Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Лабораторная работа №2
по дисциплине «Имитационное моделирование робототехнических систем»

Выполнил:
студент гр. R41341c
Борисов М. В.

Преподаватель:
Бжихатлов И. А.

Санкт-Петербург
2021 г.

Цель работы

- Написать скрипт, управляющий моделью на основе данных сенсора
- Написать аналогичный скрипт используя RemoteAPI
- Добавить в модель график и используя скрипт вывести на него данные

Ход работы

В данной работе используем модель из предыдущей лабораторной (рисунок 1).

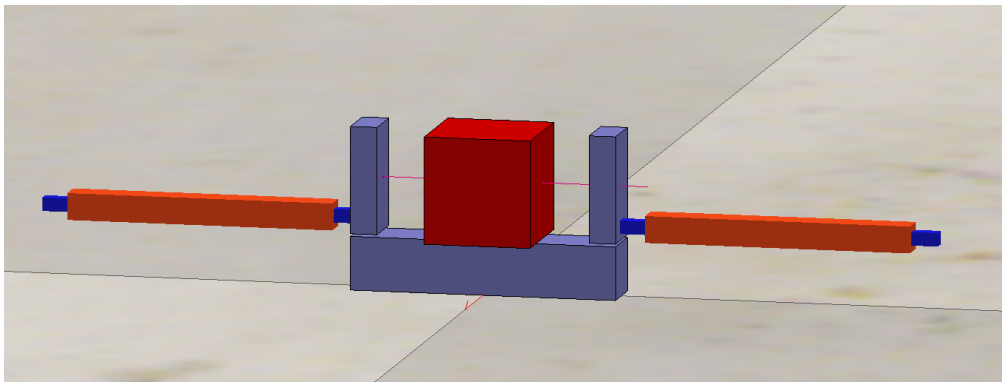

Рис. 1: Модель схвата

Напишем скрипт, который будет выполнять следующую последовательность:

1. Схват объекта
2. Интервал ожидания (5 секунд)
3. Разжатие объекта

Начнём сразу с использования RemoteAPI и выберем Python как язык клиента. Чтобы использовать связки с функциями симулятора необходимо из директории симулятора скопировать набор библиотек в свою рабочую директорию.

Для обеспечения работы клиента сперва необходимо запустить симуляцию, которая на старте открывает порт для внешнего подключения командой `simRemoteApi.start(19999)`, где аргумент функции есть номер порта.

На стороне клиента обязательно необходимо открыть подключение с помощью функции `simxStart()`, она принимает данные о сервере, как работать с подключением и возвращает ID по которому в дальнейшем нужно отправлять все функции. По окончанию работы клиент обязан закрыть подключение функцией `simxFinish()`.

Одной неприятной особенностью API стало отсутствие привязки к функции `getSimulationTime()`, поэтому пришлось писать небольшую функцию обёртку внутри модели и вызывать её через API. Нежелание использовать на клиенте, например, модуль `time` связано с тем что хотелось использовать именно время симуляции.

```
71 function getSimTime_function(inInts,inFloats,inStrings,inBuffer)
72     return {}, {sim.getSimulationTime()}, {}, ''
73 end
```

```
79     if self.response_good(res) and ready:
80         if self.initial_dist is None:
81             self.initial_dist = d
82         return d
83 
84     return None
85 
86 @property
87 def sim_time(self) -> float:
88     res, ints, floats, strings, buffer = \
89         vrepapi.simxCallScriptFunction(
90                 self.client_id,
```

Полный код приведён в приложении А.

Имея код для клиента довольно легко перенести его на Lua, однако ввиду ограниченных знаний о языке подход будет отличаться. Также как и в клиенте напишем раздельные функции, но, поскольку мы используем непотоковые скрипты, нельзя явно блокировать процесс циклами **`while`**, иначе симуляция зависнет. Поэтому будем использовать флаги для обозначения состояния. При этом, чтобы не делать две идентичные модели (одну для API другую для внутренних скриптов) мы сможем сделать так, чтобы при запуске симуляции единожды производилась демонстрация работы модели используя Lua, после чего можно будет подключиться через API.

Дополнительно напишем код для преобразования данных с сенсора. Если смотреть на выводимые графики, то видно, что на датчике присутствует шум, который скорее всего связан с дискретностью симуляции. Несмотря на то что колебания небольшие мы можем написать функцию, которая будет считать скользящее среднее.

Для этого нам потребуется написать дополнительно следующие функции для суммирования элементов массива, поскольку в стандартной библиотеке Lua такая отсутствует.

```
function sumf(a, ...) return a and a + sumf(...) or 0 end
function sumt(t) return sumf(unpack(t)) end
```

Первая выполняет суммирование произвольного количества аргументов используя рекурсию, вторая ”распаковывает” массив в отдельные аргументы для этой функции.

Получим следующие графики

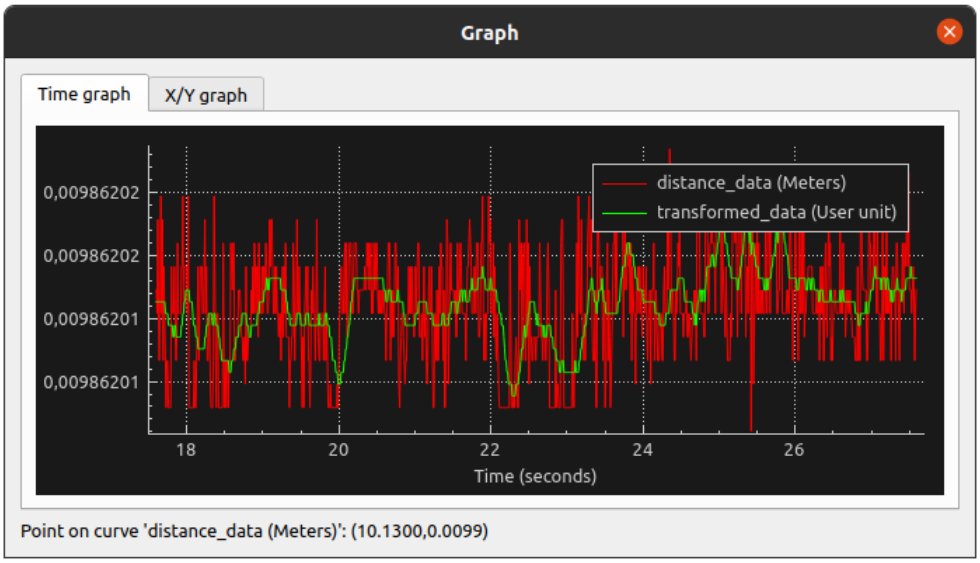

Рис. 2: Данные с сенсора от времени

Вывод

В работе была изучена работа с графиками и методы преобразования данных получаемых в процессе симуляции. Также исследован список доступных remoteAPI функций для Python и на их основе написан управляющий скрипт. Показана идентичность работы симуляции как при работе с клиентом, так и напрямую из скриптов Lua.

A. RemoteAPI скрипт

```
from abc import ABCMeta, abstractmethod
from typing import Any, Optional

import vrepapi.sim as vrepapi

```

```
 6
 7  class VREPClient(metaclass=ABCMeta):
 8      client_id: Any
 9      __host: Any
10      __port: Any
11
12      def __init__(self, host, port):
13          self.__host = host
14          self.__port = port
15
16      def __enter__(self):
17          print(f'Initiating connection with remote
             ↪  "{self.__host}:{self.__port}"...')
18          self.client_id = vrepapi.simxStart(self.__host, self.__port,
             ↪  True, True, 5000, 5)
19          print(f'Connection initiated. CLIENT_ID: {self.client_id}')
20          return self
21
22      def __exit__(self, exc_type, exc_val, exc_tb):
23          print(exc_type, exc_val, exc_tb)
24          print(f'Closing remote connection with client
             ↪  {self.client_id}')
25          vrepapi.simxFinish(self.client_id)
26
27      @abstractmethod
28      def _run(self):
29          pass
30
31      @staticmethod
32      def response_good(code: int) -> bool:
33          return code in (vrepapi.simx_return_ok,
             ↪  vrepapi.simx_return_novalue_flag)
34
35      def run(self):
36          with self:
37              self._run()
38
39
40  class L2Controller(VREPClient):
41      LEFT_HOLDER_NAME: str = 'left_holder_joint'
42      RIGHT_HOLDER_NAME: str = 'right_holder_joint'
43      SENSOR_NAME: str = 'left_holder_sensor'
44
45      D_EPS = 1e-4
46
47      def __init_handlers(self):
48          # Get handles to the objects
49          print('Initiating handlers for objects')
```

```
        _, self.left_holder = vrepapi.simxGetObjectHandle(
                self.client_id, self.LEFT_HOLDER_NAME,
                vrepapi.simx_opmode_blocking
                )
        _, self.right_holder = vrepapi.simxGetObjectHandle(
                self.client_id, self.RIGHT_HOLDER_NAME,
                vrepapi.simx_opmode_blocking
                )
        _, self.sensor = vrepapi.simxGetObjectHandle(
                self.client_id, self.SENSOR_NAME,
                vrepapi.simx_opmode_blocking
                )

    def __init_datastreams(self):
        print('Initiating data streams')
        # initialize stream to get distance
        vrepapi.simxReadProximitySensor(self.client_id, self.sensor,
         ↪  vrepapi.simx_opmode_streaming)

    @property
    def distance(self) -> Optional[float]:
        print(f'Reading from sensor...')

        res, ready, (_, _, d), *_ = vrepapi.simxReadProximitySensor(
                self.client_id, self.sensor,
                 ↪  vrepapi.simx_opmode_buffer
                )

        print(f'Readings:'
              f'\n  response: {res}; ready: {ready}; distance: {d}\n')

        if self.response_good(res) and ready:
            if self.initial_dist is None:
                self.initial_dist = d
            return d

        return None

    @property
    def sim_time(self) -> float:
        res, ints, floats, strings, buffer = \
            vrepapi.simxCallScriptFunction(
                    self.client_id,
                    'Gripper',
                    vrepapi.sim_scripttype_childscript,
                    'getSimTime_function',
                    [], [], [], bytearray(),
                    vrepapi.simx_opmode_blocking
```

```
                    )
        return floats[0] if self.response_good(res) else -1

    @property
    def initial_dist(self):
        try:
            return self.__initial_dist
        except AttributeError:
            return None

    @initial_dist.setter
    def initial_dist(self, val: float):
        self.__initial_dist = val

    def __set_gripper_velocity(self, vel: float):
        vrepapi.simxSetJointTargetVelocity(self.client_id,
         ↪  self.left_holder, vel, vrepapi.simx_opmode_oneshot)
        vrepapi.simxSetJointTargetVelocity(self.client_id,
         ↪  self.right_holder, vel, vrepapi.simx_opmode_oneshot)

    def __clench(self, vel: float):
        print(f'Clenching gripper with velocity: {vel}')
        while True:
            if self.distance is not None and self.distance <=
             ↪  self.D_EPS:
                break
            self.__set_gripper_velocity(abs(vel))
        self.__set_gripper_velocity(0)

    def __wait(self, n: int):
        print(f'Waiting for {n} sec (sim time)...')
        start = self.sim_time
        while self.sim_time - start < n:
            pass

    def __unclench(self, vel: float):
        print(f'Unclenching gripper with velocity: {vel}')
        while abs(self.distance - self.initial_dist) > self.D_EPS:
            self.__set_gripper_velocity(-abs(vel))
        self.__set_gripper_velocity(0)

    def _run(self):
        self.__init_handlers()
        self.__init_datastreams()
        self.__clench(0.008)
        self.__wait(5)
        self.__unclench(0.008)

```

```python

if __name__ == '__main__':
    SERVER_HOST = '127.0.0.1'
    SERVER_PORT = 19999

    controller = L2Controller(SERVER_HOST, SERVER_PORT)

    controller.run()
```

Б. Скрипт управления схватом на Lua

```lua
function sysCall_init()
    simRemoteApi.start(19999)
    left_holder = sim.getObjectHandle('left_holder_joint')
    right_holder = sim.getObjectHandle('right_holder_joint')
    sensor = sim.getObjectHandle('left_holder_sensor')
    D_EPS = 1e-4
    velocity = 0.008
    wait_interval = 5

    wait_start = nil
    init_distance = nil
    clenched = nil
    waited = nil
    unclenched = nil
end

function clench()
    if clenched then
        return
    end
    res, distance = sim.readProximitySensor(sensor)
    if res > 0 then
        if init_distance == nil then
            init_distance = distance
        end
        if distance <= D_EPS then
            sim.setJointTargetVelocity(left_holder, 0)
            sim.setJointTargetVelocity(right_holder, 0)
            clenched = true
            print('Done.')
            return
        end
    end
    sim.setJointTargetVelocity(left_holder, velocity)
    sim.setJointTargetVelocity(right_holder, velocity)
```

```
36      print('Clenching...')
37  end
38
39  function wait()
40      if waited or not clenched then
41          return
42      elseif start == nil then
43          start = sim.getSimulationTime()
44      elseif sim.getSimulationTime() - start > wait_interval then
45          waited = true
46          print('Done.')
47          return
48      end
49      print('Waiting...')
50  end
51
52  function unclench()
53      if unclenched or not waited then
54          return
55      end
56
57      res, distance = sim.readProximitySensor(sensor)
58      if res > 0 and math.abs(distance - init_distance) <= D_EPS then
59          sim.setJointTargetVelocity(left_holder, 0)
60          sim.setJointTargetVelocity(right_holder, 0)
61          unclenched = true
62          print('Done.')
63          return
64      end
65      sim.setJointTargetVelocity(left_holder, -velocity)
66      sim.setJointTargetVelocity(right_holder, -velocity)
67      print('Unclenching...')
68  end
69
70
71  function getSimTime_function(inInts,inFloats,inStrings,inBuffer)
72      return {}, {sim.getSimulationTime()}, {}, ''
73  end
74
75  function sysCall_actuation()
76      clench()
77      wait()
78      unclench()
79  end
80
81  function sysCall_sensing()
82      -- put your sensing code here
83  end
```

```
84
85  function sysCall_cleanup()
86      -- do some clean-up here
87  end
88
89  -- See the user manual or the available code snippets for additional
     ↪  callback functions and details
```

B. Скрипт работы с графиком

```
 1  function sysCall_init()
 2      graph = sim.getObjectHandle('Graph')
 3      sensor = sim.getObjectHandle('left_holder_sensor')
 4      buffer = {}
 5      max_points = 20
 6  end
 7
 8  function sumf(a, ...) return a and a + sumf(...) or 0 end
 9  function sumt(t) return sumf(unpack(t)) end
10
11  function sysCall_actuation()
12      -- put your actuation code here
13  end
14
15  function sysCall_sensing()
16      res, d = sim.readProximitySensor(sensor)
17      if res > 0 then
18          if #buffer >= max_points then
19              table.remove(buffer, 1)
20          end
21          buffer[#buffer + 1] = d
22          data = sumt(buffer)
23          sim.setGraphUserData(graph, 'transformed_data', data /
             ↪  #buffer)
24      end
25  end
26
27  function sysCall_cleanup()
28      -- do some clean-up here
29  end
30
31  -- See the user manual or the available code snippets for additional
     ↪  callback functions and details
```