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

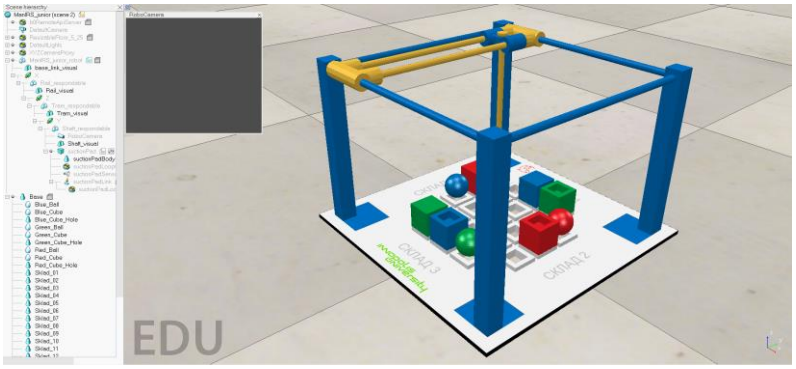
Этот архив содержит необходимые для тренировок и участия в финале модели, сцены и примеры программ для младшей возрастной группы (6-8 классы).

Содержимое архива:

- Файл ManIRS Junior.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.

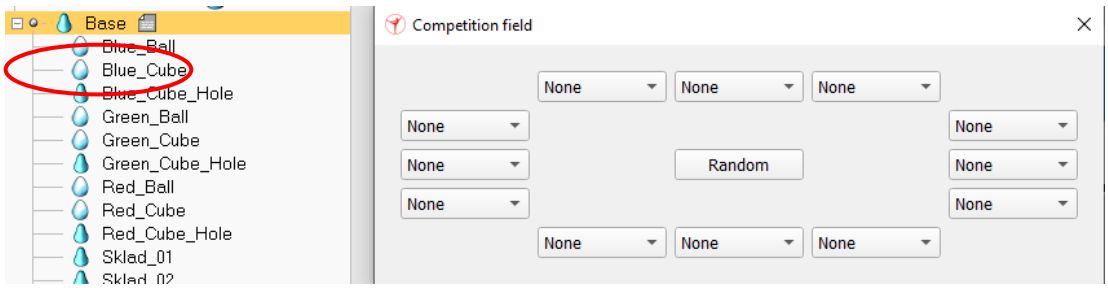
Начало работы

1. Откройте сцену «ManIRS Junior.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:

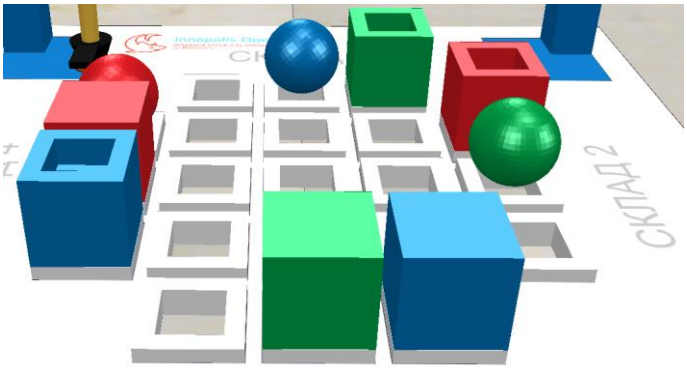


Альтернативным методом будет создание новой сцены и добавление в нее модели полигона и робота. Это может быть полезно при написании программы на языке Lua и отсутствии необходимости в сервере B0_remote_API.

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



Выбирая различные варианты в открывающихся списках, установите цвета объектов. Модель не позволит задавать одинаковые цвета или формы в соответствующих складах нескольким объектам. Модель позволяет сгенерировать случайную расстановку объектов, удовлетворяющую правилам номинации.



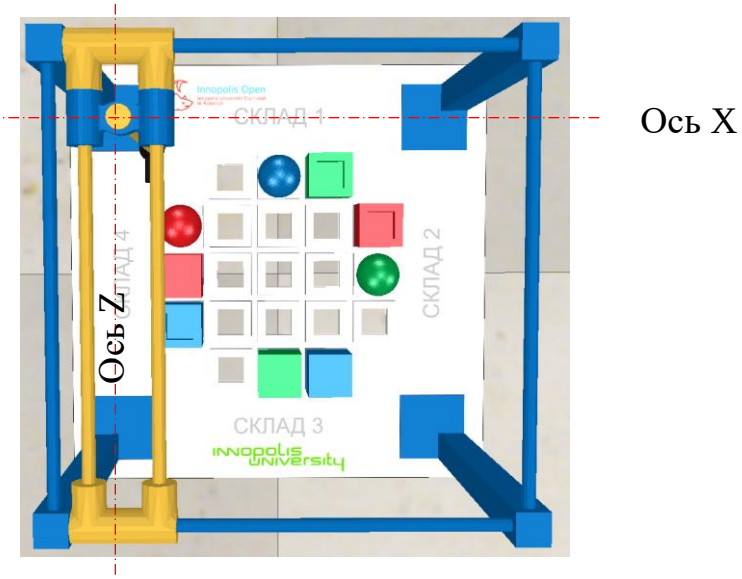
Чтобы закрыть окно настройки достаточно сбросить выбор с объекта «Base» в дереве иерархии (выбрать другой объект или снять выбор со всех объектов).

3. Напишите управляющий код для робота. На ваш выбор есть заготовки под следующие языки программирования: Lua – скрипт у объекта «ManIRS junior robot» в самом симуляторе; Python – файл ManIRS_junior.py в каталоге Programming/Python; C++ - ManIRS_junior.cpp в Programming/CPP/ManIRS_junior (решение создавалось в среде программирования VS 2019).

Описание робота

Механика робота напоминает кинематику 3D-принтера. Он имеет три звена и три привода между ними, камеру и пневматический схват.

Механические характеристики робота:

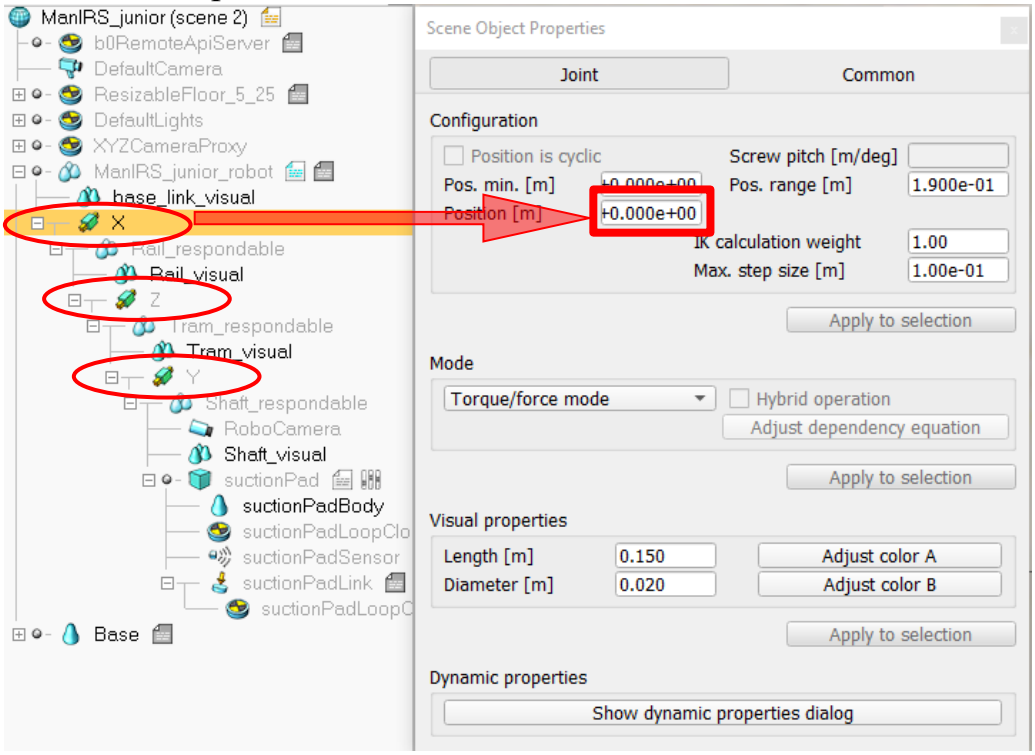


Привод X перемещает второе звено (желтое) и последующие вдоль складов 1 и 3. Диапазон перемещения от 0 до 190 миллиметров. Скорость перемещения до 100 миллиметров в секунду.

Привод Z перемещает третье звено (синяя каретка) и последующие вдоль складов 2 и 4. Диапазон перемещения от 0 до 190 миллиметров. Скорость перемещения до 100 миллиметров в секунду.

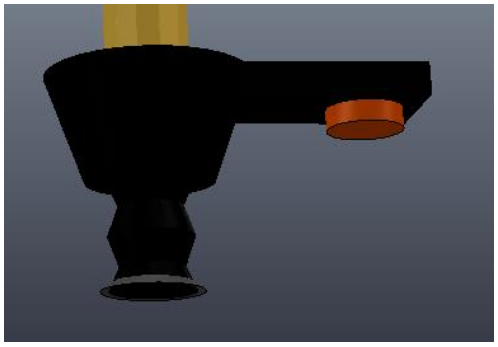
Привод Y поднимает и опускает четвертое звено (желтый стержень), схват и камеру относительно третьего и предыдущих звеньев. Диапазон перемещений – от 0 до 170 мм. Обратите внимание, что положение с большим значением поднимает звено вверх. Максимальная скорость перемещения до 100 мм в секунду.

На рисунке выше у всех приводов робота установлены нулевые значения.
Именно в этом положении оказывается вновь созданная модель робота. Вы можете выбрать другое стартовое положение. Для установки привода в требуемое положение найдите его в дереве иерархии (приводы называются X, Y и Z) и дважды щелкните по иконке. В результате откроется окно свойств, в котором можно будет установить требуемое положение привода:

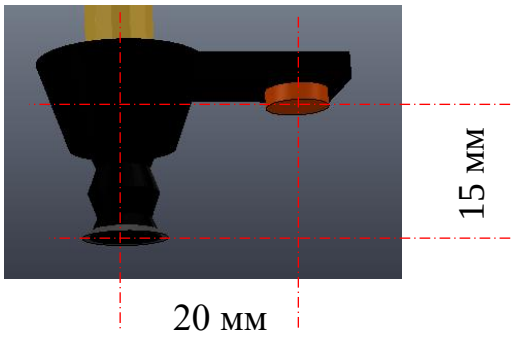


Напоминаем, что по правилам соревнований все кинематические пары, звенья и т.д. должны быть оснащены красными индикаторами нулевого положения. Инструкции по созданию индикатора или разъяснения по их альтернативному определению будут добавлены позже.
Всеми приводами робота управляют настроенные в симуляторе ПИД-регуляторы. От участника требуется указать в программе необходимое 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: kp=0.1, ki=0, kd=0. Участник может задавать собственные коэффициенты регуляторов и максимальные скорости приводов (см. раздел про программирование). Участники не могут установить скорости свыше указанных ранее (100 мм/с), это будет проверять специальная программа-судья.
Робот оборудован схватом для манипулирования объектами. Максимальное усилие отрыва составляет 4 Н, боковое усилие 3 Н.

На схвате робота дополнительно установлена камера, которую можно использовать для распознавания цветов индикаторов и объектов. Камера смотрит вниз.



Камера прикреплена к схвату и перемещается вместе с ним. 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), так 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 инициирующие 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.

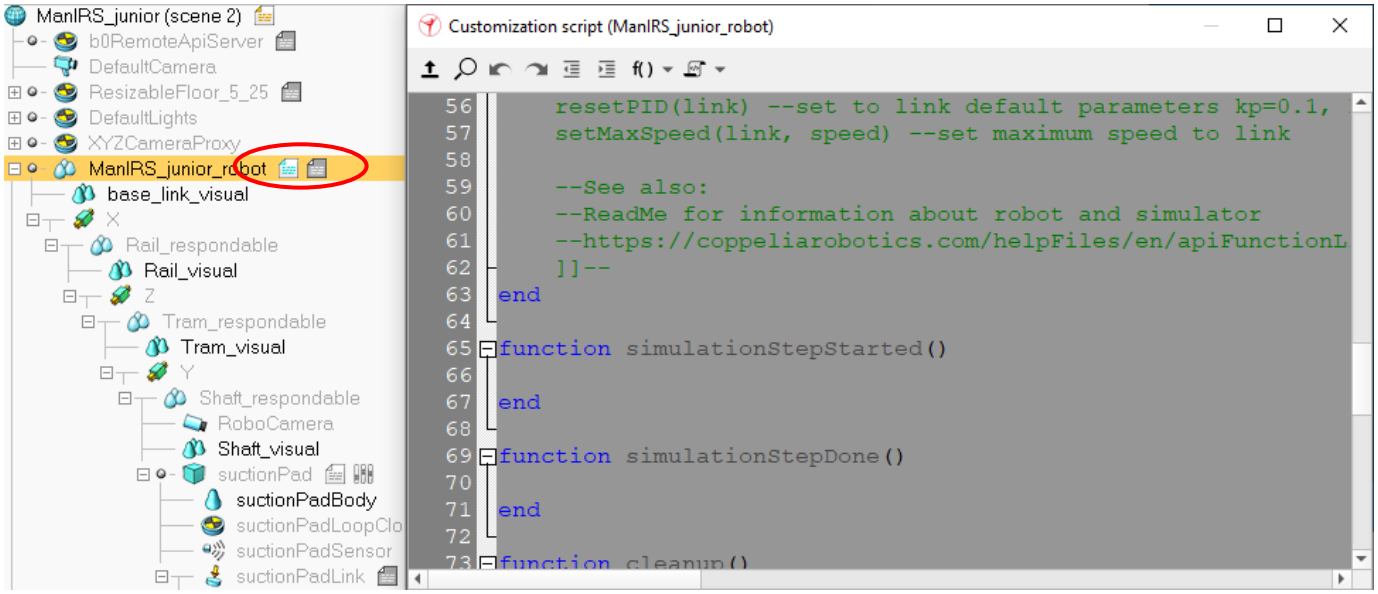
Функции *simulationStepStarted()* и *simulationStepDone()* должны быть неблокирующими. В них прописываются действия только для одного шага. Они будут вызываться в цикле самого симулятора. Все показания датчиков и положения объектов остаются неизменными на протяжении одного вызова функции *simulationStepDone()*.

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 отдельно 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_junior_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 инициирующие 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_junior.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 парметрах и возвращаемых значения см ReadMe

        #энкодеры:
        #robot.X_enc #положение привода X
        #robot.Y_enc #положение привода Y
        #robot.Z_enc #положение привода Z

        #simTime=robot.simTime
```

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_junior.cpp, находящийся в каталоге Programming/CPP/ManIRS_junior. В папке присутствуют все необходимые файлы для работы сервером B0_remote_API, 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

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

Во всех языках программирования на каждом шаге симуляции обновляются три переменных – X_enc…Z_enc. Они содержат положения приводов робота в миллиметрах. Обратите внимание, что в крайних положениях приводы могут незначительно не доходить до указанных значений. Например, положение привода камеры может быть не 190 миллиметров, а 189.999900051. 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:

- ***setPosX(pos)*** – перемещает привод X в положение pos. Параметр pos указывается в миллиметрах в диапазоне 0…190.
- ***setPosY(pos)*** – перемещает привод Y в положение pos. Параметр pos указывается в миллиметрах в диапазоне 0…190.
- ***setPosZ(pos)*** – перемещает привод Z в положение pos. Параметр pos указывается в миллиметрах в диапазоне 0…170
- ***setPositions(posX, posY, posZ)*** – 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 linkX…linkZ. 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 linkX…linkZ.
- ***setMaxSpeed(link, speed)*** – устанавливает максимально допустимую скорость привода. Параметру link заранее присваивается значение из linkX…linkZ. Параметр speed указывается в градусах или миллиметрах в секунду.
- ***grapObject()*** – указывает роботу включить схват. Объект, находящийся ближе 2 мм к рабочей поверхности схвата притянется к нему.
- ***releaseObject()*** – указывает роботу отключить схват.
- ***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 длинной 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 длинной 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 длинной 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.