Real-time Control System for a forwarder loader

Date	11.12.2018
Group	Stone
Authors	Toan Le Duc, 267945 Pallab Ganguly, 267602

Abbreviations and definitions

BCS	Boom Control System
HMI	Human-Machine Interface
GUI	Graphical User Interface
TCP	Tool Center Point
DOF	Degree of freedom

Table of contents

1 Introduction

In general, the course exercise requires students to design and implement a real-time control system for a 4 degree-of-freedom (DOF) hydraulic loader (i.e. boom) and test it in a rapid prototyping environment.

Because of workloads in this exercise, there are 4 phases for students to follow. Starting with defining Use Cases, Requirements and Preliminary Design. After that, Rapid Prototyping environment is included with IHA 3D environment. At third phase, students should complete implementing in off-line and online with Rapid Prototyping environment which includes User Interface and most of functionalities of the system should be done.

After three phases, this document presents the final work with modified model after Demonstration meeting and report of finished tasks. In this document, we present the ideas we agreed in designing real-time control system and how we improved them after each feedback meeting which are described in seven section as Use Cases, Requirements, User Interfaces, System Design and Implementation and Testing.

Finally, we come to the Future Development where we propose the idea to make our system more stable and could be applied in real-life problems.

2 General description

A 4-dof loader is a type of tractor, usually wheeled have a bucket mounted to it in front which is connected by two booms generally used to scoop up materials such as sand, gravels from the ground. The loader assembly can be removable or permanent. The major components in the loader are engine (diesel is most cases), hydraulic components (valves, hoses, pump and motors) and transmission components (gearbox, axles/wheels)

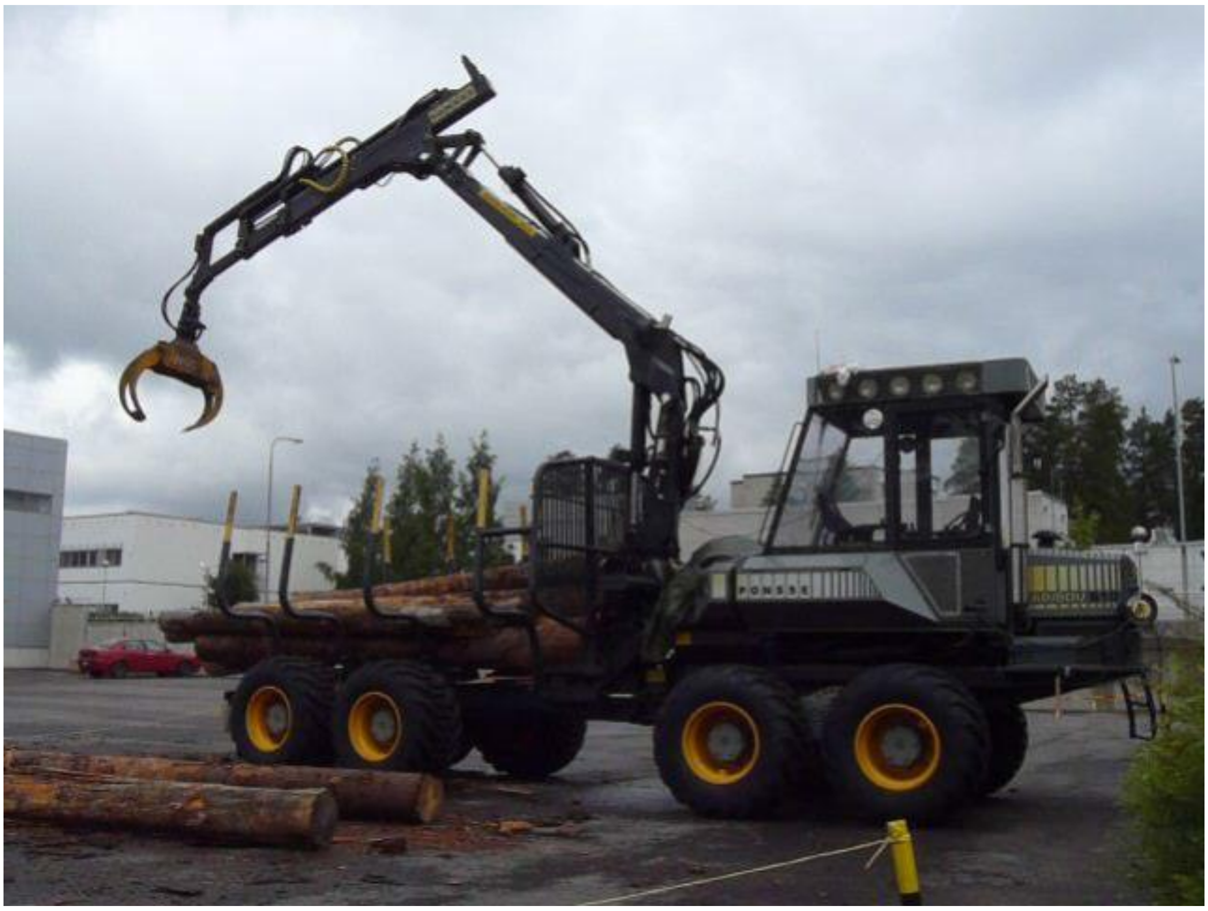


Figure 1. Forward Loader

In this exercise, the model of loader is given and the task is design a real-time control system which is running in HMI real-time machine. The system receives commands signal of joystick and GUI which are controlled by operator. The joystick and GUI are running on Host Computer. Matlab Simulink is used as the environment for implementation where GUI is designed in Simulink Real-time Explorer.

3 Use cases

Use case is a collection of possible sequences of interactions between the system under discussion and its external characters, related to achieve a particular goal.

Use cases can connect one actor's goal to a system or to another actor's responsibility. Each interaction can give a brief introduction of a low-level use case. From user view, the BCS provides an operating shift where operator has ability to control the boom in different method as control movement of boom joints, boom tip or control boom joints without influence or effect of feedback measurements and input.

3.1 Use case diagram

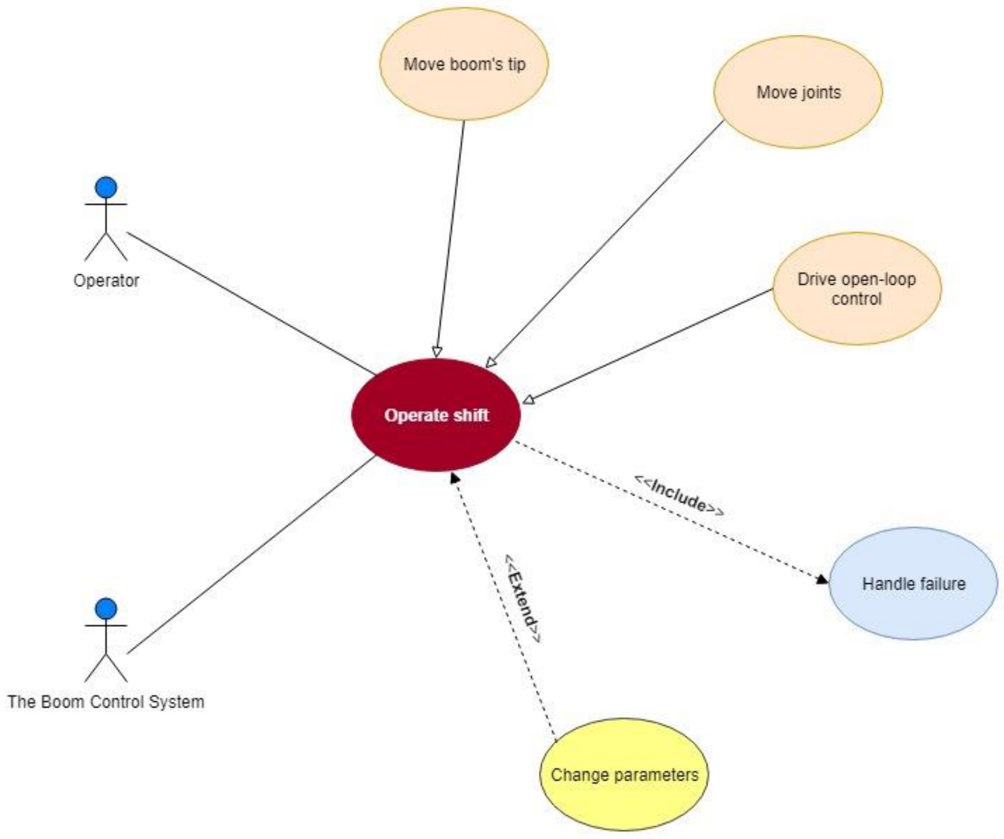


Figure 2. Use case diagram of Boom control system

3.2 Use case descriptions

3.2.1 Base use case

Use case ID	**UC-0101**
Use case name	***Operate shift***
Actors	*Operator, BCS*
Precondition	*The boom is turned on.*

Flow of events	1. Operator press *Start* button to start a shift. 2. The BCS illustrates system working details: a. The BCS shows *Operation Mode* b. The BCS shows *Joints location (q1, q2, q3, d4)* c. The BCS shows *Planar location (q1, x, y)* d. The BCS shows *Failure situation* 3. Operator chooses operation mode between *Move joints* and *Move boom's tip* and *Drive open-loop control*. 4. The BCS sets up operation mode: a. If *Move joints* mode is selected: **UC-0202 *Move joints*** b. If *Move boom's tip* mode is selected: **UC-0201 *Move boom's tip*** c. If *Drive open-loop control* mode is selected: **UC-0203 Drive open-loop control** 5. The BCS changes mode to the selected and display in user interface 6. Operator uses joystick to control the boom. 7. Operator modifies parameters for working purpose, **UC-0301 *Change parameters*** 8. Operator press *Stop* button to finish the shift.
Exceptions	The operating is stop when the system detects some errors, then the system will stop this use case and turn into **UC-0402 *Handle failure.***
Post-conditions	*Operator finishes his/her job and turn off the machine.*

3.2.2 Child use case

Use case ID	**UC-0201**
Use case	***Move boom's tip***
Actors	*Operator, The Boom Control System (BCS)*
Base Use Case	*Operate shift*
Precondition	*Operator choose Move boom's tip mode in operating shift*
Flow of Events	1. Operator uses joystick to control the boom a. Operator pushes/pulls the joystick forward/backward, the BCS moves the boom forward/backward in y-coordinated plane b. Operator turns the joystick left/right, the BCS moves the boom left/right in x-coordinated plane c. Operator rotates the joystick left/right, the BCS rotates the boom left/right (q1 joint)
Exceptions	
Post-conditions	*Operator drives the boom in planar moving.*

Use case ID	**UC-0202**
Use case	***Move joints***

Actors	*Operator, The Boom Control System (BCS)*
Base Use Case	*Operate shift*
Precondition	*Operator choose Move joints mode in operating shift*
Flow of Events	1. Operator selects drive control type between *Open-loop* and *Closed-loop*. 2. The BCS sets up for drive control type: *a.* If *Open-loop* is selected, **UC-0203 *Drive open-loop control*** b. *Closed-loop control* is default selected 3. Operator uses joystick to control the boom a. Operator rotates the joystick left/right and the BCS rotates the pillar left/right (q1 joint) b. Operator pushes/pulls the joystick in forward/backward and, the BCS extends/shortens the lift cylinder (q2 joint) c. Operator turns the joystick left/right, the BCS extends/shortens the tilt cylinder (q3 joint) d. Operator slides joystick's analog slider to left/right, the BCS extends/shortens the extension cylinder (d4 joint)
Exceptions	
Post-conditions	*Operator drives the boom by controlling each joint motion.*

Use case ID	**UC-0203**
Use case	***Drive open-loop control***
Actors	*The Boom Control System (BCS)*
Base Use Case	*Operate shift*
Precondition	*The boom is operated in Move joints mode or errors detected in system.*
Flow of Events	1. The BCS changes controller type to open-loop type 2. Operator uses joystick to control the boom movements 3. The BCS sets the boom movements and speed based on movements and acceleration of joystick
Exception	
Post-conditions	*Operator controls the boom manually.*

3.2.3 Extended use case

Use case ID	**UC-0301**
Use case	***Change parameters***
Actors	*Operator, The Boom Control System (BCS)*
Base Use Case	*Operate shift*
Extension Point	*Operate shift*
Precondition	*The boom is on operating.*
Flow of Events	1. Operator enters new parameters for operating ***a.*** Operator modifies ***controller gain***

	b. Operator modifies ***following error limit*** ***c.*** Operator modifies ***joint soft limit*** ***d.*** Operator modifies ***maximum velocity of each joint*** 2. The BCS updates its parameters
Post-conditions	*The BCS is working with new parameter from Operator input.*

3.2.4 Intended use case

Use case ID	**UC-0402**
Use case name	***Handle failure***
Actors	*The Boom Control System (BCS)*
Including Use Case	*Operate shift*
Precondition	*Errors detected in system.*
Flow of events	1. The BCS sets up solutions. a. The BCS stops all motion b. The BCS shows what is errors c. The BCS displays what has ability to work *2.* Operator selects suitable control type.
Exceptions	The system is recovered and come back to work. Change to **UC-0101 *Operate shift***.
Post-conditions	*System failure is handled and the is no dangerous issue happened.*

4 Requirements

From described ideas in Use Cases, we defined the tasks should be done and constraints of system in more details through the Requirements. This step helps us to keep on track where we should come on the work flow. The Requirements has 3 parts which are Functional Requirements, Non-functional Requirements and Constraints of the BCS. Besides, we marks where the requirements is applied in for tracking completed tasks where the number is heading of implemented section in this document.

4.1 Functional Requirements

General		*Implemented*
RF001	The BCS shall have a Power button.	5.1.1
RF002	The Operator shall start the Operate Shift by pressing Power button.	5.1.1
RF003	The BCS shall have a user interface.	5.1
RF004	The BCS shall display the joint angles of the boom in user interface.	5.1.2
RF005	The BCS shall display operating modes in user interface.	5.1.2
RF006	The BCS shall provide 3 operating options: Move boom's tip, Move joints and Drive Open-loop control.	6.8
RF007	The Operator shall select the used operating option by user interface.	5.2.1
RF008	The BCS shall let the Operator input own parameters.	5.1.3
RF009	The BCS shall display the parameters of system in user interface.	5.1.3
RF010	The Operator shall control the boom by moving joystick.	5.2
RF011	The BCS shall stop when the system has errors.	6.3, 6.5, 6.8
RF012	The Operator shall control the boom in same operating mode after the system recovering.	6.4, 6.8
RF013	The BCS shall have an Emergency button.	5.2.1
RF014	The BCS shall stop when Emergency button is pressed.	6.8
RF015	The Operator shall control the boom out of collision area.	6.5, 6.8
RF016	The Operator shall stop the Operate Shift by pressing the Power button.	5.1.1

4.2 Non-functional Requirements

		Implemented
RN001	The BCS shall not work when Emergency button is pressed or system has errors.	6.3, 6.5, 6.8
RN002	The BCS shall have one operating option is active at a time.	6.8
RN003	The BCS shall display one working operating option at a time.	5.1.2

4.3 Constraints

		Implemented
#0001	The BCS shall have the sample time is set to 2 milliseconds.	6
#0002	The BCS shall control boom without crossing physical limits.	5.1.1
#0003	The BCS shall work without vibration of boom.	6.5
#0004	The Host PC shall have Matlab Simulink 2018a	6

#0005	The Host PC shall have IHA 3D as simulation environment	6.1, 6.2
#0006	The Host PC shall have Visual Studio 2017 as compiler for Real-time model	6
#0007	The Boom Simulink model shall be provided	6.6
#0008	The joystick communication Simulink model shall be provided	6.1, 6.2, 6.4
#0009	The joystick 4x12 shall be provided	5.2, 6.1, 6.2

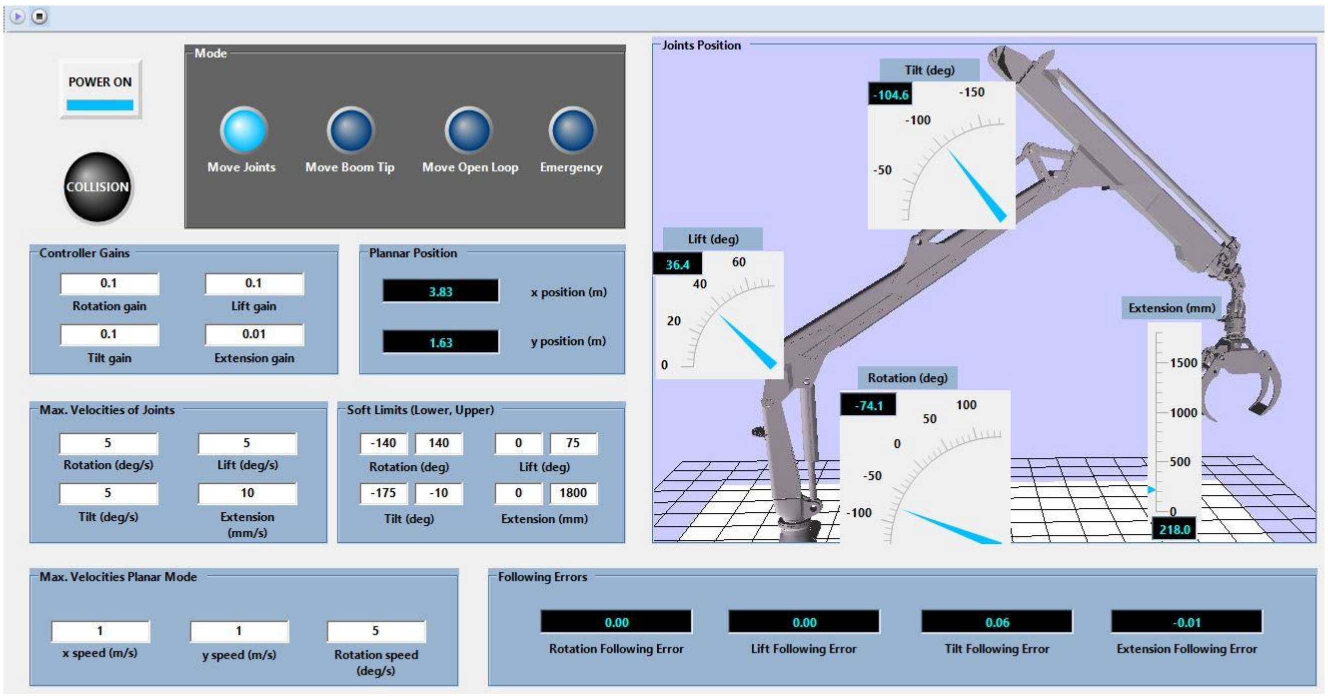
5 User Interfaces

The BCS aim is giving operator ability to control the boom in different types. Because of that, it is needed to design a user interface, which helps operator control the system easily. Besides, the boom is working under big load so friendly user interface gives a hand to reduce unwanted errors of operating.

The user interfaces includes software interface and hardware interface, which are known as Graphical User Interface and Joystick in this case. The Graphical User Interface provides information about working system and let the operator modify system parameters to keep it smoothly working or handle the errors. Whereas, the operator use the Joystick to give direction command to the boom and choose operating mode.

5.1 Graphical User Interface

The GUI is designed in Simulink Real-time Explorer. It illustrates current status of running BCS for monitoring. The operator can modify the default parameters of the BCS by entering new value. Besides, the measured data of the boom are presented in curve gauges and under digital number also which are clearer to follow. There are several sections in big panel. Each panel is described more details below.



Controller Gains

0.1 Rotation gain	0.1 Lift gain
0.1 Tilt gain	0.01 Extension gain

Plannar Position

3.83	x position (m)
1.63	y position (m)

Max. Velocities of Joints

5 Rotation (deg/s)	5 Lift (deg/s)
5 Tilt (deg/s)	10 Extension (mm/s)

Soft Limits (Lower, Upper)

-140 140 Rotation (deg)	0 75 Lift (deg)
-175 -10 Tilt (deg)	0 1800 Extension (mm)

Max. Velocities Planar Mode

1 x speed (m/s)	1 y speed (m/s)	5 Rotation speed (deg/s)

Following Errors

0.00 Rotation Following Error	0.00 Lift Following Error	0.06 Tilt Following Error	-0.01 Extension Following Error

Joints Position: Tilt (deg) -104.6; Lift (deg) 36.4; Rotation (deg) -74.1; Extension (mm) 218.0

Figure 3. Graphic User Interface

5.1.1 "POWER ON" Button and "COLLISION" LED light

"POWER ON" Button has two values, light-on and light-off. When it is light-on, the BCS is ignited and operator can use the joystick. In other hand, the button is light-off which disables joystick ability of the joystick. The operator cannot use joystick to change mode or move the boom. Although, the joystick commands have no effect on the boom, the mode is chosen before the “POWER ON” button is on will be remembered and when the button is in light-on state, the BCS will jumps into that mode instead of “Waiting” station.

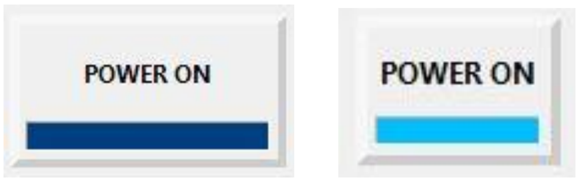


Figure 4. "POWER ON" button (left: light-off value, right: light-on value)

When the boom is moving close to collision space, the ”COLLISION” LED light is on and the BCS prevents movement control from operator which could lead the boom to the collision space.

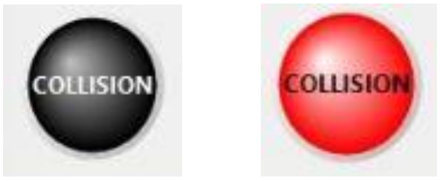
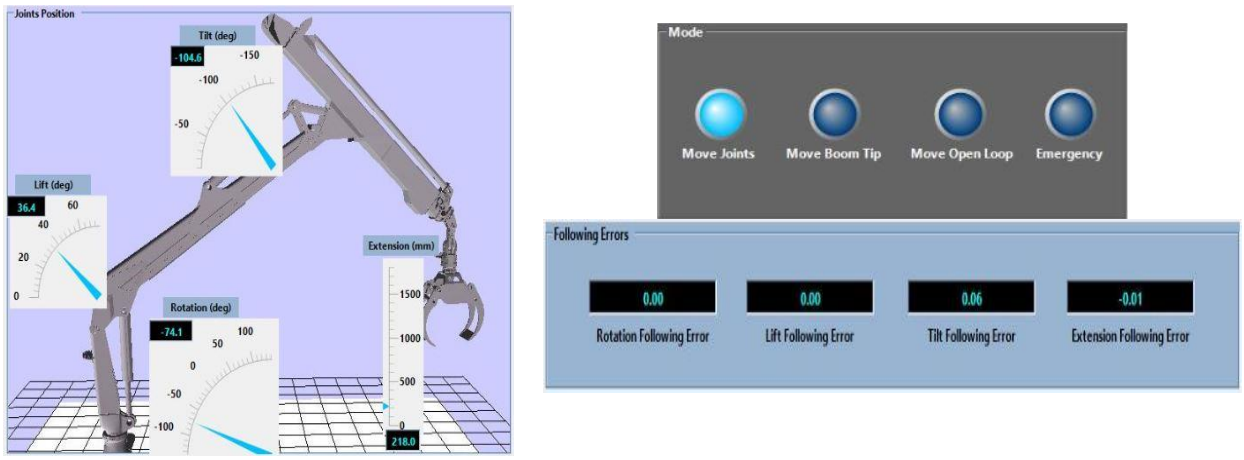


Figure 5. "COLLISION" LED light (left: no collision, right: collision)

5.1.2 Monitoring Group

We used Group Box selection in designing this group, which illustrates current status of the BCS. Furthermore, chosen operating mode is presented by LEDs in Mode section. Operator can only choose 1 operating mode at a timed. In Figure 6, Move Joints mode is running. On the right side of GUI, the big square section named Joints Position show the operator the current position of each joint through curve gauge, slide gauge and digital numbers. As following requirements, the measured data has its unit in its label. Last member of Monitoring Group is 4 numeric display which show the following errors of 4 joints.



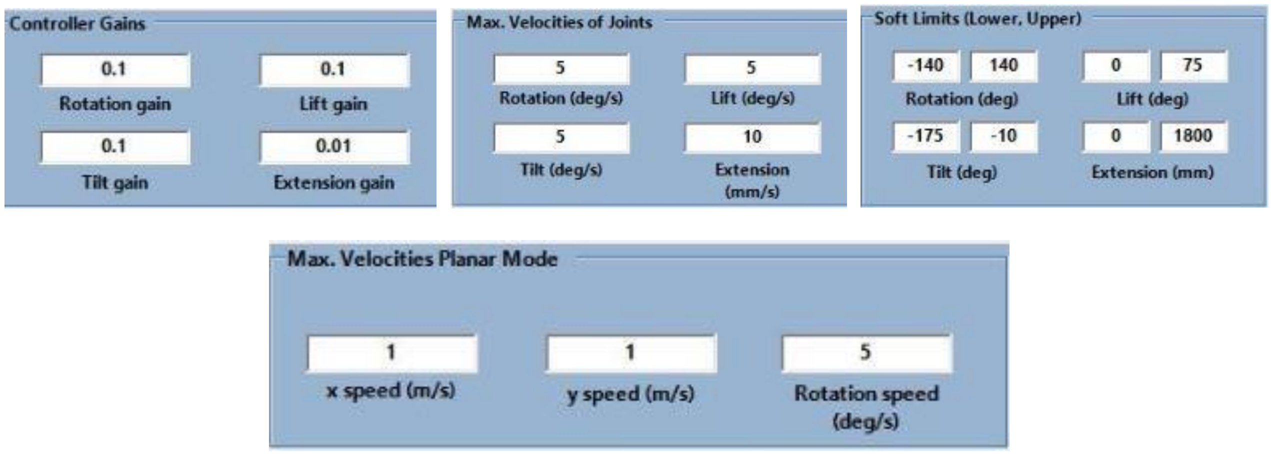
Following Errors

Rotation Following Error	Lift Following Error	Tilt Following Error	Extension Following Error
0.00	0.00	0.06	-0.01

Figure 6. Monitoring Group in GUI

5.1.3 Modifying Group

In this group, the operator has ability to change default parameters of the BCS. There are 4 sections which are Controller Gains, Maximum Velocities of Joints, Maximum Velocities Planar Mode and Soft Limits.



Controller Gains

Rotation gain	Lift gain	Tilt gain	Extension gain
0.1	0.1	0.1	0.01

Max. Velocities of Joints

Rotation (deg/s)	Lift (deg/s)	Tilt (deg/s)	Extension (mm/s)
5	5	5	10

Soft Limits (Lower, Upper)

	Lower	Upper
Rotation (deg)	-140	140
Lift (deg)	0	75
Tilt (deg)	-175	-10
Extension (mm)	0	1800

Max. Velocities Planar Mode

x speed (m/s)	y speed (m/s)	Rotation speed (deg/s)
1	1	5

Figure 7. Modifying Group in GUI

The Controller Gains are P gain of every mode. These parameters has impact roles in control velocity, stability of the boom. In additions, Max. Velocities of Joints is default joints' maximum velocities and they are used in Move Joints and Move Open Loop mode. Besides, Max. Velocities Planar Mode is used in Move Boom Tip mode, where they are desired velocities in x, y coordinates and rotation speed.

Final part of Modifying Group is Soft Limits section, which presents the lower and upper limitation of each joint. The operator has ability to change these values to make operating shift saftier. However, the current position of the boom should be included in new limitations or the BCS will count it as error and operator cannot control overpassed joint. Example, the current position of Lift is 60 degrees then we should not set new upper limitation as less-than-60 value because the BCS will stop control Lift while other joints still working.

5.2 Joystick

In this exercise, we designed joystick for picking mode and boom control purpose. The operator guilds the joystick to guild the boom moving as desired directions or presses the button on top of the joystick to choose operating mode.

5.2.1 Picking mode

As show in Figure 8, there are four buttons presenting as four operating mode. Button number 5 is Move Joints mode, button number 6 is Emergency mode, button number 3 is Move Open Loop mode and button number 4 is Move Boom Tip mode.

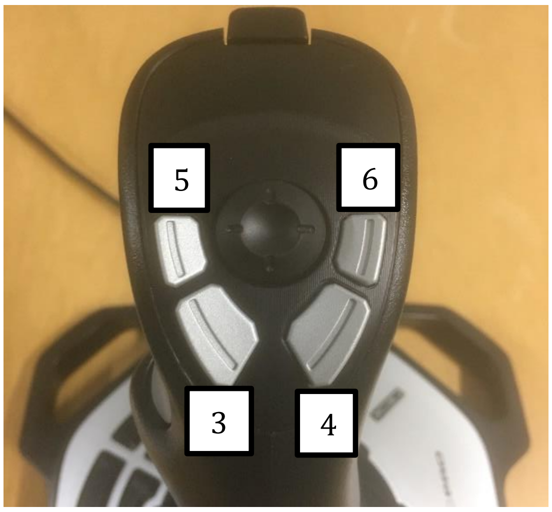


Figure 8. Operating mode buttons

5.2.2 "Move Joints" mode and "Move Open Loop" mode

Operator guilds the joystick move forward or backward to descrease or increase lift angle of boom. Guilding the joystick to left will move the boom positive rotating angle and guilding it to right then the boom will go to negative rotating angle. Rotating the joystick around its default position make the tilt angle follow as described in Figure 9.

As the extension of the boom, there is small slider, which is marked by blue rectangle, control the length of extension. Sliding up for collapse the extension and sliding down for expand the extension. This small slider is hard to put it back to "zero" place then when we implemented with Rapid Prototyping environment, we have faced unexpected pseudo errors.

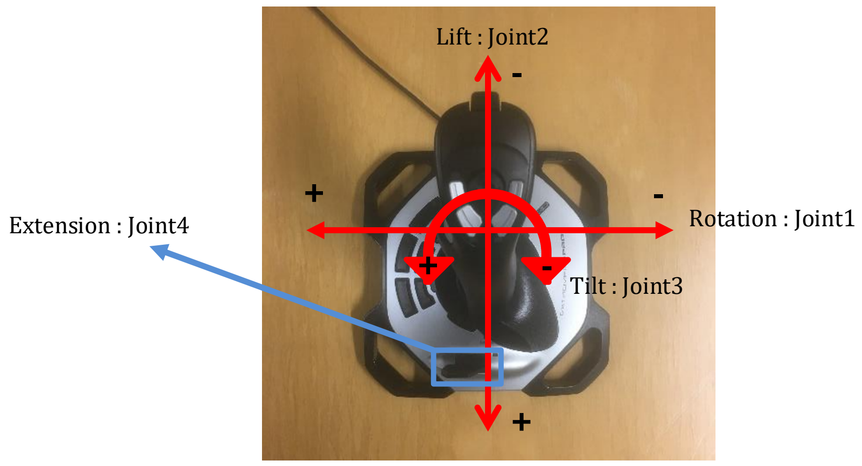


Figure 9. Joystick movements in "Move Joints" mode and "Move Open Loop" mode.

5.2.3 "Move Boom Tip" mode

There are three movements was designed in "Move Boom Tip" mode, which are move the boom along x-axis, along y-axis and rotate the boom around. Those movements are presented through movement of joystick when operator is controlling as below.

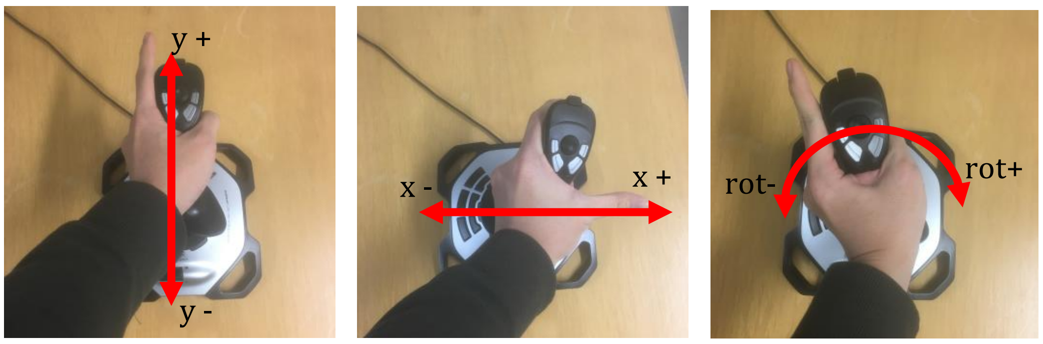


Figure 10. Directions of joystick in "Move Boom Tip" mode

(left: move x, center: move right, right: rotation)

5.3 UI functionality

The functionality of elements in UI are summarized in Table 1.

Table 1. UI Functionality

UI Element name	UI Element type	Purpose	Fulfils requirement ID
"POWER ON"	GUI button	Turn of/off the system.	RF001
"COLLISION"	GUI LED	Indicates boom's tip collision.	#0002

”Joint Positions”	GUI gauges and numeric display box	Display 4 joints’ positions.	RF004
”Mode”	GUI LED box	Display current operating mode	RF005, RN003
”Following Errors”	GUI numeric display box	Display the following errors of each joint.	RF004
”Controller Gains”	GUI box	Display editable P gain in controller of the BCS.	RF008, RF009
” Max. Velocities of Joints”	GUI box	Display editable maximum velocity of each joint in ”Move Joints” and ”Move Open Loop” mode.	RF008, RF009
” Max. Velocities Planar Mode”	GUI box	Display editable maximum desired velocity in x-axis, y-axis and rotaion speed in ”Move Boom Tip” mode.	RF008, RF009
” Soft Limits”	GUI box	Display editable upper and lower limitations of each joint.	RF008, RF009
”Button 3”	Joystick button	Turn on/off Move Open Loop mode.	RF007
”Button 4”	Joystick button	Turn on/off Move Boom Tip mode.	RF007
”Button 5”	Joystick button	Turn on/off Move Joints mode.	RF007
”Button 6”	Joystick button	Turn on/off Emergency mode.	RF007, RF013
Forward/Backward	Joystick movement	1. ”Move Joints” and ”Move Open Loop” mode: Lift angle control 2. ”Move Boom Tip” mode: y-axis movement control	RF010
Left/Right	Joystick movement	1. ”Move Joints” and ”Move Open Loop” mode: Rotation angle control 2. ”Move Boom Tip” mode: x-axis movement control	RF010
Rotate Left/Right	Joystick movement	1. ”Move Joints” and ”Move Open Loop” mode: Tilt angle control 2. ”Move Boom Tip” mode: rotation control	RF010
Slide Up/Down	Joystick slider movement	1. ”Move Joints” and ”Move Open Loop” mode: Extension control 2. ”Move Boom Tip” mode: No function	RF010

6 System Design

6.1 High-level architecture

In this course, we are introduced to Simulink Real-time system which provides as Figure 11. The dash-line connections are software running in physical components and the continuous line connections are connected by cable.

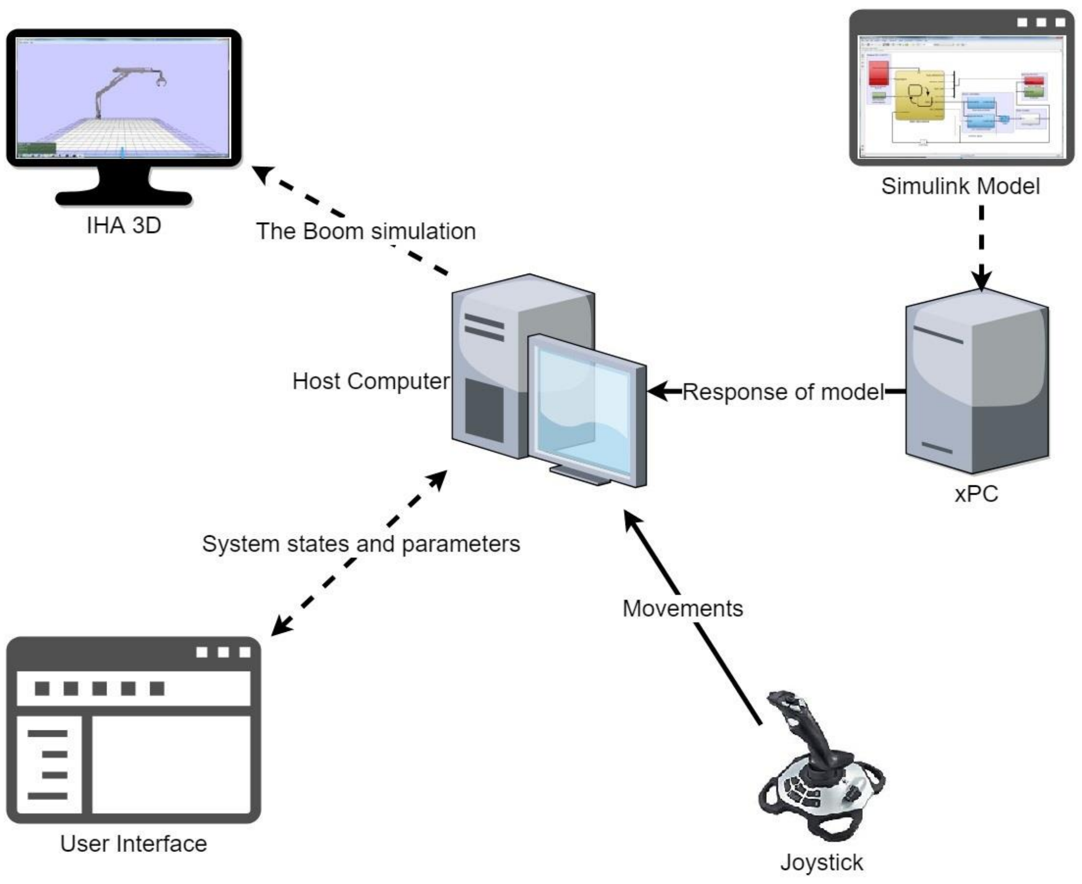


Figure 11. High-level architecture

For details, high-level software components are described below

Table 2. Hardware and software relationships

Physical Components	Software Components	Software Modules
Host PC	Matlab	Simulink Real-time
		Simulink Real-time Explorer
	IHA 3D	
Joystick	Matlab Simulink	Joystick 3D Communication
xPC	Simulink Real-time Boot	The BCS Real-time model

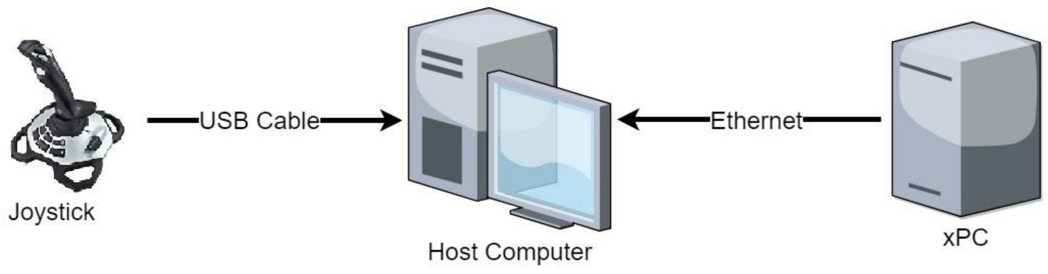


Figure 12. High-level connections

Figure 12 illustrates the set up of components where Joystick communicate with Host Computer through USB Cable and their data are sent by Joystick 3D Communication model which is mentioned in Table 2. Hardware and software relationships. Otherwise, xPC is connected to Host Computer through static IP addresses.

6.2 Real-time system architecture

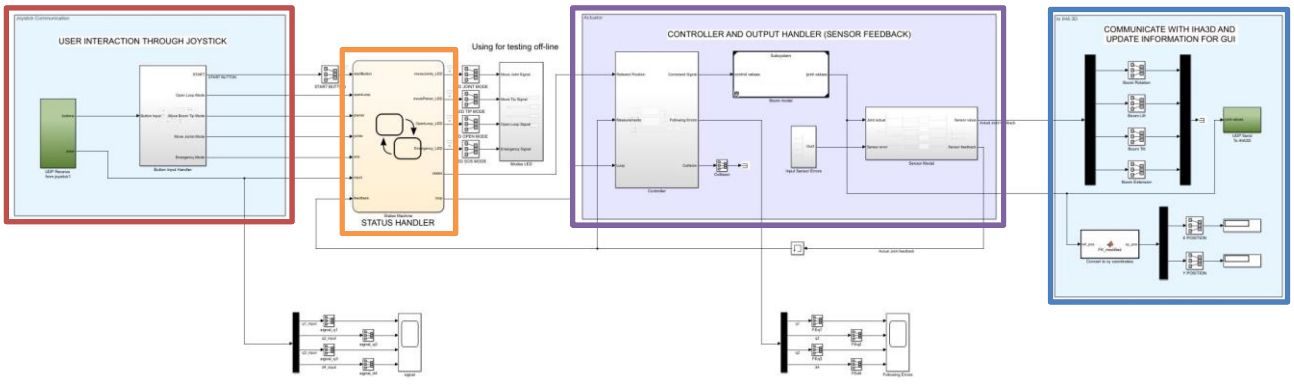

USER INTERACTION THROUGH JOYSTICK
STATUS HANDLER
Using for testing off-line
CONTROLLER AND OUTPUT HANDLER (SENSOR FEEDBACK)
COMMUNICATE WITH IHA3D AND UPDATE INFORMATION FOR GUI

Figure 13. Real-time system architecture in general view

In general view, the Real-time system architecture has 4 main part which are shown in Figure 13. Those part from left to right are User Interaction (Red rectangle), State Machine (Orange rectangle), Controller and Sensor (Purple rectangle) then the last part at right side is Communicate With IHA3D (Blue rectangle). As a flow of data from left to right, the Real-time system has its flow as description below

1. User Interaction part has purpose to convert signal from joystick and GUI into commands for next part of the architecture.

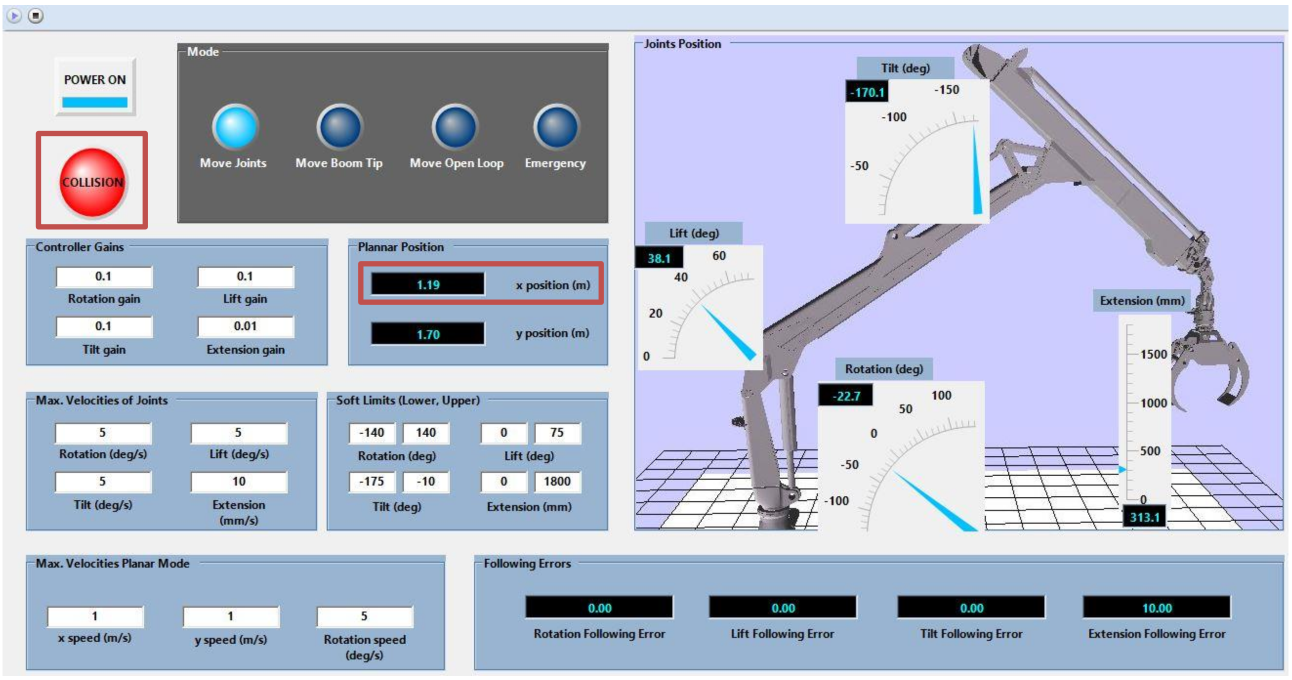
2. State Machine receives command and generate reference position for the boom while checks for saftey purpose and inform the operator by sending LED values to GUI.

3. Controller processes the reference position then give command to the boom model, which is provided. After receiving joint values from the boom model, the Sensor model will add some disturbances to the values then send the measurements back to State Machine and Controller to handle.

4. Communicate With IHA3D part gets the joint values from the boom model to visualize in rapid prototyping environment and generate those values in “xy” coordinates then sends them to GUI for operator to monitor.

6.3 Safety considerations

The BCS detects physical collisions when the boom tip moving. This detection is to make sure the boom tip is not pass the minimum edge so it could collise with other part of the boom in rapid prototyping environment. We implemented this dectection in two part State Machine and Controller. How the safety applied in these two parts will be discussed more in next sections. Besides, we also add an “Emergency” button to stop immediately the boom if the operator find some danger and want to careful check.



Controller Gains

Rotation gain	0.1
Lift gain	0.1
Tilt gain	0.1
Extension gain	0.01

Plannar Position

x position (m)	1.19
y position (m)	1.70

Max. Velocities of Joints

Rotation (deg/s)	5
Lift (deg/s)	5
Tilt (deg/s)	5
Extension (mm/s)	10

Soft Limits (Lower, Upper)

Rotation (deg)	-140	140
Lift (deg)	0	75
Tilt (deg)	-175	-10
Extension (mm)	0	1800

Max. Velocities Planar Mode

x speed (m/s)	1
y speed (m/s)	1
Rotation speed (deg/s)	5

Following Errors

Rotation Following Error	0.00
Lift Following Error	0.00
Tilt Following Error	0.00
Extension Following Error	10.00

Joints Position

Tilt (deg)	-170.1
Lift (deg)	38.1
Rotation (deg)	-22.7
Extension (mm)	313.1

Figure 14. When the TCP's x position is smaller than collision edge on x-axis (1.2m). The BCS will notice operator by LED light.

As measured in off-line implementation, we notice that the edge for collision is from 0m to 1m along x-axis and we want to make the system smoothly work then we come up with collision edge on x-axis as 1.2m. Due to the boom could be attached on 4-wheel loader so we finally chose -0.64m in y-axis as collision edge for the TCP and with that choice we could solve a problem that at starting, the TCP is lower than 0m in y-axis which cause the system stop working due to safety conditions. Generally, we have chosen minumum x position is 1.2m and minimum y position is -0.64m.

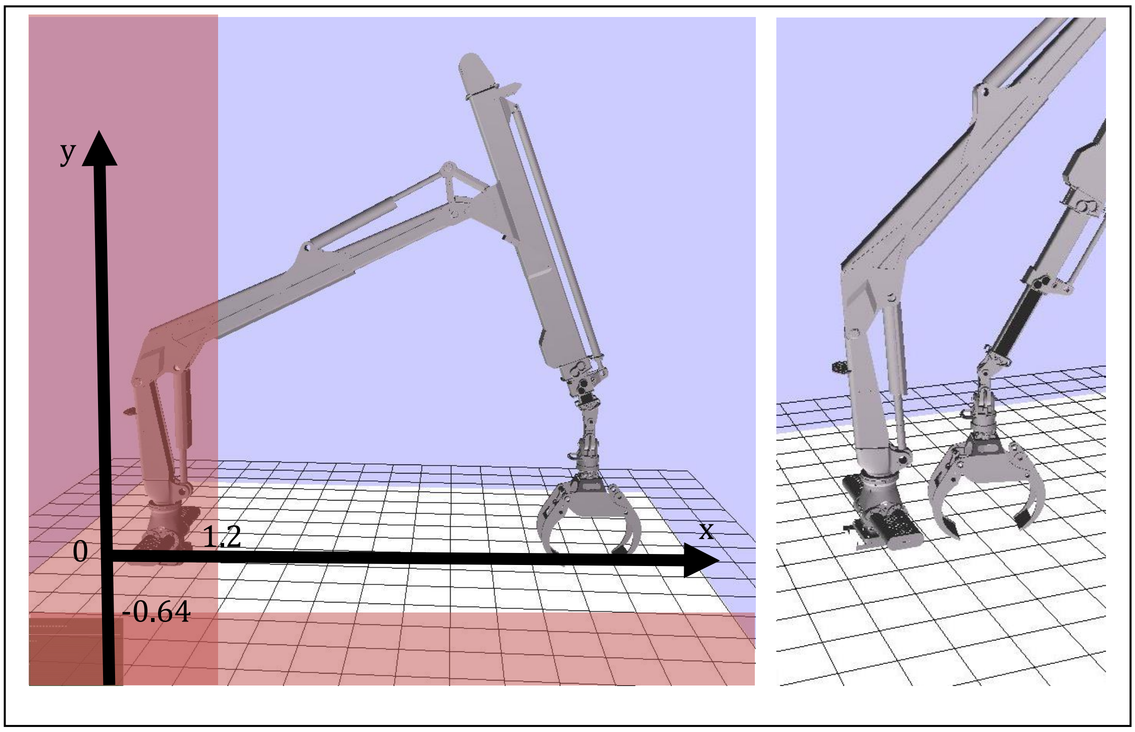


Figure 15. Collision edges as red block (left) and example of reaching x-axis edge (right)

6.4 Inputs and outputs of the Real-time system

6.4.1 Inputs

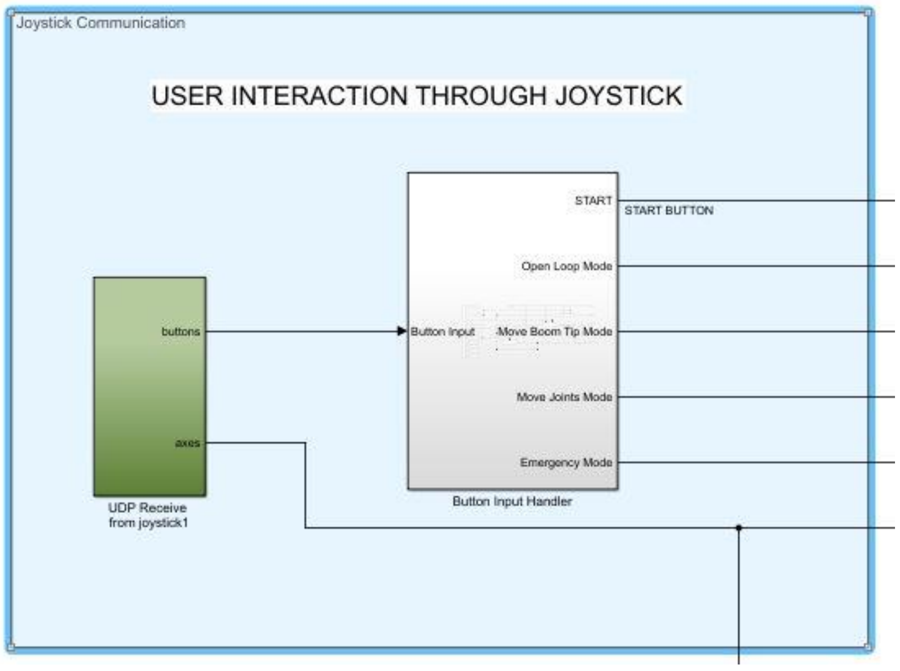


Figure 16. Inputs handling area.

This area aims to receive joystick signals by provided “UDP Receive from joystick1” block then modify the signals as State Machine command by “Button Input Handler” block.

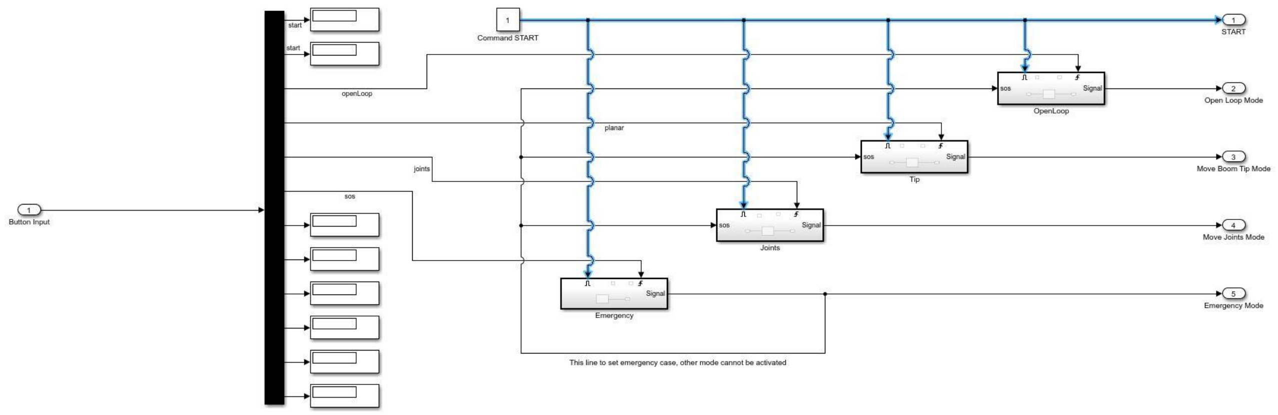


Figure 17. Architecture in "Button Input Handler" block.

The ”Command START” is parameter of ”POWER ON” button in GUI. This parameter enables other blocks. The other blocks is sub-system of Matlab function to hold the signal values until the sub-system get trigger again. The code will be presented in Appendix 3.

6.4.2 Outputs

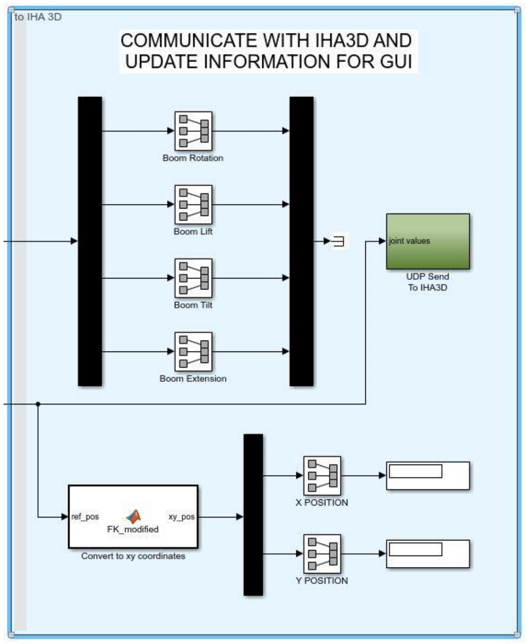


Figure 18. Output handling area

At this area, the joint values from the boom is sent to IHA3D environment, converted to GUI signals and generated as x and y position also. This area has main purpose is visualizing results in IHA3D and GUI for lively monitoring.

6.5 Controller

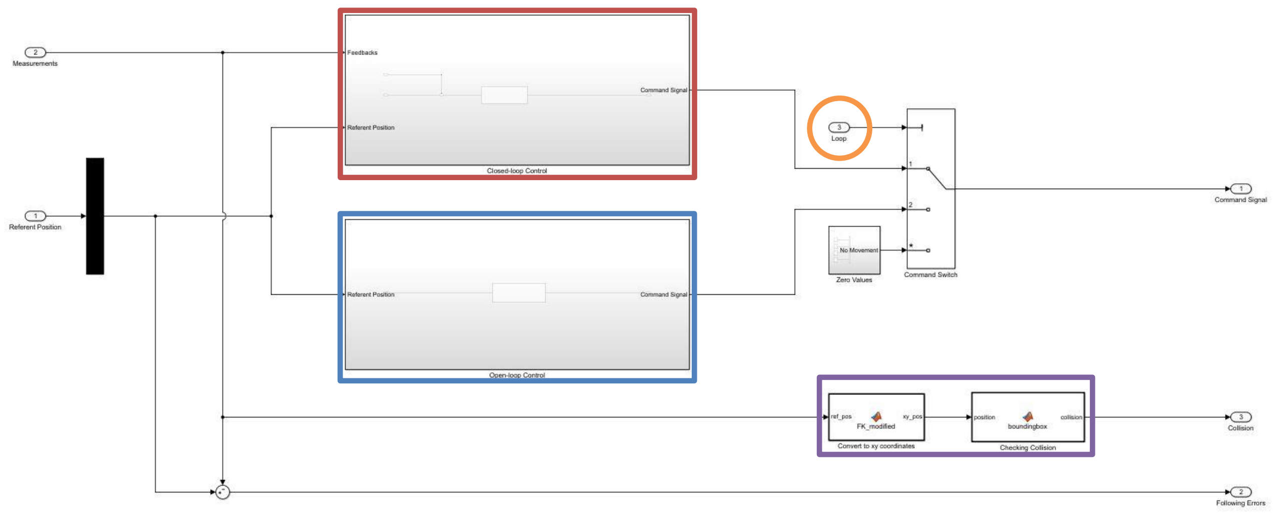


Figure 19. Architecture in "Controller" block. Closed-loop Control (Red rectangle), Open-loop Control (Blue rectangle) and Collision Detection (Purple rectangle)

The "Controller" receives reference position, control type from State Machine and measurements from Sensor then return the results which are Command Signal to the boom model, collision detected signal and following errors of joints. In both Closed-loop and Open-loop control type, the P-gain control is used for each joint. The orange circle is control type command from State Machine. This command will guild "Controller" block return Command Signal from Closed-loop Control if it is 1, from Open-loop Control if it is 2 and if it is other cases then Command Signal is zeros.

6.6 Boom model

The input if the Boom is the control values which is the output from the controller as shown in figure 6. The output if the Boom model is the joints values of all the four joints which are send to the UDP send for connecting in a real time environment.

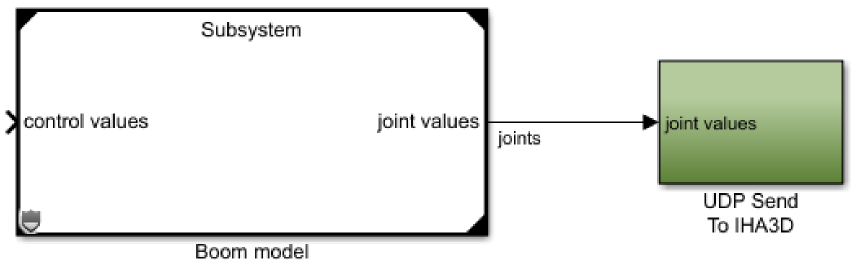
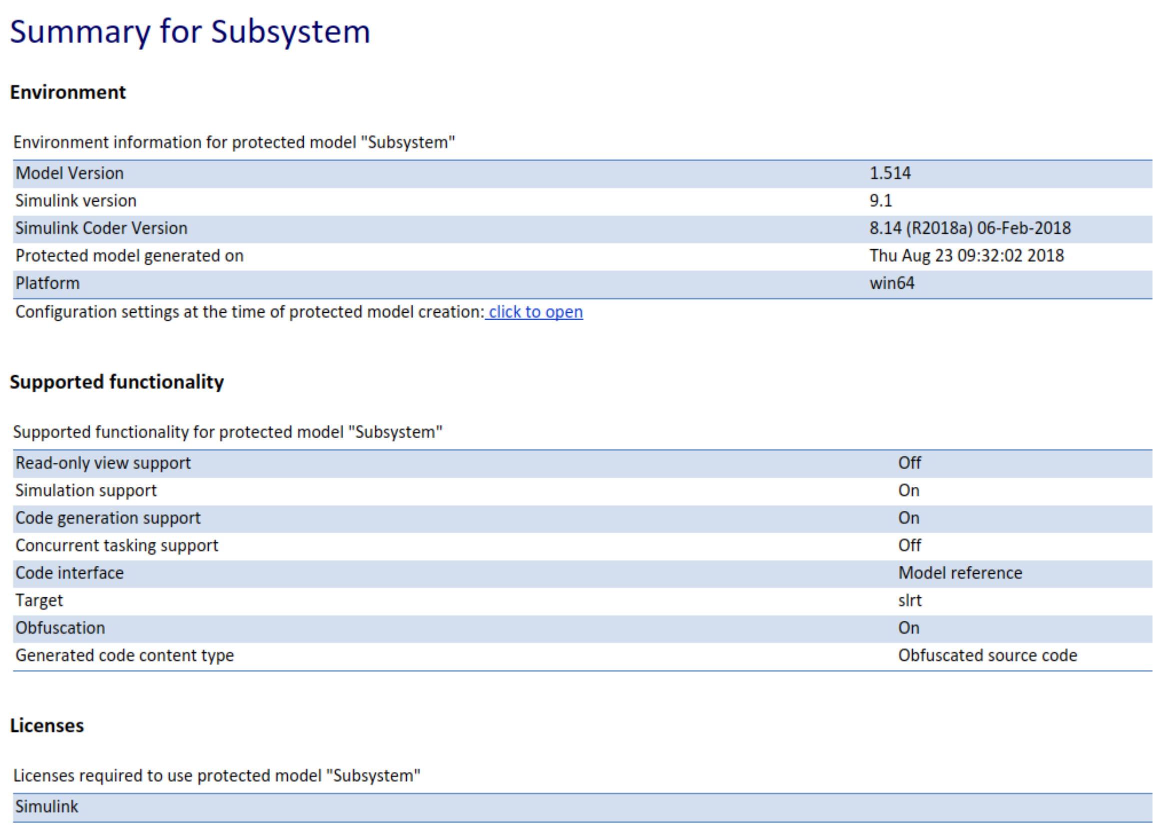


Figure 20. Boom model

The Boom model is a protected subsystem which cannot be accessed but can be used for simulation. It is made in Simulink version 2018a and can only run on 2018a as it can be seen in the figure below.



Summary for Subsystem

Environment

Environment information for protected model "Subsystem"

Model Version	1.514
Simulink version	9.1
Simulink Coder Version	8.14 (R2018a) 06-Feb-2018
Protected model generated on	Thu Aug 23 09:32:02 2018
Platform	win64

Configuration settings at the time of protected model creation: click to open

Supported functionality

Supported functionality for protected model "Subsystem"

Read-only view support	Off
Simulation support	On
Code generation support	On
Concurrent tasking support	Off
Code interface	Model reference
Target	slrt
Obfuscation	On
Generated code content type	Obfuscated source code

Licenses

Licenses required to use protected model "Subsystem"

Simulink

Figure 21. Protected Boom submodel

6.7 Sensor model

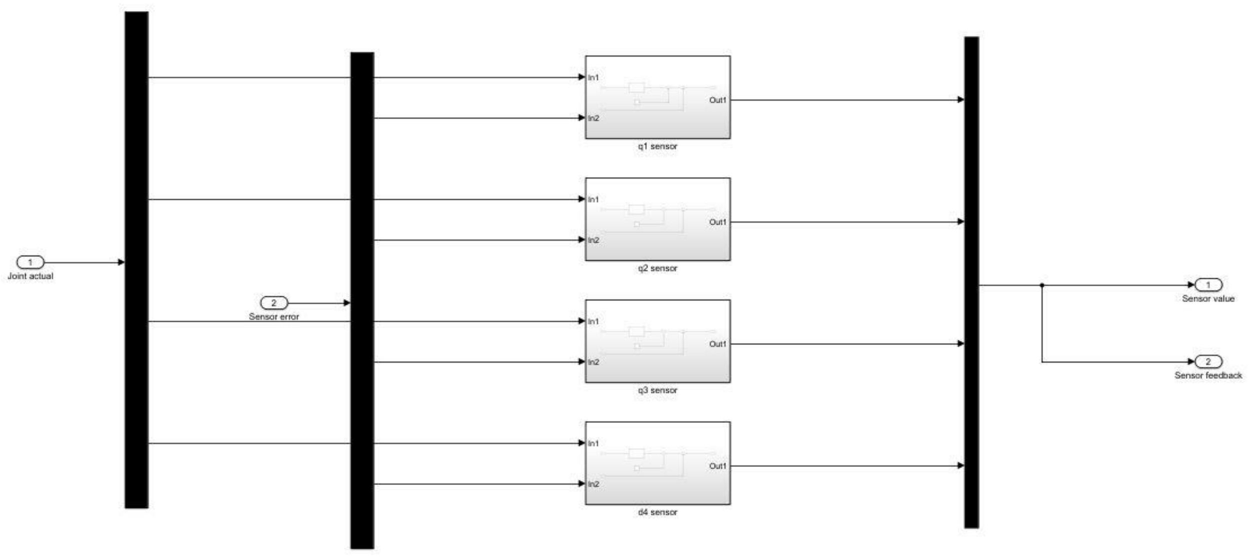


Figure 22. "Sensor model" block architecture.

Based on the sensor model presented in previous exercise, we applied that model for each joint in our “Sensor model” block. This block will add some disturbances to joint values from the boom model and send sensed values to State Machine and Controller.

6.8 State machine

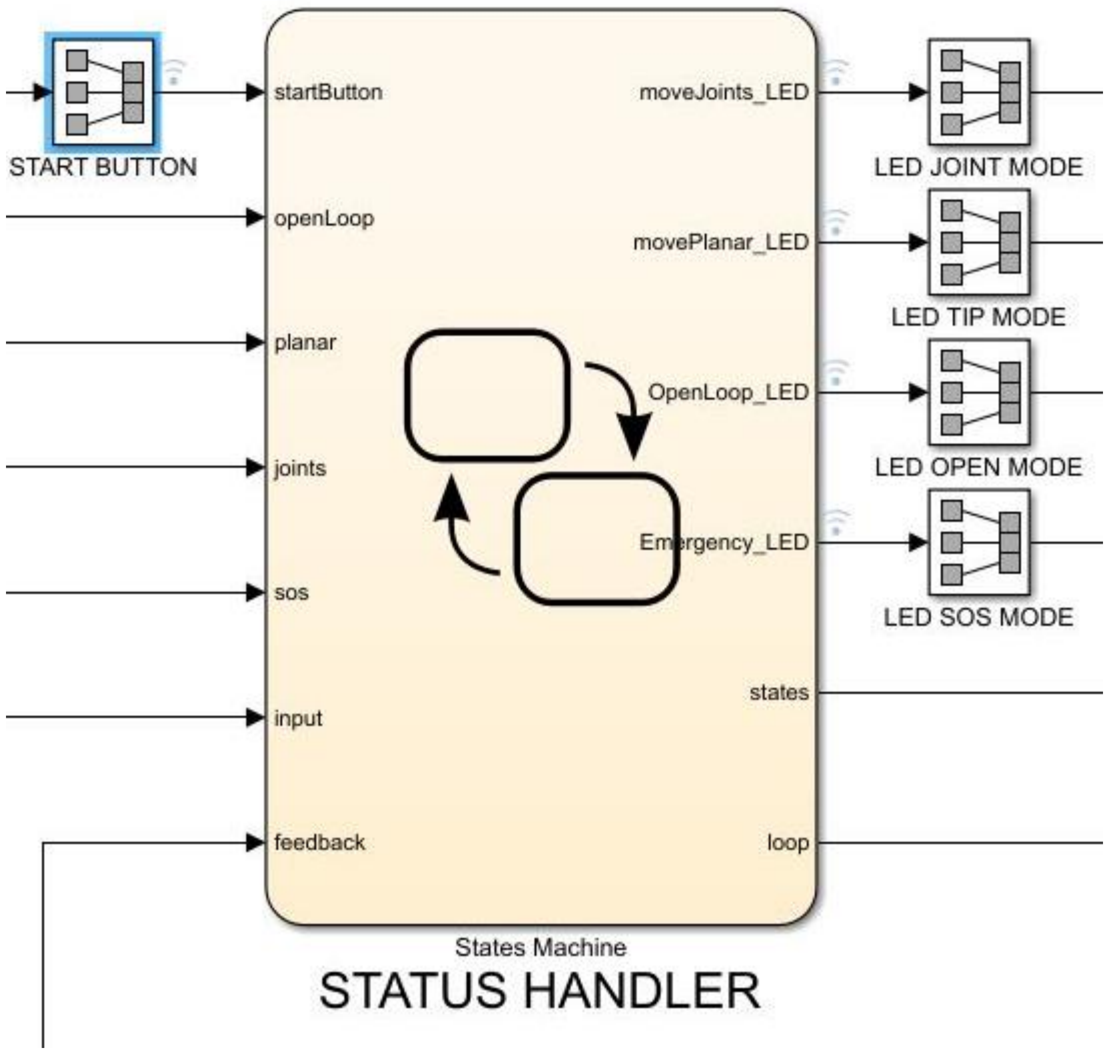
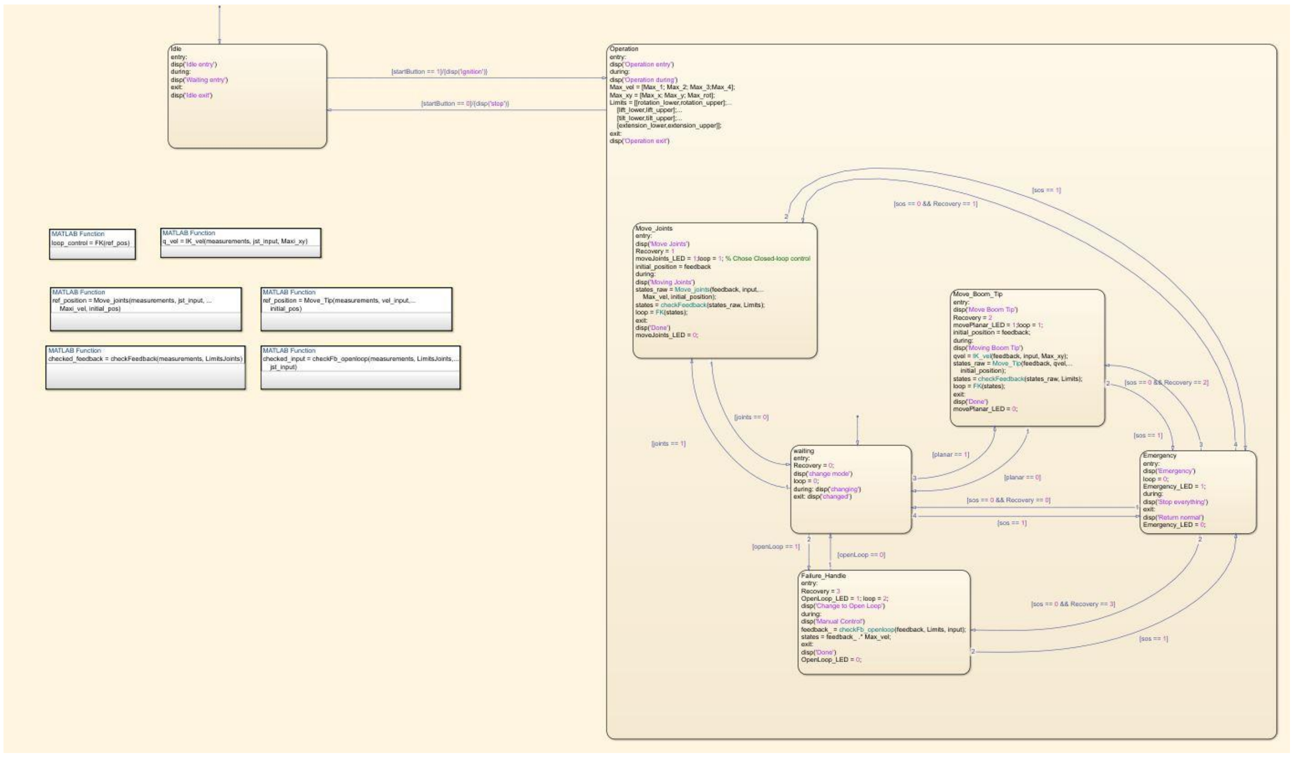


Figure 23. Overview of State Machine with inputs and outputs

State Machine receive signals from operator as which mode is chosen, joystick input and data from sensor to calculate the new states and loop for the Controller. “states” ouput is reference position and “loop” output is control type. In case Move Open Loop mode is chosen then “states” output is velocities from joystick commands.



```
Idle
entry:
disp('Idle entry')
during:
disp('Waiting entry')
exit:
disp('Idle exit')
```

```
Operation
entry:
disp('Operation entry')
during:
disp('Operation during')
Max_vel = [Max_1; Max_2; Max_3;Max_4];
Max_xy = [Max_x; Max_y; Max_rot];
Limits = [[rotation_lower,rotation_upper];...
   [lift_lower,lift_upper];...
   [tilt_lower,tilt_upper];...
   [extension_lower,extension_upper]];
exit:
disp('Operation exit')
```

```
Move_Joints
entry:
disp('Move Joints')
Recovery = 1
moveJoints_LED = 1;loop = 1; % Chose Closed-loop control
initial_position = feedback
during:
disp('Moving Joints')
states_raw = Move_joints(feedback, input,...
   Max_vel, initial_position);
states = checkFeedback(states_raw, Limits);
loop = FK(states);
exit:
disp('Done')
moveJoints_LED = 0;
```

```
Move_Boom_Tip
entry:
disp('Move Boom Tip')
Recovery = 2
movePlanar_LED = 1;loop = 1;
initial_position = feedback;
during:
disp('Moving Boom Tip')
qvel = IK_vel(feedback, input, Max_xy);
states_raw = Move_Tip(feedback, qvel,...
   initial_position);
states = checkFeedback(states_raw, Limits);
loop = FK(states);
exit:
disp('Done')
movePlanar_LED = 0;
```

```
waiting
entry:
Recovery = 0;
disp('change mode')
loop = 0;
during: disp('changing')
exit: disp('changed')
```

```
Emergency
entry:
disp('Emergency')
loop = 0;
Emergency_LED = 1;
during:
disp('Stop everything')
exit:
disp('Return normal')
Emergency_LED = 0;
```

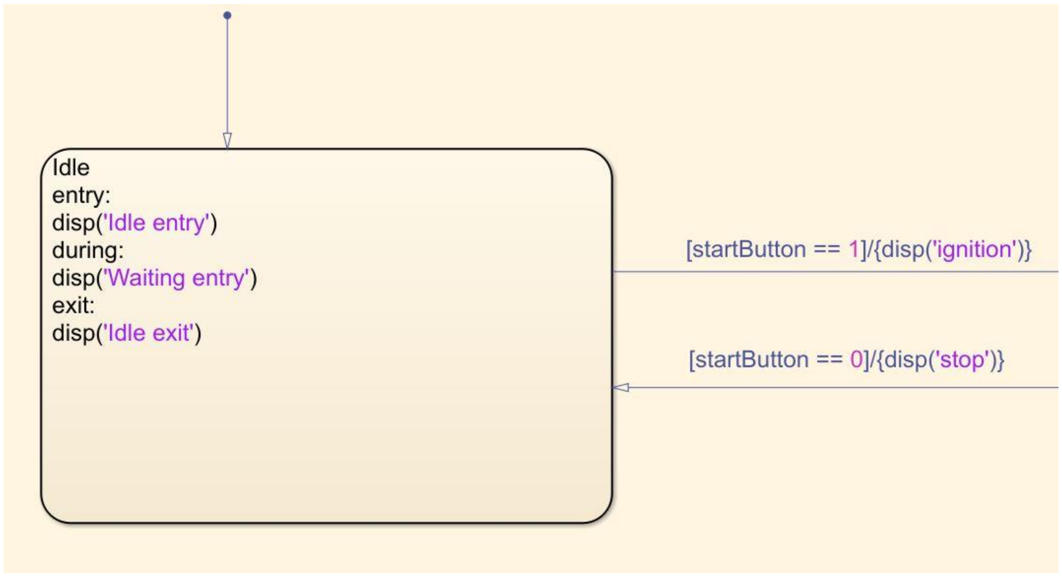
```
Failure_Handle
entry:
Recovery = 3
OpenLoop_LED = 1; loop = 2;
disp('Change to Open Loop')
during:
disp('Manual Control')
feedback_ = checkFb_openloop(feedback, Limits, input);
states = feedback_ .* Max_vel;
exit:
disp('Done')
OpenLoop_LED = 0;
```

```
MATLAB Function
loop_control = FK(ref_pos)

MATLAB Function
q_vel = IK_vel(measurements, jst_input, Maxi_xy)

MATLAB Function
ref_position = Move_joints(measurements, jst_input, ...
   Maxi_vel, initial_pos)

MATLAB Function
ref_position = Move_Tip(measurements, vel_input,...
   initial_pos)

MATLAB Function
checked_feedback = checkFeedback(measurements, LimitsJoints)

MATLAB Function
checked_input = checkFb_openloop(measurements, LimitsJoints,...
   jst_input)
```

[startButton == 1]/{disp('ignition')}
[startButton == 0]/{disp('stop')}
[sos == 1]
[sos == 0 && Recovery == 1]
[sos == 0 && Recovery == 2]
[sos == 1]
[joints == 0]
[joints == 1]
[planar == 1]
[planar == 0]
[sos == 0 && Recovery == 0]
[sos == 1]
[openLoop == 1]
[openLoop == 0]
[sos == 0 && Recovery == 3]
[sos == 1]

Figure 24. State Machine architecture

6.8.1 “Idle” state



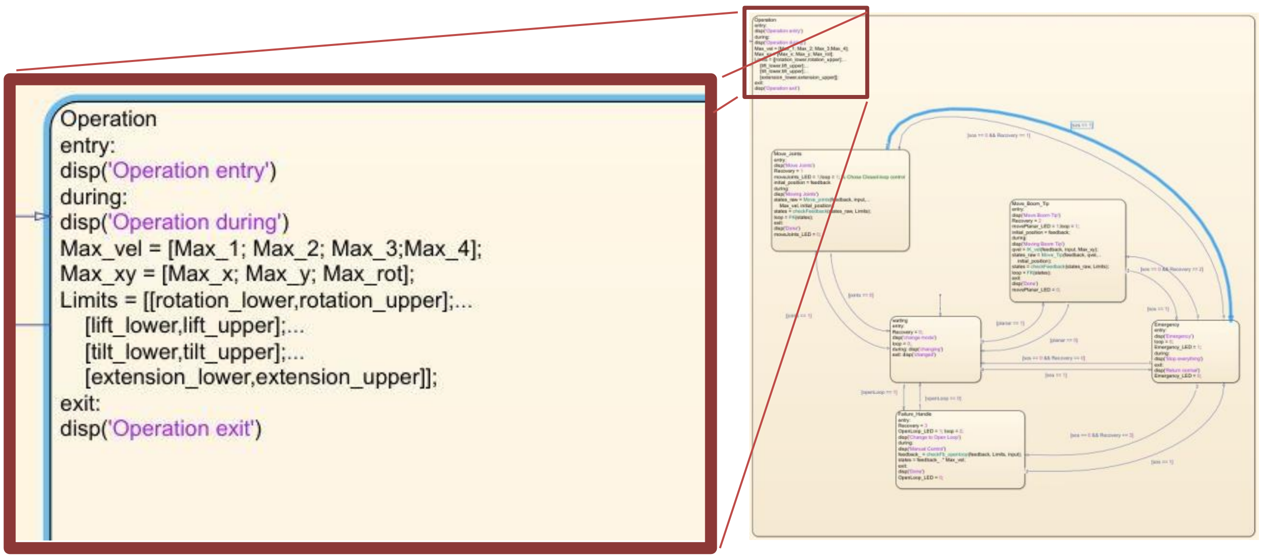
```
Idle
entry:
disp('Idle entry')
during:
disp('Waiting entry')
exit:
disp('Idle exit')
```

[startButton == 1]/{disp('ignition')}
[startButton == 0]/{disp('stop')}

Figure 25. "Idle" state.

This state runs from initial starting. If the “POWER ON” button is light-on then the State Machine will exit “Idle” state to move to “Operation” state and vice versa.

6.8.2 “Operation” state

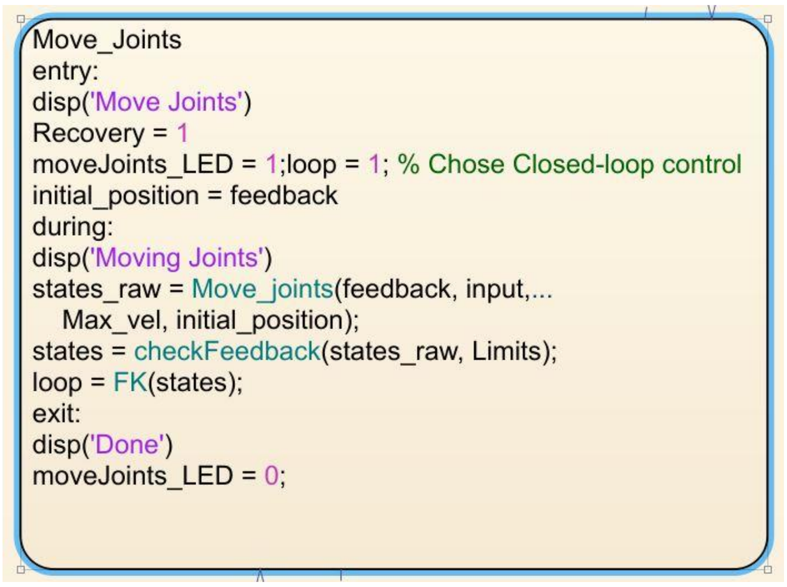


```
Operation
entry:
disp('Operation entry')
during:
disp('Operation during')
Max_vel = [Max_1; Max_2; Max_3;Max_4];
Max_xy = [Max_x; Max_y; Max_rot];
Limits = [[rotation_lower,rotation_upper];...
    [lift_lower,lift_upper];...
    [tilt_lower,tilt_upper];...
    [extension_lower,extension_upper]];
exit:
disp('Operation exit')
```

Figure 26. "Operation" state overview (right) and parameters updating in this state (left zoom image)

When this state is running, it keeps updating maximum velocities and soft limits which are editable by operator through GUI. Inside this state, there are five sub-states which are Move_Joints, Move_Boom_Tip, Move_Open_Loop, Emergency and Waiting. Initially, Waiting state is called and running when Operation state is on. The current state will move to new state whose signal is 1 first. While current state is not Waiting or Emergency, there is two ways to change state, move to Emergency then comeback to previous state when Emergency is off or move to Waiting and move to other states.

6.8.2.1 Move_Joints



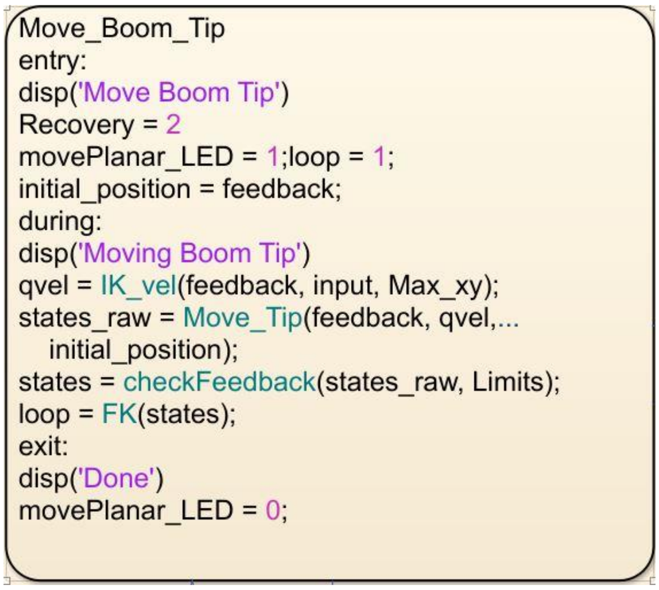
```
Move_Joints
entry:
disp('Move Joints')
Recovery = 1
moveJoints_LED = 1;loop = 1; % Chose Closed-loop control
initial_position = feedback
during:
disp('Moving Joints')
states_raw = Move_joints(feedback, input,...
    Max_vel, initial_position);
states = checkFeedback(states_raw, Limits);
loop = FK(states);
exit:
disp('Done')
moveJoints_LED = 0;
```

Figure 27. "Move_Joints" functionalities.

When this state is on, State Machine will set Move Joint mode LED light on and choose control type as Closed-loop control by set “loop” output 1. Besides, this state defines the initial position based on measured data from sensor. In additions, “Recovery” is set to 1, which is key for Emergency mode knows where to return after recovery.

When it is running, “Move_Joints” state calculates new reference position (“states_raw”) by “Move_joints” function. After that, it checks “states_raw” if it is overpass the soft limits by “checkFeedback” function and updates to “states” output the suitable reference position. To make sure it is not in collision area, “states” will be tested in “FK” function, which is modifed version of provided forward kinematics function, to get the control type. Although “states” is not overpass the soft limits, “states” will be not considered in the Controller if it move the TCP to collision area when “loop” output is set to 0.

6.8.2.2 Move_Boom_Tip

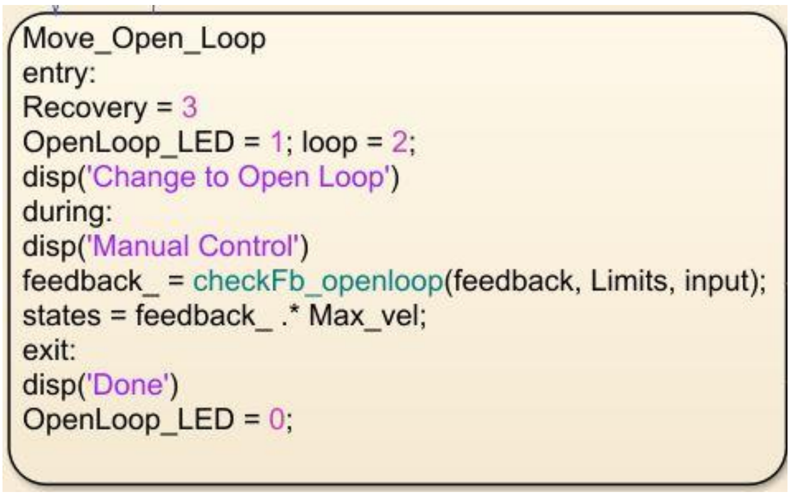


```
Move_Boom_Tip
entry:
disp('Move Boom Tip')
Recovery = 2
movePlanar_LED = 1;loop = 1;
initial_position = feedback;
during:
disp('Moving Boom Tip')
qvel = IK_vel(feedback, input, Max_xy);
states_raw = Move_Tip(feedback, qvel,...
   initial_position);
states = checkFeedback(states_raw, Limits);
loop = FK(states);
exit:
disp('Done')
movePlanar_LED = 0;
```

Figure 28. "Move_Boom_Tip" functionalities

As same as ”Move_Joints” state, entry action of this state also turn on the LED light in GUI, set initial position to current pose of the boom and ”Recovery” to key 2. When it is running, this state calculates ”qvel”, which is joints velocities, by provided function ”IK_vel”. New reference position is generated after that by ”Move_Tip” function. The new reference position alse checked with ”checkFeedback” function for acceptable reference position. The ”loop” output is calculated to make sure the boom will not follow new reference position if they guild it to collision area.

6.8.2.3 Move_Open_Loop

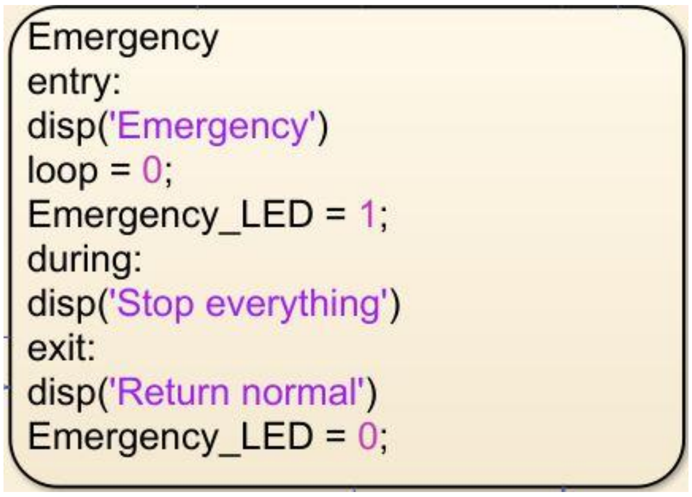


```
Move_Open_Loop
entry:
Recovery = 3
OpenLoop_LED = 1; loop = 2;
disp('Change to Open Loop')
during:
disp('Manual Control')
feedback_ = checkFb_openloop(feedback, Limits, input);
states = feedback_ .* Max_vel;
exit:
disp('Done')
OpenLoop_LED = 0;
```

Figure 29. "Move_Open_Loop" state

In this state, we use directly commands from joystick so it is needed to create new checking function to make sure the reference states will not overpass the soft limits. Besides, this state is not check the physical collision due to it could be use to guild the boom out of the collision area when there is some problems with the BCS and the TCP is in collision area. As same as two states above, we also designed this state to set parameters to its own key code to visualize in GUI and help Emergency mode work smoothly.

6.8.2.4 Emergency

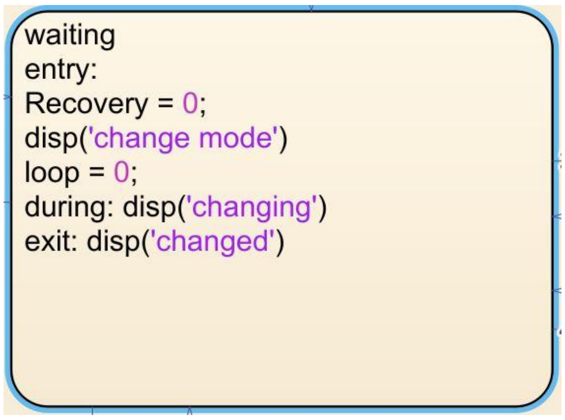


```
Emergency
entry:
disp('Emergency')
loop = 0;
Emergency_LED = 1;
during:
disp('Stop everything')
exit:
disp('Return normal')
Emergency_LED = 0;
```

Figure 30. "Emergency" state

When this state is called, it will turn on Emergency mode and stop the boom immediately by set LED light and ”loop” output to zero. This can be called from any sub-states in ”Operation” state.

6.8.2.5 Waiting



```
waiting
entry:
Recovery = 0;
disp('change mode')
loop = 0;
during: disp('changing')
exit: disp('changed')
```

Figure 31. "waiting" state is transition state between other sub-states.

7 Implementation and Testing

7.1 Off-line implementation

From starting, we divided the system into small parts and testing if each part is working as we expected. After that, we combined them together. There are some small problems with this phase when we were using Matlab 2018b where provided model is built in Matlab 2018a. We come through this phase nicely where whole system is working well with off-line implementation.

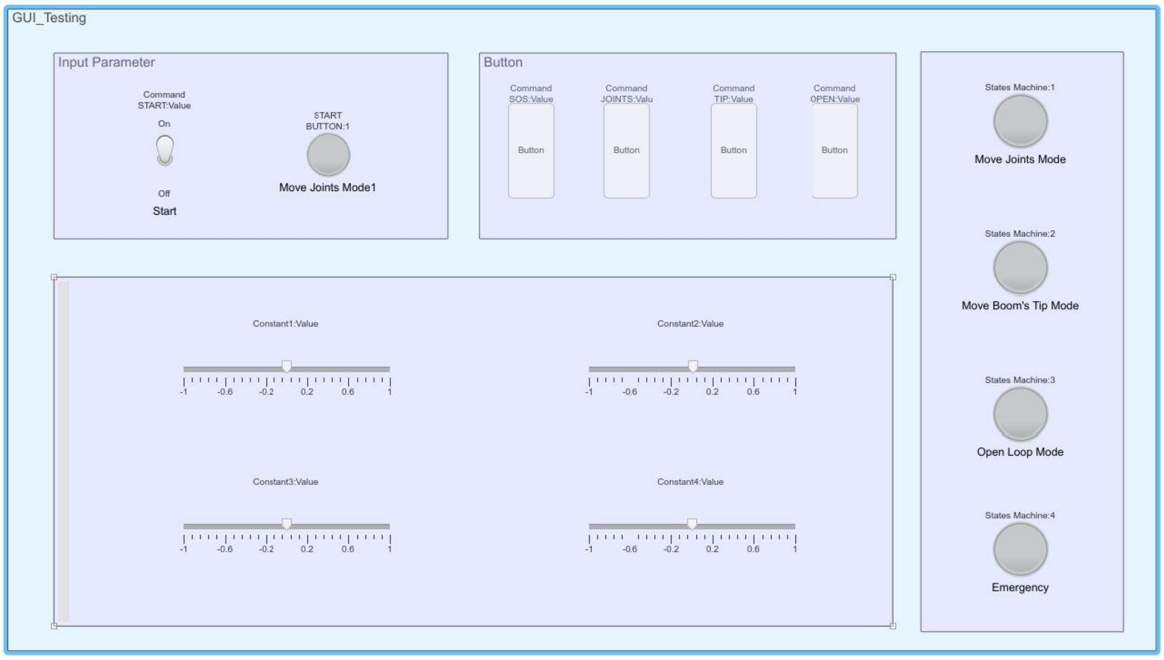


Figure 32. Testing GUI created in Off-line implementation

7.2 Real-time implementation

It took long time to test with Real-time implementation when the functions in "Operation" state are not working in real-time environment. We have to carefully check the input and output of each function by off-line testing and simple GUI display when the BCS is running to figure out where is the problem. Because the data flow is really fast and in some function, we considered the data as static variable then it causes the boom move in wrong way. Besides, we design to choose operating mode by joystick buttons and we have to figure out how to keep the keep command signal as switch-signal when command signal is button-signal, which means it automatically return to zero after releasing the button. We finally come up with self-made function which also cannot handle the really fast data flow in real-time environment, but somehow the BCS can work.

7.3 Demonstration

7.3.1 Inputs and Outputs

The buttons are not working well while they are still have problems mentioned in Real-time Implemetation. Operator has to press the button for a while until the system can catch the signal to turn on desired mode.

The outputs of the BCS is not working as expected when new modified GUI with editable parameters cannot connect to the system parameters. Besides, the mode LED is working, joints position and xy-position are presented well. Operator can modify speed of the boom through Controller Gains. ”Power On” button can show light-on, light-off value but it not turn off the whole system while the Controller still working and guild the boom. The GUI has no unit of data.

7.3.2 State Machine

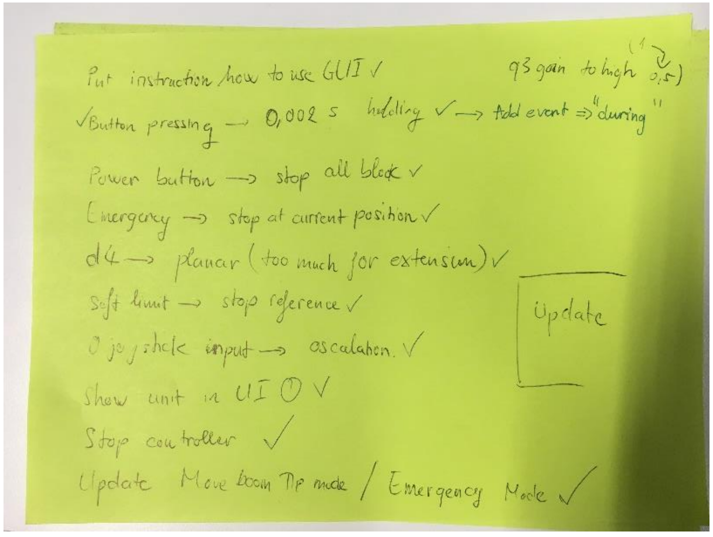
Due to mode choosing error, the State Machine shows new errors when other states can be chosen when Emergency mode activated. Emergency mode mode the boom to default position when it is called before stop working completely. When the boom reachs limits, it is oscalation although there is no input from joystick.

The Move Joints and Move Open Loop mode are working well. The Move Boom Tip mode is working when the boom can move in planar but the gain for extension is too high then the result of TCP is not correct.

7.4 Modifications after the demonstration

7.4.1 General

Fixing the feedback from demonstration. Updating the BCS model with more accuracy.



Put instruction how to use GUI ✓ q3 gain to high (1 → 0,5)

✓Button pressing → 0,002 s holding ✓ → Add event ⇒ "during"

Power button → stop all block ✓

Emergency → stop at current position ✓

d4 → planar (too much for extension) ✓

Soft limit → stop reference ✓

0 joystick input → oscalation. ✓

Show unit in UI ✓

Stop controller ✓

Update Move boom Tip mode / Emergency Mode ✓

Update

Figure 33. Note from demonstration

7.4.2 Details

1. Modify provided ”FK” function to new collision handling ”FK” function.
2. Update GUI with COLLISION LED light.
3. Update GUI, change Following Errors scopes into numeric displays.
4. Modify ”Ignition.m” script, delete unnessesary data.
5. Update GUI with joints position numeric displays.
6. Update GUI with Max. Velocities Planar Mode box.
7. Modify maximum velocities and soft limits data structure to bind them in GUI.
8. Create overpass checking function in Move_Open_Loop state.
9. Modify ”Move_joint” and ”Move_Tip” functions, delete unnessesary lines.
10. Update ”IK_vel” function with new d4 gain.
11. Update State Machine, move Max_vel, Max_xy, Limits parameters from ”Idle” state to ”during” action in ”Operation” state.
12. Create ”Button Input Handler” block to handle joystick button signal in real-time environment.
13. Update ”Convert” function to convert trigger signal from joystick to State Machine command.
14. Add collision detection in the Controller.
15. Update maximum desired velocities in planar mode.
16. Set new physical edges in the BCS to avoid collision.

8 Future development

In future work, we aim to make the system work more accuracy in Move Boom Tip mode and collision detection. The Controller is using P gains which could be updated.

Appendix 1. Self-evaluation

Our group has actively worked since starting when we set up schedule for group meetings, got milestones of each period of this exercise due to we separated the work into pieces then every members have a room to grown. Because of suitable schedule and assignments, we have completed almost the requirements of exercise.

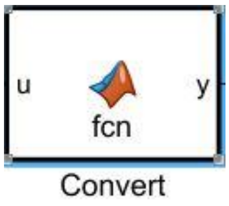
1. Toan Le: main role in model building and implementation. Has responsibility for Uses Cases, Requirements, State Machine, collision detection, real-time system architecture. In my opinion, I think the first two phases of this exercise has same level as they are 2/5 difficulty. The off-line design and testing is more difficult but we could handle it so I count it as 3/5. Finally, the big boss is real-time implementation when some small changes could lead the whole system work in unexpected way and sometimes it is hard to keep track with the running system. I count the real-time implementation as 5/5 difficulty and 5/5 interesting as it is good feeling when the system could work somehow correctly as expected.

2. Pallab Ganguly: Has responsibility for Requirements, Controller, real-time implementation, Sensor Model.

Appendix 2. Feedback

1. The real-time stations are slow.
2. Intersting exercise.

Appendix 3. Matlab Function code



```
function y = fcn(u)
% This function is triggered by joystick button signal. When the
function is
% on, it checks the status of Emergency mode. If Emergency mode
is off, it
% set its status on
%
% input:
%     u - status of Emergency mode (1 - on, 0 - off)
% output:
%     y - status of this mode

persistent count

% Set initial status
if isempty(count)
    count = 0;
end
% Check condition and set suitable status
if u == 0
    if count == 0
        count = 1;
    else
        count = 0;
    end
end
% Return the result
y = count;
end
```

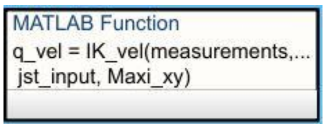
y
fcn
Convert_SoS

```
function y = fcn()
% This function is triggered by joystick button signal to turn on
Emergency
% mode
% output:
%     y - status of this mode

persistent count
% Set initial status
if isempty(count)
    count = 0;
end
% Set suitable status
if count == 0
    count = 1;
else
    count = 0;
end
% Return the result
y = count;
end
```

MATLAB Function
loop_control = FK(ref_pos)

```
function loop_control = FK(ref_pos)
% This function transforms measured joint positions to
the position of the
% Tool Center Point. In this case, the TCP is located
at the end of the
% boom, after the telesopic joint and before the
passive joint. After that,
% the function checks if TCP position over-pass the
collision space so it
% will return the suitable control loop
%
% The first joint angle, q1, has no effect on the
kinematics, so it is not
% a part of this function.
%
% Inputs:
%   q2,q3,d4     - Current joint positions [deg,m]
%
% Locals:
%   x,y          - Current TCP position position [m]
%
% Outputs:
%   loop_control - suitable control type for reference
position
%
%


% Modify input joint position
q2 = ref_pos(2);
q3 = ref_pos(3);
d4 = ref_pos(4)/1000;

% Unit conversion to radians and meters
deg_to_rad = pi/180;
q2 = q2*deg_to_rad;
q3 = q3*deg_to_rad;

% Calculate the forward kinematics
t1 = cos(q2);
t2 = q3 + pi / 0.2e1;
t3 = cos(t2);
t4 = sin(q2);
t2 = sin(t2);
t5 = t1 * t2 + t3 * t4;
t6 = d4 + 0.213e1;
t7 = 0.376348799999999983e0;
t8 = 0.350440540000000000e1;
t2 = t1 * t3 - t2 * t4;
t3 = 0.160450000000000004e1;
XY_kin = [t1 * t8 + t5 * t6 + t7 * t2 - ...
    0.208160000000000012e0 -t2 * t6 + t4 * t8 + t5 *
t7 + t3];

% Write results to outputs (in meters)
x = XY_kin(1);
y = XY_kin(2);
```

```
% Collision limitations
x_max = 1.2;
y_min = -0.64;

% Check collision limitation over-pass
collision = 0;
if x < x_max || y < y_min
    collision = 1;
end

% Generate control type
if collision == 1
    loop_control = 0;
else
    loop_control = 1;
end
```



```
function q_vel = IK_vel(measurements, jst_input,
Maxi_xy)
% This function transforms desired coordinate
velocities to desired joint
% velocities in such a way that the joints move as
little as possible. To
% do this, the function needs to also know the current
pose (joint
% positions) of the manipulator.
%
% Notice that the output is a set of desired joint
velocities, not
% absolute joint positions. If you know the discrete
time step you are
% using, you can integrate those velocities into a
change of position.
%
% Likewise, the inputs xvel & yvel are desired
coordinate velocities, not
% desired absolute coordinate positions. If you want
to explicitly control
% coordinate positions, you can calculate the time
derivative of your
% desired coordinate positions to get the desired
velocities.
%
% The first joint angle, q1, has no effect on the
kinematics, so it is not
% a part of this function.
% Inputs:
%   measurements        - feedback data of joints
position
%   jst_input           - joystick input signals
%   Maxi_xy             - Desired Maximum velocities
in planar mode
%
%  Outputs:
%   qvel                - Joints velocities
%
% Locals:
```

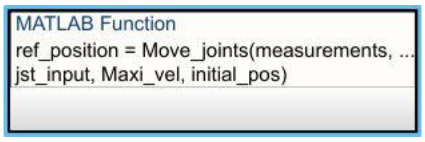
```
%   q2,q3,d4            - Current joint position
references
%                         e.g. from last iteration
[deg,m]
%   xvel,yvel           - Desired coordinate
velocities [m/s]
%
%   q2vel,q3vel,d4vel   - Desired joint velocities
[deg/s,m/s]
%

% Modify input joint position
q2 = measurements(2);
q3 = measurements(3);
d4 = measurements(4)/1000;

% Modify input xy velocities
xvel = Maxi_xy(1) * jst_input(1);
yvel = Maxi_xy(2) * jst_input(2);
q1vel = Maxi_xy(3) * jst_input(3);

% Unit conversion to radians and meters
deg_to_rad = pi/180;
rad_to_deg = 180/pi;
q2 = q2*deg_to_rad;
q3 = q3*deg_to_rad;

% Calculate the [3*2] Jacobian pseudoinverse
t1 = sin(q2);
t2 = q3 + pi / 0.2e1;
t3 = cos(t2);
t4 = cos(q2);
t2 = sin(t2);
t5 = -t1 * t2 + t3 * t4;
t6 = d4 + 0.213e1;
t7 = t4 * t2;
t8 = t1 * t3;
t9 = t7 + t8;
t10 = 0.35044054000000000e1;
t11 = t9 * 0.37634879999999983e0;
t12 = t5 * t6;
t13 = -t1 * t10 - t11 + t12;
t14 = t1 ^ 2;
t15 = t14 ^ 2;
t16 = t4 ^ 2;
t17 = t16 ^ 2;
t18 = t3 ^ 2;
t19 = t18 ^ 2;
t20 = t7 * t1;
t21 = t1 * t18 * t4;
t22 = d4 * (t2 * (t3 * (-t16 + t14) + t20) - t21);
t23 = t4 * t9;
t24 = t2 ^ 2;
t25 = t24 ^ 2;
t26 = t14 * t18 + t16 * t24;
t27 = d4 ^ 2;
t28 = -t16 + t14;
```

```
t29 = t3 * t28;
t30 = t2 * (t20 + t29) - t21;
t31 = t14 * t24 + t16 * t18;
t32 = 0.1492876700e2;
t33 = 0.7008810800e1;
t34 = 0.2637757534e1;
t35 = 0.1283276839e1;
t36 = 0.8520000000e1;
t37 = 0.9073800000e1;
t38 = (0.4e1 * d4 + 0.1704000000e2) * d4;
t39 = 0.1505395200e1 * t22;
t40 = 0.3206491776e1 * t30;
t8 = d4 * t23 * t33 + d4 * t26 * t36 + t34 * t4 * t5 +
t23 * t32 + 0.2e1 * t27 * t26 + t26 * t37 + t31 * t35
+ t4 * (0.1228085721e2 * t4 + t8 * t2 * (t38 +
0.1558104632e2)) - t39 - t40;
t23 = t18 + t24;
t23 = t2 * (t15 * t23 + t17 * t23);
t41 = t18 + t24;
t42 = t14 * t16;
t43 = t42 * t2;
t44 = t17 + t15;
t45 = t3 * t2;
t46 = t42 * t24;
t47 = t42 * t19;
t48 = t24 * (t18 * (t17 + t15) + t46) + t47;
t49 = t18 * t44;
t50 = t19 + t25;
t50 = t15 * t50 + t17 * t50;
t19 = t19 + t25;
t25 = 0.9073800003e1;
t51 = 0.1814760000e2;
t15 = d4 * t23 * t33 + 0.1401762160e2 * t43 * d4 * t41
- 0.9243771744e1 * t45 * d4 * t44 + 0.2985753400e2 *
t43 * t41 + 0.5231645170e2 * t49 * d4 - 0.1968923382e2
* t45 * t44 + 0.1228085721e2 * t49 * t27 + t51 * (t24
* (t46 + t49) + t47) + 0.8519999996e1 * d4 * t50 + t23
* t32 + t25 * (t15 * t19 + t17 * t19) + 0.1402029841e2
* t24 * t44 + 0.5571702104e2 * t49 + 0.1704000000e2 *
d4 * t48 + 0.4e1 * t27 * t48 + 0.2e1 * t27 * t50 + t42
* (t3 * (t2 * (-0.1848754340e2 * d4 - 0.3937846770e2)
+ t3 * (((0.8e1 * t24 + 0.2456171442e2) * d4 +
0.3408000000e2 * t24 + 0.1046329034e3) * d4 +
0.3629520000e2 * t24 + 0.1114340422e3)) +
0.2804059680e2 * t24);
t17 = t5 * 0.37634879999999999983e0;
t6 = t9 * t6;
t10 = t10 * t4 + t17 + t6;
t19 = t18 - t24;
t23 = 0.1603245888e1;
t2 = t22 * t36 + 0.7790523161e1 * t30 + 0.3504405400e1
* t29 * d4 - 0.7526976000e0 * d4 * (t14 * (-t18 + t24)
+ t16 * (t18 - t24)) + 0.7464383502e1 * t29 -
0.1318878767e1 * t2 * t28 + t23 * (t14 * t19 - t16 *
t19) - 0.2e1 * t27 * (t2 * (t3 * (t16 - t14) - t20) +
t21) + t1 * t4 * (t2 * (d4 * t33 + t3 *
(0.3010790400e1 * d4 + 0.6412983552e1) + t32) + t3 *
t34 + 0.1228085721e2);
```

<table>
<tr><td></td><td>

```
t4 = 0.1e1 / t15;
t14 = t1 * t5;
t1 = -t33 * t14 * d4 + t36 * d4 * t31 + t34 * t1 * t9
+ t1 * (0.1228085721e2 * t1 + t7 * t3 * (-t38 -
0.1558104632e2)) - t14 * t32 + t26 * t35 + 0.2e1 * t27
* t31 + t31 * t37 + t39 + t40;
t3 = t12 - t11;
t6 = t6 + t17;
Jacob_pinv = [t4 * (t10 * t2 + t13 * t8) t4 * (t1 *
t10 + t13 * t2); t4 * (t2 * t6 + t3 * t8) t4 * (t1 *
t6 + t3 * t2); t4 * (-t2 * t5 + t8 * t9) t4 * (-t1 *
t5 + t2 * t9);];

% Use the pseudoinverse to calculate the desired joint
velocities
IK_result = Jacob_pinv * [xvel yvel]';

% Write results to outputs, with unit conversion to
deg/s and m/s
q2vel = IK_result(1) * rad_to_deg;
q3vel = IK_result(2) * rad_to_deg;
d4vel = IK_result(3);

% Limit output to prevent infinity values
if abs(q2vel) > 1000
    q2vel = sign(q2vel)*1000;
end
if abs(q3vel) > 1000
    q3vel = sign(q3vel)*1000;
end
if abs(d4vel) > 10000
    d4vel = sign(d4vel)*1000;
end
% Update result
q_vel = [q1vel;q2vel;q3vel;d4vel*100];
end
```

</td></tr>
<tr><td>

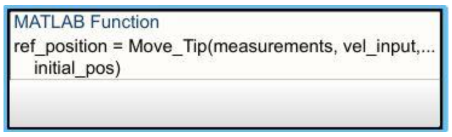

MATLAB Function
ref_position = Move_joints(measurements, ...
jst_input, Maxi_vel, initial_pos)

</td><td>

```
function ref_position = Move_joints(measurements,
jst_input, ...
    Maxi_vel, initial_pos)
% This function recieves signals from joystick and
calculate new referenced
% position.
% Parameters and result:
%   "measurements": feedback data of the boom position
- matrix 4x1
%   "jst_input": command signal from joystick - matrix
4x1
%   "Maxi_vel": Maximum velocity of joints - matrix
4x1
%   "initial_pos": initial position of the boom when
this state is called
%   "ref_position": result of this function, which is
referneced position of
%                    the boom - matrix 4x1

persistent current_position old_initial
```

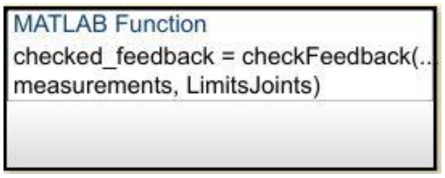
</td></tr>
</table>

```
step = jst_input .* Maxi_vel;

% If current position is null then set current
position is measured data.
if isempty(current_position)
    current_position = initial_pos;
    old_initial = initial_pos;
end

% Keep the boom position after changing states
if initial_pos ~= old_initial
    current_position = initial_pos;
    old_initial = initial_pos;
end

if sum(jst_input) ~= 0
    % Update current position with command from
joystick
    current_position = measurements + step;
end

% Keep the boom stand still
ref_position = current_position;

end
```

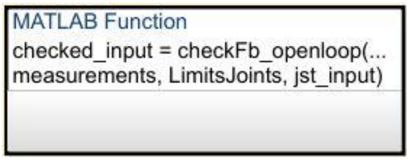

MATLAB Function
ref_position = Move_Tip(measurements, vel_input,...
initial_pos)

```
function ref_position = Move_Tip(measurements,
vel_input,...
    initial_pos)
% This function recieves signals from joystick and
calculate new referenced
% position as planar movement of the boom.
% Parameters and result:
%   "measurements": feedback data of the boom position
- matrix 4x1
%   "vel_input": joints' velocities - matrix 4x1
%   "initial_pos": initial position of the boom when
this state is called
%   "ref_position": result of this function, which is
referenced position of
%                    the boom - matrix 4x1

persistent tcp_position old_initial
travel = vel_input;

% If current position is null then set current
position is measured data.
if isempty(tcp_position)
    tcp_position = initial_pos;
    old_initial = initial_pos;
end

% Keep the boom position after changing states
if initial_pos ~= old_initial
    tcp_position = initial_pos;
    old_initial = initial_pos;
```

```
end

% Update current position with command from joystick
if sum(travel) ~= 0
    tcp_position = measurements + travel;
end

% Keep the boom stand still
ref_position = tcp_position;

end
```


MATLAB Function
checked_feedback = checkFeedback(..
measurements, LimitsJoints)

```
function checked_feedback =
checkFeedback(measurements, LimitsJoints)
% This function checks the referenced positions if
they are overpass the
% limits then scales them between the limits.
% Parameters and result:
%   "measurements": referenced positions - matrix 4x1
%   "LimitsJoints": joints' limitations - matrix 4x2
%   "checked_feedback": result of this function, which
is referenced
%                        positions of the boom - matrix
4x1

 q1 = measurements(1);
 q2 = measurements(2);
 q3 = measurements(3);
 d4 = measurements(4);

 lim1 = LimitsJoints(1,:);
 lim2 = LimitsJoints(2,:);
 lim3 = LimitsJoints(3,:);
 lim4 = LimitsJoints(4,:);

 % Upper Limits over passed
 if q1>lim1(:,2)
     q1 = lim1(1,2) - 1;
 end
 if q2>lim2(:,2)
     q2 = lim2(1,2) - 1;
 end
 if q3>lim3(:,2)
     q3 = lim3(1,2) - 1;
 end
 if d4>lim4(:,2)
     d4 = lim4(1,2) - 1;
 end

%  Lower Limits over passed
 if q1<lim1(:,1)
     q1 = lim1(1,1) + 1;
 end
 if q2<lim2(:,1)
     q2 = lim2(1,1) + 1;
 end
 if q3<lim3(:,1)
     q3 = lim3(1,1) + 1;
 end
 if d4<lim4(:,1)
     d4 = lim4(1,1) + 1;
 end
 % Update result
 checked_feedback = [q1; q2; q3; d4];

end
```


MATLAB Function
checked_input = checkFb_openloop(...
measurements, LimitsJoints, jst_input)

```
function checked_input =
checkFb_openloop(measurements, LimitsJoints,...
    jst_input)
% This function checks the referenced positions if
they are overpass the
% limits then scales them between the limits.
% Parameters and result:
%   "measurements": referenced positions - matrix 4x1
%   "LimitsJoints": joints' limitations - matrix 4x2
%   "checked_feedback": result of this function, which
is referenced
%                        positions of the boom - matrix
4x1

 q1 = measurements(1);
 q2 = measurements(2);
 q3 = measurements(3);
 d4 = measurements(4);

 u1 = jst_input(1);
 u2 = jst_input(2);
 u3 = jst_input(3);
 u4 = jst_input(4);

 lim1 = LimitsJoints(1,:);
 lim2 = LimitsJoints(2,:);
 lim3 = LimitsJoints(3,:);
 lim4 = LimitsJoints(4,:);

 % Upper Limits over passed
 if q1> (lim1(:,2) - 1)
     u1 = 0;
 end
 if q2> (lim2(:,2) - 1)
     u2 = 0;
 end
 if q3> (lim3(:,2) - 1)
     u3 = 0;
 end
 if d4> (lim4(:,2) - 1)
     u4 = 0;
 end

%  Lower Limits over passed
 if q1< (lim1(:,1) + 1)
     u1 = 0;
 end
 if q2< (lim2(:,1) + 1)
     u2 = 0;
 end
 if q3< (lim3(:,1) + 1)
     u3 = 0;
 end
 if d4< (lim4(:,1) + 1)
     u4 = 0;
 end

 checked_input = [u1; u2; u3; u4];

end
```

ref_pos
FK_modified
xy_pos
Convert to xy coordinates

```
function xy_pos = FK_modified(ref_pos)
% This function transforms measured joint positions to
the position of the
% Tool Center Point. In this case, the TCP is located
at the end of the
% boom, after the telesopic joint and before the
passive joint.
%
% The first joint angle, q1, has no effect on the
kinematics, so it is not
% a part of this function.
%
% Inputs:
%   ref_pos         - Current joint positions [deg,m] -
matrix 4x1
%
% Outputs:
%   xy_pos          - Current TCP position positions
[m] - matrix 2x1
%


% Modify input joint position
q2 = ref_pos(2);
q3 = ref_pos(3);
d4 = ref_pos(4)/1000;

% Unit conversion to radians and meters
deg_to_rad = pi/180;
q2 = q2*deg_to_rad;
q3 = q3*deg_to_rad;

% Calculate the forward kinematics
t1 = cos(q2);
t2 = q3 + pi / 0.2e1;
t3 = cos(t2);
t4 = sin(q2);
t2 = sin(t2);
t5 = t1 * t2 + t3 * t4;
t6 = d4 + 0.213e1;
t7 = 0.376348799999999983e0;
t8 = 0.350440540000000000e1;
t2 = t1 * t3 - t2 * t4;
```

	```
	t3 = 0.16045000000000000004e1;
	XY_kin = [t1 * t8 + t5 * t6 + t7 * t2 -
	0.20816000000000000012e0 -t2 * t6 + t4 * t8 + t5 * t7 +
	t3];

	% Write results to outputs (in meters)
	x = XY_kin(1);
	y = XY_kin(2);

	xy_pos = [x;y];
	end
	```

position
boundingbox
collision Checking Collision
	```
	function collision = boundingbox(position)
	% This function checks if input planar position guilds
	the TCP ovepass the
	% edges of collision area.
	% Input:
	%     position       - reference position - matrix
	2x1
	% Output:
	%     collision      - Collision status (1 - true, 0
	- false)

	% Collision edges
	x_max = 1.2;
	y_min = -0.64;

	% Initial value
	collision = 0;
	% Check collision area overpass
	if position(1) < x_max || position(2) < y_min
	    collision = 1;
	end

	end
	```