INTRODUCING KINEMATICS INTO ROBOTIC OPERATING SYSTEMS

Asad Yousuf, Savannah State University; William Lehman, Bill's Robotic Solutions;
Mir Hayder, Savannah State University; Mohamad Mustafa, Savannah State University

Abstract

The study of kinematics is essential to robotics. A robot, to perform most applications, needs to process positional data and transform data from one frame of reference to another. Robots have sensors, links, and actuators, each with its own frame of reference; so transformations between reference frames can be quite tedious. Traditionally, kinematics for robots is introduced to students using MATLAB and the Robotic Toolbox. In this study, the authors examined the introduction of kinematics for robotics with the features and tools available in the open source Robot Operating System (ROS). ROS implements tools for kinematics transforms (tf) as a key part of the ROS core libraries.

ROS defines robots with the unified robot description format (URDF) standard based upon extensible markup language (XML). URDF is, in many respects, similar to the Denavit-Hartenberg (D-H) convention but with significant additional enhancements. Students in electronic engineering technology (EET) were introduced to kinematics and ROS so they would have greater insight into engineering projects involving robotics. It was discovered that using ROS in robotics projects not only makes the projects more interesting to students, but also gives students an authentic experience with distributive systems and odometry sensors. Kinematics for robots uses linear algebra, matrices, natural logarithms (Euler's equation), imaginary numbers and trigonometry. The areas of mathematics were used to introduce kinematics for robotics to EET students to understand electricity, electric fields, and circuit theory. Emphasis was placed on matrix operations, operations involving trigonometry functions and imaginary numbers. The authors summarize here the results of this approach.

Introduction

The study of kinematics is a key tool in both industrial and mobile robotics. Robots have sensors, links, and actuators, each with its own frame of reference; so transformations between reference frames can be quite tedious. Software makes transforms easy to perform and automatic, but students need to understand kinematics to use the software [1]. Labs were designed for EET students to give them basic kinematic concepts, while gaining experience with ROS. In this paper, kinematic theory is presented to give the reader a good idea of what kinematic concepts were presented in the lab. Also in the labs were detailed descriptions of how ROS can be used to learn kinematics.

The topics covered for the kinematic labs are depicted in Figure 1. Euler angles tend to be intuitive to describe robot motion, but have issues when angles approach 90 °. Quaternions are an alternative to Euler angles but are not intuitive to use. Quaternions were treated as black boxes and Euler angles were used for inputs to the robot model to have the best of both descriptions of rotation [2]. Joints used in labs were revolute or prismatic. Other joint types were approximated in ROS using combinations of revolute and prismatic joints. The joint types were limited to keep the introduction to kinematics simple.

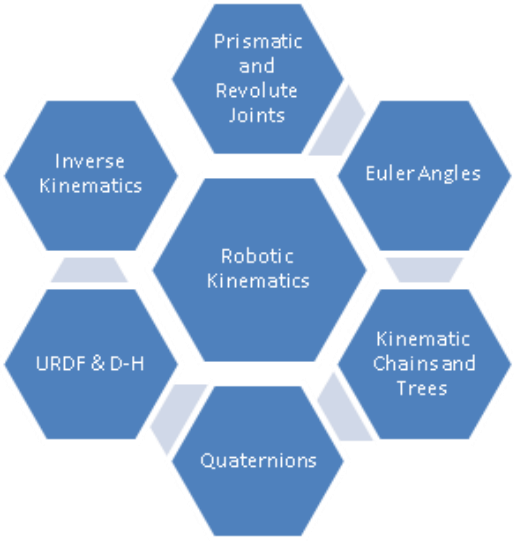


Figure 1. Robotic Kinematic Topics for Labs

Joints in robots are usually combined in a series to form chains. The robot modeling software allowed for chains to be combined into tree-like structures. A good example of a kinematic tree would be a robot with two or more arms. Forward kinematics determines the position of the robot, given the joint rotations or distance for prismatic sliding joints. Inverse kinematics is more difficult than forward kinematics since it is needed to find one or more ways to move a robot to a given point in space. Where there is usually a solution in forward kinematic problems, there may be multiple or no solutions in an inverse kinematic problem.

Robotic Operating System

Kinematics was explored using software packages. ROS has a number of software packages that deal with kinematics (see Figure 2). To demonstrate the kinematic concepts from Figure 1, more than one software package was used from Figure 2. RVIZ is a robot simulator that can display a URDF robot model in 3D along with data from other sensors such as cameras. All the labs take advantage of RVIZ to demonstrate the six kinematic concepts in Figure 1.

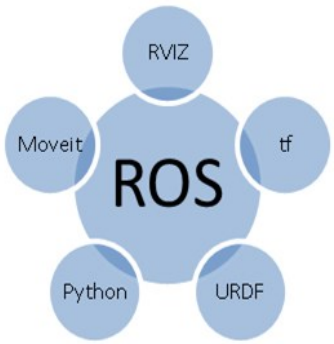


Figure 2. Software Packages in ROS for Labs

The tf software package provides a library of kinematic routines that provide all of the mathematical functions needed to transform kinematic data from one frame of reference to another in robot manipulators. Transforms in ROS are made on positional data in both space and time. There are good tutorials in ROS on how data are transformed in the temporal domain. In this study, examples and labs were based on positional data that were constant so that extrapolation in time could be ignored. Time was important but could be ignored to "keep it simple" for students [3]. URDF is a XML modeling language capable of modeling most robots. Python is a programming language with interfaces to the ROS system. Software packages in ROS are written mainly in Python, Lisp, and C++. Python interpreter was used much like a calculator. RVIZ GUI will display robot information and the results will be confirmed using calculations made in Python. MoveIt is a fascinating software package for controlling robot arms and manipulators. The labs developed to introduce students to robot kinematics are listed in Table 1.

Lab 1 dealt with analyzing views of robots from different frames of reference and converting quaternions to and from axis angle representation. In Lab 2, the students converted quaternions to and from Euler angles using the Python programming language. In Lab 3, students learned to read and write URDF files with the Linux Ubuntu screen editor gedit. The extension to Lab 3 exposed students to converting D-H tables to URDF files. Finally, students confirmed whether the URDF model was correct with the RVIZ robot simulator. Lab 4 offered the opportunity to run a number of inverse algorithms from OMPL using both the Willow Garage PR2 and 6R robot [4]. Of all the labs in Table 1, Labs 1–3 dealt with concepts in forward kinematics and Lab 4 in inverse kinematics.

Table 1. Titles of Labs

Lab	Title
1	RVIZ and PYTHON with simple robot manipulator model
2	RVIZ, TF, and PYTHON with aircraft robot model
2A	Finding position and interpolation with quaternions
3	URDF and hydra robot models
3A	Converting D-H tables to URDF models
4	MoveIt and inverse kinematics
4A	Evaluating student designed robot

Lab 1: RVIZ and PYTHON with a Simple Robot Manipulator Model

In this lab, students learned how to use the RVIZ robot simulator and convert quaternions to/from an axis angle representation. Python programming language was used to make the manual calculations and confirm whether ROS was working as expected (see Figure 3). Python is an interpreted language and supported by the ROS.

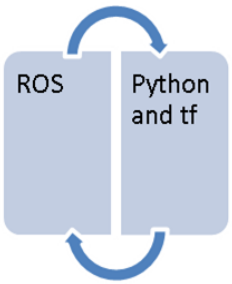


Figure 3. Setup for Lab 1

There were four sections in this lab:

1. Setup and RVIZ features
2. Converting from quaternion to axis angle representation
3. Frames of reference
4. Robot arm movement sequence

Setup guided the students through the startup of RVIZ, Robot State Publisher, and terminals. Features of the RVIZ display were explored. Quaternions to axis angle representation were converted to get useful information from the quaternion. The frames of reference in the chain of links was also changed to take different measurements. Finally, the sequences of moving arm joints were explored to set the robot to different positions in 3D space. Figure 4 represents the RVIZ robot simulator showing the robot arm. The position is shown on the RVIZ screen along with relative position using quaternions for the orientation. The robot state publisher screen is also displayed with sliders to control the robot arm [5]. Figure 5 represents the RVIS robot simulator showing the robot arm visuals turned off to reveal the axis systems of each frame.

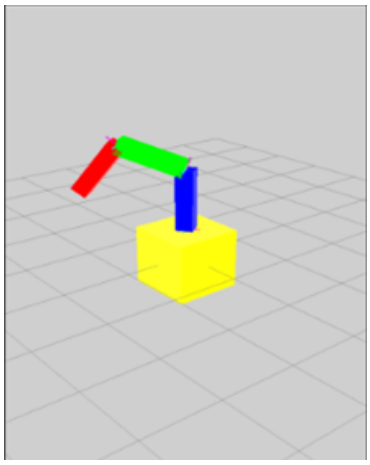


Figure 4. RVIZ Simulator and State Publisher Window

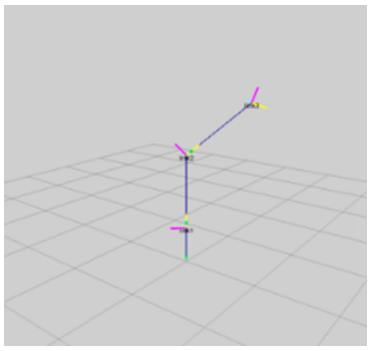


Figure 5. Joint Conventions and 3D Axis

Quaternions

Axis angle is somewhat intuitive and similar to quaternions but in 3D. Axis angle can be converted to and from Euler angles and to and from quaternions. Equations (1)-(11) convert the axis angle vector to a quaternion [6]:

$$q1 = \sin\left(\frac{\alpha}{2}\right)\cos\beta_x \tag{1}$$

$$q2 = \sin\left(\frac{\alpha}{2}\right)\cos\beta_y \tag{2}$$

$$q3 = \sin\left(\frac{\alpha}{2}\right)\cos\beta_z \tag{3}$$

The quaternion was normalized, so Equation (4) was applied.

$$q1^2 + q2^2 + q3^2 + q4^2 = 1 \tag{4}$$

Equations (5)-(8) convert a quaternion to an axis angle vector.

$$\alpha = 2\cos^{-1}(q4) \tag{5}$$

$$x = \frac{q1}{\sqrt{1-q4^2}} \tag{6}$$

$$y = \frac{q2}{\sqrt{1-q4^2}} \tag{7}$$

$$z = \frac{q3}{\sqrt{1-q4^2}} \tag{8}$$

The direction cosine angles can be found using Equations (9)-(11):

$$\beta_x = \cos^{-1} x \tag{9}$$

$$\beta_y = \cos^{-1} y \tag{10}$$

$$\beta_z = \cos^{-1} z \tag{11}$$

Example:
Given the quaternion
$q1 = 0.293802$
$q2 = -0.0957684$
$q3 = -0.294745$
$q4 = 0.904231$

The quaternion is normalized
$1 = q1^2 + q2^2 + q3^2 + q4^2 = 0.999999518$
$a = 0.427043673$
$\alpha = 2\cos^{-1}(q4) = 0.8824416 = 50.56°$
$x = 0.6829905$
$y = -0.2242590$
$z = -0.6901987$

It should be noted that $x^2 + y^2 + z^2 = 1$, and is thus normalized.

$$\beta_x = \cos^{-1} x = 46.5°$$

$$\beta_y = \cos^{-1} y = 77.4°$$

$$\beta_z = \cos^{-1} z = 133.6°$$

Lab 2: RVIZ, TF, and PYTHON with an Aircraft Robot Model

In this lab, students learned how to use the RVIZ robot simulator, Python programming interpreter, and ROS tf library to study Euler angles. The robot state publisher allows for the robot model in RVIZ to be controlled with sliders (see Figure 6). There were three sections in this lab:

1. Euler angles and gimbal lock
2. Converting quaternions to and from Euler angles
3. Frames of reference and the tf transform

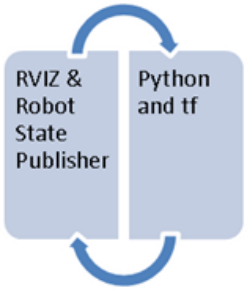
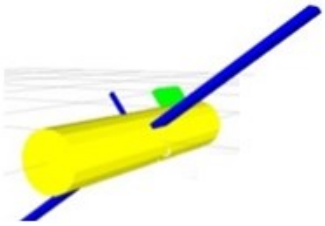


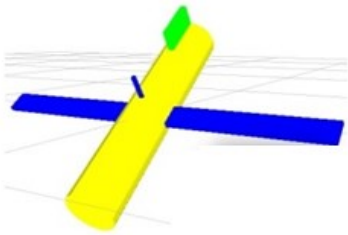
Figure 6. Setup for Lab 2

Euler angles are expressed in terms of roll, pitch, and yaw to specify the orientation of an aircraft. In ROS, roll is a rotation about the x axis, pitch is a rotation about the y axis, and yaw is a rotation about the z axis (see Figure 7). All rotations in ROS follow the right-hand rule for direction of rotation [7].

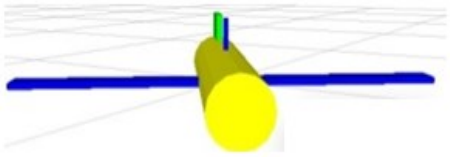
Table 2 shows the combinations of possible Euler angles. It should be noted that Tait-Bray angles are also referred to as Euler angles. The tf library provides two ways to convert Euler angles to quaternions with any of the combinations in Table 2. The tf library can also provide a convenient conversion from quaternions to Euler angles [8]. Euler angles are intuitive to use except that there are issues with Gimbal Lock. Quaternions avoid the issues of Euler angles, but are difficult to visualize. This dilemma can be solved by converting to and from Euler angles [9]. Using multiple frames of reference for a robot makes it easy to calculate angles and distances from one point in the environment to another. The tf library is used in ROS to perform this task.



(a) Roll



(b) Pitch



(c) Yaw

Figure 7. Roll, Pitch, and Yaw of an Aircraft

Table 2. Euler Angle Combinations

Proper Euler Angles	Tait-Bray Angles
RYR	RYP
RPR	RPY
PRP	PRY
PYP	PYR
YPY	YPR
YRY	YRP

Lab 2A: Finding Position and Interpolation with Quaternions

In this extension to Lab 2, position was determined from Equation (12) and interpolation between two quaternions was found with the SLERP algorithm of Equation (12):

$$Q\ P1\ Q^* = P2 \qquad (12)$$

where, Q is the quaternion; Q* is the conjugate of the quaternion, Q; Q and Q* are both normalized quaternions; and, P1 is a vector in the form [x, y, z, 0] and is not normalized. The first half of the calculation is a quaternion multiplied between Q and P1. The results of the multiplication are a quaternion, which in turn is multiplied by the Q* quaternion. Position 2 is also a vector in the form [x, y, z, 0]. Interpolation between two quaternions was found with the SLERP algorithm, which was implemented in Python [10]. Students determined the change in position between successive frames using Equation (12).

Lab 3: URDF and Hydra Robot Models

In this lab, students learned how the Unified Robot Description Format (URDF) describes robots and could use it to design their own robot. The Ubuntu Linux editor, gedit, was used to modify and create URDF text files. RVIZ, a robot simulator, and the robot state publisher in ROS, were used to display and control the robot models [11]. There were five sections in this lab:

1. URDF format and simple robot model
2. Ubuntu Linux graphical screen editor gedit
3. Hydra robot example
4. Hydra robot URDF models
5. Design a robot URDF model

URDF can be used to model a robot with links (members) connected by joints in a chain or tree. Most industrial robots can be modeled by chains of joints offset by links. Multi-arm robots can be modeled with a tree data structure of joints connected by links to a base link. The transmission element will not be covered in this lab at this point, since all the robots needed are created by chains or trees. Other elements of URDF such as sensors are also not used. There are two main types of URDF XML elements that were needed to create robot links and joints. Link elements (or blocks) can contain elements for inertial properties, visual properties, and collision properties. Joint elements can contain elements for origin, parent link name, child link name, axis of rotation/translation, calibration, dynamics, limit, mimic another joint, and safety controller information. The kinematic chain or tree of Figure 8 can be represented by a graph of links connected though joints between each link and other links.

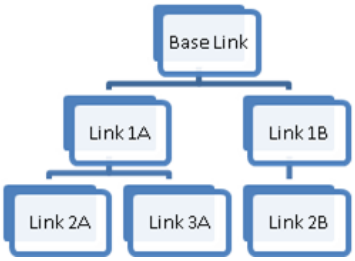


Figure 8. Kinematic Tree

Link 1A is moved by the joint between the base link and Link 1A. Link 1A is connected to a joint between Link 1A and Link 2A that moves Link 2A. Link 1A is also connected to a joint between Link 1A and Link 3A, which moves Link 3A. Each Link is moved by a single joint but may be connected to a number of joints that move other links. In URDF terminology, multiple joints can be connected to one link, but the link can only be a child in one of the joints connected to it and must be a parent to all the other joints connected to it. The base link is the first link in the tree and is special. The origin of the axis system to world coordinates x, y, and z was determined from the odometry frame (/odom) and map. In the next example, the robot arm axis origin was located at world coordinate 0, 0, 0. It should be noted the students received a graph of the kinematic tree or chain with the urdf_graphviz command entered into the terminal [12].

Link elements must have the "name" attributes for the link. The inertial and collision properties for the link are not included, except where it will be connected to a prismatic type joint. The visual information can be provided by specifying a rectangle, sphere, or cylinder shape for the link or a mesh. Although the mesh produced by a computer aided design (CAD) program can be very pleasing to the eye, it was kept simple with a rectangle or cylinder shape. As a convention in the design of the robots for the labs, all of the link and joint elements were grouped together for readability of the URDF file [13], see Figure 9. The joint element had a name and an attribute for the type of joint. Only a revolute, continuous, or prismatic type was selected for the joint. The ROS continuous joint type in actuality is a revolute. In URDF, a continuous joint is a revolute joint with the angle of rotation of 360°, where the revolute joint limits of rotation must be specified. The joint attaches a parent link to a child link. The child link can be a parent link to one or more other links in a chain or tree. The child link, however, can never be connected through a series of joints and links back to its parent link making a loop [14].

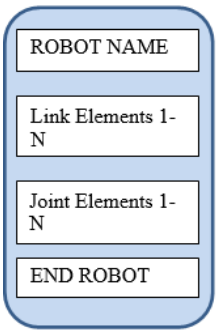


Figure 9. Types of URDF XML Blocks

An example link element can be found in Figure 10. The link could have been named anything to identify it, but it was named "link2" as it was the second link in the chain. The inertial and collision information was skipped, not because it was not useful but rather to keep the example simple. The visual element contained the geometry element, which set the type of display using the geometry element. The box element was inside the geometry element. The box element set the x, y, z sizes of the rectangle. The visual also contained the material element, which set the color of the rectangle link. This link was set to an arbitrary color to distinguish it from other links in the RVIS simulator display.

```
<link name="link2">
  <visual>
    <geometry>
      <box size="1 0.25 0.25"/>
    </geometry>
    <origin rpy="0 1.571 0" xyz="0 0 0.5"/>
    <material name="linkc2">
      <color rgba="1 0 1 1"/>
    </material>
  </visual>
</link>
```

Figure 10. URDF XML Element Link Block

The visual also has the origin element inside it. The origin rotates the linkage visual display using roll, pitch, and yaw angles. Roll is about the x axis, pitch is rotation about the y axis, and yaw is rotation about the z axis. In the above example, the rectangle was rotated 90° from the long side on the x axis to now point along the z axis. The link was offset 0.5 meters above the origin of the base link. The robot was actually buried in the floor in a hole with 0.5 meters deep. Syntax of the XML elements had the form "<label parameters>" followed by other elements and a "/>" or "</label>" [15].

Next, a joint XML element was examined (see Figure 11). The joint must have some name, which should make sense to identify where on the robot it resides. In this example, it was labeled as "link1_link2", since the link named link1 was connected to the link named link2. The joint was a revolute type with no limits on rotation angle, so a continuous type was chosen. The parent link was link1 and the child link was link2.

```
  <joint name="link1_link2"
type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
    <axis xyz="1 0 0" />
    <origin rpy="0 0 0" xyz="0 0
1.0"/>
  </joint>
```

Figure 11. URDF XML Joint Element Example

The axis element specified the axis that the joint would revolve around, which in this case was x. The origin xyz attribute was an offset of the joint from the parent joint origin to the child's joint origin. The origin rpy parameter was a roll, pitch, and yaw rotation on the child's joint frame of reference. The robot URDF examples with prismatic joints were slightly more complex since prismatic joints require limit and safety parameters not required for continuous joint types. The following command in the terminal window can be used in ROS to check the syntax of the URDF file.

```
rosrun urdfdom check_urdf filename.urdf
```

Lab 3A: Converting a D-H Table to a URDF Model

Denavit-Hartenberg (D-H) conventions ease the process of calculating the position and orientation of frames in a kinematic chain [16]. As noted earlier, URDF can be used to model robots that are kinematic trees. URDF does not require axis systems for frames to only rotate about the z axis. URDF actually uses an arbitrary axis for revolute or prismatic joints. However, the parameters of the URDF joint variables can be shown in a table analogous to D-H parameters. To demonstrate, the axis system was used in Figure 12 in order to make the D-H table for the 3R robot similar to the 3R robot introduced earlier. The θ variable represents rotation about the z axis of the joint. The d parameter is the distance along the z axis to the next joint. The d parameter is also a variable in the case of prismatic joints. The "a" parameter is the distance of each common normal or the offset between joints. Finally, the α parameter is the angle between the current joint, i, and the next joint i+1.

The parameters in the D-H table can be translated to parameters in the URDF joint element [17]. Figure 12 has the y axis going into the page for Joint 1 and the z axis coming out of the page for Joints 2 and 3. The parameters of the URDF joint elements can be enumerated in a table similar in concept to the D-H table (see Table 3). For a simple kinematic chain, there would be a single corresponding table to model the joints in the robot. For a kinematic tree, multiple joint element parameter tables can be used to represent each kinematic chain in the tree. The d_i parameters will always be placed in the corresponding Joint + 1 z axis offset between Joint i and Joint i+1. The α_i parameter is the offset between Joint i and Joint i+1 along the x axis of Joint I and joint i+1.

The robot has to be placed in world coordinates in ROS for a convenient orientation and to simplify calculations. The 3R robot is placed at the origin of the world axis system and aligned with that axis system. This means that instead of rotating Joint 2 by 90°, Joint 2 will be rotated so that the z axis comes out of the page. Joints 3 and 4 will also be rotated since Joint 2 is attached to Joint 3, which is attached to Joint 4. Joint 4 is a fixed type joint and does not move. Joint 4 is included to show the a3 parameter in the D-H table. Dummy URDF link elements are set up with simple names such as Link_1, Link_2, etc. No information is needed in the URDF link element since visual elements of the robot model are not being displayed in the lab. A base link (not included in D-H table) is needed to attach the robot model to the world coordinates. The parameters from the table to the URDF joint element are also straightforward (see Table 4). The x, y, z parameters correspond to the origin xyz parameter in the URDF joint element. The roll, pitch, and yaw parameters corresponds to the origin rpy parameter in the URDF joint element. Finally, the axis parameters define a unit vector pointing along the z axis. Since this is a revolute joint, the joint will rotate around the z axis.

After students create the URDF joint element parameter, given the D-H table they perform, they convert it to URDF and display the results in RVIZ to confirm that it matches the axis system orientation of Figure 12. The a3 parameter is between Joints 3 and 4. In Figure 13, the red axis is x, the green axis is y, and the blue axis is z.

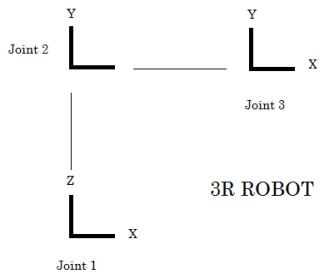


Figure 12. Axis 3R Robot

Table 3. D-H Parameters for 3R Robot

Link	θi	d_i	$\acute{a}_i$	$\acute{a}_i$
1	θ_1	d_1	0	90 °
2	θ_2	0	$\acute{a}_2$	0
3	θ_3	0	$\acute{a}_3$	0

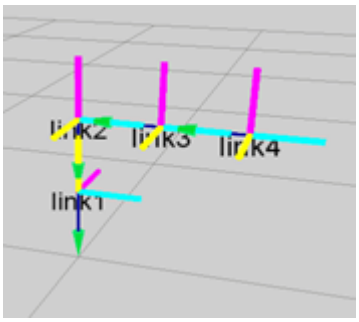


Figure 13. D-H 3R Robot Displayed in RVIZ

Lab 4: MoveIt and Inverse Kinematics

There were two sections in this lab:

1. Setup MoveIt
2. Students record benchmark information for each configuration including planning time [18-20]
3. Smoothness of trajectory is noted

Table 4. URDF Joint Elements Parameters

#	Joint Type	Joint Link #	Joint+1 Link #	x	y	z	Joint+1 Roll	Joint+1 Pitch	Joint+1 Yaw	Joint+1 Axis x	Joint+1 Axis y	Joint+1 Axis z
1	R	Base	1	0	0	d_1	0	0	0	0	0	1
2	R	1	2	α_2	0	0	90^0	0	0	0	0	1
3	R	2	3	α_3	0	0	0	0	0	0	0	1
4	Fixed	3	4	0	0	0	0	0	0	-	-	-

In this lab, the advantage of existing demonstration software that uses the PR2 and 6R robot was taken. Students set the robots to different start and end states for the PR2 and 6R robots. Students then watched the robot perform each motion using the planning algorithm from the OMPL library and recorded the time it took to calculate a solution. There is an excellent tutorial that walks the students through the setup and use of the demo software [21]. In the lab, students selected from among the following planners to perform the path selected for the PR2 or 6R robot [22]:

1. Bi-directional kinematics planning by interior-exterior cell exploration (BKPIECE)
2. Kinematics planning by interior-exterior cell exploration (KPIECE)
3. Lazy bi-directional kinematics planning by interior-exterior cell exploration (LBKPIECE)
4. Expansion space trees (EST)
5. Probabilistic roadmap method (PRM)
6. Transition-based rapidly-exploring random trees

Building upon the RVIZ plugin tutorial, the students were presented with a problem requiring building a table for the selected algorithm using different planning groups and other planning parameters. The students then recorded the results of the test in a table of planning parameters versus time and evaluated the results. Objects were inserted into the scene to demonstrate and test planning with obstacles [23]. The goal of the lab was to give them some insight into the complexity of inverse kinematics (see Figure 14).

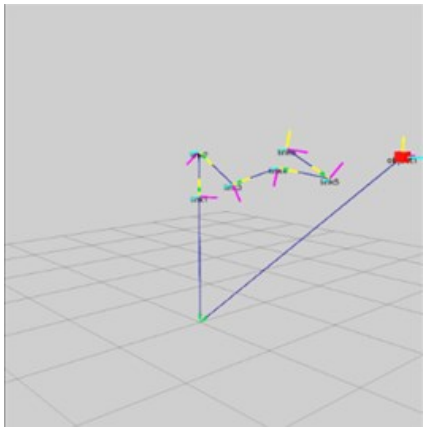


Figure 14. Demo Software Screen with 6R Robot

Lab 4A: Evaluating the Student-designed Robot

The students evaluated the robot that they designed in Lab 3. They used MoveIt setup assistant to configure their robot. The students set up pre-configured poses for their robot and followed a procedure similar to Lab 4 to evaluate the robot [24].

Conclusion

In this paper, the authors presented the major features of the seven labs developed to introduce students to kinematics using ROS. Labs were developed for students to use accelerometers and gyros to track real robots using ROS. The introduction to robotic kinematics should provide the background to understanding kinematic aspects of these labs. The robot toolbox provides a powerful system for introducing students to robotic kinematics [25]. Given the limited classroom time available for kinematics, the study focused on forward kinematics and ROS. Given more time in the robotics course, the authors would have included both MATLAB and ROS. MATLAB has recently developed a robot toolbox that allows MATLAB to connect to ROS. Although MATLAB can connect to a ROS system, there is a definite value to introducing kinematic concepts in ROS to lower the ROS learning curve and re-enforcing kinematic concepts. The labs were tested with the Hydro and Indigo versions of ROS, but future plans are to include Jade. Information on how to obtain the free open source labs outlined in this paper can be found at the website for Brazen Head Automation [26].

References

[1] Tully, F. (2013). Technologies for Practical Robot Applications (TePRA). *Proceedings of 2013 IEEE International Conference on Open-Source Software workshop. ISSN 2325-0526, pages 1-6.* doi 10.1109/TePRA.2013.6556373.

[2] James, D. (2006). Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. *Matrix, Citeseer*.

[3] Robot Operating System (n.d.). Retrieved October 25, 2015, from www.wiki.ros.org/tf

[4] Open Motion Planning Library: A Primer. http://ompl.kavrakilab.org

[5] Bruno, S., & Oussama, K. (2008). Handbook of Robotics, *Digital Design*. Springer.

[6] Maths-Rotation Conversions (n.d.). Retrieved on October 25, 2015, from www.euclideanspace.com / maths/geometry/rotations/conversions/

[7] Coordinate Frames for Mobile Platforms (n.d.). Retrieved on October 25, 2015, from http://www.ros.org/reps/rep-0105.html

[8] Standard Units Measure and Coordinate Conventions (n.d.). Retrieved on October 25, 2015, from http://www.ros.org/reps/rep-0103.html

[9] Robot Operating System (n.d.). Retrieved October 25, 2015, from wiki.ros.org/geometry/Coordinate FrameConventions
[10] van Oosten, J. (2012). Understanding Quaternions. *3D Game Engine Programming*.
[11] Robot Operating System (n.d.). Retrieved October 25, 2015, from http://wiki.ros.org/urdf/Tutorials
[12] Robot Operating System (n.d.). Retrieved October 25, 2015, from wiki.ros.org/urdfoveit.ros.org/urdf
[13] Robot Operating System (n.d.). Retrieved October 25, 2015, from wiki.ros.org/urdf/XML/link
[14] Robot Operating System (n.d.). Retrieved October 25, 2015, from wiki.ros.org/urdf/XML/joint
[15] Robot Operating System (n.d.). Retrieved October 25, 2015, from en.wikipedia.org/wiki/XML
[16] Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2004). Robot Dynamics and Control. Second Edition, *Chapter 3 Forward Kinematics: The Denavit-Hartenberg Convention.* Wiley.
[17] Thomas, F. (2012). Solved Problems in Robot Kinematics Using the Robotic Toolbox. *Universitat Politecnica DE Catalunya Barcelona Tech.* Barcelona, Spain.
[18] Corke, P. (2013). Robotics, Vision and Control. Edition 1, *Section 8.1.4 Jacobian and Manipulability*. Springer.
[19] Tsai, T. (1986). Workspace Geometric Characterization and Manipulability of Industrial Robots. *PhD thesis, the Graduate School of Ohio State University*.
[20] Vahrenkamp, N., Asfour, T., Metta, G., Sandini, G., & Dillmann, R. (2012). Manipulability Analysis. *Proceedings of IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. Osaka, Japan.
[21] Robot Operating System (n.d.). Retrieved Oct 25, 2015, from http://moveit.ros.org/documentation / tutorials/
[22] Sucan, I. A., Moll, M. & Kavraki, L. E. (2012). The Open Motion Planning Library. *IEEE Robotics and Automation Magazine*.
[23] Prats, M, Sucan, I., & Chitta, S. (n.d.). Workspace Analysis. (n.d.). Retrieved October 25, 2015, from http://moveit.ros.org/assets/pdfs/2013/ icra2013tutorial/ICRATutorial-MoveIt.pdf
[24] Chitta, S., & Sucan, I. (2013). MoveIt. *ROS Developer Conference*. Stuttgart, Germany.
[25] Corke, P. (2014). Robotic Toolbox for Matlab. *Release 9*. Retrieved October 25, 2015, from http:// www.petercorke.com/robot
[26] Brazen Head Automation (n.d.). Retrieved October 25, 2015, from http://www.brazenbot.com

Biographies

ASAD YOUSUF is a professor at Savannah State University. He earned his B.S. degree from N.E.D University, M.S. degree from the University of Cincinnati, and doctoral degree from the University of Georgia. Dr. Yousuf is a registered professional engineer in the state of Georgia. He is also a Microsoft Certified Systems Engineer. Dr. Yousuf may be reached at yousufa@savannahstate.edu

WILLIAM LEHMAN is president of Bill's Robotic Solutions, which he started in 2013. He has over 20 years of experience in software and hardware development. He has worked on numerous projects in digital communication systems, robotics, and aerospace applications. Mr. Lehman received his B.S. degree in Electrical Engineering from the Catholic University of America. Mr. Lehman may be reached at tec.teacher.lehman@gmail.com

MIR HAYDER is an assistant professor in the Department of Engineering Technology at Savannah State University. He received his Ph.D. in mechanical engineering from McGill University. His research interests include robotics, fluid-structure interaction, syngas and blended fuel combustion, and flow and structural simulations. Dr. Hayder may be reached at hayderm@savannahstate.edu

MOHAMAD MUSTAFA is a professor of civil engineering technology at Savannah State University. He had six years' of industrial experience prior to teaching at SSU. He received his B.S., M.S., and Ph.D. degrees in civil engineering from Wayne State University. His research interests include sensors applications in civil engineering. Dr. Mustafa may be reached at mustafam@savannahstate.edu