Internship report :



**Driving simulator UNLV**

[I – Project context 3](#_Toc457466270)

[II – Motion activation 4](#_Toc457466271)

[A – Refferences of hardwares 4](#_Toc457466272)

[B – The TBVST software 5](#_Toc457466273)

[C – Encoding frames 6](#_Toc457466274)

[III – Unity, 3D simulation software 9](#_Toc457466275)

[A – Link with Unity 9](#_Toc457466276)

[B – Work to be completed 11](#_Toc457466277)

[ANNEX 12](#_Toc457466278)

# 

## 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».

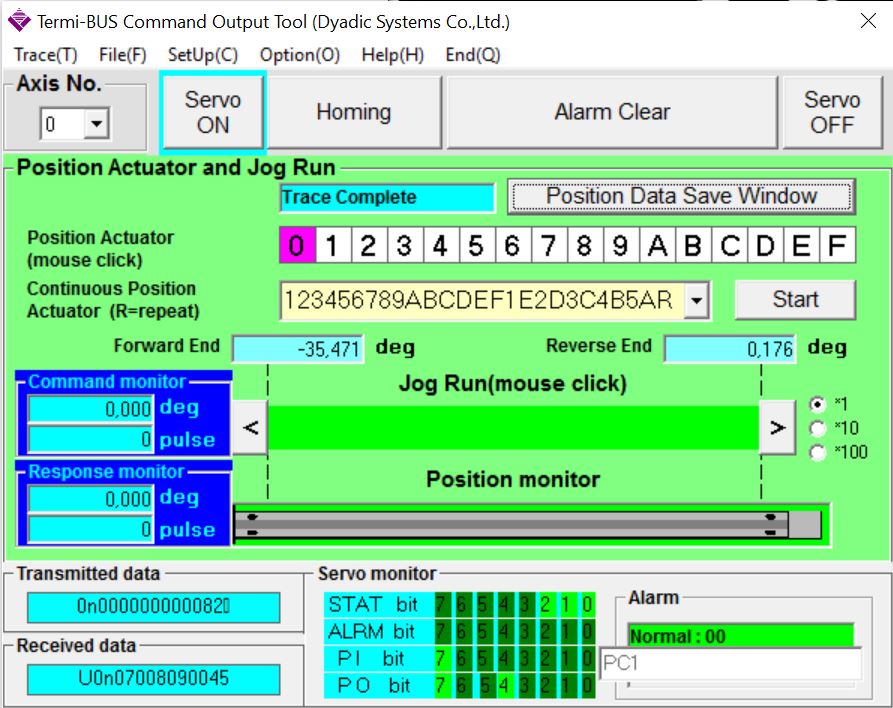
Link : <http://diysim.com/product-category/motion-systems/>

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/>

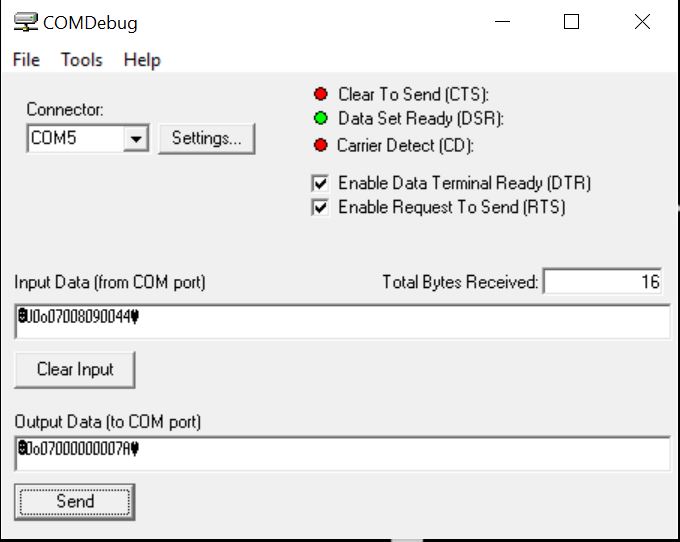
This software cannot be the final solution of the project, since the goal is to send frames through Unity directly.

### 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.

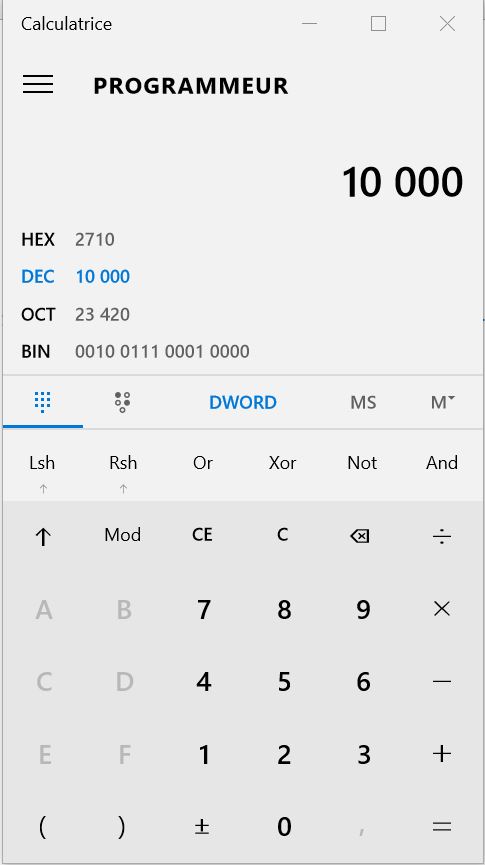
Ex : « 0o0700000000 » in ascii = « 306F30373030303030303030 » in hex

You must write in 02306F3037303030303030303003 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.

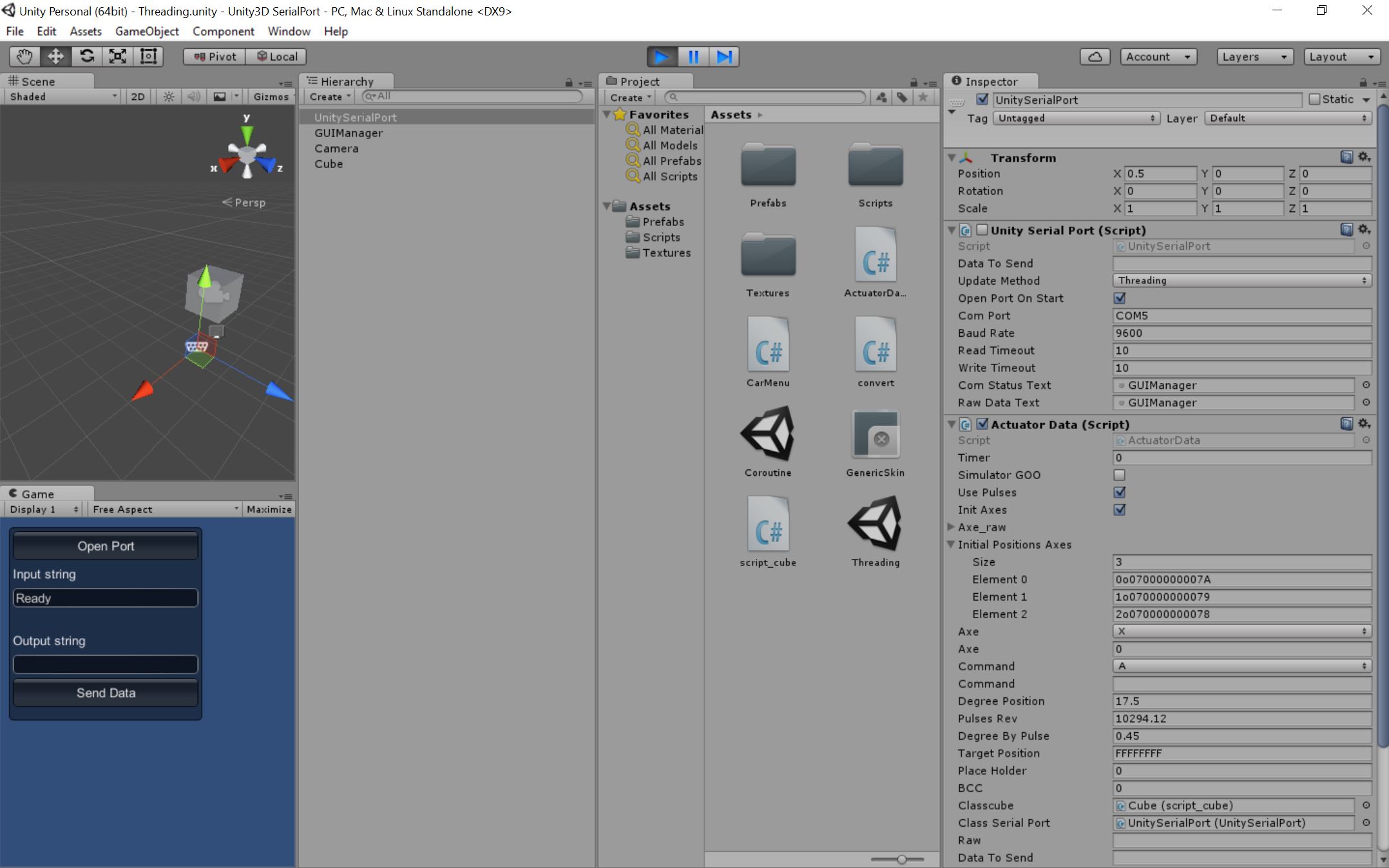
The calculation has been automated with scripts on Unity:

*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, 1 and 2. But for a reason that we still ignore, we have to run TVBST software first to initialize the actuators, but axe 0 works well without making this step.

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,   
1 and 2, the simulator receives a "homing" and stops on the 0° position in three axis.  
You must then check the "GOO Simulator" to be able to move the simulator with the arrow keys.

### B – Work to be completed

- Reproduce the sequence of initialization (without TVBST software)

- 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.

## ANNEX

Script\_cube :

using UnityEngine;  
using System.Collections;  
  
public class script\_cube : MonoBehaviour {  
  
        public GameObject cube;  
        *//public ActuatorData test\_angle;*  
  
        public float angle\_cube\_x = 0;  *//0*  
        public float angle\_cube\_y = 0;  *//2*  
        public float angle\_cube\_z = 0;  *//1*  
  
        public float angle\_in\_pulse\_x = 0;  
        public float angle\_in\_pulse\_y = 0;  
        public float angle\_in\_pulse\_z = 0;  
  
        public ActuatorData majActuatorData;  
    *// Use this for initialization*  
    void Start () {  
      
    }  
      
    *// Update is called once per frame*  
    void Update () {  
            angle\_cube\_x = cube.transform.eulerAngles.x;  
            angle\_cube\_y = cube.transform.eulerAngles.y;  
            angle\_cube\_z = cube.transform.eulerAngles.z; *//axe 1*  
  
            *//Debug.Log ("angle\_cube\_y = " + angle\_cube\_y);*  
            *//Debug.Log ("angle\_cube\_z = " + angle\_cube\_z);*  
              
            *//Debug.Log ("EnPulses = " +test\_angle.calculPulsesFromAngle (17f));*  
            *//Debug.Log ("from /script\_cube.cs : angle\_cube\_x = " + cube.transform.eulerAngles.x);*  
            *//Debug.Log ("from /script\_cube.cs : TargetPosition = " + test\_angle.calculPulsesFromAngle(angle\_cube\_x));*  
  
  
                *//convert angle axe 0*  
                if (angle\_cube\_x >= 0 && angle\_cube\_x <= 17.5) {  
                        angle\_cube\_x = 17.5f - angle\_cube\_x;  
                } else if (angle\_cube\_x >= 342.5 && angle\_cube\_x <= 360) {  
                        angle\_cube\_x = 360f - angle\_cube\_x + 17.5f;  
                } else if (angle\_cube\_x == 0) {  
                        angle\_cube\_x = 17.5f;  
                } else {  
                        angle\_cube\_x = 0f;  
                }  
  
                *//convert angle axe 2*  
                if (angle\_cube\_y >= 0 && angle\_cube\_y <= 17.5) {  
                        angle\_cube\_y = 17.5f - angle\_cube\_y;  
                } else if (angle\_cube\_y >= 342.5 && angle\_cube\_y <= 360) {  
                        angle\_cube\_y = 360f - angle\_cube\_y + 17.5f;  
                } else if (angle\_cube\_y == 0) {  
                        angle\_cube\_y = 17.5f;  
                } else {  
                        angle\_cube\_y = 0f;  
                }  
  
                *//convert angle axe 1*  
                if (angle\_cube\_z >= 0 && angle\_cube\_z <= 17.5) {  
                        angle\_cube\_z = 17.5f - angle\_cube\_z;  
                } else if (angle\_cube\_z >= 342.5 && angle\_cube\_z <= 360) {  
                        angle\_cube\_z = 360f - angle\_cube\_z + 17.5f;  
                } else if (angle\_cube\_z == 0) {  
                        angle\_cube\_z = 17.5f;  
                } else {  
                        angle\_cube\_z = 0f;  
                }  
  
  
                *//Conversion in pulses (ratio = 35° / 20 000 pulses)*  
                *//This is if you encoche the "Use pulses" square*  
                angle\_in\_pulse\_x = angle\_cube\_x/0.0017f;  
                majActuatorData.PulsesRev0 = angle\_in\_pulse\_x;  *//sending to ActuatorData.cs*  
  
                angle\_in\_pulse\_y = angle\_cube\_y/0.0017f;  
                majActuatorData.PulsesRev2 = angle\_in\_pulse\_y;  *//sending to ActuatorData.cs*  
  
                angle\_in\_pulse\_z = angle\_cube\_z/0.0017f;  
                majActuatorData.PulsesRev1 = angle\_in\_pulse\_z;  *//sending to ActuatorData.cs*  
  
  
                *//Debug.Log ("angle\_in\_pulse\_z = " + angle\_in\_pulse\_z);*  
  
                *//mooving cube with arrow*  
                *//axe 0*  
                if (Input.GetKey(KeyCode.LeftArrow)){  *//to the left*  
                        transform.Rotate(0.3f,0,0);  
                }  
                if (Input.GetKey(KeyCode.RightArrow)){  *//to the right*  
                        transform.Rotate(-0.3f,0,0);  
                }  
  
                *//axe 1*  
                if (Input.GetKey(KeyCode.UpArrow)){  *//to the left*  
                        transform.Rotate(0,0,0.3f);  
                }  
                if (Input.GetKey(KeyCode.DownArrow)){  *//to the right*  
                        transform.Rotate(0,0,-0.3f);  
                }  
  
                *//axe 2*  
                if (Input.GetKey(KeyCode.B)){  *//to the left*  
                        transform.Rotate(0,0.3f,0);  
                }  
                if (Input.GetKey(KeyCode.N)){  *//to the right*  
                        transform.Rotate(0,-0.3f,0);  
                }  
  
  
      
    }  
}

Actuators\_data :

using UnityEngine;  
using System.Collections;  
using System.Text;  
  
public class ActuatorData : MonoBehaviour {  
      
    public float timer = 0f;  
    public bool simulatorGOO = false;  
    public bool usePulses = true;  
    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 DegreePosition0 = 15;  
    public float DegreePosition1 = 15;  
    public float DegreePosition2 = 15;  
  
    public float PulsesRev = 800; *//Actualised variable by script\_cube.cs, if "use pulse" is on.*  
    public float PulsesRev0 = 800;  
    public float PulsesRev1 = 800;  
    public float PulsesRev2 = 800;  
  
    public float DegreeByPulse = 0.0017482517482517f; *//35° / 20 000 pulses*  
    *//public float DegreeByPulse = 0.45f;*  
    public string TargetPosition = "FFFFFFFF";  
    public byte PlaceHolder = 0;  
    public byte BCC = 0;  
      
    public script\_cube classcube; *//this is to get the "angle\_cube\_x" variable from script\_cube.cs*  
    public UnitySerialPort classSerialPort;  
  
    public string raw = "";  
    public string dataToSend = "";  
  
    public GUIManager S\_GUIManager;  
  
        public byte i;  
      
  
    void Start () {  
                string aa = CalculateChecksum("0T4000004000");  
                *//Debug.Log(aa+"/   "+calculPulsesFromAngle(14));*  
                i = 0;  
  
  
                classSerialPort.OpenSerialPort (); *//vrai*  
  
  
                *//init ();*  
    }  
      
      
    *// Update is called once per frame*  
    void FixedUpdate () {  
  
        *//if(Input.GetKeyDown(KeyCode.Space)){*  
                *//init ();*  
  
        *//}*  
  
        */\*if (Input.GetKey (KeyCode.RightArrow)){*  
*//classSerialPort.OpenSerialPort (); //vrai*  
*//classSerialPort.SerialPort.Open (); //ajout*  
*ouvrirPort();*  
*}\*/*  
          
        if (Input.GetKey (KeyCode.Space)) {  
                init ();      
        }  
  
          
  
        DegreePosition0 = classcube.angle\_cube\_x; *//axe 0*  
        DegreePosition1 = classcube.angle\_cube\_z; *//axe 1*  
        DegreePosition2 = classcube.angle\_cube\_y; *//axe 2*  
  
        *//Debug.Log ("from ActuatorsData.cs : DegreePosition " + DegreePosition);*  
  
        *//if(Input.GetKeyUp(KeyCode.UpArrow)){*  
                *//simulatorGOO = true;*  
  
        *//}*  
  
  
                */\*if (simulatorGOO) {*  
*timer += Time.fixedDeltaTime;*  
  
*createRaw0 ();*  
*if (timer > 0.8f) {*  
  
  
*S\_GUIManager.OutputString = raw;*  
*S\_GUIManager.sendData ();*  
*timer = 0f;*  
*}*  
  
  
  
*} \*/*  
                          
          
        if (simulatorGOO) {  
            timer += Time.deltaTime;  
                  
  
                        if (timer <= 0.4f) {  
                                createRaw0 ();  
                                S\_GUIManager.OutputString = raw;  
                                S\_GUIManager.sendData ();  
                                  
                        } else if (timer > 0.4f && timer <= 0.8f) {  
                                createRaw1 ();  
                                S\_GUIManager.OutputString = raw;  
                                S\_GUIManager.sendData ();  
  
                        } else if (timer > 0.8f && timer <= 1.2f) {  
                                createRaw2 ();  
                                S\_GUIManager.OutputString = raw;  
                                S\_GUIManager.sendData ();  
  
                        }          
                        else {  
                                timer = 0f;  
                        }  
  
                }    
  
                */\*Debug.Log ("DegreePosition0  " + DegreePosition0);*  
*Debug.Log ("DegreePosition1  " + DegreePosition1);*  
*Debug.Log ("DegreePosition2  " + DegreePosition2);*  
  
*Debug.Log ("PulsesRev0  " + PulsesRev0);*  
*Debug.Log ("PulsesRev1  " + PulsesRev1);*  
*Debug.Log ("PulsesRev2  " + PulsesRev2);*  
*\*/*      
  
    }  
      
        *//IEnumerator waitMaGueule () {*  
                *//yield return new WaitForSeconds(5f);*  
                *//simulatorGOO = true;*  
        *//}*  
  
    public void createRaw0 () { *// this function create the string data*  
  
                \_axe = 0;  
                PulsesRev = PulsesRev0;  
                \_command = convertCommand(\_Command);  
                calculTargetPosition(DegreePosition0);  
                *//Debug.Log("targetPos = "+ TargetPosition);*  
                raw = ""+\_axe+""+\_command+""+TargetPosition+""+getString2Numbers(PlaceHolder);  
                string bcc = CalculateChecksum(raw);  
                *//Debug.Log("Bcc = "+ bcc);*  
                raw = STX+""+raw+""+bcc+""+ETX;  
  
              
    }  
  
    public void createRaw1 () { *// this function create the string data*  
  
                \_axe = 1;  
                PulsesRev = PulsesRev1;  
                \_command = convertCommand(\_Command);  
                calculTargetPosition(DegreePosition1);  
                *//Debug.Log("targetPos = "+ TargetPosition);*  
                raw = ""+\_axe+""+\_command+""+TargetPosition+""+getString2Numbers(PlaceHolder);  
                string bcc = CalculateChecksum(raw);  
                *//Debug.Log("Bcc = "+ bcc);*  
                raw = STX+""+raw+""+bcc+""+ETX;  
  
  
    }  
  
    public void createRaw2 () { *// this function create the string data*  
  
                \_axe = 2;  
                PulsesRev = PulsesRev2;  
                \_command = convertCommand(\_Command);  
                calculTargetPosition(DegreePosition2);  
                *//Debug.Log("targetPos = "+ TargetPosition);*  
                raw = ""+\_axe+""+\_command+""+TargetPosition+""+getString2Numbers(PlaceHolder);  
                string bcc = CalculateChecksum(raw);  
                *//Debug.Log("Bcc = "+ bcc);*  
                raw = STX+""+raw+""+bcc+""+ETX;  
  
  
    }  
  
  
        public void init() {  
                *//classSerialPort.OpenSerialPort ();*  
                *//setAxe(\_Axe);*  
                *//S\_GUIManager.OutputString = initialPositionsAxes[\_axe];*  
  
                S\_GUIManager.OutputString = STX+"0o07000000007A"+ETX;  
                S\_GUIManager.sendData ();  
                *//yield return new WaitForSeconds(1);*  
                S\_GUIManager.OutputString = STX+"1o070000000079"+ETX;  
                S\_GUIManager.sendData ();  
                *//yield return new WaitForSeconds(1);*  
                S\_GUIManager.OutputString = STX+"2o070000000078"+ETX;  
                S\_GUIManager.sendData ();  
  
                initAxes = false;  
                *//simulatorGOO = true;*  
                *//StartCoroutine(waitMaGueule());*  
  
        }  
  
  
        public void setAxe (Axe pAxe) {  
  
                switch(pAxe) {  
                case Axe.X:  
                        \_axe = 0;  
                        break;  
                case Axe.Y:  
                        \_axe = 1;  
                        break;  
                case Axe.Z:  
                        \_axe = 2;  
                        break;  
                default :  
                        Debug.LogError("Error convert Axe ! - ActuatorData");  
                        break;  
                }  
        }  
      
    public void calculTargetPosition (float pAngle) {  
                int pulses = calculPulsesFromAngle(pAngle);  
                if(usePulses) {  
                    TargetPosition = convertPulsesInHEX((int)PulsesRev);  
                }  
                else {  
                    TargetPosition = convertPulsesInHEX(pulses);  
                }  
    }  
  
    public string getString2Numbers (byte pByte) {  
        string res = "";  
  
        if(pByte > 9) {  
            res = ""+pByte;  
        }  
        else {  
            res = "0"+pByte;  
        }  
        return res;  
    }  
  
        public string convertCommand (Command pCommand) {  
                string command = "";  
                switch(pCommand) {  
                case Command.a:  
                        command = "a";  
                        break;  
                case Command.o:  
                        command = "o";  
                        break;  
                default :  
                        Debug.LogError("Error convert Command ! - ActuatorData");  
                        break;  
                }  
                return command;  
        }  
  
    private string CalculateChecksum(string dataToCalculate)  
        {  
                byte[] byteToCalculate = Encoding.ASCII.GetBytes(dataToCalculate);  
  
                int checksum = 0;  
  
                foreach (byte chData in byteToCalculate)  
                {  
                        checksum += chData;  
                }  
  
                *//checksum &= 0xff;*  
                checksum = (0x100 - checksum) & 0xff;  
  
                return checksum.ToString("X2");  
  
        }   
  
        public int calculPulsesFromAngle (float pAngle) {  
  
                return (int)(pAngle/DegreeByPulse);  
        }  
      
        public string convertPulsesInHEX (int pPulses) {  
  
                int decValue = (int)pPulses;  
                decValue = decValue\*(-1);  
                string hexValue = decValue.ToString("X");  
                *//Debug.Log("hexvalue --------> "+ hexValue);*  
                return hexValue;  
        }  
      
}