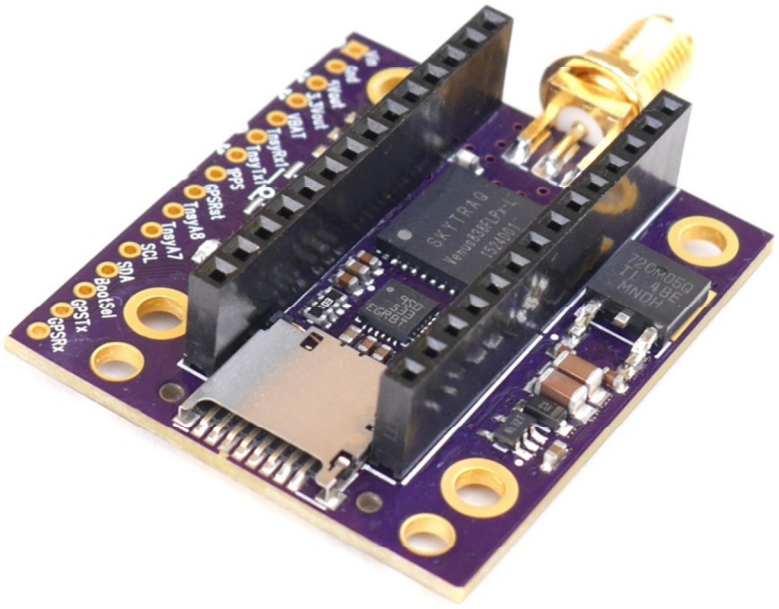
TeensyGPS Shield

Software Description Document



Contents

List of Tables

List of Tables

Purpose

The purpose of this document is to give a detailed description of the "TeensyGPS Data Logger". This will explain the purpose, functionality and system constraints necessary for the software developers to develop/modify software libraries and customize the TeensyGPS to their specific needs.

Scope

The "Teensy GPS Shield" is a module that interfaces with an Arduino Teensy 3.x microcontroller and has 4 basic features:

1. MicroSD for Data Logging
2. GPS Receiver
3. 9DOF Sensor
4. CAN Bus Transceiver

These 4 features provide the essential data needed in many real world applications including RC aircraft, RC cars, UAVs, automotive racing, motorsports and various other hobby and professional fields. In addition, having sufficient flexibility built into the data logging application would provide a system that could be useful for scientific studies such as measuring sea ice drift and at the same time being cost effective for the general hobbyist to use in RC vehicles.

The heart of the Teensy GPS Shield will be the Teensy 3.x microcontroller. The Teensy microcontroller is Arduino compatible and all software will be written for the Arduino platform. Ultimately the software will be open source and available for the general public to download and modify. This software effort is being developed by multiple individuals and an existing set of libraries will serve as a starting point for any new developers brought onto the team. The existing set of libraries can be found here: https://github.com/smokingresistor/TeensyGPS

Of the 4 features available, first basic feature is the GPS receiver which receives the satellite signal from multiple satellites and calculates current latitude, longitude, altitude and provides other useful data. The GPS receiver is based around the SkyTraq Venus838FLPX receiver chip and the datasheet can be found here: http://www.smokingresistor.com/wp-content/uploads/2014/06/Venus838FLPx_DS_v4.pdf Configuration of the GPS receiver is achieved through serial commands that will be transmitted from the Teensy microcontroller. Configuration parameters will be transmitted to the Venus838FLPX receiver chip every time the device is powered on. The existing libraries for the Teensy GPS Shield on Github already have these parameters defined.

The second feature is the 9DOF sensor based around the ST Microelectronics LSM9DS0TR chip. This chip has sensor data available in all 3 axes for acceleration, angular rates and magnetic fields. The 9DOF sensor has now been fully integrated into the Teensy GPS Shield and the data can be logged to

the MicroSD card or pushed out the CAN bus. The Arduino library for this particular LSM9DS0TR chip can be found here: https://github.com/adafruit/Adafruit_LSM9DS0_Library

The third feature is the CAN bus transceiver. The Teensy uses the FlexCAN library found at this link: https://github.com/teachop/FlexCAN_Library The Teensy communicates to a CAN bus transceiver chip part number SN65HVD232DR by Texas Instruments. The CAN bus interface is a DSUB 9 pin connector which is standard in most automotive CAN bus applications.

The final feature of the Teensy GPS shield is the data logging capability. The data from the GPS receiver and 9DOF sensor will be recorded to a MicroSD card. The GPS receiver has the capability of providing position data at a rate of 50Hz max. This data rate is configurable via a config.json file that will be read from the MicroSD card at power on. In order to provide the data logging needs for a variety of industries, a lot of flexibility has been built into the data logging library.

In addition, to these 4 primary features available on the Teensy GPS Shield, there is an additional requirement to have filtered GPS data. Due to the inaccuracies of commercial GPS position data, the 50 Hz max data rate from the GPS receiver tends to be inconsistent. For example, receiving 50 position updates within 1 second will result in 50 slightly different positions that don't overlay perfectly. To overcome these inconsistencies in the commercial GPS data, a Kalman filter has been added in an attempt to average this inconsistent position data.

Data Logging

The Teensy GPS data logger shall read the MicroSD card and use a JSON configuration file called "config.jsn" to configure the data logging parameters. The config.jsn file is a subset of the gpsd protocol standard found here: http://www.catb.org/gpsd/gpsd_json.html . The JSON file format is an open standard that uses human readable text and can be edited with any text editor. There are also many JSON viewers/editors available to make it easier to read and edit. The advantage of JSON is that it has convenient attribute-data pairs and is similar to XML but much simpler syntax for faster parsing.

The gpsd subset will be defined the config.jsn file as shown in tables 1 through 6. The user can choose which optional fields will be recorded.

CNF

The CNF object is a configuration object used to configure the Teensy GPS data logger. The information in this report will not be included in the data logs since it is input only.

Table 1: CNF Object

Name	Always?	Type	Description
class	No	string	Fixed: "CNF" for Venus838 configuration
log_en	No	numeric	Enable data logging to the MicroSD card 0: Disabled 1: Enabled
newlog	No	numeric	Generate new log file when passing through a beacon point 0: Disabled 1: Enabled
rate	No	numeric	GPS Position Update Rate with choices of 1, 2,4,5,8,10,20,25,40,50 Hz
canspeed	No	numeric	CAN bus speed defines how fast the CAN bus sends data 1000000: 1Mbps 500000: 500Kbps 250000: 250Kbps 100000: 100Kbps 50000: 50Kbps
size	No	numeric	Maximum size for each data log file (in MB) with a max of 999MB. The minimum value is 1MB.
blat	No	numeric	Beacon latitude
blong		numeric	Beacon longitude
btime	No	numeric	Beacon timeout (seconds). If you don't reach the beacon location within the timeout limit, a new log file will be generated.
btol	No	numeric	Beacon tolerance (meters). This should be no less than 15 meters.
log_type	No	numeric	Three types of data logging are possible: 0=continuous 1=event (continuous data logging will begin once event trigger is reached) 2=interval (logging only occurs within the interval that has been defined)
trig	No	numeric	Event Triggers are used to enable data logging when an event occurs. The possible events are: 0: longitude 1: latitude 2: altitude (meters) 3: speed over ground (meters per second)
trigv	No	numeric	Event Trigger Value is used for single event triggering only. Once the value in this field is obtained, logging will be enabled
intv	No	numeric	Interval Triggers are used to start and stop data logging. The possible intervals are: 0: Time elapsed after valid position fix (seconds) 1: Distance traveled after valid position fix (meters) 2: Speed over ground after valid position fix (meters/second)

Name	Always?	Type	Description
min	No	numeric	Minimum value to enable logging (ex: 60 seconds after valid position fix)
max	No	numeric	Maximum value to disable logging (ex: 3600 seconds after valid position fix)

TPV

A TPV object is a time-position-velocity report. The "class" and "mode" fields will reliably be present. The "mode" field will be emitted before optional fields that may be absent when there is no fix. Error estimates will be emitted after the fix components they're associated with. Others may be reported or not depending on the fix quality.

Table 2: TPV Object

Name	Always?	Type	Description
class	Yes	string	Fixed: "TPV"
device	Yes	string	Fixed: "Venus838"
mode	Yes	numeric	Fix mode: %d, 0=no fix, 1=2D, 2=3D, 3=3D+DGNSS
sv	No	Numeric	Number of SV in the fix (0 – 12)
time	No	string	Time/date stamp in ISO8601 format, UTC. May have a fractional part of up to .001sec precision. May be absent if mode is not 1,2 or 3.
lat	No	numeric	Latitude in degrees: +/- signifies North/South. Present when mode is 1, 2 or 3.
lon	No	numeric	Longitude in degrees: +/- signifies East/West. Present when mode is 1, 2 or 3.
alt	No	numeric	Altitude in meters. Present if mode is 2 or 3.
latf	No	numeric	Kalman filtered latitude in degrees: +/- signifies North/South. Present when mode is 1, 2 or 3.
lonf	No	numeric	Kalman filtered longitude in degrees: +/- signifies East/West. Present when mode is 1, 2 or 3.
altf	No	numeric	Kalman filtered altitude in meters. Present if mode is 2 or 3.
track	No	numeric	Course over ground, degrees from true north. This is the actual path traveled over the ground just as you were to leave “track” behind in the snow or sand.
speed	No	numeric	Speed over ground, meters per second.
speedf	No	numeric	Kalman filtered speed over ground, meters per second.
gdop	No	numeric	Geometric dilution of precision. Scaling 0.01

Name	Always?	Type	Description
pdop	No	numeric	Position dilution of precision. Scaling 0.01
hdop	No	numeric	Horizontal dilution of precision. Scaling 0.01
vdop	No	numeric	Vertical dilution of precision. Scaling 0.01
tdop	No	numeric	Time dilution of precision. Scaling 0.01
error	No	Numeric	Checksum errors that occur during communication with the Venus838

ATT

An ATT object is a vehicle-attitude report. It is returned by digital-compass and gyroscope sensors; depending on device, it may include: heading, pitch, roll, yaw, gyroscope, and magnetic-field readings.

The "class" field will reliably be present unless all other flags are set to false. Others may be reported or not depending on the specific device type.

Table 3: ATT Object

Name	Always?	Type	Description
class	Yes	string	Fixed: "ATT"
device	Yes	string	Fixed: "LSM9DS0TR"
time	No	numeric	Time/date stamp in ISO8601 format, UTC. May have a fractional part of up to .001sec precision. May be absent if mode is not 1, 2 or 3.
heading	No	numeric	Heading, degrees from true north.
pitch	No	numeric	Pitch in degrees.
yaw	No	numeric	Yaw in degrees
roll	No	numeric	Roll in degrees.
dip	No	numeric	Local magnetic inclination, degrees, positive when the magnetic field points downward (into the Earth).
mag_len	No	numeric	Scalar magnetic field strength.
mag_x	No	numeric	X component of magnetic field strength (gauss)
mag_y	No	numeric	Y component of magnetic field strength (gauss)
mag_z	No	numeric	Z component of magnetic field strength (gauss)
acc_len	No	numeric	Scalar acceleration.
acc_x	No	numeric	X component of acceleration (g)
acc_y	No	numeric	Y component of acceleration (g)

Name	Always?	Type	Description
acc_z	No	numeric	Z component of acceleration (g)
gyro_x	No	numeric	Angular rate (degrees/sec) about the X axis
gyro_y	No	numeric	Angular rate (degrees/sec) about the Y axis
gyro_z	No	numeric	Angular rate (degrees/sec) about the Z axis
quat1	No	numeric	Quaternion 1
quat2	No	numeric	Quaternion 2
quat3	No	numeric	Quaternion 3
quat4	No	numeric	Quaternion 4
temp	No	Numeric	Temperature (°C)
pressure	No	Numeric	Barometric Pressure (mbar)

CAN

The CAN object is a configuration object used to configure the Teensy GPS CAN frames. Each CAN frame is 8 bytes total. The information in this report will not be included in the data logs since it is input configuration only.

Table 4: CAN Object

Name	Always?	Type	Description
class	No	string	Fixed: "CAN" for Venus838 configuration
can00	No	array	CAN Bus Frame 0 is defined with an array of 2 parameters as follows: [can_en00, canid00] can_en00: 0=Disabled, 1=Enabled canid00: CAN Bus ID 0 to 2047 decimal
can01	No	array	CAN Bus Frame 1 is defined with an array of 2 parameters as follows: [can_en01, canid01] can_en01: 0=Disabled, 1=Enabled canid01: CAN Bus ID 0 to 2047 decimal
can02	No	array	CAN Bus Frame 2 is defined with an array of 2 parameters as follows: [can_en02, canid02] can_en02: 0=Disabled, 1=Enabled

Name	Always?	Type	Description
			canid02: CAN Bus ID 0 to 2047 decimal
can03	No	array	CAN Bus Frame 3 is defined with an array of 2 parameters as follows: [can_en03, canid03] can_en03: 0=Disabled, 1=Enabled canid03: CAN Bus ID 0 to 2047 decimal
can04	No	array	CAN Bus Frame 4 is defined with an array of 2 parameters as follows: [can_en04, canid04] can_en04: 0=Disabled, 1=Enabled canid04: CAN Bus ID 0 to 2047 decimal
can05	No	array	CAN Bus Frame 5 is defined with an array of 2 parameters as follows: [can_en05, canid05] can_en05: 0=Disabled, 1=Enabled canid05: CAN Bus ID 0 to 2047 decimal
can06	No	array	CAN Bus Frame 6 is defined with an array of 2 parameters as follows: [can_en06, canid06] can_en06: 0=Disabled, 1=Enabled canid06: CAN Bus ID 0 to 2047 decimal
can07	No	array	CAN Bus Frame 7 is defined with an array of 2 parameters as follows: [can_en07, canid07] can_en07: 0=Disabled, 1=Enabled canid07: CAN Bus ID 0 to 2047 decimal

FLS

The FLS (Finish Line Segment) object is a configuration object used to configure the Teensy GPS for motorsport racing. The information in this report will not be included in the data logs since it is input configuration only.

Table 5: FLS Object

Name	Always?	Type	Description
class	No	string	Fixed: "FLS" for Venus838 configuration

Name	Always?	Type	Description
finish	No	array	The “finish” line class defines a virtual finish line segment defined by 2 GPS latitude/longitude coordinates Point A and Point B. When the Teensy GPS passes through this virtual line segment and meets the min/max speed requirements, the output is toggled. If the min/max speeds are not defined, then these parameters are ignored. The array is defined as follows: [finish, fin_latA, fin_longA, fin_latB, fin_longB, fin_out, fin_max, fin_min] finish: 0=Disabled, 1=Enabled fin_latA: Finish line latitude Point A in decimal format fin_longA: Finish line longitude Point A in decimal format fin_latB: Finish line latitude Point B in decimal format fin_longB: Finish line longitude Point B in decimal format fin_out: Disabled, Low transition, High transition fin_max: Maximum finish line speed in m/s fin_min: Minimum finish line speed in m/s
pit_entry	No	array	The “pit_entry” line class defines a virtual finish line segment defined by 2 GPS latitude/longitude coordinates Point A and Point B. When the Teensy GPS passes through this virtual line segment and meets the min/max speed requirements, the output is toggled. If the min/max speeds are not defined, then these parameters are ignored. The array is defined as follows: [pit_entry, pitn_latA, pitn _longA, pitn _latB, pitn _longB, pitn _out, pitn _max, pitn _min] pit_entry: 0=Disabled, 1=Enabled pitn _latA: Pit entry latitude Point A in decimal format pitn _longA: Pit entry longitude Point A in decimal format pitn _latB: Pit entry latitude Point B in decimal format pitn _longB: Pit entry longitude Point B in decimal format pitn _out: Disabled, Low transition, High transition pitn _max: Maximum finish line speed in m/s pitn _min: Minimum finish line speed in m/s
pit_exit	No	array	The “pit_exit” line class defines a virtual finish line segment defined by 2 GPS latitude/longitude coordinates Point A and Point B. When the Teensy GPS passes through this virtual line segment and meets the min/max speed requirements, the output is toggled. If the min/max speeds are not defined, then these parameters are ignored. The array is defined as follows: [pit_exit, pitx_latA, pitx _longA, pitx _latB, pitx _longB, pitx _out, pitx _max, pitx _min] pit_exit: 0=Disabled, 1=Enabled pitx _latA: Pit entry latitude Point A in decimal format pitx _longA: Pit entry longitude Point A in decimal format

Name	Always?	Type	Description
			pitx _latB: Pit entry latitude Point B in decimal format pitx _longB: Pit entry longitude Point B in decimal format pitx _out: Disabled, Low transition, High transition pitx _max: Maximum finish line speed in m/s pitx _min: Minimum finish line speed in m/s

PIT

TBD

Table 6: PIT Object

Name	Always?	Type	Description
class	No	string	Fixed: "PIT" for Venus838 configuration
pit_length	No	numeric	TBD
pit_max_spd	No	numeric	TBD
pit_time	No	numeric	TBD
pit_acc	No	numeric	TBD
ir_beacon	No	numeric	TBD
gps_beacon	No	numeric	TBD

CONFIG.JSN

The following is an example of a config.jsn file that will be read by the TeensyGPS data logger. By simply adding a true or false next to the key attribute determines whether this field gets logged to the MicroSD card. Any key attributes that are omitted will be considered false and not get logged.

```
{
        "class" : "CNF",
        "log_en" : "1",
        "newlog" : "0",
        "rate" : "50",
        "canspeed" : "1000000",
        "size" : "10",
        "blat" : "",
        "blong" : "",
        "btime" : "600",
```

```
        "btol" : "15",
        "log_type" : "0",
        "trig" : "",
        "trigv" : "",
        "intv" : "",
        "min" : "",
        "max" : ""
}{
        "class" : "TPV",
        "device" : "true",
        "mode" : "true",
        "sv" : "false",
        "time" : "true",
        "lat" : "true",
        "lon" : "true",
        "alt" : "true",
        "latf" : "false",
        "lonf" : "false",
        "altf" : "false",
        "track" : "false",
        "speed" : "true",
        "speedf" : "false",
        "gdop" : "false",
        "pdop" : "false",
        "hdop" : "false",
        "vdop" : "false",
        "tdop" : "false",
        "error" : "false"
}{
        "class" : "ATT",
        "device" : "true",
        "time" : "true",
        "heading" : "false",
        "pitch" : "false",
        "yaw" : "false",
        "roll" : "false",
        "dip" : "false",
        "mag_len" : "false",
        "mag_x" : "false",
        "mag_y" : "false",
        "mag_z" : "false",
        "acc_len" : "false",
```

```
        "acc_x" : "false",
        "acc_y" : "false",
        "acc_z" : "false",
        "gyro_x" : "false",
        "gyro_y" : "false",
        "gyro_z" : "false",
        "quat1" : "false",
        "quat2" : "false",
        "quat3" : "false",
        "quat4" : "false",
        "temp" : "false",
        "pressure" : "false"
}{
        "class" : "CAN",
        "can00" : ["1","300"],
        "can01" : ["0","301"],
        "can02" : ["1","302"],
        "can03" : ["0","303"],
        "can04" : ["0","304"],
        "can05" : ["0","305"],
        "can06" : ["0","306"],
        "can07" : ["0","307"]
}{
        "class" : "FLS",
        "finish" : ["0","","","","","Disabled","0","0"],
        "pit_entry" : ["0","","","","","Disabled","0","0"],
        "pit_exit" : ["0","","","","","Disabled","0","0"]
}{
        "class" : "PIT",
        "pit_length" : "",
        "pit_max_spd" : "",
        "pit_time" : "",
        "pit_acc" : "",
        "ir_beacon" : "",
        "gps_beacon" : ""
}
```


Once the config.jsn file has been read by the Teensy GPS, the data logging files will be written to the MicroSD card in comma separated variable (CSV) format and each log file will include a single header row. The single header row starts with the TPV message class, TPV parameters being logged, ATT class and the ATT parameters being logged. If there are parameters flagged “false” in the config.jsn file,

these parameters will not be in the header row and their data will not be logged. Since the user has the option of choosing which parameters get logged, the header file will be the key to decoding the data contained within.

After the single header row is written, the data that follows will also be in CSV format and only that parameters that have been flagged as true will be written to the data log. An example of a data log will be as follows:

Example Data

class,mode,time,lat,lon,alt,track,speed,climb,class,heading,pitch,roll,mag_x
TPV,2,2005-06-08T10:34:48.283Z,46.49829337,7.567411672,1343.127,10.3788,0.091,-0.085,ATT,14223.00,169.00,-43.00,2454.000
TPV,2,2005-06-08T10:34:48.283Z,46.49829337,7.567411672,1343.127,10.3788,0.091,-0.085,ATT,14223.00,169.00,-43.00,2454.000
TPV,2,2005-06-08T10:34:48.283Z,46.49829337,7.567411672,1343.127,10.3788,0.091,-0.085,ATT,14223.00,169.00,-43.00,2454.000
TPV,2,2005-06-08T10:34:48.283Z,46.49829337,7.567411672,1343.127,10.3788,0.091,-0.085,ATT,14223.00,169.00,-43.00,2454.000

Power-On Events

When the Teensy GPS power on, here are the 3 possible scenarios that would occur on boot.

1. SD Card not detected: No data will be logged
2. No config.json file found: If the data logger cannot find the config.json file then it will revert to the existing configuration settings found in non-volatile memory. This means that if you have been using the data logger and you insert a newly formatted MicroSD card, your system will maintain its current settings that had previously been stored in non-volatile memory.
3. Config file found: During power up, the data logger will look for a config.json file. If the file is found, the data logger will use those settings and overwrite any previously stored system settings in non-volatile memory.

The data logger will create a new file each time it runs; the highest-numbered file is the most recent log. The file naming convention will be LOG00001.csv. Log files will increment with most recent number stored in non-volatile EEPROM. The max number of log files will be 65533 and then will roll over and start again with LOG00001.csv overwriting any files that currently exist with the same name. If the MicroSD card is full, then data will no longer be written to the MicroSD card.

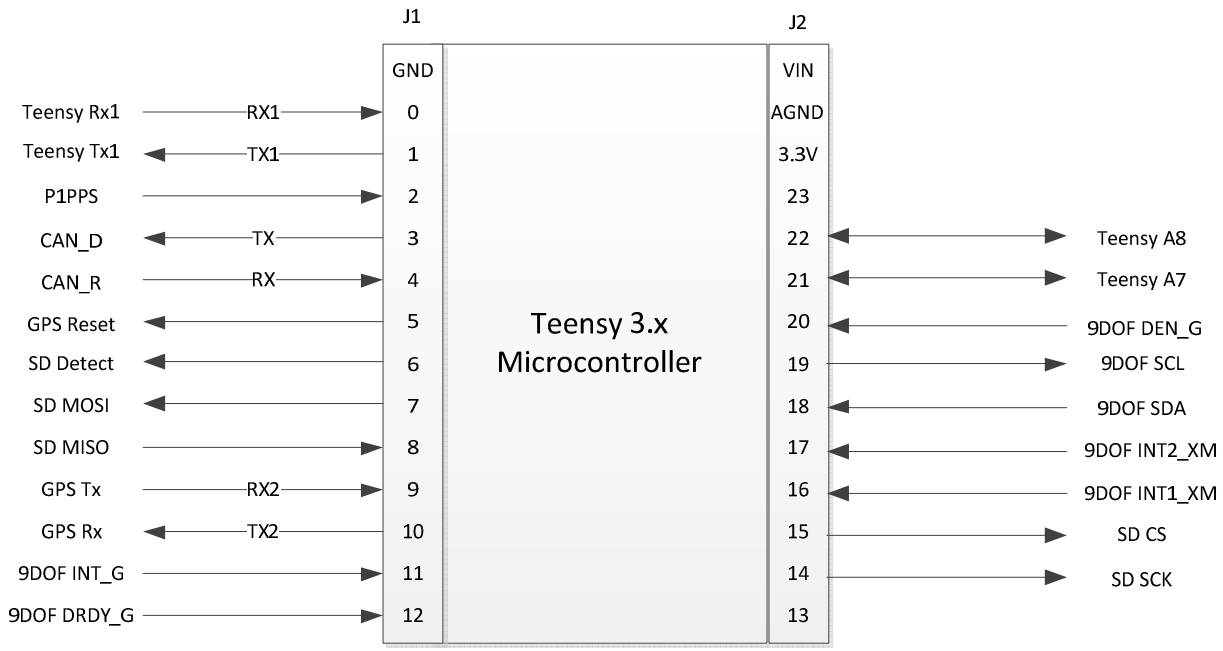


Figure 1: Teensy Pinout

MicroSD

The MicroSD card is connected to the Teensy microcontroller via an SPI bus. The SD card detect signal is connected to the Teensy on pin 6. The SPI bus is connected to the Teensy on pins 7, 8, 14, 15 as shown in Figure1.

GPS Receiver

The SkyTraq Venus838FLPX GPS receiver is connected to the Teensy microcontroller via pins 9 &10 as shown in Figure 1. The GPS receiver communicates with the Teensy via serial channel at 115,200 baud with 8 data bits, no parity and 1 stop bit. There will only be 1 command that needs to be sent from the Teensy to the Venus838 chip which will be the position update rate which varies from 1 Hz to 50 Hz and is defined in the CNF object as “rate”. After reading the desired position update rate from the config.jsn file, the update rate value will be written to the Venus838 flash configuration. Once the position update rate is sent to the GPS receiver, the Teensy will need to start processing the data from the GPS receiver. If there is no MicroSD card or there is an error reading the config.jsn file, then the command to change the position update rate will not be sent.

The data received from the GPS will be in a binary format and the binary message protocol is defined in this document: http://www.smokingresistor.com/wp-content/uploads/2016/06/AN0028_1.4.33.pdf

9DOF Sensor

The LSM9DS0 9DOF sensor connects to the Teensy via I2C bus on Teensy pins 18 and 19 as shown on Figure 1. The 9DOF sensor is initialized on boot up and configured to output the appropriate data as defined in the TPV object. Once the 9DOF sensor is configured properly, the Teensy microcontroller will read the 9DOF sensor output at the same rate which it's reading the GPS data. The 9DOF sensor data will then be logged to the MicroSD card based on the config.jsn file.

CAN Bus

The CAN bus transceiver chip SN65HVD232DR is connected to the Teensy via serial channel on Teensy pins 3 & 4 as shown in Figure 1. The config.jsn file will define the CAN bus data rate and have pre-configured CAN bus frames that can be enabled.

Table 7: CAN Bus Frames

			Bits							
CAN ID (hex)	Frame	8 Bytes	7	6	5	4	3	2	1	0
300	CAN Packet Byte	0	msb							
		1					LATITUDE			
		2								
		3								lsb
		4	msb							
		5					LONGITUDE			
		6								
		7								lsb
301	CAN Packet Byte	0	msb							
		1					LATITUDE FILTERED			
		2								
		3								lsb
		4	msb							
		5					LONGITUDE FILTERED			
		6								
		7								lsb
302	CAN Packet Byte	0	msb							
		1					ALTITUDE			
		2								lsb
		3	msb							
		4					SPEED			
		5								lsb
		6	msb							
		7					COURSE			
										lsb
			msb				CHECKSUMS ERRORS			lsb
			msb	GPS FIX		lsb	msb	SATELLITE		lsb

303	CAN Packet Byte	0	msb	ALTITUDE FILTERED	
		1			lsb
		2	msb	SPEED FILTERED	
		3			lsb
		4	msb	COURSE OVER GROUND	
		5			lsb
		6	msb	LAP DISTANCE	
		7			lsb
304	CAN Packet Byte	0	msb	ROLL	
		1			lsb
		2	msb	PITCH	
		3			lsb
		4	msb	YAW	
		5			lsb
		6	msb	Pressure	
		7			lsb
305	CAN Packet Byte	0	msb	BEACON	lsb
		1	msb	TEMPERATURE	lsb
		2	msb	ACCX	
		3			lsb
		4	msb	ACCY	
		5			lsb
		6	msb	ACCZ	
		7			lsb
306	CAN Packet Byte	0	msb	Q1	
		1			lsb
		2	msb	Q2	
		3			lsb
		4	msb	Q3	
		5			lsb
		6	msb	Q4	
		7			lsb
307	CAN Packet Byte	0	msb	Gx	
		1			lsb
		2	msb	Gy	
		3			lsb
		4	msb	Gz	
		5			lsb
		6	msb	Unused	
		7			lsb

Kalman Filter

The Kalman filter is implemented in the Teensy microcontroller and will be used to filter the GPS data. Specifically, the Kalman filter will be applied to Latitude, Longitude, Altitude and Speed. These four filtered parameters will only get logged to the MicroSD card if they are enabled in the config.jsn file.