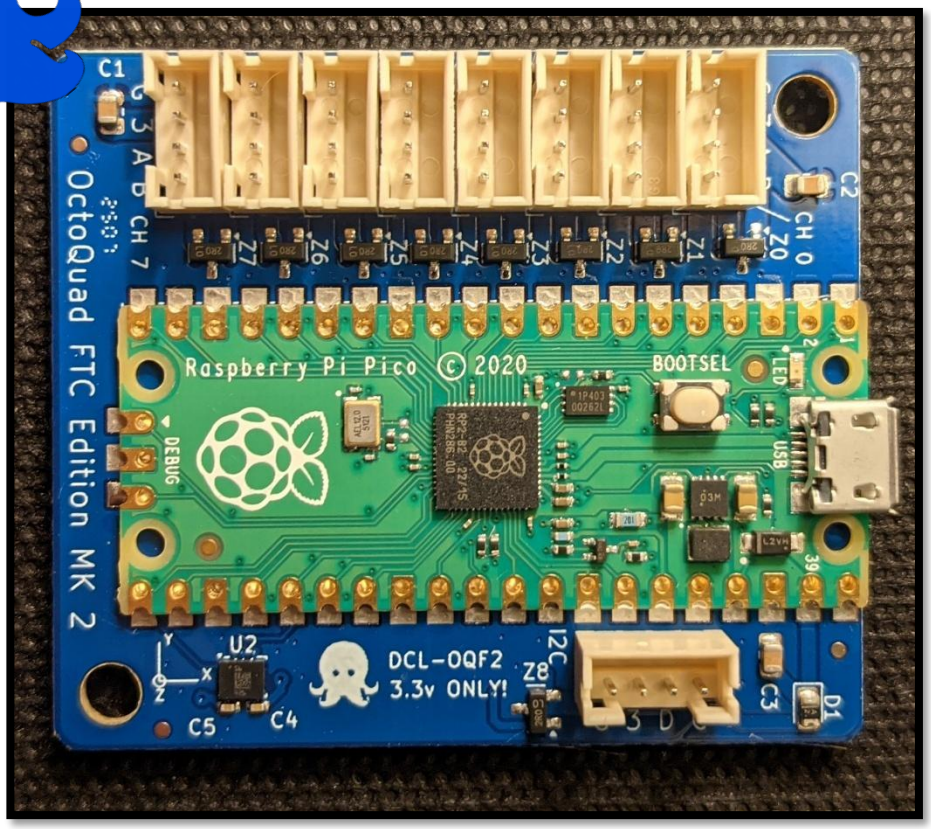
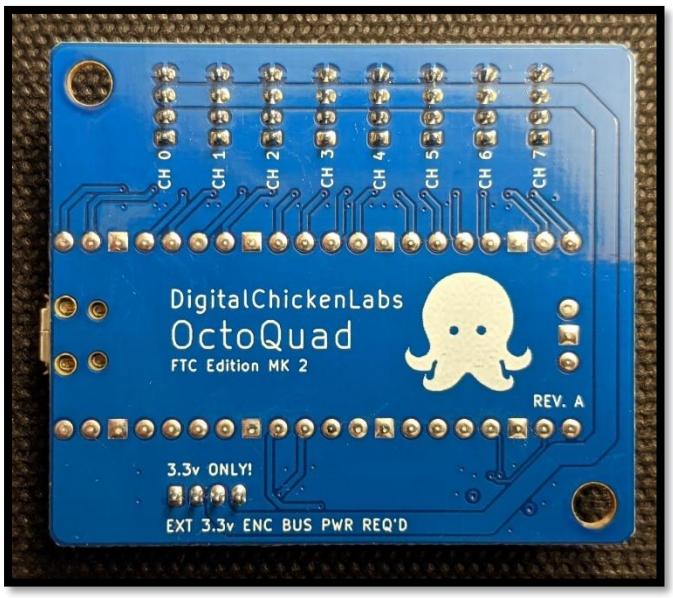
OctoQuad FTC Edition Specification





Spec. Rev. 3.0C – 3/5/2025

Table of Contents

1 Introduction

1.1 Overview

This document describes the operation of the OctoQuad FTC Edition module and the programming interface.

1.2 Board Versions

Both the OctoQuad FTC Edition MK1 and MK2 support 8 quadrature / PWM encoder inputs.

However, the MK2 board also has an onboard IMU and absolute localizer for use with passive tracking wheels (A.K.A deadwheel odometry pods).

If you are not sure which model you have, check the text on the backside of the PCB, to the left side of the octopus. It will either say “FTC Edition MK1” or “FTC Edition MK2”.

Note that the MK1 shipped with firmware v2, while the MK2 ships with firmware v3. The MK1 boards can be upgraded to firmware v3, but the MK2 boards cannot be downgraded to firmware v2.

1.3 Description

The OctoQuad provides a means to read up to eight **quadrature encoders** or **pulse width signals** on the REV Robotics Expansion Hub / Control Hub, where reading these signals from the DIO ports is not possible. The MK2 hardware revision features an onboard IMU and an absolute localizer subsystem (designed for use with passive tracking wheels) to help FTC robots navigate around the playing field with high precision.

For quadrature encoders, counts are tracked using signed 32-bit integers, and the counts for each encoder are individually resettable. Additionally, the velocity of each encoder is tracked using a signed 16-bit integer which represents the delta counts during a user-configurable sampling interval.

For pulse width measurement, pulse width is measured in microseconds with a maximum duration of 65535µs. Velocity can be measured for pulse width absolute encoders, and is measured as the change in microsecond pulse width during the user configurable sampling interval and reported as a signed 16-bit integer. The velocity calculation requires user-specified minimum and maximum pulse width values. On firmware v3 or later, absolute pulse with encoder position can be “accumulated” to allow tracking the position across multiple revolutions.

The onboard absolute localizer uses inputs from two passive tracking wheels connected to the OctoQuad along with the onboard IMU to track robot XY position and heading, XY velocity and angular velocity, at 1.92KHz.

When bulk reading encoder data or localizer data, a CRC16 is available to verify that the data has been received intact and not corrupted during transit, a common problem for I2C buses in electrically noisy conditions such as an FTC robot.

1.4 Supported Firmware Version

This document supports firmware version 3.

Tracking absolute pulse width encoder position across multiple rotations is not supported on firmware versions prior to v3.

Note that the MK1 shipped with firmware v2, while the MK2 ships with firmware v3. The MK1 boards can be upgraded to firmware v3, but the MK2 boards cannot be downgraded to firmware v2.

2 Electrical Specifications

2.1 Logic Level

The OctoQuad module uses 3.3v logic and power for I2C bus communications, as well as 3.3v power and logic for quadrature encoder signals. **The I2C and encoder connections are *NOT* 5v tolerant!**

2.2 Power

The OctoQuad FTC Edition requires external 3.3v power on the I2C header to power the encoder bus and does not supporting powering the encoder bus via USB.

2.3 Quadrature signal input

The step rate should not exceed 1 million steps/sec on any individual port (higher rates may work, but have not been tested). Note that the OctoQuad does NOT provide pull-up resistors for the A/B quadrature channels.

2.4 Pulse width signal input

The OctoQuad can measure pulse width signals from 1µs to 65535µs

2.5 ESD Protection

The encoder channels and I2C lines are protected from ESD to +/- 30kV (air) on the OctoQuad FTC Edition.

2.6 Onboard IMU

The onboard IMU is an STMicroelectronics LSM6DSV16X

3 Pinouts

3.1 4-pin JST-PH Encoder Channel Connectors

Each encoder channel connector provides power and A/B quadrature or pulse width input signal connections.

PCB Pin Label	Function
G	Ground
3	3.3v power supply for encoder
A	Quadrature channel A **OR** Pulse Width Input
B	Quadrature channel B

3.2 4-pin JST-PH I2C Connector

This connector provides connections to power the OctoQuad and exposes the I2C interface data pins. It is pin-compatible with the I2C ports on the REV Robotics Control Hub / Expansion Hub.

PCB Pin Label	Function
G	Ground
3	3.3v power input
D	I2C bus data line
C	I2C bus clock line

4 LED Status indications

4.1 Overview

The status light on the OctoQuad is used to indicate various states of communication with an I2C bus master.

4.2 LED Patterns

4.2.1 Looping sequence of one blink followed by a pause

Indicates that the OctoQuad is powered up and ready to accept communications.

4.2.2 Rapid flashing (7Hz)

Indicates that there is in-flight or recent communication on the bus

4.2.3 Slow flashing (1Hz)

Indicates that bus communication has occurred since power-up, but no recent communication has occurred.

4.2.4 Very rapid flashing (10Hz)

Internal error; please contact Digital Chicken Labs support (digitalchickenlabs@gmail.com)

5 Field-Upgradable Firmware

5.1 Obtaining firmware files

From time to time, official firmware updates for the OctoQuad may be released. Firmware binaries may be found at https://github.com/DigitalChickenLabs/OctoQuad. **WARNING:** Flashing unofficial firmware may cause permanent damage to the OctoQuad, or to devices to which it is connected. **Important note: The MK2 hardware revision is ONLY compatible with firmware v3 or newer. The MK1 hardware revision is compatible with firmware v2 or v3.**

5.2 Flashing firmware

To flash a firmware image onto the OctoQuad, follow the procedure below:

1. Remove all power and data connections from the OctoQuad.
2. Press and hold the BOOTSEL button (‘B’ on resin printed case)
3. While holding BOOTSEL, connect the OctoQuad to a computer using the micro-USB port

4. Wait until the emulated USB drive appears on the computer. The LED will remain off.
5. Drag-n-drop the firmware image onto the emulated USB drive
6. The OctoQuad will automatically flash the firmware and reboot. Flashing is complete when the emulated USB drive disappears and the status LED begins blinking an interface code.

6 I2C Interface

6.1 Overview

The OctoQuad supports operating as a slave on the standard I2C interface, using the register model. Bus clock rates of up to 400KHz are supported. **The OctoQuad's I2C address is 0x30.**

6.2 I2C Wedged Bus Recovery

The OctoQuad can be configured to attempt recovery of a stuck I2C bus in certain scenarios. See section 8.5.2 for more details.

6.3 Register access

Some registers are read-only, some are write-only, and others are read/write, as indicated in the register map. Writing to a read-only register will have no effect. Data read from a write-only register is undefined.

6.4 Register Map

Address	Type	Access	Contents
0x00	uint8_t	R	Chip ID (will read 0x51)
0x01	uint8_t	R	Firmware version (major)
0x02	uint8_t	R	Firmware version (minor)
0x03	uint8_t	R	Firmware version (engineering)
0x04	uint8_t	W	Command Register
0x05	uint8_t	RW	Command Data Register 0
0x06	uint8_t	RW	Command Data Register 1
0x07	uint8_t	RW	Command Data Register 2
0x08	uint8_t	RW	Command Data Register 3
0x09	uint8_t	RW	Command Data Register 4
0x0A	uint8_t	RW	Command Data Register 5
0x0B	uint8_t	RW	Command Data Register 6
0x0C	uint8_t	R	Localizer Yaw Axis Auto-Detect Decision
0x0D	uint8_t	R	Localizer Status
0x0E – 0x0F	int16_t	R	Localizer X axis velocity (mm/s)
0x10 – 0x11	int16_t	R	Localizer Y axis velocity (mm/s)
0x12 – 0x13	int16_t	R	Localizer Heading Axis Velocity (rad/s * 600)

0x14 – 0x15	int16_t	RW	Localizer X position (mm)
0x16 – 0x17	int16_t	RW	Localizer Y position (mm)
0x18 – 0x19	int16_t	RW	Localizer Heading (rad * 5000)
0x1A– 0x1B	uint16_t	R	Localizer CRC16 (0x0D --> 0x19)
0x1C – 0x1F	int32_t	R	Channel 0 data (quadrature count **OR** µs pulse width)
0x20 – 0x23	int32_t	R	Channel 1 data (quadrature count **OR** µs pulse width)
0x24 – 0x27	int32_t	R	Channel 2 data (quadrature count **OR** µs pulse width)
0x28 – 0x2B	int32_t	R	Channel 3 data (quadrature count **OR** µs pulse width)
0x2C – 0x2F	int32_t	R	Channel 4 data (quadrature count **OR** µs pulse width)
0x30 – 0x33	int32_t	R	Channel 5 data (quadrature count **OR** µs pulse width)
0x34 – 0x37	int32_t	R	Channel 6 data (quadrature count **OR** µs pulse width)
0x38 – 0x3B	int32_t	R	Channel 7 data (quadrature count **OR** µs pulse width)
0x3C – 0x3D	int16_t	R	Channel 0 velocity (counts per µs sampling interval)
0x3E – 0x3F	int16_t	R	Channel 1 velocity (counts per µs sampling interval)
0x40 – 0x41	int16_t	R	Channel 2 velocity (counts per µs sampling interval)
0x42 – 0x43	int16_t	R	Channel 3 velocity (counts per µs sampling interval)
0x44 – 0x45	int16_t	R	Channel 4 velocity (counts per µs sampling interval)
0x46 – 0x47	int16_t	R	Channel 5 velocity (counts per µs sampling interval)
0x48 – 0x49	int16_t	R	Channel 6 velocity (counts per µs sampling interval)
0x4A – 0x4B	int16_t	R	Channel 7 velocity (counts per µs sampling interval)
0x4C – 0x4D	uint16_t	R	Encoder data CRC16 (0x0D --> 0x19)

6.5 Register descriptions

6.5.1 Chip ID register

This register will *always* read **0x51** and may be used to confirm a proper bus connection with the OctoQuad.

6.5.2 Firmware version registers

The firmware version follows the scheme *major.minor.engineering* where each of three numbers is obtained from the corresponding register. For instance, if the registers read {2, 3, 4} then the firmware version is 2.3.4.

6.5.3 Command Register & Command Data Registers 0-6

The Command Register can be used to issue various commands to the OctoQuad, with up to 7 bytes of related data (to be written to the command operand registers). See the *Commands* section.

6.5.4 Localizer Yaw Axis Auto-Detect Decision Register

Reports the axis orientation the absolute localizer chose based on gravity detection.

Value	Meaning
0	Undecided
1	X -axis
2	X-axis (inverted)
3	Y-axis
4	Y-axis (inverted)
5	Z-axis
6	Z-axis (inverted)

6.5.5 Localizer Status Register

Reports the status of the absolute localizer routine

Value	Meaning
0	Invalid
1	Not Ready
2	Warming Up IMU
3	Calibrating IMU
4	Running Pose Integration
5	Faulted (IMU not detected)

6.5.6 Localizer X/Y Axis Velocity

Reports absolute axis velocity in mm/s, averaged over the configured localizer velocity sample interval parameter. For example, if the sampling interval is 20ms, a new velocity reading will be calculated 50 times per second.

6.5.7 Localizer Heading Axis Velocity

Reports angular velocity on the automatically chosen axis (indicated by the *Localizer Yaw Axis Auto-Detect Decision Register*) in rad/s, scaled by a factor of 600. Perform a floating-point division by 600 to obtain the value in rad/s.

6.5.8 Localizer X/Y Position

Reports the absolute position in the X axis in mm. Writing to these registers allows “teleporting” the current robot pose elsewhere.

6.5.9 Localizer Heading

Reports the heading on the automatically chosen axis (indicated by the *Localizer Yaw Axis Auto-Detect Decision Register*) in radians, scaled by a factor of 5000. Perform a floating-point division by 5000 to obtain the value in radians. Writing to this register allows “teleporting” the robot to a new heading.

6.5.10 Localizer CRC16

Contains the CRC16 for the data contained in registers 0x0D through 0x19. Note that you MUST read the *ENTIRE RANGE* of 0x0D through 0x1B in *one* operation if you wish to use this CRC value. Refer to section 10 for details on the algorithm used.

6.5.11 Channel data registers

These registers contain either quadrature counts or pulse width (in microseconds) for each channel, depending on the channel bank configuration. In either case, the value for each channel is a signed 32-bit integer. Pulse width will, of course, never be negative. If the channel is set to PWM mode and the absolute encoder wrap tracking parameter is enabled, the register will contain the pulse width accumulated across multiple revolutions of the encoder. **Note, however, that when using the wrap tracking feature, the channel pulse width min/max parameter must be set correctly.**

6.5.12 Channel velocity registers

These registers contain signed 16-bit velocity measurements for each channel.

For quadrature encoders, the velocity is defined as the net change in counts during the velocity sampling interval (see below). For example, if the sampling interval is 100ms and at the beginning of the interval the encoder count is 1234 and at the end of the interval the count is 1200, then the velocity value reported in the register will be -34. This would indicate a velocity of -34 counts/0.1s, or -340 counts/s. To determine the velocity in counts/s, user code must perform the appropriate multiplication factor based on the configured measurement interval.

The velocity sampling interval can be reduced to prevent overflow of the 16-bit counters when using encoders that output a very large number of steps per second, or, it can be increased to provide greater velocity precision on low step-rate encoders.

For pulse width input (absolute encoders) velocity is defined as the net change in microseconds pulse length during the velocity sampling interval. (See discussion of quadrature velocity above). Wrap-around is tracked internally at a much higher speed than the velocity measurement interval, so even if an absolute encoder is rotated more than a full rotation during the velocity measurement interval, the reported velocity will still be correct. **Note, however, that when using an absolute pulse width encoder, the channel pulse width min/max parameter must be set correctly.**

6.5.13 Encoder Data CRC16

Contains the CRC16 for the data contained in registers 0x1C through 0x4B. Note that you MUST read the *ENTIRE RANGE* of 0x1C through 0x4D in *one* operation if you wish to use this CRC value. Refer to section 10 for details on the algorithm used.

7 Commands

7.1 Description

The Command Register (see register map) may be used to issue various commands to the OctoQuad, with up to 7 bytes of related data (to be written to the command operand registers).

Not all commands require this extra data. For those that do, the operand register(s) *must be written in the same bus transaction in which the Command Register is written*.

7.2 Command List

The following commands are supported:

Command	Description	Operand 0	Operands 1-6
0x00	NO-OP (No command)		
0x01	Set Parameter	Parameter ID	Parameter-dependent
0x02	Get Parameter	Parameter ID	Parameter-dependent
0x03	Save Parameters to flash		
0x14	Reset Everything		
0x15	Reset Channels	8-bit channel bitfield	
0x28	Reset Localizer + Calibrate IMU		

7.2.1 Reset Everything Command

This command resets quadrature encoder counts or measured pulse width to zero, and sets all parameters to their factory defaults. NOTE: this does ***not*** save the newly reset parameters to flash.

7.2.2 Reset Channels Command

This command zeros quadrature count(s) / pulse width measurement for one or more channels. The first and only operand is a bitfield mapping to channel numbers. Each bit in the operand corresponds to a channel, e.g., bit 3 corresponds to channel 3. When issuing this command, for every bit that is set in the operand, the corresponding encoder's count will be reset.

Multiple channels can be reset in one command operation. For example, writing 01000001 as the operand will reset channel 6 and channel 0.

Reset Channel Command – Operand 1								
Bit	7	6	5	4	3	2	1	0
Effect	C7 Reset	C6 Reset	C5 Reset	C4 Reset	C3 Reset	C2 Reset	C1 Reset	C0 Reset

7.2.3 Reset Localizer + Calibrate IMU Command

This command resets the localizer pose, loads updated localizer parameters, and begins the IMU calibration routine. As part of the IMU calibration, the OctoQuad will detect the gravity vector to automatically choose the proper axis and axis sign for yaw. Once this determination is made, the chosen axis will remain fixed until the next issuance of this command. Watch the Localizer Yaw Axis Auto-Detect Decision Register and the Localizer Status Register to determine when the reset is complete.

7.2.4 Set Parameter Command

This command is used to set the value for a parameter. See below section on parameters.

7.2.5 Get Parameter Command

This command is used to get the current value of a parameter. See below section on parameters.

7.2.6 Save Parameters to Flash Command

This command may be used to save the current value of all parameters to flash, so that they will be automatically restored after a power cycle.

8 Parameters

8.1 Description

The OctoQuad supports various user-configurable options ("parameters") which affect its operation. *Parameters are not directly mapped to registers*. Parameters may optionally be saved to flash so that they are automatically restored after a power cycle. (See *Save Parameters to Flash* command).

8.2 Parameter List

Parameter ID	Name	Values
0x00	Channel directions	Channel bitfield (uint8_t)
0x01	I2C Recovery Mode	I2C Recovery mode (uint8_t)
0x02	Channel Bank Config	Channel Bank Mode (uint8_t)
0x03	Channel Velocity Interval	Interval_ms (uint8_t)
0x04	Channel Pulse Width min/max	Min_µs (uint16_t) Max_µs (uint16_t)
0x05	PWM Abs. Encoder Wrap Track	Channel bitfield (uint8_t)
0x32	Localizer X-Axis Tracking Wheel Resolution	Ticks per 1mm of travel along X axis (float32)
0x33	Localizer Y-Axis Tracking Wheel Resolution	Ticks per 1mm of travel along Y axis (float32)
0x34	Localizer X-Axis TCP Offset	Offset mm (float32)
0x35	Localizer Y-Axis TCP Offset	Offset mm (float32)
0x36	Localizer IMU scalar	IMU scalar (float32)
0x37	Localizer X-axis wheel port	Port number (uint8_t)
0x38	Localizer Y-axis wheel port	Port number (uint8_t)
0x39	Localizer velocity calculation interval	Interval_ms (uint8_t)

8.3 Setting Parameters

A Parameter may be set by writing the *Set Parameter* command ID to the command register and filling the command data registers (sequentially) with the parameter ID, followed by the value(s) for the parameter. **For parameter names in red the parameter values must be preceded by an 8-bit integer corresponding to the channel index.** (I.e. the first command data register filled after the parameter ID must be the desired channel index, then the parameter value(s) follow in subsequent command data registers). The general format for setting parameters is as follows:

Setting a Parameter (write to these registers)			
Register	Command (0x04)	Cmd Data 0 (0x05)	Cmd Data 1-6 (0x06 – 0x0B)
Data	Set Param (0x01)	Param Number	Parameter Vals. (1st may be ch idx)

8.4 Reading Parameters

Reading the current value of a parameter is accomplished in two steps. First, write the *Read Parameter* command ID to the command register and fill the command data register 0 with the parameter ID to be read. **If reading a parameter name in red, then data register 1 must be filled with a channel index.** Once the *Read Parameter* command has been issued, the current parameter value will be filled into the command data registers (starting with command data 0) which can be retrieved with a subsequent read.

8.5 Parameter Descriptions

8.5.1 Channel Directions Parameter

This parameter is used to set the encoder count direction on a per-port basis. If the channel is operating in quadrature mode, the step direction is simply reversed.

It has no effect if the channel is operating in pulse width input mode on firmware < v2.1. On firmware >= 2.1, setting this parameter will cause the reported pulse width range (for an absolute encoder) to be inverted. For example, if the pulse width range for an absolute encoder is 1-1024µs when rotated clockwise, settings this parameter will cause the reported range to be 1-1024µs when rotated counter-clockwise. **Note, however, that when using this parameter with a channel operating in pulse width input mode, the channel pulse width min/max parameter must be set correctly.**

The first and only argument is a bitfield mapping to channel numbers. Each bit in the argument corresponds to a channel, e.g., bit 3 corresponds to channel 3. When issuing this command, for every bit that is set in the operand, the corresponding encoder channel will be negated.

Multiple channels can be configured one write to this register. For example, writing 01000001 to the operand will set channel 6 and channel 0 to be negated.

Encoder Directions Parameter – Argument 0								
Bit	7	6	5	4	3	2	1	0
Effect	E7 DIR	E6 DIR	E5 DIR	E4 DIR	E3 DIR	E2 DIR	E1 DIR	E0 DIR

8.5.2 I2C Recovery Mode Parameter

This parameter is used to set how aggressively the OctoQuad will attempt to un-wedge a hung I2C bus. It has no effect if the OctoQuad is operating in any interface mode other than I2C. Three modes are supported:

- 0: The OctoQuad will not attempt to perform any type of recovery for a stuck I2C bus
- 1: An inter-byte timeout is used for I2C transactions: successive byte transfers must occur within 50ms of each other in order to prevent the timeout from expiring. If the

timeout expires, the firmware will assume that the bus has become wedged and will reset the I2C peripheral in an attempt to recover the bus.
- 2: Inter-byte timeout from Mode 1, plus pulling clock low for a small period of time if 1500ms elapses with no communications. May help to un-wedge master-side I2C hardware on an incredibly glitch/noisy bus.

8.5.3 Channel Bank Mode Parameter

The OctoQuad contains two channel banks, covering channels 0-3 and 4-7. This parameter may be used to set which mode (quadrature or pulse width measurement) each channel bank is configured for. Possible values are:

- 0: All quadrature
- 1: All pulse width
- 2: First bank quadrature; second bank pulse width

8.5.4 Channel Velocity Measurement Interval Parameter

This parameter is used to set the time interval at which the velocity is calculated for each encoder. The value is interpreted directly as milliseconds. For example, setting the value of this parameter for a channel to the decimal value “40” means that the velocity for the respective channel will be measured at 40ms intervals. The default interval is 50ms. Setting the sampling interval to 0 will be disregarded.

8.5.5 Channel Pulse Width min/max Parameter

This parameter is used to inform the firmware of the minimum/maximum pulse lengths that an absolute encoder will output, to enable accurate velocity calculation. This will default to 1µs/1024µs

8.5.6 PWM Abs. Encoder Wrap Track Parameter (Available with FW >= v3)

This parameter is used to enable or disable the absolute encoder pulse-width “wrap tracking” accumulator. It has no effect on the channel if the channel is operating in quadrature input mode.

The first and only argument is a bitfield mapping to channel numbers. Each bit in the argument corresponds to a channel, e.g., bit 3 corresponds to channel 3. When issuing this command, for every bit that is set in the operand, the feature is enabled for the corresponding channel.

Multiple channels can be configured one write to this register. For example, writing 01000001 to the operand will enable the feature for channel 6 and channel 0.

PWM Abs. Encoder Wrap Track Parameter – Argument 0								
Bit	7	6	5	4	3	2	1	0
Effect	E7 Enbl	E6 Enbl	E5 Enbl	E4 Enbl	E3 Enbl	E2 Enbl	E1 Enbl	E0 Enbl

Enabling this feature for a channel will cause the corresponding position register to report an “accumulated” microseconds count across multiple rotations of the absolute encoder. For example, suppose an absolute encoder has a minimum pulse length of 1µs and a maximum pulse length of 1024µs. Then, the delta µs per revolution is 1023. Thus, with this feature

enabled, if the encoder were started at the 1µs point and spun for 2.5 revolutions, the accumulated delta is (1023µs/rev * 2.5rev) and position register would report +/- 25575 µs (sign depending on the direction of rotation).

It is important to note that issuing a reset command for a channel with this feature enabled does NOT clear the position register to zero. Rather, it zeros the "accumulator" (i.e. the number of "wraps") such that immediately after issuing a reset command, the position register will contain the raw pulse width in microseconds. This behavior is chosen because it maintains the ability of the absolute encoder to actually report its absolute position. If resetting the channel were to really zero out the reported position register, using an absolute encoder would not behave any differently than using a relative encoder.

When using this feature, the channel pulse width min/max parameter must be set correctly.

8.5.7 Localizer X-Axis / Y-Axis Tracking Wheel Resolution

This parameter is used to inform the localizer how many encoder ticks on the axis tracking wheel correspond to 1mm of lateral travel on that axis. It is NOT recommended to calculate this value. Rather, push the robot for a couple meters and divide the number of ticks by the distance traveled to obtain this value. NOTE: This parameter does not take effect until the next issuance of the *Reset Localizer + Calibrate IMU* Command.

8.5.8 Localizer X-Axis / Y-Axis TCP Offset

This parameter allows moving the Tracking Center Point (TCP) of the localizer. The TCP is the point about which rotation does not affect X and Y position values. Without setting any offsets, the mathematical location of the TCP is at the virtual intersection of lines which run through each tracking wheel. Often, it is convenient to relocate the TCP to the center of the robot, since most often a robot will rotate about its center. The offset describes the vector moving from the mathematical TCP to the desired TCP location (usually the center of the robot). For more details, refer to the Localizer Quick Start Guide. NOTE: This parameter does not take effect until the next issuance of the *Reset Localizer + Calibrate IMU* Command.

8.5.9 Localizer IMU Scalar

Although the localizer algorithm automatically calibrates the bias of the IMU on startup, it cannot automatically calculate the angular velocity scale factor, that is, calibrating the IMU such that it reads N deg/sec if it is rotating at N deg/sec. The recommended way to calculate this scale factor is to place the robot against a wall, reset the IMU, rotate the robot 3600 degrees (10 turns), return the robot against the wall, and then observe how many degrees off 0 the IMU reports and calculate the needed correction factor. Suppose for example after rotating CCW 3600 degrees, the IMU reports a normalized heading of -10 degrees. The correction scale factor can be calculated as $k = \frac{3600 - (-10)}{3600} = 1.002$ NOTE: This parameter does not take effect until the next issuance of the *Reset Localizer + Calibrate IMU* Command.

8.5.10 Localizer X-axis / Y-axis Wheel Port

These parameters tell the localizer which encoder ports the X and Y tracking wheels are connected to. It must be in the range [0 … 7] corresponding to channels 0 through 7 on the

OctoQuad. NOTE: This parameter does not take effect until the next issuance of the *Reset Localizer + Calibrate IMU* Command.

8.5.11 Localizer Velocity Calculation Interval

This parameter is used to set the time interval at which the velocity is calculated for the localizer. The value is interpreted directly as milliseconds. For example, setting the value of this parameter to the decimal value "40" means that the velocity will be calculated at 40ms intervals. The default interval is 25ms. Setting the sampling interval to 0 will be disregarded. NOTE: This parameter does not take effect until the next issuance of the *Reset Localizer + Calibrate IMU* Command.

9 Absolute Localizer

9.1 Overview

The onboard numerical-integration absolute localizer uses inputs from 2 passive tracking wheels connected to the OctoQuad along with the onboard IMU to track robot XY position, XY velocity, Heading, and Heading Angular Velocity in the world reference frame.

The algorithm used is a Pose Exponential running synchronized to the 1.92KHz data stream from the IMU.

9.2 Robot Mounting Options (Automatic Gravity Detection)

The OctoQuad can be mounted in any of the 6 orientations orthogonal to the ground. When issuing the *Reset Localizer + Calibrate IMU* Command, the OctoQuad will perform automatic gravity detection to determine which axis of the IMU to use for robot yaw.

9.3 Sign Conventions

The +X axis points outward from the front of the robot when viewed top-down. The +Y axis points outward from the left side of the robot when viewed top-down. A CCW rotation is positive. This convention is chosen so that traveling at a heading of 0 degrees corresponds to traveling along the +X axis, just like on a Cartesian coordinate plane. See the Localizer Quick Start Guide for more details.

9.4 Required Parameters for Use

In order to use the absolute localizer, you must set the following parameters:

- *Channel Directions* Parameter – you must configure the channel direction for the ports connected to the tracking wheels such that a positive encoder tick corresponds to a positive axis movement according to the sign convention described in 9.3.
- *Localizer X-axis / Y-axis Wheel Port* Parameters
- *Localizer X-Axis / Y-Axis Tracking Wheel Resolution* Parameters
- *Localizer X-Axis / Y-Axis TCP Offset* Parameter
- *Localizer IMU Scalar* Parameter
- *Localizer Velocity Calculation Interval* Parameter
- IMPORTANT NOTE: After configuring the parameters, you must reset the localizer before they will take effect!

10 CRC16 Calculation

10.1 Algorithm

The algorithm used is the PROFIBUS CRC16, which has the following attributes:

- Polynomial: 0x1DCF
- Initial value: 0xFFFF
- Reflect input: no
- Reflect output: no
- XOR output: 0xFFFF

10.2 Sample CRC Calculation Code

```
static const uint16_t crc16_profibus_init = 0xFFFF;
static const uint16_t crc16_profibus_xor_out = 0xFFFF;
static const uint16_t crc16_profibus_table[] = {
    0x0000, 0x1DCF, 0x3B9E, 0x2651,   0x773C, 0x6AF3, 0x4CA2, 0x516D,
    0xEE78, 0xF3B7, 0xD5E6, 0xC829,   0x9944, 0x848B, 0xA2DA, 0xBF15,
    0xC13F, 0xDCF0, 0xFAA1, 0xE76E,   0xB603, 0xABCC, 0x8D9D, 0x9052,
    0x2F47, 0x3288, 0x14D9, 0x0916,   0x587B, 0x45B4, 0x63E5, 0x7E2A,
    0x9FB1, 0x827E, 0xA42F, 0xB9E0,   0xE88D, 0xF542, 0xD313, 0xCEDC,
    0x71C9, 0x6C06, 0x4A57, 0x5798,   0x06F5, 0x1B3A, 0x3D6B, 0x20A4,
    0x5E8E, 0x4341, 0x6510, 0x78DF,   0x29B2, 0x347D, 0x122C, 0x0FE3,
    0xB0F6, 0xAD39, 0x8B68, 0x96A7,   0xC7CA, 0xDA05, 0xFC54, 0xE19B,
    0x22AD, 0x3F62, 0x1933, 0x04FC,   0x5591, 0x485E, 0x6E0F, 0x73C0,
    0xCCD5, 0xD11A, 0xF74B, 0xEA84,   0xBBE9, 0xA626, 0x8077, 0x9DB8,
    0xE392, 0xFE5D, 0xD80C, 0xC5C3,   0x94AE, 0x8961, 0xAF30, 0xB2FF,
    0x0DEA, 0x1025, 0x3674, 0x2BBB,   0x7AD6, 0x6719, 0x4148, 0x5C87,
    0xBD1C, 0xA0D3, 0x8682, 0x9B4D,   0xCA20, 0xD7EF, 0xF1BE, 0xEC71,
    0x5364, 0x4EAB, 0x68FA, 0x7535,   0x2458, 0x3997, 0x1FC6, 0x0209,
    0x7C23, 0x61EC, 0x47BD, 0x5A72,   0x0B1F, 0x16D0, 0x3081, 0x2D4E,
    0x925B, 0x8F94, 0xA9C5, 0xB40A,   0xE567, 0xF8A8, 0xDEF9, 0xC336,
    0x455A, 0x5895, 0x7EC4, 0x630B,   0x3266, 0x2FA9, 0x09F8, 0x1437,
    0xAB22, 0xB6ED, 0x90BC, 0x8D73,   0xDC1E, 0xC1D1, 0xE780, 0xFA4F,
    0x8465, 0x99AA, 0xBFFB, 0xA234,   0xF359, 0xEE96, 0xC8C7, 0xD508,
    0x6A1D, 0x77D2, 0x5183, 0x4C4C,   0x1D21, 0x00EE, 0x26BF, 0x3B70,
    0xDAEB, 0xC724, 0xE175, 0xFCBA,   0xADD7, 0xB018, 0x9649, 0x8B86,
    0x3493, 0x295C, 0x0F0D, 0x12C2,   0x43AF, 0x5E60, 0x7831, 0x65FE,
    0x1BD4, 0x061B, 0x204A, 0x3D85,   0x6CE8, 0x7127, 0x5776, 0x4AB9,
    0xF5AC, 0xE863, 0xCE32, 0xD3FD,   0x8290, 0x9F5F, 0xB90E, 0xA4C1,
    0x67F7, 0x7A38, 0x5C69, 0x41A6,   0x10CB, 0x0D04, 0x2B55, 0x369A,
    0x898F, 0x9440, 0xB211, 0xAFDE,   0xFEB3, 0xE37C, 0xC52D, 0xD8E2,
    0xA6C8, 0xBB07, 0x9D56, 0x8099,   0xD1F4, 0xCC3B, 0xEA6A, 0xF7A5,
    0x48B0, 0x557F, 0x732E, 0x6EE1,   0x3F8C, 0x2243, 0x0412, 0x19DD,
    0xF846, 0xE589, 0xC3D8, 0xDE17,   0x8F7A, 0x92B5, 0xB4E4, 0xA92B,
    0x163E, 0x0BF1, 0x2DA0, 0x306F,   0x6102, 0x7CCD, 0x5A9C, 0x4753,
    0x3979, 0x24B6, 0x02E7, 0x1F28,   0x4E45, 0x538A, 0x75DB, 0x6814,
    0xD701, 0xCACE, 0xEC9F, 0xF150,   0xA03D, 0xBDF2, 0x9BA3, 0x866C
};

uint16_t crc16_profibus_compute(const uint8_t* const dat, uint32_t len)
{
    uint16_t crc = crc16_profibus_init;

    for (uint32_t i = 0; i < len; i++)
    {
        crc = (crc << 8) ^ crc16_profibus_table[(crc >> 8) ^ dat[i]];
    }

    return crc ^ crc16_profibus_xor_out;
}
```

Special Thanks To

- j5155 (Capital City Dynamics): hardware testing & RoadRunner integration
- Miriam Sinton-Remes: hardware testing
- Laina Galayde: OctoQuad logo artwork