Baxter Demonstration with VRep

Michael Shea
Mechanical Engineering
University of Illinois Urbana-Champaign
Champaign, Illinois
mjshea3@illinois.edu

Jacob Heglund
Engineering Physics
University of Illinois Urbana-Champaign
Urbana, IL
jheglun2@illinois.edu

***Abstract* — We report the progress made over the course of this past semester in ECE 470 - Introduction to Robotics. The project component of the class allowed us to implement forward and inverse kinematics, collision detection, and motion planning in the context of a V-Rep simulation of the Baxter Collaborative Robot. These methods were written in Python and interfaced with the simulator using the V-Rep Remote API. We also propose a final test scenario involving the Tower of Hanoi to show a culmination of the robotic methods we have learned.**

I. Introduction

We began the project by first choosing a simulator-programming language environment we would use for the rest of the semester. V-Rep was chosen because it includes as standard several models of real-world robots. Python was then chosen due to our previous experience with the language and rich libraries including an API that allows Python to seamlessly interface with V-Rep. Our initial goal was to have a robot thrown a ball into a basket within the simulation, so we chose the Baxter robot due to the possibilities the robot provided. We specifically wanted a robot with two arms attached to the same base with the ability for the arms to work in tandem. While basket shooting was not a feature we ended up implementing, it was the original inspiration for working with Baxter.

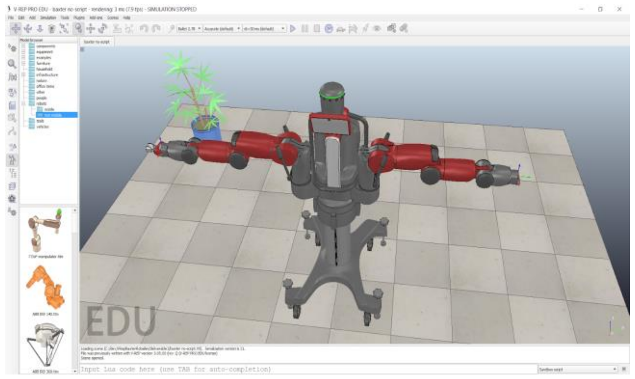

Fig 1: The Baxter robot as modelled in V-Rep

II. Forward kinematics

The first checkpoint of the project allowed us to learn the dimensions of our robot and become more familiar with the joints available on each arm. This was done for the goal of implementing forward kinematics in simulation. The product of exponentials method as shown in (1) was utilized for derivation and implementation of the forward kinematics of Baxter. The schematic we used to derive the forward kinematics is included below in *Fig 2.* During the implementation of forward kinematics in our simulation, we chose a starting pose to which Baxter would move at the beginning of each simulation. This was done to provide a constant starting position from which we could move the arms to desired joint-configurations. For ease of visualization, this was chosen to be a T-pose in which Baxter's arms are held perpendicular to the body (*Fig. 1)*

$$T(\theta) = e^{[S1]\ \theta 1}\ e^{[S2]\ \theta 2} \ldots e^{[Sn]\ \theta n} \qquad (1)$$

The parameters of the Baxter robot were learned directly from the model in V-Rep. These parameters allowed us to learn the spatial screw vectors used derive the forward kinematics. One unique challenge we faced with the Baxter robot was handling the rotating base joint. We treated this joint as part of each arm, which limited us to move one arm independently at a time.

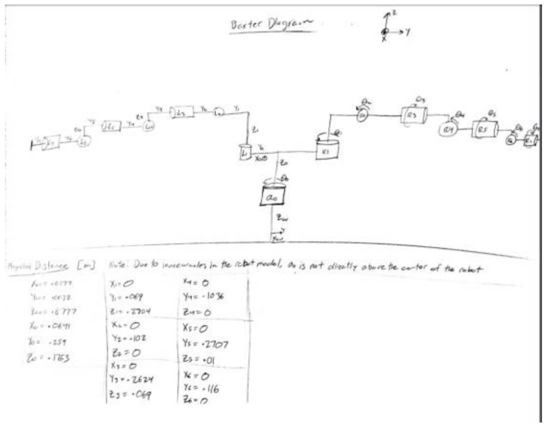

Fig 2: The forward kinematics of the Baxter robot. Included are the distances between joints and the direction of revolution.

III. Inverse Kinematics

The second checkpoint allowed us to implement inverse kinematics within our ever-growing codebase. We utilized normalized least squares as a method of minimizing the distance between our current pose and the desired pose of the end effector of one of Baxter's arms. The algorithm for inverse kinematics is analogous to a root finding algorithm and is included below.

Algorithm 1

```
Given: S, M, T_goal
Choose: initial guess for $\theta$
do...
    find T_current
    find [V] = log(T_goal (T_current)$^{-1}$)
    find J($\theta$) = $J^{-1}V$
    find $\theta = \theta + \dot{\theta}$
repeat until $||V|| < \varepsilon$
```

In cases where the inverse of the body Jacobian cannot be found, we can solve the normalized minimization problem shown below to find a $\dot{\theta}$ with minimized norm.

$$\min_{\dot{\theta}} (1/2)\,||J\,\dot{\theta} - V||^2 + (1/2)\,\mu||\theta||^2 \tag{2}$$

The solution to this problem is trivial, and the result is as follows

$$\dot{\theta} = (J^T J + \mu I)^{-1} J^T V \tag{3}$$

The implementation takes as input a 3D position in space for the tool end to reach. It then minimizes until the norm of the screw axis V is less a tolerance of .01 and norm of $\dot{\theta}$ is below a tolerance of .0001. These numbers were chosen to allow accurate positioning of the robot tool end without being sacrificing computational speed. If the algorithm does not find a set of joint-angles that satisfy these norms, a new randomly sampled initial position is chosen within the joint-angle constraints and the algorithm is run again. If this fails 7 times in a row, the position is determined to be unreachable. This is best shown in the case where the desired spatial position is too far for the robot to reach. The regularized least squares solution avoids singularities by forcing the term inside the inverse to have full column rank. For tool poses that have multiple arm-angle configurations that would reach the same pose, we completely ignore any other solutions. If the algorithm converges, we have no sense of how many other solutions would also work, and we don't really care either.

Once a set of joint angles that satisfy this minimization is found, our Python implementation first checks if the angles fall within the constraints of the robot. The constraints were found from the robot model.

IV. Motion planning

To implement motion planning, many of the equations from the previous sections where used. To start, we first came up with a way to characterize Baxter as well as outside objects with geometric objects. We chose to use spheres, as they are the easiest geometric shape to detect collisions with. All that is needed is the radius of two spheres, and their center points to detect if they are in collision, as illustrated in the equations below:

$$||P_{start} - P_{end}|| < r_{start} + r_{end} \tag{4}$$

Where P represents the position of a sphere in 3D space, and r is the radius of a sphere. In the case of the environment we used in VRep, we simulated outside objects with one big sphere each. For the robot Baxter, we placed spheres of varying sizes at every joint, including the main rotation joint that impacts the movement of both arms. The radius of the sphere for the main rotation joint was decided to be .4 meters, as it was a size that most accurately represented the physical surrounding of Baxter at the location. For the spheres located at the 7 joints of each arm, spheres of radius .1 meters were used. While a lot smaller than the rotation joint sphere, Baxter's arms contain less surface area when compared to the torso area. Also, with each arm having seven joints, seven small spheres for each arm was found to sufficiently represent the entire arm. Three Larger .4 meter radius spheres were used to represent Baxter's lower body, upper body, as well as its head. A complete set up of outside objects as well as the spheres

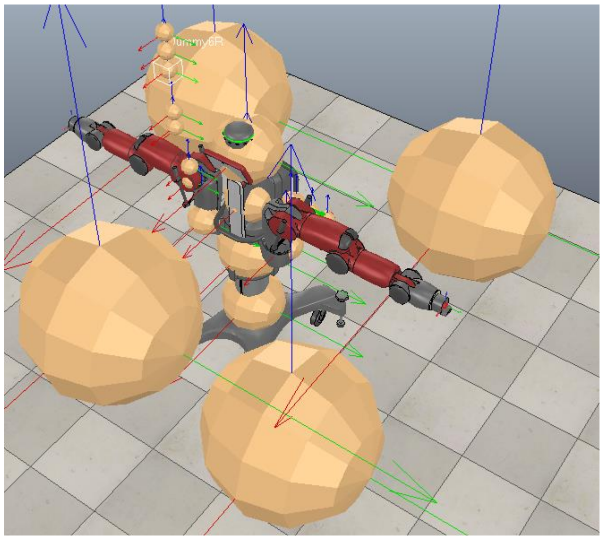

Fig 3: Spheres representing outside objects as well as Baxter's physical geometry. The spheres representing Baxter's right arms are vertical for visibility.

representing Baxter's physical presence is shown below. Note that whenever the phrase 'set of thetas' or similar terminology is used, it refers to the set of thetas that each of Baxter's joints are in for a given position.

Once the physical representation of Baxter was figured out using spheres, it was time to implement a motion planning algorithm. The process we used was as follows:

1. Prompt the user to enter the end X, Y, and Z coordinates for a given arm.
2. Use inverse kinematics to obtain the desired end thetas for the robotic arm to be in to achieve that position. Check if this position results in collision (process on how to do this is described later). If final position results in collision, notify the user and return.
3. Using the current position thetas as the starting position, calculate a collision free path to the end thetas calculated in step 2.
4. Repeat with the other arm.

The only new aspect for motion planning is in step 3, coming up with a collision free path from a set of start thetas to a set of end thetas. In order to do this, a tree structure was used with the following algorithm:

1. *Define a tree node structure that has three attributes. Those attributes are:*
    a. *A set of thetas describing the orientation of Baxter's eight joints (includes the rotation joint) for a given arm.*
    b. *A boolean value that states whether the tree is part of the starting location tree or ending location tree (meaning of this is clarified later on).*
    c. *A pointer to the current node's parent, if it has one.*
2. *Create two different trees, an initial position tree and an ending position tree.*
    a. *The root node of the starting position tree will contain the set of thetas from the starting position, a null pointer because it has no parents, and a Boolean value of true indicating it is part of the starting position tree. To check if a set of thetas for a specific arm results in collision, first calculate the resulting position of all non-static spheres described above. These are the spheres that are attached to joints besides the first rotation joint. To find the position of a sphere at a joint, act as if that joint is the end of the robotic arm, and then perform forward kinematics on the smaller theoretical arm to obtain the position. Loop through each non-static sphere, and see if it is in collision with any of the other spheres, including the static body spheres and outside objects spheres using equation 4.*
    b. *The root node of the ending position tree will contain the set of thetas from the ending position, a null pointer because it has no parents, and a Boolean value of false indicating it is part of the ending position tree.*
3. *Calculate a random set of thetas that does not have any values outside of any joints' range for a given arm and is not in collision with itself or outside objects.*
4. *Find a node from either tree that has the shortest Euclidean distance to the position obtained from the set of thetas calculated in step 3. The position can be calculated using forward kinematics described in the respective earlier section. Check if has a collision-free straight path to it. If it does, add the current node as the child to the closest found node, and, if not, discard the set of thetas and go back to step 3.*
    a. *In order to detect if there a collision along a straight-line path from one set of thetas to another, first determine how many sampling sets along the path is desired. In our case, we used the following formulas with an arbitrary small value for σ that indicated precision:*

$$Distance = ||\theta_{start} - \theta_{end}|| \quad (5)$$

$$Number\_Samplings = 1+Ciel(distance/\sigma) \quad (6)$$

    b. *Then create a loop for the number of samplings that was decided upon, and use the following formulas to come up with a set of thetas for each cycle in the loop, with s initially starting with value 0:*

$$s = s + 1/sampling \quad (7)$$

$$\theta_{iteration} = (1-s) * \theta_{start} + s * \theta_{end} \quad (8)$$

    c. *For each set of thetas from above, check if it results in collision using the method described in step 2a.*
    d. *If there is a collision detected at any of the samplings, there is no clear straight-line path between positions. If all samplings are collision free, then there is a clear line path.*
5. *For the node just added, find the closest node on the other tree based off Euclidean distance, and check if a collision free straight path exists using the method described in step 4. If it does, then both trees have been successfully connected and a collision free path from start position to end position has been achieved. If not, go to step 3.*

6. *Move the specified robotic arm along the straight-line paths of the nodes that connect the starting position and ending position.*

For the approach described above, there are some downsides that must be addressed. Because the physical properties of Baxter are represented using a limited number of spheres,

There are some areas that it will detect as a collision when there are none (when part of a spheres volume is not part of the actual robotic volume), and there will be areas that would result in a collision that it will not be able to detect (some of the physical robotic volume will not be shared by any spheres), resulting in a false positive. Due to the way the spheres were set up, and with theta limits set up as described in the inverse kinematics section, the false positive situation is very unlikely. We have yet to come across a situation where this has occurred, and we have tested dozens of different scenarios.

Sometimes, the user will enter a location that cannot be obtained with a collision free path. Most of the time this is when the end point results in a direct collision but can also occur when the points are just out of reach as well. The algorithm discussed above accounts for this in the kinematics step. If the desired end location results in collision, as described in the algorithm above, the user will be notified and the program terminates. If the point is out of reach, this is taken care of in the inverse kinematics algorithm again, and the reason for this is described in the respective inverse kinematics section above. Again, the user is notified and the program terminates.

Another important topic to discuss is the time required to run the program. In terms of the starting and ending positions, there is really no time variation if the ending position is not in collision or out of reach. But due to the fact that Baxter has seven joints to control each arm plus another rotation joint, the costliest part of the motion planning algorithm in terms of time is checking if there is a clear straight-line path between two nodes. The reason for this is that the location of each sphere must be determined using forward kinematics for each iteration along the path, and this is repeated for multiple nodes on average. Because each joint's sphere (besides the rotation joint's sphere) location is calculated with each step, the time to complete the calculation increasing exponentially with every new joint involved. On top of this, the calculation in the beginning to obtain the inverse kinematics for the ending position, as described above, also takes quite some time to find a set of thetas that are all within limit of Baxter's joints. Through multiple runs, we have found that it takes about 30 seconds to complete the entire algorithm for motion planning for a given arm position and desired end position, given the end position is valid.

V. Future Work

We now propose a plan to use the techniques we have learned for the purpose of completing a larger task. We plan to implement the Tower of Hanoi algorithm with some constraints. First, the user must be able to enter where the starting, middle, and ending location are out of many options. Second, the user will input the number of blocks, from a range of three to nine. There will also be many obstacles in the environment that Baxter will have to avoid. In order to achieve all of this, there are quite a few steps that we have to implement and figure out. These include but are not limited to:

1. Whenever Baxter picks up a block, use the volume of the moving block in the collision detection algorithm.
2. Implement Baxter's suction cup attachments to pick up and place the blocks.
3. Generating a variable number of blocks at desired locations based on user input.

VI. Acknowledgments

While completing all the work described above, we did not communicate with any faculty or staff for help with the project or code, outside of gaining inspiration from lectures of Professor Bretl. There were a few instances where Piazza post from other teams were used for inspiration and discussion.

References

[1] R.L. Williams II, "Baxter Humanoid Robot Kinematics", Internet Publication, https://www.ohio.edu/mechanical-faculty/williams/html/pdf/BaxterKinematics.pdf, April 2017.