ME 446 Laboratory #1
Kinematic Transformations

Report is due at the beginning of your lab time the week of February 12th. One report per group. Lab sessions will be held the weeks of January 22nd, 29th, and February 5th

Objectives

- Introduce the TMS320F28335 DSP controller for the CRS robot arm and using Code Composer Studio IDE to program the TMS320F28335 DSP with C.
- Derive the forward kinematic equations for CRS robot arm following the Denavit-Hartenberg (DH) convention
- Derive inverse kinematic equations for the CRS robot arm
- Use the provided Code Composer Studio project to verify your solution

NOTE: CRS robot arm has five motors/joints, in this class we are only using/controlling the first three.

Part 1: forward kinematics

In this part you will derive solutions to the forward kinematics problem for the CRS robot arm. First find parameters of the CRS robot arm following D-H convention, then verify the theoretical result using a given C program in Code Composer Studio.

Physical Dimension

Below is the physical dimension of the CRS robot arm used in this lab. Use this when defining the D-H parameters.

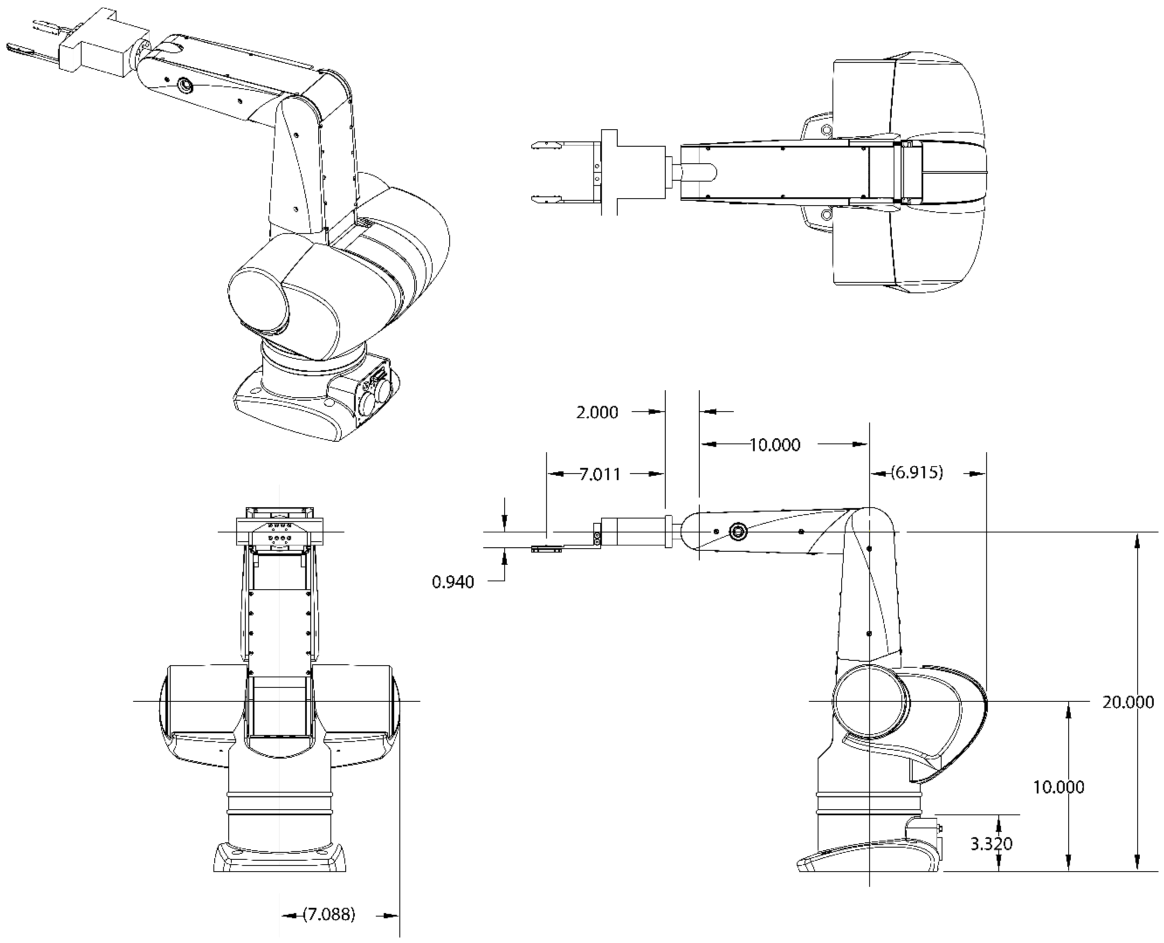

Figure 1.1 Physical Dimensions of CRS robot arm

Theoretical Solution

Find the forward kinematic equations for the CRS robot. In particular, we are interested only in the position of the end effector. We will use Matlab to find expressions for each of the three components of d_3^0.

Verification

For any provided set of joint angles $\{\theta_1, \theta_2, \theta_3\}$, we want to compare the calculated position of the end effector to the actual position of the end effector. Your C code will print the calculated position and with rulers we will roughly measure the actual positon of the robot arm for the comparison.

Procedure

1.1 Cloning your Git Repository

For your lab assignments, all the given code is stored in a GIT repository at github.com. Your instructor will walk your through the process of cloning you new repository. The steps for this are found in the document http://coecsl.ece.illinois.edu/me446/Using the ME446 Repository.pdf.

1.2 Code Composer Studio (CCS)

In this portion of the lab, you will be modifying a given Code Composer Studio project to read joint angles of your CRS robot arm.

1. Open Code Composer Studio, select the "workspace" folder created in the last step.
2. Click project → Import New CCS Project, click "Browse" then select C:\<your folder>\<reponame>C2000Ware_4_01_00_00\device_support\f2833x\examples\f28335_me446starter.
3. Once your project is loaded into CCS, rename the project ME446Labs<yourinitials>. This will differentiate your workspace and project from other groups. Then find the file me446lab.c and rename it Lab1<yourinitials>.c.
4. Power on the robot arm and run the project according to the procedure found in <yourRepoName>/Procedure for Running Robot.docx. Once your starter code is running for more than 4 seconds, you will see the LED on the emergency stop button begin to blink on and off. The robot is not moving because the starter code outputs a control effort of 0 to the waist, shoulder and elbow motors. The starter code also prints the three motor thetas to one of the DSP's UART serial ports. You can view this text in the serial terminal program "TeraTerm." Run TeraTerm and setup its Serial Port settings so that it is connected to the highest COM port (usually COM4, COM5 or COM6). Also make sure the Baud rate is set to 115200 bits per second. Now you should see text displaying in the terminal, which is the three motor thetas in degrees. Move around your robot arm and see the angles changing.
5. Now you will calibrate the start position of your robot arm as discussed in lab lecture and by your TA. There are two positions of the robot arm that need defined. One is the "zero position" and one is the "start position." The zero position is the position of the arm in Figure 1.1. Your TA will show you arrow and lines printed on the robot arm that will help you put the robot arm in the exact (or at least close) zero position. For the start position, it is nice to start at a resting position so you can easily and pretty accurately put the robot back in that position each time you start your code running. Move the robot arm so that the arm is facing forward. Shoulder joint pushed back as far as it will go. Elbow joint pushed down until it stops moving and is resting on the shoulder link. This is the "Start Position." Since this is slightly different for each robot arm we need to run a quick calibration. In your CCS project open the file lab.c. At the top of that C file you will find the lines

   ```
   float offset_Enc2_rad = -0.37;

   float offset_Enc3_rad = 0.27;
   ```

   The -0.37 and 0.27 are offsets in radians from the zero position to the start position. Enc2 is measuring the shoulder and Enc3 is measuring the elbow. You will find the offsets for your robot arm. Change these lines of code to

   ```
   float offset_Enc2_rad = 0;  //-0.37;   Keep this so you remember the sign

   float offset_Enc3_rad = 0; //0.27;  Keep this so you remember the sign
   ```

   Recompile and debug your project to download this new code to the controller. Before you run this code, put the robot arm in its zero position. You may have to have someone hold in this position. Now run your code. Once the code is running simply move the robot arm back down to its start position. View the angles printing in TeraTerm. These are the offsets for your robot arm in degrees. Convert these to radians and enter them in place of the -0.37 and 0.27 values. The Enc2 value should be set negative and the Enc3 value should be set positive. **Show your TA** your calibration by debugging and running your code with your calibrated offsets and starting the robot arm in its start position. Then once the code is running, move the links to the zero position and your theta values should be pretty close to zero degrees.

NOTE: output of optical encoders will be reset to 0 every time the CCS project is downloaded and run on the DSP (or the code restarted). Make sure your robot arm is at the home position before running your C program every time you run your code in this lab and all labs to come.

6. Additionally, we have created some interfaces between the DSP and MATLAB and the DSP and SIMULINK. These next two items will have you play with these two interfaces to see how they work. Note though that the DSP has two serial ports to interface with MATLAB and SIMULINK but the PC only has one serial port. Your TA will show you where to plug in the PC's serial port for working with the MATLAB functions and where to plug in for working in SIMULINK.
First the MATLAB functions. If you look at the top of lab.c you will see two global variables, *whattoprint* and *theta1array*. Before these variables there is a #pragma statement that tells the linker to locate these variables in a special memory section that the MATLAB functions can look in and find all the variables MATLAB can either write to or read from. For these functions to work you MUST HAVE THE DSP RUNNING and you MUST CHANGE THE CURRENT DIRECTORY of MATLAB to c:\<yourcreateddirectory>\<yourrepositorydirectory>\workspace\ME446Labs<yourinitials>\matlab. With MATLAB in this directory, it can locate the project files it needs to discover the location of these special variables. The three functions you will be using are ME446_serial_ListVars, ME446_serialwrite, ME446_serialread. Check out help on each of these functions by typing "help ME446_serialread" for example. Also before you try these functions make sure COM1 is connected to the serial port cable labeled MATLAB (discussed above). Experiment with each of these functions by first playing with the two given variables *whattoprint* and *theta1array*. (For *whattoprint*, have your instructor show you how to display its value in CCS watch expressions.) Then add one more float array to save 100 theta2 values and another float variable which you will print out to Tera Term. To change what is printed to Tera Term, change the printf statement in the function "printing" at the bottom of you lab C file. **Show your TA** that you are able to read from and write to these variables and arrays from MATLAB.
7. In addition, there is a Simulink interface that allows you to upload 4 32 bit integers from the DSP and download 7 16 bit integers to the DSP every 5ms. Probably the most useful use of this Simulink interface will be the saving of response data through the uploading of the 4 32 bit integers. To upload floating point numbers we multiply the floating point number by 10000 and upload it as an integer and then when it is received in Simulink it is divided by 10000 to give a float number but with

at the most four decimal places of precision. Feel free to change this 10000 factor if you need more precision but keep in mind the maximum number a 32 bit integer can store is 2^31 – 1, 2147483647. Notice at the top of lab.c there are four float variables, Simulink_PlotVar1, Simulink_PlotVar2, Simulink_PlotVar3, Simulink_PlotVar4. To upload data to Simulink you simply have to write values to these variables every time the lab() function is called and automatically the serial interrupt function will transfer these values to Simulink when requested. Note, due to the speed of the serial port, Simulink will only request the values every 5 ms. The lab() function is called every 1 ms., so only every 5th number written to these variables will be uploaded to Simulink. Since Simulink is performing the request and the lab() function does not know which sample the request will occur, we write to the variables every millisecond so the latest data is always ready to send. Go to the bottom of the lab() function and you will see that three of the Simulink variables are being assigned the three motor angles. Also search in F28335Serial.c for the four Simulink variables and you will see that they are being multiplied by 10000 before being sent over the serial port. Test this Simulink interface by, in MATLAB, making sure you are still in your project's matlab directory. Then launch the Simulink file "simulink5ms_plotAndGains.slx". Make sure that the robot controller is running your DSP code and then start Simulink by selecting the "DESKTOP REAL-TIME" tab and then clicking the "Run in Real Time" button. The first time you run the Simulink file it will need to build itself, so be patient until it starts. Once Simulink starts advancing in time, open Data Inspector Matlab App. Your instructor will show you how to use Data Instructor to plot and save the robot's responses. **Show your TA** by moving the robot joints and seeing the three data streams changing. If you would like you can also play around with downloading values to the DSP. Your TA can show you the variables that are updated when you download values.

8. For one more initial exercise with the robot arm, determine the positive direction of the robot joint motor's angle and the positive torque direction of each joint motor. Produce a stick figure picture showing the positive direction of the three joint angles and torques. To do this you will slightly modify the code in the function "lab" in your lab1<yourinitials>.c file. The function *lab(float theta1motor,float theta2motor,float theta3motor,float *tau1,float *tau2,float *tau3, int error)* is called by the supporting code once every millisecond. It is passed the radian value of each of the three motor joint angles and references to the three torque commands for the joint motors. Acceptable values for tau1 through tau3 are a real number between -5 and 5. The unit of this -5 to 5 number is directly proportional to torque. For this exercise, one at a time, change the line of code that assigns an open loop torque to each of the motors and run the robot while watching the theta angles displayed in Tera Term. For example change the line of code *tau1 = 0.0; to *tau1= 1.0;. Debug your code and perform the required steps to run the robot arm and you should see joint one turn in its positive direction until it goes outside of the safety region and then will stop. Repeat for joint two and three zeroing the other joints so only one joint moves at a time. Indicate on Figure 1.2 the positive direction of the three joints and **show your TA.**

1.3 D-H Coordinate Transformations. *(Completed in HW)*

1.2.1 When using the D-H convention, proper coordinate frames have to be defined. Figure 1.2 gives a sketch of the CRS robot arm with all the z axis of joints fixed. Label the rotation directions of all joints according to the positive direction you found in 1.2 above.

1.2.2 Use right hand rule to determine the direction of all z axis. Then, on figure 1.2, complete the D-H frame by properly adding x axes.

1.2.3 Finish the D-H table below. Refer to Figure 1.1 for physical dimensions of CRS robot arm.

Joint	a_i	α_i	d_i	θ_i
1				
2				
3				

1.2.4 **Show TA your D-H table.**

1.4 Theoretical Solution

1.3.1 Using the D-H values and Matlab find the forward kinematics of the CRS robot arm. BIG HELP here: So that the equations simplify nice in Matlab, wherever in your DH matrices you have "pi" substitute "sym(pi)". This helps Matlab simplify your equations when you run the "simplify" function in Matlab. Also after you run simplify Matlab may give you some fractions. If you would like to see your equations without the fractions you can run the command "vpa" on your equations.

1.3.2 Find and include in your report H_3^0.

1.3.3 These equations will be a function of the thetas defined by the D-H method. The angles of each of the three motors of the robot link will not necessarily be the same angles as define by the D-H method but we do know that the D-H thetas can be calculated by a linear combination of motor angles. The easiest way to find the relationship between theta1, theta2 and theta3 and motor1_theta, motor2_theta, motor3_theta is to move the linkage to different 90 degree joint positions and record in a table the motor theta values and the values of the D-H thetas. To find the linear combinations for motor thetas create 3 unknowns that can be solved for using the different 90 degree points. So for example using some constants c1, c2, c3 set theta3 = c1*motor_theta2 + c2*motor_theta3 + c3 create 3 or 4 equations using your above table values of thetas and motor_thetas and solve for the c1, c2 and c3 constants. You can do this using Matlab or it may be possible for you to just look at three or four equations and figure out the values of c1, c2 and c3.

1.3.4 Now that you have equations for the D-H thetas in terms of the motor thetas, substitute these values into your forward kinematic equations to find the forward kinematic equations as a function of motor thetas.

1.3.5 **Show TA your equations.**

1.5 Verification

1.4.1 For demonstration your TA will ask you to move your robot joints to different sets of joint angles $\{\theta_1, \theta_2, \theta_3\}$. Your C program must print to "Tera Term" these joint angles along with the calculated x,y,z position of the end effector's joint.

1.4.2 Using a ruler, measure and compare the position of the end effector to the x,y,z position printed in the serial port terminal. Where is x,y,z (0,0,0)?

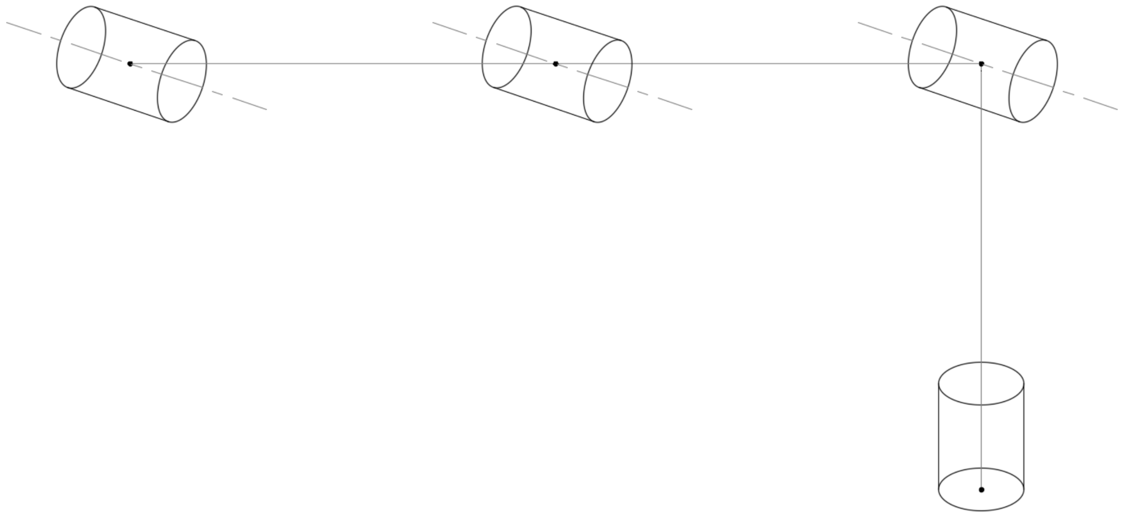

Figure 1.2 CRS robot (DH frames to be assigned)

Part 2: inverse kinematics

Geometric Approach *(Already calculated this in HW)*

Given a desired point in space (*x, y, z*), write three mathematical expressions that yield values for each of the joint angles. For the CRS robot, there are (in general) two solutions to the inverse kinematics problem. We will implement only the *elbow-up* solution.

Verification

For any provided point in space (*x, y, z*), we want to compare the joint angles of the robot arm indicated by your inverse kinematics equations, to what your CCS program gives after manually moving the end effector to the specified point.

Procedure

2.1 Theoretical Solution

2.1.1 Establish the world coordinate frame (frame *w*) centered at the center of the CRS's base. The x_w and y_w plane should correspond to the surface of the table, with the x_w axis straight ahead of the robot arm with θ_1 equal to zero. Axis z_w should be normal to the table surface, with up being the positive z_w direction and the surface of the table corresponding to $z_w = 0$.

2.1.2 Given a set of (x, y, z) coordinate, solve for the corresponding joint variables $\{\theta_1, \theta_2 . \theta_3\}$ in geometric approach. Write down the three mathematical expressions:

$$\theta_1(x_w, \mathrm{y}_w, z_w) = ?$$

$$\theta_2(x_w, \mathrm{y}_w, z_w) = ?$$

$$\theta_3(x_w, \mathrm{y}_w, z_w) = ?$$

2.1.3 You now have inverse kinematic equations to give you your defined D-H θ's. But in the control of the robot arm we will be controlling the individual motors of the robot arm. Use the equations you found in the forward kinematic calculations that equate the D-H θ's to the motor θ positions to rewrite the equations to solve for θ_{M1}, θ_{M2} and θ_{M3} given x,y,z.

2.2 Verification

For demonstration, your TA will ask you to move the robot arm to multiple x, y, z positions. Your C code should first use the forward kinematic equations from Part 1 to find the x, y, z position of the end effector given the measured motor angles. Then to verify that your inverse kinematic equations are correct, take this calculate x, y, z position and use it in your inverse kinematic equations to find calculated motor angles. These angles should match the robot's current measured motor angles. Your C code should print to "Tera Term" the motor's measured angles, the x, y, z position calculated by forward kinematics, DH angles and the motor angles calculated with your inverse kinematic equations.

Report:

Produce a lab report that details all the math used to calculate forward and inverse kinematics for the first three joints of the CRS robot arm. When explaining the forward kinematic derivation, make sure to explain how each DH parameter was found even if the parameter happens to be zero. Also detail steps taken to find the relationships between motor angles and D-H angles and note the positive torque direction of the joint motors. Pictures and sketches are a very important part in explaining your work and analysis. Hand sketches will be accepted ONLY if they are very neat and must be digitized. Computer aided drawings are preferred and will receive more points if done well. Also add to your report the COMMENTED code you wrote in Lab1<yourinitials>.c (or whatever you call your file). This report should have enough detail in it that a new student in this class would be taught how to perform the different tasks in this lab assignment.