
AC 2011-609: LEARNING ROBOTICS THROUGH DEVELOPING A VIRTUAL ROBOT SIMULATOR IN MATLAB

Yang Cao, University of British Columbia

(Aug. 2007 - Present) Instructor, School of Engineering, University of British Columbia Okanagan Campus

(Aug. 2005 - June 2007) Postdoc, Industrial and Manufacturing Systems Engineering, University of Windsor

Learning Robotics through Developing A Virtual Robot Simulator in Matlab

Abstract

Due to the expensive nature of an industrial robot, not all universities are equipped with areal robots for students to operate. Learning robotics without accessing to an actual robotic system has proven to be difficult for undergraduate students. For instructors, it is also an obstacle to effectively teach fundamental robotic concepts. Virtual robot simulator has been explored by many researchers to create a virtual environment for teaching and learning. This paper presents structure of a course project which requires students to develop a virtual robot simulator. The simulator integrates concept of kinematics, inverse kinematics and controls. Results show that this approach assists and promotes better students' understanding of robotics.

1. Introduction

Robotics course is a very common and important course for electrical and mechanical engineering students. It is also a crucial course in the curriculum of mechatronic program which is becoming popular in many North America Universities. Robot itself is a perfect example of mechatronic system. Due to the complexity of the subject, teaching of robotics has always been challenging to instructors and at the same time, learning of robotics has always been a daunting task to students. Hand-on exercises are highly appreciated by students. Institutions with adequate funding are able to provide students hands-on experience through labs¹. More recently, more sophisticated virtual lab environments were created based on various real robots^{2,3,4}. There are many benefits of these virtual lab environments such as reducing the maintenance cost due to mishandling, providing flexibilities, collaborations in finishing the lab.

However, many schools don't have budget for a real industrial robot in assisting the teaching and learning of some fundamental concepts of robotics such as forward and inverse kinematics, and control. Designing virtual robots with useful visualization tool and instructions of operation can overcome these problems⁵. One major benefit of virtual robot simulator is its ability to create a visualization of the robot model and movement in a visual 3D environment. Hence student can gain a realistic experience in visualization and modeling robots. Visualization technique is a great education value and can help to reduce the analysis and study time while it helps for a deeper understanding of the teaching material. It is also very economical to the organization Using a robot simulator.

In the robot modeling and control course (ENGR486) for 4th year engineering students at University of British Columbia (UBC) Okanagan, a project-based learning is integrated into the course. The project requires students to develop a virtual PUMA560-type robot simulator. The simulator should demonstrate the concepts of forward and inverse kinematics, and basic control techniques such as PID independent joint control. The project is divided into 3 phases in synchronize with course progress. **The first phase** requires students to model a PUMA560-type of 6 degrees-of-freedom robot using Solidworks or an other CAD software. Details such as drive systems or electrical wiring are not necessary for this CAD model. This model with assembly of each link will be imported into MATLAB as patch objects. **The second phase** is to develop a graphical user interface which displays the robot configuration as 3D model, and provides

options of demonstrating independent joint motion (forward kinematics) and trajectory following of end-effector (inverse kinematics). **The third phase** is to develop a PID controller for each joint or a model-based nonlinear controller.

Students are exposed to popular engineering tools such as Solidworks and MATLAB in modeling and simulating their robots. Besides providing students hands-on experience of simulation, the project proves to be a great aid for teaching and learning the principles of robotics. The outcome of this approach is well accepted and highly rated by students.

The paper describes the details of the three phases mentioned above and how each phase makes dry mathematic theories alive with aid of computer software. The paper will also discuss possible improvements.

2. Structure of Course Project in Developing a virtual Robot Simulator

Step 1: Create your own robotic manipulator model.

This robot model has to contain 6 degrees-of-freedom with a spherical wrist. Students can create your own model using Solidworks, AutoCad Inventor or any other CAD software. The CAD model doesn't have to be fancy as long as it contains properly assembled parts and gives 6 degrees-of-freedom. If you are not interested in solid modeling, you can also download a CAD model from industrial robot manufacturer's website such as ADEPT, FANUC, KUKA, DESO, etc. Generally, manufacturers provide CAD model for commonly used softwares. Choose a robot model with detailed documentation.

For example, a PUMA560 (not exact shape) was created using Solidworks as shown below.

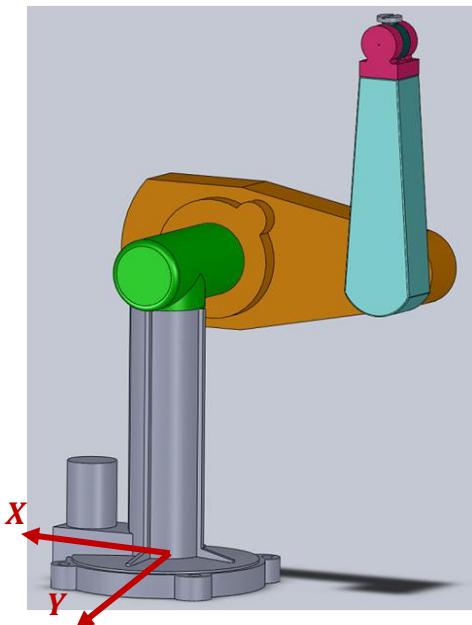


Figure 1 Simplified PUMA560 CAD model

Here are few things to note in this step:

- (1) When model the link 1 (base of the robot), make sure that the origin of the coordinate system for modeling coincides with the center of the bottom surface of link 1. And the bottom surface should be sketched in the “front plane” or XY-planes. Then the direction of extrusion of the link 1 is in the positive direction of z-axis. In actual PUMA560, the world frame has X-axis going through the joint 1 motor. Although for simulation, one can place X-axis in any directions.
- (2) When assembling all links, makes sure to use constrain condition to have the origin of part 1 (link 1 module) coincides with the origin of the coordinate system in the assembly space. You may need to “float” the part 1 first, apply the constraint and fix the part 1 in the end.
- (3) Using constraints to assemble the parts of the robot into the “READY” pose of the robot. PUMA560’s “READY” pose is shown in Figure 1, which is also the “zero position” (all joint angles are zero degree). In the next step, when we export the assembly to STL file. The STL files will contain the actual XYZ position information for each part in the “READY” pose.

Step 2: Create MAT-files for use in Matlab

Firstly, save the assembly of your robot model as “STL” files in ASCII format. Quoting from Wikipedia, “STL is a file format native to the stereolithography CAD software created by 3D Systems. This file format is supported by many other software packages; it is widely used for rapid prototyping and computer-aided manufacturing. STL files describe only the surface geometry of a three dimensional object without any representation of color, texture or other common CAD model attributes.” A STL file is basically a triangular representation of a 3D object.

When saving as STL files, select the output format as ASCII and also check the “Do not translate STL output data to positive space” to preserver originality of the data. See the Figure 2 for illustration when saving as STL files in Solidworks.

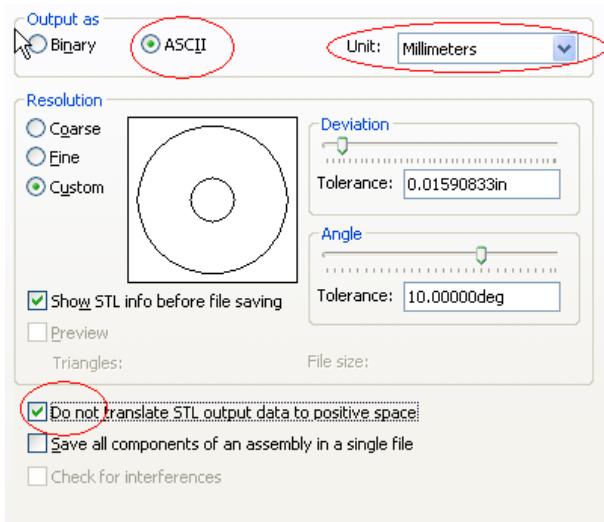


Figure 2 Saving options for STL file

Note that A list of STL files should be generated depending on the number of parts used in the assembly. These STL files contain geometric information of each part (i.e., X , Y , Z information).

Secondly, convert the STL-files into MAT-file using “Robot_CAD2MAT.m”. This m-file is a modified based on “cad2matdemo.m” which can be found on Matlab central (<http://www.mathworks.com/matlabcentral/fileexchange/3642-cad2matdemo-m>).

Open file “Robot_CAD2MAT.m” and fill in the file names as your designated file names. “Robot_CAD2MAT.m” reads each STL-file (corresponding to each link) and extracts data of faces (F), vertices (V), and colors (C) into three variables F_i , V_i , and C_i , respectively. The original size of $V1$ is $n \times 3$ where n is the number of vertices, 3 is the number of coordinates for each vertex (x , y , z). For homogeneous transformation, we need to add 1 to make it (x , y , z , 1). Thus we concat a column of 1 with n rows to matrix V_i to make it a $n \times 4$ matrix. Then the m-file saves these three variables F_i , V_i , and C_i , into a “struct” variable. For example, for link 1, we can have

```
>s1=struct('F1', F1, 'V1', V1, 'C1', C1);
```

Every link or part in the robot model will be represented using one “struct” variable. All “struct” variables will be saved into a MAT-file.

```
>save('your_robot_data.mat', 's');
```

Now this MAT-file can be loaded in the main simulation file and a “patch object” can be created corresponding to each link using the Matlab command “patch”. For example

```
>L1=patch('faces', s1.F1, 'vertices', s1.V1(:, 1:3));
```

The patch object will be used for computer visualization in Matlab environment. In the next few steps, one can modify the value of vertices by applying rotation/translation to corresponding variables (e.g., $V1$). Animated motion of the robot can be easily realized.

A few more words on the “patch” command. “patch” command is a convenient tools to create 3D graphics, which is built on window open GL. A sample usage of this command is shown above.

- “vertices”, $s1.V1(:, 1:3)$ is a $n \times 3$ matrix that contains all the vertices used to describe the link 1. n is the number of vertices, 3 is the number of coordinates for each vertex (x , y , z). Each row of this matrix corresponds to a vertex and each column to its x , y , z coordinates.
- ‘faces’ : $F1$ is a matrix that defines each face of the object in terms of the vertices that lie on the face. Each row of this matrix corresponds to a face, the column element of each row refers to the vertices that define the face.
- “L1” is a patch object created. We can use “set” command to change its properties and update the values of vertices after rotation/translation. For example,
`set(L1, 'facec', [0.717,0.116,0.123], 'EdgeColor','none');` %setting face and edge color

Step 3: Create an interface using Matlab GUI

Type “guide” to enter the Matlab GUI design environment. One can drag and drop various components (button, textbox, axis etc.) to create unique simulation user interface. The GUI should include the following functionalities

- ✓ Demonstration forward kinematics, inverse kinematics and at least one control implementation.
- ✓ For inverse kinematics, show the end-effector trajectory in real-time.
- ✓ Real-time outputs of current joint angles and end-effector’s position and orientation are preferred. The orientation can be represented using Euler’s angle (XYZ).

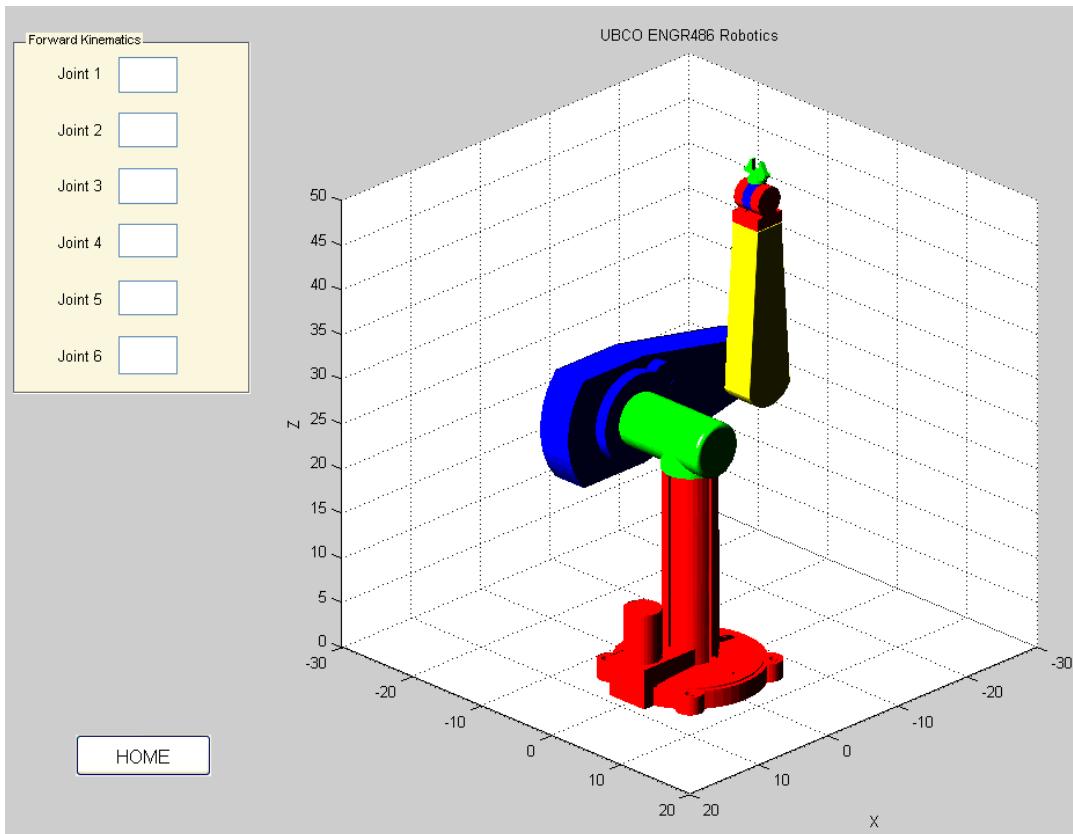


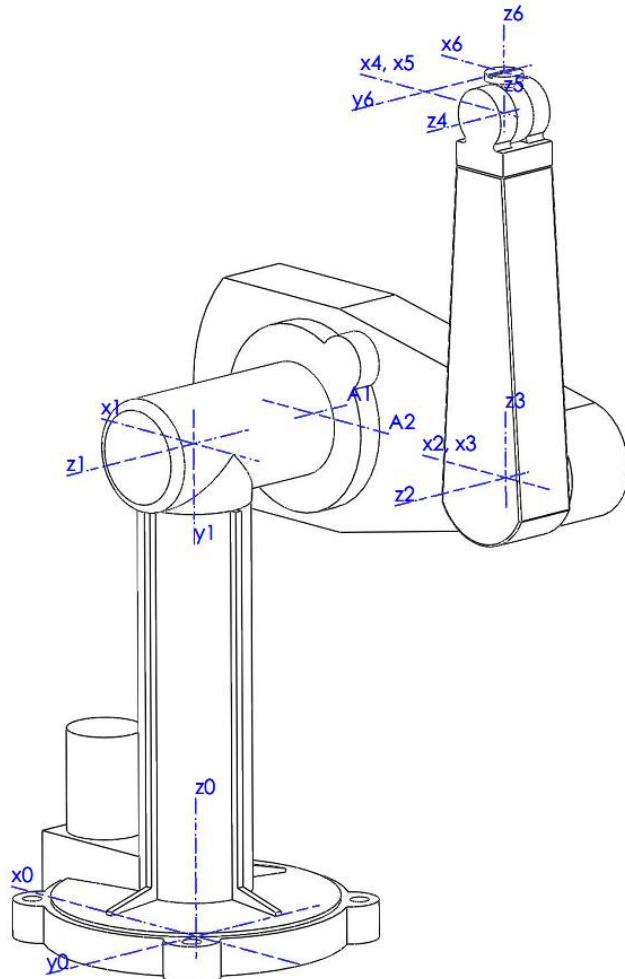
Figure 3 An sample simulator interface.

Step 4: Implement simulation of forward kinematics

- ✓ Obtain dimensions of your robot model
- ✓ Assign frames to the robot using DH convention and generate DH table.
- ✓ Realize the homogenous transformation matrix in Matlab based on the DH table
- ✓ Implement forward kinematics and demonstrate joint motion

The DH convention uses the “Distal system”^{6,7,8} (Conventional DH) as depicted in Figure 4.

Figure 4 shows how frames are assigned to the PUMA robot used for simulation and DH parameters are summarized.



PUMA robot arm link coordinate parameters				
Link	α_i	a_i	d_i	θ_i
1	-90	0	d_2	θ_1
2	0	a_2	0	θ_2
3	90	0	0	θ_3
4	-90	0	d_4	θ_4
5	90	0	0	θ_5
6	0	0	d_6	θ_6

$$a_2 = -16.39 \text{ in}, d_2 = -6.75 \text{ in}, d_4 = 19.58 \text{ in}, d_6 = 2.2 \text{ in}$$

Figure 4 Frame assignments and DH parameters for PUMA robot

Note that the DH parameters are based on a model described by Lee⁹. The actual PUMA560 robot has a small offset $a_3 = 0.8\text{in}$.

A homogeneous transformation matrix T_i^{i-1} relating the i th frame to the $(i - 1)$ th frame can be derived.

How to use homogeneous transformation matrix?

- ✓ Define matrices Link0, Link1, Link 2, ..., Link_n to contain all the vertices information for each link. Columns of these matrices are the homogeneous coordinates of vertices. Since Link0 is fixed, no change will be applied to this matrix.
- ✓ The frame 0 is designated as world frame. Homogeneous coordinates of vertices are all with respect to the world frame.
- ✓ Effect of rotation about joint 1 or rotation about $z_0 (\theta_1)$. Link1 to Link6 will all rotate about z_0 . Thus rotation of vertices on each link will be

$$T_z(\theta_1) \cdot \text{Link1}, \dots, T_z(\theta_1) \cdot \text{Link2}$$

$T_z(\theta)$ is the basic homogeneous transformation matrix for pure rotation about the current z -axis by an amount of θ .

- ✓ Effect of rotation about joint 2 or rotation about $z_1 (\theta_2)$. Link2 to Link_n will all rotate about z_1 . What we have is the homogeneous coordinates with respect to frame 0. Since this rotation about z_1 is with respect to current frame, we need to first transform the homogeneous coordinates of Link2 to Link_n into frame 1. Then apply the pure rotation about the current z_1 -axis and finally transform back to frame 0. The overall operation is

$$T_1^0 \cdot T_z(\theta_2) \cdot (T_1^0)^{-1} \cdot \text{Link}_i, \quad i = 2, \dots, n$$

- ✓ Effect of rotation about joint 3 (frame 2) or rotation about $z_2 (\theta_3)$. Link3 to Link6 will all rotate about z_2 . Similarly we need to first transform the homogeneous coordinates of Link3 to Link6 into frame 2. Then apply the pure rotation about the current z_2 -axis and finally transform back to frame 0. The overall operation is

$$T_2^0 \cdot T_z(\theta_3) \cdot (T_2^0)^{-1} \cdot \text{Link}_i, \quad i = 3, \dots, n$$

- ✓ In general, the effect of rotation about joint i (frame $i - 1$) or rotation about $z_{i-1} (\theta_i)$. Link i to Link6 will all rotate about z_{i-1} .

$$T_{k-1}^0 \cdot T_z(\theta_k) \cdot (T_{k-1}^0)^{-1} \cdot \text{Link}_k, \quad k = i, \dots, n$$

- ✓ Update the vertices of the patch objects for each link

Step5: Implement simulation of inverse kinematics

- ✓ Generate a suitable end-effector trajectory
- ✓ Implement inverse kinematics for each joint based on what was taught in class.
- ✓ Demonstrate the motion of the robot in Matlab and plot the end-effector trajectory.

Step 6: Implement a control technique

- ✓ Dynamic model of the robot is required in order to implement the controller and simulate the controlled motion. Even if it is a non-model based controlled (e.g., PID

control), one still needs the dynamic model for simulation. In view of the complexity of deriving dynamic equations of motion for a 6 degrees-of-freedom robot, you have an option here.

Option 1: if students would like to take on the challenge, you can use Lagrange formulation to derive the equations of motion. Here is a hint. Generally, CAD software will provide moment of inertia about the center of mass for each part. Also, for symbolic formulation, one can use either Maple or Matlab.

Option 2: Use a simplified robot model. Students can consider the 2nd and 3rd links as a two-link **planar** manipulator with uniform linkages. Derive the equations of motion using Lagrange's formulation for this two-link planar manipulator.

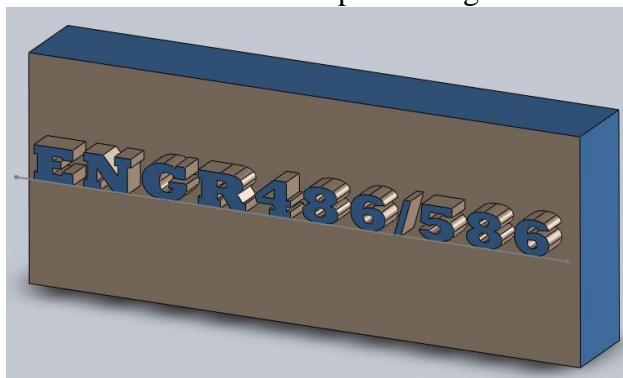
- ✓ Controller implementation and simulation
 - Option 1: independent joint controller such as PID or PD control.
 - Option2: model based control such as computer torque control

Step 7: Writing of report

Final report should be in article format, maximum 30 pages, single columns, 1.5 spaced, 12 point font, including figures and references. Programs referenced should be attached as an appendix.

3. Observations, Lesson Learned, and Future Improvement

To facilitate students learning Matlab and getting familiar with the project, assignments were designed to complement the purpose. Starting from assignment 1, students were asked to use Solidworks or other CAD software to model a simple rectangular box as shown below.



Based on this model, students went over steps (1) and (2) as described in section 2 and learned how to import STL file into Matlab. Rotational operation of this model is done on the “vertices” of this model to realize the animation in Matlab. This simple exercise raises students interest and reduces the uncertainties in their mind of how difficult this project could be. At the same time, learning rotation and transformation becomes much more interesting when students are able to see their animation working properly.

Overall, students have shown great interest in developing this robotic simulator. One of the most frequent phrase I heard when talking to students is “that was fun!” Students have used various models for their course project. Common industrial robots such as Adept s850, KUKA KR30 have been use as the simulation model. One student built a 3-degree-of-freedom robot using

Dynamixel X-12 servo-motor kit and used it for the simulator. Some snap shots of students' works are shown in Figure 5 and 6.

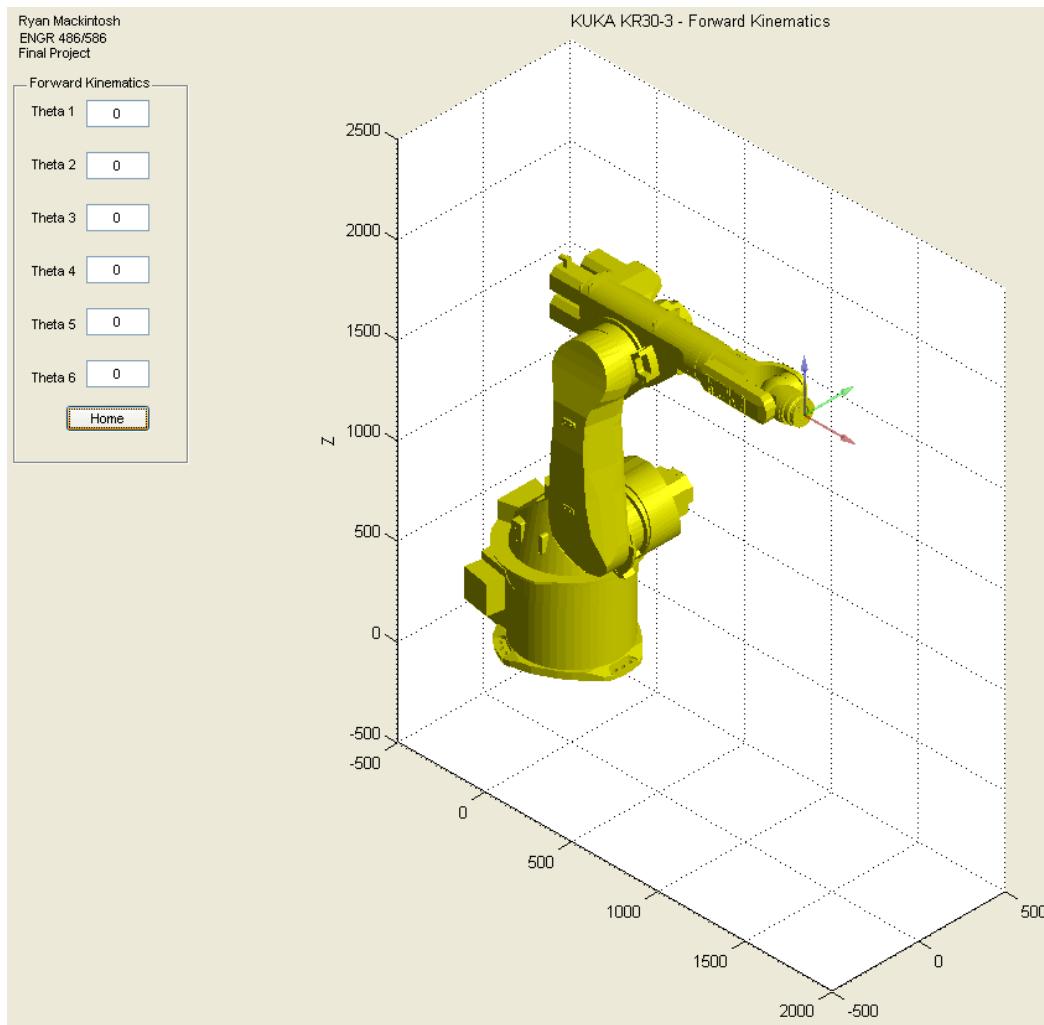


Figure 5 Snapshot of Sample student work simulating KUKA robot.

No formal surveys were conducted regarding students experience of teaching robotics through developing the simulator. However, due to the small class size, close monitoring and frequent communications with the students were maintained throughout the term. Students progresses were kept in pace with this phased project. No one is left behind with help from instructor; no matter it is programming, modeling, or algorithm implementation.

It is observed that students' frustrations mainly arise from implementing different algorithms such as inverse kinematics and controller design. Helping students overcome these difficulties is a key component in insuring the success of this course project. Addition tutorials may be necessary to enhance students Matlab programming skill to improve the efficiency of the simulation.

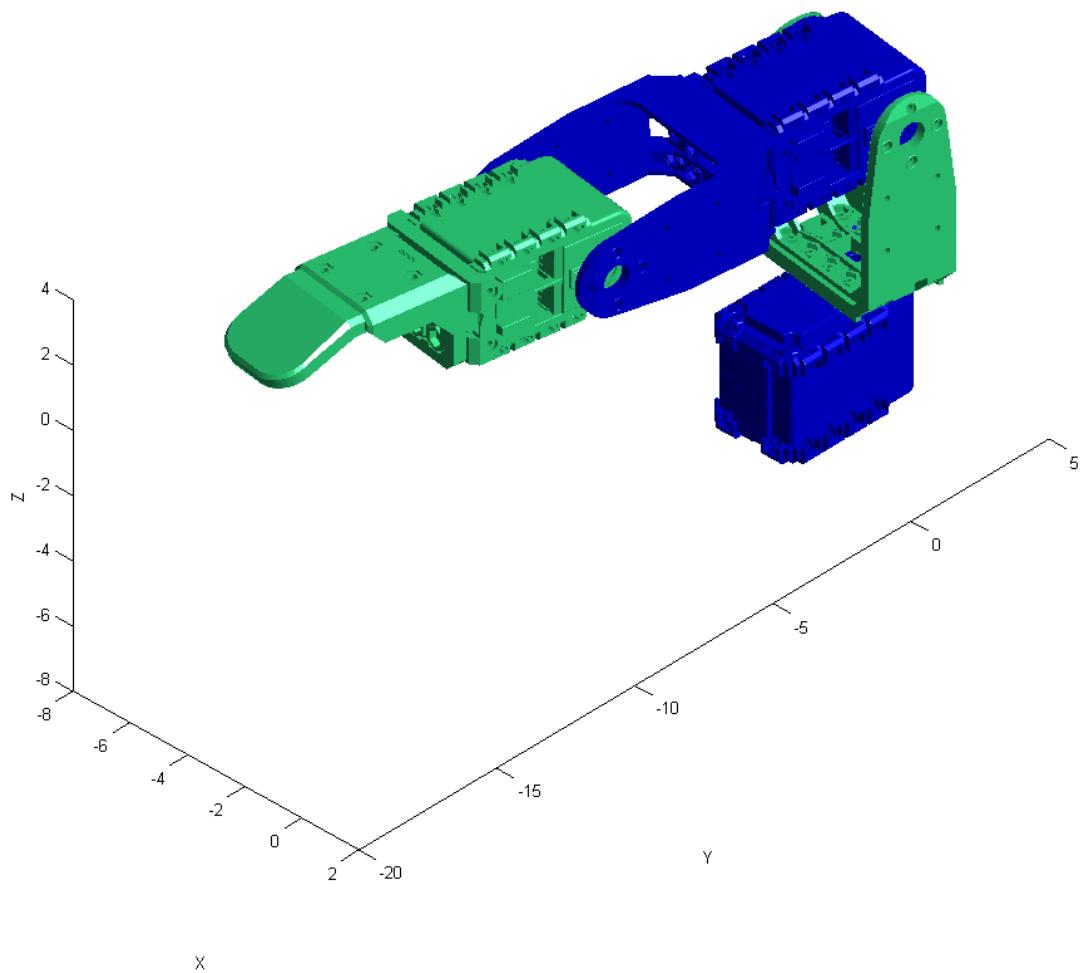


Figure 6 Snapshot of Sample student work simulating self-built robot.

Another challenge associated with the project is dynamic modeling and control of the robot. Key parameters of the robot model, such as moment of inertia of each link and location of center of mass are missing for various reasons. This makes it impossible to derive an exact dynamic model or equation of motion. Consequently, model based control will not be implemented. As an alternative, students have used assumptions and simplification in terms of modeling. For example, they can assume uniform forma and mass distribution for arms. Simulated control motions based on the simplified model.

The course project is evaluated based on the completeness and depth of students work. Coding and functionalities of the simulator are also part of the evaluate criteria. Most students have produced satisfactory work including successfully implemented forward, inverse kinematics, and simple PID control. Projects were properly documented.

Students are able to learn because they think they are having fun from doing this project. For future improvement, the focus is to make this project more flexible to suite different skill levels of students. Some students are more experienced in using Python for programming. The project

should be open to various programming options. Simulation of dynamics can be done using Matlab simMechanics. Results from simMechanics can be integrated with the robot simulator to realize automated motion.

4. Conclusions

Robotics is challenge subject involving intensive linear algebra, modeling and control. Tradition method of teaching tends to cause frustration among students. With the introduction of developing a robotic simulator, the dull math becomes alive. Throughout the practice of this project, students are able to develop this robot simulator using the robot they choose. The process using DH convention to assign frames and program forward inverse kinematics in Matlab proved to be very successful in assisting learning. It helps students understand about robot modeling, direct and inverse kinematics, joint motions, trajectories, and also workspace limitations. What's more, the virtual simulator development gains students interest and motivates student in learning robotics. It allows more lab-type of learning. Some homework can also be readily verified using the virtual robot. For future teaching plan, the develop environment will be open to students' choice. Other engineering tools, such as simMechanics, ADAMS will be considered for dynamics and control design purpose.

References

- [1] T., Hakan; G, Metin; B, Seta, “*Hardware in the Loop Robot Simulators for On-site and Remote Education in Robotics*”, International Journal of Engineering Education, Volume 22, Number 4, August 2006 , pp. 815-828(14).
- [2] Costas S. Tzafestas, Nektaria Palaiologou, “*Virtual and Remote Robotic Laboratory: Comparative Experimental Evaluation*”, IEEE Transactions On Education, Vol. 49, No. 3, August 2006.
- [3] Francisco Candelas, Santiago Puente , Fernando Torres, Francisco Ortiz, Pablo Gil and Jorge Pomares, “*A virtual laboratory for teaching robotics*”, International Journal of Engineering Education, Volume 19, Number 3, pp. 363-370, August 2003.
- [4] Rina Familia, “*A Virtual Laboratory for Cooperative Learning of Robotics and Mechatronics*”, ITHET 6th Annual International Conference, July 7 – 9, 2005, Juan Dolio, Dominican Republic.
- [5] Haslina Arshad, Jaslinda Jamal and Shahnorbanun Sahran, “Teaching Robot Kinematic in a Virtual Environment”, Proceedings of the World Congress on Engineering and Computer Science 2010 Vol.1, WCECS 2010, October 20-22, 2010, San Francisco, USA.
- [6] Reza N. Jazar, “*Theory of Applied Robotics: Kinematics, Dynamics, and Control*”, Springer; 2nd ed. edition (Jun 21 2010).
- [7] Siciliano, B., Sciacicco, L., Villani, L., Oriolo, G., “*Robotics: Modelling, Planning and Control,*” Springer, 2009.
- [8] M.W. Spong, “*Robot Modeling and Control*”, John Wiley & Sons Inc., 2006.
- [9] C.S.G. Lee, “*Robot arm kinematics dynamics and control*”, IEEE Computer, Vol.15, pp.62-80, 1982.

