

ECE 5425 Project1 Report

Group #8. Li Lin & Zhaofeng Tian

I. Abstract

In this study, a three-links robot is built with hardware components and Matlab model. An GUI for this robot is developed in Matlab App Designer, which enables users to tune DH parameters, offsets and base frame translation inside and change robot joint angle to modify the robot model, meanwhile users can observe the updated plotting in real time. Moreover, by using the error between the measured distance from the end-effector to the new base frame and the calculated one, this study designed an objective function and applied heuristic searching function to cut down the error, and made the change visualized and synchronized in the GUI. By using the GUI function and heuristic searching function, the final error is cut down from the previous value, 0.0054, to the improved value, 0.00096, which demonstrates the effectiveness of our design and robustness of GUI.

II. Introduction

As automation is growing fast all over the world, robotics is one of the most promising industries that would contribute significantly to human society. In the auto industry, the robot arm is commonly used to grab and move mechanical parts, while also help weld parts in the workspace; As a product of mechanical engineering, electrical engineering, and computer science, the robot arm can be commanded to do high accuracy motions which outperform human beings. Thus, the robot arm is used in the medical industry as well, to perform operations for the patients; In terms of STEM education, a three-link robot arm could interest kids very much, and inspire them to chase STEM dream in the future.

In this study, we propose a three-link robot arm called “G8 robot arm”, and showcase how we build it in physical world, create simulation environment in Matlab App designer, and how we optimize DH parameters using the comparisons between measurements of the distance from end-effector to clipboard frame origin and corresponding calculation results from simulation.

Rest report sections are arranged as follows. In section III, our methods of building robot in physical world and the simulation environment are proposed; In section IV, our results of DH parameter optimization are shown; Conclusion and Discussion are respectively put in sections V and VI.

III. Methods

In this section, the details of building process are illustrated step by step, which synchronizes with the project-required steps.

In the first step, we arrange the base on the clipboard such that we have the maximum joint ranges and that the robot can reach all corners of the given board with this additional criterion, we shorten one of the joint lengths of robot arm by the smallest digit from all group members student id. So, as the smallest digit in my group ids is “1”, we shorten link 1 and 2 by 1 hole from the max length shown as the Fig. 1.

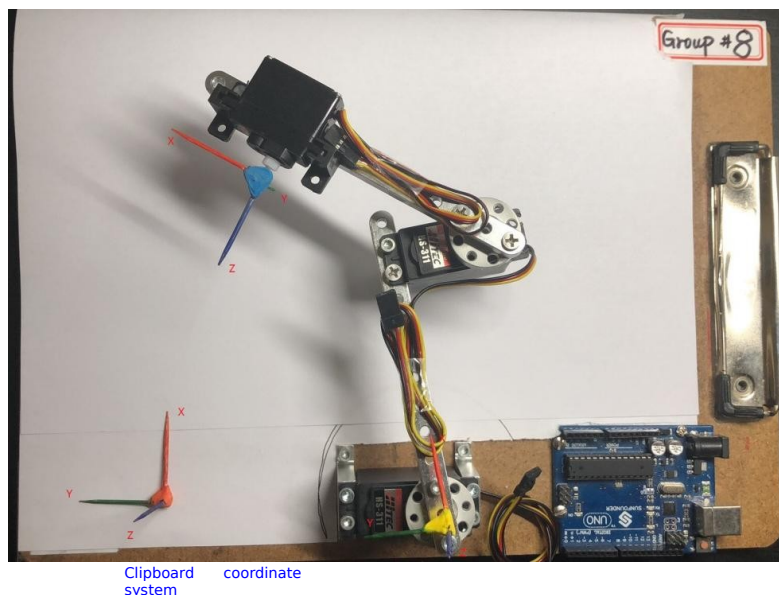


Fig. 1 Robot arm setup

As you can see, we move the base frame along the Y axis for 0.08 meters to get a clipboard frame, where the one of two toothpick-made frames locates. The other toothpick-made frame locates at the end-effector frame origin. Then all the measurements are pointed from end-effector to the clipboard frame instead of the base frame, which should be noted and is set as default in latter measurements.

In the second step, we tried several possible angles to locate the end-effector, and finally choose the current setup shown as Fig. 2(a). Meanwhile, we have drawn the range limits of the end-effector on a piece of paper.

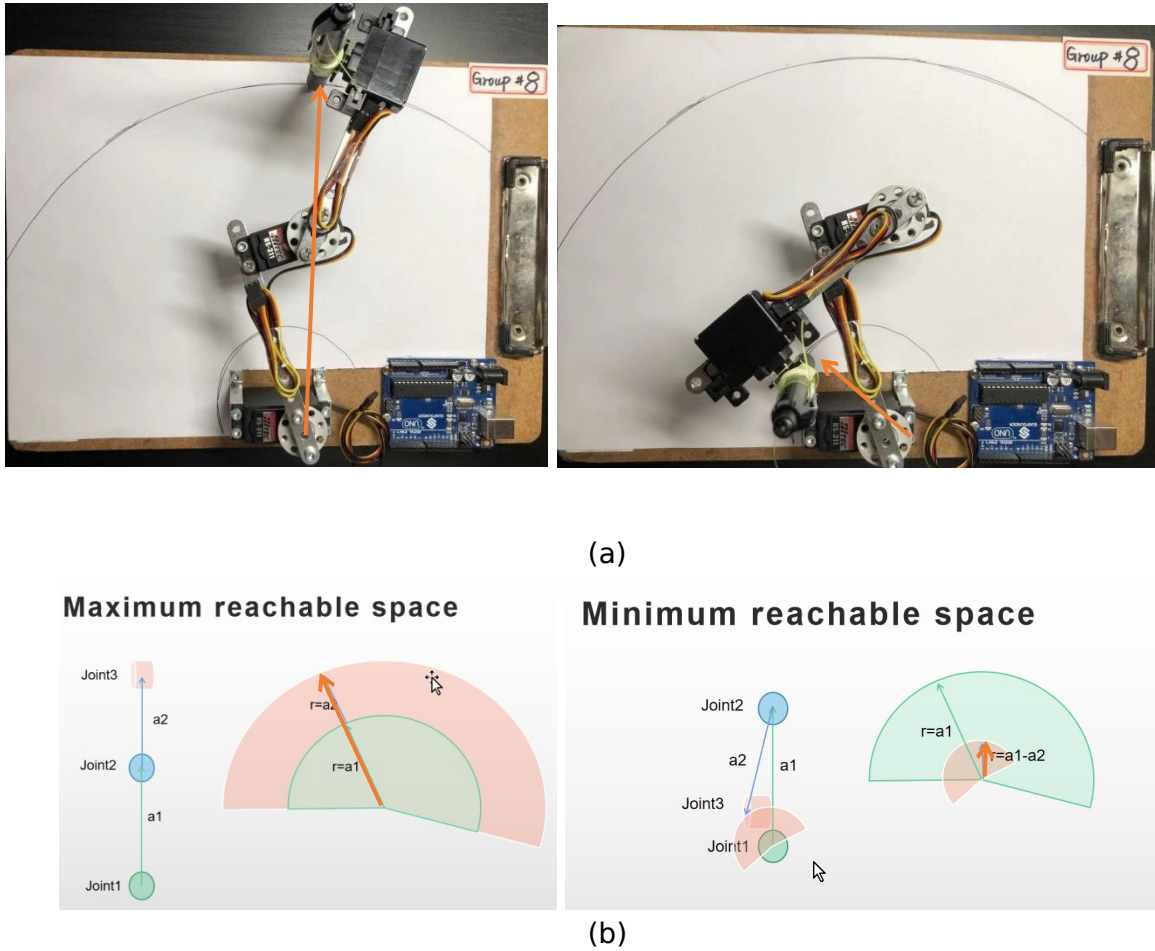


Fig. 2 Minimum and Maximum range of the end-effector.

From Fig.2 (b), we can get the reachable space of the end-effector as the red area shows.

In the step 3, we put the toothpick-made frames at the respective locations shown as Fig.1.

In the step 4, we measure the DH parameters for the robot arm following the textbook instructions. All measured DH parameters including theta value limit (joint angle limit) and offsets are shown in the Table. 1 below.

	a (m)	alpha (degree)	d (m)	theta (degree)	offset (degree)
Joint 1	0.1	0	0	-90 ~ 120	-19
Joint 2	0.073	-90	0.02	-120 ~ 90	16.8
Joint 3	0.0	0	0.02	0 ~ 135	0

Table 1. DH parameters.

1. Firstly, we create a GUI in the Matlab App designer shown as Fig. 3 to tune the DH

parameters and showcase the result by plotting the robot based on preset DH parameters. Note that Base Frame Translation section in the GUI, we set the clipboard coordinate system showing as Fig.1 which origin is [0 0 0]. So the base frame of robot arm translated with vector[0 -0.08 0.47] refer to clipboard frame. And we define the base frame and joint1 frame at the same place.

DH parameters

a1	0.1	alpha1	0	d1	0	offset1	-20
a2	0.073	alpha2	-90	d2	0.02	offset2	16.8
a3	0	alpha3	0	d3	0.02	offset3	0

Base Frame Translation

x	0	y	-0.08	z	0.047
---	---	---	-------	---	-------

Fig. 3 DH parameters GUI

2. Secondly, to explain how the DH parameters are determined, here we are using a image to illustrate it. In the Fig. 4, all DH parameters are labeled on the links and joints. In the Fig.5, offsets are labeled.

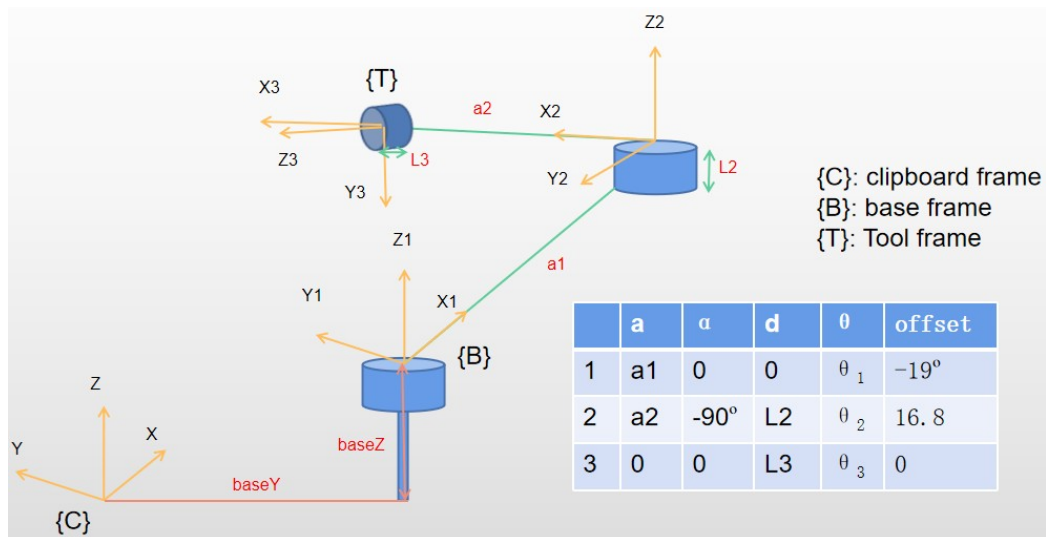


Fig. 4 DH parameters

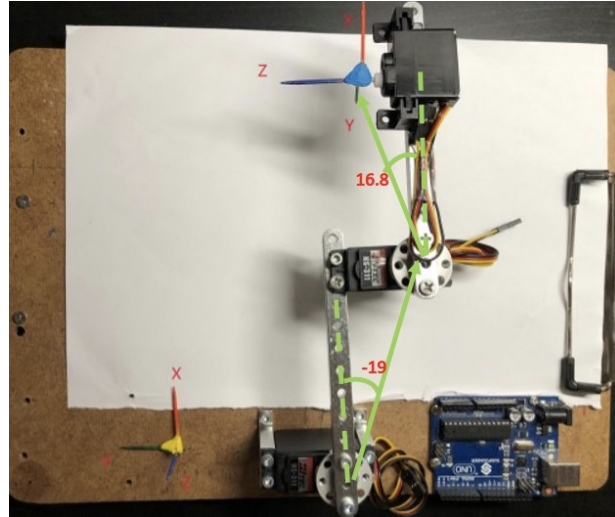
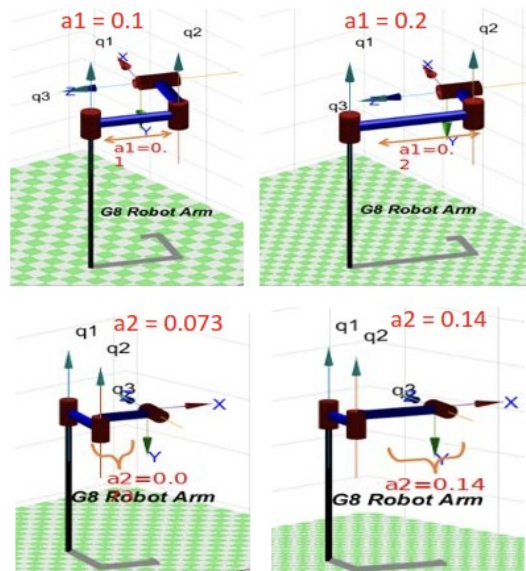
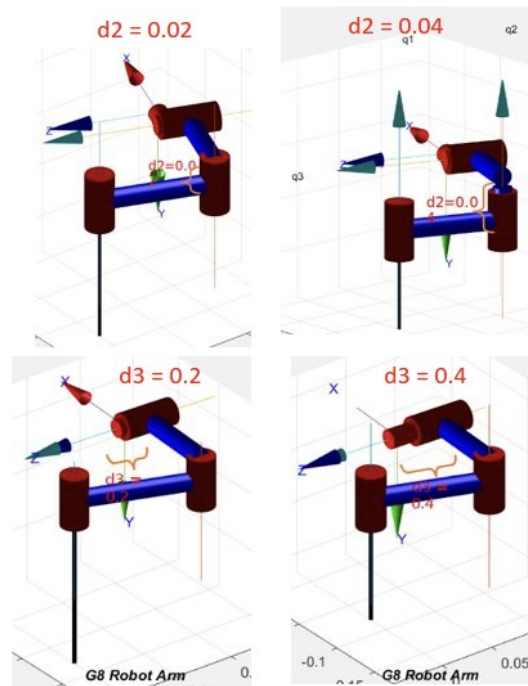


Fig. 5 offsets

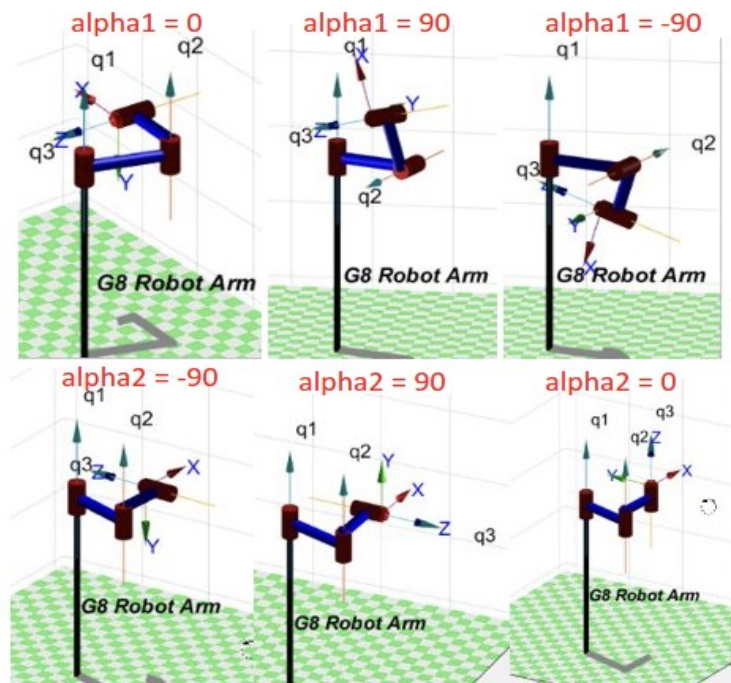
And we also use Matlab APP GUI to adjust DH parameters and plot the robot to see how each parameter's change will impact the robot arm structure. The comparisons are show as below in Fig. 6.



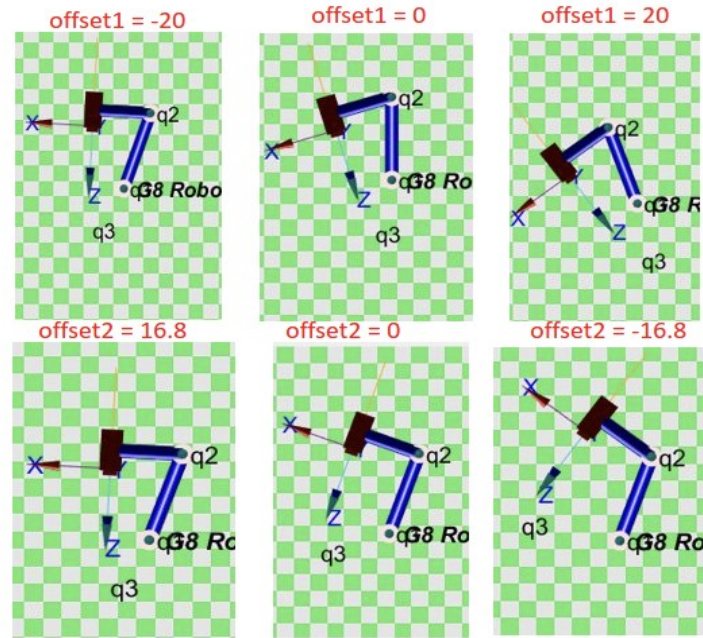
(a) parameter a



(b) parameter d



(c) parameter α



(d) parameter offset

Fig. 6 DH parameters comparisons

3. To create the robot from DH parameters, we use a CreateRobot function shown as Fig. 7. This is a callback function of “CreateRobot” button, which draws the DH parameters to each Link by using $L(i) = \text{Revolute}(\text{DH parameters})$, where “Revolute” means to create a link with a revolute joint. Then we create a robot variable app. G8Robot to store the robot information which is generated by SerialLink function, a function to create a robot from links we build earlier (combine those $L(i)$ to a full matrix, and feed it to the SerialLink() as an augment). Note that the joint angle limit is also presented in “qlim” label.

```

% Menu selected function: CreateRobotMenu
function CreateRobot(app, event)
    clear L
    deg = pi/180;
    %get the joint angles limits from slider.Limits
    qlim1 = deg2rad(app.Joint1Slider.Limits);
    qlim2 = deg2rad(app.Joint2Slider.Limits);
    qlim3 = deg2rad(app.Joint3Slider.Limits);

    % Define links of robot
    L(1) = Revolute('d', app.d1EditField.Value, ... % l.
        'a', app.a1EditField.Value, ... % l.
        'alpha', app.alpha1EditField.Value*deg, ... % l.
        'qlim', qlim1, ... % m.
        'offset',app.offset1EditField.Value*deg );

    L(2) = Revolute('d', app.d2EditField.Value, ...
        'a', app.a2EditField.Value, ...
        'alpha', app.alpha2EditField.Value*deg, ...
        'qlim', qlim2, ...
        'offset',app.offset2EditField.Value*deg );

    L(3) = Revolute('d', app.d3EditField.Value, ...
        'a', app.a3EditField.Value, ...
        'alpha', app.alpha3EditField.Value*deg, ...
        'qlim', qlim3, ...
        'offset',app.offset3EditField.Value*deg );

    %Build the robot
    app.G8Robot = SerialLink(L, 'name', 'G8 Robot Arm', .
        'tool',transl(0,0,0), ...
        'base',transl(app.xBaseTransl.Value, ...
            app.yBaseTransl.Value, ...
            app.zBaseTransl.Value));
    %Set the robot created flag
    app.IsRobotCreated = true;
    %plot the robot and update postion and distance info
    app.updateRobot();|
end

```

Fig. 7 Commands

4. The joint limits are declared in the “qlim” as an augment of Revolute(), meanwhile, in the latter design, we place a limit on the joint angle by setting limit values in the slider of the Matlab App designer.

In the step 5, we are using sliders to control the join angle and plot the robot in the GUI. Shown as Fig.8, the robot could be updated by each change in the sliders using a callback function that calls app.updateRobot() (a method in the class) to use app.G8Robot.plot function to plot the updated robot.

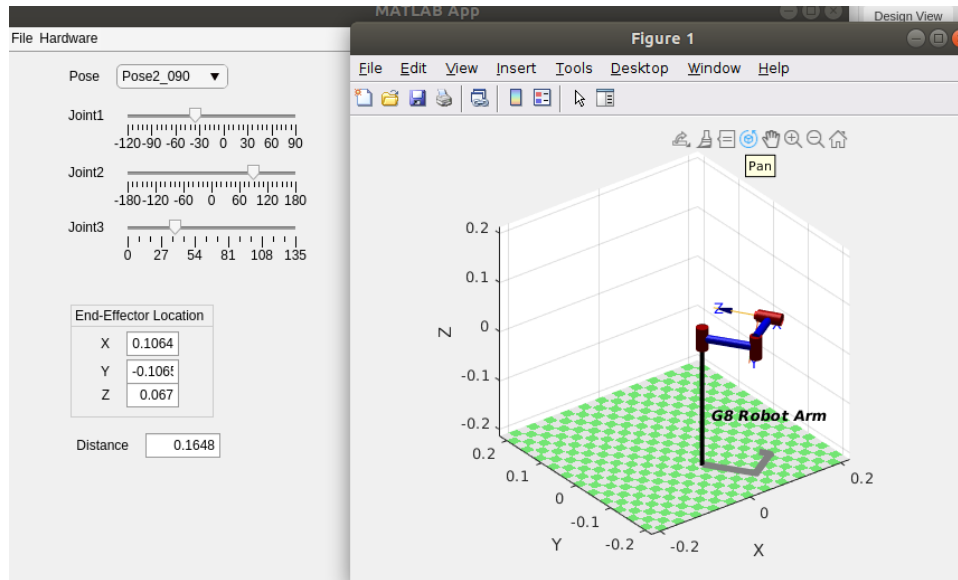


Fig. 8 Slider control

In the step 6, the codes used to implement keyboard control are shown in the Fig.9, we use `UIFigureKeyRelease(app, event)` function to get the key input in the GUI window where `event.key` can get exactly the key we input in the GUI. So, when detect arrow keys, the respective operations will be performed. In this case, we use “leftarrow” and “rightarrow” keys to control robot end-effector to move on the Y direction and use “uparrow” and “downarrow” to control robot end-effector to move on the X direction. For example, we get the “leftarrow” key as an input, the codes will update the end-effector transform matrix by changing the translation part with a motion along Y at 0.005, then get all joint angles stored in `q2` by using inverse kinematic function `ikine()`, then update the slider value and call `app.updateRobot()` to plot the new robot configuration. As the `ikine()` function may output unexpected result due to multiple solution. So, we use the previous joint angles as `q0` initial joint angles configuration. And we also added some checking condition for angles reach limits or `ikine()` function has no solution, then a message box will give a warning about it.

```

% Key release function: G8RobotArmUIFigure
function G8RobotArmUIFigureKeyRelease(app, event)
%make sure robot is created already
if app.IsRobotCreated == true
    %Get current robot EE transformation matrix
    q1 = [app.Joint1Slider.Value*pi/180 ...
          app.Joint2Slider.Value*pi/180 ...
          app.Joint3Slider.Value*pi/180];
    T = app.G8Robot.fkine(q1);
    %adjust x or y position according to key press
    if event.Key == "leftarrow"
        T.t(2) = T.t(2) + 0.005; %move to Y + 0.005m
    elseif event.Key == "rightarrow"
        T.t(2) = T.t(2) - 0.005; %move to Y - 0.005m
    elseif event.Key == "uparrow"
        T.t(1) = T.t(1) + 0.005; %move to X + 0.005m
    elseif event.Key == "downarrow"
        T.t(1) = T.t(1) - 0.005; %move to X - 0.005m
    else
        end
    %Do inverse kinematics
    q2 = app.G8Robot.ikine(T,'q0',q1,'mask',[1 1 1 0 0 0]);
    if (isempty(q2) == false)
        %update joints angle after ikine
        q2deg = rad2deg(q2);
        if((q2deg(1) <= app.Joint1Slider.Limits(2)) && ...
           (q2deg(1) >= app.Joint1Slider.Limits(1)) && ...
           (q2deg(2) <= app.Joint2Slider.Limits(2)) && ...
           (q2deg(2) >= app.Joint2Slider.Limits(1)))
            app.Joint1Slider.Value = q2deg(1);
            app.Joint2Slider.Value = q2deg(2);
            app.Joint3Slider.Value = q2deg(3);
            %plot the robot and update position and distance info
            app.updateRobot();
        else
            msg = sprintf("Joint angle reaches limit:\n" + ...
                          "Joint1 angle:%0.1f\n" + ...
                          "Joint2 angle:%0.1f",q2deg(1),q2deg(2));
            msgbox(msg);
        end
    else
        msgbox("can't reach the target");
    end
else
    msgbox("Robot not created yet")
end
end

```

Fig.9 Keyboard Control

In the step 7, we determine 9 poses and measure the distance from end-effector to the new base frame (clipboard frame), also, we calculated respective distance by firstly using the `fkine()` function to get a transform matrix of end-effector and also get the translation part and then use the translation part to calculate the distance to the base frame, which is implemented in the class method `app.updateRobot()`. All the poses, measurements and calculations are listed in the Table. 2. Also, the poses and measurements of the distance are stored in the file `DistanceData.txt`.

	Joint 1 (degree)	Joint 2 (degree)	Joint 3 (degree)	Measured Distance (meters)	Calculated Distance (meters)	Error (meters)
Pose_1	0	0	0	0.212	0.2047	0.0073
Pose_2	0	90	0	0.115	0.1095	0.0055
Pose_3	90	-90	0	0.233	0.2273	0.0057
Pose_4	90	0	0	0.118	0.1119	0.0061
Pose_5	90	90	0	0.087	0.07868	0.00832
Pose_6	90	-90	0	0.136	0.1296	0.0064
Pose_7	-90	0	0	0.264	0.2576	0.0064
Pose_8	-90	90	0	0.175	0.1759	0.0009
Pose_9	-90	-90	0	0.231	0.2292	0.0018
					Average	0.00538

Table 2. Poses and distance

In step 8, we write an objective function shown as Fig.10. As an input of this function, *dh* is a matrix that combined all DH parameters, offsets and base frame translation. Then we use the DH parameters to create our robot and then load the poses and measurements from *DistanceData.txt*. Then we use each pose to calculate a transform matrix of end-effector by using *fkine()* function, and finally calculate the distance from end-effector to the clipboard frame by get the translation part in the transform matrix and apply distance function to get the final calculated distance. Then use the average absolute value of the difference between measurements and calculated distance in different 9 poses to be the error.

In summary, the objective function is to calculate the distance error between measurements and calculations, and then the error could be optimized in the later *fminsearch()* function.

```

function error = CalcDistError(dh)
clear L
deg = pi/180;

%build parameter a
a=[dh(1,1);dh(1,2);dh(1,3)];
%build parameter alpha
alpha=[dh(2,1);dh(2,2);dh(2,3)];
%build parameter d
d=[dh(3,1);dh(3,2);dh(3,3)];
%4th column are parameter alpha
offset=[dh(4,1);dh(4,2);dh(4,3)];
baseX = dh(5,1);
baseY = dh(5,2);
baseZ = dh(5,3);

%define the links of robot
L(1) = Revolute('d', d(1), 'a', a(1), 'alpha', alpha(1)*deg, ...
    'offset',offset(1)*deg );

L(2) = Revolute('d', d(2), 'a', a(2), 'alpha', alpha(2)*deg, ...
    'offset',offset(2)*deg );

L(3) = Revolute('d', d(3), 'a', a(3), 'alpha', alpha(3)*deg, ...
    'offset',offset(3)*deg );

%Create the robot
g8robot = SerialLink(L, 'name', 'G8 Robot Arm','base',transl(baseX,baseY,baseZ));
figure(2)
clf(2)
%Plot the robot
g8robot.plot([0,0,0]);

%Load measured distance data and 3 joint angles from file
MeasData = load ('DistanceData.txt');
%get the joint angles of each pose
theta = deg2rad(MeasData(:,1:3));
%get the measured distance of each pose
MeasDist = MeasData(:,4)';

%go through 9 poses to calculate the distance
for i=1:9
    poseT(i) = g8robot.fkine(theta(i,:));
    CalcDist(i) = sqrt((poseT(i).t(1))^2 + (poseT(i).t(2))^2 + (poseT(i).t(3))^2);
end

%calculate the average error between calculated distance and
%measurement
error = mean(abs(CalcDist - MeasDist))
end

```

Fig.10 Objective Function

```

function Optimize(app, event)
%before start, set flag as false
app.optimized = false;

%read preset/measured DH parameters to DH variables
a1 = app.a1EditField.Value;
a2 = app.a2EditField.Value;
a3 = app.a3EditField.Value;
alpha1 = app.alpha1EditField.Value;
alpha2 = app.alpha2EditField.Value;
alpha3 = app.alpha3EditField.Value;
d1 = app.d1EditField.Value;
d2 = app.d2EditField.Value;
d3 = app.d3EditField.Value;
offset1 = app.offset1EditField.Value;
offset2 = app.offset2EditField.Value;
offset3 = app.offset3EditField.Value;
baseX = app.xBaseTransl.Value;
baseY = app.yBaseTransl.Value;
baseZ = app.zBaseTransl.Value;

%Assemble DH parameters for optimization
MeasDH = [a1 a2 a3;alpha1 alpha2 alpha3;d1 d2 d3;
          offset1 offset2 offset3;baseX baseY baseZ];
%set optimization options
options = optimset('Display','final','TolX',1e-6, ...
                  'PlotFcns',@optimplotfval,'MaxIter',500);
%start optimization
[x,fval,exitflag,output] = fminsearch(@CalcDistError,MeasDH, options)

app.optimized = true;
%optimization finished, fill the result to GUI
app.errorEditField.Value = fval;
app.A1EditField.Value = x(1,1);
app.A2EditField.Value = x(1,2);
app.A3EditField.Value = x(1,3);
app.ALPHA1EditField.Value = x(2,1);
app.ALPHA2EditField.Value = x(2,2);
app.ALPHA3EditField.Value = x(2,3);
app.D1EditField.Value = x(3,1);
app.D2EditField.Value = x(3,2);
app.D3EditField.Value = x(3,3);
app.OFFSET1EditField.Value = x(4,1);
app.OFFSET2EditField.Value = x(4,2);
app.OFFSET3EditField.Value = x(4,3);
app.baseXEditField.Value = x(5,1);
app.baseYEditField.Value = x(5,2);
app.baseZEditField.Value = x(5,3);
end

```

Fig. 11 Optimization

In the callback function of “Optimize” button, we perform Optimize () function which uses fminsearch() function as its core function, of which the codes are shown in the Fig. 11. The input parameters of fminsearch() function are the objective function (CalcDistError()), a matrix of DH parameters (MeasDH), options. While the objective function represents what you want to optimize, DH parameters are what can affect objective function and can be tuned. And options are about some setting of plotting requirements, tolerated error range (1e-6), and the number of iterations (500 iterations). The output contains improved parameters matrix “x” and final error value “fval”.

IV. Results

In this section, we will explain our designed App, showcasing its functionality and show the results of optimization. This project has three main files:

- G8RobotArm.mlapp
- CalcDistError.m
- DistanceData.txt

The relationship among them is that .mlapp file is our designed App, so it could be regarded as the body of the project, .m file is used as the objective function in the fminsearch() function, while the .txt file contains 9 poses and corresponding distance measurements from end-effector to the new base frame, which is to be used in the CalcDistError() function.

The below shows our GUI window,

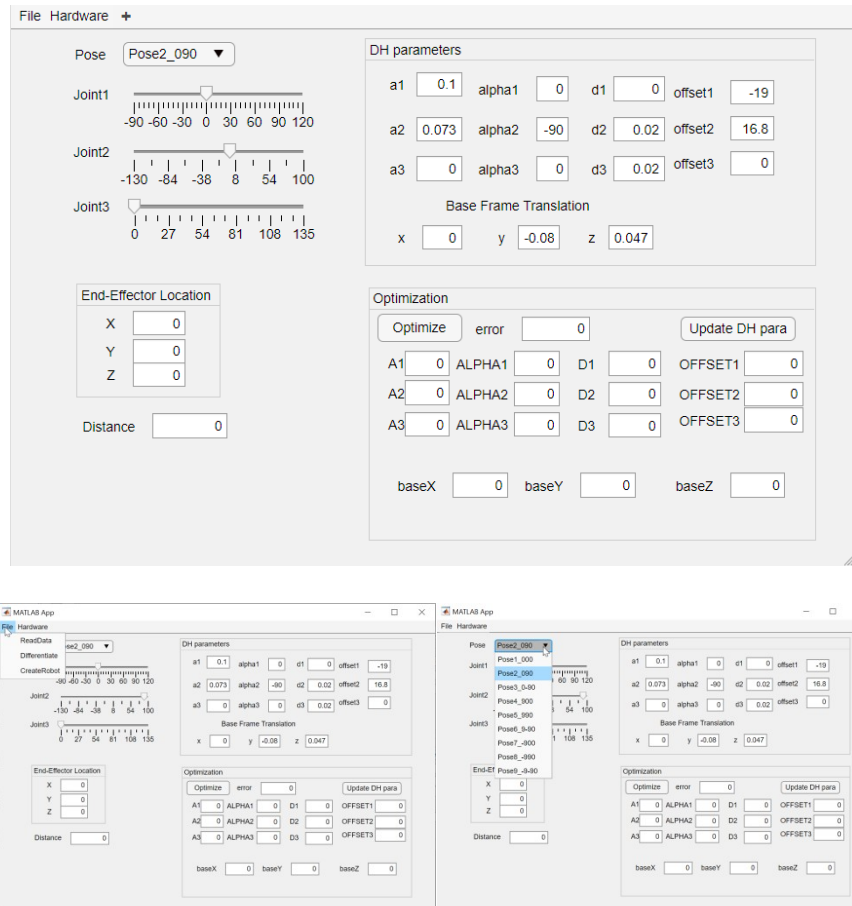


Fig. 12 GUI window.

And you can edit DH parameters, base frame translation parameters in it. Also, you can choose a pose and then press the “CreatRobot” button to create a robot. By tune the slider position, the robot can be re-plotted, which is showcased in Fig. 8. Then end-effector location and distance to the new base frame are also presented in the left side of the GUI window.

Next, we will showcase how optimization works in the GUI.

By pressing the “Optimize” button, the optimization of DH will be performed. Then the robot will be updated in the plot window, and the error will be improved in another window.

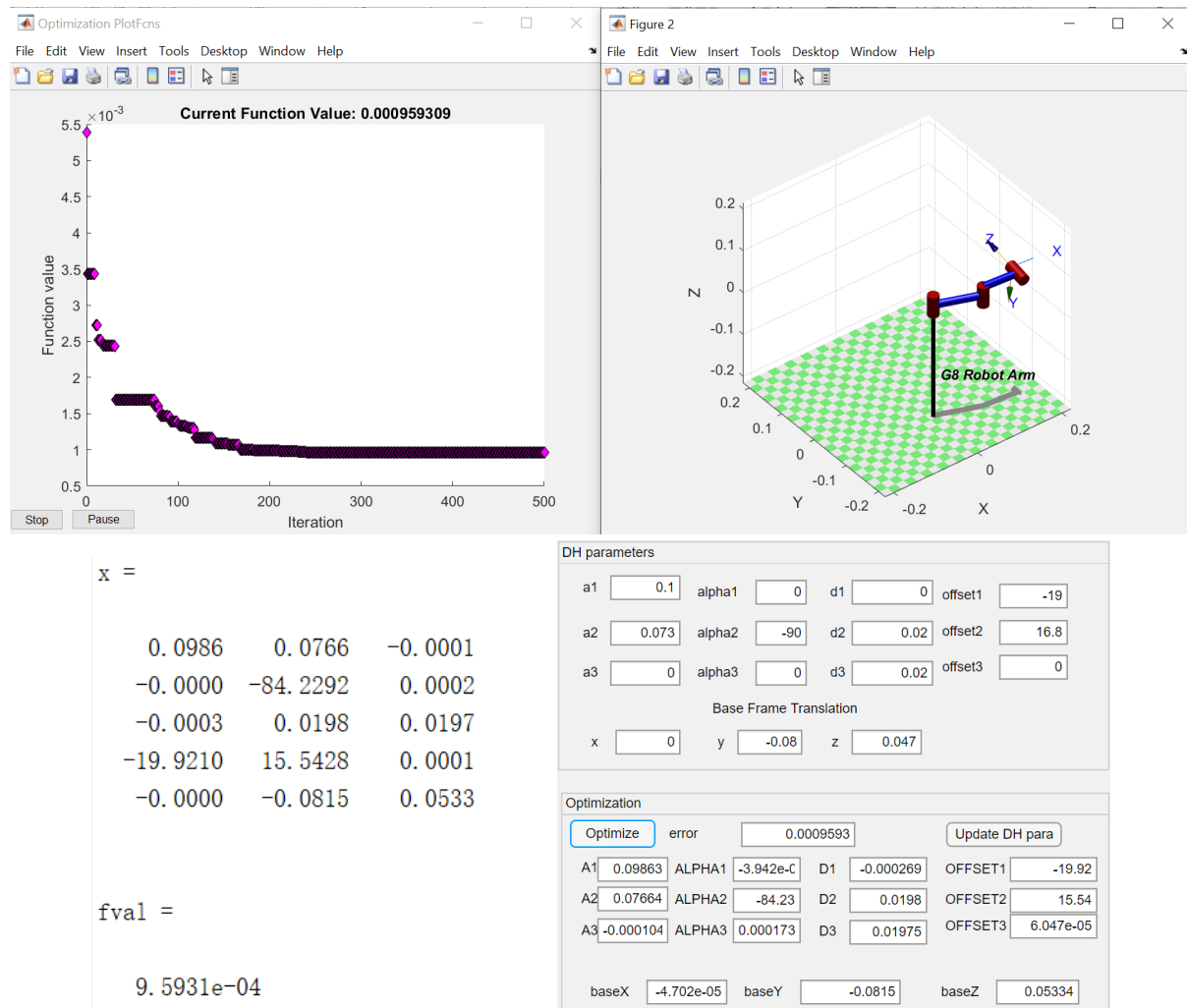


Fig. 13 Optimization window

Shown as Fig.13, after each optimization iteration, the DH parameters will be updated, and the robot will be re-plotted in the plot window. After 500 iterations, the error is reduced from 0.0054 to 0.00096, which is improved significantly.

Then we can fill optimized parameters into upper DH parameters and recalculate the distance, compare it with our measured distance, and as you can see in the Table. 3 below, the errors between measured and calculated distance are cut down obviously.


	Joint1	Joint2	Joint3	Measured	Before Optimization Calculated	error	After Optimization Calculated	error
Pose1_000	0	0	0	0.212	0.2047	0.0073	0.212	0
Pose2_090	0	90	0	0.115	0.1095	0.0055	0.115	0
Pose3_0-90	0	-90	0	0.233	0.2273	0.0057	0.2326	0.0004
Pose4_900	90	0	0	0.118	0.1119	0.0061	0.118	0
Pose5_990	90	90	0	0.087	0.07868	0.00832	0.087	0
Pose6_9-90	90	-90	0	0.136	0.1296	0.0064	0.1362	0.0002
Pose7_-900	-90	0	0	0.264	0.2576	0.0064	0.264	0
Pose8_-990	-90	90	0	0.175	0.1759	0.0009	0.1828	0.0078
Pose9_-9-90	-90	-90	0	0.231	0.2292	0.0018	0.2313	0.0003
					Avg error	0.00538		0.00096667

Table .3 DH parameters updated and corresponding error comparisons

V. Conclusion

In this section, we are summarizing our progress and experiment results. In this study, we build a real hardware robot arm with its simulation model in Matlab App Designer, which enables users to create a robot arm model simulate the hardware robot arm and tune measured DH parameters to improve the robot arm accuracy as well as performance. After the optimization of DH parameters, we get considerably smaller error values of measured distance and calculated distance in 9 poses shown as Table. 3, which demonstrate the effectiveness and accuracy of our model-based simulation and optimization.

VI. Discussion

In this section, we are going to share our experience from this study and present the promise of the future study. During the work, we learned robotics knowledge from various perspectives and enhance the understanding of the knowledge in the hands-on learning process. Meanwhile, we improved our Matlab programming skills and depend our intuitive reorganization of robotic kinematics, which benefits a lot to our course study. Moreover, we find some interesting problems to be solved, such as how to obtain an accurate reachable area and what is the reasoning inside of the `ikine()` function.

In the close future, robotics will significantly improve the productivity and efficiency of human society. Thus, as engineers, we are expected to learn more about robotics, and exploit more method paths to solve physical problems. For instance, how to apply Reinforcement Learning to make robots more intelligent that can learn how to do a task by itself. What is more, how to make robots user-friendly in the human-machine interface, which will deeply impact the universal usability of robots.

Finally, many thanks to the Professor Pandya, who provided experiment toolkit and helped us very patiently.

VII. Appendix

1. Video Link

Robot arm setup: https://www.youtube.com/watch?v=W543S_3lmbM

Robot arm GUI: <https://www.youtube.com/watch?v=ctb90lQvoHY>

2. Code

Github: <https://github.com/linliawxm/RobotArm>

G8 Robot Arm app main functions:

```
properties (Access = private)
    DataRead = false; % Initialize as false
    Data = []; %buffer for reading data
    DiffData = [];
    arduino_hardware = [];
    arduino_initialized = true;
    G8Robot = [];
    IsRobotCreated = false;
    optimized = false;

    % some useful poses
    Pose = [0 0 0; %Pose1
            0 90 0; %Pose2
            0 -90 0; %Pose3
            90 0 0; %Pose4
            90 90 0; %Pose5
            90 -90 0; %Pose6
            -90 0 0; %Pose7
            -90 90 0; %Pose8
            -90 -90 0]; %Pose9

End

methods (Access = private)
    function results = updateRobot(app)
        if app.IsRobotCreated == true
            %Re-plot robot for new pose
            app.G8Robot.plot([app.Joint1Slider.Value*pi/180 ...
                               app.Joint2Slider.Value*pi/180 ...
                               app.Joint3Slider.Value*pi/180], "jvec");
            %Recalculate the Transformation matrix after joint angle
```

```

        %changed
        T = app.G8Robot.fkine([app.Joint1Slider.Value*pi/180 ...
        app.Joint2Slider.Value*pi/180 ...
        app.Joint3Slider.Value*pi/180]);
        %Update the EE position on GUI
        app.XEditField.Value = T.t(1);
        app.YEditField.Value = T.t(2);
        app.ZEditField.Value = T.t(3);
        %Calculate the distance of EE and update on GUI
        app.DistanceEditField.Value = sqrt(T.t(1)^2 + T.t(2)^2 + T.t(3)^2);
    else
        msgbox("Robot not created yet")
    end
end
end

End

% Code that executes after component creation
function startupFcn(app)
    %Set initial joints angle according to pose selection
    poseInx = str2num(app.PoseDropDown.Value);
    app.Joint1Slider.Value = app.Pose(poseInx,1);
    app.Joint2Slider.Value = app.Pose(poseInx,2);
    app.Joint3Slider.Value = app.Pose(poseInx,3);
end

% Menu selected function: InitArduinoMenu
function Initialize(app, event)
    try
        app.arduino_hardware = arduino();
    catch
        msgbox('error on hardware initialize');
        return;
    end
    app.arduino_initialized = true;
    msgbox('Arduino initialized');
end

% Menu selected function: TestArduinoMenu
function TestArduino(app, event)
    if(app.arduino_initialized)
        app.arduino_hardware.writeDigitalPin("D13",1);
    end
end

```

```
end
```

```
% Menu selected function: ReadDataMenu
```

```
function ReadData(app, event)

    [fn pn] = uigetfile('*.txt', 'Where is the data?');
    filename = [pn fn];
    app.Data = textread(filename);
    plot(app.UIAxes, app.Data(:,1),app.Data(:,2));
    app.DataRead = true;
    app.DataReadLamp.Color = [0.0 1.0 0.0];
    app.InfoTextArea.Value = 'Data has been read.';
end
```

```
% Menu selected function: DifferentiateMenu
```

```
function Differentiate(app, event)

    if(app.DataRead)
        app.DiffData = app.Data;
        app.DiffData(1:999,2) = diff(app.Data(:,2));
        hold(app.UIAxes, 'on');
        plot(app.UIAxes, app.DiffData(:,1),app.DiffData(:,2),'g');
        app.InfoTextArea.Value = 'Green is the differential';
    else
        app.InfoTextArea.Value = 'Read the data first';
    end
end
```

```
% Menu selected function: CreateRobotMenu
```

```
function CreateRobot(app, event)

    clear L
    deg = pi/180;
    %get the joint angles limits from slider.Limits
    qlim1 = deg2rad(app.Joint1Slider.Limits);
    qlim2 = deg2rad(app.Joint2Slider.Limits);
    qlim3 = deg2rad(app.Joint3Slider.Limits);
    % Define links of robot
    L(1) = Revolute('d', app.d1EditField.Value, ... % link length (Dennavit-Hartenberg notation)
        'a', app.a1EditField.Value, ... % link offset (Dennavit-Hartenberg notation)
        'alpha', app.alpha1EditField.Value*deg, ... % link twist (Dennavit-Hartenberg notation)
        'qlim', qlim1, ... % minimum and maximum joint angle
        'offset',app.offset1EditField.Value*deg );
    L(2) = Revolute('d', app.d2EditField.Value, ...
```

```

        'a', app.a2EditField.Value, ...
        'alpha', app.alpha2EditField.Value*deg, ...
        'qlim', qlim2, ...
        'offset',app.offset2EditField.Value*deg );
L(3) = Revolute('d', app.d3EditField.Value, ...
        'a', app.a3EditField.Value, ...
        'alpha', app.alpha3EditField.Value*deg, ...
        'qlim', qlim3, ...
        'offset',app.offset3EditField.Value*deg );

%Build the robot
app.G8Robot = SerialLink(L, 'name', 'G8 Robot Arm', ...
        'tool',transl(0,0,0), ...
        'base',transl(app.xBaseTransl.Value, ...
        app.yBaseTransl.Value, ...
        app.zBaseTransl.Value));

%Set the robot created flag
app.IsRobotCreated = true;
%plot the robot and update postion and distance info
app.updateRobot();
end

% Value changed function: a1EditField
function Update_a1(app, event)
    if app.IsRobotCreated == true
        %clear current figure before re-plot
        clf;
        %update robot a1 parameter
        app.G8Robot.links(1).a = app.a1EditField.Value;
        %plot the robot and update postion and distance info
        app.updateRobot();
    end
end

% Key release function: G8RobotArmUIFigure
function G8RobotArmUIFigureKeyRelease(app, event)
    %make sure robot is created already
    if app.IsRobotCreated == true
        %Get current robot EE transformation matrix
        q1 = [app.Joint1Slider.Value*pi/180 ...
            app.Joint2Slider.Value*pi/180 ...
            app.Joint3Slider.Value*pi/180];
        T = app.G8Robot.fkine(q1);
    end
end

```



```

%adjust x or y position according to key press
if event.Key == "leftarrow"
    T.t(2) = T.t(2) + 0.005; %move to Y + 0.005m
elseif event.Key == "rightarrow"
    T.t(2) = T.t(2) - 0.005; %move to Y - 0.005m
elseif event.Key == "uparrow"
    T.t(1) = T.t(1) + 0.005; %move to X + 0.005m
elseif event.Key == "downarrow"
    T.t(1) = T.t(1) - 0.005; %move to X - 0.005m
else
end

%Do inverse kinematics
q2 = app.G8Robot.ikine(T,'q0',q1,'mask',[1 1 1 0 0 0]);
if (isempty(q2) == false)
    %update joints angle after ikine
    q2deg = rad2deg(q2);
    if((q2deg(1) <= app.Joint1Slider.Limits(2)) && ...
        (q2deg(1) >= app.Joint1Slider.Limits(1)) && ...
        (q2deg(2) <= app.Joint2Slider.Limits(2)) && ...
        (q2deg(2) >= app.Joint2Slider.Limits(1)))
        app.Joint1Slider.Value = q2deg(1);
        app.Joint2Slider.Value = q2deg(2);
        app.Joint3Slider.Value = q2deg(3);
        %plot the robot and update postion and distance info
        app.updateRobot();
    else
        msg = sprintf("Joint angle reaches limit:\n" + ...
            "Joint1 angle:%0.1f\n" + ...
            "Joint2 angle:%0.1f",q2deg(1),q2deg(2));
        msgbox(msg);
    end
else
    msgbox("can't reach the target");
end

else
    msgbox("Robot not created yet")
end

end

% Value changed function: PoseDropDown
function ChangePose(app, event)
    %make sure robot is created already
    if app.IsRobotCreated == true

```

```

        %update joints angle according to pose selection
        poseinx = str2num(app.PoseDropDown.Value);
        app.Joint1Slider.Value = app.Pose(poseinx,1);
        app.Joint2Slider.Value = app.Pose(poseinx,2);
        app.Joint3Slider.Value = app.Pose(poseinx,3);
        %plot the robot and update postion and distance info
        app.updateRobot();
    else
        msgbox("Robot not created yet")
    end
end

```

```

% Value changed function: Joint1Slider
function Joint1SliderValueChanged(app, event)
    %plot the robot and update postion and distance info
    app.updateRobot();
end

```

```

% Value changed function: Joint2Slider
function Joint2SliderValueChanged(app, event)
    %plot the robot and update postion and distance info
    app.updateRobot();
end

```

```

% Value changed function: Joint3Slider
function Joint3SliderValueChanged(app, event)
    %plot the robot and update postion and distance info
    app.updateRobot();
end

```

```

% Button pushed function: OptimizeButton
function Optimize(app, event)
    %before start, set flag as flase
    app.optimized = false;
    %read preset/measured DH parameters to DH variables
    a1 = app.a1EditField.Value;
    a2 = app.a2EditField.Value;
    a3 = app.a3EditField.Value;
    alpha1 = app.alpha1EditField.Value;
    alpha2 = app.alpha2EditField.Value;

```

```

alpha3 = app.alpha3EditField.Value;
d1 = app.d1EditField.Value;
d2 = app.d2EditField.Value;
d3 = app.d3EditField.Value;
offset1 = app.offset1EditField.Value;
offset2 = app.offset2EditField.Value;
offset3 = app.offset3EditField.Value;
baseX = app.xBaseTransl.Value;
baseY = app.yBaseTransl.Value;
baseZ = app.zBaseTransl.Value;

%Assemble DH parameters for optimization
MeasDH = [a1 a2 a3;alpha1 alpha2 alpha3;d1 d2 d3;
offset1 offset2 offset3;baseX baseY baseZ];

%set optimization options
options = optimset('Display','final','TolX',1e-6, ...
'PlotFcns',@optimplotfval,'MaxIter',500);

%start optimization
[x,fval,exitflag,output] = fminsearch(@CalcDistError,MeasDH, options)
app.optimized = true;

%optimization finished, fill the result to GUI
app.errorEditField.Value = fval;
app.A1EditField.Value = x(1,1);
app.A2EditField.Value = x(1,2);
app.A3EditField.Value = x(1,3);
app.ALPHA1EditField.Value = x(2,1);
app.ALPHA2EditField.Value = x(2,2);
app.ALPHA3EditField.Value = x(2,3);
app.D1EditField.Value = x(3,1);
app.D2EditField.Value = x(3,2);
app.D3EditField.Value = x(3,3);
app.OFFSET1EditField.Value = x(4,1);
app.OFFSET2EditField.Value = x(4,2);
app.OFFSET3EditField.Value = x(4,3);
app.baseXEditField.Value = x(5,1);
app.baseYEditField.Value = x(5,2);
app.baseZEditField.Value = x(5,3);
end

```

```

% Value changed function: a2EditField
function update_a2(app, event)
    if app.IsRobotCreated == true
        %clear current figure before re-plot
        clf;
    end
end

```

```

        %update robot a1 parameter
        app.G8Robot.links(2).a = app.a2EditField.Value;
        %plot the robot and update postion and distance info
        app.updateRobot();
    end
end

% Value changed function: d1EditField
function update_d1(app, event)
    if app.IsRobotCreated == true
        %clear current figure before re-plot
        clf;
        %update robot a1 parameter
        app.G8Robot.links(1).d = app.d1EditField.Value;
        %plot the robot and update postion and distance info
        app.updateRobot();
    end
end

% Value changed function: d2EditField
function update_d2(app, event)
    if app.IsRobotCreated == true
        %clear current figure before re-plot
        clf;
        %update robot a1 parameter
        app.G8Robot.links(2).d = app.d2EditField.Value;
        %plot the robot and update postion and distance info
        app.updateRobot();
    end
end

% Value changed function: d3EditField
function update_d3(app, event)
    if app.IsRobotCreated == true
        %clear current figure before re-plot
        clf;
        %update robot a1 parameter
        app.G8Robot.links(3).d = app.d3EditField.Value;
        %plot the robot and update postion and distance info
        app.updateRobot();
    end
end

```

end

% Value changed function: alpha2EditField

function update_alpha2(app, event)

if app.IsRobotCreated == true

%clear current figure before re-plot

clf;

%update robot a1 parameter

app.G8Robot.links(2).alpha = deg2rad(app.alpha2EditField.Value);

%plot the robot and update postion and distance info

app.updateRobot();

end

end

% Value changed function: offset1EditField

function update_offset1(app, event)

if app.IsRobotCreated == true

%clear current figure before re-plot

%clf;

%update robot a1 parameter

app.G8Robot.offset(1) = app.offset1EditField.Value*pi/180;

%plot the robot and update postion and distance info

app.updateRobot();

end

end

% Value changed function: offset2EditField

function update_offset2(app, event)

if app.IsRobotCreated == true

%clear current figure before re-plot

%clf;

%update robot a1 parameter

app.G8Robot.offset(2) = app.offset2EditField.Value*pi/180;

%plot the robot and update postion and distance info

app.updateRobot();

end

end

% Button pushed function: UpdateDHparaButton

function UpdateDHPara(app, event)

```

    if app.optimized
        app.a1EditField.Value = app.A1EditField.Value;
        app.a2EditField.Value = app.A2EditField.Value;
        app.a3EditField.Value = app.A3EditField.Value;
        app.alpha1EditField.Value = app.ALPHA1EditField.Value;
        app.alpha2EditField.Value = app.ALPHA2EditField.Value;
        app.alpha3EditField.Value = app.ALPHA3EditField.Value;
        app.d1EditField.Value = app.D1EditField.Value;
        app.d2EditField.Value = app.D2EditField.Value;
        app.d3EditField.Value = app.D3EditField.Value;
        app.offset1EditField.Value = app.OFFSET1EditField.Value;
        app.offset2EditField.Value = app.OFFSET2EditField.Value;
        app.offset3EditField.Value = app.OFFSET3EditField.Value;
        app.xBaseTransl.Value = app.baseXEditField.Value;
        app.yBaseTransl.Value = app.baseYEditField.Value;
        app.zBaseTransl.Value = app.baseZEditField.Value;
    end
end

% Value changed function: alpha1EditField
function update_alpha1(app, event)
    if app.IsRobotCreated == true
        %clear current figure before re-plot
        %clf;
        %update robot a1 parameter
        app.G8Robot.links(1).alpha = deg2rad(app.alpha1EditField.Value);
        %plot the robot and update postion and distance info
        app.updateRobot();
    end
end
end

```

Objective function:

```

%Objective function to calculate the end effector distance from the
%clipboard coordinate system origin
%Input parameter:
% DH parameters of 3 links robot
% dh = [a1 a2 a3;
%       alpha1 alpha2 alpha3;
%       d1 d2 d3;

```



```

%      offset1 offset2 offset3;
%      baseX baseY baseZ]
function error = CalcDistError(dh)

    clear L
    deg = pi/180;

    %build parameter a
    a=[dh(1,1);dh(1,2);dh(1,3)];
    %build parameter alpha
    alpha=[dh(2,1);dh(2,2);dh(2,3)];
    %build parameter d
    d=[dh(3,1);dh(3,2);dh(3,3)];
    %4th column are parameter alpha
    offset=[dh(4,1);dh(4,2);dh(4,3)];
    baseX = dh(5,1);
    baseY = dh(5,2);
    baseZ = dh(5,3);

    %define the links of robot
    L(1) = Revolute('d', d(1), 'a', a(1), 'alpha', alpha(1)*deg, ...
        'offset',offset(1)*deg );

    L(2) = Revolute('d', d(2), 'a', a(2), 'alpha', alpha(2)*deg, ...
        'offset',offset(2)*deg );

    L(3) = Revolute('d', d(3), 'a', a(3), 'alpha', alpha(3)*deg, ...
        'offset',offset(3)*deg );

    %Create the robot
    g8robot = SerialLink(L, 'name', 'G8 Robot Arm','base',transl(baseX,baseY,baseZ));
    %figure(2)
    %clf(2)
    %Plot the robot
    %g8robot.plot([0,0,0]);

    %Load measured distance data and 3 joint angles from file
    MeasData = load ('DistanceData.txt');
    %get the joint angles of each pose
    theta = deg2rad(MeasData(:,1:3));
    %get the measured distance of each pose
    MeasDist = MeasData(:,4)';

    %go through 9 poses to calculate the distance
    for i=1:9

```

```
poseT(i) = g8robot.fkine(theta(i,:));  
%T.t = [x;y;z]  
CalcDist(i) = sqrt((poseT(i).t(1))^2 + (poseT(i).t(2))^2 + (poseT(i).t(3))^2);  
end  
%calculate the avearge error between calculated distance and  
%measurement  
error = mean(abs(CalcDist - MeasDist))  
end
```