Approximate Inverse Kinematics Using a Database

Alex Henning
Worcester Polytechnic Institute (WPI)
Worcester, MA 01609
Email: adhenning@wpi.edu

Advisor:
Prof. Dmitry Berenson
Worcester Polytechnic Institute (WPI)
Worcester, MA 01609
Email: dberenson@cs.wpi.edu

A Major Qualifying Project Report submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the Degree of Bachelor of Science

December 08, 2014

Abstract

The goal of this project was to create an approximate inverse kinematics solver that generates solutions for manipulators that have less degrees of freedom (DOF) than the space in which their end-effectors move. In such situations, it is improbable that there is an exact solution to an arbitrary 6-DOF pose. Even though one cannot find an exact solution, one can often find an approximate solution. In this paper, we propose a database approach for finding the approximate inverse kinematics solution. The motivating example for this project is the iRobot PackBot. This robot only has a 5-DOF arm, but its end-effector operates in a 6-DOF task-space. Without an approximate inverse kinematics solver, it would be difficult to interface with existing algorithms, such as grasping algorithms. This approach enables interaction with algorithms that assume that 6-DOF poses can be reached.

Approximate Inverse Kinematics Using a Database

I. Introduction

The goal of this project was to create an approximate inverse kinematics (IK) solver. The solver generates solutions for manipulators that have less degrees of freedom (DOF) than the space in which their end-effectors move. In such situations, it is improbable that there is an exact solution. However, an inverse kinematics solution can often be found within some task-specific tolerance of the target pose. This tolerance allows us to generate an approximate solution to what would otherwise be an impossible target.

The motivating example is the iRobot PackBot, which can be seen in Fig. 1. This robot has an arm which only has 5 degrees of freedom, but its end-effector operates in a 6-DOF task-space. This problem is not limited to the PackBot, but is also present in other robots. Another example of a robot with the same problem is the KUKA YouBot which also has a 5-DOF arm. More generally, this applies to all non-planar arms with 4- or 5-DOF and 2-DOF planar arms which are trying to reach a target position and orientation within the plane.

In this paper, we propose a database approach to finding the approximate inverse kinematics solution. The proposed algorithm uses the iterative Jacobian method to improve the best poses found from the database and return an approximate solution if it is within a specified tolerance.

II. Background

A. Typical Approach

Typically, when approaching the problem of dimension mismatch between configuration- and task-space, one tries to change the dimensionality to make it more tractable. This change in the dimensionality of either configuration- or task-space makes the problem solvable, but at the cost of additional limitations.

An example of a task-space change in dimensionality would be using the PackBot, but only caring about the position of the end-effector. This provides more dimensions in configuration-space than task-space and allows the IK solver to take advantage of existing work on IK with redundant manipulators[1]. This approach sacrifices control over the end-effectors orientation, but it will find an exact IK solution. However, this may not be an option if orientation matters.

An example of changing the configuration-space dimensionality — often done with the KUKA YouBot — is to include one or more of its three driving degrees of freedom when solving for IK solutions[2]. This allows exact solutions to be found since the robot now has at least as many degrees in configuration-space as in task-space. This approach allows control of the orientation, but it requires accurate control over the drive base, which is not the case with many robots and does not work in all conditions. Control of the drive base is usually worse than control of a robot arm, limiting the precision. The precision further degrades over rough terrain. This can also be more difficult for robots without a holonomic drive train, making manipulation tasks difficult to perform precisely.

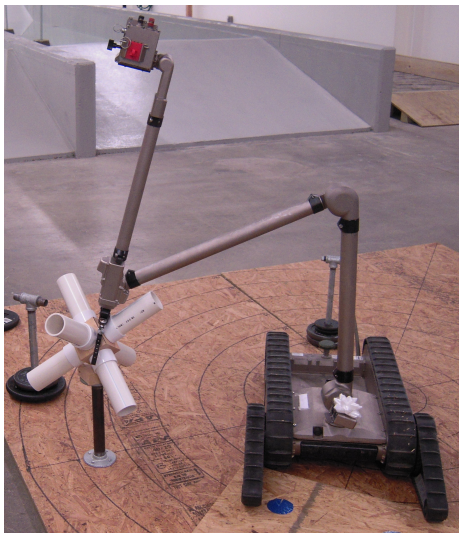


Fig. 1. The iRobot PackBot, a robot with its 5-DOF, picking up an object.

B. Approximate is Good Enough

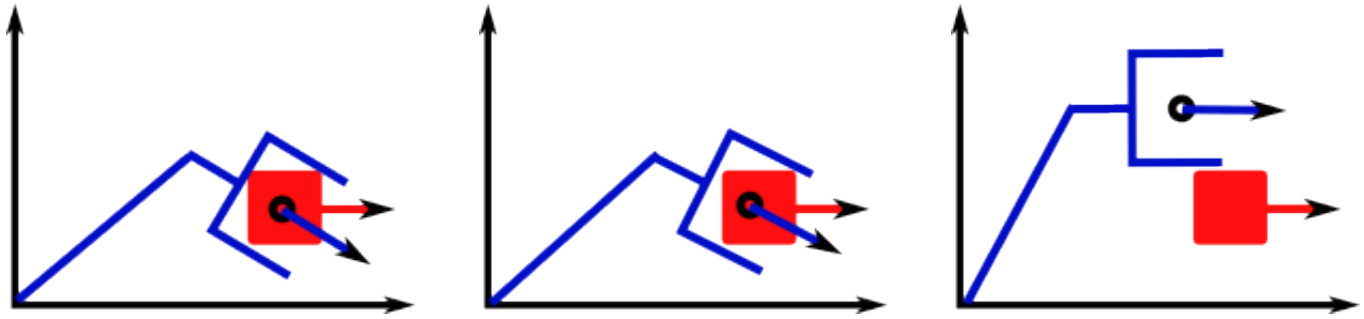
When operating robots, an approximate solution can be good enough depending on the task and the necessary precision. In fact, under standard operating conditions for most manipulators some amount of uncertainty exists in the end-effector position due to a variety of factors, including:

- Settling error in the controller
- Miscalibration of sensors
- Mechanical tolerances
- Backlash in the joints

These factors are usually ignored when doing IK, since in a well-designed system, they are within the tolerance necessary to perform the manipulator's tasks. Given that the manipulator can deal with these approximations, it is reasonable for a robot to deal with approximate IK solutions, so long as the loss in accuracy is not significant with respect to the task's tolerance.

Looking at Fig. 2, one can see a 2-DOF arm trying to reach a 3-DOF pose. When solving only for position as seen in Fig. 2(a), the solution leads to a collision with the object. When solving only for orientation as seen in Fig. 2(c), the solution is too far away to pick up the object. In Fig. 2(b), a trade-off is made which happens to be close enough to grasp. While this is not always guaranteed, by looking at approximate solutions one can often find a solution within a reasonable tolerance.

An increasing trend in robotics is to design systems that can work reliably in uncertain environments. The uncertainty robots designed for these environments allow is often much



(a) Position Only (b) Trade-off (c) Orientation Only

Fig. 2. A 2-DOF robot trying to reach a 3-DOF pose. As seen in Fig. 2(a) on the left, when only position is used, the gripper is in collision with the object. On the right in Fig. 2(c), it can be seen that orientation alone, without position, is too far away and the gripper cannot grab the object. Only in Fig. 2(b) in the middle, where there is a trade-off, is it possible for the robot to get close enough to grasp the box.

greater than robots that assume precise knowledge of the environment. For example, Deimel and Brock[3] created a compliant hand that can be used to robustly grasp a variety of objects. Some objects it could pickup include eye glasses, bottles of water and fruit. Systems using these compliant grippers can take advantage of an approximate solution due to the robustness of the gripper. This can allow the creation of a solution with less degrees of freedom which is less expensive, without a significant loss of reliability.

One example of a less expensive robot is the Baxter manufacturing robot which reduces costs by using less accurate sensors and actuators. A modified version of the Deimel and Brock compliant gripper was made at WPI[4] to work with Baxter. While Baxter's arms have 7-DOF, a similar robot with this gripper and a single 5-DOF arm could take advantage of an approximate IK solver to enable more cost-efficient yet robust manipulation.

C. Iterative Jacobian Inverse Kinematics

A simple approach to the problem of approximate IK is iterative Jacobian inverse kinematics. This method can be very effective if the current pose is near the target pose. However, the farther away the target pose gets, the more likely the iterative Jacobian method is to get stuck in a local minima or at a joint limit. Once it gets stuck, there is no way to use this method to get near the target pose without modifying the algorithm. While this algorithm can be used to help approximate IK solutions, it is insufficient on its own for most use cases.

D. Database Approaches

Database methods have a history of being used in robotics for tasks that are expensive to compute, but quick to use once computed. OpenRAVE[5] uses databases to store grasp poses relative to the target object. This allows the robot to use some criteria to quickly choose one of the known good grasps, knowing that it satisfies more computationally intensive criteria; one commonly used criteria is the grasps stability.

An example of using databases for inverse kinematics would be storing multiple analytical IK solutions in a database. OpenRAVE[6] implements such a database method for inverse kinematics as discussed in [5]. It uses the database to store analytical solvers generated by ikfast[7] which are designed for different conditions. Each analytical solver assumes that there is one "free joint" set to a known value. The database has solvers for each possible free joint. This allows for fast IK solutions using the pre-computed analytical solvers regardless of the chosen free joint.

Another example of using a database for IK is the RoboSimian robot that competes in the DARPA Robotics Challenge. It handles kinematic redundancy while walking[8] by using a database lookup to generate IK solutions that allow smooth and statically stable walking motions. The database ensures that the chosen solution is reachable in a smooth and reasonable motion, whereas a random IK solution would be difficult or impossible to reach smoothly.

III. Approach

The approach taken in this project was to pre-compute a database of known relationships between task-space and configuration-space. Then, during execution when trying to solve for a given pose, find the K nearest neighbors of the target position in task-space. For each of these neighbors, the iterative Jacobian inverse kinematics algorithm is used to get closer to the target pose. If at any point an exact solution is found, it is returned immediately. Otherwise, take the closest solution out of all the trials and return it as the solution if it is within a tolerance of the target task-space pose. The closest solution is determined using a parameter λ, which will be discussed in detail later, to control the trade-off between position and orientation.

More formally this algorithm can be expressed in pseudo-code:

1: **procedure** IK($pose$, ▷ target final pose
λ, ▷ trade-off in position vs. orientation
$threshold$, ▷ max acceptable distance
K) ▷ number of nearest neighbors to check
2: $db \leftarrow$ LoadDatabase(λ)
3: $nn \leftarrow$ NewDatabase(λ)
4: **for** $p \in db$.Nearest($pose, K$) **do**
5: **if not** InCollision(p) **then**

```
            p ← IterativeJacobianIK(p, pose)
            if IsExact(p) then
                return p
            end if
            if not InCollision(p) then
                nn.Add(p)
            end if
        end if
    end for
    nearest ← nn.Nearest(pose)
    if distance(nearest, pose) < threshold then
        return nearest
    else
        return failure
    end if
end procedure
```

A. Database Creation

One of the key aspects of this approach is the database generation. The generation of the database can have a major impact on the performance. This subsection describes how to generate the database. The initial database used by this algorithm was generated by discretizing the 5-DOF of the PackBot arm and moving it to all permutations of the discretized joint values. At each permutation the configuration-space location and the corresponding task-space pose are recorded into the database. The resulting database is uniformly distributed in configuration-space, but in task-space the distribution depends on the kinematics of the arm and is unlikely to be distributed uniformly. Target poses in areas with higher density in task-space are more likely to have a good approximate inverse kinematic solution when compared to target poses in areas with a lower density.

When generating the database, there is a trade-off to be made between the sampling density and the time to generate the database. Finer discretization is more likely to have poses close to the target pose, but doubling the discretization of each joint increases the database size and generation time by a factor of 2^{dof} — a factor of 32 for the PackBot and other 5-DOF robots. This causes diminishing returns on finer discretization. However, if the discretization is too coarse, the number of nearby poses in task-space shrinks, decreasing the likelihood of finding a solution.

One way to minimize the exponential increase in database size and generation time is to discretize the different joints with different levels of granularity depending on their significance. A small change to the base joint is likely to have a much larger effect on the end-effector pose than the final joint in the kinematic chain. By selectively choosing the discretization of each joint, it is possible to improve the database quality without increasing its size.

The naive way to discretize different joints is to guess based on an intuition and assign finer discretization to joints closer to the root of the kinematic chain. A more optimal way is to weight joints based on their swept volumes. Joint resolutions for discretization can then be calculated using the swept volumes. The result can be used to generate a database with a more useful distribution of poses in both task- and configuration-space.

When adding configurations to the database during creation, some configurations are pruned if they are impossible. In the general case, this includes all poses where the robot would be in collision with itself, such as when the manipulator is inside the body. This pruning does not affect the generality of the IK solver since these poses cannot be used as solutions. The following subsection will discuss ways to shrink the size of the database at a cost of generalizability.

B. Shrinking the Database

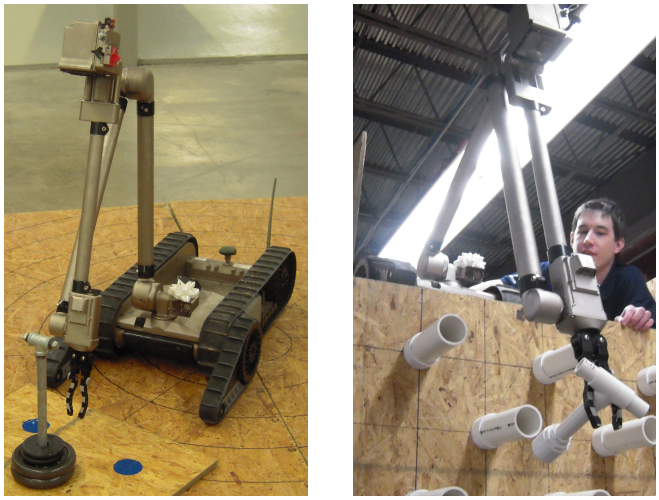
The database is likely to be quite large when generated using the above method. If the database is too large, it can take too long to lookup values in addition to using too much RAM. As a result, larger databases can make this method infeasible to actually use. This problem can be mitigated using a number of methods to remove poses from the database. These methods make the IK solver less general by taking advantage of additional information to prune database for specific use cases.

The simplest way to shrink the database is to only keep solutions that are within a task-space volume of interest. For example, with the PackBot, most manipulation happens in front of the robot. Using this knowledge, on can reduce the task-space from anywhere the robot can reach to just the area in front of the robot. By doing this, it is possible to reduce the number of poses in the database by orders of magnitude — exactly how much depends on the size of the new task-space.

Another way that poses can be reduced is by making assumptions about the environment. If you assume that the PackBot is going to be operating on relatively flat ground, it is possible to add the ground to the model and prune poses where the robot is in collision with the ground. The size reduction due to this is likely not as significant as from limiting task-space, but can still reduce the size of the database noticeably. The actual reduction depends on the kinematics of the robot.

It is also possible to have multiple databases and load the most appropriate one for the task at hand. As an example, when picking up an object in front of the robot as seen in Fig. 3(a), it can use a database that assumes there is ground and that the task-space is in front of the robot above the ground. However, when working with the NERVE Center's cliff test, as seen in Fig. 3(b), a different database optimized for a task space below the robot can be loaded. Separating these two concerns requires less resources when performing the given task, but requires that the proper database is loaded when performing that task. This is an extra step compared the more general, but resource intensive, version of this algorithm.

When applying these filters to the database, there are two ways they can be applied. One is during the initial database creation, while the other is after the fact. The latter is recommended, since it allows multiple databases to be created as described above even for tasks that may not have been intended during the initial database generation. The increase in computation time for filtering after the fact is negligible compared to the generation time.



(a) Object in Front (b) NERVE Center Cliff

Fig. 3. The iRobot PackBot manipulating objects in different environments.

For this project, we defined a volume in front of the robot and limited the database to only include poses in the volume. The volume limited the positions to be within a rectangular volume in front of the robot and the orientations are limited to be orientations used by the PackBot for manipulation tasks. This allowed us to have a database with only $18,708$ entries instead of the $1,963,585$ entries in the full database created from the above discretization method.

C. Measuring Pose Distance

When measuring the quality of an approximate pose, one must measure the distance between the approximate pose and the target pose. However, this is difficult, because there is no clear way of combining the position and orientation distances. Depending on the task at hand, the relative importance of the two components vary. Some tasks care more about position, while others care more about orientation.

For example, when using the PackBot to pick up an object, the position is generally more important than the orientation, since if the object is not between the grippers, it is impossible to pick it up. Another task is looking at objects with the PackBot's camera. For this task, orientation matters far more. In fact, the farther away the object of interest is, the less the position matters relative to orientation. However, there are tasks with a much more uncertain trade-off. Using the PackBot's camera to look through a car window requires that the position and angle are both close enough or it will not be able to see into the car.

Since there is no well-defined solution to this problem, the IK solver leaves the relationship between position and orientation as a parameter that can be specified to reflect the task at hand. The trade-off is formulated as a Pareto trade-off, using a parameter λ to determine the relative weights of position vs. orientation. This is given in the following equation:

$$\text{distance} = \lambda \Delta p + (1 - \lambda)\theta$$

The value of λ is in the interval $[0, 1]$, where a value of $\lambda = 1$ only takes the euclidean distance between the positions into account and a value of $\lambda = 0$ takes only the angle between the orientations into account. Values in between correspond to different trade-offs that the user is willing to make in order to get an approximate solution. For instance, when $\lambda = 0.5$ an error of 1 meter is considered equal to an error 1 radian. This value can be tuned as necessary for the task.

D. Nearest Neighbors

A key element to this algorithm is efficiently finding a number of the nearest poses in task-space to the target pose. This is the well studied K nearest neighbors problem and there are many libraries for finding approximate nearest neighbors. The two most popular are ANN[9][10] and FLANN[11][12]. This implementation uses FLANN, since it allows us to specify custom distance metrics. This matters for two reasons: First, since end-effector pose in 3D space is not a euclidean space due to the rotational elements of pose, it is necessary to define a custom distance metric to get the true distance. Second, in order to include the Pareto trade-off, the distance metric must use the specified λ-value.

The custom distance function calculates the distance between the position and the rotation as represented by a unit quaternion. These are then combined using the parameter λ (as discussed above) to account for the relative importance of position vs. orientation. In addition to varying the weight for position and orientation, it would also be possible to modify the FLANN distance metric to weight different axes more or less heavily, for example if the z-axis mattered more than the x- and y-axes.

Since quaternions are being used to represent rotation, the distance metric must take special care when measuring the distance between them. The proper way to get the distance between two unit quaternions representing rotations is to find the angle between them. However, this method does not allow the database to be efficiently stored by FLANN as a KD-Tree and significantly reduces the performance of database lookups. Instead, an alternative approach to the distance between quaternions is taken. The approach produces the same ordering for nearest neighbors of quaternions. The distance between quaternions is treated as a euclidean distance and the nearest neighbors are looked up in the database twice: once as its value and once with the quaternion of opposite sign which represents the same rotation. The results of these two lookups are then merged. Finally, the K nearest neighbors are returned. This method results in the same K nearest neighbors as using the proper method, but allows the database to be stored as a KD-tree. The benefit of using a KD-tree is that it enables efficient lookups, even as the database size increases.

Once the nearest neighbors are found, they are then used as the starting point for the iterative Jacobian inverse kinematics algorithm[13] to get as close to the target pose as possible. This is done from the K nearest neighbors instead of just the single nearest neighbor since it can get stuck at local minima due to joint limits and numerical issues such as singularities. However, by trying to approach from multiple

nearby neighbors, the chance of running into joint limits and singularities is greatly reduced.

E. Alternatives to Pareto

Using Pareto optimality to control the trade-off is one method to find an approximate solution. However, there are other options and we explored one of them. Instead of using the orientation directly, we treated the manipulator as two manipulators, one for each finger. From here the position Jacobians of the two fingers were used. An extra step was added where each finger's target was assigned based on which minimized the distance between the two fingers and their targets.

This setup naturally takes advantage of the fact that the PackBot's gripper is rotationally symmetric, opening up the possibility for more solutions that potentially could get closer to the target. It also removes the need to specify the λ parameter, which can be unintuitive to pick. Instead, the two position Jacobians naturally minimizes the position error between both fingers and their targets. If the orientation is defined by the finger locations, the result will be a more natural trade-off between the placement of the fingers.

One problem with just using the fingers on the PackBot is that it only has two fingers, so there is a degree of freedom left where the gripper can be at any rotation about the line between the two finger targets. This can be constrained without sacrificing the ability to choose between the two symmetric orientations of the gripper by adding a third point forming an isosceles triangle. The geometry of this triangle implicitly controls the trade-off in orientation. This alternative measure allows you to ignore measuring the orientation distance and define the error as the sum of the distances between the three points and their corresponding targets.

IV. Results

In order to verify our method and its feasibility, we performed a number of tests using three algorithms. The tests all work within a volume of interest which is in front of the robot — an area in which it would commonly be doing manipulation. This volume constrains both the position and the orientation to what one would use for manipulation in front of the robot.

The first algorithm is simply the iterative Jacobian method from a single fixed pose. The single pose has an end-effector pose as close to the center of the volume of interest as the PackBot could get. When trying to solve for a pose it sets the PackBot to this pose and tries to get as close as possible to the target.

The second algorithm is the database method described above. It takes a number of the nearest neighbors from the database, uses the iterative Jacobian method to get as close as possible to the target pose and returns the closest of all these points to the target. This has the advantage of trying from many locations that are nearby, minimizing the chance of getting stuck at a joint limit or local minima. The λ parameter was set to 0.5 for all tests as this allowed rotation and position to be treated with similar weighting.

TABLE I
Test results for leave-one-out testing.

Algorithm Used	Successes (out of 1000)	Average Time (milliseconds)	Variance (in time)
From a Fixed Point	164	35	0.0070
From Points in the Database	1000	27	0.0027
Finger Points from Database	998	115	0.31

The third algorithm tested is the finger method discussed above, which creates a triangle of points using the two fingers and stacks the three position Jacobians when running the iterative Jacobian method. This method uses the same database. The advantage of this is that the fingers can go either way since the PackBot's gripper is symmetric. It also implicitly takes care of the trade-off between position and orientation, getting rid of the need to have a lambda parameter.

A. Leave One Out Testing

The first test we did was standard leave-one-out testing. For 1000 end-effector/robot pose pairs in the database, we removed one pair at a time and ran the IK solver to get a solution for the target pose. The success rate of each algorithm and average time to solve 1000 targets was recorded in table I. This test was run to verify that the methods can get exact solutions to poses that the PackBot is known to be able to reach.

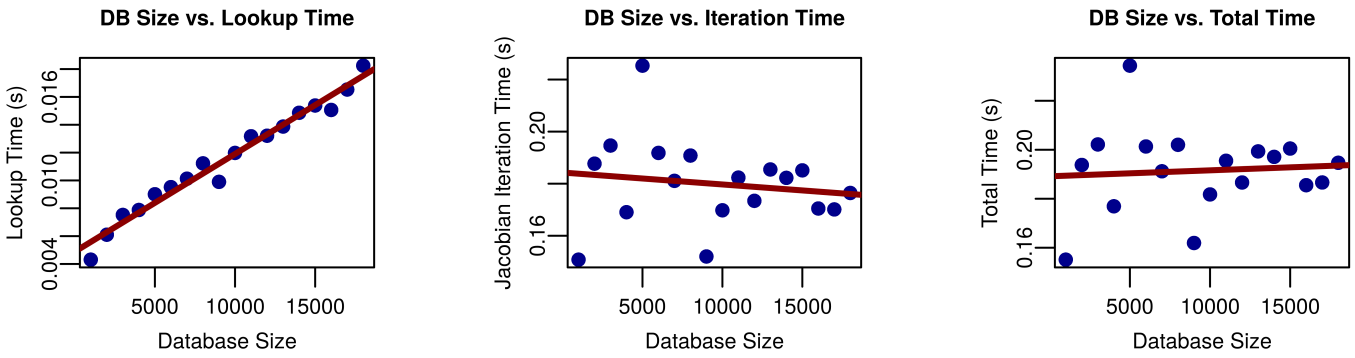
B. Sampled Grasps

To test under more realistic scenarios, we ran a test to get as close as possible to grasps from a grasp set. This allows the methods to be tested under a real usage scenario. Using grasp sets gives the solvers extra chances to get close, whereas a single target may be impossible to approximate with the kinematic structure of the PackBot.

For this test, a cylinder was placed at a random pose inside a volume in front of the robot. Thirty-two grasps were then found spaced evenly around the cylinder. The IK solver was then run on each of the target grasp poses and the closest match was returned. If the solution was within a tolerance of 0.01 or 0.05 of the target grasp, it was marked as a success. These tolerances were chosen, since they were both small. In fact, applying a small force to the PackBot's end-effector can shift the pose by more than 1 cm. Once orientation change is factored in, a tolerance of 0.05 is reasonable.

C. Effect of Database Size

For this test, new databases were created with from 1000 to 18000 poses in intervals of 1000. The points were randomly chosen from the original database. Leave-one-out testing was run for 1000 data-points and the values were recorded. They all correctly solved 100% of the poses, so that was not recorded. The values recorded included the average number of nearest neighbors and iterations of the iterative Jacobian method before finding a successful result. The corresponding times and the total time were also recorded. The raw data can be seen in table III, with the data plotted in Fig. 4.



(a) Lookup Time (b) Iteration Time (c) Total Time

Fig. 4. The effect of database size on performance. Fig. 4(a) and Fig. 4(b) show the time — averaged over 1000 leave-one-out trials — for database lookup and the iterative Jacobian method respectively. Fig. 4(c) shows the total time. All figures show a regression line in red fit to the points.

TABLE II
TEST RESULTS FOR GRASPS AROUND A RANDOMLY PLACED CYLINDER.

Algorithm Used	Cutoff	Successes (out of 1000)	Average Time (milliseconds)	Variance (in time)
From a Fixed Point	0.01	217	505	0.0042
From a Fixed Point	0.05	826	505	0.0042
From Points in the Database	0.01	251	4535	0.60
From Points in the Database	0.05	966	4535	0.60

TABLE III
RESULTS FOR TESTING THE IMPACT OF DATABASE SIZE ON PERFORMANCE.

Size	Average Nearest Neighbors	Average # Iterations	Average DB Time (milliseconds)	Average Jacobian Time (milliseconds)	Average Total Time (milliseconds)
1000	1.912	83.952	4.311	150.821	155.132
2000	2.002	93.193	6.102	187.693	193.795
3000	2.100	94.811	7.523	194.652	202.175
4000	2.078	88.151	7.879	169.044	176.923
5000	2.337	113.035	9.009	225.340	234.349
6000	2.153	98.613	9.521	191.819	201.340
7000	2.137	92.256	10.126	181.092	191.218
8000	2.212	92.571	11.228	190.806	202.034
9000	2.182	89.658	9.900	152.036	161.936
10000	2.192	89.079	11.978	169.782	181.760
11000	2.255	92.693	13.174	182.336	195.510
12000	2.260	92.260	13.213	173.419	186.632
13000	2.320	95.823	13.872	185.465	199.337
14000	2.260	94.976	14.861	182.245	197.106
15000	2.234	96.284	15.384	185.117	200.501
16000	2.249	95.568	15.064	170.466	185.530
17000	2.144	90.589	16.531	170.116	186.647
18000	2.120	89.487	18.247	176.436	194.683

V. DISCUSSION

A. IK Leave-one-out testing

Looking at the results from leave-one-out testing as seen in table I, one can see that our database method outperforms the fixed-point algorithm and finds a solution for every pose. It also runs faster than the iterative Jacobian method since nearby poses take less iterations to get close to the target and our algorithm returns immediately on an exact match.

Running the iterative Jacobian method performs significantly worse in our test volume, but the success rate is relatively optimistic in the more general case. Larger test volumes would require more iterations to be successful and the fixed point method would be more likely to run into joint limits and other problems as it approached the edges of the larger volume.

The third test method using points on fingers got a 99.8% success rate, but ran significantly slower than the other two methods tested. It is comparable in accuracy to our database method, though it runs 4 times slower and had 2 failures out of 1000 trials due to getting stuck at joint limits.

The results of this test show that our algorithm can successfully solve the inverse kinematics problem when a solution exists even though the straight iterative Jacobian method from a fixed point fails most of the time. This illustrates that our algorithm will work when an exact solution is possible and there is no loss in accuracy by using our approximate IK solver when an exact solution is available.

B. Sampled Grasps

Table II contains the result of trying to find the best grasp from a set of grasps around a cylinder at random locations. This test was run at two different tolerances, 0.01 which allows for up to 1 centimeter or up to 1 centiradian of error and a 0.05 which allows up to five times the error.

In the scenario with a very tight tolerance, both solutions did poorly. Without the database, it had 21.7% success rate and when using the database it had a success rate of 25.1%. The database method outperforms testing from a fixed point, but it can still only grasp one fourth of the target objects.

When the tolerance was increased, performance increased significantly for both methods. The database method was able to successfully get close to a grasp 96.6% of the time whereas the iterative Jacobian algorithm from a fixed point had a success rate of 82.6%. The database method performs significantly better here, succeeding in 140 out of the 1000 cases where the fixed point method was unsuccessful.

This test illustrates that the approximate solutions can usually get close enough to at least one of a set of target poses. Due to the kinematics of the PackBot and its 5-DOF, there are many targets that are not only impossible to get to, but also impossible to get within a reasonable tolerance. However, once you are willing to use a set of target poses, the approximate solver is able to come up with at least one acceptable solution for the task.

While requiring a set of targets may at first seem to make this IK solver useless, there are areas of robotics where sets of targets are not only acceptable, but already regularly used. The example used for this test was grasping. This works with one common technique, which is to generate a set of possible grasps offline. Then, during execution, try to get to one of the grasps using some criteria. These grasp sets can be combined with our proposed IK solver and generate successful IK solutions even though an analytical solver would fail. This allows our solver to be used with existing grasping software.

C. Effect of Database Size

One big factor that affects the feasibility of using the proposed algorithm is its performance. The resulting performance scales well with database size. Looking at Fig. 4(a), one can see that roughly speaking, lookup time doubles every time the database size quadruples. This is expected since the database lookup is based on KD-trees, which are an efficient way of storing nearest neighbors that should take $O(\log_2(n))$ time per look up.

For the database size used in our testing, calculating the iterative Jacobian took roughly 10x longer than the longest database lookup. As can be seen in Fig. 4(b), the larger databases performed slightly better in general since they were able to lookup closer points, but this trend was not always true. Most notably, the database with only 1000 poses spent the least amount of time running the iterative Jacobian method, this variance is likely due to the random points tested. In the end, ignoring the outliers, the time savings from a larger database is less than the increase in lookup time.

Based on these results, increasing database size is most useful for increasing the volume of interest. You can increase the volume to include significantly more points with minimal performance impact. However, increasing the size within the same volume takes slightly longer, which can be seen in Fig. 4(c). It is not until you get a very large database that lookup time becomes comparable to the time spent running the iterative Jacobian algorithm.

VI. Conclusion

We proposed a database method for finding approximate inverse kinematic solutions for robots that have less degrees of freedom (DOF) than the space in which their end-effectors move. We compared this method to the naive approach of simply using the iterative Jacobian method from a single-pose and found that our proposed method outperformed the naive approach when finding exact IK solutions that the robot was known to be able to reach. It was also shown that this method can get close enough to at least one of the targets in a grasping set 96.6% of the time. We investigated the time it took to solve and found that the database can scale to handle large number of poses without a major change in runtime performance.

The proposed algorithm was unable to get approximate IK solutions that were close to the target in the general case, due to the kinematic structure of PackBot. Our approach is limited to working within that scope and requires a number of potential targets in order to have a high chance at success. Otherwise, a high tolerance for failure is necessary. This is still useful and allows our solver to integrate with algorithms, such as grasp sets that assume that the robot is able to reach 6-DOF poses without having to modify them to support a special 5-DOF parametrization.

References

[1] N. Makondo, J. Claassens, N. Tlale, and M. Braae, "Geometric technique for the kinematic modeling of a 5 dof redundant manipulator," in *Robotics and Mechatronics Conference of South Africa (ROBOMECH), 2012 5th*. IEEE, 2012, pp. 1–7.

[2] S. Sharma, G. K. Kraetzschmar, C. Scheurer, and R. Bischoff, "Unified closed form inverse kinematics for the kuka youbot," in *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, May 2012, pp. 1–6.

[3] R. Deimel and O. Brock, "A compliant hand based on a novel pneumatic actuator," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 05 2013, pp. 01–07. [Online]. Available: http://www.robotics.tu-berlin.de/fileadmin/fg170/Publikationen_pdf/2013-icra13_Deimel_Brock.pdf

[4] T. Murcko, "Soft pneumatic hand," August 2013. [Online]. Available: http://arc.wpi.edu/#projects-14

[5] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, August 2010. [Online]. Available: http://www.programmingvision.com/rosen_diankov_thesis.pdf

[6] ——, "Openrave website," March 2013. [Online]. Available: http://openrave.org/

[7] ——, "Ikfast documentation," March 2013. [Online]. Available: http://openrave.org/docs/latest_stable/openravepy/ikfast/

[8] B. W. Satzinger, J. I. Reid, M. Bajracharya, P. Hebert, and K. Byl, "More solutions means more problems: Resolving kinematic redundancy in robot locomotion on complex terrain," in *Submitted to IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014. [Online]. Available: http://www.ece.ucsb.edu/~katiebyl/papers/Satzinger_IROS_2014.pdf

[9] S. Arya and D. M. Mount, "Ann: A library for approximate nearest neighbor searching," May 2014. [Online]. Available: http://www.cs.umd.edu/~mount/ANN/

[10] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching in fixed dimensions," College Park, MD, USA, Tech. Rep., 1995.

[11] M. Muja and D. G. Lowe, "Flann - fast library for approximate nearest neighbors," May 2014. [Online]. Available: http://www.cs.ubc.ca/research/flann/

[12] ——, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application VISSAPP'09)*. INSTICC Press, 2009, pp. 331–340.

[13] D. Berenson, "Motion planning for articulated robots 1," p. 15, May 2014. [Online]. Available: http://users.wpi.edu/~dberenson/courses/rbe595planning/lectures/Larms1.pdf