Constrained Tool Planning in Work Space Using Inverse Jacobian Control

Adam Cantor, Jonathan Martin, Leo Keselman, Stewart Butler

***Abstract*—This report details an approach to known object interaction with a Schunk Arm. A motion planning theorem is applied with a focus on workspace control. The implementation of this system was done in the GRIP/DART simulation environment designed/used by the Golems Lab at Georgia Tech.**

I. Introduction

The purpose of this work is to create a methodology for interacting with known objects in a loosely defined workspace. This goal is directly in line with the DARPA robotics challenge to apply humanoid robots to search and rescue applications by using tools and exploration algorithms.

No single robot can be equipped to deal with all environments equally well. A general purpose robot, given a concrete goal for which it is not configured correctly, must be able to use objects from its environment to reach its goals. In an anthropocentric environment this means that correctly wielding tools designed for human use gives a robot a degree of operational flexibility that would be otherwise impossible.

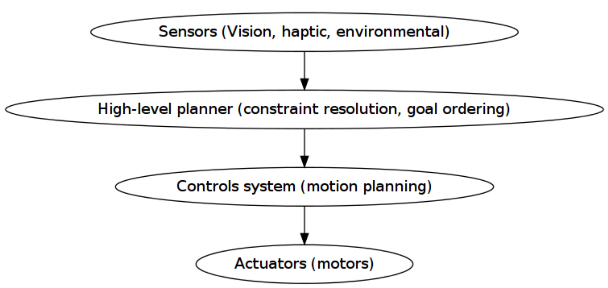

Fig. 1: Layer model of robot control systems.

Using human-configured tools requires multiple layers of control and planning, as shown in figure 1.

1) Sensors - The robot must be capable of discerning its environment, discovering what constraints that environment will impose upon it, and constructing subgoals in order to accomplish its target task. In an un-augmented anthropocentric environment, this means having a sufficiently sophisticated computer vision system (visual, radar, lidar, etc) to map the environment, recognize objects in the environment, and classify them as a given tool. It also means using haptic feedback (if possible) and torque feedback from the actuators during manipulation tasks as part of a control loop to ensure that the manipulated objects are behaving according to plan.
2) Planner - The robot needs a high level planner in order to take a goal state, break it into sub goals, match those sub goals with tasks it is able to handle, resolve any dependency conflicts, and order those goals into as efficient a plan as possible.
3) The control system is responsible for implementing each of the tasks necessary to accomplish a given sub-goal. This involves things like the lower-level motion planning necessary for manipulation, including workspace control.
4) Actuators - The robot must be capable of using its actuators to accomplish commands sent to them from the control system. Physically, one must be able to manipulate the tool in question. General purpose manipulators are not generally well suited for using tools designed for humanoid hands, and most commonly available humanoid hands have issues with grip strength, digit control, haptic sensing, and friction.

This document describes an effort to explore the challenges associated with general purpose tool usage from within a simulated environment. To reduce the scope of an otherwise unmanageable problem, the goal was simplified to be the control system for a single simulated Schunk LWA-4 arm, a 7-degree-of-freedom arm with an attachable manipulator. A screwdriver was chosen as the target tool, since it is the simplest tool that could be easily manipulated by a Schunk arm and still require a complex set of actions. In order to remove the necessity for a complicated sensor system, the work was done using the GRIP/DART simulation environment used by the Humanoid Robotics laboratory at Georgia Tech.

Complications arose during the work which prevented all our goals from being reached. The implementation of the vital control elements of the proposed system was unusable due to bugs in the implemented workspace control. These were fixed to a usable point, but there was not enough time remaining to integrate the high level planner into the decision loop. As such, the robot is capable of all the physical manipulation tasks needed to reach its goal state, but those states must be manually traversed by a human in order for the simulation to work. These issues will be further discussed in later sections.

II. Related Work

Tool manipulation in workspace is very common in both research and industry. However, most of this tool manipulation is with specialized tools that are not designed for human manipulation, and the robots are generally used in environments where the state space is largely deterministic – the planner for the robot can safely assume that a part will enter the assembly line at a certain time, reach a certain location in a certain orientation, and if there are any deviations from this expected behavior it will simply stop and alert a human of a problem.

What made our planned approach unique is that we account both for the motion planning for the robot as well as the constraints of the tool. For example, when driving the screw into the target block, we must follow the screw with the driver (held in the manipulator) as it threads into the target block by matching ‘downward’ translation with the rotation of the screw driver, then back the tool out and repeat until the screw is settled into the block. It also took into account the set of actions required to set up the environment – finding the screw, placing it in the correct location for tool use, finding the tool, and finally driving the screw into the target block.

Our chosen method of control is Resolved Rate Trajectory Planning, also known as pseudo-inverse Jacobian control. The principles behind Resolved Rate Trajectory Planning are well known in controls circles. The theory behind the implementation is not extremely complicated, making it an ideal method for workspace control of high-dimensional robots.[4] The fact that it is a trajectory-following techinque makes it critical to our work, since we do not wish for the robot to deviate from our desired trajectory for risk of collision with the workpiece or losing control of the tool.

Paul reviewed kinematic equations for simple manipulators like the one used here and explained how to determine the Jacobian for these manipulators.[3] Whitney investigated resolved rate trajectory planning and applied it to control manipulators under task constraints.[5] Cheah described how Approximate Inverse Jacobian control could be used in dynamic environments to provide workspace control.[2]

III. Methods

A. Simulation Software

This project used the DART/GRIP visualization system developed by the Golems lab at the Georgia Tech center for Research in Intelligent Machines. It is especially developed for testing algorithms on rigid body manipulators. Although the work was meant to be used on a mobile robot, it is reasonable to assume that in the tool use domain, the mobile manipulator would be stationary and the work would be done by a 7 DOF arm.

In DART/GRIP, the world is made up of objects and robots. Aside from a model, each object is described by its x,y, and z coordinate and made of links and revolute joints. The angle of each joint can be its rotation about the x,y, and z axis (roll, pitch, yaw). Robots are modified in the world inspector, and the change would automatically be reflected on the configurations of all the following links.

This visualization software was chosen because it was optimized for manipulator motion, because it was already somewhat familiar to the research team and also because its source code was available in case any changes needed to be made. It is also open source and available to anyone wishing to replicate the results discussed here.

B. Objects

The manipulator for this project was the Schunk 7 degree of freedom arm with a hand. Although this is strictly a standalone manipulator, it simulated the 7 degrees of freedom and the hand on a potential mobile robot's arm. The manipulator is shown in Figure 2.

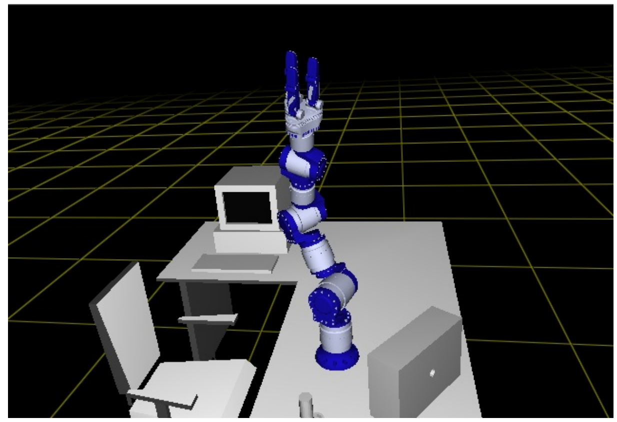

Fig. 2: Manipulator model

The block, screw, and screwdriver were developed in Solidworks and imported into the world. These are shown in one particular configuration in Figure 3.

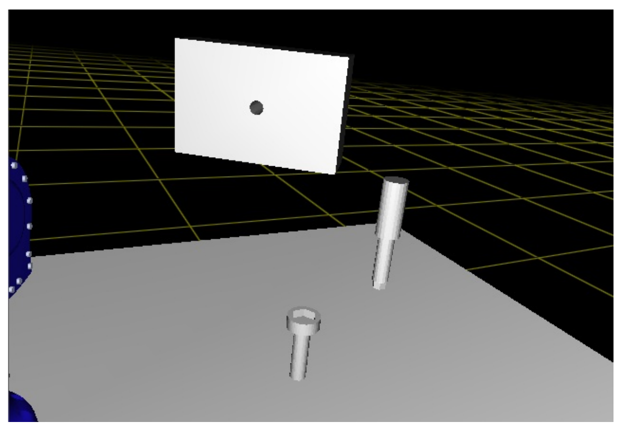

Fig. 3: Image capture of simulated environment, with objects.

C. High Level Planning

The high level plan for screwing in a bolt is designed as follows:

1) Locate screw - Align the “palm” of the manipulator with the top of the head, such that the plane of palm is parallel to the plane of the bolt head. In our case, the palm had to be slightly above the screw to be able to grasp it. Of course this offset, if necessary, will be dictated by the particular application. (See Figure 4)
2) Grasp the screw - Once the end effector and the bolt are aligned, close the fingers until they touch the screw. In a physical application, one would close the fingers until some force sensor on the fingers returns a certain

value. In our case, we simply closed the fingers until a collision was detected. Collision detection was provided by DART/GRIP.

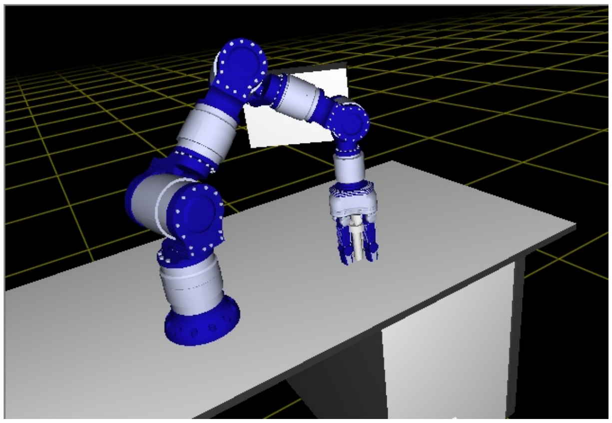

Fig. 4: Arm locating the screw object.

3) Locate the goal - Once the end effector grasps the screw, move both to the goal configuration. In our case, this was the hole in the block.

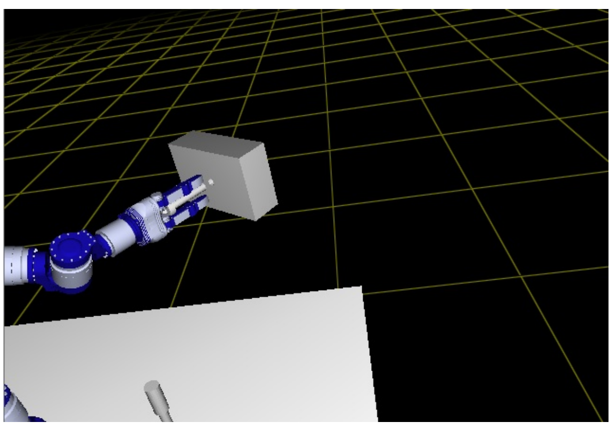

Fig. 5: Arm locating the block.

4) Start screwing the bolt - Just use the arm to screw in the bolt initially so that it stays in place while the screwdriver is retrieved. (See Figure 6)
5) Release the bolt - Leave it inside the whole, stationary.
6) Locate the screwdriver - Align the plam of the manipulator with the base of the screwdriver with an offset.
7) Grasp the screwdriver
8) Align screwdriver tip with bolt head - Here, there should be no offset between the tip and the bolt head. In a physical application, a force sensor on the hand could be used to determine when the screwdriver tip is making firm contact with the bolt head. In our case, we aligned the screwdriver and the bolt, then moved it closer until a collision was detected.
9) Screw in the bolt.

D. Low Level (Motion) Planning

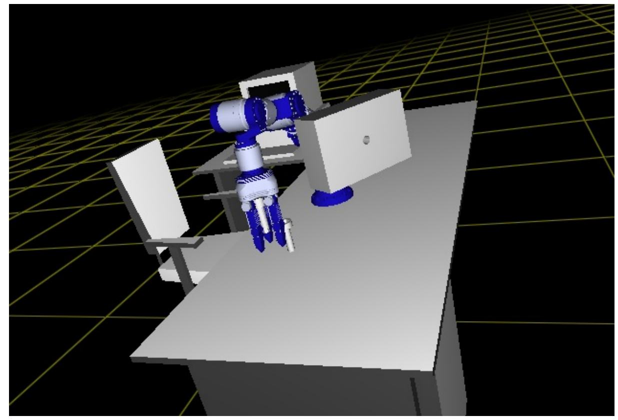

Fig. 6: Arm wielding the screw driver.

1) Resolved Rate Trajectory Planning: Resolved rate trajectory planning, or pseudo-inverse Jacobian control, was used to move the manipulator from a current world configuration $\mathbf{x_i} = \left[x_i y_i z_i R_i P_i Y_i\right]^\top$ to a target goal configuration $\mathbf{x_f} = \left[x_f y_f z_f R_f P_f Y_f\right]^\top$ in a linear fashion. The world coordinates are described as 6×1 vectors of X,Y, and Z positions with corresponding Roll, Pitch and Yaw.

Resolved rate trajectory control stems from 1, where $\mathbf{V_e}(t)$ is the end effector velocity at time t, α is the joint space position vector describing the current configuration of the robot, and $\dot{\alpha}$ is the joint space velocity vector.

$$\mathbf{V_e}(t) = \mathbf{J}(\alpha)\dot{\alpha} \tag{1}$$

This can be rearranged into 2.

$$\dot{\alpha} = \mathbf{J}(\alpha)^{-\mathbf{1}}\mathbf{V_e}(\mathbf{t}). \tag{2}$$

Thus, if the inverse Jacobian and desired end effector trajectory are known, it is possible to make a differential equation for the joint angles of the manipulator. In this case, the trajectory is to be linear, so $\mathbf{V_e}(\mathbf{t})$ was a constant 6×1 vector equal to $\mathbf{x_f} - \mathbf{x_i}$.

2) Extracting a target coordinate: The world coordinate of a given node is generated from DART/GRIP as an affine transformation described by the 4×4 homogeneous coordinate matrix $C = \begin{bmatrix} & \mathbf{R} & & \mathbf{X_{xyz}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$, so the 6×1 XYZRPY vector representation must first be extracted.

$\mathbf{X_{xyz}}$ is a 3×1 vector describing a point in 3-space, and does not require any manipulation. $\mathbf{R}$ is a 3×3 rotation matrix, so we convert this to roll-pitch-yaw as shown in equation 3:

$$\mathbf{X_{rpy}} = \begin{bmatrix} \tan^{-1}(\frac{R_{2,1}}{R_{2,2}}) \\ -\sin^{-1}(R_{2,0}) \\ \tan^{-1}(\frac{R_{1,0}}{R_{0,0}}) \end{bmatrix}, \tag{3}$$

This yields our target coordinates $\mathbf{x_f} = \begin{bmatrix} X_{xyz} \\ X_{rpy} \end{bmatrix}$.

3) Inverting the Jacobian: A direct inverse of the Jacobian is not possible in our case, as our manipulator had 7 degrees of freedom producing a 7×6 matrix. Instead, a Moore-Penrose pseudo-inverse was calculated according to equation 4.

$$J(\alpha)^{\dagger} = J(\alpha)^{\top}(J(\alpha)J(\alpha)^{\top})^{-1} \tag{4}$$

From this we compute for each time step the change in joint angles via equation 5:

$$\dot{\alpha} = J(\alpha)^{\dagger}V_e(t) \tag{5}$$

This is added to the previous time step's joint configuration and the robot is updated. The process continues until the global coordinates (both rotation and translation) of the end effector of the manipulator is less than a threshold distance ϵ of the desired position. In other words,

$$\|\mathbf{x_f} - \mathbf{x_i}\| < \epsilon \tag{6}$$

4) Joint limit avoidance: This equation for the joint velocities is not always well behaved. In order to improve the results, we will use a variation of the joint-limits avoidance strategy described in [1].

We now know from 5 the regular form of our velocity equation used to generate a trajectory toward our primary task. We will now augment that with an additional secondary task that will bias the trajectory towards keeping the joints as close to their zero points as possible.

First, we augment equation 5, so our equation describing the change in joint angles is now 7.

$$\dot{\alpha} = \mathbf{J}(\alpha)^{\dagger}\mathbf{V_e(t)} + (\mathbf{1} - \mathbf{J}(\alpha)^{\dagger}\mathbf{J}(\alpha))\mathbf{h} \tag{7}$$

The vector h is our secondary task, multiplied by $(\mathbf{1} - \mathbf{J}(\alpha)^{\dagger}\mathbf{J}(\alpha))$, the orthogonal complement of J(α). The result is a projection of $\mathbf{h}$ into the null space of $\mathbf{J}(\alpha)$, a "virtual motion" which results in a bias toward the secondary objective.

The secondary task is constructed as $\mathbf{h} = \nabla z$, where z is a fitness function described in 8.

$$\mathbf{z} = \frac{\mathbf{1}}{\mathbf{2}}(\alpha - \bar{\alpha})^{\top}\mathbf{W}(\alpha - \bar{\alpha}) \tag{8}$$

The variable $\bar{\alpha}$ is the joint position vector describing the mid-joint position; in our case, this is an all-zero vector, as the entire arm is constructed from symmetrically revolute joints centered around zero.

The matrix $\mathbf{W}$ is a diagonal weight matrix describing the acceptable deviation from this zero point – if it ;is desirable for a particular joint to have a greater latitude during movement than another, altering the corresponding row in the weight matrix will generate that effect. In our case, an identity matrix was used, as we only needed a general bias away from the joint limits.

The effect is that $\mathbf{h}$ describes a gradient where the zero positions on each joint is a minima, growing steeper as you progress further away from this center point. When projected into the null space of $\mathbf{J}_{\alpha}$, it will cause joints that are otherwise unconstrained by the target motion to descend toward their zero positions, preventing the robot from approaching too close to the joint limits and preventing some of the anomalous behavior produced by unconstrained inverse Jacobian control.

E. Object manipulation

In the real world, the actual manipulation of an object is a trivial task – once you grasp the object, it will behave as defined by the laws of physics, and can be treated as an extension of the end effector. However, the DART/GRIP simulation package does not currently support true connection of manipulated world objects as parts of the kinematic model of the robot, so the position of the world object must be manually updated according to the position of the end effector that is "grasping" it.

Maintaining the relationship between the end effector and the grasped object is difficult using XYZ-RPY coordinates since any accidental change in the order of rotation will cause resulting transform to change. However, describing the different positions and orientations of the end effector and target object as the 4×4 matrices of homogeneous coordinates describing the affine transformation used to take them from the global origin to their current global coordinates simplifies the task immensely.

Instead of making multiple operations to transform the grasped object via relative rotations and translations from its previous position, we need now only describe the affine transformation $\mathbf{R}$ which takes an object at coordinates $\mathbf{O}$ to the end effector coordinates $\mathbf{E}$, as calculated in equation 9.

$$\mathbf{R} = \mathbf{E}^{-1}\mathbf{O} \tag{9}$$

Subsequently, updating the position of object $\mathbf{O}$ to $\mathbf{O'}$ is a simple matrix multiplication of the new end effector location $\mathbf{E'}$ by the relationship $\mathbf{R}$, as shown in equation 10.

$$\mathbf{O'} = \mathbf{E'}\mathbf{R} \tag{10}$$

Setting the new location requires transforming the homogeneous coordinates back into global XYZ-RPY coordinates, as DART/GRIP do not currently expose a function for setting an objects position using homogeneous coordinates, but performing the transformations with homogeneous coordinates greatly reduces the risk of error.

F. Task motion – driving the screw

Once the screw is in place, the screw driving motion is performed to keep it stationary at the block while the screwdriver is retrieved. The screw driving motion pattern consists of a rotation of the end effector about its axis by 60 degrees, a release event, anti-rotating the end effector 120 degrees (for a net -60 degrees away from the zero point), regrasping, and repeating as necessary.

A similar motion pattern is used when the screwdriver is in use, though with a significant offset to compensate for the longer tool. Though currently unimplemented, a second modification to the motion pattern when the tool is in use is a motion away from the tool during the anti-rotation rather than a release and re-grasp motion that leaves the tool suspended in midair.

IV. Experiments

We were unable to perform any rigorous experimentation due to the broken inverse Jacobian workspace controller. However, during our attempts to debug the controller and lower motion planning, we did observe some extremely interesting behaviors in the failures produced by the inverse Jacobian controller producing bad output.

The first problem we developed appeared after adding Roll, Pitch, and Yaw to the target coordinates. If the target coordinates was near the outside of the arm's range of motion, it would attempt to drive its joints past their theoretical limits – opening up the inspection tab in GRIP showed occasional joint rotations in excess of ± 4000 degrees, and typical rotations of at least ± 200 on the primary joints. Figure 7

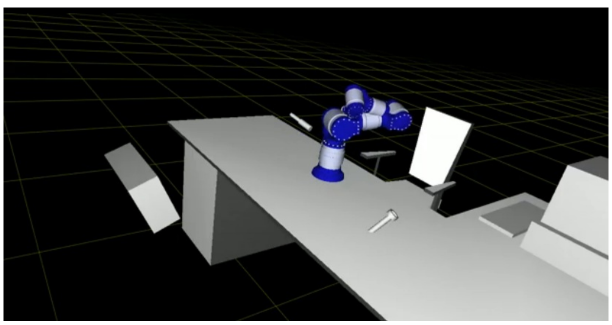

Fig. 7: Failure mode without joint limits

At first we believed this to be a result of the objects being placed outide of the end effector's range of motion, but when the orientation constraint was relaxed the controller would correctly find a solution in just the X,Y,Z axes. This is the point at which we relaxed the threshold for a correct solution, and our results improved significantly.

However, after a more complicated set of motions the arm would occasionally contort itself into a configuration that caused it to detect the shortest path to the destination coordinate to take it through its joint limits. This resulted in similar behavior as the previous figure depicted, and was prevalent enough that it still prevented us from completing the entire manipulation procedure. Further research yielded the technique dicussed in section III-D4. Implementation took quite a lot of further iteration, but when the necessary modifications had been made we activated the path finding routine again and found the arm contorting worse than before, as shown in figure 8. This behavior was caused by an inversion of the gradient values for the secondary objective, which effectively caused the arm to run as far away from its neutral joint position as possible.

After fixing this behavior, the workspace controller appears to work correctly, but no time remains for the implementation of the high level planner to permit further experimentation.

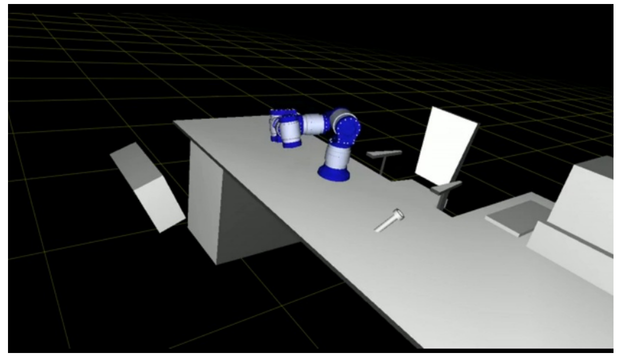

Fig. 8: Failure mode with inverted gradient joint limits

V. Analysis

A. Completeness

The main underlying principle behind our motion planning was Resolved Rate Trajectory control, as described in section III. This strategy can never be shown to be complete. It would not be able to find a goal if the start or end configuration was at a singularity. A singularity is any location in the workspace where the Jacobian loses rank. In this case, the pseudo-inverse becomes very large(inverse of a very small number). The algorithm then returns inaccurate joint velocities. This is really just a mathematical problem and is a shortcoming with the planning strategy. Singularities occur when joints are directly aligned and thus generally tend to happen near the boundaries of the workspace. Thus, Jacobian control can not be guaranteed to find a suitable plan if it exists.

B. Optimality

Jacobian control can be thought of as optimal in that sense that it follows the desired trajectory arbitrarily closely. It depends on how small of a time step is taken in integrating the differential equation for $\dot{\alpha}$. This again hinges on the start and end configurations not being singularities. In this case, the Inverse Jacobian method will not find a solution, much less an optimal one. However, if the start and end configuration are safely within the reachable workspace and a specific trajectory is necessary, this method can be very useful.

C. Efficiency

Jacobian control is quite efficient in time and space. In both of these parameters, complexity is linear to the length of the desired path. The most time and space consuming part is the inverse of the Jacobian. In our case, this always takes the same amount of time because the size of the Jacobian is always 6x7. For more degree of freedom arms, the Jacobian would have more columns, but never so many that the computation of the inverse becomes problematic.

D. Summary

Overall, the algorithms used were chosen for efficiency instead of optimality and completeness. In practice, true optimality is rarely a high priority — "second best" plans are usually good enough, often with a large enough gain in speed for real time applications. For our particular application, optimality was a low priority and thus the Jacobian method was deemed sufficient. It's also worth noting that this method is relatively easy to implement compared to other algorithms accomplishing the same thing. No inverse kinematics are required and the only lengthy part as far as setup is the synthesis of the Jacobian.

VI. Discussion

Driving a screw seemed like a reasonable task for a first experiment with constrained tool use with anthropocentric tools, since such a task is easy to decompose into a subset of simple goals. General tool planning is a hard problem with a very high computational complexity; by breaking down the problem into highly specialized subgoals, we drastically reduce the scope of the problem, but also make solutions inapplicable to other tool problems.

Though we were quick to reduce scope away from the impossible problems, we were unable to correctly estimate the amount of time required to debug our workspace control system. Without some form of control system, we were left unable to implement the higher level logic to get the desired task-planning behavior. After the demo/presentation, we were able to successfully integrate joint-limit avoidance into the Jacobian control system, which yielded proper low-level motion planning, but did not leave enough time to generalize the motion tasks and implement the high-level planner.

The inverse kinematics needed to orient tools properly was much more complicated than we expected, especially in an application with such tight constraints on the final position and orientation of the end effector (aligning the screw into the hole correctly). RRTs were sufficient when a start and end *jointspace* configuration is known to generate the desired workspace configuration, but no solution is possible when only the desired end effector position is known without some form of inverse kinematics.

Our problems with the workspace control system started when we attempted to integrate the Roll, Pitch, and Yaw coordinates into the gradient descent behavior. Augmenting the linear Jacobian with the angular Jacobian (turning it from a $3\times$ matrix into a $6\times n$ matrix) was simple enough, but resulted in very bizarre behaviors. The initial problem was an unclear conversion between degrees and radians, but after that was fixed the behavior still resulted in the robot clipping through itself and the table. With further debugging we noticed that our distance calculation was causing the issues, as the threshold was not large enough to permit an acceptable offset in all six dimensions so the controller would thrash as it approached its solution. This improved performance, but did not solve the fundamental problem.

We eventually narrowed down the underlying issue to the lack of joint limits. The motion planner would ignore the joint limits and, in searching for the correct joint-space configuration to reach its destination, would contort itself into impossible positions. After this was discovered, the joint-limit avoidance method described by Baron [1] was implemented and behavior stabilized.

At this point, the only remaining tasks are the integration of a high-level planner – easily done by integrating FF or another domain planner into the current decision loop – and working out the few remaining bugs in the coordinate system (offset from the tools, etc). Given an additional few days to a week, these changes could be easily integrated.

References

[1] L. Baron. A joint-limits avoidance strategy for arc-welding robots. In *Int. Conf. on Integrated Design and Manufacturing in Mech. Eng*, pages 16–19, 2000.

[2] C.C. Cheah, M. Hirano, S. Kawamura, and S. Arimoto. Approximate jacobian control for robots with uncertain kinematics and dynamics. *Robotics and Automation, IEEE Transactions on*, 19(4):692–702, 2003.

[3] R.P. Paul and B. Shimano. Kinematic control equations for simple manipulators. In *Decision and Control including the 17th Symposium on Adaptive Processes, 1978 IEEE Conference on*, volume 17, pages 1398–1406. IEEE, 1978.

[4] R. Samuel and SR Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *Unpublished survey. La Jolla, CA, USA: University of California, San Diego. http://math. ucsd. edu/˜ sbuss/ResearchWeb*, 2004.

[5] D.E. Whitney. Resolved motion rate control of manipulators and human prostheses. *Man-Machine Systems, IEEE Transactions on*, 10(2):47–53, 1969.

Appendix

A. How much time did your group spend on this project in total? How much of it has gone to making mistakes, experimenting, etc. as opposed to direct implementation? Reflect.

We estimate we spent around 60 hours on this project. Much of the time was spent trying to understand the inner workings of DART/GRIP. There were a couple small bugs that took a lot of time to figure out. Another issue was that we didn't quite realize the limitation of the inverse Jacobian approach and we spent a lot of time trying to figure out why the plans were misbehaving, when the whole time it was because of singularities. Another issue was understand how to extract RPY information of the end effector. We weren't sure how the transform matrix was calculated so it was difficult to extract RPY from it.

B. How much time do you think it would take to complete your initial goal?

The initial goal was what we actually implemented plus a high level planner that would decide order of operations and so on. This would have involved putting the domain into PDDL, using one of the planners attempted earlier in the semester, and then integrating this with DART/GRIP. This is theoretically simple but the details may have taken another 40 hours or so.

C. Given what you have completed and comparing your work with other groups', what grade do you think you should receive?

Some groups had significantly more impressive results, but this may have been partially aided by their own work previously done for another objective. Overall, we think we deserve around a B.