Creative Robotics Ltd

Enactive Torch Research Tool

ET-RT instructions

Document version 1.1.2

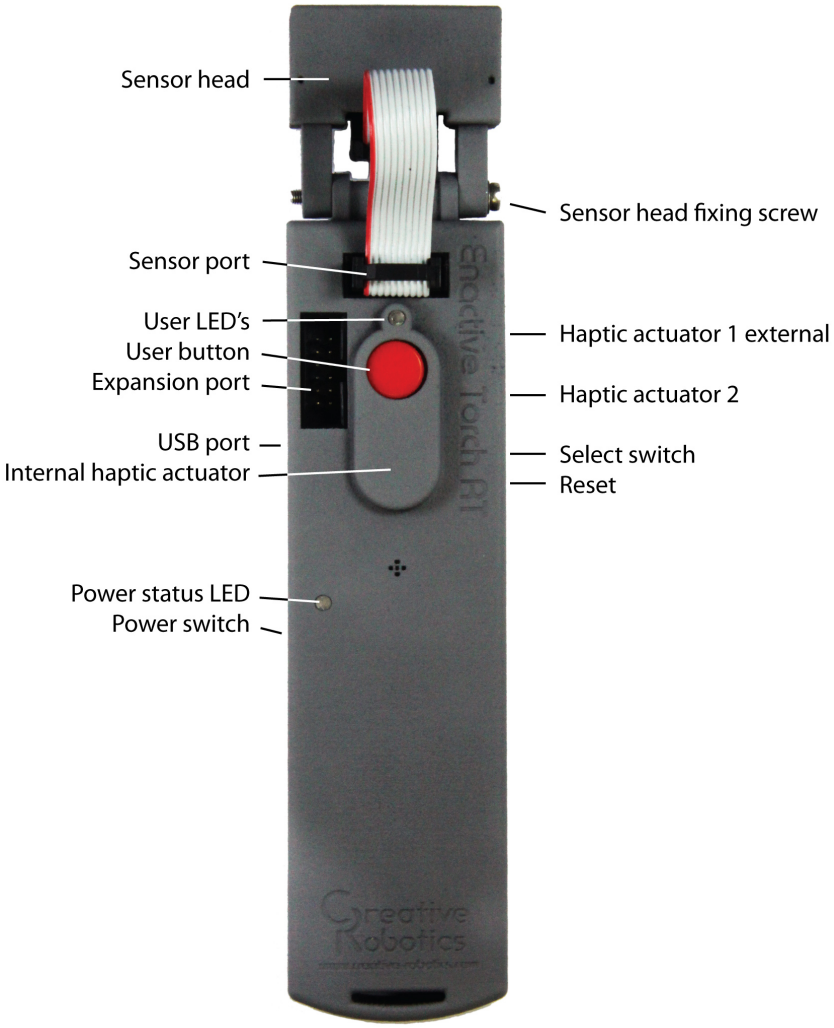
Table of Contents

Device overview

The Enactive Torch contains an internal inertial measurement unit and Bluetooth module allowing data to be streamed from the device to a computer. The internal battery is charged via the USB port. The standard sensor head contains a rangefinder with a maximum 1.5 meter range, a narrow beam LED and a light sensor, and the sensor angle relative to the body of the ET-RT can be adjusted.

The device is equipped with an internal haptic actuator called a Linear Resonant Actuator or LRA. This is mounted in the case just below the user button. A pair of 3.5mm jack sockets on the right of the device allow for two external LRA's to be plugged in and independently controlled. When an LRA is plugged into 'Haptic Actuator 1' socket it will disconnect the internal one and prevent it from operating.

In software 'Haptic Actuator 1' is called LRA1 and 'Haptic Actuator 2' is called LRA2.



Connecting to Bluetooth

The device should be marked with its Bluetooth name on a label on the rear of the case. When connecting to it via Bluetooth ensure that the ET-RT is switched on and look for the device name. Data is transferred over Bluetooth using the Synchronous Serial Protocol (SSP). The serial port needs to be configured as follows:

- Baud rate 38400
- 1 stop bit
- No parity.

Commands

The ET software incorporates a limited set of commands for configuring the device remotely. These can be sent from a serial terminal or suitable piece of software using the Bluetooth link. By default the device will begin by transmitting a device information statement that includes the firmware version and the status of the internal sensors and user input.

The following serial commands are available and should be sent as ASCII characters followed by a line feed character. These commands are defined in the ETRT-System.h file used to compile the firmware. Any adjustments made to the ET-RT using these commands will *not* persist after the device is restarted.

Software name	**Description**	**Character**
CMD_GET_DEVICE_INFO	Get device information	?
CMD_PRINT_CALIBRATION_DATA	Print calibration values for gyroscopes and accelerometers	c
CMD_DATA_STREAM_ON	Turn data streaming on	D
CMD_DATA_STREAM_OFF	Turn data streaming off	d
CMD_SET_DATA_STREAM_RAW	Set data stream type to raw data	r
CMD_SET_DATA_STREAM_YPR	Set data stream type to YPR (Yaw Pitch Roll)	y
CMD_SET_DATA_STREAM_QUA	Set data stream type to Quaternion data	q

Using the ET-RT

This section applies to devices with the default demo firmware installed

Data Formats

The example software that is supplied with the ET can output three different types of data generated by the internal sensors in addition to the data generated by the external sensor. By default it will output raw data from the internal sensors. Each packet of data is sent as human readable comma separated floating point values in a line terminated by a line feed and carriage return character. Each line is prefixed with a character indicating the type of data being sent.

Quaternion orientation data and the Roll Pitch and Yaw angles are estimates produced by combining the internal sensor reading from the gyroscopes, accelerometers and magnetometers. These estimated are computed by the device and can be subject to error. The system for estimating the orientation will also take a few seconds to converge on a solution.

Note: The system for generating Quaternions and Yaw, Pitch and Roll is under development and should not be relied on for accurate data. We recommend that you use the raw data format.

Each type of packet is formatted as follows:

Raw data format (Raw data from each internal sensor):

r, Accelerometer X, Accelerometer Y, Accelerometer Z, Gyro X, Gyro Y, Gyro Z, Magnetometer X, Magnetometer Y, Magnetometer Z, Rangefinder

YPR Format (Yaw Pitch and Roll estimate derived from internal sensors)

y, Yaw, Pitch, Roll, Rangefinder

Quaternion Format (Quaternion orientation data derived from internal sensors)

q, Quaternion 1, Quaternion 2, Quaternion 3, Quaternion 4, Rangefinder

Basic operation

Switch the ET-RT on. The status LED will illuminate red. It will begin transmitting raw data from the gyroscopes, accelerometers and sensor. This data can be read by connecting to the device from a computer via Bluetooth.

Press the red button to enable the haptic actuator (LRA). The device will map data from the rangefinder to the haptic actuator. The firmware has two modes of operation that can be changed with the selector switch.

Position 0: (switch pushed to the rear of the device) The range of an object is translated into vibration intensity where a shorter range will equate to greater intensity.

Position 1: (Switch pushed forward) The range translates into intensity but the haptic actuator delivers pulses rather than a continuous signal.

If the external LRA is plugged into haptic actuator socket 1 then it will respond. If no external actuator is plugged in then the internal one will respond.

Adjusting the sensor head angle

The sensor head can be tilted through approximately 180 degrees about the horizontal axis. This can be achieved by loosening the screw holding the head to the body. Take care not to try and adjust the head when the screw is tight. Applying too much force could cause damage to the plastic case.

Charging the battery

The battery is charged via the USB port so it can be charged by simply plugging the device into a USB port on a computer or into a USB mobile device charger. The status LED will illuminate green if the device is charged or orange if the device is charged and switched on.

Programming

To write software and program the ET-RT download the Arduino IDE from here:

https://www.arduino.cc/en/Main/Software

You will also need to install the ET-RT device library and sensor drivers in the Arduino libraries folder. The device library, sensor drivers and examples are available here:

https://github.com/CreateiveRobotics/EnactiveTorchRT

To download your software connect the ET-RT to a USB port, switch it on and allow the device drivers to install.

In the Arduino IDE select Tools > Board > LillypadUSB

Next select Tools > Port and look for your device. It should appear as a Lillypad USB.

To compile and upload your software click the upload button. When uploading the user LED will flicker orange red and green.

For more details please refer to the Arduino website.

API Documentation

The core ET-RT functions are defined in the ETRTSystem class defined in the 'ETRT-System.h' library file.

The system class only controls the internal sensors for the device. To use the sensor head a device driver class is required. The API for the standard sensor is in the next section.

ETRTSystem(void)

Class constructor

void initialise()

Initialises the ET-RT System object. This does NOT initialise the inertial measurement unit. But it will initialise the Bluetooth serial port and the USB serial port. USB serial is initiated with a baud rate of 115200k

void initialiseImu(int32_t gbx, int32_t gby, int32_t gbz, int32_t abx, int32_t aby, int32_t abz)

Initialise the inertial measurement unit with calibration values for the gyros and accelerometers.

Accepts the following arguments in this order:

Gyro bias X

Gyro bias Y

Gyro bias Z

Accelerometer bias X

Accelerometer bias Y

Accelerometer bias Z

void update()

Update the ET-RT system. This updates the inertial measurement unit, processes any commands in the serial buffer and update any haptic effects. This needs to be called approximately every 10 milliseconds Using the standard Arduino delay(time) function will interfere with this but an alternative delay function waitFor(time) is available that will continuously call the update() function whilst providing a timed delay to your software.

void setLed1(bool state)

Set the state of User LED 1.

1 = On, 0 = Off.

void setLed2(bool State)

Set the state of User LED 2.

1 = On, 0 = Off.

void Led1On()

Switch LED1 On

void Led1Off()

Switch LED1 Off

void Led2On()

Switch LED2 On

void LED2Off()

Switch LED1 Off

bool isButtonPressed()

Check the user button.

Returns 1 if pressed and 0 if not pressed.

bool getSwitchState()

Check the select switch.

Returns 1 if in the forward position and 0 if in the backward position.

bool getBluetoothState()

Read the connection status of the Bluetooth radio.

Return 1 if connected, 0 if not connected.

void beep(int pitch, int duration)

Produces an audio tone for a length of time specified in milliseconds.

void enableLra1()

Enable haptic controller number 1

void disableLra1()

Disable haptic controller number 1

void setLra1(uint16_t intensity)

Set the vibration intensity of haptic controller number 1. This will call the appropriate enable or disable function at the same time so there is no need to enable the haptic controller first.

Accepts a value in the range 0 to 1023.

void enableLra2()

Enable haptic controller number 2

void disableLra2()

Disable haptic controller number 2

void setLra2(uint16_t intensity)

Set the vibration intensity of haptic controller number 2. This will call the appropriate enable or disable function at the same time so there is no need to enable the haptic controller first.

Accepts a value in the range 0 to 1023.

void pulseLra1(int intensity, int milliTime)

Create a vibration pulse for a specified time on haptic controller 1. This function will return immediately but the update() must be called every 10ms or more to allow the haptic pulse to complete.

Accepts an intensity value in the range 0 to 1023 and a time in milliseconds.

void pulseLra2(int intensity, int milliTime)

Create a vibration pulse for a specified time on haptic controller 2. This function will return immediately but the update() must be called every 10ms or more to allow the haptic pulse to complete.

Accepts an intensity value in the range 0 to 1023 and a time in milliseconds.

void printIMURawData()

Print the raw inertial measurement data over the Bluetooth link. Data is in floating point format as comma separated values in the following order:

Accelerometer X, Y, Z, Gyroscope X, Y, Z, Magnetometer X, Y, Z

No line feeds or character returns are included.

void printIMUYPRData()

Prints the Yaw Pitch and Roll data over the Bluetooth link. Data is in floating point format as comma separated values in the following order:

Yaw, Pitch, Roll

No line feeds or character returns are included.

void printIMUQuaternionData()

Prints four quaternion values that describe the device orientation over the Bluetooth link. Data is in floating point format as comma separated values in the following order:

Q1, Q2, Q3, Q4

No line feeds or character returns are included.

void printCalibrationValues()

Prints the calibration values for the device over the Bluetooth link. Data is in floating point format as comma separated values in the following order:

Gyro X, Y, Z, Accelerometer X, Y, Z

No line feeds or character returns are included.

void printDeviceInfo()

Prints device information over the Bluetooth link. The following data is printed as individual lines:

- Device firmware version
- IMU initialisation status
- Magnetometer initialisation status
- Data stream enable state
- Button state
- Switch state

float getQ(uint8_t index)

Returns one of four quaternion values.

Accepts a value from 0 to 3 representing quaternions 1, 2, 3 and 4.

void waitFor(long int delayMillis)

A function that will delay your program for the specified time but allow the ET system to continue updating in the background. Use instead of the standard Arduino delay() function.

Accepts a delay time in milliseconds.

Standard sensor API

ETRTRangefinderS1(void)

Constructor

void initialise(bool enableState)

Initialise the sensor. Accepts an argument to determine if the rangefinder is enabled. Send 1 to enable or 0 to disable on startup.

void update(void)

Update the readings and filtered values

int getRange()

Returns the raw rangefinder reading

int getRangeFiltered()

Returns a filtered version of the rangefinder reading

int getLight()

Get the raw light sensor reading.

int getLightFiltered()

Get a filtered version of the light sensor reading.

void setLed(int ledValue)

Set the LED torch with an intensity in the range 0-255.

void LedOn()

Switch the LED torch on.

void ledOff()

Switch the LED torch off.

void enable()

Enable the rangefinder.

void disable()

Disable the rangefinder.

void setEnable(bool state)

Set the rangefinder enable state. Send 1 to enable or 0 to disable.

bool getEnableState()

Get the state of the rangefinder. Returns 1 if enabled or 0 if disabled.

void setRangeFilterLength(int length)

Set the length of the filter (Number of samples to average over)

void setLightFilterLength(int length)

Set the length of the filter (Number of samples to average over)

Sensor response

The rangefinder produces a voltage that is proportional to an object in its field of view. This response is NOT linear but conforms to the table below.

The sensor outputs a voltage that is read by an analogue to digital converter. The function getRange() returns the value generated by the analogue to digital converter.

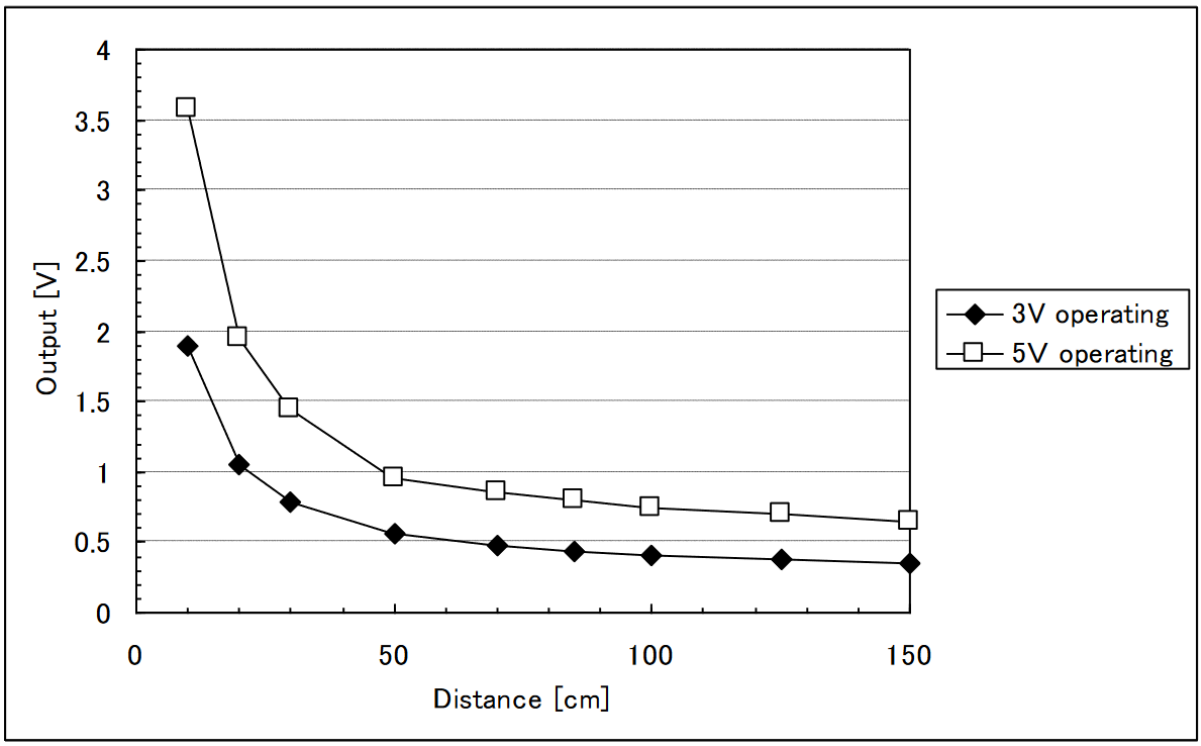
In order to convert this value to one that represents a voltage it must be scaled according to the resolution of the analogue converter which is 10 bits, or a maximum value of 1023, and the operating voltage of the device which is 3.3 volts.

This mapping can be done as follows:

Analogue to voltage scaling	3.3 ÷ 1023 = 0.0032258
Voltage to analogue scaling	1023 ÷ 3.3 = 310
To convert an analogue reading of 523 to a voltage:	Voltage = 523 × 0.0032258 = 1.7393934 Volts

To convert a voltage of 1.9 volts into the raw analogue value: ADCValue = 1.9 × 310 = 589

White paper(Reflectance ratio 90%)



Inertial Measurement Unit Axis

The diagram below illustrates the axis orientation of the gyroscope and accelerometers.

Creative Robotics Ltd

Enactive Torch Research Tool

ET-RT instructions

Document version 1.1.2

DISCLAIMER OF LIABILITY

While we always try to provide accurate information, there may be times when factual, technical or typographical inaccuracies appear in our documentation. We apologise if this happens and we reserve the right to make changes and corrections at any time, without notice. Creative Robotics expressly disclaims liability for errors or omissions in the content of this document and makes no commitment to update the information contained on these sites.

Creative Robotics expressly disclaims all liability for the interpretation and use by others of any information contained in this document. All products are subject to change without prior notice. Creative Robotics disclaims any responsibility for errors, omissions or inaccuracies in product documentation or any other data relating to our products. Creative Robotics disclaims any responsibility for loss, damage or harm caused directly or indirectly by the use of our products.

(Basically: be careful, have fun but if you mess up then don't blame us)