README

This document briefly covers the creation of a Python binding for a ROS C++ package. It uses straight boost Python library as opposed to Py++ or SIP or SWIG or Pybind11. The reason boost Python library was selected was that ROS uses it in moveit!! and has a tutorial on it, called "Using a C++ class in Python". The tutorial can be found at:

<http://wiki.ros.org/ROS/Tutorials/Using%20a%20C%2B%2B%20class%20in%20Python>

To get Python binding from the ROS C++ code, we use boost::python, which does a lot of magic under the hood. (There is still a lot of magic left undocumented!) It is important to be familiar with boost::python. This is a good tutorial to get started:

<http://boost.cppll.jp/HEAD/libs/python/doc/tutorial/> and

<https://wiki.python.org/moin/boost.python>

Other potentially helpful web sites are spelled out in the appendix.

# 1. Create a new catkin package

To make matters clean, separate the ROS C++ pacakge from the ROS python bindings to the ROS C++ pacakge. In oder to do this first create a new ROS catkin package. Configuring the package.xml and the CmakeLists.txt will be covered later.

# 2. Create cpp source file to hold C++ boost python code

Next you will need to implement a boost/python binding to your ROS C++. So you will need to include the boost python header (which does the heavy binding lifting) and your ROS package header. The fundamental Python module definition is done with the macro BOOST\_PYTHON\_MODULE(rospackage\_python) whose name "rospackage\_python" must match your eventual ROS package name!

#include <boost/python.hpp>

#include <urrospackage/PkgMainHeader.h>

BOOST\_PYTHON\_MODULE(rospackage\_python) {

}

The important things to note about this file:

1. The module file is the entry point for a boost::python module. There should only be one call to BOOST\_PYTHON\_MODULE() within any single library and this should call functions that export all the pieces of your python package.
2. The library name in the BOOST\_PYTHON\_MODULE() call must match the name of the library produced by building your code in C++. This name must match exactly or the python import statement will fail. Of course, it can fail for other reasons!

# 3. Implement python bindings

Inside of the module.cpp file you will need to implement bindings to your ROS C++ API. For simple types, for example, integer, double Boolean, the boost python library will handle the conversion. Ironically, more complicated data structures may require conversion between Python "objects" and C++ classes/struct. Most examples showed C++ API with doubles. More common, a Python "list" must be mapped into and out of a std::vector structure. Thus, depending on the list type, you will need to convert to the appropriate template type. It is not obvious from the tutorial on the C++ Python bindings

Below are some links that have C++ code to assist in the C++ Python type conversion problem.

This website in particular enlighten me to the C++ Python type conversion:

<http://shitohichiumaya.blogspot.com/2010/08/boostpython-how-to-pass-python-object.html>

# 4. Create the python \_\_init\_\_.py and setup.py file

The following describes the implementation for the gotrajcommander ROS package.

This is how I laid out my ROS package for the python bindings of gotraj. It was a normal package in that the src and include/gotrajcommander folders contained C++ header and source files. The python/gotrajcommander contained an \_\_init\_\_.py. The setup.py was in the gotrajcommander folder.

gotrajcommander:

├───src

├───scripts

├───python

│ └───gotrajcommander

└───include

└───gotrajcommander

To import the gotrajcommander library into python, we need a proper python module: a directory with an \_\_init\_\_.py file. By convention, we put these in

ROSworkspace/src/gotrajcommander/python/gotrajcommander.

Whatever name you use here, this is what people will have to use when they call import in python. So, in this case, people will have to type import gotrajcommander.

The init file must import all the c++ bindings.

The setup.py file followed the directions from the ROS tutorial ( <http://wiki.ros.org/ROS/Tutorials/Using%20a%20C%2B%2B%20class%20in%20Python> ):

from distutils.core import setup

from catkin\_pkg.python\_setup import generate\_distutils\_setup

setup\_args = generate\_distutils\_setup(

packages=['gotrajcommander'],

package\_dir={'':'python'})

setup(\*\*setup\_args)

You must be very careful typing this file.

# 5. Setup your package.xml

The package.xml tells ROS what build and runtime packages you will be using in catkin. The gotrajcommander was a python interface to the gotraj ROS package, which depended on sensor\_msgs for Joint information and tf for robot pose information. ROS python uses geometry\_msgs which is included in the boost python definitions.

<?xml version="1.0"?>

<package>

<name>gotrajcommander</name>

<version>0.1.0</version>

<description>

ROS python package to use gomotion trajectory and interpoloation motion facilities.

</description>

<author email="john.michaloski@nist.gov">John Michaloski(NIST)</author>

<maintainer email="john.michaloski@nist.gov">John Michaloski(NIST)</maintainer>

<license>NO copyright</license>

<buildtool\_depend>catkin</buildtool\_depend>

<build\_depend>roscpp</build\_depend>

<build\_depend>rospy</build\_depend>

<build\_depend>boost</build\_depend>

<build\_depend>cmake\_modules</build\_depend>

<build\_depend>python\_modules</build\_depend>

<build\_depend>gotraj</build\_depend>

<build\_depend>sensor\_msgs</build\_depend>

<build\_depend>geometry\_msgs</build\_depend>

<build\_depend>tf</build\_depend>

<run\_depend>rospy</run\_depend>

<run\_depend>roscpp</run\_depend>

<run\_depend>gotraj</run\_depend>

<run\_depend>sensor\_msgs</run\_depend>

<run\_depend>geometry\_msgs</run\_depend>

<run\_depend>tf</run\_depend>

</package>

# 6. Setup your CMakeLists.txt

Again, read the ROS tutorial ( <http://wiki.ros.org/ROS/Tutorials/Using%20a%20C%2B%2B%20class%20in%20Python> ) carefully. You will need to include both boost (boost/python.hpp) and python (python.h) to compile. The "gotraj" library is the C++ library that the Python binding are for, and the gotraj library must be linked against. To include gotraj it was a ROS catkin package, so it was a REQUIRED COMPONENTS of catkin.

As a note, I tried catkin\_simple but the package wasn't installed and I didn't find help, so I want back to the complicated brute force method.

The catkin\_python\_setup() in the CmakeLists.txt file copies the linked library to the python 2.7 destination.

cmake\_minimum\_required(VERSION 2.8.3)

project(gotrajcommander)

find\_package(catkin REQUIRED COMPONENTS

roscpp

rospy

cmake\_modules

sensor\_msgs

geometry\_msgs

tf

gotraj

)

## Both Boost.python and Python libs are required.

find\_package(Boost REQUIRED COMPONENTS python)

find\_package(PythonLibs 2.7 REQUIRED)

## Uncomment this if the package has a setup.py. This macro ensures

## modules and global scripts declared therein get installed

## See http://ros.org/doc/api/catkin/html/user\_guide/setup\_dot\_py.html

catkin\_python\_setup()

###################################

## catkin specific configuration ##

###################################

catkin\_package(

INCLUDE\_DIRS include

LIBRARIES gotrajcommander

CATKIN\_DEPENDS

roscpp

sensor\_msgs

geometry\_msgs

tf

gotraj

DEPENDS

Boost

)

###########

## Build ##

###########

include\_directories(include

${catkin\_INCLUDE\_DIRS}

${Boost\_INCLUDE\_DIRS}

${PYTHON\_INCLUDE\_DIRS}

)

set(CMAKE\_CXX\_FLAGS "-w ${CMAKE\_CXX\_FLAGS}")

set(CMAKE\_CXX\_FLAGS "-std=c++0x ${CMAKE\_CXX\_FLAGS}")

set(CMAKE\_CXX\_FLAGS "-DDEBUG ${CMAKE\_CXX\_FLAGS}")

set(CMAKE\_CXX\_FLAGS "-DBOOST\_LOG\_DYN\_LINK ${CMAKE\_CXX\_FLAGS}")

set(CMAKE\_CXX\_FLAGS "-fpermissive ${CMAKE\_CXX\_FLAGS}")

set(CMAKE\_CXX\_FLAGS "-g ${CMAKE\_CXX\_FLAGS}")

set(CMAKE\_CXX\_FLAGS "-DGO\_REAL\_DOUBLE ${CMAKE\_CXX\_FLAGS}")

set(CMAKE\_EXE\_LINKER\_FLAGS "-rdynamic ${CMAKE\_EXE\_LINKER\_FLAGS}")

set(GOTRAJCOMMANDER\_LIB\_SOURCES

src/gotrajcommander.cpp

)

add\_library(gotrajcommander ${GOTRAJCOMMANDER\_LIB\_SOURCES})

link\_directories(${CATKIN\_DEVEL\_PREFIX}/lib})

# catkin will know about gotraj and include as link

target\_link\_libraries(gotrajcommander ${catkin\_LIBRARIES} ${Boost\_LIBRARIES})

# Don't prepend wrapper library name with lib and add to Python libs.

set\_target\_properties(gotrajcommander PROPERTIES

PREFIX ""

LIBRARY\_OUTPUT\_DIRECTORY ${CATKIN\_DEVEL\_PREFIX}/${CATKIN\_PACKAGE\_PYTHON\_DESTINATION}

)

Then, compile with catkin tools.

> catkin build

OR

> catkin build -DCMAKE\_BUILD\_TYPE=Debug

OR

> catkin build -DCMAKE\_BUILD\_TYPE=Debug -DPYTHON\_VERSION=2.7

Although the third option appears to be unnecessary because the Cmake had specified the python version.

# 6. Boost Python Source

The following is the source code used to generate Python bindings to "gotraj" library using boost python. There are more capabilities but only a few features were needed. All the potential Python objects that come into or out of the bindings file has been declared in the BOOST\_PYTHON\_MODULE(gotrajcommander) code module section. In addition, the conversion routine to take JointState lists and turn them into C++ std vectors was also included, which simplified the code.

The code could be better, but works. There was a lot of confidence to things working, so things may not be as concise as possible.

The GoTraj class was wrapped with a new class that used a boost shared pointer to access this class. Where potential python objects needed to be converted either into C++ or out of C++ into objects, it was handled by the wrapper class.

The included file py\_bindings.h can be found in the ROS moveit repository:

<https://github.com/ros-planning/moveit/tree/kinetic-devel/moveit_ros/planning_interface/py_bindings_tools>

I am not sure it is necessary any more as boost python handles list to std::vector conversions automatically, but it works. And worked the first time.

#include <boost/python.hpp>

#include <boost/shared\_ptr.hpp>

#include <boost/python/converter/registry.hpp>

#include <boost/python/suite/indexing/vector\_indexing\_suite.hpp>

#include <geometry\_msgs/Pose.h>

#include "gotraj/gotraj.h"

#include "gotrajcommander/py\_conversions.h"

#include "gotrajcommander/Debug.h"

using namespace boost::python;

// Virtual functions and overloads are ignored in this implementation

//BOOST\_PYTHON\_MEMBER\_FUNCTION\_OVERLOADS(InitPose\_member\_overloads, GoTraj::InitPose, 3, 4)

//BOOST\_PYTHON\_MEMBER\_FUNCTION\_OVERLOADS(InitJoints\_member\_overloads, GoTraj::InitJoints, 3, 4)

inline geometry\_msgs::Pose ConvertPose(tf::Pose m) {

geometry\_msgs::Pose p;

p.position.x = m.getOrigin().x();

p.position.y = m.getOrigin().y();

p.position.z = m.getOrigin().z();

p.orientation.x = m.getRotation().x();

p.orientation.y = m.getRotation().y();

p.orientation.z = m.getRotation().z();

p.orientation.w = m.getRotation().w();

return p;

}

std::string DumpGoTrajParams(gomotion::GoTrajParams params) {

std::stringstream s;

s << "Vel=" << params.vel << ":Acc=" << params.acc << "Jerk=" << params.jerk << "\n";

return s.str();

}

struct GoTraj // : public gomotion::GoTraj, boost::python::wrapper<gomotion::GoTraj>

{

boost::shared\_ptr<gomotion::GoTraj> gm;

GoTraj() {

gm = boost::shared\_ptr<gomotion::GoTraj>(new gomotion::GoTraj());

}

int Init(object here, float time) {

gomotion::JointState jts;

jts.position = py\_bindings\_tools::typeFromList<double > (here.attr("position"));

gm->Init(jts, time);

return 0;

}

bool IsDone() {

return gm->IsDone();

}

int InitJoints(object here,

object there,

object params,

bool bCoordinated) {

gomotion::JointState herejts, therejts;

herejts.position = py\_bindings\_tools::typeFromList<double > (here.attr("position"));

std::cout << "Here Joints " << RCS::VectorDump<double>(herejts.position).c\_str() << "\n";

therejts.position = py\_bindings\_tools::typeFromList<double > (there.attr("position"));

std::cout << "THere Joints " << RCS::VectorDump<double>(therejts.position).c\_str() << "\n";

std::vector<gomotion::GoTrajParams> jtparams = py\_bindings\_tools::typeFromList<gomotion::GoTrajParams > (params);

std::cout << "Params " << RCS::VectorDump<double>(therejts.position).c\_str() << "\n";

return gm->InitJoints(herejts, therejts, jtparams, bCoordinated);

}

void AppendJoints(object there) {

gomotion::JointState therejts;

therejts.position = py\_bindings\_tools::typeFromList<double > (there.attr("position"));

std::cout << "Append There Joints " << RCS::VectorDump<double>(therejts.position).c\_str() << "\n";

gm->AppendJoints(therejts);

}

gomotion::JointState NextJoints() {

gomotion::JointState nextjts = gm->NextJoints();

return nextjts;

}

int InitPose(object here,

object there,

gomotion::GoTrajParams tparams,

gomotion::GoTrajParams rparams) {

tf::Pose h = ToPose(here);

tf::Pose t = ToPose(there);

std::cout << "Here Position " << RCS::DumpPoseSimple(h).c\_str() << "\n";

std::cout << "There Position " << RCS::DumpPoseSimple(t).c\_str() << "\n";

std::cout << "tparams " << DumpGoTrajParams(tparams);

std::cout << "rparams " << DumpGoTrajParams(rparams);

gm->InitPose(h, t, tparams, rparams);

return 0;

}

geometry\_msgs::Pose NextPose() {

tf::Pose pose = gm->NextPose();

//std::cout << "Next Position " << RCS::DumpPoseSimple(pose).c\_str() << "\n";

return ConvertPose(pose);

//object obj=FromPose(pose);

//return obj;

}

void AppendPose(object where) {

tf::Pose pose = ToPose(where);

gm->AppendPose(pose);

}

void InitStop() {

gm->InitStop();

}

protected:

tf::Pose ToPose(object here) {

std::vector<double> p, q;

p.push\_back(extract<double>(here.attr("position").attr("x")));

p.push\_back(extract<double>(here.attr("position").attr("y")));

p.push\_back(extract<double>(here.attr("position").attr("z")));

q.push\_back(extract<double>(here.attr("orientation").attr("x")));

q.push\_back(extract<double>(here.attr("orientation").attr("y")));

q.push\_back(extract<double>(here.attr("orientation").attr("z")));

q.push\_back(extract<double>(here.attr("orientation").attr("w")));

tf::Vector3 pos(p[0], p[1], p[2]);

tf::Quaternion quat(q[0], q[1], q[2], q[3]);

tf::Pose pose = tf::Pose(quat, pos);

return pose;

}

// This doesn't work - you need to create PyObject with correct info

object FromPose(tf::Pose pose) {

object here;

here.attr("position").attr("x") = pose.getOrigin().x();

here.attr("position").attr("y") = pose.getOrigin().y();

here.attr("position").attr("z") = pose.getOrigin().z();

here.attr("orientation").attr("x") = pose.getRotation().x();

here.attr("orientation").attr("y") = pose.getRotation().y();

here.attr("orientation").attr("z") = pose.getRotation().z();

here.attr("orientation").attr("w") = pose.getRotation().w();

return here;

}

};

typedef std::vector<double> MyList;

BOOST\_PYTHON\_MODULE(gotrajcommander) {

class\_<geometry\_msgs::Point>("Point")

.def\_readwrite("x", &geometry\_msgs::Point::x)

.def\_readwrite("y", &geometry\_msgs::Point::y)

.def\_readwrite("z", &geometry\_msgs::Point::z)

;

class\_<geometry\_msgs::Quaternion>("Quaternion")

.def\_readwrite("x", &geometry\_msgs::Quaternion::x)

.def\_readwrite("y", &geometry\_msgs::Quaternion::y)

.def\_readwrite("z", &geometry\_msgs::Quaternion::z)

.def\_readwrite("w", &geometry\_msgs::Quaternion::w)

;

class\_<geometry\_msgs::Pose>("Pose")

.def\_readwrite("position", &geometry\_msgs::Pose::position)

.def\_readwrite("orientation", &geometry\_msgs::Pose::orientation)

;

class\_<MyList>("MyList")

.def(vector\_indexing\_suite<MyList>());

class\_<gomotion::JointState >("JointState")

.def\_readwrite("header", &sensor\_msgs::JointState\_<std::allocator<void> >::header)

.def\_readwrite("name", &sensor\_msgs::JointState\_<std::allocator<void> >::name)

.def\_readwrite("position", &sensor\_msgs::JointState\_<std::allocator<void> >::position)

.def\_readwrite("velocity", &sensor\_msgs::JointState\_<std::allocator<void> >::velocity)

.def\_readwrite("effort", &sensor\_msgs::JointState\_<std::allocator<void> >::effort)

;

class\_<gomotion::GoTrajParams>("GoTrajParams", init<double, double, double>())

.def\_readwrite("vel", &gomotion::GoTrajParams::vel)

.def\_readwrite("acc", &gomotion::GoTrajParams::acc)

.def\_readwrite("jerk", &gomotion::GoTrajParams::jerk)

;

class\_<GoTraj>("GoTraj", init<>())

.def("Init", &GoTraj::Init)

.def("InitJoints", &GoTraj::InitJoints)

.def("IsDone", &GoTraj::IsDone, "Is motion done that is is queue empty")

.def("AppendJoints", &GoTraj::AppendJoints, "Append another set of joints to motion queue ")

.def("NextJoints", &GoTraj::NextJoints, "Get next joints from motion queue")

.def("InitPose", &GoTraj::InitPose, "InitPose on motion queue")

.def("NextPose", &GoTraj::NextPose, "Get next pose from motion queue")

.def("AppendPose", &GoTraj::AppendPose, "Append another pose onto motion queue")

.def("InitStop", &GoTraj::InitStop, "Stop motion in queue")

// FIXME: The time based gotraj has not been implemented in python

#if 0

.def("InitPose", &GoTraj::InitPose, InitPose\_member\_overloads(

args("here", "there", "tparams", "rparams"), "InitPose"

)

)

.def("InitJoints", &GoTraj::InitJoints, InitJoints\_member\_overloads(

args("here", "there", "params", "bCoordinated"), "Init Joints"

)

)

#endif

;

}

# Troubleshooting

This section covers a couple of the headaches that were uncovered in the generation of the Python bindings.

### Problem: python.h couldn't be found.

Use an environment variable to designate the python include for headers:

export CPLUS\_INCLUDE\_PATH=/usr/include/python2.7

### Problem: Location of test.py file!

michalos@woodsy:scripts> python test.py  
Traceback (most recent call last):  
  File "test.py", line 11, in <module>  
    from gotrajcommander  import GoTraj, GoTrajParams  
ImportError: cannot import name GoTraj

Of note, once catkin build successfully built the test.py file had to be in the

workspace/devil/lib/python2.7/dist-packages/gotrajcommander

folder in order to find the correct import the ROS python module.

devel:.

├───lib

│ ├───pkgconfig

│ ├───nist\_robotsnc

│ ├───gotraj

│ ├───python2.7

│ │ └───dist-packages

│ │ └───gotrajcommander

Catkin generated a new \_\_init\_\_.py in the folder that somehow informed python that the import name could be found and had the correct entries. The was the contents of the \_\_init\_\_.py file generated by catkin:

# -\*- coding: utf-8 -\*-

# generated from catkin/cmake/template/\_\_init\_\_.py.in

# keep symbol table as clean as possible by deleting all unnecessary symbols

from os import path as os\_path

from sys import path as sys\_path

from pkgutil import extend\_path

\_\_extended\_path = "/usr/local/michalos/nistfanuc\_ws/src/gotrajcommander/python".split(";")

for p in reversed(\_\_extended\_path):

sys\_path.insert(0, p)

del p

del sys\_path

\_\_path\_\_ = extend\_path(\_\_path\_\_, \_\_name\_\_)

del extend\_path

\_\_execfiles = []

for p in \_\_extended\_path:

src\_init\_file = os\_path.join(p, \_\_name\_\_ + '.py')

if os\_path.isfile(src\_init\_file):

\_\_execfiles.append(src\_init\_file)

else:

src\_init\_file = os\_path.join(p, \_\_name\_\_, '\_\_init\_\_.py')

if os\_path.isfile(src\_init\_file):

\_\_execfiles.append(src\_init\_file)

del src\_init\_file

del p

del os\_path

del \_\_extended\_path

for \_\_execfile in \_\_execfiles:

with open(\_\_execfile, 'r') as \_\_fh:

exec(\_\_fh.read())

del \_\_fh

del \_\_execfile

del \_\_execfiles

I'm assuming this init file changed the env variable for the python path.

### Problem: Return a Python object from a C++ object that isn't/wasn't declared in Python. That is, creating a C++ Python object dynamically

This worked but repeated the creation of the type JointState every time it was called:

struct GoTraj // : public gomotion::GoTraj, boost::python::wrapper<gomotion::GoTraj>  
{  
    boost::shared\_ptr<gomotion::GoTraj> gm;   
    GoTraj()  {  
       gm= boost::shared\_ptr<gomotion::GoTraj>(new gomotion::GoTraj());  
    }

    object NextJoints() {  
         gomotion::JointState nextjts = gm->NextJoints();  
      
         object next(class\_<JointState >("JointState")  
                .def\_readwrite("header", &sensor\_msgs::JointState\_<std::allocator<void> >::header)  
                .def\_readwrite("name", &sensor\_msgs::JointState\_<std::allocator<void> >::name)  
                .def\_readwrite("position", &sensor\_msgs::JointState\_<std::allocator<void> >::position)  
                .def\_readwrite("velocity", &sensor\_msgs::JointState\_<std::allocator<void> >::velocity)  
                .def\_readwrite("effort", &sensor\_msgs::JointState\_<std::allocator<void> >::effort)  
                );  
         next.attr("position") = py\_bindings\_tools::listFromType<double > (nextjts.position);  
        return next;  
    }

};

BOOST\_PYTHON\_MODULE(gotrajcommander){

class\_<GoTraj>("GoTraj", init<>())  
...  
.def("NextJoints",&GoTraj::NextJoints, "Get next joints on motioni queue")

}

The only problem is that a warning appears in the Python console that class\_<JointState > has already been registered.

Instead, added std::vector<double> conversion to the boost

#include <boost/python/suite/indexing/vector\_indexing\_suite.hpp>

#include "gotraj/gotraj.h"  //  has declaration of  gomotion::JointState

   gomotion::JointState NextJoints() {  
        gomotion::JointState nextjts = gm->NextJoints();  
        return nextjts;  
    }

typedef std::vector<double> MyList;  
  
BOOST\_PYTHON\_MODULE(gotrajcommander){  
    class\_<MyList>("MyList")  
        .def(vector\_indexing\_suite<MyList>() );  
  
 class\_<gomotion::JointState >("JointState")  
    .def\_readwrite("header",&sensor\_msgs::JointState\_<std::allocator<void> >::header)  
    .def\_readwrite("name",&sensor\_msgs::JointState\_<std::allocator<void> >::name)  
    .def\_readwrite("position",&sensor\_msgs::JointState\_<std::allocator<void> >::position)  
    .def\_readwrite("velocity",&sensor\_msgs::JointState\_<std::allocator<void> >::velocity)  
    .def\_readwrite("effort",&sensor\_msgs::JointState\_<std::allocator<void> >::effort)  
   ;  
  
class\_<GoTraj>("GoTraj", init<>())  
.def("Init",&GoTraj::Init )  
.def("InitJoints",&GoTraj::InitJoints )  
.def("IsDone",&GoTraj::IsDone, "Is motion done that is is queue empty")  
.def("AppendJoints",&GoTraj::AppendJoints, "Append another set of joints to motion queue ")  
.def("NextJoints",&GoTraj::NextJoints, "Get next joints on motioni queue");

### Problem: Checking whether a C++ to python converter has already been registered

From: <http://stackoverflow.com/questions/9888289/checking-whether-a-converter-has-already-been-registered#13017303>

#include <boost/python/converter/registry.hpp>

boost::python::type\_info info = boost::python::type\_id<YourType>();

boost::python::converter::registration\* reg = boost::python::converter::registry::query(info);

if (reg == NULL)

{

//registry YourType

}

Unfortunately, could not figure out how to create a Python object with embedding C++ class in BOOST\_PYTHON\_MODULE.

### Problem: The Interweb was scoured for helpful web sites on boost python.

<http://boost.2283326.n4.nabble.com/boost-python-how-to-pass-by-reference-pointer-to-python-overriden-class-functions-td3551072.html>

<http://effbot.org/pyfaq/what-is-init-py-used-for.htm>

<http://grokbase.com/t/python/cplusplus-sig/02bjg3ny0c/c-sig-passing-c-arrays-as-arguments>

<http://ompl.kavrakilab.org/python.html>

<http://shitohichiumaya.blogspot.com/2010/08/boostpython-how-to-pass-python-object.html>

<http://strattonbrazil.blogspot.com/2011/09/adding-python-support-using-boostpython.html>

<http://thejosephturner.com/blog/2011/06/15/embedding-python-in-c-applications-with-boostpython-part-2/>

<http://www.boost.org/doc/libs/1_43_0/libs/parameter/doc/html/python.html>

<http://www.boost.org/doc/libs/1_55_0/libs/python/doc/v2/faq.html>

<http://www.boost.org/doc/libs/1_63_0/libs/python/doc/html/tutorial/tutorial/functions.html>

<https://en.wikibooks.org/wiki/Python_Programming/Extending_with_C%2B%2B>

<https://github.com/boostorg/python/tree/develop/test>

<https://github.com/martwo/BoostNumpy>

<https://mail.python.org/pipermail/cplusplus-sig/2003-November/005837.html>

<https://media.readthedocs.org/pdf/pybind11/latest/pybind11.pdf>

<https://wiki.python.org/moin/boost.python/HowTo>

<https://wiki.python.org/moin/boost.python/HowTo>

<https://wiki.python.org/moin/boost.python/HowTo>  
<https://wiki.python.org/moin/boost.python/FunctionOverloading>  
 http://stackoverflow.com/questions/37818167/boost-python-add-bindings-to-existing-pyobject-for-exception-handling   
<http://stackoverflow.com/questions/3205561/binding-generic-c-libraries-to-python-with-boost-python#3208972>  
<http://imicrov.com/small-tech/python/export-c-template-class-to-python-boost>  <http://stackoverflow.com/questions/7805404/create-a-boostpythonobject-from-a-noncopyable-instance>  
https://misspent.wordpress.com/2009/09/27/how-to-write-boost-python-converters/  
https://blind.guru/tag/boostpython.html   
https://sites.google.com/site/alexeyvakimov/mini-tutorials/programming-boost-python-c  
<http://stackoverflow.com/questions/10701514/how-to-return-numpy-array-from-boostpython>  
<https://mail.python.org/pipermail/cplusplus-sig/2008-April/013167.html>  
<https://www.gamedev.net/topic/539148-integrate-python-using-boostpython/>  
<http://pyengr.readthedocs.io/en/latest/inter/bpy/>

<https://wiki.python.org/moin/boost.python/Inheritance>

<https://wiki.python.org/moin/boost.python/StlContainers>

<https://www.quora.com/How-do-I-convert-C++-vector-to-NumPy-array-using-Boost-Python>

### Problem: List of Fixmes

Why can't the python file be anywhere, not just in the ROS devel location?