MM_z = floor(60.0*(B_GF_z - double(B_DD_z)));
157     B_SS_z = 3600.0*(B_GF_z - double(B_DD_z) - double(B_MM_z)/60.0);
158
159     O_DD_x = floor(O_GF_x);
160     O_MM_x = floor(60.0*(O_GF_x - double(O_DD_x)));
161     O_SS_x = 3600.0*(O_GF_x - double(O_DD_x) - double(O_MM_x)/60.0);
162
163     O_DD_z = floor(O_GF_z);
164     O_MM_z = floor(60.0*(O_GF_z - double(O_DD_z)));
165     O_SS_z = 3600.0*(O_GF_z - double(O_DD_z) - double(O_MM_z)/60.0);
166 }
167
168 double deg2rad(double deg)
169 {
170     return deg/180.0*M_PI;
171 }
172
173 double rad2deg(double rad)
174 {
175     return rad*180.0/M_PI;
176 }
```

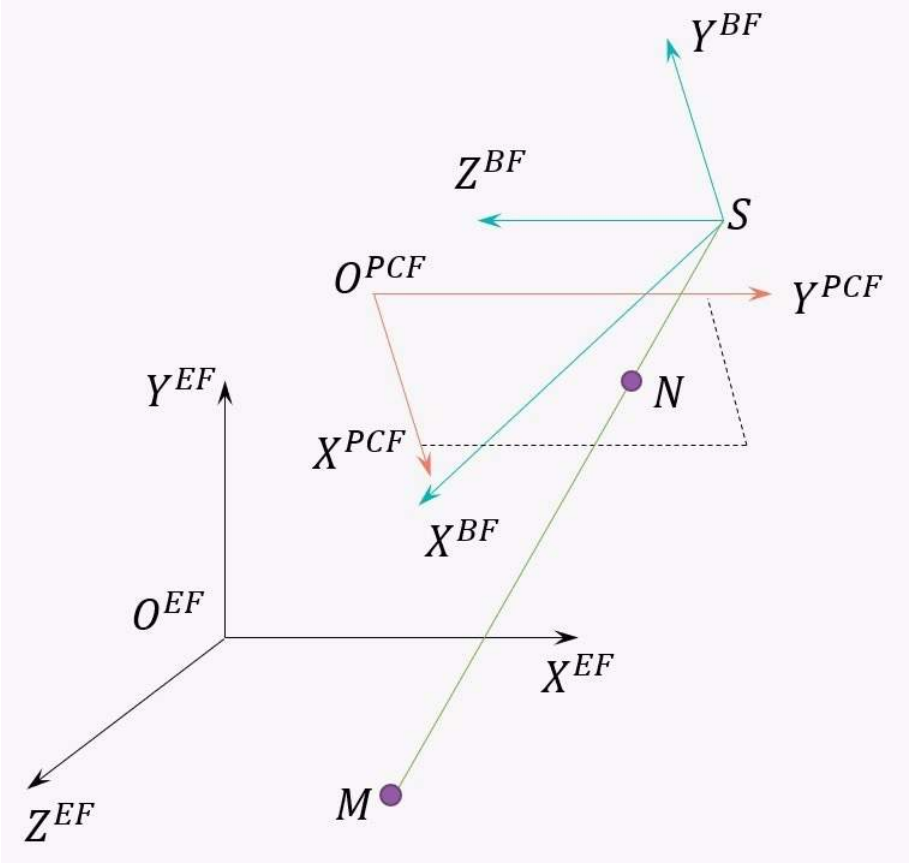
Задача 3.1.4. Определение координат наземного объекта по камере (35 баллов)

Камера жестко закреплена на корпусе БПЛА и ее оптическая ось сонаправлена с продольной осью X^{BF} связанной системы координат БПЛА (Body Frame — BF). Алгоритм технического зрения распознает искомый объект на снимке (точка N) и вычисляет его координаты в системе пиксельных координат камеры PCF (Pixel Camera Frame) в виде целочисленных пикселей (N_X^{PCF}, N_Y^{PCF}). Напишите программу, которая по известным

- координатам точки в системе пиксельных координат камеры (PCF: N_X^{PCF}, N_Y^{PCF}) [пиксели],
- углам ориентации БПЛА: рыскание ψ, крен γ, тангаж ϑ [град]
- местоположению БПЛА в нормальной земной системе координат EF (Earth Frame): S_X^{EF} (соответствует широте), S_Y^{EF}(соответствует высоте), S_Z^{EF}(соответствует долготе) [м].

определит координаты искомого объекта M в нормальной земной системе координат EF: M_X^{EF}, M_Y^{EF}[м].

При решении считать, что: координаты БПЛА и координаты точки S совпадают, оптическая ось камеры проходит через центр снимка, точка M гарантированно попадает в поле зрения камеры, подстилающая поверхность плоская и высота точки M равна 0, искажения отсутствуют.



Используемые системы координат.

Характеристики камеры:

- разрешение $RES_X \times RES_Y = 25921944$,
- размер одного пикселя $s_x = s_y = 1.4$мкм,
- фокусное расстояние $F = 3.6$ мм (между точкой S и плоскостью $O^{PCF}X^{PCF}Y^{PCF}$).

Формат входных данных

$N_X^{PCF}, N_Y^{PCF}, \psi, \vartheta, \gamma, S_X^{EF}, S_Y^{EF}, S_Z^{EF}$

Формат выходных данных

M_X^{EF}, M_Z^{EF}. Ответ дать округленным до 1 м по правилам математического округления.

Пример №1

Стандартный ввод
`2319.0000000000 160.0000000000 170.6723227638 -33.2580034791` `13.6495559557 1805.1698355052 89.1656239509 -7267.9830317087`
Стандартный вывод
`1754 -7297`

Решение

Поскольку точки M,S,N находятся на одной линии и точка M принадлежит плоскости $O^{EF}X^{EF}Z^{EF}$, то для нахождения ее координат достаточно найти точку пересечения прямой SN и плоскости $O^{EF}X^{EF}Z^{EF}$. Для этого необходимо перевести координаты точки N в систему координат двух других точек – нормальную земную.

Во-первых, проведем масштабирование координат камеры для перехода к метрическому формату:

$$N_x^{MCF} = s_x N_x^{PCF},$$
$$N_y^{MCF} = s_y N_y^{PCF},$$
$$N_z^{MCF} = 0,$$

где MCF обозначает Metric Camera Frame. Ее ось дополняет N_x^{MCF} и N_y^{MCF} до правой тройки векторов.

Во-вторых, система координат $O^{MCF}X^{MCF}Y^{MCF}Z^{MCF}$ связана со строительной системой координат $O^{BF}X^{BF}Y^{BF}Z^{BF}$ параллельным переносом:

$$N_x^{BF} = F$$
$$N_y^{BF} = -N_x^{MCF} + 0.5 s_x RES_x$$
$$N_z^{BF} = -N_y^{MCF} + 0.5 s_y RES_y$$

в результате которого становятся известными координаты точки N в строительной (связанной) системе координат.

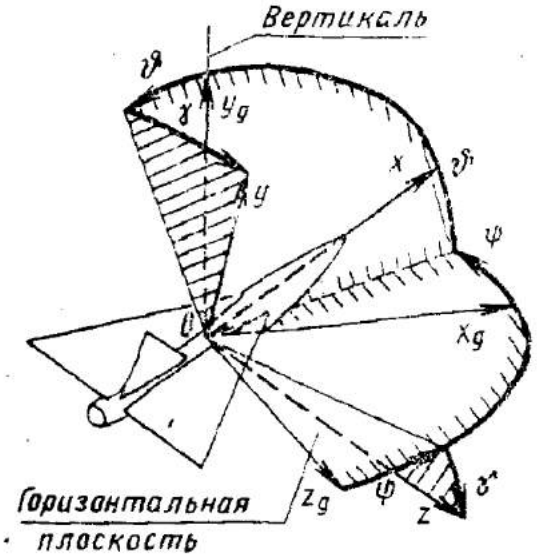
Третье заключительное преобразование состоит из двух этапов: вращение и параллельный перенос. Под вращением понимается перевод координат связанной системы в систему координат, оси которой сонаправлены осям нормальной земной $O^{EF}X^{EF}Y^{EF}Z^{EF}$, но с центром в точке S. Параллельный перенос смещает начало координат в точку O^{EF}. Преобразование имеет вид:

$$N_x^{EF} = S_x^{EF} + a_{11}N_x^{BF} + a_{12}N_y^{BF} + a_{13}N_z^{BF}$$
$$N_y^{EF} = S_y^{EF} + a_{21}N_x^{BF} + a_{22}N_y^{BF} + a_{23}N_z^{BF}$$
$$N_z^{EF} = S_z^{EF} + a_{31}N_x^{BF} + a_{32}N_y^{BF} + a_{33}N_z^{BF}$$

где

$$\begin{array}{ccc} a_{11} = c\psi c\vartheta & a_{21} = s\vartheta & a_{31} = -s\psi c\vartheta \\ a_{12} = s\psi s\gamma - c\psi s\vartheta c\gamma & a_{22} = c\vartheta c\gamma & a_{32} = c\psi s\gamma - s\psi s\vartheta c\gamma \\ a_{13} = s\psi c\gamma - c\psi s\vartheta s\gamma & a_{23} = -c\vartheta s\gamma & a_{33} = c\psi c\gamma - s\psi s\vartheta s\gamma \end{array}$$

и введены обозначения $s(\cdot) = sin(\cdot), c(\cdot) = cos(\cdot)$. Углы крена, тангажа и рыскания показаны на рисунке.



Углы ориентации ЛА.

Теперь, когда координаты точек известны в общей для них системе координат найдем координаты точки используя каноническое уравнение прямой в пространстве:

$$\frac{M_x^{EF} - S_x^{EF}}{N_x^{EF} - S_x^{EF}} = \frac{M_y^{EF} - S_y^{EF}}{N_y^{EF} - S_y^{EF}} = \frac{M_z^{EF} - S_z^{EF}}{N_z^{EF} - S_z^{EF}}$$

Из условия $M_z^{EF} = 0$ получим решение задачи:

$$M_x^{EF} = S_x^{EF} - \frac{N_x^{EF} - S_x^{EF}}{N_y^{EF} - S_y^{EF}} S_y^{EF}$$

$$M_z^{EF} = S_z^{EF} - \frac{N_z^{EF} - S_z^{EF}}{N_y^{EF} - S_y^{EF}} S_y^{EF}$$

Программная реализация сводится к последовательному выполнению описанных шагов.

Пример программы-решения

Ниже представлено решение на языке C++

```cpp
#include <iostream>
#include <iomanip>

#include <stdlib.h>
#include <time.h>
#include <math.h>

using namespace std;

///========================================================================
/// S point (aircraft) coordinates
double S_XEF, S_YEF, S_ZEF;

/// Aircraft orientation
double psi;
double theta;
double gama;   /// Euler angles
double a11, a12, a13,
       a21, a22, a23,
       a31, a32, a33;       /// Rotation matrix

/// N point coordinates
double N_XEF, N_YEF, N_ZEF;
double N_XBF, N_YBF, N_ZBF;
double N_XMCF, N_YMCF;
double N_XPCF, N_YPCF;

/// M point coordinates
double M_XEF, M_YEF, M_ZEF;

/// Camera properties
int RESX = 2592;
int RESY = 1944;
double sx = 1.4e-6;
double sy = 1.4e-6;
double F = 3.6e-3;
double WX, WY;  /// metric width and height of camera shot

extern void solve();
extern double deg2rad(double deg);
extern double rad2deg(double rad);

int main()
{
        /// get input

        cin >> N_XPCF >> N_YPCF;
        cin >> psi >> theta >> gama;
        cin >> S_XEF >> S_YEF >> S_ZEF;

        /// solve problem
        solve();

        /// set output
        cout.setf(std::ios_base::fixed, std::ios_base::floatfield);
        cout.precision(0);
```

```
57          cout << round(M_XEF) << " " << round(M_ZEF) << endl << endl;
58
59      return 0;
60  }
61
62  void solve()
63  {
64      /// solve problem
65      psi   = deg2rad(psi);
66      theta = deg2rad(theta);
67      gama  = deg2rad(gama);
68
69      /// 1. conver pixels to metric coordinates
70      N_XMCF = double(N_XPCF)*sx;
71      N_YMCF = double(N_YPCF)*sy;
72
73      /// 2. convert to body frame
74      WX = sx*double(RESX);
75      WY = sy*double(RESY);
76
77      N_XBF = F;
78      N_YBF = -N_XMCF + 0.5*WX;
79      N_ZBF = -N_YMCF + 0.5*WY;
80
81      /// 3. convert to earth frame, centered at projection of aircraft
82      /// on horizon plane
83      a11 = cos(psi)*cos(theta);
84      a12 = sin(psi)*sin(gama) - cos(psi)*sin(theta)*cos(gama);
85      a13 = sin(psi)*cos(gama) + cos(psi)*sin(theta)*sin(gama);
86
87      a21 = sin(theta);
88      a22 = cos(theta)*cos(gama);
89      a23 = -cos(theta)*sin(gama);
90
91      a31 = -sin(psi)*cos(theta);
92      a32 = cos(psi)*sin(gama) + sin(psi)*sin(theta)*cos(gama);
93      a33 = cos(psi)*cos(gama) - sin(psi)*sin(theta)*sin(gama);
94
95      N_XEF = S_XEF + a11*N_XBF + a12*N_YBF + a13*N_ZBF;
96      N_YEF = S_YEF + a21*N_XBF + a22*N_YBF + a23*N_ZBF;
97      N_ZEF = S_ZEF + a31*N_XBF + a32*N_YBF + a33*N_ZBF;
98
99      //cout << N_XEF << "\t" << N_YEF << "\t" << N_ZEF << "\n";
100
101     /// intersect SN with (0 XEF ZEF) (horizon plane)
102     M_XEF = -(N_XEF - S_XEF)/(N_YEF - S_YEF)*S_YEF + S_XEF;
103     M_YEF = 0.0;
104     M_ZEF = -(N_ZEF - S_ZEF)/(N_YEF - S_YEF)*S_YEF + S_ZEF;
105 }
106
107 double deg2rad(double deg)
108 {
109     return deg/180.0*M_PI;
110 }
111
112 double rad2deg(double rad)
113 {
114     return rad*180.0/M_PI;
115 }
```