Eta Vision System Manual

Contents

Introduction to Eta's Code Structure

The code for Eta's vision system is divided into two main directories: HPV_ArduinoMega and HPV_OSD. A wealth of standard and custom libraries also support the system's functioning. With the exception of ArduCam_Max7456 and GPS_UBLOX, there should be no need to modify any of these libraries.

HPV_ArduinoMega

HPV_ArduinoMega contains files defining the system's logic. The Arduino streams data from peripheral devices, calculates values of interest, and sends the data to the OSD. Currently, the peripherals supported are:

- ANT+ receiver configured for power pedals, crank torque frequency pedals (e.g. Todd's SRM), and heart meter
- uBLOX GPS
- Battery meter
- Temperature sensor
- SD card reader

HPV_ArduinoMega.ino

This is the central processing centre of the system. The setup initializes all the peripherals and pins and calls the calibration function to calibrate the pedals. The main loop receives data from the peripherals, calculates values of interest, and sends these values to the OSD chip via its serial link. In general, data is sent whenever a new message is received from the GPS or the ANT+ receiver. This file also takes care of most of the logging activity, writing values every two seconds to the SD card.

Every time the GPS sends data, or every one second if the GPS is not available, the program will check if the profile toggle button has been pushed. If so, the program will load the next profile.

Notes:

- Most of the variables to display are sent to the OSD when there is a GPS update. Since the GPS may not always be available, there are two lengthy cases containing nearly identical code. One block will execute whenever the GPS updates with a new message while the other block will take over if the Arduino loses its GPS connection. Remember to copy any code you add into both blocks (if applicable).
- The update frequency of logs can be set by changing `PERIOD` in ANT_interface.h
- Anything related to the `START` variable seems to have been deprecated

Ant_interface.cpp

This file contains functions for communication with the antenna. No changes should be necessary to the current code.

BatteryCheck.ino

This file contains functions to measure the temperature and battery voltage.

GPS_Func.ino

This file contains logic related to the calculations done with GPS coordinates.

SD_Func
This file contains functions to perform basic write and read operations.

TargetSpeed_Func
The functions in this file carry out a variety of tasks not necessarily related to calculating the target speed. The main function is `toggle()`, which takes care of loading new profiles and switching between real-time and simulation modes. There are also functions to calculate target speed and target power.

Logging

Currently, there are two logs created for each run of the program. The "LOG" file is a general log that is written to at regular intervals and contains information from all of the peripherals connected. This interval is defined by the `PERIOD` variable. The contents that are logged can be modified by changing the print commands at the end of `loop()` in HPV_ArduinoMega.ino. The column headers are specified in `setup()` of the same file.

The "SLG" file is written to whenever non-duplicate power meter messages are received from the SRM. The contents that are logged can be modified by changing the print commands at the end of `loop()` in HPV_ArduinoMega.ino. The column headers are specified in `setup()` of the same file.

Different types of logs from the same run will always have the same number in their file name.

Programming the OSD

Note: Do not use the OSD configuration GUI. It seems incompatible with the way Eta's OSD is currently set up and it does not support custom panels.

To program the OSD, connect it to the computer via an FTDI-USB converter. Upload the code through the Arduino IDE, making sure that the board is set to "Arduino Pro or Pro Mini" and the processor is set to "ATMega328 (5V, 16 MHz)".

Adding a Display

To add a variable to display onto the screen:

1. Define an ID at the beginning of OSD_SLIP.h and in HPV_ArduinoMega.ino
2. Perform calculations in the main loop of HPV_ArduinoMega.ino
3. Send the value in the main loop via `SlipPacketSend()`
4. Add an extern variable declaration in OSD_SLIP.cpp
5. Add the regular variable declaration in HPV_OSD.ino
6. Add a case to OSD_SLIP.cpp to extract the value from the buffer and assign it to the variable from steps 4 and 5
7. Add a panel function to OSD_panels_HPV.ino
8. Call the function in `writePanels()` (also located in OSD_panels_HPV.ino)

Parameters of Interest

Baud rate: `TELEMETRY_SPEED`, ArduCAM_OSD.ino

Start-up speed: `BOOTTIME`, ArduCAM_OSD.ino

Elevation Calculations

Elevation data is stored using a 2D array of distance and elevation pairs. To keep the data structure small, a few key points along the track were chosen so that linear interpolation could be used for a reasonable estimate of the elevation anywhere along the track. This data can be found near the beginning of simulation.cpp.

The change in elevation cannot be found without first knowing the distance travelled. The distance travelled cannot be known without calculating the net energy added to the bike, but part of this net energy is due to the change in elevation. This circular dependence necessitates an iterative algorithm.

1. First, elevation is ignored. Power, velocity, and distance travelled are calculated.
2. The distance travelled is used to find the change in elevation. Power is recalculated with the power of the elevation change included.
3. Velocity and distance calculations are repeated with the new power value.
4. The elevation change is recalculated. Power is recalculated.
5. Velocity and distance are recalculated.

And so on. Each iteration should result in a closer and closer approximation to the actual distance. The process stops after a maximum of ten iterations, or when the difference in elevation power is less than 1 W, whichever occurs first.

SRM Data

The SRM sends messages to the antenna at a non-configurable rate of every half a second. While the bike is being pedalled, the contents of the power messages will only update every full revolution. Any messages sent before the next full revolution will be a duplicate of the previous.

Commented [J1]: I can't remember if the time stamp updates or not.

Common Tasks

Add a speed profile

Create a new file on the SD card called "PF##" where the pound signs are the next unused profile number.

Note: Numbering must begin at 01 and all subsequent profile numbers must be consecutive. The program will stop looking for profiles if it cannot find a profile with the next consecutive number.

The first argument is the profile name. The name should be a maximum length of 10 characters and cannot contain spaces. A profile name longer than the limit will be truncated when it is displayed. The next six arguments are the coefficients for the polynomial approximation in order of highest power (degree 6) to lowest (constant term). Each argument must be separated by a space.

Change the target power profile

The target power profile currently uses a simple linear equation to find the target value. The starting and ending targets are defined as preprocessor variables `POWER_START` and `POWER_PRE_SPRINT`. To change the method of calculation, modify functions `calcPower()` and `calcDisplacePower()` in TargetSpeed_Func.ino.

Set the finish line coordinates

Edit the file on the SD card called "FinCoord.txt". The first argument is the latitude (in degrees), the second is the longitude (in degrees), and the third is the altitude (in metres). Each argument must be separated by a space.

Troubleshooting

No calibration data is received (display will only read "Waiting")

This is likely a problem with the antenna. If certain antenna pins short, the antenna will not work until the short is removed and the antenna is reinitialized.

1. Unplug and plug the antenna. Ensure nothing is shorting the pins. Restart the program.
2. Replace the antenna with the backup.
3. Measure the three baud rate selection pins while the Arduino is powered and ensure their logic levels match the ones specified for the baud rate used in the program (see Antenna Datasheet.pdf).
4. Comment out the call to `calibrate()` in HPV_ArduinoMega's `setup()`. Run the program using the serial monitor. If there are no ANT+ messages received after the ANT+ setup has complete, the problem is definitely with the antenna.

At this point, if the problem persists, switch to a back-up system and try again later.

Power, cadence, distance and heart rate displays freeze

These are all dependent on the antenna. See above.

The power meter will automatically shut off after a specified period of no activity (around 2 minutes). After this time, the program will stop calculating new values.

GPS not available

1. Check that the device is getting power (red LED should be lit).
2. Restart the system.
3. Try the old GPS. If it works, the new one is likely damaged or misconfigured.
4. Start U-Center and plug the GPS into the computer. Ensure the correct messages are being sent, the baud rate is configured correctly, and the message rate is reasonable. (Refer to the GPS configuration section). If U-Center can't communicate with the GPS, it may be damaged. Use the backup GPS.

OSD text flickers on screen

1. Ensure the OSD wire connections are stable and nothing is touching the OSD chip.

2. Ensure any calls to `OSD.clear()` are controlled appropriately (do not attempt to call the function without a time delay).
3. Ensure the display and OSD chip are both set on NTSC mode (see OSD section).

A Value jumps to an extremely large number

This is most likely caused by a division by zero (and not some hidden modification of variable values or overwriting memory where it shouldn't be overwritten). Check any calculations related to the variable and ensure that undefined operations cannot occur.

Known Issues

Arduino peripherals (e.g. GPS) stop working after uploading new code

Cases

Uploading the code using Justin's laptop running Arduino 1.5.5 (older, no longer supported), the code works as expected. When identical code is uploaded using Steven's laptop running Arduino 1.0.5 (newest version of the 1.0 series), the GPS will not initialize.

Analysis

Compiling on different versions of the IDE (or possibly on different computers) may break the code. In the given case, it seems to be related to the pre-processor variables that detect what Arduino board the GPS is running on (see `GPS_UBLOX_Class::Init` in GPS_UBLOX.cpp). More testing is required to determine the cause of this. We suspect this problem can be fixed by removing the pre-processor variables, forcing the code to assume an ATMega board.

Serial Buffer Overflow – Solved

Cases

The ANT+ serial will occasionally receive corrupted messages that can be up to 255 bytes long. This is highly dangerous, as the ANT+ buffer is only large enough to support messages up to 64 bytes. (In theory, no messages should be above 9 bytes in length.) Additional bytes will write into adjacent memory, overwriting any variables that may be stored there. This behaviour is completely undefined.

Analysis and Solution

It seems as though these large, "cancerous" messages are actually composed of a few 9-byte messages repeated many times.

The code relies on a sync byte (0xA4 or 0xA5, although in practice it always seems to be 0xA4 and the code ignores 0xA5 for extra security) to identify the beginning of a message. The program is usually fine at synchronizing with the ANT+ serial, but sometimes it misinterprets a 0xA4 that is part of a message payload to be the sync byte of a new message. It will then assume that the following byte is the message length, causing this error.

A check has been put into place in `receiveANT()` of ANT_interface.cpp to discard the message as soon as the message length byte is greater than 9. This means an overflow should never occur with `antBuffer`.

SD card contains strange files

Cases
The SD card is susceptible to corruption. Our testing has shown the following cases:

- A file with the name “ . ■” that Windows Explorer thinks takes up around 4GB of drive space. Pulling up the Properties window, however, shows that it is an empty (0B) file.
- A “ghost” file that takes up no space and cannot be deleted (deleting is successful, but refreshing the directory will cause the file to reappear).

Analysis
This problem likely stems from the fact that the SD card can be removed at any time, including during a write operation. There is a library function to initialize the SD card, but no function to disconnect safely from the SD.

Simulation – Solved???

Cases
The `simulate` function risks dividing by 0, which may cause undefined behaviour. On the Arduino, variables will become infinity or NaN (not a number). On the OSD, they may display as garbage values. As soon as velocity and/or distance become either of these values, they will not recover.

Analysis and Solution
Steps have already been taken to prevent floating point calculations from ruining the program. Guards are present in many `simulate()` calculations to attempt to handle unreasonable values, and `simulate()` includes a final check that will abort the current iteration without updating any of the values if any variables become infinite or undefined. Recent testing and logging have not revealed any problems with infinite or undefined variables; however, they have also not revealed any cases in which undefined variables were actually caught and handled gracefully. Therefore, it is not known if the current security measures are effective.

Shorting – Solved

Cases
When standoffs were still installed on the box and the Arduino was powered without properly locking it in place, the board heated rapidly, becoming hot enough to burn the epoxy.

Analysis and Solution
We speculate that one of the standoffs came into contact with the V_{in} pin on the bottom of the Arduino and shorted it by conducting across the carbon fibre board. **Never run the circuit while the Arduino is sitting on a conductive surface.**

The nature of the problem could have been avoided by replacing the standoffs with foam or another insulating material. The box is now taped to prevent this from happening in the future.

OSD shows rapidly flashing garbage numbers in place of meaningful ones – Solved

Cases
Whenever a value is rapidly written to the OSD (no explicit time delay between writes), the value flickers to random numbers.

Analysis and solution
The reason for this is unknown, but the problem can be solved by ensuring an adequate delay between write operations.

Miscellaneous Notes

Forward Declarations

- Arduino automatically generates forward declarations for functions. It is therefore never necessary to do so for most functions.
    - Exceptions include functions with function pointers as parameters, default arguments, and functions declared in a namespace or class. Manual declarations will be necessary.

Multi-file Programs

- Arduino (.pde and .ino) files do not work like regular .c or .cpp files. The Arduino compiler concatenates all Arduino files in the same folder into one compilation unit. (C and C++ files are treated as separate compilation units.) The concatenation begins with the main file (which shares the folder name), followed by the other Arduino files in alphabetical order (not verified, widely accepted). Therefore, **for Arduino files in the same folder**:
    - Include statements are only necessary in the main file
    - The `static` keyword will not restrict variable scope to only the local Arduino file
    - Global variables in one file do not have to be declared as `extern` in another to be used
    - Regular variable scope rules still apply. A global variable declared in "B.ino" cannot be referenced in "A.ino" because A will concatenated before B.

SD Card

- Arduino cannot log files with names longer than 8 characters (FAT 8.3 standard).
- Arduino capitalizes all letters in file names when creating a file.