Evolutionary Inverse Kinematics for ROS and MoveIt

Philipp Ruppel
May 30, 2017
Master Thesis
ROS, MoveIt

Robot Operating System

Motion planning framework

- Forward & inverse kinematics
- Perception
- Motion planning
MoveIt
Inverse Kinematics

End effect pose → joint angles
Forward Kinematics

Joint angles → end effect pose
Inverse Kinematics

End effect pose → joint angles
Inverse Kinematics

Multi-Goal IK (multiple eef, or generalized: arbitrary constraints on robot pose)
Inverse Kinematics
IK Solvers in MoveIt

- IK Plugin interface
  (kinematics::KinematicsBase)
- Analytic
  - Specialized solvers
  - ikfast
- Numeric
  - KDL
  - TRAC-IK
- Currently single-goal only!
Analytic Solution

- Manually solve equations
- Very fast at runtime
- Only practical for simple cases
- Solver is specific to a certain robot and goal configuration
- New robot or different goal types? - write new solver
Generates code automatically
(Generated solver is specific to a certain robot and goal configuration)
Only kinematic chains, only some goal types
Long compilation time
Doesn’t always work

"for 6D ik, there are still mechanisms it cannot solve"

6D = Translation (3D) + Orientation (3D)
KDL / Pseudo-Inverse Jacobian

- Pseudo-inverse jacobian
- Minimize tip frame offset

\[
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
\begin{pmatrix}
\delta x / \delta j_1 & \delta y / \delta j_1 & \delta z / \delta j_1 & \ldots \\
\delta x / \delta j_2 & \delta y / \delta j_2 & \delta z / \delta j_2 & \ldots \\
\delta x / \delta j_3 & \delta y / \delta j_3 & \delta z / \delta j_3 & \ldots \\
\vdots & \vdots & \vdots & \ddots
\end{pmatrix}
\]

\[\Delta J = -\Delta P \ast J^{-1}\]
Pseudo-Inverse Jacobian - Issues

- Joint limits
- Possibly degenerate pseudo-inverse
- Local minima
TRAC-IK

- Random restarts if stuck to mitigate local minima
- In parallel: sequential quadratic programming
- Faster than KDL
Inverse Kinematics, MoveIt

Limitations of built-in IK solvers:

- Only single-goal
- Only pose goal or position goal
- Analytic or gradient-based
- Hard to extend

Extensibility & Flexibility:

- Arbitrary cost function
- Minimize cost function
Cost Function
Cost Function

- Sum of errors (e.g. square distances)
- Goals: Position, Orientation, Pose, LookAt, MaxDistance, MinDistance, Touch, Line, AvoidJointLimits, MinimalDisplacement, JointVariable, CenterJoints, JointFunction, Balance,
BiolK
BioIK, Sebastian

- Multi-goal, arbitrary cost function
- Biologically Inspired
- Hybrid: Genetic Algorithm + Particle Swarm Optimization
- C#, Unity, Game Development
Genetic algorithm

- Population, individuals, genes
- Genome = joint values
- Mutation, recombination, selection

Particle swarm optimization

- Momentum
BioIK for ROS - Requirements

- C++
- MoveIt
- High Accuracy \((10^{-5}, 10\mu m, 0.0018\text{deg})\)
- Fail if unreachable (optional, default)
- Performance (default timeout: 5ms)
- Compatibility: Joint Types, Mimic Joints, etc.
Modified Evolution
Genome

- Genes = Joint positions (eg. joint angles)
- Only active variables
- `double[] genes;`

Mutation

- $r \times 2^{-s}$
  - $r$: random number gene
  - $s$: random scale per individual, $2^{-s}$ fast bit-shift
- Small mutations most likely $\Rightarrow$ high accuracy
- Clipping at joint limits
Genome and Mutation

NOT used:

- `int[]` genes;
- Mutation: mutate random bits
- Hamming walls: single mutated bit = 2 to 3 (20 to 21), but not 7 to 8 (0111 to 1000)
- Gray code: conversion overhead
- Different mutation scales per gene
- Floating point numbers & exponential term in mutation function = similar distribution
Fitness & Selection

- IK goals = fitness function
- Compute fitness for each individual
- Select fittest individuals
- Genotype: active joint values (e.g., angles)
- Phenotype: end effector link frames (and active joint values)
- Forward kinematics
  - = genotype-phenotype mapping
  - = most computation time
  → focus performance optimization on genotype phenotype mapping / forward kinematics
Secondary Objectives

- Primary and secondary objectives
- eg. primary: position goal, secondary: minimal displacement
- Pre-selection by secondary objectives
  Random number of survivors
Wipeouts

- Small mutations most likely
- Might temporarily be stuck at local optimum
- Wipeout if no improvement for a few generations
- Random wipeouts
- Wipeout = random reset of all genes
- Same for all individuals, to prevent premature convergence
Islands / Parallelization

- Multiple islands
- Independent evolution
- One thread per island
Multiple Species

- Problem: wipeout might accidentally destroy best solution
- Two competing species per island
- Only less fit species wiped out
Initialization

- Initial guess from MoveIt
Termination

- Timeout
- Good fitness
- Specialized pose comparison for compatibility
Particle Swarm Optimization

- Per-gene momentum

- `struct Individual {
  
  aligned_vector<double> genes;
  aligned_vector<double> momentum;

  genes += momentum

  momentum randomly doubled or erased

  }`
Memetics

- Evolution + local search
- Tried solvers from CppNumericalSolvers - slower
- Quadratic optimization
- Approximate local fitness landscape by parabola along gradient
- Find extremum & update if better

- Evolution for \( n \) generations
- Quadratic optimization, up to \( m \) steps
Extrapolated Forward Kinematics
Most computation time spent on computing forward kinematics

- Compute joint transforms
  - Trigonometry, sin/cos

- Concatenate frames
  - Quaternion or matrix operations
Forward Kinematics

MoveIt / KDL

- Rotation matrices (3x3 matrix, 3 basis vectors)
- Concatenation:
  27 multiplications, 18 additions

BioIK

- Quaternions (4 elements)
- Concatenation:
  16 multiplications, 12 additions
Extrapolated Forward Kinematics

- Compute exact forward kinematics only every n steps
- Extrapolate forward kinematics for most mutations
- Rotation: linear, $x = a + b \ast dj$
- Position: quadratic, $x = a + b \ast dj + c \ast dj^2$
Extrapolated Forward Kinematics

Extrapolation

- No trigonometry
- Only active joints, no static or inactive joints, no link frames
- No expensive quaternion-quaternion or matrix-matrix multiplications
  - only addition and multiplication by scalar
- Simple control flow (single loop)
  - Low risk of branch misprediction
  - Pipelining
Extrapolated Forward Kinematics

- FK: analytic derivatives, similar to pseudo-inverse jacobian methods
- Also 2nd order / quadratic
- IK goals: evolution
- Arbitrary fitness function
- Less problems at joint limits & with local minima
- No explicit matrix inversions
SIMD
SIMD

- SIMD (Single Instruction Multiple Data)
- load/store/add/mul/sub/... multiple values at the same time
- Vectors
- SIMD registers, SIMD instructions
- SSE, AVX, FMA
- Aligned memory access
- Programming:
  - SIMD intrinsics (eg. _mm_add_pd(a, b))
  - auto-vectorization (eg. #pragma omp simd)
- Function multiversioning
SIMD Mutation

Mutation

- SIMD auto-vectorization

```c
#pragma omp simd
aligned(genes_min:32),
aligned(genes_max:32),
...
for(size_t gene_index = 0; ...)
{
  ...
```
SIMD Forward kinematics

- Forward kinematics
- Quaternion operations: bad for SIMD - shuffle
- Can’t be efficiently vectorized
- Instruction latencies

- **double** \( r_x = \)
  \[
  ( p_w * q_x + p_x * q_w ) + \\
  ( p_y * q_z - p_z * q_y ) ;
  \]

- **double** \( r_y = \)
  \[
  ( p_w * q_y - p_x * q_z ) + \\
  ( p_y * q_w + p_z * q_x ) ;
  \]

- **double** \( r_z = \)
  \[
  ( p_w * q_z + p_x * q_y ) - \\
  ( p_w * q_x + p_y * q_w ) .
  \]
SIMD Forward kinematics

- Forward kinematics
- Extrapolation = ideal for SIMD
  
  - $px += \text{delta\_pos\_x()} \times \text{delta\_joint}$;
- $py += \text{delta\_pos\_y()} \times \text{delta\_joint}$;
- $pz += \text{delta\_pos\_z()} \times \text{delta\_joint}$;

- $\_\_m256d \ p = \_\_mm256\_load\_pd($
  $\text{tip\_frame\_ptr})$;

    $[\ldots]$ $p = \_\_mm256\_fmadd\_pd($ff, $(\_\_mm256\_load\_pd($\text{joint\_delta\_ptr}$), $p)$;

    $[\ldots]$ $\_\_mm256\_store\_pd($\text{tip\_mutation\_ptr}$, $p)$;
Neural Networks
Neural Networks

- Library: FANN
  ("Fast Artificial Neural Network", C++)

- Input: goal
- Output: joint values
- ... inaccurate!
Neural Networks

- Learn relative movements instead!
- Input:
  - normalized relative goal pose \((x,y,z, rx,ry,rz)\)
  - current joint values
- Output: joint offsets
- Multiple iterations
- Fully connected \([n, 50, m]\) network

<table>
<thead>
<tr>
<th>PR2 Arm</th>
<th>Neural Network</th>
<th>5ms</th>
<th>(10^{-5})</th>
<th>12%</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR2 Arm</td>
<td>Neural Network</td>
<td>100ms</td>
<td>(10^{-5})</td>
<td>29%</td>
</tr>
<tr>
<td>PR2 Arm</td>
<td>KDL Network</td>
<td>5ms</td>
<td>(10^{-5})</td>
<td>55%</td>
</tr>
</tbody>
</table>
Experiments
video
video
video
Shadow Hand
Shadow Hand BiolK

video
Shadow Hand Mixer Test
IK Goals

- Finger tips (Line, Touch, LookAt)
- Palm horizontal (LookAt)
- Coupled joint pair (J1, J2)
- MinimalDisplacement (secondary), CenterJoints (secondary)
Valkyrie Balancing Test

- Hands (PositionGoal, interactive)
- Feet (PoseGoal)
- Foot reflexes
- Body upright (PositionGoal, OrientationGoal, OrientationGoal)
- BalanceGoal
Valkyrie Balancing Test
Valkyrie Balancing Test
Valkyrie Balancing Test

video
Benchmark
Benchmark

• Generate random joint values
• Compute forward kinematics
• Compute inverse kinematics
• Compare results

• Timeout (5ms)
• Max error ($10^{-5}$)
• 10000 samples
• Success rate
• Average solve time
IK Test - PR2 - KDL

- 53.44% success rate
IK Test - PR2 - BiolK 1

- 70.34% success rate
IK Test - PR2 - TRAC IK

- 99.69% success rate
IK Test - PR2 - BioIK 2

- 99.99% success rate
# Benchmark

## Success Rate

<table>
<thead>
<tr>
<th></th>
<th>PR2</th>
<th>UR5</th>
<th>Valkyrie arm</th>
<th>Valkyrie foot</th>
<th>iiwa</th>
<th>avg</th>
<th>err</th>
</tr>
</thead>
<tbody>
<tr>
<td>bio2_memetic</td>
<td>100.00%</td>
<td>99.90%</td>
<td>99.93%</td>
<td>100.00%</td>
<td>99.88%</td>
<td><strong>99.94%</strong></td>
<td>0.06%</td>
</tr>
<tr>
<td>bio2_memetic_L</td>
<td>100.00%</td>
<td>99.79%</td>
<td>99.97%</td>
<td>100.00%</td>
<td>99.83%</td>
<td><strong>99.92%</strong></td>
<td>0.08%</td>
</tr>
<tr>
<td>bio2</td>
<td>99.99%</td>
<td>99.78%</td>
<td>99.74%</td>
<td>99.99%</td>
<td>99.90%</td>
<td><strong>99.88%</strong></td>
<td>0.12%</td>
</tr>
<tr>
<td>trac_ik</td>
<td>99.79%</td>
<td>99.34%</td>
<td>99.55%</td>
<td>99.98%</td>
<td>99.87%</td>
<td><strong>99.71%</strong></td>
<td>0.29%</td>
</tr>
<tr>
<td>bio2_memetic_lbfgs</td>
<td>99.98%</td>
<td>99.55%</td>
<td>98.68%</td>
<td>100.00%</td>
<td>99.56%</td>
<td><strong>99.55%</strong></td>
<td>0.45%</td>
</tr>
<tr>
<td>jac_4</td>
<td>83.31%</td>
<td>94.95%</td>
<td>71.76%</td>
<td>92.69%</td>
<td>78.17%</td>
<td><strong>84.18%</strong></td>
<td>15.82%</td>
</tr>
<tr>
<td>gd_c_4</td>
<td>82.56%</td>
<td>87.55%</td>
<td>63.41%</td>
<td>94.42%</td>
<td>84.35%</td>
<td><strong>82.46%</strong></td>
<td>17.54%</td>
</tr>
<tr>
<td>bio1</td>
<td>76.04%</td>
<td>50.79%</td>
<td>29.23%</td>
<td>70.00%</td>
<td>67.51%</td>
<td><strong>58.71%</strong></td>
<td>41.29%</td>
</tr>
</tbody>
</table>

**BioIK 99.94%, TRAC_IK 99.71%**
## Benchmark

### Average Solve Time

<table>
<thead>
<tr>
<th></th>
<th>PR2</th>
<th>UR5</th>
<th>Valkyrie arm</th>
<th>Valkyrie foot</th>
<th>iiwa</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>bio2_memetic</td>
<td>0.47ms</td>
<td>0.51ms</td>
<td>0.51ms</td>
<td>0.30ms</td>
<td>0.48ms</td>
<td>0.45ms</td>
</tr>
<tr>
<td>trac_ik</td>
<td>0.77ms</td>
<td>0.52ms</td>
<td>0.73ms</td>
<td>0.19ms</td>
<td>0.42ms</td>
<td>0.53ms</td>
</tr>
<tr>
<td>bio2_memetic_l</td>
<td>0.52ms</td>
<td>0.64ms</td>
<td>0.66ms</td>
<td>0.36ms</td>
<td>0.60ms</td>
<td>0.56ms</td>
</tr>
<tr>
<td>bio2</td>
<td>0.97ms</td>
<td>0.89ms</td>
<td>1.38ms</td>
<td>0.79ms</td>
<td>0.96ms</td>
<td>1.00ms</td>
</tr>
<tr>
<td>bio2_memetic_lbfsg</td>
<td>1.28ms</td>
<td>1.31ms</td>
<td>1.88ms</td>
<td>0.75ms</td>
<td>1.35ms</td>
<td>1.31ms</td>
</tr>
<tr>
<td>jac_4</td>
<td>2.02ms</td>
<td>0.68ms</td>
<td>1.95ms</td>
<td>0.52ms</td>
<td>1.52ms</td>
<td>1.34ms</td>
</tr>
<tr>
<td>gd_c_4</td>
<td>2.36ms</td>
<td>1.84ms</td>
<td>2.94ms</td>
<td>0.99ms</td>
<td>1.82ms</td>
<td>1.99ms</td>
</tr>
<tr>
<td>bio1</td>
<td>3.93ms</td>
<td>4.13ms</td>
<td>4.66ms</td>
<td>3.14ms</td>
<td>3.50ms</td>
<td>3.87ms</td>
</tr>
</tbody>
</table>

**BioIK**: 0.45ms, **TRAC_IK**: 0.53ms
Benchmark

BioIK

- Higher success rate
  (BioIK 99.94%, TRAC_IK 99.71%)

- Lower average solve time
  (BioIK: 0.45ms, TRAC_IK: 0.53ms)