

Содержание

1	Аналитический раздел	2
1.1	Допустимые положения пальцев рук	2
1.2	Prolog	5
2	Конструкторский раздел	6
2.1	Углы Эйлера	6
2.2	Структура проверок положения точек	7
2.3	Визуализация руки	8
3	Технологический раздел	10
3.1	Средства реализации	10
3.2	Описание логики работы модуля программы на Prolog	10
3.3	Описание структуры базы знаний	11
3.3.1	Описание фактов	11
3.3.2	Описание структур	13
3.3.3	Описание правил	14
3.4	Средства взаимодействия python и Prolog	22
3.5	Интерфейс программы	22
3.6	Отрисовка рук в Python	22
3.7	Примеры работы	23
	Список использованных источников	26

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 удаленной от лучезапястного сустава) фаланг, а большой палец - из проксимальной и дистальной фаланги. Пальцы нумеруются с единицы, начиная с большого пальца.

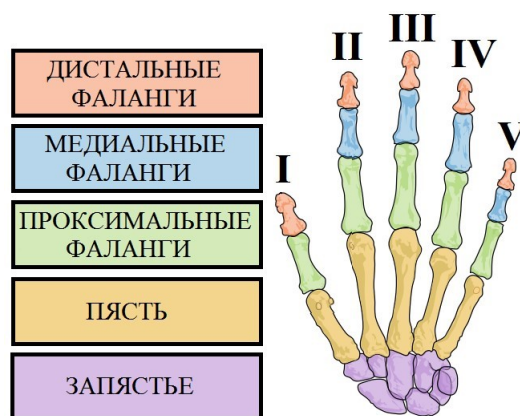


Рис. 1.1: Составные части кисти

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.

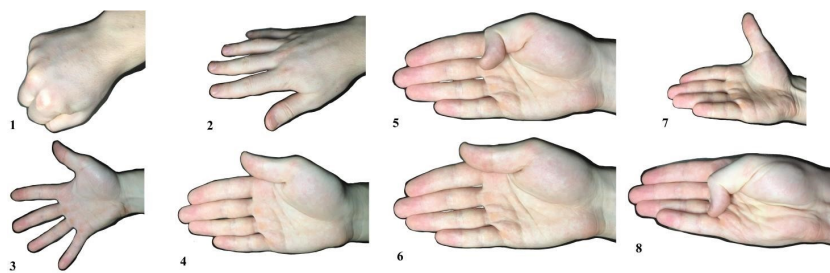


Рис. 1.2: Движения кисти. 1. Сгибание пальцев. 2. Разгибание пальцев. 3. Отведение пальцев. 4. Приведение пальцев. 5. Сгибание большого пальца. 6. Разгибание большого пальца. 7. Отведение большого пальца. 8. Противопоставление большого пальца

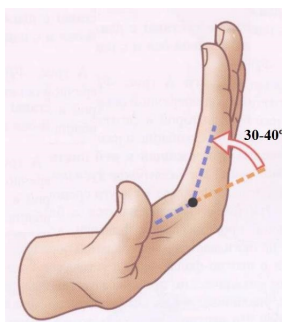


Рис. 1.3: Активное разгибание в пястно-фаланговом суставе

Простые движения состоят из:

- «Сгибания в лучезапястном суставе» или «Разгибания в лучезапястном суставе»;
- «Отведения в лучезапястном суставе» или «Приведения в лучезапястном 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., °	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	

Значения с «!» означают, что движение является возвращением к исходному положению. Значения с «+()» указывают на активное разгибание. Сокращения, используемые в таблице 1.2: Амп. – амплитуда; БП – Большой палец.

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, правил 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 резолювента, 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 выстраиваются 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 γ 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 не коммутативны 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 γ 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 $P_1(X_1, Y_1, Z_1)$, $P_2(X_2, Y_2, Z_2)$, $P_3(X_3, Y_3, Z_3)$ состоит в том что бы найти 2 вектора (Вектор $\vec{P_1P_2}$ и вектор $\vec{P_2P_3}$) что бы найти угол между ними. Чтобы найти эти 2 вектора мы воспользуемся формулой

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

Так как существует формула:

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

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

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

Конечная формула получается:

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

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:

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

2.2 Структура проверок положения точек

Входные данные в программе 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 составные термы.

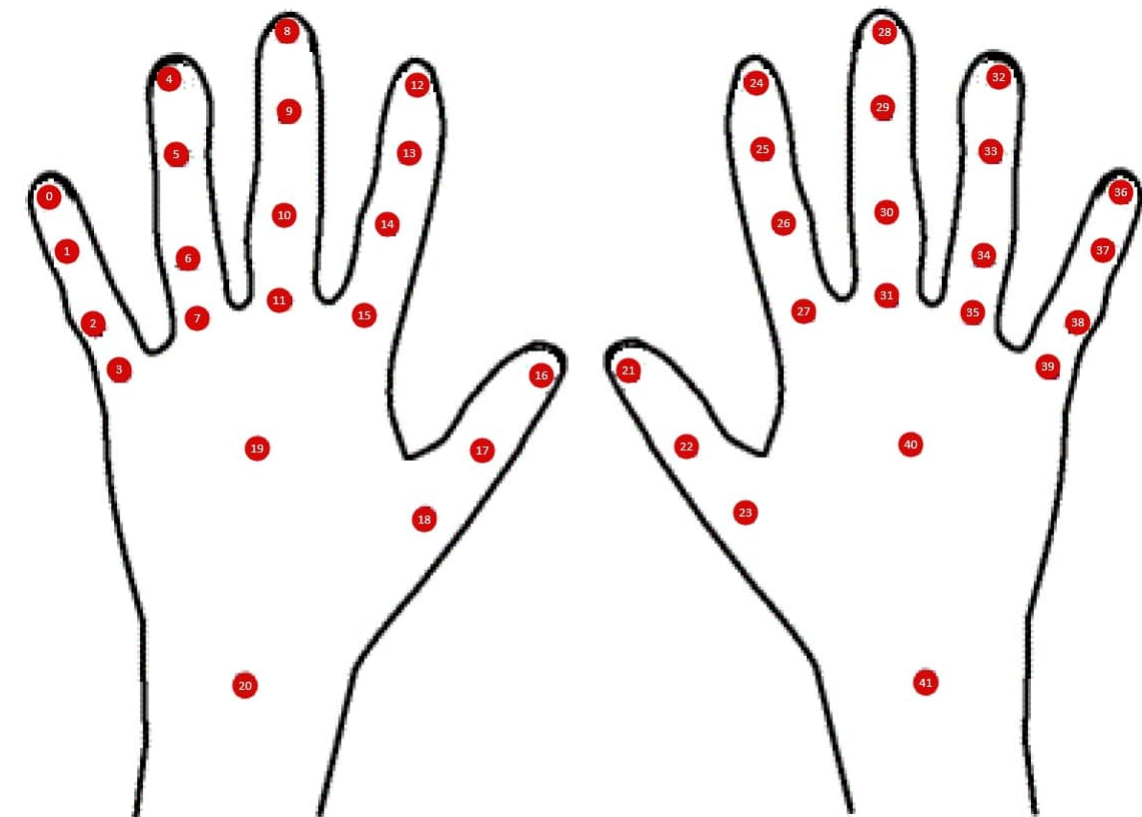


Рис. 2.1: Нумерация точек на кисти

Возьмем точки 0, 1, 2, 3. Вместе они составляют мизинец на руке. В соответствии с этим можно создать составной терм для 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 соответственно.

Далее полученные составные термы пальцев мы можем объединить в составной терм 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 правил 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 текстовый 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 составных термов для каждого из них:

Листинг 2.1: Листинг составных термов пальцев

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

где P – это номер точки на руке.

Для руки мы сделали составной терм:

Листинг 2.2: Листинг составного терма руки

```
1 Hand
2 (
3     finger(little, P0, P1, P2, P3),
4     finger(ring, P4, P5, P6, P7),
5     finger(middle, P8, P9, P10, P11),
6     finger(index, P12, P13, P14, P15),
7     finger(thumb, P16, P17, P18),
8     P19, P20
9 )
```

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 библиотеки 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.

OpenGL – API для разработки приложений, использующих 2D и 3D-графику. Эта библиотека позволяет нам 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 отображать 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.

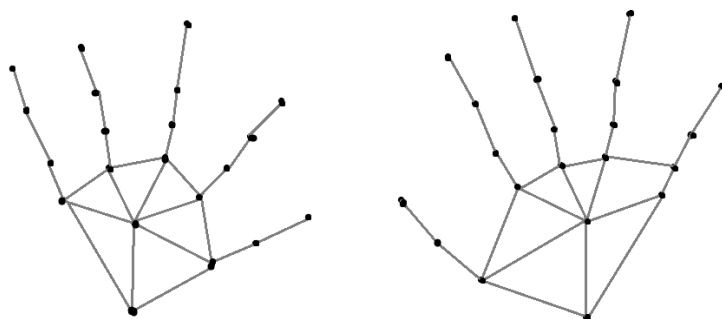


Рис. 2.2: Схематическое изображение желаемого результата рендеринга

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 был выбран в связи с наличием опыта работы с ним, что ускорило процесс разработки, а также в связи с наличием всех необходимых инструментов для отрисовки и создания GUI в нем;
- Prolog (SWI-Prolog(4)) для написания функций проверки точек на корректность.

3.2 Описание логики работы модуля программы на Prolog

Программе на Prolog задается вопрос, содержащий предикат `validate_all` (листинг 3.24) со всеми входными данными. В предикате `validate_all` точки собираются в составные термы `finger`, которые в свою очередь входят в составной терм `hand` (листинг 3.3). Для проверки точек отдельно взятой руки используется правило `validate_hand`, содержащее соответствующую структуру `hand`.

Правило `validate_hand` (листинг 3.20) возвращает истину, если выполняется правило `validate_finger` для каждого из пальцев.

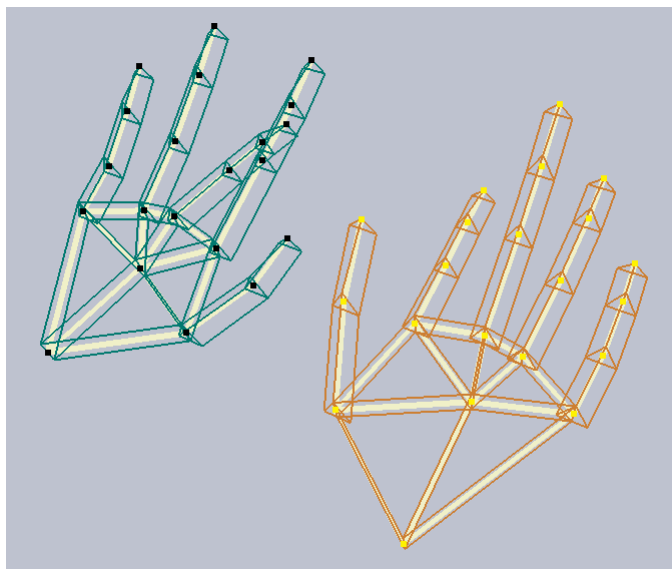
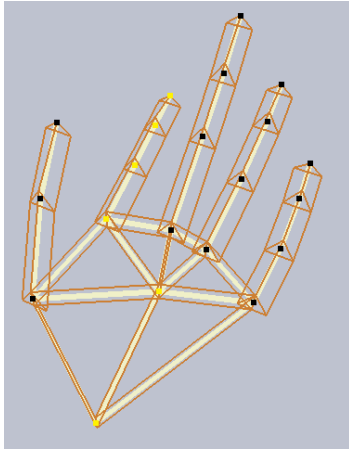
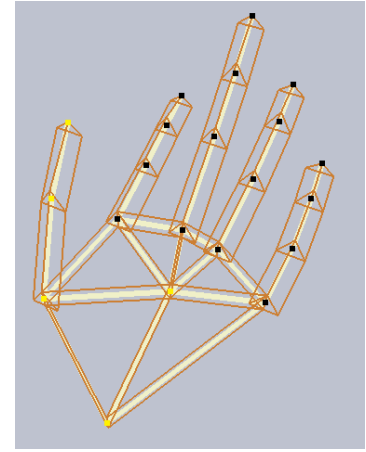


Рис. 3.1: Область работы `validate_hand` для правой руки

В правиле `validate_finger` (листинг 3.21) проверяется то, выполняются ли правила `validate_points` для различных комбинаций точек и углов заданного пальца. Проверяемые комбинации выбираются исходя из типов проверок, описанных в фактах `finger_motion_type` (листинг 3.1).



а)



б)

Рис. 3.2: Область работы validate_finger: а) для указательного, б) для большого.

В правиле validate_points (листинг 3.19) с помощью правила check_3coords (листинг 3.23) точки проверяются на существование (заданы ли значения координат числами). Если точки существуют, то они вместе с типом проверяемого угла передаются правилу validate_angle.

В правиле validate_angle (листинг 3.18) с помощью правила get_angle (листинг 3.16) вычисляется угол, образованный данными точками. Плоскость, в которой ведется вычисление, определяется из факта angle_info (листинг 3.2). Вычисленный угол проверяется на вхождение в допустимый диапазон правилом valid_angle (листинг 3.4). Если угол проходит проверку, то правило validate_angle возвращает истину.

3.3 Описание структуры базы знаний

3.3.1 Описание фактов

В данном разделе в листингах 3.1 - 3.2 описаны все факты и их назначения.

finger_motion_type

Каждый палец (кроме большого) определяется 4 точками (3 точки для большого пальца), следовательно необходимо проверять 3 различных угла при сгибе пальцев (2 для большого), а также угол отклонения пальца при отведении пальца. Таким образом, каждый палец имеет 4 уникальных вида углов для проверки: один вид при отведении и три - при сгибе.

- FingerType - тип пальца;
- AbductionType - тип углов приведения пальца (для большого пальца или других пальцев);
- Flexion1, Flexion2, Flexion3 - типы углов сгибов пальца.

Листинг 3.1: Знания о типах проверок каждого пальца

```

1 | %finger_motion_type(FingerType, AbductionType, Flexion1, Flexion2, Flexion3).
2 |
3 | finger_motion_type(thumb, bppsrib1, bppsrib2, bppsrib3).
4 | finger_motion_type(index, oprived, o2srib1, o2srib2, o2srib3).
5 | finger_motion_type(middle, oprived, o3srib1, o3srib2, o3srib3).
6 | finger_motion_type(ring, oprived, o4srib1, o4srib2, o4srib3).
7 | finger_motion_type(little, oprived, o5srib1, o5srib2, o5srib3).
```

Каждый из проверяемых углов продемонстрированы на рисунках 3.3 - 3.5. Для демонстрации был взят только указательный и большой пальцы, так как углы, образуемые точками на пальце, у указательного, среднего, безымянного и мизинца похожи, следовательно достаточно показать только один палец.

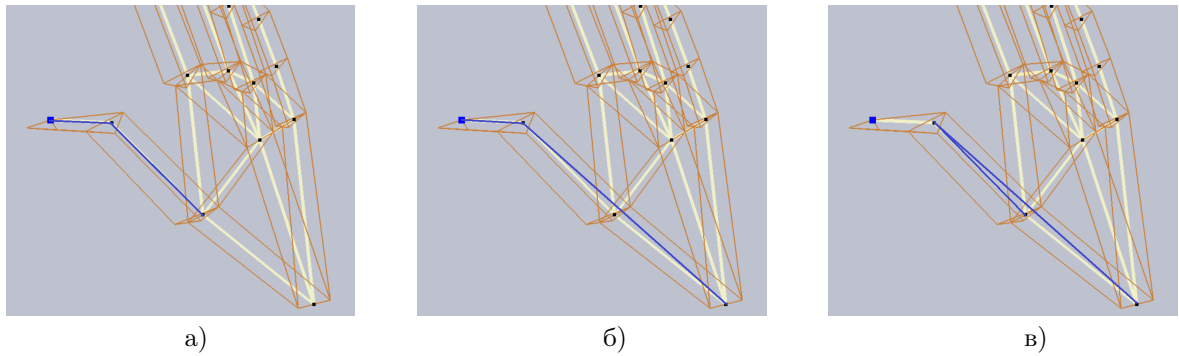


Рис. 3.3: Проверяемые углы на большом пальце: а) Flex1, б) Flex2, в) Abduction.

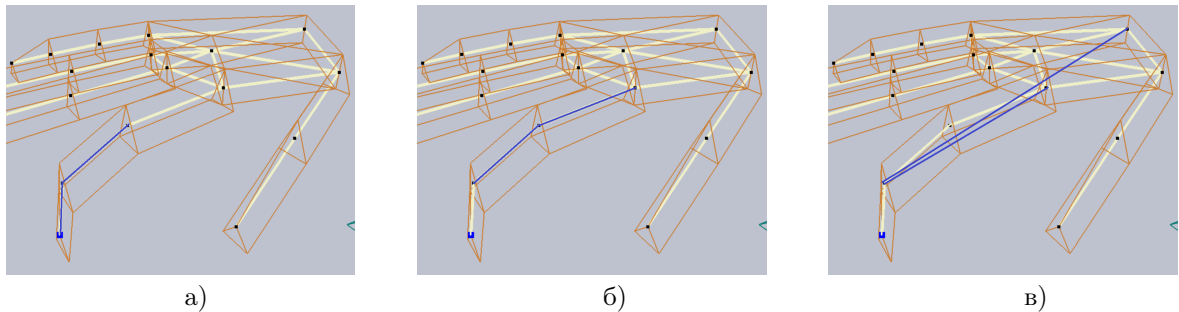


Рис. 3.4: Проверяемые углы на указательном пальце: а) oabc, б) obcd, в) Abduction.

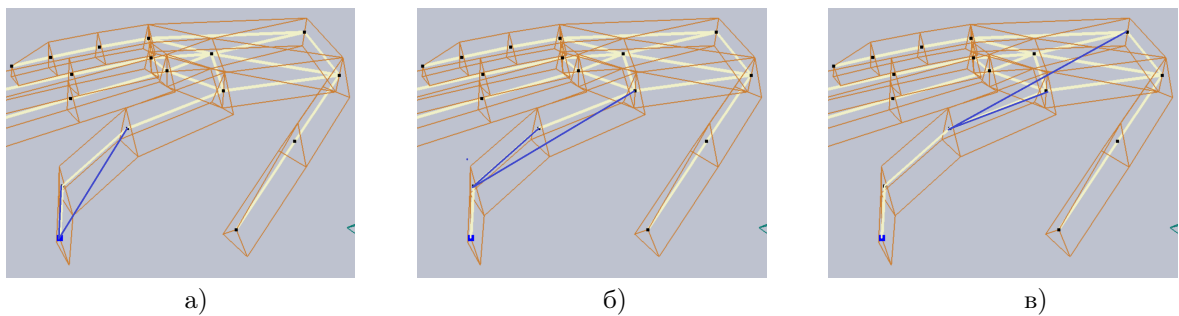


Рис. 3.5: Проверяемые углы на указательном пальце: а) Flex1, б) Flex2, в) Flex3.

angle_info

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;
- Axis - обозначает плоскость, в которой ищем угол.

Листинг 3.2: Знания об амплитудах углов и проверяемых осях

```
1| %info about angle limits in degrees for fingers and axis
```

```

2 %angle_info(Finger, MinAngle, MaxAngle, Axis)
3
4 angle_info(bpabc, -80, 80, all).
5 angle_info(bpbcd, -50, 50, all).
6 angle_info(bpcde, -90, 90, all).
7 angle_info(oabc, -80, 80, all).
8 angle_info(obcd, -100, 100, all).
9 angle_info(ocde, -90, 90, all).
10 angle_info(between, -30, 30, all).
11
12 angle_info(bpprived, -50, 50, x).
13 angle_info(oprived, -60, 60, x).
14 angle_info(bppsgib1, -50, 50, y).
15 angle_info(bppsgib2, -100, 80, y).
16
17 angle_info(o2sgib1, -120, 90, x).
18 angle_info(o2sgib2, -100, 100, x).
19 angle_info(o2sgib3, -100, 100, x).
20
21 angle_info(o3sgib1, -120, 90, x).
22 angle_info(o3sgib2, -100, 100, x).
23 angle_info(o3sgib3, -80, 80, x).
24
25 angle_info(o4sgib1, -120, 90, x).
26 angle_info(o4sgib2, -100, 100, x).
27 angle_info(o4sgib3, -80, 80, x).
28
29 angle_info(o5sgib1, -120, 90, x).
30 angle_info(o5sgib2, -100, 100, x).
31 angle_info(o5sgib3, -80, 80, x).
32
33 angle_info(bppz, -100, 100, z).
34 angle_info(middle_abduction_deduction, -10, 10, y).

```

3.3.2 Описание структур

Программой на Prolog для описания руки используются структуры Рука, Палец и Точка, описанные в листинге 3.3.

- point - структура Точка, состоит из трех координат;
- hand - структура Рука, из 5 Finger и двух опорных точек кисти;
- finger - структура Палец, состоит из 4 точек (3-х для большого пальца).

Листинг 3.3: Структуры

```

1 point(X, Y, Z).
2
3 hand(
4     finger(little, P0, P1, P2, P3),           %5|finger V
5     finger(ring, P4, P5, P6, P7),             %4|finger IV
6     finger(middle, P8, P9, P10, P11),         %3|finger III
7     finger(index, P12, P13, P14, P15),       %2|finger II
8     finger(thumb, P16, P17, P18),            %1|finger I
9     P19, P20
10 ).

```

3.3.3 Описание правил

В данном разделе в листингах 3.4 - 3.32 описаны все правила, их назначение и использованные в них переменные, а также примеры использования.

valid_angle

`valid_angle` - правило, с помощью которого определяется, входит ли данный угол в допустимый диапазон `[MinAngle;MaxAngle]`. Значения `MinAngle` и `MaxAngle` зависят от типа проверяемого угла.

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

Листинг 3.4: Реализация правила `valid_angle`

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

Листинг 3.5: Пример использования правила `valid_angle`

```
1 | ?- valid_angle(bpabc, 40).  
2 | true.
```

vec_length

`vec_length` - правило, которое определяет длину вектора.

- `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).
```

Листинг 3.7: Пример использования правила `vec_length`

```
1 | ?- vec_length(vector(2, 3, 4), Len).  
2 | Len = 5.385164807134504.
```

vec_length_sqr

`vec_length_sqr` - правило, которое определяет квадрат длины вектора. Данное правило необходимо для избежания потери точности при вычислении корня.

- `vector(X,Y,Z)` - структура «вектор», определяемая тремя координатами;
- `Len` - квадрат длины вектора.

Листинг 3.8: Реализация правила `vec_length_sqr`

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

Листинг 3.9: Пример использования правила `vec_length_sqr`

```
1 | ?- vec_length_sqr(vector(2, 3, 4), Len).
2 | Len = 29.
```

`dot_prod`

`dot_prod` - правило, которое считает скалярное произведение двух векторов.

- `vector(X,Y,Z)` - составной терм «вектор», определяемый тремя координатами;
- `DotProd` - результат скалярного произведения векторов.

Листинг 3.10: Реализация правила `dot_prod`

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

Листинг 3.11: Пример использования правила `dot_prod`

```
1 | ?- dot_prod(vector(2, 1, 1), vector(0, 1, 1), DotProd).
2 | DotProd = 2.
```

`rad_to_deg`

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

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

Листинг 3.12: Реализация правила `rad_to_deg`

```
1 | 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 угла в радианах;
- `Degrees` - величина угла в градусах.

Листинг 3.13: Реализация правила `deg_to_rad`

```
1 | 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.

- `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 3.14: Реализация правила angle_between_vectors

```

1 angle_between_vectors(Vector1, Vector2, Angle) :-
2     vec_length_sqr(Vector1, Len1Sqr),
3     vec_length_sqr(Vector2, Len2Sqr),
4     dot_prod(Vector1, Vector2, DotProd),
5     AngleRad is acos(DotProd / sqrt(Len1Sqr * Len2Sqr)),
6     rad_to_deg(AngleRad, Angle).

```

Листинг 3.15: Пример использования правила angle_between_vectors

```

1 ?- angle_between_vectors(vector(2, 1, 1), vector(0, 1, 1), Angle).
2 Angle = 54.737224657778654.

```

get_angle

get_angle - правило, которое по заданным точкам в пространстве определяет угол между векторами, которые они образуют. Вычисляется угол между векторами $P_1\vec{P}_2$ и $P_2\vec{P}_3$.

- 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 точками.

Листинг 3.16: Реализация правила get_angle

```

1 get_angle(all, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3, Z3), Angle) :-
2     AX is X2 - X1, AY is Y2 - Y1, AZ is Z2 - Z1,
3     BX is X3 - X2, BY is Y3 - Y2, BZ is Z3 - Z2,
4     angle_between_vectors(vector(AX, AY, AZ), vector(BX, BY, BZ), Angle).
5
6 get_angle(x, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3, Z3), Angle) :-
7     AY is Y1 - Y2, AZ is Z1 - Z2,
8     BY is Y3 - Y2, BZ is Z3 - Z2,
9     angle_between_vectors(vector(0, AY, AZ), vector(0, BY, BZ), Angle).
10
11 get_angle(y, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3, Z3), Angle) :-
12     AX is X2 - X1, AZ is Z2 - Z1,
13     BX is X3 - X2, BZ is Z3 - Z2,
14     angle_between_vectors(vector(AX, 0, AZ), vector(BX, 0, BZ), Angle).
15
16 get_angle(z, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3, Z3), Angle) :-
17     AX is X2 - X1, AY is Y2 - Y1,
18     BX is X3 - X2, BY is Y3 - Y2,
19     angle_between_vectors(vector(AX, AY, 0), vector(BX, BY, 0), Angle).

```

Листинг 3.17: Пример использования правила get_angle

```

1 ?- get_angle(all, point(0, 0, 0), point(2, 2, 2), point(5, 6, 0), Angle).
2 Angle = 57.58612510399082.

```


validate_angle

validate_angle - правило, которое проверяет, является ли значение угла между векторами по заданным трём точкам допустимым для заданного типа угла.

- Type - тип проверяемого угла;
- Point1, Point2, Point3 - структуры точек, заданных тремя координатами.

Листинг 3.18: Реализация правила validate_angle

```
1 validate_angle(Type, Point1, Point2, Point3) :-  
2     hand:angle_info(Type, _, _, Axis),  
3     get_angle(Axis, Point1, Point2, Point3, Angle),  
4     write_files:write_angle(Type, Angle),  
5     hand:valid_angle(Type, Angle).
```

validate_points

validate_points - правило, которое проверяет корректность координат трёх точек по заданному типу угла. Состоит из трёх правил. Первое проверяет наличие и корректность координат, второе проверяет на корректность угол между тремя точками, а третье правило выводит сообщение о том, что точки не подходят, если угол не допустим.

- 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 3.19: Реализация правила validate_points

```
1 validate_points(Type, [X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]) :-  
2     not(check_3coords([X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3])),  
3     write_files:write_invalid_data().  
4  
5 validate_points(Type, [X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]) :-  
6     check_3coords([X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]),  
7     validate_angle(Type, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3, Z3)),  
8     write_files:write_angle_is_valid(Type).  
9  
10 validate_points(Type, [X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]) :-  
11     check_3coords([X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]),  
12     not(validate_angle(Type, point(X1, Y1, Z1), point(X2, Y2, Z2), point(X3, Y3,  
13         Z3))),  
13     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 - составной терм «Рука», который состоит из 5 составных термов типа Finger, которые состоят из 4 точек для указательного, среднего, безымянного и мизинца, из 3-х точек для большого пальца, а также оставшихся двух ключевых точек на ладони: пясть и запястье;
- finger - составной терм пальца, состоящий из 3 или 4 точек;
- Wrist - точка на запястье;
- p19 - точка на пясти.

Листинг 3.20: Реализация правила validate_hand

```
1 validate_hand(hand:hand(Finger5, Finger4, Finger3, Finger2, Finger1, P19, Wrist)):-
2     validate_finger(Finger5, P19, Wrist),
3     validate_finger(Finger4, P19, Wrist),
4     validate_finger(Finger3, P19, Wrist),
5     validate_finger(Finger2, P19, Wrist),
6     validate_finger(Finger1, P19, Wrist).
```

validate_finger

validate_finger - правило, которое проверяет корректность точек на пальце.

- finger - составной терм пальца, который состоит из вида пальца (большой/безымянный/средний и т. д.) и точек-концов фаланг;
- Wrist - точка на запястье;
- p19 - точка на пясти;
- finger_motion_type - знание об амплитуде угла для конкретного типа пальца.

Листинг 3.21: Реализация правила validate_finger

```
1 validate_finger(finger(thumb, P1, P2, P3), P19, Wrist):-
2     finger_motion_type(thumb, Abduction, Flex1, Flex2, _),
3     validate_points(bpabc, P1, P2, P3),
4     validate_points(Abduction, P3, P2, Wrist),
5     validate_points(Flex1, P1, P2, P3),
6     validate_points(Flex2, P1, P2, Wrist),
7     validate_points(bppz, P1, P2, P3).
8
9 validate_finger(finger(Finger, P1, P2, P3, P4), P19, Wrist):-
10     not(Finger == thumb),
11     finger_motion_type(Finger, Abduction, Flex1, Flex2, Flex3),
12     validate_points(oabc, P1, P2, P3),
13     validate_points(obcd, P2, P3, P4),
14     validate_points(Abduction, P4, P2, Wrist),
15     validate_points(Flex1, P2, P1, P3),
16     validate_points(Flex2, P4, P2, P3),
17     validate_points(Flex3, P4, P3, Wrist),
18     validate_points(bppz, P1, P2, P3).
```

check_coords

check_coords - правило, которое проверяет наличие и валидность данных о координатах.

X, Y, Z - список координат.

Листинг 3.22: Реализация правила check_coords

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

check_3coords

check_3coords - правило, которое проверяет наличие и валидность данных о координатах трёх точек.

X1, Y1, Z1 , [X2, Y2, Z2], [X3, Y3, Z3] - списки координат.

Листинг 3.23: Реализация правила check_3coords

```

1 check_3coords([X1, Y1, Z1], [X2, Y2, Z2], [X3, Y3, Z3]) :-
2     check_coords([X1, Y1, Z1]),
3     check_coords([X2, Y2, Z2]),
4     check_coords([X3, Y3, Z3]).

```

validate_all

validate_all - правило, определяющее корректность точек двух рук. «Входное» правило в программе.

- Working_Dir - путь к рабочей директории с файлом с координатами точек;
- Result - результат работы правила;
- Point1 - составной терм «Точка», состоящий из трех координат.

Листинг 3.24: Реализация правила validate_all

```

1 validate_all(Working_Dir, Result,
2     Point1, Point2, Point3, Point4, Point5, Point6, Point7,
3     Point8, Point9, Point10, Point11, Point12, Point13, Point14,
4     Point15, Point16, Point17, Point18, Point19, Point20, Point21,
5     Point22, Point23, Point24, Point25, Point26, Point27, Point28,
6     Point29, Point30, Point31, Point32, Point33, Point34, Point35,
7     Point36, Point37, Point38, Point39, Point40, Point41, Point42
8 ) :-
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,
21                        Point12),
22                    finger(index, Point13, Point14, Point15,
23                        Point16),
24                    finger(thumb, Point17, Point18, Point19),
25                    Point20, Point21
26                ),
27            ),
28            validate_hand(
29                hand(hand(
30                    finger(little, Point37, Point38, Point39,
31                        Point40),
32                    finger(ring, Point33, Point34, Point35, Point36),
33                    finger(middle, Point29, Point30, Point31,
34                        Point32),
35                    finger(index, Point25, Point26, Point27,
36                        Point28),
37                    finger(thumb, Point22, Point23, Point24),
38                    Point41, Point42
39                ),
40            )
41        )-> Result = "Ok"; Result = "Not "
42    ).

```

write_invalid_data

`write_invalid_data` - правило, которое выводит сообщение о некорректности данных.

Листинг 3.25: 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 3.26: 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 3.27: Реализация правила `write_angle`

```
1 | 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 3.28: Реализация правила `write_angle_is_valid`

```
1 | write_angle_is_valid(Type) :-  
2 |     open('angles.txt', append, Stream2),  
3 |     string_concat(Type, " is ok", Msg),  
4 |     write(Stream2, Msg), nl(Stream2),  
5 |     close(Stream2).
```

write_invalid_points

`write_invalid_points` - правило, которое записывает некорректные точки в файл.

- Point - составной терм точки, состоит из трех координат.

Листинг 3.29: Реализация правила `write_invalid_points`

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

point_to_str

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

- Point - составной терм «Точка», состоящий из трех координат;
- Str - строковое представление точки.

Листинг 3.30: Реализация правила point_to_str

```
1 point_to_str(Point, Str) :-  
2     determ:get_coords(Point, X, Y, Z),  
3     number_string(X, StrX),  
4     number_string(Y, StrY),  
5     number_string(Z, StrZ),  
6     string_concat(StrX, ";", StrX1),  
7     string_concat(StrY, ";", StrY1),  
8     string_concat(StrX1, StrY1, StrXY),  
9     string_concat(StrXY, StrZ, StrXYZ),  
10    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 3.31: Реализация правила write_list

```
1 write_list(Stream, []).  
2 write_list(Stream, [Head|Tail]) :-  
3     write(Stream, Head),  
4     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 3.32: Реализация правила write_points

```
1 write_points(PList,Filename) :-  
2     open(Filename, write, Stream),  
3     convlist(point_to_str, PList, StrList),  
4     write_list(Stream, StrList),  
5     close(Stream).
```

3.4 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 консультировать выбранные файлы. В листинге 3.33 продемонстрированы две функции, использующиеся для взаимодействия.

Листинг 3.33: Взаимодействие python и Prolog

```
1 from pyswip import Prolog
2 import os
3
4 def append_base_stored(prolog, filename, filepath=None):
5     if filepath is None:
6         filepath = ''
7     fullpath = os.path.join(filepath, filename)
8     prolog.consult(fullpath)
9
10 def get_answer(basestored, statement, filepath=None):
11     prolog = Prolog()
12     append_base_stored(prolog=prolog,
13                        filename=basestored,
14                        filepath=filepath)
15     return prolog.query(statement)
```

append_base_stored - функция, которая по 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.

3.5 Интерфейс программы

Для удобства создания GUI для работы с кистями была выбрана библиотека tkinter с надстройками, которая позволяет рисовать 3D изображения. Также с помощью нее можно создать интерфейс взаимодействия с точками кисти, чтобы пользователь мог их изменять, загружать 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.

3.6 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

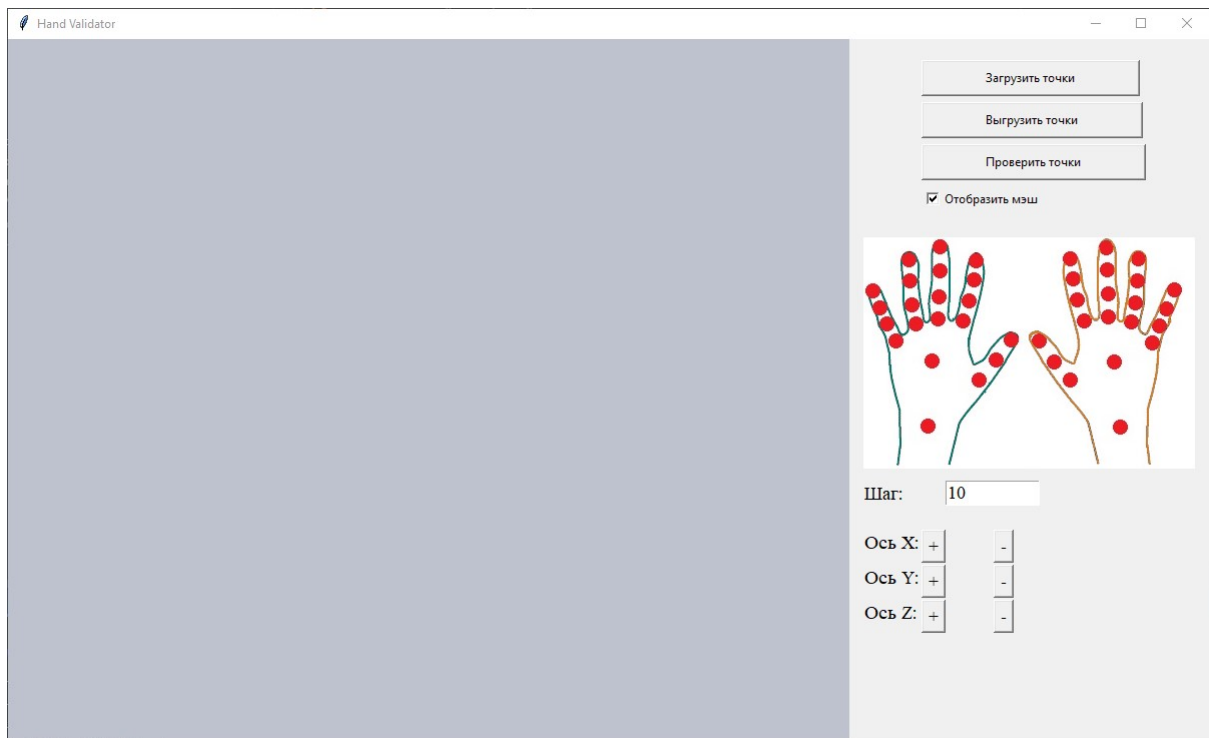


Рис. 3.6: Интерфейс программы

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 визуальнo было проще понимать, какая рука где находится, а также определять, куда "смотрит" ладонь, а где ее тыльная сторона, было решено добавить модель кисти.

Относительно каждой группы точек рисуется геометрическая фигура, отдаленно напоминающая ту часть кисти, которой соответствует. Каждая фаланга представляется призмой с основанием в форме треугольника, причём окончание пальца имеет треугольник меньшего размера.

- class PhalanxModel - класс, определяющий структуру данных точек фаланги пальца и порядок её отображения;
- class EndPhalanxModel - класс, определяющий структуру данных точек конечной фаланги пальца и порядок её отображения, является наследником класса PhalanxModel, отличается от базового класса меньшим размером конечного треугольника-основания;
- class FingerModel - класс, определяющий структуру данных точек пальца и порядок его отображения;
- class HandModel - класс, определяющий структуру данных точек руки и порядок её отображения;
- class RightHandModel - класс, определяющий структуру данных точек правой руки и порядок отображения её, является наследником класса HandModel, отличается от базового класса другой последовательностью подаваемых координат для составления структуры.

Перемещать камеру возможно с помощью стрелок на клавиатуре, масштабирование осуществляется с помощью кнопок '+' и '-'.

3.7 Примеры работы

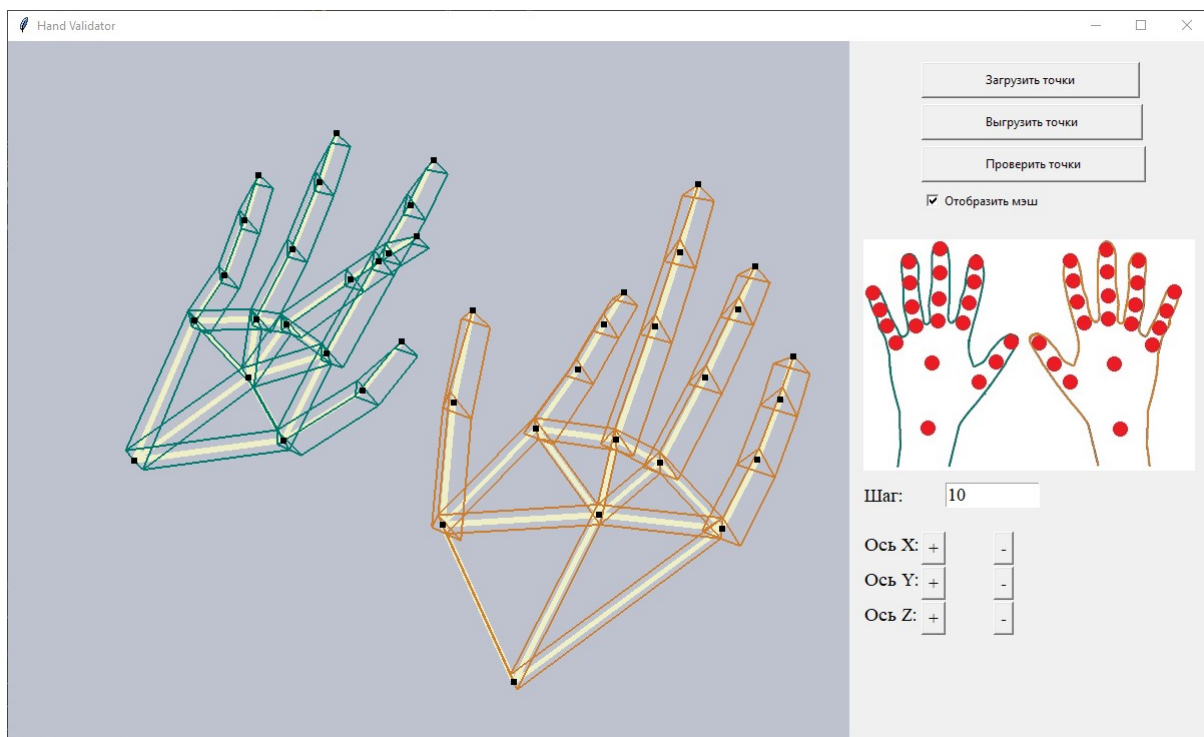


Рис. 3.7: Пример работы с корректным расположением точек

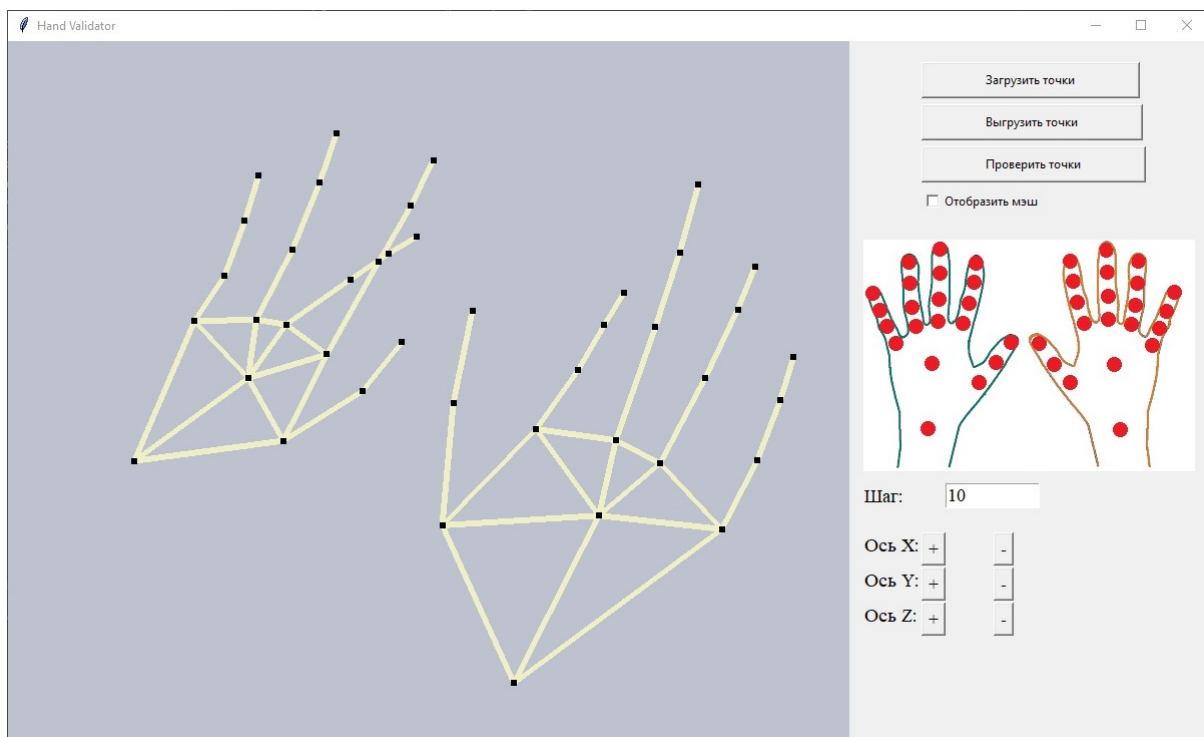


Рис. 3.8: Пример работы только с каркасом

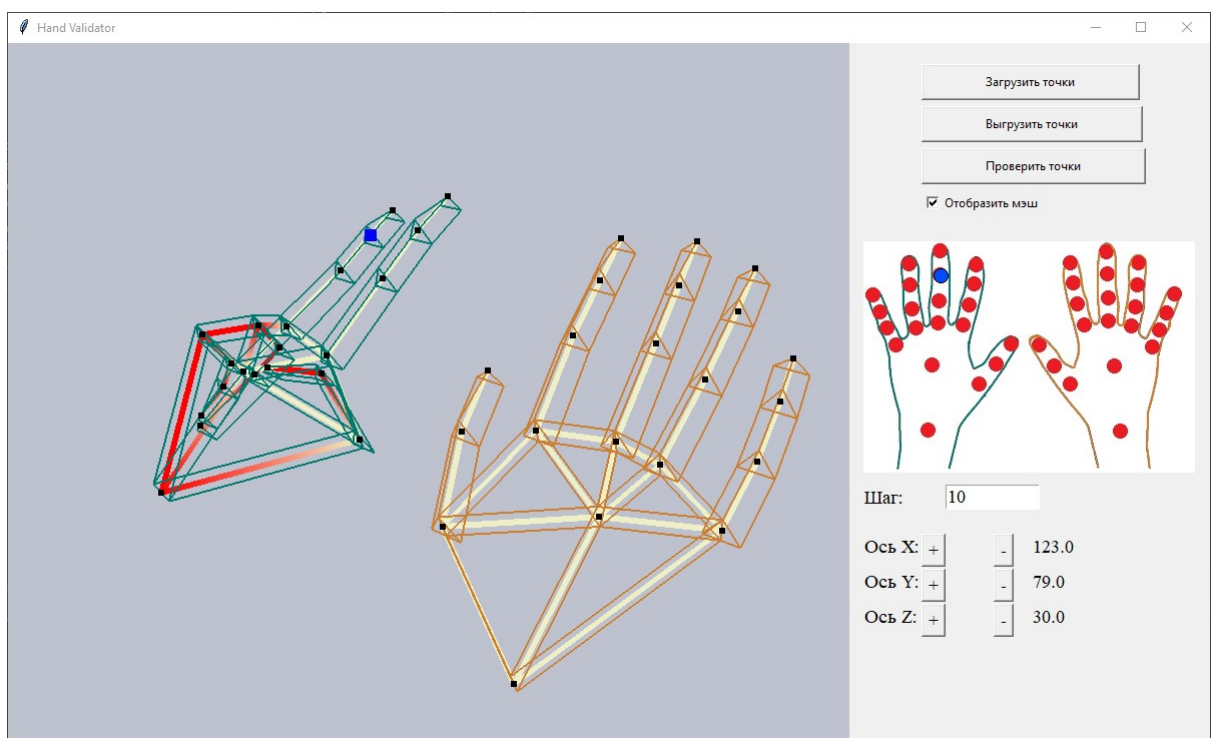


Рис. 3.9: Пример работы с некорректным расположением точек

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.)
- [5] Идрисов А. С. Дипломная работа на тему: Разработка аппаратно-программного комплекса для моделирования движений рук хирурга при проведении эндоваскулярных вмешательств Москва 2020г.