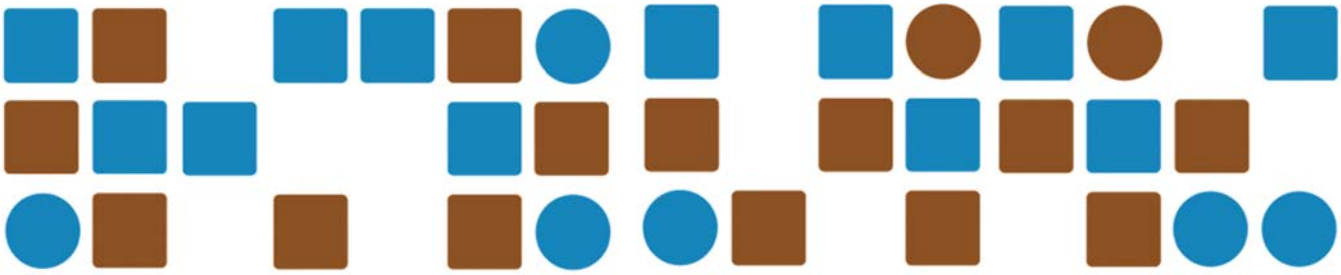


The Open Motion Planning Library (OMPL)

Integration into The Kautham Project

Planning and Implementation of Robotic Systems

Master's degree in Automatic Control and Robotics

Contents

1. OMPL overview

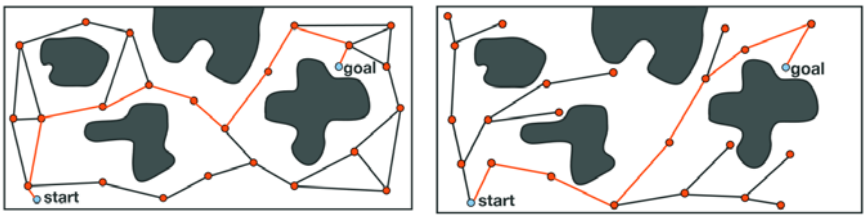
2. The Kautham Project Structure

3. Integration issues

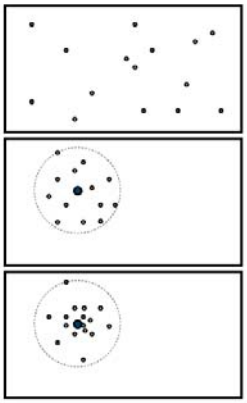
4. The benchmarking

1. OMPL overview

- OMPL contains implementations of many sampling-based algorithms such as PRM, RRT, and several variants of these planners.



- All the planners operate on very abstractly defined state spaces. Many commonly used state spaces are already implemented (e.g. $SE(2), SE(3), \mathbb{R}^n$)

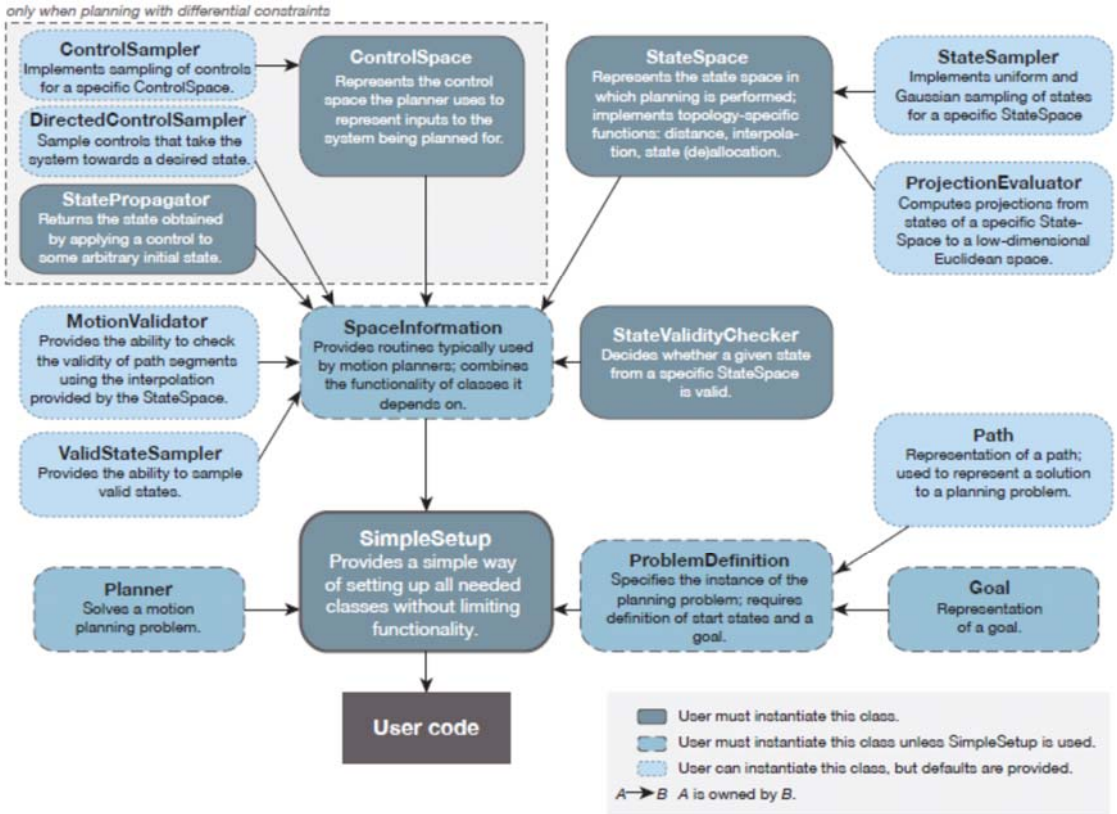


- For any state space, different state samplers can be used (e.g., uniform, Gaussian, obstacle based).

http://ompl.kavrakilab.org/

240AR032 - Planning and Implementation of Robotic Systems Jan Rosell

1. OMPL overview



Master's degree in Automatic Control and Robotics

1. OMPL overview

```
1  StateSpacePtr space(new SE3StateSpace());
2  // set the bounds (code omitted)
3
4  SimpleSetup ss(space);
5  // "isStateValid" is a user-supplied function
6  ss.setStateValidityChecker(isStateValid);
7
8  ScopedState<SE3StateSpace> start(space);
9  ScopedState<SE3StateSpace> goal(space);
10 // set the start & goal states to some values
11 // (code omitted)
12
13 ss.setStartAndGoalStates(start, goal);
14 bool solved = ss.solve(1.0);
15 if (solved)
16     setup.getSolutionPath().print(std::cout);
```


Contents

1. OMPL overview
2.The Kautham Project Structure
3. Integration issues
4. The benchmarking

2. The Kautham Project Structure

demos
doc
src
applications
external
planner
ioc
omplc
omplg
problem
sampling
util

The sources of the planners: it includes the abstract class planner and the planners libraries ioc, omplc and omplg.

ioc: Includes potential field planners computed on grids (NF1 and HF), local planners, a general PRM planner and several derived classes devoted to the planning of hand-arm robotic systems using PMDs.

Includes the following classes:
- iocPlanner, gridPlanner, HFPlanner, NF1Planner.
- LocalPlanner, LinearLocalPlanner, ConstLinearLocalPlanner
- prmPlanner
- prmPCAhandarmplanner, prmAUROhandarmplanner,

240AR032 - Planning and Implementation of Robotic Systems Jan Rosell

2. The Kautham Project Structure

demos
doc
src
applications
external
planner
ioc
omplc
omplg
problem
sampling
util

The sources of the planners: it includes the abstract class planner and the planners libraries ioc, omplc and omplg.

omplc: Contains the planners based on the control OMPL planners. Includes the following classes:

- General classes Eulerintegrator , KinematicRobotModel and omplcPlannerStatePropagator.
- The class omplcplanner that sets the solve function and the state space and its derived class omplcRRTplanner class, that sets the parameters for the OMPL RRT planner.
- The omplcRRTcarplanner and omplcRRTf16planner, derived from omplcRRTplanner , that particularize the planner to use the kinematic constraints of a car and of a f16 plane, coded in classes KinematicF16Model and KinematicCarModel that derive from KinematicRobotModel class.

Master's degree in Automatic Control and Robotics

2. The Kautham Project Structure

demos
doc
src
applications
external
planner
ioc
omplc
omplg
problem
sampling
util

The sources of the planners: it includes the abstract class planner and the planners libraries ioc, omplc and omplg.

omplg: Contains the planners based on the geometric OMPL planners. Includes the following classes:

- Abstract planner class (omplgplanner): Sets the state space and the solve function.
- Derived planner classes: omplESTplanner, omplKPIECEplanner, omplPRMplanner, omplRRTplanner, omplRRTConnectplanner, omplLazyRRTplanner omplRRTStarplanner, omplTRRTplanner, omplSBLplanner.
- omplValidityChecker

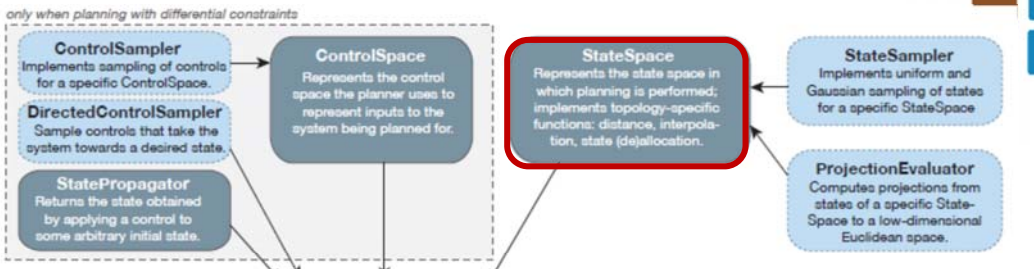
The PRM and the RRTStar planners has been reimplemented to be able to tune some parameters that are fixed in the OMPL implementations.

Contents

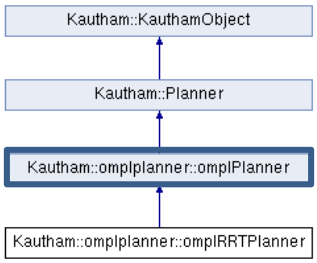
1. OMPL overview
2. The Kautham Project Structure
3. **Integration issues**
   1. **The State Space**
   2. **The Simple Setup**
   3. **The Validity Checker**
   4. **The Planner**
   5. **The Problem Definition**
4. The benchmarking

3. Integration

1. The State Space

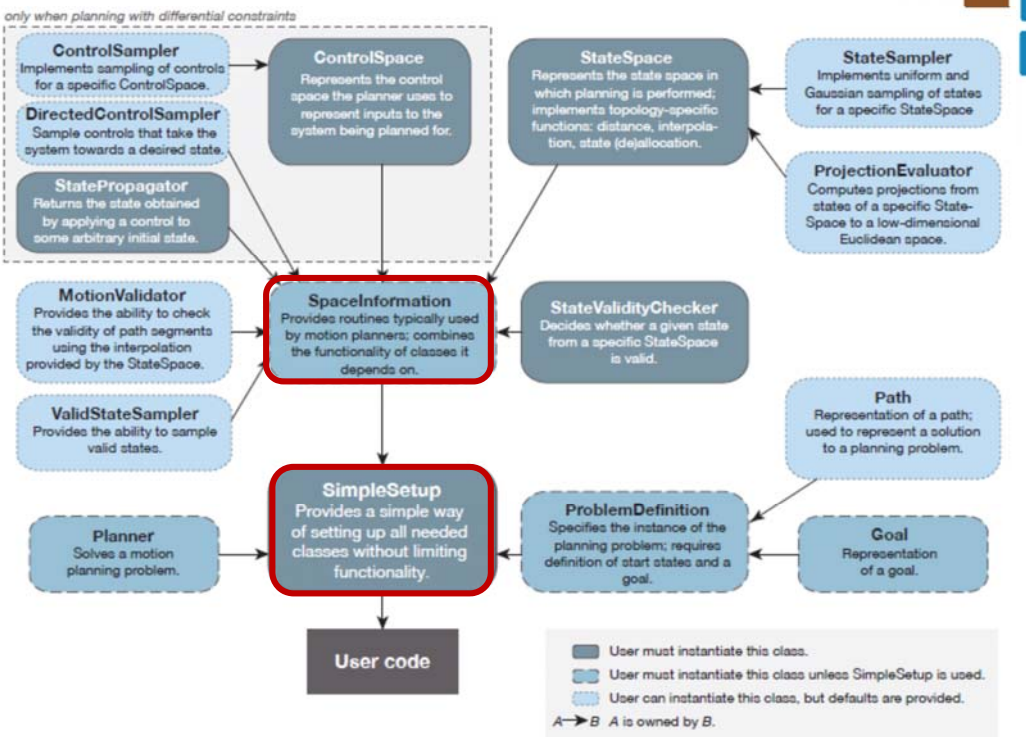


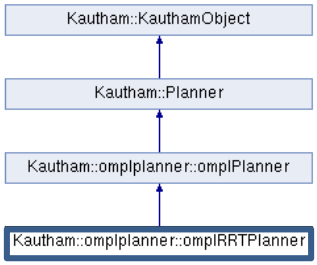

```
//! Constructor
omplPlanner::omplPlanner

//create the SE3 state space
spaceSE3[i] = ((ob::StateSpacePtr) new ob::SE3StateSpace());

//load to the compound state space of robot i
compoundspaceRob.push_back(spaceSE3[i]);
weightsRob.push_back(1);

//create the Rn state space
spaceRn[i] = ((ob::StateSpacePtr) new weigthedRealVectorStateSpace(nj));

//load to the compound state space of robot i
compoundspaceRob.push_back(spaceRn[i]);
weightsRob.push_back(1);

//the compound state space for robot i is (SE3xRn), and either SE3 or Rn may be missing
spaceRob[i] = ((ob::StateSpacePtr) new ob::CompoundStateSpace(compoundspaceRob,weightsRob));

//the state space for the set of robots. All the robots have the same weight.
space = ((ob::StateSpacePtr) new ob::CompoundStateSpace(spaceRob,weights));
```



3. Integration

2. The Simple Setup

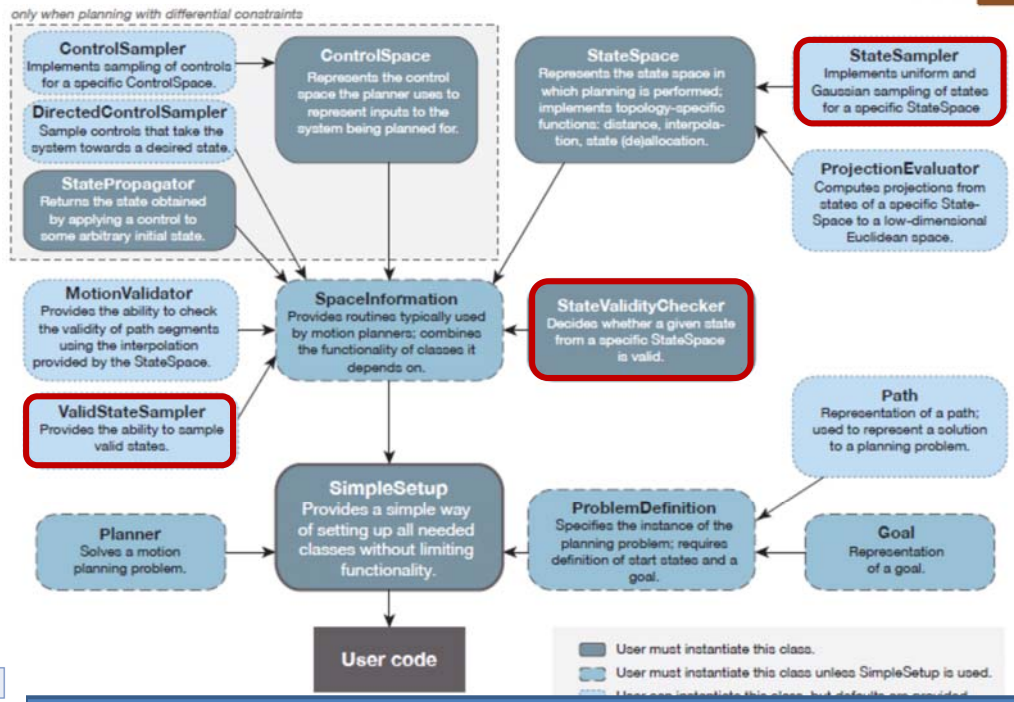


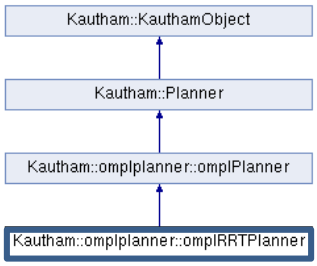



```
//! Constructor
omplRRTPlanner::omplRRTPlanner

//create simple setup
ss = ((og::SimpleSetupPtr) new og::SimpleSetup(space));
ob::SpaceInformationPtr si=ss->getSpaceInformation();
```

3. Integration

3. The Validity Checker





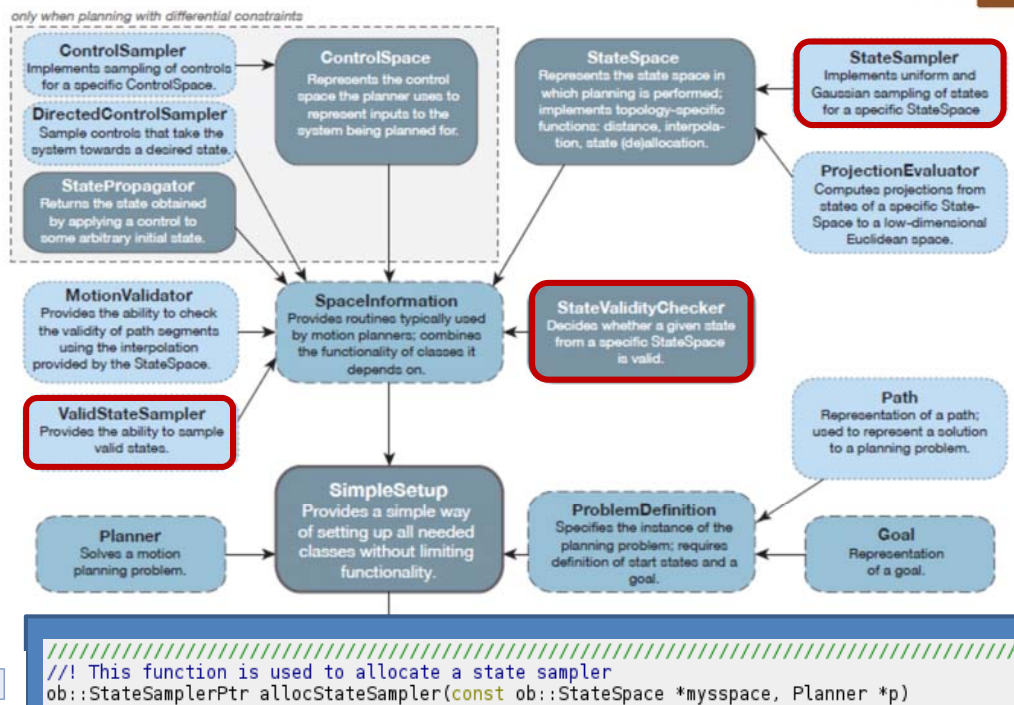

```
//! Constructor
omplRRTPlanner::omplRRTPlanner
```

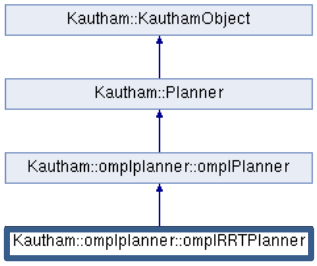
```
//set validity checker
ss->setStateValidityChecker(boost::bind(&omplplanner::isStateValid, si.get(), _1, (Planner*)this));
//alloc valid state sampler
si->setValidStateSamplerAllocator(boost::bind(&omplplanner::allocValidStateSampler, _1, (Planner*)this));
//alloc state sampler
space->setStateSamplerAllocator(boost::bind(&omplplanner::allocStateSampler, _1, (Planner*)this));
```


240AR032 - Planning and Implementation of Robotic Systems Jan Rosell

3. Integration

3. The Validity Checker

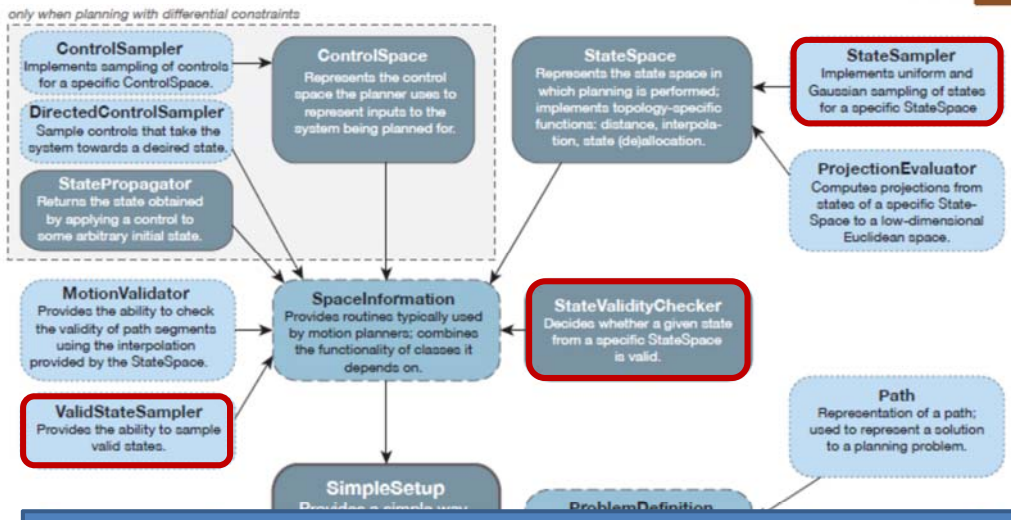




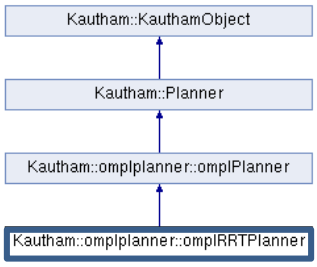

```
////////////////////////////////////////////////////////////////////////////////
//! This function is used to allocate a state sampler
ob::StateSamplerPtr allocStateSampler(const ob::StateSpace *mysspace, Planner *p)
{
    return ob::StateSamplerPtr(new KauthamStateSampler(mysspace, p));
}

////////////////////////////////////////////////////////////////////////////////
//! This function is used to allocate a valid state sampler
ob::ValidStateSamplerPtr allocValidStateSampler(const ob::SpaceInformation *si, Planner *p)
{
    return ob::ValidStateSamplerPtr(new KauthamValidStateSampler(si, p));
}
```

3. Integration

3. The Validity Checker

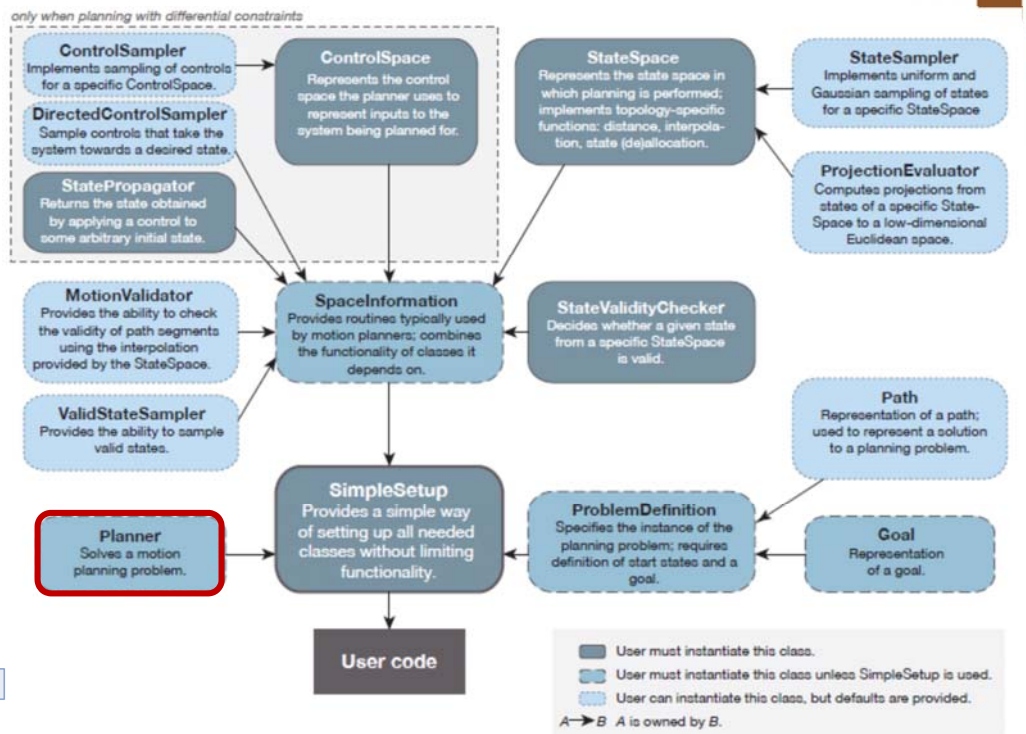


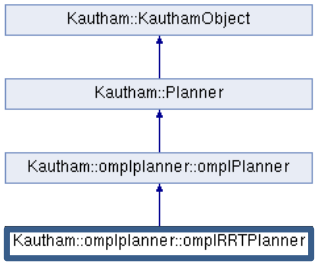

```
//! This function converts a state to a smp and tests if it is in collision or not
bool isStateValid(const ob::SpaceInformation *si, const ob::State *state, Planner *p)
{
    //verify bounds
    if(si->satisfiesBounds(state)==false)
        return false;
    //create sample
    int d = p->wkSpace()->getDimension();
    Sample *smp = new Sample(d);
    //copy the conf of the init smp. Needed to capture the home positions.
    smp->setMappedConf(p->initSamp()->getMappedConf());
    //load the RobConf of smp form the values of the ompl::state
    ((omplPlanner*)p)->omplState2smp(state,smp);
    //collision-check
    if( p->wkSpace()->collisionCheck(smp) )
        return false;
    return true;
}
```



3. Integration

4. The Planner

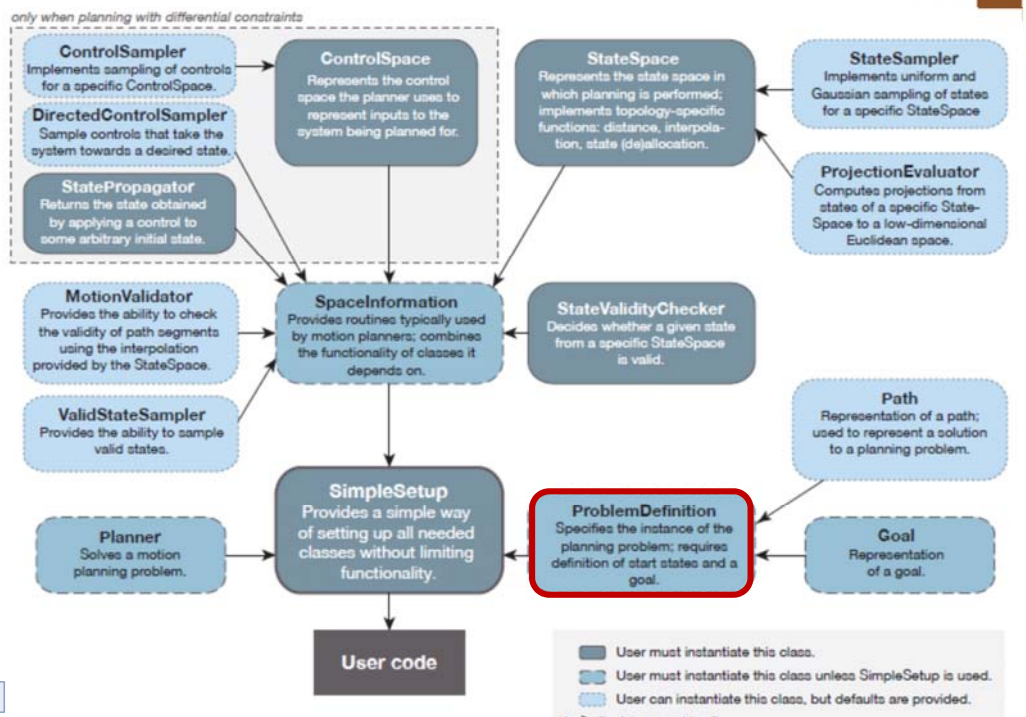


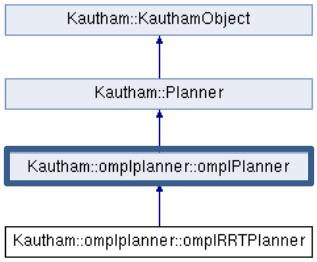



```
//! Constructor
omplRRTPlanner::omplRRTPlanner
    //create planner
    ob::PlannerPtr planner(new og::RRT(si));
    //set the planner
    ss->setPlanner(planner);
```

3. Integration

5. The Problem Definition






```
//! function to find a solution path
bool omplPlanner::trySolve()
    // set the start and goal states
    ss->setStartAndGoalStates(startompl, goalompl);
    ss->setup();
    ob::PlannerStatus solved = ss->solve(_planningTime);
```


Contents

1. OMPL overview
2. The Kautham Project Structure
3. Integration issues
4. **The benchmarking**

4. The benchmarking

OMPL has a benchmarking utility. It can be used from the KauthamConsole

a) Launch the benchmark using the KauthamConsole application as follows:

```
> ./KauthamConsole  –b absolute_path/benchmarking.xml
```

A file *result.log* is created with the data of the executed runs.

b) Visualize the results by creating a database file and the plots.

Copy the file *kautham_ompl_benchmark_statistics.py* to the folder containing the demo and cd to that folder.

```
> ./kautham_ompl_benchmark_statistics.py result.log –d result.db
> ./kautham_ompl_benchmark_statistics.py  –d result.db –p result.pdf
```