ARS

Release 0.5b1

Germán Larraín

February 28, 2017

Contents

Note: This software and its documentation are currently under development so they will be subject to changes. Contributions are welcome!

Introduction

Welcome! This is the documentation of ARS 0.5, last updated on February 28, 2017.

ARS is written in a marvelous programming language called Python. One of the many features that make it great (and popular) is its documentation. Taking that into consideration, many sections herein were taken from the official Python documentation.

So simple

To create and run your first simulation, a few lines of code are enough! Execute the following script (press key `q` or `e` to end the simulation)

```
from ars.app import Program

class FallingBall(Program):

   def create_sim_objects(self):
      # add_sphere's arguments: radius, center, density
      self.sim.add_sphere(0.5, (1, 10, 1), density=1)

sim_program = FallingBall()
sim_program.start()
sim_program.finalize()
```

Official website

The repository is hosted at BitBucket where you will find the source code and the issue tracker. We would love that you request features or improvements for ARS. Also, bug reports are more than welcome.

Because we like the Python community and the tools they use, it is registered in PyPI (Python Package Index). Although useful for organizing packages, the main benefit is to be able to install (and upgrade) ARS using the `pip` program. It takes just 3 words:

```
$ pip install ARS
```

(well, you might have to prepend `sudo` if you are using Linux and running as a user without the required priveges. Bummer, that's 4 words now...)

For support, check the Google group and join if you want to post a message.

Releases

version	date	revision
0.5b1	2013.12.13	8cde845244ae
0.5a2	2013.10.21	9fa5876718f0
0.5a1	2013.09.25	60c96b5b55ba
0.4a1	2013.04.13	f9c1381290bc
0.3a1	2012.10.17	b9190db2b909

News

What has happened lately...

Contents

Installation

ARS itself is really easy to install :) but some of its *Prerequisites* are not :(, depending on which operating system is used. The best option is Ubuntu, specially its 12.04 (i.e. *Precise Pangolin*) version. For that, follow the instructions in *Ubuntu & Debian*. If you have another version of Ubuntu, steps may be the same or very similar. For other Linux distros, the steps regarding `apt-get` will probably change.

Mac OSX is untested but it should work because all the required software has been reported to work on it. ARS itself is pure Python and OS-agnostic thus everything should work out if you comply with the requirements.

Microsoft Windows is the OS less friendly to ARS, as well as to many other open source software. Nonetheless, it is very popular so we got it to run there too! Instructions for both XP and 7 are here: *Microsoft Windows*.

Prerequisites

The software required by ARS to run (or install other parts) is listed in the following sections.

Note: On a *NIX system, some actions, such as:

- `apt-get install`
- `make install`
- `python setup.py install`
- `pip install`

require root permission because they affect it as a whole. In those cases prepend `sudo`.

Python-related packages

Obviously you need Python (i.e. the interpreter and the basic libraries) but also its header files, static library, and the 'update-python-modules'. Don't be scared by those names, they are quite common for software, even if you had not heard them before.

```
apt-get install python python-dev python-support
```

Other Python packages

Numpy (Numerical Python) is a very popular package for scientific work. It adds "support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays." (Wikipedia)

Cython is required to build ODE's Python bindings. It "is a language that makes writing C extensions for the Python language as easy as Python itself. It is based on the well-known Pyrex, but supports more cutting edge functionality and optimizations." (cython.org)

```
apt-get install python-numpy cython
```

Visualization Toolkit (VTK)

"The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, image processing and visualization. VTK consists of a C++ class library and several interpreted interface layers including Tcl/Tk, Java, and Python." (vtk.org)

```
apt-get install libvtk5.8 python-vtk
```

Note: VTK source code packages include the Python wrappers but the default setup for most systems will not install them. In general, the binary distributions of VTK do not include the Python wrappers.

Open Dynamics Engine (ODE)

"ODE is an open source, high performance library for simulating rigid body dynamics. It is fully featured, stable, mature and platform independent with an easy to use C/C++ API. It has advanced joint types and integrated collision detection with friction. ODE is useful for simulating vehicles, objects in virtual reality environments and virtual creatures. It is currently used in many computer games, 3D authoring tools and simulation tools." (ode.org)

Library

After unpacking archive `ode-0.12.tar.gz` (or `.zip`, `.tar.bz2`):

```
cd ode-0.12
./configure --enable-double-precision --with-trimesh=opcode --enable-new-trimesh --enable-shared
make
make install
```

Warning: If you downloaded ODE's source code from its Subversion repository instead of getting a release version you must 'bootstrap' the process by running `sh autogen.sh` before `configure`.

Python bindings

Run the following at the command line:

```
cd bindings/python/
python setup.py install
```

Standard installation

Once all the prerequisites are satisfied, ARS can be installed in any of these simple alternatives. **To install ARS we recommend you to use pip**.

pip

It "is a tool for installing and managing Python packages, such as those found in the Python Package Index. It's a replacement for easy_install." (pip-installer.org)

This program is not part of standard Python installations. However, it is widely used because it has `easy_install`'s functionality plus uninstall and upgrade features.

```
pip install ARS
```

Easy Install

It "is a Python module (`easy_install`) bundled with `setuptools` that lets you automatically download, build, install, and manage Python packages." (packages.python.org/distribute)

```
easy_install ARS
```

setup.py install

This is the standard way to install a module distribution. It uses the official and built-in `distutils` package.

For this, you need to download ARS, extract the contents from the archive (e.g. `.zip`, `.tar.bz2`), and run (from its root directory):

```
python setup.py install
```

Ubuntu & Debian

Ubuntu and Debian are distributions ('distros') of GNU/Linux (a.k.a. Linux).

Note: Ubuntu is Debian-based thus they share much code and packages.

For these OSs, the complete list of steps is available (and has been thoroughly checked), including ODE's compilation and installation, which are the major obstacles in the process and may scare users unaccustomed to the command line and software compilation.

Ubuntu 12.04

Ubuntu's latest long-term-support release is 12.04 (a.k.a. Precise Pangolin) and was published on April, 2012.

Run the following commands in sequence and you'll have ARS ready to go. They are grouped as: a) Python basic packages, b) VTK, c) ODE, d) ARS:

```
sudo apt-get install python-dev python-support python-pip
sudo apt-get install python-numpy cython

sudo apt-get install libvtk5.8 python-vtk

sudo apt-get install make autoconf automake libtool g++ pkg-config
wget https://downloads.sourceforge.net/project/opende/ODE/0.12/ode-0.12.tar.bz2
tar xf ode-0.12.tar.bz2
cd ode-0.12/
```

```
./configure --enable-double-precision --with-trimesh=opcode --enable-new-trimesh --enable-shared
make
sudo make install
cd bindings/python/
sudo python setup.py install
sudo ldconfig

sudo pip install ARS
```

Notice how ODE is the software piece that takes more commands and time to install. Fortunately, ARS is the exact opposite.

Note: Not all users will need to run `ldconfig` but it doesn't hurt if you do. See *ODE Import Error* for more information.

Debian 7

Debian's next stable version is 7 (a.k.a. Wheezy) and is about to be released in the first half of 2013. Debian 6 (a.k.a. Squeeze) had old versions of Python and VTK thus was not considered for these 'easy install' instructions.

The **required steps are identical** as those above, with one caveat:

Warning: Debian might not have `sudo` available or the OS user might not be in the `sudo` group. The solution (you need the root user password) for these is:

```
su
apt-get install sudo
adduser <username> sudo
exit
```

and then logout and login again.
An alternative is to replace `sudo <command>` with `su -c '<command>'`.

Microsoft Windows

Download and install the requirements:

- Numpy (mirror)
- ODE (mirror)
- VTK (mirror)

Now download the MS Windows installer of ARS from PyPI and install it.

Troubleshooting

ODE

Import error 1

```
>>> import ode
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named ode
```

It means the `ode` module could not be **found** by the Python interpreter. If the module (file `ode.so`) was built correctly it will be located in a directory named something like `~/ode-0.12/bindings/python/build/lib.linux-x86_64-2.7/`. If that's the case, you only forgot to execute:

```
~/ode-0.12/bindings/python$ sudo python setup.py install
```

which, among other things, copies `build/lib.linux-x86_64-2.7/ode.so` to directory `/usr/local/lib/python2.7/dist-packages`.

Import error 2

It means the `ode` module could not be **imported** by the Python interpreter.

```
>>> import ode
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ImportError: libode.so.3: cannot open shared object file: No such file or directory
```

This error is saying that ODE needs access access to the compiled library (as a shared object, e.g. `libode.so.3`) but can't find it. It is not about whether `sys.path` contains or not the location of the library.

Dynamically linked libraries are looked up in the system library path, i.e. the directories included in the `LD_LIBRARY_PATH` environment variable. However, instead of manually fixing this value, try running `ldconfig` –its job is to "configure dynamic linker run-time bindings" (man page)– and import `ode` again. Hopefully you'll get no output, which means it was imported correctly.

VTK

vtkXOpenGLRenderWindow

If VTK looks to be installed correctly (e.g. it can be imported with no errors) but while running a program using VTK you get an error mentioning `vtkXOpenGLRenderWindow`, then probably you have an error related to your video card, its drivers and OpenGL. You can test the latter works fine by running the programs `glxgears` or `glxinfo` at the command line (if they are not found you can install them with `sudo apt-get install mesa-utils`).

Note: If you can't get OpenGL to work, then there is no way VTK will work in your system, and probably most visualization software won't too.

About ARS

History

This project was conceived, designed and started in RAL (Robotics and Automation Laboratory) at PUC's (Pontificia Universidad Católica de Chile) engineering school.

External links

- ARS is registered at PyPI (Python Package Index).
- Its code is managed in a Mercurial repository hosted at Bitbucket.
- The documentation is hosted at ReadTheDocs and it is generated dynamically after each commit to the repository.
- RAL (Robotics and Automation Laboratory) is the organization under which this project was conceived and also obtained funds for it.
- A page in Ohloh tracks code commits and makes some analyses and estimations of the project.
- Another page in Freecode does sometings similar.

item	URL
Code repository	Bitbucket
Source code	https://bitbucket.org/glarrain/ars/src
Downloads 1	https://bitbucket.org/glarrain/ars/downloads
Downloads 2	http://sourceforge.net/projects/arsproject/files/
Downloads 3	PyPI
Issue tracker	IssueTracker
Mailing list	GoogleGroup
PyPI	PyPI
SourceForge	SourceForge
Ohloh	Ohloh
Freecode	Freecode
Crate.io	Crate

FAQ

Release 0.5

Date February 28, 2017

Well, there aren't many yet :)

ars

ars Package

`ars` Package

ARS is a physically-accurate robotics simulator written in Python. It's main purpose is to help researchers to develop mobile manipulators and, in general, any multi-body system. It is open-source, modular, easy to learn and use, and can be a valuable tool in the process of robot design, in the development of control and reasoning algorithms, as well as in teaching and educational activities.

`get_version`(**args*, ***kwargs*)

`constants` Module

`convert_color`(*R_int*, *G_int*, *B_int*)

`exceptions` Module

ARS's exceptions class hierarchy.

exception `ArsError`(*msg=None*)

Bases: `exceptions.Exception`

Base class for exceptions in this library.

Attributes: msg – explanation of the error

exception `JointError`(*joint*, *msg=None*)

Bases: *`ars.exceptions.PhysicsEngineException`*

Exception raised for errors related to physical joints.

Attributes: joint – joint in which the error occurred msg – explanation of the error

exception `PhysicsEngineException`(*msg=None*)

Bases: *`ars.exceptions.ArsError`*

Exception raised for errors in a physics engine.

Attributes: msg – explanation of the error

exception `PhysicsObjectCreationError`(*type_*, *msg=None*)

Bases: *`ars.exceptions.PhysicsEngineException`*

Exception raised for errors in physics-engine objects creation.

Attributes: type – type of the object being created msg – explanation of the error

Subpackages

app Package

app Package Main package of the software. It contains the Program class which is the core application controller.

class `ActionMap`

Bases: `object`

`add`(*key*, *value*, *repeat=False*)

`call`(*key*)

`get`(*key*, *default=None*)

`get_function`(*key*)

`has_key`(*key*)

`is_repeat`(*key*)

class `Program`

Bases: `object`

Main class of ARS.

To run a custom simulation, create a subclass. It must contain an implementation of the 'create_sim_objects' method which will be called during the simulation creation.

To use it, only two statements are necessary:

•**create an object of this class**

```
>>> sim_program = ProgramSubclass()
```

•**call its 'start' method**

```
>>> sim_program.start()
```

Constructor. Defines some attributes and calls some initialization methods to:

•set the basic mapping of key to action,

•create the visualization window according to class constants,

•create the simulation.

BACKGROUND_COLOR = (1, 1, 1)

CAMERA_POSITION = (10, 8, 10)

FLOOR_BOX_SIZE = (10, 0.01, 10)

FPS = 50

STEPS_PER_FRAME = 50

WINDOW_POSITION = (0, 0)

WINDOW_SIZE = (1024, 768)

WINDOW_TITLE = 'ARS simulation'

WINDOW_ZOOM = 1.0

create_screenshot_recorder(*base_filename*, *periodically=False*)

Create a screenshot (of the frames displayed in the graphics window) recorder.

Each image will be written to a numbered file according to `base_filename`. By default it will create an image each time *`record_frame()`* is called. If `periodically` is `True` then screenshots will be saved in sequence. The time period between each frame is determined according to *`FPS`*.

create_sim_objects()

This method must be overriden (at least once in the inheritance tree) by the subclass that will instatiated to run the simulator.

It shall contain statements calling its 'sim' attribute's methods for adding objects (e.g. add_sphere).

For example:

```
>>> self.sim.add_sphere(0.5, (1,10,1), density=1)
```

create_simulation(*add_axes=True*, *add_floor=True*)

Creates an empty simulation and:

1.adds basic simulation objects (`add_basic_simulation_objects()`),

2.**(if `add_axes` is `True`) adds axes to the visualization at the** coordinates-system origin,

3.**(if `add_floor` is `True`) adds a floor with a defined normal** vector and some visualization parameters,

4. **calls** ***create_sim_objects()*** **(which must be implemented by** subclasses),

5. **gets the actors representing the simulation objects and adds them to** the graphics adapter.

finalize()
Finalize the program, deleting or releasing all associated resources.

Currently, the following is done:

- **the graphics engine is told to** *ars.graphics.base.Engine.finalize_window()*
- all attributes are set to None or False

A finalized program file cannot be used for further simulations.

Note: This method may be called more than once without error.

on_action_selection(*key*)
Method called after an actions is selected by pressing a key.

on_pre_frame()
This method will be called before each visualization frame is created. It is meant to be, optionally, implemented by subclasses.

on_pre_step()
This method will be called before each integration step of the simulation. It is meant to be, optionally, implemented by subclasses.

record_frame()
Record a frame using a screenshot recorder.

If frames are meant to be written periodically, a new one will be recorded only if enough time has elapsed, otherwise it will return `False`. The filename index will be `time / period`.

If frames are not meant to be written periodically, then index equals simulator's frame number.

reset_simulation()
Resets the simulation by resetting the graphics adapter and creating a new simulation.

set_key_2_action_mapping()
Creates an Action map, assigns it to `key_press_functions` and then adds some `(key, function` tuples.

start()
Starts (indirectly) the simulation handled by this class by starting the visualization window. If it is closed, the simulation ends. It will restart if `do_create_window` has been previously set to `True`.

graphics Package

base Module

class **Axes**(*pos=(0, 0, 0)*, *rot=None*, *cylinder_radius=0.05*)
Bases: *ars.graphics.base.Entity*

class **Body**(*center*, *rot*)
Bases: *ars.graphics.base.Entity*

Entity representing a defined body with a given color.

get_color()

set_color(*color*)

class Box (*size*, *pos*, *rot=None*)
Bases: *ars.graphics.base.Body*

class Capsule (*length*, *radius*, *center*, *rot=None*, *resolution=10*)
Bases: *ars.graphics.base.Body*

class Cone (*height*, *radius*, *center*, *rot=None*, *resolution=100*)
Bases: *ars.graphics.base.Body*

Constructor.

Parameters **resolution** (*int*) – it is the circumferential number of facets

class Cylinder (*length*, *radius*, *center*, *rot=None*, *resolution=10*)
Bases: *ars.graphics.base.Body*

class Engine (**args*, ***kwargs*)
Bases: `object`

Abstract class. Not coupled (at all) with VTK or any other graphics library

add_object (*obj*)
Add `obj` to the visualization controlled by this adapter.

Parameters **obj** (*Body*) –

add_objects_list (*obj_list*)

finalize_window ()
Finalize window and remove/clear associated resources.

remove_object (*obj*)
Remove `obj` from the visualization controlled by this adapter.

Parameters **obj** (*Body*) –

reset ()

restart_window ()

start_window (*on_idle_callback*, *on_reset_callback*, *on_key_press_callback*)

class Entity (*pos*, *rot*)
Bases: `object`

Renderable and movable object.

It has position and orientation. The underlying object is *actor*, which connects to the real entity handled by the graphics library in use.

actor

adapter = None

set_pose (*pos*, *rot*)

class ScreenshotRecorder (*base_filename*)
Bases: `object`

calc_filename (*index=1*)
Calculate a filename using `index` for a new image.

Parameters **index** (*int*) – image's index to use for filename calculation

Returns image's filename

Return type str

file_extension = None

write (*index*, *time*)
Write render-window's currently displayed image to a file.

The image format (thus the file extension too) to use must be defined by the implementation.

Image's filename is determined by `calc_filename()`.

Parameters

- **index** (`int`) – image's index to use for filename calculation
- **time** –

class Sphere (*radius*, *center*, *rot=None*, *phi_resolution=50*, *theta_resolution=50*)
Bases: `ars.graphics.base.Body`

Constructor.

Parameters

- **phi_resolution** (`int`) – resolution in the latitude (phi) direction
- **theta_resolution** (`int`) – resolution in the longitude (theta) direction

class Trimesh (*vertices*, *faces*, *pos=None*, *rot=None*)
Bases: `ars.graphics.base.Body`

vtk_adapter Module

class Axes (*pos=(0, 0, 0)*, *rot=None*, *cylinder_radius=0.05*)
Bases: `ars.graphics.vtk_adapter.Entity`, `ars.graphics.base.Axes`

class Body (**args*, ***kwargs*)
Bases: `ars.graphics.vtk_adapter.Entity`

get_color ()
Returns the color of the body. If it is an assembly, it is not checked whether all the objects' colors are equal.

set_color (*color*)
Sets the color of the body. If it is an assembly, all the objects' color is set.

class Box (*size*, *pos*, *rot=None*)
Bases: `ars.graphics.vtk_adapter.Body`, `ars.graphics.base.Box`

class Capsule (*length*, *radius*, *center*, *rot=None*, *resolution=20*)
Bases: `ars.graphics.vtk_adapter.Body`, `ars.graphics.base.Capsule`

class Cone (*height*, *radius*, *center*, *rot=None*, *resolution=20*)
Bases: `ars.graphics.vtk_adapter.Body`, `ars.graphics.base.Cone`

class Cylinder (*length*, *radius*, *center*, *rot=None*, *resolution=20*)
Bases: `ars.graphics.vtk_adapter.Body`, `ars.graphics.base.Cylinder`

class Engine (*title*, *pos=None*, *size=(1000, 600)*, *zoom=1.0*, *cam_position=(10, 8, 10)*, *background_color=(0.1, 0.1, 0.4)*, ***kwargs*)
Bases: `ars.graphics.base.Engine`

Graphics adapter to the Visualization Toolkit (VTK) library

add_object (*obj*)

finalize_window()
Finalize and delete `renderer`, `render_window` and `interactor`.

See also:

http://stackoverflow.com/questions/15639762/ and http://docs.python.org/2/reference/datamodel.html#object.__del__

remove_object(*obj*)

reset()

restart_window()

start_window(*on_idle_callback=None*, *on_reset_callback=None*, *on_key_press_callback=None*)

class Entity(**args*, ***kwargs*)
Bases: `object`

adapter
alias of *Engine*

class ScreenshotRecorder(*base_filename='screenshot_'*, *graphics_adapter=None*)
Bases: *ars.graphics.base.ScreenshotRecorder*

Based on an official example script, very simple: http://www.vtk.org/Wiki/VTK/Examples/Python/Screenshot

file_extension = 'png'

write(*index=1*, *time=None*)

Note: Image files format is PNG, and extension is `.png`.

class Sphere(*radius*, *center*, *rot=None*, *phi_resolution=20*, *theta_resolution=20*)
Bases: *ars.graphics.vtk_adapter.Body*, *ars.graphics.base.Sphere*

VTK: sphere (represented by polygons) of specified radius centered at the origin. The resolution (polygonal discretization) in both the latitude (phi) and longitude (theta) directions can be specified.

class Trimesh(*vertices*, *faces*, *pos*, *rot=None*)
Bases: *ars.graphics.vtk_adapter.Body*, *ars.graphics.base.Trimesh*

lib Package

lib Package External libs included in ARS

Subpackages

pydispatch Package

pydispatch Package Multi-consumer multi-producer dispatching mechanism

dispatcher Module Multiple-producer-multiple-consumer signal-dispatching

dispatcher is the core of the PyDispatcher system, providing the primary API and the core logic for the system.

Module attributes of note:

Any – Singleton used to signal either "Any Sender" or "Any Signal". See documentation of the _Any class.

Anonymous – Singleton used to signal "Anonymous Sender" See documentation of the _Anonymous class.

Internal attributes:

WEAKREF_TYPES – tuple of types/classes which represent weak references to receivers, and thus must be de- referenced on retrieval to retrieve the callable object

connections – { senderkey (id) : { signal : [receivers...]}}

senders – { senderkey (id) [weakref(sender) }] used for cleaning up sender references on sender deletion

sendersBack – { receiverkey (id) [[senderkey (id)...] }] used for cleaning up receiver references on receiver deletion, (considerably speeds up the cleanup process vs. the original code.)

connect (*receiver*, *signal=_Any*, *sender=_Any*, *weak=True*)
Connect receiver to sender for signal

receiver – a callable Python object which is to receive messages/signals/events. Receivers must be hashable objects.

if weak is True, then receiver must be weak-referencable (more precisely saferef.safeRef() must be able to create a reference to the receiver).

Receivers are fairly flexible in their specification, as the machinery in the robustApply module takes care of most of the details regarding figuring out appropriate subsets of the sent arguments to apply to a given receiver.

Note: if receiver is itself a weak reference (a callable), it will be de-referenced by the system's machinery, so *generally* weak references are not suitable as receivers, though some use might be found for the facility whereby a higher-level library passes in pre-weakrefed receiver references.

signal – the signal to which the receiver should respond

if Any, receiver will receive any signal from the indicated sender (which might also be Any, but is not necessarily Any).

Otherwise must be a hashable Python object other than None (DispatcherError raised on None).

sender – the sender to which the receiver should respond

if Any, receiver will receive the indicated signals from any sender.

if Anonymous, receiver will only receive indicated signals from send/sendExact which do not specify a sender, or specify Anonymous explicitly as the sender.

Otherwise can be any python object.

weak – whether to use weak references to the receiver By default, the module will attempt to use weak references to the receiver objects. If this parameter is false, then strong references will be used.

returns None, may raise DispatcherTypeError

disconnect (*receiver*, *signal=_Any*, *sender=_Any*, *weak=True*)
Disconnect receiver from sender for signal

receiver – the registered receiver to disconnect signal – the registered signal to disconnect sender – the registered sender to disconnect weak – the weakref state to disconnect

disconnect reverses the process of connect, the semantics for the individual elements are logically equivalent to a tuple of (receiver, signal, sender, weak) used as a key to be deleted from the internal routing tables. (The actual process is slightly more complex but the semantics are basically the same).

Note: Using disconnect is not required to cleanup routing when an object is deleted, the framework will remove routes for deleted objects automatically. It's only necessary to disconnect if you want to stop routing to a live object.

returns None, may raise DispatcherTypeError or DispatcherKeyError

`getAllReceivers`(*sender=_Any*, *signal=_Any*)
Get list of all receivers from global tables

This gets all receivers which should receive the given signal from sender, each receiver should be produced only once by the resulting generator

`getReceivers`(*sender=_Any*, *signal=_Any*)
Get list of receivers from global tables

This utility function allows you to retrieve the raw list of receivers from the connections table for the given sender and signal pair.

Note: there is no guarantee that this is the actual list stored in the connections table, so the value should be treated as a simple iterable/truth value rather than, for instance a list to which you might append new records.

Normally you would use liveReceivers(getReceivers(...)) to retrieve the actual receiver objects as an iterable object.

`liveReceivers`(*receivers*)
Filter sequence of receivers to get resolved, live receivers

This is a generator which will iterate over the passed sequence, checking for weak references and resolving them, then returning all live receivers.

`send`(*signal=_Any*, *sender=_Anonymous*, **arguments*, ***named*)
Send signal from sender to all connected receivers.

signal – (hashable) signal value, see connect for details

sender – the sender of the signal

if Any, only receivers registered for Any will receive the message.

if Anonymous, only receivers registered to receive messages from Anonymous or Any will receive the message

Otherwise can be any python object (normally one registered with a connect if you actually want something to occur).

arguments – positional arguments which will be passed to *all* receivers. Note that this may raise TypeErrors if the receivers do not allow the particular arguments. Note also that arguments are applied before named arguments, so they should be used with care.

named – named arguments which will be filtered according to the parameters of the receivers to only provide those acceptable to the receiver.

Return a list of tuple pairs [(receiver, response), ...]

if any receiver raises an error, the error propagates back through send, terminating the dispatch loop, so it is quite possible to not have all receivers called if a raises an error.

`sendExact` (*signal=_Any*, *sender=_Anonymous*, **arguments*, ***named*)
Send signal only to those receivers registered for exact message

sendExact allows for avoiding Any/Anonymous registered handlers, sending only to those receivers explicitly registered for a particular signal on a particular sender.

`errors` Module

Error types for dispatcher mechanism

exception `DispatcherError`
Bases: `exceptions.Exception`

Base class for all Dispatcher errors

exception `DispatcherKeyError`
Bases: `exceptions.KeyError`, *`ars.lib.pydispatch.errors.DispatcherError`*

Error raised when unknown (sender,signal) set specified

exception `DispatcherTypeError`
Bases: `exceptions.TypeError`, *`ars.lib.pydispatch.errors.DispatcherError`*

Error raised when inappropriate signal-type specified (None)

`robust` Module

Module implementing error-catching version of send (sendRobust)

`sendRobust` (*signal=_Any*, *sender=_Anonymous*, **arguments*, ***named*)
Send signal from sender to all connected receivers catching errors

signal – (hashable) signal value, see connect for details

sender – the sender of the signal

> if Any, only receivers registered for Any will receive the message.
>
> if Anonymous, only receivers registered to receive messages from Anonymous or Any will receive the message
>
> Otherwise can be any python object (normally one registered with a connect if you actually want something to occur).

arguments – positional arguments which will be passed to *all* receivers. Note that this may raise TypeErrors if the receivers do not allow the particular arguments. Note also that arguments are applied before named arguments, so they should be used with care.

named – named arguments which will be filtered according to the parameters of the receivers to only provide those acceptable to the receiver.

Return a list of tuple pairs [(receiver, response), ...]

if any receiver raises an error (specifically any subclass of Exception), the error instance is returned as the result for that receiver.

`robustapply` Module

Robust apply mechanism

Provides a function "call", which can sort out what arguments a given callable object can take, and subset the given arguments to match only those which are acceptable.

function (*receiver*)
Get function-like callable object for given receiver

returns (function_or_method, codeObject, fromMethod)

If fromMethod is true, then the callable already has its first argument bound

robustApply (*receiver*, **arguments*, ***named*)
Call receiver with arguments and an appropriate subset of named

saferef Module Refactored "safe reference" from dispatcher.py

class BoundMethodWeakref (*target*, *onDelete=None*)
Bases: `object`

'Safe' and reusable weak references to instance methods

BoundMethodWeakref objects provide a mechanism for referencing a bound method without requiring that the method object itself (which is normally a transient object) is kept alive. Instead, the BoundMethodWeakref object keeps weak references to both the object and the function which together define the instance method.

Attributes:

key – the identity key for the reference, calculated by the class's calculateKey method applied to the target instance method

deletionMethods – sequence of callable objects taking single argument, a reference to this object which will be called when *either* the target object or target function is garbage collected (i.e. when this object becomes invalid). These are specified as the onDelete parameters of safeRef calls.

weakSelf – weak reference to the target object

weakFunc – weak reference to the target function

Class Attributes:

_allInstances – class attribute pointing to all live BoundMethodWeakref objects indexed by the class's calculateKey(target) method applied to the target objects. This weak value dictionary is used to short-circuit creation so that multiple references to the same (object, function) pair produce the same BoundMethodWeakref instance.

Return a weak-reference-like instance for a bound method

target – the instance-method target for the weak reference, must have <im_self> and <im_func> attributes and be reconstructable via:

target.<im_func>.__get__(target.<im_self>)

which is true of built-in instance methods.

onDelete – optional callback which will be called when this weak reference ceases to be valid (i.e. either the object or the function is garbage collected). Should take a single argument, which will be passed a pointer to this object.

classmethod calculateKey (*target*)
Calculate the reference key for this reference

Currently this is a two-tuple of the id()'s of the target object and the target function respectively.

safeRef (*target*, *onDelete=None*)
Return a *safe* weak reference to a callable target

target – the object to be weakly referenced, if it's a bound method reference, will create a BoundMethodWeakref, otherwise creates a simple weakref.

onDelete – if provided, will have a hard reference stored to the callable to be called after the safe reference goes out of scope with the reference object, (either a weakref or a BoundMethodWeakref) as argument.

model Package

Subpackages

collision Package

`base` Module This module defines the basic functionality for collision, as well as the base classes that compose an abstract interface to the library developers choose to use.

Both *`Space`* and *`Geom`* (parent class of *`Ray`*, *`Trimesh`*, *`Box`*, *`Sphere`*, *`Plane`*, etc) wrap the corresponding "native" object that the adapted library uses, assigned to private attribute `_inner_object`. To access (not set) it, these classes have public property `inner_object`.

This module also contains the auxiliary classes *`RayContactData`* and *`NearCallbackArgs`*.

The following are common abbreviations present both in code and documentation:

- geom: geometry object
- trimesh: triangular mesh

class **`BasicShape`**

Bases: *`ars.model.collision.base.Geom`*

Abstract class from whom every solid object's shape derive

class **`Box`** (*space*, *size*)

Bases: *`ars.model.collision.base.BasicShape`*

Box shape, aligned along the X, Y and Z axii by default

class **`Capsule`** (*space*, *length*, *radius*)

Bases: *`ars.model.collision.base.BasicShape`*

Capsule shape, aligned along the Z-axis by default

class **`Cone`**

Bases: *`ars.model.collision.base.BasicShape`*

class **`ConstantHeightfieldTrimesh`** (*space*, *size_x*, *size_z*, *height*)

Bases: *`ars.model.collision.base.HeightfieldTrimesh`*

A trimesh that is a heightfield at constant level.

Note: More than anything, this geom is for demonstration purposes, because it could be easily replaced with a *`Plane`*.

Constructor.

Parameters

- **`space`** (*`Space`*) –
- **`size_x`** (*`positive int`*) – number of cells along the X axis
- **`size_z`** (*`positive int`*) – number of cells along the Z axis

- **height** (*float*) –

static **calc_vertices** (*size_x, size_z, height=0.0*)
Return the vertices of a horizontal grid of `size_x` by `size_z` cells at a certain `height`.

Parameters

- **size_x** (*positive int*) – number of cells along the X axis
- **size_z** (*positive int*) – number of cells along the Z axis
- **height** (*float*) –

```
>>> ConstantHeightfieldTrimesh.calc_vertices(2, 4)
[(0, 0.0, 0), (0, 0.0, 1), (0, 0.0, 2), ..., (1, 0.0, 3)]
```

class **ContactGroup**
Bases: `object`

Wrapper around a collection-like class storing contact data instances.

What these instances are (attributes, behavior) is up to the implementation of the adpater.

empty ()
Remove all the stored contact data instances.

inner_object

class **Cylinder** (*space, length, radius*)
Bases: *ars.model.collision.base.BasicShape*

Cylinder shape, aligned along the Z-axis by default

class **Engine**
Bases: `object`

Collision engine abstract base class.

classmethod **are_geoms_connected** (*geom1, geom2*)
Return whether `geom1`'s body is connected to `geom2`'s body.

The *connection* is checked as whether geoms bodies are connected through a joint or not.

Parameters

- **geom1** (type of *Geom.inner_object*) –
- **geom2** (type of *Geom.inner_object*) –

Returns True if geoms' bodies are connected; False otherwise

Return type bool

classmethod **calc_collision** (*geom1, geom2*)
Calculate information of the collision between these geoms.

Check if `geom1` and `geom2` actually collide and create a list of contact data objects if they do.

Parameters

- **geom1** (type of *Geom.inner_object*) –
- **geom2** (type of *Geom.inner_object*) –

Returns contacts information

Return type list of contact data objects

classmethod **`is_ray`**(*geom*)

Return whether `geom` is a ray-like object or not.

Parameters **`geom`** (type of *`Geom.inner_object`*) –

Returns True if `geom` is an instance of the class representing a ray in the adapted library

Return type bool

classmethod **`near_callback`**(*args*, *geom1*, *geom2*)

Handle possible collision between `geom1` and `geom2`.

The responsible for determining if there is an actual collision is *`calc_collision()`*, which will return a list of contact data objects.

That information is passed to either *`process_collision_contacts()`* or *`process_ray_collision_contacts()`*, depending on whether `geom1` or `geom2` is a ray or not. It's an unhandled case that both geoms were rays.

This function is usually the callback function for *`Space.collide()`*, although it will probably be handed over to the inner object of a *`Space`* subclass.

Parameters

- **`args`** (*`NearCallbackArgs`*) – data structure wrapping the objects necessary to process the collision
- **`geom1`** (type of *`Geom.inner_object`*) –
- **`geom2`** (type of *`Geom.inner_object`*) –

classmethod **`process_collision_contacts`**(*args*, *geom1*, *geom2*, *contacts*)

Process `contacts` of a collision between `geom1` and `geom2`.

This method should create movement constraints for the bodies attached to the geoms. This is necessary for the simulation to prevent bodies' volumes from penetrating each other, making them really collide (i.e. exert mutually opposing forces).

> **Warning:** Neither `geom1` nor `geom2` can be rays. If one of them is, use method *`process_ray_collision_contacts()`*.

Parameters

- **`args`** (*`NearCallbackArgs`*) –
- **`geom1`** (type of *`Geom.inner_object`*) –
- **`geom2`** (type of *`Geom.inner_object`*) –
- **`contacts`** (*`list of contact data objects`*) – collision data returned by *`calc_collision()`*

classmethod **`process_ray_collision_contacts`**(*ray*, *other_geom*, *contacts*)

Process special case of collision between a ray and a regular geom.

See also:

For regular geoms collision, see *`process_collision_contacts()`*.

Since rays have no attached body, they can't "really" collide with other geoms. However, they do intersect, which is of interest to non-physical aspects of the simulation. A common use case is that of laser distance sensors.

> **Warning:** Collision between two rays is a singularity and should never happen.

Parameters

- **ray** (type of `Ray.inner_object`) –
- **other_geom** (type of *`Geom.inner_object`*) –
- **contacts** (*list of contact data objects*) – collision data returned by *`calc_collision()`*

class Geom

Bases: `object`

Geometry object encapsulation.

This class wraps the corresponding "native" object the adapted-to library (e.g. ODE) uses, assigned to `_inner_object`.

Subclasses must implement these methods:

- `__init__()`
- *`attach_body()`*
- *`get_position()`*, *`set_position()`*
- *`get_rotation()`*, *`set_rotation()`*

attach_body(*body*)

get_attached_body()

get_position()

Get the position of the geom.

Returns position

Return type 3-sequence of floats

get_rotation()

Get the orientation of the geom.

Returns rotation matrix

Return type 9-sequence of floats

inner_object

set_position(*pos*)

Set the position of the geom.

Parameters pos (*3-sequence of floats*) – position

set_rotation(*rot*)

Set the orientation of the geom.

Parameters rot (*9-sequence of floats*) – rotation matrix

class HeightfieldTrimesh(*space*, *size_x*, *size_z*, *vertices*)

Bases: *`ars.model.collision.base.Trimesh`*

static calc_faces(*size_x*, *size_z*)

Return the faces for a horizontal grid of `size_x` by `size_z` cells.

Faces are triangular, so each is composed by 3 vertices. Consequently, each returned face is a length-3 sequence of the vertex indices.

Parameters

- **size_x** (*positive int*) – number of cells along the X axis
- **size_z** (*positive int*) – number of cells along the Z axis

Returns faces for a heightfield trimesh based in a horizontal grid of `size_x` by `size_z` cells

Return type list of 3-tuple of ints

```
>>> HeightfieldTrimesh.calc_faces(2, 4)
[(0, 1, 4), (1, 5, 4), (1, 6, 5), (1, 2, 6), (2, 3, 6), (3, 7, 6)]
```

class NearCallbackArgs(*world=None*, *contact_group=None*, *ignore_connected=True*)
Bases: `object`

Data structure to save the args passed to *Engine.near_callback()*.

All attributes are read-only (set at initialization).

Constructor.

Parameters

- **world** (*physics.base.World*) –
- **contact_group** (*ContactGroup*) –
- **ignore_connected** (*bool*) – whether to ignore collisions of geoms whose bodies are connected, or not

contact_group

ignore_connected

world

class Plane(*space*, *normal*, *dist*)
Bases: *ars.model.collision.base.BasicShape*

Plane, different from a box

class Ray(*space*, *length*)
Bases: *ars.model.collision.base.Geom*

Ray aligned along the Z-axis by default. "A ray is different from all the other geom classes in that it does not represent a solid object. It is an infinitely thin line that starts from the geom's position and extends in the direction of the geom's local Z-axis." (ODE Wiki Manual)

clear_closer_contact()

clear_contacts()

clear_last_contact()

get_closer_contact()
Return the contact object corresponding to the collision closest to the ray's origin.

It may or may not be the same object returned by *get_last_contact*.

get_last_contact()
Return the contact object corresponding to the last collision of the ray with another geom. Note than in each simulation step, several collisions may occur, one for each intersection geom (in ODE). The object returned may or may not be the same returned by *get_closer_contact*.

get_length()

set_last_contact(*last_contact*)
Set the contact data of ray's last collision. It also checks if *last_contact* is closer than the previously existing one. The result can be obtained with the *get_closer_contact* method.

set_length(*length*)

class **RayContactData**(*ray=None*, *shape=None*, *pos=None*, *normal=None*, *depth=None*)
Bases: `object`

Data structure to save the contact information of a collision between `ray` and `shape`.

All attributes are read-only (set at initialization).

Constructor.

Parameters

- **ray** (the type of `Ray` subclass' `inner_object`) –
- **shape** (the type of `Geom` subclass' `inner_object`) –
- **pos** (*3-tuple of floats*) – point at which the ray intersects the surface of the other shape/geom
- **normal** (*3-tuple of floats*) – vector normal to the surface of the other geom at the contact point
- **depth** (*float*) – distance from the origin of the ray to the contact point

depth

normal

position

ray

shape

class **Space**
Bases: `object`

Collision space abstract base class.

This class wraps the corresponding "native" object the adapted-to library (e.g. ODE) uses, assigned to `_inner_object`.

Subclasses must implement these methods:

- `__init__()`
- `collide()`

collide(*args*, *callback*)

inner_object

class **Sphere**(*space*, *radius*)
Bases: `ars.model.collision.base.BasicShape`

Spherical shape

class **Trimesh**(*space*, *vertices*, *faces*)
Bases: `ars.model.collision.base.Geom`

A triangular mesh i.e. a surface composed of triangular faces.

Note: Note that a trimesh need not be closed. For example, it could be used to model the ground surface.

Its geometry is defined by two attributes: `vertices` and `faces`, both list of 3-tuple numbers. However, each tuple in `vertices` designates a 3D point in space whereas each tuple in `faces` is a group of indices referencing points in `vertices`.

Warning: The order of vertices indices for each face **does** matter.

Example:

```
vertices = [(0, 0.0, 0), (0, 0.0, 1), (0, 0.0, 2), (0, 0.0, 3),
            (1, 0.0, 0), (1, 0.0, 1), (1, 0.0, 2), (1, 0.0, 3)]

faces = [(0, 1, 4), (1, 5, 4), (1, 6, 5),
         (1, 2, 6), (2, 3, 6), (3, 7, 6)]
```

The, the first face is defined by points: `(0, 0.0, 0), (0, 0.0, 1), (1, 0.0, 0)`. With that order, the normal to the face is `(0, 1.0, 0)` i.e. the Y axis. The rationale to determining the *inwards* and *outwards* directions follows the well-known "right hand rule".

static **`swap_faces_indices`**(*faces*)

Faces had to change their indices to work with ODE. With the initial get_faces, the normal to the triangle defined by the 3 vertices pointed (following the right-hand rule) downwards. Swapping the third with the first index, now the triangle normal pointed upwards.

`ode_adapter` **Module** Classes and functions to interface with the collision library included in ODE.

class **`BasicShape`**

Bases: *`ars.model.collision.ode_adapter.Geom`*

class **`Box`**(*space*, *size*)

Bases: *`ars.model.collision.ode_adapter.BasicShape`*, *`ars.model.collision.base.Box`*

Box shape, aligned along the X, Y and Z axii by default

class **`Capsule`**(*space*, *length*, *radius*)

Bases: *`ars.model.collision.ode_adapter.BasicShape`*, *`ars.model.collision.base.Capsule`*

Capsule shape, aligned along the Z-axis by default

class **`ContactGroup`**

Bases: *`ars.model.collision.base.ContactGroup`*

`empty`()

class **`Cylinder`**(*space*, *length*, *radius*)

Bases: *`ars.model.collision.ode_adapter.BasicShape`*, *`ars.model.collision.base.Cylinder`*

Cylinder shape, aligned along the Z-axis by default

class **`Engine`**

Bases: *`ars.model.collision.base.Engine`*

Adapter to the ODE collision engine.

classmethod **`are_geoms_connected`**(*geom1*, *geom2*)

(see parent method)

Parameters

- **geom1** (`ode.GeomObject`) –
- **geom2** (`ode.GeomObject`) –

classmethod **`calc_collision`**(*geom1*, *geom2*)

Calculate information of the collision between these geoms.

Check if `geom1` and `geom2` actually collide and create a list of contact data objects if they do.

Parameters

- **geom1** (`ode.GeomObject`) –
- **geom2** (`ode.GeomObject`) –

Returns contacts information

Return type list of `ode.Contact`

classmethod **`is_ray`**(*geom*)

Return whether `geom` is a `ode.GeomRay` object or not.

Parameters **geom** (`ode.GeomObject`) –

Returns True if `geom` is an instance of `ode.GeomRay`

Return type bool

classmethod **`process_collision_contacts`**(*args*, *geom1*, *geom2*, *contacts*)

(see parent *`base.Engine.process_collision_contacts()`*)

Parameters

- **geom1** (`ode.GeomObject`) –
- **geom2** (`ode.GeomObject`) –
- **contacts** (list of `ode.Contact`) –

classmethod **`process_ray_collision_contacts`**(*ray*, *other_geom*, *contacts*)

(see parent *`base.Engine.process_ray_collision_contacts()`*)

Parameters

- **ray** (`ode.GeomRay`) – monkey-patched object whose attribute `outer_object` references its wrapper (a *`base.Ray`* object)
- **other_geom** (`ode.GeomObject`) –
- **contacts** (list of `ode.Contact`) –

class **`Geom`**

Bases: *`ars.model.collision.base.Geom`*

Abstract class, sort of equivalent to `ode.GeomObject`.

`attach_body`(*body*)

`get_position`()

Get the position of the geom.

Returns position

Return type 3-sequence of floats

`get_rotation`()

Get the orientation of the geom.

Returns rotation matrix

Return type 9-sequence of floats

set_position (*pos*)
Set the position of the geom.

Parameters **pos** (*3-sequence of floats*) – position

set_rotation (*rot*)
Set the orientation of the geom.

Parameters **rot** (*9-sequence of floats*) – rotation matrix

class Plane (*space, normal, dist*)
Bases: *ars.model.collision.ode_adapter.BasicShape*, *ars.model.collision.base.Plane*

Plane, different from a box

class Ray (*space, length*)
Bases: *ars.model.collision.ode_adapter.Geom*, *ars.model.collision.base.Ray*

get_length ()

set_length (*length*)

class Space
Bases: *ars.model.collision.base.Space*

Adapter to `ode.SimpleSpace`.

collide (*args, callback=None*)
Call `callback` with `args` for all potentially intersecting geom pairs.

Function `callback` must accept 3 arguments: `args, geom1, geom2`.

Parameters

- **args** (`NearCallbackArgs`) – data object passed to `callback` in each call
- **callback** (*function or None*) – a function with signature `args, geom1, geom2`

class Sphere (*space, radius*)
Bases: *ars.model.collision.ode_adapter.BasicShape*, *ars.model.collision.base.Sphere*

Spherical shape

class Trimesh (*space, vertices, faces*)
Bases: *ars.model.collision.ode_adapter.Geom*, *ars.model.collision.base.Trimesh*

ode_objects_factories Module ODE objects factories i.e. functions that create ODE objects.

create_ode_box (*space, size*)
Create an ODE box geom.

Parameters

- **space** (`ode.Space`) –
- **size** (*3-sequence of floats*) –

Returns ODE box geom

Return type `ode.GeomBox`

create_ode_capsule(*space*, *length*, *radius*)
Create an ODE capsule geom.

Note: In `GeomCCylinder` (same as `GeomCapsule`) *CCylinder* means Capped Cylinder.

Warning: ODE's constructor parameter order is different: `radius` first and then `length`.

Parameters

- **space** (`ode.Space`) –
- **length** (*float*) – of the cylindrical section i.e. caps are not included
- **radius** (*float*) –

Returns ODE capsule geom

Return type `ode.GeomCCylinder`

create_ode_cylinder(*space*, *length*, *radius*)
Create an ODE cylinder geom.

Warning: ODE's constructor parameter order is different: `radius` first and then `length`.

Parameters

- **space** (`ode.Space`) –
- **length** (*float*) –
- **radius** (*float*) –

Returns ODE cylinder geom

Return type `ode.GeomCylinder`

create_ode_hash_space()
Create a more sophisticated ODE geoms container (i.e. "space").

Note: `ode.HashSpace()` equals `ode.Space(space_type=1)`.

Returns ODE hash space

Return type `ode.HashSpace`

create_ode_joint_group()
Create an ODE joint group.

Returns ODE joint group

Return type `ode.JointGroup`

create_ode_plane(*space*, *normal*, *dist*)
Create an ODE plane (infinite) geom.

The plane equation is

$$n_0 \cdot x + n_1 \cdot y + n_2 \cdot z = dist$$

where `normal = (n0, n1, n2)`.

Warning: This object can't be attached to a body.

Parameters

- **space** (`ode.Space`) –
- **normal** (*3-sequence of floats*) – vector normal to the plane
- **dist** (*float*) – constant of the plane equation

Returns ODE plane geom

Return type `ode.GeomPlane`

create_ode_ray(*space*, *length*)

Create an ODE ray geom.

Parameters

- **space** (`ode.Space`) –
- **length** (*float*) –

Returns ODE ray geom

Return type `ode.GeomRay`

create_ode_simple_space()

Create an ODE geoms container (i.e. "space") of the simplest type.

Note: `ode.SimpleSpace()` equals `ode.Space(space_type=0)`.

Returns ODE simple space

Return type `ode.SimpleSpace`

create_ode_sphere(*space*, *radius*)

Create an ODE sphere geom.

Parameters

- **space** (`ode.Space`) –
- **radius** (*float*) –

Returns ODE sphere geom

Return type `ode.GeomSphere`

create_ode_trimesh(*space*, *vertices*, *faces*)

Create an ODE trimesh geom.

Parameters

- **space** (`ode.Space`) –
- **vertices** (*sequence of 3-sequences of floats*) –
- **faces** (*sequence of 3-sequences of ints*) –

Returns ODE trimesh geom

Return type `ode.GeomTriMesh`

signals Module This module contains string values defining different signals related to the `ars.model.collision` package.

contrib Package

contrib Package Container for contributed modules or packages that could be helpful for other ARS users (other than its own developers, which are encouraged to improve their contributions and spread the word about them).

Later on, the most popular and stable parts of the code should be integrated in the official package.

geometry Package

geometry Package

physics Package

base Module

class **Body** (*mass=None*, *density=None*, *pos=None*, *rot=None*, **args*, ***kwargs*)

Bases: `object`

attach_geom (*geom*)

calc_potential_energy (*gravity*)

Calculate the potential energy of the body due to its position (x) and the gravitational acceleration (g).

$$E_p = m \cdot g \cdot h = -m \cdot g^\top x$$

Parameters **gravity** (`tuple of 3 floats`) – gravitational acceleration vector

Returns potential energy

Return type float

calc_rotation_kinetic_energy ()

Calculate the kinetic energy of the body due to rotational movement.

$$E_{kr} = \frac{1}{2} I \omega^2 = \frac{1}{2} \omega^\top \mathbf{I} \omega$$

Returns kinetic energy

Return type float

calc_translation_kinetic_energy ()

Calculate the kinetic energy of the body due to translational movement.

$$E_{kt} = \frac{1}{2} m v^2 = \frac{1}{2} m \cdot v^\top v$$

Returns kinetic energy

Return type float

get_angular_velocity ()

get_attached_geom ()

get_center_of_gravity ()

get_inertia_tensor()

get_linear_velocity()

get_mass()

get_position()
Get the position of the body.

Returns position

Return type 3-sequence of floats

get_rotation()
Get the orientation of the body.

Returns rotation matrix

Return type 9-sequence of floats

get_saved_velocities()
Return last saved velocities (linear and angular).

inner_object

save_velocities()
Retrieve the actual velocities (linear and angular) of the body and save them.

set_position(*pos*)
Set the position of the body.

Sends `signals.BODY_PRE_SET_POSITION` and `signals.BODY_POST_SET_POSITION`.

Parameters **pos** (*3-sequence of floats*) – position

set_rotation(*rot*)
Set the orientation of the body.

Sends `signals.BODY_PRE_SET_ROTATION` and `signals.BODY_POST_SET_ROTATION`.

Parameters **rot** (*9-sequence of floats*) – rotation matrix

class **Box**(*size*, **args*, ***kwargs*)
Bases: `ars.model.physics.base.Body`

size

class **Capsule**(*length*, *radius*, **args*, ***kwargs*)
Bases: `ars.model.physics.base.Body`

length

radius

class **Cone**(*height*, *radius*, **args*, ***kwargs*)
Bases: `ars.model.physics.base.Body`

height

radius

class **Cylinder**(*length*, *radius*, **args*, ***kwargs*)
Bases: `ars.model.physics.base.Body`

length

radius

class **Engine**
Bases: object

world_class = None

class **Sphere** (*radius*, **args*, ***kwargs*)
Bases: *ars.model.physics.base.Body*

radius

class **World** (*gravity*, **args*, ***kwargs*)
Bases: object

gravity

inner_object

step (*time_step*)
Subclasses implementing this method must send the corresponding signals, defined in *ars.model.physics.signals*.

ode_adapter Module Classes and functions to interface the ODE physics engine with the API defined in physics.

class **Body** (*world*, *space*, *mass=None*, *density=None*, **args*, ***kwargs*)
Bases: object

Abstract class, sort of equivalent to ode.Body.

Constructor.

Parameters

- **world** (*base.World*) –
- **space** (*collision.base.Space*) –
- **mass** (*float or None*) –
- **density** (*float or None*) –

get_angular_velocity ()

get_center_of_gravity ()

get_inertia_tensor ()

get_linear_velocity ()

get_mass ()

get_position ()
Get the position of the body.

Returns position

Return type 3-sequence of floats

get_rotation ()
Get the orientation of the body.

Returns rotation matrix

Return type 9-sequence of floats

set_position (*pos*)
Set the position of the body.

Sends `signals.BODY_PRE_SET_POSITION` and `signals.BODY_POST_SET_POSITION`.

Parameters **pos** (*3-sequence of floats*) – position

set_rotation (*rot*)
Set the orientation of the body.

Sends `signals.BODY_PRE_SET_ROTATION` and `signals.BODY_POST_SET_ROTATION`.

Parameters **rot** (*9-sequence of floats*) – rotation matrix

class Box (*world*, *space*, *size*, *mass=None*, *density=None*)
Bases: *ars.model.physics.ode_adapter.Body*, *ars.model.physics.base.Box*

class Capsule (*world*, *space*, *length*, *radius*, *mass=None*, *density=None*)
Bases: *ars.model.physics.ode_adapter.Body*, *ars.model.physics.base.Capsule*

create capsule body (aligned along the z-axis so that it matches the Geom created below, which is aligned along the Z-axis by default)

class Cylinder (*world*, *space*, *length*, *radius*, *mass=None*, *density=None*)
Bases: *ars.model.physics.ode_adapter.Body*, *ars.model.physics.base.Cylinder*

create cylinder body (aligned along the z-axis so that it matches the Geom created below, which is aligned along the Z-axis by default)

class Engine
Bases: *ars.model.physics.base.Engine*

Adapter to the ODE physics engine

world_class
alias of *World*

class Sphere (*world*, *space*, *radius*, *mass=None*, *density=None*)
Bases: *ars.model.physics.ode_adapter.Body*, *ars.model.physics.base.Sphere*

class World (*gravity=(0.0, -9.81, 0.0)*, *ERP=0.2*, *CFM=1e-10*, **args*, ***kwargs*)
Bases: *ars.model.physics.base.World*

Adapter to `ode.World`.

See also:

Read abouth ERP and CFM in ODE's wiki http://ode-wiki.org/wiki/index.php?title=Manual:_Concepts

Constructor.

Parameters

- **gravity** (*3-sequence of floats*) – gravity acceleration vector
- **ERP** (*float*) – Error Reduction Parameter
- **CFM** (*float*) – Constraint Force Mixing

gravity

step (*time_step*)

ode_objects_factories Module ODE objects factories i.e. functions that create ODE objects.

create_ode_box(*world*, *size*, *mass=None*, *density=None*)
Create an ODE body with box-like mass parameters.

Parameters

- **world** (ode.World) –
- **size** (*3-sequence of floats*) –
- **mass** (*float or None*) –
- **density** (*float or None*) –

Returns box body

Return type ode.Body

create_ode_capsule(*world*, *length*, *radius*, *mass=None*, *density=None*)
Create an ODE body with capsule-like mass parameters.

Parameters

- **world** (ode.World) –
- **length** (*float*) –
- **radius** (*float*) –
- **mass** (*float or None*) –
- **density** (*float or None*) –

Returns capsule body

Return type ode.Body

create_ode_cylinder(*world*, *length*, *radius*, *mass=None*, *density=None*)
Create an ODE body with cylinder-like mass parameters.

Parameters

- **world** (ode.World) –
- **length** (*float*) –
- **radius** (*float*) –
- **mass** (*float or None*) –
- **density** (*float or None*) –

Returns cylinder body

Return type ode.Body

create_ode_sphere(*world*, *radius*, *mass=None*, *density=None*)
Create an ODE body with sphere-like mass parameters.

Parameters

- **world** (ode.World) –
- **radius** (*float*) –
- **mass** (*float or None*) –
- **density** (*float or None*) –

Returns sphere body

Return type `ode.Body`

`create_ode_world`(*gravity=(0.0, -9.81, 0.0)*, *ERP=0.8*, *CFM=1e-10*)

Create an ODE world object.

Parameters

- **gravity** (*3-sequence of floats*) – gravity acceleration vector
- **ERP** (*float*) – Error Reduction Parameter
- **CFM** (*float*) – Constraint Force Mixing

Returns world

Return type `ode.World`

`signals` Module This module contains string values defining different signals related to the `ars.model.physics` package.

robot Package

`joints` Module Module of all the classes related to physical joints. These are objects that link 2 bodies together.

There are two base abstract classes for all joints: *Joint* and *ActuatedJoint*. They are not coupled (at all) with ODE or any other physics or collision library/engine.

The classes that implement at least one of those interfaces are these:

- *Fixed*
- *Rotary*
- *Universal*
- *BallSocket*
- *Slider*

There is also an auxiliary class: *JointFeedback*.

class `ActuatedJoint`(*world*, *inner_joint*, *body1=None*, *body2=None*)

Bases: *ars.model.robot.joints.Joint*

A joint with an actuator that can exert force and/or torque to connected bodies.

This is an abstract class.

Constructor.

Parameters

- **world** (`physics.base.World`) –
- **inner_joint** (`ode.Joint`) –
- **body1** (`physics.base.Body`) –
- **body2** (`physics.base.Body`) –

class `BallSocket`(*world*, *body1*, *body2*, *anchor*)

Bases: *ars.model.robot.joints.Joint*

Constructor.

Parameters

- **world** (physics.base.World) –
- **body1** (physics.base.Body) –
- **body2** (physics.base.Body) –
- **anchor** (*3-tuple of floats*) – joint anchor point

class Fixed (*world*, *body1*, *body2*)

Bases: *ars.model.robot.joints.Joint*

Constructor.

Parameters

- **world** (physics.base.World) –
- **body1** (physics.base.Body) –
- **body2** (physics.base.Body) –

class Joint (*world*, *inner_joint*, *body1=None*, *body2=None*)

Bases: object

Entity that links 2 bodies together, enforcing one or more movement constraints.

This is an abstract class.

Constructor.

Parameters

- **world** (physics.base.World) –
- **inner_joint** (ode.Joint) –
- **body1** (physics.base.Body) –
- **body2** (physics.base.Body) –

body1

body2

class JointFeedback (*body1*, *body2*, *force1=None*, *force2=None*, *torque1=None*, *torque2=None*)

Bases: object

Data structure to hold the forces and torques resulting from the interaction of 2 bodies through a joint.

All attributes are private. The results (*force1*, *force2*, *torque1*, *torque2*) are all length-3 tuples of floats.

Constructor.

Parameters

- **body1** (physics.base.Body) –
- **body2** (physics.base.Body) –
- **force1** (*3-tuple of floats*) –
- **force2** (*3-tuple of floats*) –
- **torque1** (*3-tuple of floats*) –
- **torque2** (*3-tuple of floats*) –

body1

body2

force1

force2

torque1

torque2

class **Rotary** (*world*, *body1*, *body2*, *anchor*, *axis*)

Bases: `ars.model.robot.joints.ActuatedJoint`

Constructor.

Parameters

- **world** (physics.base.World) –
- **body1** (physics.base.Body) –
- **body2** (physics.base.Body) –
- **anchor** (*3-tuple of floats*) – joint anchor point
- **axis** (*3-tuple of floats*) – rotation axis

add_torque (*torque*)

Apply torque about the rotation axis.

Parameters **torque** (*float*) – magnitude

angle

Return the angle between the two bodies.

The zero angle is determined by the position of the bodies when joint's anchor was set.

Returns value ranging `-pi` and `+pi`

Return type float

angle_rate

Return the rate of change of the angle between the two bodies.

Returns angle rate

Return type float

set_speed (*speed*, *max_force=None*)

Set rotation speed to `speed`.

The joint will set that speed by applying a force up to `max_force`, so it is not guaranteed that `speed` will be reached.

Parameters

- **speed** (*float*) – speed to set
- **max_force** (*float or None*) – if not None, the maximum force the joint can apply when trying to set the rotation speed

class **Slider** (*world*, *body1*, *body2*, *axis*)

Bases: `ars.model.robot.joints.ActuatedJoint`

Joint with one DOF that constrains two objects to line up along an axis.

It is different from a Piston joint (which has two DOF) in that the Slider does not allow rotation.

Constructor.

Parameters

- **world** (physics.base.World) –
- **body1** (physics.base.Body) –
- **body2** (physics.base.Body) –
- **axis** (*3-tuple of floats*) – rotation axis

add_force (*force*)
Apply a force of magnitude `force` along the joint's axis.

position
Return position of the joint with respect to its initial position.

The zero position is established when the joint's axis is set.

Return type float

position_rate
Return position's time derivative, i.e. "speed".

Return type float

class Universal (*world*, *body1*, *body2*, *anchor*, *axis1*, *axis2*)
Bases: *ars.model.robot.joints.Joint*

Constructor.

Parameters

- **world** (physics.base.World) –
- **body1** (physics.base.Body) –
- **body2** (physics.base.Body) –
- **anchor** (*3-tuple of floats*) – joint anchor point
- **axis1** (*3-tuple of floats*) – first universal axis
- **axis2** (*3-tuple of floats*) – second universal axis

sensors Module Module of all the classes related to sensors.

There are base classes for sensors whose source is a body, joint or simulation. It also considers those which read information automatically by subscribing to certain signals.

Some abstract classes are:

- *BaseSourceSensor*
- *BaseSignalSensor*
- *BodySensor*
- *JointSensor*
- *SimulationSensor*

Some practical sensors are:

- *RotaryJointSensor*, *JointTorque*

- `Laser`
- `GPS`, `Velometer`, `Accelerometer`, `Inclinometer`
- `KineticEnergy`, `PotentialEnergy`, `TotalEnergy`, `SystemTotalEnergy`

It also contains the auxiliary classes `SensorData` and `SensorDataQueue`.

class **Accelerometer**(*body*, *time_step*)
Bases: `ars.model.robot.sensors.BodySensor`

Calculate and retrieve a body's linear and angular acceleration.

> **Warning:** The provided *time_step* is used to calculate the acceleration based on the velocity measured at two instants in time. If subsequent calls to *on_change* are separated by a simulation time period different to the given *time_step*, the results will be invalid.

class **ActuatedJointSensor**(*joint*)
Bases: `ars.model.robot.sensors.JointSensor`

Sensor whose source of data is an `ActuatedJoint` joint.

class **BaseSignalSensor**(*sender=_Any*, *autotime=False*)
Bases: `object`

Base class for sensors that handle signals with `on_send()`.

Constructor.

Parameters

- **sender** – object that will send the signal; if it is `any_sender`, subscription will be to any object
- **autotime** – if True and `_get_time()` is not overriden, every measurement's time will set to the computer time in that instant

any_sender = _Any

on_send(*sender*, **args*, ***kwargs*)
Handle signal sent/fired by `sender` through the dispatcher.

Takes care of building a data object, set time to it and save it in the `data_queue`.

Parameters

- **sender** – signal sender
- **args** – signal arguments
- **kwargs** – signal keyword arguments

sender
Return the sender of the signal to which the sensor listens.

class **BaseSourceSensor**(*source*)
Bases: `object`

Abstract base class for all sensors.

Sensor data is stored in a queue (`data_queue`), and it is usually retrieved after the simulation ends but can be accessed at any time:

```
measurement = sensor.data_queue.pull()
```

> **Warning:** Beware that `ars.utils.containers.Queue.pull()` returns the first element of the queue and **removes** it.

get_measurement()
Return a measurement of the sensor packed in a data structure.

on_change(*time=None*)
Build a `SensorData` object and stores it in the `data_queue`.

Parameters **time** (*number or None*) – if None, current (computer's) time is used

source

class BodySensor(*body*)
Bases: `ars.model.robot.sensors.BaseSourceSensor`

Abstract base class for sensors whose source of data is a body.

body

class GPS(*body*)
Bases: `ars.model.robot.sensors.BodySensor`

Retrieve a body's XYZ position.

class Inclinometer(*body*)
Bases: `ars.model.robot.sensors.BodySensor`

Retrieve a body's *pitch* and *roll*.

class JointForce(*sim_joint*, *sim*)
Bases: `ars.model.robot.sensors.SingleSignalSensor`

Sensor measuring force 'added' to a joint.

signal = 'robot joint post add force'

class JointPower(*sim_joint*, *sim*)
Bases: `ars.model.robot.sensors.MultipleSignalsSensor`

Sensor measuring power applied by a joint (due to force and torque).

signals = ['robot joint post add torque', 'robot joint post add force']

class JointSensor(*joint*)
Bases: `ars.model.robot.sensors.BaseSourceSensor`

Abstract base class for sensors whose source of data is a joint.

joint

class JointTorque(*sim_joint*, *sim*)
Bases: `ars.model.robot.sensors.SingleSignalSensor`

Sensor measuring torque added to a joint.

signal = 'robot joint post add torque'

class KineticEnergy(*body*)
Bases: `ars.model.robot.sensors.BodySensor`

Retrieve a body's kinetic energy, both due to translation and rotation.

$$E_t = \frac{1}{2}mv^2 = \frac{1}{2}m \cdot v^\top v$$
$$E_r = \frac{1}{2}I\omega^2 = \frac{1}{2}\omega^\top \mathbf{I}\omega$$

class **`Laser`**(*space*, *max_distance=10.0*)

Bases: `ars.model.robot.sensors.BaseSourceSensor`

Laser scanner.

`get_ray`()

`on_change`(*time=1488286418.410617*)

`set_position`(*pos*)

Set position of the ray.

Useful mainly when it is not attached to a body.

Parameters **`pos`** (*`3-sequence of floats`*) – position

`set_rotation`(*rot*)

Set rotation of the ray.

Useful mainly when it is not attached to a body.

Parameters **`rot`** (*`9-sequence of floats`*) – rotation matrix

class **`MultipleSignalsSensor`**(*signals*, **args*, ***kwargs*)

Bases: `ars.model.robot.sensors.BaseSignalSensor`

Abstract base class for sensors subscribed to multiple signals.

Constructor.

Parameters **`signals`** (*`iterable`*) – signals to subscribe to

class **`PotentialEnergy`**(*body*, *gravity*)

Bases: `ars.model.robot.sensors.BodySensor`

Retrieve a body's potential energy.

Calculated based on the current position (x) and world's gravitational acceleration (g).

$$E_p = m \cdot g \cdot h = -m \cdot g^\top x$$

class **`RotaryJointSensor`**(*joint*)

Bases: `ars.model.robot.sensors.ActuatedJointSensor`

Sensor measuring the angle (and its rate) of a rotary joint.

class **`SensorData`**(**args*, ***kwargs*)

Bases: `object`

Data structure to pack a sensor measurement's information.

`get_arg`(*index*)

`get_args`()

`get_kwarg`(*key*)

`get_kwargs`()

get_time()

set_time(*time*)

class **SensorDataQueue**
Bases: *ars.utils.containers.Queue*

Queue-like container for sensor measurements.

class **SimulationSensor**(*sim*)
Bases: *ars.model.robot.sensors.BaseSourceSensor*

Abstract base class for sensors whose source of data is a simulation.

Constructor.

Parameters **sim** (*ars.model.simulator.Simulation*) – simulation

sim
Return the simulation object.

Returns simulation

Return type *ars.model.simulator.Simulation*

class **SingleSignalSensor**(*signal*, **args*, ***kwargs*)
Bases: *ars.model.robot.sensors.BaseSignalSensor*

Abstract base class for sensors subscribed to one signal.

Constructor.

Parameters **signal** – signal to subscribe to

class **SystemTotalEnergy**(*sim*, *disaggregate=False*)
Bases: *ars.model.robot.sensors.SimulationSensor*

Retrieve a system's total potential and kinetic energy.

It considers all bodies in the simulation. The kinetic energy accounts for translation and rotation.

$$E_p = m \cdot g \cdot h = -m \cdot g^\top x$$
$$E_k = \frac{1}{2} m \cdot v^\top v + \frac{1}{2} \omega^\top \mathbf{I} \omega$$

class **TotalEnergy**(*body*, *gravity*, *disaggregate=False*)
Bases: *ars.model.robot.sensors.BodySensor*

Retrieve a body's potential and kinetic energy.

The kinetic energy accounts for translation and rotation.

$$E_p = m \cdot g \cdot h = -m \cdot g^\top x$$
$$E_k = \frac{1}{2} m \cdot v^\top v + \frac{1}{2} \omega^\top \mathbf{I} \omega$$

class **Velometer**(*body*)
Bases: *ars.model.robot.sensors.BodySensor*

Calculate and retrieve a body's linear and angular velocity.

signals Module This module contains string values defining different signals related to the `ars.model.robot` package.

simulator Package

`simulator` Package

class **`SimulatedBody`**(*name*, *body=None*, *actor=None*, *geom=None*)

Bases: `ars.model.simulator.SimulatedPhysicsObject`

Class encapsulating the physics, collision and graphical objects representing a body.

All the public attributes of the physics object (*_body*) are accessible as if they were from this class, by using a trick with *__getattr__*. This avoids code duplication and frequent changes to the interface.

For example, the call *sim_body.get_linear_velocity()* works if method *sim_body._body.get_linear_velocity* exists.

There are some exceptions such as the getters and setters of position and rotation because the base class *SimulatedPhysicsObject* defines those abstract methods (some other non-abstract methods use them) and requires its subclasses to implement them. Otherwise we get "TypeError: Can't instantiate abstract class SimulatedBody with abstract methods".

`body`

`get_position`()

Get the position of the body.

Returns position

Return type 3-sequence of floats

`get_rotation`()

Get the orientation of the body.

Returns rotation matrix

Return type 9-sequence of floats

`set_position`(*pos*)

Set the orientation of the body.

Parameters **`pos`** (*`3-sequence of floats`*) – position

`set_rotation`(*rot*)

Set the orientation of the body.

Parameters **`rot`** (*`9-sequence of floats`*) – rotation matrix

class **`SimulatedJoint`**(*name=None*, *joint=None*, *actor=None*)

Bases: `ars.model.simulator.SimulatedPhysicsObject`

`add_torque`(*torque*)

`get_position`()

`get_rotation`()

`joint`

`set_position`(*pos*)

`set_rotation`(*rot*)

class **`SimulatedObject`**(*name*, *actor=None*)

Bases: `object`

`actor`

`get_name`()

has_actor()

is_updatable()

set_name(*name*)

class **SimulatedPhysicsObject**(*name*, *actor=None*)
Bases: `ars.model.simulator.SimulatedObject`

get_pose()
Get the pose (3D position and rotation) of the object.

Returns pose

Return type `ars.utils.geometry.Transform`

get_position()
Get the position of the object.

Returns position

Return type 3-sequence of floats

get_rotation()
Get the orientation of the object.

Returns rotation matrix

Return type 9-sequence of floats

offset_by_object(*obj*)

offset_by_position(*offset_pos*)

rotate(*axis*, *angle*)
Rotate the object by applying a rotation matrix defined by the given axis and angle

set_pose(*pose*)
Set the pose (3D position and rotation) of the object.

Parameters **pose** (`ars.utils.geometry.Transform`) –

set_position(*pos*)
Set the orientation of the object.

Parameters **pos** (*3-sequence of floats*) – position

set_rotation(*rot*)
Set the orientation of the object.

Parameters **rot** (*9-sequence of floats*) – rotation matrix

update_actor()
If there is no actor, it won't do anything

class **Simulation**(*FPS*, *STEPS_PF*)
Bases: `object`

actors
Return a dict with each object actor indexed by the object name.

add_axes()

add_ball_socket_joint(*name*, *obj1*, *obj2*, *anchor*)
Adds a "ball and socket" joint between obj1 and obj2, at the specified anchor. If anchor is None, it will be set equal to the position of obj2.

add_basic_simulation_objects(*gravity=(0.0, -9.81, 0.0)*)
Create the basic simulation objects needed for physics and collision such as a contact group (holds temporary contact joints generated during collisions), a simulation 'world' (where physics objects are processed) and a collision space (the same thing for geoms and their intersections).

Parameters **gravity** (`3 floats tuple.`) – Gravity acceleration.

add_box(*size, center, mass=None, density=None*)

add_capsule(*length, radius, center, mass=None, density=None*)

add_cylinder(*length, radius, center, mass=None, density=None*)

add_fixed_joint(*obj1, obj2*)

add_floor(*normal=(0, 1, 0), dist=0, box_size=(5, 0.1, 5), box_center=(0, 0, 0), color=(0.2, 0.5, 0.5)*)
Create a plane geom to simulate a floor. It won't be used explicitly later (space object has a reference to it)

add_joint(*sim_joint*)

add_object(*sim_object*)
Add `sim_object` to the internal dictionary of simulated objects.

If its name equals an already registered key, it will be modified using its string representation, for example:

```
>>> add_object(sim_object)
sphere/<ars.model.simulator.SimulatedBody object at 0x3a4bed0>
```

Parameters **sim_object** (`SimulatedObject`) – object to add

Returns name/key of the object

Return type string

add_rotary_joint(*name, obj1, obj2, anchor, axis*)
Adds a rotary joint between obj1 and obj2, at the specified anchor and with the given axis. If anchor is None, it will be set equal to the position of obj2

add_slider_joint(*name, obj1, obj2, axis*)
Add a `jo.Slider` joint between `obj1` and `obj2`.

The only movement allowed is translation along `axis`.

Returns the name under which the slider was stored, which could be different from the given `name`

add_sphere(*radius, center, mass=None, density=None*)

add_trimesh_floor(*vertices, faces, center=(0, 0, 0), color=(0.2, 0.5, 0.5)*)

add_universal_joint(*obj1, obj2, anchor, axis1, axis2*)

collision_space

get_bodies()
Return a list with all the bodies included in the simulation.

Returns list of `SimulatedBody` objects

get_joint(*name*)

get_object(*name*)

gravity

joints

objects

on_idle()

perform_sim_steps_per_frame()

sim_time

time_step

update_actors()

Update pose of each simulated object's corresponding actor.

signals Module This module contains string values defining different signals related to the `ars.model.simulator` package.

utils Package

containers Module

class Queue

Bases: `object`

FIFO

clear()

Remove all elements of the queue

convert_to_list()

Return the elements in an ordered list

count()

is_empty()

pull()

Remove and return the first element of the queue

put(*element*)

Appends the element as the last object of the queue

generic Module Generic utility functions to

- write variables and tuples to file
- write and read setting from a file
- modify and create tuples.

get_current_epoch_time()

Return the current time in seconds since the Epoch.

Fractions of a second may be present if the OS provides them (UNIX-like do).

Returns number of seconds since the Epoch

Return type float

insert_in_tuple(*tuple_*, *index*, *item_*)

nested_iterable_to_tuple(*iterable_*)

read_settings(*filename*, *section*)

Read `section` from settings file at `filename`.

Parameters

- **filename** (*str*) – settings file
- **section** (*str*) – settings section

Returns settings section dictionary

Return type dict

Example:

```
>>> read_settings('test.cfg', 'mySection')
{'a_key': 1.1111, 'another_key': False}
```

write_settings (*filename*, *section*, *mapped_values*)

Write `mapped_values` at `section` in settings file at `filename`.

Parameters

- **filename** (*str*) – settings file
- **section** (*str*) – settings section
- **mapped_values** (*dict*) – values to write

Example:

```
>>> write_settings('test.cfg', 'mySection', {'a_key': 1.1111, 'another_key': False})
```

write_tuple_to_file (*text_file*, *tuple_*)

write_var_to_file (*text_file*, *var*)

geometry Module

class Transform (*pos=None*, *rot=None*)

Bases: `object`

An homogeneous transform.

It is a composition of rotation and translation. Mathematically it can be expressed as

$$\left[\begin{array}{ccc|c} & R & & T \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

where R is the 3x3 submatrix describing rotation and T is the 3x1 submatrix describing translation.

source: http://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters#Denavit-Hartenberg_matrix

Constructor.

With empty arguments it's just a 4x4 identity matrix.

Parameters

- **pos** (tuple, `numpy.ndarray` or None) – a size 3 vector, or 3x1 or 1x3 matrix
- **rot** (tuple, `numpy.ndarray` or None) – 3x3 or 9x1 rotation matrix

get_long_tuple ()

get_rotation (*as_numpy=False*)

Get the rotation component (matrix).

Parameters **as_numpy** – whether to return a numpy object or a tuple

Returns 3x3 rotation matrix

Return type tuple of tuples or `numpy.ndarray`

get_translation(*as_numpy=False*)
Get the translation component (vector).

Parameters **as_numpy** – whether to return a numpy object or a tuple

Returns 3-sequence

Return type tuple or `numpy.ndarray`

matrix
Return matrix that contains the transform values.

Returns 4x4 matrix

Return type `numpy.ndarray`

calc_compass_angle(*rot*)
Return the angle around the vertical axis with respect to the *X+* axis, i.e. the angular orientation inherent of a rotation matrix `rot`, constrained to the plane aligned with the horizon (*XZ*, since the vertical axis is *Y*).

calc_inclination(*rot*)
Return the inclination (as `pitch` and `roll`) inherent of rotation matrix `rot`, with respect to the plane (*XZ*, since the vertical axis is *Y*). `pitch` is the rotation around *Z* and `roll` around *Y*.

Examples:

```
>>> rot = calc_rotation_matrix((1.0, 0.0, 0.0), pi/6)
>>> pitch, roll = gemut.calc_inclination(rot)
0.0, pi/6
```

```
>>> rot = calc_rotation_matrix((0.0, 1.0, 0.0), whatever)
>>> pitch, roll = gemut.calc_inclination(rot)
0.0, 0.0
```

```
>>> rot = calc_rotation_matrix((0.0, 0.0, 1.0), pi/6)
>>> pitch, roll = gemut.calc_inclination(rot)
pi/6, 0.0
```

calc_rotation_matrix(*axis, angle*)
Return the row-major 3x3 rotation matrix defining a rotation of magnitude `angle` around `axis`.

Formula is the same as the one presented here (as of 2011.12.01): http://goo.gl/RkW80

$$R = \begin{bmatrix} \cos\theta + u_x^2\left(1-\cos\theta\right) & u_x u_y\left(1-\cos\theta\right) - u_z\sin\theta & u_x u_z\left(1-\cos\theta\right) + u_y\sin\theta \\ u_y u_x\left(1-\cos\theta\right) + u_z\sin\theta & \cos\theta + u_y^2\left(1-\cos\theta\right) & u_y u_z\left(1-\cos\theta\right) - u_x\sin\theta \\ u_z u_x\left(1-\cos\theta\right) - u_y\sin\theta & u_z u_y\left(1-\cos\theta\right) + u_x\sin\theta & \cos\theta + u_z^2\left(1-\cos\theta\right) \end{bmatrix}$$

The returned matrix format is length-9 tuple.

get_body_relative_vector(*body, vector*)
Return the 3-vector vector transformed into the local coordinate system of ODE body 'body'

make_OpenGL_matrix(*rot, pos*)
Return an OpenGL compatible (column-major, 4x4 homogeneous) transformation matrix from ODE compatible (row-major, 3x3) rotation matrix rotation and position vector position.

The returned matrix format is length-9 tuple.

rot_matrix_to_euler_angles(*rot*)
Return the 3-1-3 Euler angles *phi*, *theta* and *psi* (using the x-convention) corresponding to the rotation matrix *rot*, which is a tuple of three 3-element tuples, where each one is a row (what is called row-major order).

Using the x-convention, the 3-1-3 Euler angles *phi*, *theta* and *psi* (around the Z, X and again the Z-axis) can be obtained as follows:

$$\phi = \arctan 2(A_{31}, A_{32})$$
$$\theta = \arccos(A_{33})$$
$$\psi = -\arctan 2(A_{13}, A_{23})$$

http://en.wikipedia.org/wiki/Rotation_representation_(mathematics)%23Rotation_matrix_.E2.86.94_Euler_angles

rot_matrix_to_hom_transform(*rot*)
Convert a rotation matrix to a homogeneous transform.

source: transform.r2t in Corke's Robotic Toolbox (python)

Parameters **rot** (a tuple, a tuple of tuples or `numpy.ndarray`) – 3x3 rotation matrix

mathematical Module Functions to perform operations over vectors and matrices; deal with homogeneous transforms; convert angles and other structures.

acos_dot3(*vector1*, *vector2*)
Return the angle between unit 3-dimension `vector1` and `vector2`.

add3(*vector1*, *vector2*)
Return the sum of 3-dimension `vector1` and `vector2`.

calc_acceleration(*time_step*, *vel0*, *vel1*)
Calculate the vectorial substraction `vel1 - vel0` divided by `time step`. If any of the vectors is `None`, then `None` is returned.

`vel1` is the velocity measured `time_step` seconds after `vel0`.

cross_product(*vector1*, *vector2*)
Return the cross product of 3-dimension `vector1` and `vector2`.

degrees_to_radians(*degrees_*)

dist3(*vector1*, *vector2*)
Return the distance between point 3-dimension `vector1` and `vector2`.

div_by_scalar3(*vector*, *scalar*)
Return 3-dimension `vector` divided by `scalar`.

dot_product(*vec1*, *vec2*)
Efficient dot-product operation between two vectors of the same size. source: http://docs.python.org/library/itertools.html

dot_product3(*vector1*, *vector2*)
Return the dot product of 3-dimension `vector1` and `vector2`.

length2(*vector*)
Return the length of a 2-dimension `vector`.

length3(*vector*)
Return the length of a 3-dimension `vector`.

matrix_as_3x3_tuples(*tuple_9*)
Return `tuple_9` as a 3-tuple of 3-tuples.

Parameters **tuple_9** – tuple of 9 elements

Returns `tuple_9` formatted as tuple of tuples

matrix_as_tuple(*matrix_*)
Convert `matrix_` to a tuple.

Example:

```
>>> matrix_as_tuple(((1, 2), (3, 4)))
(1, 2, 3, 4)
```

Parameters **matrix** (*tuple*) – nested tuples

Returns `matrix_` flattened as a tuple

Return type tuple

matrix_multiply(*matrix1*, *matrix2*)
Return the matrix multiplication of `matrix1` and `matrix2`.

Parameters

- **matrix1** – LxM matrix
- **matrix2** – MxN matrix

Returns LxN matrix, product of `matrix1` and `matrix2`

Return type tuple of tuples

mult_by_scalar3(*vector*, *scalar*)
Return 3-dimension `vector` multiplied by `scalar`.

neg3(*vector*)
Return the negation of 3-dimension `vector`.

norm3(*vector*)
Return the unit length vector parallel to 3-dimension `vector`.

np_matrix_to_tuple(*array_*)
Convert Numpy 2D array (i.e. matrix) to a tuple of tuples.

source: http://stackoverflow.com/a/10016379/556413

Example:

```
>>> arr = numpy.array(((2, 2), (2, -2)))
>>> np_matrix_to_tuple(arr)
((2, 2), (2, -2))
```

Parameters **array** (`numpy.ndarray`) – 2D array (i.e. matrix)

Returns matrix as tuple of tuples

project3(*vector*, *unit_vector*)
Return projection of 3-dimension `vector` onto unit 3-dimension `unit_vector`.

radians_to_degrees(*radians_*)

rotate3(*rot*, *vector*)
Return the rotation of 3-dimension `vector` by 3x3 (row major) matrix `rot`.

sign(*x*)
Return `1.0` if `x` is positive, `-1.0` otherwise.

sub3 (*vector1*, *vector2*)
Return the difference between 3-dimension `vector1` and `vector2`.

transpose3 (*matrix*)
Return the inversion (transpose) of 3x3 rotation matrix `matrix`.

unitize (*vector_*)
Unitize a vector, i.e. return a unit-length vector parallel to `vector`.

vec3_degrees_to_radians (*vector_*)

vec3_radians_to_degrees (*vector_*)

vector_matrix_vector (*vector_*, *matrix_*)
Return the product of `vector_` transposed, `matrix_` and `vector` again, which is a scalar value.

$$v^\top \mathbf{M} v$$

z_axis (*rot*)
Return the z-axis vector from 3x3 (row major) rotation matrix `rot`.

version Module

get_hg_changeset ()
Return the global revision id that identifies the working copy.

To obtain the value it runs the command `hg identify --id`, whose short form is `hg id -i`.

```
>>> get_hg_changeset()
1a4b04cf687a
>>> get_hg_changeset()
1a4b04cf687a+
```

Note: When there are outstanding (i.e. uncommitted) changes in the working copy, a + character will be appended to the current revision id.

get_hg_tip_timestamp ()
Return a numeric identifier of the latest changeset of the current repository based on its timestamp.

To obtain the value it runs the command `hg tip --template '{date}'`

>> get_hg_tip_timestamp() '20130328021918'

Based on: `django.utils.get_git_changeset` @ commit 9098504, and http://hgbook.red-bean.com/read/customizing-the-output-of-mercurial.html

Django's license is included at docs/Django BSD-LICENSE.txt

get_version (*version=None*, *length=u'full'*)
Return a PEP 386-compliant version number from `version`.

Parameters

- **version** (*tuple of strings*) – the value to format, expressed as a tuple of strings, of length 5, with the element before last (i.e. version[3]) equal to any of the following: `('alpha', 'beta', 'rc', 'final')`
- **length** (*basestring*) – the format of the returned value, equal to any of the following: `('short', 'medium', 'full')`

Returns version as a string

Return type str

```
>>> get_version(version=(0, 4, 0, 'alpha', 0))
0.4.dev20130401011455
>>> get_version(version=(0, 4, 0, 'alpha', 1))
0.4a1
>>> get_version(version=(0, 4, 1, 'alpha', 0))
0.4.1.dev20130401011455
>>> get_version(version=(0, 4, 1, 'alpha', 1))
0.4.1a1
>>> get_version(version=(0, 4, 0, 'beta', 0))
0.4b0
>>> get_version(version=(0, 4, 0, 'rc', 0))
0.4c0
>>> get_version(version=(0, 4, 0, 'final', 0))
0.4
>>> get_version(version=(0, 4, 0, 'final', 1))
0.4
>>> get_version(version=(0, 4, 1, 'final', 0))
0.4.1
```

```
>>> get_version(version=(0, 4, 0, 'alpha', 0), length='medium')
0.4.dev
>>> get_version(version=(0, 4, 0, 'alpha', 0), length='short')
0.4
```

Based on: `django.utils.version` @ commit 9098504. Django's license is included at docs/Django BSD-LICENSE.txt

Changelog

0.4a1 / 2013-04-13

- Blah

0.3a1 / 2012-10-17

- Buh
- Burrito

Language election

Articles

Why Python? by Eric Raymond on Apr 30, 2000 in *Linux Journal*. http://www.linuxjournal.com/article/3882

It was written more than 12 years ago when Python was far less powerful than today but, nonetheless, his opinion and first experiences with the language may be quite similar to what happens to some people nowadays.

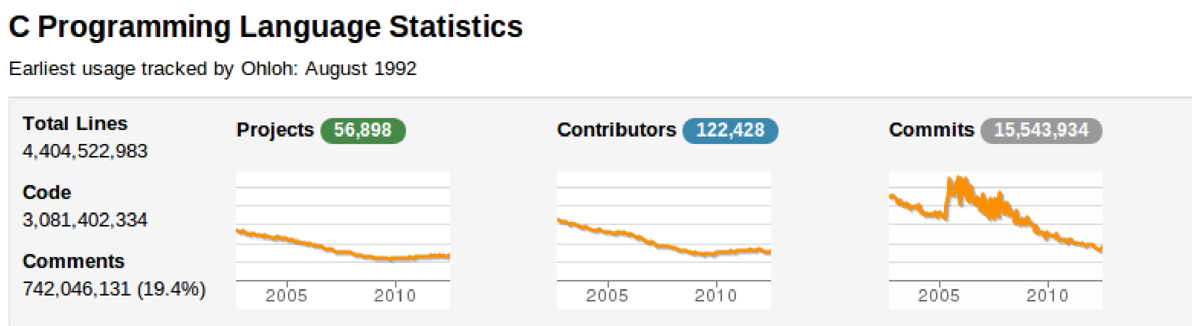
Language comparison

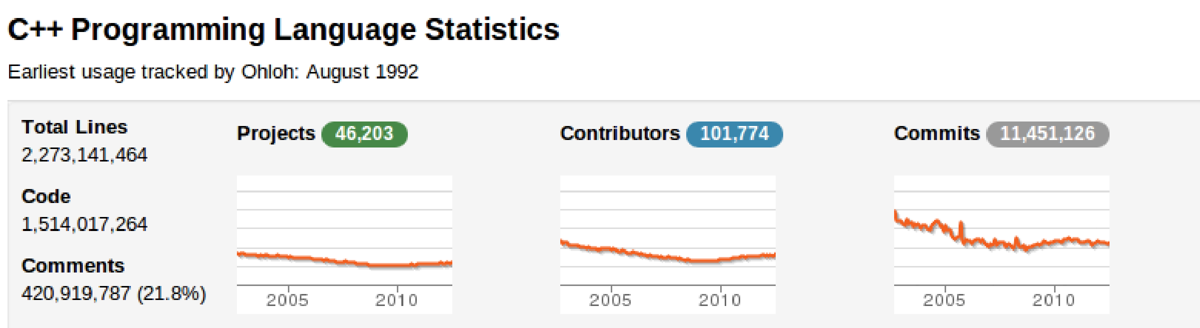
Ohloh

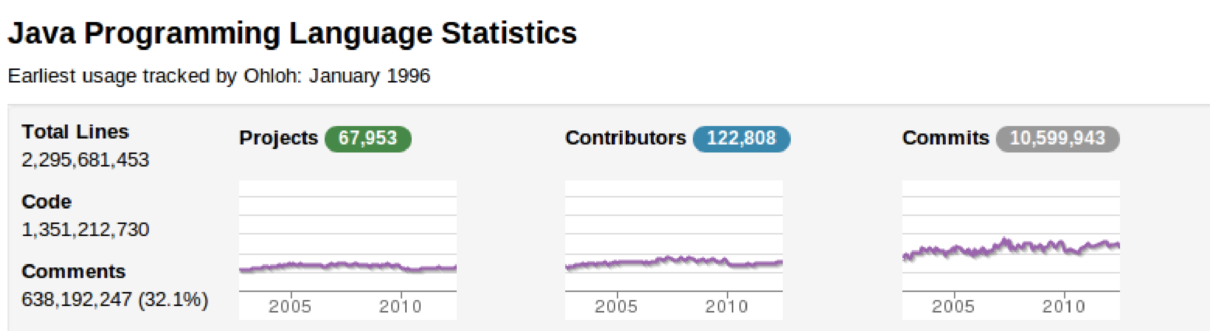
What is Ohloh? Ohloh is a free, public directory of Free and Open Source Software (FOSS) and the contributors who create and maintain it. Ohloh Code is a publicly available, free code search site that indexes most of the projects in Ohloh.

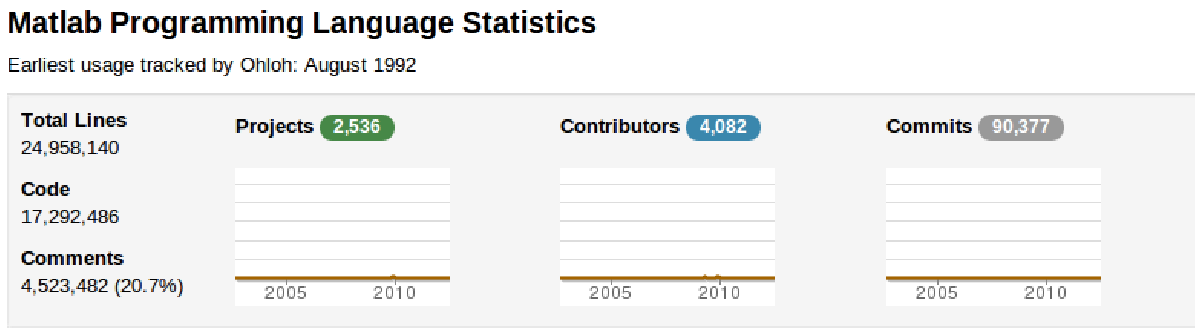
Note: Information retrieved on August 12th, 2012

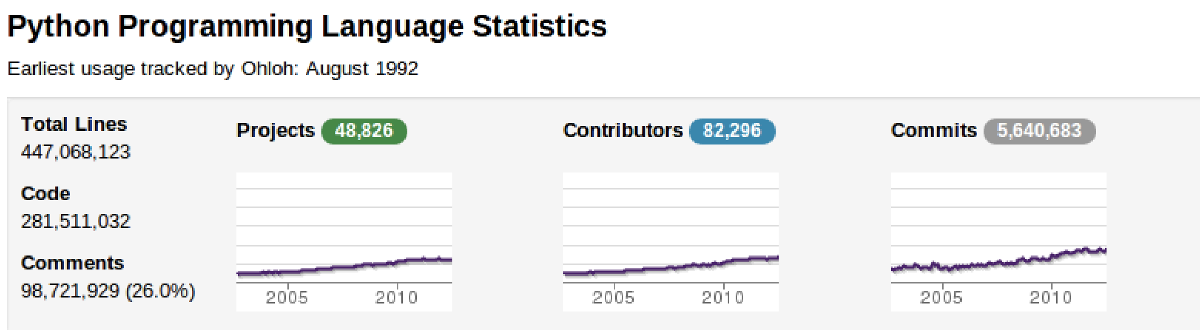
Programming Language Statistics









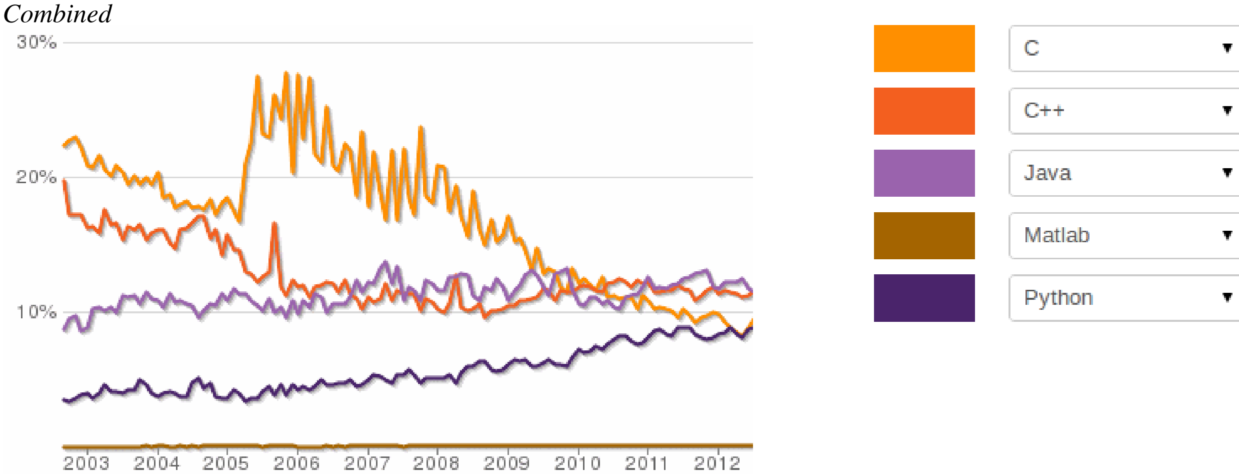


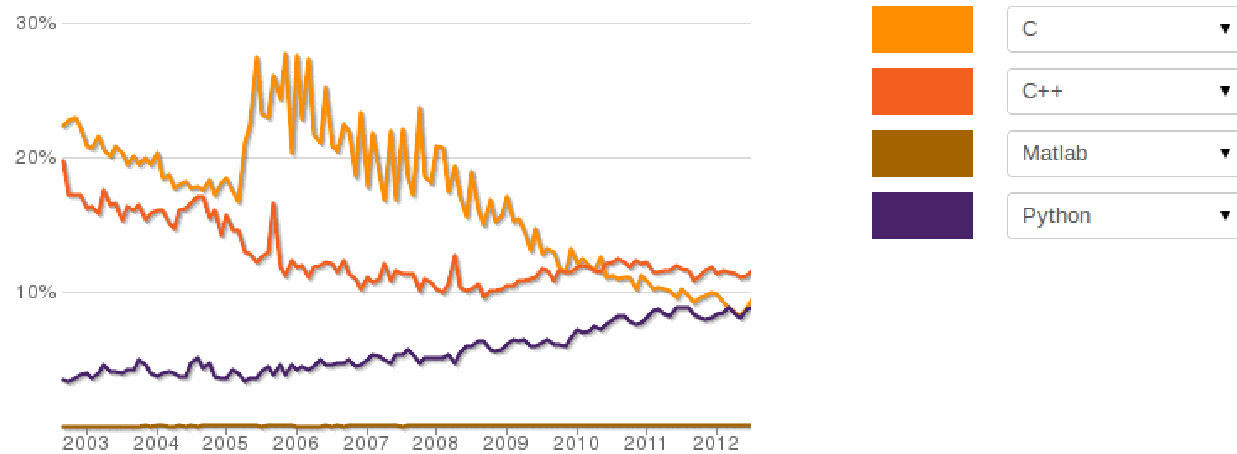
sources: C, C++, Java, Matlab & Python

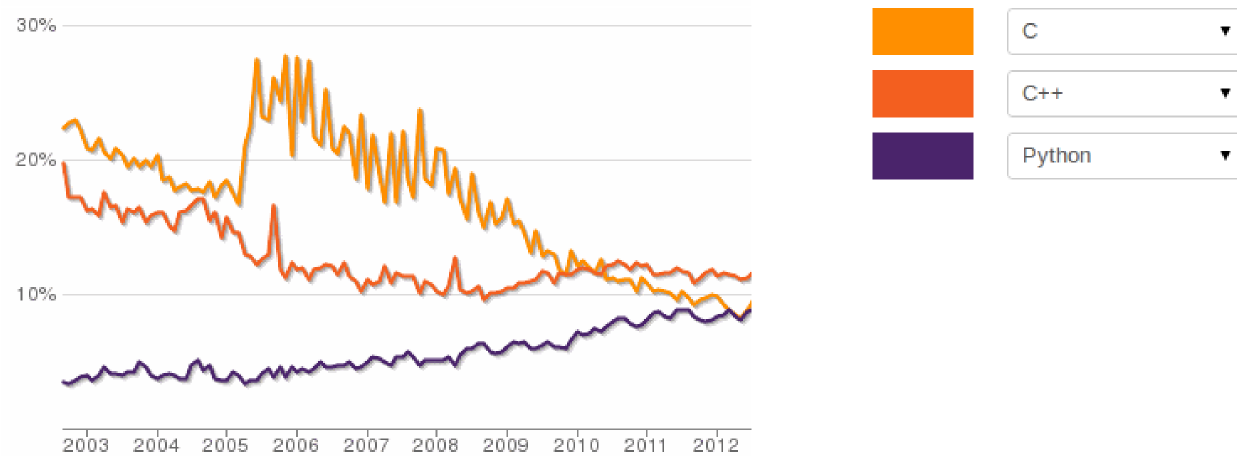
Language comparison (monthly commits as percent of total)

The lines show the count of monthly commits made by source code developers. Commits including multiple languages are counted once for each language. More

Combined





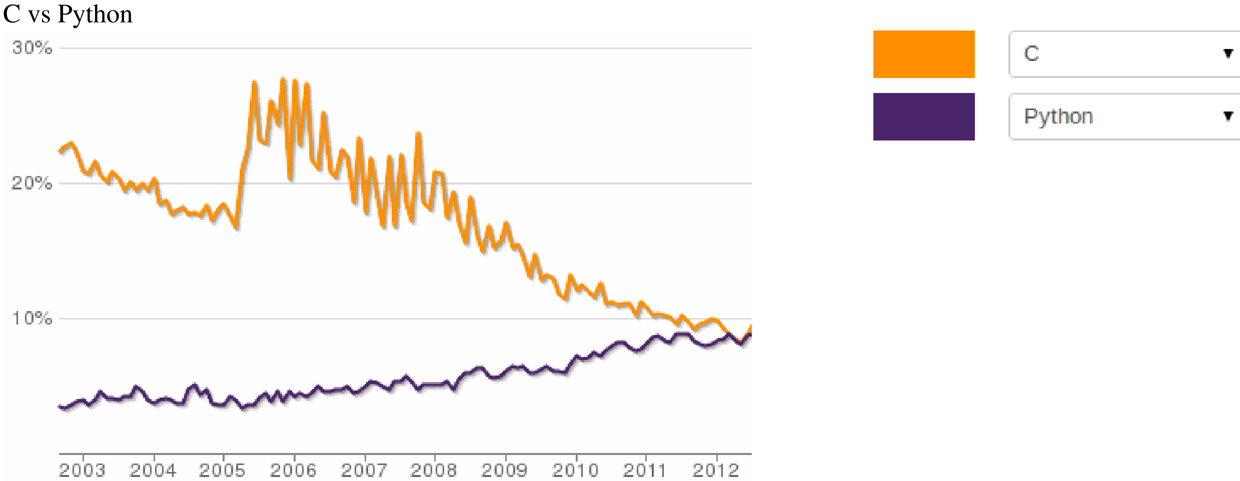


sources: C, C++, Matlab, Java & Python | C, C++, Matlab & Python | C, C++ & Python

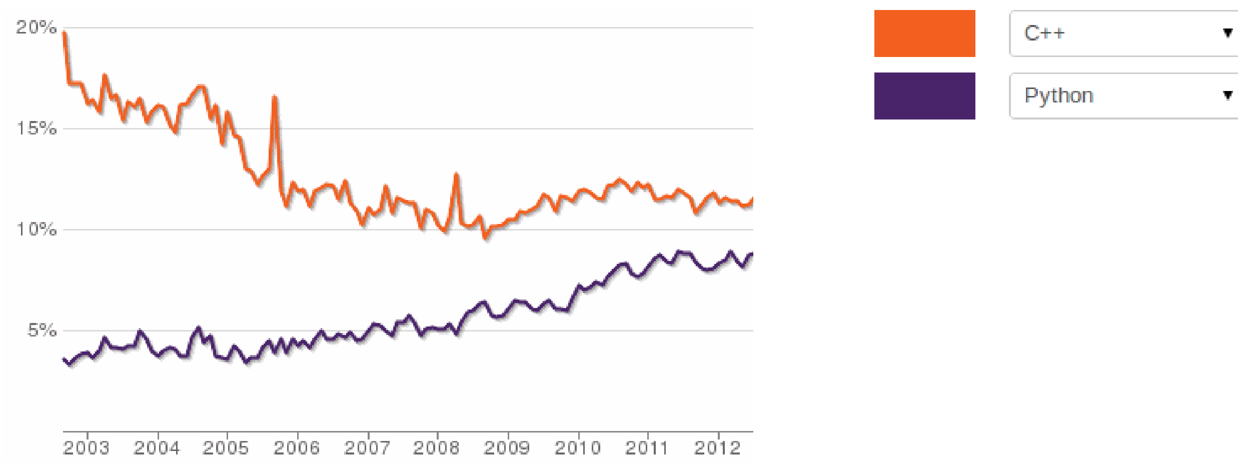
---

One on one

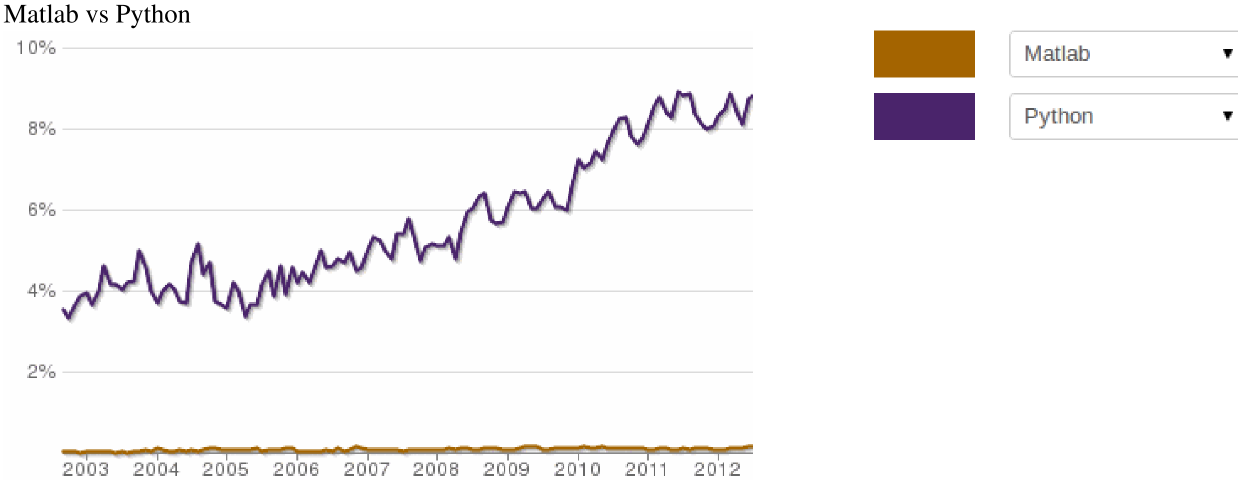
C vs Python



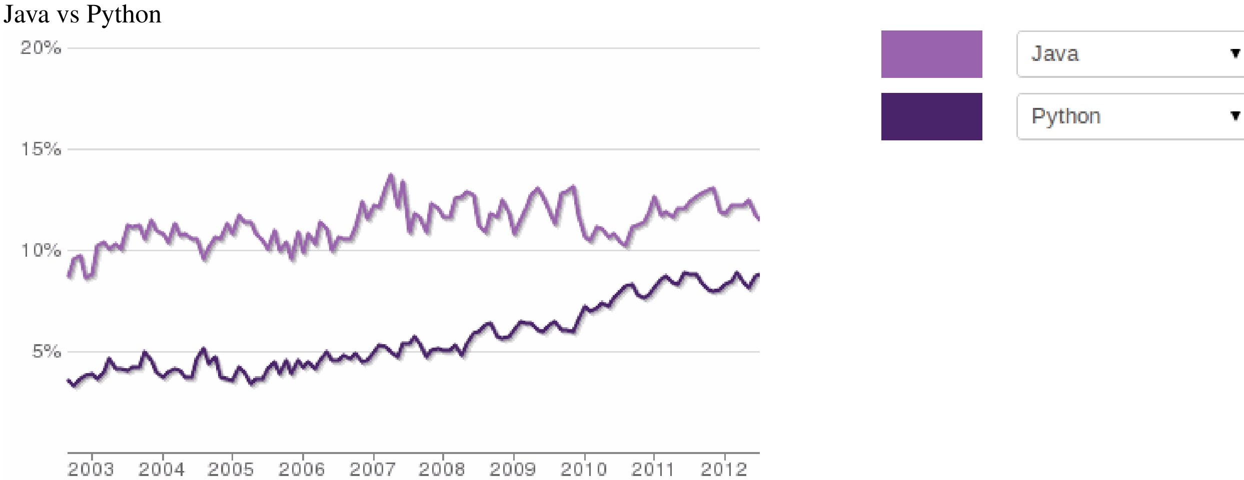
C++ vs Python



Matlab vs Python



Java vs Python



sources: C & Python | C++ & Python | Matlab & Python | Java & Python

Python

About Python

Basic information

If you are curious about Python, its history, how and where it is used, check out the official FAQ. If unfamiliar with some terms, see the glossary.

Python resources

Tutorials and guides

If you would like to get a taste of Python get started with this tutorial (included in the official documentation) which is very straight forward and does not presume any previous knowledge of the language. However, it is recommended to have at least Python installed (see next section).

Anothe way to get started from zero is the Beginner's Guide to Python.

Python on Windows

Although Python runs on Windows with no problems, sometimes it's difficult to get started in this OS because of some small details that interfere with what whould be a smooth process on a *Nix system.

Syntax (e.g. path delimiters), permissions, end-of-line character, etc, can stop the user from doing what he should be doing, i.e. learning the language, instead of dealing with Windows annoyances (for more information on this topic, go to annoyances.org).

That's why there is a special section on the Python documentation called Python on Windows FAQ. So, if you are having issues with Python stuff on a Windows OS, go read that FAQ.

Python(x,y)

> Python(x,y) is a free Python distribution providing a ready-to-use scientific development software for numerical computations, data analysis and data visualization based on Python programming language, Qt graphical user interfaces (and development framework) and Spyder interactive development environment. Its purpose is to help scientific programmers used to interpreted languages (such as MATLAB or IDL) or compiled languages (C/C++ or Fortran) to switch to Python.

It is very popular for Windows, but there is a GNU/Linux version too.

Python installation

To have Python installed means you have installed a Python interpreter. In a UNIX-like system (e.g. GNU-Linux, Mac OS, etc) you probably won't need to because it is often included in the base installation. On Microsoft Windows, however, it is not so you won't have it unless it was a dependency for some software (e.g. GIMP). Curious about that? Go read Why is Python Installed on my Computer? FAQ

Online interpreters/shells

If you can't or don't want to install Python, you can resort to online interpreters made available by some generous or commercial efforts. Here are free alternatives:

- The repl.it project is an attempt to create an online environment for interactively exploring programming languages. It provides a fully-featured terminal emulator and code editor, powered by interpreter engines for more than 15 languages. All our interpreters are written in (or compiled to) JavaScript, and run completely on the user's device, regardless or whether it's a desktop, laptop or phone.
- PythonAnywhere is a Python development and hosting environment that displays in your web browser and runs on our servers. They're already set up with everything you need. It's easy to use, fast, and powerful. There's even a useful free plan.
- Try IPython from your browser! IPython is an enhanced interactive Python interpreter, offering tab completion, object introspection, and much more. It's running on the right-hand side of this page, so you can try it out right now.
- codepad is an online compiler/interpreter, and a simple collaboration tool.
- Interactive server-side Python (2.5.2) shell for Google App Engine
- Ideone is something more than a pastebin; it's an online compiler and debugging tool which allows to compile and run code online in more than 40 programming languages.
- http://mathcs.holycross.edu/~kwalsh/python/
- http://people.csail.mit.edu/pgbovine/python/
- http://www.trypython.org/ requires Microsoft Silverlight :(

Other

http://www.rackspace.com/knowledge_center/web-resources/top-50-resources-for-programming-web-applications-with-python

Python is a general purpose, object-oriented programming language that has achieved popularity because of its readability and clear syntax. Guido van Rossum created Python in late 1980s. It is a 'high level' scripting language used by most programmers for its simplicity of use. Python can be written once and run on any computer. It's a 'multi paradigm programming language' compatible with many other programming languages such as .NET, CORBA, Java, Perl.

Dave Kuhlman's page. Open Source software projects by Dave Kuhlman.

These projects are implemented in or for Python. These projects center around XML, parsing XML, etc. They provide tools for building data mapping and Web services. Keywords are: python, xml, editor, text processing, python training.

Developers FAQ

See also:

http://docs.python.org/devguide/faq

Version control

See also:

http://docs.python.org/devguide/faq#version-control

About the documentation

When documenting ARS, some good practices are required. However, if you are not familiar with the tools or guidelines used, please do contribute by sending plain text and someone else will take care of including it in the proper way.

Style guide

Although not yet enforced, the Python documentation style guide is the reference for writing ARS's documentation.

Generation

This documentation is generated automatically by Sphinx from rst (reStructuredText) files and the docstrings in the code, both of which are in the project's repository.

For more information about Sphinx, see http://sphinx.pocoo.org/

For more information about reStructuredText, see http://docutils.sourceforge.net/rst.html and http://sphinx.pocoo.org/rest.html

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z