2. ВТОРОЙ ЭТАП

Задачи второго этапа

Наш трек, предполагает довольно обширные знания в области разработки программного обеспечения. При этом важно отметить что именно "программирование" занимает тут не очень большую часть, а основная роль — это комплексное обслуживание и администрирование различных систем.

Одна из важных компетенций для нашего трека — это работа с операционными системами семейства Linux и мета операционной системой Robot Operation System (ROS).

Robot Operating System (ROS) - это гибкая платформа (или другими словами фреймворк) для разработки программного обеспечения специально для роботов. ROS включает в себя множество разнообразных инструментов, библиотек и определенных правил, целью которых является упрощение и унификация задач разработки ПО роботов.

ROS была создана чтобы стимулировать совместную разработку программного обеспечения робототехники. Каждая отдельная команда может работать над одной конкретной задачей, но использование единой платформы, позволяет всему сообществу получить и использовать эти результаты для своих проектов.

В текущий момент все что связано с ROS, работает только под управлением Linux. Таким образом для решения задач нам необходимо установить Linux.

Установка Linux

Один из самых распространенных дистрибутивов Linux это дистрибутив Ubuntu. Именно этот дистрибутив мы и рассмотрим. Выбирайте версию **Ubuntu 18.04.1 LTS**.

Вы можете установить Linux несколькими способами.

1. Второй операционной системой, при загрузке вашего компьютера вы выберите какую ОС загружать.
   (a) `https://www.ubuntu.com/download/desktop` Скачать, записать на CD или flash и далее действовать по инструкциям инсталлятора. Это один из самых сложных и довольно радикальных способов. Подробная инструкция `https://help.ubuntu.ru/wiki/ubuntu_install`
2. Запустить вторую ОС через виртуализацию. Для windows и MacOS это может быть программа **VirtualBox**.
3. Для MacOS можно использовать бесплатную версию Parallels Desktop Lite (`https://itunes.apple.com/ru/app/parallels-desktop-lite/id1085114709?mt=12`). При установке только Linux версия не требует оплаты.

4. Для Windows 10, есть возможность запустить Ubuntu встроенными средствами, через магазин приложений Microsoft Store.

Самый простой способ начать работать с Linux, это установить программу виртуализации **VirtualBox**. Это программа позволит запустить "гостевую"ОС на уже работающую Windows.

Более подробно и с картинками можно прочитать по ссылке `https://guides.hexlet.io/ubuntu-linux-in-windows/` Часть для VirtualBox.

Краткое описание примера установки Linux на VirtualBox

1. Устанавливаем VirtualBox
2. Скачиваем дистрибутив (*.ico) файл.
3. Запускаем Виртуальную машину, указываем диск и следуем инструкциям инсталлятора.
4. Важно выполнить настройку единого буфера обмена, иначе работать будет не так удобно.

Если вы установили Linux впервые, очень рекомендуем пройти курс по работе с командной строкой от компании Хакслет `https://ru.hexlet.io/courses/cli-basics`. Работа с командной строкой может вам показаться довольно "архаичным" занятием, но это научит вас работать с любым Линуксом, при этом вне зависимости — установлен он у вас, находиться на удаленном компьютере или даже в космосе.

Установка ROS

Далее вам необходимо установить ROS. Для дистрибутива Ubuntu, есть довольно простой способ это сделать — это установить все из пакетов (Для других ОС, например Raspbian, которая по умолчанию ставить на Raspberry вам предаться компилировать все пакеты, такая установка занимает около 3 часов)

Официальная страница с инструкцией `http://wiki.ros.org/melodic/Installation/Ubuntu`

Введение в ROS на русском языке `http://docs.voltbro.ru/starting-ros/`

Запустим терминал. (Ctrl-Alt-T) или найдем его в меню запуска.

Пакеты ROS не включены в список пакетов доступных для установи в "стандартном" дистрибутиве Ubuntu. Поэтому нам надо добавить и настроить новый репозиторий.

1. Добавим репозиторий пакетов ROS в список (выполним команду в консоли)
   sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
2. Добавим ключи нового репозитория
   sudo apt-key adv –keyserver hkp://ha.pool.sks-keyservers.net:80 –recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
3. Обновим список пакетов sudo apt update
4. Установим все пакеты одной командой (это может занять несколько минут)
   sudo apt install ros-melodic-desktop-full

Инсталяция законченна, осталось произвести последние настройки

Добавим все необходимые директории в пути для доступа
echo "source /opt/ros/melodic/setup.bash">> ∼/.bashrc
source ∼/.bashrc

Чтобы изменения конфигурации начали работать, перезапустим терминал.

Запустим roscore чтобы убедиться что все работает.
roscore

Настройка завершена. У нас есть Линукс, есть ROS можем переходить к разбору задач. Все ссылки есть в описании к ролику.

Ссылки

`https://www.virtualbox.org/wiki/Downloads`

`https://www.ubuntu.com/download/desktop`

3.1. Управление роботом в симуляторе turtlesim

Задача 3.1.1. (30 баллов)

Задача проверяет базовые навыки работы с Robot Operating System, и системой управления колесных роботов. Решением задачи является присланный python скрипт. От одной команды принимается только 2 решения (две попытки)!

Выполнение задания:

1. Установить ROS (ros-melodic-desktop-full) в используя пакеты на дистрибутив Ubuntu. Инструкция `http://wiki.ros.org/melodic/Installation/Ubuntu`
2. Запустить и настроить ROS, изучить базовые принципы (Книжка "Введение в ROS" `http://docs.voltbro.ru/starting-ros/`)
3. Установить изучить пакет turtlesim `http://wiki.ros.org/turtlesim`
4. Написать python скрипт, который перемещает черепашку по квадрату с со сторонами 3 метра, с линейной скоростью 0.5 м/с, и поворачивает со скорость 0.5 рад/с. Черепаха после перемещения по всем вершинам квадрата должны вернуться в исходное положение (вершина, направление)
    (a) Старт робота должен происходить из "нижней-левой" вершины квадрата, из координат переданных эмулятором после старта работы (сбрасывать положение черепахи не надо).
    (b) Алгоритм движения робота: Движение по прямой, остановка, поворот налево, остановка, движение и тд.
    (c) В каждой вершине квадрата, программа должна вывести в консоль координаты (включая стартовую и конечную вершины)
    (d) После завершения движения черепахи, программа должна завершиться выйти
    (e) Допускается погрешность работы с координатами не более 4% относительно предыдущей и новой точки.
5. Запущенная программа должны обрабатывать значения одометрии, подготовленные симулятором, а не работать по времени.

Требования к решению и алгоритм проверки

1. Прислать исполняемый код для Python 2.7. Весь исполняемый код должен быть в одном файле. Название файла должно быть латинскими символами, соответствовать названию команды и через дефис номеру попытки (Пример: iskra-1.py).
2. На проверочной машине будут установлен пакет ros-melodic-desktop-full.
3. Оператор проверяет наличие кода обработки данных одометрии робота.
4. Оператор запускает roscore и turtlesim_node
5. Оператор запускает исполняемый файл python из командной строки: python filename.py.
6. Оператор проверяет движение робота в симуляторе.
7. Оператор ожидает завершения работы программы.

Решение

В нашей первой задаче, вам необходимо научиться работать с симулятором tourtlesim и управлять роботом-черепахой. Начнем с запуска программы tourtlesim.

Запустим консоль и выполним в ней команду roscore, которая запустит ядро ROS, командой roscore

Откроем новую вкладку (Crtl-T) или кликнем на иконку, и запустим сам эмулятор командой rosrun.
rosrun turtlesim turtlesim_node

Мы видим окно симулятора и черепаху, которой нам необходимо управлять. Но для того чтобы это выполнить, необходимо разобраться с некоторыми ключевыми сущностями ROS.

Одна из главных идей ROS, это идея что в рамках единой системы должны работать множество маленьких программ (нод) которые должны взаимодействовать друг с другом.

Любое взаимодействие между программами происходит при помощи специальных **Сообщений**. **Сообщение** это одна из базовых сущностей ROS, и представляет собой определенную структура данных — которую программы пересылают друг другу.

Также существует несколько моделей взаимодействия программ. Один из таких моделей взаимодействия, это механизм Топиков. Это самый простой способ коммуникации, в котором существуют два типа программ Издатель (Publisher) и Подписчик (Subscriber). Из названий можно догадаться — одна программа публикует данные, а вторая принимает эти данные.

Передвижение робота в симуляторе происходит если отправить специальное Сообщение в командный топик. Сообщение должно быть определенного типа (geometry_msgs/Twist), и содержать данные по скорости робота.

Мы можем посмотреть структуру сообщения выполнив команду show rosmsg show
geometry_msgs/Twist:
geometry_msgs/Vector3 linear
float64 x

float64 y
float64 z
geometry_msgs/Vector3 angular
float64 x
float64 y
float64 z

Мы видим две составные части сообщений: линейную (linear) и угловую (angular) скорости. Линейная скорость как следует из названия определяет скорости робота в плоскости XYZ, а угловая — скорости поворота.

Давайте это проверим — запустим робота по прямой. Для этого необходимо задать линейную скорость, по оси X.

Проще всего это сделать через консольную утилиту rostopic.

Запустим код. Для автоматической подсказки, необходимо нажимать Tab rostopic pub /turtle1/cmd_vel geometry_msgs/Twist "linear:
x: 0.1
y: 0.0
z: 0.0
angular:
x: 0.0
y: 0.0
z: 0.0r20

И мы видим — наша черепаха едет.

Опция -r20 означает что передача происходит с частотой 20 герц. Мы должны постоянно указывать скорость черепахе, иначе она остановится.

Если задать одновременно угловую скорость и линейную, черепаха поедет кругами. Для вращения черепахи нас интересует угловая скорость по оси Z rostopic pub /turtle1/cmd_vel geometry_msgs/Twist "linear:
x: 0.1
y: 0.0
z: 0.0
angular:
x: 0.0
y: 0.0
z: 0.1r20 Для решения задания, нам необходимо знать точное положение черепахи в пространстве. Симулятор дает нам эти данные (реальный робот действует аналогично), публикуя текущее положение черепахи в топик /turtle1/pose

Посмотрим, что мы можем получить из этого топика
rostopic echo /turtle1/pose
x: 4.37557840347
y: 6.92917776108
theta: -2.37115597725
linear_velocity: 0.10000000149
angular_velocity: 0.10000000149
—
x: 4.37443161011
y: 6.92806148529
theta: -2.36955595016

linear_velocity: 0.10000000149
angular_velocity: 0.10000000149

У нас есть координаты робота и его угол поворота. Стандартно в ROS размерность это единицы СИ. Расстояние в метрах, углы в радианах.

Итак, у нас есть понимание как управлять роботом и как получать данные о его положении. Это достаточно чтобы решить задачу про движение по квадрату.

Если рассмотреть алгоритм движения, то он очень простой — нам надо двигаться определенного расстояния по одной из осей, потом поворачиваться, двигаться, поворачиваться и так далее.

Более подробно о топиках и как с ними работать на python изложено в учебнике "Введение в рос"`http://docs.voltbro.ru/starting-ros/messaging/rabota-s-topic.html` или в официальной документации `http://wiki.ros.org/Topics`

Построим всю нашу программу отталкиваясь от поступающих данных о положении робота. Если мы получили данные — принимаем решение необходимо нам двигаться, поворачиваться или останавливаться, далее отправляем управляющие команды и ожидаем следующие обновление координат.

Чтобы организовать такой алгоритм, нам необходимо создать подписчика на топик ROS и функцию которая запускается, когда данные получены.

Наша программа начинается из блока импортов, в которой мы подключаем необходимые нам модули.

Мы подключаем библиотеку для работы с ROS на python rospy, математическией функции math.

Далее важный момент, мы подключили два модуля содержащие описание структуры "Сообщений". Pose сообщение для работы с данными о положении робота. Twist это сообщение для управления роботом,
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist

Пропустим пока класс Turtle и посмотрим на тело основной программы, в этой части мы подписываемся на данные о положении робота и создадим Ноду ROS, в это момент программа подключается к master процессу ROS
rospy.init_node('draw_square', log_level=rospy.DEBUG)

Выведем информационно сообщение
rospy.loginfo("Start Node")

Подпишемся на топик turtle1/pose.
rospy.Subscriber("turtle1/pose Pose, pose_callback, turtle)

Параметры функции Subscriber

1. Имя топика для подключения
2. Тип ожидаемых данных
3. Функция, которая вызывается если пришли данные
4. Глобальный обьект (не обязательно, но необходимо нам)

Суть происходящего довольно проста. Если в топике появятся новые данные, мы вызываем функцию pose_callback для обработки этих данных.

Зациклим нашу программу на постоянную работу по приему данных

rospy.spin()

Далее посмотрим на функцию pose_callback

Она принимает два параметра — переменная pose, в которой хранятся данные о текущем положении робота (данные из топика) и переменную turtle, в которой храниться глобальный обьект Turlte, который хранит состояние черепахи и управляет ей.

Функция pose_callback обновляет данные, которые хранит обьект Turtle, и после этого вызывает функцию Turtle.move(), которая занимается управлением робота.

Рассмотрим класс Turtle

Класс содержит несколько переменные

- pub Обьект rospy.Publisher который настроен для работы с топиком /turtle1/cmd_vel мы будем использовать его для передачи данных управления роботом
- last_pose с типом Pose() в которой мы храним значение координат вершины "отправной"точки.
- pose типом Pose() в котором мы храним текущее значение положения робота
- angels, в котором мы храним сколько вершин мы уже прошли.
- turtle_state Текущее состояние робота (находиться в точке старта, движется прямо, поворачивает)

Алгоритм работы робота для движения по квадрату довольно прост, это или движение прямо, пока не будет пройденно нужное расстояние, или повороты на 90 градусов. Если мы выполнили 4 поворота, значит мы пришли в стартовую точку — конец программы.

Данный алгоритм представлен в функции move

```
def move(self):
    if self.turtle_state == 'FORWARD':
        if(self.needMove()):
            self.turtleForward()
        else :
            self.turtle_state = 'TURN'
            self.last_pose = self.pose
            self.turtleStop()


    if self.turtle_state == 'TURN':
        if(self.needRotate()):
            self.turtleRotate()
        else :
            self.turtle_state = 'FORWARD'
            self.last_pose = self.pose
            rospy.loginfo("Point x:%s,y:%s", self.pose.x, self.pose.y)
            self.turtleStop()
            self.angels +=1
```

В зависимости от текущего состояния робота (движение или поворот) мы выполняем разные блоки. В первом блоке мы проверяем вызывая функцию needMove нужно ли нам продолжать движение (проверяем пройденной расстояние). Если заданное

расстояние не пройдено, то вызываем функцию turtleForward() которая публикует команды на продолжение движения.

Если расстояние пройдено, мы останавливаем робота и меняем статус на "поворот"Алгоритм блока поворота очень похож на алгортим движения. Если мы не повернулись на заданный угол (функция needRotate) то мы вращаемся (функция turtleRotate) Если мы достигли заданный угол, то меняем статус на "двиение"и двигаем счетчик пройденных вершин квадрата. Далее начинает работать алгоритм "движение"

Рассмотрим подробнее остальные функции

```
def turtleForward(self):
    pub_twist = Twist()
    pub_twist.linear.x = 0.5
    self.pub.publish(pub_twist)
```

Функция управления роботом — "движение в перед". Мы инициализируем пустой обьект типа Twist Для управления роботом. Устанавливаем линейну скорость по оси X (оси относительна робота) и публикуем эти данные вызывая метод pub.publish

```
def turtleRotate(self):
    pub_twist = Twist()
    pub_twist.angular.z = 0.5
    self.pub.publish(pub_twist)

def turtleStop(self):
    pub_twist = Twist()
    self.pub.publish(pub_twist)
```

Далее идут две похожие по сути функции управления движением робота. Первая задает угловую скорость вращения робота (для вращения робота против часовой стрелки). Вторая останавливает робота (передает пустой обькт с нулевыми скоростями).

```
def needMove(self):
    dist = math.sqrt(math.pow(self.pose.x-self.last_pose.x,2)+
        math.pow(self.pose.y-self.last_pose.y,2))
    rospy.logdebug("Dist :%s", dist)
    if dist < 3:
        return True
    else :
        return False
```

Функция проверки необходимости движения прямо. В функции вычисляет расстояние между начальной и конечной точками (квадратный корень из суммы квадратов разности двух координат). Если расстояние менее 3 метров то мы возвращаем True и наш алгоритм продолжает выполнять команды движения робота.

```
def needRotate(self):
    deg = self.pose.theta - self.last_pose.theta
    rospy.logdebug("Degrees :%s", deg)

    if (deg >= 0 and deg < math.pi/2):
        return True
    else :
        return False
```

Функция проверки необходимости продолжать поворот робота. Вы просто вычитаем углы начала поворота и текущие, и если угол более 90 градусов (pi/2 в радианах)

Последняя часть нашего алгоритма, это выход если мы проехали 4 вершины квадрата.

Проверка на кол-во вершин в функции move

```
if (turtle.angels == 4): rospy.signal_shutdown('STOP') # sys.exit()
```

Если мы проехали 4 вершины, то необходимо подать сигнал на отключение нашей ноды. Что вызовет завершение всей нашей программы.

Пример программы-решения

Ниже представлено решение на языке Python3

```
from collections import defaultdict
import sys
import random
import re
import string

def dfs(graph, root, p_inv=1, visited=None, leaves=None):
    visited = set() if visited is None else visited
    leaves = {} if leaves is None else leaves

    visited.add(root)
    children = graph[root]
    if len(children) == 0:
        leaves[root] = p_inv
    for child in children - visited:
        dfs(graph, child, p_inv * len(children), visited, leaves)

    return leaves


dataset = sys.stdin.read()
edges = [tuple(int(v) for v in edge.split()) for edge in dataset.splitlines()[1:]]

graph = defaultdict(set)
for src, dst in edges:
    graph[src].add(dst)

probs = dfs(graph, 0)
print('\n'.join('{}: {}'.format(v, '1' if probs[v] == 1 else '1/{}'.format(probs[v]))
    for v in sorted(probs.keys())))
```

```
import rospy, math

from turtlesim.msg import Pose
from geometry_msgs.msg import Twist

class Turtle(object):

    def __init__(self):
        self.pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
```

```
10          self.last_pose = Pose()
11          self.turtle_state = 'START'
12          self.pose = Pose()
13          self.angles = 0
14
15      def move(self):
16
17          if self.turtle_state == 'FORWARD':
18              if(self.needMove()):
19                  self.turtleForward()
20              else :
21                  self.turtle_state = 'TURN'
22                  self.last_pose = self.pose
23                  self.turtleStop()
24
25          if self.turtle_state == 'TURN':
26              if(self.needRotate()):
27                  self.turtleRotate()
28              else :
29                  self.turtle_state = 'FORWARD'
30                  self.last_pose = self.pose
31                  rospy.loginfo("Point x:%s,y:%s", self.pose.x, self.pose.y)
32                  self.turtleStop()
33                  self.angles +=1
34
35      def turtleForward(self):
36          pub_twist = Twist()
37          pub_twist.linear.x = 0.5
38          self.pub.publish(pub_twist)
39
40      def turtleRotate(self):
41          pub_twist = Twist()
42          pub_twist.angular.z = 0.5
43          self.pub.publish(pub_twist)
44
45      def turtleStop(self):
46          pub_twist = Twist()
47          self.pub.publish(pub_twist)
48
49
50      def needMove(self):
51
52          dist = math.sqrt(math.pow(self.pose.x-self.last_pose.x,2)+
53              math.pow(self.pose.y-self.last_pose.y,2))
54          rospy.logdebug("Dist :%s", dist)
55          if dist < 3:
56              return True
57          else :
58              return False
59
60      def needRotate(self):
61          deg = self.pose.theta - self.last_pose.theta
62          rospy.logdebug("Degrees :%s", deg)
63
64          if (deg >= 0 and deg < math.pi/2):
65              return True
66          else :
67              return False
68
69
```

```
70  def pose_callback(pose, turtle):
71
72      if turtle.turtle_state == 'START':
73          rospy.loginfo("Start x:%s,y:%s", pose.x, pose.y)
74          turtle.turtle_state = 'FORWARD'
75          turtle.last_pose = pose
76
77      turtle.pose = pose
78      turtle.move()
79
80      if (turtle.angles == 4): rospy.signal_shutdown('STOP') # sys.exit()
81
82  if __name__ == '__main__':
83      try:
84          turtle = Turtle()
85          rospy.init_node('draw_square')
86          rospy.loginfo("Start Node")
87          rospy.Subscriber("turtle1/pose", Pose, pose_callback, turtle)
88          rospy.spin()
89
90      except KeyboardInterrupt, e:
91          pass
92      print "exiting"
```

3.2. Подключение периферии через ROS

Задача 3.2.1. (30 баллов)

Задача проверяет базовые навыки способов подключения и управления периферией ROS на примере Arduino и сервопривода. От одной команды принимается только 2 решения (две попытки)!

Выполнение задания:

1. Установить ROS (ros-melodic-desktop-full) в используя пакеты на дистрибутив Ubuntu. Инструкция `http://wiki.ros.org/melodic/Installation/Ubuntu`
2. Запустить и настроить ROS, изучить базовые принципы (Книжка "Введение в ROS"`http://docs.voltbro.ru/starting-ros/`)
3. Установить и изучить документацию пакета rosserial `http://wiki.ros.org/rosserial`
4. Сгенерировать клиентскую библиотеку для ардуино (пакет rosserial_arduino)
5. Для Arduino микроконтроллера написать программу которая:
   (a) Создаст Подписчика (subscriber) на топик /servo_cmd с типом сообщения std_msgs/Int32 работающего через Serial порт Arduino
   (b) При получения сообщения, 0-180 переведет положения серво-машинки в установленный угол
   (c) Создаст Издателя (publisher) для топика /servo_cmd_echo с типом сообщения std_msgs/Int32
   (d) При получения сообщения в /servo_cmd издатель повторяет сообщение в топик servo_cmd_echo
   (e) При включении питания Arduino, серво машинка должны переклю-

чить положение в 90

6. Сигнальный провод сервопривода подключен к 9 пину Arduino

Алгоритм проверки

1. Проверочный стенд использует Raspberry PI и Arduino Mega (ATmega 2560)
2. Плата Arduino подключена к плате RaspberryPi через Serial и к персональному компьютеру через USB (Serial) для загрузки прошивки.
3. Оператор компилирует и загружает прошивку на Arduino, библиотека ros_lib установлена на компьютере. Файл с программой для Arduino должен назватся латинскими символами соответствовать названию команды и через дефис номеру попытки (Пример: iskra-1.ino).
4. Оператор подключает roscore и rosserial_python
5. Оператор из консоли отправляет сообщения в топик servo_cmd и контролирует

а) Угол поворота сервопривода

б) Публикацию ответного сообщения в топике servo_cmd_echo

Примечание

При использовании Arudino Nano может не хватать оперативной памяти (RAM). В таком случае в Aruino IDE будут появляться сообщения, типа:

Глобальные переменные используют 1837 байт (89%) динамической памяти, оставляя 211 байт для локальных переменных. Максимум: 2048 байт.

Недостаточно памяти, программа может работать нестабильно.

Можно сократить использование оперативной памяти уменьшив размер выделяемых буферов для передачи и приема сообщений. Для этого в самое начало программы следует поместить строку:
#define __AVR_ATmega168__ 1

Можно уменьшить количество занятой памяти еще сильнее, если вручную настроить количество publisher'ов и subscriber'ов, а также размеры буферов памяти, выделяемой для сообщений, например:

```
#include <ros.h>
// ...
typedef ros::NodeHandle_<ArduinoHardware, 3, 3, 100, 100> NodeHandle;
// ...
NodeHandle nh;
```

Решение

Предполагается что вся инфраструктура необходимая для решения задачи №1 уже развернута, и в этом разделе мы будем описывать только дополнительные действия необходимые для решения частной задчи. В этом задании вы должны научиться получать и публиковать данные в топики ROS через плату Arduino.

Ранее мы научились работать с топиками и написали первую программу питоне,

которая публикует и получает данные. Для работы этой программы необходим компьютер с линукс и РОС. Понятное дело, что на на Arduino у нас ничего такого нет.

Для решения таких задач, существует специальная библиотека rosserial, которая позволяет подключать разные микроконтроллеры через Serial порт и работать с микроконтроллером как с одной из нод ROS.

Сначала нам необходимо установить библиотеку rosserial компьютере.

Откроем терминал и выполним команду для установки пакета.
apt-get install ros-melodic-rosserial-arduino

Библиотека rosserial состоит из программы сервера для компьютера и клиента для микроконтроллера:

Программа-сервер на стороне компьютера решает все вопросы коммуникации с микроконтроллером и взаимодействия с ROS. Она получает данные по сериал порту и преобразует их в сущности рос.

Клиент — это библиотека которая реализует все стандартные механизмы ROS для обмена данными через. При помощи этой библиотеки мы можем использовать сообщения и топики ROS прямо на уровне программы в Arduino.

Чтобы работать с ардуино, нам необходимо установить и Arduino IDE. Скачаем и Установим IDE с официального сайда arduino.cc

Раздел Software Downloads.

Выберем дистрибутив Линукс 64

Скачаем и распакуем в папку arduino

Из консоли запустим установку командой install.sh с правами суперпользователя.

Займемся платой ардуино. Подключим управляющий контакт серво машинки к пину 9, и подадим питание +5в.

Подключим ардуино к компьютеру через USB кабель.

Проверим что IDE видит нашу плату и загрузим демо скетч.

Следующим этапом нам надо создать библиотеку для использования ее на Arduino. Сборку библиотеки необходимо производить самостоятельно, потому-то файлы попадающие в библиотеку сильно зависят от того, какие пакеты ROS установлены на вашем компьютере. "Сборщик"просматривает все пакеты, находит в них все сообщения которые могут использоваться в вашей программе и собирает эти файлы в единую библиотеку. Только в этом случае и со стороны клиента и со стороны сервера мы получим одинаковые сообщения.

Сборку библиотеки осуществляет python скрипт make_libraries.py пакета rosserial_arduino

Откроем терминал на компьютере и запустим команду сборки.
rosrun rosserial_arduino make_libraries.py

После работы скрипта, мы получим папку ros_lib в которой находиться библиотека для контроллера.

Чтобы Arduino IDE смогла работать с библиотекой, перепишем ее в директорию библиотек Arduino.
cp -r ros_lib /home/rosuser/Documents/Arduino/Library/lib/

У нас все готово, для того чтобы разобрать и запустить пример решения.

По заданию нам надо подписаться на топик servo_cmd если появилось сообщение, то выполнить перемещение сервомашинки и отправить данные обратно в другой топик.

Откроем В IDE нашу программу.

```
#include <Servo.h>
#include <ros.h>
#include <std_msgs/Int32.h>
```

Начало нашей программы как всегда с блока инклюдов.

Мы подключим библиотеку для работы с серво-машинкой.

Подключим библиотеки ros.

Подключим файл с типом сообщения Int32 Этот тип сообщения мы используем в топиках, важно различать что это Int32 объект. а не обычная переменная типа int Servo servo;

Инициализация объекта servo, который будет управлять серво-приводом

```
std_msgs::Int32 value_msg;

ros::NodeHandle_<NewHardware>  nh;
ros::Publisher pub("/servo_cmd_echo", &value_msg);
```

Далее мы инициализируем нужные нам объекты для ROS value_msg переменная типа std_msgs::Int32 для публикации в топик.
nh обьект "ноды"рос, для возможности использовать функции ROS
pub объект Паблишера для работы с топиком servo_cmd_echo.

```
void servo_cb( const std_msgs::Int32& cmd_msg)
{
  servo.write(cmd_msg.data); //set servo angle, should be from 0-180

  value_msg.data = cmd_msg.data;
  pub.publish( &value_msg );

  digitalWrite(13, HIGH-digitalRead(13));  //toggle led
}
```

Далее идет вызов функции servo_cb. Эта функция запускается, как только мы получим данные в топике servo_cmd. Параметр cmd_msg этой функции — это данные которые пришли из топика cmd_msg это не тип int а объект. Поэтому непосредственно значение находиться в переменной объекта cmd_msg.data

Как только мы получили данные, мы двигаем сервер командой servo.wrtite

```
servo.write(cmd_msg.data); //set servo angle, should be from 0-180
```

В данном примере мы не валидируем входные параметры, но для реальных программ хорошей практикой будет проверить что значения в пределе 0-180 градусов.

```
value_msg.data = cmd_msg.data;
pub.publish( &value_msg );
```

Далее мы присваиваем переменной value_msg.data значение которое мы получили и отправляем его через Издетель pub в топик servo_cmd_echo.

Функция вызова создана.

Далее мы создаем объект подписчик для топика servo_cmd и укажем созданную функцию вызова.

```
ros::Subscriber<std_msgs::Int32> sub("/servo_cmd", servo_cb);
```

Далее идет стандартная для Ардуино функция setup, в которой принято производить настройки.

```
pinMode(13, OUTPUT);
servo.attach(9);
servo.write(90);
```

Настраиваем обьект servo, указываем что серво на 9 пине и выставляем его в угол 90.

```
nh.initNode();
nh.subscribe(sub);
nh.advertise(pub);
```

Инициализируем ноду ROS. Объявляем, что необходимо использовать Издатель pub и Подписчик sub

```
void loop(){
  nh.spinOnce();
  delay(1);
}
```

Далее "циклическая"функция loop, которую Arduino выполняет постоянно.

Команда nh.spinOnce(); передает управление в библиотеку ROS, чтобы она обработала все свои задачи. В нашем примере, она проверит есть ли новое сообщение в топике, и если есть, то запустит нашу функцию servo_cb. Передавать управление в ROS необходимо как возможно чаще. Тут у нас нет никакого другого кода, поэтому сделаем задержку в 1 миллисекунду функцией delay.

Наша программа готова, загрузим ее на наш Arduino.

Далее зайдем в консоль, и запустим ноду rosserial которая подключиться к Arduino.
rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0

При запуске ноды, мы параметрами указали имя используемого serila порта и скорость подключения. Если команда запустилась без ошибок, то значит мы смогли подключиться к Arduino как к Ноде ROS. Откроем новое окно терминала и выведем список доступных топиков.
rostopic list

Мы увидем топик который мы создали на микроконтроллере, это топик servo_cmd и servo_cmd_echo.

Мы увидем топик который мы создали на микроконтроллере, это топик servo_cmd Пока топика servo_cmd_echo нет.

Запустим еще одно окно терминала и отправим команду на перемещение серво rostopic pub /servo_cmd 10

Серво-привод переместился.

Передадим для теста 90 и укажем -r10 для постоянной публикации в топик с частотой 10 герц.

Запустим вывод данных из топика servo_cmd_echo командой rostopic echo rostopic echo /servo_cmd_echo

Мы видим, что в топике есть данные которые мы публикуем.

На этом наше задание выполнено.

Пример программы-решения

```
#include <Servo.h>
#include <ros.h>
#include <std_msgs/Int32.h>

Servo servo;

std_msgs::Int32 value_msg;

ros::NodeHandle nh;
ros::Publisher pub("/servo_cmd_echo", &value_msg);

void servo_cb( const std_msgs::Int32& cmd_msg)
{
  servo.write(cmd_msg.data); //set servo angle, should be from 0-180

  value_msg.data = cmd_msg.data;
  pub.publish( &value_msg );
}

ros::Subscriber<std_msgs::Int32> sub("/servo_cmd", servo_cb);

void setup()
{
  servo.attach(9);
  servo.write(90);

  nh.initNode();
  nh.subscribe(sub);
  nh.advertise(pub);
}

void loop(){
  nh.spinOnce();
  delay(1);
}
```

3.3. Основы 3D-моделирования и цифровой электроники

Задача 3.3.1. (20 баллов)

Вам необходимо прорешать задачи курса "Цифровая электроника с Arduino", доступного по ссылке: `https://stepik.org/course/50222/syllabus`.

Система оценки

Баллы будут начислены после закрытия модуля пропорционально баллам, набранным в указанном курсе, то есть если участник команды решит 70%, будет выставлено 14 баллов.

Задача 3.3.2. (20 баллов)

Вам необходимо прорешать задачи курса "Основы инженерного 3D-моделирования", доступного по ссылке: `https://stepik.org/course/47687/syllabus`.

Система оценки

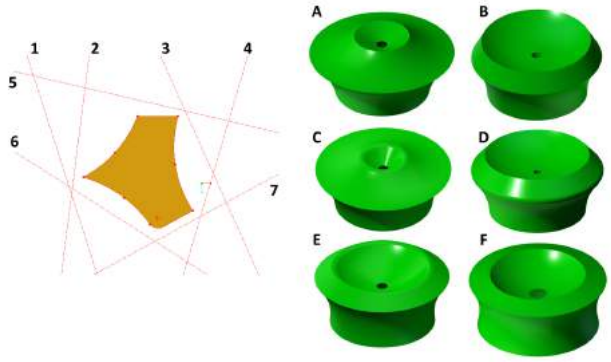
Баллы будут начислены после закрытия модуля пропорционально баллам, набранным в указанном курсе, то есть если участник команды решит 70%, будет выставлено 14 баллов.

Основы инженерного 3D-моделирования

4.1. Работа в САПР: общее понимание и базовое моделирование

Задача 4.1.1. (2 балла)

Каждое из показанных справа тел было получено из эскиза слева, вращением плоской фигуры вокруг одной из осей:



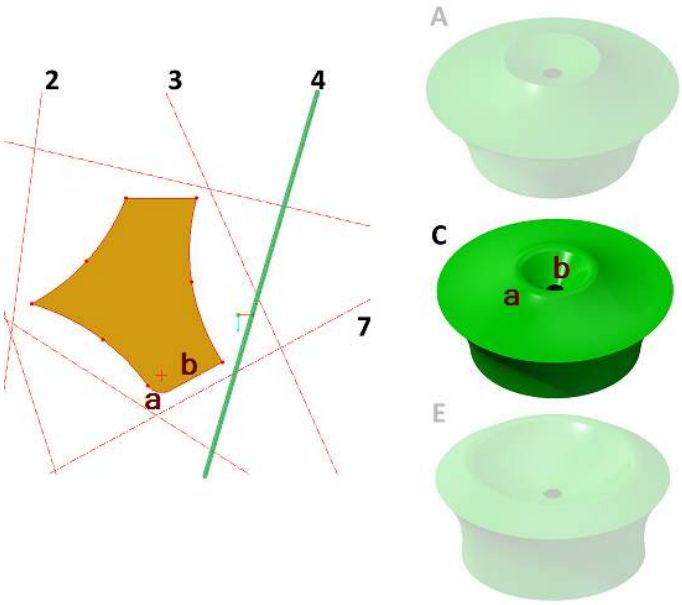
1. Тело A	а. Ось 5
2. Тело B	б. Ось 3
3. Тело C	в. Ось 6
4. Тело D	г. Ось 7
5. Тело E	д. Ось 2
6. Тело F	е. Ось 4
7. Не используется	ж. Ось 1

Решение

По эскизу определяем характерные элементы, соответствие с которыми нужно искать в телах вращения. Для каждого из показанных тел определяем ось, если надо - проверяем несколько осей, исключая неподходящие. Например, объект C имеет следующие характерные признаки:

1. сглаженный край верхнего "кратера", которому должен соответствовать скругленный угол в нижней части контура

2. короткий прямой склон "кратера", угол наклона и малый диаметр отверстия говорят о том, что "кратер" образован отрезком b, а ось должна проходить рядом с правым концом этого отрезка, под углом около 45 градусов к нему.



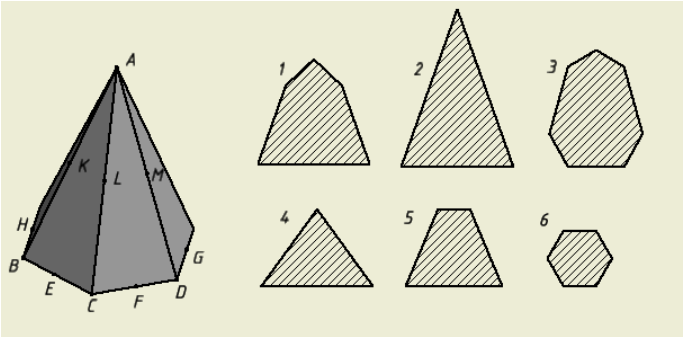
Эти признаки однозначно определяют **ось 4** как нужную нам ось. Аналогично определяются оси для каждого из остальных тел.

Ответ:

1. Тело A	б. Ось 3
2. Тело B	г. Ось 7
3. Тело C	е. Ось 4
4. Тело D	а. Ось 5
5. Тело E	д. Ось 2
6. Тело F	ж. Ось 1
7. Не используется	в. Ось 6

Задача 4.1.2. Геометрия сечений (2 балла)

Имеется пирамида, некоторые вершины и середины ребер которой обозначены буквами. Построено несколько плоскостей сечения, каждая из которых проходит, по крайней мере, через 3 точки и обозначается по этим точкам (например, "плоскость BLD"). Эти плоскости создали следующие сечения. Для каждого сечения укажите соответствующую секущую плоскость (еще 3 сечения этой же пирамиды Вы увидите на следующем шаге):



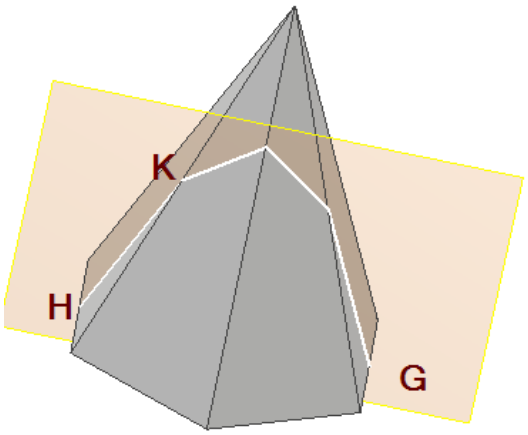
1. Плоскость FGK
2. Плоскость GHK
3. Плоскость ABD

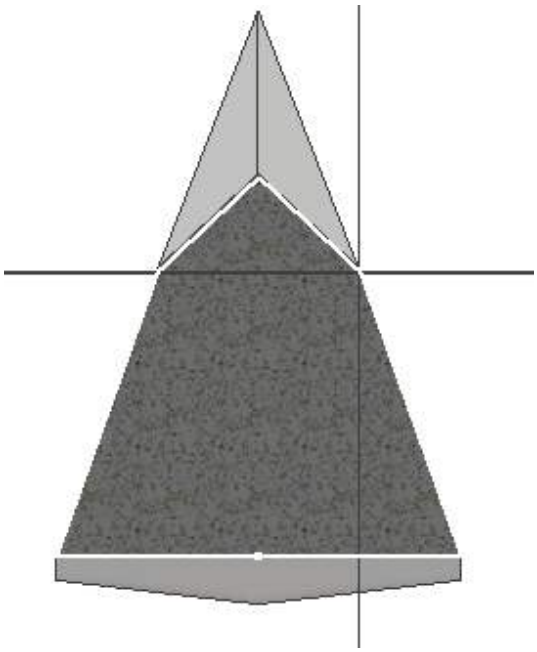
а. Сечение 1
б. Сечение 3
в. Сечение 2

Решение

Обратите внимание, что каждое из сечений данного тела может быть получено при разрезании тела по нескольким разным плоскостям, а сами плоскости могут быть обозначены разными комбинациями букв. Поэтому начинаем не с рассматривания сечений, а с построения плоскостей, предлагаемых как варианты ответов.

Например, первая из предложенных плоскостей, GHK, пройдет, как показано на рисунке. Сразу становится очевидно, что сечением в этом случае становится фигура 1:





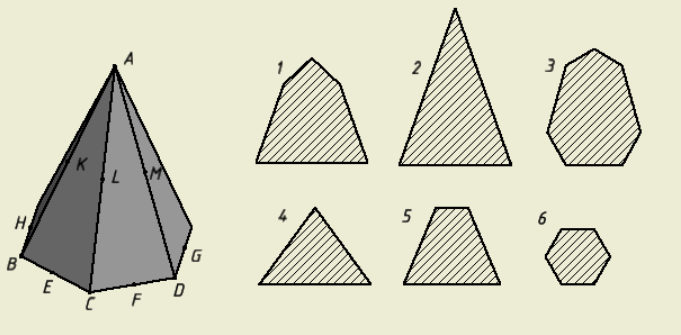
Ответ:

1. Плоскость FGK
2. Плоскость GHK
3. Плоскость ABD

б. Сечение 3
а. Сечение 1
в. Сечение 2

Задача 4.1.3. Геометрия сечений (1 балл)

Продолжаем предыдущую задачу. Сопоставьте еще 3 плоскости и 3 сечения пирамиды:



1. Плоскость EGM
2. Плоскость BDL
3. Плоскость KLM

а. Сечение 6
б. Сечение 4
в. Сечение 5

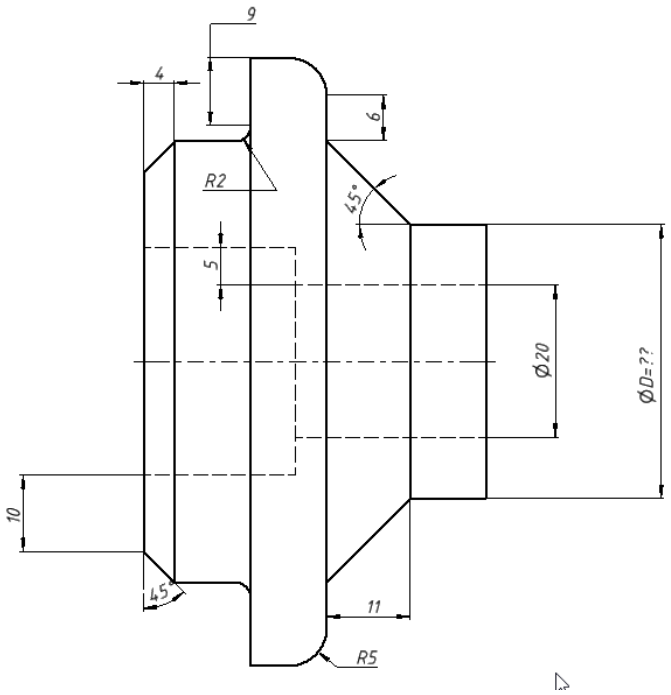
Ответ:

1. Плоскость EGM
2. Плоскость BDL
3. Плоскость KLM

в. Сечение 5
б. Сечение 4
а. Сечение 6

Задача 4.1.4. (1 балл)

По размерам, имеющимся на чертеже (все размеры даны в мм), определите диаметр d:

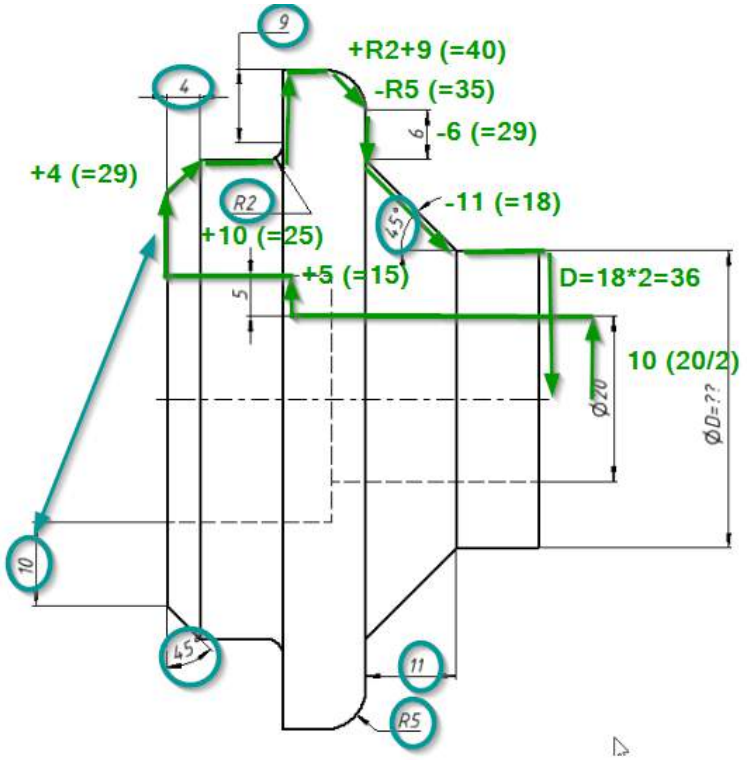


Вычислите d, введите только число (целое).

Решение

В задачах этого типа вас посылают в путешествие вокруг детали, начиная с какого-то известного размера, карабкаясь по цепочке размеров, добавляя или вычитая очередной размер. Препятствия, которые вам могут встретиться в пути:

- Размер поставлен не с той стороны, где вы его ожидаете увидеть,
- Вы "залезаете"или "съезжаете"по склону с наклоном 45 или 30 градусов, и вместо высоты склона, вам дана его ширина. Вычислять хитрые синусы/косинусы не понадобится, но кое-что про свойства треугольников надо знать,
- Вместо вертикальной стенки, вам могут встретиться скругления с заданным радиусом,
- Вы можете забыть в нужный момент перейти от радиуса к диаметру или обратно.

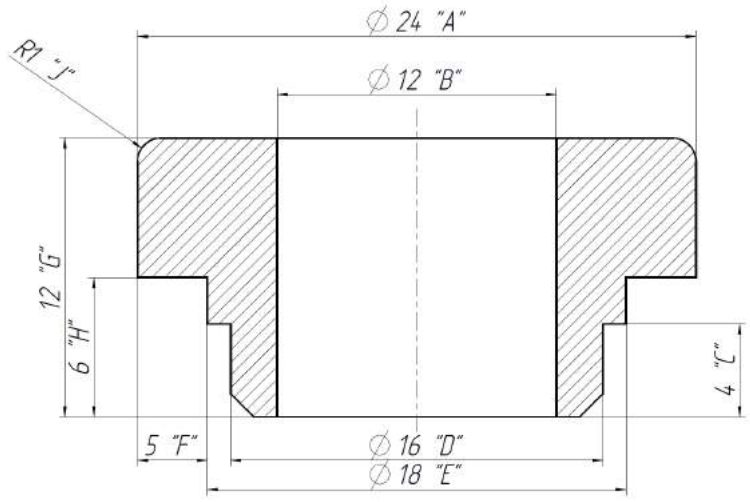


Как видно из рисунка, в данной задаче ответом будет число 36.

Ответ: 36.

Задача 4.1.5. (1 балл)

На этом чертеже неправильно указан один из размеров:



Найдите неверный размер и введите для него правильное значение, указав букву размера и верное значение в формате «буква размера>=<размер>"(например A=24). Все буквы прописные.

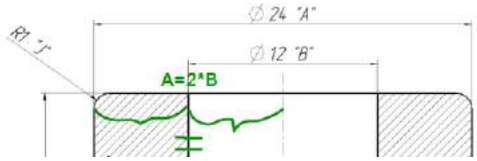
Решение

Назовем избыточным размер, который можно вычислить из других размеров, имеющихся на чертеже. Если такая цепочка размеров присутствует, то избыточным можно считать, на выбор, любой из входящих в нее размеров. При этом совсем необязательно, чтобы избыточный размер оказался неправильным. На самом деле, при генерации чертежей в САПР по 3D-модели наставить лишних размеров очень легко и просто, но нужно специально постараться (вручную изменяя размеры), чтобы один из этих размеров стал неправильным.

В данном примере есть только одна избыточная цепочка - это размеры A, E и F. Эти размеры должны удовлетворять соотношению $2 \cdot F = (A - E)$, однако на чертеже это соотношение не выполняется: $2 \cdot F = 10$, а $A - E = 24 - 18 = 6$. Значит, действительно, один из размеров в цепочке задан неверно. Очень хочется сразу решить, что неправильным (и лишним) является размер F и его настоящее значение должно быть $F = (A - E)/2 = 3$ (мм).

Однако это пока только гипотеза, с тем же успехом ошибка может быть в размере A или в размере E. Попробовать выяснить, который из этих трех размеров ошибочен можно, только сопоставляя их с другими размерами чертежа. Поэтому проверим каждый из размеров A, E, F, учитывая, что по условию задачи неправильный размер есть только один:

- Предположим, что ошибочен размер E. В этом случае его значение должно быть: $E = A - 2 \cdot F = 14$ мм, что не согласуется с чертежом: размер $D = 16$ должен быть меньше E.
- Предположим, что ошибочен размер A. Но на чертеже мы видим, что с показанными на чертеже значениями $A = 2 \cdot B$, и это подтверждается визуально равенством отмеченных отрезков. Если принять что $A = E + 2 \cdot F = 28$, то пропорции чертежа существенно изменились бы.



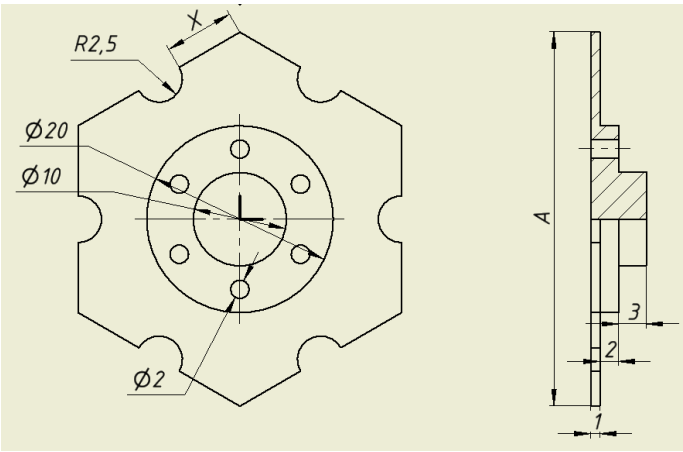
Таким образом, "неправильным"размером действительно является F и ответ, ожидаемый Stepik'ом: $F = 3$.

Ответ: F=3.

4.2. Базовое моделирование

Задача 4.2.1. Пластина (4 балла)

На чертеже изображена некоторая пластина (все размеры проставлены в мм). Определите размер, помеченный на чертеже символом X, при условии, что значение параметра A=40 мм.



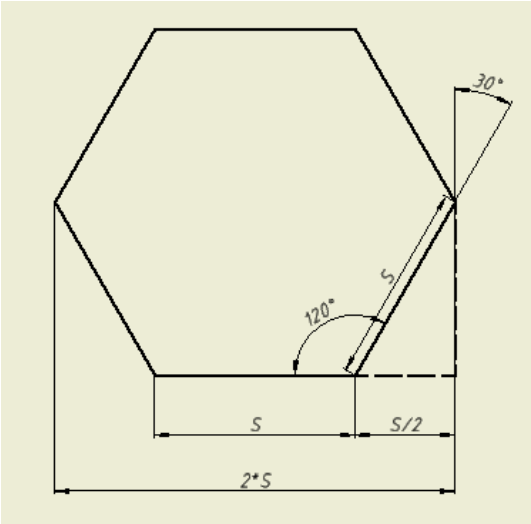
Определите размер X в мм (только число, точный ответ).

Примечание: здесь возможно как довольно простое аналитическое решение, так и решение построением в САПР. Учитывая, что в следующем же шаге та же самая деталь все равно строится полностью (с нахождением объема), быстрее просто начать строить ее и, по ходу, получить требуемый размер. Тем не менее, давайте начнем с аналитического решения.

Решение

Аналитическое

Как показано на рисунке ниже, правильный 6-угольник имеет интересное свойство: расстояние между противолежащими вершинами равно удвоенной длине стороны (т.к. $sin(30°) = 0.5$).

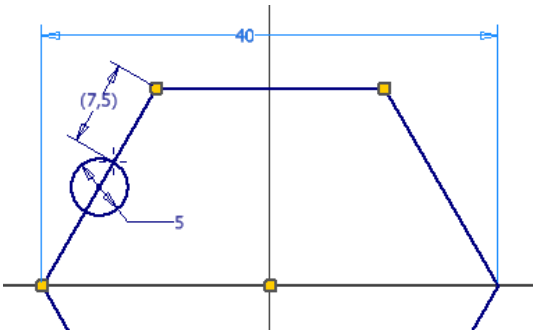


Следовательно, в нашей задаче:

$$X = (A/2 - 2 \cdot R)/2 = (20 - 5)/2 = 7.5 \text{ мм}$$

Построением

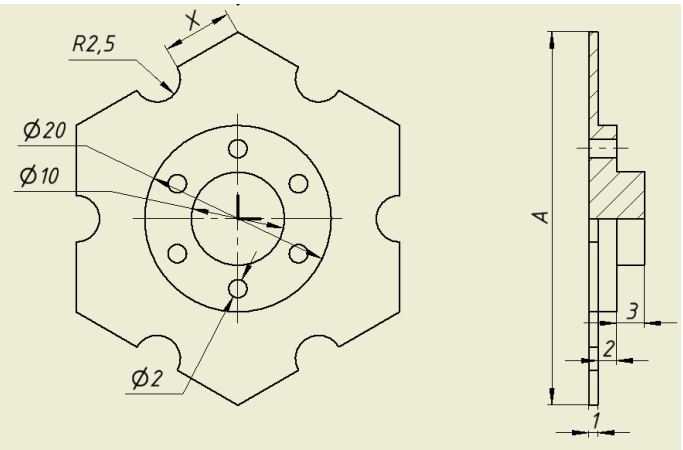
Тривиально: просто строим в эскизе в САПР 6-угольник заданного размера, строим окружность от средней точки ребра, ставим справочный размер. Получается, разумеется, те же 7.5 мм.



Ответ: 7.5

Задача 4.2.2. Пластина-2 (4 балла)

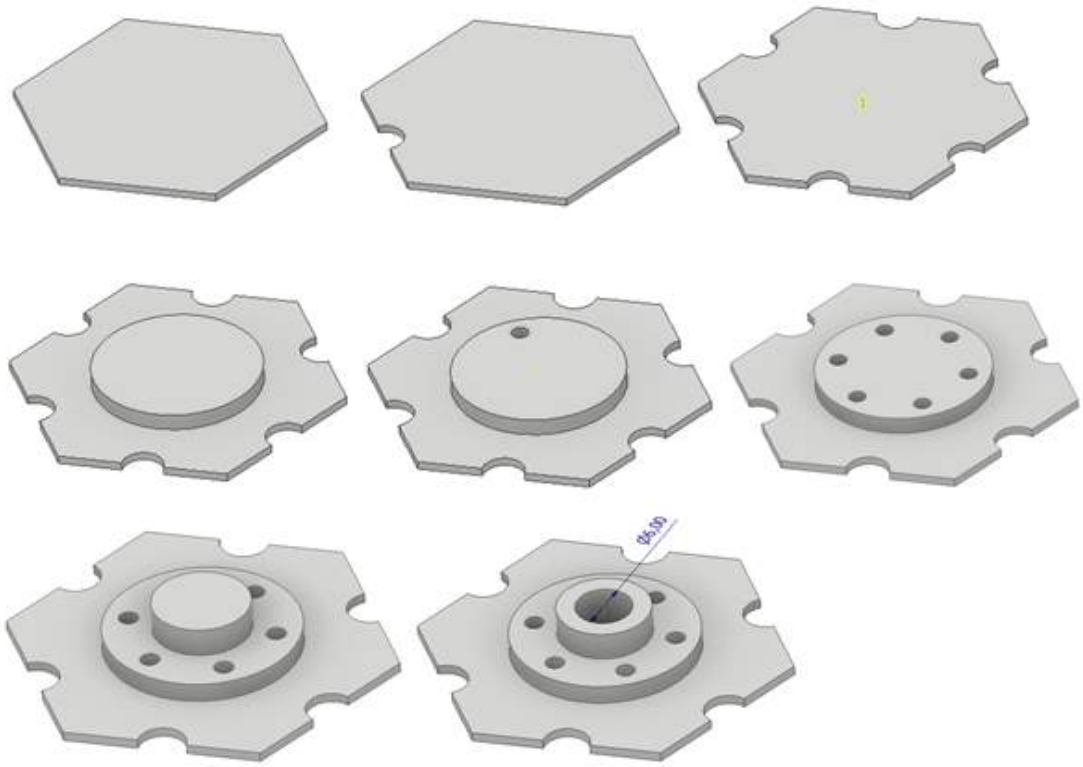
Постройте 3D-модель той же самой пластины (если Вы еще этого не сделали). Дополните модель детали недостающими элементами, чтобы пластину можно было надеть на круглый вал радиусом 3 мм.



Определите массу получившейся детали в граммах (с точностью не ниже 0.1 г) при условии, что она выполнена из высокопрочной низколегированной стали с плотностью 7.85 г/см^3.

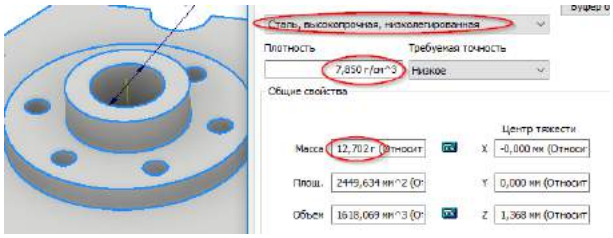
Решение

Просто строим модель по чертежу. Можно сильно упростить себе жизнь, если не пытаться делать сложные эскизы, а строить элементы модели последовательно, вот так:



Здесь применяются только выдавливания и круговые массивы. Очень важно полностью определять каждый эскиз, задавая размеры и привязки для всех его элементов. Обратите внимание на скрытую в задании "засаду": указан *радиус* отверстия, в то время как нам нужен его *диаметр*! Всегда внимательно читайте задание!

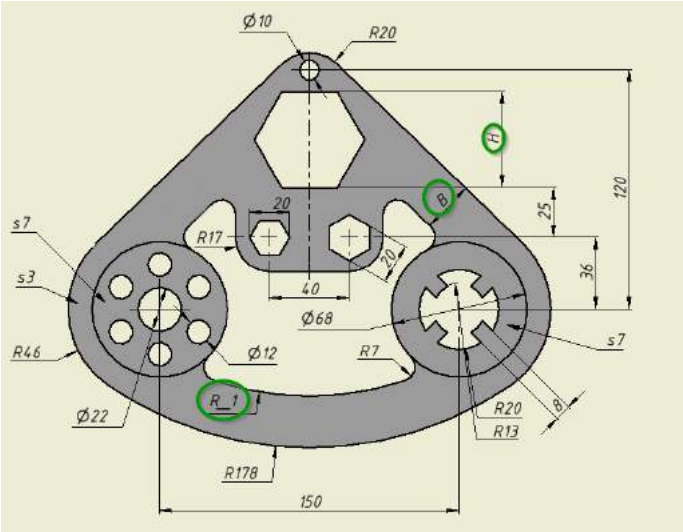
Когда модель готова, средствами вашего САПРа получаем ее массу, для заданного материала. Например, в Autodesk Inventor это будет выглядеть так:



Ответ: 12.7 ± 0.1

Задача 4.2.3. Ключ (6 баллов)

Смоделируйте ключ по приведенному чертежу. Значение параметра H взять равным 48 (мм), параметра R_1 — равным 152 (мм), значение параметра B определяется равенством B=178 – R_1 (мм). Радиусы всех сопряжений по углам внутреннего выреза одинаковы (R7).

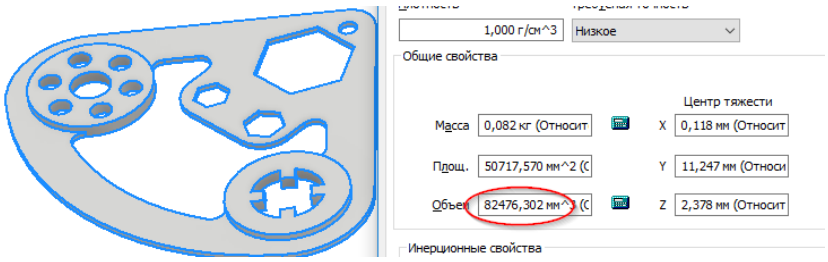


Определите объем детали в мм^3 (число, с точностью до целых).

Пояснения к ответу

Деталь почти плоская, всего лишь с двумя элементами разной толщины. Поскольку вид сбоку не показан, может быть не сразу понятно, откуда брать информацию о толщинах. Она задана выносками "s3"и "s7 т.е. основная пластина детали имеет толщину 3 мм, а два круглых утолщения - 7 мм.

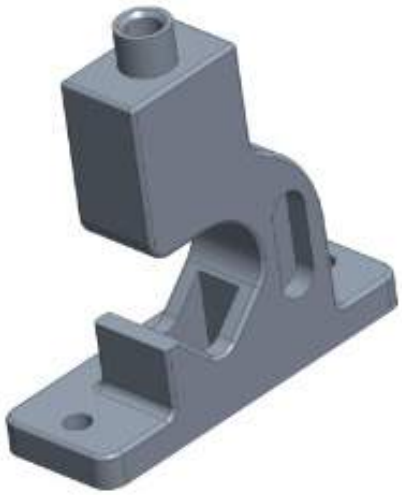
Основная сложность при моделировании этого ключа - правильно построить первый эскиз, применив все показанные на чертеже размеры и догадавшись обо всех использованных привязках и симметриях. Критерий правильности построения - полная определенность эскиза. Например, в Autodesk Inventor с обычной цветовой схемой, все линии эскиз должны стать темно-синими. По готовности модели, получаем объем. Например, в Autodesk Inventor:



Ответ: 82476 ± 20

Задача 4.2.4. Корпус тисков (6 баллов)

Скачайте чертеж корпуса тисков (`https://drive.google.com/file/d/1krX248Yo9j-rr7Zo1zUXn9mG79l2Mcp1/view`) и точно смоделируйте деталь. (На чертеже не указаны размеры: радиус паза = 9 мм, сторона фаски на верхнем отверстии = 1 мм).



Найдите объем детали, в мм^3 (вводить только число, округляя до целого, погрешность не более 5 мм^3)

Пояснения к ответу

Как и в предыдущей задаче, здесь нужно точно смоделировать деталь по чертежу и получить ее объем, однако эта деталь имеет гораздо более сложную форму.

Приводить здесь ход построения большого смысла не имеет, т.к. на следующий год в задании будут фигурировать совсем другие детали. Для тренировки, моделируйте любые детали похожей сложности из учебника по черчению, либо поищите подобные чертежи в интернете. *Следите, чтобы каждый из эскизов был полностью определен!*

Ответ: 391724 ± 20

4.3. Работа в САПР: Сборки

Задача 4.3.1. (2 балла)

Соберите головоломку, используя приложенные файлы (в формате STEP):

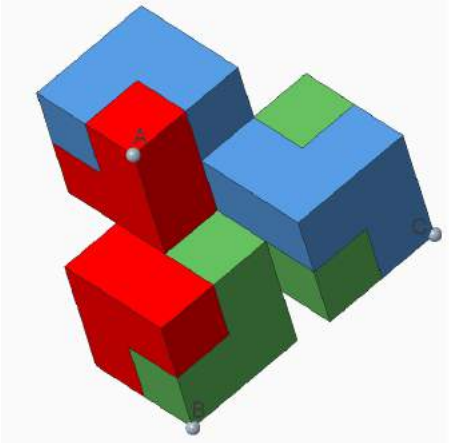
Деталь A

`https://drive.google.com/file/d/1JcXJwKiEi42S5JiICzoJdA31vLrjrEB-/view?usp=sharing`

Деталь B

`https://drive.google.com/file/d/1jTdQmNkdBCKJRBW_yd5K-0kuQvOURAFe/view?usp=sharing` Деталь C

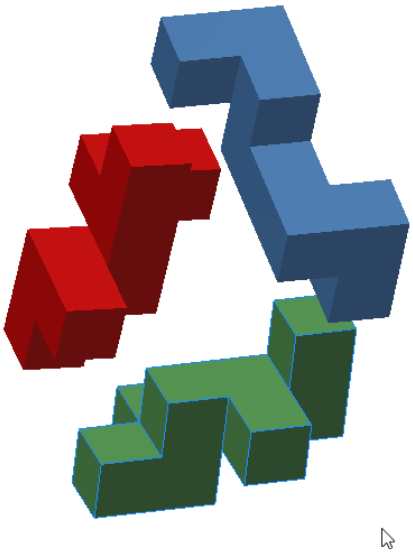
`https://drive.google.com/file/d/1prU86HaGt6WiuD1ZvGuyYfNLL1i98_tx/view?usp=sharing`



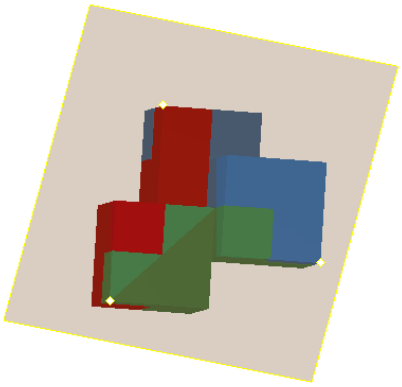
Какова площадь (в мм^2) треугольника, построенного по точкам ABC, указанным на рисунке?

Решение

Создаем сборочную модель, загружаем в нее 3 детали в формате STEP, скачанные по ссылкам. Крутим каждую деталь (в Autodesk Inventor - команда "свободный поворот"), пока она не займет положение, узнаваемое по верхнему рисунку.

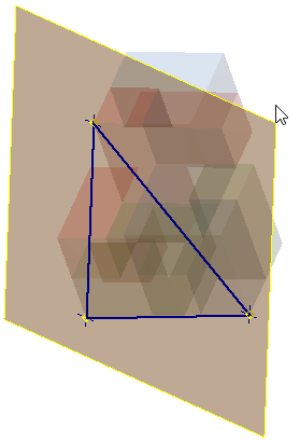


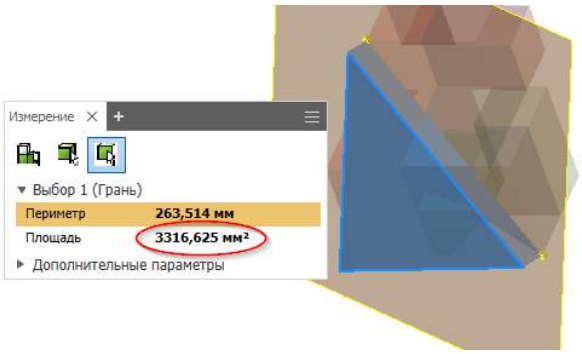
Соединяем детали сборочными зависимостями по граням, ребрам или точкам. Обычно требуется 3 зависимости для соединения каждых двух деталей.



Когда головоломка собрана, строим плоскость по 3-м точкам, показанным в задании. Учитывая, что работа идет в сборке, дальнейшее будет различаться в разных САПР.

Так, в Autodesk Inventor придется создать новую деталь в контексте сборки. После этого В плоскости строим эскиз. а в эскизе - треугольник, построенный по проекциям этих 3х точек. Слегка выдавливаем контур, чтобы измерить его площадь.

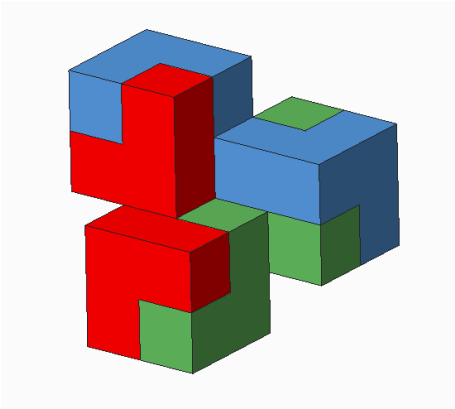




Ответ: 3316.625 ± 1

Задача 4.3.2. (2 балла)

В головоломке, собранной в предыдущей задаче, какая площадь поверхности фигуры C (красная деталь), соприкасается с другими деталями?



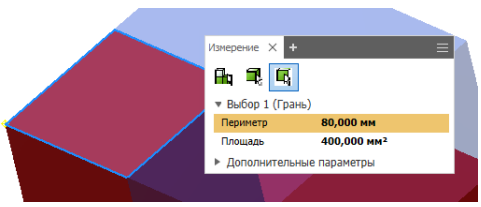
Введите площадь, в мм^2 (только число).

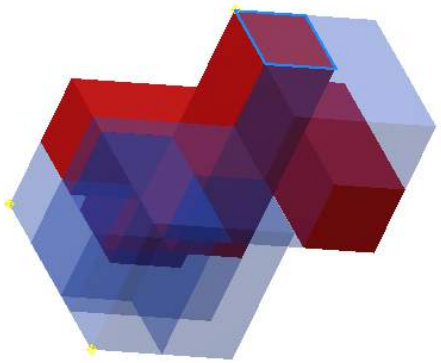
Решение

Для начала, посмотрим площадь единичного квадрата на торце элемента головоломки. Как видно, она составляет 400 мм^2.

Для облегчения подсчетов, делаем остальные 2 детали полупрозрачными. Считаем грани.

Получается 12 единичных квадратов, или 4800 мм^2.





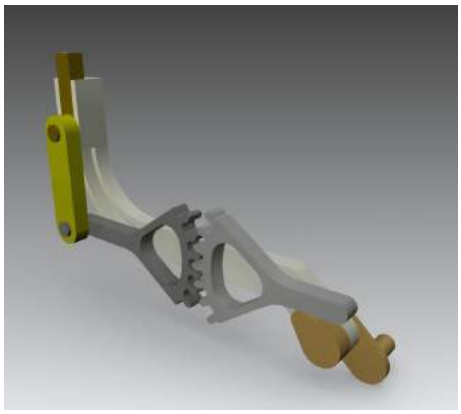
Ответ: 4800.

Задача 4.3.3. Кулачковый механизм (5 баллов)

Скачайте архив с деталями с формате STEP.

(https://drive.google.com/file/d/1NaBgymX6Suge4ll5Ia8cB15TWLGF4S6A/view?usp=sharing)

Соберите из этих деталей кулачковый механизм, как показано на рисунке:



Введите амплитуду движения поршня при вращении рукоятки (т.е. расстояние между крайними положениями поршня), в мм, только число, с точностью не хуже 0.1.

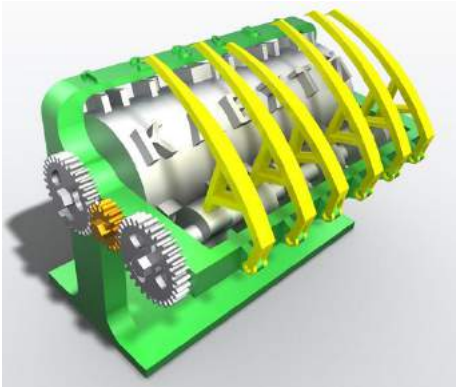
Пояснения к ответу

В задачах этого типа вам предлагается собрать механизм и вычислить диапазон движений, либо найти секретный код, определяемый по движению механизма. Требуется умение применять в сборках динамические зависимости - зубчатые передачи, кулачковые механизмы. Возможны задачи, в которых нужно изменить или добавить детали, чтобы механизм работал. В этом случае спрашивается какой-либо ключевой параметр измененной/добавленной детали. Полный процесс сборки кулачкового механизма (Вариант А) в САПР Autodesk Inventor показан в видеоуроке по адресу: https://bit.ly/2Jih7gs.

Ответ: 10.14 ± 0.1

Задача 4.3.4. Кодовая машина (8 баллов)

Скачайте комплект деталей (https://drive.google.com/file/d/162uWxhMfkUMoLGutxY6GeZiJncI3fekF/view?usp=sharing) и соберите из них показанное ниже устройство. Обратите внимание, что на зубчатых колёсах есть метки для правильной ориентации кулачкового вала относительно барабана с буквами. Если всё собрано правильно, то при вращении барабана рычаги опускаются напротив букв, составляющих слово, которое нужно ввести в ответ задания. Первый кулачок определяет первую букву слова, второй - вторую букву и так далее, всего шесть букв в слове.



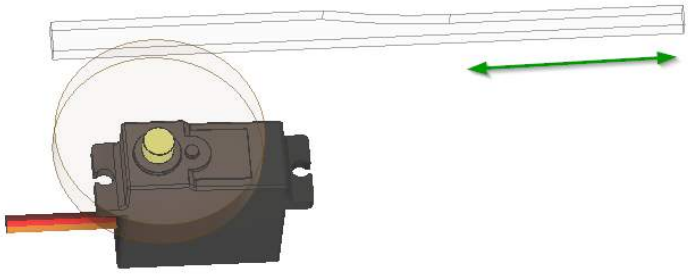
Введите кодовое слово. Ответ вписываем заглавными русскими буквами. Вместо пробела на валу используется символ "_ если в коде попался этот символ, так и вводим "_ ".

Ответ: К_ЦЕЛИ.

4.4. Механика

Задача 4.4.1. (5 баллов)

Вам надо спроектировать узел линейного перемещения для автомата, который раскладывает грузы по 8 ячейкам. Ячейки выстроены в одну линию, расстояние между соседними ячейками 40 мм. В качестве привода применен сервомотор с шестерней и зубчатой рейкой, как показано на рисунке:



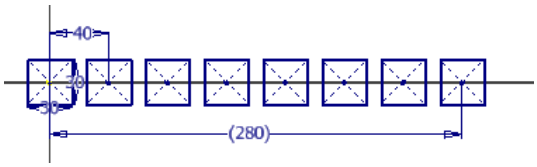
Сервопривод имеет вращательный момент (stall torque) 0.2 Н · м и диапазон поворота вала 270 градусов. Какое максимальное усилие можно получить на зубчатой рейке, при условии, что узел линейного перемещения должен обеспечивать позиционирование схвата над центром любой из ячеек?

Введите максимальное усилие, в Ньютонах (вводить только число), с точностью не ниже 0.1 Н.

Решение

Чем больше диаметр зубчатого колеса, тем больше ход зубчатой рейки, но меньше усилие. Нам нужно максимальное усилие, а значит, минимальный диаметр зубчатого колеса, при котором ход рейки оказывается достаточным для перемещения от

центра первой до центра последней ячейки. Для N ячеек число "перегонов"между ячейками составит $N-1$, а необходимый ход рейки $S = 7 \cdot 40 = 280$ мм:



Такой ход рейки должен составлять 3/4 окружности, поскольку угол поворота вала серво ограничен 270°. Следовательно, радиус зубчатого колеса будет равен:

$$R = \frac{4}{3} \cdot S/(2\pi) = \frac{2S}{3\pi} \approx 59.42 \text{ мм} \approx 0.0594 \text{ м}$$

Вращательный момент будет составлять:

$$T = \frac{Ts}{R} = \frac{0.2}{0.0594} = 3.37 \text{ Н}$$

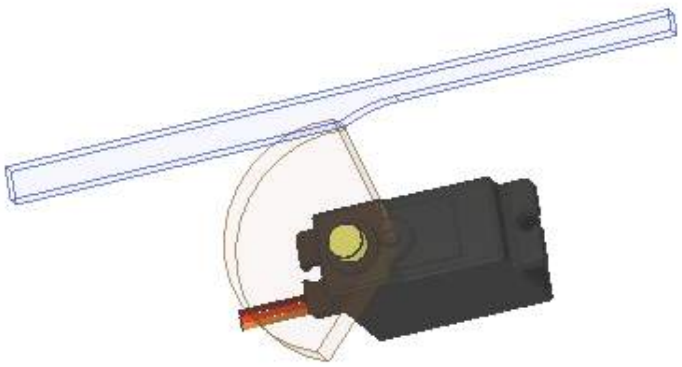
Обратите внимание, что в реальности зубчатое колесо не может иметь произвольный диаметр. При конструировании задается модуль зуба M, а диаметр делительной окружности зубчатого колеса получается умножением модуля зуба на число зубьев. Так, например, если было решено использовать зубчатую передачу с $M = 1.0$ мм, допустимые зубчатые колеса будут иметь целочисленный диаметр (а радиусы с шагом 0.5) и нам придется округлить результат до 60 мм.

В этом случае $T = 3.33$ Н. И тот и другой ответ находится в диапазоне допустимых погрешностей для данной задачи в Stepik.

Ответ: 3.33 ± 0.1

Задача 4.4.2. (4 балла)

Продолжаем конструировать тот же узел линейного перемещения. После покупки сервопривода неожиданно оказалось, что купленная модель обеспечивает поворот вала только на 120 градусов. Поэтому, и для сокращения размеров, было решено вместо полного зубчатого колеса использовать зубчатый сектор:



При использовании модуля зуба 1.5 мм, сколько зубьев будет иметь этот зубчатый сектор?

Ответ: 60 ± 1

Задача 4.4.3. (5 баллов)

Ось X 3D-принтера приводится в действие шаговым двигателем NEMA17 с вот такими параметрами:

Крутящий Момент: 4000g.cm	Угол Шага (градусы): 1.8 degree
Артикул: 17HS4401	Ток / Фаза: 1.7A
Фаза: 2	Тип: Гибридные

На валу двигателя смонтирован шкив с 20 зубьями под зубчатый ремень типа GT2 (с шагом зуба 2 мм), перемещающий каретку. В драйвере мотора используется микрошаг 1/16.

Сколько шагов мотора необходимо выполнить, чтобы переместить каретку на A мм?

Решение

В данной задаче не требуется работать с силами и моментами вращения, но нужны базовые знания о принципах работы шагового двигателя и ременной передачи.

Из таблички с данными о моторе нам понадобится только одна величина: угол шага 1.8°. Это означает, что 1 оборот вал мотора совершит за $360/1.8 = 200$ шагов, или, наоборот, за 1 шаг вал поворачивается на 1/200 оборота. Кроме того, драйвер шагового двигателя умеет работать с микрошагами, искусственными промежуточными положениями вала внутри одного шага. Например, микрошаг 1/16 означает, что для поворота вала двигателя на один шаг нужно подать 16 управляющих импульсов.

Нам не потребуется диаметр шкива и число π, чтобы вычислять длину окружности. Достаточно знать, что шкив рассчитан на ремень с шагом зуба 2 мм и имеет 20 зубцов. Диаметр шкива подобран так, чтобы именно столько зубцов укладывалось по окружности. Это означает, что за один оборот шкива ремень продвинется на $S = 20 \cdot 2 = 40$ мм.

Таким образом, чтобы переместить каретку на X мм, на драйвер мотора необходимо подать N шаговых импульсов:

$$N = \frac{X}{40} \cdot 200 \cdot 16 = 80 \cdot X \text{ (микрошагов)}$$

В задаче в Stepik вместо переменной X каждый раз появляется случайное число, и вам нужно ввести числовой ответ.

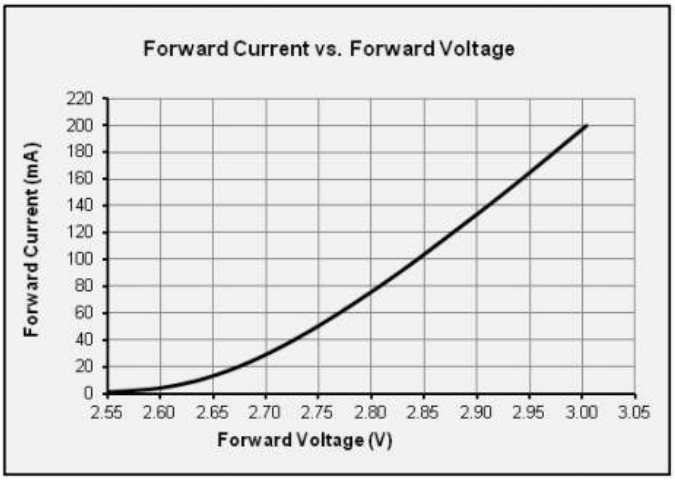
Ответ: $80 \cdot a$

Цифровая электроника с Arduino

5.1. Основы электроники

Задача 5.1.1. (3 балла)

В качестве фар машинки-робота использован осветительный светодиод LM561C, запитываемый от 2-х баночного LiPo аккумулятора (7.4 В). Последовательно со светодиодом, в цепь также включен ограничительный резистор. Используя приведенную ниже вольт-амперную характеристику светодиода, выберите номинал резистора так, чтобы ток через светодиод был по возможности ближе к 100 мА, но не превышал этого значения.



а. Выберите номинал ограничительного резистора из стандартного ряда номиналов:

1. 33 Ом
2. 39 Ом
3. 47 Ом
4. 56 Ом
5. 68 Ом
6. 82 Ом
7. 100 Ом
8. 120 Ом

б. Максимальная рассеиваемая мощность этого резистора не должна быть менее чем

1. 0.125 Вт
2. 0.25 Вт
3. 0.5 Вт
4. 0.75 Вт
5. 1 Вт
6. 2 Вт

Решение

В этой задаче вам надо понимать закон Ома и знать, что такое вольт-амперная характеристика.

В отличие от, например, резисторов, светодиоды являются нелинейными электронными компонентами. Ток через светодиод почти не протекает при малых значениях напряжения, находится в рабочем диапазоне в очень узком диапазоне напряжений, а при дальнейшем увеличении напряжения резко нарастает, до перегрева и выхода светодиода из строя. График зависимости тока от напряжения называется вольт-амперной характеристикой устройства.

Как видно из графика, требуемый ток в 100 мА получается при напряжении на диоде 2.85 В. Однако же на входе у нас 7.4 В, поэтому из них $U_R = 7.4 - 2.85 = 4.55$ В должны падать на ограничительном резисторе. Поскольку резистор и светодиод соединены последовательно, ток через резистор составляет те же 100 мА. По закону Ома:

$$R = U_R / I = 4.55 \text{ В} / 0.1 \text{ А} = 45.5 \text{ Ом}$$

Однако резисторы изготавливаются определенных номиналов, и Stepik предлагает выбрать один из них (33, 39, 47, 56, 68 Ом). Выбираем резистор с ближайшим большим значением - 47 Ом.

Какова же должна быть максимальная рассеиваемая мощность этого резистора? Мощность вычисляется по формуле:

$$P = U \cdot I = 4.55 \text{ В} \cdot 0.1 \text{ А} \approx 0.45 \text{ Вт}$$

Подбираем ближайшее большее значение (0.125, 0.25, 0.5, 0.75, 1, 2 Вт): 0.5 Вт.

Ответ: а - 3, б - 3.

5.2. Программируем Arduino: простые схемы и задачи

Задача 5.2.1. Простой blink (2 балла)

В этом шаге вам нужно будет изменить и проверить работу простейшего примера blink из стандартного набора примеров Arduino IDE, для этого следует изменить номер пина и интервал мигания, как указано в шаблоне!

Такие задачи стали возможны благодаря мини-симулятору Arduino, который имитирует действие некоторых функций библиотеки Arduino, изменяя состояние хранимых в памяти "пинов"и сообщая о каждом изменении платформе Stepik, чтобы

проверить правильность вашего решения. Вы пишете код КАК БЫ для Ардуино, на самом деле он выполняется в Степике, взаимодействуя с платформой через стандартный ввод-вывод, как и любая другая задача на программирования.

Вам НЕ НАДО трогать что-нибудь в шаблоне, кроме функций setup() и loop(). Разумеется, для решения более сложных задач могут понадобиться дополнительные функции или переменные, которые вы можете определить там же. Можно использовать большинство стандартных библиотек C++, но никакие библиотеки, специфические для Arduino, применить не получится. Мини-симулятор настраивается под задачу, и в нем, как правило, не будут работать никакие функции, не относящиеся конкретно к данной задачи.

Функции библиотеки Arduino, которые можно использовать в этой задаче:

void digitalWrite(int pin, int state): изменяет хранимое в памяти состояние пинов, и сообщает в Stepik о каждом изменении, вместе с отметкой времени. Если был задан неверный номер пина, либо этот пин не был переведен в режим OUTPUT, либо состояние пина не изменилось, то ничего не происходит.

void pinMode(int pin, int mode): устанавливает режим работы пина: INPUT, OUTPUT или INPUT_PULLUP.

void delay(int n): "Задержка"на n миллисекунд, а на самом деле - просто увеличение внутреннего счетчика времени. Мини-симулятор следит за временем, хотя, в отличие от реального микроконтроллера, счетчик времени увеличивается только и исключительно вызовами функции delay(). Любые операции, между которыми нет delay(), считаются выполняющимися мгновенно.

Если вы хотите испытать свой код на реальной схеме с Ардуино, просто перенесите в среду Ардуино написанный вами код и он, МОЖЕТ БЫТЬ, даже будет работать. Разумеется, вам сначала придется собрать соответствующую описанию схему.

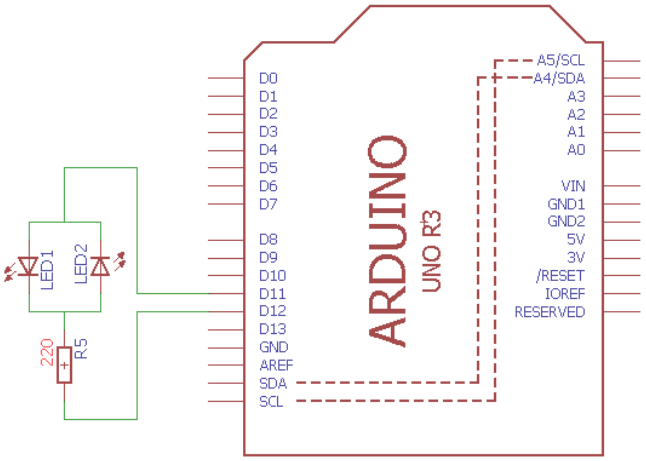
Пример программы-решения

Ниже представлено решение на языке C++

```
int LED_PIN;   // каким пином мигаем
int ON_PERIOD; // время свечения, мс
int OFF_PERIOD; // интервал между миганиями, мс
void setup()
{
    pinMode(LED_PIN,OUTPUT);
}
void loop()
{
    digitalWrite(LED_PIN, HIGH);
    delay(ON_PERIOD);
    digitalWrite(LED_PIN, LOW);
    delay(OFF_PERIOD);
}
```

Задача 5.2.2. Полицейский маячок (BLINK со встречным подключением светодиодов) (3 балла)

Два светодиода, красный и зеленый, подключены к Ардуино, как показано на схеме:



Напишите программу, которая попеременно мигает красным и зеленым светодиодами, без "темных"интервалов. Каждый светодиод включен LED_PERIOD миллисекунд (интервал меняется от теста к тесту). Какой из светодиодов зажигается первым, роли не играет. Имеются функции pinMode(), digitalWrite(), delay().

Решение

Это еще одна супер-легкая задача, рассчитанная на то, чтобы дать вам разобраться с написанием кода для мини-симулятора. Раз диоды подключены "встречно программа должна попеременно подавать LOW на D11 и HIGH на D12 (включен LED2) или HIGH на D11 и LOW на D12 (включен LED2). Для мини-симулятора "одновременными"считаются действия, выполняемые без вызова delay() между ними.

Прежде, чем писать собственно код для переключения, не забываем инициализировать соответствующие пины как выходные. При этом обязательно использовать именованные константы, приведенные в шаблоне кода, а не числа:

```
void setup()
{
    pinMode(LED_PIN1, OUTPUT);
        pinMode(LED_PIN2, OUTPUT);
}
```

Вот самое простое (и тупое) решение для поочередного мигания (и опять-таки, обязательно используем параметр LED_PERIOD, приведенный в шаблоне кода. Он будет принимать разные значения для разных тестов, и иначе программа правильный ответ не выдаст):

```
void loop()
{
    digitalWrite(LED_PIN1, HIGH);
    digitalWrite(LED_PIN2, LOW);
    delay(LED_PERIOD);
    digitalWrite(LED_PIN1, LOW;
    digitalWrite(LED_PIN2, HIGH));
    delay(LED_PERIOD);
}
```

А вот чуть более интересное решение с использованием переменной:

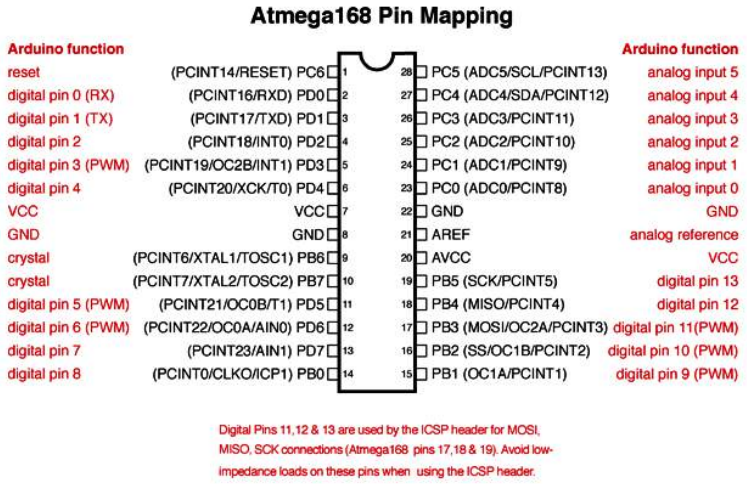
```
bool b = true;

void loop()
{
    digitalWrite(LED_PIN1, b);
    b = !b;
    digitalWrite(LED_PIN2, b);
    delay(LED_PERIOD);
}
```

Отступление (для продвинутых):

В реальном контроллере между соседними вызовами digitalWrite() будет, разумеется, некая микросекундная задержка. Для данной задачи это никакой роли не играет, но в некоторых ситуациях может быть важным. Используя только вызовы из библиотеки Ардуино, невозможно строго одновременно управлять несколькими пинами.

Сам микроконтроллер ATMEGA, тем не менее, это делать позволяет. На аппаратном уровне, "пины" (входы/выходы) контроллеры объединены в 8-битовые "порты", доступные как регистры контроллера, а на уровне языка C - как специальные переменные. Таким образом, можно строго одновременно, за 1 такт микроконтроллера, менять состояние до 8 пинов!

Если вы загуглите "ATMEGA ArduinoUNO pin mapping то найдете, например, вот такую картинку:



В контексте данной задачи, нам важен тот факт,что пинам Arduino D11 и D12 соответствуют биты 3 и 4 порта B контроллера ATMEGA. На языке C, к ним можно обратиться следующим образом (причем это действует не только в "настоящей"среде разработки AVRStudio, но и в среде Arduino!):

```
void setup()
{
   DDRB =  0b00011000;    // заменяет pinMode, устанавливая PB3 и PB4 в OUTPUT
   PORTA = 0b00010000;    // начальное состояние: PB3(D11) = 0, PB4(D12) = 1
}
void loop()
```

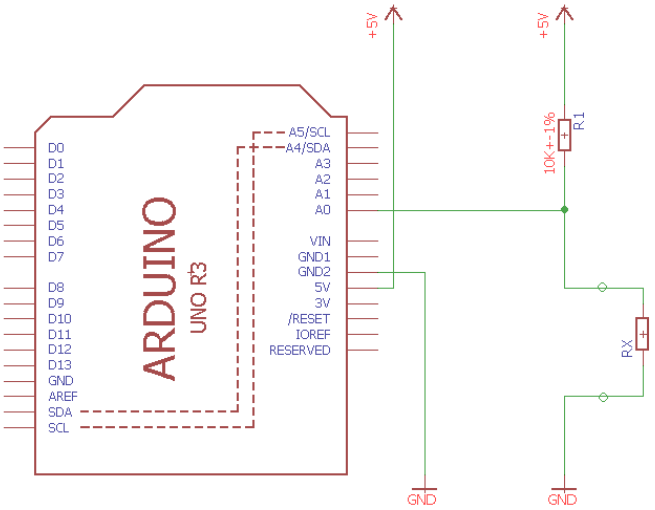
```
7  {
8      PORTA ^= 0b00011000;  // одной командой переключаем оба пина!
9      delay(LED_PERIOD);
10 }
```

Такая программа не только выполняется на реальном контроллере гораздо быстрее, но и занимает меньше места в памяти программ, чем код с вызовами функций библиотеки Ардуино. Обратите внимание, что если бы светодиоды были подключены, например, к пинам D7 и D8, то показанный выше трюк у нас бы не прошел, т.к. эти пины относятся к разным портам и не могут изменяться одной командой.

К сожалению, в мини-симуляторе низкоуровневая работа с портами не поддерживается, поэтому приведенный выше код не может быть решением данной задачи в Stepik. Тем не менее, любознательным среди вас составители задания советуют попробовать такую программку на реальном Ардуино и, по документации к контроллеру ATMega разобраться с псевдо-переменными PORTx, PINx и DDRx.

Задача 5.2.3. Простейший омметр (2 балла)

Вам нужно реализовать на Arduino простейший омметр (измеритель сопротивления), как показано на схеме.



Для этого к пину A0 подключен делитель напряжения, состоящий из прецизионного резистора 10 кОм "сверху"и измеряемого резистора "снизу". После того, как очередной измеряемый резистор подключен к схеме, контроллер включается, программа на Arduino производит измерение и через последовательный порт выводит в виде числа значение сопротивления, в кОм, с округлением до целого и с переводом строки на конце.

Для упрощения, мы принимаем, что измеряемое сопротивление всегда целое число килоом и отсутствуют любые ошибки измерения. Все значения измеряемых резисторов находятся в диапазоне от 10 кОм до 90 кОм (т.е. допускающем их сравнительно точное измерение при 10кОм подтягивающем резисторе).

Напишите программу для мини-симулятора Arduino, которая выполняет измерения.

Данные для измерения поступают из входного потока и расшифровываются мини-симулятором, а вычисленные вашей программой и отправленные на Serial.println() значения передаются в выходной поток для проверки. Мини-симулятор однократно вызывает функцию loop() для выполнения каждого измерения.

Разрешается пользоваться функциями pinMode(), analogRead() , а также упрощенным классом Serial с методами begin(), print() и println(). Передача данных должна происходить на скорости 9600 бод.

Решение

Очевидно, что это задача на использование имеющегося в микроконтроллере ATMEGA аналого-цифрового преобразователя (АЦП). На уровне библиотеки Arduino, чтение аналогового значения на входе АЦП выполняется функцией analogRead(). Эта функция возвращает значения в диапазоне от 0 (на входе - нулевое напряжение) до 1023 (напряжение питания). При подключении на аналоговый вход делителя напряжения, изображенного на схеме, напряжение на входе будет составлять:

$$V_x = V_{cc} \cdot R_x/(R_1 + R_x),$$

а значение, принятое с АЦП:

$$X_{ADC} = 1023 \cdot R_x/(R_1 + R_x)$$

Решая это уравнение относительно R_x , получаем:

$$R_x = R_1 \cdot X_{ADC}/(1023 - X_{ADC})$$

При реализации вычислений на C, всегда следует обращать внимание на правильность задания типов данных и на возможные ошибки округления. Наш код может выглядеть так:

```
#include <math.h>
Float R1 = 10.0;
void loop()
{
    int x = analogRead(A0);
    float R = R1 * x / (1023 - x);
    Serial.println(round(R));
}
```

Разумеется, нельзя забывать и про правильную инициализацию как пинов, так и последовательного порта (хотя при старте Arduino все пины уже и так находятся в режиме INPUT, желательно прописать pinMode() просто для большей ясности):

```
void setup()
{
    pinMode(A0, INPUT);
    Serial.begin(9600);
}
```

5.3. Arduino: АЦП и делители напряжения

Задача 5.3.1. Асинхронный маячок (3 балла)

К цифровым пинам контроллера Ардуино подключены 3 светодиода разных цветов (красный, желтый и зеленый). Напишите программу, которая будет одновременно мигать этими светодиодами, причем каждым - со своим периодом и скважностью. В начальный момент времени все 3 светодиода должны включиться одновременно.

Входные данные мини-симулятор считывает автоматически, но вам нужно использовать переменные, как указано в шаблоне кода. Как и в предыдущем примере, вам доступны функции pinMode(), digitalWrite() и delay().

Решение

Задачка с тремя светодиодами требует совсем другого подхода: мы должны использовать в конце функции loop() ровно одну короткую задержку (например, на 1 мс) и для каждого светодиода использовать отдельную переменную-счетчик, которая увеличивается при каждом проходе цикла. При заданных пороговых значениях счетчика светодиод может включаться или отключаться. По сравнению с фиксированной задержкой в конце цикла, время выполнения остальной части кода (проверки счетчиков и управление пинами) пренебрежимо мало.

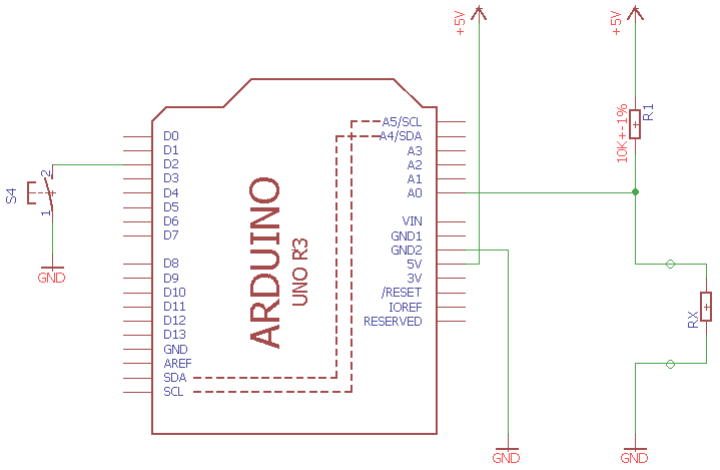
Пример программы-решения

```
int PIN_RED = 3;         // на этих пинах ваши светодиоды
int PIN_YELLOW = 4;
int PIN_GREEN = 5;

// Для каждого LED задается полный период мигания и длительность
// во включенном состояния.
// Данные считываются мини-симулятором из входного потока в эти переменные.
//и будут разными для каждого теста.

int RED_PERIOD, RED_ON_PERIOD;       // период мигания и длительность включенного
// состояния для КРАСНОГО
int YELLOW_PERIOD, YELLOW_ON_PERIOD; // то же для ЖЕЛТОГО
int GREEN_PERIOD, GREEN_ON_PERIOD;   // то же для ЗЕЛЕНОГО

int cnt_r=0, cnt_y=0, cnt_g=0; // счетчики для светодиодов

void setup()
{
    pinMode(PIN_RED, OUTPUT);
    pinMode(PIN_YELLOW, OUTPUT);
    pinMode(PIN_GREEN, OUTPUT);
}

void blink_led(int pin, int &cnt, int on_period, int period)
{
    if( !cnt ) //в начале периода ставим HIGH
        digitalWrite(pin, HIGH);
    else if( cnt >= on_period ) //по истечении времени включения - выключаем
        digitalWrite(pin, LOW);
```

```
30          cnt = (cnt+1) % period;  // циклически увеличиваем счетчик по модулю period
31      }
32
33      void loop()
34      {
35          blink_led(PIN_RED, cnt_r, RED_ON_PERIOD, RED_PERIOD);
36          blink_led(PIN_YELLOW, cnt_y, YELLOW_ON_PERIOD, YELLOW_PERIOD);
37          blink_led(PIN_GREEN, cnt_g, GREEN_ON_PERIOD, GREEN_PERIOD);
38          delay(1);
39      }
```

Задача 5.3.2. Усовершенствованный омметр (4 балла)

В задании на прошлой неделе вам предлагалось запрограммировать простой измеритель сопротивления. Вернемся вновь к этой теме, но внеся небольшие усовершенствования:



1. Теперь между пином D2 и землей подключен нормально-разомкнутый кнопочный выключатель. Измеряемые резисторы можно заменять без выключения устройства. После того, как очередной измеряемый резистор подключен к схеме, пользователь кратковременно нажимает кнопку. В момент отпускания кнопки, программа на Arduino производит измерение и выводит в виде числа значение сопротивления через последовательный порт. Если измерение проведено до отпускания кнопки, или более чем через 10ms после этого момента, analogRead() возвращает значение 1023 (как если бы измеряемый резистор еще не подключили).
2. Измеряемые сопротивления, в диапазоне от 10 кОм до 82 кОм, выбираются из номинального ряда E12 (Гугл в помощь!). Более того, как у настоящих резисторов, значения будут отклоняться от номиналов на случайную величину, не превышающую 10% номинала. Вашей программе нужно привести измеренное сопротивление к номиналу и вывести его в виде целого числа (в кОм) вызовом Serial.println(). Например, если ваша программа получила значение 54 кОм, то должно быть возвращено число 56 - ближайшее значение из номинального ряда. Данные должны передаваться на скорости 9600 бод.

Напишите программу для мини-симулятора Arduino, которая реализует описанный выше функционал: ждет нажатия и отпускания кнопки, проводит измерение, обрабатывает и выводит результат. Разрешается пользоваться функциями pinMode(),

digitalRead(), analogRead() и delay(), а также упрощенным классом Serial с методами begin(), print() и println().

ВАЖНО: в отличие от реального Arduino, в программе для мини-симулятора любые циклы ожидания должны включать вызов функции delay(), иначе в симуляторе "время не идет"и ваша программа "зациклится не сможет дождаться никаких внешних событий.

Стандартный вывод
19 818 869 518 906 523 741 510 692 556 869 517 789 509 739 844 849 614 648 500

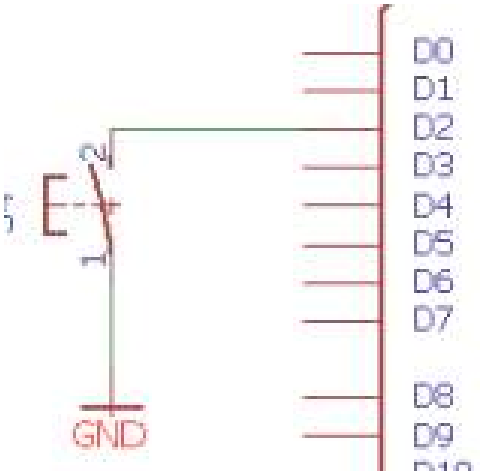
Стандартный вывод
39 56 10 82 10 27 10 22 12 56 10 33 10 27 47 47 15 18 10

Решение

Очевидно, что собственно измерение сопротивления выполняется здесь точно так же, как и в предыдущей задаче. Отличаться будет цикл ожидания нажатия кнопки до измерения и обработка ("нормализация") полученного значение после. Используя метод проектирования сверху-вниз, запишем сначала новую функцию loop(), а затем уже будем разбираться с дополнительными функциями:

```
float R1 = 10.0;
void loop()
{
    waitForButton(BUTTON_PIN);
    int x = analogRead(A0);
    float R = R1 * x / (1023 - x);
    Serial.println(normalizeResistor(R));
}
```

Чтобы поймать момент отпускания кнопки, нам сначала надо дождаться ее нажатия. Надо понимать, что при подключении кнопки между пином и землей, без внешнего подтягивающего резистора, режим пина должен быть обязательно выставлен в INPUT_PULLUP, иначе состояния пина при разомкнутой кнопке будет неопределенным. При этом, при замыкании кнопки на пине окажется 0 (LOW), а в разомкнутом состоянии - 1 (HIGH).



С учетом этого:

```
void waitForButton(int pin)
{
        while(digitalRead(pin))   // ждем нажатия кнопки
         delay(1);
        while(!digitalRead(pin))  // а затем - ее отпускания
         delay(1);
}
```

ВАЖНО: в отличие от реального Arduino, в программе для мини-симулятора любые циклы ожидания должны включать вызов функции delay(), иначе в симуляторе "время не идет"и ваша программа не сможет дождаться никаких внешних событий.

Следующее, с чем надо разобраться - это приведение измеренного значения к ближайшему стандартному значению из номинального ряда E12 (см. Википедию "Ряды номиналов радиодеталей"). Для начала, просто зададим этот ряд (в диапазоне 10 кОм - 90 кОм) в виде массива констант:

```
const int E12[] = {10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82 };
```

Самый простой способ нормализовать измеренное значение - просто сравнивать его поочередно с каждым из измеренных значений. Если отклонение не превышает $\pm 10\%$, то мы нашли нужный номинал:

```
int normalizeResistor(float R)
{
    for( int i = 0; i < 12; i++ ){
        int rn = E12[i];
        if( R >= 0.9 * rn && R < 1.1 * rn ) return rn; // номинал найден
    }
    return round(R);   // не должно случиться - возвращаем без нормализации
}
```

Такой алгоритм, однако, может давать неоднозначные ответы, поскольку 10% диапазоны для соседних значений слегка перекрываются. Например: $10 \cdot 1.1 = 11$, а $12 \cdot 0.9 = 10.8$. Таким образом, измеренное значение 10.9 кОм может соответствовать, строго говоря, номиналам как 10 кОм, так и 12 кОм. В данной задаче это не приведет к проблемам, так как генерируемые тестовые данные не будут настолько сильно отклоняться от своих номиналов, чтобы попасть в "серые зоны".

Тем не менее, чтобы избежать неоднозначности в определении номинала, можно сравнивать измеренное значение со средними арифметическими соседних номиналов. Например, для пары номиналов 12 кОм и 15 кОм естественной границей будет $(12+15)/2 = 13.5$ кОм. Вот как это можно сделать:

```
int normalizeResistor(float R)
{
  if( R < 0.9 * E12[0] ) return R; // меньше меньшего - возвращаем как есть
  if( R > 1.1 * E12[11] ) return R; // больше большего - возвращаем как есть

  for( int i = 0; i < 11; i++ ){    // не включая последний элемент
      float rn = (E12[i] + E12[i+1]) / 2.0;  // Граница между номиналами
      if( R < rn ) return E12[i]; // номинал найден
   }
   return E12[11];   // последний номинал
}
```

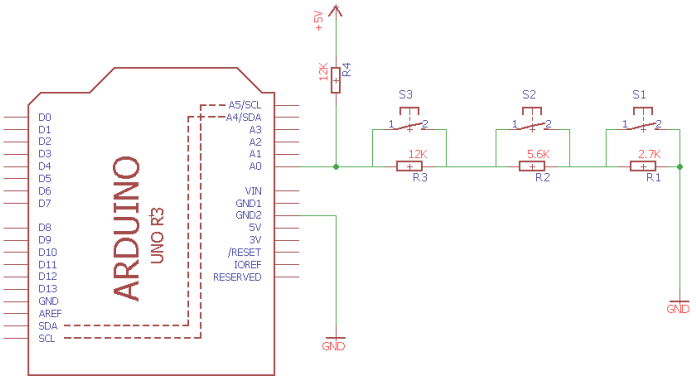
Ну и, наконец, не забываем об инициализации в функции setup(). По сравнению с предыдущей задачей, здесь добавляется установка режима INPUT_PULLUP для пина, к которому подключена кнопка:

```
void setup()
{
    pinMode(A0, INPUT);
    pinMode(BUTTON_PIN, INPUT_PULLUP);
    Serial.begin(9600);
}
```

Задача 5.3.3. Аналоговые кнопки (4 балла)

Если нужно подключить к устройству несколько кнопок, а пины контроллера приходится экономить, то подключают цепь из нескольких кнопок и резисторов к одному аналоговому входу, так что при разных комбинациях нажатий получаются разные сопротивления цепи, и, соответственно, разные напряжения на аналоговом входе.

И вот, имеется такая схема:



Напишите программу, которая сразу после включения определяет нажатые кнопки и передает через последовательный порт одно десятичное число, соответствующее их комбинации. Каждой из кнопок S1..S3 соответствует один бит в полученном числе. Например, нажатые кнопки S1 и S2 должны давать число 3 (2+1). Ожидаемая скорость передачи - 115200 бод. Программа должна учитывать, что сопротивления резисторов могут несколько (до 2.5% в каждую сторону) отклоняться от указанных на схеме значений.

В этом примере, мини-симулятор будет выполнять функцию loop() однократно для каждой комбинации нажатых кнопок. Никаких задержек и ожиданий в коде писать не надо. Используются функции analogRead(), pinMode() и упрощенный Serial, как в предыдущих заданиях.

Решение

Если нужно подключить к устройству несколько кнопок, а пины контроллера приходится экономить, то часто подключают цепь из нескольких кнопок и резисторов к одному аналоговому входу, так что при разных комбинациях нажатий получаются разные сопротивления цепи, и, соответственно, разные напряжения на аналоговом входе.

В данной схеме "верхний"резистор делителя $R4 = 12$ кОм, а "нижняя"ветка делителя состоит из 3-х последовательно соединенных резисторов ($R1 = 2.7$ К, $R2 = 5.6$ К, $R3 = 12$ К), каждый из которых может быть шунтирован при нажатии соответствующей кнопки. Для любой заданной комбинации кнопок (из 8 возможных) можно заранее вычислить напряжение на пине A0. Например, при замкнутых $S1$ и $S3$ в нижней части делителя останется только $R2 = 5.6$ К, и результат analogRead() будет равен:

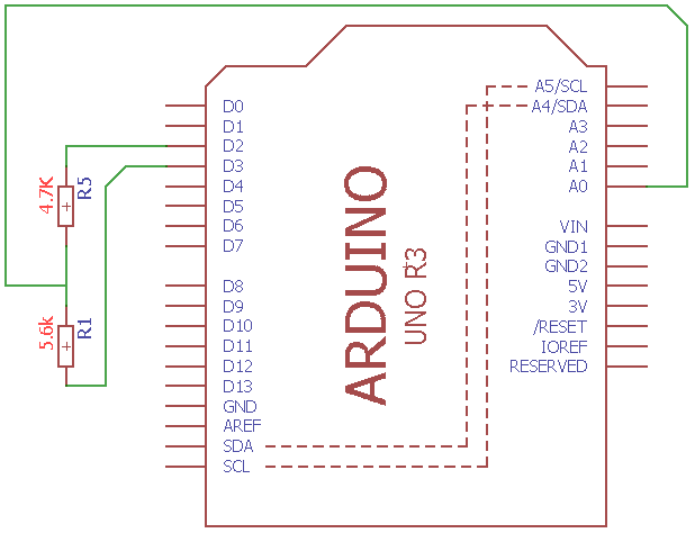
$$A = R2/(R2 + R4) \cdot 1023 = 5.6/(5.6 + 12) \cdot 1023 \approx 325$$

С учетом 2.5% допусков, диапазон значений analogRead() для этой комбинации кнопок составляет: $A_{min} = A \cdot 0.975 \approx 317$, $A_{max} = A \cdot 1.025 = 333$

Программа сравнивает полученное значение с каждым из рассчитанных таким образом диапазонов, определяя комбинацию нажатых кнопок. (Полный текст программы не приводится).

Задача 5.3.4. Загадочный делитель напряжения (1 балл)

Делитель напряжения из двух резисторов разного номинала подключен между пинами D2 и D3, а средняя точка подключена к аналоговому пину A0, как показано на схеме:



Значение напряжения измеряется при разных комбинациях настроек пинов. Считать, что напряжение на пине в выходном режиме в точности равно либо 0, либо напряжению питания (хотя в реальных схемах, логический "0"всегда чуть выше 0V, а логическая "1"чуть ниже напряжения питания). Сопротивление внутреннего подтягивающего резистора (на любом пине) принять за 10 кОм.

Нужно написать программу, которая рассчитывает все возможные напряжения, которые можно намерить на 0000000000A0, изменяя настройки пинов. В этой задаче не используется мини-симулятор Ардуино, вы просто пишете код на любом удобном для вас языке.

Формат входных данных

Два числа через пробел - сопротивления резисторов R1 и R2.

Формат выходных данных

На первой строке - число значений в ответе, на второй строке - последовательность целых чисел в диапазоне 0..1023, которые можно было бы получить функцией analogRead(A0) при различных настройках пинов.

Все числа в ответе должны быть на одной строке через пробел, упорядочены по возрастанию, повторяющиеся значения удалены.

Допускается отклонение вычисленных значений на 1 в любую сторону.

Решение

Чтобы понять эту задачу, следует помнить, что любой пин Arduino может находится в 4-х разных состояниях:

Состояние	Обозн.	Что происходит
pinMode(pin, OUTPUT); digitalWrite(pin, HIGH);	1	На пин подано напряжение питания
pinMode(pin, OUTPUT); digitalWrite(pin, LOW);	0	На пин подается 0 (подключен GND)
pinMode(pin, INPUT);	I	Пин отключен от каких-либо напряжений
pinMode(pin, INPUT_PULLUP);	U	Пин подключен к напряжению питания через подтягивающий резистор в 10K

Все 4 перечисленных режима относятся как к пинам D2, D3, так и к пину A0, на котором производятся измерения. Когда мы говорим про измерение аналогового сигнала функцией analogRead(), всегда предполагается, что аналоговый пин находится в режимах INPUT или INPUT_PULLUP. Однако ничего не мешает установить A0 в режим OUTPUT, подать на него LOW или HIGH и прочитать значение полученного напряжения, если это зачем-то вдруг понадобилось.

Итак, мы можем получить $4 \cdot 4 \cdot 4 = 64$ комбинации состояний 3х пинов. Многие из них дадут уникальные комбинации сопротивлений и уникальные значения измеряемого напряжения на средней точке делителя. Следующая функция на Python вычисляет выходное напряжение при различных комбинациях состояний трех пинов (обозначены '0', '1', 'U', 'I' - см. выше):

Пример программы-решения

Ниже представлено решение на языке Python3

```
R1 = 4.7
R2 = 5.6
R_Int = 10.0

def find_value(A, D1, D2):
  # A0 - OUTPUT, полностью определяет результат
  if A == '0':
    return 0
  elif A == '1':
    return 1023

  if (D1 in '1U') and (D2 in '1U'):  # оба HIGH or PULLUP
    return 1023

  if D1 == '0' and D2 == '0':  # R1,R2 параллельно на GND
    if A == 'I':
      return 0  #  GND независимо от R1/R2
```

```
    Else:   #  R1, R2 параллельно, R_int сверху делителя
      r_dn = 1 / (1/R1 + 1/R2)
      r_up = R_Int
      return int(1023 * r_dn / (r_up + r_dn) )

  # Считаем верх делителя: начать с бесконечности, учитывать все
  # резисторы, идущие параллельно
  r_up = math.inf
  if A == 'U':
    r_up = 1 / (1/R_Int + 1/r_up)
  if D1 == '1':
    r_up = 1 / (1/R1 + 1/r_up)
  elif D1 == 'U':  # R_Int встает последов. с R1
    r_up = 1 / (1/(R1+R_Int) + 1/r_up)

  if D2 == '1':
    r_up = 1 / (1/R2 + 1/r_up)
  elif D2 == 'U':
    r_up = 1 / (1/(R2+R_Int) + 1/r_up)


  # Аналогично с "нижними" резисторами, но тут только могут быть R1 и R2
  r_dn = math.inf
  if D1 == '0':
    r_dn = 1 / (1/R1 + 1/r_dn)
  if D2 == '0':
    r_dn = 1 / (1/R2 + 1/r_dn)


  # Все 3 пина поставлены на INPUT, неопределенное напряжение.
  if r_up == math.inf and r_dn == math.inf:
    return None

  if r_dn == math.inf:
    return 1023
  if r_up == math.inf:
    return 0

  return int(1023 * r_dn / (r_up + r_dn) )


# Полный перебор состояний пинов
results = []
for A0 in "IU01":
  for D1 in "IU01":
      for D2 in "IU01":
        v = find_value(A0, D1, D2)
        if not (v is None):
          results.append(v)


# Печатаем все уникальные решения, с сортировкой по возрастанию.
arr = [str(v) for v in sorted(set(results))]
print(len(arr))
print(' '.join(arr))
```

5.4. Шаговые двигатели и координатные устройства

Шаговые двигатели - основа самых разных координатных устройств, в том числе станков с ЧПУ, а среди них и самых распространенных - 3D-принтеров (да и обычных бумажных принтеров тоже!)

В отличие об обычных электромоторов, шаговый двигатель не просто "вращается а поворачивается на заданный угол, с весьма хорошей точностью, при подаче импульсов определенной формы, полярности и продолжительности на его 2 обмотки. Для управления шаговыми двигателями используют специальные ' микросхемы-драйверы. Типичный драйвер управляется с микроконтроллера по 3-м пинам ENABLE, DIR и STEP:

ENABLE - включение двигателя. Когда шаговый двигатель включен, ток протекает по обмоткам и блокирует движение ротора, так что прокрутить вал можно, только приложив значительное усилие. Часто сигнал ENABLE для драйвера имеет инверсную полярность, т.е. двигатель включается при установке низкого уровня сигнала.

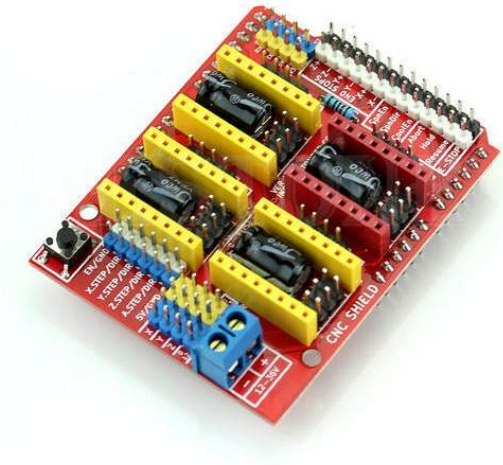
DIR - уровень сигнала определяет направление вращения.

STEP - каждый короткий импульс, подаваемый на этот пин, поворачивает вал шагового двигателя на 1 шаг.

Драйвер шагового двигателя несложно подключить к Ардуино прямо на макетной плате, но гораздо удобнее пользоваться специальными шилдами. На следующей картинке показан"бутерброд"из Arduino UNO (внизу), CNC ShieldV3 и 4-х драйверов моторов сверху. Это типичный расклад для любительского настольного фрезерного станка или плоттера. Для 3D-принтеров чаще применяют аналогичный "бутерброд но на базе Arduino Mega и шилда RAMPS.

Задача 5.4.1. Какие пины? ("Google it!") (1 балл)

Когда мы подключаем драйвер шаговых двигателей на макетной плате, то мы свободны в выборе пинов Arduino, которые будут им управлять. Но если мы используем CNC Shield, то вся "распиновка"определяется платой. Найдите в интернете информацию о CNC Shield V3 для Arduino UNO и заполните пробелы.



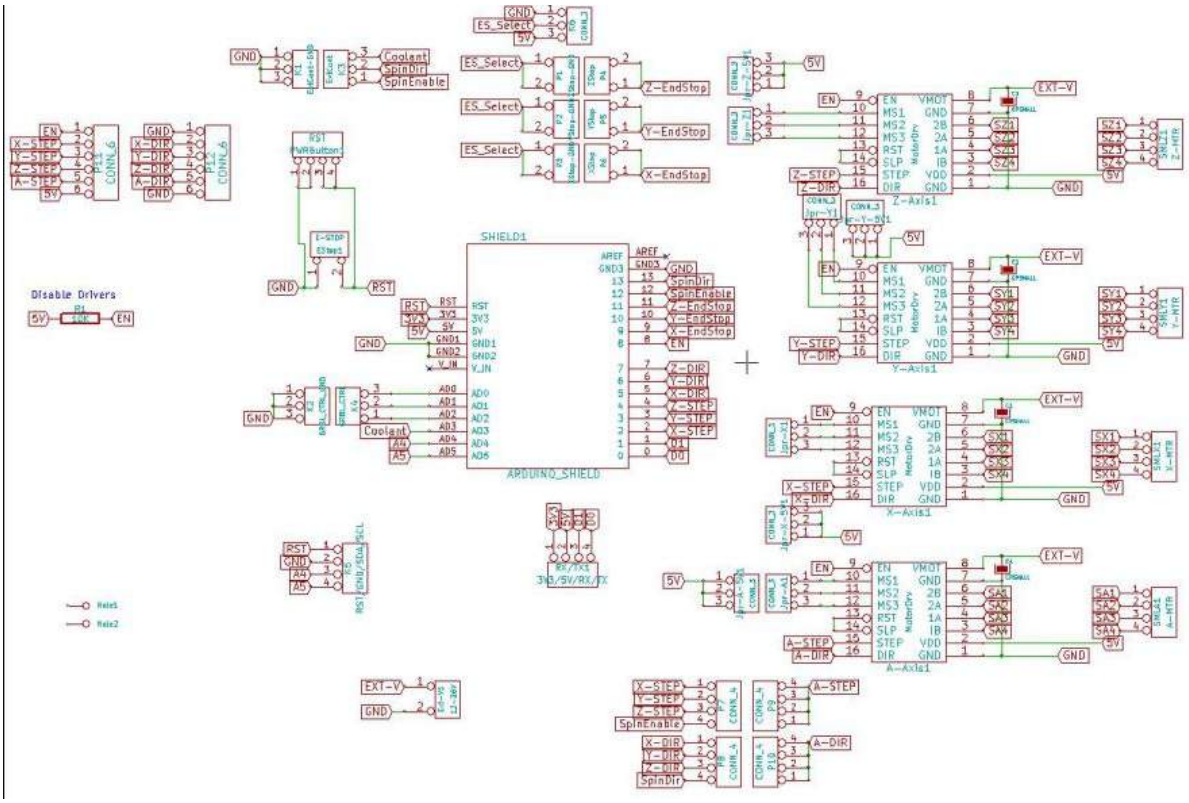
а) При использовании этой платой сигнал ENABLE:
   (a) Отдельный для каждого мотора
   (b) Общий для всех моторов
   (c) Всегда включен
   (d) Общий для каналов X и Y

б) Сигнал STEP для канала X подключен к пину Arduino:
   (a) D1
   (b) D2
   (c) D3
   (d) D4
   (e) D5
   (f) D6
   (g) D7
   (h) D8

в) Сигнал DIR для канала Z подключен к пину Arduino:
   (a) D1
   (b) D2
   (c) D3
   (d) D4
   (e) D5
   (f) D6
   (g) D7
   (h) D8
   (i) D9
   (j) D10

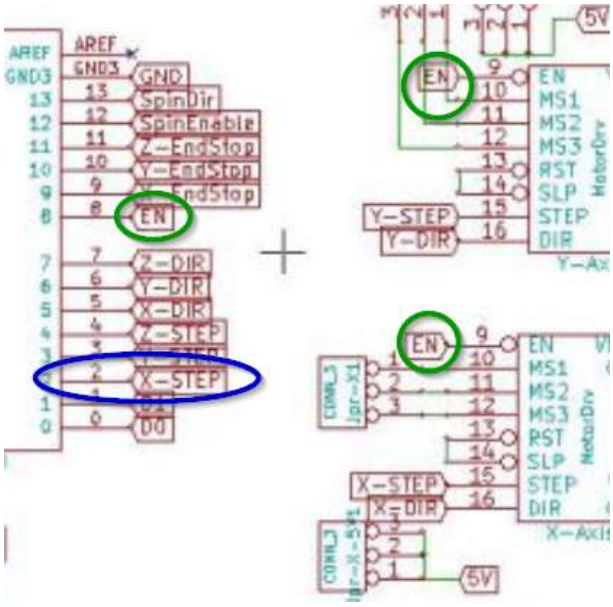
Решение

Решение этой задачи очевидно из условия. Ищем "CNC shield pinout", и находим вот такую схему:



Посмотрев подробнее на подключение драйверов, мы видим, что:

- Сигнал EN (а точнее - NOT ENABLE, т.к. драйверы включаются при подаче 0 на EN) разведен с пина 8 на все драйвера,
- Сигнал шаг X_STEP подключен к пину 2, Y_STEP к пину 3, Z_STEP к пину 4
- Сигнал направления X_DIR подключен к пину 5 и т.д.



Этого достаточно, чтобы правильно ответить на вопрос задания. На практике, следует иметь в виду, что разводка конкретной "китайской" платы может отличаться от найденной вами в интернете, поэтому крайне желательно проверять ее по дорожкам на плате и/или мультиметром.