Exemplary usage of ATF and IPDE

Grant Agreement No: 609206
Factory-in-a-Day
Author: Jonathan Hechtbauer

Pick & Place use-case

The EU project *Factory-in-a-Day* focuses on Plug & Work robots with the goal to remove the primary obstacles for robot automation, which are time and cost for installation. This document demonstrates two emerged tools, *ATF* and *IPDE*, in a Pick & Place use-case.

As shown in Image 1, the scenario consists of an industrial robot arm mounted on a table and a 3D camera (not shown) mounted at the left structure to detect objects lying in a bin. In the beginning the camera scans the bin with its content and calculates an adequate grasping pose, which is then published into the ROS system. The motion planning framework *MoveIt*!, which integrates the Open Motion Planning Library (*OMPL*), is used to generate the Pick & Place trajectory for the robot arm. Moreover, a valve is controlled via the ROS system to grasp and release the target object with a vacuum gripper.

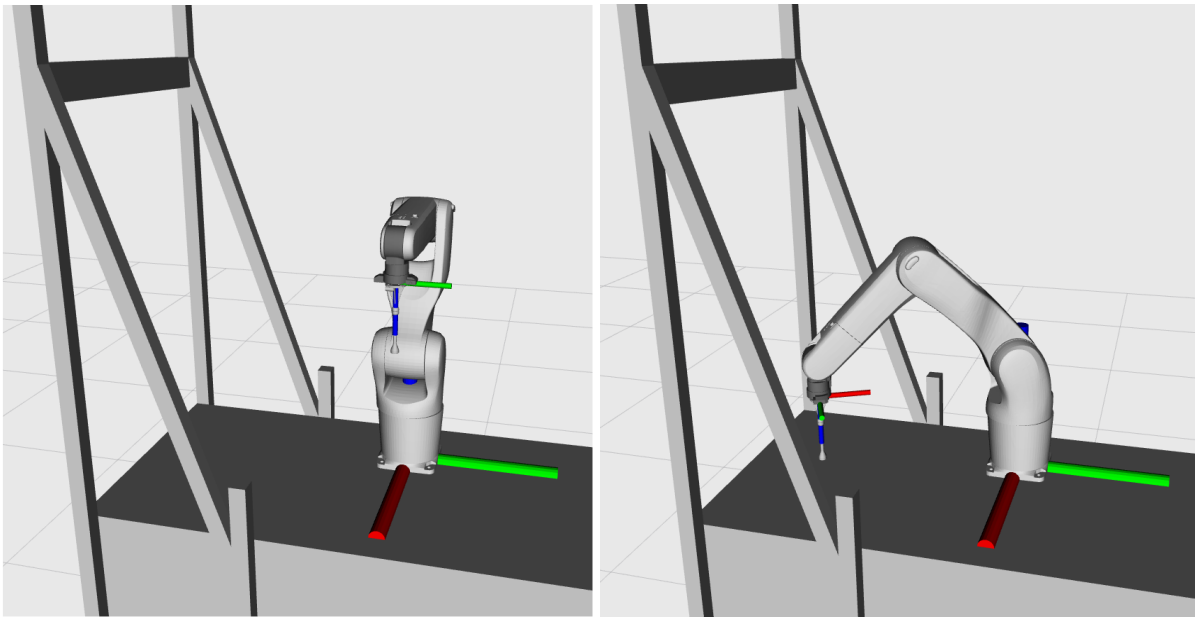

Image 1: Visualizations with RVIZ of a Pick & Place application.

In the following, the use-case is addressed to exemplarily highlight the benefits of first the *ATF* and afterwards the *IPDE* tool.

Automated Test Framework (ATF)

https://github.com/ipa-fmw/atf

Introduction

A ROS application usually consists of several components that run distributed over the system. Thus, the performance of a test scenario is on the one hand influenced by the composition of these components. On the other hand, an individual component can vary through parameters or deployment configurations. This results to a total set of test cases, which increase exponentially with existing variations. The *ATF* tool assists to automatically test all possible constellations. Besides, it provides additional performance metrics to evaluate the application. As such, there are common metrics available to measure e.g. duration, distance and frequency. Further individual ones can be created by the developer. This can be used to either do benchmarking of objective performance indicators compare to the different variations of the application. Furthermore, there is also the possibility to define ideal values bounded by a thresholds, which can be used for Continuous Integration testing. By checking that the result is not out-barrier, the developer can assure the quality for runtime behaviour. The tool also supports simulated applications e.g. with Gazebo as well as real hardware.

How-to

1) Install ATF

The *ATF* package has to be installed as any other ROS package. A detailed intruction can be found at https://github.com/ipa-fmw/atf/blob/master/doc/Installation.md.

2) Create a ATF test package

The current workflow intend the developer to have a dedicated ROS package for the examined application. The structure is standardized and good examples are provided in the *atf_test_app* repository (https://github.com/ipa-fmw/atf_test_apps). Besides the build description (CMakeList.txt) and the package information (package.xml) the package should contain the folders depicted in *Listing 1*.

```
package/
├── launch/application.launch
├── scripts/application.py
├── config/...
├── package.xml
└── CMakeList.txt
```

Listing 1: Overview of a ATF test package.

3) Define test cases

The ROS application, described in the first paragraph, consists of a multitude of components. Because of the inherent modularity, the developer is able to compose different variations. The ATF tool distinguishes between three different sources (see *Listing 2*). One

example is to evaluate different type of robots. One can also examine different robot configurations of the same model e.g. by tuning parameters of the motion configuration. Moreover, there is the possibility to simulate the robot in different worlds, which leads to a set of test environments. One more set of variation is related to the examined performance indicators, which is denoted as *test_configs*. In the Pick & Place use-case, *ATF* is used to benchmark the application regarding the execution duration and the traveled path length of the end effector.

```
config/
├── robot_envs/...
├── robots/...
├── test_configs/...
├── test_generation_config.yaml
└── test_suites.yaml
```

Listing 2: Overview of the configurations.

Finally, the user can specify which constellations of variations are of interest and generate a matrix of test suites. There is also the option to define a certain number of repetitions, in order to eliminate stochastic uncertainty.

4) Define test blocks of the application

The next step is to structure the sequence of the application and define the test blocks to be evaluated. They have to be declared in the execution function of the main *application* script. The file is located in the package's *scripts* folder and has to follow a specified format. Detailed examples can be found in the *atf_tes_apps* repository. In the provided test scenario the examined blocks are the total run, the pick and the place sequences and the one depicted as *pick*. In the latter sequence the robot moves from the approach to a dynamically calculated picking pose. The code snippet in *Listing 3* points out the usage of the ATF class for generating the test blocks. There are functions to start and end blocks as well as one to shutdown the whole test run.

```
...
 def execute(self):
     self.atf.start("total")
     self.atf.start("pick_sequence")
     ...
     self.atf.start("pick")
     ...
     self.atf.stop("pick")
     self.atf.stop("pick_sequence")
     self.atf.start("place_sequence")
     ...
     self.atf.stop("place_sequence")
     self.atf.stop("total")
     self.atf.shutdown()
 ...
```

Listing 3: Execution function with defined test blocks.

5) Run Tests

The last step is to define all the components that are necessary to run the application. As usual for ROS, these are listed in the launch file. The use case requires e.g. nodes related to hardware driver and ones for supportive tasks like the motion planning. The documentation *Using the ATF* (https://github.com/ipa-fmw/atf/blob/master/doc/Examples.md) describes further how to build and run the tests.

Results

The testing of the application is executed in the simulator *Gazebo* and with five repetitions. As mentioned previously, the execution duration and the traveled path length of the end effector are selected to benchmark the application. The *Images 2 and 3* show the measurements of the respective key performances indicators. The plots are generated by the interactive presenter of *ATF*, which also provides exact values.

The result reveals that the total run has an average duration of *13,7 sec* (*0,7 sec* deviation) and the end effector of the robot travels a path with the average length of *3,43 m* (*0,03 m* deviation). For the *pick* movement from the approach to the pick pose the execution takes *1.23 sec* (0,07 *sec deviation*) and *0,358 m* (*0,003 m* deviation). The minor deviation of the indicators is due to the randomized motion planners of *OMPL*. The result provides a better understanding of the application's performance. However, the main advantage is to compare this scenario with other variations. This can be used to benchmark new updates of the application or to investigate different configurations.

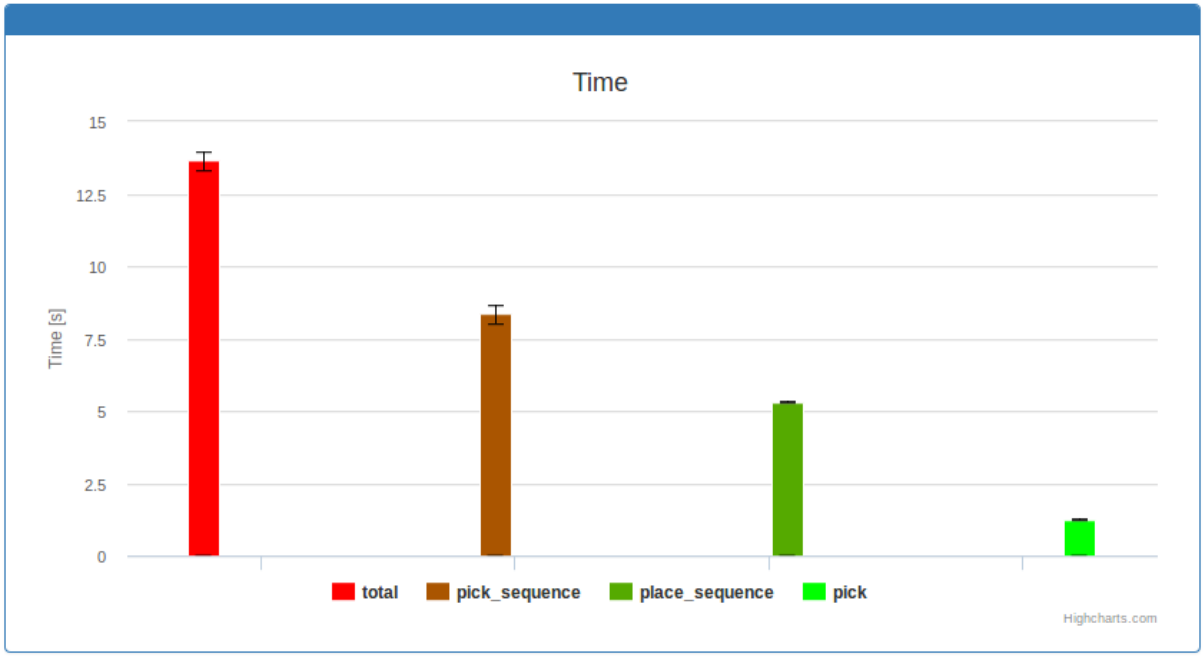

Image 2: ATF evaluation of a Pick & Place application regarding the execution time.

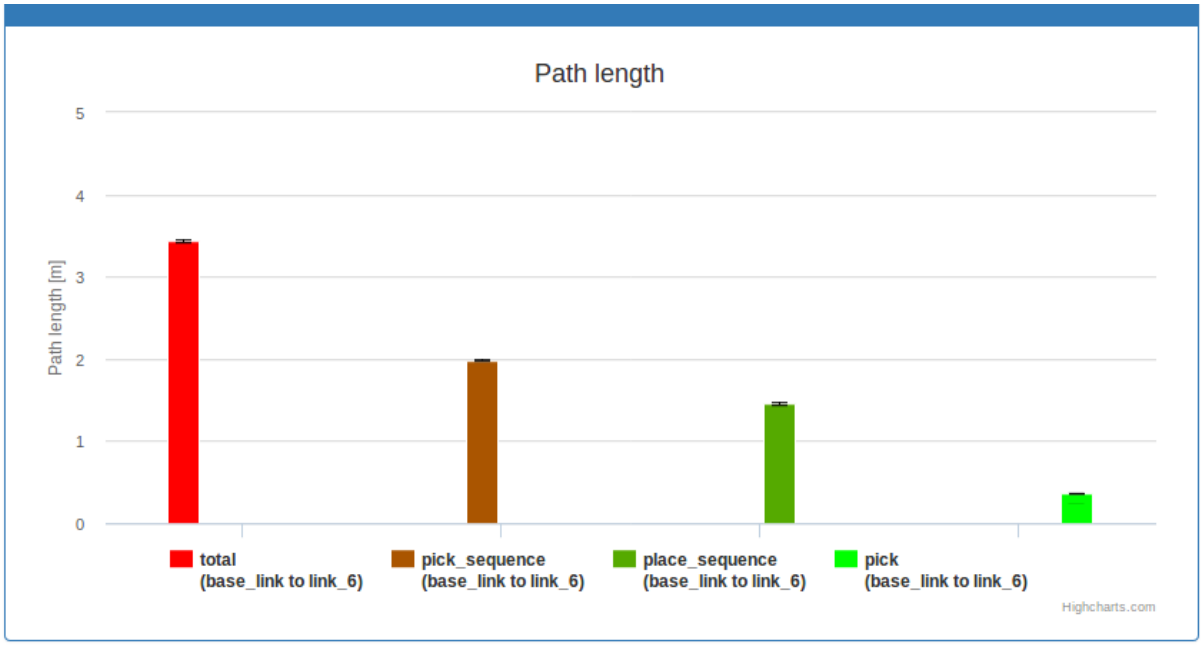

Image 3: ATF evaluation of a Pick & Place application regarding the traveled path length.

As mentioned in the introduction, *ATF* can also be used for Continuous Integration testing. In this use-case one has to define a *groundtruth* and a *groundtruth_epsilon* for specific metrics and test blocks. This basically checks after the evaluation, if the actual value is within a certain threshold of the ideal groundtruth value (see *Image 4*). The result of this test is boolean and can e.g. be integrated into the workflow of using Travis for *Continuous Integration*. Thus, the developer can define tests, which are executed every time a new commit is sent to the project's Github repository. This ensures the quality of the code and prevents the application to be broken by new updates.

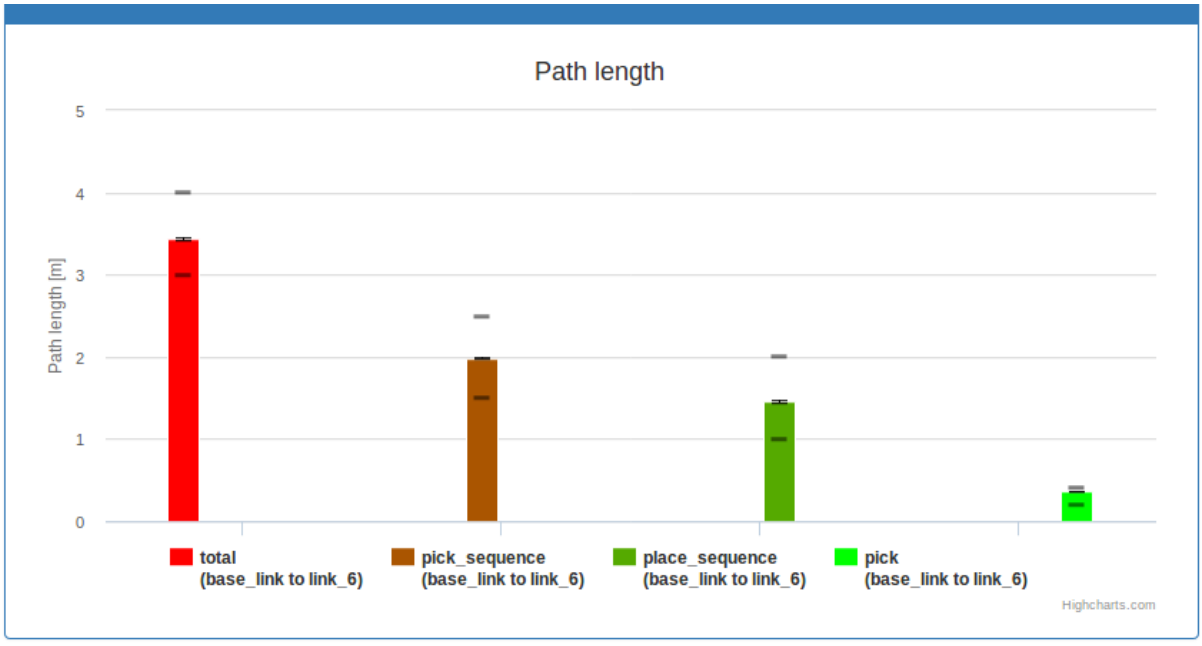

Image 4: ATF testing of a Pick & Place application regarding the traveled path length.

Integration Platform and Deployment Environment (IPDE)

https://github.com/ipa-mdl/ipde_utils

Introduction

The ROS developer workflow of using an application on a fresh computer commonly consists of several steps, which are listed in the following:

1) Install ROS environment
2) Set-up so-called catkin workspace
3) Download the desired ROS software application e.g. from source via Github.
4) Install further required packages by resolving the application's dependencies
5) Compile the catkin workspace, which contains the source files

The tool IPDE simplifies this process by automating the deployment of ROS applications. It utilizes Docker, a widely used software container engine, which offers the possibility to deploy isolated self contained application images onto a target environment. An *IPDE* python server has been developed, which handles the different steps of the deployment workflow. This server can be installed on any Docker-ready computer. Subsequently, one can easily deploy any desired application, into a secure session, even from a remote computer. Furthermore, there is the possibility to have multiple sessions of runtime applications and thus, it is very easy to switch between different configurations.

How-to

1) Install deployment backend

In order to make use of the tool's deployment environment on a the fresh machine one has to install Docker, download the *IPDE* source folder and in there start the *IPDE* backend by the command

```
$ docker-compose up -d
```

This process has to be done once. Afterwards, the host system is fully set-up and can be used as a target platform for any desired use-case. Applied on the Pick & Place application, the prepared computer is connected to the robot arm and the 3D camera.

2) Set the target IP address

The server of *IPDE* enables to easily deploy the desired ROS application from remote. Any machine, which is connected to the network, can be used for this purpose. In order to find the server, an URI with the target IP address (e.g. 192.0.2.1), was to be defined

```
$ export IPDE_SESSIONS_URI=http://192.0.2.1:8000/sessions/
```

3) Deploy and start application

As mentioned previously, the use-case requires multiple components to be installed. The IPDE tool makes the process to deploy and start the ROS application as simple as executing one script. The high level command e.g. for the demonstrated use-case is

```
$ ./launch.sh demo smartpick_demo demo.launch
 /home/user/catkin_ws/src/required_local_packages
```

Which provides the following arguments:

- `demo` specifies a label name of the session
- `smartpick_demo` is the desired ROS package name. The catkin workspace must contain the package and must be sourced to retrieve the path of the package. Alternatively, one could also provide the full path to the package.
- `demo.launch` is the desired launch file inside the package (which should be executed automatically after the deployment)
- `/home/user/catkin_ws/src/ local_packages` are the additional local packages which are necessary to build and run the application.

Further functionalities

The application will be running until it gets stopped explicitly. Likewise, the session can also be restarted

```
$ ./stop.sh demo
```

```
$ ./start.sh demo.
```

One can also remove the session and its data completely e.g. to deploy a fresh version or to clean-up:

```
$ ./rm.sh demo
```

Moreover, the user can create further sessions by choosing a different label names to e.g. run a second demo scenario that uses a different camera

```
$ ./launch.sh demo_kinect smartpick_demo demo_kinect.launch
 /home/user/catkin_ws/src/required_local_packages
```

Only one session can be running at a time. The list of available sessions can be retrieved with

```
$ ./list.sh.
```

Thus, the scripts of *IPDE* for stopping and starting of sessions make it very easy to switch between different use-cases. Just as running applications the developer can also run unit tests for the application. It is even possible to deploy the defined ATF tests, which is exemplarily shown in the following command

```
$ ./testing.sh test smartpick_demo_atf
 /home/user/catkin_ws/src/required_local_packages
 /home/user/catkin_ws/src/atf
```

One possible outlook is to design a graphical web interface for executing the shell scripts to make it more intuitive to operate the *IPDE* tool and provide the possibility to control the sessions from a tablet.