18 juillet 2016

ESIEE ENGINEERING

Internship report :

UNLV

Driving simulator UNLV

Vanhaesebroucke Geoffrey

I – Project context

The University of Nevada Las Vegas (UNLV) has in his possession for 7 years now a driving simulator. The 3D environment is realized with Unity simulation software.

The goal of my project was to successfully activate the motion simulator based on the acceleration of the car and the road incline. All this with Unity software.

The simulator has become compatible with Unity only since last year. The manufacturer «Simcraft» has developed an SDK in order to run the "motion" with Unity.
However, after making contact with them, they offered to help by subscribing to their two-year support for the cursed sum of $ 10,000, which was unthinkable for our supervisor. So we had to develop our own plugin.

II – Motion activation

A – Refferences of hardwares

The simulator structure is from the society «Simcraft» and the model of the simulator is in the range 3 APEX.

Link : http://www.simcraft.com/apex-sc830.aspx

The structure of the simulator is articulated on three axes corresponding to 3 pistons (Actuators).

The «actuators» are controlled by a «motion box» containing motors and amplifiers. These are subcontracted to the company «Diysim».

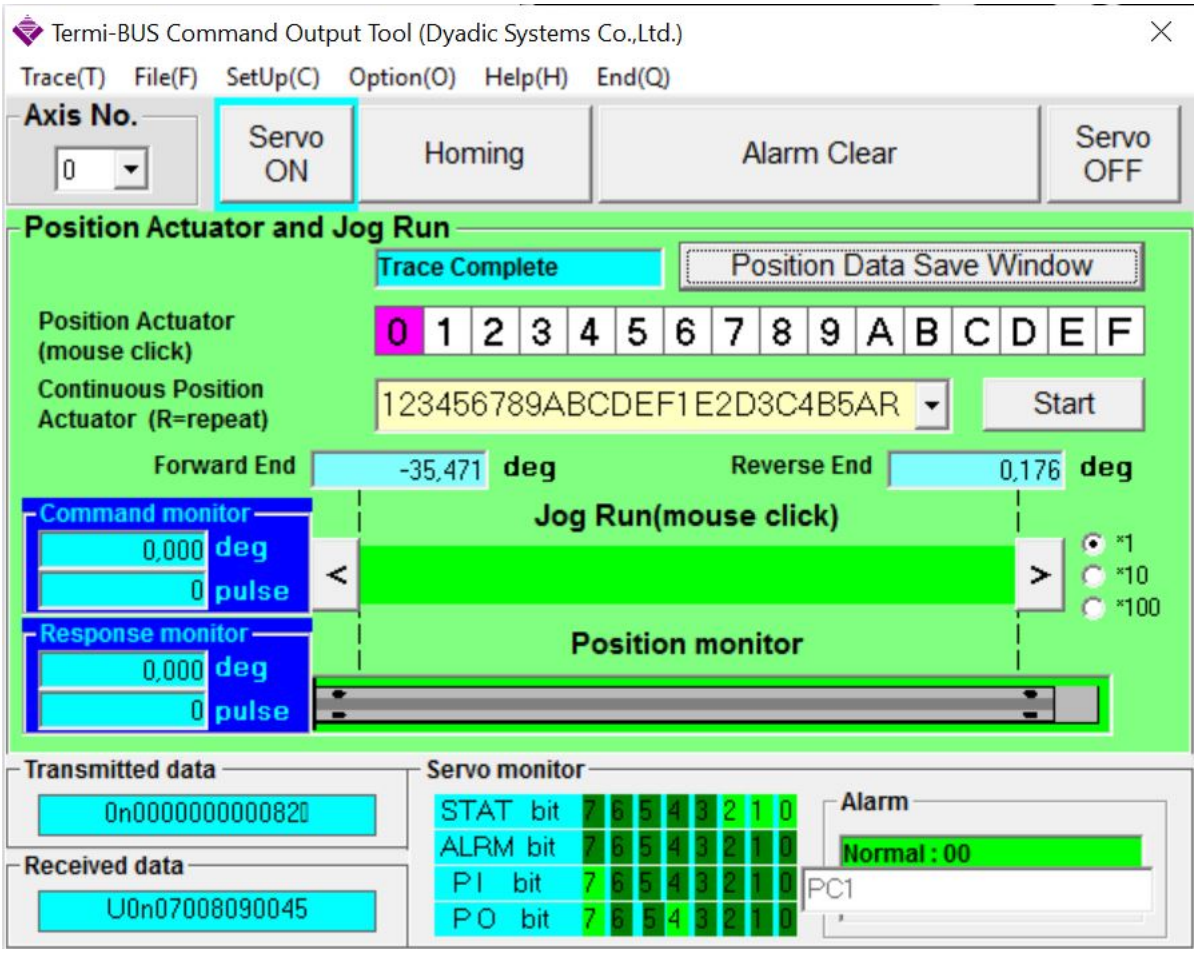
Link : http://diysim.com/product-category/motion-systems/

Specifically, the «actuators» are SCN6_040 models.

The motors of «actuators» can be controlled by sending frames in «String» through an RS232 serial link. On the motion box of the simulator, there is a USB to RS232 converter to drive it from a PC. To activate the serial link on your PC, you need to install some drivers available from the «driver_serial_port» the .zip file.

B – The TBVST software

This software from "Dyadic" is an interface for controlling the simulator on its three axes 0, 1 and 2. Specifically with this software it is possible to initialize the connection with each of the three axes (Homing) and record up to 15 positions per axis and then play sequences.



Download link :

http://www.miraiintertech.com/pctool452.zip

This software has helped me understand the encoding of data to send to the motors. The tests I've made consisted in clicking on positions while listening to what was going through the serial port to identify which command corresponded to any frame in "string".

To listen to the serial port I used the software “Serial port monitor" from Eltima (though the demo version is available for 14 days).

Download link:
http://www.eltima.com/fr/products/serial-port-monitor/

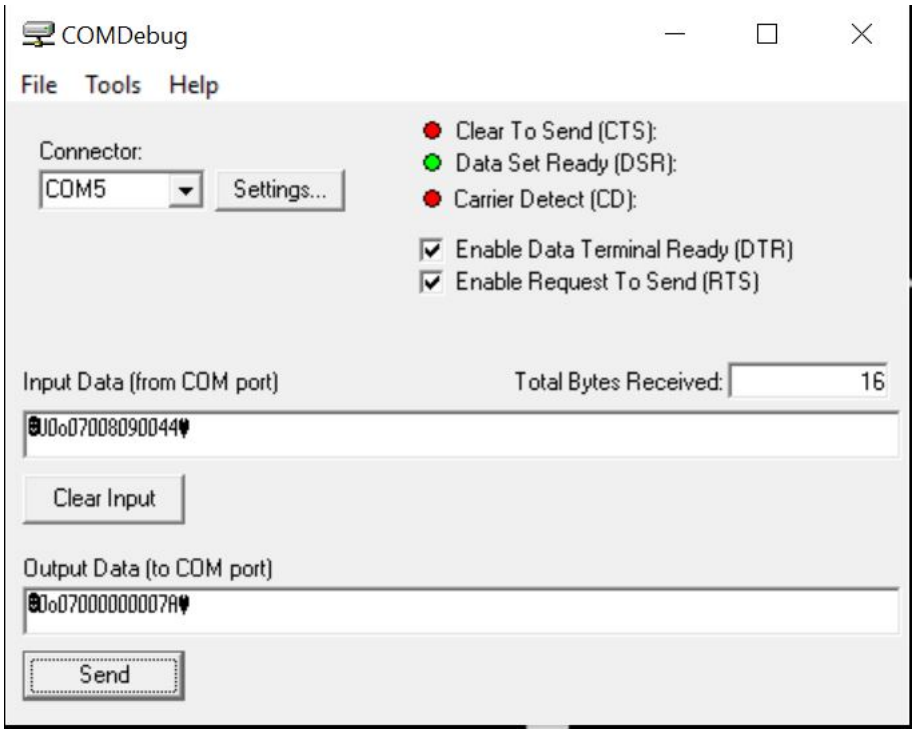
This software cannot be the final solution of the project, since the goal is to send frames through Unity directly.

C – Encoding frames

To understand the encoding frames to send to the "motion box", I made contact with the support of the company " Mirai inter-tech" which offers a kit with TBVST software.

Link: https://miraiintertech.com/e-store/products/PC%252dCTC-Tool-Kit.html

So I contacted Sid ("sid@miraiintertech.com") who forwarded me a .zip file containing datasheets that explain this encoding (page 47 of the "EE06426I.pdf" file).
To send commands, he advised me to use the software "Debug COM".



Here a « homing » on axe 0

Download link: http://www.taltech.com/freesoftware/COMDebugSetup.htm

The most used commands are the commands "o" and "a" (see STRING CALCULATOR.xls)

Command "o" called "Homing" is used to initialize the axes. Here's an example: 0o07000000007A

0 corresponds to the axis 0.
o corresponds to the "homing" command.
07 corresponds to "Homing pattern selection code" (page 49 of EE06426I.pdf).
00000000 by default.
7A corresponds to the BCC ("checksum", it must be correct for the frame to take effect).

BCC calculation (page 3 Mech_Ascii_Setup)

Personally, I used the calculator on this site: http://www.scadacore.com/field-applications/programming-calculators/online-checksum-calculator/

Take this one: 0x100 - Sum Of Bytes

Ex: For the frame above, we must enter "0o0700000000" in ASCII Input.

STX and ETX

You should also know that each frame must be enclosed between STX and ETX. These are special ASCII characters, but they are respectively 02 and 03 in hex.

Ex : « 0o0700000000 » in ascii = « 306F30373030303030303030 » in hex

You must write in 02306F30373030303030303030303 Hex Input to get special characters in the string.

Command "a" they allowed to reach a position (angle), here's an example: 0aFFFFD8F00005

0 corresponds to the axis 0.
a corresponding to the control change position.
FFFFD8F0 the 8 digits of the "target position".
00 default (place holder).
05 corresponds to the BCC ("checksum", it must be correct for the frame to take effect).

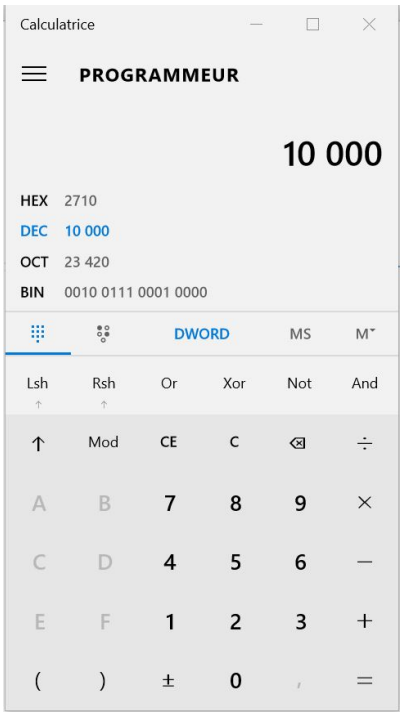
Calculation of 8 digits "target position"

The simulator can move from 0 ° to 35 ° on each axis. This tilt range consists of about 20 000 pulses.

Ex: 17.5 ° corresponds to 10 000 pulses.

It is this number of pulses to be translated in hex and is the "target position" of 8 digits seen previously.

10,000 pulses = 2710



Note: Regardless of the command, do not forget to calculate the BCC and then add the STX and ETX at the beginning and end of the frame.

Here: .0aFFFF2710002D.

Other prospects:

There are dozens more possible command, it would be interesting to look at the 'v' one to change the acceleration and speed of the simulator.

III – Unity, 3D simulation software

A – Link with Unity

After understanding the functioning of the frames, the next step was to automate the calculations of the BCC and the "target position" to form our own frames only having an angle as input. This step was achieved by inserting scripts in the Unity software.

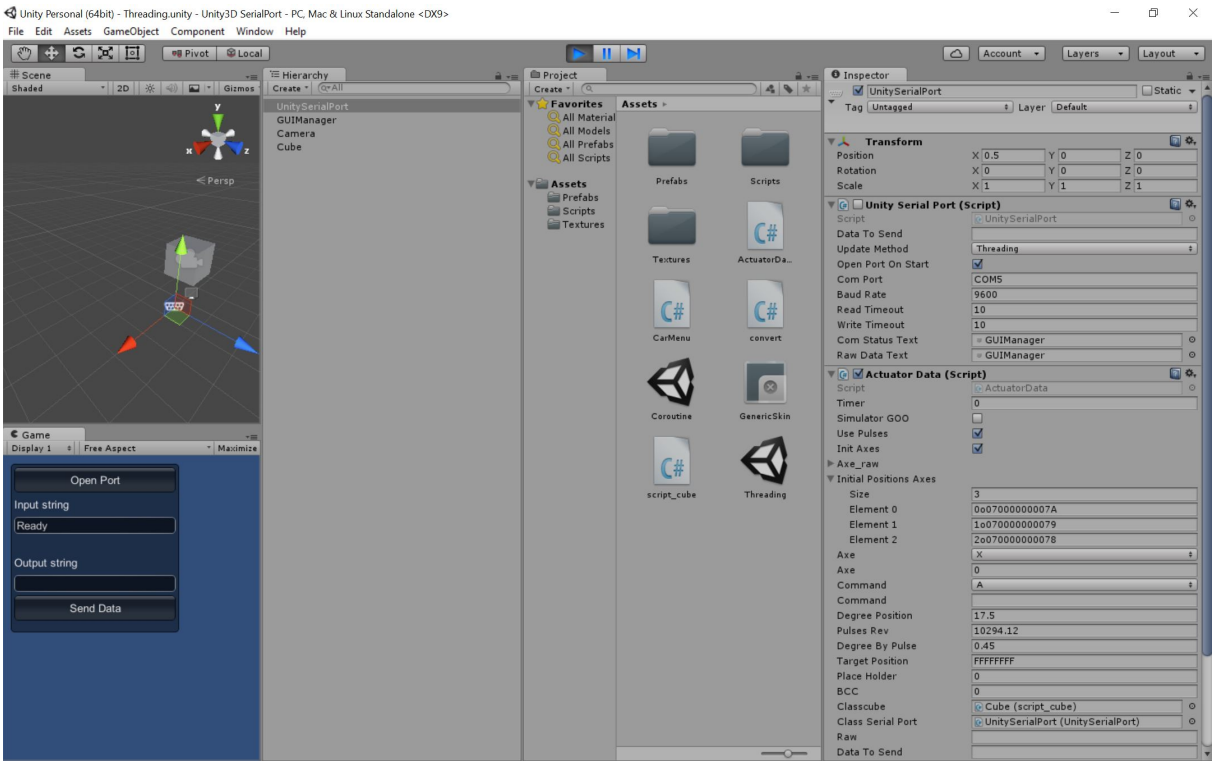
The calculation has been automated with scripts on Unity:

```
using UnityEngine;
using System.Collections;
using System.Text;

public class ActuatorData : MonoBehaviour {

    public bool usePulses = false;
    public bool initAxes = true;

    public enum Axe{X = 0, Y = 1, Z = 2};
    public Axe[] axe_raw = new Axe[] {Axe.X, Axe.Y, Axe.Z};

    public string[] initialPositionsAxes = new string[] {""+STX+"0o07000000007A"+ETX,""+STX+"1o070000000079"+ETX,""+STX+"2o070000000078"+ETX};
    public Axe _Axe = Axe.X;
    public byte _axe = 0;
    public enum Command{a,o};
    public Command _Command = Command.a;

    const char STX = '\u0002';
    const char ETX = '\u0003';

    public string _command = "";
    public float DegreePosition = 15;
    public float PulsesRev = 800;
    public float DegreeByPulse = 0.45f;
    public string TargetPosition = "FFFFFFFF";
    public byte PlaceHolder = 0;
    public byte BCC = 0;
```

frames encapsulation script part

Then we send the frames directly to the serial port of the "Motion box" with the script "UnitySerialPort.cs".
This way we managed to move the simulator through Unity on the axis 0, however, for reasons that are still unknown, moving on axes 1 and 2 remains impossible. In fact, after talking to the support at Mirai confirming that commands sent are good, the solution is still not found.

Improve of the code:
The code was improved to make it easier to use and also to automate the start of the Unity software. I create with the help of the project manager, a new script (script_cube) attached to a new "game object," a cube, which represents the future car. The idea then was to recover the tilt of the cube on its horizontal axis compared to normal. This angle is then translated into frame and sent to the simulator by the script previously seen "ActuatorData.cs".To test this script, tilting the cube is simulated by pressing the arrow keys.



Unity interface of the plugin

In this version, when making a "play", pressing "space" launches a boot on the axis 0, the simulator receives a "homing" and stops on the 0° position.
You must then check the "GOO Simulator" to be able to move the simulator with the arrow keys.

B – Work to be completed

- Managing to activate the motion on axes 1 and 2.

- Optimize the automation of code.

- Optimising the calculation pulses (precision simulator).

- Merge scripts with the latest version of the Las Vegas project.

- ++ Interest to vary the speed and the simulator-accelerated.