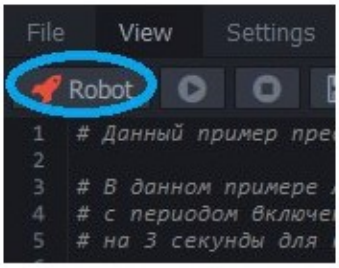
Второй отборочный этап

Индивидуальная часть

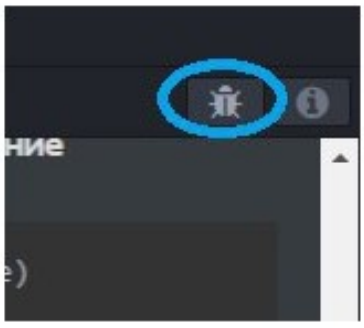
Индивидуальные задания 1 уровень сложности

Для выполнения задания вам необходимо:

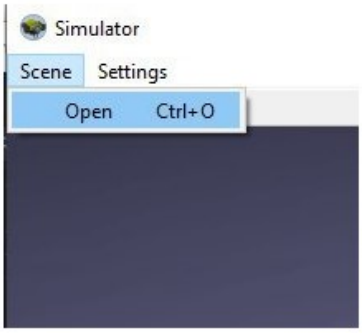
1. Скачать и установить MUR IDE: `https://murproject.com/documents/17/murIDE.exe`.
2. После установки на рабочем столе должен появиться ярлык MUR IDE. Запустите MUR IDE.
3. Для начала работы в симуляторе вам необходимо перевести режим работы IDE в Local. Для этого нажмите на кнопку с иконкой ракеты и надписью Remote в левом верхнем углу. Цвет кнопки станет синим, и надпись изменится на Local.



4. Далее вам необходимо запустить симулятор нажатием кнопки с изображением жука в правом верхнем углу. Откроется окно симулятора с черным экраном.



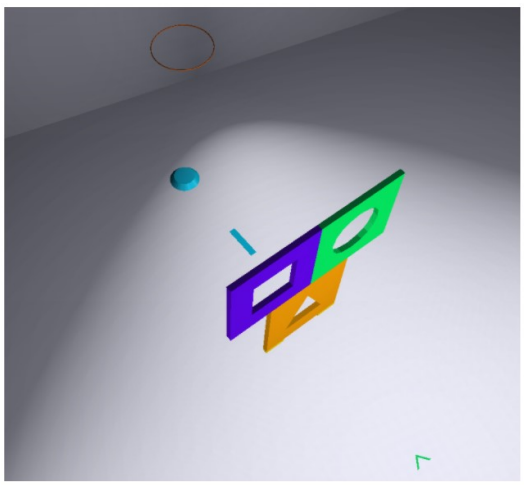
5. Скачать сцену по ссылке. Можно разместить ее в любом удобном каталоге.
6. В запущенном симуляторе переходим в меню Scene → Open. В появившемся диалоговом окне выбрать сцену «NTIx_x-x.xml» по адресу `%Путьдосцены%\NTIx_x-x.xml`.



7. Все готово! Вы можете начать программировать виртуальный аппарат на языке программирования Python.
8. В качестве решения задачи вам необходимо отправить файл с кодом в формате *.py. Для этого в меню выберите File → File save as.

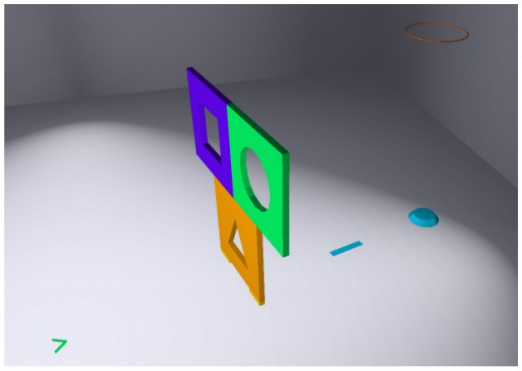
Задача II.1.1.1. Задание 1 для программистов на MUR IDE (100 баллов)

1. Задача выполняется в симуляторе и среде программирования MUR IDE. Сцены называются: NTI-Task-1_1, NTI-Task-1_2 и NTI-Task-1_3. Они доступны по ссылке (http://robocenter.net/media/nti-tasks.zip).
2. Необходимо запрограммировать подводного робота, который должен в симуляторе в автономном режиме выполнить под водой ряд задач и всплыть в заданной области.
   Время на выполнение задачи — 5 минут.
3. Как только робот всплыл или закончилось время, задача считается законченной, фиксируется количество заработанных баллов.



4. Задание:
   - Необходимо определить цвет указателя, над которым стартует робот.
   - Далее нужно последовательно запустить по одной торпеде в каждую из двух мишеней, цвет которых не совпадает с цветом указателя.
   - После этого робот должен проплыть через мишень того же цвета, что и указатель.
   - Затем проследовать по направляющей полоске голубого цвета.

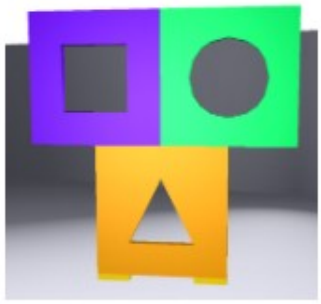
- Над круглой голубой платформой расположен обруч, в пределах которого нужно всплыть на поверхность.



5. Баллы начисляются следующим образом:
   - Попадание торпедой в правильную мишень (цвет которой не совпадает с указателем) — 25 баллов (максимум 50 баллов).
   - Попадание торпедой в неправильную мишень (цвет которой совпадает с указателем) — 10 баллов.
   - Проплыть через мишень, цвет которой совпадает с указателем — 30 баллов.
   - Проплыть через мишень, цвет которой не совпадает с указателем — 10 баллов.
   - Всплыть в обруче — 20 баллов.

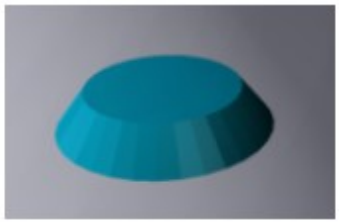
   Итого за задачу можно заработать 100 баллов.
   Штрафы:
   - Касание роботом границ мишени при прохождении через нее — минус 10 баллов.
6. Описание макетов:
   В задаче используются следующие виды объектов: указатель, стенд с мишенями, полоска, платформа, обруч.

   - Указатель может быть окрашен в один из трех цветов — синий, зеленый и оранжевый. Указатель располагается под стартовой позицией робота и указывает направление на стенд с мишенями. Цвет указателя говорит о том, через мишень какого цвета необходимо проплыть.



- Стенд с мишенями представляет собой стенд, состоящий из трех мишеней разных форм (квадрат, круг и треугольник) и цветов (синий, зеленый и оранжевый). Формы и цвета мишеней могут различаться в разных сценах. Необходимо будет проплыть в ту мишень, цвет которой совпадает с цветом указателя на стартовой позиции, перед этим выстрелив в две другие мишени. Мишень, через которую необходимо пройти, будет либо квадратом, либо кругом (не треугольником).

- Полоска (голубого цвета) указывает направление к платформе. По данной полоске необходимо будет скорректировать направление к платформе.



- Платформа (голубого цвета) служит для определения расположения обруча.

- Обруч расположен на поверхности воды и находится над голубой платформой. Аппарат должен всплыть внутри обруча.

7. Задача будет проверяться на 3 сценах. Сцены при проверке будут отличаться от сцен, доступных для написания программы. Сцены будут отличаться друг от друга цветами мишеней, цветом указателя, направлением указателя, расположением стенда с мишенями (на которые направлен указатель), расположением полоски голубого цвета, расположением голубой платформы и обруча (на которые указывает голубая полоска). В зачет попытки идет худший результат из трех сцен. При старте задачи робот располагается над указателем.
8. В качестве решения задачи вам необходимо прикрепить файл с кодом в формате *.py.

Решение

Решение (NTI-Simulator-Task1.py) доступно по ссылке: https://yadi.sk/d/KrZAbhWM9nDhAw.

Для успешного выполнения задания аппарату в виртуальной среде MUR IDE необходимо последовательно запустить по одной торпеде в каждую из двух мишеней, цвет которых не совпадает с цветом указателя на старте, после этого робот должен проплыть через мишень того же цвета, что и указатель. Затем проследовать по направляющей полоске голубого цвета. Над круглой голубой платформой расположен обруч, в пределах которого нужно всплыть на поверхность.

Приступим к написанию программы. Для начала необходимо импортировать требуемые библиотеки и инициализировать API симулятора.

```
import pymurapi as api
import cv2 as cv
import time
import numpy as np
import math

mur = api.mur_init()
```

Согласно заданию подберем диапазоны возможных цветов, которые встречаются в задании.

```
# предопределенные диапазоны цветов (HSV)
colors = {
    'green':  ((45, 50, 50), (75, 255, 255)),
    'blue':   ((130, 50, 50), (140, 255, 255)),
    'orange': ((10, 50, 50), (30, 255, 255))
}
```

Также напишем функции, которые пригодятся для дальнейших расчетов.

```
# функция для ограничения значения диапазоном
def clamp(value, min_value, max_value):
    if value < min_value:
        return min_value
    if value > max_value:
        return max_value
```

```
    return value

# функция для вычисления угла между точками
def angle_between(p1, p2):
    xDiff = p2[0] - p1[0]
    yDiff = p2[1] - p1[1]
    return math.degrees(math.atan2(yDiff, xDiff) - (np.pi / 2))

# функция для вычисления расстояния от начала координат
def length_from_center(x, y):
    return math.sqrt(x ** 2 + y ** 2)
```

Далее разработаем PD-регулятор, который будет использоваться для стабилизации курса, глубины и выравнивания над объектами.

```
# PD-регулятор
class PDRegulator(object):
    _p_gain = 0.0
    _d_gain = 0.0
    _prev_error = 0.0
    _timestamp = 0

    def __init__(self):
        pass

    def set_p_gain(self, value):
        self._p_gain = value

    def set_d_gain(self, value):
        self._d_gain = value

    def process(self, error):
        timestamp = int(round(time.time() * 1000))
        if timestamp == self._timestamp:
            return 0

        output = self._p_gain * error + self._d_gain / (timestamp - self._timestamp) *
        ↪  (error - self._prev_error)
        self._timestamp = timestamp
        self._prev_error = error
        return output
```

С использованием PD-регулятора опишем функции для регулирования курса и глубины робота.

```
# функция для установки курса робота
def keep_yaw(yaw_to_set, speed):
    def clamp_angle(angle):
        if angle > 180.0:
            return angle - 360.0
        if angle < -180.0:
            return angle + 360
        return angle

    try:
        error = mur.get_yaw() - yaw_to_set
        error = clamp_angle(error)
        output = keep_yaw.yaw_regulator.process(error)
```

```
        mur.set_motor_power(0, clamp(-output + speed, -100, 100))
        mur.set_motor_power(1, clamp(output + speed, -100, 100))
    except AttributeError:
        # создание PD-регулятора, если он отсутствует
        keep_yaw.yaw_regulator = PDRegulator()
        keep_yaw.yaw_regulator.set_p_gain(0.8)
        keep_yaw.yaw_regulator.set_d_gain(0.6)

# функция для установки глубины погружения
def keep_depth(depth_to_set):
    try:
        error = mur.get_depth() - depth_to_set
        output = keep_depth.depth_regulator.process(error)
        output = clamp(output, -100, 100)
        mur.set_motor_power(2, output)
        mur.set_motor_power(3, output)
    except AttributeError:
        keep_depth.depth_regulator = PDRegulator()
        keep_depth.depth_regulator.set_p_gain(45)
        keep_depth.depth_regulator.set_d_gain(5)
```

Напишем класс для хранения состояния робота и хода выполнения миссии.

```
# класс для хранения текущего состояния
class AUVContext(object):
    _yaw = 0.0
    _depth = 0.0
    _speed = 0.0
    _side_speed = 0.0
    _timestamp = 0
    _missions = []
    _min_area = math.inf
    _min_yaw = 0.0
    _stabilization_counter = 0
    time

    def __init__(self):
        pass

    def set_min_circle(self, yaw, area):
        if area < self._min_area:
            self._min_area = area
            self._min_yaw = yaw

    def get_min_circle_yaw(self):
        return self._min_yaw

    def get_yaw(self):
        return self._yaw

    def get_depth(self):
        return self._depth

    def get_speed(self):
        return self._speed

    def get_side_speed(self):
        return self._side_speed

    def set_yaw(self, value):
```

```
38          self._yaw = value
39
40      def set_depth(self, value):
41          self._depth = value
42
43      def set_speed(self, value):
44          self._speed = value
45
46      def set_side_speed(self, value):
47          self._side_speed = value
48
49      def get_stabilization_counter(self):
50          return self._stabilization_counter
51
52      def reset_stabilization_counter(self):
53          self._stabilization_counter = 0
54
55      def add_stabilization_counter(self):
56          self._stabilization_counter += 1
57
58      def check_stabilization(self, timeout = 3):
59          if self._stabilization_counter > timeout:
60              return True
61          else:
62              self.add_stabilization_counter()
63              return False
64
65      def push_mission(self, mission):
66          self._missions.append(mission)
67
68      def push_mission_list(self, missions):
69          for mission in missions:
70              self.push_mission(mission)
71
72      def pop_mission(self):
73          if len(self._missions) != 0:
74              return self._missions.pop(0)
75          return {}
76
77      def get_missions_length(self):
78          return len(self._missions)
79
80      def process(self):
81          timestamp = int(round(time.time() * 1000))
82          if timestamp - self._timestamp > 16:
83              keep_yaw(self._yaw, self._speed)
84              keep_depth(self._depth)
85              mur.set_motor_power(4, self._side_speed)
86              self._timestamp = timestamp
87          else:
88              time.sleep(0.05)
89
90  # объект, где будет хранится текущее состояние
91  context = AUVContext()
```

Напишем еще несколько полезных функций, которые позволяют выполнять выполнять различные действия, такие как развороты, выстрел и т. д.

```
1   # разворот на 90 градусов
2   def translate_to_90():
```

```
    yaw = mur.get_yaw() + 90
    if yaw < -180:
        yaw += 360
    if yaw > 180:
        yaw -= 360
    context.set_yaw(yaw)
    return True

# разворот на 180 градусов
def translate_to_180():
    yaw = mur.get_yaw() + 180
    if yaw < -180:
        yaw += 360
    if yaw > 180:
        yaw -= 360
    context.set_yaw(yaw)
    return True

# произвести выстрел
def shoot():
    if (context.get_speed() != 0):
        context.set_speed(0)
        return False
    else:
        mur.shoot()
        time.sleep(0.5)
        return True

# стабилизировать курс и глубину
def stabilize():
    yaw = mur.get_yaw()
    depth = mur.get_depth()

    if abs(yaw - context.get_yaw()) < 1 and abs(depth - context.get_depth()) < 0.3:
        if context.check_stabilization():
            return True
    else:
        context.reset_stabilization_counter()
    return False
```

В дальнейшем часто будет возникать задача по распознаванию контуров цветных объектов, для этого напишем соответствующую функцию.

```
# поиск на изображении контура по цвету
def find_contours(image, color_low, color_high, approx = cv.CHAIN_APPROX_SIMPLE):
    hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
    mask = cv.inRange(hsv_image, color_low, color_high)
    contours, _ = cv.findContours(mask, cv.RETR_EXTERNAL, approx)
    return contours
```

Далее идут функции, активно использующие машинное зрение. Это такие действия, как расчеты координат и углов объектов, определение цветов объектов, позиционирование и стабилизация над объектами. Для стабилизации также используется ранее написанный PD-регулятор.

```
# расчет угла прямоугольника
# для определение отклонения от полоски,
# чтобы затем скорректировать курс
```

```
4  def find_rectangle_contour_angle(contour):
5      rectangle = cv.minAreaRect(contour)
6      box = cv.boxPoints(rectangle)
7      box = np.int0(box)
8      edge_first = np.int0((box[1][0] - box[0][0], box[1][1] - box[0][1]))
9      edge_second = np.int0((box[2][0] - box[1][0], box[2][1] - box[1][1]))
10
11     edge = edge_first
12     if cv.norm(edge_second) > cv.norm(edge_first):
13         edge = edge_second
14
15     angle = -((180.0 / math.pi * math.acos(edge[0] / (cv.norm((1, 0)) *
       ↪  cv.norm(edge)))) - 90)
16     return angle
17
18 # стабилизироваться над желтым прямоугольником
19 def stabilize_on_yellow_rectangle(x, y):
20     y_center = y - (240 / 2)
21     x_center = x - (320 / 2)
22     try:
23         # определеяем и проверяем отклонение
24         length = length_from_center(x_center, y_center)
25         if length < 4.5:
26             if context.check_stabilization():
27                 return True
28         else:
29             context.reset_stabilization_counter()
30
31         output_forward =
           ↪  stabilize_on_yellow_rectangle.forward_regulator.process(y_center)
32         output_side = stabilize_on_yellow_rectangle.side_regulator.process(x_center)
33
34         output_forward = clamp(int(output_forward), -50, 50)
35         output_side = clamp(int(output_side), -50, 50)
36
37         context.set_speed(-output_forward)
38         context.set_side_speed(-output_side)
39
40     except AttributeError:
41         stabilize_on_yellow_rectangle.forward_regulator = PDRegulator()
42         stabilize_on_yellow_rectangle.forward_regulator.set_p_gain(0.5)
43         stabilize_on_yellow_rectangle.forward_regulator.set_d_gain(0.1)
44
45         stabilize_on_yellow_rectangle.side_regulator = PDRegulator()
46         stabilize_on_yellow_rectangle.side_regulator.set_p_gain(0.5)
47         stabilize_on_yellow_rectangle.side_regulator.set_d_gain(0.1)
48     return False
49
50 # определение цвета стрелки
51 def detect_arrow_color():
52     image = mur.get_image_bottom()
53     hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
54
55     # для определения цвета, выделим контуры для
56     # каждого из возможных цветов, а затем
57     # подсчитаем площадь каждого контура.
58     # цвет стрелки - это контура с наибольшей площадью.
59
60     areas = {}
61
```

```
62      for color in colors:
63          mask = cv.inRange(hsv_image, colors[color][0], colors[color][1])
64          contours, _ = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
65
66          biggest_area = 0
67
68          if contours:
69              for contour in contours:
70                  area = cv.contourArea(contour)
71                  if area > biggest_area:
72                      biggest_area = area
73
74          areas[color] = biggest_area
75
76      color = sorted(areas, key=areas.get, reverse=True)[0]
77
78      if (areas[color] > 5):
79          # устанавливаем цвет мишени, через которую нужно пройти
80          context.target_color = color
81          # а также создаем список мишеней, в которые нужно выстрелить
82          context.shooting_colors = list(set(colors.keys()) - {color})
83          return True
84      else:
85          return False
86
87  # определение угла, куда направлена стрелка
88  def detect_arrow_angle(image, contour):
89      (arrow_center_x, arrow_center_y), radius = cv.minEnclosingCircle(contour)
90      moments = cv.moments(contour)
91
92      try:
93          arrow_direction_x = int(moments['m10'] / moments['m00'])
94          arrow_direction_y = int(moments['m01'] / moments['m00'])
95
96          arrow_angle = (angle_between((arrow_direction_x, arrow_direction_y),
            ↪  (arrow_center_x, arrow_center_y)))
97          context.set_yaw(mur.get_yaw() + arrow_angle)
98
99          target_x = arrow_center_x - (320 / 2)
100         target_y = arrow_center_y - (240 / 2)
101         length = math.sqrt(target_x ** 2 + target_y ** 2)
102
103         return arrow_angle, target_x, target_y, length
104     except:
105         return False, False, False, False
106
107 # стабилизироваться над стрелкой (с учетом направления стрелки)
108 def stabilize_on_arrow():
109     image = mur.get_image_bottom()
110
111     contours = find_contours(image, colors[context.target_color][0],
        ↪  colors[context.target_color][1], cv.CHAIN_APPROX_SIMPLE)
112
113     if contours:
114         for contour in contours:
115             (arrow_angle, target_x, target_y, length) = detect_arrow_angle(image,
                ↪  contour)
116
117             if arrow_angle != False:
118                 try:
```

```
119                     output_forward =
                        ↪  stabilize_on_arrow.forward_regulator.process(target_y)
120                     output_side = stabilize_on_arrow.side_regulator.process(target_x)
121
122                     output_forward = clamp(int(output_forward), -50, 50)
123                     output_side = clamp(int(output_side), -50, 50)
124
125                     context.set_speed(-output_forward)
126                     context.set_side_speed(-output_side)
127
128                     if abs(arrow_angle) < 2 and length < 5:
129                         if context.check_stabilization():
130                             return True
131                     else:
132                         context.reset_stabilization_counter()
133
134                 except AttributeError:
135                     stabilize_on_arrow.forward_regulator = PDRegulator()
136                     stabilize_on_arrow.forward_regulator.set_p_gain(0.5)
137                     stabilize_on_arrow.forward_regulator.set_d_gain(0.1)
138
139                     stabilize_on_arrow.side_regulator = PDRegulator()
140                     stabilize_on_arrow.side_regulator.set_p_gain(0.5)
141                     stabilize_on_arrow.side_regulator.set_d_gain(0.1)
142
143     return False
144
145 # прицелиться к мишени определенного цвета.
146 # для выстрела и прохождения через мишень
147 # будет отличаться требуемая глубина,
148 # поэтому присутствует дополнительная
149 # корректировка по оси Y
150 def aim_target(color, y_correction = 0):
151     image = mur.get_image_front()
152     contours = find_contours(image, colors[color][0], colors[color][1])
153
154     if contours:
155         for contour in contours:
156             area = cv.contourArea(contour)
157             if area < 100:
158                 continue
159
160             ((x, y), (w, h), _) = cv.minAreaRect(contour)
161
162             x_center = x - (320 / 2)
163             y_center = (y - (240 / 2)) - y_correction
164
165             # у нас нет возможности точно измерить расстояние,
166             # но можно получить примерное значение из площади контура.
167             # чем ближе к мишени - тем больше ее контур и наоборотю
168
169             area = w * h
170             target_distance = 50000
171
172             try:
173                 output_forward = aim_target.forward_regulator.process(target_distance
                    ↪  - area)
174                 output_yaw = aim_target.yaw_regulator.process(x_center)
175                 output_side = aim_target.side_regulator.process(x_center)
176                 output_depth = aim_target.depth_regulator.process(y_center)
```

```
177
178                 output_forward = clamp(int(output_forward), -20, 20)
179                 output_side = clamp(int(output_side), -50, 50)
180                 output_depth = clamp(output_depth, -1, 1)
181
182                 context.set_speed(output_forward)
183                 context.set_yaw(context.get_yaw() + output_yaw)
184                 context.set_side_speed(-output_side)
185                 context.set_depth(mur.get_depth() + output_depth)
186
187                 length = length_from_center(x_center, y_center)
188
189                 # нужно проверить как расстояние до мишени,
190                 # так и центрирование (расстояние до центра изображения)
191
192                 if length < 6 and abs(target_distance - area) < 5000:
193                     if context.check_stabilization(timeout=5):
194                         context.set_depth(mur.get_depth())
195                         context.set_speed(0)
196                         return True
197                 else:
198                     context.reset_stabilization_counter()
199
200             except AttributeError:
201                 aim_target.yaw_regulator = PDRegulator()
202                 aim_target.yaw_regulator.set_p_gain(0.002)
203                 aim_target.yaw_regulator.set_d_gain(0.002)
204
205                 aim_target.forward_regulator = PDRegulator()
206                 aim_target.forward_regulator.set_p_gain(0.2)
207                 aim_target.forward_regulator.set_d_gain(0.1)
208
209                 aim_target.side_regulator = PDRegulator()
210                 aim_target.side_regulator.set_p_gain(0.5)
211                 aim_target.side_regulator.set_d_gain(0.1)
212
213                 aim_target.depth_regulator = PDRegulator()
214                 aim_target.depth_regulator.set_p_gain(0.01)
215                 aim_target.depth_regulator.set_d_gain(0.01)
216
217     return False
218
219 # обнаружение голубой полоски
220 def find_cyan_line(image):
221     contours = find_contours(image, (80, 50, 50), (100, 255, 255))
222     if contours:
223         for contour in contours:
224             area = cv.contourArea(contour)
225             if abs(area) < 100:
226                 continue
227
228             ((_, _), (w, h), _) = cv.minAreaRect(contour)
229             aspect_ratio = max(w, h) / min(w, h)
230
231             moments = cv.moments(contour)
232             try:
233                 x = int(moments['m10'] / moments['m00'])
234                 y = int(moments['m01'] / moments['m00'])
235                 return True, x, y, find_rectangle_contour_angle(contour)
236             except ZeroDivisionError:
```

```
237                 return False, 0, 0, 0
238     return False, 0, 0, 0
239 
240 # стабилизироваться по координатам, а также по углу
241 # иными словами, расположиться над объектом с заданным направлением
242 def stabilize_x_y_angle(x, y, angle):
243     y_center = y - (240 / 2)
244     x_center = x - (320 / 2)
245 
246     try:
247         length = math.sqrt(x_center ** 2 + y_center ** 2)
248         if length < 4.5:
249             if context.check_stabilization():
250                 return True
251         else:
252             context.reset_stabilization_counter()
253 
254         output_forward = stabilize_x_y_angle.forward_regulator.process(y_center)
255         output_side = stabilize_x_y_angle.side_regulator.process(x_center)
256 
257         output_forward = clamp(int(output_forward), -50, 50)
258         output_side = clamp(int(output_side), -50, 50)
259 
260         context.set_speed(-output_forward)
261         context.set_side_speed(-output_side)
262 
263         context.set_yaw(mur.get_yaw() + angle)
264     except AttributeError:
265         stabilize_x_y_angle.forward_regulator = PDRegulator()
266         stabilize_x_y_angle.forward_regulator.set_p_gain(0.5)
267         stabilize_x_y_angle.forward_regulator.set_d_gain(0.1)
268 
269         stabilize_x_y_angle.side_regulator = PDRegulator()
270         stabilize_x_y_angle.side_regulator.set_p_gain(0.5)
271         stabilize_x_y_angle.side_regulator.set_d_gain(0.1)
272     return False
273 
274 # стабилизироваться над голубой полоской
275 def stabilize_over_cyan_line():
276     found, x, y, angle = find_cyan_line(mur.get_image_bottom())
277 
278     if found:
279         print(x, y, angle)
280         if stabilize_x_y_angle(x, y, angle):
281             return True
282         else:
283             return False
284 
285 # обнаружение голубого круга
286 def find_cyan_circle(image):
287     contours = find_contours(image, (80, 50, 50), (100, 255, 255))
288 
289     if contours:
290         for contour in contours:
291             area = cv.contourArea(contour)
292             if abs(area) < 400:
293                 continue
294 
295             ((_, _), (w, h), _) = cv.minAreaRect(contour)
296             (_, _), radius = cv.minEnclosingCircle(contour)
```

```
297             rectangle_area = w * h
298             circle_area = radius ** 2 * math.pi
299             aspect_ratio = w / h
300 
301             if 0.85 <= aspect_ratio <= 1.15:
302                 if rectangle_area > circle_area:
303                     moments = cv.moments(contour)
304                     try:
305                         x = int(moments['m10'] / moments['m00'])
306                         y = int(moments['m01'] / moments['m00'])
307                         return True, x, y, area
308                     except ZeroDivisionError:
309                         return False, 0, 0, 0
310     return False, 0, 0, 0
311 
312 # стабилизироваться над голубым кругом
313 def stabilize_over_cyan_circle():
314     found, x, y, angle = find_cyan_circle(mur.get_image_bottom())
315 
316     if found:
317         if stabilize_x_y_angle(x, y, 0):
318             return True
319         else:
320             return False
321 
322 # прицелиться для выстрела в мишень
323 def aim_shooting_target():
324     # цвет целевой мишени возьмем из списка shooting_colors
325     # для выстрела нужно расположиться чуть выше центра мишени
326     if (aim_target(context.shooting_colors[-1], 25)):
327         context.shooting_colors.pop() # удаляем отработанную мишень из списка
328         return True
329     else:
330         return False
331 
332 # прицелиться для прохода через мишень
333 def aim_through_target():
334     if (aim_target(context.target_color)):
335         return True
336     else:
337         return False
```

Опишем такие действия, как движение вперед и назад, остановка движения, ожидание и всплытие.

```
1  # двигаться вперед
2  def go_forward():
3      if (context.get_speed() != 30):
4          context.set_speed(30)
5          return False
6      else:
7          return True
8  
9  # двигаться назад
10 def go_back():
11     context.set_depth(2.0)
12     if (context.get_speed() != -30):
13         context.set_speed(-30)
14         return False
```

```
    else:
        return True

# ожидание 5 секунд
def wait():
    time.sleep(5)
    return True

# ожидание 10 секунд
def wait_long():
    time.sleep(10)
    return True

# остановить движение
def stop():
    if (context.get_speed() != 0):
        context.set_speed(0)
        return False
    else:
        return True

# всплыть на поверхность
def surface():
    context.set_depth(0)
    mur.set_motor_power(2, 50)
    mur.set_motor_power(3, 50)
    time.sleep(5)
    return True
```

Далее расположен тот код, который выполняется непосредственно при запуске скрипта. Здесь будут описан алгоритм выполнения миссий, который состоит из ранее написанных функций. Миссия была разбита на подзадачи.

```
# основной код, выполняемый при запуске скрипта
if __name__ == "__main__":
    context.set_depth(2.0)
    context.set_yaw(0.0)

    # разделим большое задание на мелкие задачи

    # выстрел через мишень:
    shoot_target = (
        aim_shooting_target, # сначала прицелиться для выстрела
        shoot,               # затем выстрелить
        go_back,             # начать движение назад
        wait,                # ожидание 5 секунд
        stop,                # после этого остановиться
    )

    # пройти через мишень:
    go_through_target = (
        aim_through_target,  # прицелиться для прохода
        go_forward,          # начать движение вперед
        wait_long,           # ожидание 10 секунд
        stop,                # остановить движение
        stabilize,           # стабилизироваться (подождать, пока остановимся)
    )

    # основной алгоритм миссии
```

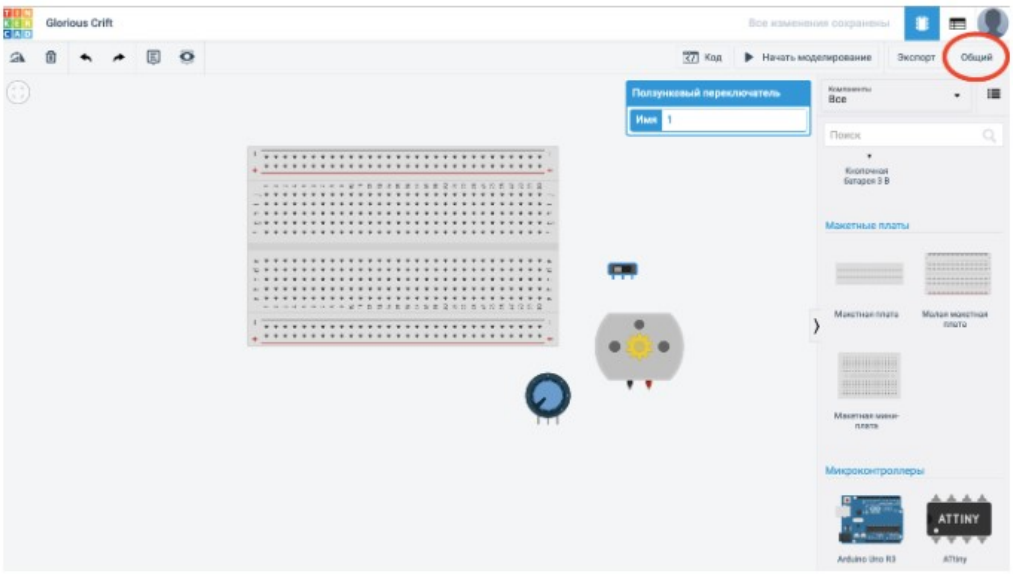
```
    # для добавления списка действий (подзадач),
    # используется распаковка значений списка (звездочка)
    missions = (
        detect_arrow_color,  # определяем цвет стрелки
        stabilize_on_arrow,  # стабилизируемся над стрелкой
        *shoot_target,       # выстрелить в первую мишень из списка
        *shoot_target,       # выстрел во вторую мишень из списка
        *go_through_target,  # пройти через последнюю мишень
        stabilize_over_cyan_line, # стабилизироваться над голубой полоской
        go_forward,          # движение вперед
        stabilize_over_cyan_circle,# стабилизироваться над голубым кругом
        stabilize,           # окончательно стабилизируемся
        surface,             # всплытие
    )

    # задаем список действий миссий
    context.push_mission_list(missions)

    print("start")

    # в цикле проходим каждое действие для выполнения миссии,
    # а также выводим в консоль отладочную информацию
    while (True):
        mission = context.pop_mission()
        print('starting', mission.__name__, '\t\ttime:', context.time)
        while not mission():
            context.process()
            context.time += 1

        if context.get_missions_length() == 0:
            break

    print("done!")

    context.set_speed(0)
    context.set_depth(2.5)

    time.sleep(3)
```

Как выполнять задания в Tinkercad и отправлять решения

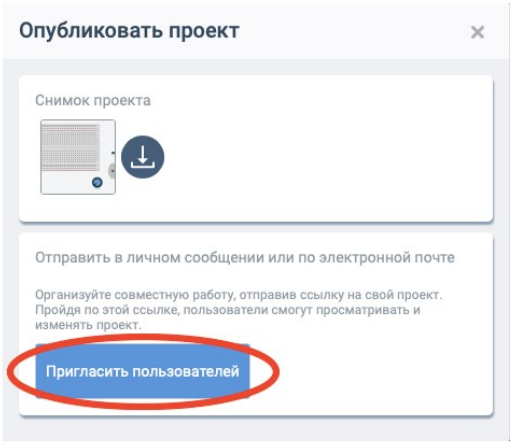
Задания выполняются в онлайн среде Tinkercad(https://www.tinkercad.com/). Для этого необходимо зарегистрироваться на сайте.

Рекомендуем окрашивать провода в разные цвета, чтобы избежать ошибок и путаницы (например: провода GND — черный, Питание — красный). Также старайтесь прокладывать провода параллельно и в обход других компонентов.

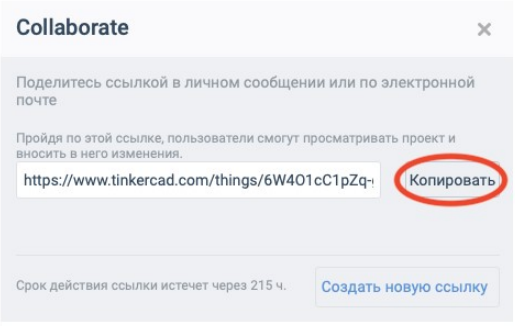
В качестве решения задач в Thinkercad необходимо прислать ссылку на ваш проект. Для этого необходимо нажать на кнопку «общий».



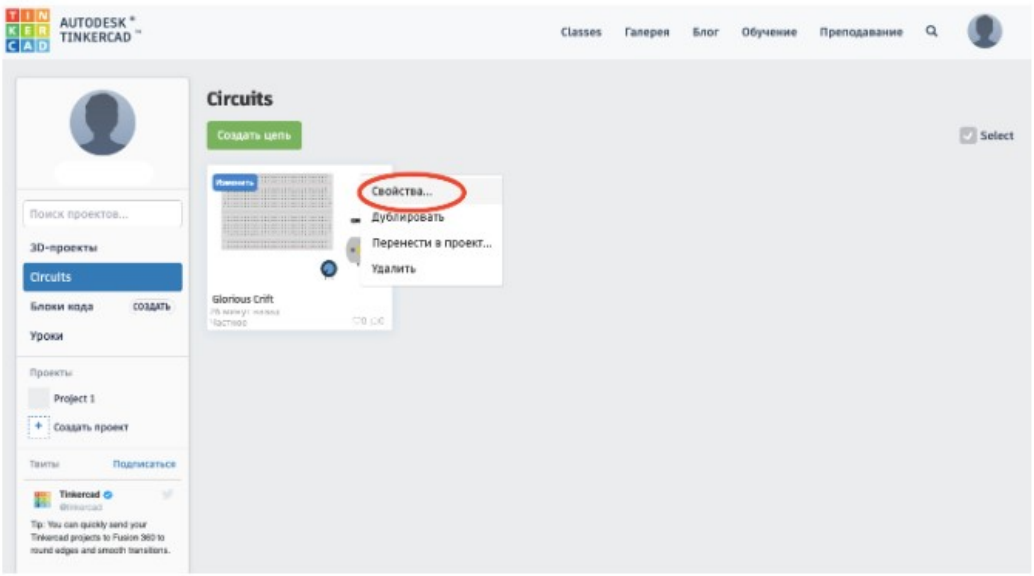
В открывшемся окне на «Пригласить пользователей».



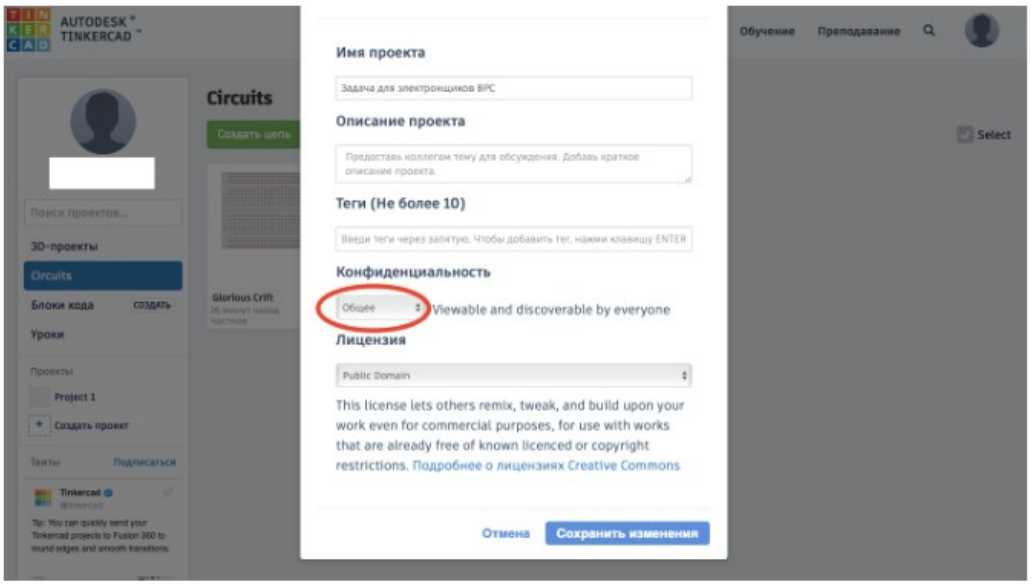
И в новом окне скопировать ссылку на проект.



Только обязательно необходимо убедиться, что проект у вас «Общий». Для того, чтобы проверить это, зайдите в свойства проекта.



И поставьте в разделе «Конфиденциальность» значение «Общее».



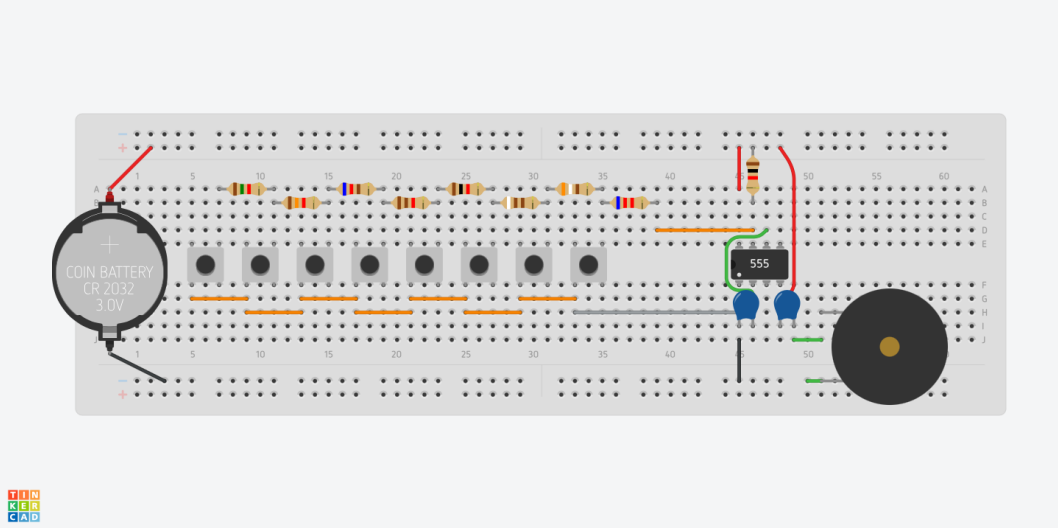
Задача II.1.1.2. Задание 1 для электронщиков (100 баллов)

1. Дано (элементы в Thinkercad)
   - Двигатель постоянного тока.
   - Потенциометр.
   - Аналоговые компоненты: резисторы, конденсаторы, индуктивности, диоды, транзисторы.
   - Элементы питания.
   - Логические микросхемы, находящиеся в разделе «Логика».
   - Макетные платы.
   - Таймер.

2. Необходимо, используя online-сервис Tinkercad (во избежание проблем с моделированием, рекомендуем пользоваться Google Chrome), создать проект электрической схемы управления коллекторным мотором. Основные требования:
    - можно использовать только те компоненты, которые приведены в пункте 1.1.
    - Необходимо реализовать управление скоростью вращения мотора при помощи потенциометра без изменения направления вращения мотора.
    - Диапазон скорости мотора не должен выходить за пределы (0; 10000).
    - Скорость вращения должна меняться с помощью широтно-импульсной модуляции.
    - Управление вращением должно осуществляться при помощи транзистора.
    - Не разрешается использование микроконтроллеров, датчиков, контроллеров управления моторами, электрических приводов с H-мостами, а также генераторов сигналов и регулируемых источников питания.
    - Изменение скорости вращения должно происходить плавно.
3. Схемы, в которых будут использоваться неразрешенные компоненты, проверяться не будут. Участник получает за него 0 баллов.
4. Баллы за задачу начисляются следующим образом.
    - 100 баллов — схема работает правильно в соответствие с требованиями. Не используются избыточные компоненты.
    - 90 баллов — схема работает правильно в соответствие с требованиями. Используются избыточные компоненты.
    - 80 баллов — схема работает правильно в соответствие с требованиями. Не используются избыточные компоненты. Но скорость меняется, но не в полном диапазоне, например, в диапазоне 0–70%. Мотор можно остановить.
    - 50 баллов — схема работает частично. Диапазон изменения скорости более 50%. Мотор можно остановить.
    - 30 баллов — схема работает частично. Диапазон изменения скорости менее 50%. Мотор нельзя остановить.
    - 0 баллов — схема не работает даже частично.
5. В качестве решения данной задачи необходимо прислать ссылку на ваш проект. Перед отправкой ссылки в ответ рекомендуем проверить работу моделирования. Инструкция по нахождению ссылки находится в 1 уроке данного модуля. Решения, ссылки на которые скопированы из адресной строки, проверяться не будут.

Решение

Данная схема является примером выполнения задачи на 100 баллов. Здесь представлены все элементы, которые можно использовать без снижения баллов за их избыточное количество. Схема состоит из таймера (микросхема таймера NE555), а также набора сопротивлений с кнопками для генерации определенной частоты.



Ссылка на пример решенного задания:

`https://www.tinkercad.com/things/4VE9SWioZjZ-pianino-nti/editel?sharecode=gsfBZBmRAjAWPFqikb3mG7nRFFhLUlCRB-HYOw1CUyI`

Задача II.1.1.3. Задание 1 для программистов микроконтроллеров (100 баллов)

1. Дано:
   - Arduino UNO.
   - 7-сегментный дисплей.
   - Микросхема CD4511.
   - Две тактовые кнопки.
   - Потенциометр.
2. Используя online-сервис Tinkercad (во избежание проблем с моделированием, рекомендуем пользоваться Google Chrome), необходимо скопировать проект по ссылке: https://www.tinkercad.com/things/3TYLXIYlBnm
   В данной схеме имеется микроконтроллер Arduino UNO, к которому подключен 7-сегментный дисплей через микросхему CD4511, а также подключены две тактовые кнопки и потенциометр. Редактирование электронной схемы не допускается.
3. Необходимо запрограммировать Arduino UNO, чтобы выполнялись следующие требования:
   - Устройство должно представлять собой таймер обратного отсчета, значение счетчика должно отображаться на дисплее.
   - С помощью потенциометра пользователь должен иметь возможность установить на дисплее число от 0 до 9 включительно.
   - С помощью кнопки «Сброс» (пин 3) отображаемое число сбрасывается к значению, которое установлено через потенциометр.
   - С помощью кнопки «Старт/Пауза» (пин 2) должна предоставляться возможность запускать таймер и ставить его на паузу.
   - После того, как значение таймера достигает нуля, должен загораться встроенный светодиод на Arduino UNO, в иных случаях светодиод дол-

жен быть выключен.

4. Баллы за задачу начисляются следующим образом:
   - Корректная работа обратного отсчета (счетчик уменьшается на единицу каждую секунду вплоть до нуля, после чего таймер останавливается и загорается светодиод) — 20 баллов.
   - Установка таймера через потенциометр — 20 баллов.
   - Функция старта таймера — 15 баллов.
   - Функция паузы — 15 баллов.
   - Функция сброса — 10 баллов.
   - Обработка пользовательского ввода без задержек — 20 баллов.

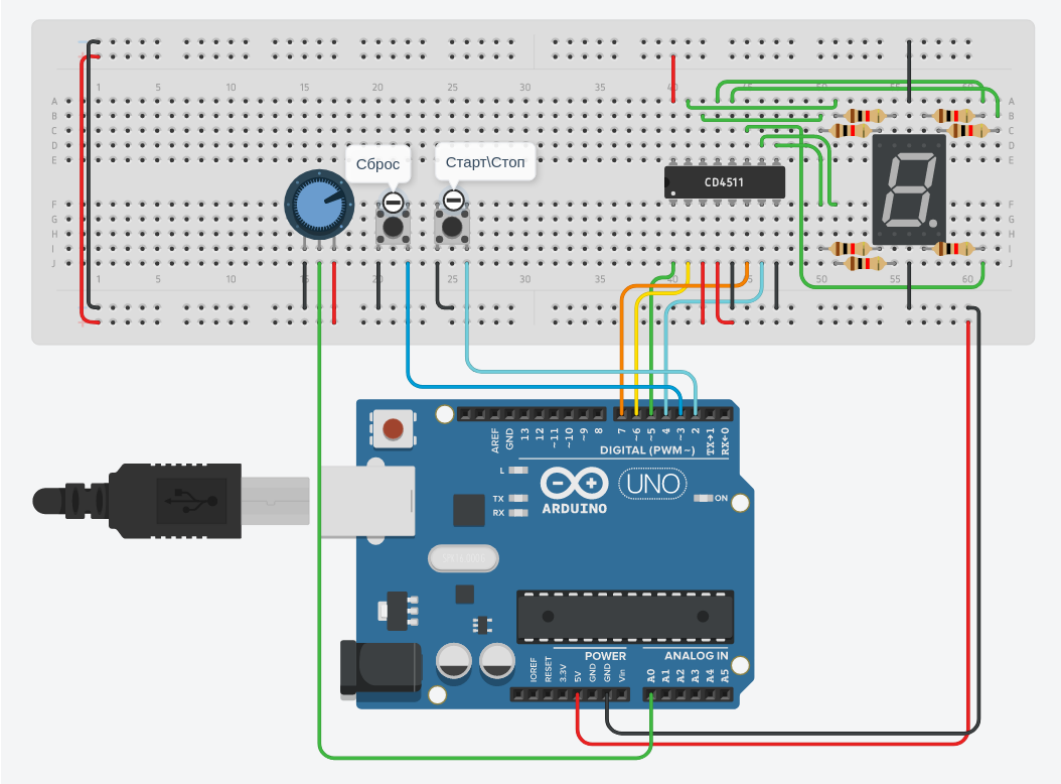
   Итого за задачу можно получить до 100 баллов.
   В случае модификации электрической схемы задание не проверятся (0 баллов за задание).
5. В качестве решения данной задачи необходимо прислать ссылку на ваш проект. Перед отправкой ссылки в ответ рекомендуем проверить работу моделирования. Инструкция по нахождению ссылки находится в 1 уроке данного модуля. Решения, ссылки на которые скопированы из адресной строки, проверяться не будут.

Решение

Решение (NTI-Arduino-Task1-Timer.ino) доступно по ссылке

https://yadi.sk/d/KrZAbhWM9nDhAw

В данной схеме имеется микроконтроллер Arduino UNO, к которому подключен 7-сегментный дисплей через микросхему CD4511, а также подключены две тактовые кнопки и потенциометр. Редактирование электронной схемы не допускается. Устройство представляет собой таймер обратного отсчета.



Ссылка на схему TinkerCad:

https://www.tinkercad.com/things/jvSEil0gZCt

Программный код для Arduino:

```
/* Таймер.
 *
 * Количество секунд для отсчета задается потенциометром.
 *
 * Кнопка "старт\стоп" запускает или останавливает таймер
 * в зависимости от его текущего состояния.
 *
 * Кнопка "сброс" останавливает и сбрасывает таймер
 * и переводит его в режим настройки потенциометром.
 *
 * При достижении нуля таймер останавливает и
 * загорается светодиод.
 */

// порты для потенциометра, кнопок и светодиода
#define POTENTIOMETER A0
#define BUTTON_START 2
#define BUTTON_RESET 3
#define LED 13

// порты для микросхемы CD4511, которая
// управляет 7-сегментным индикатором
const int digit_pins[4] = {4, 5, 6, 7};

// режимы работы таймера:
// SETTING - настройка, установка секунд отсчета
// RUNNING - отсчет времени
// PAUSE - пауза
enum modes {SETTING = 0, RUNNING, PAUSE};
```

```
30
31  // переменная текущего режима
32  int mode = SETTING;
33
34  // счетчик секунд
35  uint8_t counter = 0;
36
37  // обработка нажатия кнопки старт\стоп
38  void button_start_clicked() {
39    // если сейчас режим настройки или пауза...
40    if (mode == SETTING || mode == PAUSE) {
41      // то запускаем таймер
42      mode = RUNNING;
43    } else if (mode == RUNNING) {
44      // если же он запущен, то ставим паузу
45      mode = PAUSE;
46    }
47  }
48
49  // обработка нажатия кнопки сброса
50  void button_reset_clicked() {
51    // перевод таймера в режим настройки
52    mode = SETTING;
53    // сброс счетчика
54    counter = 0;
55  }
56
57  // функция для чтения значений с потенциометра
58  int read_potentiometer() {
59    // считывание необработанного значения с АЦП
60    int potentiometer_value = analogRead(POTENTIOMETER);
61    // перевод значения из шкалы 0...1023 в диапазон 0...9
62    return map(potentiometer_value, 0, 1023, 0, 9);
63  }
64
65  // обработка одного тика таймера
66  void timer_tick() {
67    // если счетчик больше нуля, то делаем отсчет
68    if (counter > 0) {
69      // уменьшение счетчика на единицу
70      counter--;
71      // ожидание в 1 секунду
72      delay(1000);
73    } else if (counter == 0) { // если достигли нуля
74      // то включается светодиод
75      digitalWrite(LED, HIGH);
76    }
77  }
78
79  // функция установки времени отсчета
80  void setting_time() {
81    // приравнивение счетчика значению с потенциометра
82    counter = read_potentiometer();
83    // небольшая задержка
84    delay(100);
85  }
86
87  // функция, которая один раз запускается на старте
88  void setup() {
89    // инициализация портов кнопок на вход
```

```
  pinMode(BUTTON_START, INPUT_PULLUP);
  pinMode(BUTTON_RESET, INPUT_PULLUP);
  // инициализация порта светодиода на выход
  pinMode(LED, OUTPUT);

  // привязка прерываний портов кнопок к обработчикам
  // (это позволит обрабатывать нажатия кнопок асинхронно
  // и независимо от основного цикла). благодаря этому,
  // ожидаемый сигнал не будет пропущен.

  attachInterrupt(digitalPinToInterrupt(BUTTON_START),
                  button_start_clicked, FALLING);

  attachInterrupt(digitalPinToInterrupt(BUTTON_RESET),
                  button_reset_clicked, FALLING);

  // инициализация портов индикатора
  for (int i = 0; i < 4; i++) {
    pinMode(digit_pins[i], OUTPUT);
  }
}

// основной цикл программы
void loop() {
  // если режим установки времени отсчета,
  // то вызываем соответствующую функцию-обработчик
  if (mode == SETTING) setting_time();
  // если таймер запущен, то нужно вызвать
  // функцию обработки тика таймера
  if (mode == RUNNING) timer_tick();
  // в случае паузы просто делаем небольшую задержку
  if (mode == PAUSE) delay(100);

  // вывод текущего значения счетчика на индикатор
  for (int i = 0; i < 4; i++) {
    // индикатор управляется микросхемой CD4511.
    // он преобразует двоичный код в соответствующий
    // для цифры код 7-сегментного индикатора.
    // чтобы вывести цифру, нужно подать на входы
    // микросхемы двоичный код числа.
    // для этого воспользуемся функцией bitRead,
    // которая позвоялет считать один бит числа
    // в определенной позиции.
    digitalWrite(digit_pins[i], bitRead(counter, i));
  }

  // если счетчик не равен нулю, то выключаем светодиод.
  if (counter != 0) digitalWrite(LED, LOW);
```

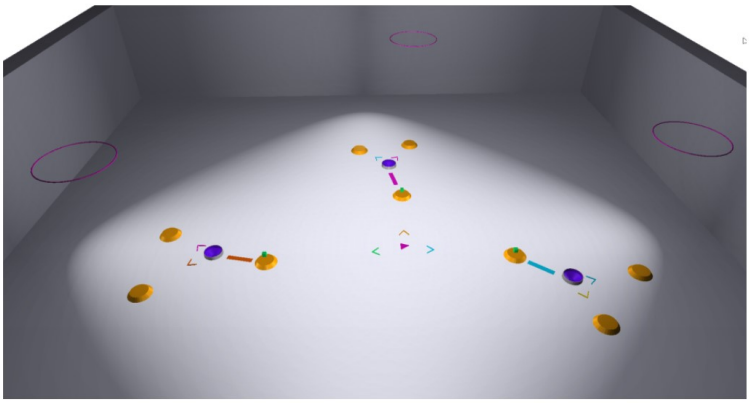
Индивидуальные задания 2 уровень сложности

Задача II.1.2.1. Задание 2 для программистов на MUR IDE (100 баллов)

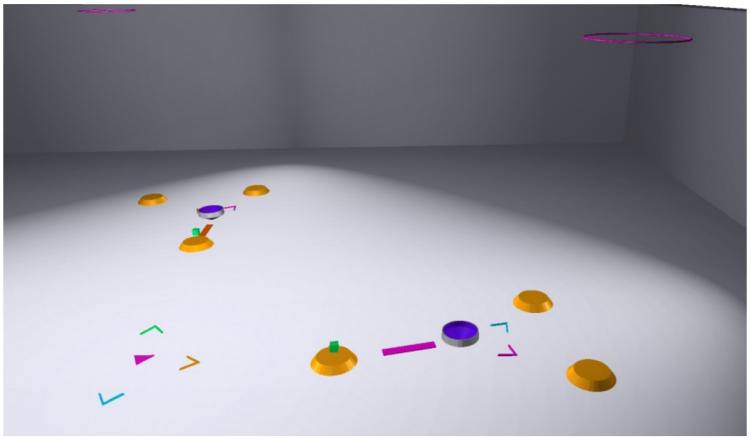
1. Задача выполняется в симуляторе и среде программирования MUR IDE. Сцены называются: NTI-Task-2_1, NTI-Task-2_2 и NTI-Task-2_3. Они доступны по

ссылке(http://robocenter.net/media/nti-tasks.zip).

2. Необходимо запрограммировать подводного робота, который должен в симуляторе в автономном режиме выполнить под водой ряд задач и всплыть в заданной области. Время на выполнение задачи — 5 минут.
3. Как только робот всплыл или закончилось время, задача считается законченной, фиксируется количество заработанных баллов.



4. Задание:
   - Робот должен определить форму навигационной метки (розового объекта на старте): треугольник, квадрат или круг.
   - Затем нужно пройти по направлению одного из трех указателей в зависимости от формы розового объекта (треугольник — зеленый указатель, квадрат — оранжевый указатель, круг — голубой указатель).
   - Каждый указатель показывает направление к платформе с кубом. Данный куб нужно подобрать и положить в синюю корзину, путь к которой указывает полоска.
   - Далее необходимо пройти к платформе по направлению одного из двух указателей, цвет которого совпадает с полоской.
   - Над данной платформой располагается обруч, в пределах которого нужно всплыть.

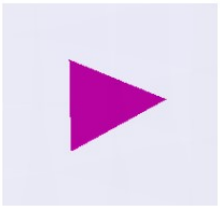


5. Баллы начисляются следующим образом:
   - Проплыть к платформе по направлению правильного указателя (в соответствии с формой навигационной метки) — 20 баллов.

- Проплыть к платформе по направлению неправильного указателя — 10 баллов.
- Взять куб с правильной платформы (в соответствии с меткой и указателем) — 30 баллов.
- Взять куб с неправильной платформы — 10 баллов.
- Положить куб в правильную корзину (в соответствии с меткой и указателем) — 25 баллов.
- Положить куб в неправильную корзину — 10 баллов.
- Всплыть в правильном обруче, в соответствии с цветом полоски — 25 баллов.
- Всплыть в неправильном обруче, в соответствии с цветом полоски — 10 баллов.

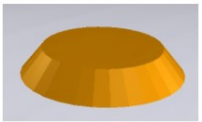
Итого за задачу можно заработать 100 баллов.

6. Описание макетов:
   В задаче используются следующие виды объектов: навигационная метка, указатель, платформа, полоска, корзина, обруч.



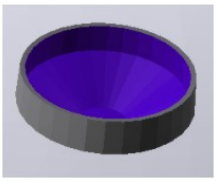
- Навигационная метка располагается на стартовой позиции, и может быть одной из трех форм: треугольник, квадрат и круг. В зависимости от формы, необходимо выбрать один из трех указателей (треугольник — зеленый указатель, квадрат — оранжевый указатель, круг — голубой указатель).

- Указатель может быть различных цветов. Первые три указателя на стартовой позиции показывают направления к платформам с кубами. Цвет нужного указателя соответствует форме розового объекта. Также имеются два указателя после синей корзины, и они показывают направления к платформам, над одной из которых находится обруч. Необходимо пройти по направлению того указателя, цвет которого совпадает с полоской.



- Платформа всегда оранжевого цвета. Первая встречающаяся платформа имеет куб, который необходимо подобрать.

- Полоска указывает направление к синей корзине. В зависимости от цвета полоски необходимо будет выбрать указатель, по которому необходимо следовать. Может быть розового, голубого или красного цвета.



- Корзина емкость, в которой должен оказаться куб. Всегда синего цвета.

- Обруч расположен на поверхности воды и находится над одной из платформ.

7. Задача будет проверяться на 3 сценах. Сцены при проверке будут отличаться от сцен, доступных для написания программы. Сцены будут отличаться друг от друга формой навигационной метки (розовый объект) на стартовой позиции, цветами и положением указателей и полосок, могут различаться направлением от платформ к корзинам (эти направления показывают полоски). В зачет попытки идет худший результат из трех сцен.
8. В качестве решения задачи вам необходимо прикрепить файл с кодом в формате *.py.

Решение

Решение (NTI-Simulator-Task2.py) доступно по ссылке:
https://yadi.sk/d/KrZAbhWM9nDhAw.

Для успешного выполнения задания аппарату в виртуальной среде MUR IDE необходимо определить форму навигационной метки для определение дальнейшего маршрута. Затем нужно пройти по направлению одного из трех указателей к платформе с кубом. Данный куб нужно подобрать и положить в синюю корзину, после чего пройти к платформе по направлению указателя, цвет которого совпадает с полоской. Над данной платформой располагается обруч, в пределах которого нужно всплыть.

Для начала импортируем библиотеки и зададим некоторые исходные данные из задания, как цвета объектов, соответствия форм и цветов объектов для определения маршрута и т. д.

```python
import pymurapi as api
import cv2 as cv
import time
import numpy as np
import math

mur = api.mur_init()

# предопределеные диапазоны цветов (HSV)
colors = {
    'green': ((45, 50, 50), (75, 255, 255)),
    'blue': ((130, 50, 50), (140, 255, 255)),
    'cyan': ((80, 50, 50), (100, 255, 255)),
    'orange': ((15, 50, 50), (30, 255, 255)),
    'red': ((0, 50, 50), (15, 255, 255)),
    'magenta': ((140, 50, 50), (160, 255, 255)),
}

# соответствие формы объекта к цвету (согласно заданию)
tag_shape_to_color = {
    'triangle': 'green',
    'square': 'orange',
    'circle': 'cyan',
}

# возможные варианты цветов полоски
line_colors = ['magenta', 'cyan', 'red']
```

Аналогично предыдущему заданию напишем несколько функций для расчетов, PD-регулятор, а также функции для задания курса и глубины погружения робота.

```python
# функция для ограничения значения диапазоном
def clamp(value, min_value, max_value):
    if value < min_value:
        return min_value
    if value > max_value:
        return max_value
    return value

# функция для расчета угла между точками
def angle_between(p1, p2):
    xDiff = p2[0] - p1[0]
    yDiff = p2[1] - p1[1]
    return math.degrees(math.atan2(yDiff, xDiff) - (np.pi / 2))

# функция для рассчета расстояния от центра до точки
def length_from_center(x, y):
    return math.sqrt(x ** 2 + y ** 2)
# PD-регулятор
class PDRegulator(object):
    _p_gain = 0.0
    _d_gain = 0.0
    _prev_error = 0.0
    _timestamp = 0

    def __init__(self):
        pass

    def set_p_gain(self, value):
```

```
        self._p_gain = value

    def set_d_gain(self, value):
        self._d_gain = value

    def process(self, error):
        timestamp = int(round(time.time() * 1000))

        if timestamp == self._timestamp:
            return 0

        output = self._p_gain * error + self._d_gain / (timestamp - self._timestamp) *
        ↪  (error - self._prev_error)
        self._timestamp = timestamp
        self._prev_error = error
        return output
# функция для установки курса робота
def keep_yaw(yaw_to_set, speed):
    def clamp_angle(angle):
        if angle > 180.0:
            return angle - 360.0
        if angle < -180.0:
            return angle + 360
        return angle

    try:
        error = mur.get_yaw() - yaw_to_set
        error = clamp_angle(error)
        output = keep_yaw.yaw_regulator.process(error)
        mur.set_motor_power(0, clamp(-output + speed, -100, 100))
        mur.set_motor_power(1, clamp(output + speed, -100, 100))
    except AttributeError:
        keep_yaw.yaw_regulator = PDRegulator()
        keep_yaw.yaw_regulator.set_p_gain(0.8)
        keep_yaw.yaw_regulator.set_d_gain(0.6)

# функция для установки глубины погружения
def keep_depth(depth_to_set):
    try:
        error = mur.get_depth() - depth_to_set
        output = keep_depth.depth_regulator.process(error)
        output = clamp(output, -100, 100)
        mur.set_motor_power(2, output)
        mur.set_motor_power(3, output)
    except AttributeError:
        keep_depth.depth_regulator = PDRegulator()
        keep_depth.depth_regulator.set_p_gain(45)
        keep_depth.depth_regulator.set_d_gain(5)
```

Также как и в предыдущем задании, здесь будет использоваться класс для хранения состояния робота и хода выполнения миссии.

```
# класс для хранения текущего состояния
class AUVContext(object):
    _yaw = 0.0
    _depth = 0.0
    _speed = 0.0
    _side_speed = 0.0
    _timestamp = 0
```

```
8       _missions = []
9       _min_area = math.inf
10      _min_yaw = 0.0
11      _stabilization_counter = 0
12      time = 0
13
14      def __init__(self):
15          pass
16
17      def set_min_circle(self, yaw, area):
18          if area < self._min_area:
19              self._min_area = area
20              self._min_yaw = yaw
21
22      def get_min_circle_yaw(self):
23          return self._min_yaw
24
25      def get_yaw(self):
26          return self._yaw
27
28      def get_depth(self):
29          return self._depth
30
31      def get_speed(self):
32          return self._speed
33
34      def get_side_speed(self):
35          return self._side_speed
36
37      def set_yaw(self, value):
38          self._yaw = value
39
40      def set_depth(self, value):
41          self._depth = value
42
43      def set_speed(self, value):
44          self._speed = value
45
46      def set_side_speed(self, value):
47          self._side_speed = value
48
49      def get_stabilization_counter(self):
50          return self._stabilization_counter
51
52      def reset_stabilization_counter(self):
53          self._stabilization_counter = 0
54
55      def add_stabilization_counter(self):
56          self._stabilization_counter += 1
57
58      def check_stabilization(self, timeout = 3):
59          if self._stabilization_counter > timeout:
60              return True
61          else:
62              self.add_stabilization_counter()
63              return False
64
65      def push_mission(self, mission):
66          self._missions.append(mission)
67
```

```
68      def push_mission_list(self, missions):
69          for mission in missions:
70              self.push_mission(mission)
71
72      def pop_mission(self):
73          if len(self._missions) != 0:
74              return self._missions.pop(0)
75          return {}
76
77      def get_missions_length(self):
78          return len(self._missions)
79
80      def process(self):
81          timestamp = int(round(time.time() * 1000))
82          if timestamp - self._timestamp > 16:
83              keep_yaw(self._yaw, self._speed)
84              keep_depth(self._depth)
85              mur.set_motor_power(4, self._side_speed)
86              self._timestamp = timestamp
87          else:
88              time.sleep(0.05)
89
90  context = AUVContext()
```

Также напишем функции для выполнения таких действий, как развороты, выстрел, движения и т. д.

```
1   # разворот на 90 градусов
2   def translate_to_90():
3       yaw = mur.get_yaw() + 90
4       if yaw < -180:
5           yaw += 360
6       if yaw > 180:
7           yaw -= 360
8       context.set_yaw(yaw)
9       return True
10
11  # разворот на 180 градусов
12  def translate_to_180():
13      yaw = mur.get_yaw() + 180
14      if yaw < -180:
15          yaw += 360
16      if yaw > 180:
17          yaw -= 360
18      context.set_yaw(yaw)
19      return True
20
21  # произвести выстрел
22  def shoot():
23      if (context.get_speed() != 0):
24          context.set_speed(0)
25          return False
26      else:
27          mur.shoot()
28          time.sleep(0.5)
29          return True
30
31  # движение вперед
32  def go_forward():
```

```
    if (context.get_speed() != 15):
        context.set_speed(15)
        return False
    else:
        return True

# движение назад
def go_back():
    if (context.get_speed() != -2.5):
        context.set_speed(-2.5)
        return False
    else:
        return True

# ожидание 5 секунд
def wait():
    time.sleep(5)
    return True

# ожидание 3 секунды
def wait_short():
    time.sleep(3)
    return True

# ожидание 12 секунд
def wait_long():
    time.sleep(12)
    return True

# прекратить движение
def stop():
    if (context.get_speed() != 0):
        context.set_speed(0)
        return False
    else:
        return True

# всплытие на поверхность
def surface():
    mur.set_motor_power(2, 50)
    mur.set_motor_power(3, 50)
    time.sleep(5)
    return True

# стабилизировать курс и глубину
def stabilize():
    yaw = mur.get_yaw()
    depth = mur.get_depth()

    if abs(yaw - context.get_yaw()) < 1 and abs(depth - context.get_depth()) < 0.3:
        if context.check_stabilization(timeout=5):
            return True
    else:
        context.reset_stabilization_counter()
    return False
```

Для выполнения этого задания также пригодится функция для определения контуров по цвету.

```
# поиск контура по цвету
```

```
def find_contours(image, color, approx = cv.CHAIN_APPROX_SIMPLE):
    hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
    mask = cv.inRange(hsv_image, color[0], color[1])
    contours, _ = cv.findContours(mask, cv.RETR_CCOMP, approx)
    return contours
```

Далее идут функции, активно использующие машинное зрение. Например, это определение цветов объектов, их углов, позиционирование и стабилизация над объектами и т. д.

```
# расчет угла прямоугольника
# для определение отклонения от полоски,
# чтобы затем скорректировать курс
def find_rectangle_contour_angle(contour):
    rectangle = cv.minAreaRect(contour)
    box = cv.boxPoints(rectangle)
    box = np.int0(box)
    edge_first = np.int0((box[1][0] - box[0][0], box[1][1] - box[0][1]))
    edge_second = np.int0((box[2][0] - box[1][0], box[2][1] - box[1][1]))

    edge = edge_first
    if cv.norm(edge_second) > cv.norm(edge_first):
        edge = edge_second

    angle = -((180.0 / math.pi * math.acos(edge[0] / (cv.norm((1, 0)) *
    ↪  cv.norm(edge)))) - 90)
    return angle

# определение цвета из списка возможных.
# рассчитывается площадь контура для каждого из цветов,
# а затем выбирается цвет по наибольшему контуру
def detect_color(available_colors):
    image = mur.get_image_bottom()
    hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)

    areas = {}

    for color in available_colors:
        mask = cv.inRange(hsv_image, colors[color][0], colors[color][1])
        contours, _ = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)

        biggest_area = 0

        if contours:
            for contour in contours:
                area = cv.contourArea(contour)
                if area > biggest_area:
                    biggest_area = area

        areas[color] = biggest_area

    color = sorted(areas, key=areas.get, reverse=True)[0]

    if (areas[color] > 5):
        return color
    else:
        return False

# определение направления стрелки
```

```python
def detect_arrow_angle(image, contour):
    (arrow_center_x, arrow_center_y), radius = cv.minEnclosingCircle(contour)
    moments = cv.moments(contour)

    try:
        arrow_direction_x = int(moments['m10'] / moments['m00'])
        arrow_direction_y = int(moments['m01'] / moments['m00'])

        arrow_angle = (angle_between((arrow_direction_x, arrow_direction_y),
        ↪  (arrow_center_x, arrow_center_y)))
        context.set_yaw(mur.get_yaw() + arrow_angle)

        target_x = arrow_center_x - (320 / 2)
        target_y = arrow_center_y - (240 / 2)
        length = math.sqrt(target_x ** 2 + target_y ** 2)

        return arrow_angle, target_x, target_y, length
    except:
        return False, False, False, False

# стабилизироваться над стрелкой (с учетом направления стрелки)
def stabilize_on_arrow(color):
    image = mur.get_image_bottom()

    contours = find_contours(image, color, cv.CHAIN_APPROX_SIMPLE)

    if contours:
        for contour in contours:
            (arrow_angle, target_x, target_y, length) = detect_arrow_angle(image,
            ↪  contour)

            if arrow_angle != False:
                try:
                    output_forward =
                    ↪  stabilize_on_arrow.forward_regulator.process(target_y)
                    output_side = stabilize_on_arrow.side_regulator.process(target_x)

                    output_forward = clamp(int(output_forward), -50, 50)
                    output_side = clamp(int(output_side), -50, 50)

                    context.set_speed(-output_forward)
                    context.set_side_speed(-output_side)

                    if abs(arrow_angle) < 2 and length < 7:
                        if context.check_stabilization(timeout=2):
                            return True
                    else:
                        context.reset_stabilization_counter()

                except AttributeError:
                    stabilize_on_arrow.forward_regulator = PDRegulator()
                    stabilize_on_arrow.forward_regulator.set_p_gain(0.5)
                    stabilize_on_arrow.forward_regulator.set_d_gain(0.1)

                    stabilize_on_arrow.side_regulator = PDRegulator()
                    stabilize_on_arrow.side_regulator.set_p_gain(0.5)
                    stabilize_on_arrow.side_regulator.set_d_gain(0.1)

    return False

```

```
106 # стабилизироваться по координатам, а также по углу
107 def stabilize_x_y_angle(x, y, angle):
108     y_center = y - (240 / 2)
109     x_center = x - (320 / 2)
110 
111     try:
112         length = math.sqrt(x_center ** 2 + y_center ** 2)
113         if length < 2.0:
114             if context.check_stabilization(timeout=10):
115                 return True
116         else:
117             context.reset_stabilization_counter()
118 
119         output_forward = stabilize_x_y_angle.forward_regulator.process(y_center)
120         output_side = stabilize_x_y_angle.side_regulator.process(x_center)
121 
122         output_forward = clamp(int(output_forward), -10, 10)
123         output_side = clamp(int(output_side), -10, 10)
124 
125         context.set_speed(-output_forward)
126         context.set_side_speed(-output_side)
127 
128         context.set_yaw(mur.get_yaw() + angle)
129     except AttributeError:
130         stabilize_x_y_angle.forward_regulator = PDRegulator()
131         stabilize_x_y_angle.forward_regulator.set_p_gain(0.5)
132         stabilize_x_y_angle.forward_regulator.set_d_gain(0.1)
133 
134         stabilize_x_y_angle.side_regulator = PDRegulator()
135         stabilize_x_y_angle.side_regulator.set_p_gain(0.5)
136         stabilize_x_y_angle.side_regulator.set_d_gain(0.1)
137     return False
138 
139 # обнаружение полоски определенного цвета
140 def find_line(image, color):
141     contours = find_contours(image, color)
142     if contours:
143         for contour in contours:
144             area = cv.contourArea(contour)
145             if abs(area) < 300:
146                 continue
147 
148             ((_, _), (w, h), _) = cv.minAreaRect(contour)
149             aspect_ratio = max(w, h) / min(w, h)
150 
151             moments = cv.moments(contour)
152             try:
153                 x = int(moments['m10'] / moments['m00'])
154                 y = int(moments['m01'] / moments['m00'])
155                 return True, x, y, find_rectangle_contour_angle(contour)
156             except ZeroDivisionError:
157                 return False, 0, 0, 0
158     return False, 0, 0, 0
159 
160 # обнаружение объекта определенного цвета
161 def find_colored_object(image, color):
162     contours = find_contours(image, color)
163     if contours:
164         for contour in contours:
165             area = cv.contourArea(contour)
```

```
166             if abs(area) < 50:
167                 continue
168 
169             ((_, _), (w, h), _) = cv.minAreaRect(contour)
170 
171             moments = cv.moments(contour)
172 
173             try:
174                 x = int(moments['m10'] / moments['m00'])
175                 y = int(moments['m01'] / moments['m00'])
176                 return True, x, y, area
177             except ZeroDivisionError:
178                 return False, 0, 0, 0
179     return False, 0, 0, 0
180 
181 # стабилизироваться над полоской
182 def stabilize_over_line(color):
183     found, x, y, angle = find_line(mur.get_image_bottom(), color)
184 
185     if found:
186         if stabilize_x_y_angle(x, y, angle):
187             return True
188         else:
189             return False
190 
191 # стабилизироваться над кубом
192 def stabilize_over_box():
193     if (context.get_depth() != 3.1):
194         context.set_depth(3.1)
195         return False
196 
197     found, x, y, area = find_line(mur.get_image_bottom(), colors['green'])
198 
199     if found:
200         if stabilize_x_y_angle(x, y + 7, 0):
201             return True
202         else:
203             return False
204 
205 # обнаружение круга
206 def find_circle(image, color):
207     contours = find_contours(image, color)
208 
209     if contours:
210         for contour in contours:
211             area = cv.contourArea(contour)
212             if abs(area) < 400:
213                 continue
214 
215             ((_, _), (w, h), _) = cv.minAreaRect(contour)
216             (_, _), radius = cv.minEnclosingCircle(contour)
217             rectangle_area = w * h
218             circle_area = radius ** 2 * math.pi
219             aspect_ratio = w / h
220 
221             if 0.85 <= aspect_ratio <= 1.15:
222                 if rectangle_area > circle_area:
223                     moments = cv.moments(contour)
224                     try:
225                         x = int(moments['m10'] / moments['m00'])
```

```
                        y = int(moments['m01'] / moments['m00'])
                        return True, x, y, area
                    except ZeroDivisionError:
                        return False, 0, 0, 0
    return False, 0, 0, 0

# стабилизироваться над кругом
def stabilize_over_circle(color):
    found, x, y, angle = find_circle(mur.get_image_bottom(), color)

    if found:
        if stabilize_x_y_angle(x, y, 0):
            return True
        else:
            return False
```

Для определения формы объекта используется подсчет площади вписанных фигур, которые сравниваются с площадью самой фигуры. По наименьшей разнице площадей можно определить, на какую из геометрических фигур объект наиболее похож.

```
# определение формы объекта определенного цвета
def detect_shape(image, color):
    contours = find_contours(image, color)

    if contours:
        for contour in contours:

            tag_area = cv.contourArea(contour)
            if (tag_area < 100):
                continue

            # после того, как найден контур подходящего цвета,
            # нужно подсчитать площади вписанных в него
            # треугольника, квадрата и круга.
            # форма объекта определяется по наибольшему
            # совпадению площади вписанной фигуры.

            (circle_x, circle_y), circle_radius = cv.minEnclosingCircle(contour)
            circle_area = circle_radius ** 2 * math.pi

            rectangle = cv.minAreaRect(contour)
            box = cv.boxPoints(rectangle)
            box = np.int0(box)
            rectangle_area = cv.contourArea(box)

            triangle = cv.minEnclosingTriangle(contour)[1]
            triangle = np.int0(triangle)
            triangle_area = cv.contourArea(triangle)

            areas = {
                'circle': circle_radius ** 2 * math.pi,
                'square': cv.contourArea(box),
                'triangle': cv.contourArea(triangle),
            }

            tag_shapes = list(areas.keys())
            shapes_areas = np.array(list(areas.values()))
            difference = abs(shapes_areas - tag_area)
```

```
            arg = np.argmin(difference)
            tag_shape = tag_shapes[arg]

            return tag_shape

    return False

# распознать навигационную метку
def detect_tag_shape():
    image = mur.get_image_bottom()
    tag_shape = detect_shape(image, colors['magenta'])
    if tag_shape != False:
        context.tag_shape = tag_shape
        # в соответствии с условиями задания,
        # определяем цвет стрелки, по которой
        # нужно следовать
        context.first_arrow_color = colors[tag_shape_to_color[tag_shape]]
        print(tag_shape, '- follow', tag_shape_to_color[tag_shape], 'arrow')
        return True
    else:
        return False

# стабилизироваться на первой стрелкой, по которой
# нужно следовать (используя ранее определенный цвет)
def stabilize_on_first_arrow():
    return stabilize_on_arrow(context.first_arrow_color)

# обнаружение оранжевого круга
def find_orange_circle():
    image = mur.get_image_bottom()
    found, x, y, area = find_circle(image, colors['orange'])

    if found:
        return True
    else:
        return False

# стабилизироваться над оранжевым кругом
def stabilize_over_orange_circle():
    return stabilize_over_circle(colors['orange'])

# установить подходящую глубину
# для захвата куба манипулятором
def set_grabbing_depth():
    mur.open_grabber()

    if (context.get_depth() != 3.62):
        context.set_depth(3.62)
        return False
    else:
        return True

# схватить куб
def grab_box():
    mur.close_grabber()
    return True

# отпустить куб
def ungrab_box():
    mur.open_grabber()
```

```
    return True

# определить цвет полоски
def detect_line_color():
    color = detect_color(line_colors)
    if color:
        context.line_color = color
        return True
    else:
        return False

# установить обычную глубину (после захвата куба)
def set_default_depth():
    if (context.get_depth() != 3.0):
        context.set_depth(3.0)
        return False
    else:
        return True

# обнаружение синей корзины (круглой формы)
def find_blue_bin():
    image = mur.get_image_bottom()
    found, x, y, area = find_circle(image, colors['blue'])

    if found:
        return True
    else:
        return False

# стабилизироваться над синей корзиной
def stabilize_over_blue_bin():
    return stabilize_over_circle(colors['blue'])

# стабилизироваться над второй полоской (ранее определенный цвет)
def stabilize_over_target_line():
    context.set_depth(2.5)
    return stabilize_over_line(colors[context.line_color])

# стабилизироваться над второй стрелкой
def stabilize_on_second_arrow():
    return stabilize_on_arrow(colors[context.line_color])
```

Далее идет код, который выполняется при запуске скрипта. Здесь описан алгоритм выполнения миссии, которая разбита на подзадачи.

```
# основной код, выполняемый при запуске скрипта
if __name__ == "__main__":
    context.set_depth(1.8)
    context.set_yaw(0.0)

    # определим подзадачи нашей миссии

    # для того, чтобы добраться до платформы с кубом, нужно:
    go_to_box_platform = (
        detect_tag_shape,            # определить маршрут по навигационной метке
        stabilize_on_first_arrow,    # стабилизироваться над первой стрелкой
        go_forward,                  # двигаться вперед
    )

```

```
    # поиск и захват куба
    find_and_grab_box = (
        find_orange_circle,           # обнаружить оранжевый круг (мы все еще движемся)
        stabilize_over_orange_circle,# стабилизироваться над этим кругом
        stabilize_over_box,           # найдя круг, можно найти и стабилизироваться над
        ↪  кубом
        stabilize,                    # окончательно стабилизируем свое положение перед
        ↪  захватом
        set_grabbing_depth,           # погружаемся ближе к кубу
        wait_short,                   # ожидание 3 секунды
        stabilize,                    # стабилизируем положение
        wait_short,                   #
        grab_box,                     # захват куба манипулятором
        wait_short,                   #
        set_default_depth,            # принимаем нормальную глубину
        go_forward,                   # двигаемся вперед
        stabilize,                    # стабилизируем курс и глубину
        wait_short,                   #
        stop,                         # останавливаем движение после 3 секунд
    )

    # найти корзину и бросить куб
    find_bin_and_throw_box = (
        detect_line_color,            # определяем цвет полоски (для определения
        ↪  маршрута)
        stabilize_over_target_line,   # стабилизируемся над нужной полоской
        ↪  (определенного цвета)
        go_forward,                   # двигаемся вперед
        find_blue_bin,                # находим синюю корзину
        stabilize_over_blue_bin,      # стабилизируемся над корзиной
        ungrab_box,                   # отпускаем куб, чтобы он упал в корзину
    )

    # добраться до последней платформы (над которой обруч)
    go_to_final_platform = (
        go_forward,                   # двигаемся вперед
        wait_short,                   # ожидание 3 секунды
        stop,                         # остановка движения
        stabilize,                    # стабилизируем положение
        stabilize_on_second_arrow,    # стабилизируемся над второй стрелкой (цвет был
        ↪  ранее определен)
        go_forward,                   # двигаемся вперед
        find_orange_circle,           # находим оранжевый круг
        stabilize_over_orange_circle,# стабилизируемся над этим кругом
        stabilize,                    # окончательно стабилизируем курс
    )

    missions = (
        *go_to_box_platform,          # сначала нужно добраться до платформы с кубом
        *find_and_grab_box,           # затем найти и схватить куб
        *find_bin_and_throw_box,      # найти корзину и отпустить куб
        *go_to_final_platform,        # добраться до последней платформы
        surface,                      # и в конце всплыть
    )

    # задаем список действий миссий
    context.push_mission_list(missions)

    print("start")

```

```
    # в цикле проходим каждое действие для выполнения миссии,
    # а также выводим в консоль отладочную информацию
    while (True):
        mission = context.pop_mission()
        print('starting', mission.__name__, '\ttime:', context.time)
        while not mission():
            context.process()
            context.time += 1

        if context.get_missions_length() == 0:
            break

    print("done!")

    context.set_speed(0)
    context.set_depth(2.5)

    time.sleep(3)
```

Задача II.1.2.2. Задание 2 для электронщиков (100 баллов)

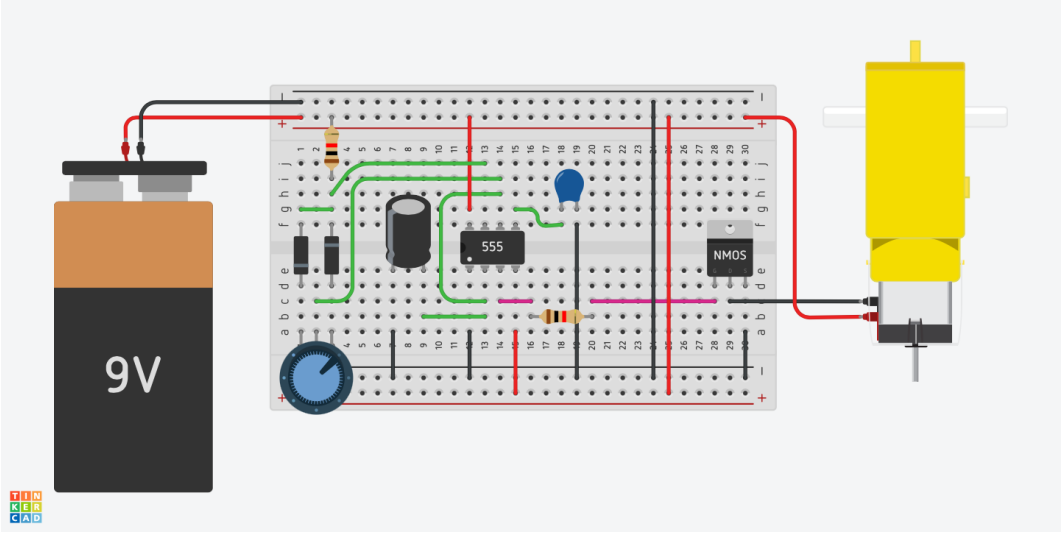
1. Дано (элементы в Thinkercad).
   - Динамик или пищалка.
   - Потенциометр.
   - Аналоговые компоненты: резисторы, конденсаторы, индуктивности, диоды, транзисторы.
   - Кнопки.
   - Элементы питания.
   - Логические микросхемы, находящиеся в разделе «Логика».
   - Макетные платы.
   - Таймер.
2. Необходимо, используя online-сервис Tinkercad (во избежание проблем с моделированием, рекомендуем пользоваться Google Chrome), создать проект электрического пианино. Основные требования:
   - Можно использовать только те компоненты, которые приведены в пункте 2.1.
   - Необходимо сделать примитивное пианино, способное генерировать от 5 до 12 нот, управляемых тактовыми кнопками.
   - Частотный диапазон нот должен соответствовать первой октаве фортепиано.
   - При нажатии на соответствующие кнопки, как и в нормальном пианино, должен издаваться звук правильной частоты (допускается округление до целых значений Гц).
   - Не разрешается использование микроконтроллеров, датчиков, контроллеров управления моторами, электрических приводов с H-мостами, а также генераторов сигналов и регулируемых источников питания.
   - Когда кнопки не нажаты посторонних звуков быть не должно.
3. Схемы, в которых будут использоваться неразрешенные компоненты, проверяться не будут. Участник получает за него 0 баллов.
4. Баллы за задачу начисляются следующим образом.

- 100 баллов — схема работает правильно в соответствие с требованиями. Не используются избыточные компоненты.
- 90 баллов — схема работает правильно в соответствие с требованиями. Используются избыточные компоненты. Соединительные проводники избыточны.
- 80 баллов — схема работает правильно в соответствие с требованиями. Не используются избыточные компоненты. Но частотный диапазон не соответствует норме.
- 50 баллов — схема работает частично. Издает звуки разной тональности.
- 30 баллов — схема работает частично. Издает какие-то звуки.
- 0 баллов — схема не работает даже частично.

5. В качестве решения данной задачи, необходимо прислать ссылку на ваш проект. Перед отправкой ссылки в ответ рекомендуем проверить работу моделирования. Инструкция по нахождению ссылки находится в 1 уроке данного модуля. Решения, ссылки на которые скопированы из адресной строки, проверяться не будут.

Решение

Данная схема является примером выполнения задачи на 100 баллов. Здесь представлены все элементы, которые можно использовать, без снижения баллов за их избыточное количество. Схема состоит из таймера (микросхема таймера NE555), а также полевого транзистора для управления нагрузкой (мотором). Для изменения скорости вращения мотора мы используем потенциометр и тем самым изменяем скважность сигнала, подаваемую на затвор полевого транзистора.



Ссылка на пример решенного задания:

`https://www.tinkercad.com/things/dK9AIVIrVtk-copy-of-pwmne555pinosa/editel?sharecode=6m4GdeLgPJ-ZTXaqcdv9XgtJIUgM0NBQV52OSbTF-GA`

Задача II.1.2.3. Задание 2 для программистов микроконтроллеров (100 баллов)

1. Дано:
   - Arduino UNO.
   - Клавиатура 4×4.
   - Символьный дисплей 16×2.
2. Используя online-сервис Tinkercad (во избежание проблем с моделированием, рекомендуем пользоваться Google Chrome), необходимо скопировать проект по ссылке: https://www.tinkercad.com/things/0DRRITK63Dt В данной схеме имеется микроконтроллер Arduino UNO, к которму подключены клавиатура 4×4 и символьный дисплей 16×2.
3. Необходимо запрограммировать Arduino UNO, чтобы выполнялись следующие требования:
   - Устройство должно представлять собой калькулятор, с помощью которого можно производить простые вычисления (складывать, вычитать, делить и умножать числа).
   - Функции кнопок:
     - цифры — числовой ввод.
     - A — сложение (+).
     - B — вычитание (-).
     - C — деление (÷).
     - D — умножение (×).
     - * — сброс.
     - # — равно (=).
   - На дисплее должно отображаться первое число (над которым производится математическая операция); символ, обозначающий математическую операцию; второе число, участвующее в математической операции.
   - После выполнения операции (при нажатии на кнопку «равно») результат является «первым числом», над которым пользователь будет производить математическую операцию (до нажатия кнопки сброса).
   - При нажатии на кнопку «сброс» очищаются оба числа и вид математической операции.
4. Баллы за задачу начисляются следующим образом:
   - Корректная работа одного вида математической операции — 15 баллов (максимум 60).
   - Функция сброса — 10 баллов.
   - Корректная обработка ошибок (программа должна быть способна корректно продолжать работу после возникновения ошибок, таких как деление на ноль или переполнение) — 30 баллов.

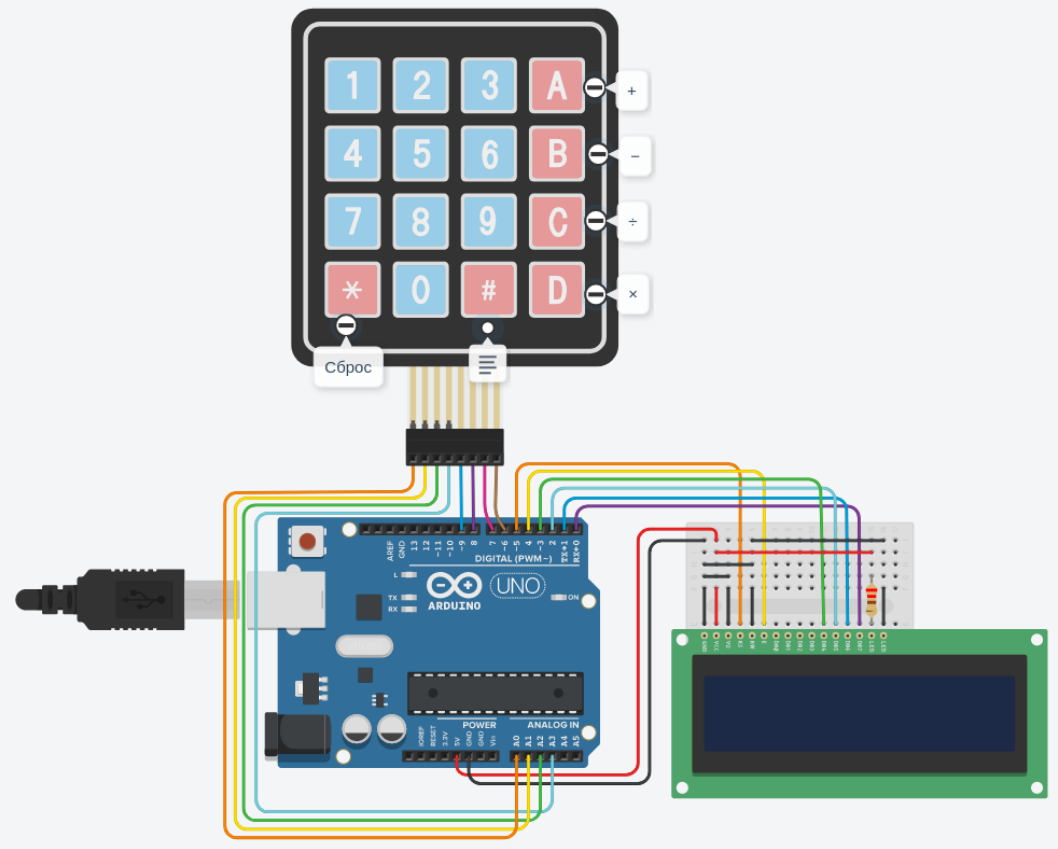
   Итого за задачу можно получить до 100 баллов. В случае модификации электрической схемы, задание не проверятся (0 баллов за задание).
5. В качестве решения данной задачи необходимо прислать ссылку на ваш проект. Перед отправкой ссылки в ответ рекомендуем проверить работу моделирования. Инструкция по нахождению ссылки находится в 1 уроке данного модуля. Решения, ссылки на которые скопированы из адресной строки, проверяться не будут.

Решение

Решение (NTI-Arduino-Task2-Calculator.ino) доступно по ссылке :

https://yadi.sk/d/KrZAbhWM9nDhAw

В данной схеме имеется микроконтроллер Arduino UNO, к которому подключены клавиатура 4×4 и символьный дисплей 16×2. Устройство представляет собой калькулятор.



Ссылка на схему TinkerCad:

https://www.tinkercad.com/things/8fSsNwQKS1C

Программный код для Arduino:

```
/* Калькулятор.
 *
 * Пользователь вводит первое число выражения
 * в верхней строке, после чего нажимает на
 * кнопку математической операции.
 *
 * После этого пользователь вводит второе число.
 *
 * Далее пользователь может нажать на кнопку "="
 * для вывода результата, или же снова нажать на
 * кнпоку математической операции, чтобы
 * продолжить вычисления над результатом.
 *
 * Также есть кнопка для сброса.
 */

```

```
// подключение библиотеки для дисплея
#include <LiquidCrystal.h>
// подключение библиотеки для клавиатуры
#include <Keypad.h>

// создание объекта для управления дисплеем
LiquidCrystal lcd(5, 4, 3, 2, 1, 0);

// кол-во строк и столбцов кнопок клавиатуры
const byte ROWS = 4;
const byte COLS = 4;

// установка соответствия между кнопками
// клавиатуры с символами на них
char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3', '+'},
  {'4', '5', '6', '-'},
  {'7', '8', '9', '/'},
  {'*', '0', '=', 'x'}
};

// задание портов для клавиатуры
byte rowPins[ROWS] = {A0, A1, A2, A3};
byte colPins[COLS] = {9, 8, 7, 6};

// создание объекта клавиатуры
Keypad customKeypad = Keypad(makeKeymap(hexaKeys),
                             rowPins, colPins, ROWS, COLS);

// доступные операции над числами
// (сложение, вычитание, деление, умножение)
enum operations {PLUS = 0, MINUS, DIVIDE, MULTIPLY};
// режимы работы (ожидание первого числа,
// ожидание второго числа, вывод результата)
enum modes {WAIT_FIRST = 0, WAIT_SECOND, RESULT};

int mode = WAIT_FIRST;  // текущий режим
int operation = 0;      // текущая операция

float num1 = 0;         // первое число
float num2 = 0;         // второе число
float result = 0;       // результат
int num_digit_count = 0;// счетчик цифр

char current_key;   // текущая нажатая кнопка

// функция очистки калькулятора
void clear_calculator() {
  // обнуление первого и второго числа
  num1 = 0;
  num2 = 0;
  // установка режима на ожидание первого числа
  mode = WAIT_FIRST;
  // обнуление счетчика цифр
  num_digit_count = 0;

  // очистка дисплея
  lcd.clear();
  lcd.setCursor(0, 0);
};
```

```
77
78  // перевод ASCII-символа в цифру
79  // (см. таблицу ASCII-кодов)
80  int ascii_to_int(int num) {
81    // если символ попадает в диапазон цифр
82    if (num >= 48 && num <= 57) {
83      // то возвращаем цифру, соответствующую символу
84      return num - 48;
85    } else {
86      // иначе возвращаем -1 (не цифра)
87      return -1;
88    }
89  }
90
91  // заполнение чисел пользователем
92  void fill_num(float &target_num) {
93
94    // если клавиша не нажата, или число слишком большое
95    if (!current_key || num_digit_count >= 14){
96      // то выходим из функции
97      return;
98    }
99
100   // получаем цифру из введенного символа
101   int num = ascii_to_int(current_key);
102
103   // если было введено число
104   if (num != -1) {
105     num_digit_count++;  // прибавляем счетчик цифр
106     lcd.print(num);     // выводим на дисплей цифру
107     target_num *= 10;   // сдвигаем десятичный разряд
108     target_num += num;  // добавляем цифру
109   }
110 }
111
112 // функция для вывода результата
113 void show_result() {
114   // очистка дисплея
115   lcd.clear();
116   result = 0;
117
118   // если было ожилание первого числа (второго еще нет),
119   // то результатом будет просто первое число
120   // (т. е. операций не было произведено)
121   if (mode == WAIT_FIRST) {
122     result = num1;
123   } else {
124
125     // иначе будет производить вычисления
126
127     // сложение
128     if (operation == PLUS) {
129       result = num1 + num2;
130     }
131
132     // вычитание
133     if (operation == MINUS) {
134       result = num1 - num2;
135     }
136
```

```
137       // умножение
138       if (operation == MULTIPLY) {
139         result = num1 * num2;
140       }
141
142       // деление
143       if (operation == DIVIDE) {
144         result = num1 / num2;
145       }
146
147     }
148
149     mode = RESULT;  // устанавливаем новый режим
150     num1 = result;  // первым числом становится результат
151     num_digit_count = 0;  // обнуляем счетчик цифр
152
153     // перевод курсора в начало верхней строки
154     lcd.setCursor(0, 0);
155     lcd.print('='); // вывод символа равенства
156     // перевод курсора в начало нижней строки,
157     // где будет отображен результат
158     lcd.setCursor(0, 1);
159
160     // проверки на ошибки
161
162     if (isnan(result)) {  // если не число (not a number)
163       lcd.print("error: nan");
164       return;
165     }
166
167     if (isinf(result)) {  // если бесконечность (infinity)
168       lcd.print("error: inf");
169       return;
170     }
171
172     // проверка на переполнение
173     if (result > 4294967040.0 ||
174         result <-4294967040.0 ) {
175       lcd.print("error: overflow");
176       return;
177     }
178
179     // вывод результата
180     lcd.print(result, DEC);
181   }
182
183   // установка математической операции
184   void set_operation(int new_operation) {
185     operation = new_operation;
186
187     // если первое число не было введено,
188     // то отобразим на его месте 0
189     if (num1 == 0) {
190       lcd.setCursor(0, 0);
191       lcd.print(0);
192     }
193
194     // если было введено второе число,
195     // то нужно произвести вычисления
196     // (это нужно, если подряд идет
```

```
197   // несколько математических операций,
198   // например, 12×34+56 можно будет посчитать,
199   // не нажимая кнопку "равно")
200   if (mode == WAIT_SECOND) {
201     // расчет и вывод результата
202     show_result();
203     // снова устанавливаем ожидание второго числа
204     // (первым числом стал результат)
205     mode = WAIT_SECOND;
206 
207     // очистка дисплея
208     lcd.clear();
209     lcd.setCursor(0, 0);
210     lcd.print(num1, DEC);
211     // обнуление второго числа
212     num2 = 0;
213     // установка курсора в начало нижней строки,
214     // куда будет вводиться второе число
215     lcd.setCursor(0, 1);
216 
217     // выход из функции
218     return;
219   }
220 
221   // для вывода знака текущей операции
222   // перемещаем курсор в конец верхней строки
223   lcd.setCursor(15, 0);
224   // вывод знака математической операции
225   lcd.print(current_key);
226 
227   // установка режима ожидания второго числа
228   mode = WAIT_SECOND;
229   // обнуление счетчика цифр
230   num_digit_count = 0;
231   // установка курсора в начало нижней строки
232   lcd.setCursor(0, 1);
233 }
234 
235 // функция, которая выполнится один раз в начале
236 void setup() {
237   // инициализация дисплея
238   lcd.begin(16, 2);
239   // включение отображения курсора
240   lcd.cursor();
241   // обнуление калькулятора
242   clear_calculator();
243 }
244 
245 // основной цикл программы
246 void loop() {
247   // получение текущей нажатой клавиши
248   current_key = customKeypad.getKey();
249 
250   // если сейчас отображается результат
251   // и при этом была нажата клавиша...
252   if (mode == RESULT && current_key) {
253     // очистка дисплея
254     lcd.clear();
255 
256     // если была нажата клавиша математической операции,
```

```
257       // то нужно произвести эту операцию над результатом
258       // (первым числом является результат)
259       if (current_key == '+' ||
260           current_key == '-' ||
261           current_key == 'x' ||
262           current_key == '/' ) {
263
264         // очистка дисплея
265         lcd.clear();
266         // установка курсора в начало первой строки
267         lcd.setCursor(0, 0);
268         // вывод первого числа (сейчас это результат
269         // от предыдущего расчета)
270         lcd.print(num1, DEC);
271         // обнуляем второе число
272         num2 = 0;
273       } else if (ascii_to_int(current_key) != -1) {
274         // если при показе результата было нажато число,
275         // то нужно обнулить калькулятор
276         // (т. е. начался ввод новых чисел)
277         clear_calculator();
278       }
279
280       // установка режима ожидания первого числа
281       mode = WAIT_FIRST;
282     }
283
284     // если была нажата клавиша
285     if (current_key) {
286
287       // проверка нажатой клавиши
288       // на соответствие действию (сброс, равно)
289       // или математической операции
290
291       if (current_key == '*') {
292         clear_calculator();
293         return;
294       }
295
296       if (current_key == '+') {
297         set_operation(PLUS);
298         return;
299       }
300
301       if (current_key == '-') {
302         set_operation(MINUS);
303         return;
304       }
305
306       if (current_key == 'x') {
307         set_operation(MULTIPLY);
308         return;
309       }
310
311       if (current_key == '/') {
312         set_operation(DIVIDE);
313         return;
314       }
315
316       if (current_key == '=') {
```

```
317         show_result();
318         return;
319       }
320     }
321
322     // если ожидается первое число,
323     // то заполняем именно его - num1
324     if (mode == WAIT_FIRST) fill_num(num1);
325
326     // если же ожидается второе число,
327     // то соответственно заполняем num2
328     if (mode == WAIT_SECOND) fill_num(num2);
329   }
```

Командная часть

Задача II.2.1. Командное задание (1 балл)

1. Важным навыком для инженеров является разработка технической документации. Когда делаешь какую-то поделку для себя, то можно не разрабатывать документацию. Но когда в процесс проектирования включено хотя бы несколько человек, когда есть заказчик или когда разрабатываемое устройство состоит из большого числа деталей и подсистем, тогда без документации не обойтись.
2. Одним из важнейших документов на начальной стадии разработки является Техническое задание (ТЗ). ТЗ обычно разрабатывается инженерами совместно с заказчиком. И именно по ТЗ заказчик потом принимает опытный образец или готовый продукт.
3. В ТЗ отражены все основные требования к устройству, поэтому чтобы написать ТЗ необходимы усилия всех видов разработчиков (электронщики, конструкторы, программисты и др.).
4. Чтобы не было разночтений в процессе разработки ТЗ и другой конструкторской документации, у нас в стране были разработаны специальные стандарты — ГОСТы. Поэтому в России конструкторская документации должна этим ГОСТам соответствовать. Стандарты сделаны прежде всего для того, чтобы облегчить жизнь разработчикам и всем вовлеченным в процесс проектирования и производства специалистам.
5. В рамках командной задачи мы предлагаем вам разработать ТЗ к существующему телеуправляемому необитаемому подводному аппарату(https://www.saabseaeye.com/solutions/underwater-vehicles/falcon). Представьте, что заказчик показывает вам картинку с аппаратом и дает краткое описание того, как должен выглядеть и работать робот, а вы должны с его слов разработать ТЗ на опытно-конструкторскую работу, результатом которой является опытный образец.
6. В качестве существующего аппараты мы выбрали аппарат Falcon компании Seaeye, это один из самых известных подводных роботов в мире, признанный стандарт качестве в своей области применения. Можете использовать любую информацию, которую найдете о нем, дня разработки ТЗ. Для написания ТЗ необходимо взять версию Falcon, рассчитанную на 300 м, без полезной нагрузки.

7. Ваше ТЗ будет оцениваться по определенным критериям. Мы подготовили специальный лист оценки(https://www.dropbox.com/s/gdrsltcwzj41e9k/%D0%9B%D0%B8%D1%81%D1%82%20%D0%BE%D1%86%D0%B5%D0%BD%D0%BA%D0%B8%20%D0%A2%D0%97.xlsx?dl=0), который поможет вам правильно разработать ТЗ.
8. В листе оценки есть формальные критерии (наличие перечня терминов, наличие перечня сокращений и т.п), а есть содержательные критерии (характеристики изделия, конструктивные требования и т.п.). И указаны пункты ГОСТов, на которые мы будем опираться при проверке ваших работ.
9. ТЗ необходимо загрузить в формате *.doc или *.docx. Размер файла должен быть менее 5 Мб. Все тексты будут проверяться на плагиат. Допустимая норма заимствований 20%.

Решение

Далее приведен лист оценки и решение одной из команд, набравшей большее количество баллов (182 балла из 200).

№	Раздел	Критерий	ГОСТ	Балл	Макс. балл
1	Оформление документа	Титульный лист	п. 6.2.2 ГОСТ 15.016-2016	3	3
2		Нумерация разделов и подразделов	п. 4.1 ГОСТ 2.105	2	2
3		Стилистика	п. 4.2.3 и 4.2.3 ГОСТ 2.105	7	8
4		Последний лист	п. 6.2.3 ГОСТ 15.016-2016	2	2
5		Поля и абзацы	п. 3.6 ГОСТ 2.105	1	2
6		Наличие перечня терминов	п. 4.2.2. ГОСТ 2.105	1	1
7		Наличие перечня сокращений	п. 4.2.6. ГОСТ 2.105	0	1
8	Общая информация	Наименование ОКР	п. 6.1.2 ГОСТ 15.016-2016	2	2
9		Цель выполнения ОКР	п. 6.1.3 ГОСТ 15.016-2016	2	2
10		Наименование и обозначение изделия	п. 6.1.3 ГОСТ 15.016-2016	2	2

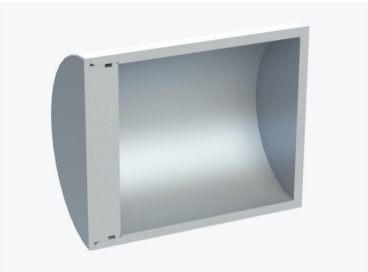
11	Технические требования к изделию	Состав изделия	п. 6.1.4.1 ГОСТ 15.016-2016	8	10
12		Характеристики изделия	п. 6.1.4.2 ГОСТ 15.016-2016	10	10
13		Технические характеристики изделия	п. 6.1.4.2 ГОСТ 15.016-2016	10	10
14		Конструктивные требования	п. 6.1.4.3 ГОСТ 15.016-2016	10	10
15		Требования к воздействию климатических условий	п. 6.1.4.5 ГОСТ 15.016-2016	5	5
16		Требования надежности	п. 6.1.4.6 ГОСТ 15.016-2016	5	5
17		Эргономические требования к организации и средствам деятельности человека-оператора	п. 6.1.4.7 ГОСТ 15.016-2016	5	5
18		Эксплуатационные режимы	п. 6.1.4.8 ГОСТ 15.016-2016	5	5
19		Численность, состав и квалификация обслуживающего персонала	п. 6.1.4.8 ГОСТ 15.016-2016	4	5
20		Состав инструментов, СИ и приспособлений для проведения технического обслуживания и ремонта, сборки и разборки изделия	п. 6.1.4.8 ГОСТ 15.016-2016	5	5
21	Технико-экономические требования	Сравнительные технико-экономические характеристики	п. 6.1.5 ГОСТ 15.016-2016	6	10
22		Стоимости выполнения ОКР	п. 6.1.5 ГОСТ 15.016-2016	5	5
23		Трудоемкость разработки	п. 6.1.5 ГОСТ 15.016-2016	5	5
24	Требования к математическому, программному и информационно-лингвистическому обеспечению	Требования к программному обеспечению	п. 6.1.6.4 ГОСТ 15.016-2016	5	10
25		Функции и задачи ПО	п. 6.1.6.4 ГОСТ 15.016-2016	10	10
26		Состав ПО		10	10
27	Требования к сырью, материалам и КИПМ	Ограничение номенклатуры применяемых материалов, КИМП и других покупных изделий	п. 6.1.7 ГОСТ 15.016-2016	4	5
28		Требования к материалам и КИМП	п. 6.1.7 ГОСТ 15.016-2016	8	10
29	Требования к учебно-тренировочным средствам	Требования к комплексным и специализированным тренажерам	п. 6.1.9 ГОСТ 15.016-2016	5	5
30		Требования к моделям, макетам, стендам, учебно-техническим плакатам	п. 6.1.9 ГОСТ 15.016-2016	5	5

31	Специальные требования	Требования к виду и составу специального оборудования и оснастки, необходимых для обеспечения эксплуатации и технического обслуживания изделия	п. 6.1.10 ГОСТ 15.016-2016	10	10
32		Требования к методам испытаний изделия при разработке	п. 6.1.10 ГОСТ 15.016-2016	5	5
33	Дополнительно	Объем документа (в словах): 1-600 слов - 0 баллов 601-700 слов - 1 балл 701-800 слов - 2 балла 801-900 слов - 3 балла 901-1000 слов - 4 балла более 1000 слов 5 баллов		5	5
34		Адекватность ТЗ (соответствие реальному изделию)		10	10
Сумма баллов				**182**	**200**

Задача II.2.2. Задание 1 для конструкторов (100 баллов)

1. Дано:
   - Алюминиевый пруток длиной 60 мм, диаметром 60 мм.
   - Алюминиевый пруток длиной 40 мм, диаметром 60 мм.
   - Резиновое уплотнительное кольцо сечением 2,5 мм и произвольного диаметра (вам необходимо самостоятельно подобрать).

   Прутки считать проточенными и подогнанными в размер.
2. Необходимо спроектировать герметичный корпус, который состоит из колбы с глухим дном и крышки. Пример приведен на картинке ниже (модель корпуса без резьбового соединения с канавкой под радиальное уплотнение). При этом крышка и колба должны соединяться с применением резьбового соединения. При проектировании предусмотрите возможность разбора корпуса (выкручивания крышки) без применения инструмента (руками).



Для герметизации необходимо использовать уплотнительное кольцо сечением 2,5 мм. Толщина стенок и дна колбы должна быть минимум 2 мм.
В корпус должен помещаться и извлекаться куб со стороной 33 мм, обязательно оставьте зазор до внутренних стенок колбы.

3. Участникам необходимо правильно рассчитать:
   - Диаметр уплотнительного кольца — 10 баллов.
   - Ширину канавки в крышке — 10 баллов.
   - Диаметр для посадки кольца в крышке — 10 баллов.
   - Внутренний диаметр колбы в месте герметизации — 10 баллов.

   Для расчетов необходимо воспользоваться ГОСТ 9833-73(https://plastinfo.ru/content/file/gosts/24fd56b18758.pdf) "Кольца резиновые уплотнительные круглого сечения для гидравлических и пневматических устройств". Особое внимание при изучении ГОСТа уделите приложению, а также (с.55).
4. Сделайте чертеж колбы — от 0 до 15 баллов
   - Должен содержать разрез детали.
   - Должен быть сохранен в формате .pdf.
   - Может быть выполнен в любой программе.
   - Должны быть проставлены все размеры, необходимые для изготовления детали, а также шероховатости поверхности и допуски.
5. Сделайте чертеж крышки — от 0 до 15 баллов
   - Должен содержать разрез детали.
   - Должен быть сохранен в формате .pdf.
   - Может быть выполнен в любой программе.
   - Должны быть проставлены все размеры, необходимые для изготовления детали, а также шероховатости поверхности и допуски.

   Исчерпывающие требования к чертежам указаны в ГОСТ 2.052-2006(http://docs.cntd.ru/document/1200045035) Тщательно изучите пункт «Нормативные ссылки», при переходе по гиперссылкам вы найдете ответы на все вопросы по оформлению чертежей.
   ГОСТ 2.311-68(http://docs.cntd.ru/document/gost-eskd-2-311-68) «ИЗОБРАЖЕНИЕ РЕЗЬБЫ»
   Для основной надписи используйте форму 2а, ее нужно также заполнить в соответствии с ГОСТами.
   Предельные отклонения и допуски указывайте только там, где это необходимо. Строгое соответствие всем пунктам ГОСТа 2.052-2006 и ГОСТам, на которые он ссылается, не нужно, достаточно соблюдение тех, которые от вас требуются (Должны быть проставлены все размеры, необходимые для изготовления детали, а также шероховатости поверхности и допуски.)
   ГОСТ 24705-2004 (http://docs.cntd.ru/document/1200038934) «Основные нормы взаимозаменяемости РЕЗЬБА МЕТРИЧЕСКАЯ Основные размеры»
   ГОСТ 11708-82 (http://docs.cntd.ru/document/1200011582) «РЕЗЬБА Термины и определения»
   ГОСТ 10549-80 (http://docs.cntd.ru/document/1200012239)«ВЫХОД РЕЗЬБЫ Сбеги, недорезы, проточки и фаски»
6. Сделайте 3Д-модель сборки крышки и колбы — от 0 до 30 баллов
   - Должна быть выполнена в любой программе. Рекомендуем использовать Autodesk Inventor.
   - Должна быть сохранена в формате .step.
   - Должна быть выполнена проточка в колбе для внутренней резьбы.
   - В деталях должны быть предусмотрены все необходимые фаски и скругления острых граней для безопасного использования

7. Решением данной задачи будет являться 3 файла:
   - pdf-файл с чертежом колбы;
   - pdf-файл с чертежом крышки;
   - step-файл с моделью сборки колба + крышка.

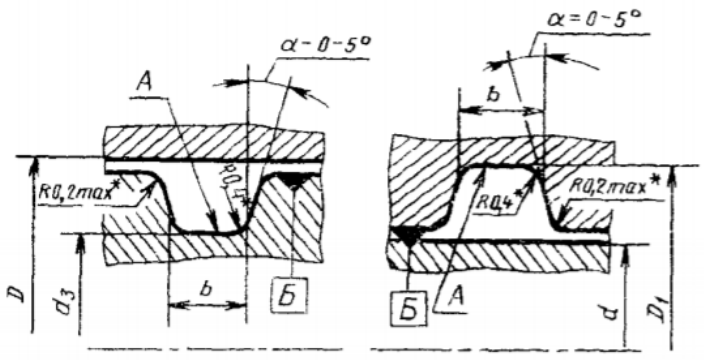
   Файлы необходимо сжать в zip-архив и приложить к ответу. В ответе необходимо указать результаты расчетов из пункта 1.3 в мм. Пример:
   Диаметр уплотнительного кольца = 1.
   Ширину канавки в крышке = 2.
   Диаметр для посадки кольца в крышке = 3.
   Внутренний диаметр колбы в месте герметизации = 4.

Решение

1. Подбираем необходимый диаметр уплотнительного кольца:
   - Исходя из ограничений по толщине стенки, размеров заготовок и в корпус должен помещаться и извлекаться куб со стороной 33 мм из дано: $60 - 2 \cdot 2 = 56$ мм — максимальный внутренний диаметр колбы, $d = a\sqrt{2} = 33 \cdot 1,4 = 46,2$ мм — минимальный внутренний диаметр колбы (формула описанной окружности около квадрата), вторая цифра в типоразмере кольца, далее D, должна быть 47 — 56 мм.
   - Открываем ГОСТ 9833-73 «Кольца резиновые уплотнительные круглого сечения для гидравлических и пневматических устройств» листаю до «Кольца уплотнительные сечением 2,5».
   - На стр. 6 определяю, что любой типоразмер кольца, от 043-047-25 до 052-056-25 подходит для проектирования корпуса, но исходя из логики «чем больше корпус, тем больше в него можно положить чего-либо для защиты этого чего-то от контакта с водой», выбираю для дальнейшего проектирования кольцо 052-056-25 с диаметром 51 мм. Диаметр уплотнительного кольца = 51 мм + 10 баллов.
2. На стр. 29 (28) ГОСТ 9833-73 «Кольца резиновые уплотнительные круглого сечения для гидравлических и пневматических устройств» нахожу таблицу с расчетами для кольца сечением 2,5 мм и вижу 2 чертежа:

С 28 ГОСТ 9833—73

Посадочные места под кольца с диаметром сечения d_2=2,5 мм



* Размер обеспечивается инструментом

Черт 7

Смотрим на изображение с образцом корпуса на странице индивидуального задания для конструкторов на Stepik'е и понимаем, что нас интересует только левая картинка, т.к. установка кольца происходит в крышке (чертеж слева), а не на распор в корпусе (чертеж справа).

3. По таблице ниже находим D, d_3, b напротив 052-056-25 в столбце «Неподвижное соединение»:
   $D = 56$ мм $d_3 = 52,3$ мм $b = 3,6$ мм + 30 баллов.
   D — внутренний диаметр колбы в месте герметизации.
   b — ширина канавки в крышке.
   d_3 — диаметр для посадки кольца в крышке.

Расчет окончен.

Диаметр уплотнительного кольца 51 мм — 10 баллов.

Ширину канавки в крышке 3,6 мм — 10 баллов.

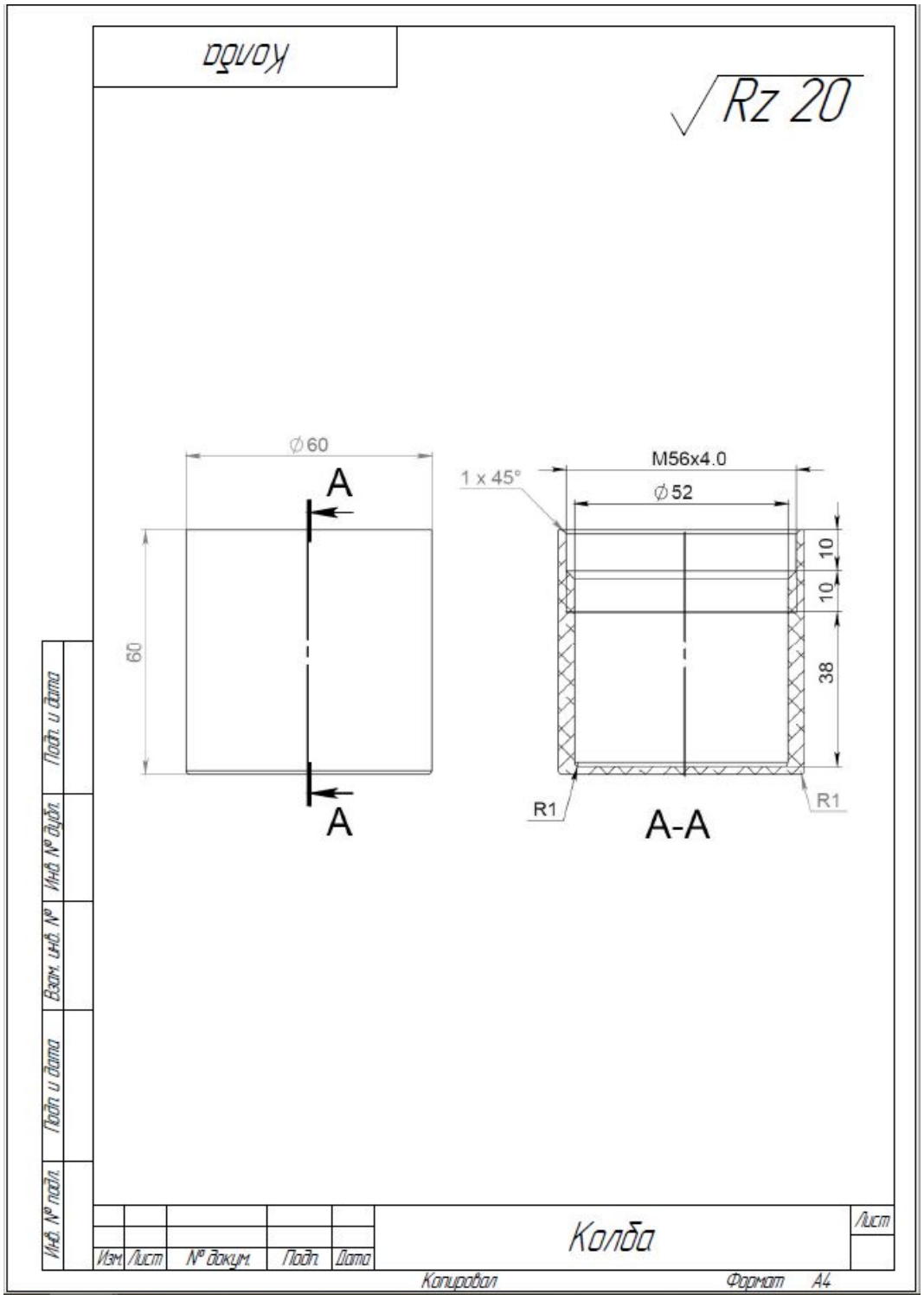
Диаметр для посадки кольца в крышке 52,3 мм — 10 баллов.

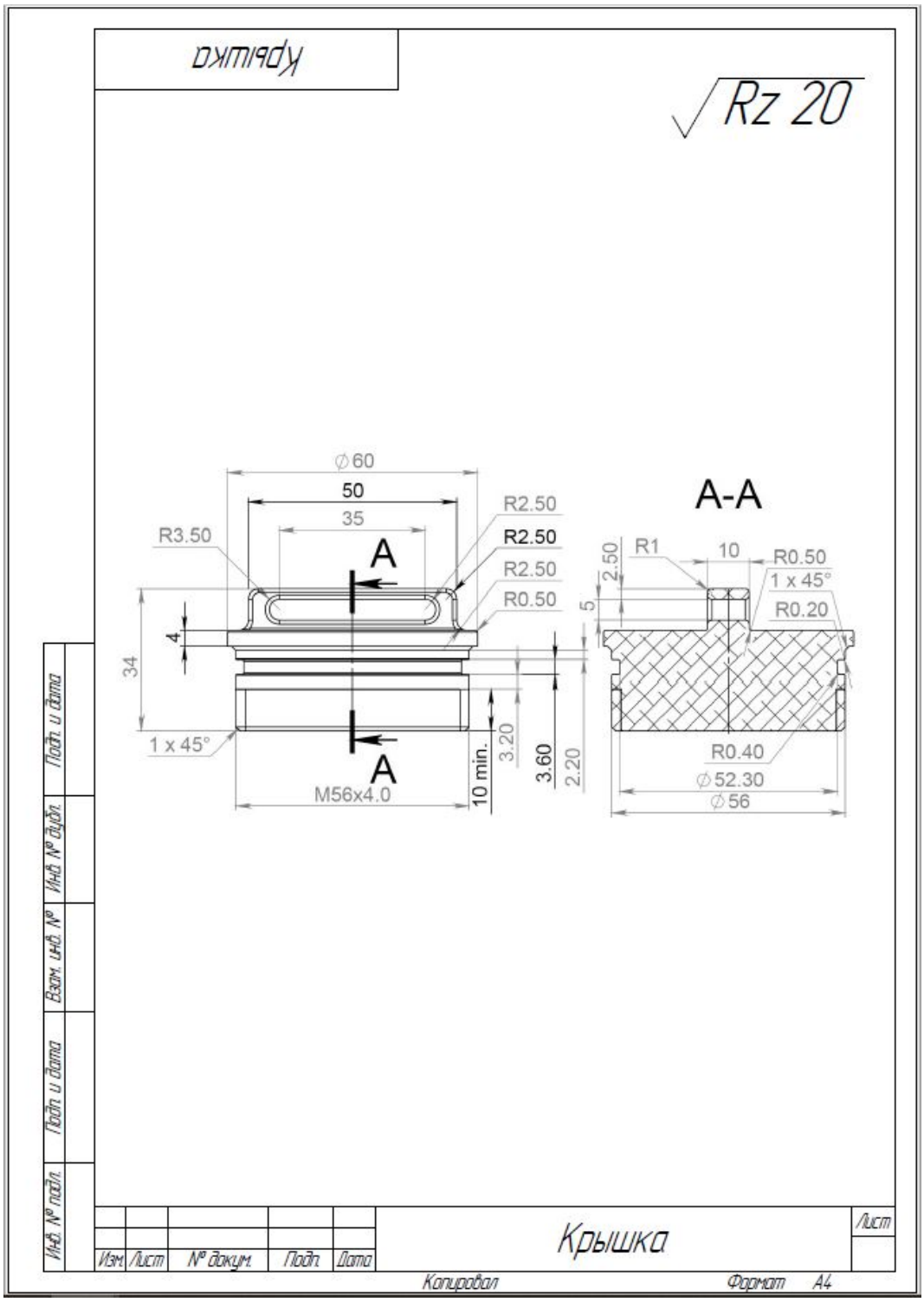
Внутренний диаметр колбы в месте герметизации 56 мм — 10 баллов.

Чертеж.

Рассчитанные по ГОСТу величины являются гарантом герметичности корпуса, при построении чертежей эти размеры нужно обязательно использовать. Построение и оформление чертежа выполняется по ГОСТ 2.052-2006

Пример:


Колба
Rz 20
∅60
A
A
60
M56x4.0
1 x 45°
∅52
10
10
38
R1
R1
A-A
Подп. и дата
Инв. № дубл.
Взам. инв. №
Подп. и дата
Инв. № подл.
Изм
Лист
№ докум.
Подп.
Дата
Колба
Лист
Копировал
Формат А4



3Д-модель сборки крышки и колбы

3д модели частей герметичного корпуса строятся отдельно в любой CAD-программе, согласно чертежам, а затем объединяются в сборку.

Задача II.2.3. Задание 2 для конструкторов (100 баллов)

1. Дано:
   - Алюминиевый пруток длиной 180 мм, диаметром 100 мм.

- Алюминиевый пруток длиной 50 мм, диаметром 180 мм.
- Резиновое уплотнительное кольцо сечением 2,5 мм произвольного диаметра (вам необходимо самостоятельно подобрать) Прутки считать проточенными и подогнанными в размер.

2. Вам предстоит спроектировать герметичный корпус, который состоит из колбы с глухим дном и крышки. Он будет использоваться под водой. Дополнительные баллы можно получить за спроектированную систему крепления крышки к корпусу (система крепления: отверстия, крепежные элементы, дополнительные детали. Что-то может быть частью колбы или крышки (детали — не идеальные тела вращения, допускаются вырезы, пазы, ушки и т. д., главное — не нарушить герметичность), это необходимо для фиксации герметизации, чтобы при ситуации, когда давление внутри корпуса больше давления окружающей среды, крышку не выдавило и не нарушилась герметизация. Несмотря на то, что система крепления не ограничена в использовании материалов (прутки только для колбы и крышки) и технологий изготовления (не задумывайтесь над тем, как это изготовить), не стоит делать ее громоздкой. Также при проектировании система должна считаться с законами физики и механики, не запрещается перенимать принципы работы готовых решений.
   Пример герметичного корпуса с системой крепления приведен на картинке ниже, однако, такая система крепления (стягивающий хомут) будет оцениваться в 0 баллов.

   Для герметизации необходимо использовать уплотнительное кольцо сечением 2,5 мм. Минимально допустимая толщина стенок и дна колбы — 3 мм. Минимальное расстояние между двумя параллельными поверхностями, которые испытывают нагрузки, 1,5 мм.
   В корпус должен помещаться и извлекаться один из кубов: куб со стороной 45 мм или 63 мм, минимальный зазор между кубом и стенками составляет $1 \pm 0{,}2$ мм.
3. Участникам необходимо правильно рассчитать в мм:
   - Диаметр уплотнительного кольца — 5 баллов.
   - Ширину канавки под кольцо — 5 баллов.
   - Глубину канавки под кольцо — 5 баллов.

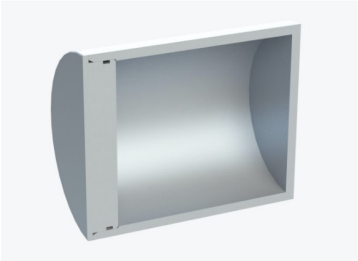
   Для расчетов необходимо воспользоваться ГОСТ 9833-73 (https://plastinfo.ru/content/file/gosts/24fd56b18758.pdf) «Кольца резиновые уплотнительные круглого сечения для гидравлических и пневматических устройств». Особое внимание при изучении ГОСТа уделите приложению.
4. Сделайте чертеж колбы — от 0 до 20 баллов
   - Должен содержать разрез детали;
   - Должен быть сохранен в формате .pdf;
   - Может быть выполнен в любой программе;

- Должны быть проставлены все размеры, необходимые для изготовления детали, а также шероховатости поверхности и допуски.

5. Сделайте чертеж крышки — от 0 до 20 баллов
   - Должен содержать разрез детали;
   - Должен быть сохранен в формате .pdf;
   - Может быть выполнен в любой программе;
   - Должны быть проставлены все размеры, необходимые для изготовления детали, а также шероховатости поверхности и допуски.

   Исчерпывающие требования к чертежам указаны в ГОСТ 2.052-2006 (http://docs.cntd.ru/document/1200045035) Тщательно изучите пункт «Нормативные ссылки», при переходе по гиперссылкам вы найдете ответы на все вопросы по оформлению чертежей.
   ГОСТ 2.311-68 (http://docs.cntd.ru/document/gost-eskd-2-311-68) «ИЗОБРАЖЕНИЕ РЕЗЬБЫ»
   Для основной надписи используйте форму 2а, ее нужно также заполнить в соответствии с ГОСТами.
   Предельные отклонения и допуски указывайте только там, где это необходимо. Строгое соответствие всем пунктам ГОСТа 2.052-2006 и ГОСТам, на которые он ссылается, не нужно, достаточно соблюдение тех, которые от вас требуются (Должны быть проставлены все размеры, необходимые для изготовления детали, а также шероховатости поверхности и допуски.)
6. Сделайте 3Д-модель сборки корпуса: колба, крышка, крепление крышки к корпусу, крепеж (винты, гайки) и пр. — от 0 до 45 баллов
   - Сборка должна быть выполнена в любой CAD программе. Рекомендуем использовать AutodeskInventor;
   - Сборка должна быть сохранена в формате .step;
   - Сборка должна содержать все элементы корпуса и системы крепления (винты гайки, шпильки обязательны, если они предусмотрены в системе);
   - Указана резьба, где это необходимо;
   - Все детали должны быть сопряжены, а система крепления должна выполнять свои функции (не нужно разносить компоненты) для наглядности;
   - В деталях должны быть предусмотрены все необходимые фаски и скругления острых граней для безопасного использования.

   Пример сборки корпуса с радиальным уплотнением без системы крепления, разрез.



7. Решением данной задачи будет являться 3 файла:
   - pdf-файл с чертежом колбы;
   - pdf-файл с чертежом крышки;

- .step файл (один файл) с моделью сборки корпуса: колба, крышка, крепление крышки к корпусу, крепеж (винты, гайки) и т. д.

Файлы необходимо сжать в zip-архив и приложить к ответу. В ответе необходимо указать обозначение типоразмера кольца, размер куба и результаты расчетов из пункта 1.3 в мм.

Примеры

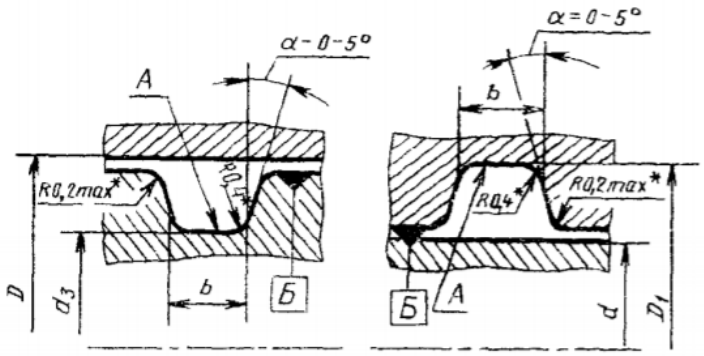
Кольцо ***-***-25, куб 63мм
Диаметр уплотнительного кольца = 1
Ширина канавки под кольцо = 2
Глубина канавки под кольцо = 3

Решение

1. Подбираем необходимый диаметр уплотнительного кольца:
   - Исходя из ограничений по толщине стенки и размеров заготовок, в корпус должен помещаться и извлекаться куб со стороной 45 мм или 63 мм из дано: 100 мм — максимальный внешний диаметр колбы, $d = a\sqrt{2} = 45 \cdot 1,4 = 63$ мм — минимальный внутренний диаметр колбы (формула описанной окружности около квадрата), вторая цифра в типоразмере кольца, далее D, должна быть 64 — 108 мм.
   - Открываем ГОСТ 9833-73 «Кольца резиновые уплотнительные круглого сечения для гидравлических и пневматических устройств» листаю до «Кольца уплотнительные сечением 2,5»
   - На стр. 6 определяю, что любой типоразмер кольца, от 060-064-25 до 102-108-25 подходит для проектирования корпуса, но исходя из логики «чем больше корпус, тем больше в него можно положить чего-либо для защиты этого чего-то от контакта с водой», выбираю для дальнейшего проектирования кольцо 098-102-25 с диаметром 96 мм. Диаметр уплотнительного кольца = 96 мм + 5 баллов.
2. На стр. 29 (28) ГОСТ 9833-73 "Кольца резиновые уплотнительные круглого сечения для гидравлических и пневматических устройств"нахожу таблицу с расчетами для кольца сечением 2,5 мм и вижу 2 чертежа:

С 28 ГОСТ 9833—73

Посадочные места под кольца с диаметром сечения d_2=2,5 мм



* Размер обеспечивается инструментом

Черт 7

Смотрим на изображение с образцом корпуса на странице индивидуального задания для конструкторов на Stepik'е и понимаем, что нас интересует только левая картинка, т. к. установка кольца происходит в крышке (чертеж слева), а не на распор в корпусе (чертеж справа).

3. По таблице ниже находим напротив 098-102-25 в столбце «Неподвижное соединение»:
   $b = 3,6$ мм + 5 баллов.
   b — ширина канавки в крышке.
   $\frac{D-d_3}{2} = 1,85$ мм глубина канавки в крышке + 5 баллов.

Расчет окончен.

Диаметр уплотнительного кольца 96 мм— 5 баллов.

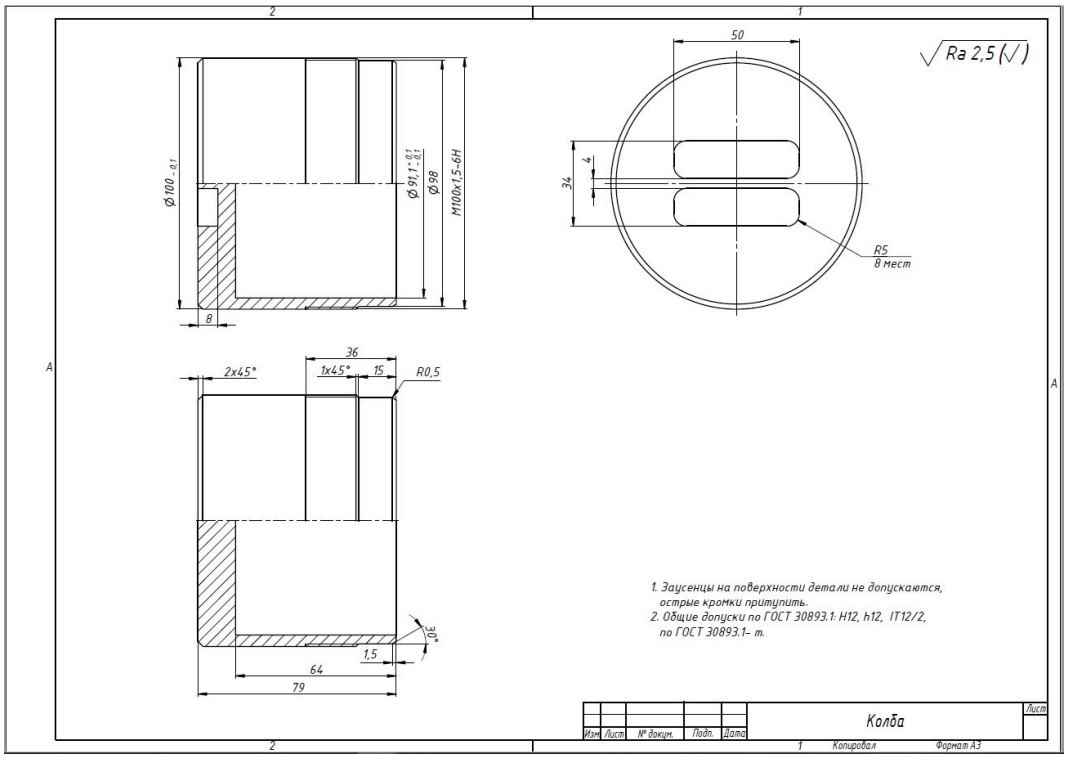
Ширину канавки в крышке 3,6 мм — 5 баллов.

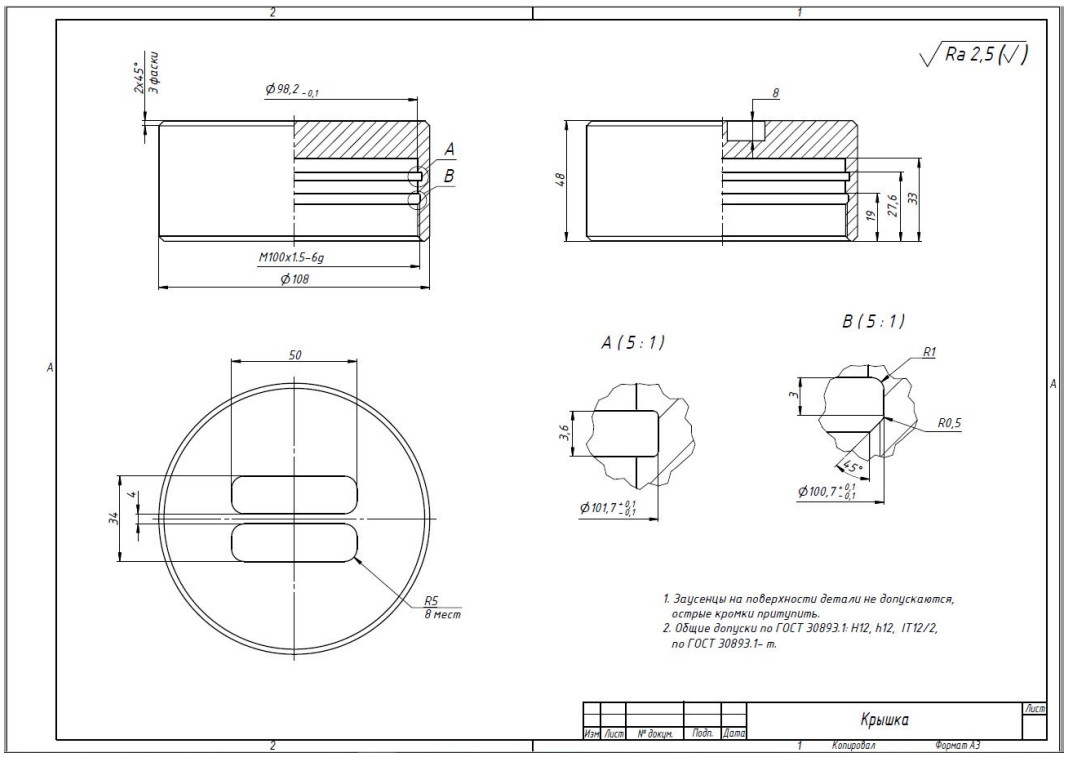
Глубина канавки в крышке 1,85 мм — 5 баллов.

Чертеж.

Рассчитанные по ГОСТу величины являются гарантом герметичности корпуса, при построении чертежей эти размеры нужно обязательно использовать. Построение и оформление чертежа выполняется по ГОСТ 2.052-2006

Пример:





3Д-модель сборки крышки и колбы 3Д модели частей герметичного корпуса строятся отдельно в любой CAD-программе согласно чертежам, а затем объединяются в сборку.