Programming the Meccanoid using the Arduino, Kinect SDK, and Visual Studio

Quick Intro

This guide outlines the process of interfacing the Meccanoid robot with Microsoft Kinect. The guide can be used, however, simply for interfacing the Kinect with the Arduino, or the Arduino with the Meccanoid, or the USB2Serial with the Arduino. The guide is split into clearly defined sections so that you can easily access just what you need.

Interfacing the Kinect with the Meccanoid is a very doable process, but without a decent guide can take weeks to figure out. There is no tutorial that goes through the entire process. There are tutorials online that help with certain parts of the process, but most are difficult to find and many of them outdated. So that's why we made this. Note, this guide **requires a device running Windows.**

Using the Kinect Software

Step 1: Install the Kinect SDK

The version of the Kinect for Windows SDK that you install will depend on your computer's operating system and the type of Kinect sensor you are using. You should download the most recent (viable) version of the Kinect SDK.

Note: If you are using the Kinect for Windows v2 or Kinect for Xbox One you MUST download the Kinect SDK 2+. Additionally, the Kinect SDK 2+ only supports Windows 8+.

Download SDK 1.8: https://www.microsoft.com/en-us/download/details.aspx?id=40278
Download SDK 2.0: https://www.microsoft.com/en-us/download/details.aspx?id=44561

Step 2: Install Microsoft Visual Studio

This is the IDE you will need to utilize the Kinect Software. You can download Visual Studio Community 2015 here: https://www.visualstudio.com/en-us/visual-studio-homepage-vs.aspx.

In case you already have another version of VS installed on your computer: Other versions of VS should also work, but some are not supported depending on which version of the Kinect SDK you are using. For example, SDK 2.0 requires VS 2012 or newer. Ensure your versions are compatible.

Step 3: Install the Kinect Drivers

Power the Kinect and plug it into your computer's USB port. Your computer should automatically install the Kinect drivers. Afterwards, check the light on the Kinect's power-box. An orange light indicates a problem. Otherwise, your Kinect is ready to go.

Step 4: Getting Started with Kinect Programming

Our guide uses C# within Visual Studio to utilize the Kinect software and create Windows applications. If you are unfamiliar with C#, this website provides a quick but comprehensive reference:
https://learnxinyminutes.com/docs/csharp/

This tutorial goes through the process of creating a basic WPF application with Kinect functionality:
http://www.egr.msu.edu/classes/ece480/capstone/spring15/group02/assets/docs/dzappnote.pdf
Note that "reader" must be declared as a "DepthFrameReader" not a "MultiSourceFrameReader."

Step 5: Advanced Use of the Kinect SDK

For more advanced usage of the Kinect SDK, see the documentation and programming examples that Microsoft provides here: https://msdn.microsoft.com/en-us/library/dn799271.aspx

Using the Arduino to Control the Meccanoid

Step 1: Install the Meccanoid Arduino Library

Thankfully, Meccano provides a library that allows the Arduino to control the Meccanoid hardware. Download it here: http://www.meccano.com/meccanoid-opensource

To install the library, open the Arduino IDE. Go to Sketch-->Include Library-->Add a .ZIP Library. Select the .zip file you downloaded.

Step 2: Understanding the Hardware

For a brief understanding of the Meccanoid hardware, including smart servos and LED's, see here:
http://cdn.meccano.com/open-source/Meccano_SmartModuleProtocols_2015.pdf

NOTE: The smart servos are EXTREMELY BUGGY when daisy chained. Sometimes servos in the same chain will "link" together. In our case for example, the 1st and 2nd servos on the left arm linked together, both moving in response to data sent to the 1st servo. The 3rd servo would respond to data being sent to the 2nd servo. We never determined the cause of this issue. You can, however, work around it by simply isolating the servo that causes issues. In our case, we removed the 2nd servo from the chain, and connected it to the Arduino as its own chain.

One last pro tip: The Arduino powered through the computer sometimes isn't strong enough to lift the Meccanoid's arms. Powering the Arduino with an external battery (we used a 7.4V) can solve this issue.

Step 3: Programming the Meccanoid

The Meccanoid Open Source Programming page (the page from Step 1 with the Arduino library) also contains an example project. Download it. The "Claw_Bot.ino" file provides a great example of how to use the library.

For further information on using the Arduino with the Meccanoid, see this post on the Arduino forum:
https://forum.arduino.cc/index.php?topic=368782.0
It's not much, but it's just about the only information out there.

Communicating between Arduino and Visual Studio

Step 1: Basic Serial Communication between Arduino and VS

This tutorial goes through the process of setting up a WPF application for basic serial communication with the Arduino IDE:
http://www.c-sharpcorner.com/UploadFile/e46423/arduino-control-using-a-windows-presentation-foundation-(wpf/
You can skip the part about using an on/off switch within VS, and just pre-program the WPF application to send either '0' or '1'.

Step 2: Advanced Serial Communication

The tutorial from Step 1 provides a great overview of setting up serial communication between the Arduino and VS, but things get a little more complicated when you're reading more than just a single char, which you will likely be doing if you're sending substantial amounts of data to the Arduino.

Read(), readString(), and readStringUntil() are all functions that Serial uses on the Arduino. They are very different functions and it's important to understand what they do if you plan on using them. Serial communication is not as simple as you might assume. This blog provides a decent explanation of how it works: https://hackingmajenkoblog.wordpress.com/2016/02/01/reading-serial-on-the-arduino/

Extra: Potential Issues

We suffered extreme latency issues (of around 3-4 seconds) while communication between VS and the Arduino. We fixed this issue by only sending data from VS every 10 frames. Additionally, we created a timer to clear the serial port out buffer every second, however, we did not determine whether this timer was needed. It may not be necessary.

Using the USB2Serial for Debugging

Purpose

A serial port on the Arduino cannot simultaneously read and write. So, if the serial port is receiving data from VS for example, it cannot write to the serial monitor. This makes debugging very difficult, as print statements become impossible. Using the USB2Serial, though, a second serial port can be opened for debugging purposes.

Hardware Setup

Plug the RX of the USB2Serial into a TX (this CANNOT be TX0, use TX1+) on the Arduino. Plug the TX into the corresponding RX. As an example, TX/RX 1 on an Arduino Mega are pins 18/19.

That's it. The 5V, ground, and reset on the USB2Serial need not be used.

Software Setup

Now, you can program the serial port you used just like you normally program "Serial". If you used TX/RX 1, for example, you simply program using "Serial1".

In order to change the output of the serial monitor to the proper serial port, go to Tools --> Port. Select the port to which USB2Serial is connected. You can only do this AFTER uploading your sketch to the Arduino.

Created on 8/26/16 by Luigi Mangione
Questions? Contact me at lnmangione@gmail.com