# Datalogger Documentation

Huy Trinh

August 2021

## 0.1 Overview

The datalogger configuration includes

**Raspberry Pi 3 Model B + Vectornav VN200**

These can be found through these links

1. RPi: `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`
2. VectorNav
   2.1 Quick start manual:`https://www.vectornav.com/docs/default-source/datasheets/vn-200-datasheet-rev2.pdf?sfvrsn=e1a7b2a0_10`
   2.2 Complete manual page: `https://2w6vmg3m8cv83pn80b2dfi9f-wpengine.netdna-ssl.com/wp-content/uploads/assets/1/7/VN200UserManual_UM004_080514.pdf`
3. PoEHat: `https://www.raspberrypi.org/products/poe-hat/`

The GNSS/VectorNav connects to the mainboard (Raspberry Pi) through UART serial connection.

## 0.2 Hardware

### 0.2.1 Circuit

The USB power input is connected to 5V pins along with GNSS power. GNSS data is connected to UART pins. Power switch is connected between GPIO18 (BCM) and ground, pullup enabled in software. Primary LED (blue) is connected to GPIO24 (BCM), secondary LED (green) is connected to GPIO25 (BCM) and tertiary LED (red) is connected to GPIO10 (BCM). All the LEDs are driven with MOSFETs/NPN-transistors.More detail can found at: `https://tinyurl.com/ygbxmabs`

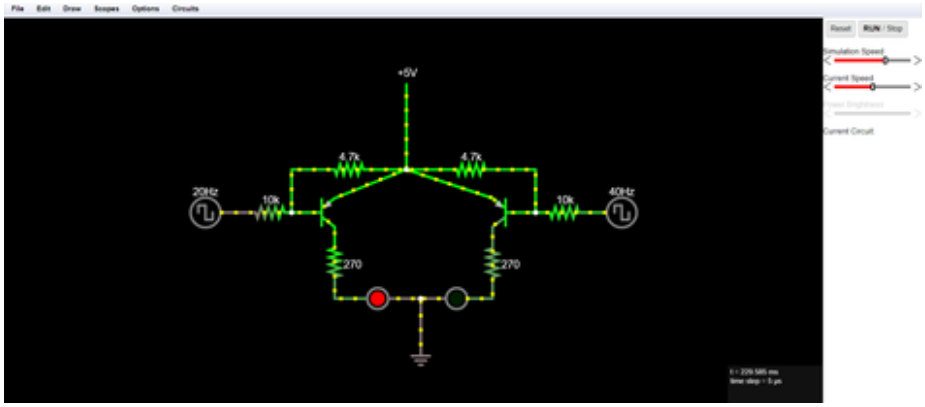


Figure 1: CircuitConnecting

### 0.2.2 SD Card

The SD should be 64 GB of space in order to store as much as possible the data collected from the datalogger.**Remember:** The origin image of the datalogger that we fetch is 32 GB, thus the minimum 32 GB space of SD card is required

### 0.2.3 USB 3G Modem

Originally Huawei E160E was used, which works well with 3G speeds. Currently Huawei E3372 4G modem is used (Hi-Link mode). Any modem, which has support on Linux probably can be made to work, some modems can draw more power though, especially LTE modems. On RPi **max_usb_current**=1 setting can be used in */boot/config.txt*.

## 0.3 Software

Software can be foud in  repository. Datalogger has accessed to the repo

### 0.3.1 Raspverry Pi Specific

The raspberry Pi run with operating system Raspbian Jessie with ROS Kinetic Kame, built from source with these instruction `http://wiki.ros.org/kinetic/Installation/Source`. More detailed instruction can be found below

**shutdown_flip.py**:
Listens to specific GPIO pin and initiates the shutdown sequence on switch flip

**led-pin.py**:
Listen to linux signals USR1 and USR2 to turn led on and off.

RPi has one or two pins for HW PWM, but apparently no sysfs interface by default

**led-linuxgpio.sh**:
Mainly control the led. Currently, the script is called in measurement script with inverse mode (mode 4). The inverse mode simply reverse the value of variable ON/OFF in its script.

**measurement.sh**
The script manage the running nodes nad check if the Internet connection. When the logger is offline, it starts the nodes and logging. When the internet connection is established, the logger uploads the measured data into github/gitlab account.

There are few setting at the beginning of the script, like log files and Leds connected. There are few functions for controlling Leds, currently, we used the led-linuxgpio.sh with inverse mode (mode 4).

Function check3G check whether there is *Huawei E160E data stick* is connected, and tries to connect to the internet using *sakis3g-script* (in our case there is not necessary since we are using the *E3372 dongle*).

When online for the first time, time is corrected using ntpdate from time1.mikes.fi ntp server and led is blinked fast. When internet is established, it will upload the data into the github/gitlab repository over SSH.

**start_script.sh**:
Start the **measurement.sh** script using flock. Flockfiles prevent multiple instances of measurement.sh from running at the same time.

### 0.3.2 Robotic Operating System ROS

ROS is an open-source, meta-operating system for your robot that providing you with services including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS distribution is a versioned set of ROS packages. In our case we use the **ROS Kinetic Kame** distro.

#### VectorNav node

Highly modified version of `https://github.com/dawonn/vectornav`. The data from INS is read over UART at maximum baudrate and then the data is published in custom ROS message (imugps) including necessary fields. THe node also automatically tries to change the baudrate if is not correct. It also includes

some configuration for the INS like GPS antenna offset and magnetometer velocity threshold. Also the LED control for turining the LED solid when the GPS fix is acquired.

### ROS Conceptual Terms

#### catkin workspace

catkin is official build system that combines CMake macros and Python script to provide some functionality on top of CMake's normal workflow. Catkin is more conventional than *rosbuild* in packages distribution, cross-compiling support and portability.

#### nodes

Nodes are processes that perform computation. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, one Node provides a graphical view of the system, and so on. A ROS node is written with the use of a ROS client library, such as roscpp or rospy.

#### rospy

rospy is a pure Python client library for ROS. The rospy client API enables Python programmers to quickly interface with ROS Topics, Services, and Parameters.

#### topics

Topics are named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes are not aware of who they are communicating with. Instead, nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic.
We can check our available topic with command *$ rostopic list* and display messages published to /topic_name by *$ rostopic echo /topic_name*

#### messages

Nodes communicate with each other by publishing the **messages** to topics. A message is a simple data structure comprise the specific fields.

- **msg files**: are simple text files for specifying the data structure of a message. These files are stored in the msg subdirectory of a package. For more information about these files, including a type specification, see the *msg format*.

- **msg types**: are understand of ROS *naming* convention: the name of package + / + name of .msg file. For instance, the *std_msgs/msg/String.msg* has the message type *std_msgs/String*
- **building**: ROS *Client Libraries* implement a message generator that translate .msg file into source code. The generator must be invoked from your build script by adding the line ***rosbuild_genmsg()*** to your CMakeList.txt
- **header**: A msg may include a special message type called 'Header', which include some common metadata field such as timestamp and frame ID.

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.
#
# sequence ID: consecutively increasing ID
uint32 seq
#Two-integer timestamp that is expressed as:
# * stamp.sec: seconds (stamp_secs) since epoch (in Python the variable is called 'secs')
# * stamp.nsec: nanoseconds since stamp_secs (in Python the variable is called 'nsecs')
# time-handling sugar is provided by the client library
time stamp
#Frame this data is associated with
# 0: no frame
# 1: global frame
string frame_id
```

Figure 2: Message Header

### bags

Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

### package

Packages might contain ROS nodes, ROS-independent library, a dataset, configuration files, a third-party software. Packages are created by hand or with tools like **catkin_create_pkg**. ROS package is a directory desended from *ROS_PACKAGE_PATH* in *ROS Environment variable* that has a *package.xml* file in it.
ROS packages commonly contain the following strucutes.

- *include/package_name*: C++ incude header (make sure it export in CMakeLists.txt)
- *msg/*: folder contain Message(msg) types
- *src/package_name/*: source file, especially Python source that are exported to other packages

- *scripts/*: executabe scripts
- *CMakeList.txt*: CMake build file
- *package.xml*: Package catkin/CMakeLists.txt

There are few tools that help to manage the packages

- *rospack*: find and retrieve information about packet
- *catkin_create_package*: create a new package
- *catkin_make*: build a workspace of package
- *rosdep*: install system dependencies of a package

**catkin_make**

A convenience tool for buidling code in workspace. Here we use
***$ cd ~/catkin_ws***
***$ catkin_make***
to compile and build our workspace. After running ***catkin_make*** we have 2 folder *build* and *devel* in our root of catkin workspace. The *build* folder is where *cmake* and *make* are invoke, and the devel folder contains any generated files and targets, plus setup *.sh file so that you can use it like it is installed.

**package.xml**

A **package manifest** XML file that must be included with any catkin-compliant package's root folder that defines that package's properties such as package name, version number, author, maintainers and dependencies on other catkin package.
Some required tags include

- *name*: ¡name¿ of package
- *¡version¿*: version of package
- *¡description¿*: description of package content
- *¡maintainer¿*: the name of person(s) that is/are maintaining the package
- *¡license¿*: software license(s) (e.g GPL,BSD,ASL) under which code is release

### 0.3.3 Installing software

First, operating system should be installed, for example (L)Ubuntu 16.04 minimal. Instructions can be found on Ubuntu website, for example: `https://help.ubuntu.com/community/Lubuntu/Documentation/MinimalInstall`. Instructions for installing ROS can be found in `http://wiki.ros.org/kinetic/Installation/`.
Brief list of commands: *adjust usernames etc. according to the local user, here openkin or pi*

# For raspbian jessie source installation: # enough swap for build to succeed

sudo fallocate -l 2G /var/swap2

sudo chmod 600 /var/swap2

sudo swapon /var/swap2

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" ¿ /etc/apt/sources.list.d/ros-latest.list'

sudo apt-key adv –keyserver 'hkp://keyserver.ubuntu.com:80' –recv-keyC1CF6E31E6BADE8868B172B4F42ED

sudo apt-key del 421C365BD9FF1F717815A3895523BAEEB01FA116

sudo apt-get install python-rosdep python-rosinstall-generator python-wstool python-rosinstall build-essential git libgps-dev i2c-tools libi2c-dev

sudo rosdep init

rosdep uodate

cd

mkdir ros_catkin_ws

cd ros_catkin_ws

rosinstall_generator ros_comm –rosdistro kinetic –deps –wet-only –tar ¿ kinetic-ros_comm-wet.rosinstall

roslocate info common_msgs –distro=kinetic ¿¿ kinetic-ros_comm-wet.rosinstall

roslocate info tf –distro=kinetic ¿¿ kinetic-ros_comm-wet.rosinstall

roslocate info tf2 –distro=kinetic ¿¿ kinetic-ros_comm-wet.rosinstall

roslocate info diagnostic_updater –distro=kinetic ¿¿ kinetic-ros_comm-wet.rosinstall

roslocate info actionlib –distro=kinetic ¿¿ kinetic-ros_comm-wet.rosinstall

roslocate info python_orocos_kdl –distro=kinetic ¿¿ kinetic-ros_comm-wet.rosinstall

roslocate info bondcpp –distro=kinetic ¿¿ kinetic-ros_comm-wet.rosinstall

nano kinetic-ros_comm-wet.rosinstall

***add to the end of file: - git: local-name:***

rosbag_migration_rule uri:
https://github.com/ros/rosbag_migration_rule.git
version: master

roslocate info pluginlib –distro=kinetic ¿¿ kinetic-ros_comm-wet.rosinstall

roslocate info angles –distro=kinetic ¿¿ kinetic-ros_comm-wet.rosinstall

roslocate info class_loader –distro=kinetic ¿¿ kinetic-ros_comm-wet.rosinstall

wstool init -j4 src kinetic-ros_comm-wet.rosinstall

(sudo cp /usr/share/cmake-3.0/Modules/FindEigen3.cmake /usr/share/cmake-3.6/Modules/)

rosdep install –from-paths src –ignore-src –rosdistro kinetic -y

sudo mkdir -p /opt/ros/kinetic

sudo chmod -R 777 /opt/ros

sudo chown -R pi:pi /opt/ros

./src/catkin/bin/catkin_make_isolated –install –install-space /opt/ros/kinetic -DCMAKE_BUILD_TYPE=Release

cd

# Normal ubuntu, skip if from source

The complete image of Ubuntu 16-04 with ROS kinetic already installed for raspberry pi 3 can be found at `https://ubiquity-pi-image.sfo2.cdn.digitaloceanspaces.com/2020-11-07-ubiquity-xenial-lxde-raspberry-pi.img.xz`

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"
¿ /etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv –keyserver 'hkp://keyserver.ubuntu.com:80' –recv-keyC1CF6E31E6BADE8868B172B4F42ED

sudo apt-key del 421C365BD9FF1F717815A3895523BAEEB01FA116

sudo apt-get update

rosdep update

source /.bashrc

sudo apt-get install libgps-dev libi2c-dev ntpdate

sudo apt-get upgrade

# Same for here

catkin_make

source devel/setup.bash

**#If you start from here with fetching from the complete datalogger image**

cd catkin_ws/src

ln -s /home/openkin/openkin/nodes/ascii_logger/ascii_logger

ln -s /home/openkin/NewGaitMaven/nodes/vectornav vectornav

# for rpi

cd

wget https://github.com/joan2937/pigpio/archive/master.zip

unzip master.zip

cd pigpio-master

make -j4

sudo make install

cd /catkin_ws

catkin_make

sudo visudo

***add line at the end: "openkin ALL=(ALL:ALL) NOPASSWD: ALL"***

sudo apt-get install screen ppp usb-modeswitch

**#if using E160E or similar**

wget http://www.sakis3g.com/downloads/sakis3g.tar.gz

tar -xzvf sakis3g.tar.gz

sudo mv sakis3g /usr/bin/sakis3g

sudo chown root:root /usr/bin/sakis3g

sudo chmod +x /usr/bin/sakis3g

***if e3372 doesn't work out of the box***

***remove old usb_modeswitch***

***install tcl, libusb-1.0-0-dev etc.***

***download latest usb_modeswitch http://www.draisberghof.de/usb_modeswitch/#install***

***sudo make install***

**Here we start to install git lfs for data storage**

cd ~/openkin/NewGaitMaven/linux-shutdown

make

sudo nano /etc/rc.local

***add line before exit 0:**
*"su -c "/home/openkin/NewGaitMaven/start_script.sh" openkin*

***for rpi add shutdown_flip***

***check permissions for serial device, pi: /boot/cmdline.txt: remove serial terminal***

***add source's to .bashrc***

***check usernames and folders in measurement.sh***

# **For mtmanager**

sudo apt-get install libqtopengl5

***download mtmanager, extract, symlink to /usr/local/bin ***

### 0.3.4 3D Printed Cases

Models can be found in 3D folder. They can be redesigned with SolidWork/FreeCad. FreeCad can be download for free. However, SolidWork require license from university and can be installed from IT-helpdesk. **Some information and settings for printers**
Get start by running the script C:\Minifactory\Config\first_time_config.bat. Then press any key to continue

Open Minifactory
**Step 1**: Start connect Z-calibration
**Step 2**: Run the nozzle up/down by pressing the arrow button until the paper slightly rubs between the nozzle and the print bed. Confirm height with "Ready" button. The height should be around 150cm
In manual control tab, uncheck the flowrate, bed temperature, extruder 1, extruder 2,. We mostly use extruder 1 for printing, the temperature is around 200 degree celsius.

In object placement, add object and rotate it that its suitable (not need to much support ) In slicer tab, choose Slicer as Slic 3r or CuraEngine (prefered). If CuraEngine, the quality should be 0.2 mm and support everywhere.. The Filament setting is PLA for both extruder.

The preview tab can show each layer in real-time printing. Also, it shows the estimated print time, layer counts. Finally press Start Print.

## 0.4 Usage

### 0.4.1 VectorNav

The datalogger is off, if light is not on. The 4G modem should also be connected to the USB port before turning the logger on. The logger can be turn on by pressing the power button on the battery, inserting the battery to the case and connecting USB-cable to the battery. The logger will turn on and off the red and green leds when powering up respectively. When the logger acquire the internet connection, the green led is turned on. When the logger is synchronizing github/gitlab server, the green led is flashing

When the logger has been on for a while and the green led is on, the logger is ready for starting measurement. The measurement can be started by disconnecting the 4G modem. The green led goes off and after a while, the red should start blinking if the VectorNav does not have fix, or if everything seem to be fine, the red led is turned on. It is still recommend to wait for GPS to acquire better stability, but the recording is running.

When measurement are done, the recording can be stopped by reconnecting the 4G modem. The red led shuts off and the green led starts to blink as files are upload to github/gitlab. When the upload is done, the green led stabilize.

The logger can be turn off by turning the power/shutdown switch off, and waiting for 10-30 seconds for all lights to go off. When logger is off, the USB power cable can be disconnected with care, and battery removed.

Note: JYU version has reverse switch direction compared to other loggers, on is out, off is towards the antenna.

## 0.5 SSH access

The datalogger have SSH server running, and they can be accessed by connecting the datalogger via ethernet. The network should have DHCP to give IP address for the datalogger. For example internet sharing on Mac and Linux can be used for connecting the datalogger straight to the computer.

The IP of the datalogger can possibly found using *nmap*: *nmap -sn 192.168.2.1-255* where IP range is subnet defined for network. The SSH connection can be mad with command *ssh pi@192.168.2.3* where IP is the IP of the datalogger, and pi is the username.

Username for Raspberry Pi based datalogger is pi. Password is raspberry for pi and openkin for openkin.