робосимулятор

v-rep

СГАУ кафедра АСЭУ
© <dponyatov@gmail.com>

8 января 2016 г.

Оглавление

Введение

1

Программирование роботов — это интересно.

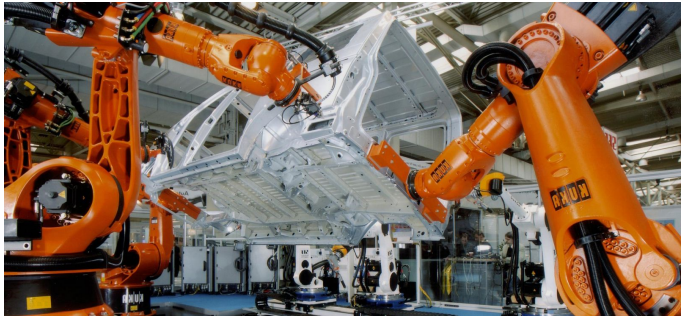
Многие наверное видели японских гуманоидных роботов, или французский учебный робот NAO, интересным выглядит проект обучаемого робота-манипулятор Baxter. Промышленные манипуляторы KUKA из Германии — это классика. Кто-то программирует системы конвейерной обработки (фильтрации, сортировки).

[1] © [2]

Дельта роботы. Есть целый пласт — управление квадрокоптером/алгоритмы стабилизации. И конечно же простые трудяги на складе — Line Follower. *И самый доступный для самостоятельного изготовления, и в то же время полезный вариант — **универсальные роботележки** на колесном или гусеничном ходу, с размерами от настольного микробота размером с мышку, до танков и карьерных робоэлектровозов.*


NAO


Вахтёр


KUKA

Но всё это как правило — не дешевые игрушки, поэтому доступ к роботам есть в специализированных лабораториях или институтах/школах где получили финансирование и есть эти направления. Всем же остальным разработчикам (кому интересна робототехника) — остаётся завистливо смотреть.

Некоторое время назад я вышел на достаточно интересную систему — 3D робосимулятор **v-rep**, от швейцарской компании Coppelia Robotics.

К своему (приятному) удивлению я обнаружил, что эта система:

- имеет большой функционал[2]
- полностью open-source[3]
- кроссплатформенная — $\boxplus Windows$, mac, $Linux$[4]
- имеет API и библиотеки для работы с роботами через C/C^{++}, $Python$, Java, Lua, Matlab, Octave или Urbi

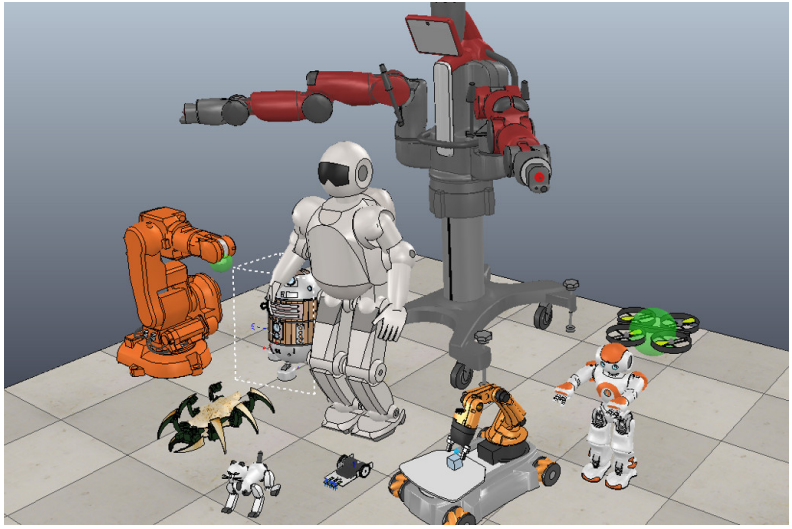
[2] система разрабатывается с марта 2010 года
[3] выложена в открытый доступ в 2013 году
[4] рализована на фреймворке Qt

- бесплатная для некоммерческого использования

Все объекты, которые программируются в этой системе — <живут> в реальном с точки зрения физических законов мире — есть гравитация, можно захватывать предметы, столкновения, датчики расстояния, видео датчики и т.п.

Поработав некоторое время с этой системой, я решил рассказать про неё читателям хабра.

Да, и на картинке скриншот из **v-rep**, и модели роботов — которые вы можете программировать, и смотреть поведение, прямо на вашем компьютере.



Установка

v-rep скачайте и установите[5] версию Pro Edu

[5] Next Next Next

MinGW для сборки плагинов и интерпретатора bI нужен компилятор C^{++}. Сам v-rep собран с помощью GNU G++ (MinGW), поэтому написание плагинов с использованием Visual C не поддерживается.

Скачайте инсталлятор MinGW и установите пакеты g++, flex и bison.

mingw-get-setup.exe » Install » Directory » **C:/MinGW**

☒ ... also install support for the GUI | ☒ ... in the start menu | ☐ ... on the desktop

Continue » Continue

Basic Setup

☒ **mingw32-base** » ◁ » Mark for Installation

☒ **mingw32-gcc-g++**

All Packages » MSYS » MinGW Developer Toolkit

☒ **msys-bison.bin**

☒ **msys-flex.bin**

Installation » Apply Changes » Apply » Close

Для запуска утилит нужно добавить пути поиска в системную переменную PATH

⊞ » Компьютер » ▷ » Свойства » Дополнительные параметры » Переменные среды

Переменные среды пользователя...

```
PATH = C:\MinGW\msys\1.0\bin;C:\MinGW\bin;...
```

Обновление

MinGW

⊞ » Все программы » MinGW Installation Manager


Installation
Update Catalogue
Close


Installation
Mark All Upgrades


Installation
Apply Changes

Глава 1

Расширение платформы

1.1 Плагины

[1]

A plugin is a shared library (e.g. a dll) that is automatically loaded by V-REP's main client application at program start-up (in a future release, it will also be possible to load/unload plugins on-the-fly). It allows V-REP's functionality to be extended by user-written functions (in a similar way as with add-ons). The language can be any language able to generate a shared library and able to call exported C-functions (e.g. in the case of Java, refer to GCJ and IKVM). A plugin can also be used as a wrapper for running code written in other languages or even written for other microcontrollers (e.g. a plugin was written that handles and executes code for Atmel microcontrollers).

Plugins are usually used to customize the simulator and/or a particular simulation. Often, plugins are only used to provide a simulation with custom Lua commands, and so are used in conjunction with scripts. Other times, plugins are used to provide V-REP with a special functionality requiring either fast calculation capability (scripts are most of

[1] © http://www.coppeliarobotics.com/helpFiles/en/plugins.htm

the times slower than compiled languages) or an interface to a hardware device (e.g. a real robot).

Each plugin is required to have following 3 entry point procedures:

```
extern "C" __declspec(dllexport) unsigned char v_repStart(void* reserved,int reservedInt);
extern "C" __declspec(dllexport) void v_repEnd();
extern "C" __declspec(dllexport) void* v_repMessage(int message,int* auxiliaryData,void* custom
```

If one procedure is missing then the plugin will be unloaded and won't be operational. Refer to the console window at start-up for the loading status of plugins. Following briefly describes above three entry point purpose:

v_repStart

This procedure will be called one time just after the main client application loaded the plugin. The procedure should:

- check whether the version of V-REP is same or higher than the one that was used to develop the plugin (just make sure all commands you use in the plugin are supported!).
- allocate memory, and prepare GUI related initialization work (if required).
- register custom Lua functions (if required).
- register custom Lua variables (if required) .
- return the version number of this plugin if initialization was successful, otherwise 0. If 0 is returned, the plugin is unloaded and won't be operational.

v_repEnd

This procedure will be called one time just before the simulation loop exits. The procedure should release all resources reserved since `v_repStart` was called.

v_repMessage

This procedure will be called very often while the simulator is running. The procedure is in charge of monitoring messages of interest and reacting to them. It is important to react to following events (best by intercepting the `sim_message_eventcallback_instancepass` message) depending on your plugin's task:

- When objects were created, destroyed, scaled, or when models are loaded: make sure you reflect the change in the plugin (i.e. synchronize the plugin with the scene content)
- When scenes were loaded or the undo/redo functionality called: make sure you erase and reconstruct all plugin objects that are linked to the scene content
- When the scene was switched: make sure you erase and reconstruct all plugin objects that are linked to the scene content. In addition to this, remember that a scene switch will discard handles of following items:
  - communication tubes
  - signals
  - banners
  - drawing objects
  - etc.
- When the simulator is in an edit mode: make sure you disable any "special functionality"provided by the plugin, until the edit mode was ended. In particular, make sure you do not programmatically select scene objects.
- When a simulation was launched: make sure you initialize your plugin elements if needed
- When a simulation ended: make sure you release any memory and plugin elements that are only required during simulation
- When the object selection state was changed, or a dialog refresh message was sent: make sure you actualize the dialogs that the plugin displays

Refer to the messages of type `sim_message_eventcallback_` for more details. When writing plugins several additional points have to be observed or taken into account:

Recommended topics

The main client application

Robot language interpreter plugin tutorial

Литература

[1] v-rep: virtual robot experimentation platform

[2] Habr Программируем роботов — бесплатный робосимулятор **v-rep**. Первые шаги