

Санкт-Петербургский государственный электротехнический университет
"ЛЭТИ" им. В. И. Ульянова (Ленина)
(СПбГЭТУ "ЛЭТИ")

Направление: **27.04.04** - Управление в технических системах
Профиль: Управление и информационные технологии в технических системах
Факультет: Компьютерных технологий и информатики
Кафедра: Автоматики и процессов управления

К защите допустить
Зав. кафедрой

Шестопалов М. Ю.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
МАГИСТРА**

**Тема: Параметрическое проектирование дельта-робота и решение
задачи координатного управления рабочим органом**

Студент _____ О.Е. Медовиков

Руководитель _____ С. Е. Абрамкин
К. Т. Н.

Санкт-Петербург
2020

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю
Заф. кафедры АПУ
_____ Шестопалов М. Ю.
«___» _____ 2020 г.

Студент Медовиков О. Е.

Группа 4391

Тема работы:

Параметрическое проектирование дельта-робота и решение задачи
координатного управления рабочим органом.

Исходные данные (технические требования):

1. Написание программы для параметрического моделирования дельта-робота
2. Написание программы для управления дельта-роботом
3. Создание рабочей модели дельта-робота

Содержание ВКР:

Перечень отчетных материалов: пояснительная записка, иллюстративный
материал, приложение.

Дополнительные разделы:

Дата выдачи задания
«___» _____ 2020 г.

Дата предоставления ВКР к защите
«___» _____ 2020 г.

Студент

_____ О.Е. Медовиков

Руководитель

К. Т. Н.

_____ С. Е. Абрамкин

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю
Заф. кафедры АПУ
Шестопалов М. Ю.
«__» _____ 2020 г.

Студент Медовиков О. Е.

Группа 4391

Тема работы:

Параметрическое проектирование дельта-робота и решение задачи
координатного управления рабочим органом.

№ п/п	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	10.12-01.02
2	Проектирование виртуальной модели	10.12 - 26.03
3	Создание физического прототипа робота	01.02 - 05.04
4	Написание прошивки для микроконтроллера	
5	Создание интерфейса для управления роботом	

Студент

_____ О.Е. Медовиков

Руководитель

к. т. н.

_____ С. Е. Абрамкин

РЕФЕРАТ

Пояснительная записка 00 стр., 00 рис., 00 табл., 00 ист., 00 прил.

Ключевые слова: параметрическое моделирование, 3д печать, дельта-робот, сортировка.

Объект исследования: кинематика дельта-робота.

Цель работы:

Основное содержание работы.

ABSTRACT

This work is a report on the work done on the design and creation of a delta robot.

Objectives of work:

Analysis of the mathematical solution of the direct and inverse problems of controlling a delta robot.

Creating a digital copy of the future robot with the ability to change parameters, such as the radius of the base, the length of the levers and the length of the articulated joints. Build a valid model based on printed parts.

Write a sketch for Arduino that implements control of stepper motors and communication with a computer via a com-port.

Write a delta robot control program capable of arbitrarily changing the coordinate of the robot carriage.

The result of the work is a robot model, analysis of structural defects, plans for subsequent modernization of the structure.

СОДЕРЖАНИЕ

Введение	9
1 Кинематика дельта-робота	10
1.1 Конструкция и устройство	10
1.2 Задача прямой кинематики дельта-робота	11
1.3 Обратная кинематика	15
2 Моделирование робота	18
2.1 ZenCad	18
2.2 Ласточкин хвост (Создание произвольного полигона) .	20
2.3 База (Создание моделей и булевы операции)	21
2.4 Глобоидная червячная передача (Параллельные вычисления)	24
2.5 Вспомогательные модели (Сборочные юниты)	30
2.6 Движение каретки (Создание анимации)	32
2.7 Расчет координат и углов объектов (Вращение координат)	35
2.8 Создание моделей (Экспорт в STL)	41
2.9 Вывод	42
3 Программирование Arduino	44
3.1 Драйвера двигателей	44
3.2 Распиновка Arduino CNC Shield	45
3.3 Основная функция	47
3.4 Функция возврата каретки домой	48
3.5 Функция изменения координаты	50
3.6 Вывод	53
4 Технико-экономическое обоснование	54
4.1 Описание проекта	54
4.1.1 Резюме	54

4.1.2	Описание продукции	55
4.1.3	Анализ рынка сбыта	56
4.1.4	Анализ конкурентов	56
4.2	План маркетинга	56
4.2.1	Анализ рынка	57
4.2.2	Ценовая политика	57
4.2.3	Сбытовая политика и мероприятия	58
4.3	План производства	58
4.4	Финансовый план	61
Заключение		63
Список		65

ПРИМЕРЫ КОДА

1	Создание полигона из массива точек	20
2	Линейная развертка (экструдирование)	21
3	Булева операция пересечения	23
4	Работа с линиями объединение	25
5	Код функции создающей червя	29
6	Пример сборочного юнита	31
7	Растановка объектов методом полярных координат . . .	32
8	Работа со временной переменной	33
9	Стробоскопический эффект на экране монитора	33
10	Рисование максимальных траекторий дельта-робота . .	34
11	Простые анимированные объекты	35
12	Решение прямой задачи управления дельта-робота . . .	37
13	Тригонометрический способ ориентирования шарниров	39
14	Векторный поворот первого шарнира	40
15	Векторный поворот второго шарнира	41
16	Создание моделей STL	42
17	Объявление пинов Arduino	46
18	Основная функция скетча	48
19	Функция возврата каретки домой	49
20	Функция управляющая шаговым двигателем	50
21	Функция изменения координаты	50
22	Усложненная функция с плавным разгоном	51

ВВЕДЕНИЕ

Данная работа является отчетом о проведенной работе по проектированию и созданию дельта-робота.

Цели работы:

Разбор математического решения прямой и обратной задачи управления дельта-роботом.

Создание цифровой копии будущего робота с возможностью менять параметры, такие как радиус базы, длины рычагов и длины шарнирных соединений. Построить на основе напечатанных деталей действующую модель.

Написать скетч для Arduino, реализующий управление шаговыми двигателями и общение с компьютером, посредством com-порта.

Написать программу управления дельта роботом, способную произвольно менять координату каретки робота.

Результатом работы является модель робота, анализ дефектов конструкции, планы на последующие модернизации конструкции.

1 Кинематика дельта-робота

1.1 Конструкция и устройство

Основанием робота является база, жёстко фиксируемая в пространстве над рабочем полем. Габариты базы очерчиваются равносторонним треугольником со стороной равной f . Середины сторон треугольника обозначают координаты осей вращения рычагов и таким образом, расстояние от центра базы до оси вращения каждого рычага равно r - радиусу вписанной окружности равностороннего треугольника. Это расстояние легко находится через соотношение:

$$f = \frac{\sqrt{3}}{2} r$$

В дальнейшей работе, при описании моделирования, будут использоваться переменные с другими названиями, например, переменная rad соответствует радиусу вписанной окружности. Это связано с удобством написания кода, так как невозможно поиск найти переменную, обозначенную одним символом, а также это неправильно, с точки зрения читаемости кода.

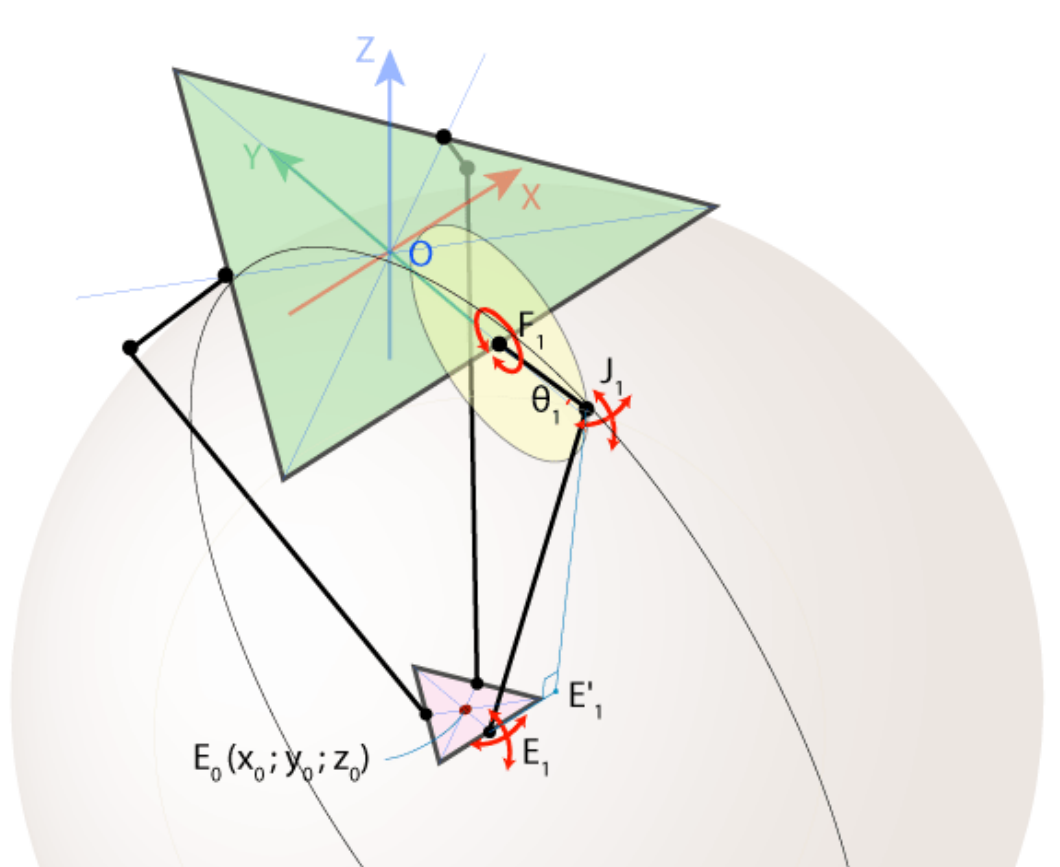


Рис. 1: Схематическое представление Дельта-робота

Начало координат располагается в центре базы, таким образом, чтобы Z координата высоты равнялась нулю для точек осей вращения рычагов, так как ко-

нежное расположение рабочего органа робота будет рассчитываться относительно этих координат. Три рычага нумеруются определённым образом. Первый рычаг двигается в плоскости YZ и направлен в противоположную оси Y сторону. Вторым рычаг повернут относительно оси Z на 120 градусов, а третий на -120 градусов. Поворот делается по правилу правой руки, где большой палец совпадает с направлением оси Z , а согнутые пальцы показывают направление вращения. Так как робот в целом абсолютно симметричен, ошибки с нумерацией рычагов закономерны, необходимо на всех этапах строго придерживаться единому правилу обозначения рычагов.

Жёстко закреплённые каждый в своей плоскости рычаги обозначаются r_{fi} , а угол на который они поворачиваются обозначают через θ_i . Точка оси вращения рычагов обозначается как F_i , а конечная точка рычага - J_i . На конце рычага находится крепление с двумя карданными шарнирами, которое всегда параллельно стороне равностороннего треугольника, обозначающего габариты рабочего органа. Две взаимно параллельные направляющие соединяются через шарниры с вершинами треугольника, образуя параллелограмм. Из-за этого, данный робот также называют разновидностью параллельного робота.

Для математического описания робота карданные шарниры и параллельные направляющие не нужны, их заменяют рычагами обозначаемыми как r_{ei} . Рычаги r_{ei} крепятся к серединам сторон треугольника, обозначающего габариты каретки, в которой закреплён рабочий орган. Габариты обозначаются, как и в случае с базой, равносторонним треугольником, длина стороны которого обозначается буквой e . Координаты точек крепления карданных шарниров к каретке называют E_i , а точкой E_0 обозначается центр каретки, то-есть координата рабочего органа.

1.2 Задача прямой кинематики дельта-робота

Решение прямой задачи кинематики дельта-робота заключается в определении координаты центра каретки E_0 при известных углах θ_i . Решение данной задачи необходима мне для определения координат расположения различных узлов машины, во время создания компьютерной модели. Сама идея решения достаточно проста. Так как рычаги, соединённые с двигателем, двигаются в одной плоскости, без возможности отклониться, это значит, что можно рассчитать координаты вершины рычага, зная координату оси вращения, длину рычага и угол поворота рычага. Координата конца рычагов обозначается буквой J_i . Подобным образом посчитать угол шарнира, соединяющего конец рычага и сторону каретки не представляется возможным, так как он вращается не вдоль одной плоскости, а в трёх измерениях.

Если допустить, что каретка не имеет размеров и представляет собой точку, то можно представить три сферы с центрами в J_i и радиусами r_{ei} . Сферы показывают область, в которой могут теоретически вращаться шарниры, при данных

значениях углов θ_i . Если внести поправки на размеры каретки, точка пересечения трех сфер - будет решением, искомой координатой каретки.

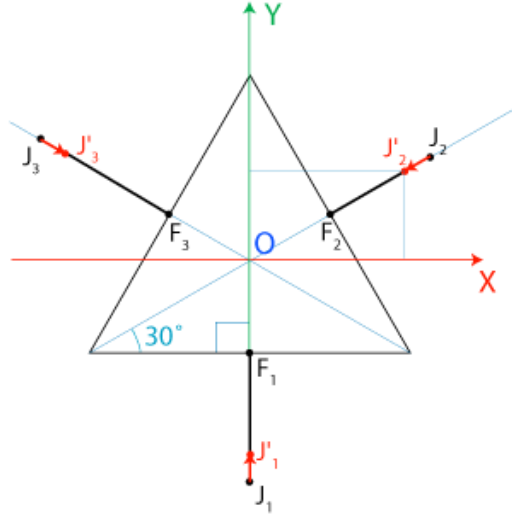


Рис. 2: Схема расчета координат рычагов

Расчет координат J_1 для первого рычага упрощается выбором системы координат. Первый рычаг параллелен оси Y и движется в плоскости YZ , поэтому координата X всегда будет равна 0. При этом Z координата оси вращения тоже равна 0, что было оговорено ранее. Значит координата F_1 будет состоять только из Y и будет равна минус радиус вписанной окружности. В этом месте нужно взять поправку на радиус каретки, и вычесть радиус каретки из радиуса базы. Таким образом, мы сможем рассчитать точки J'_i - центры сфер, с общей точкой в E_0 .

$$\begin{aligned} t &= r_{base} - r_{karet} \\ J'_1 &= (x_1; y_1; z_1) \\ J'_2 &= (x_2; y_2; z_2) \\ J'_3 &= (x_3; y_3; z_3) \end{aligned}$$

$$\begin{cases} x_1 = 0 \\ y_1 = -(t - r_f \cos(\theta_1)) \\ z_1 = -r_f \cos(\theta_1) \end{cases}$$

$$\begin{cases} x_2 = [t + r_f \cos(\theta_2)] \cos(30^\circ) \\ y_2 = [t + r_1 \cos(\theta_2)] \sin(30^\circ) \\ z_2 = -r_f \sin(\theta_2) \end{cases}$$

$$\begin{cases} x_3 = [t + r_f \cos(\theta_3)] \cos(30^\circ) \\ y_3 = [t + r_1 \cos(\theta_3)] \sin(30^\circ) \\ z_3 = -r_f \sin(\theta_3) \end{cases}$$

Теперь для нахождения координаты каретки, нужно решить систему из трех уравнений сфер с координатами центров в J'_i и радиусами r_e .

$$E_0 = (x, y, z) \quad (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = r_e^2$$

Подставим координаты J'_i , полученные ранее и получим систему уравнений вида:

$$\begin{cases} x^2 + (y - y_1)^2 + (z - z_1)^2 = r_e^2 \\ (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = r_e^2 \\ (x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = r_e^2 \end{cases}$$

Теперь раскроем скобки и немного сгруппируем переменные:

$$\begin{cases} x^2 + y^2 + z^2 - 2y_1y - 2z_1z = r_e^2 - y_1^2 - z_1^2 \\ x^2 + y^2 + z^2 - 2x_2x - 2y_2y - 2z_1z = r_e^2 - x_2^2 - y_2^2 - z_2^2 \\ x^2 + y^2 + z^2 - 2x_3x - 2y_3y - 2z_1z = r_e^2 - x_3^2 - y_3^2 - z_3^2 \end{cases}$$

Теперь можно сделать подстановку и формируем новые три уравнения, вычитая из первого сначала второе, потом третье и из второго - третье.

$$\omega_i = x_i^2 + y_i^2 + z_i^2$$

$$\begin{cases} x_2x + (y_1 - y_2)y + (z_1 - z_2)z = (\omega_1 - \omega_2)/2 \\ x_3x + (y_1 - y_3)y + (z_1 - z_3)z = (\omega_1 - \omega_3)/2 \\ (x_2 - x_3)x + (y_2 - y_3)y + (z_2 - z_3)z = (\omega_2 - \omega_3)/2 \end{cases}$$

Следующим шагом вычитаем второе уравнение из первого, частично сократив y выразив x через z . Аналогично вычитаем из второго третье, частично сокращая x и выражая y через z . Так как выражения получаются очень длинными, для компактной записи вводится подстановка a_i, b_i, d .

$$x = a_1z + b_1 \quad y = a_2z + b_2$$

$$\begin{aligned} a_1 &= \frac{1}{d}[(z_2 - z_1)(y_3 - y_1) - (z_3 - z_1)(y_2 - y_1)] \\ b_1 &= -\frac{1}{2d}[(\omega_2 - \omega_1)(y_3 - y_1) - (\omega_3 - \omega_1)(y_2 - y_1)] \end{aligned}$$

$$\begin{aligned} a_2 &= -\frac{1}{d}[(z_2 - z_1)x_3 - (z_3 - z_1)x_2] \\ b_2 &= \frac{1}{2d}[(\omega_2 - \omega_1)x_3 - (\omega_3 - \omega_1)x_2] \end{aligned}$$

$$d = (y_2 - y_1)x_3 - (y_3 - y_1)x_2$$

Теперь, имея x и y , выраженные через z , предстоит подставить их в уравнение сферы (например, первой) с центром в J_1 , раскрыть скобки, упростить и получить:

$$(a_1^2 + a_2^2 + 1) * z_0^2 + 2(a_1 + a_2(b_2 - y_1) - z_1) * z_0 + (b_1^2 + (b_2 - y_1)^2 + z_1^2 - r_e^2) = 0$$

В конечном итоге задача свелась к решению квадратного уравнения, через дискриминант, корни которого будут равны Z координате каретки. На данном этапе проводится проверка параметров дельта-робота. Человек произвольно задающий радиусы базы и каретки, длины рычагов и шарниров должен подобрать их в определённом соотношении, которое позволит роботу физически функционировать. Иначе, шарниры будут слишком короткими и не дотянутся до каретки. Решение уравнения выше позволяет определить физическую возможность создания робота при данных параметрах. Если дискриминант равен отрицательному числу, значит, что шарниры не дотягиваются до каретки и поэтому выбор параметров робота неверен. Если дискриминант равен 0 в рабочей области робота это приводит к неустойчивому равновесию. Эта координата называется точкой сингулярности параллельного робота, так как в её окрестностях находятся координаты с двумя равнозначными и очень близкими решениями. В окрестностях точки сингулярности управление роботом практически невозможно, так как движение вверх или вниз по оси Z будет случайным. Некоторые конструкции параллельных роботов, проходя точку сингулярности, "защёлкиваются" в положение, из которого не могут выйти самостоятельно. Для правильной работы робота дискриминант должен быть большим числом, в случае моей симуляции числа достигают значений $1.4 * 10^{25}$ и даже больше.

Пусть записанное выше уравнение имеет вид:

$$a * z_0^2 + b * z_0 + c = 0, \text{ тогда:}$$

$$d = b^2 - 4(ac)$$

$$z_0 = \frac{-b + \sqrt{d}}{2a}$$

$$y_0 = \frac{a_2 z_0 + b_1}{dnm}$$

$$x_0 = \frac{a_1 z_0 + b_2}{dnm}$$

Решение уравнения дает два корня и геометрически, в большинстве случаев, у каретки есть два возможных положения. Но физически, так как есть сопротивление материалов, положение только одно. У используемой конструкции параллельного робота, нет положения, когда бы дискриминант равнялся нулю, а это значит, что нельзя взять каретку и поднять её, вывернув шарниры вверх. Поэтому надо брать только нижний корень z_0 .

1.3 Обратная кинематика

Задача обратной кинематики дельта-робота заключается в нахождении углов поворота рычагов θ_i , при известной координате каретки $E_0 = (x_0, y_0, z_0)$. Данное решение основано на нахождении координат деталей первого шарнира и рычага, и вывода решения для угла θ_1 в общем виде для первого рычага с учётом правильного расположения осей координат. Два оставшихся угла будут рассчитаны аналогично, с применением вращения оси координат на 120° и -120° соответственно.

Первым шагом необходимо найти координаты крепления шарнира к каретке. Эта точка находится на стороне равностороннего треугольника и смещена от точки E_0 на величину радиуса вписанной окружности.

$$E_1(x_0, y_0 - \frac{e}{2\sqrt{3}}, z_0)$$

Так как в общем виде каретка будет иметь некое смещение по оси X , а это значит, что шарнир и рычаг не будут лежать в одной плоскости YZ . Для решения необходимо найти проекцию шарнира на плоскость YZ . Верхняя точка проекции шарнира совпадает с координатой конца рычага J_1 , а нижняя точка обозначается E'_0 .

$$E'_1(0, y_0 - \frac{e}{2\sqrt{3}}, z_0)$$

Соответственно длина проекции шарнира находится по теореме Пифагора:

$$E'_1 J_1 = \sqrt{(E_1 J_1)^2 - (E_1 E'_1)^2}$$

Так как гипотенуза равна длине шарнира, а меньший катет - смещению каретки по X , то:

$$E'_1 J_1 = \sqrt{r_e^2 - x_0^2}$$

Напомню, что координата оси вращения первого рычага F_1 смещена от центра координат на радиус вписанной окружности, таким же образом, как и крепление шарнира каретки.

$$F_1(0, \frac{-f}{2\sqrt{3}}, 0)$$

Теперь есть все необходимое, для нахождения координаты соединения рычага с шарниром J_1 . Вращаясь рычаг описывает окружность с радиусом r_f и центром в F_1 . Проекция шарнира вращаясь описывает окружность с найденным выше радиусом $E'_1 J_1$ и центром в E'_1 . Найдя точки пересечения этих двух окружностей,

мы получим две физически возможные координаты точки соединения рычага и шарнира, одна из которых ложная, а вторая (наименьшая по Y) истинная. Общие точки находятся путём решения системы уравнений двух окружностей. В данной записи y_{J_1} - это Y координата точки J_1 , «локтя» робота.

$$\begin{cases} (y_{J_1} - y_{F_1})^2 + (z_{J_1} - z_{F_1})^2 = r_f^2 \\ (y_{J_1} - y_{E'_1})^2 + (z_{J_1} - z_{E'_1})^2 = r_e^2 - x_0 \end{cases}$$

Подставляем известные координаты центров окружностей:

$$\begin{cases} (y_{J_1} + \frac{f}{2\sqrt{3}})^2 + z_{J_1}^2 = r_f^2 \\ (y_{J_1} - y_0 + \frac{e}{2\sqrt{3}})^2 + (z_{J_1} - z_0)^2 = r_e^2 - x_0 \end{cases}$$

В данном случае, так как мы работаем с окружностями и игнорируем ось X , получается система из двух уравнений с двумя неизвестными. Если раскрыть скобки и вычесть из первого уравнения второе, то можно выразить z через y и подставить во второе уравнение, получив квадратное уравнение.

$$z_{J_1} = a + b * y_{J_1}, \text{ где}$$

$$a = \frac{x_0^2 + y_0^2 + z_0^2 + r_f^2 - y_{F_1}^2}{2z_0}$$

$$b = -\frac{y_{F_1} - y_0}{z_0}$$

$$\frac{(y_1 - y_0)^2 + z_0^2}{z_0^2} * y_{J_1}^2 - \frac{x_0^2 + y_0^2 + z_0^2 + r_f^2 - r_e^2 - y_1^2 + 2y_1(y_1 - y_0)}{z_0} * y_{J_1} + r_f^2 = 0$$

Для решения уравнения находим дискриминант и корни (запишем в сокращенной форме через a и b , так как их можно вычислить заранее и сильно сократить запись).

$$a = \frac{x_0^2 + y_0^2 + z_0^2 + r_f^2 - y_{F_1}^2}{2z_0}$$

$$b = -\frac{y_{F_1} - y_0}{z_0}$$

$$d = -(a + by_1)^2 + r_f^2(b^2 + 1)$$

$$y_{J_1} = \frac{y_1 - ab - \sqrt{d}}{b^2 + 1}$$

После вычисления y_{J_1} и z_{J_1} необходимо произвести расчет угла поворота рычага.

$$\theta_1 = \arctan\left(\frac{z_{J_1}}{y_{F_1} - y_{J_1}}\right)$$

Для вычисления θ_2 и θ_3 необходимо дважды сделать поворот системы координат и проделать аналогичные действия, как для первого рычага.

Соответственно:

$$E_2 = \left((y_0 - \frac{e}{2\sqrt{3}}) \sin(120^\circ), (y_0 - \frac{e}{2\sqrt{3}}) \cos(120^\circ), z_0\right)$$

$$E_3 = \left((y_0 - \frac{e}{2\sqrt{3}}) \sin(-120^\circ), (y_0 - \frac{e}{2\sqrt{3}}) \cos(-120^\circ), z_0\right)$$

$$F_2 = \left(-\frac{f}{2\sqrt{3}} \sin(120^\circ), -\frac{f}{2\sqrt{3}} \cos(120^\circ), z_0\right)$$

$$F_3 = \left(-\frac{f}{2\sqrt{3}} \sin(-120^\circ), \frac{f}{2\sqrt{3}} \cos(-120^\circ), z_0\right)$$

Основным приемом данного метода является правильное расположение системы координат, которое позволяет не рассматривать X координату при нахождении первого угла, а поворот системы координат позволяет распространить это на остальные два угла. При компьютерной реализации в виде кода, данный метод удобно разбивается на две функции, одна из которых вызывает другую.

2 Моделирование робота

2.1 ZenCad

В качестве CAD программы для моделирования деталей робота и создания цифрового двойника была выбрана библиотека параметрического 3d моделирования ZenCad. Автор библиотеки вдохновлен программой OpenScad, проектирование в котором заключалось в написании скрипта, являющегося инструкцией для графического ядра, строящего модель. Без интерактивных инструментов изменения объектов. В ZenCad используется ядро граничного представления OpenCascade, что является преимуществом по сравнению с OpenScad, использующим математику полигональных сеток. Необходимость представлять каждую модель, как массив полигонов приводит к комбинаторному взрыву, при усложнении сцены, что в свою очередь заставляет разрабатывать модели с меньшим разрешением, чем их финальный вид, ради экономии вычислительных ресурсов. Особенностью граничного представления твёрдого тела заключается в описании модели, как набора поверхностей, с заданной точностью соединенных по границам и образующих замкнутый объем. То-есть окружность задается не как многоугольник с заданным количеством вершин, а именно как функция окружности, задаваемая двумя точками: центром и точкой на окружности, соответствующей 0 радиан. Подобное представление позволяет определять объем тела и его массово-инерционные характеристики, а также упрощает его разбиение на конечные элементы для инженерного анализа (так как можно легко определить, лежит ли точка внутри тела или за его пределами). Для удобства работы параметрические модели представляются в виде логического дерева построения. Фактически, каждая геометрическая операция (вытянуть, построить по сечениям и др.) – это набор алгоритмов, создающих набор корректно связанных поверхностей. При изменении компонента дерева построения все последующие элементы будут перестроены. Если при этом исчезнут элементы, к которым были привязаны последующие построения, то модель окажется некорректной. Поэтому, проектируя изделие, необходимо четко представлять иерархию дерева и возможные способы последующего изменения геометрии.

Для ускорения расчетов сцены используется библиотека ленивых вычислений evalcache для агрессивного кэширования вычислений. Это дает несколько преимуществ. Во-первых, расчеты внутри программы будут происходить только в том случае, если потребовался результат этих вычислений. Например, если объект не используется в конечной сцене, он не будет обсчитан. Во-вторых, объекты, параметры которых не были изменены будут исключены из расчетов, а данные о них загружены из кэша, где хранятся предыдущие вычисления. В том числе и одинаковые объекты сцены не будут обсчитаны несколько раз. Ради демонстрации моделей с разными значениями параметров, можно заранее их обсчитать и соот-

ветственно разные версии модели будут храниться в кэше. В теории возможно переносить папку кэша с одного компьютера на другой, если это целесообразно (очень сложный проект и маломощный компьютер). С другой стороны, моделирование в ZenCad представляет собой программирование, то-есть изменение текстового документа, для чего возможно использовать любой компьютер и текстовый редактор.

Созданный мной цифровой двойник написан на языке «Python», который является скриптовым языком. Это обозначает, что язык не имеет собственного компилятора, а программы представляют собой скрипт. Скрипт - это записанная в виде текста последовательность команд, которые будет выполнять интерпритатор Python, при условии, если он сможет обнаружить все необходимые для выполнения библиотеки. Текст моего проекта разделён на четыре логических секций:

Константы здесь размещены все константы, используемые в скрипте и их часто используемые отношения, для облегчения расчетов и экономии места.

Функции для удобства работы с деталями робота, все они создаются в виде функций, к которым происходит обращение в нужный момент. К некоторым функциям, бывают обращения внутри других функций, например, для выреза под гайку, которая используется в нескольких деталях.

Файлы STL Так как целью проекта является создание физического макета, детали которого будут напечатаны на 3д принтере, то существует специальный раздел, где собраны все функции, создающие STL файлы. STL - это широкоиспользуемый формат для обмена 3д моделями в виде готовой полигональной сетки. Данный формат не подходит для дальнейших преобразований моделей (кроме линейного масштабирования). Это удобно тем, что автор, не желающий выдавать секреты проектирования своей модели, может разместить в открытом доступе конечный результат, с которым возможно работать, только как с законченной моделью. В частности, напечатать на 3д принтере. К слову, создание проектов в ZenCad заключается в написании текста, а значит попадают под законодательство об авторском праве, что тоже может быть немаловажно для определенных задач.

Интерактивные объекты и анимация особенностью ZenCad является то, что сами по себе модели не будут присутствовать в сцене до тех пор, пока не будут превращены в интерактивные объекты. Чрезвычайно глобальное отличие от OpenScad, что можно создавать различные сцены и привязывать к ним интерактивные объекты, при этом объекты, не задействованные в создании интерактивных объектов, не будут обсчитаны совсем.

Есть несколько способов создания интерактивных объектов, я использую функцию `n = disp(m)` (сокращение от Display), которая из объекта, хранящейся в переменной `m`, создает интерактивный объект `n`. Отличие объекта от интерактивного объекта можно рассмотреть на примере создания анимации. Если координату куба представить в виде массива из 10 значений, а после создать из него интерактивный объект, то мы увидим, как в сцене будет сгенерировано 10 кубов. Для создания анимации движения, необходимо менять координату именно интерактивного тела, что делается с помощью специальных преобразований.

2.2 Ласточкин хвост (Создание произвольного полигона)

В проекте несколько раз используется крепление типа «Ласточкин хвост», для создания которого используется возможность построения многоугольника `polygon` по массиву точек `pnts`, состоящего из координат точек. В приведенной ниже функции, первым шагом происходит проверка флага `f` на True/False, что символизирует будет ли данных объект непосредственно ласточкиным хвостом или пазом под него. Это важно, так как паз должен иметь зазор, определяемый переменной `dh`. Зазор подобран экспериментально, напечатанные детали имеют достаточно шершавые поверхности, в следствие послойного наплавления. При данном зазоре одна деталь с трудом стыкуется с другой и держится силой трения. При объявлении двумерного массива `pnts`, вносятся координаты в формате `[[x1, y1], [x2, y2]]`, которые автоматически воспринимаются как 2 координаты X и Y, фигура соответственно получается плоской и в плоскости XY. Если бы стояла задача использовать все три координаты их можно было записывать как `[[x1, y1, z1], [x2, y2, z2]]`. После получения плоской фигуры (трапеции), происходит ее экструдирование с помощью функции `linear_extrude()` на вектор `vec=(0,0,20)`.

```

1      def hvost(f): # Ласточкин хвост
2          if f == 1 :
3              dh = 0.3
4          else :
5              dh = 0
6          pnts=      [[ 20 + dh , 10  + dh] ,
7                      [ 10 + dh , -10 - dh] ,
8                      [-10 - dh , -10 - dh] ,
9                      [-20 - dh , 10  + dh]]
10         o=polygon(pnts=pnts , wire=False)
11         m=linear_extrude(proto=o , vec=(0,0,20) , center=True)
12     return m

```

Пример кода 1: Создание полигона из массива точек

2.3 База (Создание моделей и булевы операции)

База представляет собой основную конструкционную часть робота, задающей конструкционную жесткость для всех остальных элементов конструкции. Основной сложностью для создания дельта робота, заключается в необходимости с большой точностью располагать рычаги под углом в 120° друг к другу. От точности углов, чрезвычайно сильно зависит возможность управления роботом в дальнейшем. К счастью, 3д печать, как и ЧПУ фрезеровка, позволяет обойти эту сложность, убирая человеческий фактор, так как расположение всех креплений и отверстий задается станком с высокой точностью. В следствие моего решения печатать все детали робота на 3д принтере, я обязан уместить их в рамках печатной поверхности своего принтера, а именно в квадрат 20x20 сантиметров. В проекте присутствует дополнительное построение (квадрат из красных линий), обозначающий размер печатного стола принтера. Так как база не умещается в данные габариты, мне пришлось сделать ее разборной. Центральная цилиндрическая часть имеет 3 паза под ласточкин хвост, в которые встают "лепестки" с креплением для двигателей, червячного редуктора, концевиков и оси рычага.

Вырез под гайку позволяет упростить процесс сборки и симпатично спрятать гайки внутри конструкции. Сам по себе, он имеет форму шестигранного цилиндра. Для создания шестигранника используется специальная функция `ngon()` с параметром `n` (количество граней) равным шести. Параметр `wire` отвечает за то, является ли двумерный объект полноценным, заполненным полигоном или только контуром. Основная сложность остается в определении радиуса. В данном случае подразумевается радиус описанной окружности многоугольника. Если измерить гайку штангель-циркулем, то получится диаметр вписанной окружности шестигульника. Так как в шестигульнике радиус описанной окружности относится к радиусу вписанной как $\sin(60^\circ)$, получаем соотношение:

$$r_{ngon} = \frac{0.5*2}{\sqrt{3}} * d_{gaiika} + dh_{zazor}$$

Итоговая величина зазора подбиралась экспериментально, чтобы гайка входила в гнездо плавно и не выпадала под собственным весом. Конкретно $dh = 0.225$. Стоит отметить, что подобная подгонка имеет смысл, когда деталь печатается шестигранным отверстием кверху, когда оно свободно от поддержек или не деформировано, как любое небольшое горизонтальное отверстие.

```
1 | def pos_gaiki(): # функция выреза под гайку
2 |     m = ngon(r=4.15,n=6,wire=False)
3 |     n = linear_extrude(proto=m, vec=(0,0,5), center=True)
4 |     return n
```

Пример кода 2: Линейная развертка (экструдирование)

Центральная часть робота помимо скрепления "лепестков" также имеет важную функцию крепления дельта-робота к его рабочему месту. В данном случае, так как стояла задача создания макета робота, была создана легкая рама из алюминиевого квадратного профиля, непосредственно стыкуемого с центральным элементом робота. В его рабочей версии, крепление придется значительно переработать.

Для нейро-сетевого процесса управлением роботом, в базе предусмотрено посадочное место для расположения камеры точно по центру.

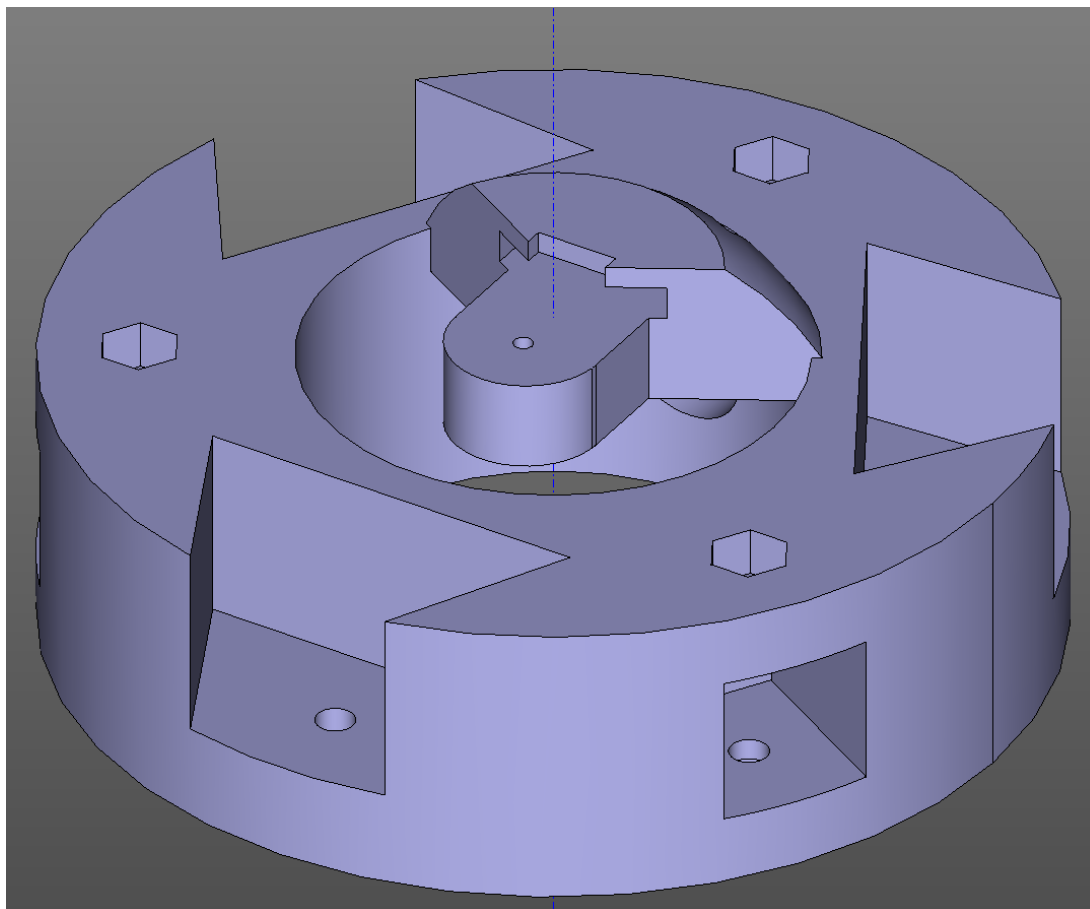


Рис. 3: Центральная деталь дельта-робота

Все крепления осуществляются винтами М4 с шляпками под внутренний шестигранник и самоконтрящимися гайками.

Лепесток отвечает за один из важнейших параметров дельта-робота, а именно за радиус базы, переменную rad . Изменение радиуса базы (расстояние от центра робота, до одной из осей вращения рычагов) меняется за счет изменения длины данной детали. В данном случае, предоставлено значение $rad = 150$ миллиметров, которое можно считать предельным значением. Минимальным значением можно считать 90 миллиметров, если не менять конфигурацию центральной части. При значениях меньше 90 мм., целесообразно печатать базу единой деталью.

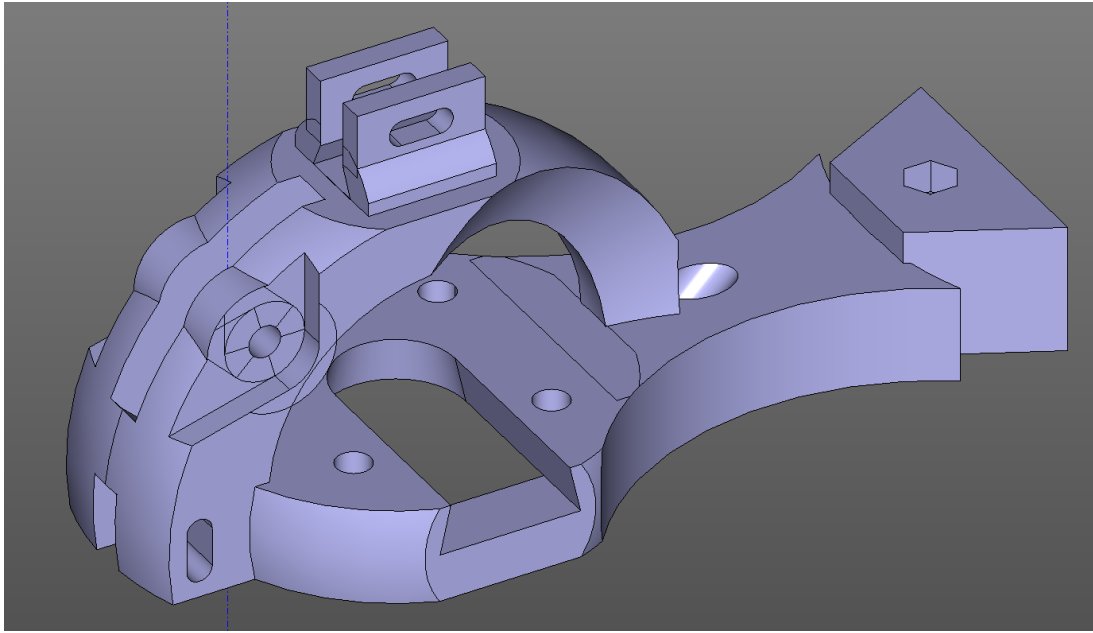


Рис. 4: Вторая деталь базы дельта-робота

Для увеличения жесткости напечатанной детали, желательно придавать ей как можно больше сложных криволинейных поверхностей. В данном случае использовано максимальное количество сферических поверхностей. Все прямоугольные поверхности проектировались по принципу пересечения сферы с кубом, или вычитания сфер из формы.

```
1 | krep = box((xz_dvig+20,xz_dvig+20,15),center=True)\
2 | ^ sphere(r=0.8*xz_dvig)
```

Пример кода 3: Булева операция пересечения

В частности главная центральная часть данной детали (крепеж шагового двигателя), это прямоугольный параллелепипед со сторонами X и Y на 20 миллиметров больше, чем габариты двигателя, и Z стороной равной 15 мм. С помощью оператора \wedge находится пересечение со сферой радиусом 80 % от величины габаритов двигателя. Таким образом вместо прямоугольных форм получают округлые, способные противостоять большим скручивающим нагрузкам. Расположенная перпендикулярно арка также представляет собой полусферу, пересеченную с кубоидом. Пространство под аркой представляет собой вычтенную сферу, так как располагающийся там глобоидный червь тоже имеет габариты сферы. В следствие этого, края арки находятся немного ниже высоты червя, и тот на свое место встает с щелчком, за счет упругости пластика. Это один из способов удержания червя, использованный тут.

У каждого лепестка есть два настраиваемых крепления для концевиков, позволяющие немного регулировать крайние положения рычага. При настройке робота крайне важно выставить рычаги в начальные позиции (в моем случае это -15°)

и для облегчения настройки, есть физическое ограничение для поворота рычага соответствующие положениям -15° и 90° . Таким образом, выставив рычаг в одно из крайних положений, есть возможность отрегулировать позицию концевика на срабатывание в нужный момент.

2.4 Глобоидная червячная передача (Параллельные вычисления)

Глобоидная червячная передача является разновидностью червячных передач, когда зона механического соприкосновения червя и колеса изогнута по вогнутой (глобоидной) траектории. Это обеспечивает большое сцепление по количеству зубьев в несколько раз (в сравнении с цилиндрическим червем) в зависимости от конфигурации количества зубьев и диаметра колеса. В промышленных механизмах данная передача используется редко из-за некоторых минусов, связанных со сложностью изготовления и настройки глобоидной передачи. Также её характерной особенностью является повышенное тепловыделение, а следовательно повышенный износ и сниженный КПД, в сравнении с цилиндрическим червём. Тем не менее, в данной конструкции глобоидная передача позволит в небольшом объеме сделать пластиковый редуктор с максимально большими зубьями у рабочих колес. Максимально возможный размер зубьев, подразумевает возможность повысить надежность пластиковой конструкции и упростить требования к печати деталей.

Двумерное шестеренчатое колесо является необходимым элементом, для выбранного мной способа моделирования червя. Идея построения заключается в том, чтобы взять некую форму заготовки (цилиндр или в моем случае полусферу) и, поворачивая ее вокруг своей оси, вычесть из нее форму шестеренчатого колеса, которая при этом сама поворачивается вокруг собственной оси. Таким образом, можно быть точно уверенным, что моделируемый червь и колесо идеально подходят друг к другу.

Поэтому, в первую очередь, предстояло выбрать форму шестеренчатого колеса. Самая простая форма для зубьев (если не рассматривать процесс изготовления, а с точки зрения воображения) это зубья сферической формы. Рассмотрим функцию построения двумерной шестерни из окружностей. Так как необходимые значения шага резьбы и количества зубьев изначально не определены, функция должна генерировать шестерню с произвольными значениями этих переменных. Данные две переменные являются одними из основных и вынесены в первый блок проекта. Называются `teeth` (количество зубов зубчатого колеса) и `step` (шаг резьбы червячной передачи).

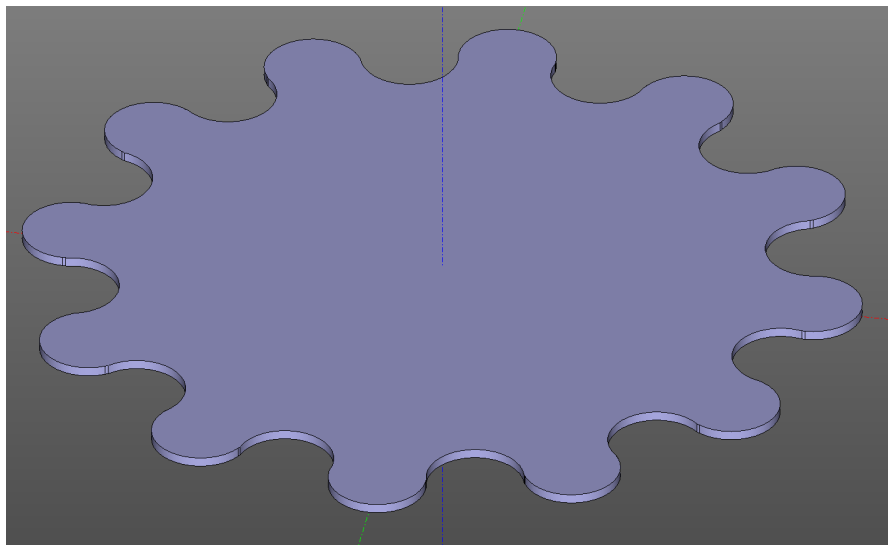


Рис. 5: Пример работы функции (добавлен объем)

```

1 def gear(teeth, step): # Функция шестерни
2     angle = pi/teeth
3     a = i*2*angle
4     rad_opis = (step/2) / math.sin(angle/2)
5     rad_vpis = (step/2) / math.tan(angle/2)
6     c=[]
7     def c1(i): # Функция зуба колеса
8         c1 = circle(step/2, angle=(1.5*pi, 0.5*pi), wire=True)\
9             .moveX(rad_opis).rotateZ(a)
10        return c1
11    def c2(i): # Функция впадины колеса
12        c2 = circle(step/2, angle=(0.5*pi, 1.5*pi), wire=True)\
13            .moveX(rad_vpis).rotateZ(a+angle)
14        return c2
15    for i in range(teeth):
16        o = sew([c1(i),\
17            segment(c1(i).endpoints()[-1], c2(i).endpoints()[-1]),\
18                c2(i) ])
19        p = segment(c2(i).endpoints()[0],\
20            c1(i+1).endpoints()[0])
21        c.append(o)
22        c.append(p)
23    return sew(c)

```

Пример кода 4: Работа с линиями объединение

Основная задача данной функции заключается в том, чтобы расставить по-

ловинки окружностей определенным образом в пространстве. Для формирования формы шестеренчатого колеса, можно представить многоугольник в вершинах которого расположены центры окружностей зубьев, а серединах сторон - центры окружностей, вырезающие впадины. Количество вершин многоугольника будет соответствовать количеству зубьев колеса (teeth), а сторона - шагу резьбы (step).

Внутри функции объявлены две отдельные функции `c1(i)` и `c2(i)` для построения *i*-той половинки окружности зуба и впадины соответственно. А внутри них располагается функция `circle()`, которая имеет 3 параметра: радиус окружности, угол (angle), с помощью которого создается не окружность целиком, а дуга, ограниченная начальным и конечным углом, и `wire` - является объект контуром или сегментом. Радиус окружностей равен половине шага резьбы. Чтобы поместить центр окружности во все вершины многоугольника, нужно один раз сместить его на радиус описанной окружности и `teeth` раз повернуть на удвоенный угол `angle`. Таким образом используется удобство полярных координат. Для впадин аналогично, но переместить нужно на радиус вписанной окружности. Оба радиуса и угол находятся с помощью тригонометрии в самом начале.

Самое сложное в данной функции - это получение из разрозненных кусочков, единый двумерный объект. Для соединения нескольких линий в одну, существует функция `sew([line1,line2])`, которая обрабатывает массивы, состоящие из линий. Соответственно, необходимо объявить массив `s=[]` и циклично добавлять в него полуокружности, чтобы потом объединить их в единое целое. К сожалению, в месте, где должны соединяться полуокружности рядом находится большое количество точек, и алгоритм функции `sew()` не способен самостоятельно решить, какие точки необходимо скреплять, поэтому ему необходимо помочь. Необходимо воспользоваться функцией `segment(point1, point2)`, которая создает отрезок, соединяющий точки `point1` и `point2`. В качестве точек нужно взять граничные точки линий, для чего существует преобразование `line.endpoints()[0]`, где 0 - это номер точки объекта `line`. Конечно, таким образом можно обратиться к любой точке, из которой состоит линия, но как обратиться к последней точке? Так как линия является массивом из точек, то задача состоит в обращении к последнему элементу массива, номера которого мы не знаем. В python мы можем это сделать, вызвав переполнение, то-есть обратившись к -1 элементу массива.

В контексте данной задачи, линия `O` представляет собой сумму зуба, впадины и отрезка, соединяющих их последние точки. Линия `P` представляет собой отрезок, соединяющий первую точку впадины с первой точкой следующего зуба. Когда цикл `teeth` раз добавит эти линии к массиву `s`, будет возможно полностью объединить массив в одну линию и вернуть ее, как результат работы функции - `return sew(s)`. Обращаю внимание, что алгоритм делает все операции за один цикл

В данном случае явно продемонстрированы сложности, связанные с тем, что ZenCad использует ядро граничного представления. В случае с OpenScad мелкие погрешности несовпадения точек в пространстве, были бы нивелированы, пред-

ставлением окружностей, как многогранников с четко определенными величинами минимальных деталей. В ZenCad даже ничтожный разрыв линии превратит замкнутую фигуру в незамкнутую, за чем необходимо очень пристально следить. Так как к незамкнутым фигурам нельзя применить те операции, которые необходимо сделать в дальнейшем.

Создание тора из вращающейся шестерни Если взять полученную выше шестерню и замкнуть её в трёхмерный тор, параллельно поворачивая её вокруг своей оси, то можно получить форму, обратную форме червя. Достаточно будет вычесть этот тор из сферы или цилиндра, чтобы получить конечную форму глобоидного червя. Задача состоит в том, чтобы разработать алгоритм построения данной фигуры и, так как её расчёт будет достаточно затратным для вычислительных мощностей, распараллелить вычисления, благо python предлагает инструменты для этого.

Идея создания фигуры заключается в том, чтобы построить выше смоделированную шестерню, сдвинуть ее от центра на радиус будущего тора. Следующую построить аналогично, но при этом повернуть по оси Z относительно центра на угол "b" и одновременно повернуть ее вокруг собственной оси на угол "a". При этом переменная "n" отвечает за детализацию будущей фигуры. Переменная "n_s" отвечает за количество оборотов, которое сделает шестерня, пройдя всю окружность по тору. Это очень важная переменная, так как от нее зависит не гладкость поверхности, а передаточное число червячной передачи. "n_s" задает количество зубьев на теле червяка, в моем проекте это три восходящие спирали. Данное решение сильно уменьшает передаточное число, что положительно сказывается на скорости робота, но при этом такой червяк плохо стопорит шестерню и в некоторых позициях возможна передача движения со стороны шестерни. То есть по мере увеличения количества зубьев у червя, передача превращается в экстравагантно выглядящую угловую зубчатую передачу. Но самое главное, что в небольшом объеме, мы можем создать зубчатые колеса с очень большими зубьями, которые просто напечатать на 3д принтере и износостойкость, которых будет велика за счет отсутствия мелких (меньше 4 мм.) деталей.

Для распараллеливания вычислений используется модуль futures. Идея заключается в том, чтобы фигуру, похожую на тор, разделить на 4 равные части и рассчитывать эти части параллельно в 4 потока. Поэтому в начале функции вставлена проверка значения переменной x, от которой зависят переменные da и db - начальный сдвиг углов a и b соответственно. Функция chast(x) в зависимости от x будет строить четвертинку итоговой фигуры и её необходимо запустить 4 раза. В данном случае, ThreadPoolExecutor() создает пул из количества потоков, равному количеству элементов в массиве x, а именно 4 потока, внутри которых будет выполняться функция chast() со своим значением переменной x. Более элегантным

решением было бы в массив *x* занести значения одного из углов сдвига, а второй считать через формулу, это позволит отказаться от части с проверкой и легко менять количество потоков, редактированием массива. В данном случае нельзя изменить количество потоков, без правки проверок внутри функции *chast()*.

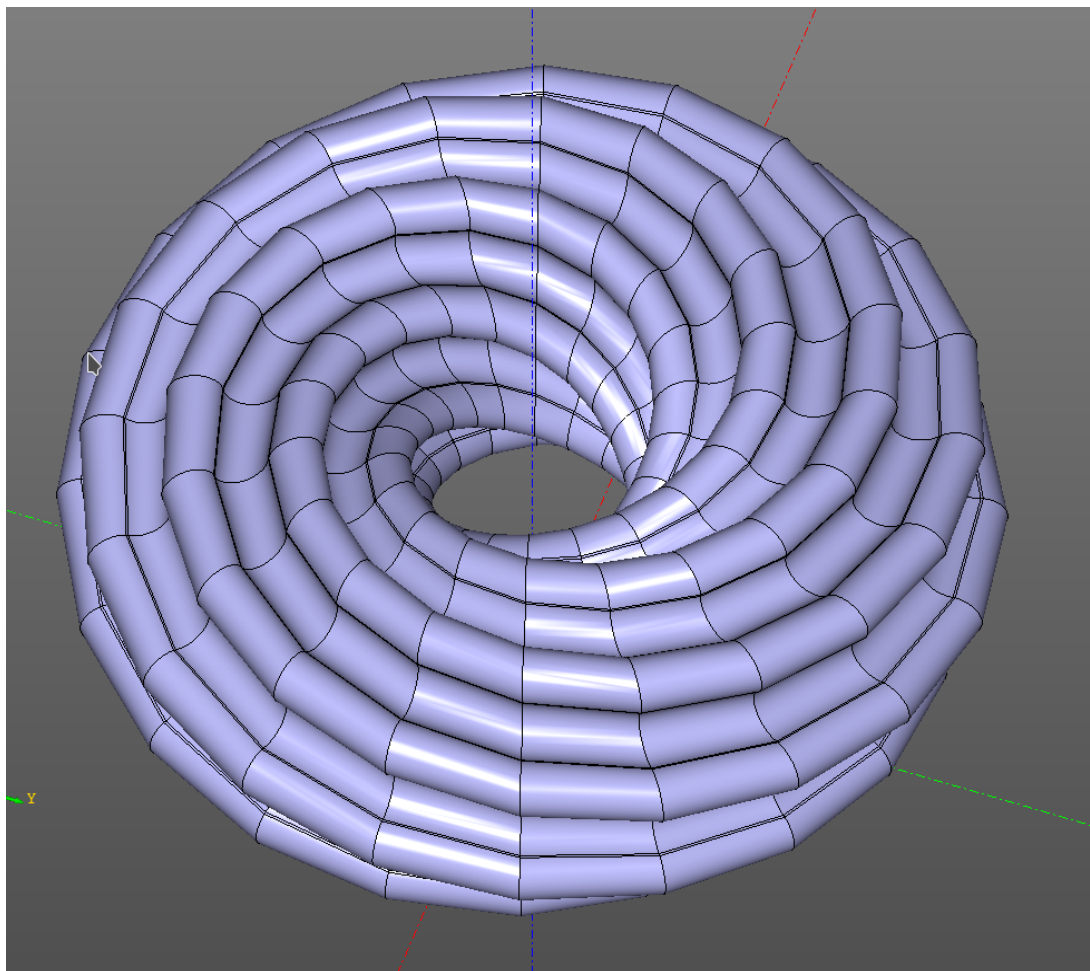


Рис. 6: Обратная форма глобоидного червя

После получения обратной формы червя, нужно взять фигуру-заготовку цилиндра или полусферы и вычесть из нее эту форму, что не является ресурсоемкой операцией и происходит мгновенно. Остается только добавить некоторые дополнительные детали червя: отверстие под вал двигателя с прямоугольным скосом, отверстие для отвертки, позволяющее прикрутить двигатель с надетой шестерней и конусный «ласточкин хвост», дополнительно фиксирующий червя в пазу. Для уменьшения вероятности того, что вал провернется в теле червя, предусмотрено место для запрессовывания специальной латунной зубчатой гайки, которая крепится к валу двигателя винтом. С червем она соприкасается зубьями и держится за счет трения.

```

1 from concurrent import futures
2 def worm (): # Червь
3     angle = pi/teeth
4     rad_opis =(step/2) / math.tan( angle/2)
5     a = (n_s*angle*0.5)/n
6     b = 0.5*pi/n
7     def chast(x): # выбор четверти в зависимости от x
8         if x==
9             da = 0
10            db = 0
11        if x==1:
12            da = a*n
13            db = pi/2
14        if x==2:
15            da = 2*a*n
16            db = pi
17        if x==3:
18            da = 3*a*n
19            db = 1.5*pi
20        g=[]
21        r=0.5*(xz_dvig+10)
22        for i in range(n+1): # цикл расстановки двумерных шестере-
нок
23            g.append(gear(teeth,step).rotateY(i*a + da)\
24                    .translate(r,0,20).rotateZ(i*b + db))
25        w = loft(g) # "обтягивание"двумерных фигур
26        return w
27    x = [i for i in range(0,4,1)] # создание и запуск 4 потоков
28    with futures.ThreadPoolExecutor() as executor:
29        worm = executor.map(chast, x)
30    o = cylinder(r=3,h=30,center=True).up(15)\ #отверстие под вал
31        ^cube([6,6,20],center=True).translate(1,0,10)
32    w = sphere(r=0.5*(xz_dvig+10),pitch=(0,pi/2))\ # полусфера
33        - union(list(worm))\ # объединенный в 1 объект тор
34        - cylinder(2,50,True).translate(0.5*otverstia,0.5*otverstia,
35    w = w - o + cone(r1=10.4, r2=9.2, h=6,center=True).down(3)\
36        - cylinder(11.2/2,11,True).down(0.5)\
37        - cylinder(1.5,12,True).rotateY(deg(90))\
38        .translate(-5,0,-5.5+2.7)

```

39 | `return w.rotateZ(deg(90)).rotateY(deg(180))`

Пример кода 5: Код функции создающей червя

Объемная шестерня для ее создания, как это ни странно, не используется функция `gear()`. Для плавного скольжения по желобам червя, зубья шестерни должны иметь форму сферы. Плоской шестеренку можно сделать только при небольшой толщине, в пару мм. при больших значениях она перестает вставляться в червя. При сферической форме зуба толщину шестеренки можно приравнять к шагу резьбы, тем самым сделав её массивной и достаточно прочной.

2.5 Вспомогательные модели (Сборочные юниты)

К вспомогательным построениям можно отнести вещи, добавленные в сцену ради визуального наполнения. К ним относятся модель платы `arduino` с дополнением `cnc shield v3`. Не смотря на то, функционально данная модель только показывает габариты и расположение отверстий крепления, при моделировании возникли сложности. `ZenCad` в его сегодняшнем исполнении, не подразумевает наличия у моделей текстуры или цвета. Функцию `color()` возможно применить только к интерактивным объектам сцены, привязав каждому объекту конкретный цвет. Как же тогда создать разноцветную модель, или как перемещать сборную модель, объединяющую детали разных цветов, как единое целое? Необходимо обратиться к сборочным юнитам.

Сборочный юнит - это объект, имеющий собственную локальную систему координат, относительно которой позиционируются связанные с этим юнитом интерактивные объекты и другие юниты. Их следует структурировать в виде дерева, отсчитывая положение относительно положения юнита-предка (`unit.parent`). Если юнит не имеет предка, его положение отсчитывается от глобальной системы координат. Юнит содержит имеет два объекта координатного преобразования - `location` и `global_location`.

`Location` - задаёт положение юнита относительно положения юнита предка. `location` может быть обновлён или непосредственно, или с помощью метода `relocate`. `globallocation` - это положение юнита относительно глобальной системы координат. `globallocation` используется при отрисовке объекта. `globallocation` строится на основе дерева `unit.location` и может быть обновлён с помощью метода `locationupdate`, `relocate` и, с помощью других операций.

Конечно, смысл объединения объектов в древовидное дерево, не создание разноцветных моделей, а удобное размещение их в пространстве сцены. Мы вынуждены каждый элемент определенного цвета создавать отдельно, но имеем возможность манипулировать ими, как группой. К такой группе могут быть применены все аффинные преобразования, в том числе и масштабирование.

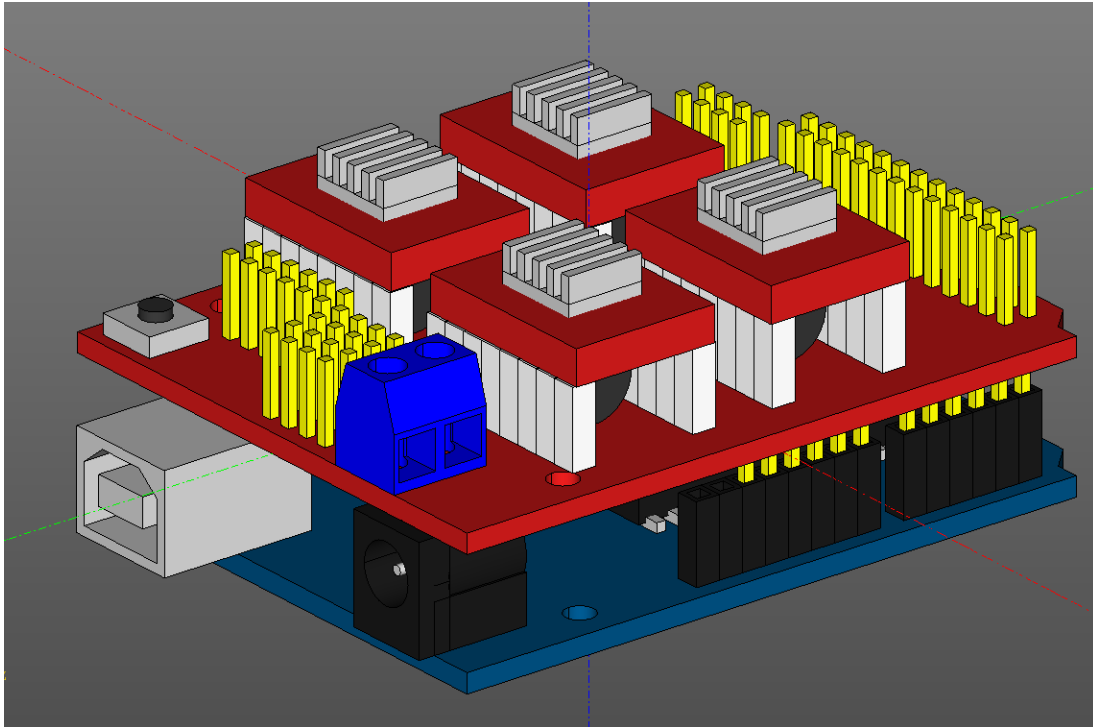


Рис. 7: Модель платы Arduino cnc shield v3

```

1 | arduino = zencad.assemble.unit()
2 | arduino.add_shape(plate(1), color(0,0.31,0.5))
3 | arduino.add_shape(usb(), color(0.8,0.8,0.8))
4 | arduino.add_shape(pit(), color(0.1,0.1,0.1))
5 | arduino.add_shape(pit2(), color(0.8,0.8,.8))
6 | arduino.add_shape(knopka(), color(0.8,0.8,.8))
7 | arduino.add_shape(knopka2(), color(0.8,0.2,0.2))
8 | arduino.add_shape(micro(), color(0.1,0.1,0.1))
9 | arduino.add_shape(micro2(), color(0.8,0.8,0.8))
10 | arduino.add_shape(raz1(), color(0.1,0.1,0.1))
11 |
12 | shild = zencad.assemble.unit()
13 | shild.add_shape(plate(0), color(0.8,0.1,0.1))
14 | shild.add_shape(shtik(), color.yellow)
15 | shild.add_shape(knopka(), color(0.8,0.8,.8))
16 | shild.add_shape(knopka2(), color(0.2,0.2,0.2))
17 | shild.add_shape(kleim(), color.blue)
18 | shild.add_shape(raz2(), color.white)
19 | shild.add_shape(kond(), color(0.2,0.2,0.2))
20 | shild.add_shape(drivers(), color(0.8,0.1,0.1))
21 | shild.add_shape(radiators(), color(0.8,0.8,.8))
22 |

```

```

23 | arduino.relocate(moveZ(-12.5))
24 | disp(shild)
25 | disp(arduino)

```

Пример кода 6: Пример сборочного юнита

В качестве примера показываю, как объединял плату arduino и плату shild в два сборных юнита, состоящих из набора функций, строящих определенные элементы, наполняющие плату. К слову, функция plate(), отрисовывающая кусок текстолита, имеет логический флаг 0 или 1, отвечающий за количество отверстий в платах, так как оно разное. Строка arduino.relocate(moveZ(-12.5) * rotateY(deg(90))) перемещает и вращает систему координат для юнитов, зависящих от предка arduino, поэтому их расположение относительно друг друга не меняется.

Таким же методом были созданы концевики, состоящие из двух частей.

2.6 Движение каретки (Создание анимации)

Последняя треть проекта оставлена под создание объектов сцены с помощью функции disp() (сокращение от display). По умолчанию display добавляет интерактивный объект в главную сцену, но есть возможность создавать различные сцены, прикреплять юниты к ним и позже отобразить командой show().

Объекты разделены на две группы: неподвижные и подвижные.

Неподвижные объекты сцены имеют конкретную позицию относительно глобальной системы координат, которая рассчитывается один раз, до следующего изменения глобальных параметров дельта-робота. На данный момент, все они зависят от размера радиуса базы (rad).

```

1 | disp(krepej().moveY(-rad), color(1, 0.4, 0))
2 | disp(krepej().moveY(-rad).rotateZ(deg(120)), color(1, 0.4, 0))
3 | disp(krepej().moveY(-rad).rotateZ(deg(-120)), color(1, 0.4, 0))

```

Пример кода 7: Растановка объектов методом полярных координат

На самом деле, формула расчета координат сложнее, но все константы я оставил внутри функции, а глобальные переменные использовал здесь. Самой главной идеей было разместить геометрический центр модели krepej в точке, где находится ось рычага. Тогда перемещение на константу rad, поместит ось рычага в необходимую точку.

Подвижные объекты сцены собраны в отдельную функцию animate(widget). Все вычисления внутри функции зависят от переменной t, которая отсчитывает время анимации. Для изменения переменной во времени используется библиотека time.


```

1 | import time
2 |
3 | nulltime = time.time()
4 |
5 | def animate(widget):
6 |     t = (20*(time.time() - nulltime))%210
7 |     if (t<105):
8 |         teta1 = t-15
9 |         teta2 = t-15
10 |        teta3 = t-15
11 |     else:
12 |         teta1 = 210-t-15
13 |         teta2 = 210-t-15
14 |         teta3 = 210-t-15

```

Пример кода 8: Работа со временной переменной

Функция `time.time()` возвращает текущее время, выраженное через количество секунд, прошедшее с 1 января 1970-го года. Чтобы не работать с этим большим числом, а начать отсчитывать с нуля вводится начальная точка отсчета `nulltime`. При каждом последующем вызове функции времени, из неё будет вычитаться начальное время. Но на самом деле, в данном случае, это ни на что не влияет и можно опустить, так как для вычисления `t` используется операция `%210` (нахождение остатка от деления на 210). Моей задачей было менять углы отклонения рычагов робота в их крайних пределах, а именно от -15° до 90° . Если считать перемещение на один градус за один такт, то движение в одну сторону будет равным $15 + 90 = 105$ тактам. Добавляем движение назад и получаем $105 * 2 = 210$ число тактов на всю анимацию. Так как время течет не останавливаясь, программа будет отрабатывать анимацию циклично до конца своей работы. Коэффициент 20 перед скобкой отвечает только за скорость анимации. Так как ядро отрисовывает объекты много чаще раза в секунду, а функция времени возвращает миллисекунды, в качестве тысячных секунды, анимация не становится прерывистой от ввода коэффициента. По сути, ядро не ограничивает нас в скорости анимации. Ради эксперимента можно попробовать заставить объект вращаться со скоростью, кратной герцовки экрана монитора и добиться стробоскопического эффекта. Ниже пример программы, где я заставляю тело вращаться со скоростью 589.95 оборотов в секунду, превратив `propeller` в мигающий, но почти неподвижный восьмелистник. При небольшом изменении частоты, кажущееся направление вращения меняет свое направление, по часовой или против часовой стрелки. Число 589.95 не кратно 60 Гц экрана, так как очевидно есть задержки, необходимые процессору для вычислений и на разных компьютерах придется подбирать свою частоту вращения.

```

1 | from zencad import *

```

```

2 import time
3
4 m=box((100,10,10),center=True)
5
6 propeler = disp(m)
7
8 def animate(widget):
9     t = (589.95*time.time())%36
10    propeler.relocate(rotateY(deg(10*t)))
11
12 show(animate=animate)

```

Пример кода 9: Стробоскопический эффект на экране монитора

Для создания анимации необходимо понимать разницу между объектами и интерактивными объектами. Можно попробовать создать объект, координата которого зависит от времени. Тогда мы увидим, как каждый такт создаются копии этого объекта, которые постепенно заполняют сцену. Потому что мы создали интерактивный объект, состоящий из размножающихся во времени объектов. Данный прием я использую для рисования траектории движения каретки, в качестве объекта используя точку (point(X,Y,Z)).

```

1 def animate(widget):
2     t = (20*(time.time() - nulltime))
3     k = 105
4     d = 120 #20*t/360
5     p = 15
6
7     teta1= k*math.fabs(math.sin(deg(t))) - p
8     teta2= k*math.fabs(math.sin(deg(t - d))) - p
9     teta3= k*math.fabs(math.sin(deg(t + d))) - p

```

Пример кода 10: Рисование максимальных траекторий дельта-робота

В качестве демонстрации движения, я придумал такой небольшой алгоритм, где углы отклонения рычагов меняются по синусоидальному закону, и имеют сдвиг по фазе относительно друг друга (d). С помощью параметров k и p задается диапазон значений [-15:90]. Функция fabs возвращает модуль от синусоиды, чтобы избавиться от отрицательных значений, которое здесь не нужны. Если сделать сдвиг по фазе равный 120°, то можно добиться максимальных отклонений: когда один рычаг будет в минимальной позиции, два других будут в максимальных и наоборот. Если сдвиг по фазе будет с разными знаками, как в примере выше, то мы увидим максимальную по радиусу круговую траекторию, на которую способен робот. Если знаки у сдвига будут одинаковыми, то мы увидим максимальное боко-

вое смещение каретки. На рисунке траектории обозначены чёрной линией. Можно отметить, что граница рабочей зоны дельта-робота имеет в горизонтальном сечении треугольную форму, что следует учитывать при расположении робота на его рабочем месте.

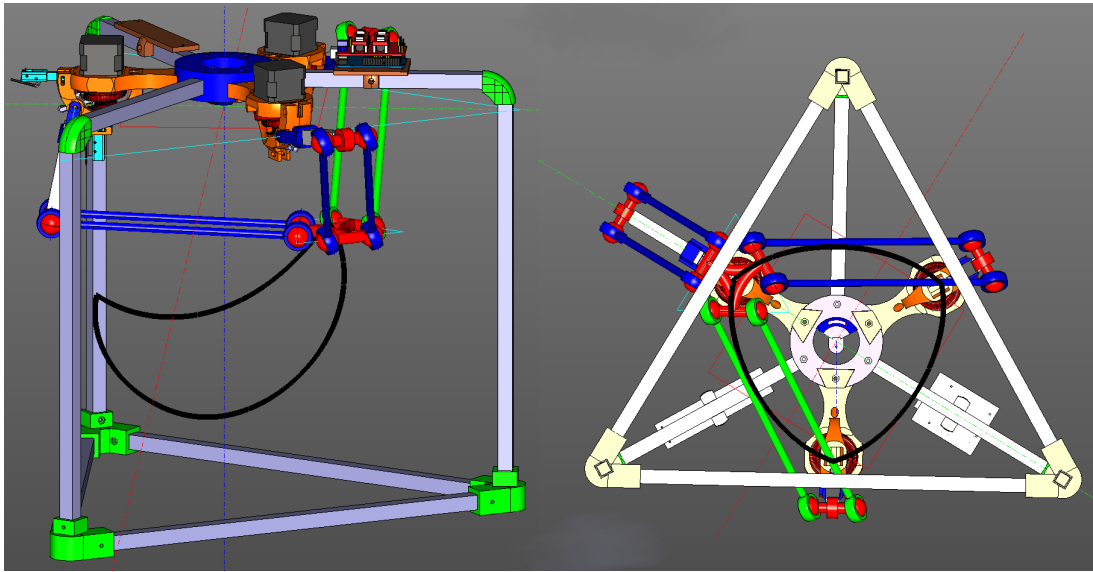


Рис. 8: Максимальные траектории движения дельта-робота

2.7 Расчет координат и углов объектов (Вращение координат)

Простые вычисления. Рассмотрим пример размещения первой группы анимированных объектов. Это именно те объекты, положение которых меняется в одной плоскости и его возможно напрямую (в одно действие) рассчитать из углов θ_i . А именно червяка, шестеренки, рычага и верхней части локтя (часть шарнира карданного крепления).

```

1 | #=====1 рука
2 | worm1.relocate(translate(0,-rad+rich_x,19.5) * rotateZ(deg(-3*teta1
3 | gear3d1.relocate(moveY(-rad) * rotateX(deg(teta1)))
4 | plecho1.relocate(moveY(-rad) * rotateX(deg(teta1)))
5 | plecho_krep1.relocate(moveY(-rad) * rotateX(deg(teta1)))
6 | lokot_verh1.relocate(moveY(-rad) * rotateX(deg(teta1)))
7 | #=====2 рука
8 | worm2.relocate(translate(1.732*0.5*(rad-rich_x),0.5*(rad-rich_x),19.
9 |     * rotateZ(deg(-3*teta2 + 120)))
10 | gear3d2.relocate(translate(1.732*0.5*rad,0.5*rad,0)\
11 |     * rotateZ(deg(120)) * rotateX(deg(teta2)))

```

```

12 | plecho2.relocate(translate(1.732*0.5*rad,0.5*rad,0)\
13 |     * rotateZ(deg(120)) * rotateX(deg(teta2)))
14 | plecho_krep2.relocate(translate(1.732*0.5*rad,0.5*rad,0)\
15 |     * rotateZ(deg(120)) * rotateX(deg(teta2)))
16 | lokot_verh2.relocate(translate(1.732*0.5*rad,0.5*rad,0)\
17 |     * rotateZ(deg(120)) * rotateX(deg(teta2)))
18 | #####3 рука
19 | worm3.relocate(translate(-1.732*0.5*(rad-rich_x),0.5*(rad-rich_x),19
20 |     * rotateZ(deg(-3*teta3 - 120)))
21 | gear3d3.relocate(translate(-1.732*0.5*rad,0.5*rad,0)\
22 |     * rotateZ(deg(-120)) * rotateX(deg(teta3)))
23 | plecho3.relocate(translate(-1.732*0.5*rad,0.5*rad,0)\
24 |     * rotateZ(deg(-120)) * rotateX(deg(teta3)))
25 | plecho_krep3.relocate(translate(-1.732*0.5*rad,0.5*rad,0)\
26 |     * rotateZ(deg(-120)) * rotateX(deg(teta3)))
27 | lokot_verh3.relocate(translate(-1.732*0.5*rad,0.5*rad,0)\
28 |     * rotateZ(deg(-120)) * rotateX(deg(teta3)))

```

Пример кода 11: Простые анимированные объекты

Если присмотреться к данному отрывку кода, то можно заметить общее единообразие в том, как происходит размещение объектов. В первой руке происходит сдвиг по Y координате на -rad (иногда встречается отступ rich_x, который зависит от форм-фактора двигателя), и поворот на угол θ_1 . У второй и третьей руки все происходит аналогично, но добавляется поворот по Z на 120° и -120° соответственно и некие коэффициенты 1.732 и 0.5. На самом деле координаты деталей второй и третьей руки рассчитываются по формуле вращения координат.

$$\begin{aligned}
 x_2 &= x_1 \cos(120^\circ) + y_1 \sin(120^\circ) \\
 y_2 &= -x_1 \sin(120^\circ) + y_1 \cos(120^\circ) \\
 x_3 &= x_1 \cos(-120^\circ) + y_1 \sin(-120^\circ) \\
 y_3 &= -x_1 \sin(-120^\circ) + y_1 \cos(-120^\circ)
 \end{aligned}$$

Как видно вторая и третья рука строятся аналогичным образом, что и первая, только через преобразование координат по формуле вращения системы координат, относительно оси Z. Но это только размещение на нужной координате, при этом объект остается повернутым, относительно своей новой системы координат. Ему требуется поворот вокруг собственной оси на 120° , для выравнивания со своей новой системой координат. Поэтому и появляется во второй и третьей руке дополнительная функция rotateZ(deg(120)). Можно отметить, что здесь заложено,

что скорость вращения червяка в 3 раза больше, скорости вращения остальных деталей.

Вычисления прямой задачи управления. Вторая группа анимированных деталей поворачиваются, следуя за кареткой, либо непосредственно присоединены к ней. Поэтому вычисление их положения, невозможны без вычисления координат каретки. Вычисляются они по формулам, приведенным в главе про кинематику.

```

1 | t = rad - 0.5*E  # радиус вписанной окружности
2 |
3 | x1 = 0
4 | y1 = -rad + e - Rf*math.cos(deg(teta1))
5 | z1 = -Rf*math.sin(deg(teta1))
6 |
7 | x2 = (-rad + e - Rf*math.cos(deg(teta2)))*math.sin(deg(120))
8 | y2 = (-rad + e - Rf*math.cos(deg(teta2)))*math.cos(deg(120))
9 | z2 = -Rf*math.sin(deg(teta2))
10 |
11 | x3 = (-rad + e - Rf*math.cos(deg(teta3)))*math.sin(deg(-120))
12 | y3 = (-rad + e - Rf*math.cos(deg(teta3)))*math.cos(deg(-120))
13 | z3 = -Rf*math.sin(deg(teta3))
14 |
15 | dnm = (y2-y1)*x3-(y3-y1)*x2
16 |
17 | w1 = y1**2 + z1**2
18 | w2 = x2**2 + y2**2 + z2**2
19 | w3 = x3**2 + y3**2 + z3**2
20 |
21 | a1 = (z2-z1)*(y3-y1) - (z3-z1)*(y2-y1)
22 | b1 = -( (w2-w1)*(y3-y1) - (w3-w1)*(y2-y1) )/2
23 |
24 | a2 = -(z2-z1)*x3 + (z3-z1)*x2
25 | b2 = ((w2-w1)*x3 - (w3-w1)*x2)/ 2
26 |
27 | a = a1**2 + a2**2 + dnm**2
28 | b = 2*(a1*b1 + a2*(b2-y1*dnm) - z1*dnm**2)
29 | c = (b2-y1*dnm)**2 + b1**2 + (dnm**2)*(z1**2 - Re**2)
30 |
31 | d = b**2 - 4*a*c
32 | Z = -0.5*(b + math.sqrt(d))/a

```

$$33 \quad X = (a1*Z + b1)/dnm$$

$$34 \quad Y = (a2*Z + b2)/dnm$$

Пример кода 12: Решение прямой задачи управления дельта-робота

В конце этих вычислений получаются координаты (X, Y, Z) , которые присваиваются каретке и определяют её конечное положение в пространстве.

Карданное соединение (векторная алгебра). Вычисления выше начинаются с расчета координат (x_i, y_i, z_i) , которые являются координатами локтей (обозначены красным цветом на рис. 8). Но локти были расставлены уже до этого другим методом. Сложность возникает с карданным крепежом (обозначен синим цветом на рис. 8), который в проекте обозначен, как `arm1p`. Где цифра обозначает номер руки робота, а «р» и «л» - «правая» и «левая». Всего их шесть штук. Задача заключается в том, чтобы по двум координатам (координате локтя и каретки) рассчитать положение в пространстве каждой из `arm`.

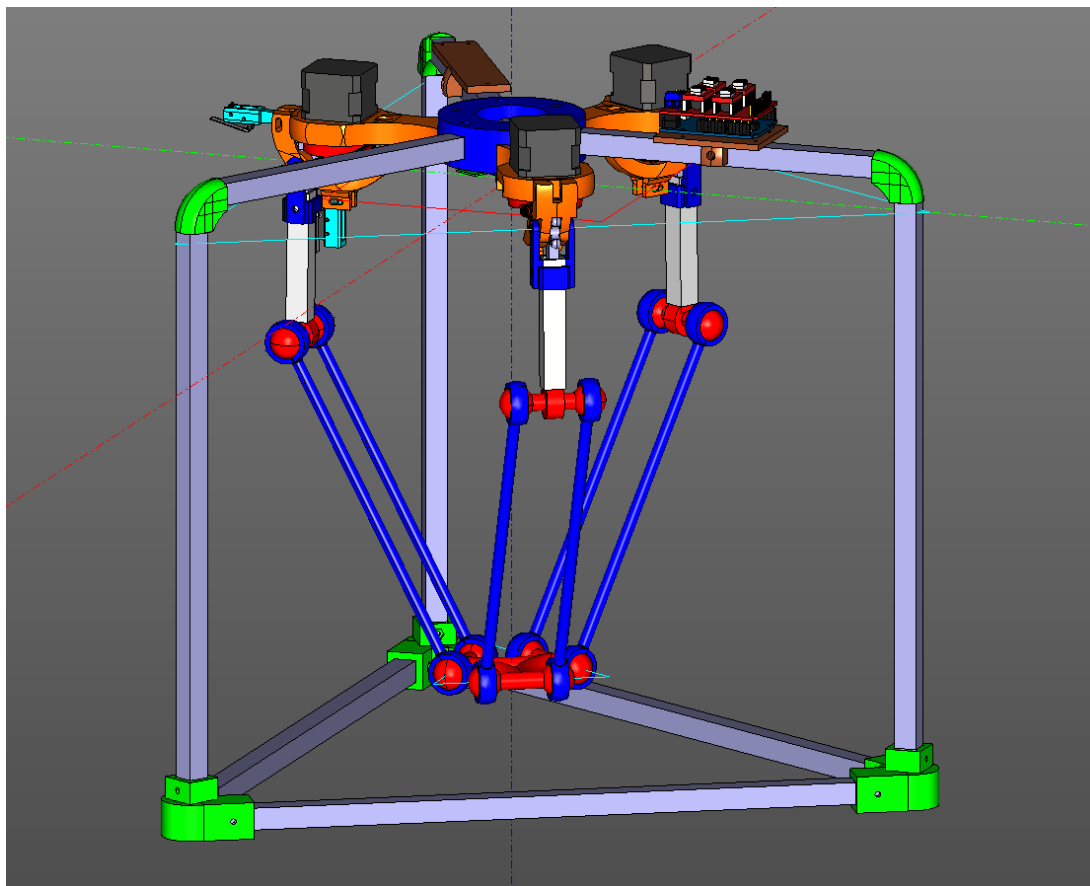


Рис. 9: Общий вид дельта-робота

`Arm` состоит из верхнего и нижнего кольца, охватывающих сферу шарнира и трубки их соединяющих. Функция строит `arm` таким образом, чтобы центр верхнего кольца был в начале координат. Поэтому первоначальное расположение будет

соответствовать координатам (x_i, y_i, z_i) , со смещением вправо или влево. Остается повернуть его в двух плоскостях таким образом, чтобы нижнее кольцо «наделось» на шарнир каретки. В случае с первой рукой координата для нижнего кольца легко рассчитывается из (X, Y, Z) . Каретка не поворачивается в пространстве, при движении все её точки рисуют взаимно параллельные линии. Поэтому координаты креплений не будут меняться, если их выразить через координаты центра каретки. Рассчитав координаты верхнего и нижнего кольца и, зная расстояние между ними, так как это константа, можно рассчитать проекции угла на плоскость ZX и ZY . Соответственно рассчитав значение проекций, можно будет найти углы поворота.

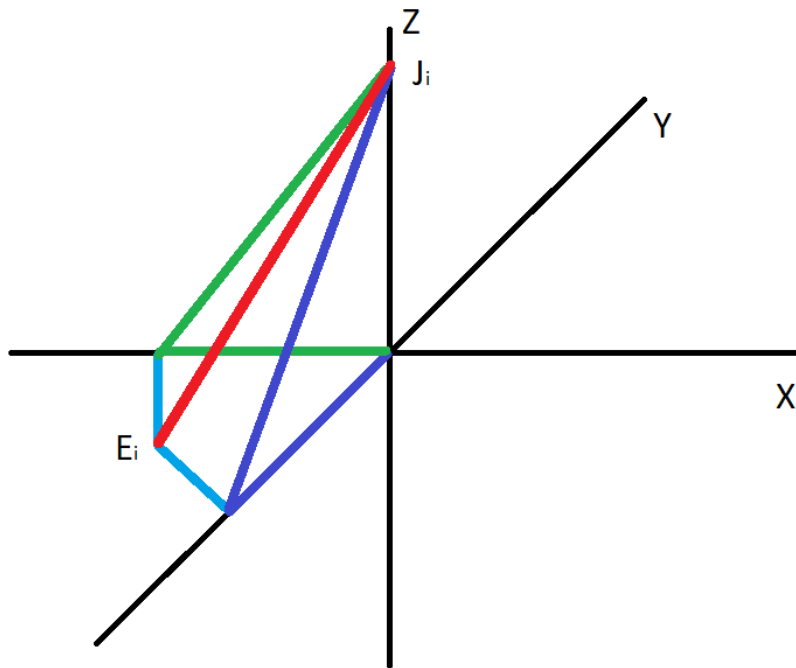


Рис. 10: Проекции

Пример для первой руки:

```

1 | arm1p.relocate(translate(x1+0.5*l_karetki,y1-0.5*E,z1)\
2 |     * rotateY(-math.asin((X-x1)/Re))\
3 |     * rotateX(math.asin((Y-y1)/Re)))
4 | arm1l.relocate(translate(x1-0.5*l_karetki,y1-0.5*E,z1)\
5 |     * rotateY(-math.asin((X-x1)/Re))\
6 |     * rotateX(math.asin((Y-y1)/Re)))

```

Пример кода 13: Тригонометрический способ ориентирования шарниров

К сожалению, данный метод перестал работать в нынешней версии ZenCad, так второй поворот будет происходить относительно системы координат, смещенной первым поворотом, а не относительно глобальной системы координат. Более того,

очередность поворотов тоже влияет на конечный результат. Это произошло потому, что правила вращения интерактивных и обычных объектов привели к единообразию. Зато появился более правильный и простой способ повернуть карданный крепеж на требуемый угол, а именно - векторный поворот. Ниже представлен пример векторного поворота для двух карданных креплений, соединяющих первый рычаг с кареткой. Необходимо рассчитать координаты, где крепление стыкуется с локтем (точка $pN = (x_l, y_l, z_l)$), и точку стыковки с кареткой ($pK = (Xk, Yk, Z)$).

```

1 | x_l = x1 + 0.5*l_karetki
2 | y_l = y1 - e
3 |
4 | Xk = X + 0.5*l_karetki
5 | Yk = Y - e
6 |
7 | pN = point(x_l, y_l, z_l)
8 | pK = point(Xk,Yk,Z)
9 | arm1p.relocate(translate(*pN)\
10 |                * short_rotate(f=(0,0,-1), t=pK-pN) )
11 |
12 |
13 | x_l = x1 - 0.5*l_karetki
14 | y_l = y1 - e
15 |
16 | Xk = X - 0.5*l_karetki
17 | Yk = Y - e
18 |
19 | pN = point(x_l, y_l, z_l)
20 | pK = point(Xk,Yk,Z)
21 | arm1l.relocate(translate(*pN)\
22 |                * short_rotate(f=(0,0,-1), t=pK-pN) )

```

Пример кода 14: Векторный поворот первого шарнира

После простого расчета координат точек, необходимо создать объект точку (`point()`). После чего переместить объект `arm1p` в точку `pN`, где находится его соединение с локтем. Тут использована ссылка на адрес переменной (`translate(*pN)`). И вторым действием, повернуть тело `arm1p` на вектор `t`, относительно вектора `f`. Вектор `f` показывает начальное положение тела `arm1p` (оно направлено вертикально вниз), а вектор `t` - необходимая позиция. Чтобы перенести вектор `t` в начало координат, необходимо из координат конца вектора, вычесть координаты начала вектора. В итоге мы получаем, что тело `arm1p` своим началом находится в точке `pN`, а своим концом смотрит на точку `pK`. По большому счету, в данном примере, мы сделали тоже самое, что и в примере выше, только тригонометрические вы-

числения углов спрятаны внутри функции `short_rotate()`. И пользователю теперь не нужно думать о тригонометрии.

В случае со вторым и третьим рычагами, расчеты аналогичные, но координаты точек `pN` и `pK` рассчитываются с применением приема вращения координат.

```

1 | x_l = x2 - 0.866*e - 0.25*l_karetki
2 | y_l = y2 + 0.5*e - 0.433*l_karetki
3 |
4 | Xk = X - 0.866*e - 0.25*l_karetki
5 | Yk = Y + 0.5*e - 0.433*l_karetki
6 |
7 | pN = point(x_l, y_l, z2)
8 | pK = point(Xk,Yk,Z)
9 | arm2p.relocate(translate(*pN)\
10 |                * short_rotate(f=(0,0,-1), t=pK-pN) )
11 |
12 | x_l = x2 - 0.866*e + 0.25*l_karetki
13 | y_l = y2 + 0.5*e + 0.433*l_karetki
14 |
15 | Xk = X - 0.866*e + 0.25*l_karetki
16 | Yk = Y + 0.5*e + 0.433*l_karetki
17 |
18 | pN = point(x_l, y_l, z2)
19 | pK = point(Xk,Yk,Z)
20 | arm2l.relocate(translate(*pN)\
21 |                * short_rotate(f=(0,0,-1), t=pK-pN) )

```

Пример кода 15: Векторный поворот второго шарнира

2.8 Создание моделей (Экспорт в STL)

ZenCad удобен тем, что позволяет массово экспортировать детали в нужные директории и делать это при каждом обновлении геометрии деталей. При процессе преобразования модели из объекта представлений в полигональную сетку необходимо задать минимальный размер сегментов сетки, который в данном случае составляет 0.01 миллиметра. Неудобство заключается в том, что на данный момент функция требует ввода полного пути от корня до имени файла. Нельзя указать текущую директорию через точку или домашнюю директорию через тильду. В системе виндовс, путь прописывается аналогично, начиная с диска и заканчивая именем файла. При переносе файл в другую систему, необходимо убедиться, что прописанные в скрипте директории существуют, иначе выполнить скрипт не получится.

```

1  def  tostl():
2      m1=krepej()
3      m2=basa()
4      m3=plecho_krep()
5      m4=ugolok_verh()
6      m5=ugolok_niz()
7      m6=shablon()
8      m7=lokot_niz()
9      m8=lokot_verh()
10     m9=krep_konch()
11     m10=krep_konch_niz()
12     m11=ploshadka(1)
13     m12=ploshadka(2)
14     m13=worm()
15     to_stl(m1, '/home/oleg/krepej.stl',0.01)
16     to_stl(m2, '/home/oleg/basa.stl',0.01)
17     to_stl(m3, '/home/oleg/plecho_krep.stl',0.01)
18     to_stl(m4, '/home/oleg/ugolok_verh.stl',0.01)
19     to_stl(m5, '/home/oleg/ugolok_niz.stl',0.01)
20     to_stl(m6, '/home/oleg/shablon.stl',0.01)
21     to_stl(m7, '/home/oleg/lokot_niz.stl',0.01)
22     to_stl(m8, '/home/oleg/lokot_verh.stl',0.01)
23     to_stl(m9, '/home/oleg/krep_konch.stl',0.01)
24     to_stl(m10, '/home/oleg/krep_konch_niz.stl',0.01)
25     to_stl(m11, '/home/oleg/ploshadka1.stl',0.01)
26     to_stl(m12, '/home/oleg/ploshadka2.stl',0.01)
27     to_stl(m13, '/home/oleg/worm.stl',0.01)
28
29  tostl()

```

Пример кода 16: Создание моделей STL

2.9 Вывод

После изготовления модели дельта-робота были выявлена недостаточная жесткость конструкции. «Лепестки», на которых расположены двигатели, имеют явный люфт в месте сочлинения с базой, который обусловлен неправильным позиционированием пазов. На данный момент паз сделан так, что эффективно препятствует движению в горизонтальной плоскости, но при этом дает вертикальные колебания. «Ласточкин хвост» требуется перевернуть на 90°, чтобы это исправить. Изучив движения дельта-робота, я пришел к выводу, что величина радиуса базы

должна быть значительно больше. Не в пределах от 90 мм. до 150 мм., а начинаться от 150 мм. и доходить до 300 мм. При таких плечах, делать детали целиком из пластика не имеет смысла. В новом видении дизайна робота, база должна иметь металлический скелет, а пластик будет играть роль сухожилий, связывающих все прочие детали. На стоимость робота это не должно повлиять, но упростит и ускорит процесс печати.

3 Программирование Arduino

3.1 Драйвера двигателей

Для управления шаговыми двигателями используют платы-драйвера, в моем случае это самые распространенные A4988. Их функция заключается в формировании псевдосинусоидальных токов в обмотках шагового двигателя, заставляя его делать шаги или микрошаги. Благодаря подаче сигналов на контакты MS1, MS2, MS3 можно выставить дробление шага от $\frac{1}{2}$ до $\frac{1}{32}$. Дробление шага позволяет увеличить в разы точность позиционирования, путем увеличения количества шагов на оборот, но от этого страдает скорость вращения и момент. В данном проекте, скорость и момент важнее точности, поэтому функция дробления шага не была задействована.

Управление драйвером происходит с помощью трех контактов. Первый, отвечает за включение (enable), без подачи напряжения на этот контакт, драйвер будет игнорировать команды на движение. Наличие сигнала на контакте dir определяет направление последующих шагов. Каждый сигнал на контакте step является командой драйверу сделать шаг двигателем.

Питание на драйвер приходит в обход платы Arduino, в моем проекте используется 12-ти вольтовый адаптер питания. Если отключить обмотки двигателя, то можно настроить ток драйвера, вращая отверткой потенциометр на плате. Для разных драйверов токи считаются по-своему, в случае с A4988 она выглядит так:

$$I_{raschet} = \frac{I_{nominal_dvig}}{2.5} = \frac{1.7}{2.5} = 0.68A$$

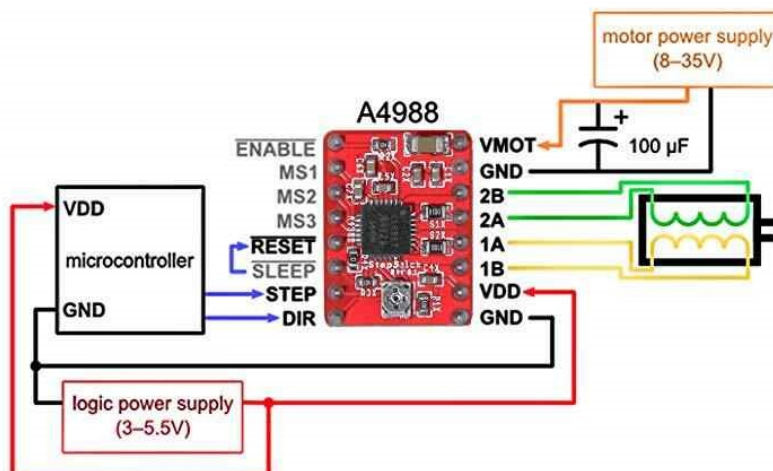


Рис. 11: Схема контактов драйвера шагового двигателя

Выставленное значение тока влияет на момент, с которым работает двигатель, на его шумность и силу нагрева. При завышении токов можно увеличить момент,

но нагреть двигатель до 90° , что может размягчить пластик, который не держит температуры выше 80° . Занижение токов приведет к понижению шума, вибраций и температуры, но двигатель может начать пропускать шаги из-за недостаточного момента.

3.2 Распиновка Arduino CNC Shield

Arduino CNC Shield продается как готовое решение для самодельных трехосевых станков. Плата создана специально под открытую прошивку GRBL для Arduino. Для взаимодействия с этой прошивкой энтузиасты написали несколько программ для различного использования CNC станков (фрезеровка, графировка или рисование). Общение с платой происходит с помощью команд gcode, отправляемых по com-порту. Поддерживаются функции плавного набора скорости шаговых двигателей, плавное движение по окружностям и спиралям и другие возможности.

Тем не менее специфика движения дельта-робота заключается в том, что все 3 шаговых двигателя должны работать одновременно. В то время, как в линейных станках одновременно работают только две оси и только в предварительно заложенных функциях, например, по рисованию окружностей. С помощью gcode можно управлять углами поворотов рычагов, через координаты x, y, z. Но нужно понимать, что внутри программы будут одновременно переменные x, y, z, которые обозначают координату в пространстве и x, y, z, которые являются углами.

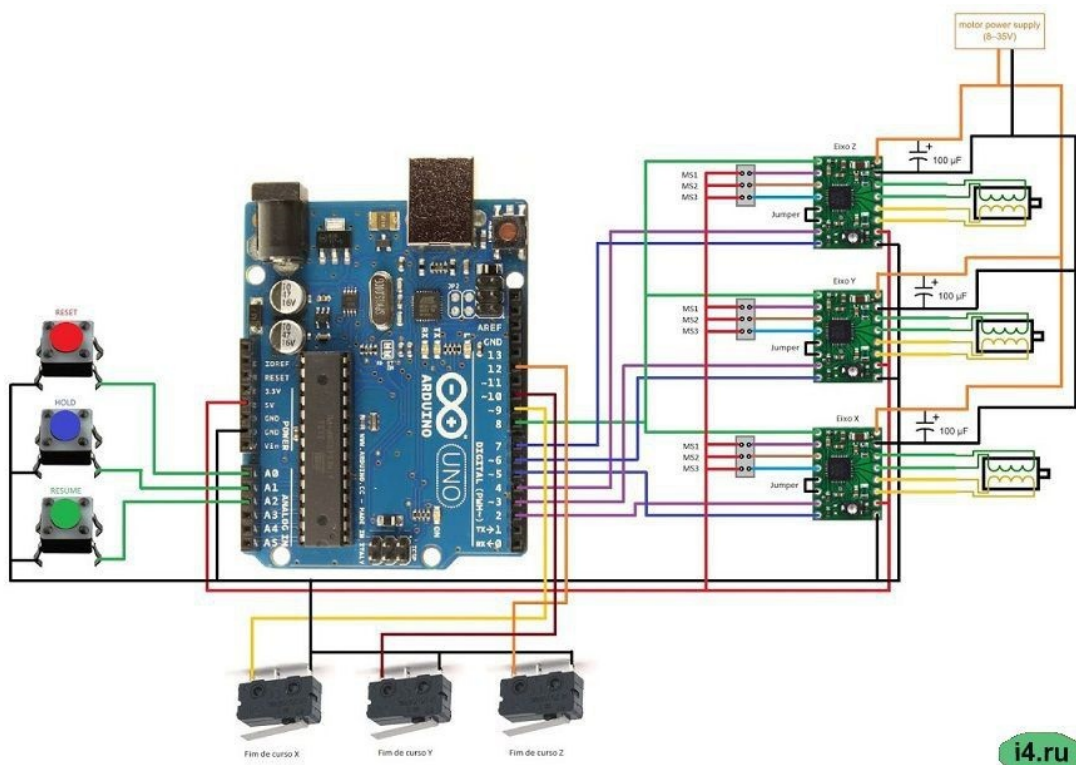


Рис. 12: Схема распиновки Arduino CNC Shield

В качестве тренировки и для набора практического навыка, решил самостоятельно написать скетч. Первым делом необходимо объявить рабочие пины Arduino:

```
1 int step1 = 2;
2 int step2 = 3;
3 int step3 = 4;
4
5 int dir1 = 5;
6 int dir2 = 6;
7 int dir3 = 7;
8
9 int pwr = 8;
10
11 int kon1 = 9;
12 int kon2 = 10;
13 int kon3 = 11;
14
15 void setup() {
16     Serial.begin(115200);
17     pinMode(step1, OUTPUT);
18     pinMode(step2, OUTPUT);
19     pinMode(step3, OUTPUT);
20     pinMode(dir1, OUTPUT);
21     pinMode(dir2, OUTPUT);
22     pinMode(dir3, OUTPUT);
23     pinMode(pwr, OUTPUT);
24     pinMode(kon1, INPUT);
25     pinMode(kon2, INPUT);
26     pinMode(kon3, INPUT);
27 }
```

Пример кода 17: Объявление пинов Arduino

Я создаю переменные, обозначающие номер порта, чтобы иметь возможность в случае необходимости менять порт, не переписывая код скетча целиком. Порты с 2 по 8 назначаю выходами и с помощью них буду отправлять управляющие сигналы на драйвера. Порты с 9 по 11 назначаются входами, в случае если в процессе выполнения движения рычаг нажмет на концевик, то на один из этих портов придет сигнал. К сожалению, оба концевика: минимальное положение и максимальное положение, у данной паты выведены на 1 пин. Поэтому определить, какой именно концевик сработал можно только по направлению движения двигателя.

3.3 Основная функция

Под основной функцией в скетче подразумевается управление Arduino с помощью команд, принимаемых через com-порт. Так как ком порт ограничен тем, что мы можем отправлять только восьмибитные сообщения, представляющие собой число от 0 до 256, конвертируемое в символ ASCII кодировки, то и команды представляют собой 1 символ. Для передачи числа в формате Int, представляющее собой 4 байта, необходимо со стороны компьютера разбить переменную Int на массив из 4 символов float, и передовать их по очереди. Со стороны Arduino существует функция Serial.parseInt, которая собирает 4 символа float, обратно в переменную типа Int. На данный момент, я ограничился двумя командами: возврат каретки в исходную позицию и перемещение каретки в новую координату. Начальная позиция в данном случае это углы θ_i равные -15° . Углы необходимо перевести в шаги двигателей и работать с ними.

$$n_{shagov} = \frac{\theta_{max} - \theta_{min}}{360^\circ} * N_{dvig} * K_{reductor} = \frac{90^\circ + 15^\circ}{360^\circ} * 200 * 6 = 350$$

Количество шагов от минимального до максимального положения является отношением полного угла к полной окружности, домноженное на количество шагов на оборот шагового двигателя и на коэффициент передачи редуктора. Получается, что для прохождения рычага до крайнего положения необходимо сделать 350 шагов, что больше 256, поэтому мы не можем использовать 8 битные числа для передачи изменения координаты, а вынуждены использовать переменные типа Int.

Для того, чтобы узнать на какие углы необходимо переместить рычаги необходимо дважды решить обратную задачу управления. Первый раз мы рассчитываем углы θ_i для настоящего положения в пространстве, а второй раз для последующего положения. И только вычитая результат первого решения из второго, мы получим значение изменения углов. В скетче ардуино, я завожу 9 переменных, отвечающих за хранение состояния углов: текущие значения θ_i , новые значения, полученные по com-порту, и Δ_i , разница между ними.

Плата Arduino в бесконечном цикле ожидает сигнала по com-порту. Если будет получена буква H, что обозначает home, то будет вызвана функция move_home(). Важно, что именно перед вызовом функции движения, я отключаю порт pwr, тем самым разрешая работу драйверам. После выполнения функции движения, возвращаем драйверам запрет на движение и посылаем в com-порт букву Y, подтверждение для компьютера, что функция выполнена адекватно

Если получена буква C, что обозначает change position, Arduino по очереди запрашивает значение координат, в которые необходимо перенести каретку. Происходит расчет Δ_i , снимается запрет на работу драйверов и выполняется функция движения каретки.

Подсчет текущего значения углов θ_i будет происходить внутри функций движения.

```

1 void loop() {
2     if (Serial.available() > 0) {
3
4         val = Serial.read();
5         Serial.write(val);
6
7         if (val == 'H') {
8             digitalWrite(pwr,LOW);
9             move_home();
10            digitalWrite(pwr,HIGH);
11            Serial.print('Y');}
12
13        if (val == 'C') {
14            Serial.print('X');
15            NEW_teta1 = Serial.parseInt();
16            Serial.print('Y');
17            NEW_teta2 = Serial.parseInt();
18            Serial.print('Z');
19            NEW_teta3 = Serial.parseInt();
20
21            delta1 = NEW_teta1 - teta1;
22            delta2 = NEW_teta2 - teta2;
23            delta3 = NEW_teta3 - teta3;
24
25            digitalWrite(pwr,LOW);
26            change_location();
27            digitalWrite(pwr,HIGH);
28
29            Serial.print('Y');
30
31        }
32    }
33 }

```

Пример кода 18: Основная функция скетча

3.4 Функция возврата каретки домой

Функция возврата каретки домой устроена по принципу, движения двигателя до тех пор, пока не появится сигнал на соответствующем порту, к котрому подклю-чен концевик. Цикл while выполняется до тех пор пока все концевики не будут ак-

тивированы. В условии использовано логическое ИЛИ и утверждение, что концевики в неактивированном состоянии. Пока хоть один из них будет неактивирован, цикл будет выполняться. Внутри цикла происходит дополнительная проверка на концевики для каждого двигателя, перед выполнением функции движения. Это делается потому, что скорее всего рычаги придут в крайние положения не синхронно.

После выполнения цикла, все рычаги должны оказаться в начальной позиции, соответственно можно обнулить глобальные переменные текущих значений θ_i .

```

1 int move_home() {
2     while( digitalRead(kon1)==LOW || //неправильный перенос строки
3           digitalRead(kon2)==LOW ||
4           digitalRead(kon3)==LOW) {
5         if( digitalRead(kon1)==LOW) { move_teta1( -1); }
6         if( digitalRead(kon2)==LOW) { move_teta2( -1); }
7         if( digitalRead(kon3)==LOW) { move_teta3( -1); }
8     }
9     teta1 = 0;
10    teta2 = 0;
11    teta3 = 0;
12 }

```

Пример кода 19: Функция возврата каретки домой

Функции изменения угла рычага однотипны, создают два сигнала на нужные выходы, с некоторой задержкой. В зависимости от необходимого направления движения, задается положительное или отрицательное k . В зависимости от знака, на пин dir либо подается, либо не подается напряжение. После подачи сигнала на направление, необходимо дать задержку, в данном случае это t миллисекунд. После подачи сигнала step, делается вторая задержка для совершения шага. Величины задержек зависят от скорости срабатывания платы микроконтроллера. Для драйвера A4988 задержки очень большие, двигатель не может начать движение, если задать их меньше 500 микросекунд. Это связано с резким спадом момента на роторе, так как управляющий ток не успевает нарастать за такой промежуток времени. Если уменьшать задержки уже вращающегося двигателя, то можно плавно опустить их до 250 микросекунд, но при любом толчке происходит пропуск шагов и двигатель останавливается. Так как мне важна стабильная работа двигателей, без пропуска шагов и с хорошим моментом, я решил использовать задержки в 1 миллисекунду. В своем предыдущем проекте, когда использовал трехканальный драйвер SMD-303 совсем другой ценовой категории, то использовал задержки в 50-60 микросекунд. Этот опыт оказался вредным, так как выставляя низкие задержки, я потратил много времени, считая что двигатели не могут повернуть редуктор, так как клинит шестерни. Я повышал задержки до 200 микросекунд, но это не дава-

ло результата, я пересобирал конструкцию, печатал новые шестеренки и тратил время и силы впустую.

```
1 int move_tetal(int k){
2     if (k>0) {
3         digitalWrite(dir1 ,HIGH);
4         delay(t);
5         digitalWrite(step1 ,HIGH);
6         delay(t);
7         tetal++; }
8     if (k<0) {
9         digitalWrite(dir1 ,LOW);
10        delay(t);
11        digitalWrite(step1 ,HIGH);
12        delay(t);
13        tetal--; }
14 }
```

Пример кода 20: Функция управляющая шаговым двигателем

3.5 Функция изменения координаты

Логика данной функции заключается в поочередном совершении шагов двигателями до тех пор, пока разница в координатах Δ_i не станет равной нулю. Цикл while выполняется тогда, когда есть отличная от нуля разница в координатах. Используется логическое ИЛИ. Внутри цикла происходит повторная проверка, что разница не равно нулю и одновременно не задействованны концевики. После этого, в зависимости от знака разницы, вызывается функция движения в определенную сторону, а разница уменьшается по модулю на единицу. Если разница по одной из координат станет равной нулю раньше остальных или будет задействован концевик, этот двигатель прекратит движение и будет ждать остальных. В случае, если рычаг активировал концевик до того, как разница стала нулевой, то на этот случай происходит обнуление разницы, чтобы цикл не превращался в бесконечный.

```
1 int change_location(){
2     while(delta1 != 0||delta2 != 0||delta3 != 0){
3         if( delta1 !=0 && digitalRead(kon1)==LOW) {
4             if( delta1 > 0 ) {move_tetal(1); delta1 --;}
5             else {move_tetal(-1); delta1 ++;}}
6         else{delta1 = 0;}
7         if( delta2 !=0 && digitalRead(kon2)==LOW) {
8             if( delta2 > 0 ) {move_tetal(1); delta2 --;}}
```

```

9         else {move_teta2(-1); delta2++;}}
10     else{delta2 = 0;}
11     if( delta3 !=0 && digitalRead(kon3)==LOW) {
12         if( delta3 > 0 ) {move_teta3(1); delta3--;}
13         else {move_teta3(-1); delta3++;}}
14     else{delta3 = 0;}
15 }
16 }

```

Пример кода 21: Функция изменения координаты

Слабое место данной функции в том, что двигатели поочередно делают шаги. Один шаг занимает 2 миллисекунды, а это значит, что пока один двигатель двигается, два других будут простаивать в течение 4 миллисекунд. Экспериментируя с данным скетчем, в какой-то момент я придумал, каким образом надо изменить функцию, чтобы двигатели двигались действительно одновременно. Изначально, мне не удавалось победить то обстоятельство, что у каретки есть шесть степеней свободы, что подразумевает необходимость написания шести разных функций, но решение нашлось.

В новой функции нет вызова других функции управления двигателями, так как это оказалось ненужным. Цикл `while` как и в прошлый раз выполняется до тех пор, пока хоть одна из переменных $Delts_i$ отлична от нуля. Внутри цикл делится на две половины. В первой половине, в зависимости от знака разницы посылаются сигналы `dir`, отвечающие за направление движения. Если разница равна нулю, сигнал будет послан, но к ошибке это не приведет. После этой операции происходит необходимая задержка. Во второй половине цикла происходит проверка на отличие разницы от нуля и срабатывание концевика. Если разница отлична от нуля и концевик не сработал, то можно отправлять сигнал на шаг. После чего остается только изменить глобальное значение угла и разницы на единицу. Если сработал концевик, но разница при этом не равна нулю, она приравнивается к нулю, чтобы цикл `while` не становился бесконечным.

Таким образом мне удалось реализовать одновременное функционирование двигателей, в независимости от направления движения и разнице количества шагов. Более того, так как двигатели работают одновременно и имеют общую задержку, ее можно очень просто менять во времени, тем самым разгоняя двигатели. В моем случае, драйвера очень плохо реагируют на разгон, поэтому я меняю задержку с 1000 микросекунд до 500 микросекунд в течение 22 шагов, по квадратичному закону. Возможно, дальнейшие эксперименты позволят выжать из драйверов большие показатели, возможно стоит завязать токи, но это требует дальнейших испытаний. И важно придумать способ отлавливать пропущенные шаги, так как двигатель иногда может пропускать шаги без остановок, но отследить это без обратной связи невозможно.

```

1  int hard_change_location(){
2      int k = 0;
3      while(delta1 != 0 || delta2 != 0 || delta3 != 0){
4
5          if(delta1 > 0) {digitalWrite(dir1 ,HIGH);}
6              else {digitalWrite(dir1 ,LOW);}
7          if(delta2 > 0) {digitalWrite(dir2 ,HIGH);}
8              else {digitalWrite(dir2 ,LOW);}
9          if(delta3 > 0) {digitalWrite(dir3 ,HIGH);}
10             else {digitalWrite(dir3 ,LOW);}
11
12         delayMicroseconds(1000 - k*k);
13
14         if(delta1 !=0 && digitalRead(kon1)==LOW) {
15             digitalWrite(step1 ,HIGH);
16             if(delta1 > 0) {delta1--; teta1++;}
17                 else{delta1++; teta1--;}
18         else {delta1=0;}
19
20         if(delta2 !=0 && digitalRead(kon2)==LOW) {
21             digitalWrite(step2 ,HIGH);
22             if(delta2 > 0) {delta2--; teta2++;}
23                 else{delta2++; teta2--;}
24         else {delta2=0;}
25
26         if(delta3 !=0 && digitalRead(kon3)==LOW) {
27             digitalWrite(step3 ,HIGH);
28             if(delta3 > 0) {delta3--; teta3++;}
29                 else{delta3++; teta3--;}
30         else {delta3=0;}
31
32         delayMicroseconds(1000 - k*k);
33
34         if (k < 22) {k++;}
35     }
36 }

```

Пример кода 22: Усложненная функция с плавным разгоном

3.6 Вывод

Данным скетчем реализован минимальный функционал, необходимый для управления роботом. Arduino умеет возвращать каретку в начальное положение, отсчитывать координаты, и перемещаться на заданные через com-порт величины углов. Важно, что благодаря реализованной функции, двигатели совершают шаги одновременно, без простоя. Есть пространство для улучшения механизма разгона двигателей, возможно, получится реализовать дополнительно торможение по квадратичному закону. Ощущается упор в возможности драйвера a4988, возможно стоит рассмотреть другие варианты драйверов.

4 Технико-экономическое обоснование

4.1 Описание проекта

4.1.1 Резюме

Бизнес план посвящен разработке и вывод на рынок сверхдешевого дельта-робота с оригинальным управлением. Стоимость самого робота в 50 тыс. руб., программное обеспечение к нему 300 тыс. руб.

Для фирмы требуется два специалиста с начальным уровнем зарплаты в 105 тыс. руб.

Начальное капиталовложение не менее 600 тыс. руб.

Себестоимость одного робота может варьироваться, но принята за 7137 рублей. При продаже за год 36-и роботов и 8 комплектов ПО, чистая прибыль составит 1583 тыс. руб.

4.1.2 Описание продукции

Готовым продуктом выступает параллельный робот, а конкретно легкий дельта-робот с грузоподъемностью не более одного килограмма. Данная машина востребована в широком спектре производственных задач, связанных с манипулированием материалами, сортировкой продукции или отбраковки. В виду своей конструкции может быть монтирован непосредственно над линией конвейера, занимает мало пространства.

Пример названия продукта:

Дельта-робот модель подвесная

Характеристики:

Количество осей: 3

Грузоподъемность: 1 кг.

Радиус действия: 300 мм.

Повторяемость: 2 мм.

Потребляемая мощность 200 Вт.

Вес: 4 кг.

Целью моего проекта было создание параметрической модели робота, с помощью которой можно относительно просто подобрать необходимые размеры будущего робота под нужды заказчика. Так как расчет рабочей области данного типа роботов является не тривиальной задачей, наглядная 3д модель, интерактивно меняющие свои параметры, такие как: радиус базы, форм-фактор шаговых двигателей, длины рычагов и шарниры, размеры каретки - является большим подспорьем. В сцену можно легко добавить объекты имитирующие будущее рабочее место робота, а анимация покажет, какие траектории он сможет совершать. Габариты и параметра робота можно и нужно менять до оптимальных для конкретного заказчика, после чего только подходить к процессу создания самого продукта. Так как рабочая зона ограничена в радиусе выполняемой операции, персонал производственной линии может безопасно работать в непосредственной близости от робота.

Дельта-робот спроектирован для печати большинства своих деталей на 3д принтере, что позволяет сильно удешевить себестоимость изделия буквально во всем. Пластик не обладает высокой жесткостью, поэтому робот не обладает большой точностью позиционирования, что не мешает использовать его в задачах, переноса небольших объектов. Зато робот чрезвычайно прост в изготовлении, все его детали легко заменяемы, масштабируемы в любых пропорциях, для создания замены требуется только время.

Потенциальными покупателями могут выступать производители пищевой продукции, в операциях дозирования теста на жаровой поверхности, выдавливании кремов определенным рисунком или фасовке готовой продукции. Так как робот

состоит большей частью из нейтрального пластика, без смазочных материалов, он способен работать непосредственно с пищей. Максимальная дешевизна работа может помочь его массовому внедрению в центры переработки мусор, где скромные технические показатели будут невелированы массовостью. По той же причине, робот можно использовать в качестве учебного пособия, поставляя его в школьные классы, молодежные кружки. Робот может служить умным освещением, следя за работой человека и освещая рабочее место в условиях, когда человек не может отвлекаться.

4.1.3 Анализ рынка сбыта

4.1.4 Анализ конкурентов

4.2 План маркетинга

Интересующиеся данными технологиями могут быть завлечены с помощью демонстрации 3д моделей или видеороликов работы готовых моделей. Для этого следует заводить тематические каналы на видеохостингах, с наглядным описанием того, как это работает и как это можно использовать. Обязательно продемонстрировать работу с заказчиком, так как необходимо доказать, что пластиковые роботы способны совершать полезную работу и окупаться. Основным рекламным ходом планируется использовать открытость продукта, рекламой будет служить возможность заказчику самостоятельно производить новых роботов. И конечно большую степень гибкости и возможность самостоятельного апгрейда.

Рекламировать роботов необходимо с сильным упором на общую "экологичность" пластмассовых механизмов, состоящих из пластика PetG - основного пластика для производства бутылок. Участвующие в сортировке мусора роботы, могут быть частично или полностью состоять из отсортированных бутылок или другого вторичного материала. Рекламные ролики лучше всего распространять в тематических группах в социальных сетях, где обитают люди заинтересованные в самоделках, экологии и технологиях 3д печати. Например, групп: "Ветряки своими руками" "Зеленая энергетика" "Все о 3д печати" "Собираю ЧПУ станок" и подобные.

Также в рекламных чрезвычайно важно попасть на тематическую выставку:

- VendExpo и WRS5-2021 (15-я международная выставка вендинговых технологий и систем самообслуживания).

- RosBuild 2021 (Международная специализированная выставка строительных, отделочных материалов и технологий в рамках «Российской строительной недели-2021»).

- ЭЛЕКТРО-2021 (30-я международная выставка. Электрооборудование. Светотехника. Автоматизация зданий и сооружений).

В качестве сервисного ремонта следует предлагать бесплатную замену любых напечатанных частей, если поломка произошла в процессе эксплуатации, так как

себестоимость таких частей примерно равняется стоимости пластика.

4.2.1 Анализ рынка

Прямые конкуренты на рынке не представлены, так как идея создания роботов из пластика витает в воздухе, но не доведена до ума. Конкурировать необходимо со стандартными роботами, которые представляют собой устройства другого класса, с точностью позиционирования в 0.02 мм., огромным запасом по надежности и прочности. Но такие роботы стоят хороших денег.



Рис. 13: Примеры конкурентов на рынке

Робот «OEM ODM» китайского происхождения, стоит 744 тыс. рублей, но без представительства в России. Робот «Fanuc» японского происхождения, но собирается в России. Стоимость зависит от заказчика, мне удалось выяснить, что начинается она от 600 тыс. руб. Оба робота имеют грузоподъемность в килограмм, высокую точность позиционирования, и массивную базу. У «Fanuc» база весит 17 кг., позволяя ему избегать вибраций, возникающих при работе. Данные роботы предназначены для чистых производств, маловероятно увидеть их на сортировке мусора. Ниша сверхдешёвых дельта-роботов на данный момент свободна, что связано со сложностью написания программного обеспечения для управления параллельной машиной.

4.2.2 Ценовая политика

Основной статьей заработка следует считать разработку программного продукта для решения конкретной задачи, поставленной заказчиком. Так как свобод-

ное ПО для управления данным типом робота отсутствует в принципе, программы придется писать в любом случае. Особенно в начале, когда не будет еще большого опыта и наработанной базы проектов. Поэтому за программный продукт логично брать больше, чем стоимость самого робота, более того, возможно, передать электронные модели заказчику.

4.2.3 Сбытовая политика и мероприятия

Показатели	Квартал				Всего
	I	II	III	IV	
Разработка ПО					
Ожидаемы объем продаж, ед.	1	2	2	3	8
Цена с НДС, т.р.	300	300	300	300	
Выручка с НДС, т.р.	300	600	600	900	2400
Нетто-выручка (без НДС), т.р.	250	500	500	750	2000
Сумма НДС, т.р.	50	100	100	150	400
Дельта-робот					
Ожидаемы объем продаж, ед.	3	6	9	18	36
Цена с НДС, т.р.	50	50	50	50	
Выручка с НДС, т.р.	150	300	450	900	1800
Нетто-выручка (без НДС), т.р.	125	250	375	750	1500
Сумма НДС, т.р.	25	50	75	150	300

Таблица 1: План продаж

4.3 План производства

Группы комплектующих, из которых будет состоять система:

- 1) Комплектующие которые печатаются на 3д принтере: основные конструкционные детали, редуктор, шарниры, крепление, каретка и части рабочего органа.
- 2) Электроника, включающая в себя arduino cnc shield, драйвера двигателей A4988, шаговые двигатели, orange Pi, orange Pi camera, импульсный адаптер питания, концевики.
- 3) Силовые элементы, в качестве которых выступает квадратная алюминиевая труба 15x15 мм. и круглая алюминиевая труба 8 мм. В зависимости от требований заказчика, диаметры труб можно варьировать.
- 4) Прочие расходные материалы: подшипники, винты, самоконтрящиеся гайки, провода, стяжки.

Основной процесс изготовления заключен в печати деталей базы, последующей механической обработке (снятие поддержек, ошкуривание и подобное) и самого

процесса сборки. Для печати выбран пластик PetG, как самый распространенный пластик, имеющий неплохие прочностные характеристики, не разлагающийся под действием ультрафиолета и работающего при температурах до 80°C. Конечно, возможен переход на промышленные пластики, показатели которых на совершенно ином уровне. Но это приведет к необходимости изменить геометрию деталей, сделать их более грацильными, так как во многих местах прочность будет излишняя.

Модель stl	Время печати	Штук	Расход г.	Общее время	Общий расход
basa	12h 58m	1	132.77	12h 58m	132.77
krepej	11h 10m	3	96.68	1d 9h 30m	290.04
worm	3h 35m	3	31.74	10h 45m	95.22
ploshadka1	3h 1m	1	24.47	3h 1m	24.47
ploshadka2	2h 53m	1	24.21	2h 53m	24.21
krep_konch_niz	31m	3	3.92	1h 33m	11.76
krep_konch	30m	3	3.97	1h 30m	11.91
lokot_verh	2h 18m	3	24.18	6h 54m	72.54
lokot_niz	37m	12	6.68	7h 24m	80.16
plecho_krep	1h 26m	3	11.56	4h 18m	34.68
Итого:				3d 12h 46m	777.76

Таблица 2: Расход времени и пластика на печать деталей

Время, приведенное в таблице 1, отображает минимальное время требуемое для печати, без учета времени на разогрев, постановку на печать следующей детали и возможных прерываний печати, связанных с застреванием филамента, плохой адгезией или сдвигом по слоям. Для достижения приемлемых временных рамок, требуется использовать параллельную печать, минимум на 2 принтерах. При получении крупного заказа, выполнить его в разумные сроки будет возможно только при создании фермы из принтеров.

Под производства необходимо два помещения: офисное для проектирования и мастерская для сборки и печати. В офисе необходимо два компьютера:

- для менеджера по продажам и закупкам продукции
- рабочее место проектировщика с CAD-приложением, слайсером для 3d печати.

В мастерской необходимы:

- место электрика, оборудованное паяльными принадлежностями, набором коробок и стеллажей для комплектующих.
- верстак с рабочими инструментами, где можно будет скручивать соединения, обрабатывать детали, проводить сборку.
- место для размещения 3d принтеров.

Наименование	Штук	Цена за шт. руб.	Стоимость руб.
Arduino Uno	1	300	300
Arduino CNC Shield	1	250	250
driver A4988	3	100	300
Адаптер питания 500 Вт.	1	1250	1250
Orange Pi с камерой	1	1400	1400
Двигатели Nema 17	3	475	1425
Концевой переключатель	6	10	60
Подшипник 5x8x2.5	6	32	192
Труба квадратная 15x15 мм. 2м.	1	200	200
Труба круглая 8 мм. 1м	2	80	160
Винты и прочий крепеж	1	300	300
Филамент PetG кг.	1	1300	1300
Итого:			7137

Таблица 3: Стоимость расходников

Минимальная цена аренды промышленного помещения в 50 м.² в Санкт-Петербурге стоит около 40 тыс. рублей в месяц. 480 тыс. рублей в год.

Наименование	Штук	Цена тыс. руб.	Стоимость тыс. руб.
Автоматизированное рабочее место	2	80	160
3д принтер	2	50	100
Рабочее место электрика	1	100	100
Верстак с инструментом	1	100	100
Итого:			420

Таблица 4: Стоимость оборудования

Основным процессом производства будет проектирование деталей, для реализации пожеланий заказчика и печать деталей. С двумя принтерами изготовление комплекта для одного робота должно укладываться в рабочее время 5-6 календарных дней. Разброс неизбежен, так как время зависит от размеров итоговых деталей. Для реализации большого заказа необходимо докупать дополнительные принтеры, либо отдавать печать на аутсорс в ателье 3д печати, что может рассматриваться, как очень выгодный вариант.

По моему мнению, необходимый персонал может состоять из двух человек, каждый из которых берет на себя сразу по две роли. Конечно, роли поделены достаточно условно, каждый должен быть готов подменить другого и понимать работу товарища.

Наименование	Ставка	Зарплата тыщ. руб.	Всего тыщ. руб.
Инженер проектировщик	0.5	80	40
Оператор 3д принтера	0.5	50	25
Электрик-сборщик	0.5	50	25
Менеджер по продажам	0.5	50	25
Итого зарплата в месяц:			105

Таблица 5: Первоначальный уровень зарплат

4.4 Финансовый план

Показатели	Источник информации	Квартал				Всего
		I	II	III	IV	
1. Выручка-нетто (без учета НДС) от реализации	План продаж	450	900	1050	1800	4200
2. Переменные производственные затраты						
2.1 Переменные материальные затраты	План материально-производственных затрат	21.411	42.822	64.233	128.466	256.932
2.2 Переменные затраты на оплату труда	План затрат на оплату труда производственного персонала	315	315	315	315	1260
2.3 Переменные общепроизводственные затраты	План общепроизводственных затрат	540	120	120	120	480
3. Валовая прибыль		428.589	857.178	985.767	1671.534	4943.068
4. Переменные управленческие и коммерческие затраты	План управленческих и коммерческих затрат	0	0	0	0	0
5. Маржинальная прибыль						

5.1 Разработка ПО		250	500	500	750	2000
5.2 Дельта-робот		103.589	257.178	385.767	771.534	1543.063
6.1 Постоянные общепроизводственные затраты	План общепроизводственных затрат	75	150	175	300	700
6.2 Постоянные управленческие и коммерческие затраты	План управленческих и коммерческих затрат	0	0	0	0	0
7. Прибыль от продаж		375	750	875	1500	3500
8. Прибыль до налогообложения		450	900	1050	1800	4200
9. Налог на прибыль		75	150	175	300	700
10. Чистая (нераспределенная) прибыль		-626.411	422.178	550.767	1236.534	1583.068

Таблица 6: Финансовый план

Расчет показателя NPV.

$$NPV = \frac{-206.411 + 422.178 + 550.767 + 1236.534}{1.1} - 420 - 480 = 820,97$$

Так как NPV получился положительным, значит финансирование проекта является целесообразным.

ЗАКЛЮЧЕНИЕ

После изготовления модели дельта-робота были выявлена недостаточная жесткость конструкции. «Лепестки», на которых расположены двигатели, имеют явный люфт в месте сочленения с базой, который обусловлен неправильным позиционированием пазов. На данный момент паз сделан так, что эффективно препятствует движению в горизонтальной плоскости, но при этом дает вертикальные колебания. «Ласточкин хвост» требуется перевернуть на 90° , чтобы это исправить. Изучив движения дельта-робота, я пришел к выводу, что величина радиуса базы должна быть значительно больше. Не в пределах от 90 мм. до 150 мм., а начинаться от 150 мм. и доходить до 300 мм. При таких плечах, делать детали целиком из пластика не имеет смысла. В новом видении дизайна робота, база должна иметь металлический скелет, а пластик будет играть роль сухожилий, связывающих все прочие детали. На стоимость робота это не должно повлиять, но упростит и ускорит процесс печати.

Написанный скетч реализует минимальный функционал, необходимый для управления роботом. Arduino умеет возвращать каретку в начальное положение, отсчитывать координаты, и перемещаться на заданные через com-порт величины углов. Важно, что благодаря реализованной функции, двигатели совершают шаги одновременно, без простоя. Есть пространство для улучшения механизма разгона двигателей, возможно, получится реализовать дополнительно торможение по квадратичному закону. Ощущается упор в возможности драйвера a4988, возможно стоит рассмотреть другие варианты драйверов.

СПИСОК ИСТОЧНИКОВ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. какая-то статья