LEAP Motion Sensor Implementation in Robotics Control Systems

Joe Long
Masters of Science in Computer Science Program

Faculty Adviser:
Dr. CJ Chung

12/15/2016

Abstract

The LEAP Motion sensor, as a means to pass commands on to robots, presented a number of potential benefits to experiment with. One such basic implementation is standard steering and navigation from remote locations. To simulate this, a PC was setup with a LEAP Motion sensor. Meanwhile, another laptop was setup to act as a controller on the demo L2Bot. With both laptops connected via UDP sockets, the control station laptop can read in the LEAP sensor data, interpret it, then pass along commands to the remote laptop. This experiment is to test the feasibility of the LEAP sensor in such an environment as well as the general viability of virtual control systems for future implementation.

Table of Contents

LEAP Motion Sensor Implementation in Robotics Control Systems........1

Abstract........1

Method........2

Technologies Used........2

About the Transmitter / Control Station........3

Communications........6

About the Receiver and Robot Control........6

Appendices, Additional Resources, and Demonstrations........7

Special Thanks to… 7

Method

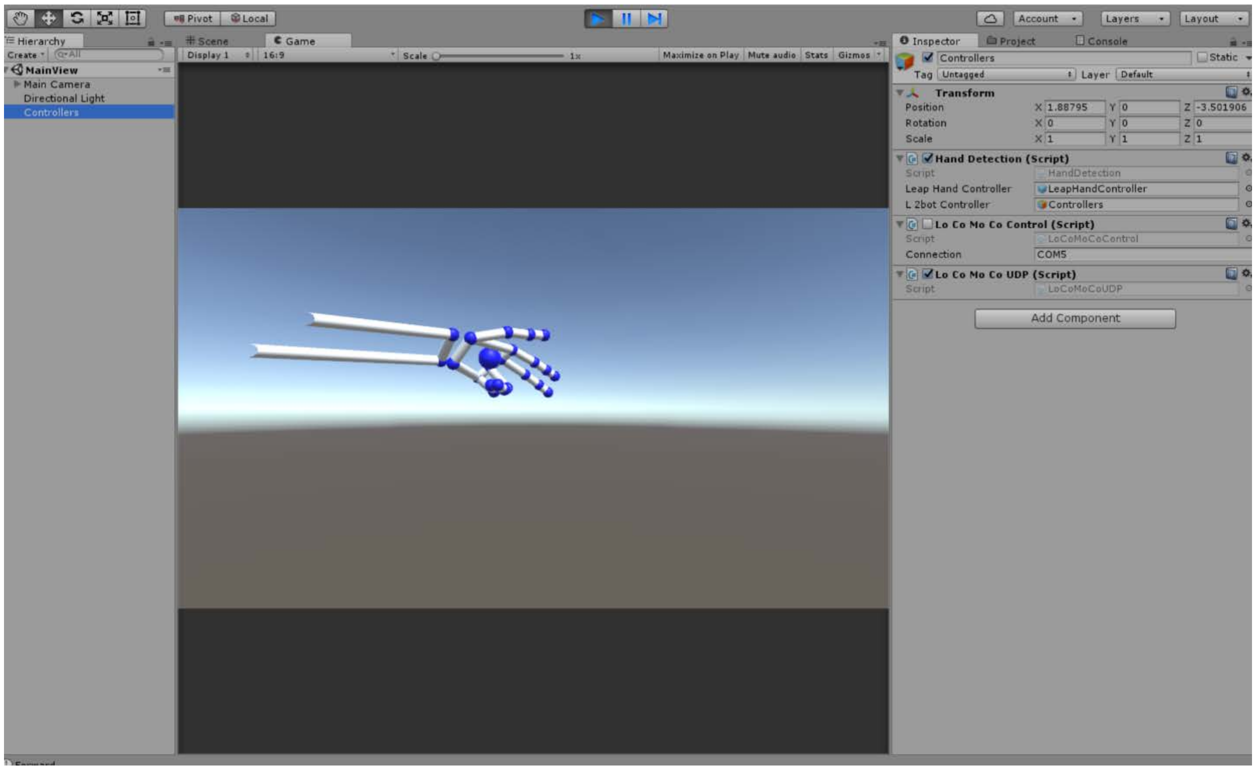
As a means of testing the LEAP sensor's viability in acting as a robot control mechanism, I've setup the LEAP sensor in tabletop mode, facing up toward the operator's hand. When only one hand is detected, it will then run through a series of conditionals to find out which pattern fits the finger positions at that moment. In the long run a more dynamic interpretation engine would prove helpful, but as a means to test the sensor itself this more than meets the need.

To simplify execution and keep the focus of the project on the LEAP sensor's interpretation layer, the communications and workflow of this project have been heavily simplified. Commands are read in by the C# script in Unity that then uses sockets to communicate with the PC on the robot.

Technologies Used

- Unity (using C# and .NET 2.0)
- LEAP Motion Sensor
- Two laptops, one as the receiver with the receiver's .exe file copied to it, one as the sender with the sender.exe file on it.
- L2Bot
- Standard home router, no special configuration

About the Transmitter / Control Station



Controllers Static
Tag Untagged Layer Default

Transform

Position	X	1.88795	Y	0	Z	-3.501906
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Hand Detection (Script)
Script HandDetection
Leap Hand Controller LeapHandController
L 2bot Controller Controllers

Lo Co Mo Co Control (Script)
Script LoCoMoCoControl
Connection COM5

Lo Co Mo Co UDP (Script)
Script LoCoMoCoUDP

Introduction and Implementation in the Unity Environment

The transmitter uses the Unity environment, primarily because we found that the Unity implementation of the LEAP Motion libraries offered the most accurate results. It also allowed for a pre-built method of visualizing the hand model on the screen and having scripts continue to run in the background. Unity itself is only getting the most basic of uses, the screen display only shows the primary frames and no other wireframe elements are in scope short of the hand objects being controlled by the LEAP Sensor prefab objects.

Directional Interpretation

With Unity handling most of the timing and visual support, the development could focus solely on the LEAP libraries, how hand and joint positions are detected, and using this data to interpret gestures and direction. The current system is very rudimentary steering system, interpreting hand gestures into one of seven commands that are then sent over the network to the receiver.

Gestures are made over the sensor while standing directly behind it and the operating laptop, making sure that the sensor is perpendicular to the operator’s body. The nine gesture commands detected in the current version are:

- Stop: Hold a closed fist over the sensor

```
if(fingers.Count(f => f.IsExtended) == 0)
{
    Debug.Log("Stop");
    SendRobotCommand(LoCoMoCo.RobotCommmands.Stop);
}
```

- Coast: No hands detected over the sensor

```
if(currentFrame.Hands.Count == 0)
{
    Debug.Log("No hands found: Coast");
    SendRobotCommand(LoCoMoCo.RobotCommmands.Coast);
}
```

- Forward: Point two or three fingers forward

```
if(pointerFinger.Direction.Yaw >= 2.9 || pointerFinger.Direction.Yaw <= -2.9)
{
    Debug.Log("Forward");
    SendRobotCommand(LoCoMoCo.RobotCommmands.Forward);
}
```

- Bank Left: Point two or three fingers about 20 degrees to the left of straight forward

```
// Yaw is between -2.6 and -2.9
if (pointerFinger.Direction.Yaw <= -2.6 && pointerFinger.Direction.Yaw > -2.9)
{
    Debug.Log("Bank Left");
    SendRobotCommand(LoCoMoCo.RobotCommmands.BankLeft);
}
```

- Bank Right: Point two or three fingers about 20 degrees to the right of straight forward

```
// Yaw is between 2.6 and 2.9
if (pointerFinger.Direction.Yaw >= 2.6 && pointerFinger.Direction.Yaw < 2.9)
{
    Debug.Log("Bank Right");
    SendRobotCommand(LoCoMoCo.RobotCommmands.BankRight);
}
```

- Turn Left: Point two or three fingers about 45 degrees to the left of straight ahead

```
// Yaw is greater than than -2.6
if (pointerFinger.Direction.Yaw > -2.6)
{
    Debug.Log("Turn Left");
    SendRobotCommand(LoCoMoCo.RobotCommmands.TurnLeft);
}
```

- Turn Right: Point two or three fingers about 45 degrees to the right of straight ahead

```
// Yaw is less than 2.6
if (pointerFinger.Direction.Yaw < 2.6)
{
    Debug.Log("Turn Right");
    SendRobotCommand(LoCoMoCo.RobotCommmands.TurnRight);
}
```

- Pivot Left: Hold a closed fist over the sensor with your thumb extended to the left

```
if(thumb.Direction.Roll < -1.1 && thumb.Direction.Roll > -1.8)
{
    Debug.Log("Pivot left");
    SendRobotCommand(LoCoMoCo.RobotCommmands.PivotLeft);
}
```

- Pivot Right: Hold a closed fist over the sensor with your thumb extended to the right

```
if ((thumb.Direction.Roll > 1.1 && thumb.Direction.Roll < 3.1) ||
    (thumb.Direction.Roll < -2 && thumb.Direction.Roll > -3.1))
{
    Debug.Log("Pivot right");
    SendRobotCommand(LoCoMoCo.RobotCommmands.PivotRight);
}
```

A demonstration of the gesture controls can be found at:
https://youtu.be/s0ScYDyKMQs

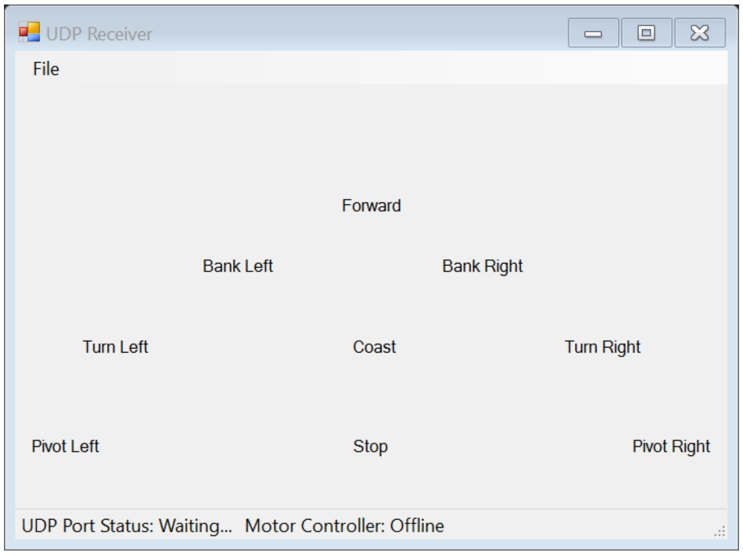
The Need for Additional Communication Methods
The original concept for this was on a riding robot, where the operator could be physically on the robot steering it using these hand gestures on a virtual dashboard of sorts. When moving the test case over to the L2Bot though it was apparent that wireless communication of some sort would be needed, as the operator can't exactly be following around the L2Bot while holding their hand over it and get accurate results.

The project scope was then expanded to include close range, wireless communication between a control station then an additional PC that would be on the L2Bot, receive the commands, then control the L2Bot accordingly. The simplest method was to use a UDP socket connection over a home router as a general proof of concept. More advanced versions could include sockets over cellular connections, TCP to confirm connection states, or additional socket connections to allow for two-way communication, but the current proof of concept is a one-way street simply passing direction instructions at 30 messages per second.

Communications

The two laptops communicate using a rebuilt version of the existing LoCoMoCo class. On its own, LoCoMoCo allows for direct communication between a laptop and a motor controller. The modified version is a DLL that packages the LoCoMoCo class along with additional classes for master/slave UDP connections, along with the various support functions needed to store and send commands, then translate the commands back into the instructions passed to the L2Bot. Future plans are slated to include also include Bluetooth, TCP, or other protocols.

About the Receiver and Robot Control



UDP Port Status: Waiting... Motor Controller: Offline

The UDP Receiver application is the simplest component of the set. It listens for UDP connections coming in over a set port, translates the command flag into an executable robot command, then passes that command onto the LoCoMoCo motor controller. As a visual, as commands are received, the corresponding directional command in the form UI lights up.

The primary component in this module is the implementation of a robot command library and testing receiving rates for different commands and how passing commands can be optimized. This variant uses integers that correspond with C# enum values as shown below:

Stop:	Both motors stop rotating
Coast:	Both motors fall into a coast state, will take ~5" to coast to a stop
Forward:	Both motors forward at full power
Bank Left:	Left motor on coast, right on full power forward
Bank Right:	Right motor on coast, left on full power forward
Turn Left:	Left motor stopped, right on full power forward
Turn Right:	Right motor on coast, left on full power forward
Pivot Left:	Left motor full power reverse, right motor full power forward
Pivot Right:	Right motor full power reverse, left motor full power forward

With this success, it's also possible to start sending more complex data, possibly serialized objects that correspond with motor directions or simple XML data being sent as strings. The primary weakness with the current structure is that enum elements would have to be created for every permutation of each wheel being in a forward, reverse, coast, or stop states, making this integer sending method quite restrictive.

Appendices, Additional Resources, and Demonstrations

To learn more about using the LEAP sensor with Unity:

- LEAP Motion's C# Documentation: https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html
- Getting started in Unity: https://unity3d.com/learn/tutorials/topics/developer-advice/how-start-your-game-development
- LEAP and Unity Setup Training: https://developer.leapmotion.com/unity

A full demonstration of this test robot control system can be found at:
https://youtu.be/rSMKtIB0Gtw

Special Thanks to…

Prof. Gordon Stein
For sample material integrating Unity with the LEAP sensor

Adeline Miller
For development and research assistance