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 к. т. н. _____ С. Е. Абрамкин

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 управление 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 управление 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

speak from my heart

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Меня зовут дундук

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 поиском найти переменную, обозначенную одним символом, а также это неправильно, с точки зрения читаемости кода.

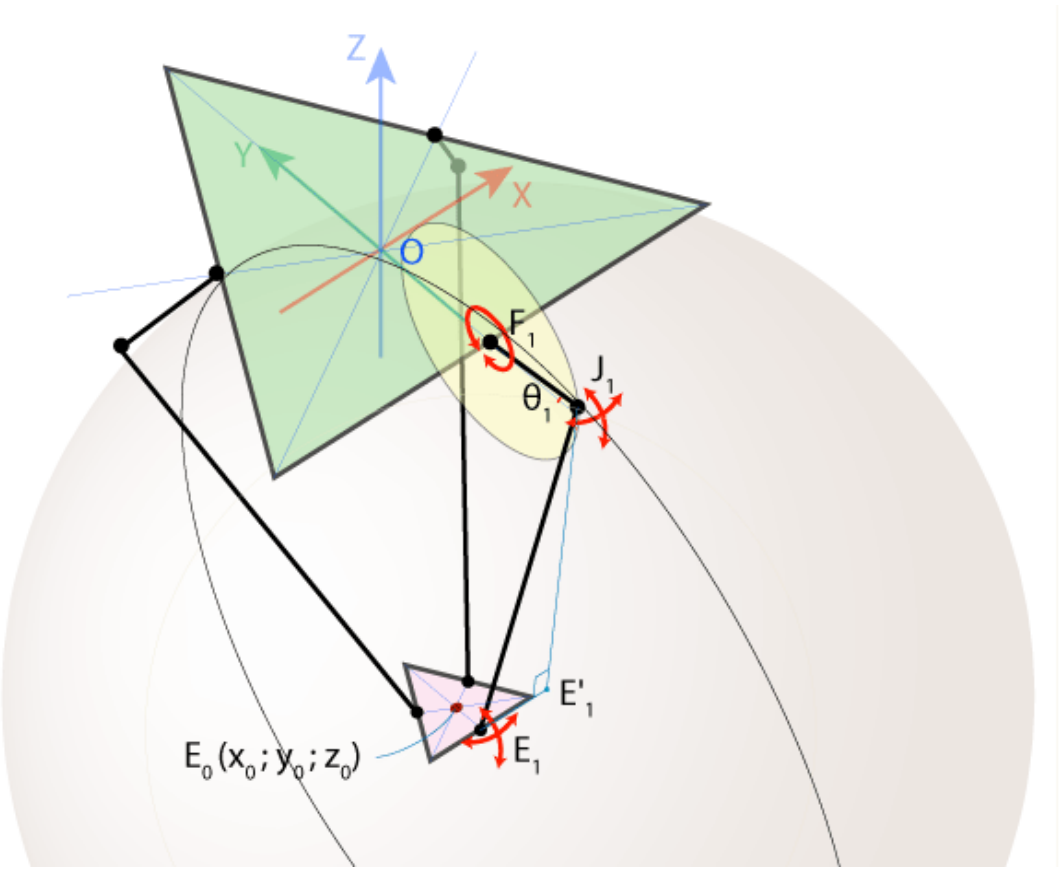

Рис. 1: Схематическое представление Дельта-робота

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

нечное 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. Второй 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 закреплен рабочий орган. Габариты обозначаются, как и в случае с базой, равносторонним треугольником, длина стороны которого обозначается буквой e. Координаты точек крепления карданных шарниров к каретки 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 достаточна проста. Так как рычаги, соединенные 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 могут вращаться шарниры, при данных

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

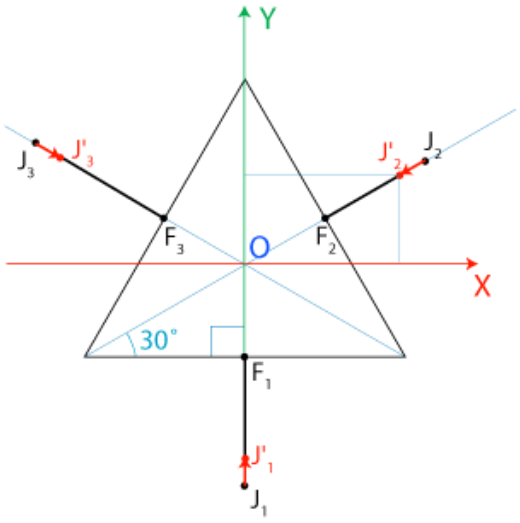

Рис. 2: Схема расчета координат рычагов

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.

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

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

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

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

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

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

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:

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

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

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

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

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.

$$x = a_1 z + b_1 \qquad y = a_2 z + b_2$$

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

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

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

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:

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

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.

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.

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

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 длинна проекции шарнира находится по теореме Пифагора:

$$E'_1J_1 = \sqrt{(E_1J_1)^2 - (E_1E'_1)^2}$$

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

$$E'_1J_1 = \sqrt{r_e^2 - x_0^2}$$

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'_1J_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 решения системы уравнений двух окружностей.

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

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.

$$\frac{x_0^2+y_0^2+z_0^2+r_e^2+r_f^2-y_{F_1}^2}{2z_0} y^2 - \frac{y_{F_1}-y_0}{z_0} y = 0$$

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. Объекты, параметры которых не были изменены будут исключены из расчетов и загружены из кэша. В том числе и одинаковые объекты сцены не будут обсчитываться несколько раз. Ради демонстрации моделей с разными значениями параметров, можно заранее их обсчитать и соответственно 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 измене-

ние текстового документа, для чего возможно использовать любой компьютер и текстовый редактор.

Созданный мной цифровой двойник представляет собой программу на языке «Python», текст которой разделён на три 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.

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. Есть несколько способов создания интерактивных объектов, я использую функцию n = disp(m), которая из объекта, хранящейся в переменной m, создает интерактивный объект n. В случае с анимацией, если координату куба представить в виде массива из 10 значений, то в сцене будет генерироваться 10 кубов. Для создания анимации, необходимо менять координату именно интерактивного тела, что делается с помощью специальных функций.

2.2 База

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. 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.

Ласточкин хвост для создания которого используется возможность построения многоугольника polygon по массиву pnts, состоящего из координат точек. Первым делом 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, я вносил координаты в формате [[1,2],[3,4]] которые автоматически воспринимаются как 2 координаты Х и У, фигура соответственно получается плоской и в плоскости ХУ. Если бы стояла задача использовать все три координаты их можно было записывать как [[1,2,3],[4,5,6]]. После получения плоской фигуры (трапеции), происходит ее экструдирование с помощью функции linear_extrude() на вектор vec=(0,0,20).

```
def hvost(f): # Ласточкин хвост
    if f == 1 :
        dh = 0.3
    else:
        dh = 0
    pnts=      [[ 20 + dh , 10  + dh],
                [ 10 + dh , -10 - dh],
                [-10 - dh , -10 - dh],
                [-20 - dh , 10  + dh]]
    o=polygon(pnts=pnts,wire=False)
    m=linear_extrude(proto=o,vec=(0,0,20),center=True)
return m
```

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 шестиугольника. Так как в шестиугольнике радус описанной окружности относится к радиусу вписанной как $sin(60°)$, получаем соотношение:

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

Итоговая величина зазора подбиралась экспериментально, чтобы гайка входила в гнездо плавно и не выпадала под собственным весом. Конкретно $dh = 0.225$

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

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.

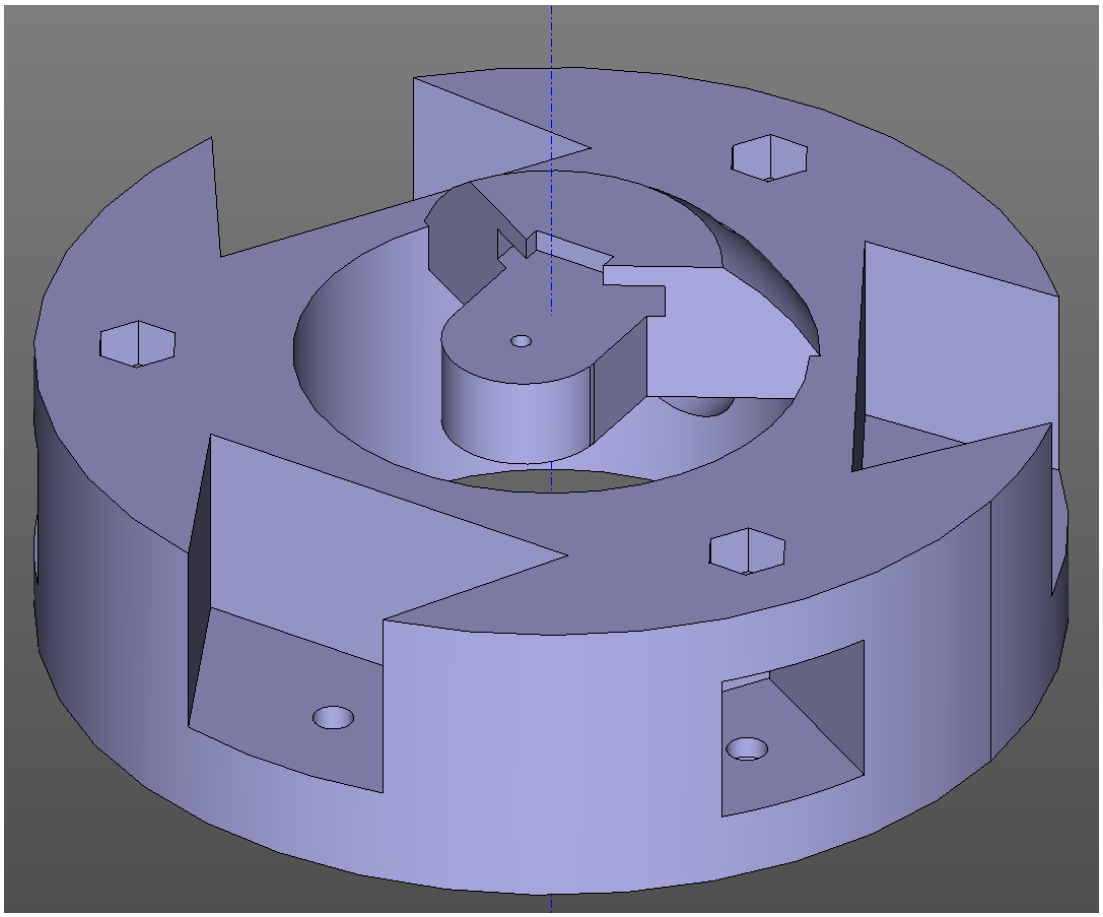

Рис. 3: Центральная деталь дельта-робота

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 длинны данной детали. В данном случае, предоставлено значение rad = 150 милиметров, 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.

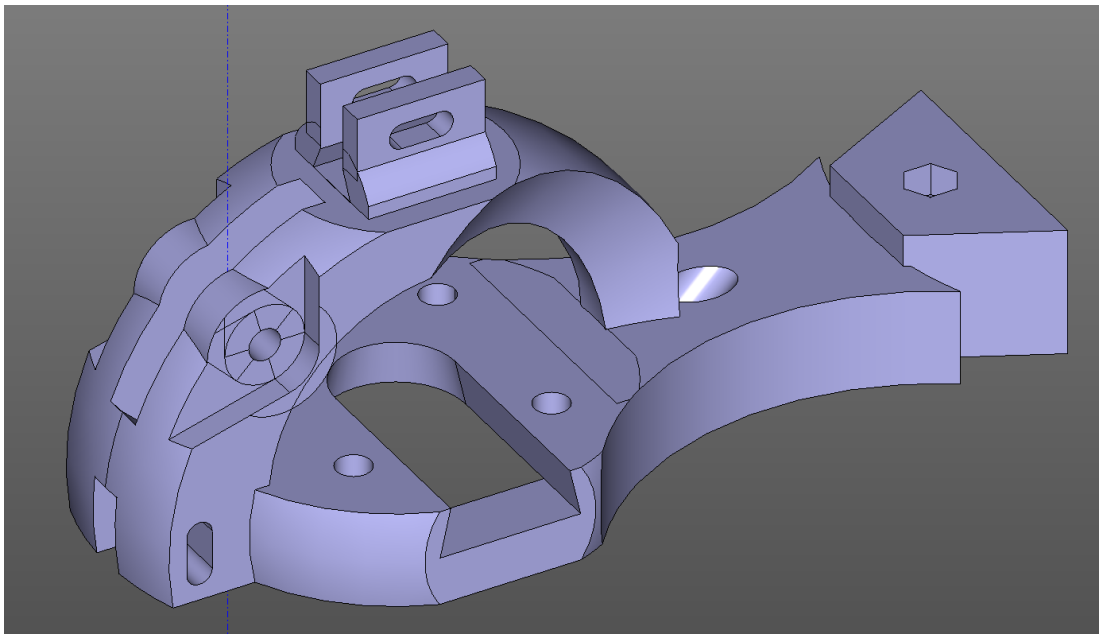

Рис. 4: Вторая деталь базы дельта-робота

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.

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

В частности главная центральная часть данной детали (крепеж шагового двигателя), это прямоугольный параллелепипед со сторонами X и У 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 получаются 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.

2.3 Глобоидная червячная передача

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 сравнении с обычной передачей в несколько раз 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 сильно сэкономить пространство и расходуемый пластик.

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).

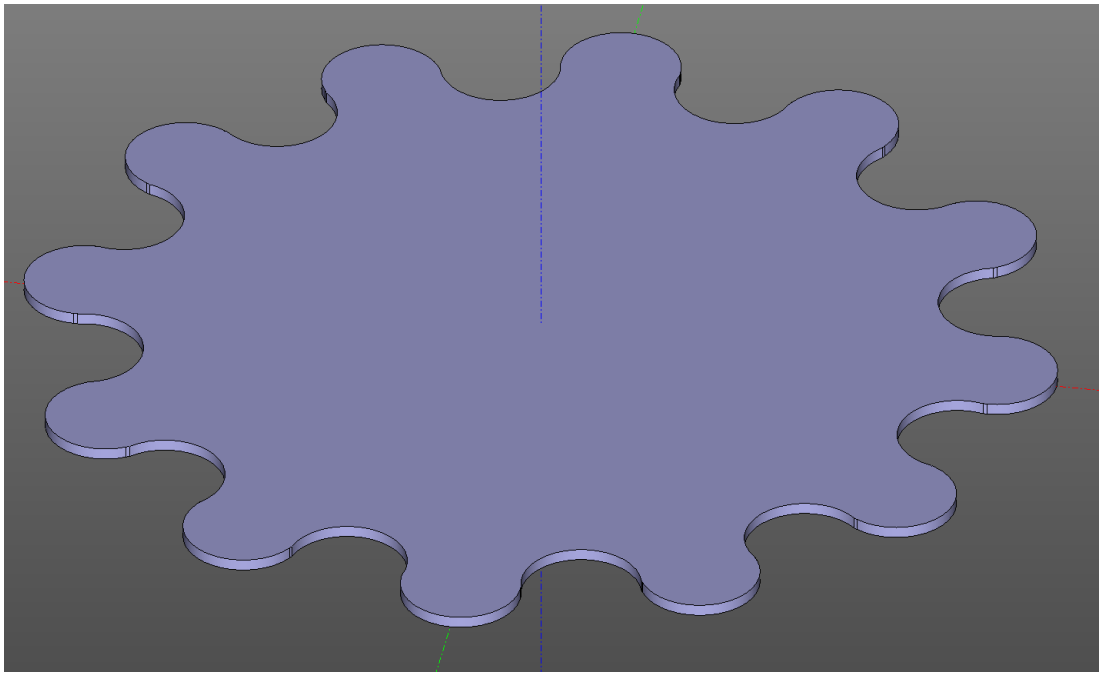

Рис. 5: Пример работы функции (добавлен объем)

```
def gear(teeth,step): # Функция шестерни
    angle = pi/teeth
    rad_opis = (step/2) / math.sin(angle/2)
    rad_vpis = (step/2) / math.tan(angle/2)
    c=[]
    def c1(i): # Функция зуба колеса
        a  = i*2*angle
        c1 = circle(step/2,angle=(1.5*pi,0.5*pi), wire=True)\
            .moveX(rad_opis).rotateZ(a)
        return c1
    def c2(i): # Функция впадины колеса
        a  = i*2*angle
        c2 = circle(step/2,angle=(0.5*pi,1.5*pi), wire=True)\
             .moveX(rad_vpis).rotateZ(a+angle)
        return c2
    for i in range(teeth):
        o = sew([c1(i),\
                 segment(c1(i).endpoints()[-1],c2(i).endpoints()[-1]),\
                 c2(i)  ])
        p = segment(c2(i).endpoints()[0], c1(i+1).endpoints()[0])
        c.append(o)
        c.append(p)
    return sew(c)
```

Основная задача данной функции заключается в том, чтобы расставить половинки 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 цент 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 c=[] 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 c, будет возможно полностью объединить массив в одну линию и вернуть её, как результат работы функции - return sew(c). 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, представлением 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.

Для распаралелливания 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.

```
from concurrent import futures
def worm (): # Червь
    angle = pi/teeth
    rad_opis =(step/2) / math.tan(angle/2)
    a = (n_s*angle*0.5)/n
    b = 0.5*pi/n
    def chast(x):
        if x==
                da = 0
                db = 0
        if x==1:
                da = a*n
                db = pi/2
        if x==2:
                da = 2*a*n
                db = pi
        if x==3:
                da = 3*a*n
                db = 1.5*pi
        g=[]
        r=0.5*(xz_dvig+10)
        for i in range(n+1):
```

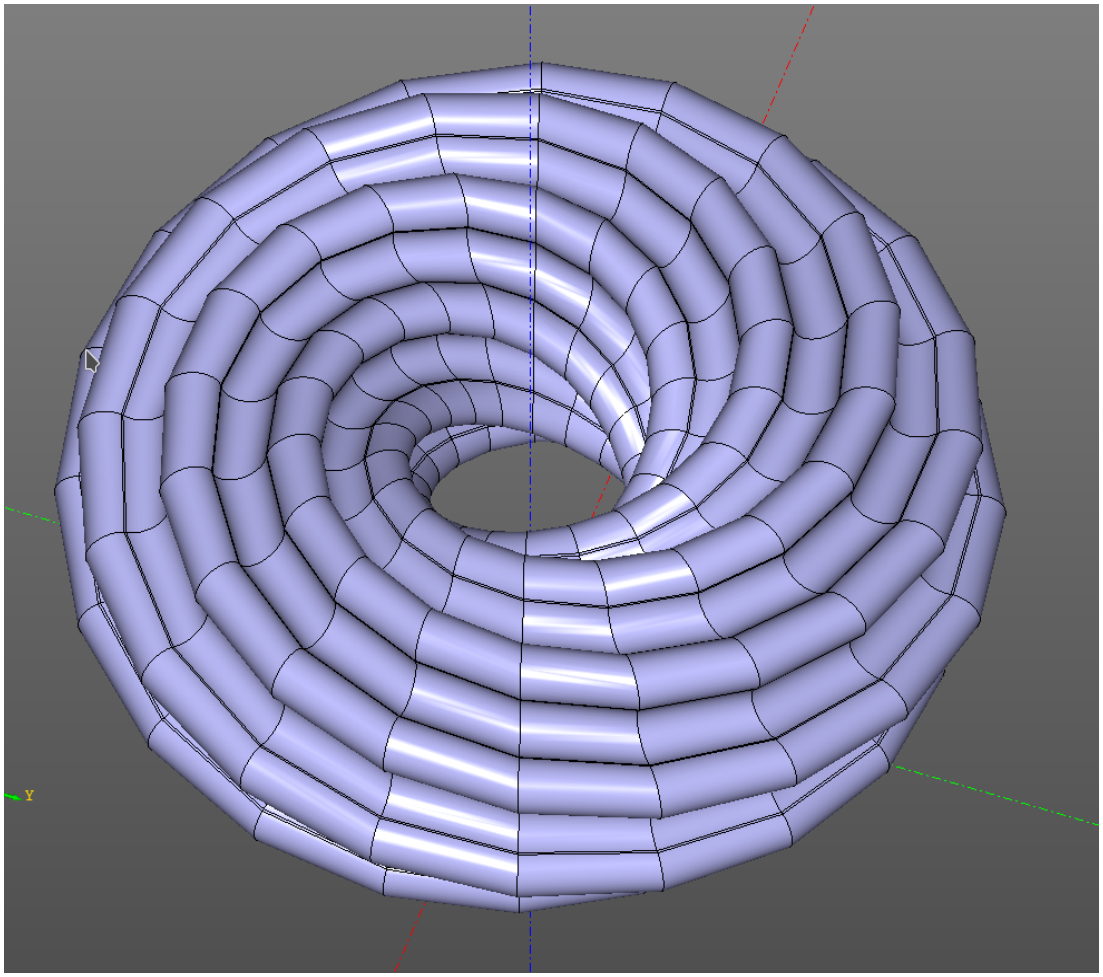

Рис. 6: Обратная форма глобоидного червя

```
            g.append(gear(teeth,step).rotateY(i*a + da)\
                                .translate(r,0,20).rotateZ(i*b + db))
    w = loft(g)
    return w
x = [i for i in range(0,4,1)]
with futures.ThreadPoolExecutor() as executor:
    worm = executor.map(chast, x)
```

ЗАКЛЮЧЕНИЕ

Бла-бла-бла просто гений

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. какая-то статья