Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Лабораторная работа №3
по дисциплине «Имитационное моделирование робототехнических систем»

Выполнил:
студент гр. R41341c
Борисов М. В.

Преподаватель:
Бжихатлов И. А.

Санкт-Петербург
2021 г.

Цель работы

- Импортировать модель манипулятора в симулятор
- Собрать визуальную и физическую части манипулятора
- Написать скрипт для поиска зелёного куба в рабочем пространстве
- Реализовать касание манипулятором до куба при обнаружении

Ход работы

Воспользуемся моделью манипулятора доступной по ссылке https://grabcad.com/library/r
arm-manipulator-fanuc-m-20ib-industrial-robot-1 и изображённом на рисунке 1.

Рис. 1: Рендер манипулятора

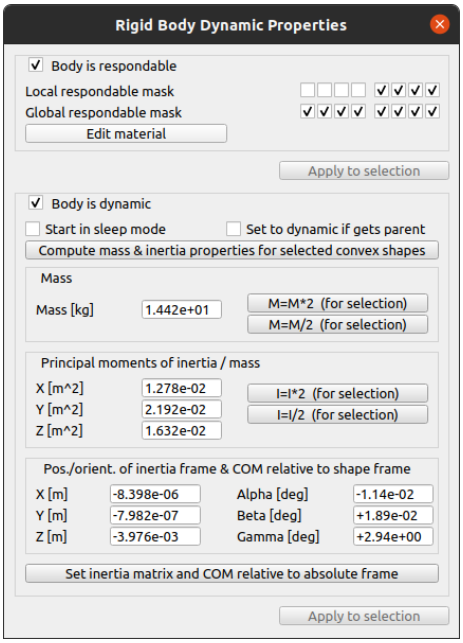
Для уменьшения потребления ресурсов на рендер уменьшим число полигонов модели с помощью опции `Edit → Decimate selected shape`. Покрасим отдельные детали в различные цвета для удобства, после чего сделаем ещё более простые модели деталей. Эти детали будут служить физической репрезентацией манипулятора и на их основе будут считаться коллизии и взаимодействия.

Базу робота заменим обычным цилиндром, остальные детали заменим их выпуклой оболочкой. Используя редактор вершин и мешей разметим места для установки осей вращения с помощью компонента `Dummy`. По ним в дальнейшем будет проще ориентироваться.

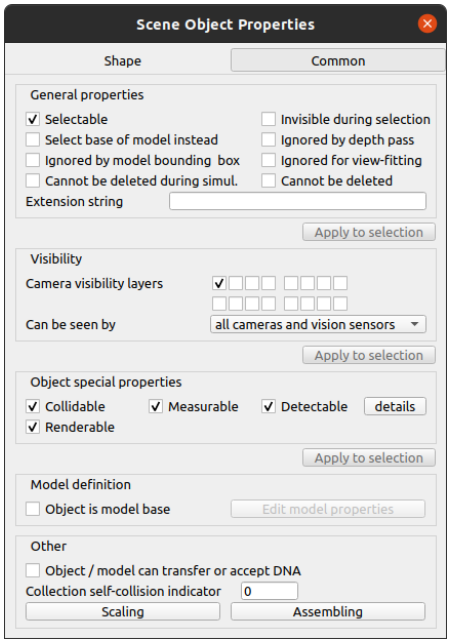
Также с помощью этого компонента отметим конец последнего звена манипулятора, куда обычно крепятся инструменты. В это место мы прикрепим датчики.

Все детали манипулятора необходимо называть шаблонно (рисунок 3b), в коде это упростит работу.

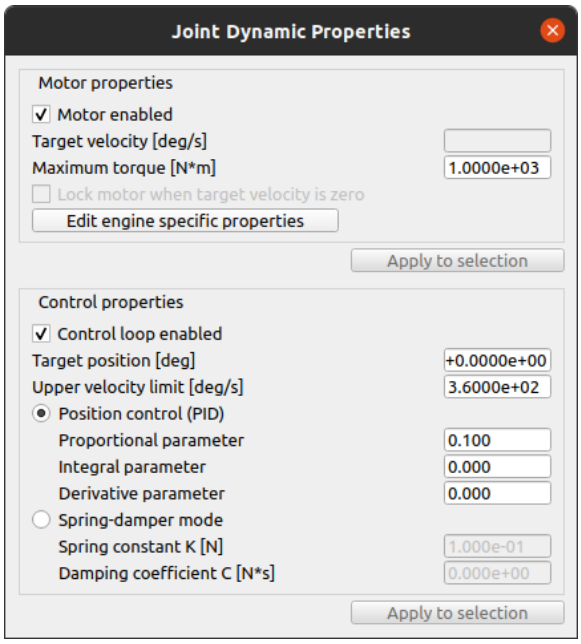
В конечном итоге получим модель манипулятора и соответствующее ей дерево (рисунок 3).



(a) Настройки динамических компонентов



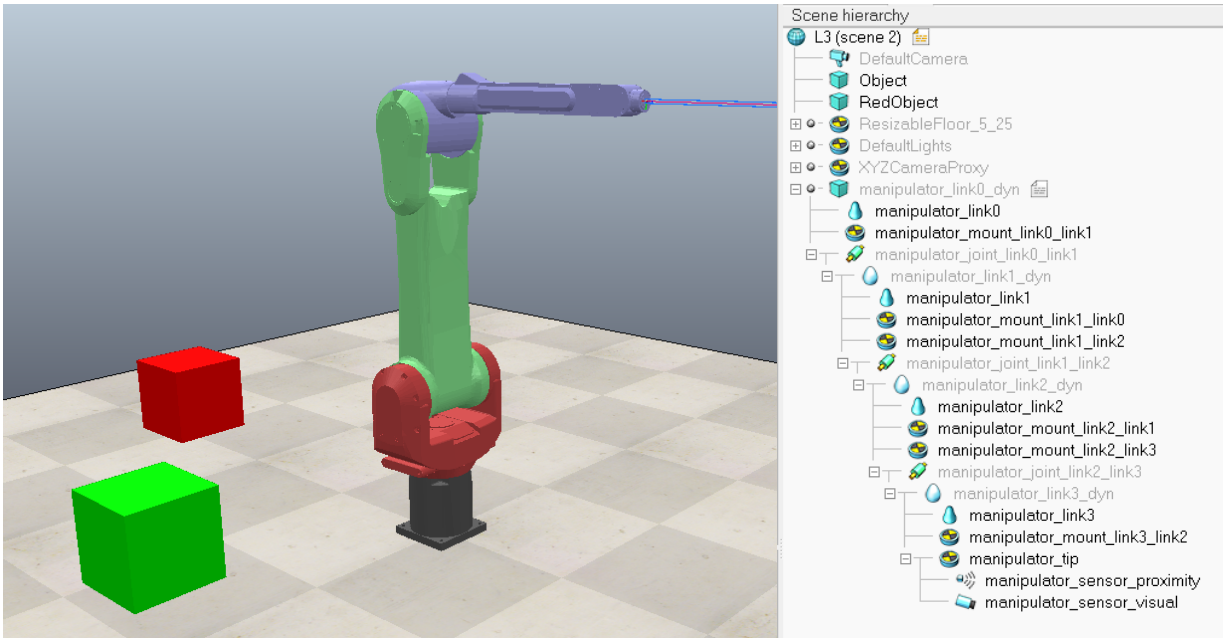
(b) Настройки объекта измерения (куба)



(c) Настройки сочленений

Рис. 2: Настройки тел и сочленений

На рисунке 2 показаны настройки для различных тел модели. При том для динамических тел, которые участвуют в расчётах важно чередование локальной



(a) Собранный манипулятор (b) Дерево модели манипулятора

Рис. 3: Модель

маски соседних элементов для избежания ложных коллизий. Для элементов визуализации галочки динамики и взаимодействия сняты, поскольку в расчёте они не участвуют.

Скрипт напишем с использованием remote API на python. Код целиком приведён в приложении А. Алгоритм следующий:

1. Начать вращение вокруг своей оси (первое сочленение)
2. Совершать осциллирующее движение последним звеном (сканировать пространство)
3. Как только обнаружен зелёный куб остановиться
4. Медленно приблизиться и коснуться куба
5. Вернуться в исходное положение

У данного алгоритма есть множество недостатков, в частности для ”честного” касания куба необходимо решать обратную задачу кинематики. Также для ”честного” определения зелёного предмета желательно переводить получаемый цвет в HSV пространство, которое даёт более надёжные измерения цвета. Однако в рамкой лабораторной работы такого алгоритма достаточно.

Для каждого этапа алгоритма в скрипте прописана соответствующая функция, что позволяет при необходимости легко модифицировать и улучшить код (например, для последующих лабораторных работ).

Вывод

В работе изучен алгоритм импорта сторонних файлов и создания из них моделей для симуляции. Изучена работа с новым визуальным сенсором. Разработан простой алгоритм взаимодействия с окружением и рассмотрены его недостатки.

A. RemoteAPI скрипт

```
from typing import Any, List, NewType, Optional
from math import pi, isclose

import vrepapi.sim as vrepapi

from L2.src.main import VREPClient

VREPHandle = NewType('VREPHandle', Any)


class L3Controller(VREPClient):
    __basename = 'manipulator'
    __sep = '_'
    __link_basename = 'link'
    __joint_basename = 'joint'
    __sensor_basename = 'sensor'

    __n_links = 4

    __base_handle: VREPHandle
    __links_handles: List[VREPHandle]
    __joint_handles: List[VREPHandle]
    __tip_handle: VREPHandle

    __prox_sensor_handle: VREPHandle
    __vis_sensor_handle: VREPHandle

    __D_EPS: float = 1e-2
    __ANGLE_EPS: float = 0.1
    __G_THRESH: float = 0.9

    __joint_ranges: List[float] = [
            [None, None],
            [-pi/3, pi/2],
            [0, pi/3]
            ]

    __joint_dp: List[float] = [
```

```
            .08,
            .08,
            .08
            ]

    @classmethod
    def __construct_name(cls, *args) -> str:
        return cls.__sep.join([cls.__basename, *args])

    @classmethod
    def get_base_name(cls) -> str:
        return cls.get_link_name(0)

    @classmethod
    def get_link_name(cls, n: int) -> str:
        return cls.__construct_name(f'{cls.__link_basename}{n}')

    @classmethod
    def get_joint_name(cls, n: int) -> str:
        """
        :param n: index of the link that is controlled by the joint
        :return: name of the joint
        """
        links = [f'{cls.__link_basename}{_}' for _ in (n, n + 1)]
        return cls.__construct_name(cls.__joint_basename, *links)

    @classmethod
    def get_tip_name(cls):
        return cls.__construct_name('tip')

    @classmethod
    def get_sensor_name(cls, type: str) -> str:
        return cls.__construct_name(cls.__sensor_basename, type)

    def __init_handles(self):
        self.__base_handle, *self.__links_handles = [
                vrepapi.simxGetObjectHandle(self.client_id,
                 ↪  self.get_link_name(_),
                 ↪  vrepapi.simx_opmode_blocking)[1]
                for _ in range(self.__n_links)]
        self.__joint_handles = [
                vrepapi.simxGetObjectHandle(self.client_id,
                 ↪  self.get_joint_name(_),
                 ↪  vrepapi.simx_opmode_blocking)[1] for
                _ in range(0, self.__n_links - 1)]
        _, self.__tip_handle = vrepapi.simxGetObjectHandle(
                self.client_id, self.get_tip_name(),
                 ↪  vrepapi.simx_opmode_blocking
```

```
82                  )
83          _, self.__prox_sensor_handle = vrepapi.simxGetObjectHandle(
84                  self.client_id, self.get_sensor_name('proximity'),
                     ↪  vrepapi.simx_opmode_blocking
85                  )
86          _, self.__vis_sensor_handle = vrepapi.simxGetObjectHandle(
87                  self.client_id, self.get_sensor_name('visual'),
                     ↪  vrepapi.simx_opmode_blocking
88                  )
89
90      def __init_datastreams(self):
91          vrepapi.simxReadProximitySensor(self.client_id,
             ↪  self.__prox_sensor_handle,
             ↪  vrepapi.simx_opmode_streaming)
92          vrepapi.simxReadVisionSensor(self.client_id,
             ↪  self.__vis_sensor_handle, vrepapi.simx_opmode_streaming)
93
94      def __init(self):
95          self.__init_handles()
96          self.__init_datastreams()
97
98      @property
99      def distance(self) -> Optional[float]:
100         print(f'Reading from proximity sensor...')
101
102         res, ready, (_, _, d), *_ = \
103             vrepapi.simxReadProximitySensor(
104                     self.client_id,
105                     self.__prox_sensor_handle,
106                     vrepapi.simx_opmode_buffer
107                     )
108
109         print(f'Readings:\n'
110               f'  response: {res}; ready: {ready}; distance: {d}\n')
111
112         return d if self.response_good(res) and ready else None
113
114     @property
115     def greenness(self):
116         print(f'Reading from visual sensor...')
117
118         res, ready, packets =
             ↪  vrepapi.simxReadVisionSensor(self.client_id,
             ↪  self.__vis_sensor_handle,
119                                             vrepapi.simx_opmode_buffer)
120
121         greenness = packets[0][12] if self.response_good(res) and
             ↪  packets else None  # get average green channel
```

```
122
123             print(f'Readings:\n'
124                   f'  response: {res}; ready: {ready}; greenness:
                      ↪  {packets}\n')
125
126             return greenness
127
128         @property
129         def object_found(self):
130             return self.greenness and self.greenness >= self.__G_THRESH
131
132         def __move_to_next_point(self):
133             self.move_joint(self.__joint_handles[0], self.__joint_dp[0])
134             self.set_joint_pos(self.__joint_handles[1], pi/4)
135             min_, max_ = self.__joint_ranges[-1]
136             if abs((cur_pos :=
                      ↪  self.get_joint_pos(self.__joint_handles[-1])) - max_) <=
                      ↪  self.__ANGLE_EPS:
137                 self.__joint_dp[-1] = -abs(self.__joint_dp[-1])
138             elif abs(cur_pos - min_) <= self.__ANGLE_EPS:
139                 self.__joint_dp[-1] = abs(self.__joint_dp[-1])
140             self.move_joint(self.__joint_handles[-1],
                      ↪  self.__joint_dp[-1])
141
142         def get_joint_pos(self, joint_handle: VREPHandle):
143             _, curr_coord = vrepapi.simxGetJointPosition(self.client_id,
                      ↪  joint_handle, vrepapi.simx_opmode_blocking)
144             print(f'Getting joint pos... pos: {curr_coord}')
145             return curr_coord if self.response_good(_) else None
146
147         def move_joint(self, joint_handle: VREPHandle, delta: float):
148             curr_pos = self.get_joint_pos(joint_handle)
149             if curr_pos is None:
150                 return
151             self.set_joint_pos(joint_handle, curr_pos + delta)
152
153         def set_joint_pos(self, joint_handle: VREPHandle, pos: float):
154             print(f'Setting joint pos: {pos}')
155             vrepapi.simxSetJointTargetPosition(self.client_id,
                      ↪  joint_handle, pos, vrepapi.simx_opmode_oneshot)
156
157         def set_joint_vel(self, joint_handle: VREPHandle, vel: float):
158             vrepapi.simxSetJointTargetVelocity(self.client_id,
                      ↪  joint_handle, vel, vrepapi.simx_opmode_oneshot)
159
160         def __touch_object(self):
161             while True:
```

```
162             if self.distance is not None and self.distance <=
                ↪  self.__D_EPS:
163                 break
164             self.move_joint(self.__joint_handles[1], 0.001)
165             self.move_joint(self.__joint_handles[2], -0.001)
166
167     def __home(self):
168         for joint_handle in self.__joint_handles:
169             self.set_joint_pos(joint_handle, 0)
170         while not all(isclose(self.get_joint_pos(_), 0,
                ↪  abs_tol=self.__D_EPS) for _ in self.__joint_handles):
171             pass
172
173     def _run(self):
174         self.__init()
175         while not self.object_found:
176             self.__move_to_next_point()
177         self.__touch_object()
178         self.__home()
179
180
181 if __name__ == '__main__':
182     SERVER_HOST = '127.0.0.1'
183     SERVER_PORT = 19999
184
185     controller = L3Controller(SERVER_HOST, SERVER_PORT)
186
187     controller.run()
```