A Guide to the Arduino YUN for Network Robotics

UVA Stage Robot Documentation

Section by Andy Carluccio

Introduction

The Arduino YUN is an extraordinary piece of hardware that is made infuriatingly difficult to use because of a poor software deployment and objectively false documentation. Regardless of its many issues, the YUN is the right tool for many networked robotics projects. This section of the Stage Robot Documentation attempts to alleviate the stress on the developer by explaining the exact deployment of the YUN that was found to be reliable. The developer will hopefully be able to create a mental model of the operation of the YUN through exploration of this guide. It is important to read this section of the documentation completely through before attempting to follow the setup instructions.

Device Overview – What is the YUN?

The Arduino YUN Rev2 (not to be confused with the Rev1, the YUN Shield, or the YUN mini, all of which are commonly referred to as "the YUN" online, much to the dismay of debuggers) is a microcontroller consisting of a ATmega32u4 processor for Arduino sketches and a Atheros AR9331 MIPS processor responsible for running a Linux distribution, a modified flavor of OpenWRT. The ATmega32u4, hereon the "Arduino processor," controls the device-facing side of the YUN, controlling pin voltages and serial communication as any Arduino Uno does. The Atheros AR9331, heron the "Linux processor," controls the world-facing functionality of the YUN, which includes the SD card, the USB port, the flash file system, and most notably, the WiFI module.

The genius of the YUN is that both of these processors can communicate with each other over a modified serial connection called the "Bridge." This opens the door to a massive world of possibilities for networked robotics projects. Obviously, being able to access web data from an Arduino sketch is a major benefit of this setup, but having a full Linux deployment in close communication with the Arduino microcontroller means that complex scripts can be written to interpret data from the world-facing devices and produce incredibly low-level results through execution from the Arduino processor.

The Achilles Heel of the YUN is its documentation, or lack thereof. The information published on the Arduino website is often factually incorrect about facets of current YUN devices being shipped, and many critical components of operating the YUN to its fullest potential are missing from the documentation and have instead been inferred from a web of forum posts where YUN users try to piece the project back together. Some have even gone as far as to author entirely new system images for the YUN, a fact this guide will exploit in the setup of the Stage Robot. Thankfully, the Arduino-side of the YUN shares the majority of its documentation with the YUN, and while few YUN-specific examples worth anything exist on the Arduino website, the API for Arduino sketches is fairly complete.

Into the Tutorial – What You Will Need

In order to complete this tutorial, please secure the following resources if you have not used them yet in other sections of the documentation:

1. An Arduino YUN Rev2
2. A network router with both WiFi 2.4GHz and LAN connections and an outbound connection to the Internet (for now, and this can also be bridged from another computer).
3. A Windows computer on the network with the following software installed:
    a. Arduino IDE
    b. WinSCP
    c. PuTTY
    d. TFTPUtil
4. An Ethernet cable connecting the YUN to the router
5. Your robot, configured following this documentation
6. Completed download of the YUN images, pip archive file, and Python and Arduino code

Phase 0 – Get the YUN on the Correct Network

If you have not played with the Arduino YUN yet, you need to get it on your robot's network before you can follow this tutorial. By default, the YUN will emit a local WiFi hotpot if it does not already have a network to connect to (or if that network disappears at any time). The YUN will remain "stranded" if this happens until explicitly reminded of the correct network to connect to.

Connect to the Arduino's hotspot, and then access its web control interface by either typing in http://arduino.local into a web browser or, more reliably, entering the default IP address of the YUN, which is 192.168.240.1 . The default password is "arduino" without the quotation marks. Inside the editor, you must configure the YUN to connect to your network. You should also give the YUN a name, such as myYUN or robotBrain, and set a new password. After you fill out this information, the YUN will prompt you to change networks, and if you do this before the progress bar completes, you might be able to access the web interface again by clicking the hyperlink, but if that fails, you need to figure out what the IP of the YUN was set to by your router so you can access the web interface again in the future. Note this IP, as it will be needed later as well.

Phase 1 – Flash the YUN

The version of the YUN that ships at the time of the writing of this documentation does not seem to comply with any known documentation, and for this purpose, it is essential to flash the YUN to a known-working state where the rest of the tutorial can build from.

This phase draws from this tutorial, though not exactly:
https://www.arduino.cc/en/Tutorial/YunUBootReflash

The YUN has three sections of flash data, each of which needs to be erased and replaced in order. Follow the instructions for setting up a TFTP server on your computer, but target the images from this repository instead of the ones provided by Arduino.

With your TFTP server online and connected to the robot's network and the YUN connected to your computer over USB and to the network over both WiFi and Ethernet, upload the included YunSerialMonitor sketch to the YUN board and open the serial monitor window.

At this time, possibly by using the ipconfig command in Windows Command Prompt, determine the IP address of your computer.

The following steps occur in quick succession. With serial monitor open, press the physical button on the YUN labeled "YUN Reset" or some equivalent. When you press this button, the serial monitor should rapidly populate, and you have around four seconds to type "ard" (or whatever the last prompt on screen says depending on your version) and enter it into the YUN. If you are successful, you will get access to the bootloader's terminal.

Send the following commands over the serial monitor in order, waiting for the previous to complete and prompt you to enter the next command until you finish this list:

setenv serverip 192.168.1.4; ← replace this IP with your computer's IP, where TFTP server runs

setenv ipaddr 192.168.1.7; ← the YUN's IP

tftp 0x80060000 openwrt-ar71xx-generic-linino-u-boot.bin;

erase 0x9f000000 +0x40000;
cp.b $fileaddr 0x9f000000 $filesize;
erase 0x9f040000 +0x10000;

tftp 0x80060000 openwrt-ar71xx-generic-yun-16M-kernel.bin;

erase 0x9fEa0000 +0x140000;
cp.b $fileaddr 0x9fea0000 $filesize;

tftp 0x80060000 openwrt-ar71xx-generic-yun-16M-squashfs-sysupgrade.bin

erase 0x9f050000 +0xE50000;
cp.b $fileaddr 0x9f050000 $filesize;

bootm 0x9fea0000;

The YUN will now reboot itself and forget which network it was connected to, so you must repeat Phase 0. The YUN is now flashed to the proper image for this documentation.

Phase 2 – Install Python Requests Library

The Linux processor's feeble Bridge to the Arduino processor is managed by a mysterious Python library called, you guessed it, Bridge. This Bridge library is the main attraction for

development in Python on the YUN, because the lower-level access routines required for C/C++ to use the hardware bridge is usually beyond the scope of hobby electronics. That said, installing Python libraries is tedious. Thankfully, the heavy lifting has been accomplished already during the flash, so it is now possible to install the only library this tutorial requires, *requests*, somewhat "easily."

First, connect to the file hierarchy of the YUN using WinSCP. For reference:

File Protocol: SCP
Host name: the IP address of the YUN
Port number: 22
User name: root
Password: the password you set in the web interface

Go to the usr/lib folder and create a new folder. The tutorial names it "andyCode" for simplicity. Inside andyCode, drag and drop the pip tar file from your downloads. Creating it was a labor of love.

Open a PuTTY session. Assuming you know basic BASH commands, CD to the andyCode folder. If you do not, go learn some basic BASH commands. In your terminal, type:

```
opkg update
opkg install distribute
okpg install python-openssl
easy_install pip-6.0.8.tar.gz
```

These will take a fair amount of time to run. When they complete, delete the pip tar file from the YUN. Now run:

```
pip install requests
```

This will install the *requests* library on the YUN.

Phase 3 – Migrating the Python Code and Understanding the Bridge
Open the Python code in your IDE of choice, perhaps Visual Studio 2017. It is important to understand how it works in case it needs to be adapted. The first line is required for execution, and the following lines import the required libraries for web and bridge access and configure each. The infinite loop scrapes the robot's control website and parses it into individual strings separated by commas. There are eight such arguments and an additional clock report at the end. The clock is intended to check if the data has become "stale," possibly due to a network error. If the data is stale, a set of zeros will be printed, indicating an implicit stop command. Otherwise, the data will be updated to the integer representation offset by 127 to ensure that all values are positive or zero. Note that the buffer is flushed after each print. This is for Arduino processing.

The time is updated at the end of the loop, and a short sleep is added to keep the bridge from becoming backlogged with many more updates than the slower Arduino processor can parse in a reasonable amount of time. The value of this sleep command is entirely experimental and

constitutes a careful balancing act between the two processors to make sure the bridge does not hold data between the processors for too long before being acted upon. This supports the design philosophy of only sending data when it is needed, which is very helpful for reducing the latency between a drive command sent from the physical controller and the response from the robot. Note that this example sketch currently ignores encoder value parsing back from the Arduino processor to the Linux processor, and this functionality will be added in a near future update.

Drop the Python middleware code in the andyCode folder on the YUN. It will need privileges to execute:

chmod +x Middleware_for_Controller.py

Verify that the file is executable (will now appear green when using ls command)

Test it with the code:

python Middleware_for_Controller.py

and make sure it prints as expected. Interrupt it with CTRL+C when the functionality is confirmed. It will require the robot data server being active.

The Python code is now configured for execution on the YUN's Linux processor.

Phase 4 – Uploading and Understanding the Arduino Sketch

Open the provided Arduino sketch in the Arduino IDE and make sure the serial monitor is open. Understanding this firmware is also important for debugging and modification in the future.

The firmware uses the Software Serial library to implement the Sabertooth library because there is potential for a collision between Sabertooth and the Bridge library that can be avoided by routing through Software Serial. The Bridge library (now the Arduino Bridge library, not the mysterious Python Bridge library) is imported and configured. Process is where most of the Bridge communication will occur, though, so it is also imported.

Setup completes required tasks for the encoders and motors and runs the Python code on the Linux processor. This is achieved through the Process library's runShellCommandAsynchronously method, and this will essentially send a shell command to the Linux processor and then continue execution of the Arduino code regardless of what happens with that shell command on the Linux processor. As a result, the code must wait for p.available() to become true, which indicates that values are being printed from the Linux processor to the console (and thus, to the bridge). The code is now ready to enter the loop body.

Emergency stop is checked at the beginning to determine if a flag should be raised to activate the breaks. Note that regardless of if this code is executing properly, the segregated emergency stop system will cut power to the motors. Having the Arduino code running simply adds in an

additional breaking action so the robot will stop immediately versus drifting a bit further with momentum.

Each button on the controller has an integer value in the loop, and these are each updated by reading an integers from the bridge in sequential order or by implicitly setting them to zero if no new data is available. If both bumpers are depressed, execute a deliberate break. Otherwise, if the right trigger “dead man” is depressed, unlock the break and execute a drive based on the joystick data. If neither of the above but the aButton is depressed, execute some action for the lift depending on the dPad data. If no condition has yet been met, execute an implicit break.