Implementation of KDL Inverse Kinematics Routine on the Atlas Humanoid Robot

Karan Khokar*, Patrick Beeson, Rob Burridge

TRACLabs Inc., Webster, Texas, 77598, USA

Abstract

In this work, we describe the implementation of an inverse kinematics (IK) routine, from the open-source Kinematics and Dynamics library, on the Atlas humanoid robot. We begin by discussing the theory behind the IK routine and then give details of its implementation. Results demonstrate the robustness, accuracy, and speed of the IK routine. An average success rate of around 85% in solving for IK and an average time of less than 2.4 milliseconds per solve was observed. We discuss the possible reasons for the low success rate and present our plans to develop a more robust inverse kinematics solver.

1. Introduction

The DARPA Robotics Challenge (DRC)\(^1\) is a global competition in which robots operate in disaster response scenarios under human supervisory control. It began in the year 2013 and was divided into three stages. The first stage was a virtual competition, held in June 2013. Teams that qualified entered the second stage and competed in December 2013 at the DRC Trials, in a real environment with real robots. There were seven teams that had the Atlas robot and the remaining nine developed their own robot. The tasks were cutting a hole in a wall using a drill, clearing debris and walking through a doorway, opening multiple doors in series, turning valves, engaging a hose in a wye, walking over uneven terrain, climbing a ladder, and driving a vehicle. The top eight teams in the second stage

* Karan Khokar. Tel.: +1-281-724-9617.
E-mail address: karan@traclabs.com

\(^1\) Karan Khokar. Tel.: +1-281-724-9617.
E-mail address: karan@traclabs.com
qualified to compete in the third and final stage (the DRC Finals), to be held in June 2015. Team TRACLabs successfully cleared the first two stages and is preparing for the final stage. Through the second stage, our approach at the Trials was to utilize as much human intelligence, human decision making, and off-the-shelf software libraries as possible. The Finals will require the robots to finish tasks faster, without any human intervention and with stricter network constraints. Keeping these requirements in mind, we are attempting to introduce more autonomy into Atlas, including perceptual autonomy and planning.

Atlas (Fig. 1(a)) is a humanoid robot developed by Boston Dynamics Inc. (BDI). It is 1.88m tall and weighs approximately 150 kg. There are twenty-eight hydraulically actuated joints: six in each arm, six in each leg, three in the back, and one in the neck (Fig. 1(b)). Each joint has encoders for position feedback, the feet have force sensors, and the wrists have six-axis force-torque sensors. The Multisense head (by Carnegie Robotics) consists of a pair of RGB cameras for stereo vision, a spinning light detection and ranging (LIDAR) sensor, and an inertial measurement unit (IMU). Additionally, an IMU is located in the lower back. Two fish-eye lens cameras were fixed to its chest looking left and right to provide a wider view and to compensate for lack of panning in the neck. It has an onboard computer and an electric pump for pressurizing hydraulic fluid. A tether carries a 3-phase 480 V external power, water for cooling the pump, and data from a field computer. The water runs through an off-board air-cooled heat exchanger, and the field computers are connected to remote operator control stations. The tests carried out for this paper were done both on the real robot as well as in simulation. However, the quantitative results are presented from simulation.

Throughout the competition, we have used Robot Operating System (ROS) as the middleware in our software development. ROS provides message passing between modules, libraries for sensors, and build utilities. We used the C++ and Python programming languages, as they are both compatible with ROS. Walking and balancing controllers were provided by BDI. Since many of the tasks require manipulation of the environment, efficient arm motion control becomes necessary. This results in the need for robust trajectory generation algorithms. Inverse kinematics is an important part of trajectory generation and hence, having a robust inverse kinematics routine is necessary.

Fig. 1: (a) Atlas humanoid robot by Boston Dynamics Inc. (b) Diagram showing the joints of the Atlas robot, along with the global frame

2. Inverse Kinematics as a Linear Least Square Problem

Inverse Kinematics (IK) is the method of determining a set of joint angles that will satisfy a given end-effector pose in the Euclidean space. The space of joint angles is called the configuration space whereas the Euclidean space
is called the Cartesian space. Joint angle values are needed by the robot’s motion controller in order to move the end-effector of a robot to a desired point in space, or through multiple points in case of a trajectory. Humans specify destinations and trajectories for robots in Cartesian space and not in configuration space. Hence, methods like inverse kinematics are necessary to make the conversion.

One of the ways to solve for inverse kinematics is numerically. For instance, as a solution to a linear least squares fit problem. Numerical methods must be used when a closed form solution does not exist. Even when a closed form solution exists, computation times can be much longer than those for numerical methods. The numerical solution is not always accurate but, when it exists, the accuracy is sufficient for arm motions. At times the numerical solution does not exist or the solution fails to converge within the permissible tolerance limits. In these scenarios, other methods need to be looked into or be developed. In this paper, we will focus on numerical methods based on the least squares solution.

Consider an $n$-jointed kinematic chain. Each Cartesian coordinate is a function of each of the joint angles. Let $y_1, y_2, \ldots, y_6$ be the Cartesian coordinates and let $x_1, x_2, \ldots, x_n$ be the joint angles of the $n$ joints. Then,

$$y_1 = f_1(x_1, x_2, x_3, \ldots, x_n)$$
$$y_2 = f_2(x_1, x_2, x_3, \ldots, x_n)$$
$$\ldots$$
$$y_6 = f_6(x_1, x_2, x_3, \ldots, x_n)$$

(1)

In matrix form this is represented by $Y = F(X)$, where $X$ represents the vector of joint values, and $Y$ represents the vector of Cartesian coordinates. If we take the partial derivatives of each Cartesian coordinate with respect to each joint angle, we get,

$$\delta y_1 = \frac{\delta f_1}{\delta x_1} \delta x_1 + \frac{\delta f_1}{\delta x_2} \delta x_2 + \ldots + \frac{\delta f_1}{\delta x_n} \delta x_n$$
$$\delta y_2 = \frac{\delta f_2}{\delta x_1} \delta x_1 + \frac{\delta f_2}{\delta x_2} \delta x_2 + \ldots + \frac{\delta f_2}{\delta x_n} \delta x_n$$
$$\ldots$$
$$\delta y_6 = \frac{\delta f_6}{\delta x_1} \delta x_1 + \frac{\delta f_6}{\delta x_2} \delta x_2 + \ldots + \frac{\delta f_6}{\delta x_n} \delta x_n$$

(2)

Or in matrix form,

$$\hat{Y} = J(X)\hat{X},$$

where $J(X)$ is the Jacobean of $F$.

(3)

The solution of (3) is that of a linear least squares problem that minimizes $\|\hat{Y} - J(X)\hat{X}\|$. An exact solution to this problem does not exist, and only a best fit can be determined through this $n$-dimensional hyperspace. In order to minimize this value, some form of gradient descent must be applied. The Newton-Raphson (NR) method offers such a form of minimization. In NR, the intersection of the gradient of the curve in question with the primary dimension is determined. This intersection becomes the guess for the next iteration. Since the Jacobian is the gradient of the function $(F)$ that relates all the joint angles to a particular Cartesian coordinate, it lends itself to NR minimization method. Thus, starting from an initial guess, $X$, successive NR iterations are applied until an $X$ which is within permissible tolerance bounds is obtained.

The Jacobian is time varying because as $X$ changes, $J(X)$ changes too. The number of rows of the Jacobian
equals the degrees of freedom (DOF) in the Cartesian space and the number of columns is equal to the number of joints of the manipulator. So, for an \( n \) DOF manipulator, the size of the Jacobian will be \( 6 \times n \). If we represent the joint angles by \( \theta \) and Cartesian velocity by \( v \), (3) becomes,

\[
\dot{\theta} = J(\theta)^{-1} \cdot v
\]

(4)

\( \dot{\theta} \) is an \( nx1 \) joint velocity vector and \( v \) is a \( 6x1 \) Cartesian velocity vector containing a \( 3x1 \) linear velocity vector and a \( 3x1 \) rotational velocity vector stacked together. Thus, in order to determine the joint angles at a given instant, we need to determine the inverse of the Jacobian. This is how the guess for the next iteration in the NR method is obtained. A direct inverse is only possible when \( J(X) \) is a square matrix i.e. when the number of joints is equal to the number of DOF in Cartesian space. This is true in the case of the 6 DOF Atlas arm. For redundant arms, a pseudo-inverse must be used to approximate the inverse of the Jacobian. Methods like Singular Value Decomposition (SVD) can be used to determine the pseudo-inverse.

3. Implementation of KDL Inverse Kinematics

We used the Kinematics and Dynamics (KDL) library\(^4\) of the Open Robotics Control Software (OROCOS) toolkit to implement inverse kinematics for the Atlas kinematic chains. KDL solves for the joint angles by solving the linear least squares problem, as described in the previous section. This is done over a number of iterations, using Newton-Raphson (NR) for gradient descent minimization. The pseudo-inverse of the Jacobian is determined using Singular Value Decomposition (SVD).

We used the Robot Operating System (ROS)\(^2\) on Linux Ubuntu 12.04 to provide infrastructure. ROS features used include the ROS master, package management, build utilities, and ROS topics. C++ was the primary programming language used with ROS. ROS Visualizer (Rviz)\(^3\) was used to provide visual feedback during implementation and testing (Fig. 2(a)). KDL was added as a third-party library. The Atlas robot model used was, as provided by Boston Dynamics Inc. (BDI), in the form of a Unified Robot Description File (URDF). The URDF file was uploaded to the ROS parameter server using the launch utility, and then read in the C++ program as a URDF model. The launch utility also started the ROS core. The URDF model was then converted to a KDL tree, and then eventually to a KDL chain.

Various kinematic chains were tested. We began analyzing the 6 DOF chain of the left arm. The six joints consist of three pairs of alternating roll and pitch (in that order) joints in the shoulder, elbow and wrist (Fig. 1(b)). To test whether an improvement in the results would be obtained, we increased the number of joints in the kinematic chain. It is commonly believed that increasing the number of joints in an arm increases its manipulability. Thus, the back joints were integrated into the kinematic chain of the 6 DOF left arm. First, the back roll was added to make the chain have 7 DOF. Then, the back pitch joint was added to make the chain have 8 DOF. Finally, the back yaw was added and the 9 DOF chain was analyzed.

Excessive roll and pitch of the back causes the center of gravity (CoG) of the Atlas to displace considerably from its longitudinal axis, causing the Atlas to become unbalanced and eventually fall over. To prevent this, we used the weighted damped least square (WDLS) method of solving for inverse kinematics, instead of simply the linear least square method. WDLS provides the ability to add weights to those joints for which less movement is desired compared to the other joints. The matrix that we use to assign the weights is called the joint-space matrix. The method also gives the option to constrain a Cartesian DOF, from among movements along \( x, y, z, \) roll, pitch and yaw, by adding weights to a task-space matrix. The damping in the WDLS is a way of avoiding singularity. However, analysis for singularity was not carried out, and thus singularity avoidance was not implemented and tested. There has been some work in the past on these lines\(^6\).

Joint limits were taken into consideration during the tests, otherwise the results would not have been useful for practical purposes. As a result, KDL’s NR and Joint Limit (JL) based iterative solver was used. For joint limits, the solver simply assigns the limiting value to the joint whose value increases beyond its limit as a result of the inverse kinematic solution. Methods that use weighted joint limit avoidance exist\(^7\), but they have not been used or analyzed in this work.
4. Experiments

We conducted experiments to determine the robustness, accuracy and speed of the implemented KDL inverse kinematics routine. Tests were conducted in simulation as well as on the real robot. The quantitative results presented are from simulation. We randomly sampled the configuration space of the left arm 1000 times and obtained the metrics for the 6 DOF kinematic chain. Tests for 7, 8 and 9 DOF chains were done similarly. To form these chains, the back joints were appended to the left arm, as mentioned in the previous section. Upper-torso, of the left arm, was the base link for the 6 DOF chain. Mid-torso link, when added as the base to the chain, resulted in back roll joint being included and the chain turning into a 7 DOF one. Similarly, lower-torso added the back pitch and made the chain an 8 DOF one. And finally, addition of pelvis as the base resulted in addition of back yaw and the resulting chain became a 9 DOF one.

<table>
<thead>
<tr>
<th>Joint</th>
<th>Min Limit (rad)</th>
<th>Seed (rad)</th>
<th>Max Limit (rad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back yaw</td>
<td>-0.663225</td>
<td>0</td>
<td>0.663225</td>
</tr>
<tr>
<td>Back pitch</td>
<td>-0.610691</td>
<td>0</td>
<td>0.438427</td>
</tr>
<tr>
<td>Back roll</td>
<td>-0.698132</td>
<td>0</td>
<td>0.698132</td>
</tr>
<tr>
<td>Shoulder roll</td>
<td>-1.5708</td>
<td>0.77</td>
<td>0.785398</td>
</tr>
<tr>
<td>Shoulder pitch</td>
<td>-1.5708</td>
<td>0.75</td>
<td>1.5708</td>
</tr>
<tr>
<td>Elbow roll</td>
<td>0</td>
<td>2.15</td>
<td>3.14159</td>
</tr>
<tr>
<td>Elbow pitch</td>
<td>0</td>
<td>2.14</td>
<td>2.35619</td>
</tr>
<tr>
<td>Wrist roll</td>
<td>0</td>
<td>0.01</td>
<td>3.14159</td>
</tr>
<tr>
<td>Wrist pitch</td>
<td>-1.1781</td>
<td>0.3</td>
<td>1.1781</td>
</tr>
</tbody>
</table>

Fig. 2: (a) Seed configuration chosen in our experiments (b) Seed value and comparison with joint limits

For each of the 1000 samples, KDL Newton-Raphson (NR) and joint limit (JL) based Weighted Damped Least Square (WDLS) inverse kinematics was carried out. 1000 iterations of NR minimization were used and the tolerance bounds were kept at 0.001. At the end of the 1000 iterations, if any of the six Cartesian coordinates had a difference of more than 0.001 units (meters or radians) with the desired pose, that run for the sample was considered a failure. If it was a failure, the differential twist was computed as the error value. A differential twist is the difference between the position and orientation of the end-effector, expressed as a 6x1 vector. The twist error was averaged over the total number of samples that failed to converge. Moreover, the time it took the KDL to solve for each of the sample poses was measured and the time averaged over the 1000 samples was presented. The computer that ran the algorithm had eight Intel i7 cores, running at 3.60 GHz, and it had 16 GB of RAM. For the kinematic chains that included the back joints, 0.1 was the comparative weight used for roll and pitch joints in the WDLS joint-space matrix, whereas the other joints used 1.0 as the weight. All elements of the task-space matrix had uniform weight, that of 1.0.

The seed configuration for the first iteration of every sample was the one shown in Fig. 2(a). It positioned the end-effector near the environment to be manipulated. Once, the first iteration was over, the joint angle configuration
computed by KDL became the seed for the next iteration, and so on. The chosen seed as shown in Fig. 2(a), however, is not the most desirable from the kinematic standpoint, because the shoulder and wrist roll joints are very close to their limits (Fig. 2(b)). Due to the same reason, many trajectories failed to execute. It would be interesting to see whether using random initial seeds would improve the results. It would also be interesting to test with an initial seed in which each joint is at the centre of its range, and another that uses a look-up table to set the seed from a pose closest to the desired pose.

5. Results and Discussion

In this section we present the results from the experiments described in the previous section. From Fig. 3, we see that the percentage success increases as the number of DOF are increased. In other words addition of the back joints to the arm is beneficial for solving for inverse kinematics (IK). There is a negligible drop in the success rate when the back yaw joint is added. This is unexpected since the yaw joint increases the workspace by a substantial amount and is the only back joint that does not have movement constraints in the joint-space matrix of the weighted damped least square (WDLS) solver. It will be interesting to test this for different seeds and for complete trajectories. Increasing the number of Newton-Raphson (NR) iterations, changing the tolerance bounds, and changing the value of members of task-space matrix are other factors that could impact the results.

![Success measure of KDL](image)

For the samples that failed to achieve 0.001 tolerance bound for each of the Cartesian coordinate, error in the form of Cartesian twist was computed, as explained in the previous section. The error was averaged over all the failed samples. We can see from Fig. 4 that the error reduces as the back joints are added to the left arm. It will be interesting to compare the error for different seeds, for trajectories, and for different NR iterations and tolerance bounds.
Next, we present the results from the time taken by the KDL IK routine to arrive at a solution. The times are averaged over all the 1000 samples for each kinematic chain type. We observe from Fig. 5 that the speed of solving for IK reduced as the number of DOF increased. However, the loss in speed is negligible compared to the gain in accuracy. The maximum average time for all the chains is still less than 2.4 milliseconds. The increase in time is expected, since as the number of degrees of freedom increase, the size of matrices and vectors increases, thus resulting in more computation time. The use of a high-end computer may have impacted the results, but computing was not made efficient algorithmically.

Overall, the results are not very promising as far as the robustness is concerned. The percentage success seems too low to expect robust trajectory execution, because failure to converge to even a single point in a trajectory can lead to its failure. Adding constraints to the task-space matrix for WDLS routine may not increase the robustness. In fact it may increase the failure rate of trajectory generation. Having more than 6 DOF in the arms would improve the
success rate if IK solves but it seems that the main problem is the design of the Atlas arms. The shoulder roll joint, which is also the base joint in the arm chain, begins at its maximum joint limit at the initial seed configuration we have used (refer Fig. 2(a)). As a result, any backward movement of the arm, i.e., along the negative global $x$ axis, will exceed the joint limit. The same problem exists with the wrist roll joint. At the configuration shown in the figure, the joint is at its lower limit. So any rotation along the negative global $x$ axis is impossible.

Moreover, the last joint of the arm is a pitch joint and not a roll joint. This is a very uncommon design and makes end-effector roll motions difficult to achieve. A very common design to achieve a variety of end-effector orientations is the Piper’s design\textsuperscript{9}. In this design, the Cartesian frames of the last three joints (i.e., the wrist joints) coincide. This gives a full range of orientation motion. It does produce problems of singularity when the last and the third-to-last joints align, but singularity avoidance techniques have been implemented successfully in the past\textsuperscript{6}.

In these scenarios, it seems that joint error minimization techniques that relax tolerances by different amounts on different individual Cartesian coordinates, and allow for deviations from the actual path, would be useful. Adding more DOF to the arms is desirable but changing the kinematics design, especially that of the wrist roll joint, is very important for improving robustness of the IK solvers.

6. Conclusions and Future Work

In summary, we have presented the results of implementation of KDL inverse kinematics routine on Atlas humanoid robot. Although the results demonstrate that the robustness, accuracy, and speed of determining inverse kinematics solutions increase with the addition of the back joints to the kinematic chain, they also show that the robustness remains poor. Changing the initial seed location, and changing the number of Newton-Raphson iterations and tolerance bounds may change the results considerably. Tests on complete trajectories will demonstrate the utility of KDL for inverse kinematics. Currently, the Atlas is undergoing hardware upgrades that include adding an extra DOF to the arm at the wrist and changing the mounting orientation of the shoulder roll joint. It will be worthwhile to test KDL inverse kinematics routines on the new design. Tests on singularity and comparison with other inverse kinematics methods, such as non-linear optimization techniques and techniques that relax constraints on specific Cartesian coordinates, will be done in the future. If the new design improves the workspace of the Atlas end-effector, then implementing weighted least norm techniques to optimize motion based on priorities will be carried out. Comparison with MoveIt!’s version of KDL inverse kinematics will be done to determine whether it performs better than the currently implemented KDL solver.

References

5. ROS Visualization (Rviz). http://wiki.ros.org/rviz