Table of Contents

1. Laboratory on LEGO Mindstorms - EV3 C# Programming Theory & Practice 2
1.1. Introduction 2
1.2. C# Example - Controlling EV3 movement with hand gestures 2
2. Laboratory on LEGO Mindstorms - Programming EV3 in C# 3
2.1. Leap Motion Overview 3
2.1.1. Hand 3
2.1.2. Arm 4
2.1.3. Finger 4
2.1.4. Bone 4
2.2. Coordinates System 5
2.3. System Architecture 5
2.3.1. Native Application Interface 5
2.3.2. WebSocket Interface 6
2.4. Setting Up a Project 6
2.4.1. Hello World C# 6
2.5. Sample source code 7
2.6. Citations 7

1. Laboratory on LEGO Mindstorms - EV3 C# Programming Theory & Practice

1.1. Introduction

This tutorial shows the basics of programming theory and practice of LEGO EV3 robot in C# language. The physical behaviour and parameters of the robot are represented by set of methods and fields assigned to appropriate classes. In order to control the behaviour of the robot, appropriate classes have to be instantiated and initialized and appropriate methods have to be called. For example, the EV3 brick is represented in code by the class Brick.

1.2. C# Example - Controlling EV3 movement with hand gestures

The example is program which navigates the EV3 depending on the hand movement detected by kinect device. The EV3 robot brick is represented in the C# code by class brick. The following code creates a new brick instance and initializes it with the new blutooth connection assigned to given port (COM4 in this case). Then, asynchronous bluetooth connection is established and confirmation sound is played on the robot side:

```
_brick = new Brick(new BluetoothCommunication("COM4"));
_brick.ConnectAsync();
_brick.DirectCommand.PlayToneAsync(10, 1000, 300);
```

The idea is such: asynchronously detect the movement of hands and take into account joints placement - basing on its coordinates the relative position of hand with respect to shoulders will be recognized. All neccessary data is given in argument “joints”

```
this.DrawHandAsync(body.HandLeftState, jointPoints[JointType.HandLeft], dc,
joints);
this.DrawHandAsync(body.HandRightState, jointPoints[JointType.HandRight],
dc, joints);
```

In the above method, hand and shoulder variables (both right) should be defined:

```
Joint rShoulder = joints[JointType.ShoulderRight];
Joint rHand = joints[JointType.HandRight];
```

Basing on the relative placement of the hand with respect to the shoulder, the appropriate action is performed (movement method is called). The mathematical comparisons are simply horizontal and vertical displacements between shoulders and hands.

```
drawingContext.DrawEllipse(this.handClosedBrush, null, handPosition,
HandSize, HandSize);
if (Math.Abs(rShoulder.Position.X - rHand.Position.X) <
Math.Abs(rShoulder.Position.Y-rHand.Position.Y))
{
```

```
   if (rHand.Position.Y > rShoulder.Position.Y)
      goForward();
   else
      goBackward();
} else
{
   if (rHand.Position.X > rShoulder.Position.X)
      turnRight();
   else
      turnLeft();
}
```

The movement functions are self explanatory and designed in the following way, calling the direct brick commands with appropriate parameters to move the robot as expected.

```
private async void goForward()
{
   _brick.DirectCommand.TurnMotorAtPowerAsync(OutputPort.B|OutputPort.D,20);
}


private async void turnRight()
{
   _brick.DirectCommand.TurnMotorAtPowerAsync(OutputPort.B, 20);
   _brick.DirectCommand.TurnMotorAtPowerAsync(OutputPort.D, 0);
}
```

2. Laboratory on LEGO Mindstorms - Programming EV3 in C#

2.1. Leap Motion Overview

The Leap Motion is an optic hand tracking device. It is able to track both hands and all 10 fingers as they move through the open space between the user and the controller, letting interaction with over 140 built-in apps for motion control. The Leap Motion Controller (LMC) itself uses optical sensors and InfraRed (IR) light imaging to define the position of predefined objects in real time. Updates are provided as a set (frame) of data. Each Frame object, representing a frame, contains properties (positions, orientations, gestures and motions) of all tracked objects at single moment of time. Then, the sensor software analyzes the objects movements detected in the device’s field of view and translates it into actions to be developed. Actually a very small amount of processing is done on the device itself, in order to keep the cost of the units low. All the images are post-processed on the developer’s computer. There the noises are being removed and the model of all objects is being created.

There are basically four classes that can be treated as the data model for the Leap Motion:

2.1.1. Hand

- Reports the physical characteristics of a detected hand.
- It's tracking data includes a palm position and velocity, vectors for the palm normal and direction to the fingers and lists of the attached fingers.

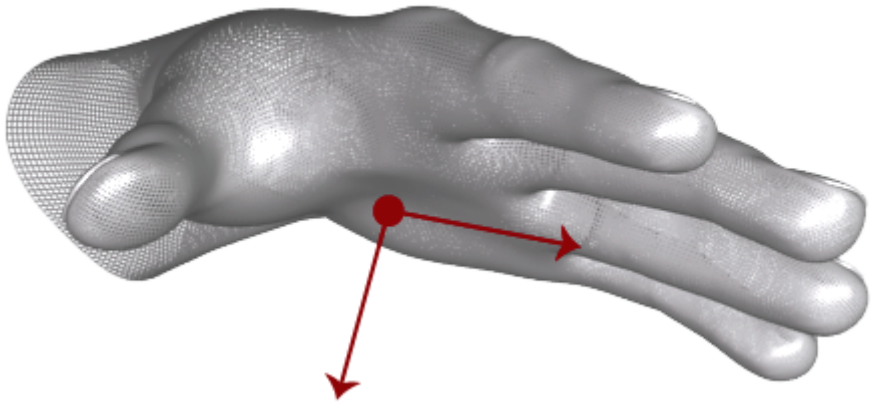


Fig. 1: The Hand PalmNormal and Direction define the orientation of the hand. Source

2.1.2. Arm

- Bone-like class that represents the forearm.
- Provides the orientation, length width and end points of an arm.

2.1.3. Finger

- Class representing a tracked finger.
- Valid finger objects can be obtained from a Frame or Hand object.
- Information is provided for each finger on the tracked hand (characteristics or partially/not visible fingers are estimated)
- Fingers are identified by type name, i.e. thumb, index.



Fig. 2: Finger TipPosition and Direction vectors provide the position of a finger tip and the general direction in which a finger is pointing. Source

2.1.4. Bone

- Class representing a tracked bone.
- Provided by a Finger object.

- Bones are ordered from base to tip, indexed from 0 to 3 (the thumb does not have a base metacarpal bone and therefore contains a valid, zero length at that location).

2.2. Coordinates System

The Leap Motion system employs a right-handed Cartesian coordinate system. The origin is centered at the top of the Leap Motion Controller. The x- and z-axes lie in the horizontal plane, with the x-axis running parallel to the long edge of the device. The y-axis is vertical, with positive values increasing upwards (in contrast to the downward orientation of most computer graphics coordinate systems). The z-axis has positive values increasing toward the user.

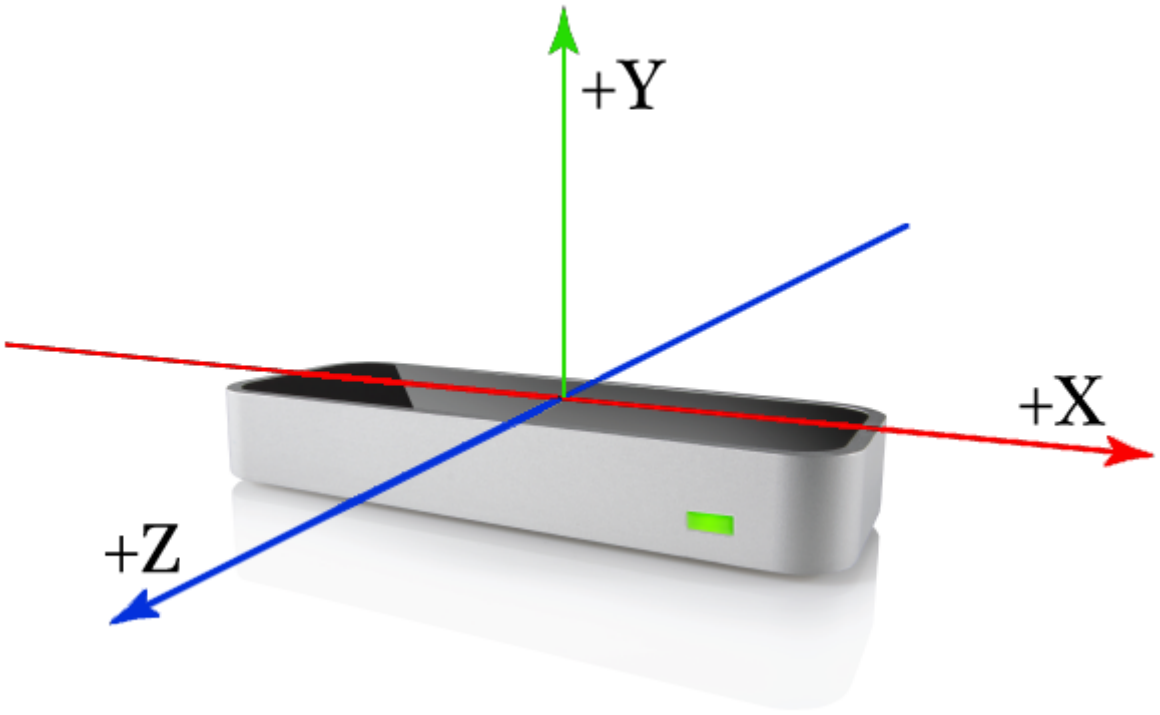


Fig. 3: The Leap Motion right-handed coordinate system. Source

2.3. System Architecture

On Windows, the Leap Motion software runs as a service. The physical connection between the Leap Motion Controller and the computer takes place via USB bus. Leap-enabled applications access the Leap Motion service to receive motion tracking data. The Leap Motion SDK provides two different APIs to access the tracking data: a native interface and the WebSocket interface.

2.3.1. Native Application Interface

- Unordered List ItemProvided through dynamically loaded library, that connects to the Leap Motion service and provides the data to the application.
- In case of C# the library is not linked directly, but with use of the special wrapper for binding

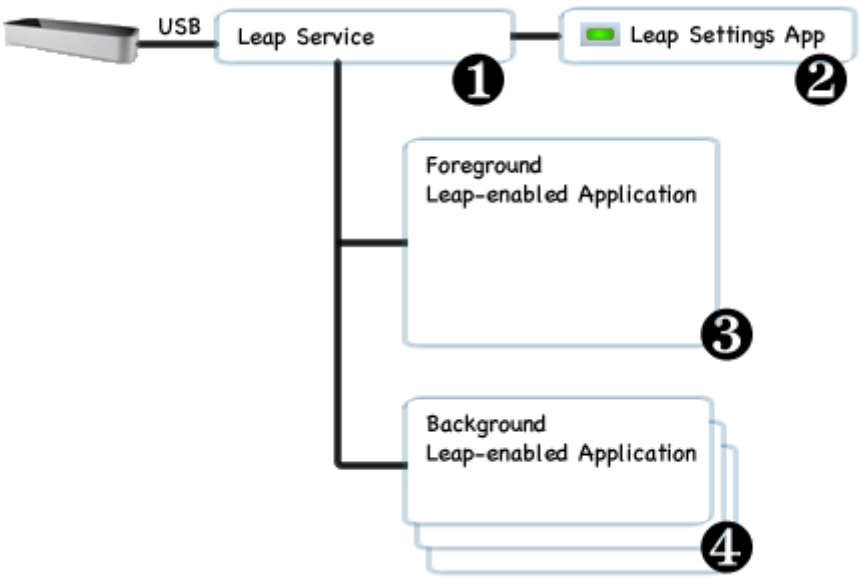


Fig. 4: Leap-enabled application. Source

2.3.2. WebSocket Interface

- Unordered List ItemService runs a WebSocket server on the localhost, port: 6437
- Tracking data in the form of JSON messages
- JavaScript client is available for convenience

2.4. Setting Up a Project

The Motion Leap libraries (file LeapCSharp.NETx.x.dll located in the lib folder of the Leap SDK package) are expected to be loaded from the same directory as the executable of the application, so it’s required that SDK is downloaded. The linking step will be covered later.

2.4.1. Hello World C#

- Create new project of the WPF or Windows Forms type.
- Add a reference to one of the pre-built libraries:
- Select Project > Add Reference.
- In the References dialog, select Browse.
- Browse to Leap Motion SDK folder.
- Choose the appropriate library for your project (depending on the version of .NET being used)
- Set the target platform to x86 or x64
- Select Project > Properties.
- Select Build page.
- Set the Configuration to All Configurations.
- Under General, set Platform target to x86 or x64.
- Edit the Post Build Event to copy the native libraries to the project’s target executable directory.



If you are creating a 64-bit application, use the libraries in the lib/x64 directory, not the 32-bit libraries in the lib/x86 directory.

- Open the project properties page.
- Select the Build Events page.
- Click Edit Post-build.
- Edit the event field to copy the libraries:
- For 32-bit projects add:

```
xcopy /yr "$(LEAP_SDK)\lib\x86\LeapC.dll" "$(TargetDir)"
```

- For 64-bit projects add:

```
xcopy /yr "$(LEAP_SDK)\lib\x64\LeapC.dll" "$(TargetDir)"
```

- Add your source code!

2.5. Sample source code

Sample source code can be found here:

leapexample.zip



Make sure that the reference to Leap Motion dll library provided in project is valid

2.6. Citations

All citations, used images and more information about Leap Motion can be found here:

- Sample_Tutorial
- Leap_Overview
- Leap_Architecture
- Leap_Coordinate_Mapping

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:macro:lego:lab:csharp

Last update: **2018/01/28 05:59**