The coordination_oru Java Library

A Framework for Multi-Robot Motion Planning, Coordination and Control

Federico Pecora
Center for Applied Autonomous Sensor Systems, Örebro Unievrsity
federico.pecora@oru.se

Website:	https://federicopecora.github.io/coordination_oru
GitHub repo:	https://github.com/FedericoPecora/coordination_oru
Stable release:	0.4.2

1 Summary

For practitioners. This software API provides a solution to the problem of deploying fleets of autonomous robots in generic environments. The aim of the method is to minimize deployment costs, which is achieved by minimizing the need to re-engineer the environment, and making as few assumptions as possible on the robots and how their low-level controllers are implemented. Most notably, the approach is appropriate for heterogeneous fleets; it makes no assumptions on the process that decides goals, facilitating integration with existing workflow management systems; and guarantees that robots will reach their destinations, and that they will do so safely. The method is shown to scale well to several dozens of robots. An implementation of the approach is provided as an open source library which does not depend on a particular robot middleware.

For academics. A standing challenge in multi-robot motion planning and coordination is to safely yet efficiently coordinate robot motions while accounting for uncertainties and contingencies. The challenge is rendered harder by the fact that robots may be heterogeneous and that their goals may be posted asynchronously. Most existing approaches require constraints on the infrastructure, unrealistic assumptions on robot models, or specific forms of motion planning. The coordination_oru framework provides a centralized, loosely-coupled supervisory controller that overcomes these limitations. The approach responds to newly-posed constraints and uncertainties during trajectory execution, ensuring at all times that planned robot trajectories remain kino-dynamically feasible, that the fleet is in a safe state, and that there are no deadlocks or livelocks. This is achieved without the need for hand-coded rules, fixed robot priorities, or environment modification. All but the most basic assumptions are made on robot models and on the environment, and algorithm is proved both empirically and formally to guarantee safety and liveness.

2 Overview

This software implements an online coordination method for multiple robots. Its main features are:

- Goals can be posted and paths computed online;
- Precedences are inferred online, accounting for robot dynamics via provided dynamic models;
- Very few assumptions are made on robot controllers;
- The coordination method is not specific to a particular motion planning technique.

The software includes a basic 2D robot simulation and a simple built-in motion planner (which depends on the OMPL [9] and MRPT [2] libraries). A separate interface package [6] is provided to enable the use of this software in conjunction with ROS [8] and the navigation_oru stack [1] to obtain a fully implemented stack for multi-robot coordination and motion planning.

3 The Algorithm

The algorithm provided by this implementation is detailed in [7], and the corresponding talk at ICAPS 2018 is available on YouTube[1]. The approach makes very few assumptions on robot controllers, and can be used with any motion

[1] Please see https://www.youtube.com/watch?v=XVSAqH3gt_s.

planning method for computing kinematically-feasible paths. Coordination is seen as a high-level control scheme for the entire fleet. Heuristics are used to update precedences of robots through critical sections while the fleet is in motion, and the dynamic feasibility of precedences is guaranteed via the inclusion of user-definable models of robot dynamics. Several examples with simulated and real robots are shown on the Semantic Robots YouTube channel[2].

The coordination method is based on the trajectory envelope representation provided by the Meta-CSP framework [5]. This representation is detailed in [4].

In short, a trajectory envelope is a set of spatio-temporal constraints on a robot's trajectory. A trajectory envelope spans over a path, which is a sequence of poses $\langle p_1, \ldots, p_n \rangle$. In the current implementation, the spatial constraints defining a trajectory envelope are computed as the sweep of the robot's footprint over the path.

4 Tutorials

The approach is discussed in more detail in the tutorial on Integrated Motion Planning, Coordination and Control for Fleets of Mobile Robots, given at the 2018 International Conference on Automated Planning and Scheduling (ICAPS) by F. Pecora and M. Mansouri. Slides and source code of the tutorial are available online[3].

5 Using the `coordination_oru` library

A simple way to use the `coordination_oru` API as a library in your Java project is to use Gradle and Jitpack. To do so, simply include the following in the repositories section of your `build.gradle` file:

```
repositories {
  jcenter()
  maven { url 'https://jitpack.io' }
  maven { url 'https://github.com/rosjava/rosjava_mvn_repo/raw/master/' }
}
```

and add the following compile-time dependency in the dependencies section of your `build.gradle` file:

```
dependencies {
    compile 'com.github.FedericoPecora:coordination_oru:0.4.2'
}
```

You may want to check the current stable release of the framework and change it accordingly (it was 0.4.2 at the time of writing).

6 Installation from Source

Alternatively, you can install the framework from source. To do so, clone the repository available on Github and compile the source code with gradle (redistributable included):

```
$ git clone https://github.com/FedericoPecora/coordination_oru.git
$ cd coordination_oru
$ ./gradlew install
```

7 Running an example

A number of examples are provided. Issue the following command from the source code root directory for instructions on how to run the examples:

```
$ ./gradlew run
```

In the example `TestTrajectoryEnvelopeCoordinatorThreeRobots`, missions are continuously posted for three robots to reach locations along intersecting paths. The paths are stored in files provided in the `paths` directory. The poses of locations and pointers to relevant path files between locations are stored in the self-explanatory `paths/test_poses_and_path_data.txt` file.

[2] Please see `http://www.youtube.com/watch?v=jCgrCVWf8sE`.

[3] Please see `https://gitsvn-nt.oru.se/fopa/coordination-tutorial-src-ICAPS-2018`.

8 Visualizations

The API provides three visualization methods:

- `BrowserVisualization`: a browser-based visualization.
- `JTSDrawingPanelVisualization`: a Swing-based visualization.
- `RVizVisualization`: a visualization based on the ROS visualization tool RViz [3].

All three visualizations implement the abstract `FleetVisualization` class, which can be used as a basis to create your own visualization.

Most examples use the `BrowserVisualization`. The state of the fleet can be viewed from a browser at `http://localhost:8080`. Figure 8 shows this visualization for the `TestTrajectoryEnvelopeCoordinatorThreeRobots` example.

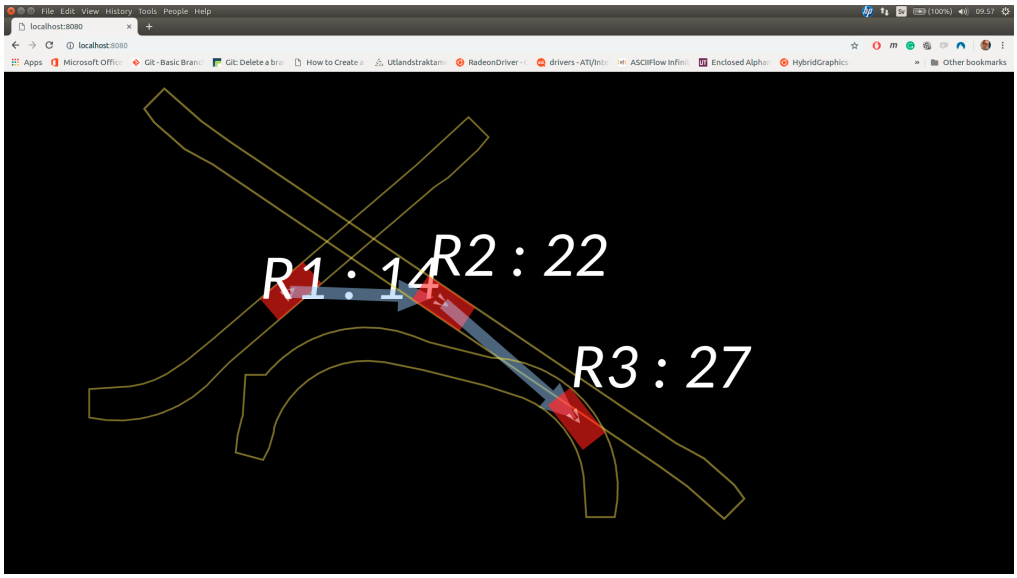

Figure 1: A screenshot of the browser-based GUI.

An arrow between two robots indicates that the source robot will yield to the target robot. Priorities are computed based on a heuristic (which can be provided by the user) and a forward model of robot dynamics (which can also be provided, and is assumed to be conservative [7]). The specific poses at which robots yield are also updated online, based on the current positions of robots and the intersecting areas of their trajectory envelopes (critical sections). This makes it possible to achieve “following” behavior, that is, the yielding pose of a robot is updated online while the "leading" robot drives.

The a Swing-based GUI provided by class `JTSDrawingPanelVisualization` is shown in Figure 8. This GUI allows to take screenshots in SVG, EPS and PDF formats by pressing the `s`, `e` and `p` keys, respectively (while focus is on the GUI window). Screenshots are saved in files named with a timestamp, e.g., `2017-08-13-11:13:17:528.svg`. Note that saving PDF and EPS files is computationally demanding and will temporarily interrupt the rendering of robot movements; SVG screenshots are saved much quicker.

The `RVizVisualization` visualization publishes visualization markers[4] that can be visualized in RViz. The class also provides the static method `writeRVizConfigFile(int ... robotIDs)` for writing an appropriate RViz confiuration file for a given set of robots. An example of the visualization is shown in Figure 8.

The visualization with least computational overhead is the `RVizVisualization`, and is recommended for fleets of many robots. The `BrowserVisualization` class serves an HTML page with a Javascript which communicates with the coordinator via websockets. Although rendering in this solution is less efficient than in RViz, the rendering occurs on the client platform (where the browser is running), so its computational overhead does not necessarily affect the coordination algorithm. The `JTSDrawingPanelVisualization` is rather slow and not recommended for fleets of more than a handful of robots, however it is practical (not requiring to start another process/program for visualization) and relatively well-tested.

[4]Please refer to the guide available at `http://wiki.ros.org/rviz/DisplayTypes/Marker`.

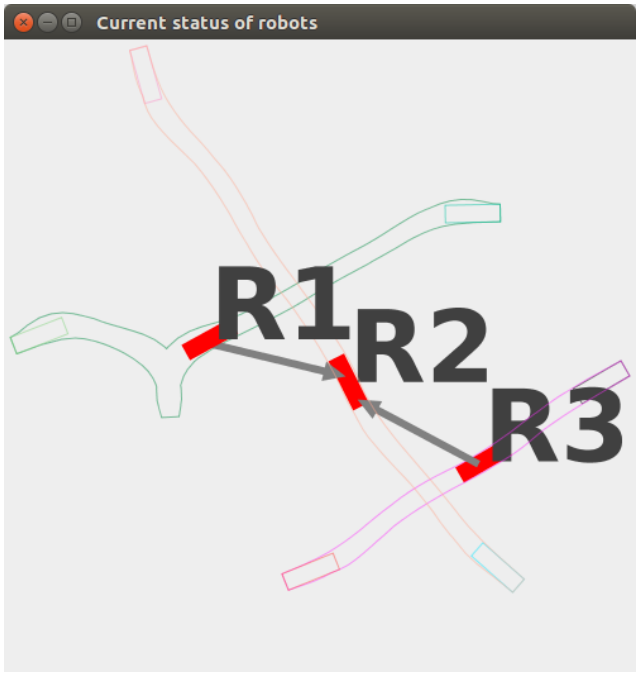

Figure 2: A screenshot of the Swing-based GUI.

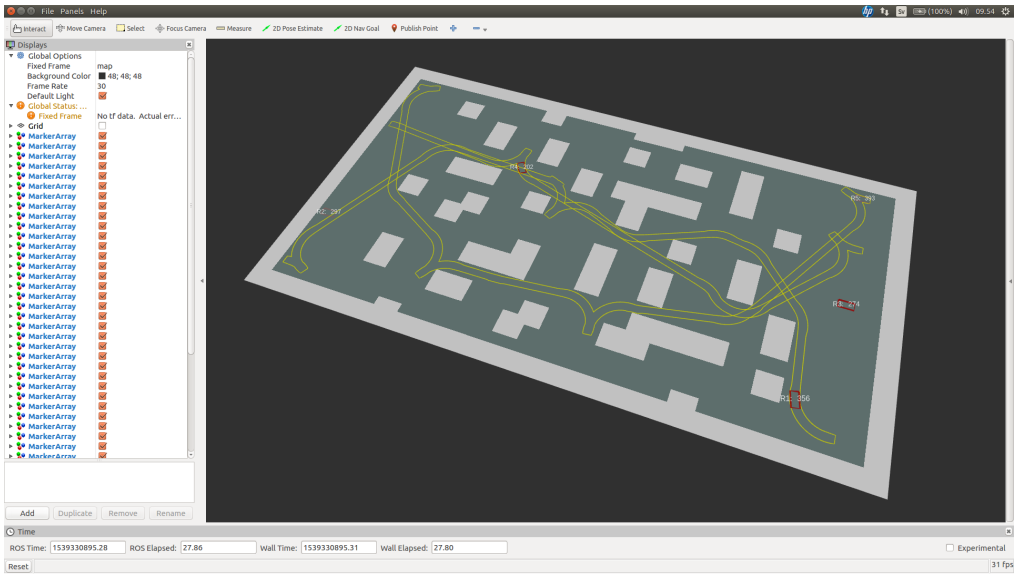

Figure 3: A screenshot of the RViz-based GUI.

9 Logging

More detailed information about execution is posted in the terminal and saved to log files. Log files can be inspected offline by running class `coordination_oru.util.BrowseLogs`, which opens a log browsing GUI (see Figure 9). Each panel in the GUI shows the output of one of the class instances that ran in the previous execution of the test program. Several of these classes are instantiated in separate threads, and messages produced concurrently are highlighted when the caret position in one of the panels is updated by the user. The key-bindings `Alt-<X>` and `Ctrl-Alt-<X>` can be used to quickly select panel `<X>` in the top and bottom pane, respectively.

10 The `SimpleReedsSheppCarPlanner` motion planner

A simple motion planner is provided for testing the coordination framework without the need for pre-computed path files. The planner can be used to obtain paths for robots with Reeds-Shepp kinematics (Dubin's car-like robots that can move both forwards and backwards), and is used in several of the included demos.

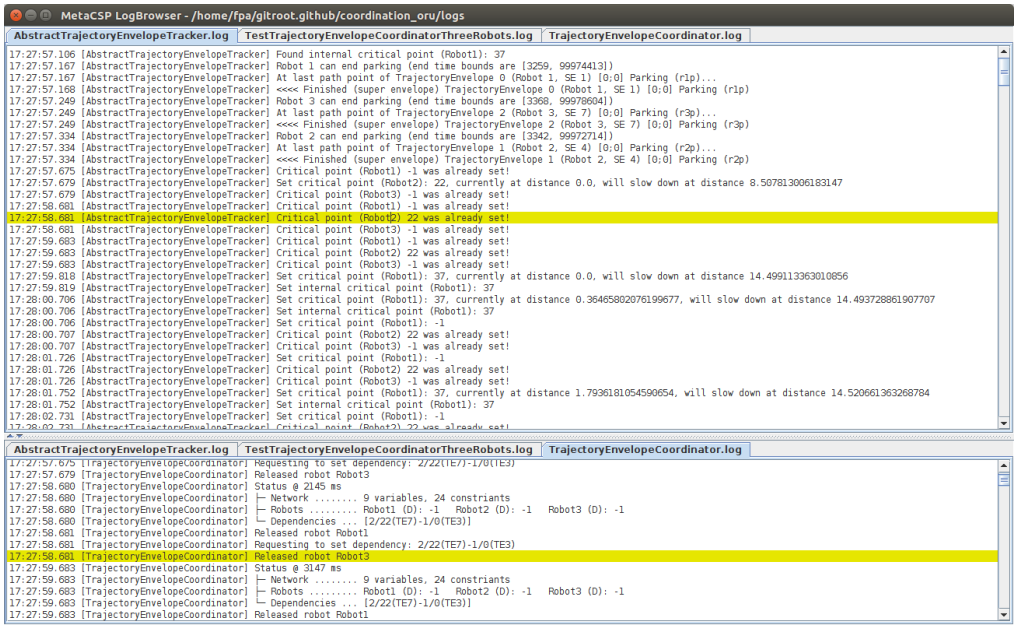

Figure 4: A screenshot of the log inspection interface.

The provided motion planner depends on the Open Motion Planning Library [9], and the Mobile Robot Programming Toolkit [2]. The motion planner and its Java interface are purposefully kept very simple. It performs rather poorly in terms of the quality of paths it returns, and is not suited for anything beyond simple examples. Please consider developing a more performing and principled integration with your motion planning software of choice, as done in the `coordination_oru_ros` package [6].

11 Installing the `SimpleReedsSheppCarPlanner` motion planner

Please install the OMPL and MRPT libraries. Both are present in the official Ubuntu repositories (tested on Ubuntu 16.04):

```
$ sudo apt-get install libompl-dev
$ sudo apt-get install mrpt-apps libmrpt-dev
```

Then, compile and install the `simplereedssheppcarplanner` shared library as follows:

```
$ cd coordination_oru/SimpleReedsSheppCarPlanner
$ cmake .
$ make
$ sudo make install
$ sudo ldconfig
```

This will install `libsimplereedssheppcarplanner.so` in your `/usr/local/lib` directory. A simple JNA-based Java interface to the library is provided in package `se.oru.coordination.coordination_oru.motionplanning`. The Java class `ReedsSheppCarPlanner` in the same package can be instantiated and used to obtain motions for robots with Reeds-Shepp kinematics.

12 Using the `SimpleReedsSheppCarPlanner` motion planner

A simple example showing how to invoke the motion planner is provided by class `TestReedsSheppCarPlanner` in package `se.oru.coordination.coordination_oru.motionplanning.tests`.

Most of the coordination examples make use of the motion planner (see screenshot in Figure 12). Issue command

```
$ ./gradlew run
```

for a list of all provided examples and instructions on how to run them, and see package

`se.oru.coordination.coordination_oru.tests`

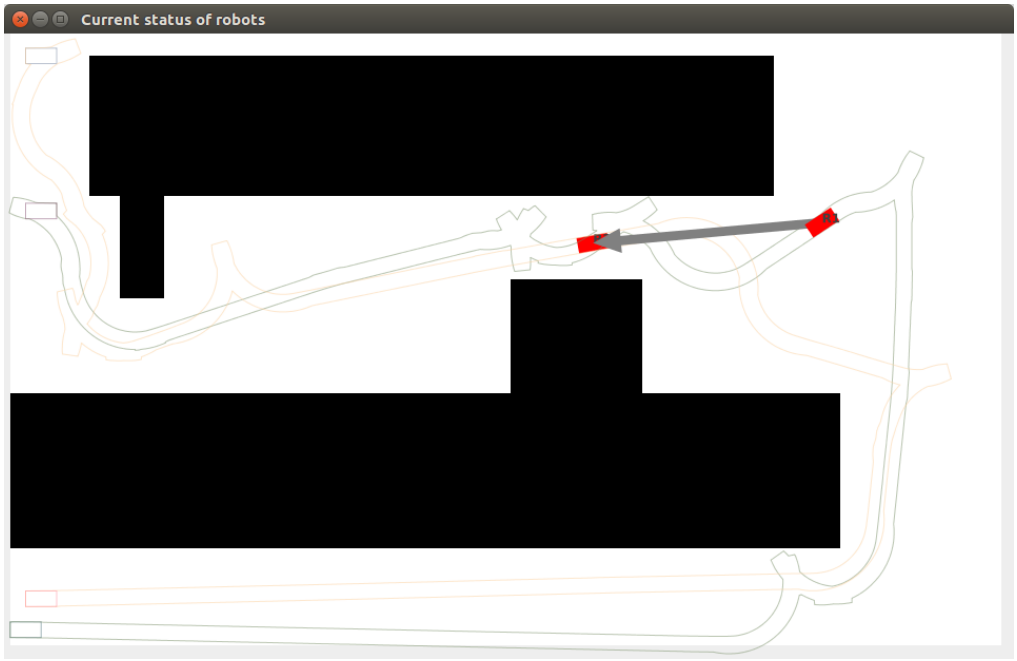

Figure 5: Example paths obtained via the `SimpleReedsSheppCarPlanner`.

13 Sponsors

This project is supported by

- The Semantic Robots Research Profile, funded by the Swedish Knowledge Foundation, see `http://semanticrobots.oru.se` for details.
- The ILIAD Project, funded by the EC H2020 Program, see `https://iliad-project.eu` for details.
- The iQMobility Project, funded by Vinnova, see `https://www.vinnova.se/en/p/iqmobility---automated-bus-service` for details.

14 License

References

[1] Henrik Andreasson. *The* `navigation_oru` *NAvigation Stack*, 2019. `https://github.com/OrebroUniversity/navigation_oru-release`.

[2] Jose-Luis Blanco-Claraco. *Mobile Robot Programming Toolkit (MRPT)*, 2019. `https://www.mrpt.org/`.

[3] Hyeong Ryeol Kam, Sung-Ho Lee, Taejung Park, and Chang-Hun Kim. Rviz: A toolkit for real domain data visualization. *Telecommun. Syst.*, 60(2):337–345, October 2015.

[4] F. Pecora, M. Cirillo, and D. Dimitrov. On mission-dependent coordination of multiple vehicles under spatial and temporal constraints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[5] Federico Pecora. *The Meta-CSP Framework: a Java API for Meta-Constraint Reasoning*, 2012. `http://github.com/FedericoPecora/meta-csp-framework`, also available on Maven Central.

[6] Federico Pecora. *The* `coordination_oru_ros` *Multi-Robot Coordination Package*, 2017. `http://github.com/FedericoPecora/coordination_oru_ros`.

[7] Federico Pecora, Henrik Andreasson, Masoumeh Mansouri, and Vilian Petkov. A loosely-coupled approach for multi-robot coordination, motion planning and control. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2018.

[8] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[9] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. `http://ompl.kavrakilab.org`.