Optimizing MoveIt

Costs, Constraints and Betterments

October 19, 2023

Lars Henning Kayser
MoveIt Chief Architect
henningkayser@picknik.ai
Quick facts

about me

- **2018**
  - M.Sc. CS at University of Hamburg, TAMS robotics lab
  - Hired by PickNik after introduction at ROSCon Madrid

- **Since then**
  - 20+ clients, leading 6 projects
    - industrial, medical, construction, agriculture, logistics, …
    - primarily consulting and R&D, motion planning - MoveIt, C++
  - MoveIt ROS 2 migration, ROSin project (EU Horizon 2020)

- **Now: MoveIt Chief Architect (or Archeologist?)**
  - OSS Maintenance, internal R&D, TSC member
Optimizing MoveIt

01 **Inverse Kinematics**
Solving / Sampling / Optimizing

02 **Motion Planning**
Searching / Optimizing / Ranking

03 **Miscellaneous**
Projects / Python / Parameters / PRs
Example: Turtle Cleaning Robot

repo: https://github.com/henningkayser/roscon23_moveit

Problem: Find initial robot state to gently apply scrubber!
Solver Plugins

IKFast
repo: MoveIt

KDL
repo: MoveIt

trac_ik
repo: https://bitbucket.org/traclabs/trac_ik/src/rolling-devel/

bio_ik
repo: https://github.com/TAMS-Group/bio_ik
ros2: https://github.com/PickNikRobotics/bio_ik/tree/ros2

pick_ik
repo: https://github.com/PickNikRobotics/pick_ik

---

Configuration: kinematics.yaml

```yaml
manipulator:
  kinematics_solver: kdl_kinematics_plugin/KDLKinematicsPlugin
```

C++ Implementation

```cpp
// ... initialize:
//    RobotState target_state
//    JointModelGroup arm_group
//    PoseStamped turtle_pose

PoseStamped target_pose = turtle_pose;
target_pose.pose.position.z += turtle_radius;

target_state.setFromIK(arm_group, target_pose);
```
Very often

Sometimes
IK - Limitations

We don’t always want or need fully constrained target poses

- **Tools can often be applied with some tolerance**
  - Suction grippers, laser scanning, spin scrubbers …
- **We may compromise orientation accuracy for position accuracy**
  - Laser cutting, welding, assembly
- **Unstructured robot environments may require**
  - Additional safety margins, collision clearance
- **Reachability issues may lead to**
  - Joint flips, high failure rate, solutions near singularities or joint limits

We can increase the solution space using problem-specific constraints!
IK - Constraints

Constraints are rules that decide the binary validity of a state

- **Implementation types**
  - Threshold functions with target value and tolerance range
  - Bool functions that perform on non-gradient metrics

- **MoveIt supports**
  - Position, Orientation, Joint, Visibility constraints
  - Collisions, joint limits are implicitly constraining solutions
We can model the space of valid poses as single position constraint!

**PositionConstraint (moveit_msgs)**

- `std_msgs/Header header`: turtle pose frame
- `string link_name`: IK link
- `geometry_msgs/Vector3 target_point_offset`: IK frame offset, turtle radius along Z-axis
- `moveit_msgs/BoundingVolume constraint_region`: tiny primitive shape, sphere at center of turtle
- `float64 weight (unused for now)`
#include <moveit/constraint_samplers/constraint_sampler_manager.h>
#include <moveit/kinematic_constraints/utils.h>

// ... init std::string link_name, PlanningScene scene

geometry_msgs::msg::PointStamped target_point;
target_point.header = turtle_pose.header;
target_point.point = turtle_pose.pose.position;

geometry_msgs::msg::Point link_offset;
link_offset.z = turtle_radius;

using kc = kinematic_constraints;
auto constraints = kc::constructGoalConstraints(link_name, link_offset, target_point);

constraint_samplers::ConstraintSamplerManager sampler_manager;
auto goal_sampler =
    sampler_manager.selectSampler(scene, arm_group->getName(), constraints);

goal_sampler->sample(target_state);
Obviously, collision checks are not enabled here. We either have to reject a lot of samples (costly!) or add additional constraints.
What if we want quality criteria like …

1. reducing the joint distance from the current configuration
2. repeatable or at least similar solutions
3. preference for contact points near an ideal target

… and all that at the same time?
Plugin Implementation

- Thread-safe reimplementation of bio_ik
- Provides gradient descent (local) and memetic (global) optimization
- Built-in cost objectives
  - minimal displacement
  - center joints
  - avoid joint limits
- Supports dynamic parameter updates

Configuration: kinematics.yaml

```yaml
manipulator:
  kinematics_solver: pick_ik/PickIkPlugin
  mode: global  # global, local
  position_scale: 1.0  # factor for position distance cost
  rotation_scale: 0.5  # factor for rotation distance cost
  position_threshold: 0.005  # max allowed position cost
  orientation_threshold: 0.01  # max allowed orientation cost
  minimal_displacement_weight: 0.0  # minimize seed distance
  center_joints_weight: 0.0  # keep joint values centered
  avoid_joint_limits_weight: 0.0  # penalize states near limits
```

... and implements MoveIt’s new IK Cost function API!
Inject quality metrics into IK solver plugins

- IK callback for computing cost values for solver-internal samples
- Currently, only supported by pick_ik, bio_ik (ros2 PickNik fork)

```
// moveit_core/kinematics_base/.../kinematics_base.h, class KinematicsBase

using IKCostFn = std::function<double(const geometry_msgs::msg::Pose& target_pose, 
                                   const moveit::core::RobotState& sample_state, 
                                   const moveit::core::JointModelGroup* group, 
                                   const std::vector<double>& seed_positions)>;
```
// ...

kc::KinematicConstraintSet constraints_validator(robot_model);
constraints_validator.add(constraints, scene->getTransforms());

auto constraints_cost_fn = [&](const geometry_msgs::msg::Pose& /* target_pose */,
    const RobotState& sample_state,
    const JointModelGroup* /* group */,
    const std::vector<double>& /* seed_positions */)
{
    return constraints_validator.decide(sample_state).distance;
};

target_state.setFromIK(arm_group,
    target_pose,
    0.05, /* seconds timeout */
    GroupStateValidityCallback(),
    KinematicsQueryOptions(),
    constraints_cost_fn);
We still pass the initial target pose to the IK call
• Setting `return_approximate_solution` to `true` allows diverging from it
• `pick_ik` provides additional `approximate_*` parameters for tuning cost thresholds

```cpp
auto ik_options = kinematics::KinematicsQueryOptions();
ik_options.return_approximate_solution = true;

target_state.setFromIK(arm_group,
    target_pose,
    0.05,  /* seconds timeout */
    GroupStateValidityCallback(),
    ik_options,
    constraints_cost_fn);
```
IK - pick_ik Solutions

Same start state (20 solutions)

Random start state (20 solutions)
Advanced Use Cases

- **Cartesian Interpolation**
  
  MoveIt’s Cartesian Interpolator supports IK cost functions!

- **Visual Servoing**
  
  “local” modes are feasible for computing controller waypoints online (requires post-processing)

- **Collision Clearance**
  
  A collision distance check as cost function allows “pushing” the robot away from obstacles
Limitations

- **Conflicting Cost Terms**
  “Too many cooks…” - already position and orientation targets may conflict, produce offsets

- **Performance**
  cost functions need to be very efficient, otherwise solver time explodes

- **Weighting**
  Cost terms are balanced by weight. Tuning them may come close to “magic numbers”
Fully constrained IK can have undesired side effects
- restricts solution space too much
- can produce reachability issues, joint flips
- “bad” IK solutions can cause path sampling and motion planning issues

Constraints define some IK problems more elegantly
- increase the solution space
- enable trade offs between solution accuracy and quality criteria
- IK constraints can be sampled and filtered

Cost Functions allow optimizing quality metrics
- cost functions can be derived from constraints with distance metrics
- IK Solvers can optimize for multiple weighted cost functions at the same time
- Optimization can be global or local, depending on the problem
Motion Planning

Searching / Optimizing / Ranking
… is very difficult!

- OMPL used to sample states in **joint space** or **Cartesian space**
  - joint samples require validation = Rejection Sampling
  - Cartesian samples require IK
- Interpolated states need to be validated as well
- Best approach so far was using a search space (= constraint manifold) approximation
  - pre-computed database with valid states
  - [Sucan et al, IROS 2012](https://example.com/sucan-irostext)!
OMPL Constrained Planning

- New adapters to OMPL’s constrained planning framework ([Kingston, 2019](#))
- Projects sampled states into the constraint manifold using error Jacobian
  - Optimization on constraint derivative gradient
- Supports any OMPL planning algorithm
- Implemented for Position (BOX), Orientation, and Equality

STOMP

Stochastic Trajectory Optimization for Motion Planning

- Finds smooth collision free paths using probabilistic optimization
- Starts with an initial (maybe infeasible) guess
- The initial path is iteratively optimized by minimizing individual waypoint costs over randomized samples
- Advantages:
  - Cost function does not need to be differentiable
  - Can incorporate additional cost terms
Stochastic Trajectory Optimization for Motion Planning

- Finds smooth collision free paths using probabilistic optimization
- Starts with an initial (maybe infeasible) guess
- The initial path is iteratively optimized by minimizing individual waypoint costs over randomized samples
- Advantages:
  - Cost function does not need to be differentiable
  - Can incorporate additional cost terms

Complete reimplementation!
- C++ callbacks instead of plugins
  - noise, costs, filter, post conditions
  - supports arbitrary constraints
    - caveat: probably more useful for post processing OMPL if problem is challenging
- NOTE: cost function API similar to CostIKFn() is WIP!

Solution quality depends on the planning algorithm
-> Picking the best algorithm for a given problem is not intuitive

Even the “best” algorithm can fail
-> In this case we need a fallback planner

Approach
Run a Portfolio of Planners in parallel and pick the best!
Parallel Planning

Semi-autonomous choice of the most suitable planner for a given problem

- Customizable stopping criteria
- Customizable solution selection
- Good default but no “real” optimization
Miscellaneous

Projects / Python / Parameters / PRs
**Major Contributions**

- **GSoC: Python Bindings**
  - Peter David Fagan

- **GSoC: IK Benchmarking**
  - Mohamed Raessa & Sebastian Castro

- **GSoC: Servo Refactor**
  - Mohammed Ibrahim & Sebastian Castro

- **Iron Release**
  - Henning Kayser

- **Isaac Integration**
  - Marq Rasmussen & Jafar Abdi

- **Unifying Parameter Approach**
  - Tyler Weaver
generate_parameters_library

- declarative, validatable and (almost) self-documenting ROS 2 parameters
- repo: https://github.com/PickNikRobotics/generate_parameter_library

Visit Tyler Weaver’s talk “Parameters should be boring”!

tomorrow, 2:10 PM CST, “ROS Development” track
Python Bindings

moveit_py

2022 GSoC - Peter David Fagan

- Python bindings to MoveItCpp and moveit_core classes
- Goal: facilitate integrating and prototyping with other Python libraries

Tutorial


```python
rclpy.init()
logger = rclpy.logging.get_logger("moveit_py")

# instantiate MoveItPy instance and get planning component
robot = MoveItPy(node_name="moveit_py")
robot_arm = robot.get_planning_component("arm")
logger.info("MoveItPy instance created")

# set plan start and goal states using predefined states
robot_arm.set_start_state(configuration_name="ready")
robot_arm.set_goal_state(configuration_name="extended")

# plan to goal
arm_motion = robot_arm.plan()

# execute trajectory
if (arm_motion):
    robot.execute(arm_motion.trajectory)
```
Advanced Use Cases

Learning

Simulation

Parameter Tuning

Perception

Benchmarking

Data Visualization
Grow Community - We want YOU to start playing with this!

What use cases are you interested in?
Which Python library would you like to integrate?
What interfaces do you need for that?
Weekly Developer Standup
Tuesdays, 8:30AM Mountain Time

Monthly Working Group / MoveIt Maintainer Meeting
4th Thursday, 9AM Mountain Time

All contributors are welcome! Request an invite via henningkayser@picknik.ai
Get Involved!

Contribute to MoveIt
Review and file PRs
Engage in issue discussions
Join the meetings
Become a Core Contributor or Maintainer

Apply for GSoC 2024
12+ weeks focused programming
Mentored by MoveIt maintainers
Details will be shared end of 2023