Development of a C/C++ Interface for a Balancing Arm System

Nima Soroush, Kiran Roy

June 19, 2013

Supervised by Dr. Guillaume Ducard

Abstract

We present the design of an Interface and control strategies on the framework for the actual implementation of one degree balancing arm system. In our framework, hardware and software are integrated together to realize the hardware-in-the-loop simulation. Real-time hardware-in-the-loop simulation is one of the most effective methods for the verification of the overall control performance.

Balancing arm system is a pre-designed one degree balancing arm system, connected over sensor, micro controller and simulator. Simulator is a software, designed and programmed on C++ interface.The simulator calculates and simulates the arm movement and also trace important behaviours of the one degree balancing arm Ssstem.

The idea is to satisfy following two main concepts:

- To observe the system behaviour in GUI (without connecting between simulator and system)
- To trace the behaviour of the system in the real-time when simulator and the system are connected.

Contents

Chapter 1

Introduction

Our supervisor Dr. Guillaume Ducard has constructed the Balancing Arm System, shown in figure1. This is utilized for receiving of real time data and compare those data to a virtual model. Our missions are to track the angle with respect to thrust and replicate as well into the graph.
This system is designed to show the difference between real-time simulation with virtual simulation of one degree moving arm.

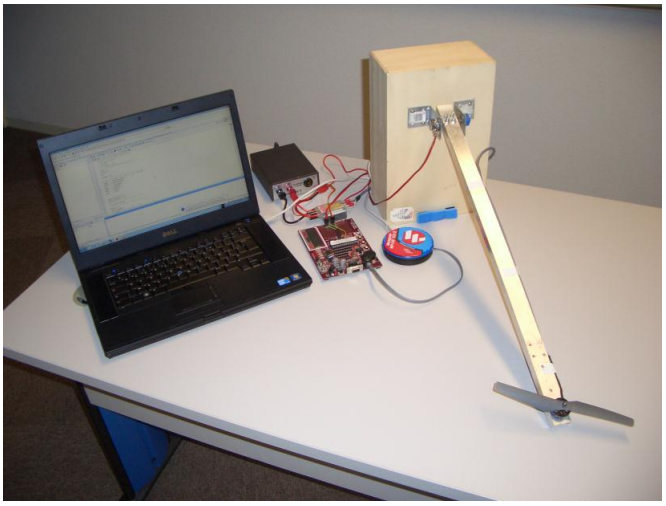


figure1: Balancing Arm System

The given one degree balancing arm system has three parts - base, sidebar and an arm. The sidebar is at one end of the base and the sidebar can be rotated along its own axis. At the top of the sidebar, there is the aluminium arm. At the end of the arm, there is a plastic propeller. The propeller helps the arm to actuate. At This balancing arm system is connected with a 8 pin micro-controller PIC Board The angle of the sidebar and arm is measured by a linear circular potentiometer

which is at the joint of the sidebar and the arm. The speed of the propeller is controlled by a dedicated motor controller. The communication between our GUI and the PIC board is established over usb serial communicator.

1.1 Project Goal

The potentiometer takes the voltage in the range of 0-5v and send to PIC board. In the board there is analogue to digital converter and converts the 0-5v into 0-1023. The PIC board then sends the value to the GUI and we mapped that value in a time vs θ graph. The motor-controller also sends the PWM (pulse with modulation) signal to the PIC board and the PIC board then convert that signal and sends that to the GUI.

- To develop framework for:

  the acquisition of the measurement data coming from the balancing arm system over a serial port

  to animate a 3D simulator of the system.

- Implement an user Interface where some physical data are being displayed on the screen such as the arm angle, the propeller speed and thrust.

- One object is to perform hardware in-the-loop simulation, meaning that the C++ simulator will be used to test some code implemented on an external micro controller.

- Document the architecture of the overall framework/interface.

Here is the graphics view of the real system.

1.2 Model View

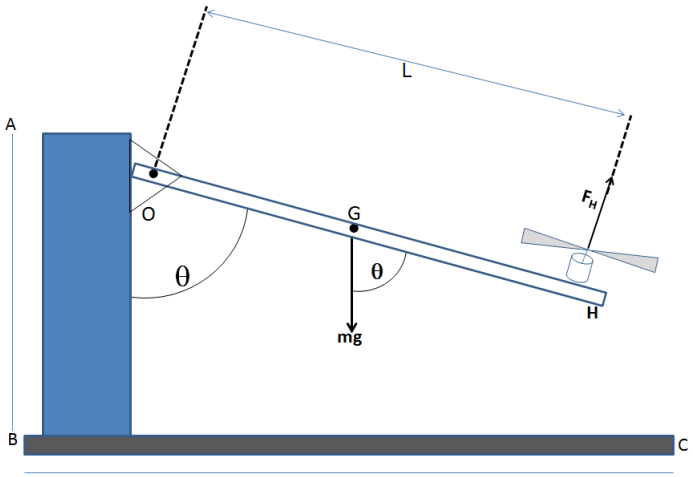


figure2: Balancing Arm System

- AB = Sidebar
- BC = Base
- OH = Arm (L)
- F_H = Thrust of the propeller
- g = gravity (9.81 m/s^2)
- θ = Angle

1.3 Working Principle of propeller

The propeller "propels" the bar or makes it move. It does this in much the same way that the air-plane's wing produces lift, only instead of being pushed forward through the air, it is spun in circles through the air.
A helicopter flies by means of the thrust that is created by the rotation of the blades of a main rotor. As the blades rotate, an airflow is created over them, resulting in lift. This raises the helicopter.
Thrust is produced by the expulsion of a reaction mass. In an optimum situation (see below), thrust equals the product of the mass expelled from the propeller force in unit time (the propellant mass flow rate) and the velocity .
If F is the thrust, m_p the propellant flow rate, and V_e the effective velocity, then **F** = m_p V_e

In our project the thrust is obtained by rotation of the propeller according to F = $\mu.\Omega^2$, where Ω^2 = propeller speed in rad/s and μ = lifting coefficient.

1.4 Computation of θ

θ is the function of J_z, L, F_H, where

- J_z = moment of inertia (m^2kg)
- L = length of the arm
- F_H = propeller thrust
- $\vec{OG}$ =$\vec{OH}$ = $\frac{L}{2}$

To compute the angle θ, we need to compute $\ddot{\theta}$ and $\dot{\theta}$.

$\dot{\theta}$ and $\ddot{\theta}$ are first derivative and second derivative of angle θ respectively.

From the fundamental theorem of dynamics for system in rotation - we derived the following equations

$J_{\vec{z}}\ddot{\theta} = \vec{OG} \wedge \vec{mg} + \vec{OH} \wedge \vec{F_H}$
$J_{\vec{z}}\ddot{\theta}$ = -$\vec{OG}$.mgsinθ + L.$\vec{F_H}$
$\therefore \ddot{\theta} = \frac{[L\vec{F_H} - \vec{OG}.mgsin\theta]}{J_z}$
$\ddot{\theta}$(k) =[$\dot{\theta}$(k) - $\dot{\theta}$(k-1)]/ΔT [k= time]
$\ddot{\theta}$(k) = [θ(k) - θ(k-1) - θ(k-1) - θ(k-2)]/ΔT^2
$\ddot{\theta}$(k) = [θ(k) - 2θ(k-1) + θ(k-2)]/$(\Delta T)^2$

$\dot{\theta}$(k) = $\ddot{\theta}$(k)ΔT + $\dot{\theta}$(k-1)
θ(k) = $\dot{\theta}$(k)ΔT + θ(k-1)

Chapter 2

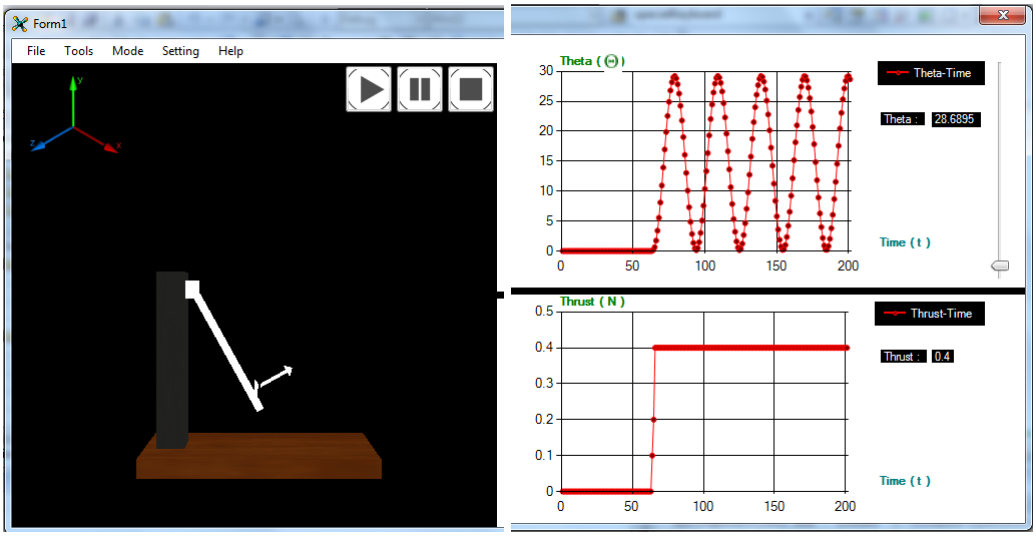
Technology

The user interface is designed with C++ language over visual studio on windows form application. The simulation designed on OpenGL with C++ language.
The technology behind the user interface divided into two parts. First part that is to implement the propeller equation done with C++ language using Microsoft Visual Studio and second part corresponds to arm simulator based on OpenGL basic C++ language.
For the first part we used the windows form application and add chart component. The chart component gives the diagram in graph form of angle (θ) vs thrust of propeller.
For the 3D arm simulator we used OpenGL component for the visual studio that is called OpenCS . For calculation of angle (θ), we send the value to OpenGL component and we render the arm position.
The 3D arm drawing with the angle vs time and thrust vs time graph

2.1 User Interface

The GUI has tow parts. Left is showing the virtual model of our balancing arm system and right is showing the graphs of Angle vs Time and Thrust vs Time.
In our UI,we have file, tools, mode, setting, helps.
In file option, user will able to save the graphs in .png format and also able to open file. In mode option We have different modes of simulation(Described in chapter 4). In tools, we have start, stop, pause, restart mode. In help, there is details about the UI software.

Below is the picture of our implemented GUI.



(a) GUI (b) Graphs

Chapter 3

Serial Communication

The word serial means "one after the other". Serial data transfer means when we transfer data one bit at a time, one right after the other. Information is passed back and forth between the computer and the other device, essentially, setting a pin high or low.

Serial communication is often used either to control or to receive data from an embedded microprocessor. Serial communication is a form of I/O in which the bits of a byte is transferred one after the other in a timed sequence on a single wire.

Serial communication enables different equipments to communicate with their outside world. Data bits are sent in a serial way over a single line. A personal computer has a serial port known as communication port or COM Port used to connect a modem(for example) or any other device, there could be more then one COM Port in a PC. Serial ports are controlled by a special chip called UART (Universal Asynchronous Receiver Transmitter). Different applications use different pins on the serial port and this basically depend of the functions required.

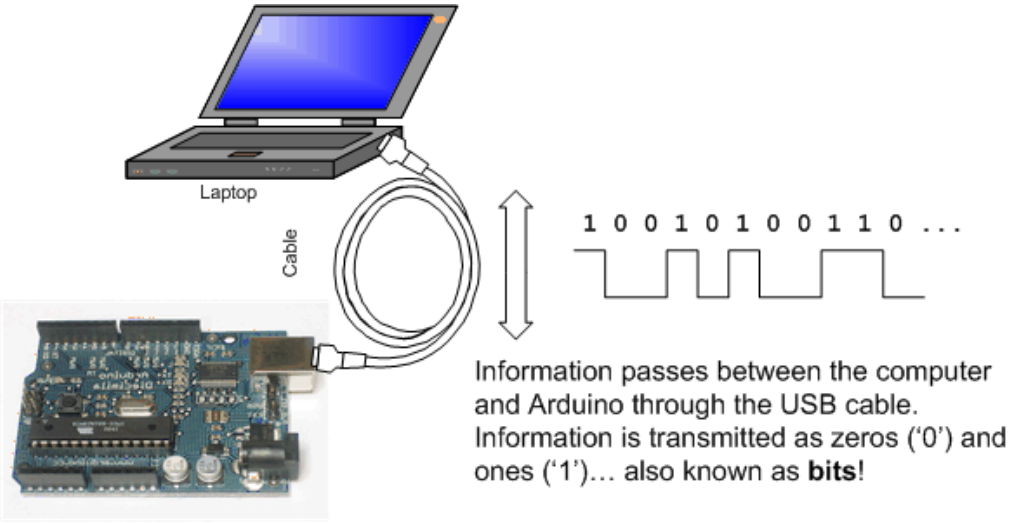


figure4: Serial Data Communication

The serial communication architecture of our project

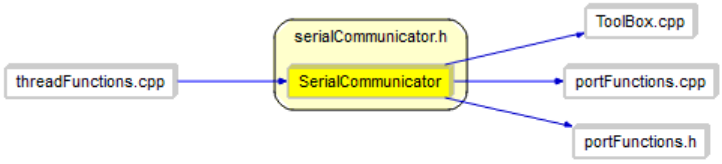


figure5: Thread and Serial Communication

In our project, the serial communicator depends on

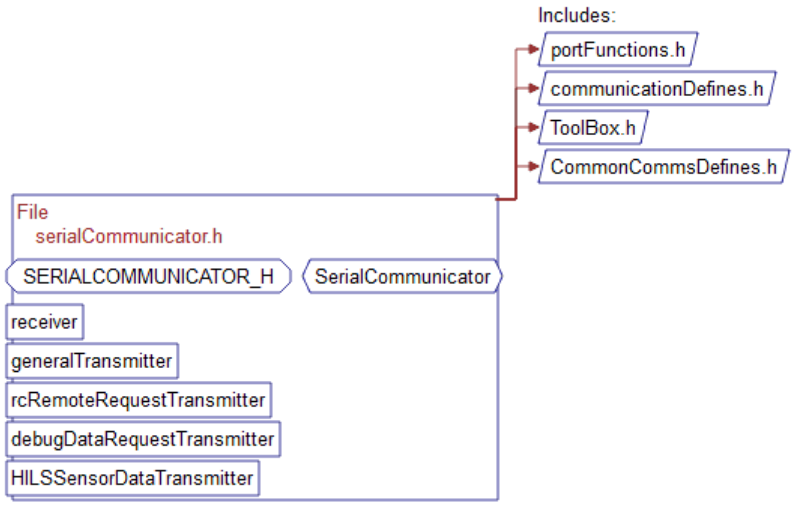


figure6: Serial Communicator

3.1 UART

The Universal Asynchronous Receiver/Transmitter (UART) controller is the key component of the serial communications. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Serial transmission is commonly used with modems and for non -networked communication between computers, terminals and other devices.

3.2 Asynchronous Serial Transmission

Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters in advance and special bits are added to each word which are used to synchronize the sending and receiving units.

Here is how the Data packet is created and how to make the communication shown in figure 7

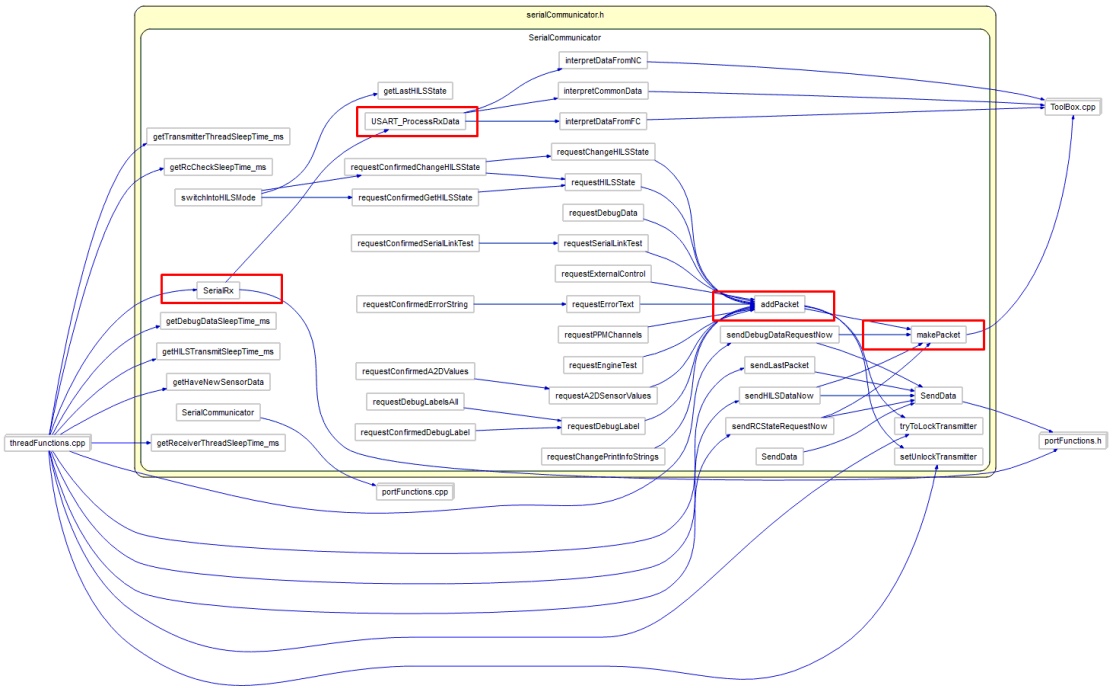


figure7: Serial Communicator

The following figure shows thread function works with serial communication

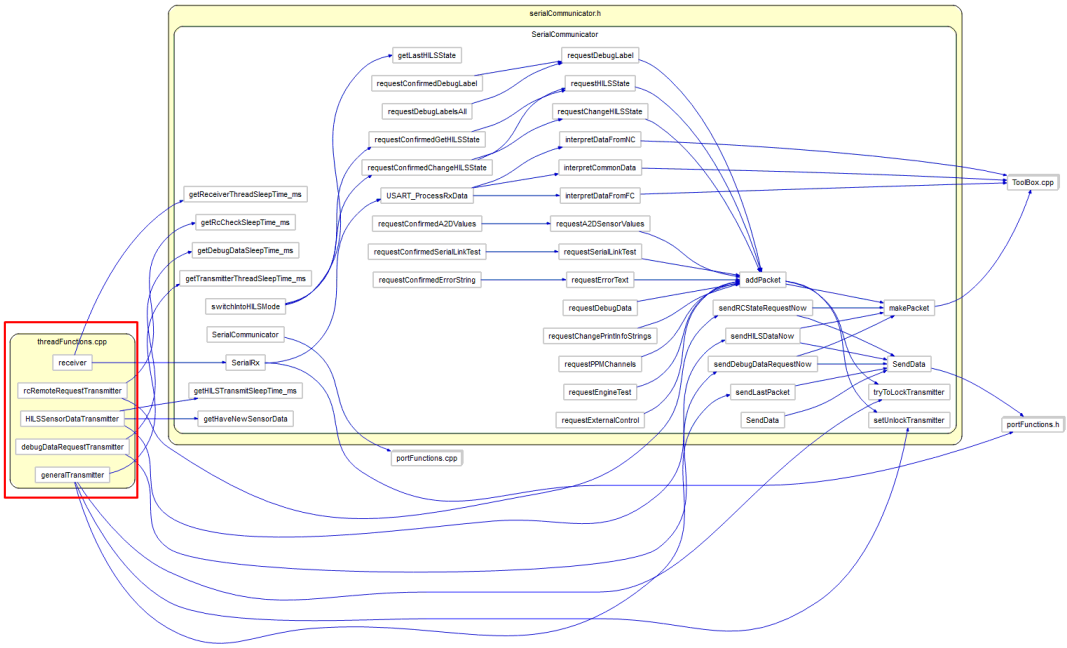


figure8: Thread to Serial Communicator

3.3 Code Architecture

The below is the graphical representation of code architecture. we have two main package named as SerialCommunication and MovingArm.

The architecture of serial communication is bellow.

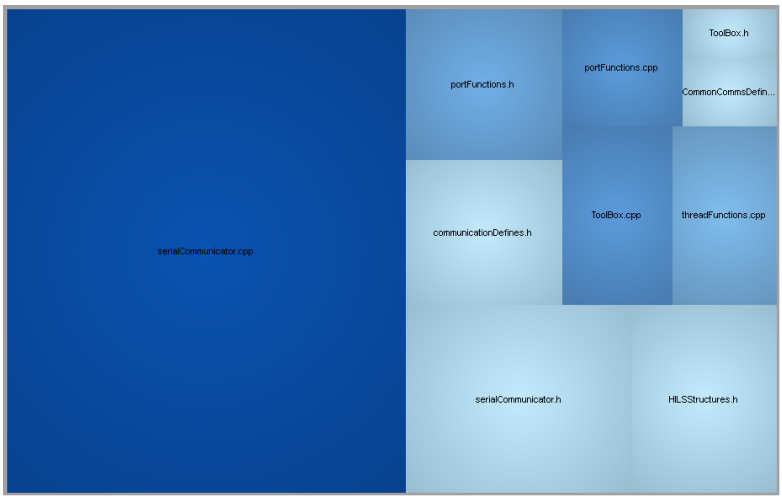


figure9: Code Architecture

The whole communication is designed by Dr. Guillaume Ducard The following figure depicts the creation of threads and threa-loop

3.4 Communication Thread

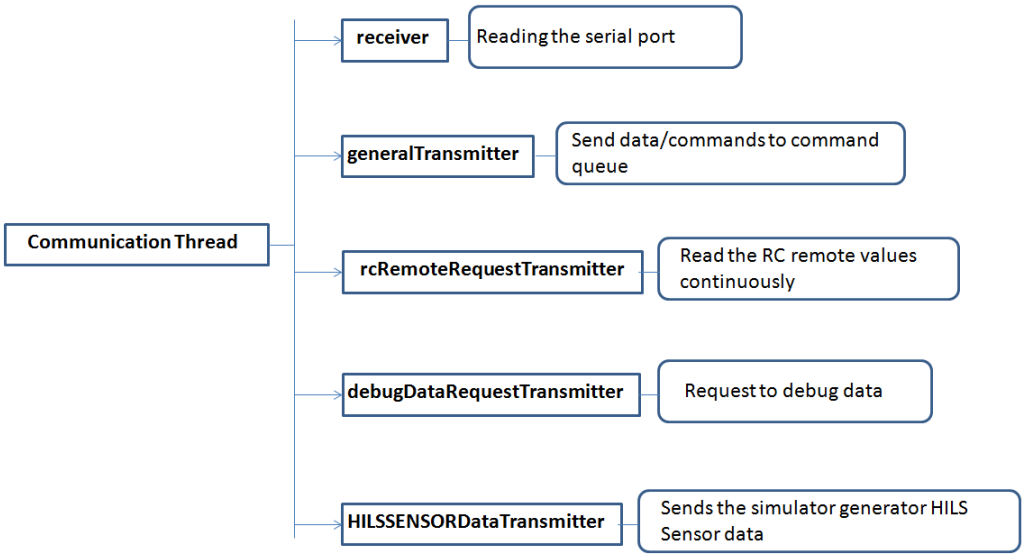


figure10: Generation of Thread Communication

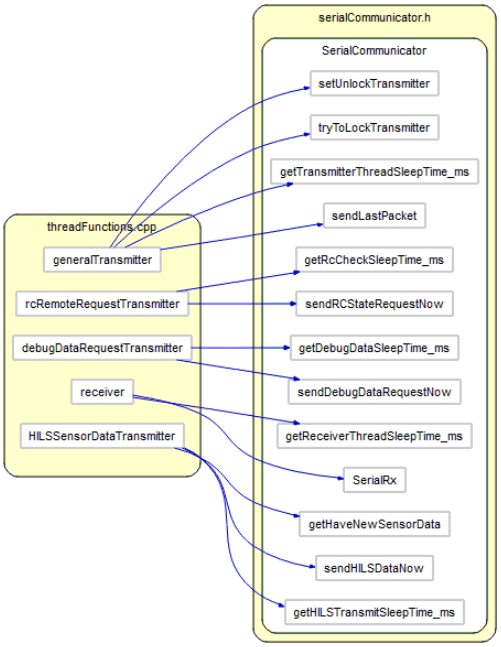


figure11: Connection of Thread Communication

The serial communication package is connected with MovingArm package. Below is the graphical representation of MovingArm Package

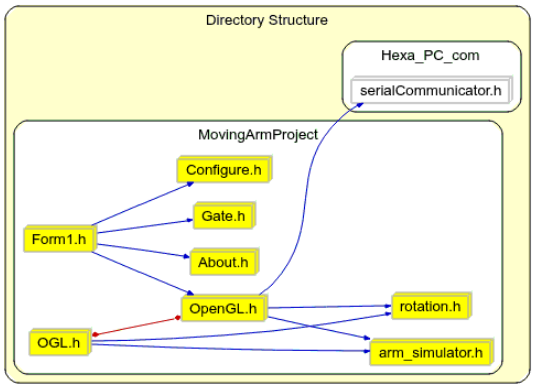


figure12: MovingArm Connected with SerialCommunication

Chapter 4

Hardware in the Loop Simulation

The purpose of Hardware in the Loop Simulation(HIL) simulation is to provide an effective platform for developing and testing real-time embedded systems.
HIL simulation adds the complexity of the plant under control to the test platform. The complexity of the plant under control is included in test and development by adding a mathematical representation of all related dynamic systems. These mathematical representations are referred to as the "plant simulation."
An HIL simulation also include electrical emulation of sensors and actuators. These electrical emulations act as the interface between the plant simulation and the embedded system under test. The value of each electrically emulated sensor is controlled by the plant simulation and is read by the embedded system under test.

This framework includes the following three modules (1) onboard hardware module; (2) control module; and (3) software module.

4.1 Design of HIL simulation framework

The framework of hardware-in-the-loop simulation is depicted in figure 10. This framework includes: (a) an onboard hardware module that activates the model actuators and output sensors; (b) a control module for executing automatic control algorithms; (c) a interface module for generating task commands and monitoring the model through data view and 3D view interfaces; and (d) a software module for integrating all the previous three modules to perform real-time hardware-in-the-loop simulation.

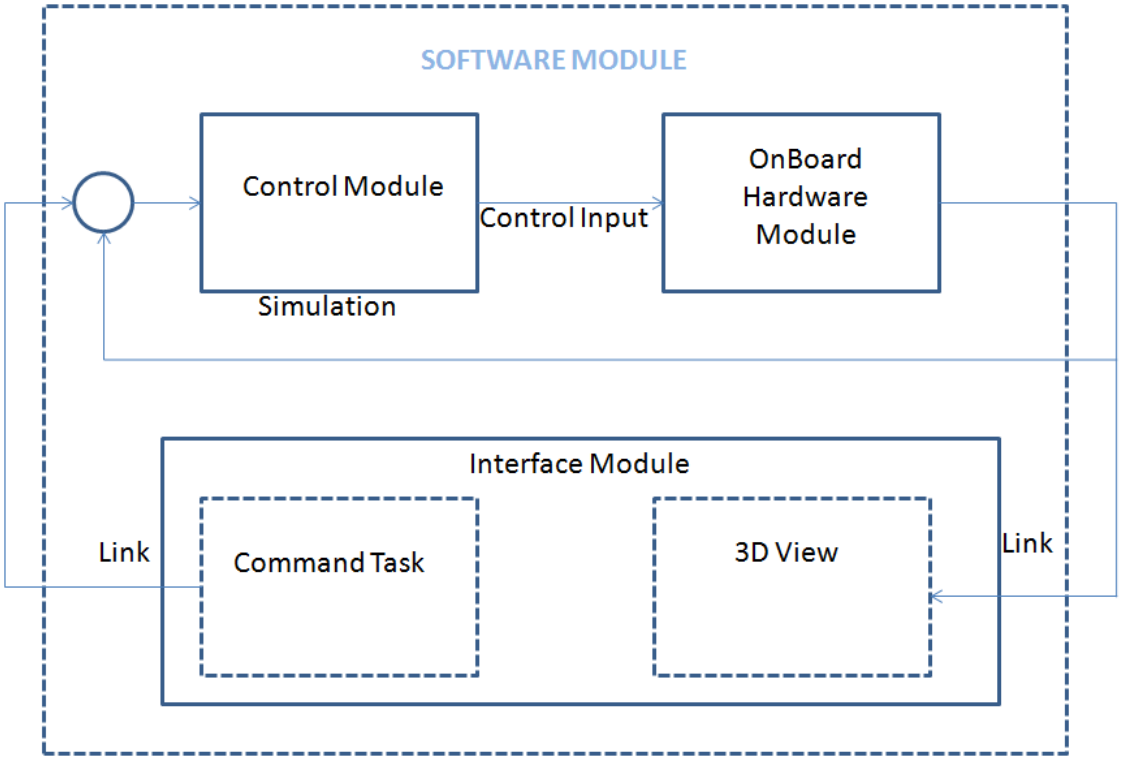


figure13: Framework of the HIL simulation

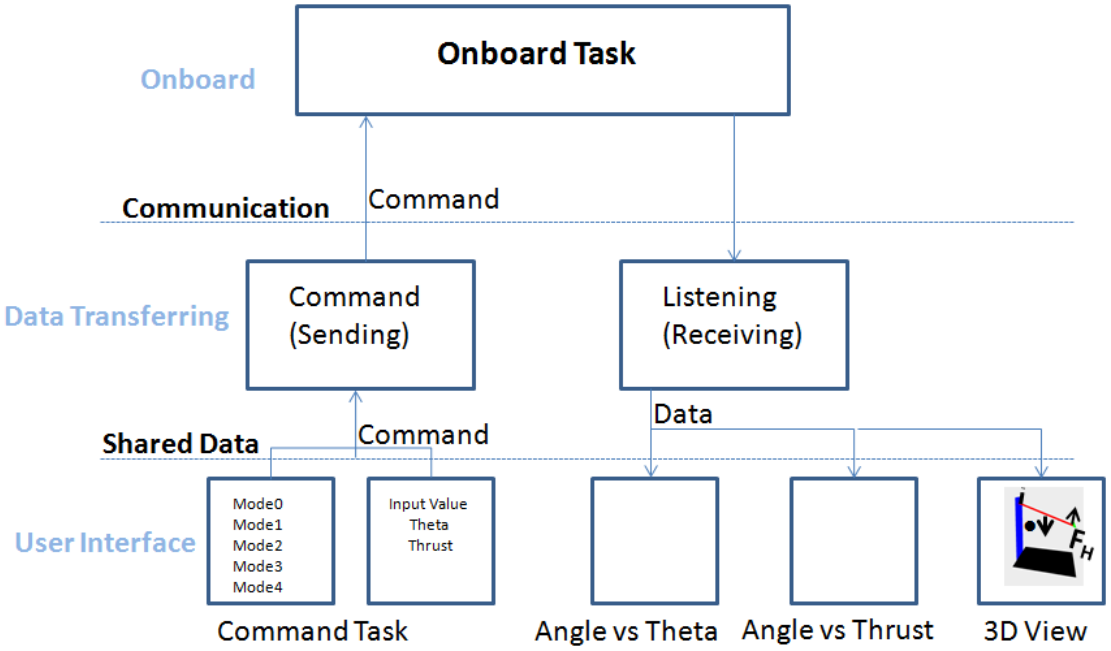


figure14: Structure of Software System

4.2 Module communication

The project consists of two main parts. Hardware module and software module. The software module has also two main parts including Hardware control software and Interface Software. Here we describe the communication of hardware control software and interface software. The main function which is actually invoked by form1.h works as communication manager between serial communication and the graphics interface.The graphics interface is invoked by OpenGL.h. The OpenGL consists of serial communication and invokes two other methods - arm simulator and arm rotation. The arm simulator consists the methods of mathematical formula that calculates the angle and thrust(see Computation of θ). It invokes also the Mode0 where PC simulator runs on its own and the user can control the thrust and the angle θ. To control the thrust, user has two predefined control gate named as Cascade Controller and Leadlag Controller, where the user can put value of K_1, K_2 or A_1, A_2, A_3,A_4 and the controller function will calculate the angle θ. The arm rotation moves the position of arm according to the angel θ and thrust.

4.3 Different Simulation Modes

We have four different modes. MODE0,MODE1,MODE2,MODE3

- MODE0 - containing two different concepts. In first concept that is called **Open loop Thrust Input** The PC simulator runs on its own. User can control the thrust by changing the sliding bar. The thrust that is generated by the slider will send to function of equation for calculating the theta by that thrust(See Computation of θ). The calculated thrust is sent to OpenGL rendering function to be shown in GUI.

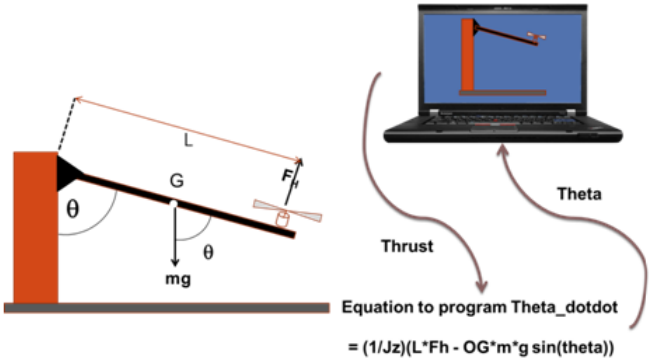


figure15: Open loop thrust input

Second concept is **Closed-loop with local controller** has two predefined control gate named as Cascade Controller and Leadlag Controller, where the user can put value of K_1, K_2 or A_1, A_2, A_3, A_4 that will be define by user and will be send to the function of equation of one of these two controller and the controller function will calculate the desired angle between moving arm and the sidebar.

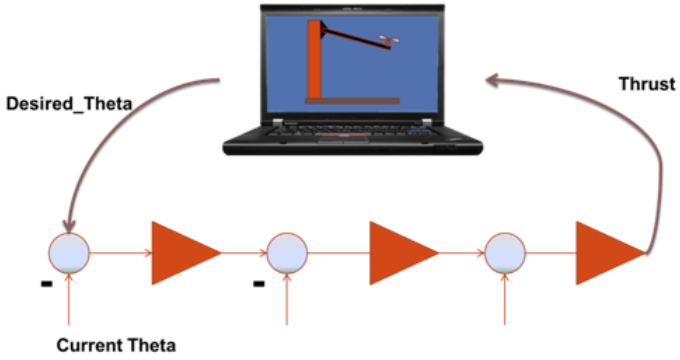


figure16: Closed loop with local controller

The desired angle is set by sliding bar in user interface and this desired theta is calculated by one of the Cascade or Leadlag 1controller depend on user selection. The equation for Cascade and Leadlag controller is shown below.

Equation for Cascade controller

$\ddot{\theta}_d$ = Double derivative Desired Theta
θ_d = Desired Theta
θ_C = Current theta
$\dot{\theta}_l$ = last derivative of theta
K = Constant

$\ddot{\theta}_d$ = ((θ_d - θ_C) $*$ K_1)-$\dot{\theta}_l$))$*$ K_2

Equation for Leadlag controller

Leadlag controller is a recursive method for calculating the $\ddot{\theta}_d$. Let 't' be time for each step of calculation of $\ddot{\theta}_d$ and 'e' be the error function for calculating the θ

$\ddot{\theta}_d$ = ($A_1 * \ddot{\theta}_d$ in (t-1))+ ($A_2 * \ddot{\theta}_d$ in (t-2))+ ($b_1 * e$)+ ($b_2 * e(t-1)$)+($b_3 * e(t-3)$))

The $\ddot{\theta}_d$ and θ_d are set by slider in UI, are sent to function of equation of thrust that is needed to reach for θ_d

This equation to calculate the Thrust F_H is shown bellow

$F_H = \frac{[J_{\vec{z}} * \ddot{\theta} + \vec{OG} * m * g * \sin\theta]}{L}$

The arm rotation moves the position of arm according to the angel that is given by slider and thrust.In this case we can observe that arm will move to exact degree and the thrust will be shown in the plot.

MODE1 - **Real hardware in the loop**In this mode, The PC sends the sensor data to the Board that is current angle of arm (θ) and $\dot{theta}$. The board replies with motor commands (thrust) that is calculated by θ and $\dot{theta}$ and PC will receive this thrust and in GUI, the theta and thrust will be displayed. This loop will continue frequently by this mode which treated by the PC simulator and displayed in the 3D animation. The simulation for this mode shown bellow

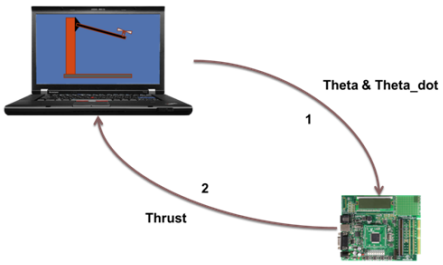
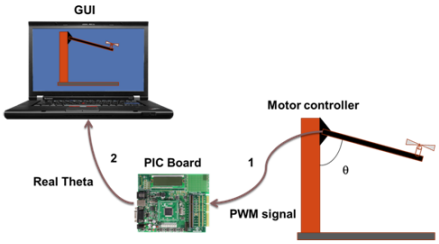
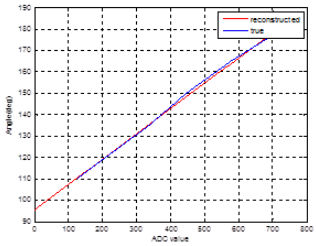


figure17: Mode1- Real Hardware in the loop

- MODE2 - **Real-angle display mode**- In this mode, the 3D GUI is only used to display the real angle of the balancing arm. The PC simulator makes the connection with PIC board, the board ask for the angle(the position of the arm) from the potentiometer. The potentiometer send the PWM signal (pulse with the modulation signal)in the range from 0 to 5 volt that is correspond the angle between arm and sidebar. When the PIC board receives the voltage from the potentiometer, it converts into digital value and the value to the software module in streams of bytes (0 to 1023 byte) through serial communicator and in the simulator by angle sensor calibration that is shown as bellow the exact position of arm is then calculated and shown in the GUI. So we can see that when we move the arm the exact movement and position will be shown in the GUI and the plot.



(c) Real angle display mode



(d) Calibration of angle sensor

- MODE3 - **Real PC control** In this mode, the PC will send the desired thrust to the board through serial communicator and board will transfer this value to PWM signal and will send it to motor controller. The motor controller by receiving the signal start to turn the propeller and this cause the arm start to move. The potentiometer will measure the angle between the moving arm and the column and will send this the data as PWM signal to the board, and finally board send the angle in stream of byte to PC. And simulator convert this value to real angle in radian and will display the arm movement, theta

and thrust in GUI and plot. Mode3 follow same steps as mode1 with this different that in mode1 there is no motor controller and movement of arm but in mode3 the motor controller and real arm movement will be replaced. The steps are shown below in the figure 18

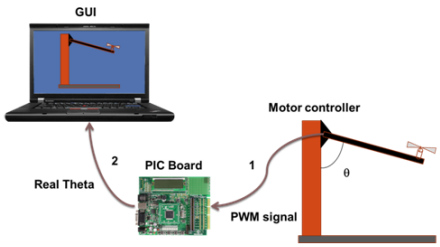


figure19:Real angle display mode

Chapter 5

Conclusion and Future Work

We have presented here a design procedure of a hardware-in-the-loop simulation system for our Balancing Arm System that is designed by our supervisor Dr. Guillaume Ducard. We have designed a Graphical User Interface for that system and also implemented four important mode to make the simulation. Simulation results are obtained for MODE0 and MODE2, compared to those of the actual system. The proposed hardware-in-the-loop simulation system is capable of accurately and efficiently predicting the real system situations and useful tool for us in education research. The framework of MODE1 and MODE3 has already been done. We need to implement that and test with the real system.

Acknowledgement

Most sincere thanks to our supervisor Dr. Guillaume Ducard for his help, guidance and encouragement. We are grateful to got the opportunity to work with him and his guidance led us to accomplish this project.

Bibliography

[1] http://msdn.microsoft.com/en-us/vstudio/hh386302.aspx

[2] http://www.opengl.org/documentation/
http://nehe.gamedev.net/

[3] http://www.mikroe.com/pic/development-boards/

[4] http://www.ftdichip.com/Drivers/D2XX.htm

[5] Hardware in-the-loop-simulation