PROGRAM_NAME 0.1

User manual

Contents

1 Copyright and License

2 Introduction

PROGRAM_NAME is an application for recording and processing data from inertial measurement units (IMUs). It consists of two parts. Microcontroller code running on an Arduino board collects data from the IMUs and transmits it to a PC. A Python program running on the PC receives data from the Arduino and provides users with a graphical interface. This interface allows users to interact with the Arduino and to view and manipulate IMU data.

3 Hardware

PROGRAM_NAME interconnects three pieces of computational hardware: a PC, an Arduino, between 1 and 3 IMUs.

3.1 PC

PROGRAM_NAME has been tested on PCs running Linux Mint 18 Sarah, OS X El Capitan 10.11, and Windows 10. It is expected to work with other versions of these operating systems and with other Linux distributions.

3.2 Arduino

PROGRAM_NAME has been tested with an Arduino UNO. It may work on other models with a 16MHz clock speed and sufficient storage.

3.3 IMUs

The mpu9250 by InvenSense is a nine-axis (gyroscope, accelerometer, compass) motion tracking device.

For applications requiring minimum package size and weight, the mpu9250 can be wired directly to the Arduino. A separate manual documents the procedure we have used to prepare the mpu9250 for use with PROGRAM_NAME: http://www.url_for_cassandras_manual.com

For testing purposes, or if the added size and weight are acceptable, an mpu9250 mounted on a circuit-board can be used with PROGRAM_NAME.

3.4 SPI

The Arduino communicates with the IMUs using the Serial Peripheral Interface bus (SPI) protocol. With SPI, one or more slave devices (IMUs in this application) exchange data with a master device (the Arduino) over a single shared bus, consisting of 2 data lines and a clock line. Each slave has a separate *chip select* line, used to control access to the shared bus.

3.5 Wiring the Arduino

Table 1 summarizes the process of connecting IMUs to the Arduino. It may be necessary to use a breadboard, especially if multiple IMUs are used. Figure 3.5 shows an Arduino wired to a single IMU. Pins 8, 9, and 10 are used for *chip select* lines for up to three IMUs. Pin 11 carries data traveling from the Arduino to the IMUs, i.e. Master-Out, Slave-In (MOSI). Pin 12 carries data traveling from the IMUs to the Arduino, i.e. Master-In, Slave Out (MISO). Pin 13 carries a clock signal which regulates timing of the communication protocol. Each IMU should be connected to 3.3V power (available on the Arduino) and to ground.

If using a trigger, connect it to Pin 4 and to ground.

Connect the Arduino to the PC with a USB cable. To ensure adequate power, especially if using multiple IMUs, it is recommended to power the Arduino with an external power source instead of relying on the USB port.

4 Installation

4.1 Arduino IDE

While there are many tools for installing program code onto the Arduino, we have tested PROGRAM_NAME using the Arduino IDE version 1.8.1. Earlier versions might lack some of library definitions used by PROGRAM_NAME.

The software can be downloaded from: https://www.arduino.cc/en/Main/Software

description	label	color	pin
IMU 1 chip select	NCS / CS / SS	white	8
IMU 2 chip select	NCS / CS / SS	white	9
IMU 3 chip select	NCS / CS / SS	white	10
data from Arduino to IMU	MOSI / SDI / SDA	green	11
data from IMU to Arduino	MISO / SDO / ADO	blue	12
clock	SCL / CLK / SCK	yellow	13
power	VCC	red	3.3V
ground	GND	black	GND
trigger			4

Table 1: Instructions for wiring the IMUs to the Arduino. The first column describes the purpose of each line. The second column provides names commonly used to describe each line. For example, if the IMU is mounted on a board, the board's pins might have labels matching these values. The third column provides the colors specified in the **Document Name** manual. The fourth column specifies pins on the Arduino Uno.

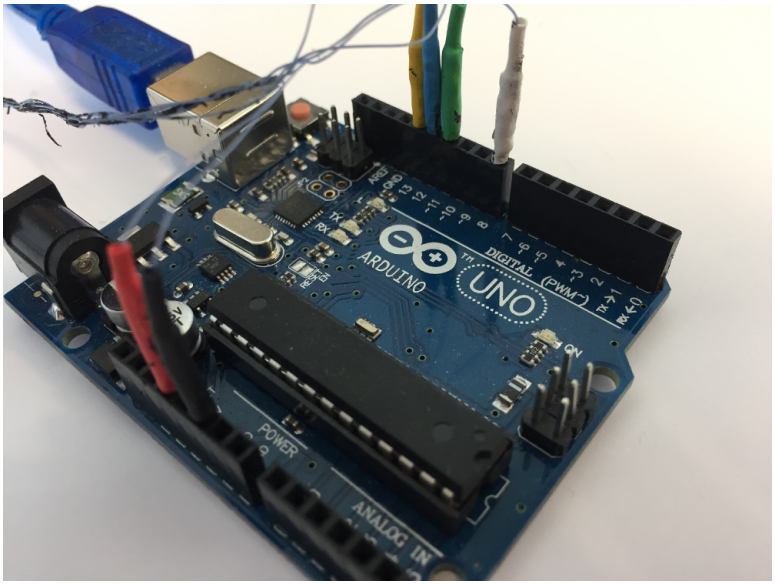

Figure 1: Wiring a single IMU to the Arduino

If your operating system has a package management system, it might be able to install the Arduino IDE. For example, on a Debian-based system you can use `apt`:

```
# apt install Arduino-core
```

4.2 Install PROGRAM_NAME on the Arduino

Transfer the file `am_tx/am_tx.ino` onto Arduino. This can be accomplished using the Arduino IDE graphical user interface. Alternatively, the Arduino IDE can be used to program the Arduino directly from the command line:

```
# arduino --upload am_tx/am_tx.ino
```

For more information about the Arduino command line interface, see: https://github.com/arduino/Arduino/blob/master/build/shared/manpage.adoc

4.3 PC

PROGRAM_NAME requires Python 3. It has been tested with Python version 3.5. Thus, it is recommended to use Python version 3.5 or later.

Consider using `virtualenv` to isolate your installation of PROGRAM_NAME from other packages and from other Python versions: https://pypi.python.org/pypi/virtualenv

We recommend using pip to install PROGRAM_NAME: https://pypi.python.org/pypi/pip

```
# cd PROGRAM_NAME
# pip install --upgrade pip
# pip install ./
```

Pip will automatically install any of the following dependencies if needed:

- h5py
- numpy
- pyqtgraph
- pyserial
- pyqt5

Once PROGRAM_NAME has been installed, it can be started from the command line:

```
# PROGRAM_NAME
```

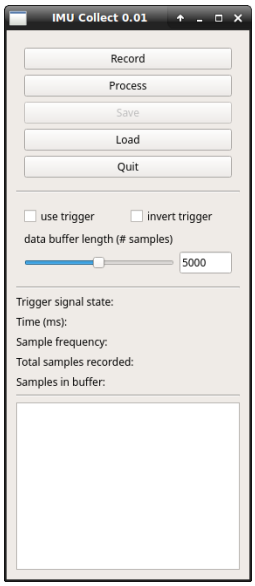

Figure 2: Control panel

5 The data buffer

PROGRAM_NAME uses a single data buffer to hold IMU data. When data is recorded from the IMUs, the data buffer is first erased. Then each sample is added to the buffer as it is recorded. The buffer can be saved to a file, or a file can be loaded into the buffer, erasing any previous contents. Data processing operations can be performed on the data buffer, altering it irreversibly. Therefore, any time the data buffer contains valuable data it is recommended to save it to file before collecting new data, loading another file, or executing data processing operations.

The *data buffer length (# samples)* slider adjusts the size of the data buffer. When each new sample is received, it is added to the data buffer. If the buffer was already full (the number of samples in the buffer was equal to the size of the buffer), the oldest sample in the buffer is deleted. If the size of the buffer is adjust to be smaller than the number of samples in the buffer, the oldest samples in the buffer are deleted until the number of samples in the buffer is equal to the buffer length.

6 Collecting data

To begin collecting data, press the *record* button. The *record* button changes into the *stop* button. The PC establishes communication with to the Arduino and instructs it to begin collecting data from the IMUs at 200Hz. This sample rate is hard-coded, and was determined to be near the maximum possible with the Arduino Uno.

As samples received from the Arduino are recorded in the data buffer, they become visible in the data

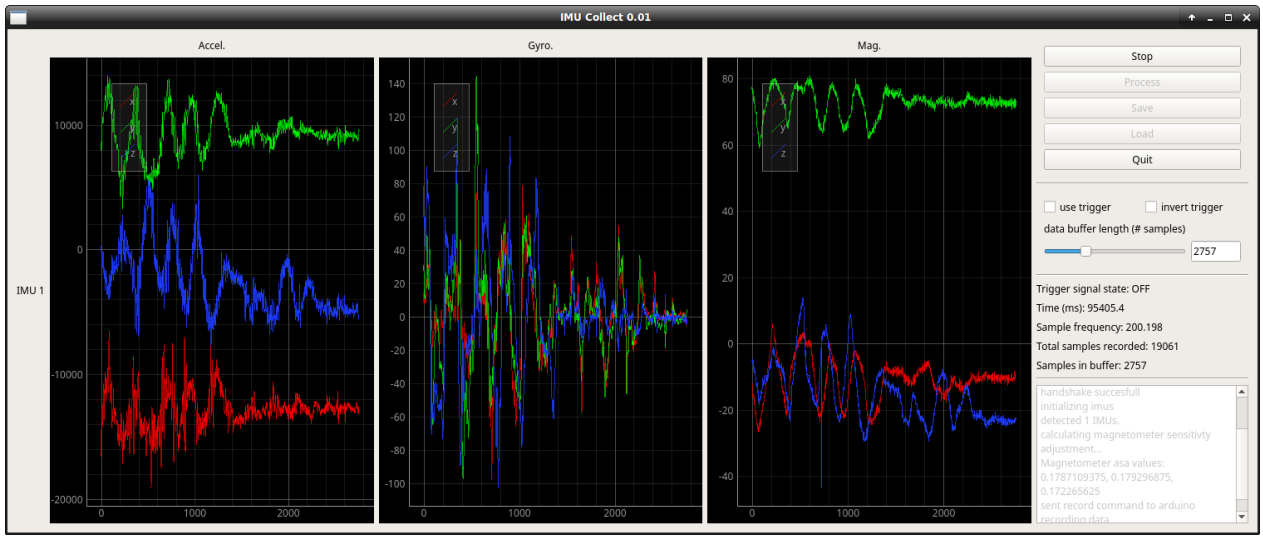

Figure 3: Collecting data from a single IMU

visualization plots. For each IMU, one row of three plots is displayed. The left-most plot shows accelerometer data, the center plot shows gyroscope data, and the right-most plot shows magnetometer data. For each plot, red, green, and blue lines show x, y, and z axis measurements, respectively.

The *stop* button (or the trigger, as described in Section 6.1), causes the PC first to instruct the Arduino to stop collecting data and then to halt communication with the Arduino. The *stop* button changes back into the *record* button.

6.1 Trigger

Optionally, a trigger attached to the Arduino (Section 3.5) can be used to stop recording. If the *use trigger* checkbox is not checked, the trigger is ignored. Otherwise, if an active trigger is detected, recording will stop, i.e., the same effect as pressing the *stop* button during recording. If the *Record* is pressed while the *use trigger* checkbox is checked and the trigger is active, the recording immediately ends with zero data stored.

If the *invert trigger* checkbox is not checked, the trigger is considered active when the associated Arduino pin is set high. If the *invert trigger* checkbox is checked, the trigger is considered active when the associated Arduino pin is set low.

If no trigger is connected then the value of the pin is undefined. Thus, to ensure reliable behavior, the *use*

trigger checkbox should not be checked unless there is a trigger connected to the Arduino.

7 Saving and loading

7.1 File formats

The `.csv` file format stores data in plain text, using newlines to delimit samples and commas to delimit measurements within each sample. PROGRAM_NAME uses the CSV module in the Python Standard Library with default formatting parameters: https://docs.python.org/3/library/csv.html

The `.hdf5` format is designed to store large amounts of data. It is generally more compact than `.csv`. PROGRAM_NAME uses the h5py package to read and write `.hdf5` files: http://www.h5py.org/

7.2 Saving

The *save* button opens a dialog allowing the user to select a filesystem location, a filename, and a filetype (either `.csv` or `.hdf5`). The the data buffer is saved to file according to the selections made in the dialog. The *save* button is disabled when the data buffer is empty or while data is being recorded.

7.3 Loading

The *load* button opens a dialog allowing the user to select a filetype (either `.csv` or `.hdf5`) and a file to load. The file will be loaded into the data buffer, overwriting any data stored there. The *load* button is disabled while data is being recorded.

8 Processing data

The *process* button opens the data processing control panel (Figure 4).

8.1 Process mode

All data transformation algorithms are applied to the data buffer. If the *Use current data* radio button is selected, the current contents of the data buffer are used. This option is not available when the data buffer is empty.

Alternatively, if the *Batch process* radio button is selected, the current contents of the data buffer are ignored. Instead, batch processing iterates over a list of one or more input files, loading each file into the data buffer, applying a transformation algorithm, and saving the result to a new output file before proceeding to the next input file in the list. The *Select files* button launches a dialog for selecting input files for batch processing. The batch processing output filetype can be set, or can vary to match the filetype of each input file. The output suffix string is appended to the filename of each input file (before the file extension) to compose the output

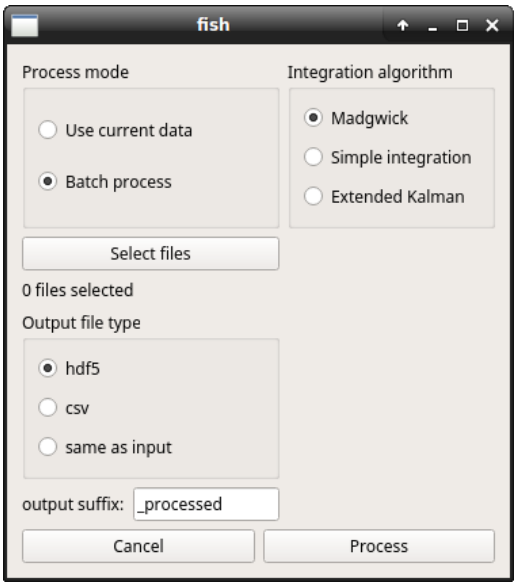

Figure 4: Data processing control panel

filename. If the output suffix string is empty and the output filetype matches the input filetype, then the input file is overwritten.

With either processing mode, the contents of the data buffer are overwritten by any data processing.

8.2 Integration algorithm

The *Integration algorithm* radio buttons select the data transformation algorithm.

Madgwick
: We need to write about what this does.

Simple integration
: We need to write about what this does.

Extended Kalman
: We need to write about what this does.

9 Troubleshooting

9.1 Installation

If you encounter problems during installation, make sure you are using Python 3.5 or later and that pip is working with the correct Python version. Consider using `virtualenv` as discussed in Section 4.3.

9.2 Error messages

This section provides explanation and troubleshooting tips for each error message produced by PROGRAM_NAME. Error messages are printed in red type in the main text window.

ASA read failed, using 1 adjustment

For each magnetometer axis, a sensitivity adjustment value (ASA) is stored in ROM by the manufacturer. This error message is reported when the Arduino is unable to read the ASA values from an IMU. It likely indicates that the Arduino is not communicating correctly with the magnetometer. Any magnetometer data recorded despite this message should be discarded. Try resetting the Arduino. Make sure that the Arduino is correctly powered and connected to the PC (Section 3.5), and that the correct code is installed on the Arduino (Section 4.2).

failed to create connection, aborting

The program failed to establish a serial connection with the Arduino. Try resetting the Arduino. Make sure that the Arduino is correctly powered and connected to the PC (Section 3.5), and that the correct code is installed on the Arduino (Section 4.2).

handshake failed

Even though the PC may have established a valid serial connection to the Arduino, the data exchange protocol used by PROGRAM_NAME failed to establish a communication handshake with the Arduino. Try resetting the Arduino. Make sure that the Arduino is correctly powered and connected to the PC (Section 3.5), and that the correct code is installed on the Arduino (Section 4.2).

invalid csv file

The program attempted to read a .csv file (Section 7.1), but the format of the data in the file was not valid.

invalid file type:...

There was an attempt either to save or to load a filetype other than `.csv` and `.hdf5`. As discussed in Section 7.1, `.csv` and `.hdf5` are the only file formats supported by PROGRAM_NAME.

no Arduino found

The program searched for Arduinos on all serial ports but didn't find any. PROGRAM_NAME uses `serial.tools.list_ports` to scan for Arduinos: http://pyserial.readthedocs.io/en/latest/tools.html

no IMUs detected, aborting

The Arduino did not detect any attached IMUs. After the Arduino is initialized, it attempts to determine the number of IMUs by sending a WHOAMI request while signaling each of the three legal chip select pins (Section 3.5). It then sends a message to the PC reporting the number of responses received. This error is reported if the Arduino does not receive any WHOAMI responses. Try resetting the Arduino.

Make sure that the Arduino is correctly powered and connected to the PC (Section 3.5), and that the correct code is installed on the Arduino (Section 4.2).

rx failed, no data read from serial

The PC expected to receive data from the Arduino but failed. Perhaps no data was transmitted, or perhaps only data that violates the communication protocol used by PROGRAM_NAME was received. Try resetting the Arduino. Make sure that the Arduino is correctly powered and connected to the PC (Section 3.5), and that the correct code is installed on the Arduino (Section 4.2).

unknown sample received:...

The PC expected to receive a packet containing a data sample, but the packet either had the wrong type or the wrong length. This message can be ignored if it occurs only briefly at the beginning of recording. Otherwise, try resetting the Arduino.

unable to determine number of IMUs, aborting

The program failed to determine how many IMUs are attached to the Arduino. After the Arduino is initialized, it attempts to determine the number of IMUs by sending a WHOAMI request while signaling each of the three legal chip select lines (Section 3.5). It then sends a message to the PC reporting the number of IMUs detected. This error is reported if the PC sends a command to the Arduino to initialize, but does not receive a message reporting the number of IMUs detected. Try resetting the Arduino. Make sure that the Arduino is correctly powered and connected to the PC (Section 3.5), and that the correct code is installed on the Arduino (Section 4.2).