

Командный практический тур

Задача IV.1. (15 баллов)

Описание задачи: во время разведки местности БПЛА обнаруживает зону чрезвычайной ситуации (ЧС) и совершает облет этой зоны по ее периметру. В процессе полета измерения местоположения БПЛА спутниковой навигационной системой записываются в «черный ящик». По возвращении БПЛА на базу данные из «черного ящика» анализируются для определения площади зоны ЧС, обнаруженной БПЛА.

Формулировка задачи: написать программу на языке Python, вычисляющую площадь зоны ЧС (в м²) по измерениям спутниковой навигационной системы, записанным бортовой аппаратурой БПЛА.

Входные данные: текстовый .csv файл, содержащий измерения широты и долготы бортовым GPS приемником.

Выходные данные: программа, реализующая расчет площади зоны ЧС в квадратных метрах.

Решение

Поскольку измерения спутниковой навигационной системы имеют шумовую составляющую, амплитуда которой может достигать 2–3 метров, целесообразно перед вычислением площади зоны ЧС провести фильтрацию полученных измерений. Одним из самых простых и распространенных алгоритмов фильтрации — упрощенный фильтр скользящего среднего (Simple Moving Average, SMA).

Преимуществом данного фильтра является то, что его очень просто настроить, так как он содержит всего один параметр — размер окна n . Значения n всегда нечетные. Выходным значением фильтра будет среднее арифметическое всех значений, лежащих в пределах половины окна от заданного значения:

$$y_i = \frac{1}{n} \sum_{j=i-k}^{i+k} x_j, \quad k = \frac{n-1}{2}$$

При разработке эталонного решения задачи использовался размер окна $n = 15$.

При переводе координат из градусов в метры следует учитывать, что в отличие от длины одного градуса широты, которая всегда одинакова и равна (при допущении, что Земля это шар) $\frac{2\pi R_{\text{зем}}}{360}$, длина одного градуса долготы (длина дуги произвольной параллели) в метрах изменяется при изменении широты:

$$l_{\lambda_\varphi} = \frac{2\pi R_{\text{зем}}}{360} \cos \frac{\varphi\pi}{180},$$

где: $R_{\text{зем}}$ — радиус Земли, φ — широта нужной параллели.

Для расчета площади зоны ЧС, которая представляет собой многоугольник на плоскости с известными координатами вершин, используется формула Гаусса:

$$S = \frac{1}{2} \left| \sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n \right|$$

Код программы для расчета площади зоны ЧС

```

1  import numpy as np
2  import math
3  import csv
4  R = 6371 * 1e+3 # meters
5  def moving_average(a, n = 3) :
6      ret = np.cumsum(a, dtype = float)
7      ret[n:] = ret[n:] - ret[:-n]
8      return ret[n - 1:] / n
9      csvfile = open('gps_log.csv', newline = '')
10     reader = csv.reader(csvfile, delimiter = ';')
11     time = []
12     lat = []
13     lon = []
14     header = True
15     for row in reader :
16 if header :
17     header = False
18 else :
19     time.append(float(row[0].replace(',', ' ')))
20     lat.append(float(row[1].replace(',', ' ')))
21     lon.append(float(row[2].replace(',', ' ')))
22     print(lat[0:10])
23     length = len(time)
24     print("Found {} measurements.".format(length))
25     avglat = sum(lat) / length
26     avglon = sum(lon) / length
27     print("Average coords are: {} {}".format(avglat, avglon))
28     lat2m = 2 * math.pi * R / 360.0
29     lon2m = 2 * math.pi * R / 360.0 * math.cos(avglat * math.pi / 180.0)
30     north = []
31     east = []
32     latf = moving_average(lat, n = 15)
33     lonf = moving_average(lon, n = 15)
34     lend = len(latf)
35     for i in range(0, lend) :
36         north.append((latf[i] - avglat) * lat2m)
37         east.append((lonf[i] - avglon) * lon2m)
38         S = 0.5 * (north[0] * east[lend - 1] - north[lend - 1] * east[0]);
39 for i in range(0, lend - 1) :
40     S += 0.5 * (north[i] * east[i + 1] - north[i + 1] * east[i])
41     S = abs(S)
42     print("Area equals to {}".format(S))

```

Система оценки

- Максимальный балл за решение задачи — 11 баллов.
- Оценивается точность нахождения площади зоны ЧС. За каждые 50 кв. метров ошибки списывается 1 балл.
- Правильный ответ на каждый из двух вопросов по теории прибавляет 2 балла.

Задача IV.2. (18 баллов)

Описание задачи: произошла чрезвычайная ситуация (ЧС), координаты границы зоны ЧС известны. На разведку ситуации отправляется БПЛА. Задачей БПЛА

является облет зоны ЧС по периметру в автоматическом режиме и осуществление наблюдения (фотографирования) границы указанной зоны. Для повышения качества медиа-данных, собранных БПЛА, требуется осуществить полет БПЛА по траектории как можно ближе к заданной. Слишком большое отклонение от траектории приведет к потере фрагмента данных о зоне ЧС.

Формулировка задачи: реализовать полет БПЛА по заданной пятию контрольными точками (ППМ) траектории в автоматическом режиме, при помощи внешнего контроллера.

Условия задачи:

Среда разработки: Arduino IDE.

Участникам предоставляется возможность управления БПЛА в боковом и вертикальном каналах.

Управление по вертикальному каналу осуществляется при помощи функции `HeightReg()` — алгоритм управления предоставлен разработчиками задачи.

Управление по боковому каналу осуществляется при помощи функции `GammaReg()` — алгоритм управления необходимо разработать участникам.

Ограничение по заданным значениям каналов управления:

- Боковой канал управления — угол крена $|\gamma^{\text{зад}}| \leq 30^\circ$;
- Вертикальный канал управления — угол тангажа $|\vartheta^{\text{зад}}| \leq 30^\circ$;

Приблизительная скорость полета БПЛА: $V \approx 15 - 22$, м/с.

Входные данные: массив ППМ — содержит координаты точек в системе координат симулятора, хранится в переменной `_PointsArray`, иницируется в функции `init()`.

Массив телеметрии БПЛА — содержит в себе параметры движения БПЛА, обращение к структуре массива осуществляется при помощи функции:

`getPcktTele().getBody()`.

Выходные данные: программа для внешнего контроллера в среде Arduino IDE.

Решение

Управление по курсу осуществляется через изменение угла крена БПЛА. Другими словами, входом системы управления является рассогласование $\Psi - \Psi^{\text{зад}}$, где Ψ — текущий угол курса, $\Psi^{\text{зад}}$ — требуемый угол курса, а выходная величина — заданный угол крена $\gamma^{\text{зад}}$.

Подходящим контроллером является пропорциональный регулятор (ПД-регулятор):

$$u = K_{\Psi} \epsilon$$

где ϵ — сигнал ошибки, u — управляющий сигнал, K_{Ψ} — коэффициент усиления.

Оба угла Ψ и $\Psi^{\text{зад}}$ изменяются в пределах $[-180^\circ, +180^\circ]$. Для обеспечения кратчайшего разворота сигнал ошибки ϵ рассчитывается как:

$$\epsilon = \text{atan2}(\sin(\Psi - \Psi^{\text{зад}}), \cos(\Psi - \Psi^{\text{зад}})).$$

Чтобы заданный угол крена не превышал по модулю 30 градусов, управляющий

сигнал регулятора ограничивается следующим образом:

$$\gamma^{\text{зад}} = \{u, |u| \leq 30^\circ\}$$

Коэффициент K_Ψ подбирается при таком малом сигнале рассогласования, что $|\gamma^{\text{зад}}| < 30^\circ$. Увеличение коэффициента ускоряет переходный процесс, однако, при больших значениях K_Ψ регулятор курса может работать быстрее, чем система стабилизации крена, что приведет к перерегулированию и колебательности. Удовлетворительное качество процесса регулирования достигается при $K_\Psi = 5..8$.

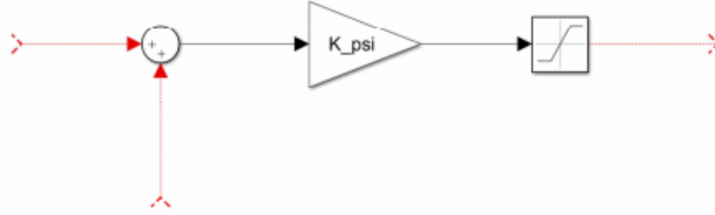


Рис. 1: Структурная схема системы управления углом крена

Управление по высоте осуществляется через изменение угла тангажа БПЛА. Другими словами, входом системы управления является рассогласование $\alpha - \alpha^{\text{зад}}$, где α — текущий угол атаки, $\alpha^{\text{зад}}$ — требуемый угол атаки, а выходная величина — заданный угол тангажа $\vartheta^{\text{зад}}$.

Подходящим контроллером является пропорциональный регулятор (ПД-регулятор):

$$\Delta H = H^{\text{зад}} - H^{\text{тек}}$$

$H^{\text{тек}}$ получаем из пакета телеметрии `_tele.H`.

V_y получаем из пакета телеметрии `_tele.dH`.

$$k_{h1} = H_h \cdot \Delta H,$$

$$\text{pitch_direct} = k_{h1} \cdot K_{vy1} - v_y \cdot K_{vy2}$$

Чтобы заданный угол тангажа не превышал по модулю 30 градусов, управляющий сигнал регулятора ограничивается следующим образом:

$$\vartheta^{\text{зад}} = \{\text{pitch_direct}, |\text{pitch_direct}| \leq 30^\circ\}$$

$$V_{y\text{max}} = \{k_{h1}, |k_{h1}| \leq 7,5 \text{ м/с}\}$$

Коэффициент K_h подбирается при таком малом сигнале рассогласования, что $|\vartheta^{\text{зад}}| < 30^\circ$. Увеличение коэффициента ускоряет переходный процесс, однако, при больших значениях K_h регулятор высоты может работать быстрее, чем система стабилизации, что приведет к перерегулированию и колебательности. Для удержания высоты и ограничения вертикальной скорости используется дифференциальное звено регулятора — коэффициенты K_{vy1} и K_{vy2} . Удовлетворительное качество процесса регулирования достигается при $K_h = 0,5..0,6$, $K_{vy1} = 7..8$ и $K_{vy2} = 9..10$.

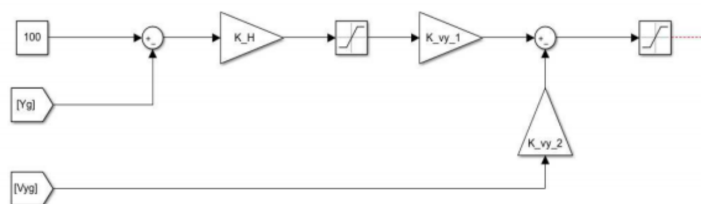


Рис. 2: Структурная схема системы управления углом тангажа.

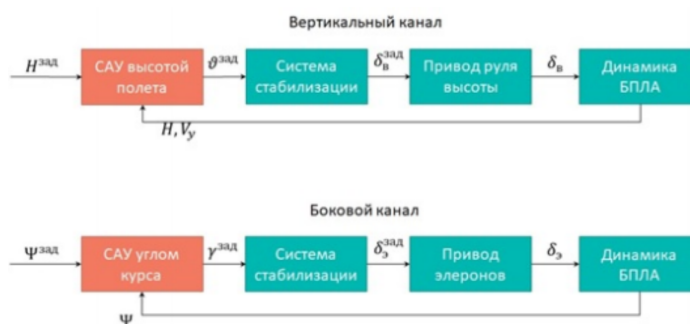


Рис. 3: Обобщенная возможная система автоматического управления

Код программы полета БПЛА по заданной траектории (пользовательский)

```

1  #include "UARTControlSystem.h"
2  #include <avr/eeprom.h>
3  #define sign(x) ({ decltype(x) _x = (x); _x < 0 ? -1 : _x ? 1 : 0; })
4  //##### УЧАСТНИКАМ ОЛИМПИАДЫ - Задача 2 ##### //
5  // ЗДЕСЬ ПИШЕМ РЕГУЛЯТОР
6  float UARTControlSystemClass::HeightReg(const Skywalker2015PacketTelemetry& _tele,
7  // const float& Hz) // _tele - структура принятого пакета телеметрии, Hz - заданная
8  // высота {
9  float K_H = 0.56;
10 float K_vy_1 = 7.7591;
11 float K_vy_2 = 9.8941;
12 float Yg = _tele.H; // Получаем из принятого пакета телеметрии текущую высоту БПЛА, м.
13 // float Vy = _tele.Vy1; // Получаем из принятого пакета телеметрии текущую
14 // вертикальную скорость БПЛА, м/с.
15 float kh1;
16 float _Pitch_direct;
17 kh1 = Saturation(K_H * (Hz - Yg), -7.5, 7.5); // Интегральное звено + Ограничитель
18 // _Pitch_direct = Saturation((kh1 * K_vy_1 - Vy * K_vy_2), -20, 20); //
19 // Дифференциальное звено + Ограничитель
20 return _Pitch_direct; // Возвращаем заданный тангаж.
21 }
22 float UARTControlSystemClass::ToPointXY(const Skywalker2015PacketTelemetry& _tele,
23 // const float& a_Xg, const float& a_Yg)
24 {
25     float _Xt = _tele.L;
26     float _Yt = _tele.Z;
27     float Psi_cmd = -atan2(a_Yg - _Yt, a_Xg - _Xt); // курс на точку, радианы
28     // Psi_cmd *= 57.3; // Переведем в градусы
29     return GammaReg(_tele, Psi_cmd);
30 }

```

```

21 }
22 bool lockDir = false;
23 float Direction = 0;
24 float UARTControlSystemClass::GammaReg(const Skywalker2015PacketTelemetry& _tele,
    ↪ const float& Psi_direct_deg)
25 {
26     static bool _doonce = true;
27     static bool _doonce2 = true;
28     static float Psi_direct;
29     float gamma_cmd;
30     float Psi_dir = Psi_direct_deg;
31     float Xg_cmd = _PointsArray[_Point_Index].L;
32     float Zg_cmd = _PointsArray[_Point_Index].Z;
33     float Yg_cmd = _PointsArray[_Point_Index].H;
34     float delX = Xg_cmd - _tele.L;
35     float delY = Zg_cmd - _tele.Z;
36     float delZ = Yg_cmd - _tele.H;
37     float Size = sqrtf(pow(delX, 2) + pow(delY, 2) + pow(delZ, 2));
38     // Регулятор курса
39     float Kpsi = 1.0;
40     float Psi = _tele.Psi; // Получаем с принятого пакета телеметрии текущий угол
    ↪ курса(в связанной СК), град.
41     gamma_cmd = Kpsi * (AngDefines(Psi - Psi_dir));
42     // Problem 2
43     //gamma_cmd = gamma_cmd - 0.45 * _Trajectory_Offset - 1.1 *
    ↪ (_Trajectory_Offset - last_cte);
44     last_cte = _Trajectory_Offset;
45     gamma_cmd = Saturation(gamma_cmd, -30, 30);
46     return gamma_cmd; // Заданный угол крена, град
47 }
48 float UARTControlSystemClass::PointnDrop(const Skywalker2015PacketTelemetry& _tele) {
49     return 0.0f;
50 }
51 //#####
52 //##### УЧАСТНИКАМ ОЛИМПИАДЫ - Отправка пакета сигнала ##### //
    ↪ Стоит учитывать что частота обмена данными с симулятором - 100 Гц. // Тело
    ↪ функции "SendSignalPacket()" вызываемой при удачном приеме пакета телеметрии //
    ↪ переменным getPcktSignal().getBodyMutable().Gamma_direct и
    ↪ getPcktSignal().getBodyMutable().Tang_direct присваивается результат расчета
    ↪ регулятора void UARTControlSystemClass::SendSignalPacket()
53 {
54     //##### Задача 2 #####
    ↪ _Point_Index = GetNowPointIndex(getPcktTele().getBody()); // Обновляем
    ↪ текущий индекс точки
55     getPcktSignal().getBodyMutable().Gamma_direct =
    ↪ PointsFlyGam(getPcktTele().getBody(), _Point_Index); // Значение
    ↪ заданного угла крена поступающего на стабилизатор, в градусах!
56     getPcktSignal().getBodyMutable().Tang_direct =
    ↪ PointsFlyTan(getPcktTele().getBody(), _Point_Index); // Значение заданного
    ↪ угла тангажа поступающего на стабилизатор, в градусах!
57     //#####
58     if (counter++ % 10 == 0)
59     {
60         SendTele(getPcktTele().getBody());
61         counter = 0;
62     }
63     //##### Данные вызовы функций изменению не подлежат. #####
    ↪ // Обновляем контрольную сумму пакета.
64     getPcktSignal().updateChecksum();
65     // Отсылаем пакет по UART

```

```

66     sendBuf(getPcktSignal().getData(), getPcktSignal().dataSizeBytes);
67     ↪ //##### }
68     ↪
69     void UARTControlSystemClass::SendANSIMsg(::Ids a_packetId)
70     {
71         switch (a_packetId)
72         {
73             case bogusPacketAtPosfixLoss:
74                 Serial.println("bogusPacketAtPosfixLoss");
75                 break;
76             case malformed:
77                 Serial.println("Wrong structure.");
78                 break;
79             case invalid:
80                 Serial.println("Checksum Error");
81                 break;
82             case unknown:
83                 Serial.println("unknown id packet");
84                 break;
85             case cmd:
86                 Serial.println("these packet not place to be here");
87                 break;
88             case Telemetry:
89                 break;
90             case Signal:
91                 Serial.println("these packet not place to be here");
92                 break;
93             case Protection:
94                 Serial.println("ProtectionCheck");
95                 break;
96             default:
97                 break;
98         }
99     }
100     void UARTControlSystemClass::SendCmdPacket()
101     {
102         getPcktCommand().getBodyMutable().cmd = Delivery::Drop;
103         getPcktCommand().updateChecksum();
104         sendBuf(getPcktCommand().getData(), getPcktCommand().dataSizeBytes);
105     }
106     float UARTControlSystemClass::PointsFlyGam(const Skywalker2015PacketTelemetry
107     ↪ & _tele, const size_t & a_Point_index)
108     {
109         float Xg_cmd = _PointsArray[_Point_Index].L;
110         float Zg_cmd = _PointsArray[_Point_Index].Z;
111         if (_User_Control_Enabled)
112         {
113             Xg_cmd = _New_Waypoint.L;
114             Zg_cmd = _New_Waypoint.Z;
115         }
116         return ToPointXY(_tele, Xg_cmd, Zg_cmd);
117     }
118     float UARTControlSystemClass::PointsFlyTan(const Skywalker2015PacketTelemetry
119     ↪ & _tele, const size_t & a_Point_index)
120     {
121         float H_z = _PointsArray[_Point_Index].H;
122         if (_User_Control_Enabled)
123         {
124             H_z = _New_Waypoint.H;
125         }
126     }

```

```

122         return HeightReg(_tele, H_z);
123     }
124     size_t UARTControlSystemClass::GetNowPointIndex(const
    ↪ Skywalker2015PacketTelemetry & _tele)
125     {
126         size_t max_index = 13;
127         float Xg_cmd = _PointsArray[_Point_Index].L;
128         float Zg_cmd = _PointsArray[_Point_Index].Z;
129         float Yg_cmd = _PointsArray[_Point_Index].H;
130         float delX = Xg_cmd - _tele.L;
131         float delY = Zg_cmd - _tele.Z;
132         float delZ = Yg_cmd - _tele.H;
133         float Size = sqrtf(pow(delX, 2) + pow(dely, 2) + pow(delZ, 2));
134         // Calculate crosstrack error
135         float x_og = 0.0;
136         float z_og = 0.0;
137         if (_Point_Index > 0)
138         {
139             x_og = _PointsArray[_Point_Index - 1].L;
140             z_og = _PointsArray[_Point_Index - 1].Z;
141         }
142         else
143         {
144             x_og = _PointsArray[max_index - 1].L;
145             z_og = _PointsArray[max_index - 1].Z;
146         }
147         float relx = _tele.L - x_og;
148         float rely = _tele.Z - z_og;
149         float delx = Xg_cmd - x_og;
150         float dely = Zg_cmd - z_og;
151         float U = (relx * delx + rely * dely) / (delx * delx + dely * dely);
    ↪ float cte = (rely * delx - relx * dely) / (delx * delx + dely *
    ↪ dely); _Trajectory_Offset = cte * 1500.0;
152         //Serial3.println(cte);
153         if (Size < 7.5)
154         {
155             _Point_Index++;
156             _User_Control_Enabled = false;
157         }
158         // Выход за границы массива
159         if (_Point_Index >= max_index)
160         {
161             _Point_Index = 0;
162         }
163         return _Point_Index;
164     }
165     inline float UARTControlSystemClass::AngDefines(const float& angl)
166     {
167         float z1 = angl * M_PI / 180;
168         float z2 = atan2(sin(z1), cos(z1)) * 180 / M_PI;
169         return z2;
170     }
171     inline float UARTControlSystemClass::Saturation(const float& c, const float&
    ↪ a, const float& b)
172     {
173         if (c < a)
174             return a;
175         else if (c > b)
176             return b;
177         else

```

```

178         return c;
179     }
180     void UARTControlSystemClass::onUartPacketReceived(::Ids a_packetId)
181     {
182         switch (a_packetId)
183         {
184             case Telemetry:
185                 // при получении телеметрии сразу отправляем пакет сигнала. В
186                 // эту функцию можно записать так же отправку командного
187                 // пакета (на сброс груза)
188                 SendSignalPacket();
189                 break;
190             case Protection:
191                 AccessCheckBody _prot;
192                 eeprom_read_block((void*)&_prot, 10, sizeof(_prot));
193                 getPcktProtection().getBodyMutable() = _prot;
194                 getPcktProtection().updateChecksum();
195                 sendBuf(getPcktProtection().getData(),
196                     // getPcktProtection().dataSizeBytes); break;
197                 );
198             default:
199                 // Пишем в порт сведения о неправильном приеме пакета.
200                 SendANSIMsg(a_packetId);
201                 break;
202         }
203     }
204     void UARTControlSystemClass::onUartPostfixLost()
205     {
206         // Serial.flush();
207         Serial.println("onUartPostfixLost");
208     }
209     void UARTControlSystemClass::init()
210     {
211         // Problem 4
212         // _PointsArray[4] = (FlyPoints(0, 0, 120));
213         // _PointsArray[0] = (FlyPoints(-230*2, 300*2, 120));
214         // _PointsArray[1] = (FlyPoints(0, 800, 120));
215         // _PointsArray[2] = (FlyPoints(300, 600, 120));
216         // _PointsArray[3] = (FlyPoints(300, 200, 120));
217         // Problem 2
218         // Expanded trajectory
219         _PointsArray[0] = (FlyPoints(600, -500, 100)); // wp
220         _PointsArray[1] = (FlyPoints(700, 100, 100)); // wp
221         _PointsArray[2] = (FlyPoints(750, 300, 115));
222         _PointsArray[3] = (FlyPoints(750, 145, 115));
223         _PointsArray[4] = (FlyPoints(50, -500, 100)); // wp
224         _PointsArray[5] = (FlyPoints(0, -600, 100));
225         _PointsArray[6] = (FlyPoints(100, -550, 100));
226         _PointsArray[7] = (FlyPoints(-600, -500, 100)); // wp
227         _PointsArray[8] = (FlyPoints(-750, -500, 100));
228         _PointsArray[9] = (FlyPoints(-650, -460, 100));
229         _PointsArray[10] = (FlyPoints(-100, -800, 100)); // wp
230         _PointsArray[11] = (FlyPoints(-50, -830, 100));
231         _PointsArray[12] = (FlyPoints(-150, -950, 100));
232         // RAW trajectory
233         // _PointsArray[0] = (FlyPoints(600, -500, 100)); // wp
234         // _PointsArray[1] = (FlyPoints(700, 100, 100)); // wp
235         // _PointsArray[2] = (FlyPoints(50, -500, 100)); // wp
236         // _PointsArray[3] = (FlyPoints(-600, -500, 100)); // wp
237         // _PointsArray[4] = (FlyPoints(-100, -800, 100)); // wp
238     }

```

```

235 void UARTControlSystemClass::sendBuf(const uint8_t * a_data, uint8_t
    ↪ a_byteCount) {
236     uint8_t _sendBuf[200];
237     UartTransportPacketFormatterClass _Formatter(a_data, a_byteCount);
    ↪ for (size_t i = 0; i < _Formatter.getBytesCount(); ++i)
238     {
239         _sendBuf[i] = _Formatter.getBytes(i);
240     }
241     Serial.write(_sendBuf, _Formatter.getBytesCount());
242 }
243 void UARTControlSystemClass::SendTele(const Skywalker2015PacketTelemetry &
    ↪ _tele) {
244     float H = _tele.H; // Высота, м
245     float Z = _tele.Z; // Отклонение по оси Z, м
246     float L = _tele.L; // Дальность, м
247     float Vx1 = _tele.Vx1; // Скорость Vx1, м/с
248     float Vy1 = _tele.Vy1; // Скорость Vy1, м/с
249     float Vz1 = _tele.Vz1; // Скорость Vz1, м/с
250     float wx = _tele.wx; // Угловая скорость в связанной СК, 1/с
251     float wy = _tele.wy; // Угловая скорость в связанной СК, 1/с
252     float wz = _tele.wz; // Угловая скорость в связанной СК, 1/с
253     float Theta = _tele.Tan; // Угол наклона траектории град.
254     float Psi = _tele.Psi; // Угол курса в связанной СК, град.
255     float Gamma = _tele.Gam; // Угол крена в связанной СК, град.
256     float V = _tele.V; // Скорость БПЛА, м/с
257     char s[128];
258     //snprintf_P(s, sizeof(s), PSTR("tele: %f %f %f %f %f %f %f %f %f %f
    ↪ %f %f \n"), H,Z,L, Theta,Psi,Gamma, dH,dZ,dL, wx,wy,wz);
259     //Serial3.write(s);
260     Serial3.print("tele: ");
261     Serial3.print(L);
262     Serial3.print(" ");
263     Serial3.print(Z);
264     Serial3.print(" ");
265     Serial3.print(H);
266     Serial3.print(" ");
267     Serial3.print(Theta);
268     Serial3.print(" ");
269     Serial3.print(Psi);
270     Serial3.print(" ");
271     Serial3.print(Gamma);
272     Serial3.print(" ");
273     Serial3.print(Vx1);
274     Serial3.print(" ");
275     Serial3.print(Vy1);
276     Serial3.print(" ");
277     Serial3.print(Vz1);
278     Serial3.print(" ");
279     Serial3.print(wx);
280     Serial3.print(" ");
281     Serial3.print(wy);
282     Serial3.print(" ");
283     Serial3.print(wz);
284     Serial3.println(" ");
285 }
286 UARTControlSystemClass UARTControlSystem;
287 }

```

Система оценки

- Максимальный балл за решение задачи — 14 баллов.
- По заданной траектории равномерно распределены 27 путевых точек. За прохождение каждой точки (области в радиусе 5 м. от нее) начисляется 0,5 балла.
- При подсчете итогового результата баллы округляются вверх. Таким образом, при прохождении всех точек 13,5 баллов округляются до 14.
- Правильный ответ на каждый из двух вопросов по теории прибавляет 2 балла.

Задача IV.3. (27 баллов)

Описание задачи: БПЛА, оснащенный грузом, направляется для устранения чрезвычайной ситуации (ЧС). Задачей БПЛА является сброс груза в центр зоны ЧС. Однако из-за отсутствия возможности наблюдения за развитием событий в режиме реального времени точные координаты точки сброса неизвестны. БПЛА пролетает над зоной ЧС и осуществляет фотосъемку подстилающей поверхности для анализа текущей ситуации и вычисления точных координат точки сброса груза.

Формулировка задачи: написать программу на языке Python, определяющую широту и долготу центра (среднего арифметического) зоны ЧС (в град.), ограниченной контрастными ориентирами на предоставленном снимке с бортовой камеры БПЛА.

Входные данные: изображение с бортовой камеры; параметры бортовой камеры; навигационные параметры БПЛА (координаты и ориентация) на момент осуществления съемки.

Выходные данные: программа реализующая расчет географических координат (широты и долготы в град.) среднего арифметического центра зоны ЧС с использованием алгоритмов технического зрения и библиотеки OpenCV.

Решение

При переводе координат из градусов в метры следует учитывать, что в отличие от длины одного градуса широты, которая всегда одинакова и равна (при допущении, что Земля это шар) $\frac{2\pi R_{зем}}{360}$, длина одного градуса долготы (длина дуги произвольной параллели) в метрах изменяется при изменении широты:

$$l_{\lambda\varphi} = \frac{2\pi R_{зем}}{360} \cos \frac{\varphi\pi}{180}$$

где: $R_{зем}$ — радиус Земли, φ — широта нужной параллели.

При помощи матрицы направляющих косинусов можно перевести любой вектор из географической системы координат $XgYgZg$ в систему координат $XbYbZb$, жестко связанную с бортом БПЛА. Перед использованием этой матрицы следует проверить направления осей.

$$\begin{bmatrix} X_b \\ Y_b \\ Z_b \end{bmatrix} = M_g^b \times \begin{bmatrix} X_g \\ Y_g \\ Z_g \end{bmatrix} \quad (1)$$

При осуществлении последовательных поворотов БПЛА на углы рыскания Ψ , тангажа θ и крена γ выглядит так:

$$M_g^b = \begin{bmatrix} \cos\Theta\cos\Psi & \sin\gamma\sin\Theta\cos\Psi - \cos\gamma\sin\Psi & \sin\gamma\sin\Psi + \cos\gamma\sin\Theta\cos\Psi \\ \cos\Theta\sin\Psi & \cos\gamma\cos\Psi + \sin\gamma\sin\Theta\sin\Psi & \cos\gamma\sin\Theta\sin\Psi - \sin\gamma\cos\Psi \\ -\sin\Theta & \sin\gamma\cos\Theta & \cos\gamma\cos\Theta \end{bmatrix} \quad (2)$$

Воспользовавшись операцией умножения матриц, использование матрицы направляющих косинусов для поворота вектора можно записать в скалярном виде:

$$X_b = (\cos\Theta\cos\Psi)X_g + (\sin\gamma\sin\Theta\cos\Psi - \cos\gamma\sin\Psi)Y_g + (\sin\gamma\sin\Psi + \cos\gamma\sin\Theta\cos\Psi)Z_g$$

$$Y_b = (\cos\Theta\sin\Psi)X_g + (\cos\gamma\cos\Psi + \sin\gamma\sin\Theta\sin\Psi)Y_g + (\cos\gamma\sin\Theta\sin\Psi - \sin\gamma\cos\Psi)Z_g$$

$$Z_b = (-\sin\Theta)X_g + (\sin\gamma\cos\Theta)Y_g + (\cos\gamma\cos\Theta)Z_g$$

При наличии углов крена и тангажа БПЛА возникает оптическое искажение изображения, полученного бортовой камерой. Это явление возникает, когда плоскость камеры параллельна плоскости подстилающей поверхности.

Если углы тангажа и крена равны нулю — искажения не возникает и $l_{\text{л}} = l_{\text{п}}$. Если же углы тангажа и крена не равны нулю — длина $l_{\text{л}}$ на поверхности земли будет больше длины $l_{\text{п}}$. При этом разрешение снимка камеры остается постоянным. Это означает, что пикселям, лежащим в левой части кадра (в примере на рисунке 4) будет соответствовать большее расстояние в метрах на поверхности, чем пикселям в правой части кадра, так как они находятся дальше от БПЛА.

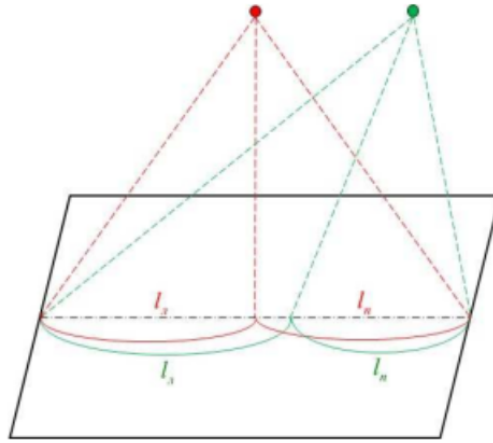


Рис. 4: Геометрия искажения изображения при ненулевом тангаже и/или крене

Код программы для поиска координат центра зоны ЧС

```

1 import cv2 # used for image processing, duh
2 import numpy as np # used to convert images from PIL to cv2
3 import math # used for trigonometry functions
4 # Параметры БПЛА
5 uavLat = 56.097977 # degrees
6 uavLon = 35.877670 # degrees
7 uavYaw = 35 # degrees

```

```

8  uavPitch = 5 # degrees
9  uavRoll = 3 # degrees
10 H = 50 # meters
11 # Параметры камеры
12 F = 5.2 * 1e-3 # meters
13 sx = 7.6 * 1e-6 # meters
14 sy = 8.1 * 1e-6 # meters
15 width = 2400 # pixels
16 height = 1800 # pixels
17 fieldHor = 80; # degrees
18 fieldVer = 30; # degrees
19 # Параметры Земли
20 R = 6371 * 1e+3 # meters
21 lat2m = 2 * math.pi * R / 360.0
22 lon2m = 2 * math.pi * R / 360.0 * math.cos(uavLat * math.pi / 180.0)
23 c1 = math.cos(0 * math.pi / 180.0)
24 c2 = math.cos(uavPitch * math.pi / 180.0)
25 c3 = math.cos(uavRoll * math.pi / 180.0)
26 s1 = math.sin(0 * math.pi / 180.0)
27 s2 = math.sin(uavPitch * math.pi / 180.0)
28 s3 = math.sin(uavRoll * math.pi / 180.0)
29 Rgb = np.array([[c2 * c1, s3 * s2 * c1 - c3 * s1, c3 * s2 * c1 + s3 * s1], [c2 * s1,
    ↪ s3 * s2 * s1 + c3 * c1, c3 * s2 * s1 - s3 * c1], [-s2, s3 * c2, c3 * c2]] )
30 G = np.array([[0], [0], [H]] )
31 vect = np.matmul(Rgb, G)
32 print(vect)
33 dx = vect[0] / vect[2] * H
34 dy = vect[1] / vect[2] * H
35 print("Image center is displaced by {} and {} meters".format(dx, dy)) #
36 #color definition
37 red_lower = np.array([0, 120, 150], dtype = "uint8") red_upper = np.array([20, 255,
    ↪ 255], dtype = "uint8") red_lower_v = np.array([235, 120, 150], dtype = "uint8")
    ↪ red_upper_v = np.array([255, 255, 255], dtype = "uint8")
38 # Main cycle
39 rawFrame = cv2.imread('cameraShot.JPG')
40 # Normalize image brightness
41 cv2.normalize(rawFrame, rawFrame, 0, 255, cv2.NORM_MINMAX) # Blur the image to reduce
    ↪ noise
42 blur = cv2.medianBlur(rawFrame, 5)
43 # Convert BGR to HSV
44 hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
45 # Find color masks
46 red3_mask = cv2.inRange(hsv, red_lower, red_upper)
47 red2_mask = cv2.inRange(hsv, red_lower_v, red_upper_v) full_mask = red2_mask +
    ↪ red3_mask
48 cv2.imwrite("deb.png", hsv)
49 # Find counters
50 contours, hierarchy = cv2.findContours(full_mask, cv2.RETR_LIST,
    ↪ cv2.CHAIN_APPROX_SIMPLE)
51 numCon = 0
52 orient_x = []
53 orient_y = []
54 for icontour in contours :
55 if len(icontour) > 50:
56 numCon += 1
57 # Draw circles around found contours on the raw image ellipse =
    ↪ cv2.fitEllipse(icontour)
58 cv2.ellipse(rawFrame, ellipse, (0, 0, 0), 2)
59 # find location of the orientier's center
60 sum_x = 0

```

```

61 sum_y = 0
62 for i in range(0, len(icontour)) :
63     sum_x += icontour[i][0][0]
64     sum_y += icontour[i][0][1]
65     orient_x.append(sum_x / len(icontour))
66     orient_y.append(sum_y / len(icontour))
67     #print("Found {} orientiers! Their coords are:".format(numCon))    avg_x = 0
68     avg_y = 0
69     for i in range(0, numCon) :
70         #print("{} {}".format(orient_x[i], orient_y[i]))
71         avg_x += orient_x[i]
72         avg_y += orient_y[i]
73         if numCon > 0:
74             # Coordinates of the center
75             avg_x = int(round(avg_x / numCon))
76             avg_y = int(round(avg_y / numCon))
77             print("Point location: {}, {}".format(avg_x, avg_y))
78             dyb = (avg_x - width / 2) * sx / F * H
79             dxb = (height / 2 - avg_y) * sy / F * H
80             print("Dislocation of the drop point is {} and {} meters".format(dxb, dyb))
81             ↪ print("Sum shift in base frame: {}, {}".format(dx + dxb, dy + dyb))    dybt = dyb;
82             dxbt = dxb;
83             # account for image distortion
84             fowx = abs((avg_x - width / 2) * fieldHor / width) * math.pi / 180.0    fowy =
85             ↪ abs((avg_y - height / 2) * fieldVer / height) * math.pi / 180.0    if avg_x < width
86             ↪ / 2:
87             dybt = math.sin(math.pi / 2 + fowx) * dyb / math.sin(math.pi / 2 - fowx - uavRoll *
88             ↪ math.pi / 180.0)
89             else:
90             dybt = math.sin(math.pi / 2 - fowx) * dyb / math.sin(math.pi / 2 + fowx - uavRoll *
91             ↪ math.pi / 180.0)
92             if avg_y < height / 2 :
93             dxbt = math.sin(math.pi / 2 + fowy) * dxb / math.sin(math.pi / 2 - fowy - uavPitch
94             ↪ * math.pi / 180.0)
95             else :
96             dxbt = math.sin(math.pi / 2 - fowy) * dxb / math.sin(math.pi / 2 + fowy - uavPitch
97             ↪ * math.pi / 180.0)
98             print("Image distortion: {}, {}".format(dxbt - dxb, dybt - dyb))    shift =
99             ↪ np.array([dx[0] + dxbt, dy[0] + dybt, [0.0]] )
100             c1 = math.cos(uavYaw * math.pi / 180.0)
101             c2 = math.cos(0 * math.pi / 180.0)
102             c3 = math.cos(0 * math.pi / 180.0)
103             s1 = math.sin(uavYaw * math.pi / 180.0)
104             s2 = math.sin(0 * math.pi / 180.0)
105             s3 = math.sin(0 * math.pi / 180.0)
106             Rgby = np.array([[c2 * c1, s3 * s2 * c1 - c3 * s1, c3 * s2 * c1 + s3 * s1], [c2 *
107             ↪ s1, s3 * s2 * s1 + c3 * c1, c3 * s2 * s1 - s3 * c1],
108             [-s2, s3 * c2, c3 * c2]] )
109             #Rbgy = np.linalg.inv(Rbgy)
110             vect2 = np.matmul(Rgby, shift)
111             print("Sum shift in geo frame: {}, {}".format(vect2[0], vect2[1]))    # finde geo
112             ↪ shift
113             shiftLat = vect2[0] / lat2m;
114             shiftLon = vect2[1] / lon2m;
115             print("GEO shift: {}, {}".format(shiftLat, shiftLon))
116             centerLat = uavLat + shiftLat
117             centerLon = uavLon + shiftLon
118             print("GEO coords: {}, {}".format(centerLat, centerLon))    # Draw the center of the
119             ↪ area
120             rect_size = 10

```

```

110 p1 = (avg_x - rect_size, avg_y - rect_size)
111 p2 = (avg_x + rect_size, avg_y + rect_size)
112 cv2.rectangle(rawFrame, p1, p2, (0, 0, 0), 2)
113 # Draw lines from orientiers to the center
114 for i in range(0, numCon) :
115     cv2.line(rawFrame, (int(orient_x[i]), int(orient_y[i])), (avg_x, avg_y), (100,
        ↪ 100, 100), 1)
116     cv2.imwrite("sol_3.png", rawFrame)

```

Система оценки

- Оценивается точность определения центра зоны ЧС. Если величина ошибки составляет меньше 50 см, за решение задачи начисляется 23 балла.
- Если величина ошибки лежит в диапазоне 50–200 см — начисляется 20 баллов.
- Если величина ошибки лежит в диапазоне 2–5 м — начисляется 18 баллов.
- Если величина ошибки лежит в диапазоне 5–10 м — начисляется 15 баллов.
- Если величина ошибки лежит в диапазоне 10–25 м — начисляется 10 баллов.
- Если величина ошибки лежит в диапазоне 25–50 м — начисляется 5 баллов.
- Если величина ошибки превышает 50 метров — начисляется 0 баллов.
- Правильный ответ на каждый из двух вопросов по теории прибавляет 2 балла.

Задача IV.4. (40 баллов)

Описание задачи: БПЛА, оснащенный грузом, совершает патрулирование по заданному маршруту. В случае обнаружения зоны чрезвычайной ситуации (ЧС), БПЛА должен автоматически определить местонахождение центра этой зоны, изменить маршрут полета и осуществить сброс груза.

Формулировка задачи: написать программу на языке Python анализирующую изображение с бортовой камеры БПЛА для расчета координат точки сброса груза. Определив местоположение точки сброса, программа должна отправить команду на полетный контроллер для изменения траектории движения БПЛА. При приближении к точке сброса необходимо отправить команду на сброс груза.

Входные данные: снимок бортовой камеры БПЛА; параметры БПЛА на момент осуществления съемки; полетный контроллер с программой автопилота.

Выходные данные: программа реализующая изменение траектории полета БПЛА и сброс груза в центре зоны ЧС. Координаты точки сброса груза должны определяться системой технического зрения с использованием библиотеки OpenCV.

Решение

Для разворота по углу курса используется упрощенная матрица поворота вектора, которую можно записать в скалярном виде следующим образом:

$$X_{\text{вперед}} = X_{\text{север}} \cos \Psi - Y_{\text{восток}} \sin \Psi$$

$$Y_{\text{вправо}} = X_{\text{север}} \sin \Psi + Y_{\text{восток}} \cos \Psi$$

```

1 import cv2 # used for image processing
2 import numpy as np # used for linear algebra math

```

```

3 import math # used for trigonometry functions
4 import time # used to track time, duh
5 from collections import namedtuple # used to conviniently store detection data import
  ↳ serial # used to exchange data with arduino flight controller from threading
  ↳ import Thread # used to work with multiple threads running in parallel
6 # Data structure contains all the detection information
7 Telemetry = namedtuple("Telemetry", "X Y H Pitch Yaw Roll Vx1 Vy1 Vz1 wx wy wz")
8 cameraWidth = 1280
9 cameraHeight = 720
10 ser = serial.Serial(
11     port='/dev/ttyUSB0',
12     baudrate = 115200,
13     parity = serial.PARITY_NONE,
14     stopbits = serial.STOPBITS_ONE,
15     bytesize = serial.EIGHTBITS,
16     timeout = 0.1 )
17 # Extracts a single Telemetry object from the string.
18 # The input string should have 'tele: X Y H Pitch Yaw Roll Vx1 Vy1 Vz1 wx wy wz \r\n'
  ↳ format
19 def parseIncomingMessage(receivedString):
20     data = Telemetry(0,0,0,0,0,0,0,0,0,0,0,0)
21     list = receivedString.split(" ")
22     length = len(list)
23     if length == 14:
24         if list[0] == "tele:":
25             try:
26                 X = float(list[1])
27                 Y = float(list[2])
28                 Z = float(list[3])
29
30                 Pitch = float(list[4])
31                 Yaw = float(list[5])
32                 Roll = float(list[6])
33
34                 Vx = float(list[7])
35                 Vy = float(list[8])
36                 Vz = float(list[9])
37
38                 wx = float(list[10])
39                 wy = float(list[11])
40                 wz = float(list[12])
41                 data = Telemetry(X,Y,Z,Pitch,Yaw,Roll,Vx,Vy,Vz,wx,wy,wz)
42             except ValueError:
43                 print("ValueError occured:")
44                 print(receivedString)
45                 data = Telemetry(0,0,0,0,0,0,0,0,0,0,0,0)
46     return data
47
48 def sendNewWaypoint(newx, newy):
49     ser.write("wpnt {} {} \n".format(newx, newy).encode())
50     print("wpnt {} {} \n".format(newx, newy))
51
52 def sendDropCargo():
53     ser.write("drop \n".encode())
54     print("drop \n")
55 # Текущие Параметры БПЛА
56 uavX = 0 # meters (north)
57 uavY = 0 # meters (east)
58 uavYaw = 0 # degrees
59 H = 120 # meters

```

```

60
61 def telemetryGathering_thread():
62     global ser, uavX, uavY, uavYaw, uavPitch, uavRoll, H    # Main cycle
63     while True:
64         x = ser.readline()
65         msg = parseIncomingMessage(x.decode())
66         # Параметры БПЛА
67         if (msg.H != 0):
68             uavX = msg.X # meters
69             uavY = msg.Y # meters
70             uavYaw = -msg.Yaw # degrees
71             uavPitch = msg.Pitch # degrees
72             uavRoll = msg.Roll # degrees
73             H = msg.H # meters
74
75 def uav_control_thread():
76     global ser, uavX, uavY, uavYaw, uavPitch, uavRoll, H
77     width = 1920
78     height = 1080
79     X = -264
80     Y = 339
81     H = 120
82     Heading = -127
83     #color definition
84     red_lower = np.array([5, 20, 180])
85     red_upper = np.array([17, 80, 255])
86     img = cv2.imread('simulatorCamera.png')
87     dx = 0
88     dy = 0
89     # Normalize image brightness
90     cv2.normalize(img, img, 0, 255, cv2.NORM_MINMAX)    # Blur the image to reduce
91     ↪ noise
92     blur = cv2.medianBlur(img, 5)
93     # Convert BGR to HSV
94     hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
95     # Find color masks
96     red3_mask = cv2.inRange(hsv, red_lower, red_upper)    full_mask = red3_mask
97     # Find counters
98     contours, hierarchy = cv2.findContours(full_mask, cv2.RETR_LIST,
99     ↪ cv2.CHAIN_APPROX_SIMPLE)
100     numCon = 0
101     orient_x = []
102     orient_y = []
103     maxlen = 0
104     coords = [0, 0]
105     if len(contours) > 0:
106         truecont = contours[0];
107         for icontour in contours:
108             if len(icontour) > maxlen:
109                 maxlen = len(icontour)
110                 truecont = icontour
111         if len(truecont) > 300:
112             ellipse = cv2.fitEllipse(truecont)
113             cv2.ellipse(img, ellipse, (0,0,255),2)    sum_x = 0
114             sum_y = 0
115             for i in range(0, len(truecont)):
116                 sum_x += truecont[i][0][0]
117                 sum_y += truecont[i][0][1]
118             avg_x = int(round(sum_x / len(truecont)))    avg_y = int(round(sum_y /
119             ↪ len(truecont)))

```

```

117     print("shift_hor {}".format(avg_x - width/2))    print("shift_ver
        ↳ {}".format(height/2 - avg_y))
118     Mx = (avg_y - height/2) / 11.496
119     My = (avg_x - width/2) / 8.4
120     print("shift in base frame: {}, {}".format(Mx, My))    Mx1 =
        ↳ Mx*math.cos(Heading * math.pi / 180.0) + My*math.sin(Heading *
        ↳ math.pi / 180.0)
121     My1 = -Mx*math.sin(Heading * math.pi / 180.0) + My*math.cos(Heading *
        ↳ math.pi / 180.0)
122     print("Sum shift in geo frame: {}, {}".format(Mx1, My1))    print("Sum
        ↳ coords in geo frame: {}, {}".format(Mx1 + X, My1 + Y))
123     coords[0] = X + Mx1
124     coords[1] = Y + My1
125     rect_size = 5
126     p1 = (avg_x - rect_size, avg_y - rect_size)
127     p2 = (avg_x + rect_size, avg_y + rect_size)
128     cv2.rectangle(img, p1, p2, (0,0,0), 2)
129     cv2.imwrite("sol_3.png", img)
130
131     # found everything
132     sendNewWaypoint(coords[0], coords[1])
133     cargoReady = True
134     while cargoReady:
135         dist = math.sqrt((uavX - coords[0])**2 + (uavY - coords[1])**2)    if dist <
        ↳ 90:
136             #print("Drop cargo!")
137             for i in range(0,3):
138                 sendDropCargo()
139                 cargoReady = False
140
141 if __name__ == '__main__':
142     cv2.setNumThreads(2)
143     control_thr = Thread(target=uav_control_thread, args=[])
144     control_thr.daemon = True
145     control_thr.start()
146     process_thr = Thread(target=telemetryGathering_thread, args=[])
        ↳ process_thr.daemon = True
147     process_thr.start()
148     try:
149         while True:
150             time.sleep(1)
151         except KeyboardInterrupt:
152             print('\ninterrupted!')

```

Система оценки

- Оценивается точность попадания груза в центр зоны ЧС. Если величина ошибки меньше 1 метра, за решение задачи начисляется 26 баллов.
- Если величина ошибки лежит в диапазоне от 1 до 2 м — начисляется 23 балла.
- Если величина ошибки лежит в диапазоне от 2 до 5 м — начисляется 20 баллов.
- Если величина ошибки лежит в диапазоне от 5 до 15 м — начисляется 17 баллов.
- Если величина ошибки лежит в диапазоне от 15 до 30 м — начисляется 15 баллов.
- Если величина ошибки лежит в диапазоне от 30 до 100 м — начисляется 10 баллов.
- Если величина ошибки лежит в диапазоне от 100 до 300 м — начисляется 5

баллов.

- Если величина ошибки больше 300 м, баллы за решение задачи не начисляются.
- Дополнительно 10 баллов начисляется за предоставление графического результата работы системы технического зрения (изображение с координатами точки сброса).
- Правильный ответ на каждый из двух вопросов по теории прибавляет 2 балла.