An Educational Tool to Support Introductory Robotics Courses

Dr. Fernando Garcia Gonzalez, Florida Golf Coast University

Dr. Fernando Gonzalez joined FGCU as an Assistant Professor in the Software Engineering Program in the fall of 2013. Previously he has worked at Texas A&M International University in Laredo, Texas, the U.S. Department of Energy at Los Alamos National Laboratory in Los Alamos, New Mexico and at the University of Central Florida in Orlando, Florida. Dr. Gonzalez graduated from the University of Illinois in 1997 with a Ph.D. in Electrical Engineering. He received his Master's degree in Electrical Engineering and his Bachelor's degree in Computer Science from Florida International University in 1992 and 1989. Dr. Gonzalez research interest includes the intelligent control of large scale autonomous systems, autonomous vehicles, discrete-event modeling and simulation and human signature verification.

Dr. Janusz Zalewski, Florida Gulf Coast University

Janusz Zalewski, Ph.D., is a professor of computer science and software engineering at Florida Gulf Coast University. Prior to an academic appointment, he worked for various nuclear research institutions, including the Data Acquisition Group of Superconducting Super Collider and Computer Safety and Reliability Center at Lawrence Livermore National Laboratory. He also worked on projects and consulted for a number of private companies, including Lockheed Martin, Harris, and Boeing. Zalewski served as a chairman of the International Federation for Information Processing Working Group 5.4 on Industrial Software Quality, and of an International Federation of Automatic Control Technical Committee on Safety of Computer Control Systems. His major research interests include safety related, real-time embedded and cyberphysical computer systems, and computing education.

Mr. Gerardo Javier Pinzon P.E., Texas A&M International University

An Educational Tool to Support Introductory Robotics Courses

With the rising popularity of robotics in our modern world there is an increase in the number of engineering programs that do not have the resources to purchase expensive dedicated robots but find a need to offer a basic course in robotics. This common introductory robotics course generally covers the fundamental theory of robotics including robot kinematics, dynamics, differential movements, trajectory planning and basic computer vision algorithms commonly used in the field of robotics. The nature of this material almost necessitates the use of robotic hardware to allow the students to practice implementing the theory they learn in class.

This paper introduces a software based educational tool designed to be used in introductory robotics courses. The software simulates the geometry of motion (kinematics) of any multilink industrial robotic arm and is to be used in place of or along with an actual robotic arm. The students can use this tool to support their learning much the same way they use an actual robotic arm. The tool includes an integrated development environment that models the environments that typically included with robotics packages. This tool allows the student to input the characteristics of the arm they wish to program allowing the student to program any type of arm they wish. This tool provides a low cost solution to situations where purchasing expensive robotic arms typically needed for this course is not possible, where the existing equipment does not allow for direct joint programming, for on-line robotics courses and for non-software intensive programs where the students are not to produce robotic software.

Keywords—robotics, software tool, kinematics, programming

I. INTRODUCTION

In this paper we present a new software tool that is specifically designed to teach the basic Introduction to Robotics course. Many robotics books[1-8] cover this material. The course generally covers robotics fundamentals including history, robot types, and degrees of freedom, robot kinematics including the transformation matrix, forward and inverse kinematics, and the Denavit-Hartenberg (D-H) parameters, differential motions, robot dynamics, trajectory planning, actuators and sensors, and robot vision.

This course is generally a first course in robotics and may be an undergraduate or graduate level course. In many programs, this course is offered as the first in a sequence of several robotics courses in a robotics program with the goal of graduating students who specialize in robotics. This course has gained a reputation of being very interesting and attractive to students which has led many institutions to offer this course as an independent single course with an alternate goal not related to producing robotics experts. These goals range from teaching students how different systems work together such as in the Systems Engineering Program at Texas A&M International University to simply providing an interesting elective course such as in the Software Engineering Program at Florida Gulf Coast University.

This tool supports teaching the topics which include robot degrees of freedom, forward and some limited inverse kinematics, Denavit-Hartenberg parameters, some path planning, and simple robot programming. The tool has many features that lend itself to teaching this course. Some of these features are generally not found in general purpose robotic simulators. The

learning outcomes this tool is designed to support are listed below. This tool is designed to support the student with learning:

- The relationship between the standard Devanit-Hartenberg representation and the corresponding arm.
- The forward kinematics equations.
- How to use the inverse kinematic equations to program the arm.
- How to program the arm at a high level by defining points and using the move instruction.
- How to design a trajectory.

The tool presented in this paper was created by the authors for the sole purpose of supporting the instructor in the Introduction to Robotics course. The tool is written in C++ using wxWidgets for its graphical user interface (GUI) and OpenGL for its graphics. The tool also has a component to support learning robotic vision but this component is not presented here. The two components are not integrated at this time.

II. Relation to Other Work

There exist a large number of general purpose robotic simulators both free and commercially available[10-13]. These tools are used for professional robotics research and related work as well as for educational purposes. The problem with using these general tools for teaching an introductory robotics course is first, there is a relatively large learning curve needed to get sufficiently familiar with the tool before the student can use them for learning robotics. Our tool is specifically designed to allow a student that has never used the tool before to input the specifications of the robotic arm and get to the point where the student can move the links of the robot and adjust the viewing position in less than 5 minutes. Secondly actual robots are programmed using an included environment that uses a custom scripting language that performs all the inverse kinematics required for the robot. While this is how real robots are programmed, it does not lend itself to learning introductory robotics since the logic in these preexisting software components is precisely what the student needs to learn how to create. This tool differs from these robotic simulation tools in that the presented tool is specifically designed to teach this specific course and therefore has a much smaller learning curve and does not do the work the students need to do to learn.

Peter Corke[8] has developed a library of MATLAB functions and has made it available free[9]. This library is very popular but requires the student to write programs in MATLAB. While writing programs is far superior in helping the student learn robotics, it is not always feasible especially at institutions that do not have a software intensive program such as a robotics program. For example in the Systems Engineering Program at Texas A&M International University programming is a very small part of the engineering curriculum where students are not expected to be able to create whole programs yet they still offer the robotics course as required for the major. It is not reasonable to add learning how to program to the robotics course contents. This course in the Systems Engineering Program was the initial motivation for creating our tool.

III. SPECIFYING THE ARM

One of the best features of this tool is the user's ability to specify an arm by simply entering 6 numbers per link. There is no need to draw or design the arm and any arm the user can dream up can be used with the tool. These numbers include the four (D-H) parameters along with the lower and upper limits of the link's motion. Using the forward kinematic equations formed by the D-H parameters, an arm with the correct kinematics can be totally specified. This arm may not look realistic but its kinematics are correct. That is, if the joints are moved to a specific set of angles, the location and orientation of the hand will be exactly the same as any robot, real or simulated, with that same set of D-H parameters. Excluding the limit of movement of each joint, the relationship between the joint angles and the position and orientation of the end-effector is dictated only by the D-H parameters and any two arms with the same parameters will have the same relationship. That is why those and the joint limits are the only parameters the tool needs to specify the arm. This results in the user being able to simulate any arm in the text book and furthermore being able to enter the arm in just a few minutes. In Figure 1 the specifications for the three DOF arm with two revolute joints and one prismatic joint (R2P) is shown entered into the arm creation pane. The rendered arm in shown in Figure 2a.

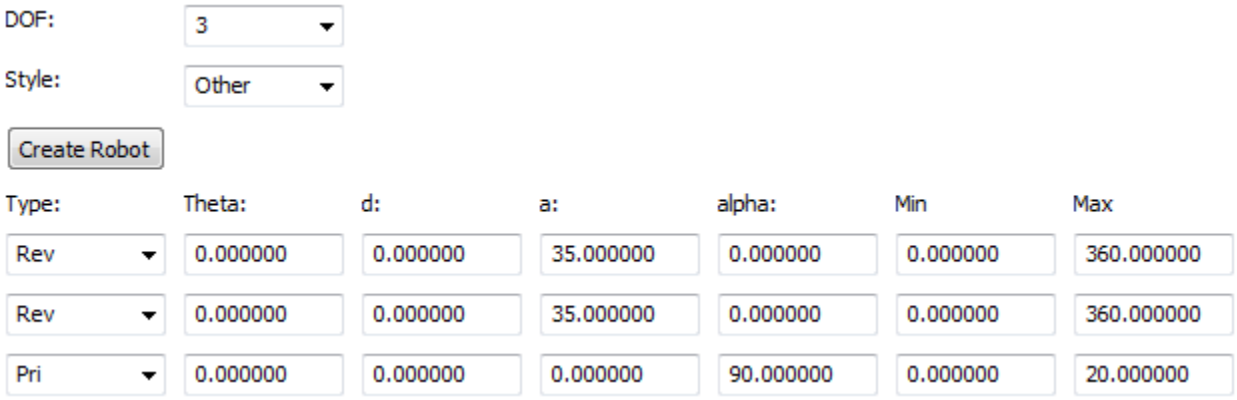
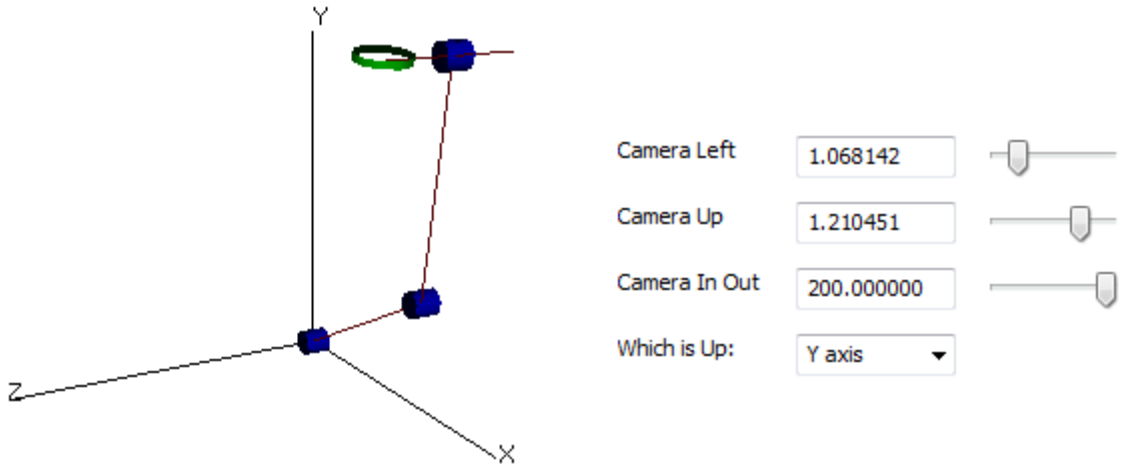


Figure 1: The control panel to create the robotic arm.

IV. RENDERING THE ARM IN THREE DIMENSIONS

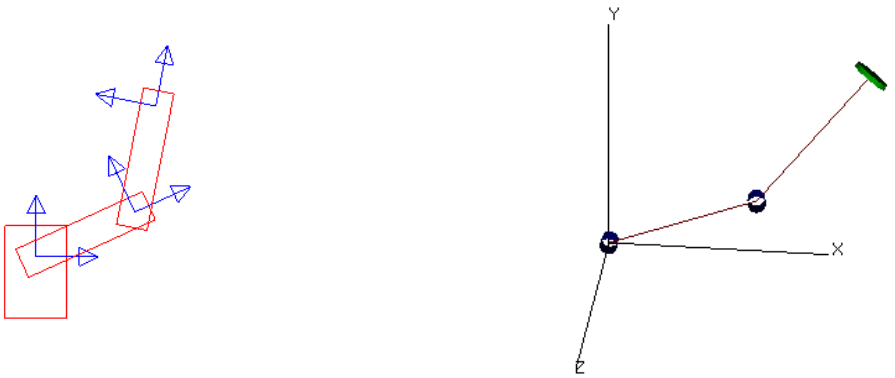
The arm is rendered using three dimensional (3D) graphics, see Figure 2a. The user can zoom in and out, select the coordinate axis that points up and move the eye position vertically and horizontally using the controls shown in Figure 2b.



a b

Figure 2: (a) The 3D rendering of a 3 link R2P arm. (b) The 3D view controls.

There is a two dimensional (2D) view as well. While any arm can be rendered in 2D, the motivation is to display the popular two link rotational arm shown in Figure 3a that appears in many books because of its simplicity. Figure 3b shows the same arm rendered in 3D.



a b

Figure 3: (a) A two link arm rendered in 2D. (b) The same arm rendered in 3D.

V. MOVING THE JOINTS USING SLIDERS

Each joint is given a slider that the user can use to move that joint. The arm's rendering changes as the slider moves in real time. In Figure 4a, the sliders have not moved and the arm is in its home position. In Figure 4b the sliders have moved and the arm is rendered in the position corresponding to the joint angle specified in the sliders. The angles are in degrees.

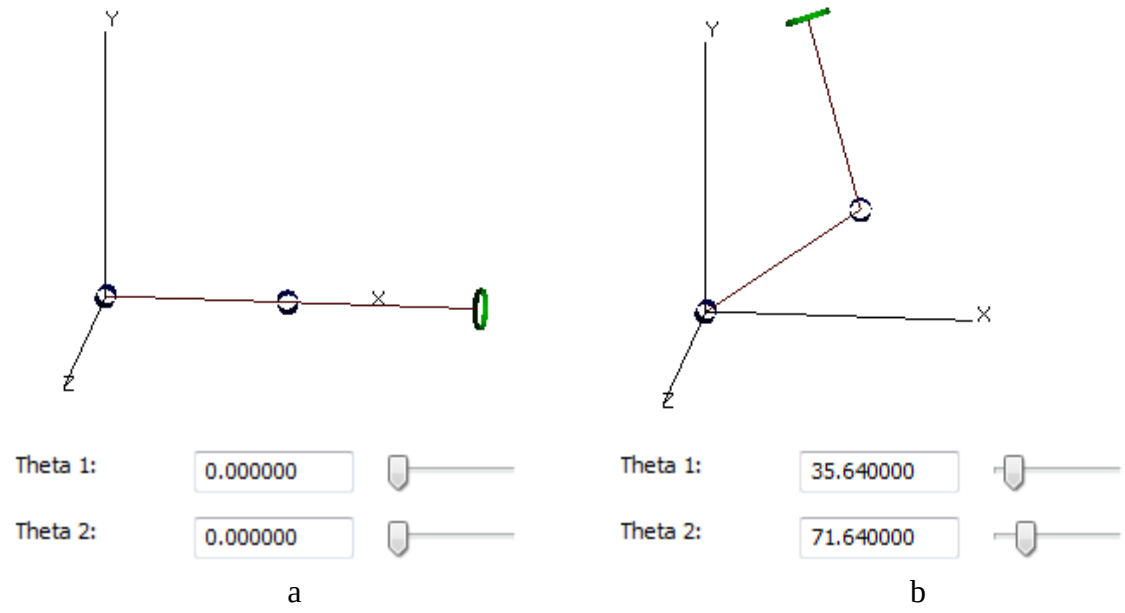


Figure 4: (a) The slider have not moved and the arm is rendered in its home position. (b) The two sliders have moved and the corresponding arm rendering is shown.

VI. Running the Tool in Simulation Mode

In simulation mode each joint moves in proportion to the power applied to the simulated joints motor. This models the movement of a real arm. The joint motors must be given a desired velocity for them to move. An object manages the movement of each joint and only moves the joint if it is given a velocity versus time function. The desired velocity must be reached by accelerating the joint to the desired velocity. Each joint is given a maximum acceleration which determines the time it takes to reach the desired velocity. The joint also needs to decelerate to reduce its velocity. The joint's position is therefore a function of the joints power setting over time. The user can chose to stop simulation mode and move the joints by hand using the sliders. However the joint's position in simulation mode remain unchanged and the arm will return to its last moved position once the user returns back to simulation mode. Figure 5 shows buttons that are used to move the joints in simulation mode. Pressing a button moves the joint by the degree specified in the box to the right. Pressing a button creates a 2-1-2 type trajectory to go from its current position to one that is the number of degrees specified away. This type of manual movement is available in most robot controllers.

Figure 5: The buttons move each joint by the degrees specified in the box to the right.

VII. Using the Tool to Teach Robot Programming

Most robotics systems contain a compiler or interpreter along with its corresponding programming language that can be used to program the robot. The language may be proprietary to that manufacturer or may be one of several standard robotic languages. These languages tend to

be very simple. They generally provide an instruction to move the robot given a predefined point. These machines are programed by first defining a list of points then the programmer can use the move instruction to move the robot to the predefined points. The points are defined by physically moving the robot to the position and recording the point.

In our tool we are modeling the same process. The user can write a simple program in our robot language to move the arm. The program runs in simulation mode. The language includes a set of instructions to perform task such as to move to a predefined position, open and close the gripper, jump to an instruction and some variable manipulation. The process of writing the program starts with defining a set of positions. A position is a point in the 6 dimensional joint space where a point is represented by a 6-tuple with each number being an angle for the corresponding joint. The user then moves the arm manually using move buttons shown in Figure 5 to the desired position and records the position. The user can override the default name by providing a unique name as shown in Figure 6. In Figure 7 the user can see and modify the list of points. After this list of points is created the user can write the program. The program generally moves between points as specified by the point parameter. The user must define points so that the movements between points will yield a path that does not hit any obstacles.

Figure 6: To record a point the user clicks the add button. The user can override the default point name.

Name Point	J1	J2	J3	J4	J5	J6
P0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
P1	27.000000	40.680000	0.000000	0.000000	0.000000	0.000000
P2	40.680000	74.880000	0.000000	0.000000	0.000000	0.000000

Figure 7: The user can see and change the list of points.

The list of instructions currently implemented is shown in Table 1. This part of the tool is not fully developed and more instruction may be added. All of the instructions have a line or instruction number assigned to it. Jumps are made to the instruction numbers. While this is not a structured programming language it is simple and is how real robots are generally programmed. The program is created by selecting the instruction from the pull down menu of the last empty instruction. Once the instruction is selected the rest of the parameters' widgets are displayed and the user can then select the appropriate parameters. All the variables and point are available in pull down menus. Adding and deleting instructions are done by clicking the instruction number button to select the instruction then clicking the delete or insert button. In Figure 8 a program listing is shown. Note the widgets available to the programmer. The following is a detailed explanation of each instruction.

- The MOVE instruction causes the arm to move from the current position to the destination position. A destination point and velocity must be specified. A 2-1-2 blend type path plan is created for each joint. The velocity functions produced by the path plan

are given to the simulator to power the simulated joint motors. The MOVE instruction takes two parameters, the destination point and the travel velocity. All the joints travel at the same velocity. As a result the movement are not linear.

- The OPENGRIPPER and CLOSEGRIPPER instruction open and close the gripper. No parameters are needed. The gripper is not currently implemented.

- The LET instruction is used to assign a value to a variable. Variables do not need to be declared and all can accommodate a floating point value. The parameters are the variable name which can be new or an existing name and its value.

- The INCR and DECR instructions add or subtract 1 from the variable. At this time there is no support for implementing mathematical expressions and therefore these instructions are the only way to modify variable values such as to implement counters. The only parameter is the variable.

- The IF instruction takes a condition and if true jumps to the destination instruction. There is no else option implemented at this time. The condition consists of two variables and a Boolean operator. No support for complex mathematical Boolean expressions are implemented at this time. The parameters are the two variable names, the Boolean operator and the instruction number to jump to if the condition is true.

- The JUMP instruction is an unconditional jump. The only parameter is the instruction number to jump to.

Table 1: Current Instruction Set

Instruction	Description	Parameters
MOVE	Moves the arm from the current position to the specified position	The destination point name, the velocity.
OPENGRIPPER	Opens the gripper	None.
CLOSEGRIPPER	Closes the gripper	None.
LET	Assigned a value to a variable.	The variable name, the value.
INCR	Increments the variable by 1.	The variable name.
DECR	Decrements the variable by 1.	The variable name.
IF	Conditional execution. The condition consists of a left and right variable and a Boolean operator. If the condition is true a jump to the instruction number is performed.	Condition (left variable, Boolean operator, right variable, instruction number.
JUMP	Unconditional jump.	The instruction number to jump to.

In Figure 8 the listing of a program that moves the arm between **P1** and **P2** and back 2 times is shown. This program is described in more detail in the example presented next.

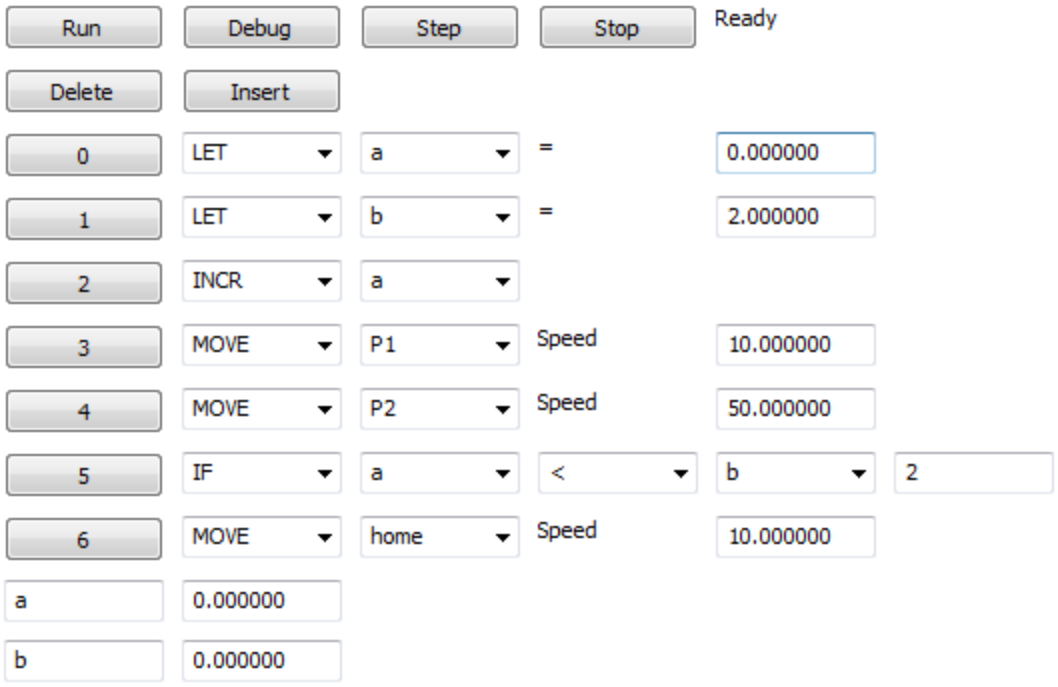


Figure 8: The program listing in the example.

Program execution is performed by the interpreter. When an instruction is executed its line number button turns blue to let the user know which instruction is currently being executed. The debug option displays a list of all the variables along with their current values. The user can see how these values change during program execution. The user can also step the program. That is execute one line and wait for the user to step again. This is useful for debugging.

The following is an example program. The user starts by creating the arm. Figure 9 shows the setup window the user uses to input the specifications of the arm. The window shows the setup for a 2 revolute joint arm with link length of 35 units each and $0°$ to $360°$ joint limits. Clicking the Create Robot button creates the robotic arm shown in Figure 12a and Figure 12b.

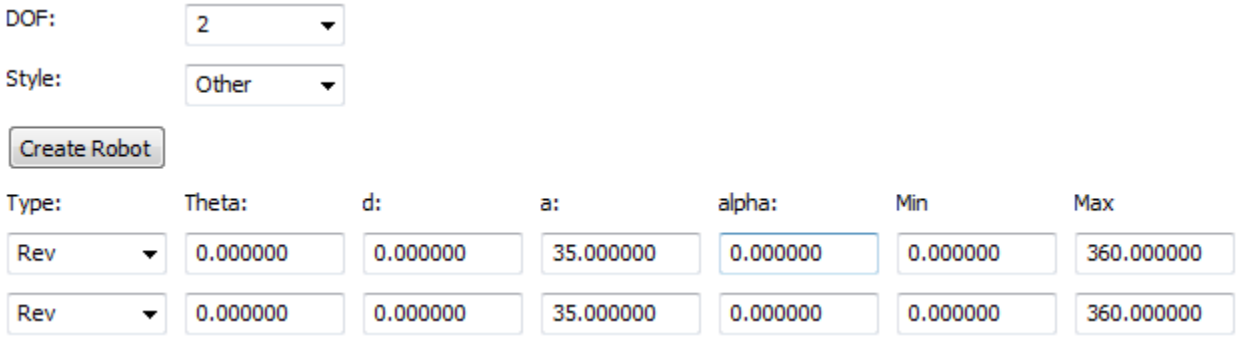


Figure 9: The window used to enter the specifications including the D-H parameters of the arm.

The program shown in Figure 8 above has two variables, "**a**" and "**b**". To loop 2 times we use the variable a as the loop counter. The **IF** statement cannot accept a number in its condition so the variable **b** is created and initialed with the number **2**. The **IF** statement forms a loop that loops until variable **a** reaches **2** which is stored in **b**. In the loop the variable "**a**" is incremented, the arm moves to position **P1** then it moves to position **P2**. The list of positions is shown in the position window of the tool shown in Figure 10.

Name Point	J1	J2	J3	J4	J5	J6
home	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
P1	21.600000	28.440000	0.000000	0.000000	0.000000	0.000000
P2	107.640000	50.040000	0.000000	0.000000	0.000000	0.000000

Figure 10: The list of points. Note there are only 2 joints, J1 and J2, used with this arm.

Figure 11 shows the listing as the program executes. At the bottom is the debugging information that includes the list of variables along with their current values. The current instruction being executed is highlighted by making its button red.

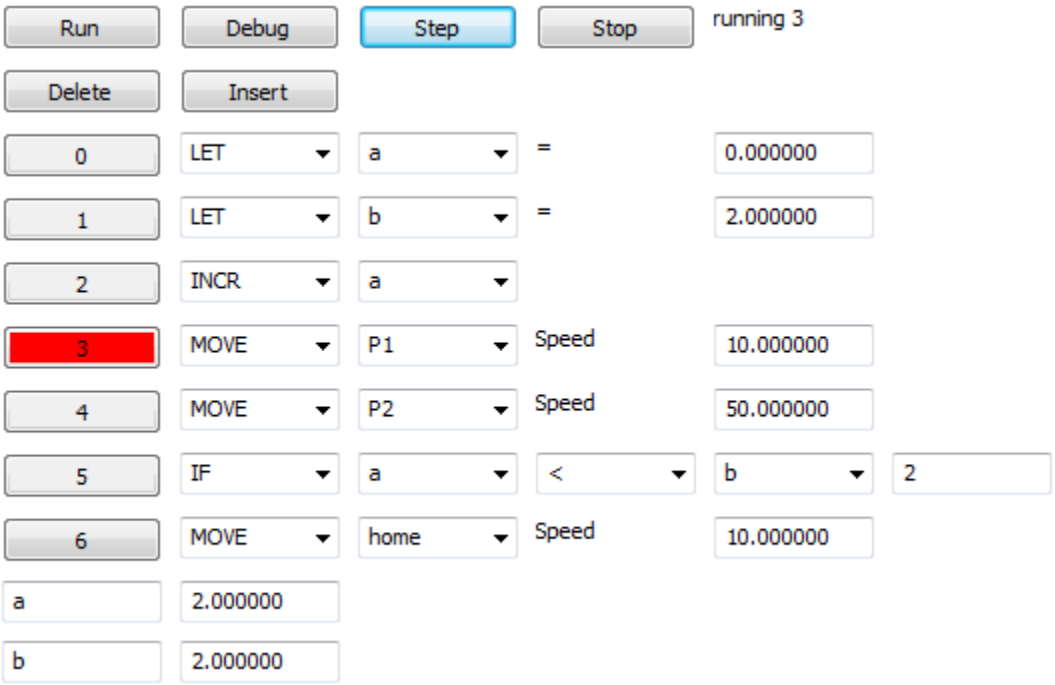
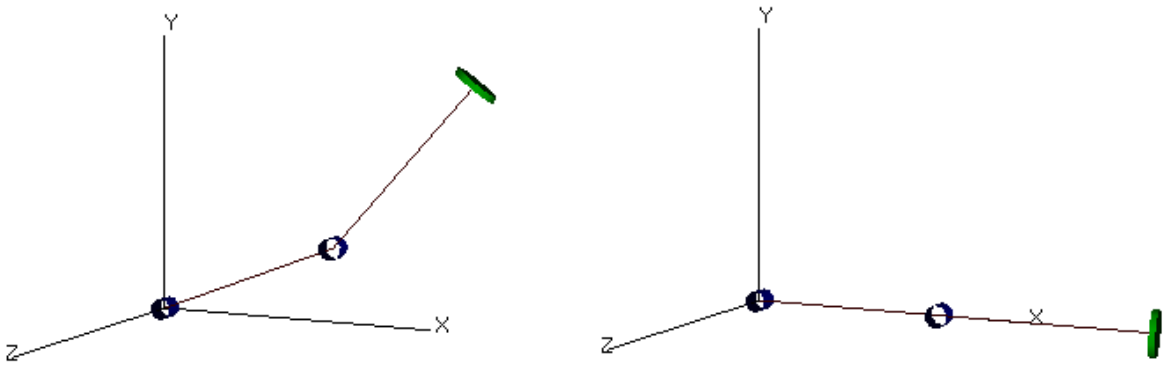


Figure 11: The listing shown while executing. Instruction 4 is highlighted in red indicating that instruction is being executed. Debugging information is listed below. Note a = 2 and b = 2.

Figure 12a shows the arm after executing the current instruction, instruction 3 which moves the arm to position P1. At the end, the program returns the arm back to the home position. Figure 12b shows the arm after returning to the home position.



a b

Figure 12: (a) The arm in position P1 after executing instruction 3. (b) The arm in the home position after executing the last instruction in the program.

The complete tool screen is shown in Figure 13 after executing instruction 3 and in Figure 14 after executing instruction 4. Note the arm used in this example was entered by the user. Any arm entered by the user can be used to program.

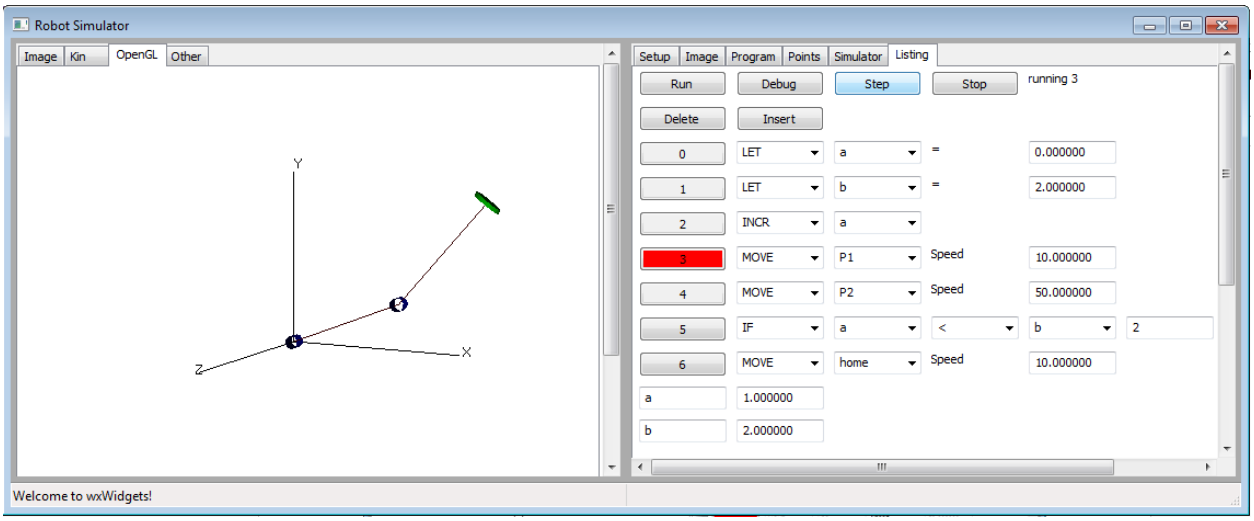


Figure 13: The complete tool screen after executing instruction 3.

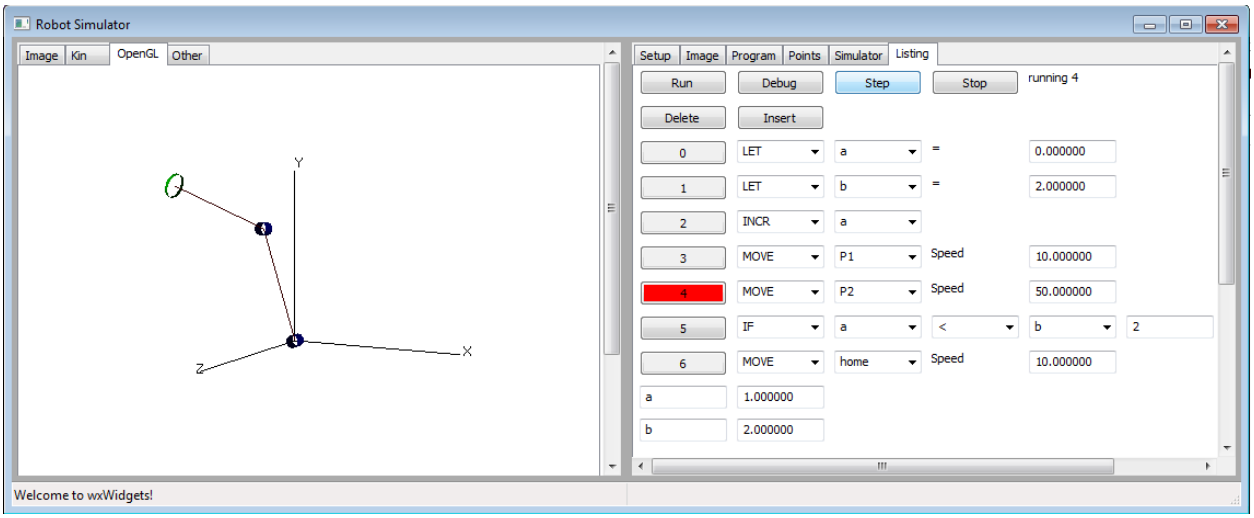
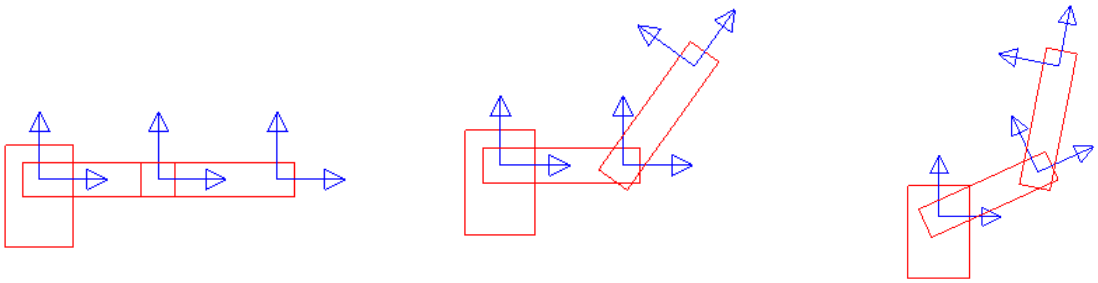


Figure 14: The screen after executing instruction 4.

VIII. Using the Tool to Teach the Forward Kinematic Equations

Robotic arms consist of links connected by joints. Each joint is connected to an actuator that can rotate or translate the link. Prismatic joints translate linearly as the actuator moves while revolute joints rotate. In a two revolute link robot where link 1 is connected to the stationary base and link 2 is connected to link 1, see Figure 15a, as the joint for link 2 moves link 2 rotates relative to link 1,Figure 15b. When Link 1 rotates it rotates relative to the base but it moves link 2 since it's attached to it Figure 15c. This dependency is represented by the product of transformation matrices explained below.



a b c

Figure 15: (a) A 2 link planner arm. (b) Link2 moves relative to link 1. (c) Link 1 move relative to the base.

The goal of the study of kinematics is to derive a set of equations that when given the desired position of the robot's end-effector, its hand for example, the angles for each joint that will result in the hand having this desired position (location and orientation) can be computed. That is given (x, y, z) the desired location of the hand in three dimensional Cartesian coordinates and the desired orientation of the hand, (r, p, y) roll, pitch and yaw then the angle that each joint must have $(\Theta_1, \cdots \Theta_n)$ to achieve can be computed.

The fundamental mathematical tool that is used to model the robot's kinematics is the transformation matrix. This matrix represents the location and orientation of a reference coordinate frame relative to some other frame. The idea is to attach a reference frame to each link of the robot then represent the location of the frame of each link relative to the frame of the link it is attached to. Each link is attached to another link with the first link attached to the base. The next few paragraphs are a brief explanation of the theory behind forward and inverse kinematic equations using homogenous transformation matrices. After this brief explanation, the description of how to use the tool to support the instructor in teaching this material is presented.

For example: Let P_A be point P represented in coordinate frame A, P_B be point P represented in Frame B and ${}^{A}T_{B}$ be the transformation matrix that represents frame B represented in the coordinates of frame A. Then

$$P_A = {}^{A}T_{B} P_B \tag{1}$$

So when a transformation matrix is multiplied by a vector that represents a point in some frame the resultant product is a vector representing the same point represented in the other frame.

Furthermore if in addition to ${}^{A}T_{B}$ we have ${}^{B}T_{C}$ the transformation matrix that represents frame C represented in the coordinates of frame B then we have

$$P_B = {}^{B}T_{C} P_C \tag{2}$$

and combining the two equations we get

$$P_A = {}^{A}T_{B}\ {}^{B}T_{C} P_C \tag{3}$$

where P_C is the point represented in frame C. In general we use A_i to represent ${}^{i-1}T_{i}$ the representation of frame i in terms of the coordinates of frame $i-1$. Each link is represented by a matrix called A_i for link i. The complete robot is then represented by the product of all the A matrices or $T = A_1 A_2 \cdots A_n$.

Note that when a joint angle changes the position of the link's frame relative to the link it's attached to changes and therefore the A_i matrix for that link changes. The A matrices are functions of the four D-H parameters and include the rotation the link has relative to the link its attached to, Θ the length of the link, d the joint offset, a and the joint twist, α. All are constants describing the physical characteristics of the link except for either Θ for revolute joints or d for prismatic joint.

The matrix $T = A_1 A_2 \cdots A_n$ is called the forward kinematic equations and is a function of actuator settings either Θ or d. Given the current position of all of the joint angles and displacements the forward kinematic equations produces the transformation matrix that relates the joint angles and displacement to the position of the end-effector. Our goal is the opposite that is we know the desired end-effector position. It's the joint angles and displacements that need to be calculated so that they can be given to the actuators to move. Our goal is then to produce the inverse kinematic equations that give the joint angles and displacements given the desired position of the end-effector.

Currently the tool does not have a way to allow the user to input a set of inverse kinematic equations or to program the arm by directly controlling the simulated joint motors. While this will

change in future versions, at this time the tool can only be used to verify the student's work. The tool does compute the forward kinematic equations and renders the arm using these equations. The student can derive the forward kinematic equations, then given a set of joint angles or displacements the tool can compute the coordinates of the hand in the Cartesian world frame. Figure 16a shows the sliders the student can use to input the joint angles. Figure 16b shows the resultant world coordinates produced by the tool's forward kinematic equations. Figure 17 shows the tool's complete screen including the rendering of the arm corresponding to the inputted joint angles. The tool does not need to be in simulation mode for this part to work. The student can also derive the inverse kinematic equations and use the tool to verify their work as well. The student can input a point in the world frame, compute the joint angles then enter these joint angles in to the tool and have the tool move the hand to the position specified by the inputted joint angles.

a b

Figure 16: (a) The joint angle sliders where the student can enter the joint angles computed by hand. (b) The corresponding world coordinates produced by the forward kinematic equations of the tool.

I. Using the Tool to Teach Path Planning

Path planning is the creation of polynomials that given the time as input returns the position, velocity and acceleration of each joint. It is used to plan a smooth path when moving the arm from one position to another. The inverse kinematic equations only tell the angles at the destination. It does not specify how the angles change over time in going from their current angle to their destination angle. Typically a robotic arm in which one override the controller and get access to the joint angles can be programmed by specifying the velocity of each joint. The student moves the arm by creating polynomial that continuously telling each joint the velocity it needs to have. The joint's feedback controller then adjusts the power to track this velocity.

To study path planning the tool has a mode where it models a real robotic arm which needs to be controlled at the joint level. Each joint can only be moved by specifying a velocity. The links then moves from their current location based on the input velocity. One cannot simply tell the joint to instantaneously be at a particular angle. It must be moved by commanding it to move at a specific velocity and wait until it works its way to the desired angle. Furthermore the velocity is limited by the maximum acceleration. An actual joint motor can only provide a limited amount of force that translates to a maximum acceleration given its fixed mass. The maximum acceleration is input to the tool and the velocity is then limited by this acceleration.

At this time the tool does not have the capability to allow the user to enter these polynomials. This is a similar to the situation where the tool does not allow the student to enter the inverse kinematic equations.

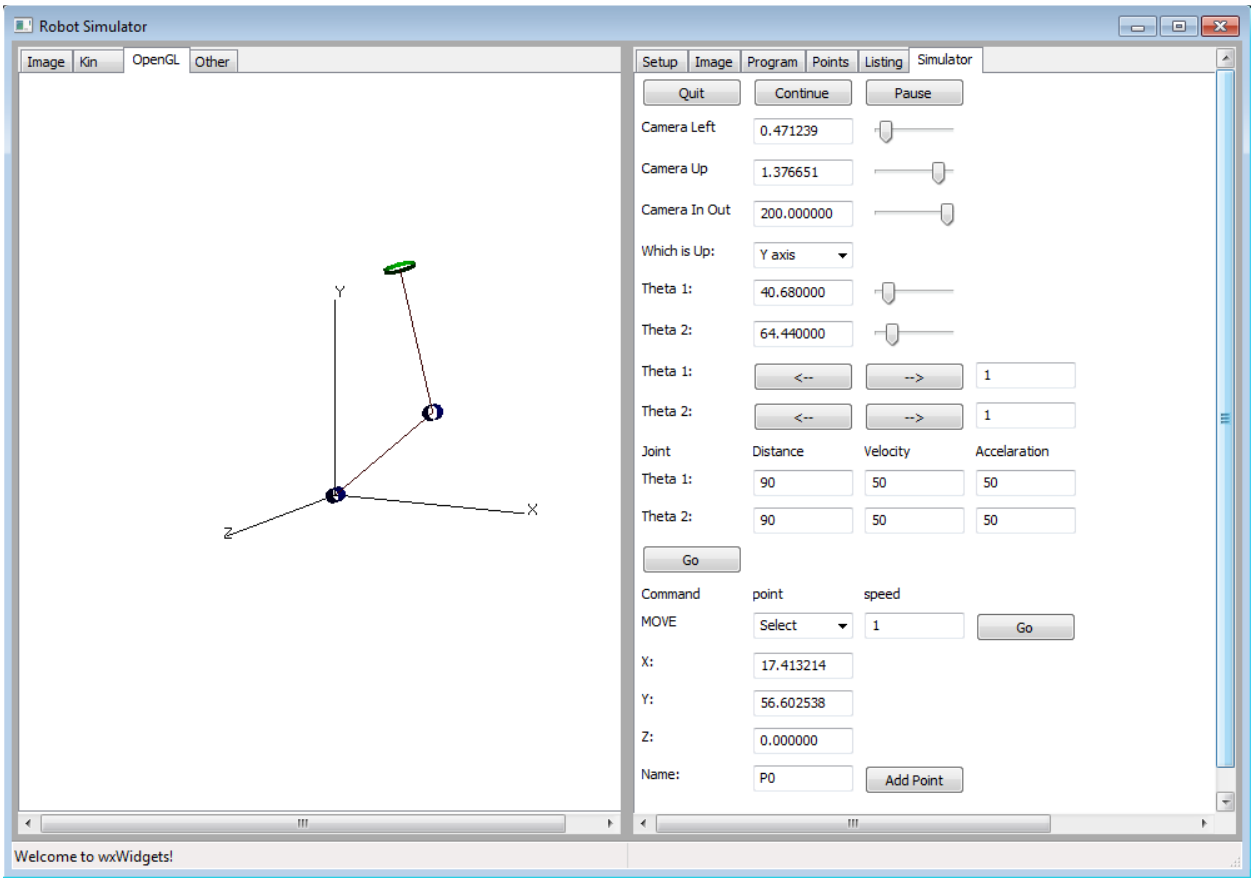
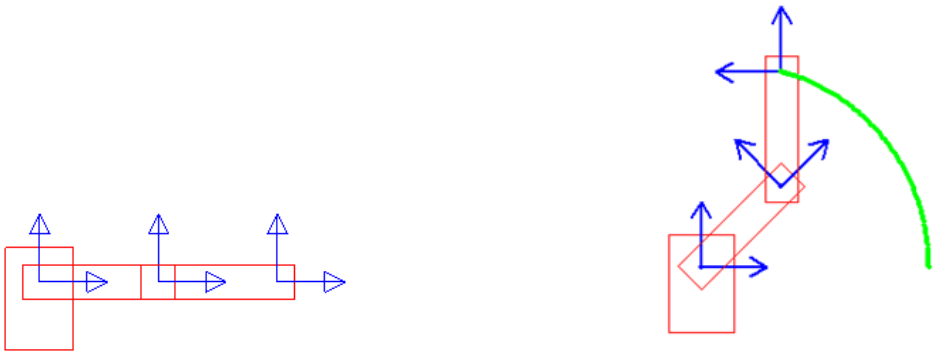


Figure 17: The tool's screen with the inputted joint angles, the outputted world coordinates and the corresponding image of the arm given the joint angles.

Eventually the student will be able to move the arm by creating a path plan which specifies the velocity of each joint. A poor plan will result in the arm not moving as desired and possibly overshooting the target destination angle or perhaps the arm may run into an obstacle if the various joint's movement are not coordinated properly. This will be evident by observing the movement just as one will do with a real robotic arm.

Currently however there is support to allow the student to design a single 2-1-2 blend type path plan. By single we mean that a single trajectory may be planned as opposed to a sequence of movements as is done when creating a general program. To create a path plan, for each joint the student must enter the distance to move in degrees, the velocity to cruse at and the acceleration. Then the arm moves according to the path plan. Note the student only specifies the parameters of the plan and does not actually develop the polynomials. Developing the polynomials, while harder, will result in the student obtaining a better understanding of path planning. This is why this is planned for a future version. The 2-1-2 blend is a very common path to plan and the easiest of the blend types. The student will learn how to set the parameters that yields the desired path. In the example below a path was created from the home position, see Figure 18a to a position where link 1 is at $45°$ degree relative to the base and link 2 is $45°$ degrees relative to link 1. The speed was set to 5 degrees per second and the total time to arrive it 10 seconds, see Figure 18b. The path of the hand as it moves is recorded by painting green circles as it moves. It's as though the hand has a pen and is drawing as it moves. This gives the student a record of the resultant path. The

movement can be seen including how the hand accelerates at the beginning and slows down towards the end of the path.



a b

Figure 18: (a) The arm in its home position. (b) The arm after joints 1 and 2 move by 45° **while the hand's path is traced.**

II. ASSESSING THE TOOL

The tool is being used for the first time in spring 2015. The instructor will present the lecture as usual without the use of this tool. The students will be tested on the material covered. Then after these exams the tool will be available for the students to use. The final exam will be comprehensive and will retest them on this same material. No additional instruction will be given to the students in these areas and so the only difference in their preparation is the use of this tool and any additional studying they may perform. The design of this approach was influenced by the fact that the tool was not ready until after the corresponding material was covered in the class and the students were tested. Unfortunately the results of the exams before and after the use of the tool will not be available for the printing of this paper.

In addition a survey will be given to the students at the end of the course to ask about such issues as the usefulness of the tool in helping them learn the material, the appropriateness of the tool in general, the ease of use of the tool, its user friendliness, and what features or changes would they recommend. However please note that the only other resource that is really competing with this tool is the library of MATLAB functions offered by Peter Corke and this resource require the students to have previous programming skills making it not suitable to programs where the students do not have this programming skill. Consider also that some instructors do not want to have the students invest the extra time it will take to have them write programs even if they know how. This tool is not competing with this and possible other libraries but rather filling a gap where no other tool exist with this focused application. Since all the students must be treated equally, separating the students into two groups and giving each group a different tool will lead to some student having to write programs while the other do not. Therefore a survey is the only practical assessment tool we can use. The questions follow. The answer choices are (strongly agree, agree, neutral, disagree, and strongly disagree) numbered from 4 down to 0.

1. This tool gave me support in learning the D-H representation.
2. This tool was not aligned with the course content.
3. This tool was difficult to use.

4. Seeing the arm produced by supplying the tool the D-H parameters help me understand how to represent an arm using the DH representation.
5. The tool did not support me in understanding the use of the inverse kinematic equations.
6. This tool help me learn how to program a robotic arm using its own robotic language.
7. Assignments using this tool was better than simple pencil and paper homework problems.
8. I learned how to use the tool very quickly.
9. The tool help me learn how to create a trajectory for a joint.
10. The tool did not help me in understanding how to assign the D-H axis and compute its parameters.
11. The tool did not offer support in understanding how to write a robotic program.
12. I prefer to use this tool than to write a program using a library of image processing routines.
13. Using this tool was easy.
14. Even though I can represent any robot arm I wish I did not take advantage of this and only used the arms preexisting in the tool.
15. The tool was not much help in understanding trajectory planning.
16. I spent a lot to time learning to use the tool.
17. The tool help me understand how to use the inverse kinematic equations to program the robot at the joint level.
18. I would have preferred to write a program using a library of routines than using this tool.
19. Please indicate the time it took you to figure out how to use the tool.
20. Please indicate the time it took you to input a robotic arm once you had the D-H parameters determined.
21. What changes, improvements, or new features do you recommend for this tool?

Note many question are redundant and phrased differently to make sure they do not change their view based on how the question is worded. The survey will be given after they complete all the assignment which is towards the end of the course so the results will not be ready in time for this publication but will be presented in the presentation.

III. The Development of the Tool

One would think that one of the hardest parts of implementing this tool is to draw the robot. Since the robot moves the drawing function needs to know the joint angles and be able to know the position of each link. Then each individual line making up the links must be drawn such that the position is correct. However the very same robot theory that this tool is teaching the students can be used to draw the robot. Recall that the forward kinematic equations determines where the location of the hand is in the world frame given all the joint angles. As the arm moves, the joint angles change and the forward kinematic equations give the position of the hand at that time. The hand can then be drawn knowing where it is to be drawn. The rest of the links are drawn in the same way by generating the forward kinematic equation from the base to the particular link.

For example the following code is all that is needed to know how to draw a 2 DOF arm to its current position.

```
FillA(A1,theta1,0,L1,0);
FillA(A2,theta2,0,L2,0);

l1 = A1*Link1;
```

```
l2 = A1*A2*Link2
```

Note the **FillA**(.) function produces the *A* transformation matrix given the 4 D-H parameters that totally describe the link including its current position. Link1 and Link2 are matrices of points that define the boundary of the link represented in the links own frame. And **l1** and **l2** are matrices of the boundary points but represented in the world frame already corrected for the links relative position within the robot and for the arms current position. So drawing the robot simply consists of drawing lines between all pairs of points in **l1** and **l2**. This feature allows the tool to dynamically draw any arm specified by the user.

The graphical user interface (GUI) and drawing windows were implemented using wxWidgets. This allows the tool to be ported to many platforms in future implementations. A library was created to interface between the wxWidgets and the tool's code so that the program does not have any code specific to any particular GUI tool. Then porting it to other languages is also easier. For example porting to Java will require replacing all the wxWidgets calls with Java Swing calls however only the library created will need to be changed. The tool's actual code will remain the same. Due to the amount of existing code we have from previous projects all written in C++ we decided to implement this tool in C++.

The most difficult part of this project was to justify the time and effort in creating the tool. Since there are many preexisting tools that are used for both image progressing and robotic simulations a careful study was performed to investigate the appropriateness of using these tools. The merit of this tool then became clear; to create a tool that is specifically designed to support the instructor and students in teaching and learning in the Introduction to Robotics course. This includes a tool that does not require a significant effort to learn how to use, allows the students to use any robotic arm in the book or elsewhere, allows for programming at the joint level, and finally a tool that performs just the right amount of functionality leaving the students to implement the rest.

IV. Future Enhancements

There are many improvements planned for future versions. The first is to integrate the vision component with the rest of the tool. This will allow the student to attach a camera to the arm or ceiling looking down at the table. The image from the camera will then correspond to where the camera is pointing and what is in its view. The student will then be able to implement a program that includes robotic vision algorithms to identify and locate the part within the image, then use the kinematic equations to locate the part in the world frame. The student will then be able to program the arm to reach and grab the part.

The second major improvement is to add capabilities to decipher equations. This will allow the student to enter the inverse kinematic equations. This involves adding a compiler to the code to compile the equations.

Most all commercially available robots, even those used for educational purposes, 14 do not allow one to program the joint angles directly. Instead their controller is programmed using a robotic language such as the one presented above where one commands it to move to a specific location at a specific speed and the controller performs al of the inverse kinematics. One generally does not have the option to override the controller and control the joint angles directly. There are safety concerns in overriding the controller as well. The third major improvement is to expand the compiler to include other programming structures such as if-else, loops and so on to allow the student to program the arm at the joint level.

V. Conclusions

In this paper we presented a tool to support the instructor and students in the basic Introduction to Robotics course. This tool allows students to input any robotic arm by just entering the arm's D-H parameters and the joint movement limits. Then the student can program the arm using a robotic type of language, use the arm to verify their forward and inverse kinematic equation computations and design a 2-1-2 type path plan. The tool supports 3D graphics with zoom and pan features.

References

1. Robotics Control, Sensing, Vision, and Intelligence, K.S.Fu, R.C.Gonzalez, C.S.G.Lee, McGraw Hill, 1887.
2. Fundamentals of Robotics Analysis & Control, Robert J. Schilling, Prentice Hall, 1990.
3. Applied Robotic Analysis, Robert E. Parkin, Prentice Hall, 1991.
4. Introduction to Robotics Mechanics and Control, 2nd Ed. John J. Craig, Addison Wesley, 1989
5. Robot Dynamics and Control, Mark W. Spong, M. Vidyasagar, Wiley, 1989.
6. Robot Modeling and Control, Mark W. Spong, Seth Hutchinson, M. Vidyasagar, 2006
7. Introduction to Robotics Analysis, Control, Applications, 2nd Ed. Saeed B. Niku, Wiley, 2011
8. Robotics, Vision and Control: Fundamental Algorithms in MATLAB, Peter Corke, Springer Science & business Media, 2011.
9. Peter Corke MATLAB tool box for robotics. URL: http://www.petercorke.com/RVC/top/toolboxes/
10. GAZEBO Robot Simulation Software. URL: http://www.gazebosim.org
11. Robologix Logic Design Inc. URL: http://www.robologix.com
12. Webots 7 by Cyberbotics Professional Mobile Robot Simulator. URL: http://www.cyberbotics.com
13. Robotics Developer Studio. URL: http://msdn.microsoft.com/en-us/library/bb648760.aspx
14. Intelitek Robotics and Educational Systems. URL: http://www.intelitek.com