Содержание

1 Аналитический раздел

1.1 Допустимые положения пальцев рук

Каждый палец, кроме первого, состоит из проксимальной (более приближенной к лучезапястному суставу), медиальной (средней) и дистальной (наиболее удалённой от лучезапястного сустава) фаланг, а большой палец - из проксимальной и дистальной фаланги. Пальцы нумеруются с единицы, начиная с большого пальца.

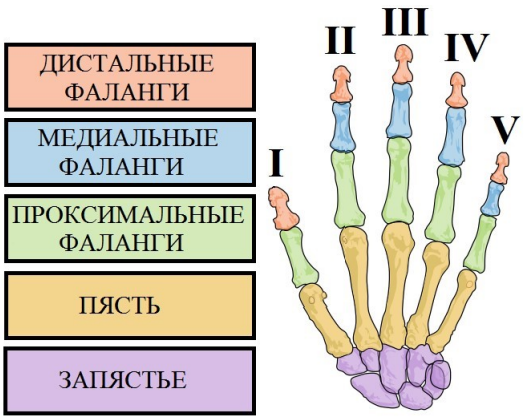

Рис. 1.1: Составные части кисти

В каждом из суставов пальцев кисти выполняется сгибание и разгибание, но только в пястных суставах - отведение и приведение. Движения могут быть следующими: активными (вызванными собственными сокращениями мышц кисти) и пассивными (вызванными внешней причиной). Большой палец, который имеет наиболее сложную структуру, способен на такие движения, как сгибания и разгибание, отведение и приведение, а также противопоставление при помощи пястной кости. В пястно-фаланговом и межфаланговом суставах большого пальца возможны только сгибание и разгибание.

Простые движения кисти приведены на рисунках 1.2 и 1.3. Амплитуда сгибания/разгибания и отведения/приведения измеряется из нейтрального положения, при котором ось кисти, проходящая через средний палец и третью пястную кость, коллинеарна продольной оси предплечья.

Вокруг вертикальной оси в горизонтальной плоскости осуществляются вращение внутрь (пронация) и наружу (супинация) большого пальца вместе с пястной костью при помощи пястно-запястного сустава. Большой палец при сгибании вокруг сагиттальной оси во фронтальной плоскости смещается в сторону ладони, при этом возможно противопоставление остальным пальцам. Движение вокруг фронтальной оси определяется как приведение и отведение большого пальца.

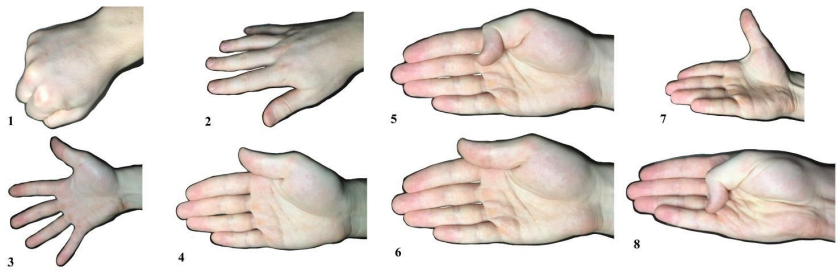

Рис. 1.2: Движения кисти. 1. Сгибание пальцев. 2. Разгибание пальцев 3. Отведение пальцев. 4. Приведение пальцев. 5. Сгибание большого пальца 6. Разгибание большого пальца. 7. Отведение большого пальца. 8. Противопоставление большого пальца

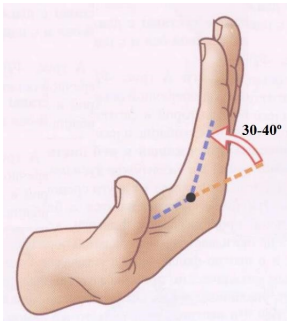

Рис. 1.3: Активное разгибание в пястно-фаланговом суставе

Простые движения состоят из:

- «Сгибания в лучезапястном суставе» или «Разгибания в лучезапястном суставе»;
- «Отведения в лучезапястном суставе» или «Приведения в луче- запястном суставе»;
- «Сгибания пальцев» или «Разгибания пальцев» (большой палец рассматривается отдельно);
- «Отведения пальцев» или «Приведения пальцев» (большой палец рассматривается отдельно);
- «Сгибания большого пальца» или «Разгибания большого пальца»;
- «Отведения большого пальца» или «Приведения большого пальца»;
- «Противопоставления большого пальца».

Были составлены таблицы 1.1 и 1.2 амплитудных углов кисти и пальцев от нейтрального положения. Также, принимая во внимание факт, что движения отведения или приведения совершаются самостоятельно только проксимальными фалангами, все другие фаланги данного движения самостоятельно не выполняют и являются ведомыми. II палец отличается способностью производить отведение в 60° от начального положения, сохраняя возможность выполнять приведение и отведение в диапазоне 30°.

Таблица 1.1: Угловая амплитуда движений ладони

Движение	**Амплитуда, °**
Сгибание	80 - 85
Разгибание	70 - 85
Отведение	15 - 25
Приведение	30 - 45

Таблица 1.2: Угловая амплитуда движений пальцев

Движение			Сгибание	Разгибание	Отведение	Приведение
Амп., ^{o}	БП	Пястная кость и запястье	20	!20	20	20
		Проксимальная фаланга и пястная кость	50	!50	-	-
		Дистальная и проксимальная фаланги	80	!80+(20)	-	-
	II палец	Проксимальная фаланга и пястная кость	-	-	60	!60
	II, III, IV, V пальцы	Проксимальная фаланга и пястная кость	90	!90+(30)	30	30
		Медиальная и проксимальная фаланги	100	!100	-	-
		Дистальная и медиальная фаланги	80	!80	-	-
	Ладонь	Ладонь и предплечье	70		80	

1.2 Prolog

Prolog — язык и система логического программирования, основанные на языке предикатов математической логики дизъюнктов Хорна, представляющей собой подмножество логики предикатов первого порядка. Язык сосредоточен вокруг небольшого набора основных механизмов, включая сопоставление с образцом, древовидного представления структур данных и автоматического перебора с возвратами. Хорошо подходит для решения задач, где рассматриваются объекты (в частности структурированные объекты) и отношения между ними. Пролог, благодаря своим особенностям, используется в области искусственного интеллекта, компьютерной лингвистики и нечислового программирования в целом. В некоторых случаях реализация символьных вычислений на других стандартных языках вызывает необходимость создавать большое количество кода, сложного в понимании, в то время как реализация тех же алгоритмов на языке Пролог даёт простую программу, легко помещающуюся на одной странице.

Prolog является декларативным языком программирования: логика программы выражается в терминах отношений, представленных в виде фактов и правил. Для того чтобы инициировать вычисления, выполняется специальный запрос к базе знаний, на которые система логического программирования генерирует ответы «истина» и «ложь». Для обобщённых запросов с переменными в качестве аргументов созданная система Пролог выводит конкретные данные в подтверждение истинности обобщённых сведений и правил вывода. Иначе говоря, предикат можно определить как функцию, отображающую множество произвольной природы в множество булевых значений ложно, истинно. Задача пролог-программы заключается в том, чтобы доказать, является ли заданное целевое утверждение следствием из имеющихся фактов и правил.

Основными понятиями в языке Пролог являются факты, правила логического вывода и запросы, позволяющие описывать базы знаний, процедуры логического вывода и принятия решений. В логическом программировании, как оно реализовано в прологе, используется только одно правило вывода — резолюция. В языке пролог исходное множество формул, для которого ищется пустая резольвента, представляется в виде так называемых «дизъюнктов Хорна»:

Программа на Прологе описывает отношения, определяемые с помощью предложений. Как и в любом другом языке, ориентированном на символьные вычисления, предложения выстраива-

ются из термов, которые в свою очередь подразделяются на атомы, числа, переменные и структуры. Структуры представляют собой совокупности термов, заключённые в круглые скобки, в том числе и другие структуры. Структура обозначается именем (функтором), которое располагается перед круглыми скобками.

book('Название', '2009', 'Спб', authors('Первый автор', 'Второй автор')).

Ещё одной конструкцией являются списки, элементы которых заключаются в квадратные скобки. В основе списков в Пролог лежат связные списки.

Правила в Прологе записываются в форме правил логического вывода с логическими заключениями и списком логических условий. В чистом Прологе предложения ограничиваются дизъюнктами Хорна. Факты в языке Пролог описываются логическими предикатами с конкретными значениями. Факты в базах знаний на языке Пролог представляют конкретные сведения (знания). Обобщённые сведения и знания в языке Пролог задаются правилами логического вывода (определениями) и наборами таких правил вывода (определений) над конкретными фактами и обобщёнными сведениями. Предложения с пустым телом называются фактами.

2 Конструкторский раздел

2.1 Углы Эйлера

Углы Эйлера определяют три поворота системы, которые позволяют привести любое положение системы к текущему. Обозначим начальную систему координат как (x, y, z) , конечную как (X,Y,Z). Пересечение координатных плоскостей xy и XY называется линией узлов N.

- Угол a между осью x и линией узлов — угол прецессии.
- Угол b между осями z и Z — угол нутации.
- Угол y между линией узлов и осью X— угол собственного вращения.

Повороты системы на эти углы называются прецессия, нутация и поворот на собственный угол (вращение). Такие повороты некоммутативны и конечное положение системы зависит от порядка, в котором совершаются повороты. В случае углов Эйлера производится серия из трёх поворотов:

1. На угол a вокруг оси z. При этом ось x переходит в N.
2. На угол b вокруг оси N. При этом ось z переходит в Z.
3. На угол y вокруг оси Z. При этом ось N переходит в X.

Иногда такую последовательность называют 3,1,3 (или Z,X,Z), но такое обозначение может приводить к двусмыслице. Для вычисления этих углов используются векторы. Генеральная идея если у нас есть 3 точки P1(X1, Y1, Z1), P2(X2, Y2, Z2), P3(X3, Y3, Z3) состоит в том что бы найти 2 вектора (Вектор $P_1\vec{P}_2$ и вектор $P_2\vec{P}_3$) что бы найти угол между ними. Чтобы найти эти 2 вектора мы воспользуемся формулой

$$\vec{AB} = (X_2 \breve{} X_1, Y_2 \breve{} Y_1, Z_2 \breve{} Z_1)\vec{BC} = (X_3 \breve{} X_2, Y_3 \breve{} Y_2, Z_3 \breve{} Z_2) \tag{2.1}$$

Так как существует формула:

$$\vec{AB} \cdot \vec{BC} = |\vec{AB}| \cdot |\vec{BC}| * \cos(\theta), \tag{2.2}$$

где θ – угол между этими векторами, то из этой формулы мы получаем этот угол помощу arccos. Длину вектора находим по формуле:

$$|\vec{AB}| = \sqrt{(X_{AB}^2 + Y_{AB}^2 + Z_{AB}^2)}. \tag{2.3}$$

Конечная формула получается:

$$\theta = \arccos(\frac{\vec{AB} * \vec{BC}}{|\vec{AB}| * |\vec{BC}|}). \tag{2.4}$$

Если мы хотим найти угол по оси X или оси Y, то мы просто не включаем эти координаты у уравнение, т.е. если хотим например найти по оси X то вектор $P_1\vec{P}_2$ можем вычислить вот так:

$$P_1\vec{P}_2 = (0, Y_2 - Y_1, Z_2 - Z_1). \tag{2.5}$$

2.2 Структура проверок положения точек

Входные данные в программу представляют собой 42 точки в трехмерном пространстве. Из 42 точек, 21 определяет одну кисть.

Проверка кисти на правильность осуществляется с помощью группы проверок отдельных пальцев, а также точек расположенных непосредственно на ладони. Для реализации проверок лучше использовать язык программирования Prolog, поскольку это мощный инструмент именно для работы с логическими конструкциями.

Для удобства представления входных данных их можно разделить на структуры.

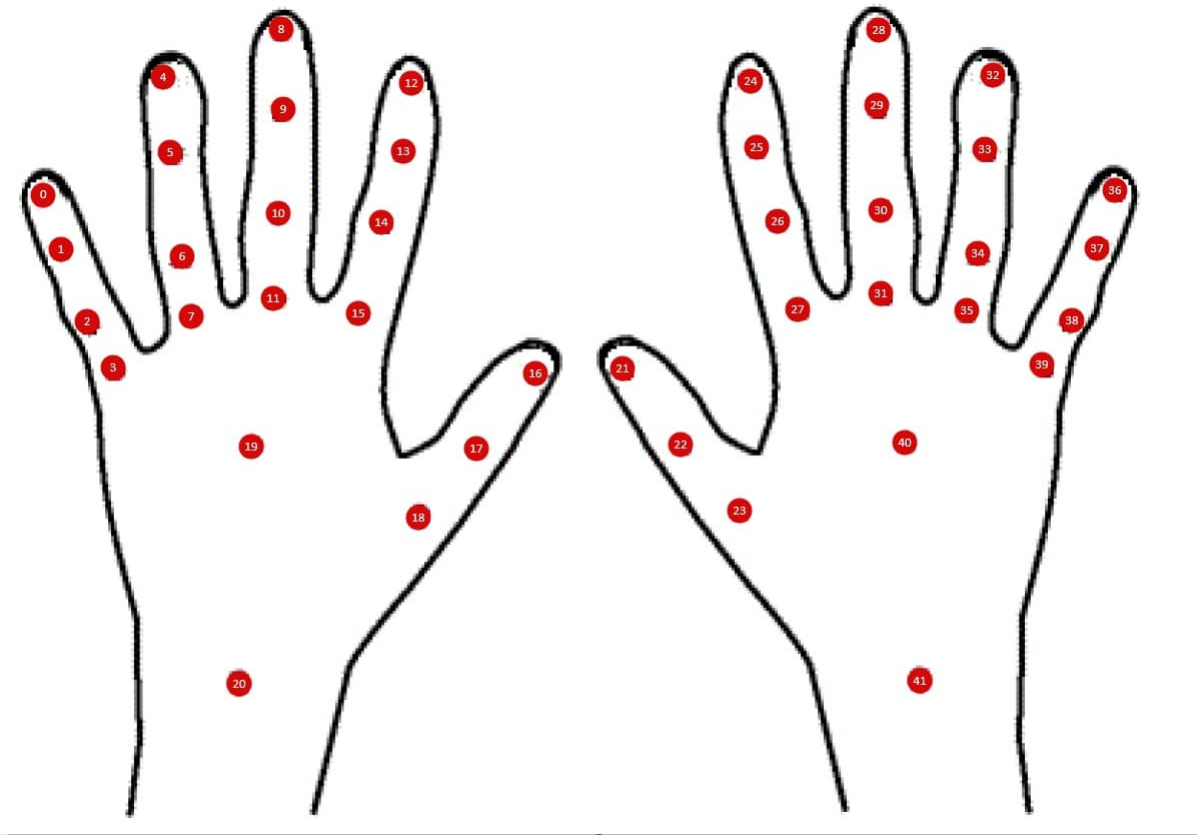

Рис. 2.1: Нумерация точек на кисти

Возьмем точки 0, 1, 2, 3. Вместе они составляют мизинец на руке. В соответствии с этим можно создать структуру мизинца. Схожим образом можно объединить оставшиеся точки в безымянный, средний, указательный и большой пальцы. Останутся только точки 19, 20 на левой руке и 40, 41 на правой. Эти точки являются ключевыми для своих кистей соотвественно.

Далее полученные структуры пальцев мы можем объединить в структуру руки. Каждая рука будет состоять из пальцев и оставшимся двум точкам соответственно для левой и правой кисти.

Данные преобразования необходимо провести для упрощения понимания структуры кода при его чтении, а также облегчения работы при написании процедур проверок.

Сами проверки в своей основе опираются на проверку углов между определенными точками в пальце.

Мы создали структуры проверок для всех пальцев в Прологе, где проверяли является ли угол между тремя точками меньше или больше предполагаемого, и если да то это положение недопустимо. При этом мы находили углы по оси X и по оси Y так как:

- Сгибание - Это когда три точки создают угол по оси X от 0 до -180;
- Разгибание -Это когда три точки создают угол по оси X от 0 до 180;
- Отведение - Это когда три точки создают угол по оси Y от 0 до -180;
- Приведение - Это когда три точки создают угол по оси Y от 0 до 180;

Эта проверка вызывается 2 раза для обе руки.

Как можно увидеть из таблицы 1.2, у нас различаются углы для большого пальца, 4 остальных пальцев и самой ладони. Поэтому мы сделали 3 различные структуры проверок.

Если какие то проверки не проходят допустимость, то мы записываем эти точки в текстовой файл что бы питон потом мог отобразить не допустимые кости руки.

Структуру мы разбили на несколько частей: Три точки -> Палец -> Рука

Структуру трёх точек мы представляем как Point(X, Y, Z)

Для пальца мы сделали 5 структуру для каждого из них:

Листинг 2.1: Листинг структур пальцев

```
        finger(little, P0, P1, P2, P3),
        finger(ring, P4, P5, P6, P7),
        finger(middle, P8, P9, P10, P11),
        finger(index, P12, P13, P14, P15),
        finger(thumb, P16, P17, P18)
```

где P – это номер точки на руке.

Для руки мы сделали структуру:

Листинг 2.2: Листинг структуры руки

```
Hand
(
        finger(little, P0, P1, P2, P3),
        finger(ring, P4, P5, P6, P7),
        finger(middle, P8, P9, P10, P11),
        finger(index, P12, P13, P14, P15),
        finger(thumb, P16, P17, P18),
        P19, P20
)
```

где P_i - точка на руке, i – это номер точки на руке.

Так как углы допустимости одинаковы для пальцев 2, 3, 4, 5, то для них мы сделали одинаковую проверку.

2.3 Визуализация руки

Визуализация руки написана на python, с использованием библиотек TkInter и OpenGL. Для визуализации руки нам нужно приложение, которое может загружать точки из текстового файла, а так же редактировать уже имеющиеся точки с помощью мышки ради изменения их положения.

TkInter – Библиотека для Питона, которая позволяет создавать интерфейс.

OpenGL – API для разработки для рендеринга 3д и 2д графики. Оно позволяет нам вместе с Tkinter визуализировать кисть руки, и показать инвалидные ребра этой кисти. С помощью OpenGL

мы избавились от большого количества формул и алгоритмов, которые бы нам потребовались, если бы мы сами писали свой движок.

Точки будем брать из текстового файла, которых должно быть 42. После проверки допустимости мы будем рендерить руки на экран. Если при проверке будут найдены точки, которые не прошли проверку, надо их запомнить, а при рендеринге их ребра будем окрашивать красный цвет. Точки будем изображать черными квадратами.

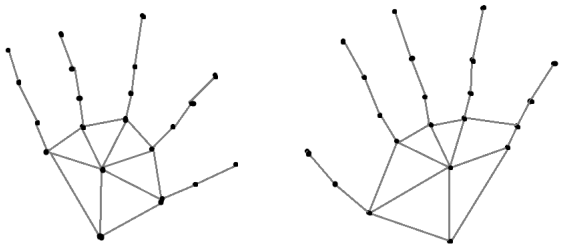

Рис. 2.2: Схематическое изображение желаемого результата рендеринга

3 Технологический раздел

3.1 Средства реализации

Для реализации программы были следующие языки программирования:

- Python (v.3.9(3)) для написания интерфейса программы и отрисовки рук. Python является простым в использовании средством для выполнения небольших задач, таких как чтение и запись, отрисовка оконного интерфейса;
- Prolog (SWI-Prolog(4)) для написания функций проверки точек на корректность.

3.2 Описание структуры базы знаний

3.2.1 Описание фактов

В данном разделе описаны все факты и их назначения.

finger_motion_type

Каждый палец (кроме большого) определяется 4 точками (3 точки для большого), следовательно необходимо проверять 3 различных угла при сгибе пальцев (2 для большого), а также угол отклонения пальца при отведении пальца. Таким образом, каждому пальцу соответствует 4 типа проверок.

- FingerType - тип пальца;
- AbductionType - тип сгиба, отведение или привидение;
- Flexion1 - обозначение, для определения места на пальце, в котором происходит сгиб;

Листинг 3.1: Знания о типах проверок каждого пальца

```
%finger_motion_type(FingerType, AbductionType, Flexion1, Flexion2, Flexion3).

finger_motion_type(thumb, bpprived, bppsgib1, bppsgib2, bppsgib2).
finger_motion_type(index, oprived, o2sgib1, o2sgib2, o2sgib3).
finger_motion_type(middle, oprived, o3sgib1, o3sgib2, o3sgib3).
finger_motion_type(ring, oprived, o4sgib1, o4sgib2, o4sgib3).
finger_motion_type(little, oprived, o5sgib1, o5sgib2, o5sgib3).
```

angle_type_limits

Каждому типу проверок соответствуют диапазоны углов, которые допустимы при том или ином движении пальца.

- Finger - тип пальца;

- MinAngle - минимальное значение угла отклонения;
- MaxAngle - максимальное значение угла отклонения;

Листинг 3.2: Знания об амплитудах углов

```
%angle_type_limits(Finger, MinAngle, MaxAngle)

angle_type_limits(bpabc, -80, 80).
angle_type_limits(bpbcd, -50, 50).
angle_type_limits(bpcde, -90, 90).
angle_type_limits(oabc, -80, 80).
angle_type_limits(obcd, -100, 100).
angle_type_limits(ocde, -90, 90).
angle_type_limits(between, -30, 30).

angle_type_limits(bpprived, -50, 50).
angle_type_limits(oprived, -60, 60).
angle_type_limits(bppsgib1, -50, 50).
angle_type_limits(bppsgib2, -100, 80).

angle_type_limits(o2sgib1, -120, 90).
angle_type_limits(o2sgib2, -100, 100).
angle_type_limits(o2sgib3, -100, 100).

angle_type_limits(o3sgib1, -120, 90).
angle_type_limits(o3sgib2, -100, 100).
angle_type_limits(o3sgib3, -80, 80).

angle_type_limits(o4sgib1, -120, 90).
angle_type_limits(o4sgib2, -100, 100).
angle_type_limits(o4sgib3, -80, 80).

angle_type_limits(o5sgib1, -120, 90).
angle_type_limits(o5sgib2, -100, 100).
angle_type_limits(o5sgib3, -80, 80).

angle_type_limits(bppz, -100, 100).
```

angle_det_type

Также каждый из типов проверок отвечает за конкретную ось пространства, по которой проводится проверка.

- Type - место сгиба на пальце;
- Axis - обозначает плоскость, в которой ищем угол;

Листинг 3.3: Знания об осях типов проверок

```
%angle_det_type(Type, Axis)

angle_det_type(bpabc, all).
angle_det_type(bpbcd, all).
angle_det_type(bpcde, all).
angle_det_type(oabc, all).
angle_det_type(obcd, all).
angle_det_type(ocde, all).
angle_det_type(between, all).

```

```
angle_det_type(bpprived, x).
angle_det_type(oprived, x).
angle_det_type(bppsgib1, y).
angle_det_type(bppsgib2, y).

angle_det_type(o2sgib1, x).
angle_det_type(o2sgib2, x).
angle_det_type(o2sgib3, x).

angle_det_type(o3sgib1, x).
angle_det_type(o3sgib2, x).
angle_det_type(o3sgib3, x).

angle_det_type(o4sgib1, x).
angle_det_type(o4sgib2, x).
angle_det_type(o4sgib3, x).

angle_det_type(o5sgib1, x).
angle_det_type(o5sgib2, x).
angle_det_type(o5sgib3, x).

angle_det_type(bppz, z).
```

hand, finger, point

Программой на Prolog для описания руки используются структуры Рука, Палец и Точка.

- point - структура Точка, состоит из трех координат;
- hand - структура Рука, из 5 Finger и двух опорных точек кисти;
- finger - структура Палец, состоит из 4 ил 3 точек;

Листинг 3.4: Структуры

```
point(X, Y, Z).

hand(
        finger(little, P0, P1, P2, P3),         %5|finger V
        finger(ring, P4, P5, P6, P7),           %4|finger IV
        finger(middle, P8, P9, P10, P11),       %3|finger III
        finger(index, P12, P13, P14, P15),      %2|finger II
        finger(thumb, P16, P17, P18),           %1|finger I
        P19, P20
).
```

3.2.2 Описание правил

В данном разделе описаны все правила, их назначение и использованные в них переменные.

valid_angle

valid_angle - правило, с помощью которого определяется, входит ли переданный угол в его возможный диапазон. Диапазон задан от MinAngle до MaxAngle. Значения MaxAngle и MinAngle зависят от типа проверяемого угла.

- Type - тип проверяемого угла;
- Angle - заданное значение угла;
- MinAngle - минимальный возможный угол;
- MaxAngle - максимальный возможный угол.

Листинг 3.5: Реализация правила valid_angle

```
1 valid_angle(Type, Angle):-
2         angle_type_limits(Type, MinAngle, MaxAngle),
3         MinAngle =< Angle, Angle =< MaxAngle.
```

vec_length

vec_length - правило, которое определяет длину вектора и помещает ее в Len.

- vector(X,Y,Z) - структура «вектор», определяемая тремя координатами;
- len - длина вектора;

Листинг 3.6: Реализация правила vec_length

```
1 vec_length(vector(X, Y, Z), Len) :- Len is sqrt(X * X + Y * Y + Z * Z).
```

vec_length_sqr

vec_length_sqr - правило, которое определяет квадрат длины вектора и помещает ее в Len.

- vector(X,Y,Z) - структура «вектор», определяемая тремя координатами;
- len - длина вектора;

Листинг 3.7: Реализация правила vec_length_sqr

```
1 vec_length_sqr(vector(X, Y, Z), Len) :- Len is X * X + Y * Y + Z * Z.
```

dot_prod

dot_prod - правило, которое получает две структуры vector и помещает в DotProd значение их перемножения.

- vector(X,Y,Z) - структура «вектор», определяемая тремя координатами;
- DotProd - результат скалярного произведения векторов.

Листинг 3.8: Реализация правила dot_prod

```
dot_prod(vector(X1, Y1, Z1), vector(X2, Y2, Z2), DotProd) :-
        DotProd is X1 * X2 + Y1 * Y2 + Z1 * Z2.
```

rad_to_deg

rad_to_deg - правило, которое получает значение в радианах и переводит его в градусы.

- Radian - величина в радианах;
- Degrees - величина в градусах.

Листинг 3.9: Реализация правила rad_to_deg

```
rad_to_deg(Radian, Degrees) :- Degrees is Radian * 180 / 3.1415.
```

deg_to_rad

deg_to_rad - правило, которое получает значение в градусах и переводит его в радианы.

- Radian - величина в радианах;
- Degrees - величина в градусах.

Листинг 3.10: Реализация правила deg_to_rad

```
deg_to_rad(Degrees, Radian) :- Radian is Degrees * 3.1415 / 180.
```

angle_between_vectors

angle_between_vectors - правило, которое определяет угол между двумя заданными векторами и помещает значение Angle.

- Vector1, Vector2 - структуры векторов, которые состоят из 3 переменных X,Y,Z;
- Angle - значение угла между векторами в градусах;
- Len1Sqr, Len2Sqr - значения квадратов длин Vector1 и Vector2 соответственно;
- DotProd - значение скалярного произведения векторов Vector1, Vector2;
- AngleRad - значение угла в радианах.

Листинг 3.11: Реализация правила angle_between_vectors

```
angle_between_vectors(Vector1, Vector2, Angle) :-
        vec_length_sqr(Vector1, Len1Sqr),
        vec_length_sqr(Vector2, Len2Sqr),
        dot_prod(Vector1, Vector2, DotProd),
        AngleRad is acos(DotProd / sqrt(Len1Sqr * Len2Sqr)),
        rad_to_deg(AngleRad, Angle).
```

get_angle

get_angle - правило, которое по заданным точкам в пространстве, определяет угол между векторами, которые они образовывают.

- all - обозначение того, что искать необходимо угол по всем осям сразу;
- x - обозначение того, что искать необходимо угол по оси X;
- y - обозначение того, что искать необходимо угол по Y;
- z - обозначение того, что искать необходимо угол по Z;
- point(X, Y, Z) - структура, заданная тремя координатами;
- Angle - значение угла между векторами, образованными первой, второй и третьей точками.

Листинг 3.12: Реализация правила get_angle

```
get_angle(all, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3, Z3), Angle) :-
        AX is X2 - X1, AY is Y2 - Y1, AZ is Z2 - Z1,
        BX is X3 - X2, BY is Y3 - Y2, BZ is Z3 - Z2,
        angle_between_vectors(vector(AX, AY, AZ), vector(BX, BY, BZ), Angle).

get_angle(x, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3, Z3), Angle) :-
        AY is Y1 - Y2, AZ is Z1 - Z2,
        BY is Y3 - Y2, BZ is Z3 - Z2,
        angle_between_vectors(vector(0, AY, AZ), vector(0, BY, BZ), Angle).

get_angle(y, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3, Z3), Angle) :-
        AX is X2 - X1, AZ is Z2 - Z1,
        BX is X3 - X2, BZ is Z3 - Z2,
        angle_between_vectors(vector(AX, 0, AZ), vector(BX, 0, BZ), Angle).

get_angle(z, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3, Z3), Angle) :-
        AX is X2 - X1, AY is Y2 - Y1,
        BX is X3 - X2, BY is Y3 - Y2,
        angle_between_vectors(vector(AX, AY, 0), vector(BX, BY, 0), Angle).
```

validate_angle

validate_angle - правило, которое проверяет направленность угла, его размер и является ли значение угла допустимым.

- Type - тип проверяемого угла;
- Point - структура точки, заданная тремя координатами.

Листинг 3.13: Реализация правила validate_angle

```
validate_angle(Type, Point1, Point2, Point3) :-
        hand:angle_det_type(Type, Axis),
        get_angle(Axis, Point1, Point2, Point3, Angle),
        write_files:write_angle(Type, Angle),
        hand:valid_angle(Type, Angle).
```

validate_points

validate_points - правило, которое проверяет корректность данных X1,Y1,Z1, а также 2, 3. Есть три реализации. Первая проверяет корректность или наличие значений, вторая проверяет на корректность угол между тремя точками, а третья на случай, если угол неправильный пишет сообщение о том, что точки не подходят.

- Type -тип угла (между какими точками проверяется угол и какие заданы на него ограничения);
- X,Y,Z - список координат точки по X, Y, Z.

Листинг 3.14: Реализация правила validate_points

```
validate_points(Type, [X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]):-
        not(check_3coords([X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3])),
        write_files:write_invalid_data().

validate_points(Type, [X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]):-
        check_3coords([X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]),
        validate_angle(Type, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3, Z3)),
        write_files:write_angle_is_valid(Type).

validate_points(Type, [X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]):-
        check_3coords([X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]),
        not(validate_angle(Type, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3,
            Z3))),
        write_files:write_invalid_points([X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]).
```

validate_hand

validate_hand - правило, которое проверяет переданные ему конструкции finger.

- hand - структура рука, которая состоит из 5 структур типа Finger, которые состоят из 4 точек для указательного, среднего, безымянного и мизинца, из 3 для большего пальца, а также точек оставшихся двух ключевых точек на ладони;
- finger - структура пальца состоящая из 3 или 4 точек;
- Wrist - точка на запястье;
- p19 - точка на пясти.

Листинг 3.15: Реализация правила validate_hand

```
validate_hand(hand:hand(Finger5, Finger4, Finger3, Finger2, Finger1, P19, Wrist)):-
        validate_finger(Finger5, P19, Wrist),
        validate_finger(Finger4, P19, Wrist),
        validate_finger(Finger3, P19, Wrist),
        validate_finger(Finger2, P19, Wrist),
        validate_finger(Finger1, P19, Wrist).
```

validate_finger

validate_finger - правило, которое проверяет корректность точек на пальце. .

- finger - структура пальца, которая состоит из вида пальца (большой/безымянный/средний и т. д.) и точек;
- Wrist - точка на запястье;

- p19 - точка на пясти;
- finger_motion_type - знание об амплитуде угла для конкретного типа пальца.

Листинг 3.16: Реализация правила validate_finger

```
validate_finger(finger(thumb, P1, P2, P3), P19, Wrist):-
        finger_motion_type(thumb, Abduction, Flex1, Flex2, _),
        validate_points(bpabc, P1, P2, P3),
        validate_points(Abduction, P3, P2, Wrist),
        validate_points(Flex1, P1, P2, P3),
        validate_points(Flex2, P1, P2, Wrist),
        validate_points(bppz, P1, P2, P3).
```

check_coords

check_coords - правило, которое проверяет наличие данных в x,y,z.

x,y,z - список координат;

Листинг 3.17: Реализация правила check_coords

```
check_coords([X, Y, Z]):- number(X), number(Y), number(Z).
```

check_3coords

check_3coords - правило, которое проверяет наличие данных в трех списках координат.

x,y,z - список координат;

Листинг 3.18: Реализация правила check_3coords

```
check_3coords([X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]) :-
        check_coords([X1, Y1, Z1]),
        check_coords([X2, Y2, Z2]),
        check_coords([X3, Y3, Z3]).
```

validate_all

validate_all - правило, которое правило, с которого начинается вся работа в программе. Оно начинает проверку корректности рук, которые задаются в качестве структур hand.

- Working_Dir - путь к рабочей директории с файлом точек;
- Result - результат работы правила;
- Point1 - структура точки, состоит из трех координат.

Листинг 3.19: Реализация правила validate_all

```
validate_all(Working_Dir, Result,
        Point1, Point2, Point3, Point4, Point5, Point6, Point7,
        Point8, Point9, Point10, Point11, Point12, Point13, Point14,
        Point15, Point16, Point17, Point18, Point19, Point20, Point21,
        Point22, Point23, Point24, Point25, Point26, Point27, Point28,
        Point29, Point30, Point31, Point32, Point33, Point34, Point35,
        Point36, Point37, Point38, Point39, Point40, Point41, Point42
) :-
```

```
9         working_directory(_, Working_Dir),
10        open('points.txt', write, Stream),
11        open('angles.txt', write, Stream2),
12        close(Stream2),
13        close(Stream),
14        (
15                (
16                        validate_hand(
17                                hand:hand(
18                                        finger(little, Point1, Point2, Point3, Point4),
19                                        finger(ring, Point5, Point6, Point7, Point8),
20                                        finger(middle, Point9, Point10, Point11,
                                              Point12),
21                                        finger(index, Point13, Point14, Point15,
                                              Point16),
22                                        finger(thumb, Point17, Point18, Point19),
23                                        Point20, Point21
24                                )
25                        ),
26                        validate_hand(
27                                hand:hand(
28                                        finger(little, Point37, Point38, Point39,
                                              Point40),
29                                        finger(ring, Point33, Point34, Point35, Point36),
30                                        finger(middle, Point29, Point30, Point31,
                                              Point32),
31                                        finger(index, Point25, Point26, Point27,
                                              Point28),
32                                        finger(thumb, Point22, Point23, Point24),
33                                        Point41, Point42
34                                )
35                        )
36                )-> Result = "Ok"; Result = "Not"
37        ).
```

write_invalid_data

write_invalid_data - правило, которое выводит сообщение о некорректности данных.

Листинг 3.20: Реализация правила write_invalid_data

```
1 write_invalid_data() :- write("Invalid data "), nl.
```

write_angle

write_angle - правило, которое выводит значение угла.

- Angle - значение угла.

Листинг 3.21: Реализация правила write_angle

```
1 write_angle(Angle) :- write("Angle is "), write(Angle), nl.
```

write_angle

write_angle - правило, которое выводит значение и тип угла.

- Angle - значение угла;
- Type - тип угла.

Листинг 3.22: Реализация правила write_angle

```
write_angle(Type, Angle):- write(Type), write(" angle is "), write(Angle), nl.
```

write_angle_is_valid

write_angle_is_valid - правило, которое пишет о корректности угла в файл.

- Type - тип данного угла.

Листинг 3.23: Реализация правила write_angle_is_valid

```
write_angle_is_valid(Type) :-
        open('angles.txt', append, Stream2),
        string_concat(Type, " is ok", Msg),
        write(Stream2, Msg), nl(Stream2),
        close(Stream2).
```

write_invalid_points

write_invalid_points - правило, которое записывает не корреткные точки в файл.

- Point - структура точки, состоит из трех координат.

Листинг 3.24: Реализация правила write_invalid_points

```
write_invalid_points(Point1, Point2, Point3) :-
        open('points.txt', append, Stream),
        write(Stream, [Point1, Point2, Point3]), nl(Stream),
        close(Stream).
```

point_to_str

point_to_str - правило, которое приводит значение точки к строковому представлению.

- Point - структура точки, состоящая из трех координат;
- Str - строковое представление точки.

Листинг 3.25: Реализация правила point_to_str

```
point_to_str(Point, Str) :-
        determ:get_coords(Point, X, Y, Z),
        number_string(X, StrX),
        number_string(Y, StrY),
        number_string(Z, StrZ),
        string_concat(StrX, ";", StrX1),
        string_concat(StrY, ";", StrY1),
        string_concat(StrX1, StrY1, StrXY),
        string_concat(StrXY, StrZ, StrXYZ),
        string_concat(StrXYZ, "\n", Str).
```

write_list

write_list - правило, которое пишет в список в файл.

- Stream - поток записи;
-

  - список;
- Head - голова списка;
- Tail - хвост списка.

Листинг 3.26: Реализация правила write_list

```
write_list(Stream, []).
write_list(Stream, [Head|Tail]) :-
        write(Stream, Head),
        write_list(Stream, Tail).
```

write_points

write_points - правило, которое пишет точки из списка в файл.

- Filename - путь к файлу с координатами;
- PList - список координат, полученных из файлов.

Листинг 3.27: Реализация правила write_points

```
write_points(PList,Filename) :-
    open(Filename, write, Stream),
    convlist(point_to_str, PList, StrList),
        write_list(Stream, StrList),
    close(Stream).
```

3.3 Средства взаимодействия python и Prolog

Для обеспечения взаимодействия частей программы на языках Python и Prolog, была использована библиотека PySwip, которая позволяет делать запросы со входными данными из части Python, а также консультровать выбранные файлы. В листинге 3.28 продемонстрированы две функции. использующиеся для взаимодействия.

Листинг 3.28: Взаимодействие python и Prolog

```
from pyswip import Prolog
import os

def append_base_stored(prolog, filename, filepath=None):
    if filepath is None:
        filepath = ''
    fullpath = os.path.join(filepath, filename)
    prolog.consult(fullpath)

def get_answer(basestored, statement, filepath=None):
    prolog = Prolog()
    append_base_stored(prolog=prolog,
```

```
13                          filename=basestored,
14                          filepath=filepath)
15      return prolog.query(statement)
```

append_base_stored - функция, которая по имени пути и имени файла, консультирует файл по этому пути, тем самым запуская реализацию части программы на Prolog.

get_answer - функция, которая отправляет в программу запрос и возвращает состояние, после того как она обработала запрос.

3.4 Интерфейс программы

Для удобства создания GUI для работы с кистями была выбрана библиотека tkinter, с надстройками позволяющими рисовать 3D изображения, а также добавить интерфейс взаимодействия с точками кисти, возможностью их изменять, сохранять или записывать в файл; Интерфейс на tkinter состоит из:

- Холста, на котором происходит отрисовка;
- Небольшом изображении в углу экрана, которое служит для выбора точек на кисти, координаты которых пользователь может поменять;
- Кнопок изменения координат выбранной точки по X, Y, Z;
- Кнопок для загрузки и сохранения координат в файл;
- Кнопки для проверки текущих точек на корректность;

3.5 Отрисовка рук в Python

Отрисовка кисти совершается следующим образом. Точки загружаются из текстового файла, которых должно быть 42. Полученные 42 точки делятся по 21, для каждой кисти соответственно. После необходимых проверок на Prolog, по этим точкам рисуется каркас, показывающий их схематичное расположение относительно друг друга. При этом если ребра окрашены красным цветом, то это означает, что положение отдельной части руки в этих трех точках недопустимо. Левая рука окрашена зелёным цветом, а правая - рыжим. Точки обозначены чёрными квадратами. Чтобы визуально было проще понимать, какая рука где находится, а также определять, куда "смотрит"ладонь, а где ее тыльная сторона, было решено добавить модель кисти. Относительно каждых групп точек рисуется геометрическая фигура, отдаленно напоминающая ту часть кисти, которой соответствует. Например возьмем указательный палец. Его можно разбить на два прямоугольника и треугольник, как показано на рисунке ниже.

- class PhalanxModel - класс, в котором происходит рендеринг фаланги пальца, но не на дальнем от ладони конце пальца;
- class EndPhalanxModel(PhalanxModel) - класс, в котором происходит рендеринг конца фаланги пальца;
- class FingerModel - класс, в котором происходит рендеринг пальца;
- class HandModel - класс, в котором происходит рендеринг руки;
- class RightHandModel(HandModel) - класс, в котором происходит рендеринг правой руки. Этот класс нужен, потому что данные для левой и правой рук отличаются последовательностью подаваемых на вход координат;

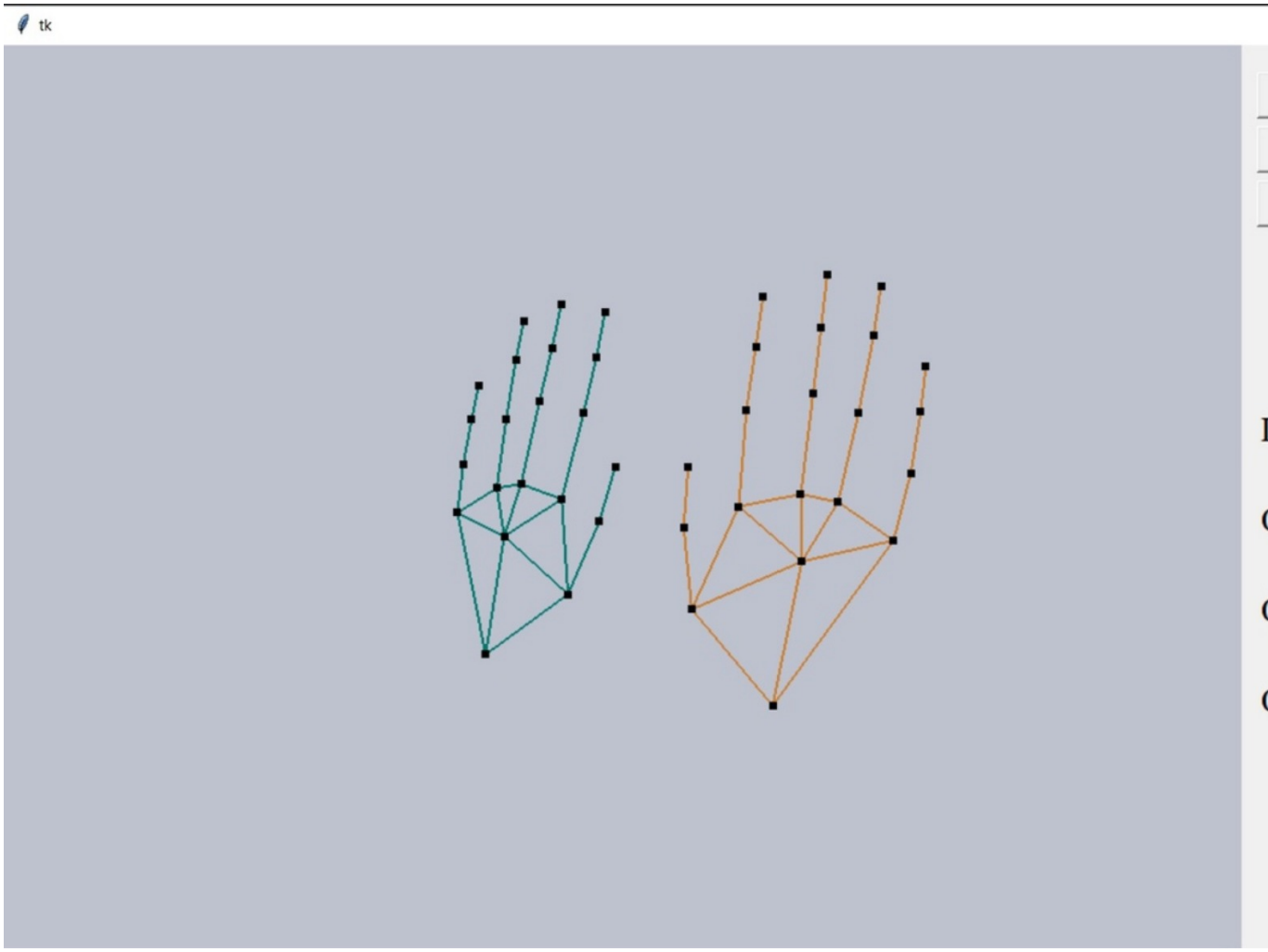

Рис. 3.1: Интерфейс программы

Перечисленные выше классы отвечают за рендеринг кисти и ее деталей. Перемещать камеру возможно с помощью стрелок на клавиатуре, масштабирование осуществляется с помощью кнопок ’+’ и ’-’.

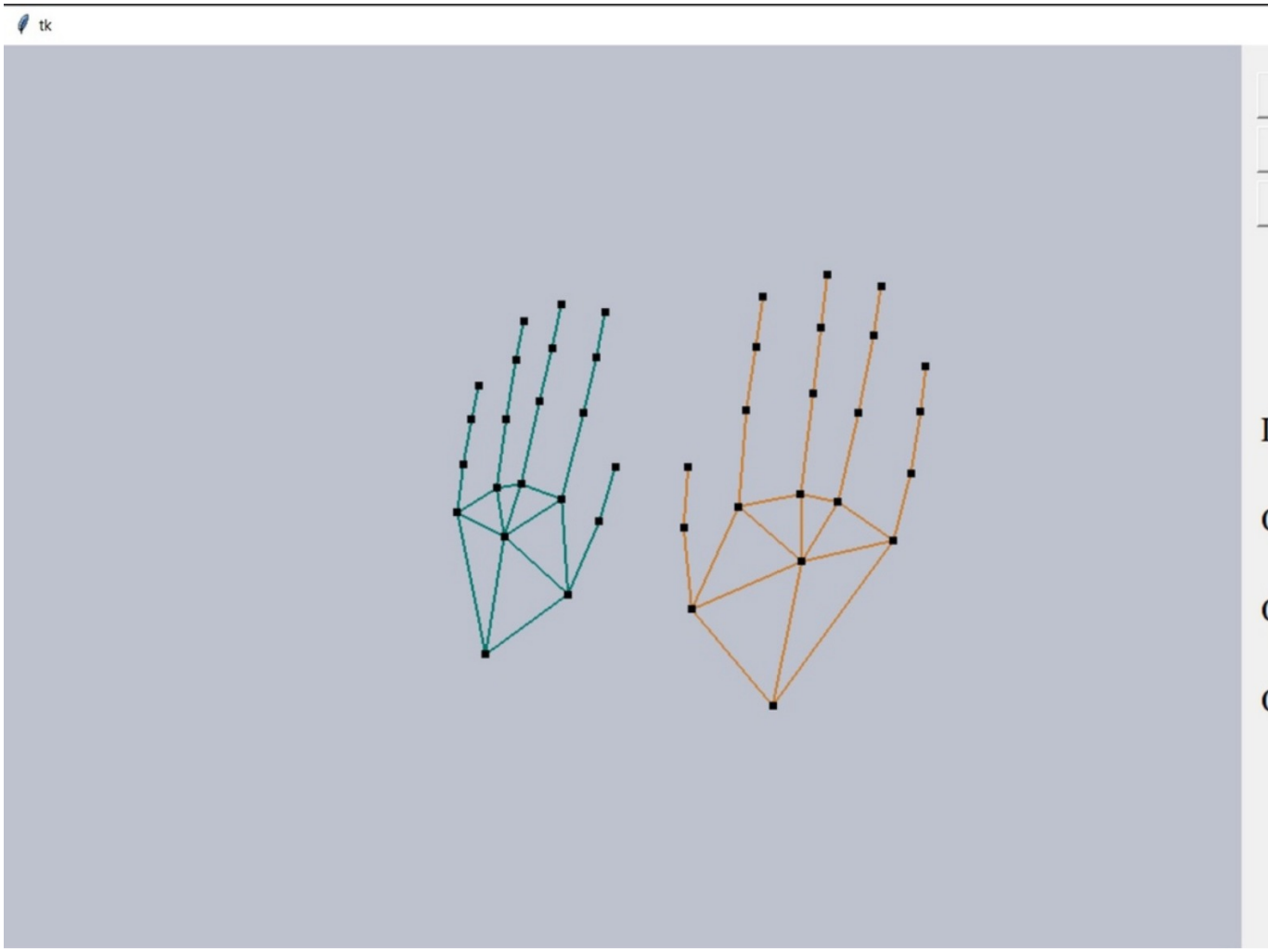

Рис. 3.2: Интерфейс программы 1

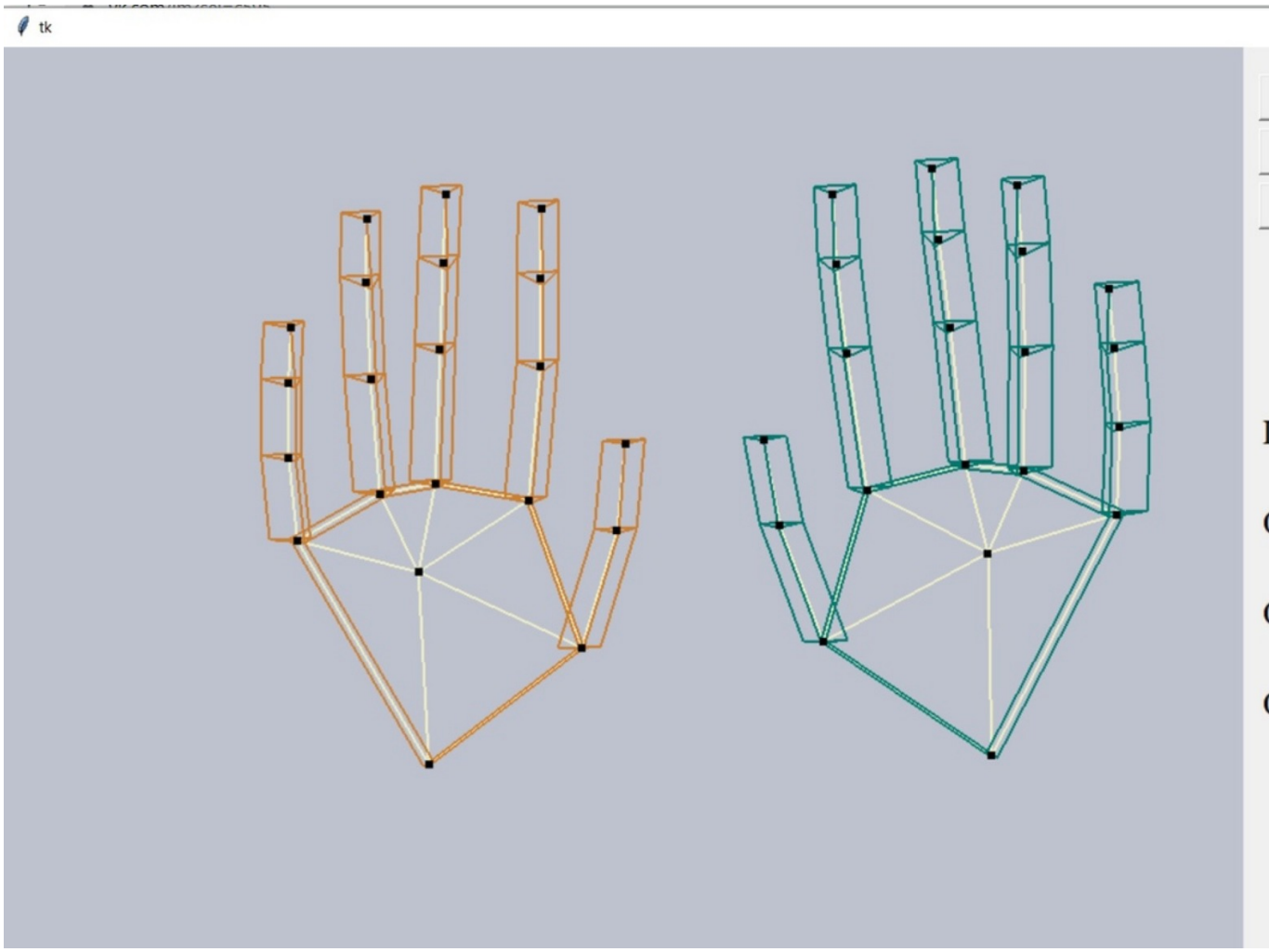

Рис. 3.3: Интерфейс программы 2

Список использованных источников

[1] Анатолий Адаменко, Андрей Кучуков. Логическое программирование и Visual Prolog. - СПб.: БХВ-Петербург, 2003 - 990 с. - ISBN 5-94157-156-9.

[2] Марков, В. Н. Современное логическое программирование на языке Visual Prolog 7.5. - СПб.: БХВ-Петербург, 2016 - 544 с. - ISBN 978-5-9775-3487-1.

[3] Документация Python 3.9 [Электронный ресурс]. - Режим доступа: https://docs.python.org/3/index.html, свободный. (Дата обращения: 25.09.2021 г.)

[4] Документация SWI-Prolog [Электронный ресурс]. - Режим доступа: https://www.swi-prolog.org/pldoc/index.html, свободный. (Дата обращения: 25.09.2021 г.)