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.

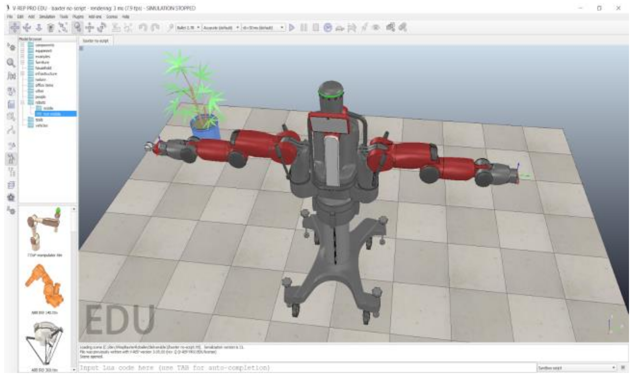


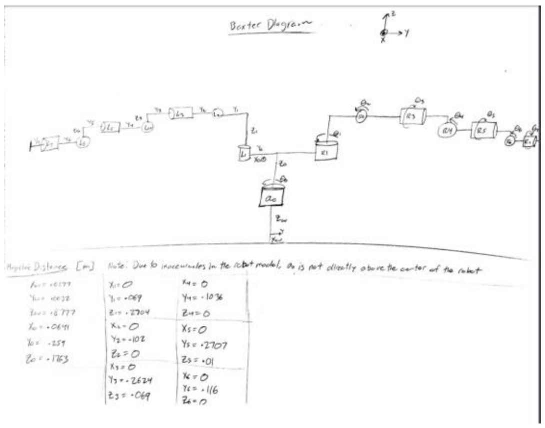
Fig 1: The Baxter robot as modelled in V-Rep

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.


Baxter Diagram
a0
Note: Due to inaccuracies in the robot model, a0 is not directly above the center of the robot

Absolute Distances [m]

Xw = .0177	X1 = 0	X4 = 0
Yw = .0032	Y1 = .069	Y4 = .1036
Zw = .8777	Z1 = .2704	Z4 = 0
X0 = .0641	X2 = 0	X5 = 0
Y0 = .259	Y2 = .102	Y5 = .2707
Z0 = .1763	Z2 = 0	Z5 = .01
	X3 = 0	X6 = 0
	Y3 = .2624	Y6 = .116
	Z3 = .069	Z6 = 0

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 $J(\dot\theta) = J^{-1}V$
    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 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.

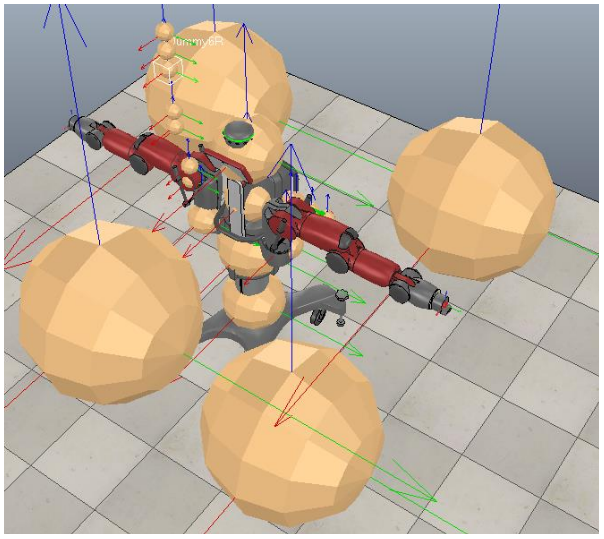


Fig 3: Spheres representing outside objects as well as Baxter's physical geometry. The spheres representing Baxter's right arms are vertical for visibility.

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:*

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

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

    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. Results

For the final part of this project, we were given the task to, "at minimum, … create a dynamic simulation (i.e., a simulation of real physics) in which at least one robot---with at least one arm that has at least six joints---moves at least one object (e.g., by pushing or grasping) from a random initial pose to a given final pose." To accomplish this task, while at the same time showcasing all work done described in previous sections, we decided to have Baxter complete the classic Tower of Hanoi problem with three and four blocks.

We first started by setting up the environment, which can be seen in figure 4. The setup consisted of a table which Baxter used as a surface to complete the algorithm, along with the number of cubes that were used in the algorithm.

The first step was modifying our collision detection functions. We represented each cube as a sphere that had the radius of half the side length of each cube, just as we did with Baxter's joints. We did not use spheres to represent the table's physical nature however, as it is not a completely symmetrical shape like the cubes are (which can therefore easily be represented as a sphere). We instead wrote another function that checks the location of all the spheres that represent Baxter's physical body and all the other outside objects and sees if any of those locations are within the bounds of the table's dimensions.

We then used one of the suction grippers that is available within VRep, and attached one to each of Baxter's arms. The process to turn on and off each suction cup was quite simple. All that was required was to send a message request to the handle tag of the object along with a 1 or 0, which represented turning on and off the suction cup, respectively. The child scripts of each suction cup, written a language called LUA (VRep's language), would then receive the signal and act accordingly. Instead of using the physics engine to pick up the object, the script searched for the closest item and then made the suction cup the parent of that object, so extra caution had to be taken to ensure the correct blocks were detected by the suction cup.

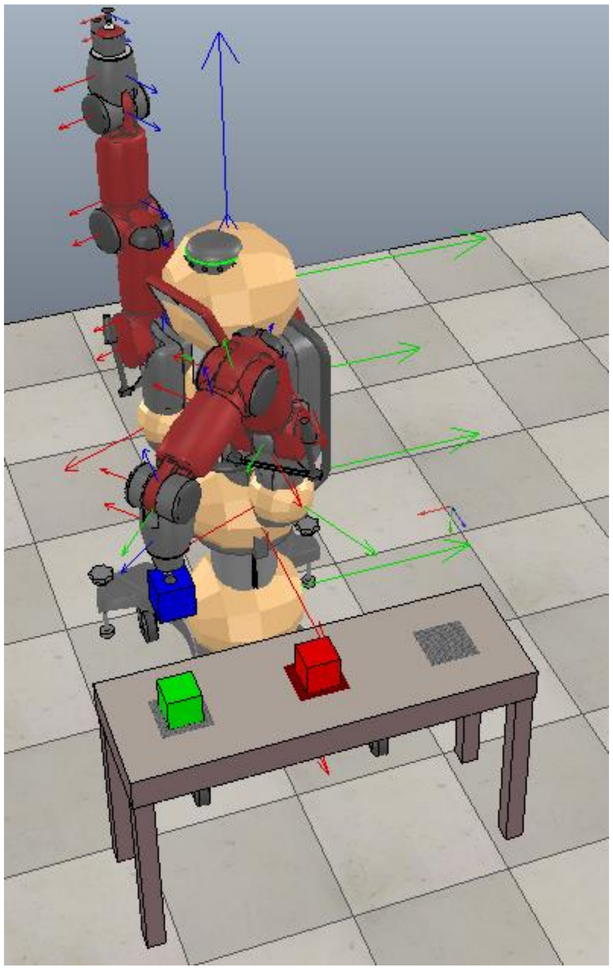


Fig 4: Baxter completing the Tower of Hanoi problem. The red pad on the table represents the ending position, and the other two grey pads are the starting and middle positions.

We then programmed the function for Baxter to complete the Tower of Hanoi. When the program is called, it asks the user to specify if they want Baxter to complete the three or four block version of the algorithm (figure 4 shows Baxter solving the three-block version), and then the user would also specify the starting, middle, and end locations for the algorithm within the limit dimensions of the table top.

The code then uses those locations to place pads to show the user visually the points they picked, and proceeds to solve the algorithm. Currently, the steps for the algorithm are hardcoded. The simple pseudocode for the Tower of Hanoi is as follows:

```
FUNCTION MoveTower(disk, current_position, desired_position , spare):
IF disk == 0, THEN:
    move disk from , current_position to , desired_position
ELSE:
    MoveTower(disk - 1, current_position, spare, , desired_position)
    move disk from current_position to desired_position
    MoveTower(disk - 1, spare, desired_position, current_position)
END IF
```

For each step of the algorithm, the function has Baxter move to the top of the desired location, move the tip of the suction cup down to the exact height of the block, either pick up or drop off the block, and then move back above the desired position. Each time Baxter is commanded to move to a specific location, the functions and algorithms described in the motion planning section are used to create a collision free path, which in turn also uses the forward and inverse kinematics algorithms described in the above sections as well.

The video of Baxter completing the Tower of Hanoi problem can be found following the link in reference [11]. Baxter could plan the paths each cube exactly as expected, with no reported collisions.

VI. Future Work

Given the time constraints for the video, we were able to implement and display a basic version of an environment that demonstrated inverse kinematics, path planning, and collision avoidance with a payload. We now put forward a plan for future extensions of the work presented in the video displayed during the end-of-semester party for ECE 470.

Our implementation of path planning was based on building trees stochastically in the space of joint variables. This often gave quick results a path, but it is often the case that a shorter path is desired for robotic arms to reduce overall energy use. Therefore, a future implementation would involve the RRT or RRT* as the motion planning algorithm. These algorithms have convergence guarantees in certain spaces, whereas a stochastic tree-building algorithm may struggle to find a path in very high dimensional spaces [2].

If we wanted to implement more modern techniques, we could also pursue a reinforcement learning solution to the path planning problem. Reinforcement learning provides a vast area of exploration and research that is only now able to be utilized due to the recent advances in deep learning [4,5,7,8]. Once we are in the realm of machine learning, there are several methods we could also implement to keep track of the optimal policy found by searching space of joint angles. One method would be to take a tabular representation of the action each joint angle should take at each possible position in the space of joint angles. However, this is often used in simple grid world problems rather than higher dimensional problems such as the path planning problem with Baxter. Instead, we would implement a Gaussian Process (GP) or Radial Basis Function (RBF) method to estimate the optimal policy and the cost function in the space of joint angles [3].

As another extension of this work, we could also utilize the camera features of V-Rep to give Baxter vision, and the ability to sense objects. In the context of the Tower of Hanoi, the application of visual sensing would allow the robot to first examine the workspace to determine the total amount of blocks that are to be moved, to autonomously find the “middle” and “end” points for the blocks given they are visually marked. We could even implement fiducial markers in the simulation so that the robot would autonomously determine the order in which the blocks need to be moved. This is a simple example, but it shows the power that a robot such as Baxter could have in a manufacturing environment.

As a final idea to extend this work, it may be the case that environment does not allow an arm to reach an object. For example, the right arm may not be able to reach an object it is supposed to pick up, and the left arm may not be able to place the object in the end location it is supposed to end at. If Baxter is able to hand objects between arms, the utility of the robot would greatly increase. This would be an interesting challenge, since the type of object transfer would depend entirely on the object being moved and the type of hands or grippers attached to the Baxter Robot. Handing objects between Baxter’s hands would nonetheless be a useful feature for a future implementation of our code.

We have discussed several routes of future research and implementation for our Baxter robot. These ideas are not the only possibilities, but they are obvious next-steps that could be taken given our experiences and existing codebase.

VII. 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.

[2] Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. The international journal of robotics research, 30(7), 846-894.

[3] Csató, L., & Opper, M. (2002). Sparse on-line Gaussian processes. Neural computation, 14(3), 641-668.

[4] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press.

[5] Pattanaik, A., Tang, Z., Liu, S., Bommannan, G., & Chowdhary, G. (2017). Robust Deep Reinforcement Learning with Adversarial Attacks. arXiv preprint arXiv:1712.03632.

[6] Atkeson, C. G., & Santamaria, J. C. (1997, April). A comparison of direct and model-based reinforcement learning. In Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on (Vol. 4, pp. 3557-3564). IEEE.

[7] Sutton, R. S., Barto, A. G., & Williams, R. J. (1992). Reinforcement learning is direct adaptive optimal control. IEEE Control Systems, 12(2), 19-22.

[8] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[9] Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.".

[10] Szeliski, R. (2010). Computer vision: algorithms and applications. Springer Science & Business Media.

[11] Heglund, J. April 30, 2018. *ECE 470 Final Deliverable: Tower of Hanoi with Baxter. https://www.youtube.com/watch?v=MNY7lrb8RS0&t=6s*