Depth Imaging with the Asus Xtion Pro Live Sensor and the KIPR Link (Part 1)
Jeremy Rand
Team SNARC (Sooners / Norman Advanced Robotics Coalition)
jeremy.rand@ou.edu

Depth Imaging with the Asus Xtion Pro Live Sensor and the KIPR Link (Part 1)

1 Introduction

Botball has historically not changed much over the years, particularly in terms of sensors available. If you participated successfully in Botball circa 2006, you probably would do fine in 2013 with minimal retraining. The differing game board layouts and occasional game mechanic quirks do not hide the fact that the same skill set has been used in Botball for years, because the robots are still using the same sensors (rangefinders, brightness sensors, touch sensors, and color blob trackers).

The most recent time that an entirely new sensor showed up in Botball was in 2005, when the Xport Botball Controller (XBC) introduced PID motor control. PID motor control allows extremely precise dead-reckoning with almost no programming effort by the user. Prior to 2005, it was rare to see a Botball robot that could make accurate turns or drive in a straight line. These days, we've observed experienced Botball programmers show newbies how to do both things reliably in about 10 minutes.

Needless to say, PID motor control made an enormous impact on how the students spent time solving problems. Instead of worrying about whether a robot can drive straight, teams instead focus on more high-level tasks, e.g. finding scoring objects and areas effectively.

We think another 2005-like event is coming. Technology has advanced since 2005, and we need a new sensor in the Botball kit, one which should change the game once more. We think the Asus Xtion Pro Live may be that sensor. But first, the obligatory disclaimer.

DISCLAIMER: Using the Xtion Pro Live with the KIPR Link requires hacking the Link via SSH and SFTP. These are powerful tools, and if you do something wrong, there is a risk of impairing the operation of the Link. It appears that a full brick is unlikely, since in theory you should be able to reflash a firmware image to restore to factory default settings. However, KIPR does not guarantee this, and nor do we. No warranty is implied! (However, please do notify us if something bad happens, since we'd like to investigate any such occurrences.) If this scares

you, consider it a sign that using the Xtion Pro Live with the KIPR Link may not be for you.

2 Background

The Asus Xtion Pro Live sensor is a combination of an RGB camera with a depth camera. Its underlying technology will be familiar to many, because it uses the same sensing hardware as the Microsoft Kinect. Unlike the Kinect, the Xtion is USB-powered and is much smaller, making it more suitable to Botball-style robotics.

The Xtion is capable of generating a 640×480 image where each pixel contains a distance from the Xtion camera (precision 1mm). This image can be updated at 30fps. It also returns RGB color data, but this paper will primarily focus on the depth sensing capabilities.

3 Choice of Controller

Our goal was to use standard Botball hardware to interface with the Xtion. We initially attempted this on the CBC Botball Controller in 2012, but its ancient Linux kernel and libusb version made the project impractical. Because the 2012 KIPR Open game did not yield an immediate advantage for this type of sensing, we decided to scrap the project for the 2012 season.

We were excited when the KIPR Link was announced, because its much newer kernel and more standard architecture meant that we could probably use existing software as a jumping point. As soon as we got our hands on a Link and an Xtion, we started hacking away.

4 Loading Files onto the Link

The Link has built-in WiFi and runs an SSH daemon. This means that, similarly to the CBC [1], it is possible to get a terminal prompt from a PC to perform shell commands. However, the Link is running a much newer version of the SSH daemon than the CBC, and as a result also supports the SFTP protocol. This means that a hacker can simply download an SFTP client (we're fans of Filezilla [2]), connect to the Link's IP (username is root, no password), and get browsable, drag-and-drop, read/write access to the Link's filesystem. This is much more user-friendly than the SCP-based methods which the CBC required.

We decided to place our files in the /kovan/binaries/ directory. We did this because the Link doesn't do anything automatically with these files, and also we can browse these files from the Link's file manager to see that the files are present.

5 Choice of Library

We had to choose a library with which to access the Xtion. The two that we examined were libfreenect and OpenNI. Unfortunately, a quick Google search revealed that, as recently as a year ago, many people were having issues accessing the Xtion from libfreenect (only the Kinect hardware seemed to be well-supported). OpenNI did not appear to have these issues, so we decided to go with OpenNI.

6 Building OpenNI For the Link

OpenNI provides ARM-Linux binaries for their libraries and example programs. Since we're lazy (like most programmers), we downloaded the binaries and copied them to the Link using SFTP. Unfortunately, when we ran the "SimpleRead" example, we got an "Illegal Instruction" error.

After doing this, we noticed that the OpenNI release notes included with the download state that the supported ARM processors are "Arm Cortex A8 and above." This makes sense, since the Link does not meet that requirement and therefore would not be able to execute the instructions in the binary.

As a result, we decided to try to build OpenNI ourselves. We downloaded a .zip archive of the OpenNI2 GitHub repo [3], and copied its contents to the Link via SFTP. We ran the "make" command from an ssh terminal, and after waiting a few minutes, we were greeted with an error about libusb-1.0 not being found.

Conveniently, the Link uses the Angstrom distribution [4], which includes a package manager. This is much better than the CBC, which made installing libraries extremely difficult. After Googling for the syntax [5], we found that this command will install libusb-1.0 from an SSH terminal:

```
opkg update
opkg install libusb-1.0-dev
sync
```

After installing libusb-1.0, we ran the "make" command again from an SSH terminal, and the operation finished without errors after waiting a few minutes. However, running the "SimpleRead" example gave us the same "Illegal Instruction" error we had gotten with the pre-built binaries.

We then noticed that in the output of the "make" command, the compiler was being passed some suspicious flags, such as `-mtune=cortex-a8`. After looking through the makefiles, we found that the file `OpenNI2-master/ThirdParty/PSCommon/BuildSystem/Platform.Arm` was passing the following flags:

```
# Hardware specifying flags
CFLAGS += -march=armv7-a -mtune=cortex-a8 -mfpu=neon -mfloat-abi=softfp #-
mcpu=cortex-a8
```

This confirmed that the library was building by default for the Cortex A8 platform, which is incompatible with the Link. We weren't sure why the makefile would force a specific ARM hardware target, but we figured it couldn't hurt to try removing the offending flags. We removed all of the flags except for `-mfloat-abi=softfp`, since we were fairly sure that the Link used software floating-point. After running "make" from an SSH terminal, we *still* got the "Illegal Instruction" error upon running the "SimpleRead" example.

Getting frustrated, we Googled for what exactly the "softfp" flag did, and learned that contrary to what we assumed, the "softfp" flag actually forces usage of hardware floating-point. (We have no idea why they used that name....) So, we removed the entire line of hardware specifying flags, and reran the "make" command from an SSH terminal. And finally, we ran the "SimpleRead" example from SSH, and were delighted to find that the example ran, and promptly told us that no device was plugged in (which was correct; the Xtion wasn't plugged into the Link at that point).

We then plugged the Xtion into the Link's USB port, reran the example via SSH, and the example worked perfectly, showing us the depth reading of the center pixel, updating at approximately 30fps.

7 Running from the Link GUI

While the binary of the example program was visible from the Link's file browser, we couldn't find an easy way to execute it from the GUI. We had two choices: hack the Link's GUI, or code a bootloader in C. While hacking the Link's GUI is probably a more robust solution, we were in a hurry, so we decided to use a bootloader.

The basic idea of a bootloader is to write a simple C program, which can be loaded via flash drive or USB download, whose sole purpose is to boot another program (the OpenNI example program). Our bootloader consists of:

```
int main()
{
```

```
    printf("OpenNI Bootloader for Kovan\n\n");

    printf("Killing old process...\n");
    system("killall SimpleRead");

    printf("\nChanging working directory...\n");
    chdir("/kovan/binaries/OpenNI/OpenNI2-master/Bin/Arm-Release/");

    printf("\nRunning program...\n\n");
    system("./SimpleRead");

    printf("\n\nExecution finished.\n");
}
```

We weren't sure if the emergency-stop feature would properly kill the OpenNI example since it wasn't the program that the Link launched (that would be the bootloader), so we included a "killall" command to stop any existing instance of the example program prior to running it. As it turned out, preliminary testing suggests that this isn't necessary, but we decided to err on the side of caution and leave that code in there.

We perform a "chdir" operation before running the program because the working directory is used to find some shared library files. Running the program without "chdir" results in a missing library error.

After compiling this code via the Link GUI, running the bootloader program resulted in the OpenNI example executing. However, the text output lagged horribly, and only updated around once per 5 seconds. We decided to investigate that later.

8 End of Part 1

This paper was too big to fit, so we've split it into two parts. Part 2 will continue where we left off here. See you in Part 2!

References

[1] J. Rand, M. Thompson, B. McDorman. *Hacking the CBC Botball Controller: Because It Wouldn't Be a Botball Controller if It Couldn't Be Hacked*. Proceedings of the 2009 Global Conference on Educational Robotics, 2009.
[2] Tim Kosse. FileZilla. https://filezilla-project.org/ , 2013.
[3] OpenNI. OpenNI2. https://github.com/OpenNI/OpenNI2 , 2013.
[4] Angstrom Contributors. The Angstrom Distribution: Embedded Power. http://www.angstrom-distribution.org/building-angstrom , 2012.

[5] mochad Contributors. BeagleBone running Angstrom Linux. http://sourceforge.net/apps/mediawiki/mochad/index.php?title=BeagleBone_running_Angstrom_Linux , 2012.