

## Практическая работа № 2

### Программирование мобильного робота в среде CoppeliaSim

**Цель работы:** получить практические навыки программирования дочерних сценариев среды CoppeliaSim для обработки сигнала оптических датчиков и релейного регулирования движения мобильного робота.

#### 1. Основы программирования роботов в CoppeliaSim

##### Обзор программных средств CoppeliaSim

CoppeliaSim поддерживает 6 различных подходов к программированию, каждый из которых имеет определенные преимущества по сравнению с другими, но все шесть взаимно совместимы:

1. **Встроенный сценарий (Embedded script)** - это способ управления симуляцией работы сцен или моделей с помощью сценариев на языках Lua или Python. Он очень прост и гибок, и гарантирует совместимость со всеми другими установками CoppeliaSim по умолчанию. Это самый простой и часто используемый подход программирования в среде симулятора.
2. **Надстройка (Add-on) и Sandbox script.** Эти методы использует сценарии на языках Lua или Python для расширения функциональных возможностей самого симулятора. Sandbox script могут запускаться автоматически и работать в фоновом режиме, либо вызываться как функции (например, удобно при написании сценариев импорта/экспорта). Надстройки не привязываются к определенной симуляции или модели. Они расширяют общие функции, среды CoppeliaSim.
3. **Плагины** – это метод расширения функциональных возможностей симулятора CoppeliaSim и способ программирования работы моделей и сцены на языках C/C++. Этот способ используется для предоставления CoppeliaSim специальной функциональности, требующей либо возможности быстрых вычислений (сценарии в большинстве случаев медленнее, чем компилируемые языки), либо специального интерфейса для аппаратного устройства (например, реального робота), либо специального коммуникационного интерфейса с внешним миром.
4. **Клиент удаленного API.** Данный подход к программированию позволяет внешнему приложению (например, расположенному на роботе, другом компьютере и т. д.) подключиться к среде CoppeliaSim и использовать ее функционал на основе команд удаленного API.
5. **Узел ROS** – метод, позволяющий внешнему приложению (например, расположенному на роботе, другой машине и т. д.) подключаться к CoppeliaSim через операционную систему роботов (Robot Operation System - ROS).
6. **Сетевой узел на базе TCP/IP, ZeroMQ и др.** Данный метод позволяет внешнему приложению (например, расположенному на роботе или на другом компьютере) подключаться к CoppeliaSim с помощью сетевых технологий.

## Сценарии CoppeliaSim

Для программирования действий и динамического изменения свойств объектов модели или сцены в среде CoppeliaSim используются сценарии различного типа (рис. 1).

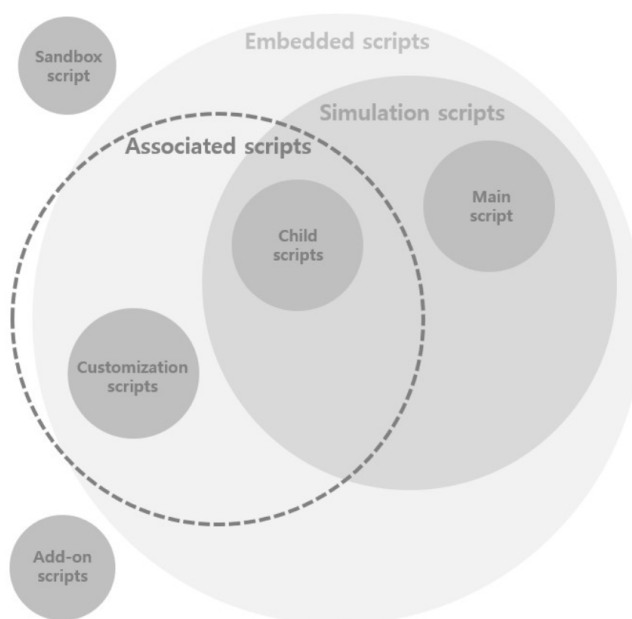


Рис. 1. Элементы и структура программных средства CoppeliaSim

Для выполнения сценариев используется механизм вызова callback-функций, который работает в следующей последовательности:

- Главный и дочерние сценарии моделирования (main and child scripts);
- Сценарии настройки сцены или модели (customization scripts);
- Sandbox scripts;
- Надстройки симулятора (add-on scripts).

Основной и дочерние сценарии сцены являются сценариями моделирования и работают только когда запущен процесс симуляции. Сценарии настройки, надстройки и sandbox работают с объектами сцены постоянно, даже когда моделирование остановлено. Кроме того, сценарии надстройки и sandbox продолжают работать даже после переключения на другую сцену. Вышеупомянутая последовательность имеет смысл, поскольку сценарии родительских объектов могут быть спроектированы так, чтобы полагаться на данные, созданные сценариями дочерних объектов, и работать с ними. Если сцена имеет несколько дочерних сценариев, то в первую очередь выполняются сценарии объектов на концах ветвей иерархии, а в последнюю запускаются сценарии корневых объектов.

Сценарии моделирования играют центральную роль в моделировании и выполняются только во время выполнения имитации. Существует два типа сценариев моделирования:

- **Основной сценарий.** По умолчанию каждая сцена имеет основной сценарий, который обрабатывает всю функциональность (то есть отвечает за вызов дочерних скриптов). Без основного сценария симуляция не запустится.

Основной сценарий сцены не связан с ее объектами. Изменение его программы возможно, но нежелательно. Для программирования действий объектов сцены или моделей в Coppelia Sim следует опираться на дочерние сценарии.

- **Дочерние сценарии.** Каждый объект сцены может быть связан с дочерним сценарием, который выполняет определенную часть моделирования. Чаще всего дочерний сценарий используется для управления моделью, состоящей из нескольких объектов (например, роботом). Дочерние сценарии привязаны к объектам сцены. Поэтому они дублируются во время операции копирования и вставки, что является важной функцией, позволяющей легко масштабировать сцену моделирования. Связанные дочерние сценарии составляют основу распределенной архитектуры управления CoppeliaSim. Их количество неограниченно.

Основной и дочерний сценарии по умолчанию обычно делятся на 4 callback-функции:

- **Функция инициализации: sysCall\_init.** Эта callback-функция не является обязательной. Для основного сценария она будет выполнена один раз только в начале симуляции. Ее код отвечает за подготовку симуляции. В случае дочернего сценария данная функция вызывается единственный раз при первом вызове дочернего сценария.
- **Функция действия объекта/модели: sysCall\_actuation.** Эта часть основного или дочернего сценария будет выполняться на каждом шаге моделирования во время фазы выполнения действия объектом или моделью.
- **Функция регистрации сигналов датчика: sysCall\_sensing.** Выполняется на каждом шаге симуляции, во время фазы считывания сигнала датчиков.
- **Функция очистки (восстановления состояния объекта): sysCall\_cleanup.** Эта часть будет выполняться один раз непосредственно перед окончанием симуляции или перед уничтожением сценария.

Сценарий настройки (customization script) связаны с объектом сцены и работают независимо от того запущена симуляция или нет. Они используются для конфигурирования среды моделирования.

## Особенности языка Lua

Лексика Lua:

- язык чувствителен к регистру при обозначении переменных;
- ключевые слова: **and, break, do, else, elseif, end, false, for, function, if, in, local, nil, not, or, repeat, return, then, true, until, while;**
- операторы: **+ - \* / % ^ # == ~= <= >= < > = ( ) { } [ ] ; : , .. ...**
- Текстовые строки могут быть разделены одинарными или двойными кавычками (например, 'Текст', "Текст");

- Комментарий начинается с двойного дефиса в любом месте строки команды. Например:

***a=4 --А это уже комментарий после присвоения переменной a значения 4***

Типы и значения:

- Lua является языком с динамической типизацией. Это означает, что тип переменных определяется по присваиваемому значению, а не назначается посредством ключевых слов;
- Язык имеет 8 основных типов данных:
  - ✓ **nil** – отсутствие какого-либо значения;
  - ✓ **bool** – логический тип переменных, которые могут принимать значения **false** или **true** (и **nil**, и **false** делают условие ложным; любое другое значение делает его истинным);
  - ✓ **number** – обозначает как целые числа, так и числа с плавающей запятой (внутри имеет два различных представления: целое число и число с плавающей запятой);
  - ✓ **string** - массивы символов (строки могут содержать любой 8-битный символ, включая встроенные нули);
  - ✓ **function** - тип, обозначающий функцию на языке Lua;
  - ✓ **userdata** – переменная этого типа может содержать произвольные данные в нотации языка «C» (соответствует блоку необработанной памяти);
  - ✓ **thread** – переменная этого типа представляет независимый поток выполнения, используемый для реализации сопрограмм;
  - ✓ **table** - массивы, которые могут содержать значения любого типа, кроме **nil**;

Переменные:

- существует 3 типа переменных: **глобальные переменные**, **локальные переменные** и **поля таблицы**. Любая переменная считается глобальной, если она явно не объявлена как локальная.
- перед первым присвоением переменной ее значение равно нулю;
- квадратные скобки используются для индексации таблицы (например, *value=speedMotor[x]*). Первое значение в таблице находится в позиции 1 (а не 0, как для массивов языка C).

Выражения:

- Lua допускает множественные присвоения значений переменным. Синтаксис присваиваний определяет список переменных слева и список значений справа. Элементы в обоих списках разделены запятыми:

***x,y,z = myTable[1],myTable[2],myTable[3]***

- Операторы отношений (**==** оператор равно; **~=** не равно; **<** меньше; **>** больше; **<=** меньше или равно; **>=** больше или равно) всегда дают **false** или **true**;
- Структура оператора **if** (на основе примера):

***if value1==value2 then***

```
print('value1 and value2 are same!')  
end
```

- Структура оператора **for**:

```
for i=1,4,1 --счетчик от 1 до 4 с шагом 1  
print(i)  
end
```

- Структура оператора **while**:

```
i=0  
while i~=4 do  
  i=i+1  
end
```

- Структура оператора **repeat**:

```
i=0  
repeat  
  i=i+1  
until i==4
```

- Структура оператора **table**:

```
myTable={'firstValue',2,3} -- построить таблицу с 3 переменными  
print(myTable[1]) -- напечатать 1 элемент таблицы  
table.insert(myTable,4) -- добавить 4 элемент таблицы
```

- Операция соединения строк:

```
a='hello'  
b='world'  
c=a..b -- c содержит строку 'hello world'
```

- Операция определения размера переменной и таблицы:

```
stringLength=#'hello world'  
tableSize=#{1,2,3,4,5}
```

Побитовые операторы:

- **&** – побитовое логическое «И» (AND);
- **|** – побитовое логическое «ИЛИ» (OR);
- **>>** – сдвиг вправо
- **<<** – сдвиг влево
- **~** – побитовая инверсия (отрицание, NOT).

## 2. Задание

### 2.1. Обучающая часть

**Проект «Движение трехопорного робота вдоль линии».** Разработать сценарий на языке Lua, реализующий релейное управление движением трехопорного робота вдоль линии на основе сигнала датчика освещенности.

1. Создайте новую сцену и дайте ей название в соответствии с шаблоном: **Work2(<Фамилия студента-группа (на латинице)>-task1.ttt**. Например, студент Иванов из группы 1101б должен именовать файл сцены CoppeliaSim: **Work2(Ivanov-1101b)-task1.ttt**

## Крепление и конфигурирование датчика освещенности

2. Откройте проект Work1(...)-task-1.ttt и скопируйте все элементы трехопорного робота в проект Work2(...)-task-1.ttt.
3. Постройте крепление для датчика освещенности и поместите его в передней части платформы робота. Для этого создайте кубоид с размерами (**1e-2; 4e-2; 1e-3**) и поместите его в точку с мировыми координатами (**0; 7e-2; 3.76e-2**). Назовите новый объект «*SensorSupport*» и сделайте его дочерним по отношению к объекту «*Base\_visual*». «*SensorSupport*» отнесите к первой группе визуального слоя, отключите у него динамические и специальные свойства, а спектральные характеристики (цвет поверхности) настройте произвольным образом.
4. Создайте оптический сенсор и поместите его на крепление. Для этого щелкните в рабочей области сцены правой кнопкой мыши и во всплывающем меню выберите пункт «*Add – Vision Sensor – Perspective type*». Появится цветной кубоид и скелет призмы, изображающей область наблюдения датчика. Назовите объект «*Visian\_sensor*» и сделайте его дочерним по отношению к объекту «*SensorSupport*». Линия, которая будет направлять движение робота во время симуляции, будет находиться на поверхности сцены. Поэтому датчик следует перевернуть так, чтобы в его область наблюдения попадала поверхность сцены. Используя инструмент поворота переверните объект «*Visian\_sensor*» на **180** градусов вокруг оси X. Затем поместите сенсор в точку с мировыми координатами: (**0; 8.5e-2; 3.2e-2**). Откройте свойства объекта «*Visual\_sensor*» и измените его размеры с помощью полей «*Object size X – Y – Z*» в группе свойств «*Visual properties*»: (**0.005; 0.005; 0.007**). В этом же диалоговом окне в группе свойств «*Main properties*» измените поля «*Near / far clipping plane*» (Расстояние от сенсора до ближней и дальней плоскостей, ограничивающих область наблюдения): (**1e-2; 3.3e-2**) и «*Persp. Angle [deg]*» (Угол обзора в градусах): **15**. Указанные параметры определяют границы области наблюдения, изображаемые в сцене скелетными линиями цвета датчика. В данном случае область наблюдения представляет собой призму. Контролируемый датчиком объект обязательно должен находиться в пределах этой призмы. Поэтому ее основание должно быть немного дальше от датчика, чем поверхность сцены, на которой будет изображена направляющая линия. Наибольший размер основания призмы следует выбирать так, чтобы он был меньше ширины линии. Установите поля свойства «*Resolution X / Y*» равными (**1; 1**) соответственно. Это позволит получить из цветной видеокамеры простой фотоэлемент датчика цвета. В сценарии управления роботом параметры цвета будут преобразованы к уровню яркости (освещенности), что фактически сделает используемый сенсор датчиком освещенности. Для свойства «*Entity to detect*» (Объект для обнаружения) установите значение «**all renderable objects in the scene**» (Все визуализируемые объекты на сцене). Наконец, произвольным образом настройте спектральные характеристики оптического датчика с помощью свойств: «*Adjust color (passive)*» и «*Adjust color (active)*». По окончании настройки робот со смонтированным креплением и датчиком дол-

жен выглядеть аналогично рисунку 1. При запуске симуляции робот вместе с датчиком должен двигаться по поверхности сцены. Закройте диалоговое окно свойств объекта «*Visian\_sensor*».

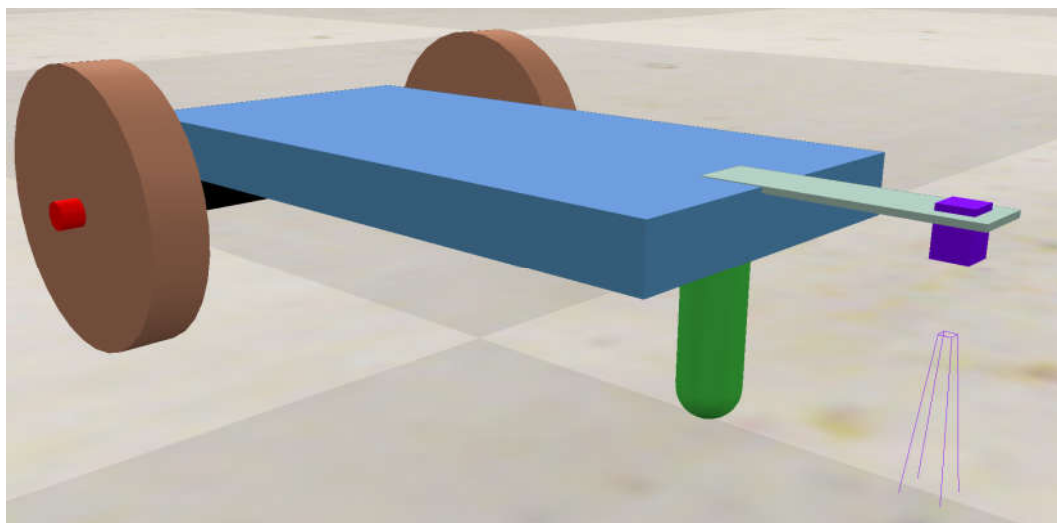


Рис. 1. Трехопорный робот после монтажа датчика

Для отслеживания сигнала датчика щелкните правой кнопкой мыши в рабочей области сцены и во всплывающем меню выберите пункт «*Add – Floating view*». Появится новое (пустое) окно. Щелкните внутри него правой кнопкой мыши и во всплывающем меню выберите пункт «*View – Associate view with selected visual sensor*». При заданных параметрах объекта «*Visual\_sensor*» в этом окне будет изображен цвет, усредненный датчиком по области наблюдения. Потянув за правый нижний угол окна «*Visual\_sensor*» можно уменьшить его до нужных размеров. При запуске симуляции и движении робота цвет, регистрируемый датчиком, должен меняться.

#### Проецирование рисунка трассы на поверхность сцены

5. Поверхность сцены представлена в проекте двумя объектами «*Floor*» и «*Floor\_visible*». Первый используется для моделирования взаимодействия с другими (динамическими) объектами сцены, а второй отвечает за изображение на поверхности сцены. Объект «*Floor\_visible*» является результатом группировки множества объектов и перестроить текстуру на его поверхности довольно сложно. Поэтому упростим задачу. Используя свойства объекта «*Floor\_visible*» можно определить его геометрические характеристики, в том числе размеры по X и Y: (5e+0; 5e+0). Добавьте на поверхность сцены новый объект типа «*Plane*» (Плоскость). Для этого во всплывающем меню выберите пункт «*Add – Primitive Shape - Plane*». В появившемся диалоговом окне укажите значения по X и Y 4e+0 и 4e+0 соответственно. Размер у объектов типа «*Plane*» по оси Z равен нулю. Нажмите «Ok» для построения объекта. Созданному объекту дайте название «*Path\_plane*» и поместите в точку с мировыми координатами (0; 0; 1e-3). Откройте диалоговое окно свойств объекта «*Path\_plane*» и отнесите его к 8 группе визуального слоя. Из специальных свойств объекта оставьте включенными только свойства «*Detectable*» и «*Renderable*». В диалоговом окне управления динамическими


свойствами объекта «*Path\_plane*» нужно снять флажок «*Body is dynamic*», но установить «*Body is respondable*» (Тело отвечает на воздействие других динамических тел). Последнее действие позволит роботу двигаться непосредственно по поверхности объекта «*Path\_plane*».

6. Для проецирования изображения трассы робота на поверхность объекта «*Path\_plane*» используйте файл «*PathRobot.png*», прилагаемый к заданию настоящей практической работы. В окне свойств объекта «*Path\_plane*» на закладке «*Shape*» нажмите кнопку «*Adjust texture*». Появится диалоговое окно «*Shape*». В нем нажмите кнопку «*Load new texture*». Далее выберите файл «*PathRobot.png*» и нажмите «*Ok*». Теперь в диалоговом окне «*Texture Load Option*» установите флажок «*Scale texture to*», выберите «*max. 2048x2048*» и нажмите «*Ok*». Изображение «*PathRobot.png*» размером 2048x2048 пиксел будет спроецировано на площадку объекта «*Path\_plane*», размеры которой составляют 1x1 метр (1 м<sup>2</sup>). Чтобы растянуть изображение трассы на всю поверхность объекта «*Path\_plane*» нужно в окне «*Shape*» свойствам «*Along U*» и «*Along V*» (коэффициенты масштабирования по осям X и Y) присвоить значение 4. В результате выполнения описанных операций должна сформироваться сцена, аналогичная рисунку 2.



Рис. 2. Изображение трассы на поверхности сцены

### Сценарий работы робота

7. В окне иерархии сцены щелкните на корневом объекте «*Base*» правой кнопкой мыши и во всплывающем меню выберите пункт «*Add – Associated child script – Non threaded*». Рядом с названием объекта должна появиться пиктограмма . Щелкните по ней. Откроется окно редактирования сценария, в котором по умолчанию содержатся четыре пустых callback-функции.
8. Для управления мобильным роботом в функции **sysCall\_init()** нужно инициализировать дескрипторы (указатели на контролируемый объект) сле-



дующих объектов: датчик контроля освещенности («*Vision\_sensor*»), двигатель левого колеса («*Left\_joint*»), двигатель правого колеса («*Right\_joint*»). Данное действие выполняется с помощью функции **getObjectHandle**, которая определена в модуле **sim**. В итоге тело функции **sysCall\_init()** должно содержать следующие строки:

```
--получить дескриптор объекта 'Vision_sensor' и поместить его в переменную
--hSensor
hSensor=sim.getObjectHandle('Vision_sensor');
--получить дескриптор мотора левого колеса и поместить его в переменную
--hLMotor
hLMotor=sim.getObjectHandle('Left_joint');
--получить дескриптор мотора правого колеса и поместить его в переменную
--hRMotor
hRMotor=sim.getObjectHandle('Right_joint');
```

9. Теперь запрограммируем считывание сигнала с датчика освещенности. Для получения и обработки данных в процессе симуляции предназначена функция **sysCall\_sensing()**. Она вызывается из основного сценария на каждом шаге моделирования с интервалом по умолчанию  $dt=50$  мс. Получить результаты измерения оптического сенсора в виде таблицы помогает библиотечная функция языка Lua **sim.getVisionSensorImage**. В случае, когда фотодатчик имеет разрешение 1x1 элемент, в функции **sysCall\_sensing()** достаточно использовать две строки кода:

```
--считать данные сенсора в таблицу im
im = sim.getVisionSensorImage(hSensor);
--присвоить переменной level значение уровня сигнала «красного» слоя
--изображения, умноженного на 1000. По умолчанию сигнал сенсора
--изменяется в пределах от 0 до 1. Индекс 1 (im[1]) таблицы
--соответствует R (red), 2 – G(green), 3 – B(blue).
level=im[1]*1000;
```

10. При обработке дочернего сценария по окончании работы **sysCall\_sensing()** последует вызов функции **sysCall\_actuation()**. Код этой функции реализует алгоритм управления роботом, который опирается на значения переменных, обновленных в функции **sysCall\_sensing()**. Для управления мобильным роботом используем простейший метод **релейного регулирования**. Суть его заключается в одновременном включении мотора только одного колеса. Какое из колес включено определяется в результате сравнения текущего уровня освещенности с заданным порогом. Например, если уровень сигнала превышает значение 500, то следует включить левое колесо и выключить мотор правого колеса. Если же уровень сигнала датчика будет ниже 500, то нужно включить правое колесо и выключить мотор левого. В описанном случае, когда датчик «видит» белое поле слева от линии, робот совершает поворот

направо, а датчик по дуге приближается к черной линии. При попадании линии в область наблюдения датчика уровень его освещенности начнет понижаться пропорционально доли черного в области наблюдения. Когда текущий уровень освещенности станет меньше 500 нужно переключить моторы и заставить робота по дуге двигаться от черной линии на белое поле. Регулярное считывание сигнала датчика и переключение двигателей робота приведет к его поступательно-колебательному движению. В целом робот будет двигаться вдоль линии, а его датчик будет совершать колебания около границы белого поля и черной линии. Для управления моторами колес используйте функцию **sim.setJointTargetVelocity()**, а сравнение текущего уровня освещенности с заданным порогом (500) следует выполнять с помощью оператора **if**. Чтобы реализовать описанный выше алгоритм релейного управления мобильным роботом поместите в **sysCall\_actuation()** приведенный ниже код:

```
if level>500 then
    sim.setJointTargetVelocity(hLMotor,speed);
    sim.setJointTargetVelocity(hRMotor,0);
else
    sim.setJointTargetVelocity(hLMotor,0);
    sim.setJointTargetVelocity(hRMotor,speed);
end
```

11. Переменные **speed** и **level** перед использованием в callback-функциях дочернего сценария требуется определить как глобальные и инициализировать в функции **sysCall\_init()**. Полный код дочернего сценария представлен на рисунке 3.

```
function sysCall_init()
    hSensor=sim.getObjectHandle('Vision_sensor');
    hLMotor=sim.getObjectHandle('Left_joint');
    hRMotor=sim.getObjectHandle('Right_joint');
    speed=-12; --Wheel speed [radians per second]
    level=1000;--Initial light level
end

function sysCall_actuation()
    if level>500 then
        sim.setJointTargetVelocity(hLMotor,speed);
        sim.setJointTargetVelocity(hRMotor,0);
    else
        sim.setJointTargetVelocity(hLMotor,0);
        sim.setJointTargetVelocity(hRMotor,speed);
    end
end

function sysCall_sensing()
    im=sim.getVisionSensorImage(hSensor);
    level=im[1]*1000;
end

function sysCall_cleanup()
    -- do some clean-up here
end
```

Рис. 3. Код дочернего сценария трехопорного мобильного робота

По завершении, описанных выше действий, сценарий управления роботом готов и окно его редактора можно закрыть.

12. Поместите мобильного робота недалеко от трассы, чтобы датчик находился слева от нее по ходу движения. Запустите симуляцию и протестируйте работу робота. Робот должен «увидеть» линию и начать движение вдоль нее.
13. Если робот «теряет» линию и меняет направление движения вдоль нее на противоположное, уменьшите значение переменной **speed**. При правильно собранной конструкции робот должен двигаться вдоль линии неограниченно долго. На устойчивость движения робота вдоль трассы влияет размер основания области наблюдения датчика. По регулируйте величину угла проекции датчика и понаблюдайте как ведет себя робот на крутых поворотах и прямолинейных участках трассы. С каким углом проекции датчика можно добиться максимальной скорости движения робота вдоль направляющей линии? Каково максимальное значение скорости робота, с которой ему удастся проходить трассу без «потери» направляющей?

#### Создание модели мобильного робота

14. В первой практической работе построена конструкция мобильного трехопорного робота, которая представлена иерархически связанным набором объектов с визуальными и динамическими характеристиками. Выполнение шагов настоящей практической работы способствовало программированию реакции робота на воздействие окружающей среды. Таким образом, совокупность свойств конструкции и программных средств робота представляют его в виде законченной модели в среде CoppeliaSim. Подобные наборы объектов сцены можно экспортировать в библиотеку автономных моделей и строить произвольное количество экземпляров в любой сцене CoppeliaSim.
15. Для организации набора объектов трехопорного робота в виде автономной модели выберите корневой элемент «*Base*», откройте диалоговое окно его свойств и на закладке «*Common*» в группе «*Model definition*» (Определение модели) пометьте флажок «*Object is model base*» (Объект является модельной базой). Затем нажмите кнопку «*Edit model properties*». В появившемся диалоговом окне «*Model properties*» (Свойства модели) никакие флажки устанавливать не требуется, но следует нажать кнопку «*Select model thumbnail*» (Выберите миниатюру модели). В окне «*Model thumbnail image*» (Миниатюрное изображение модели) нажмите кнопку «*No*» и с помощью кнопок вращения и сдвига в окне «*Model thumbnail selection*» (Выбор эскиза модели) отрегулируйте вид пиктограммы созданного мобильного робота, которой он будет представлен в библиотеке моделей среды CoppeliaSim (рис. 4). Нажмите кнопку «*Ok*». Закройте диалог «*Model thumbnail selection*» кнопкой «*Close*» и окно свойств объекта «*Base*».

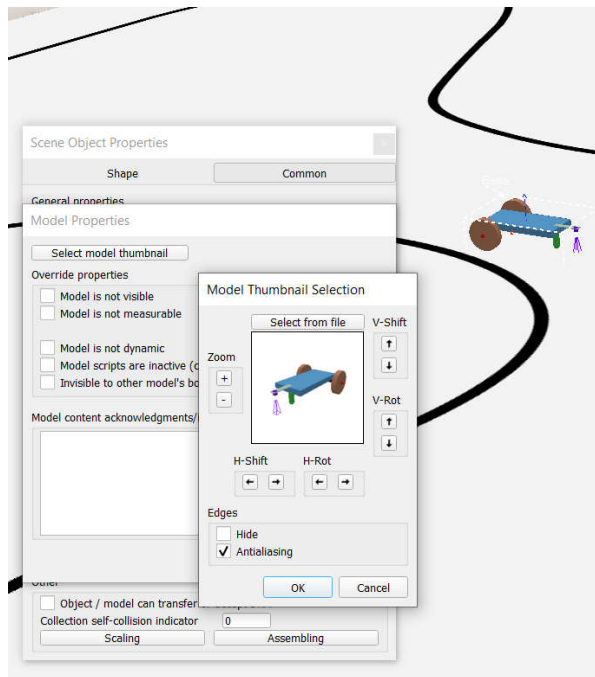



Рис. 4. Эскиз модели разработанного трехопорного робота

16. При выбранном корневом элементе «Base» выберите пункт основного меню «File – Save model as – Coppeliasim model...». В появившемся диалоговом окне «Model thumbnail image» щелкните «Yes», а в окне «Saving model...» выберите папку «...\CoppeliasimRobotics\CoppeliasimEdu\models\robots\mobile», дайте название файлу с расширением «.ttm» и нажмите кнопку «Сохранить».
17. На левой панели инструментов в главном окне Coppeliasim нажмите пиктограмму , чтобы открыть «Model browser». В списке браузера выделите пункт «robots - mobile» и в коллекции найдите модель построенного трехопорного робота (рис. 5).

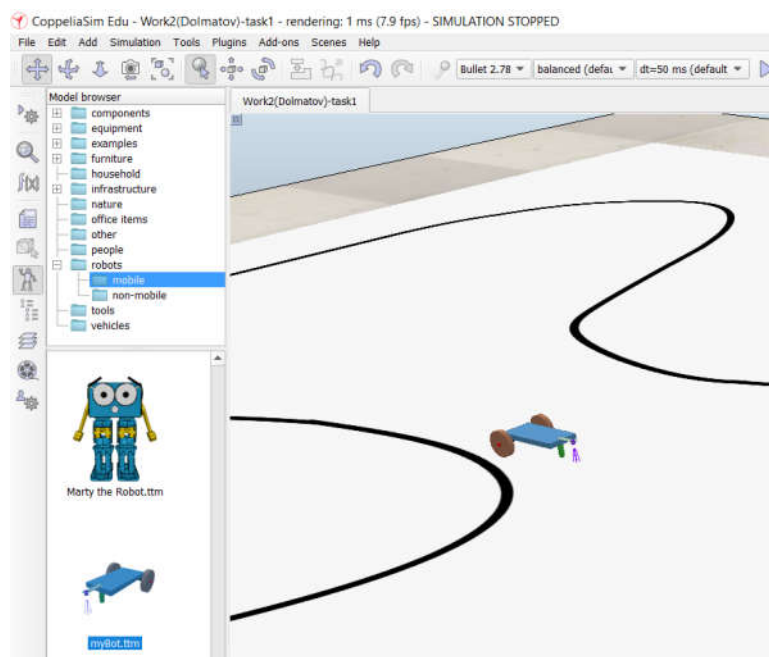


Рис. 5. Модель трехопорного робота в коллекции Coppeliasim

18. Из коллекции CoppeliaSim с помощью мыши перетащите несколько экземпляром модели трехопорного робота на сцену разрабатываемого проекта и поместите в разных местах трассы около направляющей линии. Запустите симуляцию. Понаблюдайте за поведением каждого экземпляра. Что произойдет, если направить роботов по трассе навстречу друг другу? Есть ли ограничение на количество экземпляров модели на сцене?
19. Сохраните разработанный проект сцены CoppeliaSim в файле и продемонстрируйте результаты работы преподавателю.

## 2.2. Самостоятельная работа

**Проект «Мобильный робот с двумя датчиками освещенности».** Построить модель мобильного трехопорного робота, осуществляющего релейное регулирование движения вдоль направляющей линии по двум датчиками освещенности.

1. Создайте новую сцену и дайте ей название в соответствии с шаблоном: Work2(<Фамилия студента-группа (на латинице)>)-task2.ttt.
2. Скопируйте объекты трехопорного робота из проекта Work2(<Фамилия студента-группа (на латинице)>)-task1.ttt на сцену разрабатываемого проекта.
3. Закрепите в передней части платформы робота два датчика освещенности с разрешением 1x1 так, чтобы они располагались по обе стороны от линии трассы и их области наблюдения не пересекались.
4. Разработайте сценарий работы робота так, чтобы левый датчик освещенности контролировал вращение только левого колеса, а сигнал правого датчика использовался для управления вращением правого колеса. Уровень сигнала обоих датчиков освещенности должен изменяться в пределах от 0 до 255, а порог включения моторов колес должен составлять 150. Моторы могут иметь только два состояния, отличающиеся абсолютной скоростью вращения [радиан в секунду]: (0; 10). Таким образом, пока оба датчика «смотрят на белое» (начальное положение), робот должен двигаться прямо. Если в область наблюдения одного из датчиков попадает черная линия трассы, то соответствующий датчику мотор должен выключиться, и способствовать возвращению робота в начальное положение.
5. Добейтесь максимальной скорости устойчивого движения робота по трассе с помощью регулировки расстояния между датчиками.
6. По окончании тестирования постройте автономную модель робота и разместите ее в коллекции CoppeliaSim.
7. Пропредмонстрируйте преподавателю одновременную работу на трассе нескольких экземпляров разработанной модели мобильного робота с двумя датчиками освещенности.

## 2.3. Индивидуальное задание

***Проект «Управление четырехколесным мобильным роботом с поворотным передним мостом на основе двух датчиков освещенности».***

1. Создайте новую сцену и дайте ей название в соответствии с шаблоном: Work2(<Фамилия студента-группа (на латинице)>)-task3.ttt.
2. Скопируйте объекты четырехколесного робота из проекта Work1(<Фамилия студента-группа (на латинице)>)-task2.ttt на сцену разрабатываемого проекта.
3. С помощью графического редактора разработайте оригинальную трассу для мобильного робота в виде растрового изображения размером 2048x2048 элементов. Трасса должна включать минимум 3 правых и 3 левых поворота. Нанесите разработанное изображение трассы на поверхность сцены.
4. Закрепите в передней части платформы робота два датчика освещенности с разрешение 1x1 так, чтобы они располагались по обе стороны от линии трассы и их области наблюдения не пересекались.
5. Разработайте сценарий работы робота, который позволит поворачивать колеса переднего моста в сторону датчика, обнаружившего черную линию трассы. После выхода линии трассы из поля зрения датчика колеса должны возвращаться в первоначальное положение, обеспечивающее движение робота прямо. Для управления углом поворота рулевого шарнира в сценарии используйте функцию `sim.setJointTargetPosition()`. При повороте переднего моста колеса не должны касаться платформы. Привод робота должны осуществлять колесами переднего моста, вращающиеся с одинаковой скоростью. Колеса заднего моста должны вращаться свободно.
6. Подрегулируйте свойства объектов мобильного робота, чтобы добиться устойчивого движения по трассе.
7. Создайте автономную модель робота и разместите в коллекции CoppeliaSim.
8. Продемонстрируйте преподавателю работу экземпляров модели на оригинальной трассе.

## 3. Подготовка отчета, представление и оценка работы

### Структура отчета

В качестве отчета по каждой части задания необходимо предоставить готовый проект CoppeliaSim. В отчете (проекте CoppeliaSim) оценивается точность параметров модели (названий, значений, иерархии и др.), для которых в задании определена уникальная величина или идентификатор. Произвольно определяемые объекты или их параметры могут иметь любое значение. В задании они оговариваются отдельно.

Загрузку проектов на сайт Eluniver следует выполнять после демонстрации всех частей задания преподавателю. Желательно загружать все проекты одновременно либо отдельными файлами, либо общим архивом.

### Представление и защита работы

Представлением работы является ее демонстрация преподавателю. Каждый проект может быть представлен отдельно от других частей задания. В ходе представления преподаватель может задать вопрос по любому пункту соответствующей части задания или попросить выполнить какие-либо построения на основе навыков, полученных при разработке проекта. Оценка за представление задания выставляется на основе работоспособности проекта, правильности ответа студента на вопросы по проекту и готовности выполнить дополнительное задание без использования методического материала.

Защита работы заключается в ответе на два контрольных вопроса, выбранных произвольно преподавателем из списка контрольных вопросов (п. 4). Оценивается детальность и точность ответа. Во время ответа пользоваться методическим материалом нельзя. Возможность ответа на контрольные вопросы дается студенту после представления всех частей задания.

### Структура оценки практической работы

№	Вид оценки	Максимальный балл
1.	Проект «Обучающая часть»	10
2.	Проект «Самостоятельная работа»	15
3.	Проект «Индивидуальное задание»	25
4.	Отчет «Обучающая часть»	5
5.	Отчет «Самостоятельная работа»	5
6.	Отчет «Индивидуальное задание»	10
7.	Контрольный вопрос 1	15
8.	Контрольный вопрос 2	15
<b>Итого:</b>		<b>100</b>

## **4. Контрольные вопросы**

1. Какие языки программирования используются во встроенных сценариях?
2. Какие типы сценариев применяются в среде CoppeliaSim?
3. Для каких целей в среде CoppeliaSim используются плагины?
4. Какой тип сценарием продолжает работать даже после переключения между сценами?
5. С помощью каких инструментов возможно управление роботом из среды CoppeliaSim?
6. Для чего предназначены дочерние сценарии объектов сцены?
7. Как называется механизм, используемый для вызова дочернего сценария?
8. В какой последовательности вызываются дочерние сценарии, привязанные к нескольким объектам сцены?

9. Какие функции по умолчанию содержатся в дочернем сценарии?
10. Для чего предназначена функция *init* в сценарии?
11. Как называются функции сценария, которым передается управление роботом на регулярной основе?
12. Какая функция сценария вызывается при завершении работы симуляции?
13. Какие типы переменных используются в языке Lua?
14. Как определяется тип данных в языке Lua?
15. Для чего предназначен тип *table* в языке Lua? Какой тип данных он может хранить?
16. Какие ключевые слова языка Lua применяются для описания алгоритмической конструкции «если ...то ...иначе»?
17. Для чего используется ключевое слово *repeat* в языке Lua?
18. Сколько видов циклов можно задействовать в сценариях на языке Lua?
19. Какова структура цикла с постусловием в языке Lua?
20. Что называется телом цикла *while*?
21. Сколько итераций будет выполнено в цикле *for i=132,4,-2* ?
22. Каков результат операции  $16 \ll 2$ ?
23. Для чего в языке Lua применяются побитовые операции? Приведите пример.
24. Как включить Lua-commander в среде CoppeliaSim?
25. Каким образом можно привязать к объекту сцены дочерний сценарий?
26. Как получить доступ к программному управлению объектом сцены?
27. Каковы принципы релейного управления движением мобильного робота?
28. Как оборудовать робота в среде CoppeliaSim датчиком освещенности?
29. За что отвечает свойство визуального датчика «Near/far clipping plane»?
30. Что такое разрешение визуального датчика? Как оно настраивается в среде CoppeliaSim?
31. Какие свойства визуального датчика задают его область наблюдения?
32. Как в Lua-программе получить данные с визуального сенсора?
33. Что станет результатом операций: `im=sim.getVisionSensorImage(hSensor); data=im[3]*1000; ?`
34. Как в среде CoppeliaSim выполняется построение модели и размещение ее в стандартной библиотеке?
35. После создания нескольких экземпляров модели робота каждая из них будет иметь собственный сценарий работы или для всех будет действовать единый сценарий?
36. Каким образом осуществляется настройка пиктограммы модели робота, которая отображается в стандартной библиотеке?
37. Каков принцип управления мобильным роботом на основе двух датчиков освещенности?
38. Какой критерий в программе управления определяет поведение робота?
39. Возможна ли при симуляции работы робота ситуация когда он вопреки работы релейного регулятора движется от направляющей линии?
40. Каким образом формируется трасса робота на поверхности сцены?