IMU-Capture 0.1

User manual

September 20, 2017

Contents

1 Copyright and License

2 Introduction

IMU-Capture is an application for recording and processing data from inertial measurement units (IMUs). The software has 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 IMU via the Arduino and to view and manipulate IMU data.

3 Hardware

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

3.1 PC

IMU-Capture has been tested on PCs running Linux Mint 18 Sarah, OS X El Capitan 10.11, and Windows 10. It may work with other operating systems.

3.2 Arduino

IMU-Capture has been tested with an Arduino UNO. It may work on other models that use a 16MHz clock speed and have 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 IMU-Capture: 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 IMU-Capture.

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 bus consisting of 2 data lines and a clock line. In addition, each slave device 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 Uno) 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 connect the Arduino to an external power source instead of relying on the USB port for power.

4 Installation

4.1 Arduino IDE

While there are many ways to install program code onto the Arduino, we have tested IMU-Capture using the Arduino IDE version 1.8.1. Earlier versions may lack some of library definitions used by IMU-Capture.

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

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

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. The third column lists the colors specified in the **Document Name** manual. The fourth column specifies pins on the Arduino Uno.

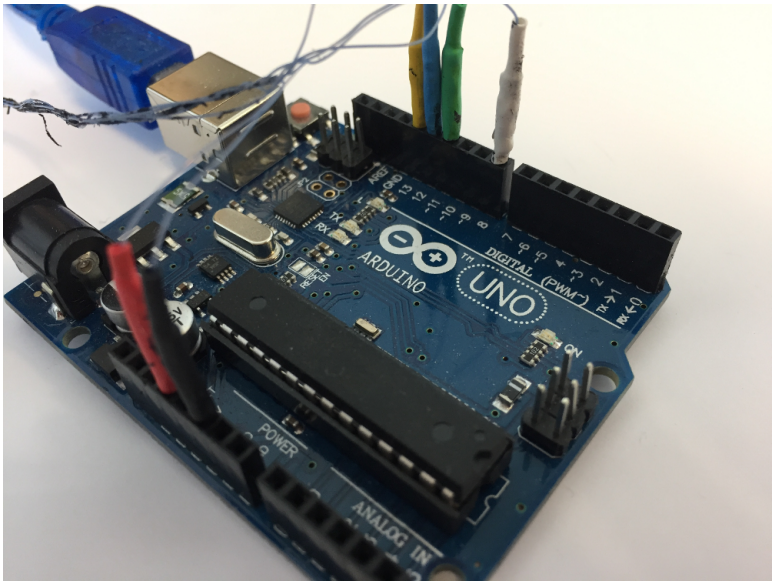

Figure 1: Wiring a single IMU to the Arduino

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

4.2 Install IMU-Capture on the Arduino

Transfer the file `ic_tx/ic_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 ic_tx/ic_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

IMU-Capture requires Python 3. It has been tested with Python version 3.5. Thus, it is recommended to use Python version 3.5 or later. On windows, IMU-Capture has been successfully installed with the Anaconda Python distribution (https://www.anaconda.com), but tests failed to install IMU-Capture with a standalone Python installation.

It is recommended to use using pip (the Python Package Index) to install IMU-Capture: https://pypi.python.org/pypi/pip

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

Pip will automatically install any of the following dependencies, and any of their sub-dependencies, if needed:

- h5py
- pyqtgraph
- pyserial
- pyqt5
- scipy
- numpy-quaternion

Once IMU-Capture has been installed, it can be started from the command line:

```
# IMU-Capture
```

On Windows, informational and error messages will not be displayed unless IMU-Capture is invoked by the Python interpreter explicitly:

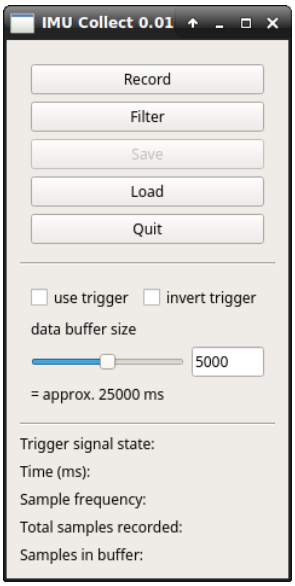

Figure 2: Control panel

```
# python IMU-Capture
```

When IMU-Capture starts, the main control panel will be visible (Figure 2).

5 The data buffer

IMU-Capture 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. When data is not being 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 is already full (the number of samples in the buffer is 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

instrument	modality	units
accelerometer	acceleration	meters per second squared
gyroscope	rotational speed	radians per second
magnetometer	magnetic flux density	microteslas

Table 2: Units used by IMU-Capture

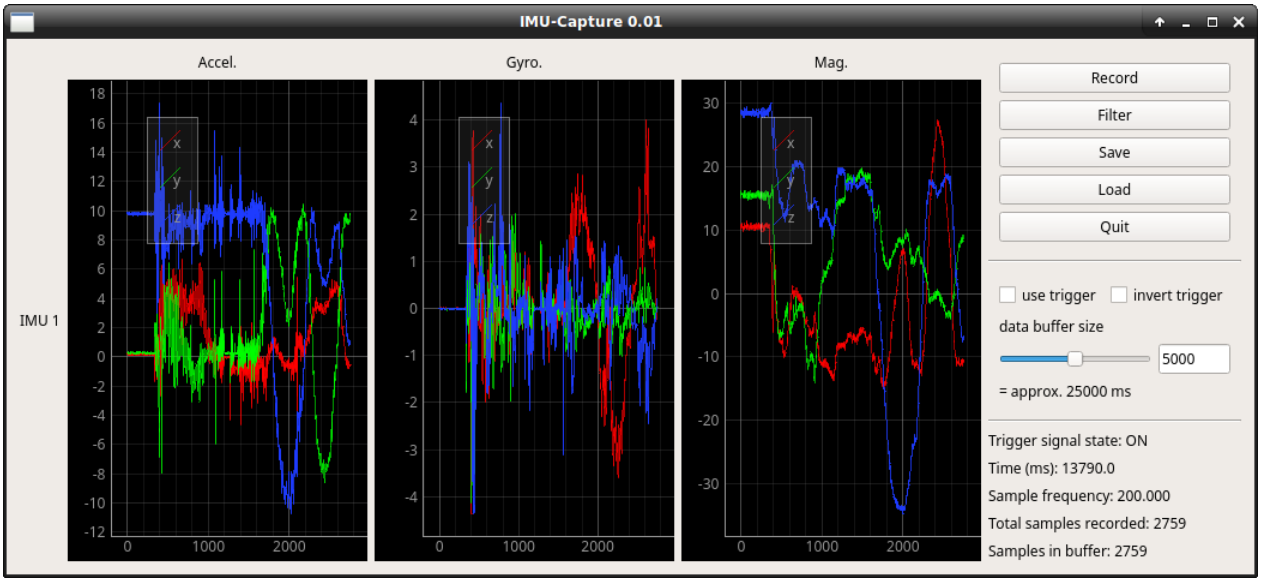

Figure 3: Collecting data from a single IMU

200Hz. This sample rate is hard-coded, and was determined to be near the maximum possible with the Arduino Uno.

The *stop* button (or the trigger, as described in Section 6.2), causes the PC first to instruct the Arduino to stop collecting data and then to halt communication with the Arduino.

Table 2 lists the units used for each sensor modality.

6.1 Plots

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

Each plot will scale automatically as the data buffer is updated. Plots can also be adjusted manually using the mouse.

6.2 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, the value of the pin is undefined. Thus, to ensure reliable behavior, the *use trigger* checkbox should be unchecked 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. IMU-Capture 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`. IMU-Capture uses the h5py Python 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 file system location, a file name, and a file type (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 file type (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 Filter data

Due to factors such as the effect of gravity upon the accelerometer, sensor drift, and noise, the raw data collected by IMU-Capture will not reflect the actual dynamics of the IMUs. IMU-Capture includes several

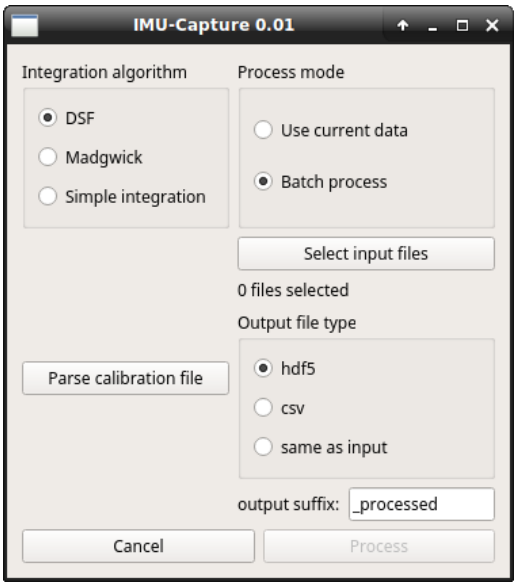

Figure 4: Data filtering control panel

algorithms that can be applied to IMU data after it has been recorded. Currently, it is not possible to apply these algorithms in real time as the data is recorded. Running one of these algorithms will have to main effects upon recorded data: 1) The effect of gravity is removed from the acceleration data, leaving only *dynamic* acceleration. 2) The rotational speed data is converted into rotational angle data, that is, the rotation of the IMUs is integrated over time.

Currently, filtering algorithms only apply to a single IMU.

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

8.1 Calibration

Before any of the filtering algorithms can be applied, it is necessary to load a calibration file by clicking the *Parse calibration file* button. Three main pieces of information are extracted from the calibration file:

1. Three orthogonal orientations are used to construct a set of basis vectors
2. The first of the three orientations is used to determine the initial gravity vector
3. The first of the three orientations is used to determine noise covariance matrices

Follow these steps to create a calibration file.

1. Wire up the IMU as you will use it to collect data

2. Hold the IMU in an orientation to be considered “upright”
3. Begin recording
4. Hold the IMU upright, in the first orthogonal orientation, for about 5 seconds
5. Rotate the IMU 90 degrees so that the new orientation is orthogonal to the first
6. Hold the IMU in the second orthogonal orientation for about 5 seconds
7. Rotate the IMU 90 degrees again so that the new orientation is orthogonal to both previous orientations
8. Hold the IMU in the third orthogonal orientation for about 5 seconds
9. Stop recording

If IMU-Capture is unable to load a calibration file, error messages explaining why can help determine if there are problems with the file and help to record a correct calibration file. See Section 9.3 for more discussion about these messages.

8.2 Filter mode

All data filtering 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 filtering 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 file type can be set, or can vary to match the file type of each input file. The output suffix string is appended to the file name of each input file (before the file extension) to compose the output file name. If the output suffix string is empty and the output file type matches the input file type, then the input file is overwritten.

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

8.3 Integration algorithm

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

DSF The Dynamic Snap Free algorithm developed by Vikas: (cite)

Madgwick
An implementation of the filter developed by Madgwick: http://x-io.co.uk/res/doc/madgwick_internal_report.pdf

Simple integration

Use the Madgwick algorithm with the beta parameter set to 0.

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.

9.2 General error messages

This sections discusses some of the error messages that may be reported by IMU-Capture.

This section provides explanation and troubleshooting tips for each error message produced by IMU-Capture.

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 after 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 IMU-Capture 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). It is possible the IMU-Capture incorrectly identified a serial device as the Arduino. Any device that the pySerial library (`https://pythonhosted.org/pyserial/#`) identifies as manufactured either by "Arduino" or by "Microsoft" will be identified as an Arduino by IMU-Capture. Try disconnecting all serial devices and then reconnecting them, starting with the Arduino.

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 file type other than `.csv` and `.hdf5`. As discussed in Section 7.1, `.csv` and `.hdf5` are the only file formats supported by IMU-Capture.

no Arduino found

The program searched for Arduinos on all serial ports but didn't find any. IMU-Capture uses the pySerial library to scan for Arduinos: https://pythonhosted.org/pyserial/#

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 data that violates the communication protocols used by IMU-Capture 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).

9.3 Calibration error messages

This section discusses error messages that may be reported when IMU-Capture fails to load a calibration file.

fewer than 3 steady intervals

A "steady interval" is a contiguous duration of at least 3 seconds when the change in accelerometer

measurement remains below a threshold and and the gyroscope measurement remains below a threshold. Possibly the IMU was not held still enough, or the duration of one or more hold was too short.

too many steady intervals, the limit is…

See the discussion for "fewer than 3 steady intervals" for a definition of "steady interval". IMU-Capture can remove some some extra steady intervals if they are not part of a triple of orthogonal vectors, but too many will cause this error. When recording the calibration file, start recording when with the IMU in the first orientation, end recording immediately after 5 seconds of the third orientation, and make smooth, quick, transitions between the orientations.

found more than 1 triple of orthogonal vectors

More than three orientations were recorded, and more than one triple of orthogonal vectors can be identified. This is similar to "too many steady intervals, the limit is…". See the discussion of that message for possible solutions.

could not find 3 orthogonal vectors

Even though at least 3 "steady intervals" (see the discussion of the "fewer than 3 steady intervals" message) were found, no 3 intervals are orthogonal. That is, the difference between 90 degrees and the angles separating the intervals was greater than a threshold. When recording the calibration file, ensure that that three orientations are mutually orthogonal.