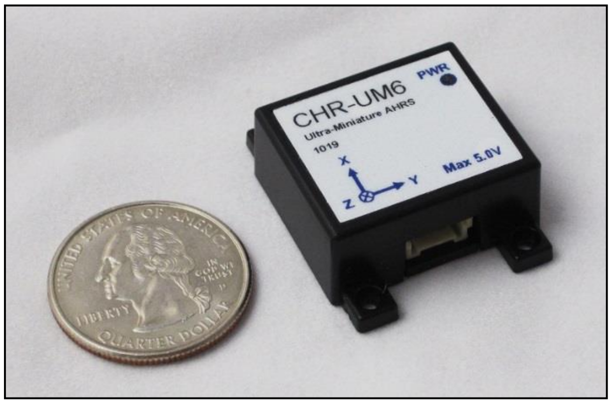
UM6 Ultra-Miniature Orientation Sensor Datasheet

Document rev. 2.4, updated 10/14/2013

1. Introduction

Device Overview



The UM6 Ultra-Miniature Orientation Sensor combines sensor measurements from rate gyros, accelerometers, and magnetic sensors to measure orientation at 500 Hz. The UM6 also has the capability to interface with external GPS modules to provide position, velocity, course, and speed information.

Communication with the UM6 is performed over either a TTL (3.3V) UART or a SPI bus.

The UM6 is configured by default to automatically transmit data over the UART. The UM6 can be configured to automatically transmit raw sensor data, processed sensor data, angle estimates, and angle estimate covariances at user configurable rates ranging from 20 Hz to 300 Hz in roughly 1 Hz increments. The UM6 can also receive and parse GPS packets, automatically transmitting new GPS position, velocity, and satellite data whenever it is available. Alternatively, the UM6 can operate in "silent mode," where data is transmitted only when specific requests are received over the UART. Regardless of the transmission mode and rate, internal angle estimates are updated at 500 Hz to improve accuracy.

The UM6 simplifies integration by providing a number of automatic calibration routines, including rate gyro bias calibration, magnetometer hard and soft iron calibration, and accelerometer "zeroing" to compensate for sensor-platform misalignment. All calibration routines are triggered by sending simple commands over the serial interface.

The UM6 comes factory-calibrated to remove soft and hard iron distortions present in the enclosure. When integrated into the end-user system, additional calibration may be necessary to correct other magnetic field distortions. Magnetometer calibration can be performed using the UM6 interface software, available for free download from www.chrobotics.com/downloads.

Temperature compensation of rate gyro biases is also supported by the UM6. An internal temperature sensor is used to measure temperature, and third-order compensation is applied to remove the effects of temperature-induced bias. By default, the terms used in compensation are all zero, which means that no temperature compensation is performed. The compensation terms must be determined experimentally by the end-user. On special request, compensation can be performed on each device at the factory.

The UM6 can be configured to use either Euler Angles or quaternions for attitude estimation. In Euler Angle mode, magnetometer updates are restricted to yaw alone. This can be useful in cases where distortions are possible or even expected, and where it would be undesirable for those distortions to affect pitch and roll angles (i.e. on a flying rotorcraft). In quaternion mode, Euler Angles are still available, but there are no restrictions on what angles the magnetometer is allowed to influence.

The UM6 is available in an OEM version (the UM6-LT) that has a slightly larger footprint and does not include an enclosure. The UM6-LT is functionally equivalent to the UM6, but magnetometer calibration is not performed at the factory.

A variety of expansion boards are also available to simplify integration of the UM6 into a wide variety of platforms.

Sensor Outputs (User-Configurable)

- GPS position, heading, and velocity (with connected GPS)
- GPS position in meters from customizable home location
- Euler Angles
- Quaternions
- Raw gyro, accel, and mag data
- Processed sensor data (scale factors applied, biases removed)
- Attitude estimate covariance

Summary of Features

- Supports GPS connectivity and data parsing
- GPS position can be reported in meters from user-customizable home position
- Automatic gyro bias calibration
- Cross-axis misalignment correction[1]
- Rate gyro temperature compensation[2]
- TTL UART or SPI bus communication
- Adjustable serial output rates (20 Hz - 300 Hz) and baud rate (up to 115200 baud)
- Onboard 3.3V regulator simplifies integration
- Open-source firmware with free development tools
- Open-source PC software for data visualization and AHRS configuration

Common Applications

- Robotics
- Platform Stabilization
- Motion Tracking
- Enhanced GPS Navigation
- General Motion Sensing
- Image Stabilization
- Human Limb Tracking

[1] Soft and hard iron calibration matrices and cross-axis alignment matrices must be determined and set by the end user. The UM6 configuration software includes routines for computing soft and hard iron calibration data.
[2] Temperature calibration terms are not computed at the factory unless specifically requested.

Table of Contents

List of Tables

2. Revision History

Rev. 1.0 - Initial Release
Rev. 1.1 - Removed information describing unsupported self-test feature
Rev. 2.0 - Added descriptions of GPS communication and SPI communication features
Rev. 2.1 - Added SPI bus information and fixed various small discrepancies
Rev. 2.2 - Adjusted SPI documentation
Rev. 2.3 - Fixed SPI bus clock polarity information
Rev. 2.4 – Added more details to the SPI communication section. Added UART examples.

3. Absolute Maximum Ratings

Table 1 - UM6 Absolute Maximum Ratings
Note: operating the UM6 at or above its maximum ratings can cause permanent damage to the device and is not recommended.

Symbol	Ratings	Maximum Value	Unit
Vdd	Supply voltage	-0.3 to +6.5V	V
Vin	Input voltage on any digital IO pin	-0.3 to 5.1 V	V
A	Acceleration	3000 *g* for .5 ms	
		10000 *g* for .1 ms	
Top	Operating temperature range	-40 to +85	°C
Tstg	Storage temperature range	-40 to +125	°C

4. Electrical Characteristics

Table 2 - UM6 Electrical Characteristics

Symbol	Parameter	Test condition	Min.	Typ.	Max.	Unit
Vdd	Supply voltage		3.5	3.5 - 5.0	5.0	V
Idd	Supply current	+5V supply voltage	50	52	58	mA

Table 3 - Gyro Characteristics

Symbol	Parameter	Test Condition	Min.	Typ.	Max.	Unit
FSA	Measurement Range			+/- 2000		°/s
CAS	Cross-axis sensitivity				2	%
SSF	Sensitivity Scale Factor		13.513	14.375	15.238	LSB/°/s
SSV	Sensitivity Scale Factor Variation over Temperature	-40°C to +85°C		+/- 10		%
OffDr	Zero-rate level change vs. temperature	-40°C to +85°C		+/- 40		°/s

NL	Non linearity	Best fit straight line		+/- 0.2		% FS
LnAS	Linear Acceleration Sensitivity	Static		0.1		°/s/g
Rn	Rate noise density			0.03		°/s/rt Hz

Table 4 - Accelerometer Characteristics

Symbol	Parameter	Test Condition	Min.	Typ.	Max.	Unit
aFS	Measurement Range			+/- 2		*g*
SoXYZ	Sensitivity	Vdd = 3 V	0.9	1	1.1	*mg/LSB*
Stemp	Sensitivity change due to temperature			0.01		%/°C

5. Pin Configuration and Functional Descriptions

Table 5 - UM6 Pin Descriptions (bottom pin headers)

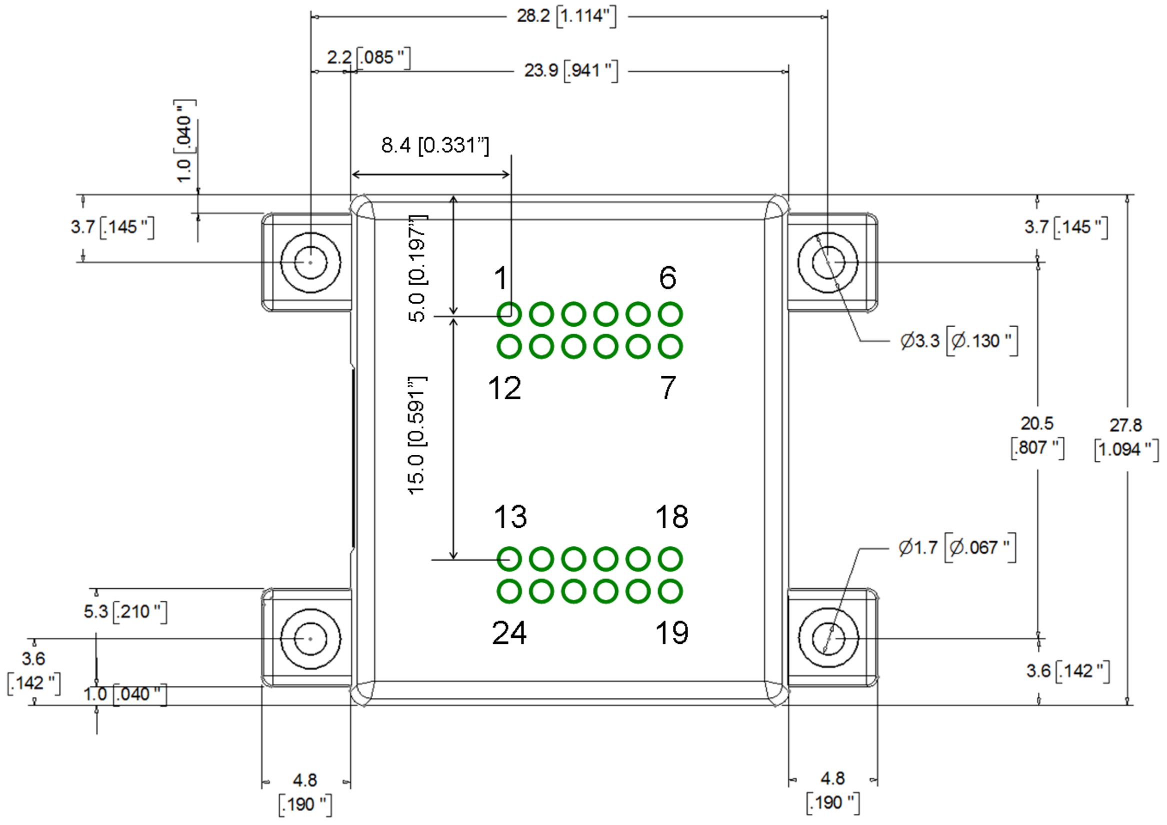
Pin Description		
Pin #	Pin Name	Description
1	+3.3V	3.3V output
2	GND	Ground
3	MOSI	SPI MOSI pin
4	MISO	SPI MISO pin
5	SCK	SPI SCK pin
6	SS	SPI SS pin
7	Vin	Input voltage: must be between 3.5 V and 5V
8	GND	Ground
9	SCL	i2c SCL line (10k internal pullup)
10	SCL	Connected internally to pin 9
11	SDA	i2c SDA line (10k internal pullup)
12	BOOT0	Programming pin for upgrading firmware. Pulling this pin high before applying power causes the firmware bootloader to start.
13	PA0	MCU pin PA0
14	RX2	RX pin for UART2 (Output from GPS)
15	PA4	MCU pin PA4
16	PA6	MCU pin PA6
17	PB0	MCU pin PB0
18	RX1	RX pin for UART1
19	TX1	TX pin for UART1
20	PB1	MCU pin PB1
21	PA7	MCU pin PA7
22	PA5	MCU pin PA5
23	TX2	TX pin for UART2 (Input from GPS)
24	PA1	MCU pin PA1

Table 6 - UM6 Pin Descriptions (side connector)

Pin #	Pin Name	Description
1	Vdd	Input voltage (3.5V to 5V)
2	GND	Supply ground
3	RX	UART RX pin (input)
4	TX	UART TX pin (output)

6. Mechanical Drawings

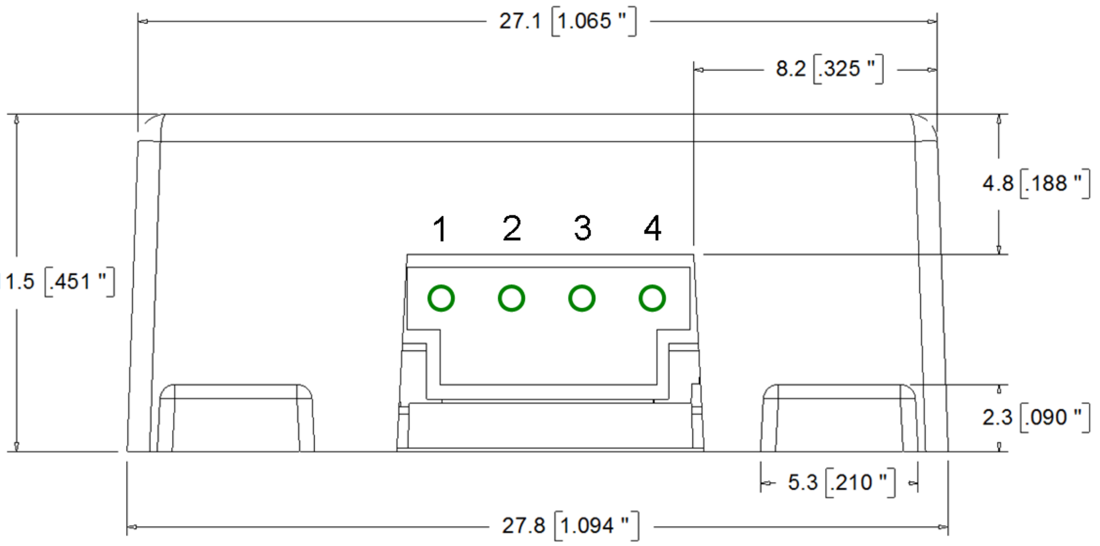
Top View
mm (inches)



Part number 609-2890-ND from Digi-Key is a compatible 2mm pin header for connecting to the bottom connectors on the UM6.

Side View

mm (inches)



Part number A98655-ND from Digi-Key is a compatible connector for the 4-pin side output of the sensor (compatible connecting cables are also available from CH Robotics).

7. UM6-LT Pin Configuration and Functional Descriptions

Top View (note: drawing is not to scale)

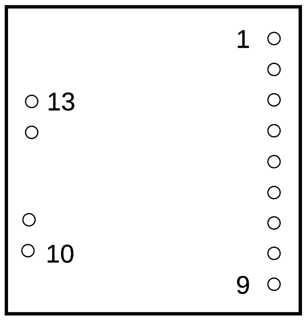


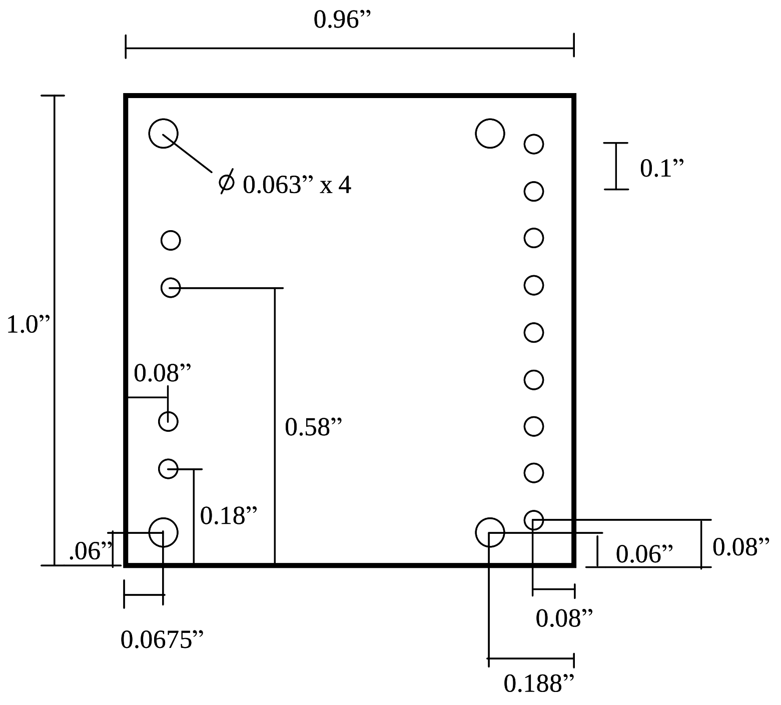
Table 7 - UM6-LT Pin Descriptions

Pin Description		
Pin #	Pin Name	Description
1	TX	TX (output)
2	RX	RX (input)
3	PGM	Bootloader activation pin. Pull high to activate bootloader - leave disconnected for normal operation.
4	D1	GPIO pin 1 (TX2)
5	D2	GPIO pin 2 (RX2)
6	D3	GPIO pin 3 (MOSI)

7	D4	GPIO pin 4 (MISO)
8	D5	GPIO pin 5 (SCK)
9	D6	GPIO pin 6 (SS)
10	Vdd	Input voltage (3.3 - 5.0 V)
11	GND	Ground
12	GND	Ground
13	+3.3V	+3.3V output.

8. UM6-LT Mechanical Drawing

Top View (Not to Scale)



9. Basic Operation

9.1. Powering and communicating with the UM6

The UM6 requires an input voltage (Vin) ranging from +3.5 V to +5.0V. Supply voltages outside this range are not supported, and high voltages in particular could damage the device. For best results, the supply voltage should be conditioned to remove supply noise.

Communication with the UM6 takes place over a logic level TTL UART at 3.3V, or over a SPI bus. The UART TX and RX pins and the SPI bus pins are +5V tolerant, so +5V systems can be integrated with the UM6 without requiring any external level-shifting hardware. This presumes that the 5V system recognizes 3.3V as above the high logic threshold. In some cases, it may be necessary to add pull-up resistors to the SPI and UART outputs to interface properly with +5.0V systems.

The UM6 can be integrated into other systems either through a side-oriented 4-pin connector, or through two 2mm 12-pin connectors exposed on the bottom of the device. The 4-pin connector provides access to the UART TX and RX pins in addition to Vin and GND. The 2mm 12-pin connectors on the bottom of the UM6 also provide access to the UART pins, in addition to the SPI bus pins, pins for interfacing the unit with an external GPS module, and other pins for custom development. If it is desired to use the SPI bus for communication or to connect a GPS module to the device, the bottom connectors of the UM6 must be used.

The UM6-LT has all pins routed out to a 0.1" header that is easily accessible. This includes pins for SPI communication, UART communication, and GPS integration.

9.2. Sampling, Filtering, and Angle Estimation

9.2.1. EKF Assumptions and Limitations

The UM6 uses a discrete implementation of the EKF, where rate gyro measurements are used to drive the prediction step, and the accelerometer and magnetometer are used for drift correction in the update step. Two major assumptions are made:

1. On average, the accelerometers measure the gravity vector. The EKF knows which way is down by observing the direction of the gravity vector. There are a number of cases where this assumption breaks down, particularly in aircraft. When an aircraft makes a coordinated turn, the measured acceleration will always point in the direction of the bottom of the fuselage. Over a long turn, the pitch and roll angle estimates will therefore tend to zero. To correct this problem, additional sensors must be used for centripetal compensation. The UM6 has no native support for centripetal compensation. However, the firmware is open source, and GPS inputs can be used to add centripetal compensation (this may be a feature in future firmware revisions).

2. Distortion of the magnetic field measurement is predictable. Distortion from objects that rotate and translate with the AHRS can be corrected, but other distortions (from high current wires, ferrous metal objects, etc.) will cause erroneous angle measurements.

If either of the two preceding assumptions do not hold, then angle estimates from the UM6 become less reliable.

All UM6 angle measurements are made with respect to a North-East-Down (NED) inertial frame. The inertial frame x-axis is aligned with magnetic north, the y-axis is aligned with magnetic east, and the z-axis points down toward the center of the Earth. The "pitch" angle represents positive rotation about the y-axis, "roll" represents positive rotation about the x-axis, and "yaw" represents positive rotation about the z-axis.

The sequence of rotations used to give the orientation of the UM6 is first yaw, then pitch, then roll.

The UM6 supports angle estimation using Euler Angles or quaternions. In Euler Angle estimation mode, there is a singularity in pitch angle at +/- 90 degrees. This means that, at 90 degrees, the orientation of the sensor can be represented in more than one way, and the estimation algorithm fails. This is a fundamental limitation of an Euler Angle attitude representation, and can be corrected only by adopting a different representation.

In Euler Angle mode, the UM6 restricts magnetometer updates to yaw only. This can be useful in environments where the external magnetic field varies significantly and it is important to keep accurate pitch and roll angle estimates (i.e. on a flying rotorcraft).

In quaternion estimation mode, a quaternion is used to represent the rotation from the inertial frame to the body frame of the sensor. Quaternions do not suffer from the singularity problem, and can represent any orientation of the sensor. When quaternion estimation is enabled, Euler Angles are still computed after the fact using the estimated attitude quaternion - regardless of the estimation mode being used, Euler Angles can always be retrieved from the device.

In quaternion mode, the UM6 does NOT restrict magnetometer updates to yaw - field distortions can therefore effect all angle estimates, yaw, pitch, and roll. Quaternion estimation mode is enabled by default. This setting can be changed by writing to the UM6_MISC_CONFIG register.

In some cases, it may be necessary to ignore data received by the accelerometers or by the magnetometer. For example, if the UM6 is to be used in an environment where magnetic field readings aren't reliable (ie. indoors or next to an RF transmitter), then magnetometer inputs to the EKF can be disabled. Accelerometer inputs can be similarly disabled, and EKF updates can be turned off entirely if desired. Accelerometer and magnetometer inputs are enabled by default, but they can be disabled by writing to the UM6_MISC_CONFIG register. In general, it should not be necessary to disable the accelerometers or the magnetic sensor, and in many cases, doing so can cause the filter to fail.

9.2.2. EKF tuning

EKF performance can be tuned by adjusting the process noise covariance matrix and the measurement noise covariance matrices (there are two measurement noise matrices - one for the accels, and one for the magnetometer). Generally the measurement noise matrices should remain fixed at the factory default, and the process noise matrix should be adjusted.

Loosely speaking, the process noise matrix is used to specify how much the EKF trusts data from the gyros with respect to data from the magnetic sensors and accelerometers. The lower the values along the diagonal of the matrix, the more the rate gyros are trusted. Conversely, if the diagonal elements are large, the gyros are trusted less and the accels and magnetometers are weighted more heavily.

Trusting gyros more heavily increases the effect of time-varying gyro output biases, essentially increasing DC errors in the angle estimates. Trusting accels and magnetic sensors more heavily increases AHRS sensitivity to vibrations and distorted magnetic field measurements.

Optimal selection of the process noise matrix depends on the specific environment in which the sensor will be operating. If the UM6 will be subjected to large accelerations that interfere with gravity vector measurement, then the diagonal terms of the process noise matrix should be small. On a more stationary platform, on the other hand, the accels can be trusted more. Increase the diagonal terms of the process noise matrix to reduce bias errors.

The process variance can be adjusted by writing to the UM6_EKF_PROCESS_VARIANCE register. This register specifies the value of all the diagonal elements of the process covariance matrix. The off-diagonal elements are assumed to be zero.

The magnetometer and accelerometer measurement variances can be set by writing to the UM6_EKF_MAG_VARIANCE and the UM6_EKF_ACCEL_VARIANCE registers, respectively. These registers specify the values of the diagonal elements of the covariance matrices. The off-diagonal elements are assumed to be zero.

9.3. Calibration

The UM6 comes ready to report angles immediately, but in precision applications it is often beneficial to perform extra calibration to improve accuracy. There are a variety of calibration procedures that can be performed ranging from very simple to more complex. Possible calibration options are:

1. Rate gyro bias calibration
2. Rate gyro scale factor calibration
3. Accelerometer bias calibration
4. Magnetometer soft and hard iron calibration
5. Accelerometer and magnetometer reference vector adjustment
6. Accelerometer and rate gyro cross-axis misalignment correction

Details about each calibration procedure are given below.

9.3.1. Gyro and Accelerometer Calibration

The zero-rate output the rate gyros and the zero-acceleration output of the accelerometers is device-dependent, and the gyro biases vary significantly with temperature. On the UM6, the angular rates used for angle estimation are given by

rate = R_gyro(measured_rate - bias)*

where *R_gyro* is a calibration matrix that scales the measurements and performs axis alignment, *measured_rate* is a vector of raw data returned by the rate gyros, and *bias* is a vector of estimated biases for each rate gyro. While the default biases may be sufficient for the accelerometers, the biases for the rate gyros should generally be reconfigured at run-time.

There are two components of gyro bias calibration - temperature compensation terms, and a constant term used to make slight adjustments at run-time. If the variable *t* represents the measured rate gyro temperature, then the bias is computed as

$$bias = C + (C_0 + C_1 t + C_2 t^2 + C_3 t^3)$$

The term *C* is used to trim the gyro bias compensation at run-time, while the terms C_x are used to compensate for temperature-induced bias changes. By default, the terms C_x are all zero, which means that the only bias compensation performed by the UM6 is based on the value of the trimming term, *C*. The temperature compensation terms can be determined experimentally by the user by logging temperature data and gyro data, and fitting a curve to the measured biases. Alternatively, calibration can be performed for each device at the factory if requested.

The UM6 can automatically measure and set the bias trim term *C*. Automatic bias trim measurement is triggered by sending a ZERO_RATE_GYROS command to the AHRS. During automatic calibration, which takes approximately three seconds, the AHRS should be stationary. The gyro bias trim can also be set manually by writing to the UM6_GYRO_BIAS_XY and UM6_GYRO_BIAS_Z registers.

The UM6 can be configured to automatically calculate bias trim when the sensor starts up by writing to the UM6_MISC_CONFIG register.

The gyro calibration matrix is, by default, diagonal with the diagonal elements corresponding to scale factors that convert raw rates to actual angular rates in degrees per second. The calibration matrix can be changed by writing to the UM6_GYRO_CAL_xy registers, where x corresponds to the matrix row and y corresponds to the matrix column.

The equation describing accelerometer correction is identical to that of the rate gyro equation, but with unique biases and a unique calibration matrix.

9.3.2. Magnetometer Hard and Soft Iron Calibration

Metal and magnetic objects near the UM6 distort magnetometer measurements, creating significant errors in estimated angles. Distortions from objects that are not in a fixed position relative to the UM6 cannot generally be corrected. On the other hand, distortions from objects that are in a fixed position with respect to the sensor can be detected and corrected. For example, if the sensor is mounted to a platform using steel screws, the magnetic field will be distorted; but, since the screws rotate and translate with the sensor, the distortions can be corrected. In contrast, if a screwdriver were placed close to the UM6, then the resulting distortions could not be corrected because the screwdriver does not move with the sensor.

For magnetometer soft and hard iron calibration the common procedure is to mount the UM6 in its final configuration and then perform calibration.

Magnetic field inputs to the EKF are first multiplied by a field correction. By default, the correction matrix scales the sensor measurements so that they are close to unit-norm (ie. no calibration is performed). The UM6 configuration utility can be used to determine the field correction matrix and write it to the sensor.

The magnetometer calibration matrix can also be set manually by writing to the UM6_MAG_CAL_xy registers, where x corresponds to the matrix row and y corresponds to the matrix column.

9.3.3. Gyro Cross-Axis Alignment Calibration

The UM6 measures rotation rates about three axes. Under ideal circumstances, these three axes would be perfectly orthogonal, so that rotation about a single sensor axis would only produce outputs on the rate gyro for that axis. Due to manufacturing tolerances, however, the axes are never perfectly aligned. While cross-axis misalignment is often small enough that it can be ignored, precision applications may require that misalignment be corrected.

Cross-axis misalignment correction can be understood in terms of a change of basis from a non-orthogonal basis formed by the axes of the rate gyros to an orthogonal basis formed by the body frame of the sensor (see Section 9.2 for details about coordinate frames used by the AHRS). We will say that the basis vectors for the body coordinate frame are given by the elementary basis vectors $\{e_1 \quad e_2 \quad e_3\}$. The basis vectors for the non-orthogonal rate gyro axes, expressed in terms of the elementary basis vectors, are given by $\{v_x \quad v_y \quad v_z\}$.

The rotation rates measured by the rate gyros are expressed in terms of the basis $\{v_x \quad v_y \quad v_z\}$. Let the vector **x** is be composed of the *x*, *y*, and *z* rotation rates measured by the rate gyros at any given time. Then the rotation rates expressed in the body frame are given by

$$\mathbf{x}_e = [v_x \quad v_y \quad v_z]\mathbf{x} = T\mathbf{x}$$

This is simply a change of basis operation. By default, *T* is the identity matrix (ie. no cross-axis calibration is performed). If cross-axis calibration is to be performed, *T* must be determined experimentally.

To determine *T* experimentally, the UM6 should be mounted on the center of a turn-table that can rotate the sensor at a constant rate about any body frame axis. The exact rate is not important. It is important, however, that the sensor is mounted so that when the turn-table is spinning, there is only rotation about one body frame axis at a time. The mechanical fixture used to mount the sensor must therefore have tight tolerances.

By rotating the UM6 about one body frame axis and measuring the response of all the rate gyros, it is possible to determine the vectors needed to perform the change of basis. Let $\hat{v}_x$ be the measured response of the x-axis rate gyro to rotation about the body frame x-axis. Let $\hat{v}_y$ and $\hat{v}_z$ be similarly defined. Then the matrix *T* is given by

$$T = [\hat{v}_x \quad \hat{v}_y \quad \hat{v}_z]^{-1}.$$

To prevent unwanted scaling, each column vector in T should be normalized.

There are many ways to perform calibration. The explicit method described in this section is perhaps the most intuitive, but there may be methods that are easier or more accurate depending on what tools are available.

9.3.4. Accel Cross-Axis Alignment Calibration

The accelerometers on the UM6 would ideally be perfectly aligned to the body frame of the device. Much like the onboard gyros, however, manufacturing tolerances cause small misalignments in the accelerometer axes with respect to the body frame. While cross-axis misalignment is often small enough that it can be ignored, precision applications may require that misalignment be corrected.

Cross-axis calibration of the accelerometers is performed in the same fashion as rate gyro cross-axis calibration, except that the AHRS should be aligned with the gravity vector for each axis instead of being rotated about the body frame axes.

9.3.5. Magnetometer and Accelerometer Reference Vectors

The UM6 uses two reference vectors to determine its orientation. The accelerometer reference vector corresponds to the measured gravity vector when pitch and roll angles are zero. The magnetometer reference vector represents the measured magnetic field of the Earth when yaw, pitch, and roll angles are zero. When the accelerometer and magnetometer reference vectors match the output of the sensors, the sensor is oriented with zero yaw, pitch, and roll.

By default, the accelerometer reference vector corresponds to a zero reading on the X and Y accel axes, and the acceleration of gravity on the Z axis. Thus, if the sensor is perfectly flat with respect to the Earth, pitch and roll angles will remain at zero. In some cases, the AHRS will not be mounted perfectly level. The accelerometer reference vector can be reset so that the current orientation of the sensor corresponds to zero pitch and roll. This is accomplished by sending the UM6 a UM6_SET_ACCEL_REF command. When the packet has been received, the sensor will take the current accelerometer readings and use them as the reference vector. The accel reference vector can also be set manually by writing to the configuration register directly.

By default, the magnetometer reference vector is set as the magnetic field measurement when yaw, pitch, and roll angles are zero (ie. the x-axis of the sensor is aligned with magnetic north). Since the magnetic field of the Earth changes depending on location, the magnetometer reference vector should be reset to correspond to its location. This can be accomplished by orienting the sensor so that it is pointed magnetic north (y-axis reading should be zero) and sending an UM6_SET_MAG_REF command to the unit. Alternatively, the reference vector can be set manually by writing to the configuration register directly.

9.4. Writing Settings to Flash

UM6 settings can be written to onboard flash so that they persist when the power is cycled. To write the current configuration to flash, the UM6_FLASH_COMMIT command should be sent to the device either over the UART.

9.5. Restoring Default Settings

The UM6 comes with preconfigured settings that can be changed by the end-user. In some cases, it may be desirable to restore factory settings to the device. This can be accomplished by sending a UM6_RESET_TO_FACTORY packet to the sensor. Note that this command will cause the UM6 to load factory default settings to RAM. To cause these changes to persist when the power is cycled, a UM6_FLASH_COMMIT command should be sent to the device.

Factory configuration and calibration settings are stored in a unique location in FLASH memory. Reprogramming the firmware of the UM6 can erase factory default FLASH settings. In case this happens, the UM6 will load hard-coded factory settings, but these settings will not include calibration that may have been performed at the factory.

9.6. Getting the Firmware Version

The UM6 can be queried to determine the firmware version on the device by sending a UM6_GET_FW_VERSION command to the sensor. The UM6 firmware version is stored as a four-byte sequence of characters. For example, the initial firmware release for the sensor is version "UM1A". Subsequent release versions increment the last letter: "UM1B", "UM1C", etc.

9.7. Automatic Gyro Bias Calculation

The UM6 can automatically measure rate gyro biases when the sensor is stationary. The automatic bias estimation is triggered by sending a UM6_ZERO_GYROS command to the sensor. The UM6 can also be configured to automatically measure gyro biases on startup. This feature can be enabled or disabled by writing to the UM6_MISC_CONFIG register.

9.8. Resetting the Onboard EKF

The onboard EKF can be reset (error covariance re-initialized, states "zeroed") by sending a UM6_RESET_EKF command to the sensor.

9.9. Interfacing the UM6 with External GPS

Only devices with firmware revision UM2B or later support communication with external GPS.

The UM6 and the UM6-LT have a spare UART bus that can be connected directly to any GPS module that communicates using industry standard NMEA strings. The UART pins on both the UM6 and the UM6-LT are 5V tolerant, allowing the unit to interface with GPS modules that communicate using either 3.3V or 5.0V TTL. The UM6 can parse most standard NMEA packet types, including GGA, GLL, GSA, GSV, RMC, and VTG packets.

When a GPS module is connected to the secondary UART on the UM6 or the LT, NMEA packets are automatically received and parsed by the module. No additional configuration is required by the user beyond setting the GPS baudrate and connecting the GPS module. The GPS module baud rate can be set in the UM6_COMMUNICATION register.

GPS position information can be retrieved from the UM6 as raw latitude and longitude in degrees and altitude in meters, and/or as relative position in meters north, east, and above a configurable GPS "home" position. The GPS home position is stored in the GPS_ HOME_LAT, GPS_HOME_LON, and GPS_HOME_ALTITUDE configuration registers. These registers can be set manually or by sending a SET_HOME_POSITION command to the sensor. The SET_HOME_POSITION command takes the most recent position and altitude reported by the GPS module and writes them to the home configuration registers. After writing to the GPS home configuration registers, the changes can be made permanent by executing a WRITE_TO_FLASH command, which will cause the UM6 to store the settings in non-volatile flash.

GPS data retrieved by the UM6 is stored in fourteen different data registers which can be accessed in the same fashion as any other data register (see Table Table 8 - Description of GPS Data Registers). In the communication configuration register (UM6_COMMUNICATION), there are five bits that control which (if any) GPS data is automatically transmitted over the UART when it is received by the UM6. If the POS bit is set, then raw GPS longitude and latitude in degrees and altitude in meters is transmitted. If the REL bit is set, then the relative position and altitude with respect to the GPS home position is transmitted. If the VEL bit is set, then the GPS course and speed (reported by GPS in a NMEA VTG packet) is transmitted. If the SUM bit is set, then the satellite summary register contents are automatically transmitted. Finally, if the SAT bit is set, then all 6 satellite detail registers are transmitted whenever there is new data.

When automatic data transmission is enabled for any GPS data (as described above), data is only automatically transmitted when new GPS data is available. For example, if the UM6 broadcast rate is set to 100 Hz, then every 10 milliseconds, when all other enabled data is being transmitted, the UM6 checks to determine whether new GPS data is available. If it is, and if GPS data transmission is enabled, the data is transmitted. If new data is not available, no additional action is taken.

Table 8 - Description of GPS Data Registers

GPS Data Register	Address	Description
UM6_GPS_LONGITUDE	0x77	Measured GPS longitude in degrees. Stored as a 32-bit IEEE Floating Point.
UM6_GPS_LATITUDE	0x78	Measured GPS latitude in degrees. Stored as a 32-bit IEEE Floating Point.
UM6_GPS_ALTITUDE	0x79	Measured GPS altitude in meters. Stored as a 32-bit IEEE Floating Point.
UM6_GPS_POSITION_N	0x7A	GPS north position in meters, referenced to the GPS home latitude set in the UM6_GPS_HOME_LAT configuration register. Stored as a 32-bit IEEE Floating Point.
UM6_GPS_POSITION_E	0x7B	GPS east position in meters, referenced to the GPS home latitude set in the UM6_GPS_HOME_LAT configuration register. Stored as a 32-bit IEEE Floating Point.
UM6_GPS_POSITION_H	0x7C	GPS altitude in meters, referenced to the GPS home altitude set in UM6_GPS_HOME_ALTITUDE configuration register. Stored as a 32-bit IEEE Floating Point.
UM6_GPS_COURSE_SPEED	0x7D	Course and speed of the GPS unit as reported by a VTG packet.
UM6_GPS_SAT_SUMMARY	0x7E	Summary of satellite information.
UM6_GPS_SAT_1_2	0x7F	Satellite ID and SNR for satellites 1 and 2.
UM6_GPS_SAT_3_4	0x80	Satellite ID and SNR for satellites 3 and 4.
UM6_GPS_SAT_5_6	0x81	Satellite ID and SNR for satellites 5 and 6.
UM6_GPS_SAT_7_8	0x82	Satellite ID and SNR for satellites 7 and 8.
UM6_GPS_SAT_9_10	0x83	Satellite ID and SNR for satellites 9 and 10.
UM6_GPS_SAT_11_12	0x84	Satellite ID and SNR for satellites 11 and 12.

10. Communicating with the UM6

Communication with the UM6 is performed using a 3.3V TTL UART or using a SPI bus. Both communication methods can be used without doing anything specific to enable it. By default, data is transmitted automatically at user-adjustable intervals over the UART. In contrast, the SPI bus only sends data when that data has been requested over the bus (i.e. there is no automatic transmission of data. The UM6 is a SPI bus slave, and only responds to requests from a master device). Both the SPI bus and the UART can be used regardless of whether the other bus is in use - for example, data could be retrieved from the SPI bus even when the UART is transmitting data.

Only UM6 sensors with firmware revision UM2A or later support communication via the SPI bus.

Regardless of whether the UART or the SPI bus is used, data is retrieved from the UM6 by reading from onboard 32-bit registers. Configuration is similarly performed by writing data to the onboard 32-bit configuration registers. Commands can also be sent to the UM6 to perform specific functions (like auto-zeroing the rate gyros, writing configuration settings to flash, etc.)

Registers can be read and written individually or in "batch" operations. In a batch operation, more data is read/written at once to reduce communication overhead for large transfers. In a batch read, the packet address specifies the starting address of the read. For the remainder of the batch, the address of the register to be read increments by one until the batch length is reached. The behavior of a batch write is similar.

The length of the batch is specified in registers, NOT in bytes. For example, a batch length of 4 specifies that 16 bytes will be read/written in the batch (there are 4 bytes in each register, so a batch length of 4 registers results in 4*4 = 16 bytes).

10.1. UART Communication

The UM6 UART operates at a 3.3V logic level with 8 data bits, 1 stop bit, and no parity. While the logic level is 3.3V, the pins are 5V tolerant to simplify integration with 5V systems (ie. no extra external hardware is generally necessary to integrate with 5V systems). By default, the serial baud rate is set at 115200 baud, but the baud rate can be changed by the end user if desired by writing to the UM6_COMMUNICATION register.

UART communication can operate in one of two modes: Broadcast and Listen mode.

In Broadcast Mode, the UM6 will automatically transmit sensor data at a user-configurable rate ranging from 20 Hz to 300 Hz. The broadcast rate can be adjusted by writing to the UM6_COMMUNICATION register.

In Broadcast Mode, any combination of raw or processed sensor data, orientation estimates, and estimator covariance can be transmitted automatically. The data channels that are enabled are called "active channels." By default, only processed rate gyro data, accelerometer data, magnetometer data, and estimated angles are transmitted active. This can be changed by writing to the UM6_COMMUNICATION register. (Note: "Processed" data is sensor data to which calibration has been applied. "Raw" data is the data measured directly by the sensor). Each batch of data is transmitted in its own packet - thus, rate gyro data occupies one packet, accelerometer data occupies another, etc.)

Depending on the UART baud rate, the broadcast rate, and the amount of data being transmitted in each batch of data, the UART can be overloaded with requests to transmit data. This effects not only the ability of the sensor to transmit states at the desired broadcast rate, but it can also affect the sensor's ability to receive packets from a host. It is therefore recommended that when adjusting the broadcast rate, baud rate, and active channel settings, care should be taken to not overload the UART hardware. If more channels are being made active, leave the broadcast rate low to start and gradually increase to ensure that communication is not lost. If the UART becomes overloaded, simply reset the device to restore settings. DO NOT WRITE NEW COMMUNICATION SETTINGS TO FLASH UNTIL IT HAS BEEN VERIFIED THAT THE SENSOR CONTINUES TO COMMUNICATE PROPERLY.

In contrast to Broadcast Mode, Listen Mode causes the UM6 to wait for requests over the UART before transmitting data. A GET_DATA command causes all active channel data to be transmitted (the same data that would be transmitted if Broadcast Mode were enabled). Alternatively, specific data registers can be queried while the device is in Listen Mode. The UM6 is in Broadcast Mode by default. This setting can be changed by writing to the UM6_COMMUNICATION register.

10.1.1. UART Serial Packet Structure

When communication is performed over the UART, data transmitted and received by the UM6 is formatted into packets containing:

1. The three character start sequence 's', 'n', 'p' to indicate the start of a new packet (i.e. **s**tart **n**ew **p**acket)
2. A "packet type" (PT) byte describing the function and length of the packet
3. An address byte indicating the address of the register or command
4. A sequence of data bytes, the length of which is specified in the PT byte
5. A two-byte checksum for error-detection

Table 9 - UART Serial Packet Structure

's'	'n'	'p'	packet type (PT)	Address	Data Bytes (D0...DN-1)	Checksum 1	Checksum 0

All packets sent and received by the UM6 must conform to the format given above.

The PT byte specifies whether the packet is a read or a write operation, whether it is a batch operation, and the length of the batch operation (when required). The PT byte is also used by the UM6 to respond to commands. The specific meaning of each bit in the PT byte is given below.

Table 10 - Packet Type (PT) byte

7	6	5	4	3	2	1	0
Has Data	Is Batch	BL3	BL2	BL1	BL0	RES	CF

Table 11 - Packet Type (PT) Bit Descriptions

Bit(s)	Description
7	Has Data: If the packet contains data, this bit is set (1). If not, this bit is cleared (0).
6	Is Batch: If the packet is a batch operation, this bit is set (1). If not, this bit is cleared (0)
5-2	Batch Length (BL): Four bits specifying the length of the batch operation. Unused if bit 7 is cleared. The maximum batch length is therefore 2^4 = 16
1	Reserved
0	Command Failed (CF): Used by the UM6 to report when a command has failed. Unused for packet sent to the UM6.

The address byte specifies which register will be involved in the operation. During a read operation (Has Data = 0), the address specifies which register to read. During a write operation (Has Data = 1), the address specifies where to place the data contained in the data section of the packet. For a batch read/write operation, the address byte specifies the starting address of the operation.

The "Data Bytes" section of the packet contains data to be written to one or more registers. There is no byte in the packet that explicitly states how many bytes are in this section because it is possible to determine the number of data bytes that should be in the packet by evaluating the PT byte.

If the Has Data bit in the PT byte is cleared (Has Data = 0), then there are no data bytes in the packet and the Checksum immediately follows the address. If, on the other hand, the Has Data bit is set (Has Data = 1) then the number of bytes in the data section depends on the value of the Is Batch and Batch Length portions of the PT byte.

For a batch operation (Is Batch = 1), the length of the packet data section is equal to 4*(Batch Length). Note that the batch length refers to the number of registers in the batch, NOT the number of bytes. Registers are 4 bytes long.

For a non-batch operation (Is Batch = 0), the length of the data section is equal to 4 bytes (one register). The data section lengths and total packet lengths for different PT configurations are shown below.

Table 12 - Packet Length Summary

Has Data	Is Batch	Data Section Length (bytes)	Total Packet Length (bytes)
0	NA	0	7
1	0	4	11
1	1	4*(Batch Length)	7 + 4*(Batch Length)

Note that if a packet is a batch operation, the batch length must be greater than zero.

Read Operations
To initiate a serial read of one or more registers aboard the sensor, a packet should be sent to the UM6 with the "Has Data" bit cleared. This tells the UM6 that this will be a read operation from the address specified in the packet's "Address" byte. If the "Is Batch" bit is set, then the packet will trigger a batch read in which the "Address" byte specifies the address of the first register to be read.

In response to a read packet, the UM6 will send a packet in which the "Has Data" bit is set, and the "Is Batch" and "Batch Length" bits are equivalent to those of the packet that triggered the read operation. The register data will be contained in the "Data Byte" section of the packet.

Write Operations
To initiate a serial write into one or more registers aboard the sensor, a packet should be sent to the UM6 with the "Has Data" bit set. This tells the UM6 that the incoming packet contains data that should be written to the register specified by the packet's "Address" byte. If a batch write operation is to be performed, the "Is Batch" bit should be set, and the "Batch Length" bits should indicate the number of registers that are to be written to.

In response to a write packet, the UM6 will update the contents of the specified register(s) with the contents of the data section of the packet. The UM6 will then transmit a COMMAND_COMPLETE packet to indicate that the write operation succeeded. A COMMAND_COMPLETE packet is a packet with PT = 0 (no data, no batch) and with an address matching the address of the register to which the write operation was made, or the start address of the write operation if this was a batch write.

Note that the COMMAND_COMPLETE packet is equivalent to a packet that would cause the UM6 to initiate a read operation on the address to which data was just written. Since the packet is going from the sensor to the host, however, its meaning is different (it would not make sense for the UM6 to request the contents of one of its registers from an external host).

Command Operations
There are a variety of register address that do not correspond with actual physical registers aboard the UM6. These "command" addresses are used to cause the UM6 to execute specific commands (there are commands for executing calibration operations, resetting the onboard filters, etc. See the UM6 Register Overview in this document for more details).

To initiate a command, simply send a packet to the UM6 with the command's address in the packet "Address" byte. The PT byte should be set to zero for a command operation.

If the UM6 successfully completes the specified command, then a COMMAND_COMPLETE packet is returned with the command address in the "Address" byte of the response packet. If the command fails, the UM6 responds by sending a COMMAND_FAILED packet. The COMMAND_FAILED packet is equivalent to the COMMAND_COMPLETE packet except that the "Command Failed" bit in the PT byte is set (CF = 1).

In some cases, a command will cause specific packets to be sent other than the COMMAND_COMPLETE packet. A UM6_GET_FW_VERSION command will, for example, return a packet containing the version of the firmware installed on the sensor. In this and similar cases, the COMMAND_COMPLETE packet is not sent.

10.2. Example UART Communication

Receiving Data from the UM6
There are a lot of ways to parse the incoming data from the UM6. Often, it is easiest to write a generalized parser that takes all incoming data and extracts the data, address, and packet type information and then makes it easily accessible to the user program. The following code shows an example of a good general parser that can be used to extract packet data.

```
// Structure for holding received packet information
typedef struct UM6_packet_struct
{
    uint8_t Address;
    uint8_t PT;
    uint16_t Checksum;

    uint8_t data_length;
    uint8_t data[30];
} UM6_packet;

// parse_serial_data
// This function parses the data in ‘rx_data’  with length ‘rx_length’ and attempts to find a packet
// in the data.  If a packet is found, the structure ‘packet’ is filled with the packet data.
// If there is not enough data for a full packet in the provided array, parse_serial_data returns 1.
// If there is enough data, but no packet header was found, parse_serial_data returns 2.
// If a packet header was found, but there was insufficient data to parse the whole packet,
// then parse_serial_data returns 3.  This could happen if not all of the serial data has been
// received when parse_serial_data is called.
// If a packet was received, but the checksum was bad, parse_serial_data returns 4.
// If a good packet was received, parse_serial_data fills the UM6_packet structure and returns 0.
uint8_t parse_serial_data( uint8_t* rx_data, uint8_t rx_length, UM6_packet* packet )
{
    uint8_t index;

    // Make sure that the data buffer provided is long enough to contain a full packet
    // The minimum packet length is 7 bytes
    if( rx_length < 7 )
    {
        return 1;
    }

    // Try to find the ‘snp’ start sequence for the packet
    for( index = 0; index < (rx_length – 2); index++ )
    {
        // Check for ‘snp’.  If found, immediately exit the loop
        if( rx_data[index] == ‘s’ && rx_data[index+1] == ‘n’ && rx_data[index+2] == ‘p’ )
        {
            break;
        }
    }

    uint8_t packet_index = index;

    // Check to see if the variable ‘packet_index’ is equal to (rx_length - 2).  If it is, then the above
    // loop executed to completion and never found a packet header.
    if( packet_index == (rx_length – 2) )
    {
        return 2;
    }

    // If we get here, a packet header was found.  Now check to see if we have enough room
    // left in the buffer to contain a full packet.  Note that at this point, the variable ‘packet_index’
    // contains the location of the ‘s’ character in the buffer (the first byte in the header)
    if( (rx_length – packet_index) < 7 )
    {
        return 3;
    }

    // We’ve found a packet header, and there is enough space left in the buffer for at least
    // the smallest allowable packet length (7 bytes).  Pull out the packet type byte to determine
    // the actual length of this packet
    uint8_t PT = rx_data[packet_index + 3];

    // Do some bit-level manipulation to determine if the packet contains data and if it is a batch
    // We have to do this because the individual bits in the PT byte specify the contents of the
    // packet.
    uint8_t packet_has_data = (PT >> 7) & 0x01;     // Check bit 7 (HAS_DATA)
    uint8_t packet_is_batch = (PT >> 6) & 0x01;     // Check bit 6 (IS_BATCH)
    uint8_t batch_length = (PT >> 2) & 0x0F;        // Extract the batch length (bits 2 through 5)
```

```
// Now finally figure out the actual packet length
uint8_t data_length = 0;
if( packet_has_data )
{
    if( packet_is_batch )
    {
        // Packet has data and is a batch.  This means it contains 'batch_length' registers, each
        // of which has a length of 4 bytes
        data_length = 4*batch_length;
    }
    else    // Packet has data but is not a batch.  This means it contains one register (4 bytes)
    {
        data_length = 4;
    }
}
else    // Packet has no data
{
    data_length = 0;
}

// At this point, we know exactly how long the packet is.  Now we can check to make sure
// we have enough data for the full packet.
if( (rx_length – packet_index) < (data_length + 5) )
{
    return 3;
}

// If we get here, we know that we have a full packet in the buffer.  All that remains is to pull
// out the data and make sure the checksum is good.
// Start by extracting all the data
packet->Address = rx_data[packet_index + 4];
packet->PT = PT;

// Get the data bytes and compute the checksum all in one step
packet->data_length = data_length;
uint16_t computed_checksum = 's' + 'n' + 'p' + packet_data->PT + packet_data->Address;
for( index = 0; index < data_length; index++ )
{
    // Copy the data into the packet structure's data array
    packet->data[index] = rx_data[packet_index + 5 + index];
    // Add the new byte to the checksum
    computed_checksum += packet->data[index];
}

// Now see if our computed checksum matches the received checksum
// First extract the checksum from the packet
uint16_t received_checksum = (rx_data[packet_index + 5 + data_length] << 8);
received_checksum |= rx_data[packet_index + 6 + data_length];

// Now check to see if they don't match
if( received_checksum != computed_checksum )
{
    return 4;
    }

    // At this point, we've received a full packet with a good checksum.  It is already
    // fully parsed and copied to the 'packet' structure, so return 0 to indicate that a packet was
    // processed.
    return 0;
}
```

Once the packet has been parsed and copied into the packet structure, accessing the desired data is as simple as watching for packets with the desired address, checking the data length to make sure it is as long as you expect, and then pulling the data out of the packet's data array.

Getting the Firmware Revision

To read the firmware revision from the UM6, a GET_FW_REVISION command should be sent to the Hydra over the serial port. Recall that to initiate a command on the UM6, a read packet should be sent to the Hydra using the command's address in the 'Address' byte of the packet. For a GET_FW_REVISION command, the address is 170 (0xAA).

C-code for constructing and sending the command is shown below:

```
uint8_t tx_data[20];

tx_data[0] = 's';
tx_data[1] = 'n';
tx_data[2] = 'p';
tx_data[3] = 0x00;        // Packet Type byte
tx_data[4] = 0xAA;        // Address of GET_FW_REVISION register
tx_data[5] = 0x01;        // Checksum high byte
tx_data[6] = 0xFB;        // Checksum low byte

USART_transmit( tx_data, 7 );
```

The preceding code assumes that a function called USART_transmit(uint8_t* data, uint8_t length) exists that transmits 'length' characters from the provided buffer over the UART.

Once the UM6 receives the above packet, it will respond with a packet containing the firmware revision. Example code for receiving the firmware revision packet is given below. Note that this code assumes that the serial data is being received and transferred to a buffer before the example code is executed.

```
UM6_packet new_packet;
char FW_revision[5];

// Call the parse_serial_data function to handle the incoming serial data.  The serial data should
// be placed in 'rx_data' and the length in 'rx_data_lenbgth' before this function is called.
if( !parse_serial_data( rx_data, rx_data_length, &new_packet )
{
    // We got a good packet!  Check to see if it is the firmware revision
    if( packet.Address == 0xAA )
    {
        // Extract the firmware revision
        FW_revision[0] = packet.data[0];
        FW_revision[1] = packet.data[1];
        FW_revision[2] = packet.data[2];
        FW_revision[3] = packet.data[3];
        FW_revision[4] = '\0';     // Null-terminate the FW revision so we can use it like a string

        // Print the firmware revision to the terminal (or do whatever else you want…)
        printf("Got the firmware revision: %s\r\n", FW_revision);
    }



    // TODO: Check to see if any of the other packets that we care about have been found.
    // If so, do stuff with them.

}
```

Note that it is not always sufficient to simply check the address of the data register that you want to read. In most cases, data automatically transmitted by the UM6 is sent in batch operations to improve efficiency. For example, when processed rate gyro is transmitted, it is sent in one batch packet containing both registers 0x5C (Gyros X and Y) and 0x5D (Gyro Z). Thus, the address of the packet is 0x5C, but it is a batch packet with batch length 2.

10.3. SPI Bus Communication

Communicating with the UM6 via the SPI bus allows for much higher data rates and lower latency than would be available through the UART.

The UM6 SPI bus operates at a +3.3V logic level. The UM6 is a slave on the bus, remaining inactive unless queried by a master device. Since the SPI bus will not always be used, bus inputs (MOSI, SCK, SS) are pulled to +3.3V internally. This prevents noise from being registered by the UM6 as attempts to communicate with the sensor. While all SPI pins on the UM6 are +5V tolerant, interfacing with +5V devices may require level-shifting hardware to allow the logic high output of the UM6 to register properly.

The UM6 SPI clock (SCK) is active high, with data clocked in on the first rising edge (usually labeled SPI Mode 0 on microcontrollers and other devices). The maximum clock rate is 400kHz. The master should place its data on the MOSI line on the clock falling edge.

All SPI operations begin when the master writes two control bytes to the bus. The first byte indicates whether the operation is a register read (0x00) or a write (0x01). The second byte is the address of the register being accessed.

A read operation is performed by writing the control byte 0x00 to the MOSI line, followed by the address of the register to be read. During the next four transfers, the UM6 will write the contents of the register to the MISO line starting with the most-significant byte in the register as shown in Figure 2 - Single Register Read Operation. The master should pull the MOSI line low during the remainder of the read.

A read operation can be extended to read more than one register at a time as shown in Figure 3 - Multiple Register Read Operation To initiate the batch read, the master should write the address of the next desired register to the MOSI line while last byte of the previous register is being transmitted by the UM6.

A write operation is performed by writing the control byte 0x01 to the MOSI line, followed by the address of the register to modify. During the next four transfers, the UM6 will read the data from the MOSI line and write it to the specified register. During a write operation, the UM6 will pull the MISO line low to indicate that it is receiving data. There is no batch write operation. The structure of a write operation is illustrated in Figure 4 - Single Register Write Operation.

Commands are initiated over the SPI bus by sending a **write** operation to the command address, with all four data bytes at zero. The UM6 will not report that a command was executed over the SPI bus except for during a GET_FW_REVISION command, which causes the firmware revision to be sent over the MISO line during the next four byte transfers on the bus.

For example, to execute a zero rate gyros command over the SPI bus, the following data bytes should be sent over the bus: 0x01 0xAC 0x00 0x00 0x00 0x00.

Similarly, to query the UM6 to get the firmware revision over the SPI bus, the following sequence should be used: 0x01 0xAA 0x00 0x00 0x00 0x00. The UM6 will send the firmware revision over the MISO line during the last four byte transfers.

Figure 1 - SPI Bus Timing

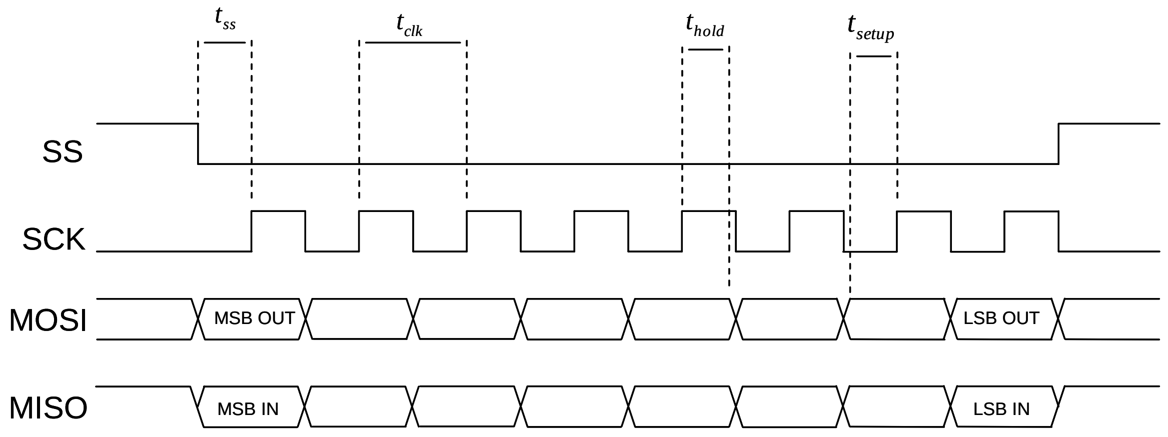


Table 13 - SPI Bus Timing

Name	Description	Min	Max
t_{ss}	Slave-select setup time	1 us	NA
t_{clk}	Clock period	2 us	5 us
f_{clk}	Clock frequency	200 kHz	500 kHz

t_{hold}	Data hold time	10 ns	NA
t_{setup}	Data setup time	10 ns	NA

Figure 2 - Single Register Read Operation

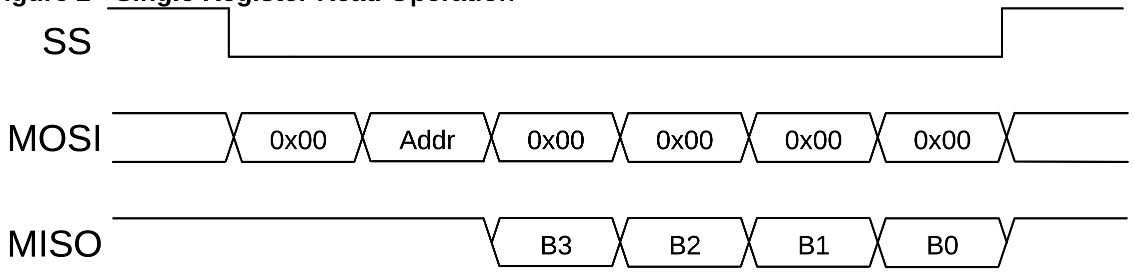


Figure 3 - Multiple Register Read Operation

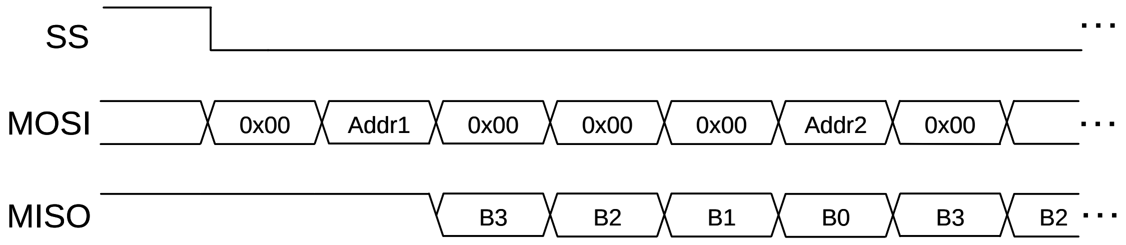
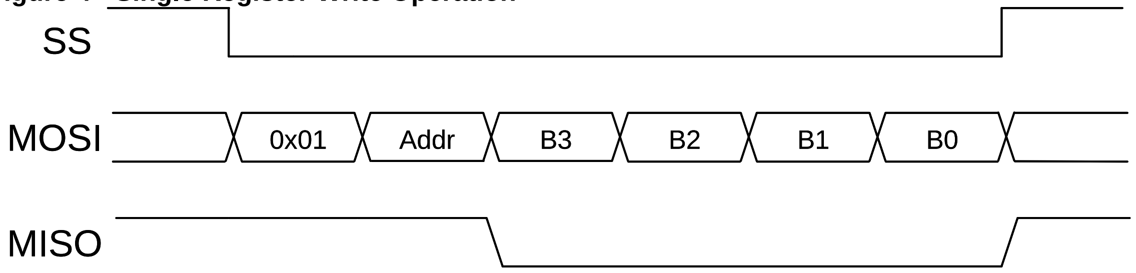


Figure 4 - Single Register Write Operation



11. UM6 Register Overview

There are three types of registers onboard the UM6: configuration registers, data registers, and command registers.

Configuration registers begin at address 0x0 and are used to configure UM6 operation. Configuration register contents can be written to onboard flash to allow settings to be maintained when the device is powered down.

Data registers begin at address 0x55, and store raw and processed data from the sensors along with estimated states. Unlike configuration registers, data register contents cannot be written to flash.

Command registers technically aren't registers at all, but they provide a convenient way to send commands to the UM6 when those commands do not require additional data beyond the command itself. For example, a command to run an onboard gyro bias calibration routine is triggered by querying the UM6_ZERO_GYROS command register. By using a unique register address for each command, the same communication architecture used to read from and write to data and configuration registers can be used to send commands to the UM6. Command registers begin at address 0xAA.

11.1. UM6 Configuration Registers

UM6 configuration registers are used to adjust the performance of the UM6 sensor. Configuration register contents can be written to FLASH by sending a UM6_FLASH_COMMIT command to the sensor.

Table 14 - UM6 Configuration Register Summary

Address	Register Name	Description
0x00	UM6_COMMUNICATION	Communication configuration settings
0x01	UM6_MISC_CONFIG	Misc. sensor configuration settings
0x02	UM6_MAG_REF_X	x-component of the magnetic field reference vector. Stored as a 32-bit IEEE floating point value.
0x03	UM6_MAG_REF_Y	y-component of the magnetic field reference vector. Stored as a 32-bit IEEE floating point value.
0x04	UM6_MAG_REF_Z	z-component of the magnetic field reference vector. Stored as a 32-bit IEEE floating point value.
0x05	UM6_ACCEL_REF_X	x-component of the accelerometer reference vector. Stored as a 32-bit IEEE floating point value.
0x06	UM6_ACCEL_REF_Y	y-component of the accelerometer reference vector. Stored as a 32-bit IEEE floating point value.
0x07	UM6_ACCEL_REF_Z	z-component of the accelerometer reference vector. Stored as a 32-bit IEEE floating point value.
0x08	UM6_EKF_MAG_VARIANCE	Variance used during the magnetometer update step in the EKF. Stored as a 32-bit IEEE floating point value.
0x09	UM6_EKF_ACCEL_VARIANCE	Variance used during the accelerometer update step in the EKF. Stored as a 32-bit IEEE floating point value.
0x0A	UM6_EKF_PROCESS_VARIANCE	Variance used during the prediction step in the EKF. Stored as a 32-bit IEEE floating point value.
0X0B	UM6_GYRO_BIAS_XY	Gyro bias on the x and y axes. Stored as 16-bit signed integers.
0x0C	UM6_GYRO_BIAS_Z	Gyro bias on the z axis. Stored as 16-bit signed integer.
0x0D	UM6_ACCEL_BIAS_XY	Accelerometer bias on the x and y axes. Stored as 16-bit signed integers.

0x0E	UM6_ACCEL_BIAS_Z	Accelerometer bias on the z axis. Stored as 16-bit signed integer.
0x0F	UM6_MAG_BIAS_XY	Magnetometer bias on the x and y axes. Stored as 16-bit signed integers.
0x10	UM6_MAG_BIAS_Z	Magnetometer bias on the z axis. Stored as 16-bit signed integer.
0x11	UM6_ACCEL_CAL_00	Element in row 0, column 0 of the accelerometer calibration matrix
0x12	UM6_ACCEL_CAL_01	Element in row 0, column 1 of accelerometer calibration matrix
0x13	UM6_ACCEL_ CAL _02	Element in row 0, column 2 of accelerometer calibration matrix
0x14	UM6_ACCEL_ CAL _10	Element in row 1, column 0 of accelerometer calibration matrix.
0x15	UM6_ACCEL_ CAL _11	Element in row 1, column 1 of accelerometer calibration matrix
0x16	UM6_ACCEL_ CAL _12	Element in row 1, column 2 of accelerometer calibration matrix
0x17	UM6_ACCEL_ CAL _20	Element in row 2, column 0 of accelerometer calibration matrix
0x18	UM6_ACCEL_ CAL _21	Element in row 2, column 1 of accelerometer calibration matrix
0x19	UM6_ACCEL_ CAL _22	Element in row 2, column 2 of accelerometer calibration matrix
0x1A	UM6_GYRO_ CAL _00	Element in row 0, column 0 of rate gyro calibration matrix
0x1B	UM6_GYRO_ CAL _01	Element in row 0, column 1 of rate gyro calibration matrix
0x1C	UM6_GYRO_ CAL _02	Element in row 0, column 2 of rate gyro calibration matrix
0x1D	UM6_GYRO_ CAL _10	Element in row 1, column 0 of rate gyro calibration matrix
0x1E	UM6_GYRO_ CAL _11	Element in row 1, column 1 of rate gyro calibration matrix
0x1F	UM6_GYRO_ CAL _12	Element in row 1, column 2 of rate gyro calibration matrix
0x20	UM6_GYRO_ CAL _20	Element in row 2, column 0 of rate gyro calibration matrix
0x21	UM6_GYRO_ CAL _21	Element in row 2, column 1 of rate gyro calibration matrix
0x22	UM6_GYRO_ CAL _22	Element in row 2, column 2 of rate gyro calibration matrix
0x23	UM6_MAG_CAL_00	Element in row 0, column 0 of magnetometer calibration matrix
0x24	UM6_MAG_CAL_01	Element in row 0, column 1 of magnetometer calibration matrix
0x25	UM6_MAG_CAL_02	Element in row 0, column 2 of magnetometer calibration matrix
0x26	UM6_MAG_CAL_10	Element in row 1, column 0 of magnetometer calibration matrix

0x27	UM6_MAG_CAL_11	Element in row 1, column 1 of magnetometer calibration matrix
0x28	UM6_MAG_CAL_12	Element in row 1, column 2 of magnetometer calibration matrix
0x29	UM6_MAG_CAL_20	Element in row 2, column 0 of magnetometer calibration matrix
0x2A	UM6_MAG_CAL_21	Element in row 2, column 1 of magnetometer calibration matrix
0x2B	UM6_MAG_CAL_22	Element in row 2, column 2 of magnetometer calibration matrix
0x2C	UM6_GYROX_BIAS_0	Coefficient C0 used in x-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x2D	UM6_GYROX_BIAS_1	Coefficient C1 used in x-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x2E	UM6_GYROX_BIAS_2	Coefficient C2 used in x-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x2F	UM6_GYROX_BIAS_3	Coefficient C3 used in x-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x30	UM6_GYROY_BIAS_0	Coefficient C0 used in y-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x31	UM6_GYROY_BIAS_1	Coefficient C1 used in y-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x32	UM6_GYROY_BIAS_2	Coefficient C2 used in y-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x33	UM6_GYROY_BIAS_3	Coefficient C3 used in y-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x34	UM6_GYROZ_BIAS_0	Coefficient C0 used in z-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x35	UM6_GYROZ_BIAS_1	Coefficient C1 used in z-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x36	UM6_GYROZ_BIAS_2	Coefficient C2 used in z-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x37	UM6_GYROZ_BIAS_3	Coefficient C3 used in z-axis gyro temperature compensation. This register is only available in firmware revisions UM2A and later.
0x38	UM6_GPS_HOME_LAT	Latitude in degrees used as the home position for computing relative position in meters. Stored as a 32-bit IEEE Floating Point. This register is only available in firmware revisions UM2B and later.
0x39	UM6_GPS_HOME_LON	Longitude in degrees used as the home position

		for computing relative position in meters. Stored as a 32-bit IEEE Floating Point. This register is only available in firmware revisions UM2B and later.
0x40	UM6_GPS_HOME_ALTITUDE	Altitude in meters used as the home position for computing relative position in meters. Stored as a 32-bit IEEE Floating Point. This register is only available in firmware revisions UM2B and later.

11.1.1. UM6_COMMUNICATION Register (0x00)

Description

Specifies communication settings on the UM6.

Table 15 - Communication Register Definition

B3								B2							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RES	BEN	GR	AR	MR	GP	AP	MP	QT	EU	COV	TMP	POS	REL	VEL	SUM

B1								B0							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SAT	RES	*GPS_baud_rate*			*baud_rate*			*broadcast_rate*							

Table 16 - Communication Register Bit Definitions

BITS	NAME	DESCRIPTION
0-7	*broadcast_rate*	When in broadcast mode, these bits specify how often a data packets are automatically transmitted over the serial port. The actual broadcast frequency is given by $freq = (280/255)*broadcast\_rate + 20$
8-10	*baud_rate*	Used to set sensor serial port baud rate. The three configuration bits set the desired baud rate as shown below: 000 -> 9600 Baud 001 -> 14400 Baud 010 -> 19200 Baud 011 -> 38400 Baud 100 -> 57600 Baud 101 -> 115200 Baud 110 -> NA 111 -> NA Note that if the desired baud rate is changed, the new configuration must be written to FLASH in order for the new baud rate to persist when power is cycled. A write to FLASH must be triggered by

		sending a WRITE_TO_FLASH command with the currently selected baud rate. This ensures that the baud rate is never mistakenly changed permanently.
11-13	*GPS_baud_rate*	Used to set the baud rate used by the connected GPS receiver (if used). The three configuration bits set the desired baud rate as shown below: 000 -> 9600 Baud 001 -> 14400 Baud 010 -> 19200 Baud 011 -> 38400 Baud 100 -> 57600 Baud 101 -> 115200 Baud 110 -> NA 111 -> NA Note that if the desired baud rate is changed, the new configuration must be written to FLASH in order for the new baud rate to persist when power is cycled. A write to FLASH must be triggered by sending a WRITE_TO_FLASH command with the currently selected baud rate. This ensures that the baud rate is never mistakenly changed permanently.
14	RES	Reserved. These bits are ignored.
15	SAT	Detailed satellite status information enabled. This is available in firmware revisions UM2B and later.
16	SUM	Satellite summary transmission enabled. This is available in firmware revisions UM2B and later.
16	VEL	GPS course and velocity transmission enabled. This is available in firmware revisions UM2B and later.
18	REL	GPS relative position transmission enabled. This is available in firmware revisions UM2B and later.
19	POS	GPS position transmission enabled. This is available in firmware revisions UM2B and later.
20	TEMP	Temperature measurement transmission enabled. This is only available in firmware revision UM2A and later.
21	COV	Covariance matrix transmission enabled
22	EU	Euler angle transmission enabled
23	QT	Quaternion transmission enabled (only outputs valid data if quaternion estimation is enabled in the MISC_CONFIG register)
24	MP	Processed magnetometer data transmission enabled
25	AP	Processed accelerometer data transmission enabled
26	GP	Processed gyro data transmission enabled
27	MR	Raw magnetometer data transmission enabled
28	AR	Raw accelerometer data transmission enabled
29	GR	Raw gyro data transmission enabled
30	BEN	Broadcast mode enabled
31	RES	Bit reserved

11.1.2. UM6_MISC_CONFIG Register (0x01)

Description

Miscellaneous configuration options.

Table 17 - Misc. Settings Register Definition

B3								**B2**							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MUE	AUE	CAL	QUAT	PPS	RES										

B1								**B0**							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES															

Table 18 - Misc Settings Bit Definitions

BITS	**NAME**	**DESCRIPTION**
0-26	RES	Reserved. The contents of these bits are ignored.
27	PPS	Specifies whether PPS timing is enabled. This will be implemented in future firmware revisions.
28	QUAT	Specifies whether quaternion state estimation is enabled.
29	CAL	Enables startup gyro calibration. If this bit is set, then gyros will be calibrated automatically on sensor startup.
30	AUE	EKF accelerometer updates enabled (pitch and roll angle correction)
31	MUE	EKF magnetometer updates enabled (yaw angle correction)

11.1.3. UM6_MAG_REF_X (0x02)

Description

X-component of the magnetic field reference vector. This register stores a 32-bit IEEE floating point value.

Table 19 - Mag Reference X Register Definition

B3	**B2**	**B1**	**B0**
32-bit IEEE floating point value			

11.1.4. UM6_MAG_REF_Y (0x03)

Description

Y-component of the magnetic field reference vector. This register stores a 32-bit IEEE floating point value.

Table 20 - Mag Reference Y Register Definition

B3	B2	B1	B0
32-bit IEEE floating point value			

11.1.5. UM6_MAG_REF_Z (0x04)

Description

Z-component of the magnetic field reference vector. This register stores a 32-bit IEEE floating point value.

Table 21 - Mag Reference Z Register Definition

B3	B2	B1	B0
32-bit IEEE floating point value			

11.1.6. UM6_ACCEL_REF_X (0x05)

Description

X-component of the accelerometer reference vector. This register stores a 32-bit IEEE floating point value.

Table 22 - Accel Reference X Register Definition

B3	B2	B1	B0
32-bit IEEE floating point value			

11.1.7. UM6_ACCEL_REF_Y (0x06)

Description

Y-component of the accelerometer reference vector. This register stores a 32-bit IEEE floating point value.

Table 23 - Accel Reference Y Register Definition

B3	B2	B1	B0
32-bit IEEE floating point value			

11.1.8. UM6_ACCEL_REF_Z (0x07)

Description

Z-component of the accelerometer reference vector. This register stores a 32-bit IEEE floating point value.

Table 24 - Accel Reference Z Register Definition

B3	B2	B1	B0
32-bit IEEE floating point value			

11.1.9. UM6_EKF_MAG_VARIANCE (0x08)

Description

Variance of magnetometer noise. This value is used by the EKF during the magnetometer update step to compute the Kalman Gain and to propagate the error covariance. This value typically needn't be modified by the end user.

Table 25 - EKF Mag Variance Register Definition

B3	B2	B1	B0
32-bit IEEE floating point value			

11.1.10. UM6_EKF_ACCEL_VARIANCE (0x09)

Description

Variance of accelerometer noise. This value is used by the EKF during the accelerometer update step to compute the Kalman Gain and to propagate the error covariance. This value typically needn't be modified by the end user.

Table 26 - EKF Accel Variance Register Definition

B3	B2	B1	B0
32-bit IEEE floating point value			

11.1.11. UM6_EKF_PROCESS_VARIANCE (0x0A)

Description

Variance of process noise. This value is used by the EKF during the predictions to propagate the error covariance. This value can be tuned to adjust the performance of the filter. A high process variance causes the rate gyros to be trusted more, while a low process variance causes the magnetic and acceleration sensors to be trusted more.

Table 27 - EKF Process Variance Register Definition

B3	B2	B1	B0
32-bit IEEE floating point value			

11.1.12. UM6_GYRO_BIAS_XY (0x0B)

Description

Stores the values used to compensate for rate gyro bias on the X and Y gyro axes. The bias values are stored as 16-bit 2's complement signed integers.

Table 28 - Gyro Bias X/Y Register Definition

B3	B2	B1	B0
X gyro bias (16-bit, little-endian, 2's complement)		Y gyro bias (16-bit, little-endian, 2's complement)	

11.1.13. UM6_GYRO_BIAS_Z (0x0C)

Description

Stores the values used to compensate for rate gyro bias on the rate gyro Z axis. The bias value is stored as a 16-bit 2's complement signed integer.

Table 29 - Gyro Bias Z Register Definition

B3	B2	B1	B0
Z gyro bias (16-bit, little-endian, 2's complement)		Unused	

11.1.14. UM6_ACCEL_BIAS_XY (0x0D)

Description

Stores the values used to compensate for bias on the X and Y accelerometer axes. The bias values are stored as 16-bit 2's complement signed integers.

Table 30 - Accel Bias X/Y Register Definition

B3	B2	B1	B0
X accel bias (16-bit, little-endian, 2's complement)		Y accel bias (16-bit, little-endian, 2's complement)	

11.1.15. UM6_ACCEL_BIAS_Z (0x0E)

Description

Stores the values used to compensate for bias on the accelerometer Z axis. The bias value is stored as a 16-bit 2's complement signed integer.

Table 31 - Accel Bias Z Register Definition

B3	B2	B1	B0
Z accel bias (16-bit, little-endian, 2's complement)		Unused	

11.1.16. UM6_MAG_BIAS_XY (0x0F)

Description

Stores the values used to compensate for bias on the X and Y magnetometer axes. The bias values are stored as 16-bit 2's complement signed integers.

Table 32 - Mag Bias X/Y Register Definition

B3	B2	B1	B0
X mag bias (16-bit, little-endian, 2's complement)		Y mag bias (16-bit, little-endian, 2's complement)	

11.1.17. UM6_MAG_BIAS_Z (0x10)

Description

Stores the values used to compensate for bias on the magnetometer Z axis. The bias value is stored as a 16-bit 2's complement signed integer.

Table 33 - Mag Bias Z Register Definition

B3	B2	B1	B0
Z mag bias (16-bit, little-endian, 2's complement)		Unused	

11.1.18. UM6_ACCEL_CALxx (0x11 - 0x19)

Description

A set of 9 registers storing the accelerometer calibration matrix. This matrix is used to correct cross-axis misalignment of the accelerometer axes and to scale raw data to correct acceleration values. By default, this matrix is a diagonal matrix with scale factors for each diagonal entry (ie. no cross-axis calibration is performed).

The matrix indices are stored in row-major order. Addresses 0x17 through 0x19 store the first row, 0x1A-0x1C the second, and 0x1D-0x1F the third.

Table 34 - Accel Calibration Matrix Register Definition

B3	B2	B1	B0
UM6_ACCEL_ALIGNxx (32-bit IEEE floating point value)			

11.1.19. UM6_GYRO_CALxx (0x1A - 0x22)

Description

A set of 9 registers storing the rate gyro calibration matrix. This matrix is used to correct cross-axis misalignment of the rate gyro axes, and to scale raw data to the correct angular rates. By default, this matrix is the a diagonal matrix with default scale factors in each diagonal entry (ie. no gyro axis alignment is performed).

The matrix indices are stored in row-major order. Addresses 0x20 through 0x22 store the first row, 0x23-0x25 the second, and 0x26-0x28 the third.

Table 35 - Gyro Calibration Matrix Register Definition

B3	B2	B1	B0
UM6_GYRO_ALIGNxx (32-bit IEEE floating point value)			

11.1.20. UM6_MAG_CALxx (0x23 - 0x2B)

Description

A set of 9 registers storing the magnetometer calibration matrix. This matrix is used to correct soft-iron distortions of the measured magnetic field, in addition to correcting cross-axis misalignment. By default, this is a diagonal matrix with scale factors for each axis that should reduce the measured field vector to a unit-norm vector. In general, the magnetometer calibration matrix should be determined experimentally and set to minimize errors from hard and soft iron distortions.

The matrix indices are stored in row-major order. Addresses 0x29 through 0x2B store the first row, 0x2C-0x2E the second, and 0x2F-0x31 the third.

Table 36 - Magnetometer Calibration Matrix Register Definition

B3	B2	B1	B0
UM6_MAG_CALxx (32-bit IEEE floating point value)			

11.1.21. UM6_GYROX_BIASx (0x2C - 0x2F)

Description

A set of four registers storing coefficients used for temperature compensation of the x-axis rate gyro. The register UM6_GYROX_BIAS0 corresponds to the term *C0*, UM6_GYROX_BIAS1 corresponds to the term *C1*, etc.

Each register stores a 32-bit IEEE floating point value. The rate gyro bias is computed as

$$bias = C + (C_0 + C_1 t + C_2 t^2 + C_3 t^3)$$

where the term *C* is used for trimming the bias value at run-time. By default, the temperature compensation coefficients are all zero. Factory temperature compensation can be requested, or the coefficients can be determined experimentally by the end-user.

These registers are only available in firmware revision UM2A and later.

Table 37 - Gyro X Bias Temperature Calibration Register Definition

B3	B2	B1	B0

UM6_GYROX_BIASx (32-bit IEEE floating point value)

11.1.22. UM6_GYROY_BIASx (0x30 - 0x33)

Description

A set of four registers storing coefficients used for temperature compensation of the y-axis rate gyro. The register UM6_GYROXY_BIAS0 corresponds to the term *C0*, UM6_GYROY_BIAS1 corresponds to the term *C1*, etc.

Each register stores a 32-bit IEEE floating point value. The rate gyro bias is computed as

$$bias = C + (C_0 + C_1 t \ \ + C_2 t^2 + C_3 t^3)$$

where the term *C* is used for trimming the bias value at run-time. By default, the temperature compensation coefficients are all zero. Factory temperature compensation can be requested, or the coefficients can be determined experimentally by the end-user.

These registers are only available in firmware revision UM2A and later.

Table 38 - Gyro Y Bias Temperature Calibration Register Definition

B3	B2	B1	B0
UM6_GYROY_BIASx (32-bit IEEE floating point value)			

11.1.23. UM6_GYROZ_BIASx (0x34 - 0x37)

Description

A set of four registers storing coefficients used for temperature compensation of the z-axis rate gyro. The register UM6_GYROZ_BIAS0 corresponds to the term *C0*, UM6_GYROZ_BIAS1 corresponds to the term *C1*, etc.

Each register stores a 32-bit IEEE floating point value. The rate gyro bias is computed as

$$bias = C + (C_0 + C_1 t \ \ + C_2 t^2 + C_3 t^3)$$

where the term *C* is used for trimming the bias value at run-time. By default, the temperature compensation coefficients are all zero. Factory temperature compensation can be requested, or the coefficients can be determined experimentally by the end-user.

These registers are only available in firmware revision UM2A and later.

Table 39 - Gyro Z Temperature Compensation Register Definition

B3	B2	B1	B0
UM6_GYROX_BIASx (32-bit IEEE floating point value)			

11.2. UM6 Data Registers

11.2.1. UM6 Data Register Overview

Table 40 - Summary of UM6 Data Registers

Address	Register Name	Description
0x55	UM6_STATUS	UM6 Status Register. Stores results of self-test operations, and indicates when internal errors have occurred.
0x56	UM6_GYRO_RAW_XY	Stores the most recently acquired raw data from the X and Y axis rate gyros. No bias compensation or scaling is performed. Data is stored as 16-bit 2's complement integers.
0x57	UM6_GYRO_RAW_Z	Stores the most recently acquired raw data from the Z axis rate gyro. No bias compensation or scaling is performed. Data is stored as a 16-bit 2's complement integer.
0x58	UM6_ACCEL_RAW_XY	Stores the most recently acquired raw data from the X and Y axis accelerometers. No bias compensation or scaling is performed. Data is stored as 16-bit 2's complement integers.
0x59	UM6_ACCEL_RAW_Z	Stores the most recently acquired raw data from the Z axis accelerometer. No bias compensation or scaling is performed. Data is stored as a 16-bit 2's complement integer.
0x5A	UM6_MAG_RAW_XY	Stores the most recently acquired raw data from the X and Y axis magnetic sensors. No bias compensation or scaling is performed. Data is stored as 16-bit 2's complement integers.
0x5B	UM6_MAG_RAW_Z	Stores the most recently acquired raw data from the Z axis magnetometer. No bias compensation or scaling is performed. Data is stored as a 16-bit 2's complement integer.
0x5C	UM6_GYRO_PROC_XY	Stores the body-frame x and y-axis angular rates in degrees per second. Computed using raw gyro data, bias compensation, scaling, and cross-axis alignment. Stored as 16-bit signed integers to be scaled to obtain the actual values.
0x5D	UM6_GYRO_PROC_Z	Stores the body-frame z-axis angular rate in degrees per second. Computed using raw gyro data, bias compensation, scaling, and cross-axis alignment. Stored as 16-bit signed integer to be scaled to obtain the actual value.
0x5E	UM6_ACCEL_PROC_XY	Stores the body-frame x and y-axis acceleration. Computed using raw accel data, bias compensation, scaling, and cross-axis alignment. Stored as 16-bit

		signed integers to be scaled to obtain the actual values.
0x5F	UM6_ACCEL_PROC_Z	Stores the body-frame z-axis acceleration. Computed using raw accel data, bias compensation, scaling, and cross-axis alignment. Stored as 16-bit signed integer to be scaled to obtain the actual value.
0x60	UM6_MAG_PROC_XY	Stores the body-frame x and y-axis magnetic field components. Computed using raw magnetometer data, bias compensation, scaling, and soft-iron calibration. Stored as 16-bit signed integers to be scaled to obtain the actual values.
0x61	UM6_MAG_PROC_Z	Stores the body-frame z-axis magnetic field component. Computed using raw magnetometer data, bias compensation, scaling, and soft-iron calibration. Stored as 16-bit signed integer to be scaled to obtain the actual value.
0x62	UM6_EULER_PHI_THETA	Stores the estimated roll and pitch angles. Each element is stored as a 16-bit signed integer to be scaled to obtain the actual angle.
0x63	UM6_EULER_PSI	Stores the estimated pitch angle. This is a 16-bit signed integer to be scaled to obtain the actual angle.
0x64	UM6_QUAT_AB	Stores the first two components of the estimated quaternion. Each element is stored as a 16-bit signed integer to be scaled to obtain the actual quaternion value.
0x65	UM6_QUAT_CD	Stores the third and fourth components of the estimated quaternion. Each element is stored as a 16-bit signed integer to be scaled to obtain the actual quaternion value.
0x66	UM6_ERROR_COV_00	Stores the element in row 0 column 0 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x67	UM6_ERROR_COV_01	Stores the element in row 0 column 1 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x68	UM6_ERROR_COV_02	Stores the element in row 0 column 2 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x69	UM6_ERROR_COV_03	Stores the element in row 0 column 3 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x6A	UM6_ERROR_COV_10	Stores the element in row 1 column 0 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x6B	UM6_ERROR_COV_11	Stores the element in row 1 column 1 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x6C	UM6_ERROR_COV_12	Stores the element in row 1 column 2 of the error covariance matrix. This is a 32-bit IEEE floating point value.

0x6D	UM6_ERROR_COV_13	Stores the element in row 1 column 3 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x6E	UM6_ERROR_COV_20	Stores the element in row 2 column 0 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x6F	UM6_ERROR_COV_21	Stores the element in row 2 column 1 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x70	UM6_ERROR_COV_22	Stores the element in row 2 column 2 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x71	UM6_ERROR_COV_23	Stores the element in row 2 column 3 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x72	UM6_ERROR_COV_30	Stores the element in row 3 column 0 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x73	UM6_ERROR_COV_31	Stores the element in row 3 column 1 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x74	UM6_ERROR_COV_32	Stores the element in row 3 column 2 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x75	UM6_ERROR_COV_33	Stores the element in row 3 column 3 of the error covariance matrix. This is a 32-bit IEEE floating point value.
0x76	UM6_TEMPERATURE	Stores the measured temperature used for rate gyro temperature compensation. This is a 32-bit IEEE floating point value. This register is only available on firmware version UM2A and later.
0x77	UM6_GPS_LONGITUDE	Measured GPS longitude in degrees. Stored as a 32-bit IEEE Floating Point.
0x78	UM6_GPS_LATITUDE	Measured GPS latitude in degrees. Stored as a 32-bit IEEE Floating Point.
0x79	UM6_GPS_ALTITUDE	Measured GPS altitude in meters. Stored as a 32-bit IEEE Floating Point.
0x7A	UM6_GPS_POSITION_N	GPS north position in meters, referenced to the GPS home latitude set in the UM6_GPS_HOME_LAT configuration register. Stored as a 32-bit IEEE Floating Point.
0x7B	UM6_GPS_POSITION_E	GPS east position in meters, referenced to the GPS home latitude set in the UM6_GPS_HOME_LAT configuration register. Stored as a 32-bit IEEE Floating Point.
0x7C	UM6_GPS_POSITION_H	GPS altitude in meters, referenced to the GPS home altitude set in UM6_GPS_HOME_ALTITUDE configuration register. Stored as a 32-bit IEEE Floating Point.
0x7D	UM6_GPS_COURSE_SPEED	Course and speed of the GPS unit as reported by a VTG packet.
0x7E	UM6_GPS_SAT_SUMMARY	Summary of satellite information.

0x7F	UM6_GPS_SAT_1_2	Satellite ID and SNR for satellites 1 and 2.
0x80	UM6_GPS_SAT_3_4	Satellite ID and SNR for satellites 3 and 4.
0x81	UM6_GPS_SAT_5_6	Satellite ID and SNR for satellites 5 and 6.
0x82	UM6_GPS_SAT_7_8	Satellite ID and SNR for satellites 7 and 8.
0x83	UM6_GPS_SAT_9_10	Satellite ID and SNR for satellites 9 and 10.
0x84	UM6_GPS_SAT_11_12	Satellite ID and SNR for satellites 11 and 12.

11.2.2. UM6_STATUS (0x55)

Description
UM6 Status Register. Stores results of self-test operation and reports other internal errors.

Table 41 - Status Register Definition

B3								B2							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MAG INI	ACC INI	GYR INI	ST GX	ST GY	ST GZ	ST AX	ST AY	ST AZ	ST MX	ST MY	ST MZ	BUS GYR	BUS ACC	BUS MAG	EKF DIV

B1								B0							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GYR DEL	ACC DEL	MAG DEL	RES												ST

Table 42 - Status Register Bit Definitions

BITS	NAME	DESCRIPTION
0	ST	If asserted, indicates that a self-test operation was completed.
1-12	RES	Reserved. These bits are ignored.
13	MAG DEL	If asserted, indicates that the processor did not receive data from the magnetic sensor for longer than expected.
14	ACC DEL	If asserted, indicates that the processor did not receive data from the accelerometer for longer than expected.
15	GYR DEL	If asserted, indicates that the processor did not receive data from the rate gyros for longer than expected.
16	EKF DIV	If asserted, indicates that the EKF state estimates became divergent and the EKF was forced to restart.
17	BUS MAG	Indicates that there was a bus error while communicating with the magnetic sensor.
18	BUS ACC	Indicates that there was a bus error while communicating with the accelerometers.
19	BUS GYR	Indicates that there was a bus error while communicating with the rate gyros.
20	ST MZ	Indicates that the self-test operation failed on the magnetometer z-axis.
21	ST MY	Indicates that the self-test operation failed on the magnetometer y-axis.
22	ST MX	Indicates that the self-test operation failed on the magnetometer x-axis.
23	ST AZ	Indicates that the self-test operation failed on the accelerometer z-

		axis.
24	ST AY	Indicates that the self-test operation failed on the accelerometer y-axis.
25	ST AX	Indicates that the self-test operation failed on the accelerometer x-axis.
26	ST GZ	Indicates that the self-test operation failed on the rate gyro z-axis.
27	ST GY	Indicates that the self-test operation failed on the rate gyro y-axis.
28	ST GX	Indicates that the self-test operation failed on the rate gyro x-axis.
29	GYR INI	Indicates that rate gyro startup initialization failed. Usually indicates that the rate gyro is damaged.
30	ACC INI	Indicates that accelerometer startup initialization failed. Usually indicates that the accelerometer is damaged.
31	MAG INI	Indicates that magnetometer startup initialization failed. Usually indicates that the magnetometer is damaged.

11.2.3. UM6_GYRO_RAW_XY (0x56)

Description
Stores the most recent data acquired from the X and Y rate gyro axes. This register contains unmodified data from the sensor. The raw sensor data for each axis is stored as a 16-bit 2's complement integer.

Table 43 - Raw Gyro X/Y Register Definition

B3	B2	B1	B0
X gyro data (16-bit, little-endian, 2's complement)		Y gyro data (16-bit, little-endian, 2's complement)	

11.2.4. UM6_GYRO_RAW_Z (0x57)

Description
Stores the most recent data acquired from the Z rate gyro axis. This register contains unmodified data from the sensor. The raw sensor data is stored as a 16-bit 2's complement integer.

Table 44 - Raw Gyro Z Register Definition

B3	B2	B1	B0
Z gyro data (16-bit, little-endian, 2's complement)		Reserved	

11.2.5. UM6_ACCEL_RAW_XY (0x58)

Description
Stores the most recent data acquired from the X and Y accelerometer axes. This register contains unmodified data from the sensor. The raw sensor data for each axis is stored as a 16-bit 2's complement integer.

Table 45 - Raw Accel X/Y Register Definition

B3	B2	B1	B0
X accel data (16-bit, little-endian, 2's complement)		Y accel data (16-bit, little-endian, 2's complement)	

11.2.6. UM6_ACCEL_RAW_Z (0x59)

Description
Stores the most recent data acquired from the Z accelerometer axis. This register contains unmodified data from the sensor. The raw sensor data is stored as a 16-bit 2's complement integer.

Table 46 - Raw Accel Z Register Definition

B3	B2	B1	B0
Z accel data (16-bit, little-endian, 2's complement)		Reserved	

11.2.7. UM6_MAG_RAW_XY (0x5A)

Description
Stores the most recent data acquired from the X and Y magnetometer axes. This register contains unmodified data from the sensor. The raw sensor data for each axis is stored as a 16-bit 2's complement integer.

Table 47 - Raw Mag X/Y Register Definition

B3	B2	B1	B0
X mag data (16-bit, little-endian, 2's complement)		Y mag data (16-bit, little-endian, 2's complement)	

11.2.8. UM6_MAG_RAW_Z (0x5B)

Description
Stores the most recent data acquired from the Z magnetometer axis. This register contains unmodified data from the sensor. The raw sensor data is stored as a 16-bit 2's complement integer.

Table 48 - Raw Mag Z Register Definition

B3	B2	B1	B0
Z mag data (16-bit, little-endian, 2's complement)		Reserved	

11.2.9. UM6_GYRO_PROC_XY (0x5C)

Description
Stores the most recent processed data acquired from the X and Y axis rate gyros. To obtain the processed data, the UM6 subtracts biases from the raw data and then multiplies the gyro

measurement vector by the gyro calibration matrix to perform scaling and axis alignment operations:

processed_data = R_gyro(raw_gyro - gyro_biases)*

where *R_gyro* is the gyro calibration matrix, *raw_gyro* is a vector containing the raw rate gyro data from all three axes, and *gyro_biases* is a vector containing the gyro biases to be removed from the measurements.

The processed gyro data are stored internally as IEEE floating point values, but the UM6_GYRO_PROC_XY register stores the data as 16-bit 2's complement integers to minimize the amount of data that must be transmitting when communicating with the sensor.

To convert the register data from 16-bit 2's complement integers to actual angular rates in degrees per second, the data should be multiplied by the scale factor 0.0610352 as shown below.

*angular_rate = register_data*0.0610352*

Table 49 -Processed Gyro X/Y Register Definition

B3	B2	B1	B0
X gyro data (16-bit, little-endian, 2's complement)		Y gyro data (16-bit, little-endian, 2's complement)	

11.2.10. UM6_GYRO_PROC_Z (0x5D)

Description
Stores the most recent processed data acquired from the Z axis rate gyro. To obtain the processed data, the UM6 subtracts the Z-axis bias from the raw data and then multiplies the gyro measurement vector by the gyro calibration matrix to perform scaling and axis alignment operations:

processed_data = R_gyro(raw_gyro - gyro_biases)*

where *R_gyro* is the gyro calibration matrix, *raw_gyro* is a vector containing the raw rate gyro data from all three axes, and *gyro_biases* is a vector containing the gyro biases to be removed from the measurements.

The processed gyro measurement is stored internally as an IEEE floating point value, but the UM6_GYRO_PROC_Z register stores the data as a 16-bit 2's complement integer to minimize the amount of data that must be transmitting when communicating with the sensor.

To convert the register data from a 16-bit 2's complement integer to the actual angular rate in degrees per second, the data should be multiplied by the scale factor 0.0610352 as shown below.

*angular_rate = register_data*0.0610352*

Table 50 - Processed Gyro Z Register Definition

B3	B2	B1	B0
Z gyro data (16-bit, little-endian, 2's complement)		Reserved	

11.2.11. UM6_ACCEL_PROC_XY (0x5E)

Description
Stores the most recent processed data acquired from the X and Y axis accelerometers. To obtain the processed data, the UM6 subtracts biases from the raw data and then multiplies the accelerometer measurement vector by the accelerometer calibration matrix to perform scaling and axis alignment operations:

processed_data = R_accel(raw_accel - accel_biases)*

where *R_accel* is the accelerometer calibration matrix, *raw_accel* is a vector containing the raw accelerometer data from all three axes, and *accel_biases* is a vector containing the accelerometer biases to be removed from the measurements.

The processed accelerometer data are stored internally as IEEE floating point values, but the UM6_ACCEL_PROC_XY register stores the data as 16-bit 2's complement integers to minimize the amount of data that must be transmitting when communicating with the sensor.

To convert the register data from 16-bit 2's complement integers to actual acceleration in gravities, the data should be multiplied by the scale factor 0.000183105 as shown below.

acceleration = register_data 0.000183105*

Table 51 - Processed Accel X/Y Register Definition

B3	B2	B1	B0
X accel data (16-bit, little-endian, 2's complement)		Y accel data (16-bit, little-endian, 2's complement)	

11.2.12. UM6_ACCEL_PROC_Z (0x5F)

Description
Stores the most recent processed data acquired from the Z axis accelerometer. To obtain the processed data, the UM6 subtracts biases from the raw data and then multiplies the accelerometer measurement vector by the accelerometer calibration matrix to perform scaling and axis alignment operations:

processed_data = R_accel(raw_accel - accel_biases)*

where *R_accel* is the accelerometer calibration matrix, *raw_accel* is a vector containing the raw accelerometer data from all three axes, and *accel_biases* is a vector containing the accelerometer biases to be removed from the measurements.

The processed gyro measurement is stored internally as an IEEE floating point value, but the UM6_ACCEL_PROC_Z register stores the data as a 16-bit 2's complement integer to minimize the amount of data that must be transmitting when communicating with the sensor.

To convert the register data from a 16-bit 2's complement integer to the actual acceleration in gravities, the data should be multiplied by the scale factor 0.000183105 as shown below.

*acceleration = register_data*0.000183105*

Table 52 - Processed Accel Z Register Definition

B3	B2	B1	B0
Z accel data (16-bit, little-endian, 2's complement)		Reserved	

11.2.13. UM6_MAG_PROC_XY (0x60)

Description
Stores the most recent processed data acquired from the X and Y axis magnetometers. To obtain the processed data, the UM6 subtracts biases from the raw data and then multiplies the magnetometer measurement vector by the magnetometer calibration matrix to perform scaling and axis alignment operations:

processed_data = R_mag(raw_mag - mag_biases)*

where *R_mag* is the magnetometer calibration matrix, *raw_mag* is a vector containing the raw magnetometer data from all three axes, and *mag_biases* is a vector containing the magnetometer biases to be removed from the measurements.

The processed accelerometer data are stored internally as IEEE floating point values, but the UM6_MAG_PROC_XY register stores the data as 16-bit 2's complement integers to minimize the amount of data that must be transmitting when communicating with the sensor.

To convert the register data from 16-bit 2's complement integers to a unit-norm (assuming proper calibration) magnetic-field vector, the data should be multiplied by the scale factor 0.000305176 as shown below.

magnetic field = register_data 0.000305176*

Table 53 - Processed Mag X/Y Register Definition

B3	B2	B1	B0
X mag data (16-bit, little-endian, 2's complement)		Y mag data (16-bit, little-endian, 2's complement)	

11.2.14. UM6_MAG_PROC_Z (0x61)

Description
Stores the most recent processed data acquired from the Z axis magnetometer. To obtain the processed data, the UM6 subtracts magnetometer biases from the raw data and then multiplies the resulting vector by the magnetometer calibration matrix to perform scaling and axis alignment operations:

processed_data = R_mag(raw_mag - mag_biases)*

where *R_mag* is the magnetometer calibration matrix, *raw_mag* is a vector containing the raw magnetometer data from all three axes, and *mag_biases* is a vector containing the magnetometer biases to be removed from the measurements.

The processed gyro measurement is stored internally as an IEEE floating point value, but the UM6_MAG_PROC_Z register stores the data as a 16-bit 2's complement integer to minimize the amount of data that must be transmitting when communicating with the sensor.

To convert the register data from 16-bit 2's complement integers to a unit-norm (assuming proper calibration) magnetic-field vector, the data should be multiplied by the scale factor 0.000305176 as shown below.

*magnetic field = register_data*0.000305176*

Table 54 - Processed Mag Z Register Definition

B3	B2	B1	B0
Z mag data (16-bit, little-endian, 2's complement)		Reserved	

11.2.15. UM6_EULER_PHI_THETA (0x62)

Description
Stores the most recently computed roll (phi) and pitch (theta) angle estimates. The angle estimates are stored as 16-bit 2's complement integers. To obtain the actual angle estimate in degrees, the register data should be multiplied by the scale factor 0.0109863 as shown below

angle estimate = register_data 0.0109863*

Note that the contents of this register are valid even when quaternion estimation mode is enabled - in quaternion mode, the attitude is computed using quaternions, and Euler Angles are computed after the fact so that they can be used if desired.

Table 55 - Euler Angle Roll/Pitch Register Definition

B3	B2	B1	B0
Roll angle (phi)		Pitch angle (theta)	

11.2.16. UM6_EULER_PSI (0x63)

Description
Stores the most recently computed yaw (psi) angle estimate. The angle estimate is stored as a 16-bit 2's complement integer. To obtain the actual angle estimate in degrees, the register data should be multiplied by the scale factor 0.0109863 as shown below

angle estimate = register_data 0.0109863*

Note that the contents of this register are valid even when quaternion estimation mode is enabled - in quaternion mode, the attitude is computed using quaternions, and Euler Angles are computed after the fact so that they can be used if desired.

Table 56 - Euler Angle Yaw Register Definition

B3	B2	B1	B0
Yaw angle (psi)		Reserved	

11.2.17. UM6_QUAT_AB (0x64)

Description
Stores the first two components of the most recent quaternion attitude estimate. The estimates are stored as 16-bit 2's complement integers. To obtain the quaternion components, the register data should be multiplied by the scale factor 0.0000335693as shown below

quaternion = register_data 0.0000335693*

Note that the contents of this register are only valid when quaternion estimation mode is enabled.

Table 57 - Quaternion A/B Register Definition

B3	B2	B1	B0
First component of quaternion vector (a)		Second component of quaternion vector (b)	

11.2.18. UM6_QUAT_CD (0x65)

Description
Stores the second two components of the most recent quaternion attitude estimate. The estimates are stored as 16-bit 2's complement integers. To obtain the quaternion components, the register data should be multiplied by the scale factor 0.0000335693as shown below

quaternion = register_data 0.0000335693*

Note that the contents of this register are only valid when quaternion estimation mode is enabled.

Table 58 - Quaternion C/D Register Definition

B3	B2	B1	B0
First component of quaternion vector (c)		Second component of quaternion vector (d)	

11.2.19. UM6_ERROR_COVxx (0x66 - 0x75)

Description

A set of 16 registers storing the error covariance matrix. When the UM6 is in quaternion estimation mode, this matrix corresponds to the error covariance matrix for the quaternion vector *[a b c d]*. When quaternion estimation is disabled (ie. Euler Angles are being used), the matrix corresponds to the error covariance matrix for the state vector *[phi theta psi]*, where *phi* is the roll angle, *theta* is the pitch angle, and *psi* is the yaw angle. In the latter case, only the first three rows and columns of the error covariance matrix contain relevant data.

The matrix indices are stored in row-major order. Addresses 0x66 through 0x69 store the first row, 0x6A-0x6D the second, 0x6E-0x71 the third, and 0x72 - 0x75 the fourth.

Table 59 - Error Covariance Register Definition

B3	B2	B1	B0
UM6_MAG_CALxx (32-bit IEEE floating point value)			

11.2.20. UM6_TEMPERATURE (0x76)

Description

Stores the temperature measured by the onboard rate gyro temperature sensors. This value is used for temperature compensation of the rate gyros.

This register is only available in firmware revision UM2A and later.

Table 60 - Temperature Register Definition

B3	B2	B1	B0
UM6_TEMPERATURE (32-bit IEEE floating point value)			

11.2.21. UM6_GPS_LONGITUDE (0x77)

Description

If a GPS module is attached to the UM6, this register stores the measured longitude in degrees.

Table 61 - GPS Longitude Register Definition

B3	B2	B1	B0
UM6_GPS_LONGITUDE (32-bit IEEE floating point value)			

11.2.22. UM6_GPS_LATITUDE (0x78)

Description

If a GPS module is attached to the UM6, this register stores the measured latitude in degrees.

Table 62 - GPS Latitude Register Definition

B3	B2	B1	B0
UM6_GPS_LATITUDE (32-bit IEEE floating point value)			

11.2.23. UM6_GPS_ALTITUDE (0x79)

Description

If a GPS module is attached to the UM6, this register stores the measured altitude in meters.

Table 63 - GPS Altitude Register Definition

B3	B2	B1	B0
UM6_GPS_ALTITUDE (32-bit IEEE floating point value)			

11.2.24. UM6_GPS_POSITION_N (0x7A)

Description

If a GPS module is attached to the UM6, this register stores the relative position in meters north of the home position set in the UM6_GPS_HOME_LAT register.

Table 64 - GPS Relative North Position Register Definition

B3	B2	B1	B0
UM6_GPS_POSITION_N (32-bit IEEE floating point value)			

11.2.25. UM6_GPS_POSITION_E (0x7B)

Description

If a GPS module is attached to the UM6, this register stores the relative position in meters east of the home position set in the UM6_GPS_HOME_LON register.

Table 65 - GPS Relative East Position Register Definition

B3	B2	B1	B0
UM6_GPS_POSITION_E (32-bit IEEE floating point value)			

11.2.26. UM6_GPS_POSITION_H (0x7C)

Description

If a GPS module is attached to the UM6, this register stores the altitude in meters relative to the home position set in the UM6_GPS_HOME_ALTITUDE register.

Table 66 - GPS Relative Altitude Register Definition

B3	B2	B1	B0
UM6_GPS_POSITION_H (32-bit IEEE floating point value)			

11.2.27. UM6_GPS_COURSE_SPEED (0x7D)

Description

If a GPS module is attached to the UM6, this register stores the course and speed of the GPS module as reported by the NMEA VTG packet.

The ground course is a 16-bit signed integer representing the ground course angle in hundredths of a degree. To retrieve the actual ground course in degrees, divide the register value by 100.

The ground speed is a 16-bit unsigned integer representing the ground speed in hundredths of meters per second. To retrieve the actual ground speed in meters per second, divide the register value by 100.

Table 67 - GPS Course/Speed Register Definition

B3	B2	B1	B0
Ground Course (16-bit signed integer)		Ground Speed (16-bit unsigned integer)	

11.2.28. UM6_GPS_SAT_SUMMARY (0x7E)

Description

If a GPS module is attached to the UM6, this register stores the a summary of satellite information, including HDOP and VDOP, number of satellites used in the fix, and the GPS mode (2D, 3D, or no fix). The data fields in this register are only valid if the GPS module is configured to transmit the NMEA packets containing the relevant information.

Table 68 - GPS Satellite Summary Register Definition

B3								B2							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE		Satellite Count				HDOP									

B1								B0							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VDOP										RES					

Table 69 - GPS Satellite Summary Bit Definitions

BITS	NAME	DESCRIPTION
0-5	*Reserved*	These bits are unused.
6-15	*VDOP*	Vertical Dilution of Precision as reported by the GPS receiver. This is an unsigned, 10-bit integer. The actual VDOP value is given by *vdop = register_value/10*
16-25	*HDOP*	Horizontal Dilution of Precision as reported by the GPS receiver. This is an unsigned, 10-bit integer. The actual HDOP value is given by *hdop = register_value/10*
26-29	*Satellite Count*	Number of satellites used in the position fix. This is an unsigned 4-bit integer.
30 - 31	*MODE*	GPS mode as reported by GSA packet. 0 = No MODE information received

		1 = No fix 2 = 2D fix 3 = 3D fix

11.2.29. UM6_GPS_SAT_x_y (0x7F - 0x84)

Description

This set of six registers contains the satellite ID and SNR of each satellite being tracked by the GPS receiver (up to 12 satellites). This information is provided in NMEA GSV packets from the GPS receiver.

Each register stores data for up to two satellites, with both the satellite ID and the satellite SNR stored in 8-bit unsigned integers.

Table 70 - Satellite Data Register Description

B3	B2	B1	B0
Satellite x ID	Satellite x SNR	Satellite y ID	Satellite y SNR

11.3. UM6 Command Registers

Command registers are technically not registers at all, but they provide a convenient way to trigger specific commands onboard the UM6 using the same communication interface used to write to and read from registers. Commands are initiated by executing a read command for the relevant command address using the UART.

Table 71 - Summary of UM6 Commands

Address	Packet Name	Description
0xAA	UM6_GET_FW_VERSION	Causes the UM6 to report the firmware revision.
0xAB	UM6_FLASH_COMMIT	Causes the UM6 to write all current configuration values to flash (makes configuration persist when power is cycled)
0xAC	UM6_ZERO_GYROS	Causes the UM6 to start executing the zero gyros command. Once the "zero gyros" operation begins, a COMMAND_COMPLETE packet is sent. After the "zero gyros" command finishes, the gyro bias registers are transmitted over the UART.
0xAD	UM6_RESET_EKF	Causes the estimation algorithm to reset itself (resets orientation estimates and error covariance estimates)
0xAE	UM6_GET_DATA	Causes the UM6 to transmit data from all active channels over the UART.
0xAF	UM6_SET_ACCEL_REF	Sets the accelerometer reference vector used by the EKF during accelerometer updates.
0xB0	UM6_SET_MAG_REF	Sets the magnetometer reference vector used

		by the EKF during magnetometer updates.
0xB1	UM6_RESET_TO_FACTORY	Causes the UM6 to load factory default settings.
0xB2	RESERVED	RESERVED
0xB3	UM6_SET_HOME_POSITION	When GPS is connected, this command causes the most recently received GPS position to be used as the home position.
0xFD	UM6_BAD_CHECKSUM	Not technically a command. A packet with this address is transmitted by the UM6 when it receives a packet with a bad checksum. This address should only be used by the UM6.
0xFE	UM6_UNKNOWN_ADDRESS	Not technically a command. A packet with this address is transmitted by the UM6 when it receives a packet referencing a register address that does not exist.
0xFF	UM6_INVALID_BATCH_SIZE	Not technically a command. A packet with this address is transmitted by the UM6 when it receives a batch read or write that would cause the UM6 to read from or write to an address that does not exist.

11.3.1. UM6_GET_FW_VERSION (0xAA)

Causes the UM6 to report the firmware revision operating on the sensor. The firmware version is stored on the UM6 as a four-byte sequence of characters (ie. one register in length). For example, when the command is received over the UART, a packet containing the firmware revision is transmitted in response (the PT byte = 0, the Address byte = 0xAA, and the data section contains the firmware version).

11.3.2. UM6_FLASH_COMMIT (0xAB)

Causes the UM6 to write all current configuration values to flash. This makes the current configuration persist after power is cycled.

The UM6 will respond by sending a COMMAND_COMPLETE packet over the UART if it succeeds, and COMMAND_FAILED packet otherwise.

11.3.3. UM6_ZERO_GYROS (0xAC)

Causes the UM6 to start averaging gyro measurements to automatically determine the gyro biases. The UM6 should be stationary during this operation. When the UM6_ZERO_GYROS command is received, the UM6 responds by transmitting a COMMAND_COMPLETE packet. In fact, the command is not complete and is just beginning, but the response packet is used to indicate that the command was received. When the zero gyros operation completes, the new biases are applied and the contents of the gyro bias registers are transmitted over the UART.

11.3.4. UM6_RESET_EKF (0xAD)

Causes the estimation algorithm to reset itself (resets orientation estimates and error covariance estimates). When the UM6_RESET_EKF command is received over the UART, the UM6 responds by transmitting a COMMAND_COMPLETE packet.

11.3.5. UM6_GET_DATA (0xAE)

Causes the UM6 to transmit data from all active channels over the UART.

11.3.6. SET_ACCEL_REF (0xAF)

Sets the current accelerometer measurement as the accelerometer reference vector used by the EKF during accelerometer updates. This vector indicates the expected accelerometer output when the UM6 pitch, roll, and yaw angles are zero. When the UM6 receives the SET_ACCEL_REF command over the UART, a COMMAND_COMPLETE packet is transmitted over the UART in response.

11.3.7. UM6_SET_MAG_REF (0xB0)

Sets the current magnetometer measurement as the magnetometer reference vector used by the EKF during magnetometer updates. This vector indicates the expected magnetometer output when the UM6 pitch, roll, and yaw angles are zero. When the UM6 receives the SET_MAG_REF command over the UART, a COMMAND_COMPLETE packet is transmitted over the UART in response.

11.3.8. UM6_RESET_TO_FACTORY (0xB1)

Causes the UM6 to load factory default configuration settings to RAM. Factory default settings are automatically loaded on startup unless new settings have been saved to FLASH. Once new settings have been written to FLASH, the UM6_RESET_TO_FACTORY command must be sent to the sensor to retrieve the default settings.

11.3.9. UM6_SET_HOME_POSITION (0xB3)

Causes the UM6 to use the last received GPS position as the "home" position. All future north, east, and altitude positions will be calculated with respect to the new home reference.

11.3.10. UM6_BAD_CHECKSUM (0xFD)

A packet with this address is transmitted by the UM6 when a packet with an invalid checksum is received by the sensor.

11.3.11. UM6_UNKNOWN_ADDRESS (0xFE)

A packet with this address is transmitted by the UM6 when it receives a packet referencing a register address that does not exist.

11.3.12. UM6_INVALID_BATCH_SIZE (0xFF)

A packet with this address is transmitted by the UM6 when it receives a batch read or write that would cause the UM6 to read from or write to an address that does not exist.

Disclaimer

This document is provided as reference only; "typical" device specifications must be evaluated by the end-user. CH Robotics reserves the right to modify this document and the products it describes without notice.

CH Robotics products are not intended for use in weapons systems, aircraft, life-saving or life-sustaining systems, automobiles, or any other application where failure could result in injury, death, property damage, or environmental damage.