

Финал номинации Манипуляционные ИРС олимпиады Innopolis Open in Robotics 2020 будет проводиться с использованием симулятора роботов CoppeliaSim.

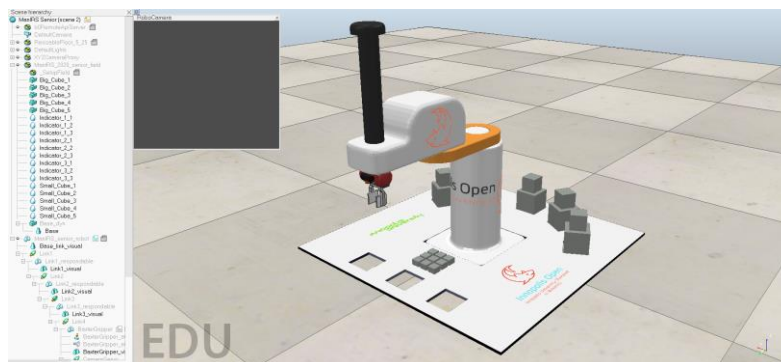
Этот архив содержит необходимые для тренировок и участия в финале модели, сцены и примеры программ для старшей возрастной группы (9-11 классы).

Содержимое архива:

- Файл ManIRS Senior.ttt - сцена с полигоном и стандартным роботом.
- Каталог Models - содержит модели робота и игрового поля. Советуем перенести его содержимое в каталог симулятора. Для ОС Windows чаще всего это C:\Program Files\CoppeliaRobotics\CoppeliaSimEdu\models\любой\_удобный\_каталог
- Каталог Programming - содержит заготовки программ на различных языках программирования.

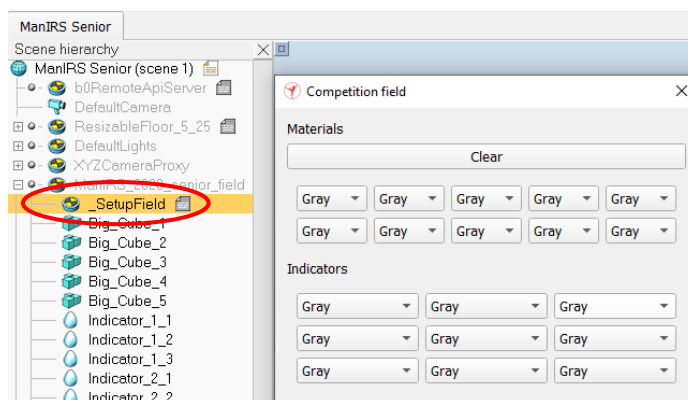
## Начало работы

1. Откройте сцену «ManIRS Senior.ttt» в симуляторе CoppeliaSim. Сцена уже содержит стандартную модель робота, полигон с возможностью предварительной настройки, сервер B0\_remote\_API для подключения и управления моделью из внешних программ, дополнительное окно для отображения данных с камеры:

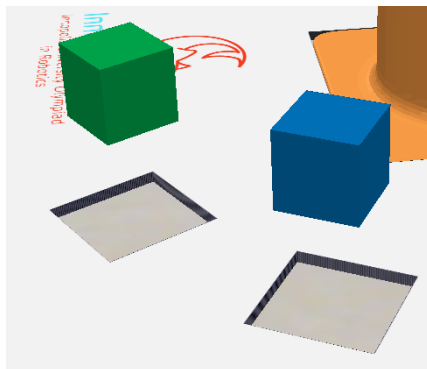


Альтернативным методом будет создание новой сцены и добавление в нее модели полигона и робота. Это может быть полезно при написании программы на языке Lua и отсутствии необходимости в сервере B0\_remote\_API.

2. Настройте стартовые цвета объектов. Для этого в дереве иерархии модели найдите объект с именем «\_SetupField» и щелкните по нему. Появится окно для настройки сцены:



Выбирая различные варианты в открывающихся списках, установите цвета объектов. Модель позволяет задавать одинаковые цвета нескольким объектам. Модель позволяет убрать объект вариантом None, при этом вышестоящий объект упадет на основание полигона при запуске симуляции. Можете использовать это для тренировок выполнения дополнительных заданий.



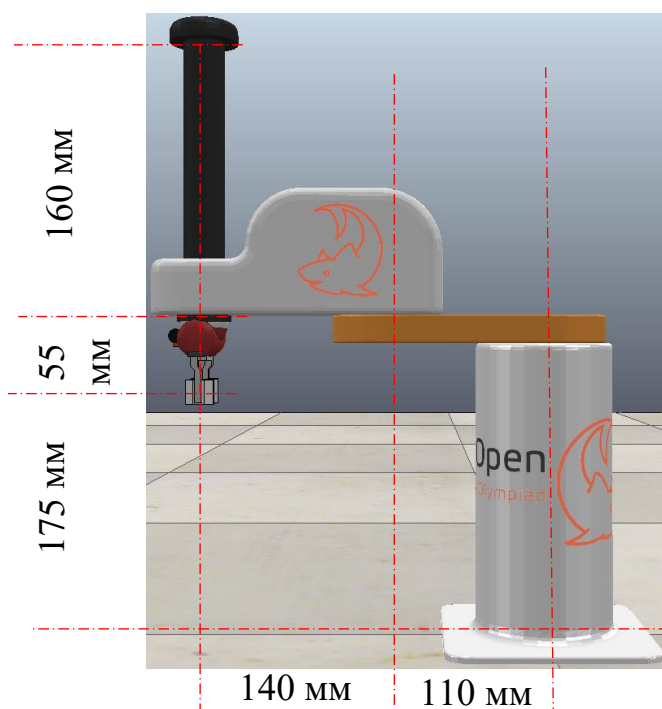
Сброс до исходного состояния производится кнопкой «Clear» интерфейса настройки полигона. Чтобы закрыть окно настройки достаточно сбросить выбор с объекта «\_SetupField» в дереве иерархии (выбрать другой объект или снять выбор со всех объектов).

3. Напишите управляющий код для робота. На ваш выбор есть заготовки под следующие языки программирования: Lua – скрипт у объекта «ManIRS senior robot» в самом симуляторе; Python – файл ManIRS\_senior.py в каталоге Programming/Python; C++ – файл ManIRS\_senior.cpp в каталоге Programming/CPP/ManIRS\_senior.

## Описание робота

Робот построен с использованием кинематической схемы SCARA. Он имеет четыре звена и четыре привода между ними, камеру с подвижным основанием и схват.

Механические характеристики робота:



Привод 1 поворачивает второе звено (оранжевое) и последующие вокруг основания. Угол поворота от -180 градусов до 180 градусов. Скорость поворота до 180 градусов в секунду.

Привод 2 поворачивает третье звено (белое, с акулой) и последующие вокруг второго. Угол поворота от -150 до 150 градусов. Скорость поворота до 180 градусов в секунду.

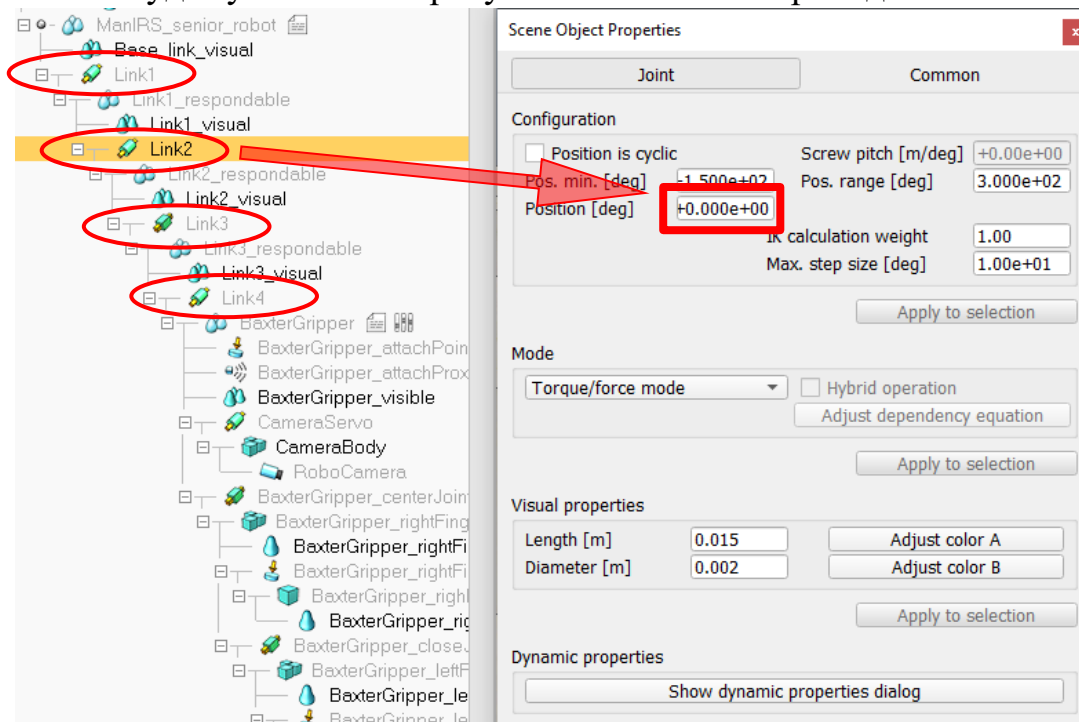
Привод 3 поднимает и опускает четвертое звено (черное), схват и камеру относительно третьего и предыдущих звеньев. Диапазон перемещений – от 0 до 160 мм. Обратите внимание, что положение с большим значением поднимает звено вверх. На рисунке выше третий привод установлен в максимальное положение, 160 мм. Максимальная скорость перемещения до 250 мм в секунду.

Привод 4 поворачивает схват (красный) и камеру относительно четвертого звена. Угол поворота от -180 до 180 градусов. Скорость поворота до 540 градусов в секунду.

На рисунке выше у приводов робота установлены следующие значения:

- привод 1 = 0 градусов,
- привод 2 = 0 градусов,
- привод 3 = 160 мм,
- привод 4 = 0 градусов.

Именно в этом положении оказывается вновь созданная модель робота. Обратите внимание, что робот не соответствует требованиям к стартовому положению (п. 1.3. правил). Вы можете выбрать собственное стартовое положение. Для установки привода в требуемое положение найдите его в дереве иерархии (приводы называются Link1..Link4) и дважды щелкните по иконке. В результате откроется окно свойств, в котором можно будет установить требуемое положение привода:



Напоминаем, что по правилам соревнований все кинематические пары, звенья и т.д. должны быть оснащены красными индикаторами нулевого положения. Инструкции по созданию индикатора или разъяснения по их альтернативному определению будут добавлены позже.

Всеми приводами робота управляют настроенные в симуляторе ПИД-регуляторы. От участника требуется указать в программе требуемое положение одного или всех звеньев, робот сам установит свои звенья в указанное положение. Обратите внимание, что на максимальных скоростях звенья двигаются так быстро, что робот может выронить переносимый в схвате объект!

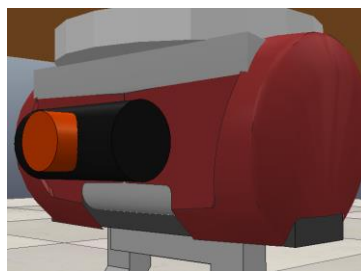
Изначально всем звеньям заданы следующие коэффициенты ПИД-регулятора:  $k_p=0.1$ ,  $k_i=0$ ,  $k_d=0$ . Участник может задавать собственные коэффициенты регуляторов и максимальные скорости приводов (см. раздел про программирование). Участники не могут установить скорости выше указанных ранее (180 г/с для приводов 1 и 2, 540 г/с для привода 4 и 250 мм/с для привода 3), это будет проверять специальная программа-судья.

Робот оборудован схватом для манипулирования объектами. Максимальное раскрытие пальцев схвата составляет 55 мм. Скорость движения пальцев схвата – 75 мм/с. Полное закрытие или открытие схвата происходит примерно за 0.75 секунды. Учитывайте это при управлении роботом!

На схвате робота дополнительно установлена камера, которую можно использовать для распознавания цветов индикаторов и объектов. Камера может смотреть вниз или вперед. Поворот камеры из одного положения в другое выполняется примерно за 0.2 секунды.

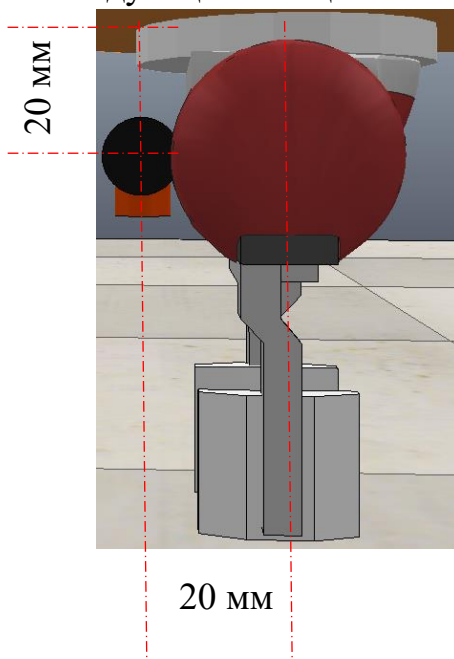


Камера направлена вниз



Камера направлена вперед

Камера прикреплена к схвату и вращается вместе с ним с помощью привода 4. При решении задачи учитывайте следующие смещения:



## Программирование робота

Программное управление моделью робота в симуляторе может проходить в синхронном и асинхронном режиме управления. Асинхронный напоминает работу с роботом, имеющим два контроллера: низкоуровневый для управления моторами и считывания показаний датчиков (реализован средствами симулятора) и высокоуровневый для выполнения основной логики программы. Высокоуровневый контроллер имеет собственный поток выполнения, который может быть синхронизирован с низкоуровневым различными сигналами и «семафорами». Для имитации высокоуровневого контроллера используется отдельный поток выполнения программы, который можно синхронизировать с симулятором различными системными командами.

Синхронный режим управления напоминает работу на одном контроллере, содержащем функции, вызываемые по таймеру. Так, на каждом шаге симуляции вызываются соответствующие функции. Даже на слабых компьютерах симулятор будет останавливаться и ожидать завершения всех вычислений данного шага.

Научно-методический комитет подготовил функции, выполняемые как на каждом шаге симуляции (для полной синхронизации управляющей программы и модели) и в отдельном потоке. Участники могут выбирать удобный для них способ и вписывать собственный код в соответствующие функции. При работе в синхронном режиме порядок выполнения симуляции следующий:

1. Выполняются иницирующие функции симулятора и его объектов;
2. Выполняется пользовательская функция *init()*;
3. Начинается шаг, выполняются просчет и перемещение объектов физическим движком симулятора;
4. Продолжается шаг, выполняется пользовательская функция *simulationStepStarted()*;
5. Продолжается шаг, выполняются считывания и пересчет значений всех датчиков;
6. Завершается шаг выполнением пользовательской функции *simulationStepDone()*;
7. Начинается шаг, выполняется просчет и перемещение объектов....
- ...
- n-1. Выполняются очищающие функции симулятора и его объектов;
- n. Выполняется пользовательская функция *cleanup()*.

Таким образом, задача участников сводится к написанию действий в четырех функциях: *init()* –выполнится один раз перед симуляцией, *simulationStepStarted()* – будет выполняться в начале каждого шага симуляции, *simulationStepDone()* – будет выполняться в конце каждого шага симуляции, *cleanup()* – выполнится один раз после остановки симуляции.

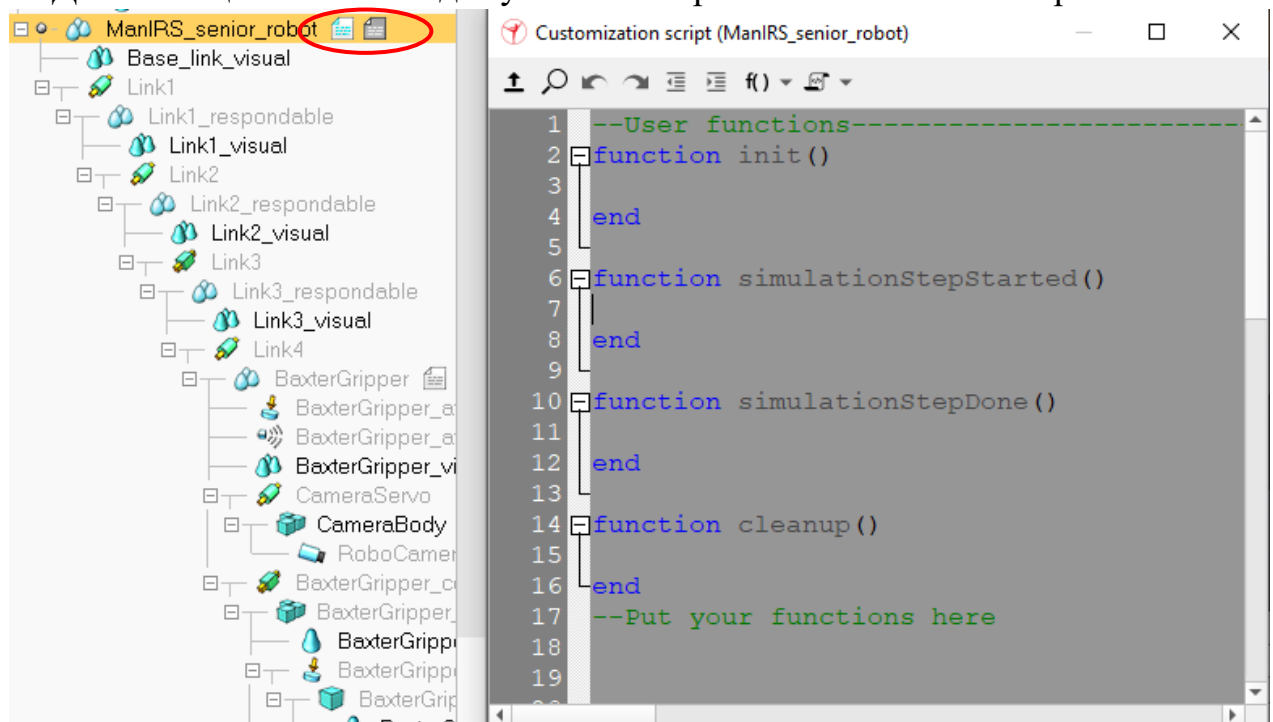
Функции *simulationStepStarted()* и *simulationStepDone()* должны быть неблокирующими. В них прописываются действия только для одного шага. Они будут вызываться в цикле самого симулятора. Все показания датчиков и положения объектов остаются неизменными на протяжении одного вызова функции *simulationStepDone()*.

Нельзя просто написать в функцию *simulationStepStarted()* или *simulationStepDone()* все действия манипулятора подряд. Для этого можно использовать асинхронный режим управления и вписать управляющие команды в соответствующую область или функцию (отличаются у различных языков программирования, помечены комментариями). В этом случае создается отдельный поток для выполнения программы. В начале потока могут быть вызваны те же команды, что и в функции *init()*. Далее могут вызываться блокирующие и неблокирующие команды, ожидание времени или событий с объектами симулятора. После выполнения всех действий поток завершается. При этом симуляция может продолжать работать, если удалить команду его остановки. Стоит так же обратить внимание, что в асинхронном режиме изначально потоки программы и симуляции не ожидают друг друга, что может вызвать некоторые трудности. Например, во время длительной обработки изображения с камеры в потоке программы поток симулятора может передвинуть робота слишком далеко. Рекомендуем для управления роботом использовать установку необходимых положений приводов. В этом случае, даже если поток программы задержится на долгое время, ПИД-регулятор потока симулятора приведет привод в заданное положение и будет ожидать дальнейших команд.

Для большего удобства переноса кода между тренировками и зачетным выполнением рекомендуется писать программу только на одном языке программирования, без смешивания скриптов Lua с программой Python и т.д.

В зависимости от используемого языка программирования участники должны выполнить следующие действия:

- Lua – управляющая программа пишется в один из двух внутренних скриптов. Найдите в дереве иерархии корневой объект робота с именем «ManIRS\_senior\_robot» и рядом с ним иконки текстового файла (скрипта). Двойной щелчок по каждому из них откроет окно с текстом скрипта:



Светлый скрипт – потоковый дочерний – содержит функцию *sysCall\_threadmain()*, в начале которой прописываются иницилирующие команды, а далее основной поток



выполнения задания. После запуска симуляции эта функция будет выполняться в отдельном потоке.

Темный скрипт – кастомизирующий скрипт – содержит четыре функции *init()*, *simulationStepStarted()*, *simulationStepDone()*, *cleanup()*.

Пользователь может прописать действия в оба скрипта. Или удалить неиспользуемый скрипт.

В каталоге Programming/Lua находится файл RoboFunctions.lua с заранее подготовленными функциями управления роботом. Для адекватной их работы необходимо в обоих скриптах найти команду

*require "Programming/Lua/RobotFunctions"*

и в ней указать верный путь к файлу. Если данный способ не работает, то можно просто скопировать содержимое файла RoboFunction.lua и вставить в один или оба скрипта.

- Python – управляющая программа пишется в файл ManIRS\_senior.py, находящийся в каталоге Programming/Python. Важно запускать этот файл из 64-битной версии IDLE (так как и симулятор 64-битный):

```
import os
os.add_dll_directory(r'C:\Program Files\CoppeliaRobotics\CoppeliaSimEdu')

import b0RemoteApi
from RoboFunctions import ManRobot
import time

with b0RemoteApi.RemoteApiClient('b0RemoteApi_pythonClient','b0RemoteApi_manirs') as client:
    doNextStep=True
    robot=ManRobot(client)
    step=0 #переменная для подсчета шагов симуляции

    def init():
        #пример использования функций управления роботом:
        #подробнее о всех функциях, их параметрах и возвращаемых значениях см ReadMe

        #энкодеры:
        #robot.link1_enc
        #robot.link2_enc
        #robot.link3_enc
        #robot.link4_enc
        #robot.grip_enc
        #robot.cam_enc
```

Во второй строке необходимо указать актуальный путь к корневому каталогу симулятора (содержит все необходимые для работы библиотеки .dll). Файл содержит сразу функции *init()*, *simulationStepStarted()*, *simulationStepDone()*, *cleanup()* и раздел «Put your main action here». Изначально в этом разделе указан цикл, после каждого шага которого управление будет переходить к симулятору. Далее вызываются функции *simulationStepStarted()*, *simulationStepDone()* и управление снова возвращается к циклу программы. В комментариях файла указан способ выполнения действий без цикла и синхронизации.

Для корректной работы с B0\_remote\_API\_server необходимо в Python установить пакет msgpack командой: `pip install msgpack`

Запуск симуляции производится не из CoppeliaSim, а запуском программы – она подключается к серверу B0, производит предварительные настройки и присылает команду на запуск симуляции.

- C++ - управляющая программа пишется в файл ManIRS\_senior.cpp, находящийся в каталоге Programming/CPP/ManIRS\_senior. В папке присутствуют все необходимые файлы DLL, файлы ManRobot.h и

ManRobot.cpp, описывающие соответствующий класс манипуляционного робота и основных действий с ним. В проект (управляющую роботом программу) должны быть включены файлы ManRobot.h и ManRobot.cpp и все зависимости, описанные в этом видео: [https://youtu.be/GpCJrI\\_ohYI](https://youtu.be/GpCJrI_ohYI)

Запуск симуляции производится не из CoppeliaSim, а запуском программы – она подключается к серверу V0, производит предварительные настройки и присылает команду на запуск симуляции.

Во всех языках программирования на каждом шаге симуляции обновляются шесть переменных – link1\_enc...link4\_enc, grip\_enc, cam\_enc. Они содержат положения приводов робота, раскрытие губок схвата и поворот камеры в градусах или миллиметрах. Обратите внимание, что в крайних положениях приводы могут незначительно не доходить до указанных значений. Например, положение привода камеры может быть не 90 градусов, а 89.999900051641. Это связано с ограничениями физического движка симулятора, учитывайте подобные моменты при организации циклов или округляйте значения по собственному желанию.

Для более удобного написания программ научно-методический комитет номинации подготовил следующие функции:

- **setPos1(pos)** – поворачивает привод 1 в положение pos. Параметр pos указывается в градусах в диапазоне -180...180
- **setPos2(pos)** – поворачивает привод 2 в положение pos. Параметр pos указывается в градусах в диапазоне -150...150
- **setPos3(pos)** – перемещает привод 3 в положение pos. Параметр pos указывается в миллиметрах в диапазоне 0...160
- **setPos4(pos)** – поворачивает привод 4 в положение pos. Параметр pos указывается в градусах в диапазоне -180...180
- **setPositions(pos1, pos2, pos3, pos4)** – устанавливает все приводы в указанные положения. Параметры имеют аналогичные значения и ограничения, как и в предыдущих функциях.
- **setPID(link, kp, ki, kd)** – устанавливает новые параметры ПИД-регулятора соответствующего привода. Параметру link заранее присваивается значение из link1...link4. Параметры kp, ki и kd задаются дробными числами
- **resetPID(link)** – сбрасывает параметры ПИД-регулятора соответствующего привода к значениям по-умолчанию: kp=0.1, ki=0, kd=0. Параметру link заранее присваивается значение из link1...link4.
- **setMaxSpeed(link, speed)** – устанавливает максимально допустимую скорость привода. Параметру link заранее присваивается значение из link1...link4. Параметр speed указывается в градусах или миллиметрах в секунду.
- **openGripper()** – указывает роботу открыть схват. Схват полностью откроется не на следующем шаге, а примерно через 3.6 секунды. Аналогично работе с реальным сервомотором – после установки нового положения сервопривод некоторое время перемещается в него.
- **closeGripper()** – указывает роботу закрыть схват. Схват полностью закроется не на следующем шаге, а примерно через 3.6 секунды. Аналогично работе с



реальным сервомотором – после установки нового положения сервопривод некоторое время перемещается в него.

- ***turnCameraToFront()*** – указывает роботу повернуть камеру вперед. Камера полностью повернется вперед не на следующем шаге, а примерно через 0.2 секунды. Аналогично работе с реальным сервомотором – после установки нового положения сервопривод некоторое время перемещается в него.
- ***turnCameraToDown()*** – указывает роботу повернуть камеру вниз. Камера полностью повернется вниз не на следующем шаге, а примерно через 0.2 секунды. Аналогично работе с реальным сервомотором – после установки нового положения сервопривод некоторое время перемещается в него.
- ***setCameraResolution(x, y)*** – устанавливает разрешение камеры робота. При отличающихся значениях *x* и *y* область видимости становится прямоугольной – «сжимается» по короткой стороне. Кроме того, изменяется количество пикселей и размер массива, возвращаемого командами *getCameraImage()*, *getCameraField()* и *robot.cam\_image*. Параметр *x* задает горизонтальное разрешение (ширину), параметр *y* – вертикальное разрешение (высоту) кадра. Параметры должны быть указаны в виде целых чисел. Доступный диапазон разрешения камеры: 1...1024 пикселя по каждому измерению.

На Lua доступны следующие функции:

- ***img=getCameraImage()*** – возвращает полное изображение с камеры в режиме RGB, с разрешением 256 x 256 пикселей. Возвращает массив длиной 256\*256\*3=196608 элементов. Структура возвращаемого массива: {*pix\_0x0\_red*, *pix\_0x0\_green*, *pix\_0x0\_blue*, *pix\_0x1\_red*, *pix\_0x1\_green*, *pix\_0x1\_blue*...} Каждое значение находится в диапазоне 0...1, например: {0.80000001192093, 0.7843137383461, 0.76470589637756...}
- ***pixel=getCameraPixel(x,y)*** – возвращает RGB-компоненты одного пикселя кадра с камеры. Функция отсутствует для языков Python и C++. Возвращает массив из трех элементов: {*pixel\_red*, *pixel\_green*, *pixel\_blue*} Каждое значение находится в диапазоне 0...1, например: {0.80000001192093, 0.7843137383461, 0.76470589637756}. Параметры – координаты X и Y пикселя, находятся в диапазоне 0...255 (при разрешении камеры 256x256 пикселей).
- ***imgField=getCameraField(x, x\_size,y, y\_size)*** – возвращает часть изображения (кадра) с камеры в режиме RGB. Функция отсутствует для языков Python и C++. Суммы *x+x\_size* и *y+y\_size* должны быть меньше или равны 255 (при разрешении камеры 256x256 пикселей). Возвращает массив длиной *x\_size\*y\_size\*3* элементов: {*pix\_X\_Y\_red*, *pix\_X\_Y\_green*, *pix\_X\_Y\_blue*, *pix\_X\_Y+1\_red*, *pix\_X\_Y+1\_green*, *pix\_X\_Y+1\_blue*...} Каждое значение находится в диапазоне 0...1, например: {0.80000001192093, 0.7843137383461, 0.76470589637756...}

На Python и C++ доступны следующие аргументы объекта ***robot***:

- ***simTime=robot.simTime*** – возвращает время симуляции на текущем шаге. Программа на Lua полностью с ним синхронизирована, а внешняя может выполняться значительно быстрее или медленнее симулятора. Время возвращается в секундах.

- ***img=robot.cam\_image*** – возвращает кадр с камеры на текущем шаге. Для Python кадр представлен в виде массива байт размером  $256*256*3=196608$  элементов. Для C++ кадр представлен строкой длиной в 196608 символов. Структура возвращаемого массива и символов строки: {pix\_0x0\_red, pix\_0x0\_green, pix\_0x0\_blue, pix\_0x1\_red, pix\_0x1\_green, pix\_0x1\_blue...} Каждое значение находится в диапазоне 0...1 для Python и 0...255 для C++.
- ***robot.disconnect*** – возвращает логическое значение (*true* или *false*), показывающее, произошел ли разрыв соединения с симулятором. Значение станет *true* в момент остановки симулятора (например, из программы-судьи) или в случае, если от симулятора не пришел корректный ответ.
- ***robot.stopDisconnect*** – устанавливает, должна ли программа участника автоматически выключаться при разрыве соединения. По умолчанию установлено в *true*, если перенастраивается на *false* участник должен позаботиться в своем алгоритме о принудительной остановке программы.