Младшая категория

Участникам было предоставлено одинаковое оборудование (наборы «ZMROBO Intelligence Storm 2108» с комплектом дополнительных датчиков) и одинаковые мобильные платформы. Но так как перед проведением финала проводились трехдневные учебно-тренировочные сборы, то к началу финала команды располагали значительно отличающимися версиями роботов, с различиями в установке датчиков, схватов и прочей периферии. Вследствие этого единое эталонное решение, гарантировано и одинаково исполняемое на роботах всех команд, подготовить невозможно. Поэтому предлагается декомпозировать задачу на составляющие и в дальнейшем описать решение ключевых ее элементов.

Финальная задача предполагала подготовку командами управляющих алгоритмов, реализующих следующие операции:

1. перемещение робота с по заданным векторам, дугам, линиям;
2. перемещение робота вдоль линий, стен;
3. считывание и распознавание бинарного кода;
4. захват объектов схватом и выгрузка их в определенных зонах.

Рассмотрим решение этих операций подробнее.

Одна из команд разделила свою программу на два файла: библиотеку полезных функций и основной исполняемый файл. Рассмотрим выполнение указанных операций на примере этой библиотеки (язык программирования Python):

(Здесь и далее для наглядности и лучшей структурности в коде участников может быть изменен порядок объявления функций).

(Здесь и далее в код участников могут вноситься незначительные правки, такие как удаление комментариев, команд вывода сообщений на экран, прочих отладочных инструментов).

(Здесь и далее преимущественно сохранены комментарии участников, с незначительными орфографическими правками).

```
import rcu

d_kolesa = 65 #диаметр колеса в миллиметрах
pi = 3.14 #число пи
baza = 135 #база в миллиметрах

# переменные для линии
black = 445 # проверить
white = 900
s2_min = 800
s2_max = 4000
s3_min = 800
s3_max = 4000
err = 0
err_old = 0
gray = 2500
sum_err_old = 0
kp = 0.035 #0.005
kd = 0.025 #0.002
ki = 0.001
s2 = 0
s3 = 0
s = 0

enc_b=0
enc_c=0

x=0
y=0

x3=0
y3=1
x4=2
y4=3
x5=0
y5=1
x6=2
y6=3
# для того чтоб работал wait
wait = rcu.SetWaitForTime

# стоп моторов
def mstop():
    rcu.SetMotor(1,0)
    rcu.SetMotor(2,0)

# пикаем
def beep(time = 0.1):
    rcu.SetInBeep(1)
```

```
    wait(time)
    rcu.SetInBeep(0)

# получить энкодеры мотора
def getTacho(n_port):
    if n_port == 1:
       enc = rcu.GetMotorCode(1)
    elif n_port == 2:
       enc = rcu.GetMotorCode(2)
    return enc

# едем прямо (есть регулятор по энкодерам)
def run_enc(speed=40):
    global sum_err_old, err_old
    if abs(speed)<50:  #здесь надо ставить подобранные kp, kd для медленной скорости (30) и быстрой (90)
        kp=0.5 #1.2
        kd=5  #6
    else:
        kp=0.3 #0.8
        kd=4   #5
    enc1 = getTacho(1)
    enc2 = getTacho(2)
    err = enc1-enc2
    sum_err_old = (0.5*sum_err_old) + err
    u = kp*err +kd*(err_old - err) + ki*sum_err_old
    power_l = speed - u
    power_r = speed + u
    err_old = err
    rcu.SetMotor(1, power_l)
    rcu.SetMotor(2, power_r)

# сброс энкодеров
def prepare_motors():
    rcu.SetMotorCode(1)
    rcu.SetMotorCode(2)

# ехать до чёрной линии
def drive_to_line(speed):
    while normal(3) > black:  # пока датчик видит значение больше черного
        run_enc(speed)  # езда прямо
    mstop()  # стоп моторы

# выводим среднее значение энкодеров
def get_avg_enc():
    global enc_b, enc_c
    enc_b = rcu.GetMotorCode(2)
    enc_c = rcu.GetMotorCode(1)
    enc = (abs(enc_c) + abs(enc_b))/2
    return enc
```

```
99.
100.  # едем заданное кол-во энкодеров БЕЗ линии
101.  def driveEnc(enc,speed=40,stop =1):
102.      global kp,kd,ki
103.      sum_err_old=0
104.      err_old=0
105.      prepare_motors()
106.      a = 0
107.      while a < enc:
108.          run_enc(speed)
109.          a = get_avg_enc()
110.      if stop == True:
111.          mstop()
112.
113.  # едем заданное кол-во сантиметров БЕЗ линии
114.  def driveSm(sm,v=40,stop =1):
115.      global pi
116.      global d_kolesa
117.      mm = sm*10 # Из см в мм
118.      enc = mm*(360/(pi*d_kolesa)) # пересчёт из мм в градусы
119.      driveEnc(enc,v,stop)
120.
121.  # едем до препятствия
122.  def drive_to_uz(sm):
123.      while sm < uz():
124.          run_enc(40)
125.      mstop()
126.
127.  # едем вдоль препятствия и узнаём его размер
128.  def size_uz():
129.      distance = 5
130.      size = 0
131.      while True:
132.          run_enc(40)
133.          if distance > uz():
134.              mstop()
135.              wait(1)
136.              prepare_motors()
137.              run_enc(40)
138.              #сделал чтоб видеть где он нашёл край объекта
139.              while uz() < 10:
140.                  enc = get_avg_enc()
141.                  size = (enc/360)*(pi*d_kolesa) # ДОЛЖЕН ВЫДАТЬ РАЗМЕР ПРЕПЯТСТВИЯ В МИЛЛИМЕТРАХ
142.                  wait(0.1)
143.              mstop()
144.              rcu.SetDisplayString(5,str(size), 0xFFFFF, 0x00000)
145.              return size
146.
147.  # нормализация показания датчика
```

```
def normal(s_number):
    global s2_max
    global s2_min
    global s3_max
    global s3_min
    if s_number == 2:
        Snorm = 1000 * (rcu.GetLightSensor(2) - s2_min) / (s2_max - s2_min) #читаем датчик 2 и приводим к шкале 0..100
    elif s_number ==3:
        Snorm = 1000 * (rcu.GetLightSensor(3) - s3_min) / (s3_max - s3_min) #читаем датчик 3 и приводим к шкале 0..100
    return int(Snorm)

# езда по двум датчикам
def line(v=40):
    global kp,kd,ki
    global sum_err_old, err_old, err
    global s2, s3
    s2 = normal(2)
    s3 = normal(3)
    err = s3-s2
    sum_err_old = (0.5*sum_err_old) + err
    u = kp*err +kd*(err_old - err) + ki*sum_err_old
    power_l = v - u
    power_r = v + u
    err_old = err
    rcu.SetMotor(2, power_l)
    rcu.SetMotor(1, power_r)

# езда по одному датчику
def line_s1(v=40):
    global kp,kd,ki
    global s2 , s3, err, err_old, gray
    s2 = normal(2)
    s3 = gray
    err = s3-s2
    u = kp*err
    power_l = v - u
    power_r = v + u
    err_old = err
    rcu.SetMotor(2, power_l)
    rcu.SetMotor(1, power_r)

# езда по линии по энкодерам
def lf_enc(enc,speed=40,stop=1):
    global sum_err_old, err_old
    err_old=0
    sum_err_old=0
    prepare_motors()
    a=0
    while enc>a:
```

```
            line(speed)
            a = get_avg_enc()
        if stop == True:
            mstop()

    # езда по энкодерам на одном датчике
    def lf_enc_s1(enc,speed=40):
        global sum_err_old, err_old
        err_old=0
        sum_err_old=0
        prepare_motors()
        a=0
        while enc>a:
            line_s1(speed)
            a = get_avg_enc()

    # едем по линии определённое кол-во перекрёстков
    def lf_cross(n_cross,v=40,stop=1):
        global sum_err_old, err_old,s2,s3,black
        err_old=0
        sum_err_old = s3+s2
        per = 0
        while n_cross > per:
            while normal(2)>black and normal(3)>black :
                line(v)
            mstop()
            per+=1
            beep()
            lf_enc(100,v)
        if stop==True:
            mstop()

    # едем по линии определённое кол-во перекрёстков одним датчиком
    def lf_cross_s1(n_cross,v=40,stop=40):
        global sum_err_old, err_old
        err_old=0
        sum_err_old = s3+s2
        per = 0
        while n_cross > per:
            while (sum_err_old-(s2+s3))<7:
                sum_err_old = s3+s2
                line_s1(v)
            per+=1
            rcu.SetInBeep(1)
            wait(0.1)
            rcu.SetInBeep(0)
            lf_enc_s1(120,v)
        if stop==True:
            mstop()
```

```
246.
247.  # ехать без линии по энкодерам ДУГОЙ
248.  # deg - сколько всего проехать от 0...+10 000,
249.  # vb -скорость B; vc -скорость C; от -100...+100
250.  # start_distance, stop_distance - расстояния разгона/замедления; stop - тип тормоза жесткая (1), простая (2), не выключать моторы (0)
251.  def drive_arc(deg, vb, vc, start_distance, stop_distance, stop):
252.      global enc_b,enc_c
253.      cur_enc = 0
254.      err_old = 0
255.      err = 0
256.      upr=0
257.      prepare_motors()
258.      kb_r = abs(vb)-abs(vb-1) # определяем знак для мотора B - +1 или -1 ==koef_B_reverse
259.      kc_r = abs(vc)-abs(vc-1) # определяем знак для мотора C - +1 или -1 ==koef_C_reverse
260.      if abs(vb)>13:
261.          rcu.SetMotor(2,0) # типа сет повер)
262.      else:
263.          rcu.SetMotor(2,0)
264.          kb_r =0
265.      if abs(vc)>13 :
266.          rcu.SetMotor(1,0)
267.      else:
268.          rcu.SetMotor(1,0)
269.          kc_r=0
270.      if vb<50:  # здесь надо ставить подобранные kp, kd для медленной скорости (30) и быстрой (90)
271.          kp= 0.45 #0.45
272.          kd= 0 #0
273.      else:
274.          kp= 0.45 #0.45
275.          kd= 0 #0
276.      rat = abs(vb/vc) # RATIO=соотношение скоростей по модулю, такое же соотношение должно быть между проездом энкодеров моторов = наша цель
277.      if start_distance>0 and deg>start_distance:  # плавный разгон
278.          while cur_enc < start_distance:
279.              cur_enc = get_avg_enc() #взять средний энкодер с моторов AB и каждый из них
280.              vb_temp = (25*kb_r)+cur_enc/start_distance*(vb-(25*kb_r))
281.              vc_temp = (25*kc_r)+cur_enc/start_distance*(vc-(25*kc_r))
282.              if vc != 0 :
283.                  err = enc_b*kb_r-enc_c*rat*kc_r #ошибка это энкодерB*знакB минус энкодерC*знакC*соотношение. например для дуги вперед вправо va=60 vb=30. за счет rat показания энкодера домножаться на 2 и будут как бы равны энкодеру мотора которые едет быстрее и длиннее. множители знаков нужны для случаев когда надо крутиться вокруг оси. наример -40,+40 для разворота на месте
284.                  upr = kp*err+kd*(err-err_old)
285.              else:
286.                  upr=0
287.              rcu.SetMotor(2,vb_temp-upr*kb_r/2*rat) #регулируем моторB с учетом его инверсии *(-1) и с учетом его знака *kb_r
288.              rcu.SetMotor(1,vc_temp+upr*kc_r/2) #регулируем моторC с учетом его знака *kc_r
```

```
289.              err_old = err
290.      while cur_enc <= (deg-stop_distance): # основной проезд
291.          cur_enc = get_avg_enc() #взять средний энкодер с моторов АВ и каждый из них
292.          if vc != 0:
293.              err = enc_b*kb_r-enc_c*rat*kc_r #ошибка это энкодерВ*знакВ минус энкодерС*знакС*соотношение. например для дуги вперед вправо va=60 vb=30. за счет rat показания энкодера домножаться на 2 и будут как бы равны энкодеру мотора которые едет быстрее и длиннее. множители знаков нужны для случаев когда надо крутиться вокруг оси. наример -40,+40 для разворота на месте
294.              upr = kp*err+kd*(err-err_old)
295.          else:
296.              upr=0
297.          rcu.SetMotor(2,vb-upr*kb_r*rat) #регулируем моторВ с учетом его инверсии *(-1) и с учетом его знака *kb_r
298.          rcu.SetMotor(1,vc+upr*kc_r) #регулируем моторС с учетом его знака *kc_r
299.          err_old = err
300.      if stop_distance>0:  # плавный стоп
301.          while cur_enc < deg:
302.              cur_enc = get_avg_enc() #взять средний энкодер с моторов ВС и каждый из них
303.              temp_enc=stop_distance-(deg-cur_enc) #сколько проехали за тормозной путь
304.              vb_temp = vb-temp_enc/stop_distance*(vb-(25*kb_r))
305.              vc_temp = vc-temp_enc/stop_distance*(vc-(25*kc_r))
306.              if vc != 0:
307.                  rat = abs(vb/vc) #соотношение скоростей по модулю, такое же сооношение должно быть между проездом энкодеров моторов = наша цель
308.                  err = enc_b*kb_r-enc_c*rat*kc_r #ошибка это энкодерВ*знакВ минус энкодерС*знакС*соотношение. например для дуги вперед вправо va=60 vb=30. за счет rat показания энкодера домножаться на 2 и будут как бы равны энкодеру мотора которые едет быстрее и длиннее. множители знаков нужны для случаев когда надо крутиться вокруг оси. наример -40,+40 для разворота на месте
309.                  upr = kp*err+kd*(err-err_old)
310.              else:
311.                  upr=0
312.                  rcu.SetMotor(2,(vb_temp-upr*kb_r/2*rat)) #регулируем моторВ с учетом его инверсии *(-1) и с учетом его знака *kb_r
313.                  rcu.SetMotor(1,(vc_temp+upr*kc_r/2)) #регулируем моторС с учетом его знака *kc_r
314.                  err_old = err
315.      if stop==True: # торможение, если надо
316.          mstop()
317.
318.  # ехать без линии Дугу;
319.  # mode - b+, b-, c+, c- в сторону какого мотора дуга и вперед или назад;
320.  # R- радиус дуги в см;   deg - сколько градусов ДУГИ проехать;
321.  # stop - тип тормоза жесткий (1), не выключать моторы (0)
322.  def drive_radius(mode, R, deg, stop=1):
323.      global baza, d_kolesa
324.      deg=abs(deg)
325.      R=R*10 #переводим из см в мм
326.      k_dugi=(R+baza/2)/(R-baza/2)
327.      motor_deg=R*deg*2/d_kolesa*1.05 #1.05 подобранный коэф для полного ровного круга 360 гр радиусом 30см
328.      if mode=="b+":
329.          #дуга налево вперед
```

```
            vb=50
            vc=vb*k_dugi
            drive_arc(motor_deg,vb,vc,150,150,stop)
        elif mode=="b-":
            #дуга налево вперед
            vb=-50
            vc=vb*k_dugi
            drive_arc(motor_deg*1.05,vb,vc,150,150,stop)
        elif mode=="c+":
            #дуга направо вперед
            vc=50
            vb=vc*k_dugi
            drive_arc(motor_deg,vb,vc,150,150,stop)
        elif mode=="c-":
            #дуга направо назад
            vc=-50
            vb=vc*k_dugi
            drive_arc(motor_deg*1.05,vb,vc,150,150,stop) #1.06 подобранный коэф для полного ровного круга 360 гр радиусом
    30см

    # считываем показания с уз
    def uz(port = 1):
        distance =  rcu.GetUltrasound(port)
        return distance

    # едем вдоль стенки и читаем код(близко - 0; далеко - 1)
    def bin_uz(count):
        global x3,y3,x4,y4,x5,y5,x6,y6
        dist = 510
        n = 1
        dist_uz = 0
        number = ''
        otv = ''
        lf_enc(190,40,1)
        while n<=count:
            mstop()
            wait(0.2)
            dist_uz = uz()
            if dist_uz < 18:
                number += '1'
            else:
                number += '0'
            if n%2==0:
                #number = number[::-1] #переворачиваем массив, если так надо
                beep()
                rcu.SetDisplayString(n//2,number, 0xFFFFF, 0x00000)
                otv = int(number,2)
                if n//2==1:
                    x3 = otv
```

```
            elif n//2==2:
                y3 = otv+1
            elif n//2==3:
                x4 = otv+2
            elif n//2==4:
                y4 = otv+3
            otv=''
            number=''
        if n<7: # вынесено в отдельное условие тк надо проехать всего n-1 раз а цикл крутится n раз
            lf_enc(dist,40,1)
        if n==7:
            lf_enc(dist-50,40,1)
        n+=1
    x5 = y3
    y5 = x3
    x6 = y4
    y6 = x4
    beep()
    wait(6)

# поворот двумя колесами по градусам РОБОТА
# mode - "2w" колеса, "1w_b" колесо B, "1w_c" колесо C, ; degrees - сколько градусов РОБОТА повернуть (-360...+360);
# v - скорость поворота (20...90); stop - остановка жесткая (1), простая (2), не выключать моторы (0)
def turn_deg( mode, degrees , v , stop):
    global baza, d_kolesa
    side=degrees/abs(degrees) #в какую сторону крутить +1 по часовой, -1 против часовой
    degrees=abs(degrees)
    v=abs(v)
    if mode == "2w":
        motor_deg=(baza*degrees)/d_kolesa #расчитываем сколько крутить мотор без поиска линии
        drive_arc(motor_deg,v*side,-1*v*side,0,0,1) #поворот
        beep()
    elif mode=="1w_b" or mode=="1w_c":
        motor_deg=(2*baza*degrees)/d_kolesa #расчитываем сколько крутить мотор без поиска линии
        motor_deg/=2 #делим на 2 так как средний энкодер при стоячем двигателе больше в 2 раза чем энкодер мотора
    if mode=="1w_b":
        drive_arc(motor_deg,v*side,0,150,150,stop) #поворот
    elif mode=="1w_c":
        drive_arc(motor_deg, 0, v*side, 150, 150,stop) #поворот

def turn2wL (degrees, v, stop):
    global baza, s, black, enc_c, enc_b, white
    side=degrees/abs(degrees) #в какую сторону крутить +1 по часовой, -1 против часовой
    v=abs(v)
    motor_deg=baza*abs(degrees)/d_kolesa #расчитываем сколько крутить мотор без поиска линии
    drive_arc(motor_deg,v*side,-1*v*side,100,0,0) #съехать с черной линии, поворот на месте
    beep()
    sens=(side+1)/2+2 #вычисляем нужный датчик +1=3;-1=2
    s = normal(sens) #измеряет показания датчика и приводит шкалу к 0...+100, результат в переменной @s -ГЛОБАЛЬНОЙ
```

```
    err_old = 0
    err = 0
    if v<50: #здесь надо ставить подобранные kp, kd для медленной скорости (30) и быстрой (90)
        kp=0.45
        kd=1
    else:
        kp=0.45
        kd=1
    while s>black*1.5: #ищем линию внешним датчиком
        cur_enc=get_avg_enc() #взять средний энкодер с моторов ВС и каждый из них
        s = normal(sens)
        err =  abs(enc_b)-abs(enc_c)  #расчет ошибки, направление не важно, главное по модулю одинаково чтобы было
        upr = kp * err + kd * (err - err_old) #управляющее воздействие
        rcu.SetMotor(2,(v-upr)*side) #МоторВ перевернут поэтому домножаем на (-1),
        rcu.SetMotor(1,(v+upr)*side*(-1)) #тут *(-1) для вращения по часовой при side=1
        err_old = err
    while s<white*0.7: #ищем линию внешним датчиком
        cur_enc = get_avg_enc() #взять средний энкодер с моторов ВС и каждый из них
        s = normal(sens)
        err =  abs(enc_b)-abs(enc_c)   #расчет ошибки, направление не важно, главное по модулю одинаково чтобы было
        upr = kp * err + kd * (err - err_old) #управляющее воздействие
        rcu.SetMotor(2,(v-upr)*side) #МоторВ перевернут поэтому домножаем на (-1),
        rcu.SetMotor(1,(v+upr)*side*(-1)) #тут *(-1) для вращения по часовой при side=1
        err_old = err
    if stop == True: #торможение, если надо
        mstop()

def navigator(x1,y1):
    proesd = 15
    driveSm(proesd*y,40,1)
    if x<3:
        turn_deg('2w',-90,40,1)
    elif y==2:
        a=a
    else:
        turn_deg('2w',90,40,1)
    driveSm(proesd*x,40,1)
```

В строке 1 производится подключение библиотеки для работы с оборудованием – контроллером робота и периферией.

В зависимости от конструкции робота подбираются и вычисляются различные коэффициенты, например, коэффициенты ПИД-регуляторов. В некоторых местах программы участники оставили поясняющие комментарии, в которых отражена необходимость подбора коэффициентов даже для движения с разными скоростями.

Функции *drive_arc()* и *drive_radius()* позволяют перемещать робота по дуге на заданное расстояние или на заданный сектор дуги. Совместно с функциями *turn_deg()*, *turn2wL()* и *driveEnc()* они позволяют реализовать проезд к выбранному сектору зоны выгрузки. Диаметры дуг, векторную меру секторов и требуемые к проезду тики энкодеров подбираются экспериментальным путем исходя из конструкции робота.

Имея подобную библиотеку функций, можно собрать программу для выполнения итогового задания.