Predictive Model Development For Welding Mini-robot

[1]Oladebeye D. H, [2]Adejuyigbe S. B., [3]Kareem B
[1]Professor, [2]Professor, [3]Professor
[1]Department of Mechanical Engineering Technology, Federal Polytechnic, Ado-Ekiti, Ekiti State, Nigeria,
[2]Mechatronics Engineering Department, Federal University, Oye-Ekiti, Ekiti State, Nigeria,
[3]Industrial and Production Engineering Department, Federal University of Technology, Akure, Ondo State, Nigeria

Abstract **- In this analysis, MATLAB software and Response Surface Methodology (RSM) were used to optimize the welding parameters, namely voltage, current, material thickness and arc length. A predictive model was subsequently obtained using a modified heat input function to help assess and then predict the welding time and welding speed of the established welding robot in relation to the welding parameters adopted. Results have shown that, at welding time (4.7-32.94s), the built welding robot can weld mild steel plates linearly along the length of the guide 470 mm on X-axis, 350 mm on Y-axis and 110 mm on Z-axis. Welding speed range for 0.5 mm thick plate to 1.0 mm mild steel plate is 4.409-5.32 mms-1 while electric arc (manual) welding speed range is 2.330-5.500 mms-1. The welding time ranges from 4.7s to 32.94s for 0.5 mm thick plate to 1.0 mm mild steel plate welded with modeed built welding robot while the welding time ranges from 15s to 45s for 0.5 mm thick plate to 1.0 mm mild steel plate welded with electric arc welding (handbook). The results showed that the predictive model built for welding mini-robot would reduce the cost of production and increase the quality as well as the efficiency of welding during welding process compared to manual electric arc welding. For the same steel plate specimen, the weld length in the established robot welding (Automated) is obviously greater than in the electric arc welding (Manual). Welding speed increases from all signs, on average as the thickness of the steel plate increases. The welding speeds of the built welding robot, however, are much greater than those of the electric arc welding device. It is a benefit which should be of interest to the Nigerian manufacturing industry in the areas of precision machining, mass production of products, reduction of machining time which production costs, improved efficiency, income maximization.**

keywords **- Predictive, model, development, welding, mini-robot**

I. Introduction

What makes robotics so fascinating is that it is a science of ingenious machines, precision-built, powered by a permanent power source, and versatile from a programming perspective. This does not simply mean open source, but rather the use of powerful APIs, and de facto standards for both hardware and software, enabling unrestricted access to device capacity. This is particularly important in research environments, where good resource access is required in order to introduce and test new ideas. If that is usable, then an integrator device (or even a researcher) does not need open source software, at least for the conventional robotics fields (industrial robot manipulators and mobile robots). In fact, that could also be very hard to achieve as these robotics fields have decades of engineering effort, achieving very good results and reliable machines that are not easy to match. That open source problem is nevertheless very relevant as a way to spread and accelerate development for emerging robotics research (such as humanoid robotics, space robotics, robots for medical use, etc.)

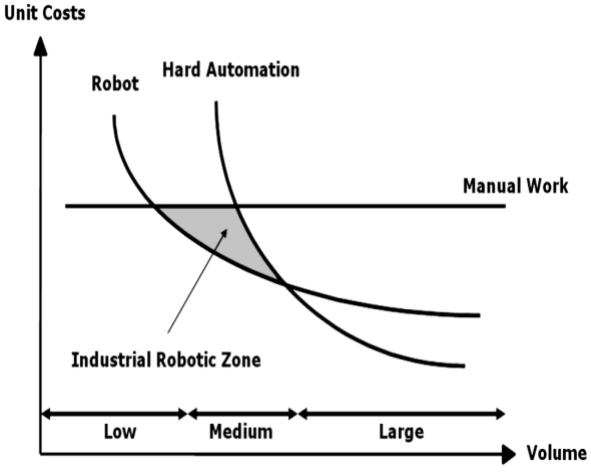

Figure 1. Industrial robot zone

Industrial Robotic Welding is by far the most common robotics application in the world [1]. In reality, their assembly processes require a huge number of products which require welding operations. Perhaps the most important example is the automotive industry, with the spot and MIG / MAG welding operations in assembly line car body workshops. Arc welding can be used

during assembly operation to weld the components of railcar brackets [2]. Nonetheless, there are growing number of smaller companies, consumer-oriented, manufacturing small series or exclusive goods tailored for each consumer. Such users need a good and highly automated welding process to respond in a timely and high quality manner to client needs. The Agile Design methods by Kusiak [3] and Kusiak [4] apply the most to these firms, obviously backed by flexible fabrication setups (Figure 1). For all that interest, industrial robotic welding has evolved slightly and is far from being, at least in a general way, a solved technical operation. The welding process is dynamic, hard to parameterise and to track and control effectively [5-9]. In addition, most of the welding techniques, including the effects on the welding joints, are not fully known, and are used based on observational models obtained under particular conditions through practice. At the moment the results of the welding process on the welded surfaces are not completely understood. Welding can in most cases (i.e. welding with MIG / MAG) place extremely high intense temperatures in small areas. Physically, this allows the material undergo extremely high and localized thermal expansion and contraction cycles, which causes material changes that can influence its mechanical behavior along with plastic deformation [10-12].

Such improvements need to be well documented for mitigating the consequences. It is not easy to use welding robots, and has been the focus of numerous R&D efforts [13-17]. And that's because a large range of items are created by the industrial world using welding to assemble some of their parts (Figure 2). If the percentage of welding connections in the product is sufficiently high then some kind of automation should be used to perform the welding operation. This will lead to cheaper goods, as efficiency and quality can be improved, and manpower and manufacturing costs can be reduced. Nevertheless the problems increase in number and complexity when a robot is connected to a welding system. Robots are still difficult to use and program by everyday operators, have restricted remote installations and programming environments, and are managed with closed systems and restricted software interfaces [18-22]. In view of some of the reasons mentioned above, this work is set out to develop a model based on a modified heat input equation and with the aid of a MATLAB program to predict the welding time and speed for the developed Cartesian welding robot operations.

II. PREDICTIVE MODEL DEVELOPMENT FOR THE WELDING ROBOT

This model was developed based on a modified heat input equation and with aid of a MATLAB program to predict the welding speed and welding time for the developed welding robot operations, these two parameters being very fundamental to the performance of the developed welding robot as bases for comparison with the conventional manual electric arc welding technique. The total heat input (Q) transferred to the weldness per unit time and per unit length of the weld is given by Singh [23] as in Equation (1) and leads to Equations 2 – 4. The welding time is given as Equation (6).

$$Q = \eta \frac{IV}{S} \quad (1)$$

$$Q = \eta \frac{60\,IV}{S} \quad (2)$$

$$\text{hence, } S = \frac{L}{t} \quad (3)$$

$$Q = \eta \frac{60\,IVt}{L} \quad (4)$$

$$t = \frac{Q \times L}{60\,\eta IV} \quad (6)$$

I is the current in Ampere (A)
V is the voltage in Voltage (V)
S is the welding speed in mm/s
η is the heat transfer efficiency
L is length of weld in mm
t is the welding time or time required to perform the welding operation in second (s)

This model interface using MATLAB Programme (See Appendix I) was a product of modification of the heat input equation and it was used to predict the welding speed and welding time for the developed welding robot operation as shown in Figure 1.

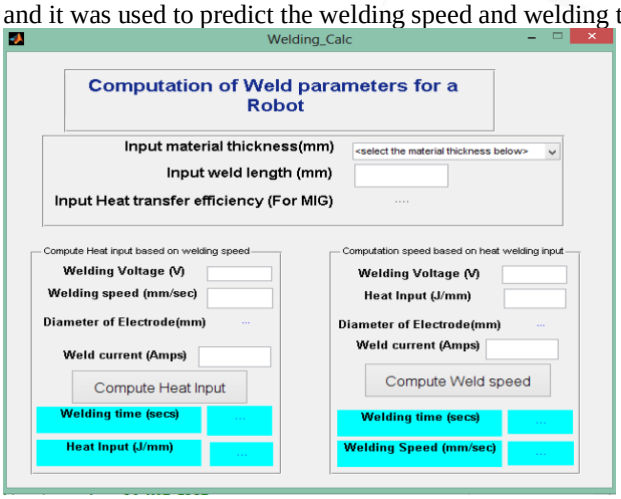

Figure 1: Welding Model for the Developed Robot. **Figure 2:** Developed Welding Robot Operation Interface

III. Modeling of the Developed Welding Robot Operation Interface

This is the Graphical user interface for the input of parameters, such as length of weld in the X-axis, weld position in the Y-axis and the positioning of the welding tong in the Z-axis, all to control the operation of the welding robot developed as shown in Figure 2. A MATLAB programme was used for the modeling.

IV. Simulation of the Developed Welding Robot

This was used to show the various positions in the X, Y and Z axes that the welding robot developed could be moved to perform the flux MIG welding operations as shown in Figure 3. A MATLAB programme (See Appendix II) was used for the simulation.

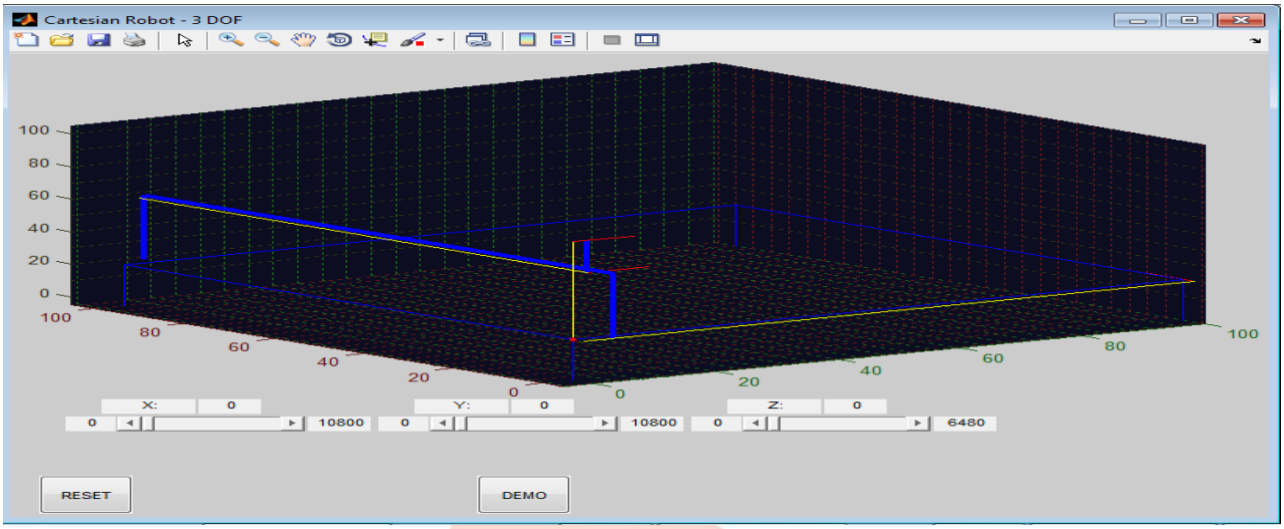

Figure 3: Simulation of the Developed welding Robot

V. WELDED MILD STEEL PLATES FOR TEST USING THE MODELED DEVELOPED WELDING ROBOT

Table 1 shows the various times of weld and welding speeds at some set lengths of weld for the welded mild plates of different sizes used as test specimens for the quality of weld of the developed welding robot.

Table 1: Time, Length of Weld and Welding Speed of Developed Welding Robot

Welding Operation using the Developed Welding Robot						
Length of Weld (mm)	25	50	75	100	125	150
0.5 mm Mild Steel Plate						
Time of Weld (s)	5.37	10.94	17.01	21.82	26.53	32.94
Welding Speed (mm/s)	4.66	4.57	4.41	4.58	4.71	4.55
0.6 mm Mild Steel Plate						
Time of Weld (s)	5.25	10.89	16.99	21.74	26.45	32.82
Welding Speed (mm/s)	4.76	4.59	4.41	4.60	4.73	4.57
0.7 mm Mild Steel Plate						
Time of Weld (s)	5.12	10.77	16.92	21.69	26.33	32.44
Welding Speed (mm/s)	4.88	4.64	4.43	4.61	4.75	4.62
0.8 mm Mild Steel Plate						
Time of Weld (s)	5	10.66	16.84	21.64	26.3	32.43
Welding Speed (mm/s)	5.00	4.69	4.45	4.62	4.75	4.63
0.9 mm Mild Steel Plate						
Time of Weld (s)	4.88	10.58	16.75	21.42	26.02	32.42
Welding Speed (mm/s)	5.12	4.73	4.48	4.67	4.80	4.63
1.0 mm Mild Steel Plate						
Time of Weld (s)	4.7	10.13	16.51	21.3	25.74	31.52
Welding Speed (mm/s)	5.32	4.94	4.54	4.69	4.86	4.76

Table 1 indicates, for developed welding robot, welding speed range for 0.5 mm thick plate, (4.409-4.655) mms^{-1}; 0.6 mm mild steel plate (4.414-4.762) mms^{-1}; 0.7 mm plate, (4.433-4.883) mms^{-1} ; 0.8 mm plate (4.454-5.000) mms^{-1}; 0.9 mm mild steel plate, (4.478-5.123) mms^{-1} and for 1.0 mm mild steel plate, (4.543-5.319) mms^{-1}. Table 1 also indicates, for developed welding robot, welding time range for 0.5 mm thick plate, (5.37-32.94) s; 0.6 mm mild steel plate (5.25-32.82) s; 0.7 mm plate, (5.12-32.44) s; 0.8 mm plate (5-32.43) s; 0.9 mm mild steel plate, (4.88-32.42) s and for 1.0 mm mild steel plate, (4.7-31.52) s.

VI. WELDED MILD STEEL PLATES FOR TEST USING ELECTRIC ARC WELDING (MANUAL)

Table 2 shows the various lengths of weld and welding speeds at some set time of weld for welded mild plates of different sizes used as test specimens for the electric arc welding (Manual).

Table 2: Time, Length of Weld and Welding Speed of Electric Arc Welding Machine

Welding Operation using Electric Arc Welding							
Time of Weld (s)	15	20	25	30	35	40	45
	0.5 mm Mild Steel Plate						
Length of Weld (mm)	35	45	56	75	88	101	107
Welding Speed (mm/s)	2.33	2.25	2.24	2.50	2.51	2.53	2.38
	0.6 mm Mild Steel Plate						
Length of Weld (mm)	45	53	61	78	92	107	135
Welding Speed (mm/s)	3.00	2.65	2.44	2.60	2.63	2.68	3.00
	0.7 mm Mild Steel Plate						
Length of Weld (mm)	46	56	67	95	106	117	148
Welding Speed (mm/s)	3.07	2.80	2.68	3.17	3.03	2.93	3.29
	0.8 mm Mild Steel Plate						
Length of Weld (mm)	62	88	114	123	144	165	174
Welding Speed (mm/s)	4.13	4.40	4.56	4.10	4.11	4.13	3.87
	0.9 mm Mild Steel Plate						
Length of Weld (mm)	65	97	129	158	175	184	192
Welding Speed (mm/s)	4.33	4.85	5.16	5.27	5.00	4.60	4.27
	1.0 mm Mild Steel Plate						
Length of Weld (mm)	70	102	134	165	180	187	195
Welding Speed (mm/s)	4.67	5.10	5.36	5.50	5.14	4.68	4.33

Table 2 indicates, for electric arc welding, welding speed range of (2.330-2.517) mms^{-1} for 0.5 mm mild steel; (2.440-3.000) mms^{-1} for 0.6 mm mild steel plate; (2.680-3.289) mms^{-1} for 0.7 mm plate but for thicker mild steel plates 0.8 mm -1.0 mm, the welding speeds appreciated. For 0.8 mm plate, the welding speed ranges from (3.867-4.560) mms^{-1}; for 0.9 mm plate, (4.267-5.267) mms^{-1} and for 1.0 mm plate, (4.333-5.500) mms^{-1}. Table 2 also indicates, for electric arc welding (manual), welding time range of (15-45) s for 0.5 mm thick plate to 1.0 mm mild steel plate.

VII. SUMMARY OF THE RESULT DISCUSSION

At reduced or lesser time, for the same steel plate specimen, length of weld is clearly greater in developed robot welding (Automated) than in the electric arc welding (Manual). This shows that more time would be saved, under the same operating conditions, using the developed welding robot.

From all indications, welding speed, on the average, increases as the thickness of steel plate increases. This is more pronounced in the electric arc welding where welding slows down when welding operation is carried out on less thicker plates. However, the welding speeds of developed welding robot are far greater than those of the electric arc welding. This is an advantage and should be of interest to the Manufacturing Industry in Nigeria in the areas of precision machining, mass production of items, reduction of machining time and cost of production, increased productivity, maximization of profit.

VIII. CONCLUSION

In conclusion, the results of the predictive model developed for welding mini-robot when compared with the manual electric arc welding indicated that welding with the developed welding mini-robot based on the developed model will reduce the production cost and increase the quality as well as the reliability of weld during welding processes.

IX. ACKNOWLEDGMENT

The authors are thankful to all doctorates thesis supervisors and technologists at the Federal University of Technology, Akure and the Federal Polytechnic Ado Ekiti who played a vital role in making this research project a success.

X. References

[1]. United Nations and International Federation of Robots, (2000) "World industrial robots 1996: statistics and forecasts", New York: ONU/IFR, 2000.

[2] Ilesanmi, A. D., Khumbulani, M. and Adefemi, O. A. (2019). Optimization of welding parameters using Taguchi and response surface methodology for rail car bracket assembly, *The International Journal of Advanced Manufacturing Technology*, 100(1): 2221–2228.

[3] Kusiak, A., (1986). Modelling and design of flexible manufacturing systems. *Elsevier*: Amsterdam, 1986.

[4] Kusiak, A., (2000). Computational intelligence in design and manufacturing, John Wiley & Sons: New York.

[5] Rosheim, M. E., (1994). *Robot evolution: the development of anthrobots*. John Willey & Sons: New York.

[6] Rosheim, M. E., (1997). In the footsteps of Leonardo. *Robotics & Automation Magazine, IEEE, 4* (2), 12-14.

[7] Pedretti, C., (1981). *Leonardo architect*. Rizzoli International Publications: New York, 1981.

[8] Tesla, N., (1983). *My inventions: autobiography of Nicola Tesla*. Willinston, VT: Hart Brothers.

[9] Myhr, M., (1999). "Industrial new trends: ABB view of the future", International Workshop on Industrial Robotics, New Trends and Perspectives (http://robotics.dem.uc.pt/ir99/), Parque das Nações, Lisbon.

[10] Bolmsjo, G., (1997) "Sensor system in arc welding", Technical Report, Lund Institute of Technology, Production and Materials Engineering Department, 1997.

[11] Bolmsjö, G.; Olsson M.; Nikoleris G.; Brink K., (1995). In *Task programming of welding robots*. In Proceedings of the Int. Conf. on the Joining of Materials, JOM-7, pages 573-585, May-June 1995.
[12] Loureiro, A.; Velindro M.; Neves F., (1998). The influence of heat input and the torch weaving movement on robotized MIG weld shape. *International Journal for the Joining of Materials*, *10* (3), 86-91.
[13] Agren, B., *Sensor integration for robotic arc welding*. PhD thesis, Lund University, 1995.
[14] Richardson, R. W., (1986). Robotic weld joint tracking systems - theory and implementation methods. *Welding Journal*, *65* (11) 43-51.
[15] Books, B., (1991). "Welding robots - the state of the art", Welding and Metal Fabrication, June 1991.
[16] Drews, P.; Starke G., 91986). "Development approaches for advanced adaptive control in automated arc welding", Internal Report XII-970, Mechatronics Department, Achen, Germany, 1986.
[17] Koyama, T.; Takahashi, Y.; Kobayashi, M.; Morisawa, J., (1989). Development of real time welding control system by using image processing. *Quarterly Journal of the Japan Welding Society*, *7*(3), 79-83.
[18] Pires, J. N.; Sá da Costa, J., (2000). Object-oriented and distributed approach for programming robotic manufacturing cells. *Robotics and Computer-Integrated Manufacturing*, *16* (1), 29-42.
[19] Pires, JN.; Monteiro, P.; Schölzke, V., (2001). "Using robot manipulators on high efficient wrapping machines for paper industry", ISR'2001, Seoul, Korea, April 2001.
[20] Pires, J. N., (2000). Using Matlab to interface industrial robotic and automation equipment. *IEEE Robotics and Automation Magazine*, *7* (3).
[21] Pires, J. N., (2000). Programming industrial robotic and automation equipment. *Industrial Robot, An International Journal*, MCB University Press, July.
[22] Sá da Costa, J. M. G.; Pires, JN., (2001). Future welding robot developments, *Journal Robótica*, *No 41*, January.
[23] Singh, S. (2010)**:** Mechanical Engineer's Handbook, Khanna Publishers, Darya Ganj, New Delhi-110002.

APPENDIX I
MATLAB PROGRAMME FOR THE CARTESIAN WELDING ROBOT MODEL

```
function varargout = Welding_Calc(varargin)
% WELDING_CALC M-file for Welding_Calc.fig
%      WELDING_CALC, by itself, creates a new WELDING_CALC or raises the existing
%      singleton*.%
%      H = WELDING_CALC returns the handle to a new WELDING_CALC or the handle to
%      the existing singleton*.%
%      WELDING_CALC('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in WELDING_CALC.M with the given input arguments.%
%      WELDING_CALC('Property','Value',...) creates a new WELDING_CALC or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Welding_Calc_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Welding_Calc_OpeningFcn via varargin.%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help Welding_Calc
% Last Modified by GUIDE v2.5 10-Feb-2016 13:44:52
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Welding_Calc_OpeningFcn, ...
                   'gui_OutputFcn',  @Welding_Calc_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before Welding_Calc is made visible.
function Welding_Calc_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Welding_Calc (see VARARGIN)
% Choose default command line output for Welding_Calc
handles.output = hObject;
set(handles.uitable1,'data',{1,1,1;1,1,1;1 1 1; 1 1 1; 1 1 1; 1 1 1; 1 1 1; 1 1 1; 1 1 1;1 1 1; 1 1 1;1 1 1;1 1 1;1 1 1;1 1 1;1 1 1 })
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes Welding_Calc wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line.
function varargout = Welding_Calc_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function length_Callback(hObject, eventdata, handles)
% hObject    handle to length (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of length as text
%        str2double(get(hObject,'String')) returns contents of length as a double
% --- Executes during object creation, after setting all properties.
function length_CreateFcn(hObject, eventdata, handles)
% hObject    handle to length (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global value
%tik=get(handles.thickness,'string')
let=get(handles.length,'string');
tick=str2num(value);
lent=str2num(let);
eff=0.8;
set(handles.effi,'string',eff);
WeldV=get(handles.WeldVolt,'string');
WeldS1=get(handles.WeldSpeed1,'string');
```

```
WeldVo=str2num(WeldV);
WeldSp1=str2num(WeldS1);
Dia=0.8;
set(handles.Diameter,'string',Dia);
Curre=get(handles.curr,'string');
Curren=str2num(Curre);
HI1=((eff*Curren*WeldVo)/(WeldSp1));
set(handles.Heatinpout,'string',HI1);
WT=lent/WeldSp1;
set(handles.Weldtimeout,'string',WT);
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (se GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%tik=get(handles.thickness,'string');
global value
let=get(handles.length,'string');
tick=str2num(value);
lent=str2num(let);
eff=0.8;
set(handles.effi,'string',eff);
WeldV=get(handles.WeldVolt,'string');
Heatin1=get(handles.HeatInp,'string');
WeldVo=str2num(WeldV);
Heatin1=str2num(Heatin1);
Dia=0.8;
set(handles.Diameter,'string',Dia);
Curre=get(handles.curr,'string');
Curren=str2num(Curre);
WeldSp2=((eff*Curren*WeldVo)/(Heatin1));
set(handles.Weldspeedout,'string',WeldSp2);
WT=lent/WeldSp2;
set(handles.Weldtimeout2,'string',WT);
function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double
% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end
function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double
% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double
% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
data=get(handles.uitable1,'data');
xdata=data(:,1);
ydata=data(:,2);
zdata=data(:,3);
save 'xdata.mat' xdata
save 'ydata.mat' ydata
save 'zdata.mat' zdata
%call the graph
close(gcf)
cartes_init
% --- Executes during object creation, after setting all properties.
function text26_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text26 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over text26.
function text26_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to text26 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit9 as text
%        str2double(get(hObject,'String')) returns contents of edit9 as a double
% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function HeatInp_Callback(hObject, eventdata, handles)
% hObject    handle to HeatInp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of HeatInp as text
%        str2double(get(hObject,'String')) returns contents of HeatInp as a double
% --- Executes during object creation, after setting all properties.
function HeatInp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to HeatInp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function WeldVolt_Callback(hObject, eventdata, handles)
% hObject    handle to WeldVolt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of WeldVolt as text
%        str2double(get(hObject,'String')) returns contents of WeldVolt as a double
% --- Executes during object creation, after setting all properties.
function WeldVolt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to WeldVolt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function WeldSpeed1_Callback(hObject, eventdata, handles)
% hObject    handle to WeldSpeed1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of WeldSpeed1 as text
%        str2double(get(hObject,'String')) returns contents of WeldSpeed1 as a double
% --- Executes during object creation, after setting all properties.
function WeldSpeed1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to WeldSpeed1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function Heateff_Callback(hObject, eventdata, handles)
% hObject    handle to Heateff (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of Heateff as text
%        str2double(get(hObject,'String')) returns contents of Heateff as a double
% --- Executes during object creation, after setting all properties.
function Heateff_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Heateff (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
% --- Executes on selection change in thickness.
function thickness_Callback(hObject, eventdata, handles)
% hObject    handle to thickness (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String') returns thickness contents as cell array
%        contents{get(hObject,'Value')} returns selected item from thickness
global value
contents = get(hObject,'String');
     value= contents{get(hObject,'Value')};
% --- Executes during object creation, after setting all properties.
function thickness_CreateFcn(hObject, eventdata, handles)
% hObject    handle to thickness (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit15 as text
%        str2double(get(hObject,'String')) returns contents of edit15 as a double
% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function curr_Callback(hObject, eventdata, handles)
% hObject    handle to curr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of curr as text
%        str2double(get(hObject,'String')) returns contents of curr as a double
% --- Executes during object creation, after setting all properties.
function curr_CreateFcn(hObject, eventdata, handles)
% hObject    handle to curr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

APPENDIX II
MATLAB PROGRAMME OF THE CARTESIAN WELDING ROBOT SIMULATION

```
% Cartesian Robot Simulator - with X-Y-Z actuators
%
% This is a simple simulator of a 3 DOF cartesian robot.
%
% This code basically draws the robot accurately, and refreshes the drawing
% based on the relations between the links and the motors. So it is rather
% ideal than realistic. However, simulation results on physical models
% can be visualized using this type of approach or models can be
% implemented, as future work. It is also possible to use this as an
% interface of a small scale prototype which is picN microcontroller
% based (this was why this code was written actually).
%
% You can freely use this code, but I would appreciate any credit too.
%
% 2013 - Eindhoven
%
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% INITIALIZATION CODE
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
close all;
clear all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%
% PARAMETERS
%
%% call the initial figurs
test
%initial position
ee_x = 0; %%ee-end effector (desired)
ee_y = 0;
ee_z = 0;
%plot axis
x_max = 100.0;
x_min = -5.0;
y_max = 105.0;
y_min = -5.0;
z_max = 105.0;
z_min = -5.0;
%frame
fr_x0 = 0.0;
fr_x1 = 100.0;
fr_y0 = 0.0;
fr_y1 = 100.0;
fr_height = 20.0;
fr_upper2_height = 20.0;
%links
link_x_offsetx = 0;
link_x_offsety = 2.5;
link_x_offsetz = 0;
link_y_offsetx = 2.5;
link_y_offsety = 0;
link_y_offsetz = 40.0;
link_z_length = link_y_offsetz+fr_upper2_height;
link_z_offsetx = 2.5;
link_x_stepspercm = 0.3;
link_y_stepspercm = 0.3;
link_z_stepspercm = 0.3;
%motors
motor_x_step = 360;
motor_y_step = 360;
```

```
motor_z_step = 360;
motor_x_curstep = 0;
motor_y_curstep = 0;
motor_z_curstep = 0;
motor_x_line_length = 10.0;
motor_y_line_length = 10.0;
motor_z_line_length = 10.0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%initial calculations
%

link_x_height = fr_height;
link_x_length = fr_x1 - fr_x0 + 2*link_x_offsetx;
link_x_x0 = fr_x0-link_x_offsetx;
link_x_x1 = fr_x1+link_x_offsetx;
link_x_y0 = fr_y0-link_x_offsety;
link_x_y1 = fr_y0-link_x_offsety;
link_x_z0 = link_x_height;
link_x_z1 = link_x_height;
link_x_steps = link_x_stepspercm*link_x_length*motor_x_step;
ee_motor_x=(ee_x/link_x_length)*link_x_steps;
ee_x = (ee_motor_x/link_x_steps)*link_x_length;
motor_x_curstep = mod(ee_motor_x,motor_x_step);
link_y_height = fr_height+link_y_offsetz;
link_y_length = fr_y1 - fr_y0 + 2*link_y_offsety;
link_y_x0 = (ee_motor_x/link_x_steps)*link_x_length+link_z_offsetx;
link_y_x1 = (ee_motor_x/link_x_steps)*link_x_length+link_z_offsetx;
link_y_y0 = fr_y0-link_y_offsety;
link_y_y1 = fr_y1+link_y_offsety;
link_y_z0 = link_y_height;
link_y_z1 = link_y_height;
link_y_steps = link_y_stepspercm*link_y_length*motor_y_step;
ee_motor_y=(ee_y/link_y_length)*link_y_steps;
ee_y = (ee_motor_y/link_y_steps)*link_y_length;
motor_y_curstep = mod(ee_motor_y,motor_y_step);
% motors
motor_x_posx = fr_x1+link_x_offsetx;
motor_x_posy = fr_y0-link_x_offsety;
motor_x_posz = link_x_height;
motor_y_posx = link_y_x0;
motor_y_posy = link_y_y0;
motor_y_posz = link_y_z0;
motor_x_ang = -(motor_x_curstep/motor_x_step)*2.0*pi;
motor_y_ang = (motor_y_curstep/motor_y_step)*2.0*pi;
motor_x_line_x0 = motor_x_posx;
motor_x_line_x1 = motor_x_posx;
motor_x_line_y0 = motor_x_posy;
motor_x_line_y1 = motor_x_posy+cos(motor_x_ang)*motor_x_line_length;
motor_x_line_z0 = motor_x_posz;
motor_x_line_z1 = motor_x_posz+sin(motor_x_ang)*motor_x_line_length;
motor_y_line_x0 = motor_y_posx;
motor_y_line_x1 = motor_y_posx+cos(motor_y_ang)*motor_y_line_length;
motor_y_line_y0 = motor_y_posy;
motor_y_line_y1 = motor_y_posy;
motor_y_line_z0 = motor_y_posz;
motor_y_line_z1 = motor_y_posz+sin(motor_y_ang)*motor_y_line_length;
% upper frame
fr_up_x0 = link_y_x0+link_y_offsetx;
fr_up_x1 = link_y_x0+link_y_offsetx;
fr_up_y0 = link_x_y0;
fr_up_y1 = fr_y1+link_x_offsety;
fr_up_z0 = link_x_z0;
fr_up_z1 = link_y_z0;
```

```
fr_up2_x0 = link_y_x0;
fr_up2_x1 = link_y_x0;
fr_up2_y0 = (ee_motor_y/link_y_steps)*link_y_length;
fr_up2_y1 = (ee_motor_y/link_y_steps)*link_y_length;
fr_up2_z0 = link_y_z0;
fr_up2_z1 = link_y_z0+fr_upper2_height;
link_z_steps = link_z_stepspercm*link_z_length*motor_z_step;
ee_motor_z=(ee_z/link_z_length)*link_z_steps;
ee_z = fr_height + (ee_motor_z/link_z_steps)*link_z_length;
motor_z_curstep = mod(ee_motor_z,motor_z_step);
motor_z_ang = (motor_z_curstep/motor_z_step)*2.0*pi;
link_z_x0 = link_y_x0-link_z_offsetx;
link_z_x1 = link_z_x0;
link_z_y0 = fr_up2_y0;
link_z_y1 = link_z_y0;
link_z_z0 = fr_height+(ee_motor_z/link_z_steps)*link_z_length;
link_z_z1 = link_z_z0 + link_z_length;
motor_z_posx = link_z_x0;
motor_z_posy = link_z_y0;
motor_z_posz = link_z_z1;
motor_z_line_x0 = motor_z_posx;
motor_z_line_x1 = motor_z_posx+cos(motor_z_ang)*motor_z_line_length;
motor_z_line_y0 = motor_z_posy;
motor_z_line_y1 = motor_z_posy+sin(motor_z_ang)*motor_z_line_length;
motor_z_line_z0 = motor_z_posz;
motor_z_line_z1 = motor_z_posz;
drawing=[ee_x ee_y ee_z];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
% Drawing
%
%initialize
fig = figure('Name', 'Cartesian Robot - 3 DOF', 'NumberTitle', 'off',...
        'Position', [5,60,800,940]);
set(gca, 'drawmode', 'fast');
set(fig, 'MenuBar', 'none', 'BackingStore', 'off');
% set(fig, 'BackingStore', 'off');
plot_main = subplot('Position', [0.05,0.28,0.90,0.70]);
axis([x_min x_max y_min y_max z_min z_max]); grid minor; hold on;%axis equal;
set(gca,'Color',[0.05 0.05 0.13]);
set(gca,'XColor',[0.1 0.4 0.1]);
set(gca,'YColor',[0.4 0.1 0.1]);
set(gca,'ZColor',[0.2 0.2 0.1]);
rotate3d(gca,'on');
set(gcf,'toolbar','figure');
% Frame
line_framex0 = line('xdata', [fr_x0  fr_x1],'ydata', [fr_y0 fr_y0],'zdata', [fr_height fr_height],'erasemode','normal','LineWidth', 1,'color', 'blue');
line_framex1 = line('xdata', [fr_x0  fr_x1],'ydata', [fr_y1 fr_y1],'zdata', [fr_height fr_height],'erasemode','normal','LineWidth', 1,'color', 'blue');
line_framey0 = line('xdata', [fr_x0  fr_x0],'ydata', [fr_y0 fr_y1],'zdata', [fr_height fr_height],'erasemode','normal','LineWidth', 1,'color', 'blue');
line_framey1 = line('xdata', [fr_x1  fr_x1],'ydata', [fr_y0 fr_y1],'zdata', [fr_height fr_height],'erasemode','normal','LineWidth', 1,'color', 'blue');
line_frame_leg11 = line('xdata', [fr_x0  fr_x0],'ydata', [fr_y0 fr_y0],'zdata', [-5.0 fr_height],'erasemode','normal','LineWidth', 1,'color', 'blue');
line_frame_leg12 = line('xdata', [fr_x1  fr_x1],'ydata', [fr_y0 fr_y0],'zdata', [-5.0 fr_height],'erasemode','normal','LineWidth', 1,'color', 'blue');
line_frame_leg21 = line('xdata', [fr_x0  fr_x0],'ydata', [fr_y1 fr_y1],'zdata', [-5.0 fr_height],'erasemode','normal','LineWidth', 1,'color', 'blue');
line_frame_leg22 = line('xdata', [fr_x1  fr_x1],'ydata', [fr_y1 fr_y1],'zdata', [-5.0 fr_height],'erasemode','normal','LineWidth', 1,'color', 'blue');
line_upperframe = line('xdata', [fr_up_x0  fr_up_x1],'ydata', [fr_up_y0  fr_up_y1],'zdata', [fr_up_z1 fr_up_z1],'erasemode','normal','LineWidth', 3,'color', 'blue');
```

```
line_upperframeleg0 = line('xdata', [fr_up_x0  fr_up_x1],'ydata', [fr_up_y0  fr_up_y0],'zdata', [fr_up_z0
fr_up_z1],'erasemode','normal','LineWidth', 3,'color', 'blue');
line_upperframeleg1 = line('xdata', [fr_up_x0  fr_up_x1],'ydata', [fr_up_y1  fr_up_y1],'zdata', [fr_up_z0
fr_up_z1],'erasemode','normal','LineWidth', 3,'color', 'blue');
line_upperframe2 = line('xdata', [fr_up2_x0  fr_up2_x1],'ydata', [fr_up2_y0  fr_up2_y1],'zdata', [fr_up2_z0
fr_up2_z1],'erasemode','normal','LineWidth', 3,'color', 'blue');
% Links
line_link_x = line('xdata', [link_x_x0  link_x_x1],'ydata', [link_x_y0 link_x_y1],'zdata', [link_x_z0
link_x_z1],'erasemode','normal','LineWidth', 1,'color', 'yellow');
line_link_y = line('xdata', [link_y_x0  link_y_x1],'ydata', [link_y_y0 link_y_y1],'zdata', [link_y_z0
link_y_z1],'erasemode','normal','LineWidth', 1,'color', 'yellow');
line_link_z = line('xdata', [link_z_x0  link_z_x1],'ydata', [link_z_y0 link_z_y1],'zdata', [link_z_z0
link_z_z1],'erasemode','normal','LineWidth', 1,'color', 'yellow');
% Motors
line_motor_x = line('xdata', [motor_x_line_x0 motor_x_line_x1],'ydata', [motor_x_line_y0 motor_x_line_y1],'zdata',
[motor_x_line_z0 motor_x_line_z1],'erasemode','normal','LineWidth', 1,'color', 'red');
line_motor_y = line('xdata', [motor_y_line_x0 motor_y_line_x1],'ydata', [motor_y_line_y0 motor_y_line_y1],'zdata',
[motor_y_line_z0 motor_y_line_z1],'erasemode','normal','LineWidth', 1,'color', 'red');
line_motor_z = line('xdata', [motor_z_line_x0 motor_z_line_x1],'ydata', [motor_z_line_y0 motor_z_line_y1],'zdata',
[motor_z_line_z0 motor_z_line_z1],'erasemode','normal','LineWidth', 1,'color', 'red');
point_ee = plot3(ee_x,ee_y,ee_z,'r.');
DRAW = plot3(drawing(:,1),drawing(:,2),drawing(:,3),'g');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
% GUI
 sli_X = uicontrol(fig,'Style','slider','Pos',[70 100 120 17],...
     'Min',0,'Max',link_x_steps,'SliderStep',[1/200 1/200],...
     'Val',ee_motor_x,'CallBack',['ee_motor_x=(get(sli_X,"Val"));,'...
       'set(X_cur,"String",num2str(ee_motor_x)),','cartes_draw;']);
sli_Y = uicontrol(fig,'Style','slider','Pos',[270 100 120 17],...
     'Min',0,'Max',link_y_steps,'SliderStep',[1/200 1/200],...
     'Val',ee_motor_y,'CallBack',['ee_motor_y=(get(sli_Y,"Val"));,'...
       'set(Y_cur,"String",num2str(ee_motor_y)),','cartes_draw;']);
sli_Z = uicontrol(fig,'Style','slider','Pos',[470 100 120 17],...
     'Min',0,'Max',link_z_steps,'SliderStep',[1/100 1/100],...
     'Val',ee_motor_z,'CallBack',['ee_motor_z=(get(sli_Z,"Val"));,'...
       'set(Z_cur,"String",num2str(ee_motor_z)),','cartes_draw;']);
 X_min = uicontrol(fig,'Style','text','Pos',[037 100 32 17],'String','0');
 Y_min = uicontrol(fig,'Style','text','Pos',[237 100 32 17],'String','0');
 Z_min = uicontrol(fig,'Style','text','Pos',[437 100 32 17],'String','0');
 X_max = uicontrol(fig,'Style','text','Pos',[192 100 40 17],'String',num2str(link_x_steps));
 Y_max = uicontrol(fig,'Style','text','Pos',[392 100 40 17],'String',num2str(link_y_steps));
 Z_max = uicontrol(fig,'Style','text','Pos',[592 100 40 17],'String',num2str(link_z_steps));
 X_label = uicontrol(fig,'Style','text','Pos',[60 120 60 17],'String','X:');
 Y_label = uicontrol(fig,'Style','text','Pos',[260 120 60 17],'String','Y:');
 Z_label = uicontrol(fig,'Style','text','Pos',[460 120 60 17],'String','Z:');
  X_cur = uicontrol(fig,'Style','text','Pos',[122 120 40 17],'String',...
     num2str(get(sli_X,'Value')));
 Y_cur = uicontrol(fig,'Style','text','Pos',[322 120 40 17],'String',...
     num2str(get(sli_Y,'Value')));
 Z_cur = uicontrol(fig,'Style','text','Pos',[522 120 40 17],'String',...
     num2str(get(sli_Z,'Value')));
 pbreset = uicontrol(fig,'Style','push','Pos',[20 5 60 44],...
     'String','RESET','CallBack',[...
      'set(sli_X,"Value",0),','set(sli_Y,"Value",0),','set(sli_Z,"Value",0),'...
'set(X_cur,"String","0"),','set(Y_cur,"String","0"),','set(Z_cur,"String","0"),'.
      'ee_motor_x = 0; ee_motor_y = 0; ee_motor_z = 0;ee_x=0;ee_y=0;ee_z=0;','drawing=[0 0 fr_height];',...
      'refresh(fig);cartes_draw;']);
pbdemo = uicontrol(fig,'Style','push','Pos',[300 5 60 44],...
     'String','DEMO','CallBack','draw_demo');
 % end
% 10.02.14
```