§4 Заключительный этап: командная часть

В современном мире БПЛА всё более востребованы, как в военной, так и в гражданской сферах. Область применения БПЛА в гражданской сфере очень широка: это аэрофотосъемка, обследование крупных промышленных объектов, сельское хозяйство и многое другое. В связи с этим растут и требования, предъявляемые к БПЛА: это длительность полета, автоматический полет по маршруту, наличие системы предупреждения о столкновении, устойчивый полет при внезапной ветровой нагрузке, возможность распознавания объектов и т.п.

Основной задачей командного этапа инженерной олимпиады было научиться разрабатывать упрощенный режим автоматического полета БПЛА самолетного типа; наличие возможности проверить его работоспособность в реальном полете на аэродроме, а также познакомиться с элементами технического зрения для распознавания объектов с БПЛА на местности.

Для решения этой основной задачи учащимся необходимо было иметь соответствующие знания по физике и информатике, а также научиться (и принимать участие):

• разбираться в устройствах БПЛА самолетного типа, принципах его работы;

• получать и обрабатывать (разрабатывать алгоритмы фильтрации сигналов) реальные сигналы с датчиков углов крена и тангажа, высоты;

• работать с математическими моделями полета БПЛА самолетного типа (для этого им предстояло сначала решить математические задачи по управлению БПЛА, а затем в финале поработать с математической моделью полета БПЛА);

• разбираться в органах управления БПЛА и написать программу запуска и управления электродвигателем;

• поучаствовать в проведении летных испытаний БПЛА самолетного типа;

• понимать основные принципы построения элементов системы технического зрения.

Основной акцент при формировании командной задачи был направлен на четкое разделение основной задачи на подзадачи, при успешном решении которых участники достигали поставленной цели. Такой подход даёт возможность участникам глубоко понять объект исследования и принцип его работы.

Ниже представлены подзадачи заключительного этапа олимпиады.

Продолжительность командной части — 3,5 дня.

Задача 4.1

Условие: Используя электронную печатную плату с установленными датчиками написать программу в среде Arduino получения следующих данных в реальном времени:

- крен

- тангаж

- высота

- вертикальная скорость"

Дополнительно – ответить на дополнительный вопрос.

При наличии времени – выполнить дополнительное задание.

Вариант решения:

Основная часть:

1. вывести формулы определения углов крена и тангажа по известному вектору кажущихся ускорений;
2. подключить библиотеки работы с трехосевым акселерометров в программе;
3. запрограммировать и протестировать чтение вектора кажущихся ускорений с трехосевого акселерометра;
4. написать функцию преобразования приборных осей к связанным и протестировать ее;
5. рассчитать по выведенным в п.1. формулам углы крена и тангажа;
6. запрограммировать вывод результатов в терминальное окно через последовательный порт.

Код программы:

```
#define P0 101325
#define N 10
#define g 9.81
#define s 0.64
#include "MPU9250.h"
#include "MS5611.h"
MPU9250 mpu;
MS5611 ms5;
float rH = 203;
float ffilt(float* a){
  float res = 0;
  for(int i=0; i<N; i++){
    res += a[i];
  }
  res /= N;
  return res;
}
int32_t ifilt(int32_t* a){
  int32_t res = 0;
  for(int i=0; i<N; i++){
    res += a[i];
  }
  res /= N;
  return res;
}
float Height(int32_t p){
    return 44330 *(1 - pow(p*1.0/P0, 1/5.225));
}
void setup() {
  mpu.begin();
  ms5.begin();
  int32_t AP[N];
  for(int i =0; i<N; i++){
    AP[i] = ms5.readPressure();
    delay(10);
    mpu.readSensor();
  }
  int32_t P = ifilt(AP);
  rH = Height(P);
```

```
  Serial.begin(9600);
  while(!Serial){
    ;
  }
  //int32_t P = ms5.readPressure();
  //H0 = Height(P);
}
float T0 = millis();
float V0 = 0;
float H0 = 0;
void loop() {
  float Aax[N];
  float Aay[N];
  float Aaz[N];
  int32_t AP[N];
  for(int i =0; i<N; i++){
    AP[i] = ms5.readPressure();
    mpu.readSensor();
    Aax[i] = -mpu.getAccelX_mss();
    Aay[i] = -mpu.getAccelZ_mss();
    Aaz[i] = -mpu.getAccelY_mss();
    delay(1);
  }
  int32_t P = ifilt(AP);
  mpu.readSensor();
  float ax = ffilt(Aax)+0.11;
  float ay = ffilt(Aay)-0.35;
  float az = ffilt(Aaz)+0.17;
  float tg = atan2(ax, ay)*57;
  float kr = -atan2(az, ay)*57;//kren
  float H = Height(P)- rH;
  float T = millis();
  float dT = T - T0;
  float dH = H-H0;
  float V = (1-s)*V0 + s*(dH)/dT*1000;
  V0 = V;
  T0 = T;
  H0 = H;
   Serial.print(ax);
   Serial.print("\t");
   Serial.print(ay);
   Serial.print("\t");
   Serial.print(az);
   Serial.print("\t\n");
}
```

Задача 4.2

Условие: Подобрать параметры статического и астатического регулятора высоты, так чтобы:

- обеспечить устойчивость замкнутой системы
- обеспечить время переходного процесса по высоте в пределах 15-40 сек
- перерегулирование не более 5 %
- вертикальная скорость не более 5 м/с

Дополнительно – ответить на дополнительный вопрос.

При наличии времени – выполнить дополнительное задание.

Вариант решения:

Основная часть:

1. методом последовательных приближений изменять коэффициенты и параметры ограничений так, чтобы выполнить условие задачи для статического регулятора;
2. методом последовательных приближений изменять коэффициенты и параметры ограничений так, чтобы выполнить условие задачи для астатического регулятора.

- Статический регулятор

KH = 0.6

KVy = 7

Ограничение на вертикальную скорость = +-7

Ограничение на угол тангажа = +-20

- Астатический регулятор

KIH = 0.15

KH = 1.2

KVy = 1.4

Ограничение на ошибку высоты = +-50

Ограничение на угол тангажа = +-20

Задача 4.3

Условие: Запуск электродвигателя на самолете на заданное время (5 секунд)

Дополнительно – ответить на дополнительный вопрос.

При наличии времени – выполнить дополнительное задание.

Вариант решения:

Основная часть:

1. написать подпрограмму запуска двигателя через широтно-импульсно-модулированный (ШИМ) сигнал;
2. написать подпрограмму управления двигателя через ШИМ сигнал;
3. проверить работу программы на осциллографе;
4. протестировать работу программы на электродвигателе.

Код программы:

```
#define pin 5
#define T 20000
void setup() {
  pinMode(pin, OUTPUT);
}
void shim(int t){
  long int t0 = micros();
  digitalWrite(pin, HIGH);
  while (micros() - t0 < t){
    ;
  }
  digitalWrite(pin, LOW);
  while(micros() - t0 < T){
    ;
  }
}
void loop() {
  delay(5000);
  for(int i=0; i<300; i++){
    shim(1000 + i);
  }
  long int t0 = millis();
  while(millis() - t0 < 5000){
    shim(1300 + (millis() - t0)/25);
  }
  shim(1000);
}
```

Задача 4.4

Условие: Написание программы выхода самолета на заданную высоту в реальном полете.

Дополнительно – ответить на дополнительный вопрос.

При наличии времени – выполнить дополнительное задание.

Вариант решения:

Основная часть:

1. в функции с пользовательским названием и стандартным прототипом:
2. запрограммировать расчет барометрической высоты;
3. запрограммировать расчет управляющих воздействий с помощью статического и/или астатического регулятора, подобранного в задаче 2;
4. запрограммировать расчет скважности управляющего ШИМ сигнала;
5. протестировать работу описанной функции на программе моделирования движения самолета.

Код программы:

```
void team_11_1(float &H_cmd_m, float &P_pa, float &P0_pa, float &H_m, float
&Vy_ms, float &theta_cmd_deg, uint16_t &pwm_cmd_ms, tf1011_2 &diff, integral_1
&integrator, float time_from_last_call_s)
{
  H_m = 44330 *(1 - pow(P_pa/P0_pa, 1/5.225));
  Vy_ms = diff.f_update(H_m, time_from_last_call_s);
  theta_cmd_deg = (2*(H_cmd_m-H_m) - Vy_ms)*0.9;

  if(theta_cmd_deg > 20){
    theta_cmd_deg = 20;
  }else if (theta_cmd_deg < -20){
    theta_cmd_deg = -20;
  }
  pwm_cmd_ms = uint16_t(5.0f*theta_cmd_deg) + 1500;
}
void team_11_2(float &H_cmd_m, float &P_pa, float &P0_pa, float &H_m, float
&Vy_ms, float &theta_cmd_deg, uint16_t &pwm_cmd_ms, tf1011_2 &diff, integral_1
&integrator, float time_from_last_call_s)
{
  H_m = 44330 *(1 - pow(P_pa/P0_pa, 1/5.225));
    Vy_ms = diff.f_update(H_m, time_from_last_call_s);
  theta_cmd_deg = (integrator.f_update(0.3*(H_cmd_m-H_m),
time_from_last_call_s)) - 1.41421356237309504880168872420 97*H_m - Vy_ms*1.2;

  if(theta_cmd_deg > 20){
    theta_cmd_deg = 20;
  }else if (theta_cmd_deg < -20){
    theta_cmd_deg = -20;
  }
  pwm_cmd_ms = uint16_t(5.0f*theta_cmd_deg) + 1500;
}
```

Задача 4.5

Условие: Летные испытания.

Решение:

Проверка в реальном полете, как самолет отрабатывает программу из задачи 4.

Основная часть:

- участники, успешно завершившие задачу 4, т.е. более, чем на 1 балл – наблюдение за полетом;
- участники, не завершившие задачу 4 успешно, т.е. не больше, чем на 1 балл – ответ на вопрос по пройденным темам.

Задача 4.6

Условие: Техническое зрение. Поиск определенного объекта по видеофрагменту полета БПЛА.

Дополнительно – ответить на дополнительный вопрос.

Решение:

Основная часть:

- определить назначение данных программ;
- выполнить снимок указанного объекта;
- настроить HSV фильтр;
- продемонстрировать работу алгоритма определения центра искомого объекта на реальном видеопотоке.

- определить назначение трех программ

- сделать с помощью первой программы снимок экрана

Код программы:

```
from picamera import PiCamera
from time import sleep
camera = PiCamera()
camera.start_preview()
sleep(10)
camera.capture('/home/pi/Documents/image.jpg')
camera.stop_preview()
```

- настроить фильтр цвета с помощью второй программы

HSV – Цвет, Насыщенность, Яркость.

Код программы:

Создание окон:

«settings» – шкалы фильтра HSV.

«result» - изображение с корректировками фильтра.

```
import sys
sys.path.append('/usr/local/lib/python3.5/site-packages')
import cv2
import numpy as np
if __name__ == '__main__':
    def nothing(*arg):
      pass
cv2.namedWindow( "result" ) # создаем главное окно
cv2.namedWindow( "settings" ) # создаем окно настроек
# создаем 6 бегунков для настройки начального и конечного цвета фильтра
cv2.createTrackbar('h1', 'settings', 0, 255, nothing)
cv2.createTrackbar('s1', 'settings', 0, 255, nothing)
cv2.createTrackbar('v1', 'settings', 0, 255, nothing)
cv2.createTrackbar('h2', 'settings', 255, 255, nothing)
cv2.createTrackbar('s2', 'settings', 255, 255, nothing)
cv2.createTrackbar('v2', 'settings', 255, 255, nothing)
crange = [0,0,0, 0,0,0]
```

```
while True:
    img = cv2.imread('image.jpg')
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV )
    # считываем значения бегунков
    h1 = cv2.getTrackbarPos('h1', 'settings')
    s1 = cv2.getTrackbarPos('s1', 'settings')
    v1 = cv2.getTrackbarPos('v1', 'settings')
    h2 = cv2.getTrackbarPos('h2', 'settings')
    s2 = cv2.getTrackbarPos('s2', 'settings')
    v2 = cv2.getTrackbarPos('v2', 'settings')
    # формируем начальный и конечный цвет фильтра
    h_min = np.array((h1, s1, v1), np.uint8)
    h_max = np.array((h2, s2, v2), np.uint8)
    # накладываем фильтр на кадр в модели HSV
    thresh = cv2.inRange(hsv, h_min, h_max)
    cv2.imshow('result', thresh)
    ch = cv2.waitKey(5)
    if ch == 27:
        break
cv2.destroyAllWindows()
```

- в третьей программе сделать снимок с использованием полученного фильтра

Код программы:

```
# import the necessary packages
from picamera import PiCamera
import time
import cv2
import numpy as np
# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 50
camera.hflip = True
rawCapture = PiRGBArray(camera, size=(640, 480))
# allow the camera to warmup
time.sleep(0.1)
saved_images_counter = 0
pause_counter = 0
OBJECT_PRESENT_THRESHOLD = 100
# capture frames from the camera
for frame in camera.capture_continuous(rawCapture, format="bgr",
      use_video_port=True):
      # grab the raw NumPy array representing the image, then initialize the
      timestamp
      # and occupied/unoccupied text
        image = frame.array
        blur = cv2.blur(image, (3,3))
        #hsv to complicate things, or stick with BGR
        hsv = cv2.cvtColor(blur,cv2.COLOR_BGR2HSV)
        #thresh = cv2.inRange(hsv,np.array((0, 200, 200)), np.array((20, 255,
255)))
        lower = np.array([52,67,87], dtype="uint8")
        upper = np.array([71,159,179], dtype="uint8")
        thresh = cv2.inRange(hsv, lower, upper)
        thresh2 = thresh.copy()
        # find contours in the threshold image
        image, contours,hierarchy =
cv2.findContours(thresh,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
        # finding contour with maximum area and store it as best_cnt
        max_area = 0
        best_cnt = 1
        for cnt in contours:
```

```
            area = cv2.contourArea(cnt)
            if area > max_area:
                    max_area = area
                    best_cnt = cnt
    # finding centroids of best_cnt and draw a circle there
    M = cv2.moments(best_cnt)
    cx,cy = int(M['m10']/M['m00']), int(M['m01']/M['m00'])
    #if best_cnt>1:
    cv2.circle(blur,(cx,cy),10,(0,0,255),-1)
    # show the frame
    cv2.imshow("Frame", blur)
    #cv2.imshow('thresh',thresh2)
    Dst = cv2.inRange(hsv, lower, upper)
    nb_green = cv2.countNonZero(dst)
    if nb_green > OBJECT_PRESENT_THRESHOLD:
        if pause_counter > 0:
              pause_counter = pause_counter - 1
        else:
              cv2.imwrite( 'found_%04d.png' % saved_images_counter, img )
            pause_counter = 10
            saved_images_counter = saved_images_counter + 1
key = cv2.waitKey(1) & 0xFF
# clear the stream in preparation for the next frame
    rawCapture.truncate(0)
# if the `q` key was pressed, break from the loop
    if key == ord("q"):
              break
```