Санкт-Петербургский государственный университет
информационных технологий, механики и оптики
Факультет информационных технологий и программирования
Кафедра компьютерных технологий

С. А. Алексеев

Программно-аппаратный комплекс для исследования автоматного управления мобильными роботами

Бакалаврская работа

Руководитель: В. О. Клебан

Санкт-Петербург
2010

Оглавление

Введение

Роботы – это физические агенты, которые выполняют поставленные перед ними задачи, проводя манипуляции в физическом мире. Управление роботами является сложной задачей. Большой интерес вызывает автоматный подход к управлению роботами [1, 2, 3]. Для исследования и проектирования автоматных программ управления мобильным роботом предложен специальный программно-аппаратный комплекс. Важной его особенностью является то, что он содержит удобную реализацию графического языка автоматного программирования [4], позволяющую проектировать различные системы управления. При решении задачи управления мобильным роботом возникает ряд проблем. Наиболее важной проблемой является необходимость в специальной сцене, построить которую в реальном мире занимает много времени и средств. Также во время реальных испытаний есть вероятность повредить оборудование. Для решения этих проблем предлагается использовать эмулятор реального робота и среды окружения. На практике достаточно часто встречаются задачи, для которых известно, что они могут быть решены при помощи конечных автоматов, но эвристически построить для них автомат чрезвычайно сложно. Попытки решения подобных задач выполняются, в частности, при помощи различных методов автоматического синтеза программ. Одним из таких методов является генетическое программирование [5]. Для применения генетического программирования необходим «быстрый» эмулятор робота, так как получение необходимого числа значений функции приспособленности для реального робота требует больших временных затрат.

1. Постановка задачи

В рамках данной работы ставилась задача создать эффективный комплекс для сквозного проектирования и отладки автоматных систем управления мобильными роботами. В состав комплекса должны входить:

- реальный робот;
- графический язык проектирования автоматных программ;
- модель робота в среде твердотельного моделирования;
- среда эмуляции и способ конвертирования модели робота в объект этой среды;
- контроллер робота, с функцией отладки по шагам и функцией интерпретирования графической программы;
- протокол связи;

Система сквозного проектирования должна предоставлять пользователю возможность осуществлять полный цикл создания и отладки мобильного робота, от чертежей деталей до программного обеспечения.

Для построения такого комплекса, целесообразно использовать технологию автоматного программирования [6]. Применение автоматного подхода при создании подобных систем управления обладает рядом достоинств: документируемость, возможность верификации, упрощение внесения изменений и т. д.

2. Обзор доступных лабораторий

На данный момент существует несколько лабораторий, предназначенных для эмуляции различных роботов и управления ими. Наиболее известные из них – это *Microsoft Robotics Developer Studio*, *Player/Stage/Gazebo* и *Webots*.

2.1. Описание лабораторий

2.1.1. *Microsoft Robotics Developer Studio*

Краткое описание

Microsoft Robotics Developer Studio (в дальнейшем *MRDS*) – продукт компании *Microsoft*, предназначенный для разработки в области робототехники. *MRDS* включает в себя несколько компонентов:

- *Visual Programming Language* (*VPL*) – визуальный язык программирования.
- Эмулятор робота и среды окружения.
- *Concurrency & Coordination Runtime* (*CCR*) – библиотека для работы с параллельными и асинхронными потоками данных.
- *Decentralized System Services* (*DSS*) – сервисный подход к созданию слабо связанных распределенных приложений.

Особенности лаборатории

Все компоненты в *Robotics Studio* представляют собой независимо исполняемые сервисы. Для разработчика не существует физических устройств – есть сервис с интерфейсом, с помощью которого можно изменять состояние устройства, или считывать с него информацию.

Среда, в которой выполняется приложение в *Microsoft Robotics Studio*, носит название *Runtime Environment*. В основе *Runtime* лежит *CLR 2.0*, что дает возможность писать приложения, используя любые языки программирования платформы *Microsoft .NET*.

Сам по себе *Runtime* состоит из двух ключевых технологий: *Concurrency & Coordination Runtime* (*CCR*) — это библиотека для работы с параллельными и асинхронными потоками данных, и *Decentralized System Services* (*DSS*) – средство создания распределенных приложений на основе сервисов.

К достоинствам симулятора можно отнести следующие:

- его достаточно легко начать использовать;
- подходит для обучения и исследований;
- сценический подход позволяет моделировать окружающий мир, создавая предметы и расставляя их в нужных местах;
- точное моделирование визуальной составляющей окружающего мира;
- симуляция физики поддерживается в полном объеме.

Ограничения эмулятора:

- мир идеализирован – нет искажения данных, как в реальном мире. Например, поверхность, по которой ездит робот, не может создавать трения. Однако при написании сервисов для конкретных «виртуальных» сенсоров можно добавлять шум, похожий на экспериментальный;
- описывая робота в симуляторе, можно постараться сделать его максимально похожим на реального, но модель всегда будет лишь приближением. Таким образом, в симуляторе приходится иметь дело с незаконченными или неточными моделями;
- точная настройка требует очень много времени;

- модель физики в *Microsoft Robotics Developer Studio* упрощена, поэтому симуляция не подойдет там, где требуются сверхточные расчеты.

2.1.2. *Player/Stage/Gazebo*

Краткое описание

Лаборатория *Player/Stage/Gazebo* предназначена, в основном, для исследования мульти-агентных систем и состоит из трех основных модулей:

1. *Player* – это сетевой сервер для управления роботами. Работая в роботе, *Player* предоставляет простой интерфейс ко всем сенсорам и устройствам робота через IP-сеть. Клиентская программа общается с *Player* через *TCP*-сокет, считывает информацию с сенсоров, пишет команды, и конфигурирует устройства. *Player* поддерживает различные составные части робота. Классическая платформа *Player* ориентирована на семейство роботов *Pioneer2*, но также поддерживается некоторое число других роботов, и множество сенсоров. Модульная архитектура *Player* позволяет легко добавлять поддержку новых устройств.

2 *Stage* – эмулятор популяции мобильных роботов, сенсоров и устройств на двумерной растровой карте. Stage разработан, чтобы поддержать исследования мульти-агентных автономных систем, поэтому он предоставляет множество устройств с простой моделью, вместо того, чтобы эмулировать каждое устройство с высокой точностью. Может быть использован, как *C++* модуль, для эмуляции роботов внутри пользовательского приложения.

3. *Gazebo* – эмулятор множества роботов в трехмерном окружении. Как и *Stage*, он позволяет эмулировать популяцию роботов, сенсоров и объектов, но делает это в трехмерном мире. Он генерирует достаточно реалистичную обратную связь сенсоров и физически правдоподобные взаимодействия между объектами.

Может быть использован в отдельности от сервера *Player*, взаимодействуя с приложением через библиотеку *libgazebo*.

Особенности лаборатории

Проект *Player/Stage* предоставляет два эмулятора: *Stage* и *Gazebo*. Ввиду их совместимости с *Player*, клиентские программы, написанные в одном эмуляторе, обычно могут быть запущены в другом с незначительными изменениями, или без них. Ключевое различие между этими двумя эмуляторами заключается в том, что *Stage* предназначен для эмуляции огромного числа роботов, с невысокой точностью, а Gazebo эмулирует небольшое число роботов, но с высокой точностью. Пользователи могут менять эмуляторы, с учетом их требований.

Stage и *Gazebo* обычно используются, как плагин к *Player*, предоставляя популяцию виртуальных устройств для *Player*. Это, так называемая, «Player/Stage» система. Пользователи пишут контроллеры роботов, как клиенты *Player*-сервера. Обычно, пользователи не могут заметить разницы между реальным роботом и их эмуляцией. Это позволяет экспериментировать на устройствах, которыми пользователи не обладают в действительности.

2.1.3. *Webots*

Краткое описание

Webots – среда разработки, предназначенная для моделирования, программирования и эмулирования мобильных роботов [7]. При помощи Webots, пользователь может проектировать сцены с различным числом объектов и роботов. Настройки объектов, такие, как масса, размер, цвет и т. д., также задаются пользователем. Благодаря большому разнообразию сенсоров и материалов, можно создавать любые модификации робота. Контроллеры робота могут быть запрограммированы с помощью интегрированной среды разработки. Возможности робота могут быть протестированы в физически реалистичном мире. Контроллер робота, при необходимости, может быть загружен в коммерческого робота *e-puck*.

Особенности лаборатории

Основной элемент разработки в *Webots* – это, так называемая, сцена. Она состоит из различных объектов окружения, настроек физических параметров виртуального мира, а также роботов. Настройки физических параметров достаточно гибкие, они позволяют исследовать поведение робота при различных условиях. Например, можно изменять силу трения, изменять состояние среды – эмулировать водные объекты, добавлять ветер и т.д. Для проектировки сцены существует встроенный редактор, он позволяет проектировать сцены с различными геометрическими объектами, а также создавать уникальные модели роботов. Важной особенностью *Webots* является возможность ускоренной эмуляции, что позволяет за короткий срок собрать важную информацию, например о поведении робота при различных

условиях. Встроенная среда разработки позволяет создавать контроллеры робота на различных языках программирования (*C++*, *Java* и т.д.).

2.2. Сравнительная таблица, рассмотренных лабораторий

Название лаборатории / Параметры, для сравнения	*MSRS*	*Player/Stage/ Gazebo*	*Webots*
Гибкость физических настроек	Низкая	Высокая	Высокая
Возможность проектирования произвольных роботов	Да	Да	Да
Удобство импортирования моделей из средств твердотельного моделирования	Низкое	Низкое	Низкое
Открытый исходный код	Нет	Да	Нет
Наличие визуального языка программирования	Да	Нет	Нет
Возможность загрузки контроллера в произвольного робота	Нет	Нет	Нет

3. Реализация комплекса

3.1. Архитектура комплекса

Комплекс включает в себя шесть основных частей:

- робота;
- систему управления с инструментом графического проектирования;

- контроллер робота (реального и виртуального);
- протокол связи;
- модель робота в среде твердотельного моделирования *Solidworks* [8] с возможностью конвертирования в объект среды *Webots*;
- эмулятор *Webots*;

Функционирование комплекса осуществляется за счет взаимодействия системы управления, контроллера робота и реального (виртуального) робота. Взаимодействие между системой управления и контроллером робота осуществляется по проводной или беспроводной связи при помощи команд протокола. В упрощенном смысле архитектура имеет вид, изображенный на рис. 1.

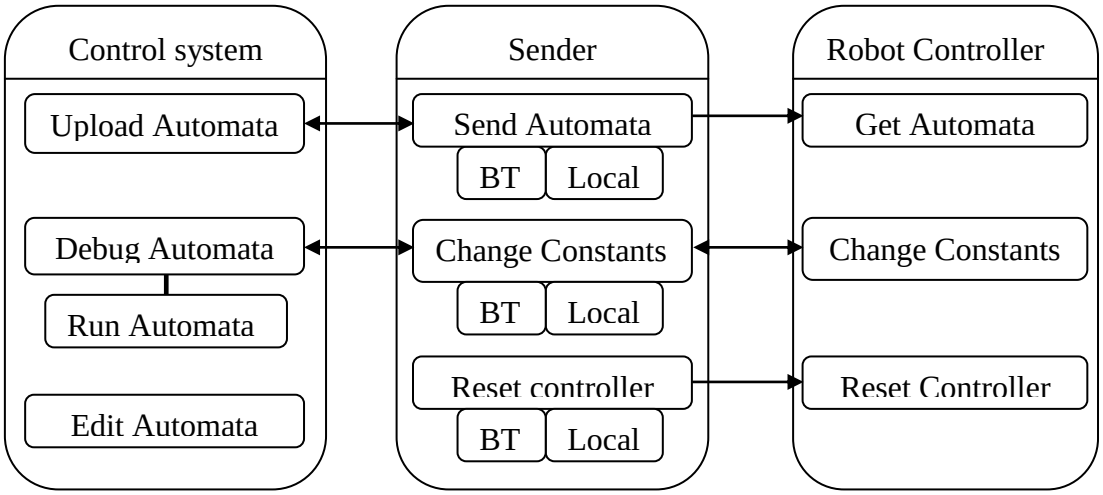


Рис. 1. Схема взаимодействия системы управления и контроллера робота

3.2. Робот

Объектом управления в данном комплексе является робот. Он включает в себя: аппаратную вычислительную платформу *Arduino*, выполненную на основе микроконтроллера *Atmel AVR*, два электромотора с редукторами и колесами, литий-ионный аккумулятор, дальномер, четыре датчика линии.

Для данного робота был выбрана готовая аппаратная платформа *Arduino Diecimila* (рис. 2), обладающий интерфейсом *RS-232* и микросхемой конвертера *USB-to-Serial FT232R*, благодаря которой, связь с ПК осуществляется через *USB*-интерфейс.

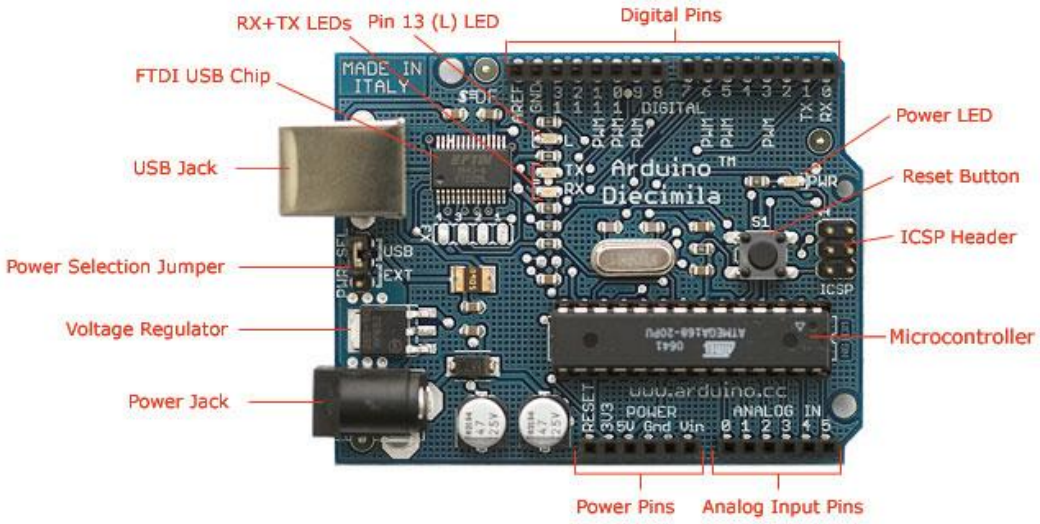


Рис. 2. Основные объекты вычислительной платформы *Arduino*

Впоследствии, к данной плате был припаян модуль *BTM-11-CS-96*, позволяющий осуществлять беспроводную связь с роботом, что увеличивает эффективность работы комплекса. Решение о выборе вычислительной платформы было сделано на основе ее функциональности и удобства использования. Вместе с вычислительной платформой также поставляется среда разработки *Arduino*, представляющая собой компилятор *C++* с некоторой дополнительной функциональностью, такой как наличие модуля *Serial Monitor*, который позволяет писать и читать из последовательного порта связи, наличие модуля *avrdude*, предназначенного для прошивки в микроконтроллер требуемой программы. Важными функциональными особенностями самой платы *Arduino Diecimila* являются возможность питания платы за счет соединения *USB*, возможность замены отказавшего микроконтроллера *Atmel AVR*, наличие *UART* интерфейса, позволяющего

добавить к платформе различные передающие модули. Помимо этого, документация и чертежи *Arduino* распространяются под лицензией «Creative Commons Attribution Share-Alike 2.5» и доступны на официальном сайте *Arduino.* Схемы печатной платы *Arduino* также доступны. Исходный код для интегрированной среды разработки и библиотек опубликован и доступен под лицензией «GPLv2».

Готовый робот изображен на рис. 3.

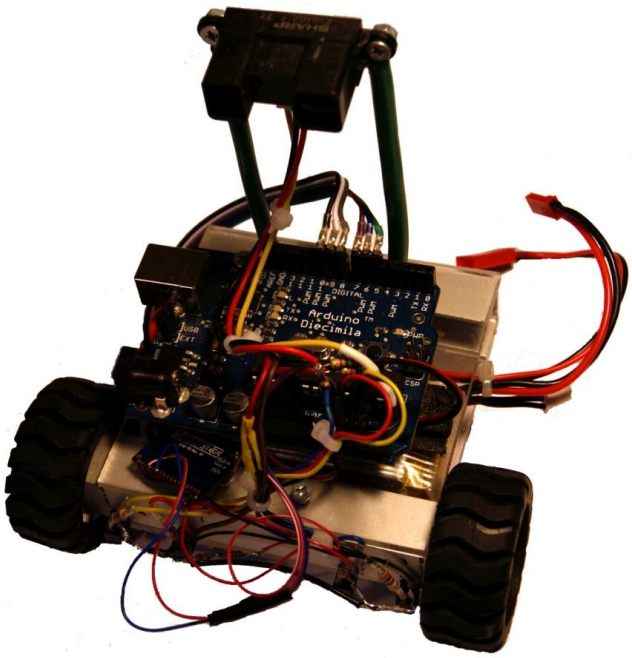


Рис. 3. Фотография реального робота

3.3. Контроллер

Управление роботом осуществляется за счет изменения различных переменных, таких как скорость вращения двигателей, направление их вращения и т. д. Как реальным роботом, так и виртуальным, управляет специально разработанные контроллеры. Они имеют два режима работы. Сразу после включения, контроллер находится в состоянии ожидания, оно изменяется в случае получения контроллером какой-либо команды от

системы управления. В это же состояние контроллер переходит при получении от системы управления команды сброса.

Первый режим включается при получении контроллером одной из команд протокола, относящейся к взаимодействию с переменными («Leftspeed», «GetDistance» и т. д.). В этом режиме система управления либо запрашивает показания различных датчиков робота, либо устанавливает необходимые значения переменных контроллера. При таком режиме работы робот фактически управляется непосредственно из системы управления на ПК, что позволяет отлаживать исследуемый автомат.

Второй режим – автономный. Он включается, если контроллер получает команду «SetAutomata». В этом режиме контроллер получает от системы управления представление конечного автомата и интерпретирует его, автоматически изменяя значения переменных и опрашивая датчики. Контроллер спроектирован в интегрированной среде разработки «Arduino», которая является приложением на *Java*, включающим в себя редактор кода, компилятор и модуль передачи прошивки в аппаратную вычислительную платформу *Arduino*.

3.4. Протокол связи

Как реальный, так и виртуальный роботы управляются при помощи специального, единого протокола. Это позволяет использовать практически идентичные контроллеры, как для реального робота, так и для виртуального. Благодаря этому сильно упрощается процесс модификации контроллера, так как нет необходимости производить изменения в контроллерах с различными архитектурами. Перечислим команды протокола, передаваемые роботу из системы управления на ПК:

1. «Reset» – осуществить сброс контроллера в исходное состояние;
2. «LeftSpeed **a»** – установить значение скорости левого двигателя, равной **a**, где **a** – целое число;
3. «RightSpeed **a»** – установить значение скорости правого двигателя, равной **a**, где **a** – целое число;
4. «GetDistance» – передать показание дальномера в систему управления на ПК;
5. «GetLine **a»** – передать показание датчика линии под номером **a**, где **a** – целое число от нуля до трех;
6. «SetAutomata» – включить процедуру приема автомата управления из системы управления на ПК;
7. «Delay **a»** – установить продолжительность работы моторов **a**, где **a** – целое число;
8. «END» – закончить прием команд и перейти к их выполнению;

После каждой команды должен идти символ ‘\r\n’. В контроллере робота реализован процесс разбора строк. В результате, полученная команда, разбивается на номер команды и значение, если оно предусмотрено для данной команды.

Перечислим команды протокола, передаваемые системе управления на ПК из контроллера робота:

1. «Distance **a»** – показания дальномера равно **a**, где **a** – целое число.
2. «Line **a b»** – показание датчика линии **a** равно **b**, **a** – целое число от нуля до трех, **b** – ноль или единица.
3. «OK!» – контроллер робота получил команду «END» и выполнил указанные инструкции.

В управляющем приложении на ПК протокол реализован в классе *ParametersControl*. Этот класс имеет методы *synchronizeIn* и *synchronizeOut*, отвечающие за синхронизацию переменных экземпляра класса *ParametersControl* со значениями переменных контроллера робота. Метод *synchronizeIn* отправляет контроллеру робота следующую последовательность команд: «GetDistance», «GetLine 1...4» и команду «END», после получения которой, контроллер начинает выполнение принятых команд. Метод *synchronizeOut* отправляет контроллеру робота последовательность, состоящую из команд: «Leftspeed s», «Rightspeed r», «Delay d» и «END», где *s, r, d* – целые числа. Для отправки используется класс *Sender*. Этот класс может быть инициализирован двумя способами: в первом случае связь с реальным роботом будет осуществляться через com-порт, а во втором – связь с эмулятором будет осуществляться через сокет. Для приема и передачи данных в этом классе реализованы методы *write(string)* и *ReadLine()*. Метод *write* отправляет контроллеру робота строку, а *ReadLine* считывает строку.

В контроллере робота для связи используется класс *Serial*, обладающий методами *read()* и *println(val)*, где *val* – любой тип данных. Метод *read()* возвращает один байт, или возвращает -1, в случае, если не было считано ни одного символа. Метод *println(val)* сначала отправляет значение *val*, а затем символ конца строки. Каждой команде протокола сопоставляется единственный номер. Для удобства в контроллере создана структура *Command*, содержащая два целых числа: номер команды и параметр команды. Также в контроллере реализован метод *Cmp* для сравнения массивов символов, метод *ParseCommand*, возвращающий номер команды в

зависимости от строки, полученной на вход и метод *GetValue*, возвращающий значение параметра команды.

3.5. Система управления

Система управления предназначена для создания, редактирования и отладки конечных автоматов. Она позволяет проектировать конечные автоматы при помощи встроенных инструментов, а также запускать и отлаживать их. Для запуска и отладки автоматов используется «модуль связи». В режиме отладки, в системе управление хранится и обновляется набор переменных, отвечающих за управление роботом. Набор переменных делится на две части: переменные, осуществляющие входные воздействия, и переменные, осуществляющие выходные воздействия. На основе переменных, отвечающих за входные воздействия, выбирается следующее состояние автомата, а результатом выходных воздействий является изменение соответствующих переменных. После этого набор переменных в системе управления синхронизируется при помощи передающего модуля с набором переменных в роботе. При проектировании автомата, используется метамодель, изображенная на рис. 4.

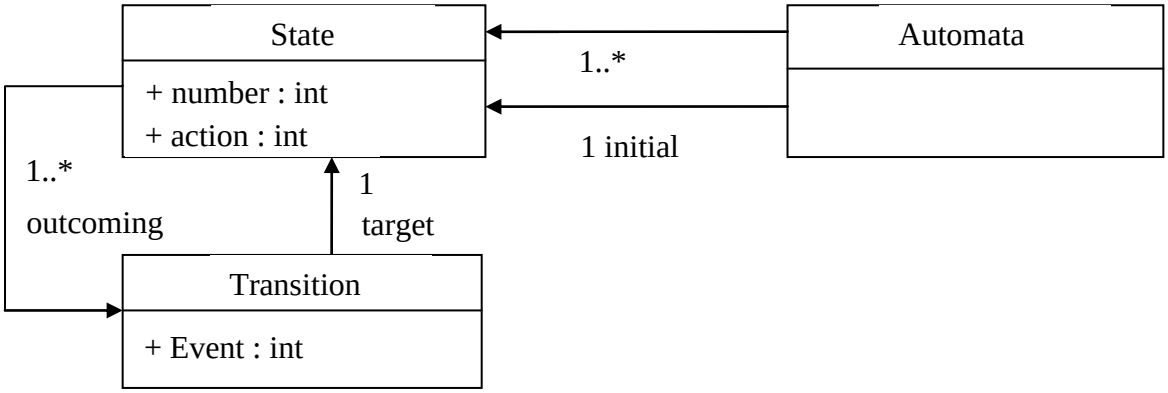


Рис. 4. Метамодель представления автомата, использованного в данной работе

Скриншот системы управления изображен на рис. 5.

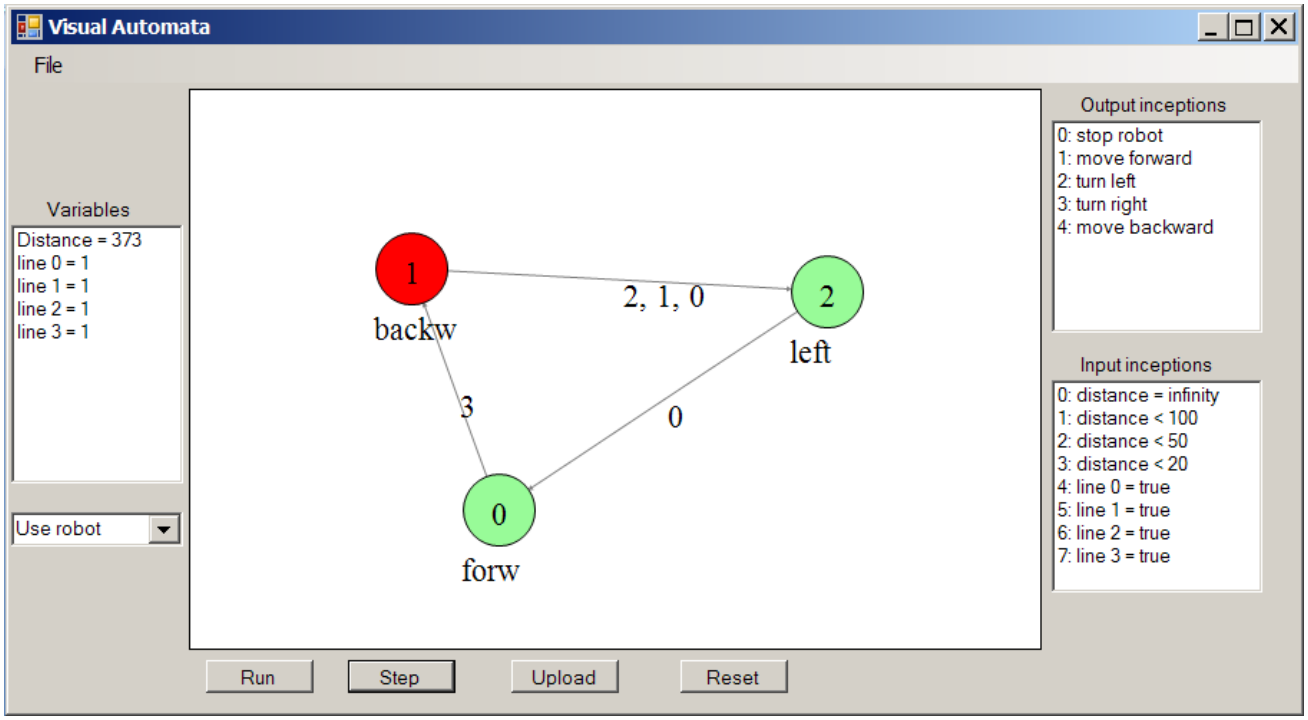


Рис. 5. Скриншот системы управления

Рассмотрим подробно составляющие этой среды. В центре расположена панель редактирования автоматов. Для того чтобы добавить новое состояние к редактируемому автомату, необходимо щелкнуть по панели левой кнопкой мыши при нажатой клавише <Ctrl>. Для удаления состояния, необходимо щелкнуть по нему левой кнопкой мыши с нажатой клавишей <Alt>. Перемещение состояний по панели осуществляется их перетаскиванием с нажатой левой кнопкой мыши. Для добавления перехода из состояния A в B требуется нажать клавишу <Ctrl> и при нажатой левой кнопке мыши переместить курсор мыши из состояния A на состояние B. Для удаления перехода достаточно с нажатой клавишей <Alt> совершить щелчок левой кнопкой мыши на переходе, подлежащем удалению.

Слева окна системы управления расположен список с названием «Variables». В нем отображены значения датчиков робота в данный момент

времени. При этом «Distance» – показания дальномера, «Line **a»** – показание датчика линии **a,** где **a** – целое число.

Под списком «Variables» размещен выпадающий список, который позволяет выбрать контроллер, с которым будет взаимодействовать система управления.

Справа в окне системы управления находятся два списка. Первый – «Output inceptions» позволяет выбрать выходное воздействие, совершаемое при переходе в данное состояние. Для того чтобы назначить состоянию выходное воздействие достаточно щелкнуть левой кнопкой мыши по требуемому состоянию, а затем выбрать одно из выходных воздействий в списке. После этого под состоянием отобразится сокращение, обозначающее выбранное воздействие.

Второй список – «Input inceptions» позволяет выбрать входное воздействие, при условии выполнения которого осуществляется переход.

Помимо перечисленного выше, в окне системы управления присутствуют четыре кнопки: «Run», «Step», «Upload» и «Reset». Кнопка «Step» осуществляет один шаг автомата. После нажатия на нее система управления опрашивает показания датчиков робота, а затем на их основе осуществляет переход в следующее состояние. Выходное воздействие, совершаемое при переходе в следующее состояние, влечет за собой изменение значения переменных робота. Эти переменные система управления отправляет контроллеру робота, который их записывает непосредственно в цифровые выходы. Текущее состояние автомата на каждом шаге выделяется красным цветом.

Кнопка «Run» выполняет такие же действия, как и «Step», за исключением того, что ее действие выполняется непрерывно – от

пользователя не требуется нажатия кнопки для выполнения следующего шага.

Кнопка «Upload» предназначена для выгрузки в контроллер робота интерпретации автомата, созданного при помощи редактора. Это позволяет отключить робота от системы управления. В таком режиме входные воздействия, следующие состояния и выходные воздействия рассчитываются непосредственно вычислительной платформой. Такой режим позволяет роботу осуществлять автономное функционирование без связи с системой управления.

В меню системы управления доступны функции создания нового автомата, сохранения текущего автомата, а также загрузки уже имеющегося автомата в систему управления.

3.6. Модель робота в среде *Solidworks*. Конвертирование модели в объект эмулятора *Webots*

Виртуальная модель робота была спроектирована в системе твердотельного моделирования *Solidworks*. Это позволяет экспортировать ее в среду *Webots*, а также получать чертежи для серийного изготовления реальных роботов (приложение 1). Кроме того, проектирование в среде *Solidworks* обладает рядом других достоинств.

Во-первых, при помощи встроенного в *Webots* 3D-редактора нельзя получить столь точные модели, как в среде *Solidworks*, так как встроенный редактор включает в себя лишь несколько примитивных объектов, таких как сфера, цилиндр, параллелепипед и т. д.

Во-вторых, в среде *Solidworks* есть различные инструменты для оценки спроектированной модели, такие как проверка интерференции, расчет массовых характеристик и т.д.

В *Webots* была загружена модель мобильного робота, спроектированная в *Solidworks* (рис. 6).

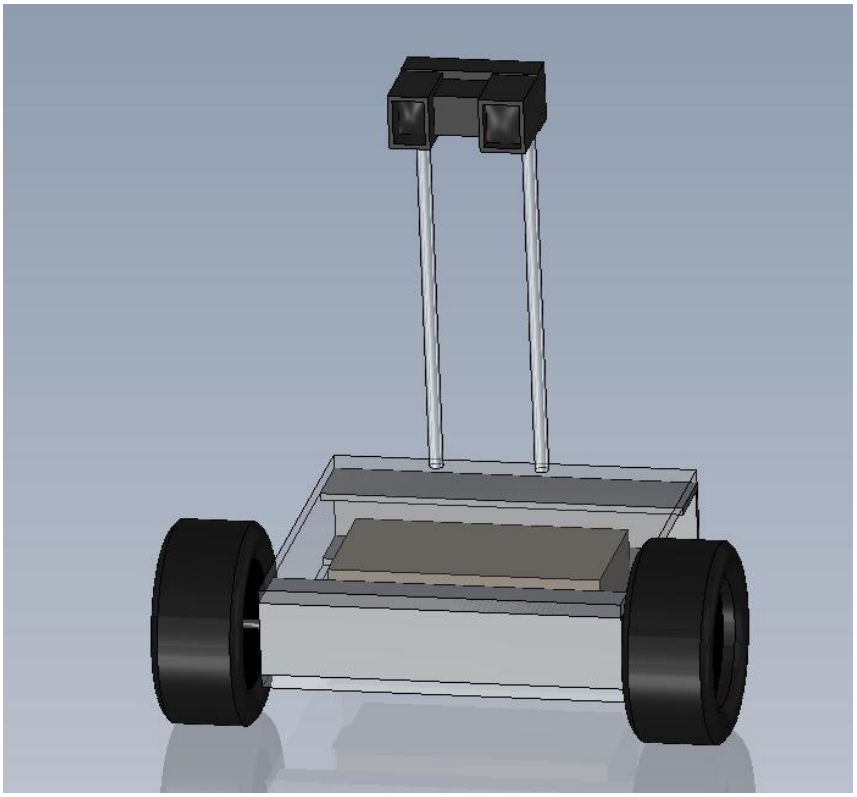


Рис. 6. Модель робота в среде *Solidworks*

Загрузка модели в *Webots* осуществлялась в несколько этапов:

1. Выгрузка модели из *Solidworks*.
2. Загрузка трехмерных объектов в *Webots.*
3. Создание модели робота в *Webots* на основе загруженных объектов.

Из системы *Solidworks* робот был выгружен в файл формата *VRML V2.0 utf8*. Этот формат фактически является массивом индексированных точек. Каждый массив соответствует конкретной детали. Затем этот файл был импортирован в среду *Webots*. В результате импорта получилось 16 объектов. После этого в среде *Webots* был создан объект класса *DifferentialWheels* –

класса двухколесных роботов. Трехмерным объектам колес импортируемого робота были присвоены значения «left wheel» и «right wheel» в *DifferentialWheels*. Дальномер стал объектом класса *DistanceSensor*.

Остальные трехмерные объекты робота сделаны декоративными.

3.7. Эмулятор *Webots*

Webots – среда разработки, предназначенная для проектирования, управления и эмулирования мобильных роботов. При помощи *Webots*, пользователь может проектировать сцены с различным числом объектов и роботов. Настройки объектов (масса, размер, цвет и т. д.) также задаются пользователем. Благодаря большому разнообразию сенсоров и материалов, можно создавать любые модификации робота. Контроллеры робота могут быть запрограммированы с помощью интегрированной среды разработки. Возможности робота могут быть протестированы в физически реалистичном мире. На рис. 7 изображен виртуальный робот в среде эмуляции *Webots*.

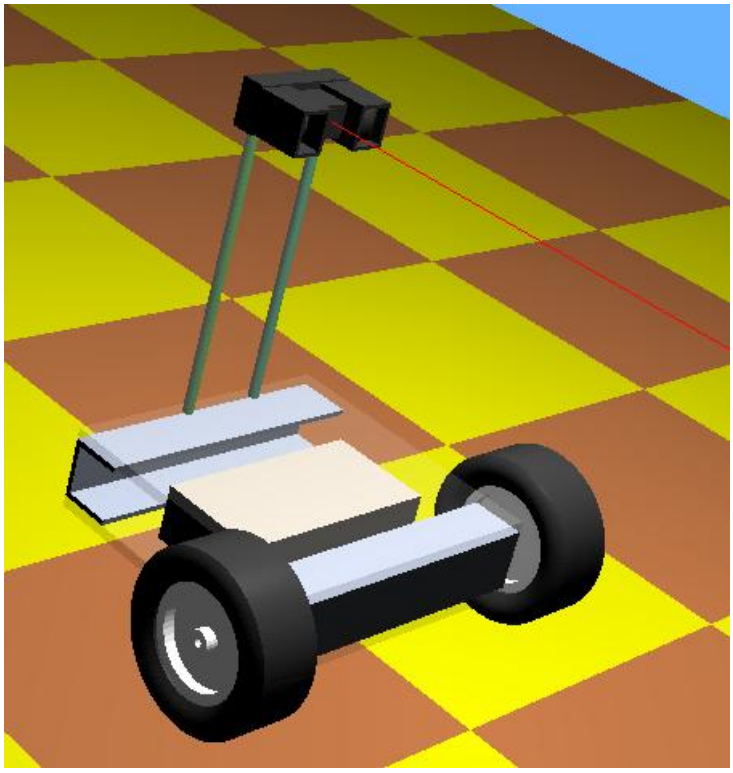


Рис. 7. Модель робота в среде эмуляции *Webots*

В среде *Webots* эмулируется определенная «сцена». Она задается объектом класса *WorldInfo*. В этом классе задается основная информация о поведении объектов на этой «сцене» и отображении на экран. Также необходимо указать источники света. Они задаются объектами класса *PointLight*. Фон задается объектом класса *BackGround*. Параметры прорисовки, такие как дальность и т.д., задаются объектом класса *ViewPoint*.

Виртуальный робот задается классом *DifferentialWheels*. Это стандартный класс для реализации двухколесного робота. Опишем наиболее важные поля этого класса:

- *Translation* – поле, содержащее три координаты, которые отвечают за смещение моделей в этом классе относительно начала координат;
- *Rotation* – поле, содержащее четыре числа, первые три – коэффициенты поворота относительно осей X, Y, Z соответственно. Последнее число – угол поворота. В нашем случае во второе поле (Y) установлена единица, так как робот поворачивает во время движения только вокруг оси Y;
- *Children* – ветка, в которой содержится информация о трехмерных объектах и их назначении. В нашем случае *Transform* содержит декоративные элементы робота, два объекта *Solid* содержат трехмерные объекты колес. Они называются «left wheel» и «right wheel». *DistanceSensor* содержит трехмерный объект дальномера. *BODY Transform* содержит объект цилиндр, который используется, как граница робота при взаимодействии с другими объектами сцены;
- *BoundingObject* – поле, содержащее объект, который будет использован, как граница робота. В нашем случае там содержится определенный ранее объект *BODY Transform*;

- *Controller* – поле, содержащее информацию об используемом контроллере.

4. Генетическое программирование

Эволюционный алгоритм – алгоритм, использующий и моделирующий процесс биологической эволюции, задача которого – максимизировать значение целевой функции. Генетические алгоритмы – один из видов эволюционных алгоритмов.

В данной работе приводится пример автоматического синтеза системы управления роботом для задачи «Кегельринг» и ее проверка на реальном мобильном роботе.

4.1. Постановка задачи

Цель системы управления роботом в задаче «Кегельринг» – вытолкнуть из ринга расположенные в нем кегли (рис.8), не выходя за пределы круга, ограничивающего ринг, и сделать это необходимо за наиболее короткое время.

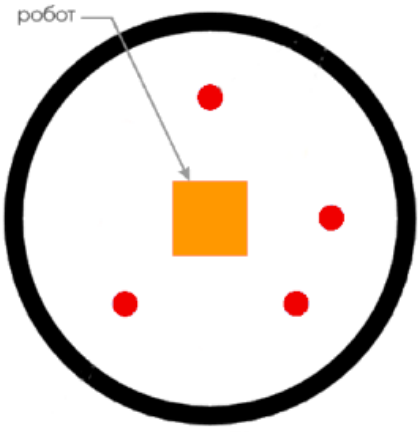


Рис. 8. Ринг с кеглями и робот

Порядок расстановки кеглей в ринге определяется следующим образом:

– перед началом состязания на ринге расставляют восемь кеглей;

– робот устанавливается в центр ринга;

– методом жеребьевки из ринга убирают четыре кегли.

4.2. Схема работы генетического алгоритма

Для решения задачи «Кегельринг» разработан программно-аппаратный комплекс (рис.9), который включает в себя среду эмуляции и приложение, реализующее генетический алгоритм (в дальнейшем просто генетический алгоритм).

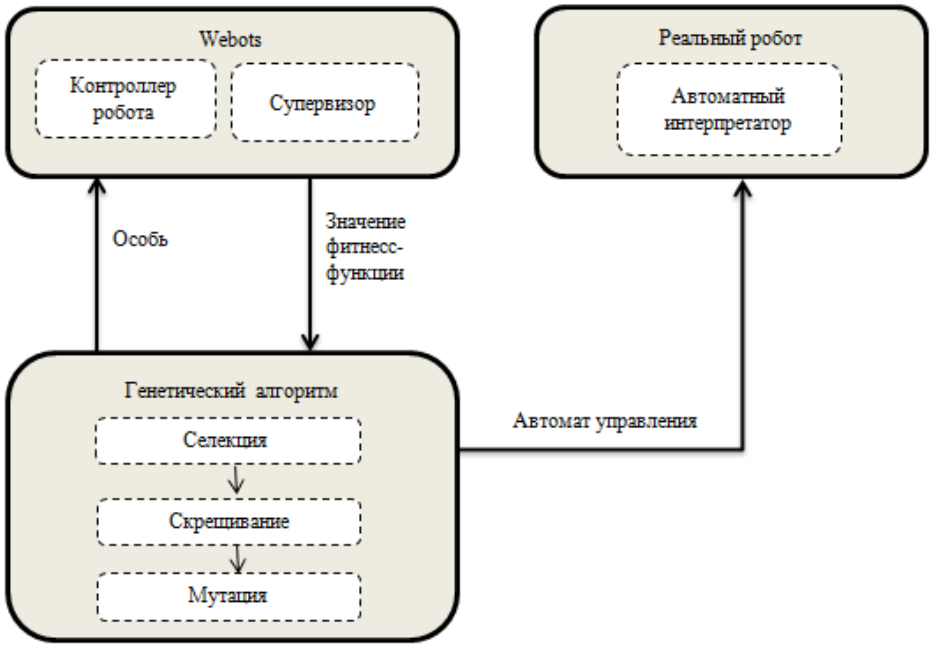


Рис. 9. Структурная схема разработанного комплекса

Среда эмуляции используется для вычисления значения функции приспособленности особи, которая подается ей на вход. При этом желательно обеспечить наиболее точное соответствие среды эмуляции и реальной среды.

В качестве среды эмуляции в данной работе использована программная среда *Webots*. Эта среда позволяет создавать виртуальные модели роботов,

окружение для них, а также программировать физическое поведение и выполнять симуляцию.

Виртуальный робот может быть оснащен моторами, сервоприводами и набором датчиков. Специально для решения поставленной задачи была спроектирована виртуальная модель робота и виртуальный ринг с кеглями.

Виртуальная модель робота была оснащена двумя моторами, дальномером и датчиком линии эквивалентными тем, которые установлены на реальном роботе.

В качестве элемента обеспечения работы виртуальной модели был разработан специальный интерпретатор, представляющий собой программу на языке *Java*. С помощью этого интерпретатора исполняется автомат управления роботом.

Кроме того, был создан контроллер-супервизор, который отслеживает перемещения модели робота и кеглей, и на основе этих данных вычисляет значение функции приспособленности.

Генетический алгоритм осуществляет процесс эволюции особей. Вначале он генерирует случайную стартовую популяцию. После этого особи стартовой популяции подаются на вход среде эмуляции. Среда эмуляции рассчитывает значение функции приспособленности для каждой из особей и предоставляет эти данные генетическому алгоритму. На следующем шаге генетический алгоритм осуществляет селекцию, скрещивание и мутацию в соответствии с полученными значениями функции приспособленности.

Любую из полученных в популяции особей при необходимости можно передать на вход интерпретатору реального робота для проверки функциональности особи в условиях реальной среды.

4.3. Способ кодирования хромосом

Для применения генетического алгоритма к решению задач автоматического синтеза систем управления мобильными роботами, необходима модель хромосомы. В нашем случае хромосома представляет собой закодированный специальным образом конечный автомат. Рассмотрим способ кодирования более подробно.

Результат кодирования – байт-строка, задающая конечный автомат единственным способом (рис. 10).

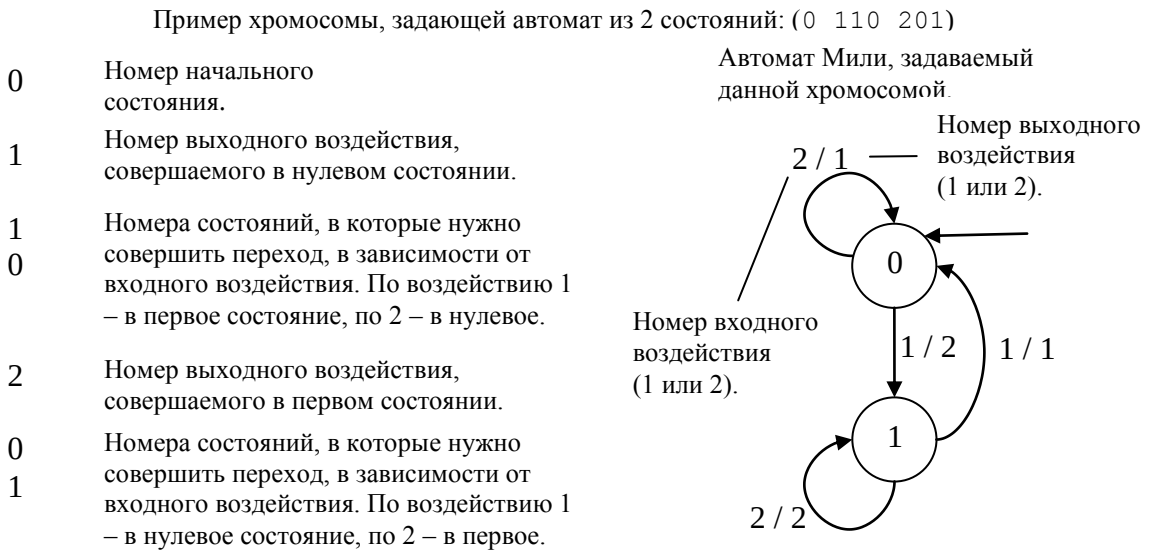


Рис. 10. Пример кодирования хромосомы

Рассмотрим представление байт-строки для конечного автомата из N состояний с L выходными воздействиями. Первый байт – номер начального состояния, его значение меньше N. Каждое из состояний кодируется следующим образом: первый байт – номер выходного воздействия (ехать вперед/назад, поворачивать), затем L байт – номера состояний для перехода по входным воздействиям. Таким образом, в начале байт-строки идет один байт – номер начального состояния, а затем $N*L$ байт – описание состояний.

Полученная модель хромосомы является одной из самых простых. Это позволяет эффективно применять операторы скрещивания и мутации.*Операторы скрещивания и мутации*

Как отмечалось выше, генетический алгоритм осуществляет процесс эволюции особей, который похож на биологическую эволюцию. В ходе этого процесса применяются два основных инструмента – селекция и скрещивание. Для скрещивания особей применяется специальный оператор скрещивания, задача которого – создание особей нового поколения на основе особей предыдущего поколения.

В качестве оператора скрещивания применим операцию одноточечного кроссовера. Он работает следующим образом: генерируется случайное натуральное число *C*, меньшее длины хромосомы. Затем обе родительские хромосомы делятся на две части, первая из которых содержит первые *C* генов, а вторая – оставшийся генотип. Две дочерние особи получаются путем составления нового генотипа на основе частей генотипа обоих родителей.

Рассмотрим пример (рис. 11) применения операции одноточечного кроссовера к хромосомам *A(0110201)* и *B(1201101)*.

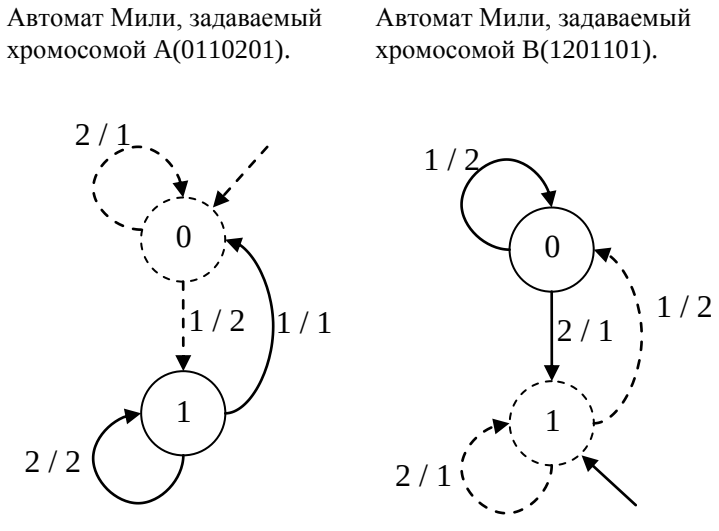


Рис. 11. Применение оператора скрещивания к хромосомам *A*(0 110 201) и *B*(1 201 101)

Длина каждой из хромосом равна семи. Допустим, что число *C* равно четырем, тогда первая дочерняя хромосома будет состоять из четырех первых байт хромосомы *A* и трех последних хромосомы *B*. Вторая дочерняя хромосома будет состоять из четырех первых байта хромосомы *B* и трех последних хромосомы *A*. Части, которыми обмениваются конечные автоматы, выделены пунктирными линиями.

Оператор мутации осуществляет случайные изменения в генотипе случайных особей. В нашем случае, оператор мутации изменяет один (случайно выбранный) байт в хромосомах пяти случайных особей.

4.4. Функция приспособленности

Для эффективной работы генетического алгоритма необходима специально подобранная функция приспособленности. Функция приспособленности или фитнес функция – отображение из множества хромосом в множество численных значений их приспособленности.

В нашем случае функция приспособленности зависит от следующих параметров:

- число кегель, которые в какой-либо момент времени находились за пределами ринга;
- число кегель, которые не покинули ринг;
- расстояние, пройденное роботом;
- время, за которое робот решил задачу;
- условие: покидал ли робот пределы ринга?

Особенностью функции приспособленности, использованной в этой задаче, является то, что способ ее расчета зависит от приспособленности робота. Поясним это. В первом поколении, когда хромосомы строятся

случайным образом, робот редко может вытолкнуть хотя бы одну кеглю. Таким образом, если оставить для расчета функции приспособленности только такие параметры, как время, число кегель, сформулированное выше условие, то большее значение функции приспособленности часто будут иметь роботы, не предпринимающие каких-либо действий и, как следствие, не покидающие ринг.

Получается, что на первом этапе выращивания целесообразно использовать функцию приспособленности, включающую в себя «бонус» за пройденное расстояние. Тогда получат преимущество роботы, предпринимающие активные действия, например, использующие датчик линии, или перемещающиеся внутри круга.

Для вычисления функции приспособленности используется процедура, приведенная в Листинге 1.

Листинг 1

```
fitness = 0;                    //Значение функции
//приспособленности
current_time = 0; //Текущее время модели
max_time   =   const;   //Максимальное   время
//выполнения задания
distance = 0;     //Путь проделанный роботом
while current_time < max_time do
    if кегли_убраны and робот_не_покидал_ринг
    then
        fitness = max_time - current_time;
```

```
        return fitness;
    else
        fitness = fitness + distance;
        if кегля_покинула_ринг then
            fitness = fitness + 1;
        if кегля_вернулась_в_ринг then
            fitness = fitness – 0.5;
return fitness;
```

4.5. Калибровка модели и реального робота

Одной из серьезных проблем при применении генетических алгоритмов является несоответствие между моделью задачи, используемой при синтезе системы управления, и реальным миром.

Автомат, отлично решающий поставленную задачу в эмуляторе, может полностью не справляться с ней в реальной среде. При разработке модели даже для такой простой задачи как «Кегельринг» невозможно учесть все внешние факторы, влияющие на работу алгоритма: погрешности датчиков, особенности работы электродвигателей, материалы предметов, покрытие ринга и многое другое. Робот, выращенный на одной модели, должен демонстрировать работоспособность и на другой, немного отличающейся от исходной.

Рассмотрим возможные методы решения этой проблемы:

- обучение робота на различных моделях. Недостатком данного метода является увеличение пространства поиска, что негативно сказывается на времени работы алгоритма;

– обучение робота с зашумленными входами. Данный метод не увеличивает пространство поиска, но при этом адаптирует робота к выходу в реальный мир;
– изменение функции приспособленности в сторону меньшей зависимости от параметров среды;

В данной работе для решения этой проблемы был предложен способ «обмена» переменными. При использовании этого способа фактически детерминируются результаты выходных воздействий автомата. Это осуществляется за счет установки фиксированного расстояния, пройденного колесами, для каждого воздействия. Данный способ реализуется за счет установки скорости двигателей и времени их работы. В результате имеет место практически точное соответствие между реальной и виртуальной моделями.

Результаты

- Разработан графический язык автоматного программирования с возможностью отладки, ориентированный на мобильных роботов.
- Разработана методика для конвертирования модели из среды *SolidWorks* в среду эмулятора *Webots*.
- На основе модели робота в среде *SolidWorks* изготовлен и протестирован мобильный робот.
- Разработана и опробована система для генетического программирования на основе эмулятора *Webots*.

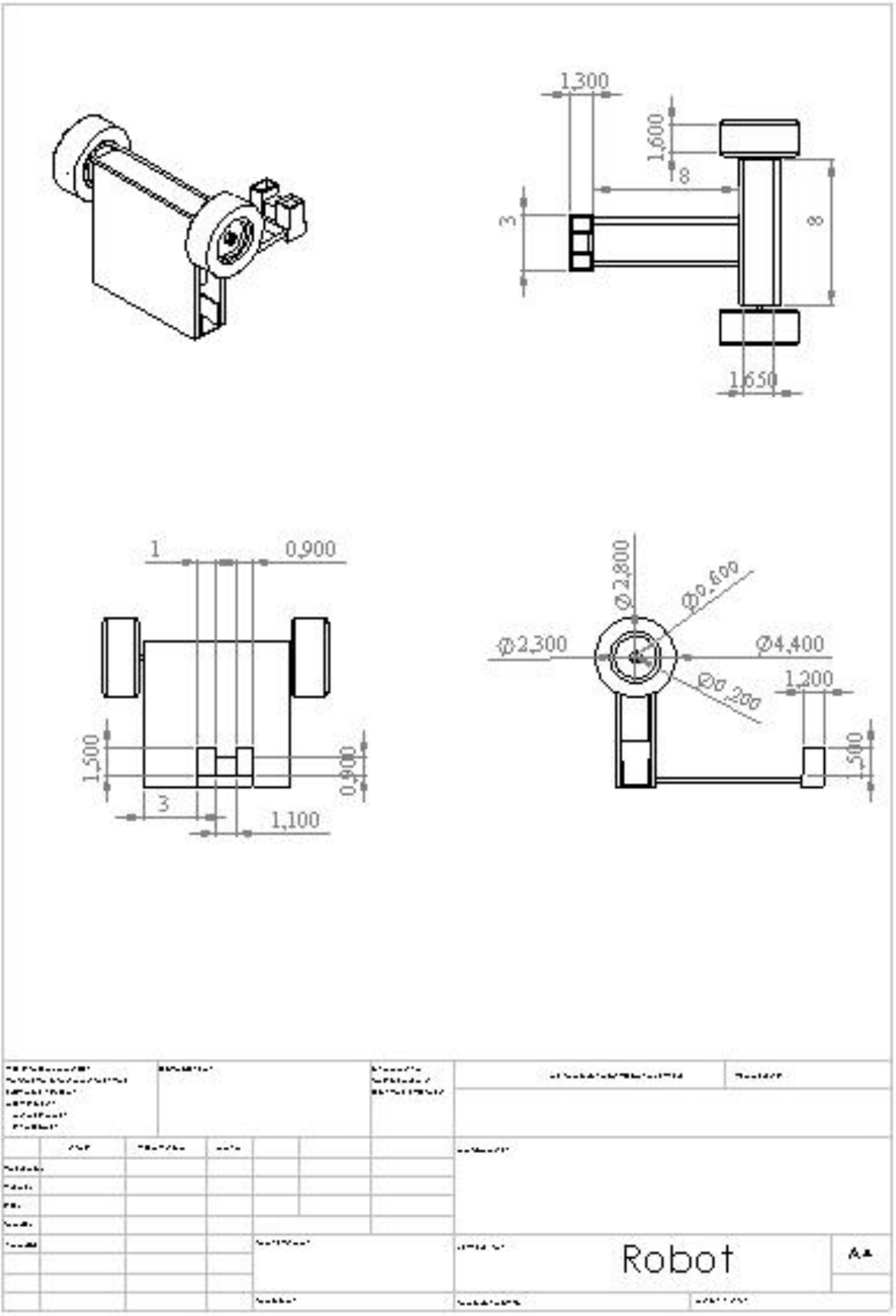
Литература

1. *Шалыто А. А.* Технология автоматного программирования // Труды Всероссийской научной конференции «Методы и средства обработки информации». М.: МГУ. 2003. http://is.ifmo.ru/works/tech_aut_prog/
2. *Клебан В. О., Шалыто А. А., Парфенов В. Г.* Построение системы автоматического управления мобильным роботом на основе автоматного подхода //Научно-технический вестник СПбГУ ИТМО. Выпуск 53. Автоматное программирование. 2008, с. 281 – 285.
3. *Клебан В. О., Шалыто А. А.* Использование автоматного программирования для построения многоуровневых систем управления мобильными роботами / Экстремальная робототехника. СПб: ЦНИИ РТК, 2008.
4. *Гуров В. С., Мазин М. А, Нарвский А. С, Шалыто А. А.* UML. SWITCH-технология. Eclipse // Информационно-управляющие системы. 2004. № 6, с.12–17. http://is.ifmo.ru/works/uml-switch-eclipse
5. *Царев Ф. Н., Шалыто А. А.* О построении автоматов с минимальным числом состояний для задачи об «умном муравье» / Сборник докладов X международной конференции по мягким вычислениям и измерениям. СПбГЭТУ «ЛЭТИ». Т.2. 2007, с. 88 – 91. http://is.ifmo.ru/download/ant_ga_min_number_of_state.pdf
6. *Шалыто А. А., Туккель Н. И.* SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем //Программирование. 2001. № 5, с. 22-28.

7. Cyberbotics Ltd. Webots reference manual. 2009. http://www.cyberbotics.com/cdrom/common/doc/webots/reference/reference.html

8. *Тику Ш.* Эффективная работа: SolidWorks 2004. СПб.: Питер, 2005.

Приложение 1. Чертеж робота



Приложение 2. Исходный код комплекса

MainForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using automatPanel;
using System.Collections;
using System.IO.Ports;
using System.Threading;
using System.Net;
using System.Net.Sockets;

namespace VisualAutomata
{
    public partial class Form1 : Form
    {
        ArrayList states;
        ArrayList arrows;
        bool reset;
        public Form1()
        {
            InitializeComponent();
            com = new SerialPort("COM7");
            com.Open();
            com.DiscardInBuffer();
            com.DiscardOutBuffer();
            com.Write("END\r\n");
            com.ReadLine();
            //sock = ConnectSocket("127.0.0.1", 6666);
            reset = true;
            listBox1.Items.Add(new inception(0, "stop robot"));
            listBox1.Items.Add(new inception(1, "move
forward"));
            listBox1.Items.Add(new inception(2, "turn left"));
            listBox1.Items.Add(new inception(3, "turn right"));
            listBox1.Items.Add(new inception(4, "move
backward"));

            listBox2.Items.Add(new inception(0, "distance =
infinity"));
            listBox2.Items.Add(new inception(1, "distance",
100));
            listBox2.Items.Add(new inception(2, "distance",
50));
            listBox2.Items.Add(new inception(3, "distance",
20));
```

```
            listBox2.Items.Add(new inception(4, "line 0",
"true"));
            listBox2.Items.Add(new inception(5, "line 1",
"true"));
            listBox2.Items.Add(new inception(6, "line 2",
"true"));
            listBox2.Items.Add(new inception(7, "line 3",
"true"));


        }

        private static Socket ConnectSocket(string address, int
port)
        {
            Socket s = null;
            IPAddress add = IPAddress.Parse(address);
            IPEndPoint ipe = new IPEndPoint(add, port);
            Socket tempSocket =
                new Socket(ipe.AddressFamily, SocketType.Stream,
ProtocolType.Tcp);
            tempSocket.Connect(ipe);
            if (tempSocket.Connected)
            {
                s = tempSocket;
            }
            return s;
        }



        class inception
        {
            int number;
            string description;
            string addinf;
            int tostring;
            int distance;


            public inception(int number, string description)
            {
                this.number = number;
                this.description = description;
                tostring = 0;
            }

            public inception(int number, string description,
string additional_information)
            {
                this.number = number;
                this.description = description;
```

```
            this.addinf = additional_information;
            tostring = 1;
        }

        public inception(int number, string description, int
distance)
        {
            this.number = number;
            this.description = description;
            this.distance = distance;
            tostring = 2;
        }

        public override string ToString()
        {
            switch (tostring)
            {
                case 0: return number.ToString() + ": " +
description;
                case 1: return number.ToString() + ": " +
description + " = " + addinf;
                case 2: return number.ToString() + ": " +
description + " < " + distance.ToString();
            }
            return "error";
        }
    }

    private void listBox1_SelectedIndexChanged(object
sender, EventArgs e)
    {
        if (userControl11.s_selected != null)
        {

userControl11.s_selected.setoutput(listBox1.SelectedIndex);
            switch (listBox1.SelectedIndex)
            {
                case 0:
userControl11.s_selected.setName("stop");
                    break;
                case 1:
userControl11.s_selected.setName("forw");
                    break;
                case 2:
userControl11.s_selected.setName("left");
                    break;
                case 3:
userControl11.s_selected.setName("right");
                    break;
                case 4:
userControl11.s_selected.setName("backw");
                    break;
            }
```

```
            }
        }

        private void listBox2_SelectedIndexChanged(object
sender, EventArgs e)
        {
            if (userControl11.a_selected != null)
            {
                if
(!userControl11.a_selected.inceptions.Contains(listBox2.Selected
Index))

userControl11.a_selected.inceptions.Add(listBox2.SelectedIndex);
            }
        }

        private TableAutomata TA;
        private SerialPort com;
        private ParametersControl pctrl;
        private Socket sock;

        private void button1_Click(object sender, EventArgs e)
        {
            userControl11.setDesignMode(false);
            reset = false;
            if (TA == null)
            {
                pctrl = new ParametersControl();
                generateTA();
            }
            do_next(400, 200);
        }

        private void generateTA()
        {
            states = new ArrayList(userControl11.getStates());
            arrows = new ArrayList(userControl11.getArrows());

            TA = new TableAutomata(states.Count);
            for (int i = 0; i < states.Count; i++)
            {
                State st = (State)states[i];
                TableState ts = new TableState(st.getnumber(),
st.outArrows.Count, st.getoutput());
                for (int j = 0; j < st.outArrows.Count; j++)
                {
                    Arrow ar = (Arrow)st.outArrows[j];
                    for (int k = 0; k < ar.inceptions.Count;
k++)
                        ts.addnewtransition(ar.inceptions[k],
ar.e_state.getnumber());
                }
```

```
                TA.addstate(ts);
            }


        }

        class variable
        {
            string str;
            int val;

            public variable(string line, int value)
            {
                this.str = line;
                this.val = value;
            }

            public override string ToString()
            {
                return (str + val.ToString());
            }
        }

        private void do_next(int turn_delay, int forw_delay)
        {
            int distance = 0;
            int input = 0;
            int output = 0;

            pctrl.synchronizeIn(com);
            distance = pctrl.getdistance();
            if (turn_delay + forw_delay != 0)
            {
                listBox3.Items.Clear();
                listBox3.Items.Add(new variable("Distance = ",
distance));
                listBox3.Items.Add(new variable("line 0 = ",
pctrl.getline(0)));
                listBox3.Items.Add(new variable("line 1 = ",
pctrl.getline(1)));
                listBox3.Items.Add(new variable("line 2 = ",
pctrl.getline(2)));
                listBox3.Items.Add(new variable("line 3 = ",
pctrl.getline(3)));
            }

            if (distance < 130) input = 0;
            else
                if (distance < 165) input = 1;
                else
                    if (distance < 260) input = 2;
                    else
                        if (distance < 550) input = 3;
```

```
            int cs = TA.get_current_state();
            for (int i = 0; i < states.Count; i++)
            {
                if (((State)states[i]).getnumber() == cs)
                    ((State)states[i]).bcg_color.Color =
Color.PaleGreen;
            }

            output = TA.donext(input);

            cs = TA.get_current_state();
            for (int i = 0; i < states.Count; i++)
            {
                if (((State)states[i]).getnumber() == cs)
                    ((State)states[i]).bcg_color.Color =
Color.Red;
            }

            if (turn_delay + forw_delay != 0)
                userControl11.Refresh();

            if (output == 0) pctrl.setspeed(0, 0);
            else
                if (output == 1)
                {
                    pctrl.setdelay(forw_delay);
                    pctrl.setspeed(50, 50);
                }
                else
                    if (output == 2)
                    {
                        pctrl.setdelay(turn_delay);
                        pctrl.setspeed(0, 50);
                    }
                    else
                        if (output == 3)
                        {
                            pctrl.setdelay(turn_delay);
                            pctrl.setspeed(50, 0);
                        }
                        else
                            if (output == 4)
                            {
                                pctrl.setdelay(forw_delay);
                                pctrl.setspeed(-50, -50);
                            }
            pctrl.synchronizeOut(com);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            if (reset != true)
```

```
    {
        userControl11.setDesignMode(true);
        reset = true;
        userControl11.resetcolor();
        userControl11.Refresh();
        Thread.Sleep(500);
        com.Write("Reset\r\n");
        com.DiscardInBuffer();
        com.DiscardOutBuffer();
        TA = null; ;
    }
}

private void button3_Click(object sender, EventArgs e)
{

    if (reset == true)
    {
        userControl11.setDesignMode(false);
        reset = false;
        if (TA == null)
        {
            pctrl = new ParametersControl();
            generateTA();
        }
        ThreadStart thr = new ThreadStart(run_do_next);
        thread = new Thread(thr);
        thread.Start();
    }
}

void run_do_next()
{
    while (!reset)
    {
        do_next(0, 0);
    }
}

private Thread thread;

private void button4_Click(object sender, EventArgs e)
{
    if (reset != false)
    {
        userControl11.setDesignMode(false);
        reset = false;
        if (TA == null)
        {
            generateTA();
            com.Write("SetAutomata\r\n");
            Console.WriteLine(com.ReadLine());
        }
```

```
                TA.UploadAutomata(com);
            }
        }

        private void toolStripMenuItem1_Click(object sender,
EventArgs e)
        {
            saveFileDialog1.ShowDialog();
            userControl11.save_auto(saveFileDialog1.FileName);
        }

        private void toolStripMenuItem2_Click(object sender,
EventArgs e)
        {
            TA = null;
            openFileDialog1.ShowDialog();
            userControl11.load_auto(openFileDialog1.FileName);
        }

        private void toolStripMenuItem3_Click(object sender,
EventArgs e)
        {
            TA = null;
            userControl11.new_auto();
        }
    }
}
```

ParametersControl.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;

namespace VisualAutomata
{
    class ParametersControl
    {
        int leftspeed;
        int rightspeed;
        int delay;

        int distance;
        int line0, line1, line2, line3;

        public void synchronizeIn(SerialPort port)
        {
            port.Write("GetDistance" + "\r\n");
            port.Write("GetLine " + 0 + "\r\n");
            port.Write("GetLine " + 1 + "\r\n");
            port.Write("GetLine " + 2 + "\r\n");
```

```
            port.Write("GetLine " + 3 + "\r\n");
            port.Write("END\r\n");
            ParseInput(port);
            port.ReadLine();
        }

        public void synchronizeOut(SerialPort port)
        {
            port.Write("LeftSpeed " + leftspeed + "\r\n");
            port.Write("RightSpeed " + rightspeed + "\r\n");
            port.Write("Delay " + delay + "\r\n");
            port.Write("END\r\n");
            port.ReadLine();
        }

        void ParseInput(SerialPort port)
        {
            for (int i = 0; i < 5; i++)
            {
                string str = port.ReadLine();
                if (str.Substring(0, 4) == "Line")
                {
                    int num = Int16.Parse(str.Substring(5, 1));
                    int val = Int16.Parse(str.Substring(7, 1));
                    switch (num)
                    {
                        case 0:
                            line0 = val;
                            break;
                        case 1:
                            line1 = val;
                            break;
                        case 2:
                            line2 = val;
                            break;
                        case 3:
                            line3 = val;
                            break;
                    }
                }
                else
                {
                    distance = Int16.Parse(str.Substring(9,
str.Length - 9));
                }
            }
        }

        public void reset(SerialPort port)
        {
            port.Write("Reset");
            port.Write("\r\n");
        }
```

```
        public void setspeed(int left, int right)
        {
            if(left < 0)
                this.leftspeed = Math.Abs(left) + 255;
            else
                this.leftspeed = left;
            if (right < 0)
                this.rightspeed = Math.Abs(right) + 255;
            else
                this.rightspeed = right;
        }

        public int getdistance()
        {
            return distance;
        }

        public int getline(int number)
        {
            switch (number)
            {
                case 0: return line0;
                case 1: return line1;
                case 2: return line2;
                case 3: return line3;
            }
            return -1;
        }

        public void setdelay(int delay)
        {
            this.delay = delay;
        }

        public ParametersControl()
        {
            leftspeed = 0;
            rightspeed = 0;
            delay = 0;

            distance = -1;
            line0 = -1;
            line1 = -1;
            line2 = -1;
            line3 = -1;
        }
    }
}
```

TableAutomata.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;

namespace VisualAutomata
{
    class TableAutomata
    {
        int currentstate;
        List<TableState> Automata;


        public void addstate(TableState state)
        {
            this.Automata.Add(state);
            this.Automata.Sort(new StComp());
        }

        public int get_current_state()
        {
            return this.currentstate;
        }

        public TableAutomata(int states)
        {
            Automata = new List<TableState>(states);
        }

        public int donext(int input)
        {
            Transition trans =
Automata.ElementAt(currentstate).gettransition(input);
            if (trans.newstate != -1)
                currentstate = trans.newstate;
            return
(Automata.ElementAt(currentstate).getoutput());
        }

        public void setcurrstate(int num)
        {
            this.currentstate = num;
        }

        class StComp : IComparer<TableState>
        {
            public int Compare(TableState x, TableState y)
            {
                return (x.number - y.number);
            }
        }
```

```
        public void UploadAutomata(SerialPort com)
        {
            byte[] buff = new byte[1];

            //Console.WriteLine("Writing current state");
            buff[0] = (byte)currentstate;
            com.Write(buff, 0, 1);
            //Console.WriteLine(currentstate + " " +
com.ReadLine());

            //Console.WriteLine("Writing states count");
            buff[0] = (byte)Automata.Count;
            com.Write(buff, 0, 1);
            //Console.WriteLine(Automata.Count + " " +
com.ReadLine());

            for (int i = 0; i < Automata.Count; i++)
            {
                int[] trans = new int[2];

                //Console.WriteLine("Output");
                buff[0] =
(byte)Automata.ElementAt(i).getoutput();
                com.Write(buff, 0, 1);

//Console.WriteLine(Automata.ElementAt(i).getoutput() + " " +
com.ReadLine());

                int count =
Automata.ElementAt(i).gettransitioncount();

                //Console.WriteLine("Writing transitions
count");
                buff[0] = (byte)count;
                com.Write(buff, 0, 1);
                //Console.WriteLine(count + " " +
com.ReadLine());

                for (int j = 0; j < count; j++)
                {
                    //Console.WriteLine("Transition");

                    trans =
Automata.ElementAt(i).getinttrans(j);
                    buff[0] = (byte)trans[0];
                    com.Write(buff, 0, 1);
                    //Console.WriteLine(trans[0] + " " +
com.ReadLine());

                    buff[0] = (byte)trans[1];
                    com.Write(buff, 0, 1);
                    //Console.WriteLine(trans[1] + " " +
com.ReadLine());
```

```
                }
            }

        }
    }
}
```

State.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;

namespace VisualAutomata
{
    class TableState
    {

        class TransComp : IComparer<Transition>
        {

            public int Compare(VisualAutomata.Transition x,
VisualAutomata.Transition y)
            {
                return (y.input - x.input);
            }

        }

        public int number;
        List<Transition> transitions;
        int output;

        public TableState(int number, int inputcount, int
output)
        {
            this.number = number;
            this.transitions = new List<Transition>(inputcount);
            this.output = output;
        }

        public TableState(int number, int output)
        {
            this.number = number;
            this.output = output;
        }

        public void addnewtransition(int input, int newstate)
        {
            Transition act = new Transition(input, newstate);
            this.transitions.Add(act);
```

```
        }

        public Transition gettransition(int input)
        {
            Transition act;
            act = transitions.Find(
                delegate(Transition trans)
                {
                    return trans.input == input;
                }
            );
            if (act != null)
                return act;
            else return (new Transition());
        }

        public int gettransitioncount()
        {
            return transitions.Count;
        }

        public int[] getinttrans(int number)
        {
            int[] trans = new int[2];
            trans[0] = this.transitions.ElementAt(number).input;
            trans[1] =
this.transitions.ElementAt(number).newstate;
            return trans;
        }

        public int getoutput()
        {
            return (this.output);
        }
    }
}
```

Action.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace VisualAutomata
{
    class Transition
    {
        public int input;
        public int newstate;

        public Transition(int input, int newstate)
        {
```

```
            this.input = input;
            this.newstate = newstate;
        }

        public Transition()
        {
            this.input = -1;
            this.newstate = -1;
        }
    }
}
```

RoboController.pde

```
typedef unsigned int uint;

//PIN DESCRIPTION
const int LEFT_PWM = 5;
const int LEFT_A = 6;
const int LEFT_B = 7;
const int RIGHT_PWM = 10;
const int RIGHT_A = 9;
const int RIGHT_B = 8;
const int SHARP_PIN = 5;

//--------------------------------------------------------------------
--------
//Hardware utilites
//--------------------------------------------------------------------
--------
void move(int vl, int vr)
{
  vr = -vr;
  if (vl > 0){
    digitalWrite(LEFT_A, HIGH);
    digitalWrite(LEFT_B, LOW);
  }
  else {
    digitalWrite(LEFT_A, LOW);
    digitalWrite(LEFT_B, HIGH);
    vl = -vl;
  }
  analogWrite(LEFT_PWM, vl);

  if (vr > 0){
    digitalWrite(RIGHT_A, HIGH);
    digitalWrite(RIGHT_B, LOW);
  }
  else {
    digitalWrite(RIGHT_A, LOW);
    digitalWrite(RIGHT_B, HIGH);
    vr = -vr;
  }
```

```
  analogWrite(RIGHT_PWM, vr);
}

int GetDistance()
{
  return analogRead(SHARP_PIN);
}

boolean GetLine(char n)
{
  if (n==0 | n==1)
    return analogRead(n) > 300;
  if (n==2 | n==3)
    return analogRead(n) > 950;
}

void setup()
{
  pinMode(LEFT_A, OUTPUT);
  pinMode(LEFT_B, OUTPUT);
  pinMode(RIGHT_A, OUTPUT);
  pinMode(RIGHT_B, OUTPUT);
  Serial.begin(9600);
  //initialize();
}

int ReadLine(char (*chars))
{
  while(Serial.available() == 0)
    delay(5);

  char ch1;
  char ch2;
  int size = 0;
  while(true)
  {
    while(Serial.available() == 0)
      delay(5);
    ch1 = Serial.read();
    if(ch1 == '\r')
    {
      while(Serial.available() == 0)
        delay(5);
      ch2 = Serial.read();

      if(ch2 == '\n') break;
      else
      {
        chars[size++] = ch1;
        chars[size++] = ch2;
      }
    }
    else
```

```
    chars[size++] = ch1;
  }
  return (size);
}

void WriteLine(char chars[], int beginindex, int endinindex)
{
  for(int i = beginindex; i < endinindex; i++)
    Serial.print(chars[i]);
  Serial.println(chars[endinindex]);
}

void CpyLine(char from[], int beginindex, int endindex, char *
dest)
{
  for(int i = 0; i < endindex - beginindex; i++)
    dest[i] = from[i+beginindex];
}

boolean Cmp(char src[], char cmp[], int length)
{
  for(int i = 0; i < length; i++)
  {
    if(src[i] != cmp[i]) return false;
  }
  return true;
}

int ParseCommand(char src[], int length)
{
    if(length >= 3){
      if(Cmp(src, "END", 3)) return 7;

      if(length >= 5)
        if(Cmp(src, "Reset", 5)) return 0;
      if(length >= 7)
        if(Cmp(src, "Delay", 5)) return 6;
      if(length >= 9)
        if(Cmp(src, "GetLine", 7)) return 4;

      if(length >= 11){
        if(Cmp(src, "LeftSpeed", 9)) return 1;
        if(Cmp(src, "GetDistance", 11)) return 3;
        if(Cmp(src, "SetAutomata", 11)) return 5;
      }
      if(length >= 12)
        if(Cmp(src, "RightSpeed", 10)) return 2;
    }

  return (-1);
}

int Degree(int base, int degree)
```

```
{
  int val = 1;
  if(degree == 0) return 1;
  else
  {
    for(int i = 0; i < degree; i++)
      val *= 10;
    return val;
  }
}

int ReadByte()
{
  int tmp = -1;
  while(Serial.available() == 0)
    delay(50);
  tmp = Serial.read() - 0;
  return(tmp);
}

int GetValue(char src[], int command, int length)
{
  int n = 0;
  switch(command)
  {
    case 1:
      n = 10;
      break;
    case 2:
      n = 11;
      break;
    case 4:
      n = 8;
      break;
    case 6:
      n = 6;
      break;
  }

  if(n == 0) return (-1);

  int value = 0;
  for(int i = n; i < length; i++)
  {
    value += (src[i] - '0') * (Degree(10, length - i - 1));
  }

  return value;
}

const unsigned short MAX_STATES = 7;
const unsigned short MAX_INPUTS = 10;
```

```
struct State{
  short newstate[MAX_INPUTS];
  short output;
};

void GetAuto(int states, struct State *Automata[])
{
    int output = 0;
    int transition = 0;
    for(int i = 0; i < states; i++)
    {
      Automata[i]->output = -1;
      for(int j = 0; j < MAX_INPUTS; j++)
        Automata[i]->newstate[j] = -1;
    }
    for(int i = 0; i < states; i++)
    {
      output = ReadByte();
      Automata[i]->output = output;
      transition = ReadByte();
      short newstate = 0;
      short input = 0;
      for(int j = 0; j < transition; j++)
      {
        input = ReadByte();
        newstate = ReadByte();
        Automata[i]->newstate[input] = newstate;
      }
    }
}




void loop()
{
  char line[256];
  int command = -1;
  int accsize = 0;
  int value = -1;
  boolean reset = false;
  while(true)
  {
    reset = false;
    while((accsize = ReadLine(line)) == 0)
      delay(50);
    command = ParseCommand(line, accsize);
    value = GetValue(line, command, accsize);

    if(command == 5)
    {
      int defstate = -1;
      int states = -1;
```

```
  Serial.println("Getting auto");
  defstate = ReadByte();
  states = ReadByte();

  State*Automata[states];
  GetAuto(states, Automata);

  while(true)
  {
    short input = -1;
    int d = GetDistance();

    if(d < 100) input = 0;
      else
      if(d < 135) input = 1;
        else
        if(d < 230) input = 2;
          else
          if(d < 520) input = 3;

    //Serial.println(input);
    short newst = Automata[defstate]->newstate[input];
    //Serial.println(newst);
    if(newst != -1) defstate = newst;
    short output = Automata[defstate]->output;

    if(output == 0) move(0, 0);
      else
      if(output == 1) move(128, 128);
        else
        if(output == 2) move(-128, 128);
          else
          if(output == 3) move(128, -128);
            else
            if(output == 4) move(-128, -128);

     if(Serial.available() != 0)
     {
       accsize = ReadLine(line);
       command = ParseCommand(line, accsize);
       if(command == 0)
       {
         move(0, 0);
         break;
       }
     }
  }
}
else
{
  int leftspeed = 0;
  int rightspeed = 0;
  int dist = -1;
```

```
int del = 0;

boolean lin = false;
while(true){
  while(command != 7)
  {
    switch(command)
    {
      case 0:
        reset = true;
        move(0, 0);
        break;
      case 1:
        leftspeed = value;
        break;
      case 2:
        rightspeed = value;
        break;
      case 3:
        dist = GetDistance();
        Serial.print("Distance ");
        Serial.println(dist);
        break;
      case 4:
        lin = GetLine(value);
        Serial.print("Line ");
        Serial.print(value);
        Serial.print(" ");
        if(lin) Serial.println("1");
        else Serial.println("0");
        break;
      case 6:
        del = value;
        break;
    }
    if(reset) break;
    if(leftspeed > 255) leftspeed = 255 - leftspeed;
    if(rightspeed > 255) rightspeed = 255 - rightspeed;
    accsize = 0;

    while((accsize = ReadLine(line)) == 0)
      delay(50);
    command = ParseCommand(line, accsize);
    value = GetValue(line, command, accsize);
  }

  Serial.flush();
  if(reset) break;
  move(leftspeed, rightspeed);
  if(del != 0)
  {
    delay(del);
    leftspeed = 0;
```

```
          rightspeed = 0;
          move(0,0);
        }
        Serial.println("OK!");
        while((accsize = ReadLine(line)) == 0)
          delay(50);
        command = ParseCommand(line, accsize);
        value = GetValue(line, command, accsize);
      }
    }
  }
}
```

UserControl.cs

```
using System;
using System.IO;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Data;
using System.Text;
using System.Windows.Forms;

namespace automatPanel
{

    public partial class UserControl1 : UserControl
    {
        private Graphics            cur_g;
        private int                 height;
        private int                 width;
        private SolidBrush          cur_brush;
        private Pen                 cur_pen;
        private Rectangle           work_place;
        private Rectangle           cur_rect;
        private ArrayList           drawing_states;
        private ArrayList           drawing_arrows;
        public State               s_selected;// one of state is
selected
        private int                 mouse_x;
        private int                 mouse_y;
        private bool                ctrl_pressed;
        private bool                alt_pressed;
        public Arrow               a_selected;//выбранное ребро
        private bool                design_mode;
        private int                 StateNumber;

        public UserControl1()
        {
            InitializeComponent();
```

```
            height              = this.Height;
            width               = this.Width;
            drawing_states      = new ArrayList();
            drawing_arrows      = new ArrayList();
            cur_brush           = new SolidBrush(Color.White);
            cur_pen             = new Pen(Color.Black);
            work_place          = new Rectangle(0, 0, width,
height);
            cur_rect            = new Rectangle(0, 0, 0, 0);
            s_selected          = null;
            ctrl_pressed        = false;
            alt_pressed         = false;
            design_mode         = true;
            StateNumber         = 0;
        }

        public void resetcolor()
        {
            for (int i = 0; i < drawing_states.Count; i++)
                ((State)drawing_states[i]).bcg_color.Color =
Color.PaleGreen;
        }

        public bool save_auto(string fname)
        {
            TextWriter tw = new StreamWriter(fname);
            tw.WriteLine(drawing_states.Count);
            foreach (State s in drawing_states)
            {
                Point p = s.getCenter();
                int number = s.getnumber();
                int output = s.getoutput();
                tw.WriteLine(p.X + "," + p.Y + "," + number +
"," + output + "," + s.getname());
            }

            tw.WriteLine(drawing_arrows.Count);
            foreach (Arrow a in drawing_arrows)
            {
                State beg = a.b_state;
                State end = a.e_state;


                tw.WriteLine(beg.getnumber() + "," +
end.getnumber() + "," + a.inceptions.Count);
                for (int i = 0; i < a.inceptions.Count - 1; i++)
                {
                    tw.Write(a.inceptions[i] + ",");
                }
                tw.WriteLine(a.inceptions[a.inceptions.Count -
1]);
            }
```

```
            tw.Close();

            return true;
        }

        public bool load_auto(string fname)
        {
            StateNumber = 0;
            TextReader tr = new StreamReader(fname);
            int states = 0;
            int arrows = 0;
            string tmp = "";
            drawing_states.Clear();
            drawing_arrows.Clear();

            states = int.Parse(tr.ReadLine());

            for (int i = 0; i < states; i++)
            {
                tmp = tr.ReadLine();
                string[] sites = tmp.Split(',');
                int px = int.Parse(sites[0]);
                int py = int.Parse(sites[1]);
                int number = int.Parse(sites[2]);
                int output = int.Parse(sites[3]);
                State st = new State(px, py);
                st.setnumber(number);
                st.setoutput(output);
                st.setName(sites[4]);
                drawing_states.Add(st);
            }

            arrows = int.Parse(tr.ReadLine());
            for (int i = 0; i < arrows; i++)
            {
                tmp = tr.ReadLine();
                string[] sites = tmp.Split(',');
                int beg = int.Parse(sites[0]);
                int end = int.Parse(sites[1]);
                int incount = int.Parse(sites[2]);
                Arrow ar = new Arrow((State)drawing_states[beg],
(State)drawing_states[end]);
                tmp = tr.ReadLine();
                string[] qqq = tmp.Split(',');
                for (int j = 0; j < incount; j++)
                {
                    ar.inceptions.Add(int.Parse(qqq[j]));
                }

                ((State)drawing_states[beg]).outArrows.Add(ar);
                ((State)drawing_states[end]).inArrows.Add(ar);
```

```
                drawing_arrows.Add(ar);
            }
            Refresh();
            return true;
        }

        public void new_auto()
        {
            StateNumber = 0;
            drawing_states.Clear();
            drawing_arrows.Clear();
            Refresh();
        }

        private void OnPaint(object sender, PaintEventArgs e)
        {

            cur_g = e.Graphics;

            height = this.Height;
            width  = this.Width;
            work_place.Width = width;
            work_place.Height = height;
            work_place.X = 0;
            work_place.Y = 0;
            cur_g.FillRectangle(cur_brush, work_place);

            //border
            work_place.Width = width - 1;
            work_place.Height = height - 1;
            cur_g.DrawRectangle(cur_pen, work_place);
            work_place.X = 1;
            work_place.Y = 1;

//
cur_g.DrawString(drawing_arrows.Capacity.ToString() + " " +
i.ToString(), new Font("Verdana", 20), new
SolidBrush(Color.Black), new PointF(20, 20));
            foreach (Shape elem in drawing_arrows)
            {
                elem.drawing(cur_g);
            }

            foreach (Shape elem in drawing_states)
                elem.drawing(cur_g);

        }

        //попали в состояние - выделяем(true), нет -
создаём(false)
        private bool state_mouse_down(int px, int py)
        {
            short res;
```

```
            foreach (State elem in drawing_states)
                if ((res = elem.inside(px, py)) == 1)
                {
                    elem.select_begin();
                    s_selected = elem;
                    return true;
                }
                else if (res == -1)
                    return false;

            //если нажат ctrl, то добавдяем
            if (ctrl_pressed)
            {
                State s = new State(px, py);
                s.setnumber(StateNumber++);
                drawing_states.Add(s);
            }

            return false;
        }

        private bool state_mouse_up(int px, int py)
        {
            if (s_selected != null)
            {
                short res;
                if (ctrl_pressed)
                {
                    foreach (State elem in drawing_states)
                        if ((res = elem.inside(px, py)) == 1)
                        {
                            elem.select_end();

                            Arrow a = new Arrow(s_selected,
elem);
                            drawing_arrows.Add(a);
                            s_selected.outArrows.Add(a);
                            elem.inArrows.Add(a);
                            return true;
                        }
                }
            }

            return false;
        }

        public void setDesignMode(bool b)
        {
            foreach (Shape e in drawing_states)
                e.set_design_mode(b);

            foreach (Shape e in drawing_arrows)
                e.set_design_mode(b);
```

```
        }

        public bool getDesignMode()
        {
            return design_mode;
        }

        private void delete_state(int px, int py)
        {
            ArrayList al = new ArrayList(drawing_arrows.Count);

            foreach(State e in drawing_states)
                if (e.inside(px, py) == 1)
                {
                    drawing_states.Remove(e);
                    for (int i = e.getnumber(); i <
drawing_states.Count; i++)
                        ((State)drawing_states[i]).setnumber(i);

                    StateNumber--;
                    foreach (Arrow a in drawing_arrows)
                        if (!(a.b_state == e || a.e_state == e))
                            al.Add(a);
                        else
                        {
                            a.b_state.outArrows.Remove(a);
                            a.e_state.inArrows.Remove(a);
                        }


                    drawing_arrows = al;

                    return;
                }

        }

        private void OnKeyDown(object sender, KeyEventArgs e)
        {
            ctrl_pressed = e.Control;

            if(!ctrl_pressed)
                alt_pressed = e.Alt;
        }

        private void OnKeyUp(object sender, KeyEventArgs e)
        {
            ctrl_pressed = e.Control;

            if(!ctrl_pressed)
                alt_pressed = e.Alt;
        }
```

```
        public ArrayList getVertexs()
        {
            ArrayList res = new ArrayList(drawing_states.Count);

            foreach (State s in drawing_states)
                res.Add(new Vertex(s));
            return res;
        }

        public ArrayList getStates()
        {
            ArrayList res = new ArrayList(drawing_states.Count);

            foreach (State s in drawing_states)
                res.Add((s));
            return res;
        }

        public ArrayList getEdges()
        {
            ArrayList res = new ArrayList(drawing_arrows.Count);

            foreach (Arrow a in drawing_arrows)
                res.Add(new Edge(a));

            return res;
        }

        public ArrayList getArrows()
        {
            ArrayList res = new ArrayList(drawing_arrows.Count);

            foreach (Arrow a in drawing_arrows)
                res.Add(a);
            return res;
        }

        private void UserControl1_Resize(object sender,
EventArgs e)
        {
            this.Refresh();
        }

        private void OnMouseDown(object sender, MouseEventArgs
e)
        {
            //cur_g.DrawString("Mouse down", new Font("Verdana",
20), new SolidBrush(Color.Black), new PointF(20, 80));
            mouse_x = e.X;
            mouse_y = e.Y;

            if (a_selected != null)
            {
```

```
                a_selected.Select(false);
                a_selected = null;
            }

            MouseButtons mb = e.Button;

            if (mb == MouseButtons.Right)
            {
                if (design_mode)
                    design_mode = false;
                else
                    design_mode = true;

                setDesignMode(design_mode);
            }



            if (design_mode)
            {

                //убираем все выделения
                foreach (State elem in drawing_states)
                    elem.set_pen();

                if (!state_mouse_down(mouse_x, mouse_y))
                    s_selected = null;

                //выделили ли контрольную точку
                foreach (Arrow elem in drawing_arrows)
                    if (elem.point_at(mouse_x, mouse_y))
                    {
                        elem.Select(true);
                        a_selected = elem;
                    }

            }
            else
            {
                foreach (State elem in drawing_states)
                    if (elem.inside(mouse_x, mouse_y) == 1)
                        elem.select_end();
                    else
                        elem.set_pen();

            }

            this.Invalidate();
            //new System.Threading.Thread(new
System.Threading.ThreadStart(this.Refresh)).Start();
            //cur_g.DrawEllipse(cur_pen, 1, 1, 10, 10);

        }
```

```
        private void OnMouseUp(object sender, MouseEventArgs e)
        {
            mouse_x = ((MouseEventArgs)e).X;
            mouse_y = ((MouseEventArgs)e).Y;

            if (alt_pressed && design_mode)
                delete_state(mouse_x, mouse_y);

            if (design_mode)
            {

                if (!state_mouse_up(mouse_x, mouse_y) &&
s_selected != null)
                {
                    s_selected.refresh(mouse_x, mouse_y);
                }


                if (a_selected != null)
                {
                    a_selected.refresh(mouse_x, mouse_y);
                    //a_selected.Select(false);
                }

            }
            //new System.Threading.Thread(new
System.Threading.ThreadStart(this.Refresh)).Start();
            this.Invalidate();

        }

    }

    public class Edge
    {
        public bool Selected()
        {
            return cur_a.Selected();
        }

        public void setText(String t)
        {
            cur_a.setName(t);
        }

        public void setColor(Color c)
        {
            cur_a.setColor(c);
        }

        public Vertex getBeginV()
        {
```

```
            return new Vertex(cur_a.b_state);
        }

        public Vertex getEndV()
        {
            return new Vertex(cur_a.e_state);
        }

        public Edge(Arrow a)
        {
            cur_a = a;
        }

        public void Select(bool b)
        {
            cur_a.Select(b);
        }

        private Arrow cur_a;
    };

    public class Vertex
    {
        public bool Selected()
        {
            return cur_s.Selected();
        }

        public void setR(int r)
        {
            cur_s.setR(r);
        }

        public void setText(String t)
        {
            cur_s.setName(t);
        }

        public void setColor(Color c)
        {
            cur_s.bcg_color.Color = c;
        }

        public void setPosition(Point p)
        {
            cur_s.refresh(p.X, p.Y);
        }

        public ArrayList getInEdges()
        {
           ArrayList res = new ArrayList(cur_s.inArrows.Count);

           foreach (Arrow a in cur_s.inArrows)
```

```
                res.Add(new Edge(a));

            return res;
        }

        public ArrayList getOutEdged()
        {
            ArrayList res = new
ArrayList(cur_s.outArrows.Count);

            foreach (Arrow a in cur_s.outArrows)
                res.Add(new Edge(a));

            return res;
        }

        public void Select(bool b)
        {
            if (b)
                cur_s.select_end();
            else
                cur_s.set_pen();
        }

        public Vertex(State st)
        {
            cur_s = st;
        }

        private State cur_s;
    };

    public abstract class Shape
    {
        public void set_pen(Pen p)
        {
            shape_pen = p;
        }

        public void set_pen()
        {
            shape_pen.Color = Color.Black;
            shape_pen.Width = 1;
        }

        public void set_rect(Rectangle r)
        {
            shape_rect = r;
        }

        public void drawing(Graphics g)
        {
            g.SmoothingMode = SmoothingMode.HighQuality;
```

```
            draw(g);
            g.SmoothingMode = SmoothingMode.None;
        }

        public abstract void Select(bool b);

        public abstract void set_design_mode(bool b);

        public bool isSelected()
        {
            return selected;
        }

        protected abstract void draw(Graphics g);

        protected Pen shape_pen;
        protected Rectangle shape_rect;
        protected bool selected;
        protected bool design_mode;
    };

    class ControlPoint : Shape
    {
        //
        public ControlPoint(Point b)
        {
            selected = false;
            shape_pen = new Pen(Color.LightGray);

            p = new Point(b.X, b.Y);
            shape_rect = new Rectangle(p.X - size, p.Y - size, 2
* size, 2 * size);
            design_mode = true;
        }

        public bool inside(int px, int py)
        {
            if ((px >= p.X - size) && (px <= p.X + size)
               && (py >= p.Y - size) && (py <= p.Y + size))
                return true;

            return false;
        }

        public void refresh(int px, int py)
        {
            p.X = px;
            p.Y = py;
            shape_rect.X = px - size;
            shape_rect.Y = py - size;
        }

        public override void Select(bool b)
```

```
        {
            selected = b;

            if (selected)
                shape_pen.Color = Color.YellowGreen;

        }

        public override void set_design_mode(bool b)
        {
            design_mode = b;
            //throw new Exception("The method or operation is
not implemented.");
        }

        public Pen getPen()
        {
            return shape_pen;
        }

        protected override void draw(Graphics g)
        {

            if(design_mode)
                g.DrawRectangle(shape_pen, shape_rect);

            //throw new Exception("The method or operation is
not implemented.");
        }

        public Point p; // сама точка

        private const int size = 5;
    };

    public class Arrow : Shape
    {
        public Arrow(State s1, State s2)
        {
            inceptions = new List<int>();
            name = "";
            b_state = s1;
            e_state = s2;
            selected = false;
            design_mode = true;
            cur_c = Color.Gray;

            begin = new Point(s1.getCenter().X,
s1.getCenter().Y);
            end = new Point(s2.getCenter().X, s2.getCenter().Y);
            shape_pen = new Pen(cur_c);

            //задаём контрольные точки
```

```
            double cur_x1 = 0, cur_y1 = 0;
            double cur_x2 = 0, cur_y2 = 0;
            double k_x, k_y;

            if (s1 != s2)
            {

                k_x = Math.Abs((double)(end.X - begin.X) / 3);
                k_y = Math.Abs((double)(end.Y - begin.Y) / 3);

                if (end.X > begin.X)
                {
                    cur_x1 = begin.X + k_x;
                    cur_x2 = begin.X + 2 * k_x;
                }
                else
                {
                    cur_x1 = begin.X - k_x;
                    cur_x2 = begin.X - 2 * k_x;
                }

                if (end.Y > begin.Y)
                {
                    cur_y1 = begin.Y + k_y;
                    cur_y2 = begin.Y + 2 * k_y;
                }
                else
                {
                    cur_y1 = begin.Y - k_y;
                    cur_y2 = begin.Y - 2 * k_y;
                }
            }
            else
            {
                cur_x1 = begin.X + 2 * s1.getR();
                cur_y1 = begin.Y + s1.getR();
                cur_x2 = begin.X + s1.getR();
                cur_y2 = begin.Y + 2 * s1.getR();
            }

            p1 = new ControlPoint(new
Point((int)cur_x1,(int)cur_y1));
            p2 = new ControlPoint(new
Point((int)cur_x2,(int)cur_y2));

        }


        /* выбрали контрольную точку, изменили её
         * позицию, обновляем линию, передавая
         * последние координаты.
         */
```

```
public void refresh(int px, int py)
{
    ControlPoint sp = null;

    if (p1.isSelected())
        sp = p1;
    else if (p2.isSelected())
        sp = p2;
    else
        return;

    sp.refresh(px, py);
}

public bool point_at(int px, int py)
{
    if (p1.inside(px, py))
    {
        p1.Select(true);
        p2.getPen().Color = Color.LightPink;
        return true;
    }

    if (p2.inside(px, py))
    {
        p2.Select(true);
        p1.getPen().Color = Color.LightPink;
        return true;
    }

    return false;
}

public override void Select(bool b)
{
    selected = b;

    if (!selected)
    {
        p1.Select(false);
        p2.Select(false);
        p1.getPen().Color = Color.LightGray;
        p2.getPen().Color = Color.LightGray;
    }
}

public bool Selected()
{
    return selected;
}

public override void set_design_mode(bool b)
{
```

```
            p1.set_design_mode(b);
            p2.set_design_mode(b);
            //throw new Exception("The method or operation is
not implemented.");
        }

        protected override void draw(Graphics g)
        {
            p1.drawing(g);
            p2.drawing(g);

            g.SmoothingMode = SmoothingMode.HighQuality;

            if (selected)
                shape_pen.Color = Color.Red;
            else
                shape_pen.Color = cur_c;

            shape_pen.EndCap = LineCap.ArrowAnchor;
            shape_pen.Width = 1;
            clip();
            g.DrawBezier(shape_pen, begin, p1.p, p2.p, end);
            FontFamily ff = new
FontFamily(System.Drawing.Text.GenericFontFamilies.Serif);
            Font f = new Font(ff, 12);
            SolidBrush sb = new SolidBrush(Color.Black);
            String incept = "";
            for (int i = 0; i < inceptions.Count; i++)
                if (i != inceptions.Count - 1)
                    incept += inceptions[i].ToString() + ", ";
                else
                    incept += inceptions[i].ToString();

            g.DrawString(incept, f, sb, new Point((p1.p.X +
p2.p.X) / 2, (p1.p.Y + p2.p.Y) / 2));

            g.DrawString(name, f, sb, new Point((p1.p.X +
p2.p.X)/2,(p1.p.Y + p2.p.Y)/2));


            //throw new Exception("The method or operation is
not implemented.");
        }

        private void refreshBeginEnd()
        {
            begin.X = b_state.getCenter().X;
            begin.Y = b_state.getCenter().Y;

            end.X = e_state.getCenter().X;
            end.Y = e_state.getCenter().Y;
        }
```

```
        public void setName(String s)
        {
            name = s;
        }

        public string getName()
        {
            return (name);
        }

        public void setColor(Color c)
        {
            cur_c = c;
        }

        //вырезаем части, которые находяться внутри круга
        //устанавливаем начало и конец на границу круга
        private void clip()
        {
            refreshBeginEnd();
            int r = e_state.getR();

            double l = Math.Sqrt((p2.p.X - end.X) * (p2.p.X -
end.X) +
                                 (p2.p.Y - end.Y) * (p2.p.Y -
end.Y));

            double k = (l - r) / l;

            Point p = new Point((int)(p2.p.X + (end.X - p2.p.X)
* k),
                                          (int)(p2.p.Y + (end.Y -
p2.p.Y) * k));
            end = p;
        }

        private Point begin;
        private Point end;
        public List<int> inceptions;
        private Color cur_c;
        private ControlPoint p1, p2;
        public State b_state, e_state;
        private String name;
    }

    public class State : Shape
    {

        public State(int pX, int pY)
        {
            shape_pen = new Pen(Color.Black);
            shape_rect = new Rectangle(pX - r, pY - r, 2 * r, 2
* r);
```

```
            name = "";
            design_mode = true;
            bcg_color = new SolidBrush(Color.PaleGreen);
            inArrows = new ArrayList();
            outArrows = new ArrayList();
        }

        public void setR(int rad)
        {
            r = rad;
        }

        //попали 1
        //нет, но расстояние от этой точки меньше 2х радиусов -
1(т.е. рисовать нельзя)
        //больше, тогда 0
        public short inside(int pX, int pY)
        {
            double l = Math.Sqrt((pX - shape_rect.X - r) * (pX -
shape_rect.X - r) +
                                 (pY - shape_rect.Y - r) * (pY
- shape_rect.Y - r));

            if ((r - l) >= 0)
                return 1;
            else if (l <= 2 * r)
                return -1;
            else
                return 0;
        }

        public Point getCenter()
        {
            return new Point(shape_rect.X + r, shape_rect.Y +
r);
        }

        public void setName(String s)
        {
            if (s.Length < 5)
                name = s;
            else
                name = s.Substring(0, 5);
        }

        public string getname()
        {
            return name;
        }

        public int getR() { return r; }

        public void refresh(int px, int py)
```

```
        {
            shape_rect.X = px - r;
            shape_rect.Y = py - r;
        }

        public override void set_design_mode(bool b)
        {
            design_mode = b;
            //throw new Exception("The method or operation is
not implemented.");
        }


        public override void Select(bool b)
        {
            selected = b;
        }

        public bool Selected()
        {
            return selected;
        }

        public void select_begin()
        {
                Select(true);
                shape_pen.Color = Color.Green;
                shape_pen.Width = 2;
        }

        public void select_end()
        {

             Select(true);
             shape_pen.Color = Color.Red;
             shape_pen.Width = 2;
        }

        public void setnumber(int number)
        {
            this.number = number;
        }

        public int getnumber()
        {
            return this.number;
        }

        public void setoutput(int output)
        {
            this.output = output;
        }
```

```
        public int getoutput()
        {
            return this.output;
        }

        protected override void draw(Graphics g)
        {
            g.FillEllipse(bcg_color, shape_rect);
            g.DrawEllipse(shape_pen, shape_rect);


            FontFamily ff = new
FontFamily(System.Drawing.Text.GenericFontFamilies.Serif);
            Font f = new Font(ff, 12);
            SolidBrush sb = new SolidBrush(Color.Black);
            g.DrawString(this.number.ToString(), f, sb, new
Point(shape_rect.X + shape_rect.Width/3, shape_rect.Y +
shape_rect.Height/3));
            g.DrawString(name,f,sb,new Point(shape_rect.X - 5,
shape_rect.Y + shape_rect.Height + 5));
           // g.DrawString(name,

            //g.DrawRectangle(shape_pen, shape_rect);
            //  throw new Exception("The method or operation is
not implemented.");
        }

        public SolidBrush bcg_color;
        private static int r = 25;
        private String name;
        private int number;
        private int output;
        public ArrayList inArrows;
        public ArrayList outArrows;
    };

}
```

RoboController.java

```
import com.cyberbotics.webots.controller.Robot;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.net.ServerSocket;
```

```
import java.net.Socket;
import java.util.Random;
import com.cyberbotics.webots.controller.DifferentialWheels;
import com.cyberbotics.webots.controller.DistanceSensor;


public class RoboController extends DifferentialWheels {
  Socket sock;
  BufferedReader br;
  BufferedWriter bw;
  final int MAX_INPUTS = 10;
  DistanceSensor sens;

  void Init() {
      try {
          ServerSocket ssock = new ServerSocket(777);
          sock = ssock.accept();
          br = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
          bw = new BufferedWriter(new
OutputStreamWriter(sock.getOutputStream()));
      } catch (IOException e) {
          System.out.println(e.toString());
      }
  }

  class State {
    int newstate[];
    int output;

    public State() {
```

```
    newstate = new int[MAX_INPUTS];
    for (int i = 0; i < MAX_INPUTS; i++) {
      newstate[i] = -1;
    }
    output = -1;
  }
}

class Command {
  public int value;
  public int command;

  Command() {
    value = -1;
    command = -1;
  }
}

int GetValue(String line, int startindex) {
  int val = 0;
  String tmp = line.substring(startindex, line.length());
  val = Integer.parseInt(tmp);
  return val;
}

Command ParseCommand(String line) {
  Command comm = new Command();
  if (line.length() >= 3) {
    if (line.substring(0, 3).matches("END")) {
      comm.command = 7;
    }
```

```
if (line.length() >= 5) {
  if (line.substring(0, 5).matches("Reset")) {
    comm.command = 0;
  }
}

if (line.length() >= 7) {
  if (line.substring(0, 5).matches("Delay")) {
    comm.command = 6;
    comm.value = GetValue(line, 6);
  }
}

if (line.length() >= 9) {
  if (line.substring(0, 7).matches("GetLine")) {
    comm.command = 4;
    comm.value = GetValue(line, 8);
  }
}

if (line.length() >= 11) {
  if (line.substring(0, 9).matches("LeftSpeed")) {
    comm.command = 1;
    comm.value = GetValue(line, 10);
  }

  if ((line.substring(0, 11)).matches("GetDistance")) {
    comm.command = 3;
  }
```

```
      if (line.substring(0, 11).matches("SetAutomata")) {
        comm.command = 5;
      }
    }

    if (line.length() >= 12) {
      if (line.substring(0, 10).matches("RightSpeed")) {
        comm.command = 2;
        comm.value = GetValue(line, 11);
      }
    }
  }

  return comm;
}

int readint() {
  try {
    int tmp = 0;
    while ((tmp = br.read()) == -1) {
      try {
        Thread.sleep(10);
      } catch (InterruptedException ex) {
        System.out.println(ex.toString());
      }
    }

    return tmp;
  } catch (IOException ex) {
    System.out.println(ex.toString());
  }
```

```
  return (-1);
}

public RoboController() {


  super();

}

void move(int leftspeed, int rightspeed) {
  System.out.println(leftspeed + " " + rightspeed);
  setSpeed(leftspeed, rightspeed);
}

int distance() {
  int tmp = (int)(sens.getValue() /1.5);
  //System.out.println("Distance: " + tmp);
  return(tmp);
}

void delay(int delay) {
  int tmp = delay / 32;
  tmp *= 32;
  step(tmp);
}

boolean line(int number) {
  return(true);
}
```

```
public void run() {
try{
  Init();
  sens = getDistanceSensor("SENS");
  sens.enable(64);
  String line;
  Command comm = new Command();
  boolean reset = false;

  do {
    line = br.readLine();
    comm = ParseCommand(line);
    reset = false;
    if (comm.command == 5) {
      int defstate = -1;
      int states = -1;
      int output = -1;
      int transition = -1;

      defstate = readint();
      states = readint();

      State Automata[] = new State[states];

      for (int i = 0; i < states; i++) {
        Automata[i] = new State();
        output = readint();
        Automata[i].output = output;

        transition = readint();
```

```
    int newstate = -1;
    int input = -1;

    for (int j = 0; j < transition; j++) {
      input = readint();
      newstate = readint();
      Automata[i].newstate[input] = newstate;
    }

  }

  while (true) {
    int input = -1;
    int d = distance();

    if (d < 100) {
      input = 0;
    } else if (d < 135) {
      input = 1;
    } else if (d < 230) {
      input = 2;
    } else if (d < 1000) {
      input = 3;
    }

    int newst = Automata[defstate].newstate[input];
    if (newst != -1) {
      defstate = newst;
    }
    int outp = Automata[defstate].output;
```

```
      if (outp == 0) {
        move(0, 0);
      } else if (outp == 1) {
        move(128, 128);
      } else if (outp == 2) {
        move(-128, 128);
      } else if (outp == 3) {
        move(128, -128);
      } else if (outp == 4) {
        move(-128, -128);
      }

      delay(32);

      if (sock.getInputStream().available() > 0) {
        br.readLine();
        move(0, 0);
        break;
      }
    }

  } else {
    int leftspeed = 0;
    int rightspeed = 0;
    int dist = -1;
    int del = 0;
    boolean lin = false;

    while (true) {
      while (comm.command != 7) {
        switch (comm.command) {
```

```
                case 0:
                  reset = true;
                  move(0, 0);
                  break;
                case 1:
                  leftspeed = comm.value;
                  //System.out.println("Accept leftspeed: " +
leftspeed);
                  break;
                case 2:
                  rightspeed = comm.value;
                  //System.out.println("Accept rightspeed: " +
rightspeed);
                  break;
                case 3:
                  dist = distance();
                  bw.write("Distance " + dist);
                  bw.newLine();
                  bw.flush();
                  break;
                case 4:
                  lin = line(comm.value);
                  bw.write("Line " + comm.value + " ");
                  if (lin) {
                    bw.write("1");
                  } else {
                    bw.write("0");
                  }
                  bw.newLine();
                  bw.flush();
                  break;
```

```
    case 6:
      del = comm.value;
      break;
  }
  if (reset) {
    break;
  }
  if (leftspeed > 255) {
    leftspeed = 255 - leftspeed;
  }
  if (rightspeed > 255) {
    rightspeed = 255 - rightspeed;
  }
  line = br.readLine();
  comm = ParseCommand(line);
}

if (reset) {
  break;
}
move(leftspeed, rightspeed);
if(del == 0)
  delay(32);
if (del != 0) {
  delay(del);
  leftspeed = 0;
  rightspeed = 0;
  move(0, 0);
}
bw.write("OK!");
bw.newLine();
```

```
            bw.flush();

            line = br.readLine();
            comm = ParseCommand(line);

          }
        }
      } while (step(64) != -1);
      } catch (IOException e) {
        System.out.println(e.toString());
      }
    }
    public static void main(String[] args) {
      RoboController controller = new RoboController();
      controller.run();
    }
}
```