navX-MXP Robotics Navigation Sensor User Guide

Kauai Labs

Table of Contents

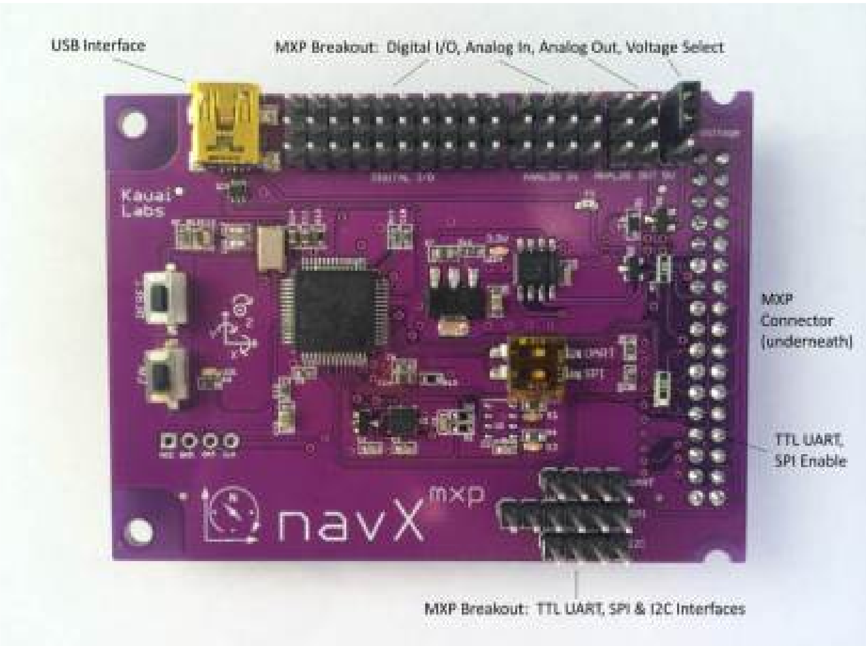
Overview
navX-MXP

navX-MXP is a **9-axis inertial/magnetic sensor** and **motion processor**. Designed for **plug-n-play** installation onto a National Instruments RoboRIO™, navX-MXP also provides RoboRIO **I/O Expansion**.

navX-MXP is a must-have add on to any RoboRIO-based control system, and includes free software libraries, example code and many more features.

navx-MXP works with the Kauai Labs Sensor Fusion Framework (SF2) to provide even more advanced capabilities.

Super-charge your robot:



- Field-Oriented Drive
- Auto-balance
- Auto-rotate to angle
- Motion Detection
- Collision Detection
- and more…

Expand your RoboRIO™:

- 10 Digital I/Os
- 4 Analog Inputs
- 2 Analog Outputs
- I2C, SPI & UART Interfaces
- Selectable Output Voltage

Features

Sophisticated Motion Processing

- High Accuracy, Low-latency Yaw, Pitch and Roll Angles
- Automatic Accelerometer/Gyroscope Calibration
- Gravity-corrected Linear Acceleration
- High-sensitivity Motion Detection
- Tilt-compensated Compass Heading
- 9-Axis absolute heading w/Magnetic disturbance detection
- Configurable Update Rate from 4 to 200Hz

Easy to Use

- Plug-n-Play Installation via RoboRIO MXP Interface
- USB, TTL UART, I2C and SPI communication interfaces
- RoboRIO libraries and sample code
- Tools for Magnetometer Calibration
- Conformal-coated circuit board

Protective Enclosure

- A custom navX-MXP enclosure can be created with a 3D printer using provided Enclosure design files
- Alternatively, the navX-MXP enclosure can be purchased online.

Open-Source

- board schematics and bill of materials.
- The Eclipse IDE and a free version of an ARM compiler can be downloaded for those wishing to customize the firmware.
- Firmware updates can be easily downloaded to the navX-MXP circuit board via the USB port.

Technical Specifications

The navX-MXP circuit board and official firmware provide inertial and magnetic measurements, with a range, accuracy and update rate as described on this page.

Note that certain performance specifications are only valid after a start-up Gyroscope/Accelerometer Calibration *period, during which time the navX-MXP circuit board must be held still.*

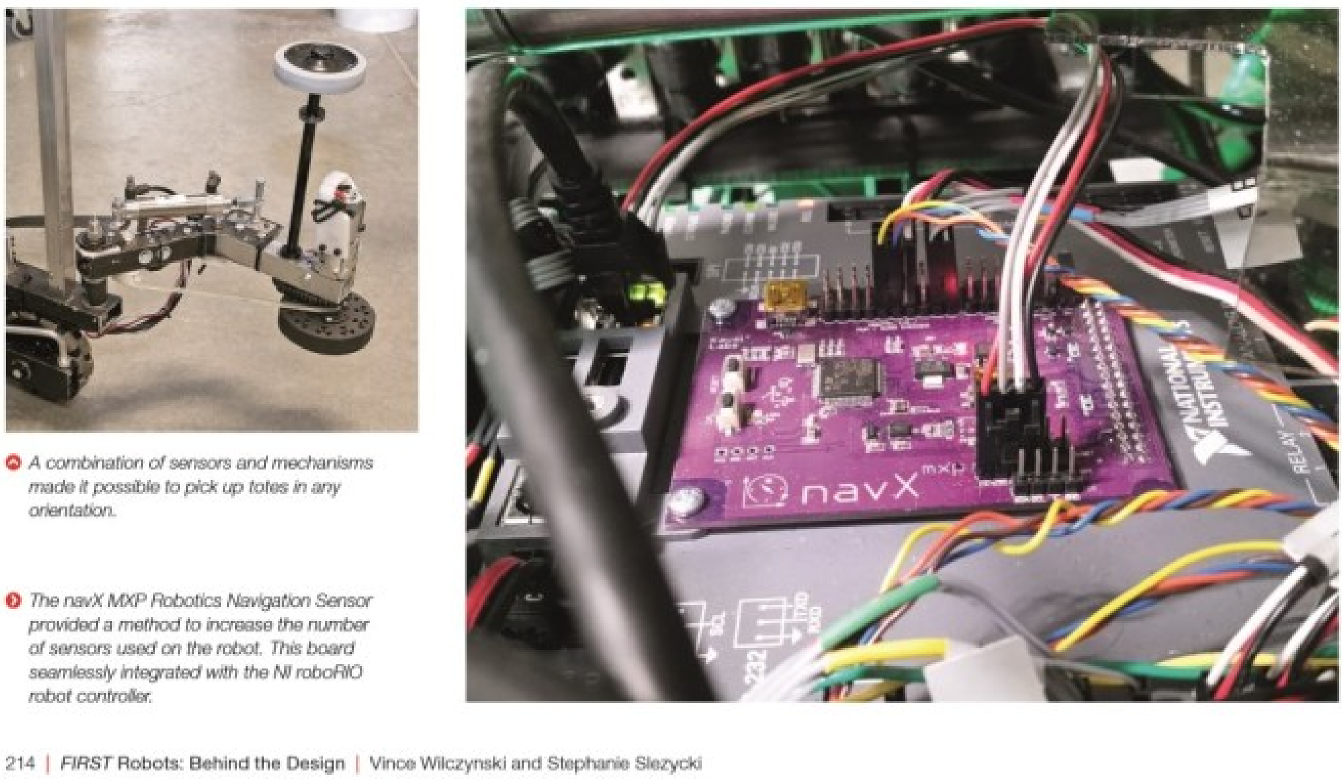
Additional details can be found in the navX-MXP datasheet.

Electrical Specifications	
Voltage:	5V DC
Current Consumption:	50 millamps
Communications Interface:	USB, TTL UART, SPI, I2C
Power Connector:	USB and/or 5VDC/GND Pins on MXP Connector
USB Connector:	USB Mini-B

"Behind the Design"

navX-MXP is mentioned several times (pages 214-217, 227 and 231) within “FIRST Robots – Behind the Design – 30 Profiles of Design, Manufacturing and Control” (2015, USFIRST).

Team 624’s 2015 Robot



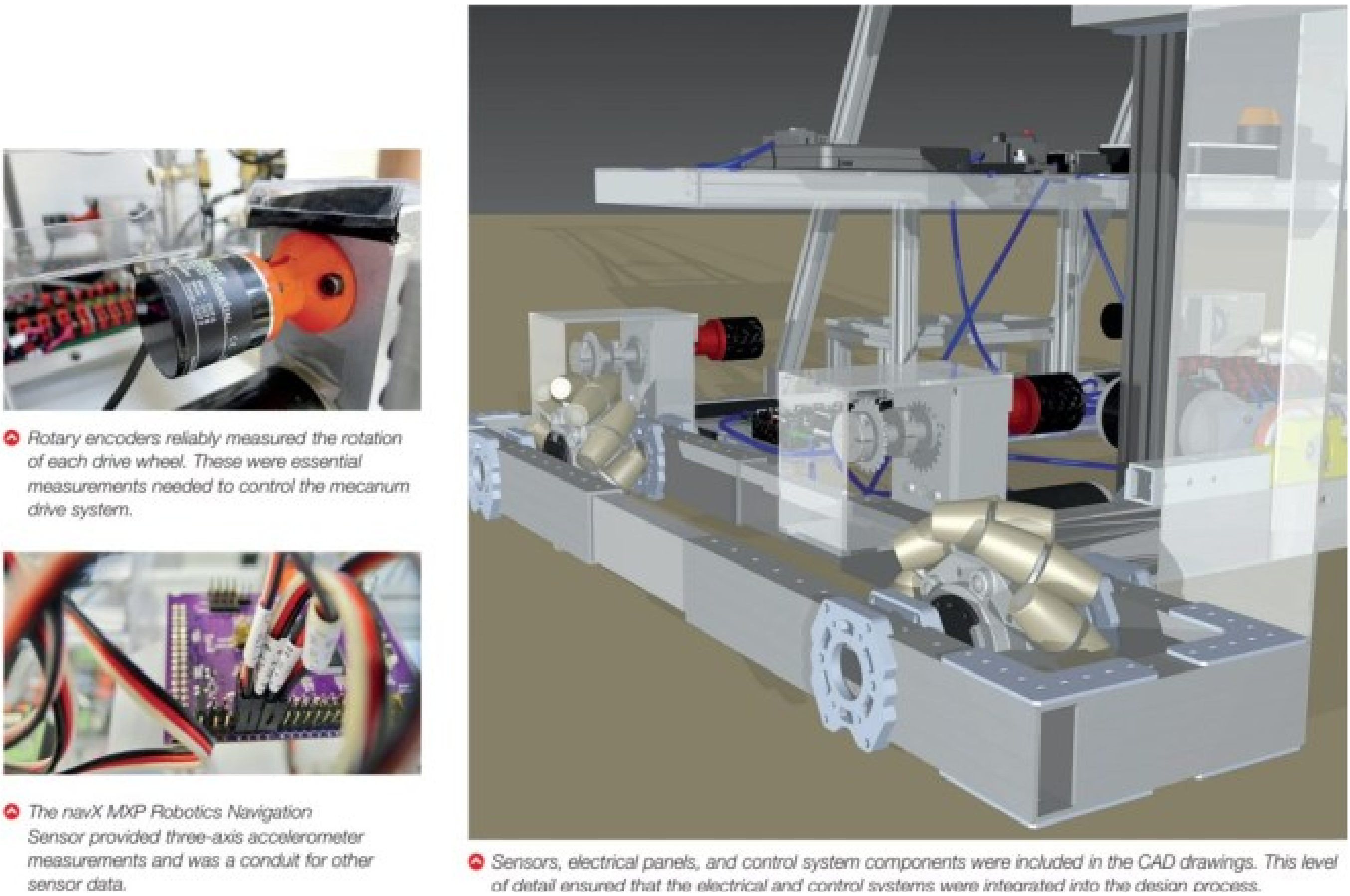
A combination of sensors and mechanisms made it possible to pick up totes in any orientation.

The navX MXP Robotics Navigation Sensor provided a method to increase the number of sensors used on the robot. This board seamlessly integrated with the NI roboRIO robot controller.

214 | *FIRST* Robots: Behind the Design | Vince Wilczynski and Stephanie Slezycki

navX-MXP on Team 624's 2015 Robot

Team 2062's 2015 Robot


Rotary encoders reliably measured the rotation of each drive wheel. These were essential measurements needed to control the mecanum drive system.

The navX MXP Robotics Navigation Sensor provided three-axis accelerometer measurements and was a conduit for other sensor data.

Sensors, electrical panels, and control system components were included in the CAD drawings. This level of detail ensured that the electrical and control systems were integrated into the design process.

navX-MXP on Team 2062's 2015 Robot

About the "Behind the Design" Book

"Behind the Design – 30 Profiles of Design, Manufacturing and Control" has six chapters that focus on CAD modeling, traditional machining, CNC mills and lathes, CNC cutting, 3D printing, and sensors/control. Each chapter profiles five FRC teams to illustrate how these technologies apply to robot design, manufacturing, and control. The book also includes vignettes between the chapters that illustrate the purpose of FIRST and its impact.

Frequently-asked Questions

Will navX-MXP work with the National Instruments RoboRIO™?

Yes, the navX-MXP is designed specifically to work with the RoboRIO. Please see the instructions for installing navX-MXP onto a FIRST FRC robot for more details, as there are several installation options.

Will navX-MXP work with the Android-based FTC Control System?

Yes, navX-MXP can be used with the Android-based FTC Control System, via its I2C interface. For more information, please see the FTC Robot Installation instructions and the description of the Android Libraries.

Will navX-MXP work with the older National Instruments CRIO™ robot controller?

Yes, navX-MXP can be used with the National Instruments CRIO robot controller by using the nav6 libraries, which are available on the . You will need a USB-to-RS232 serial converter in order to connect navX-MXP to the CRIO's RS-232 port, and you will also need a 12VDC-to-5VDConverter to provid power/ground to the power pins on the navX-MXP's MXP Connector.

Please note that the legacy nav6 libraries only use the navX-MXP serial interfaces, and do not provide access to new navX-MXP features including 9-axis headings, magnetometer calibration and magnetometer disturbance detection.

What interface/installation options are available for the navX-MXP?

- Plug-n-play install to the RoboRIO MXP port
- Connection to the RoboRIO MXP port via a male-to-female floppy-disk-style ribbon cable
- Connection to one of the RoboRIO USB Connectors via a USB Cable
- Connection of power (+5VDC)/ground to the navX-MXP's MXP Connector, and direct connection to the TTL UART, I2C or SPI pins.

Aren't the magnetometer (compass heading) readings unreliable when the navX-MXP is used on a Robot with powerful motors?

Yes, this is correct. If navX-MXP is mounted nearby any energized motors, the magnetometer's ability to measure the (weak) earth's magnetic field is severely diminished.

However, at the beginning of each FIRST FRC match, the robot is turned on for about a minute before the match begins. During this time period, the motors are not energized and thus do not add magnetic interference that would disturb the magnetometer readings. Once the magnetometer is calibrated, navX-MXP will return either an accurate magnetometer reading, or an indication that its measurement of the earth's magnetic field has been disturbed.

Magnetometer readings taken at the beginning of a match, when combined with the navX-MXP yaw measurements, enable a robot's pose and absolute heading to be maintained throughout the match. This feature of the navX-MXP is referred to as a "9-axis" heading.

Why do the Yaw angles provided by the navX-MXP drift over time?

The short answer is that the yaw angle is calculated by integrating reading from a gyroscope which measures changes in rotation, rather than absolute angles. Over time, small errors in the rotation measurements build up over time. The navX-MXP features sophisticated digital motion processing and calibration algorithms that limit this error in the yaw angle of ~1 degree per minute. For further details, please see the Yaw Drift page.

Can the navX-MXP "Displacement" estimates be used for tracking a FRC or FTC robot's change in position (dead-reckoning) during autonomous?

Accelerometer data from the navX-MXP's onboard MPU-9250 are double-integrated by the navX-MXP firmware to estimate displacement, *and are accurate to approximately 1 meter of error during a 15 second period*.

To track a FRC or FTC robot's position during autonomous requires an accuracy of about 1 cm of error per 15 seconds. While the accuracy of the navX-MXP displacement estimates might be good enough to track the position of an automobile on a road, it is too low for use in tracking a FRC or FTC robot's position during the 15 second autonomous period.

The root cause of the displacement estimate error rate is *accelerometer noise*. Estimating displacement requires first that each acceleration sample be multiple by itself twice (cubed), and then integrated over time. Practically, if a noisy signal is cubed, the result is very noisy, and when this very noisy value is integrated over time, the total amount of error grows very quickly.

The current noise levels (approximately 150 micro-g per square-root-hertz) would need to be reduced by a factor of 100 (two orders of magnitude) before displacement estimates with 1 cm of error per 15 seconds can be achieved by double-integration of accelerometers.

MEMS accelerometers which feature these low noise levels are beginning to emerge, but are currently very expensive. KauaiLabs is actively researching these technology developments and projects that MEMS technology that is both (a) low noise (1 micro-g per square root hertz) and (b) available at low cost will be available in approximately 5 years (~2020). KauaiLabs plans to develop a product which can be used for accelerometer-based dead-reckoning at that time.

Did Invensense finally publicly release a description of the DMP (Digital Motion Processor) and interface specs, or are you using what other people reverse engineered a while ago?

The navX-MXP firmware uses the officially released Invensense MotionDriver version 6.1. Kauai Labs has ported this driver to work correctly on the navX-MXP’s STM32F411 micro-controller.

What’s the difference between navX-MXP and the navX-MXP Aero?

navX-MXP and the navX-MXP Aero share a single design. navX-MXP Aero adds a pressure sensor (MS5611) providing additional altitude measurements with a resolution of 10 cm.

Since the pressure sensor is an expensive component, this sensor was left off of navX-MXP, decreasing the cost for those not desiring an altitude measurement.

Installation
Installation

Plug-n-play: navX-MXP is designed for rapid, plug-n-play installation on a National Instruments RoboRIO™, making it easy to install and integrate onto robots including a FIRST FRC Robot. navX-MXP and supports plug-n-play installation onto an Android-based FTC Robot.

Orientation: Tips and tricks for ensuring navX-MXP measurements are aligned with your robot, including the new Omnimount flexible mounting feature.

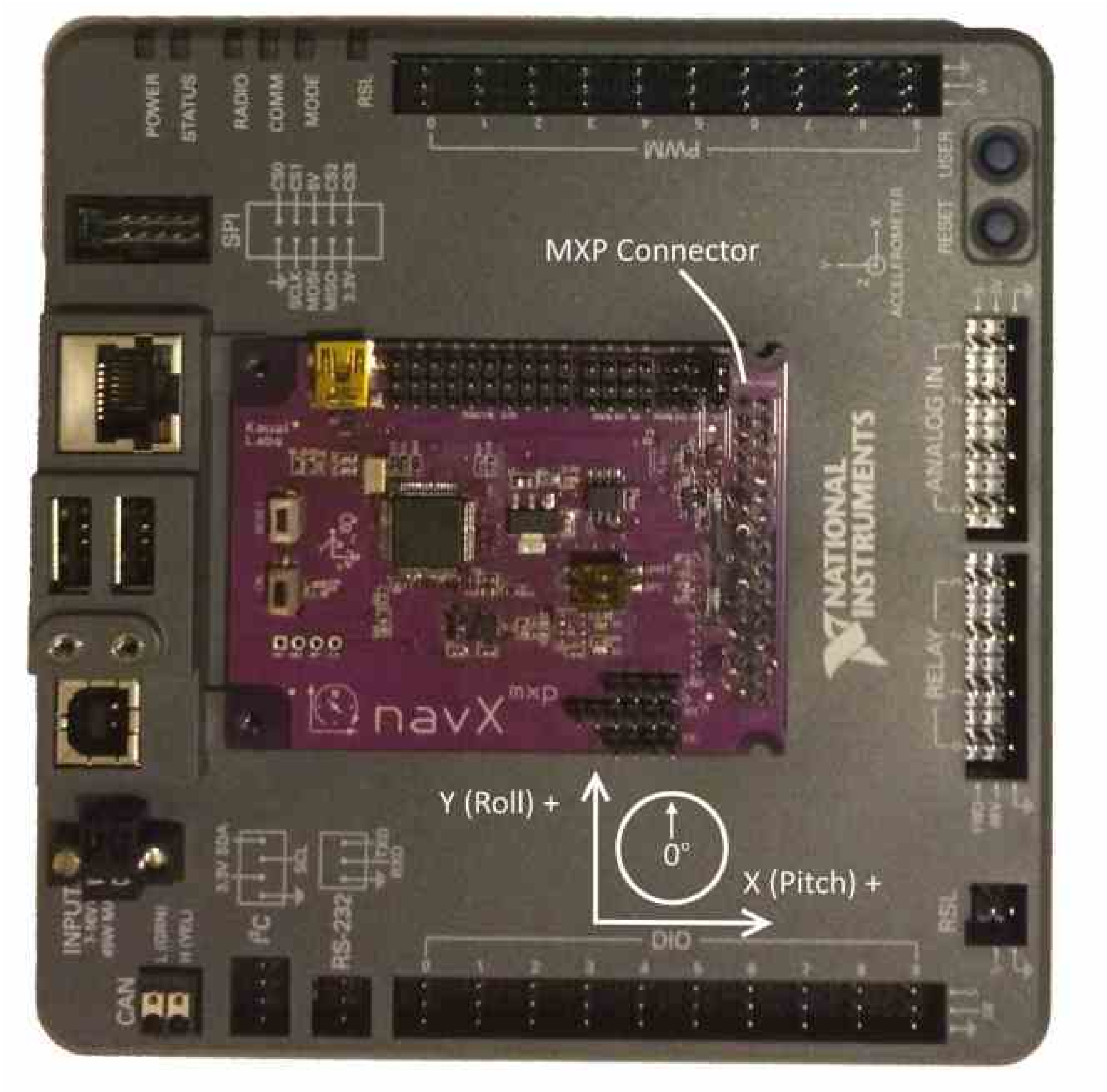
I/O Expansion: In addition to sophisticated motion processing, navX-MXP also provides analog and digital I/O expansion on a RoboRIO.

Flexibility: To allow flexible customization, navX-MXP also supports several alternative installation options as well as several communication options, providing flexibility when integrating with other components.

Enclosure: To protect an installed navX-MXP, an enclosure is available – which can be either purchased, or printed on a 3D printer using open-source design files.

RoboRIO Installation

navX-MXP is designed for plug-n-play installation onto the National Instruments RoboRIO™. This installation takes about only a minute. To install, simply place the 34-pin “MXP” Connector on the bottom of the navX-MXP circuit board into the corresponding MXP slot on the top of the RoboRIO, as shown below.


MXP Connector
Y (Roll) +
0°
X (Pitch) +
navX mxp
NATIONAL INSTRUMENTS
PWM
DIO
ANALOG IN
RELAY
USER
RESET
POWER
STATUS
RADIO
COMM
MODE
RSL
SPI
INPUT
CAN
I²C
RS-232

Securing navX-MXP to the RoboRIO

Next, secure navX-MXP to the RoboRIO using two #4-40 screws, each with a length of 3/16th inch. You can also use a 1/4 inch-long screw if you place a small washer between it and the top of the navX-MXP circuit board.

Image not found

Securing the navX-MXP circuit board and RoboRIO to the robot chassis

The navX-MXP circuit board should be mounted such that it is firmly attached to the robot chassis. ***The quality of this mounting will be directly reflected in the quality of navX-MXP inertial measurements.*** To ensure quality, carefully follow these guidelines:

- The RoboRIO on which the navX-MXP circuit board is placed should be tightly mounted; it should be a part of the chassis mass, and should move exactly as the chassis moves. Avoid mounting the navX-MXP circuit board in an area of the chassis that might be flexible, as this could introduce vibration to the inertial sensors that does not represent the chassis inertial properties.

- The navX-MXP circuit board should be mounted in the center of the chassis, which ensures the origin of the yaw/pitch/roll axes truly represent the chassis center.

- Be sure to understand the orientation of the navX-MXP circuit board, relative to the chassis, and decide whether OmniMount is needed.

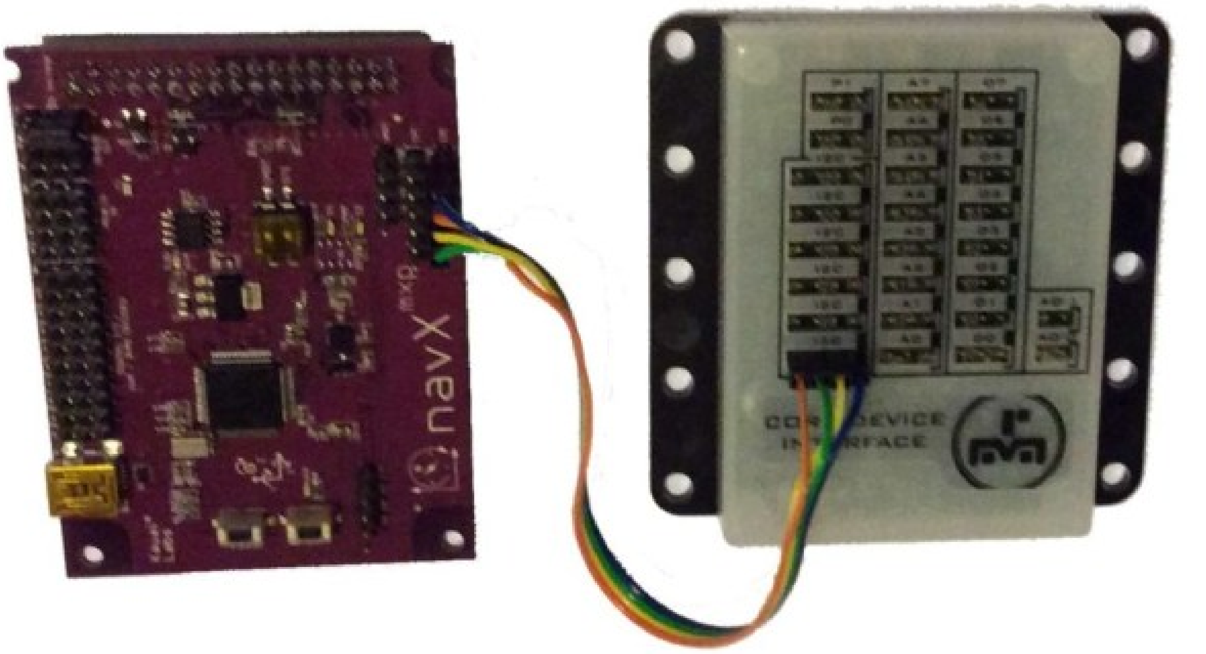
- Housing the navX-MXP circuit board in some form of protective enclosure is highly recommended, to protect it from damage. This should both protect the circuit board from damage, and provide strain relief for the cables that connect to the navX-MXP circuit board.

(Note that there are several other installation options available.)

FTC Installation

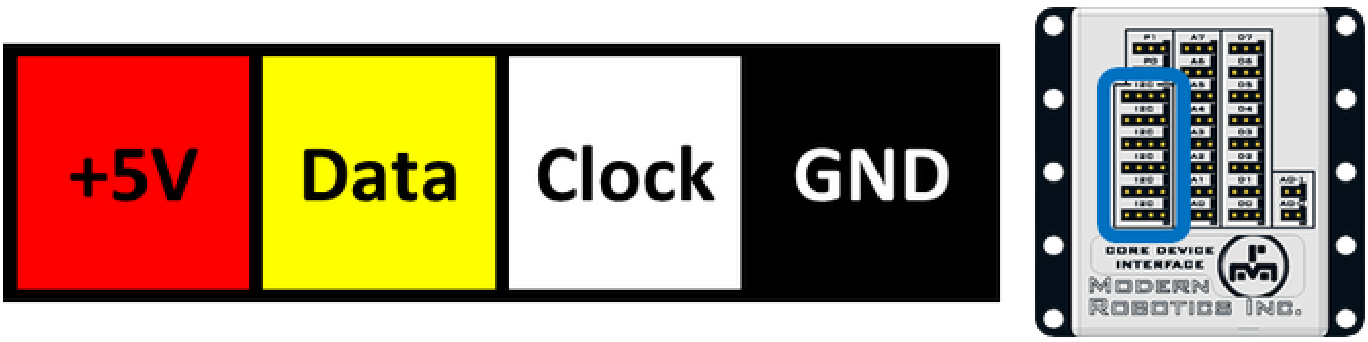
Note: navX-MXP firmware version 2.2 or higher is required to use navX-MXP w/the FTC Android-Based Robot Control System.

navX-MXP can be used with the FTC Android-Based Robot Control System released in 2015. Both power to and signaling to/from navX-MXP occurs via the I2C interface by way of the Core Device Interface Module (DIM) from Modern Robotics, Inc, as shown in the below diagram:



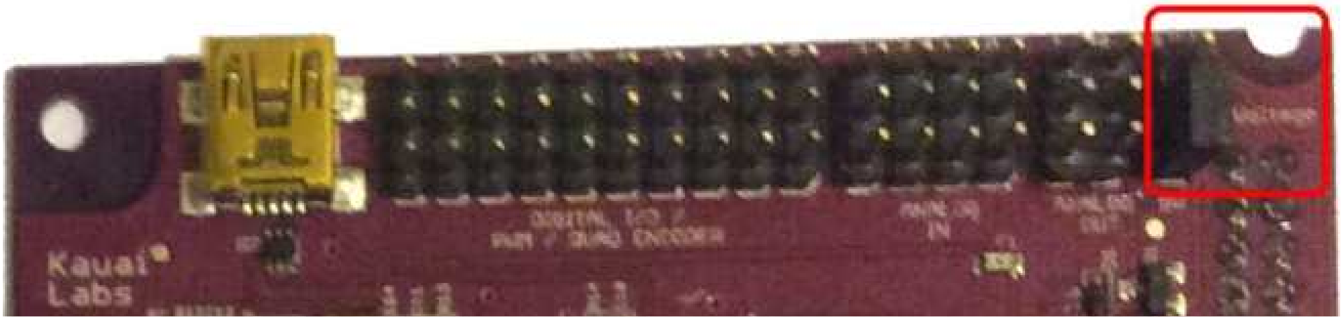
Electrical Wiring Instructions

- Select one of the 6 I2C ports on the DIM, as shown below. Note that the ports are numbered from 0, starting at the bottom-most port on the left-hand side of the DIM.



- Connect the +5V, Data (SDA), Clock (SCL) and GND pins on the selected DIM I2C port to the corresponding pins on the navX-MXP External I2C Port Connector.

- To ensure the +5V from the DIM is used to power the navX-MXP board circuitry, ensure that the “Voltage Select” jumper is set to 5V.



Electrical Wiring Verification

If properly wired, when power is applied to the DIM, the Red 3.3V LED on the navX-MXP should light up.

Double-check that the SDA and the SCL wires on the DIM match the corresponding pins on the navX-MXP circuit board.

Physical Installation on the Robot

The navX-MXP circuit board should be mounted such that it is firmly attached to the robot chassis. ***The quality of this mounting will be directly reflected in the quality of navX-MXP inertial measurements.*** To ensure quality, carefully follow these guidelines:

- Whereever the navX-MXP circuit board is placed, it should be tightly mounted; it should be a part of the chassis mass, and should move exactly as the chassis moves. Avoid mounting the navX-MXP circuit board in an area of the chassis that might be flexible, as this could introduce vibration to the inertial sensors that does not represent the chassis inertial properties.

- The navX-MXP circuit board should be mounted in the center of the chassis, which ensures the origin of the yaw/pitch/roll axes truly represent the chassis center.

- Be sure to understand the orientation of the navX-MXP circuit board, relative to the chassis, and decide whether OmniMount is needed.

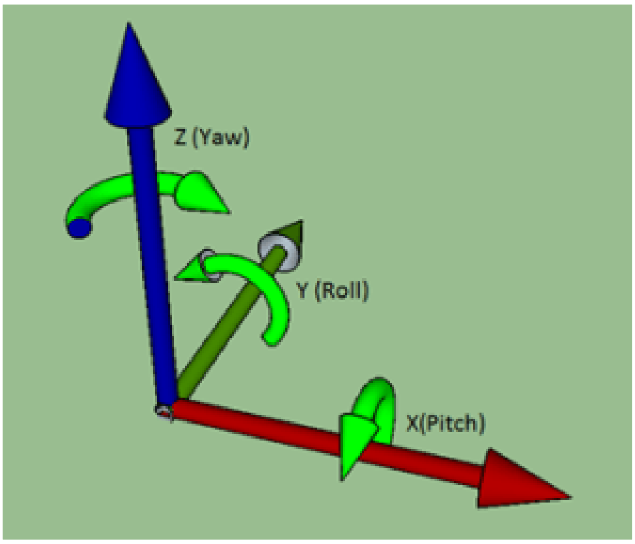
- Housing the navX-MXP circuit board in some form of protective enclosure is highly recommended, to protect it from damage. This should both protect the circuit board from damage, and provide strain relief for the cables that connect to the navX-MXP circuit board.

Orientation

navX-MXP measures a total of 9 sensor axes (3 gyroscope axes, 3 accelerometer axes and 3 magnetometer axes) and fuses them into a 3-D coordinate system. In order to effectively use the values reported by navX-MXP, a few key concepts must be understood in order to correctly install navX-MXP on a robot.

3-D Coordinate System

When controlling a robot in 3 dimensions a set of 3 axes are combined into a 3-D coordinate system, as depicted below:



In the diagram above, the green rounded arrows represent Rotational motion, and the remaining arrows represent Linear motion.

Axis	Orientation	Linear motion	Rotational Motion
X (Pitch)	Left/Right	– Left / + Right	+ Tilt Backwards
Y (Roll)	Forward/Backward	+ Forward / – Backward	+ Roll Left
Z (Yaw)	Up/Down	+ Up / – Down	+ Clockwise/ – Counter-wise

More details are available on the Terminology page.

Reference Frames

Note that the 3-axis coordinate system describes relative motion and orientation; it doesn't specify the orientation with respect to any other reference. For instance, *what does "left" mean once a robot has rotated 180 degrees?*

To address this, the concept of a [reference frame] was developed. There are three separate three-axis “reference frames” that should be understood:

Coordinate System	Technical Term	X Axis	Y Axis
Field	World Frame	Side of Field	Front (Head) of Field
Robot	Body Frame	Side of Robot	Front (Head) of Robot
navX MXP	Board Frame	See diagram Below	See diagram below

Joysticks and Reference Frames

Since a three-axis joystick is typically used to control a robot, the robot designer must select upon which Reference Frame the driver joystick is based. This selection of Reference Frame typically depends upon the drive mode used:

Drive mode	Reference Frame	Coordinate Orientation
Standard Drive	Body Frame	Forward always points to the front (head) of the robot
[Field-oriented Drive]	World Frame	Forward always points to the front (head) of the field

navX-MXP Board Orientation (Board Frame)

Aligning Board Frame and Body Frame

In order for the navX-MXP sensor readings to be easily usable by a robot control application, the navX-MXP Coordinate System (Board Frame) must be aligned with the Robot Coordinate system (Body Frame).

Aligning the Yaw (Z) axis and Gravity

The navX-MXP motion processor takes advantage of the fact that gravity can be measured with its onboard accelerometers, fusing this information with the onboard gyroscopes to yield a very accurate yaw reading with a low rate of drift. In order to accomplish this, ***the yaw (Z) axis must be aligned with the "gravity axis" (the axis that points directly up and down with respect to the earth).***

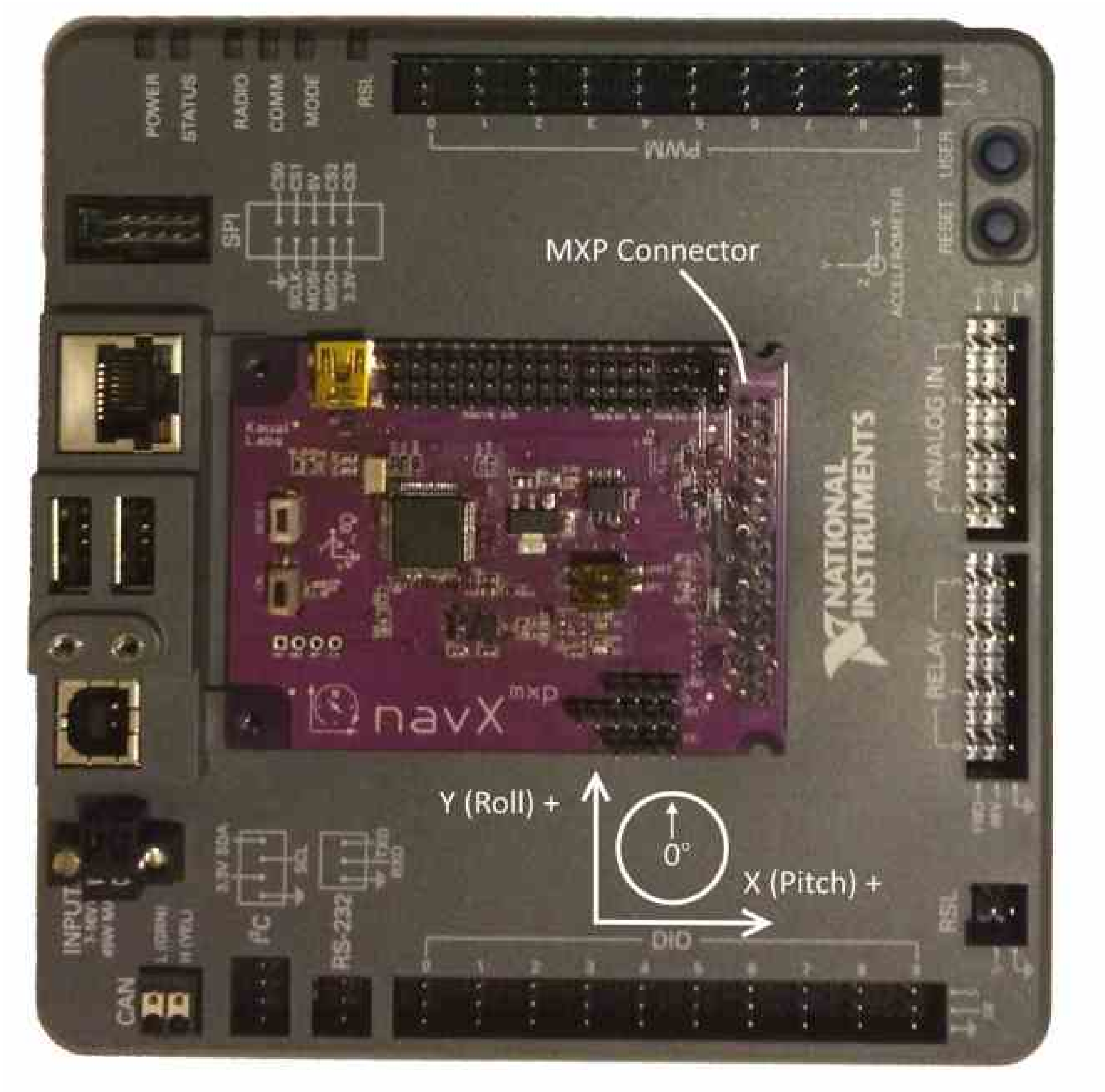
When installing navX-MXP on a robot, the navX-MXP yaw (Z) axis and the gravity axis must be aligned.

Default navX-MXP Board Orientation

The default navX-MXP circuit board orientation is with the navX-MXP logo on the *Rear Left, with the top of the circuit board pointing up (with respect to the earth)*.

Since Body Frame and Board Frame coordinates should be aligned, and because the Yaw axis must be aligned with gravity, by default you must orient the navX-MXP with the top of the board facing up, and with the Y axis (on the circuit board) pointing to the front of the robot.

If you need to mount the navX-MXP circuit board in a different orientation (vertically, horizontally, or upside down), you can use the new* OmniMount *feature to transform the orientation.

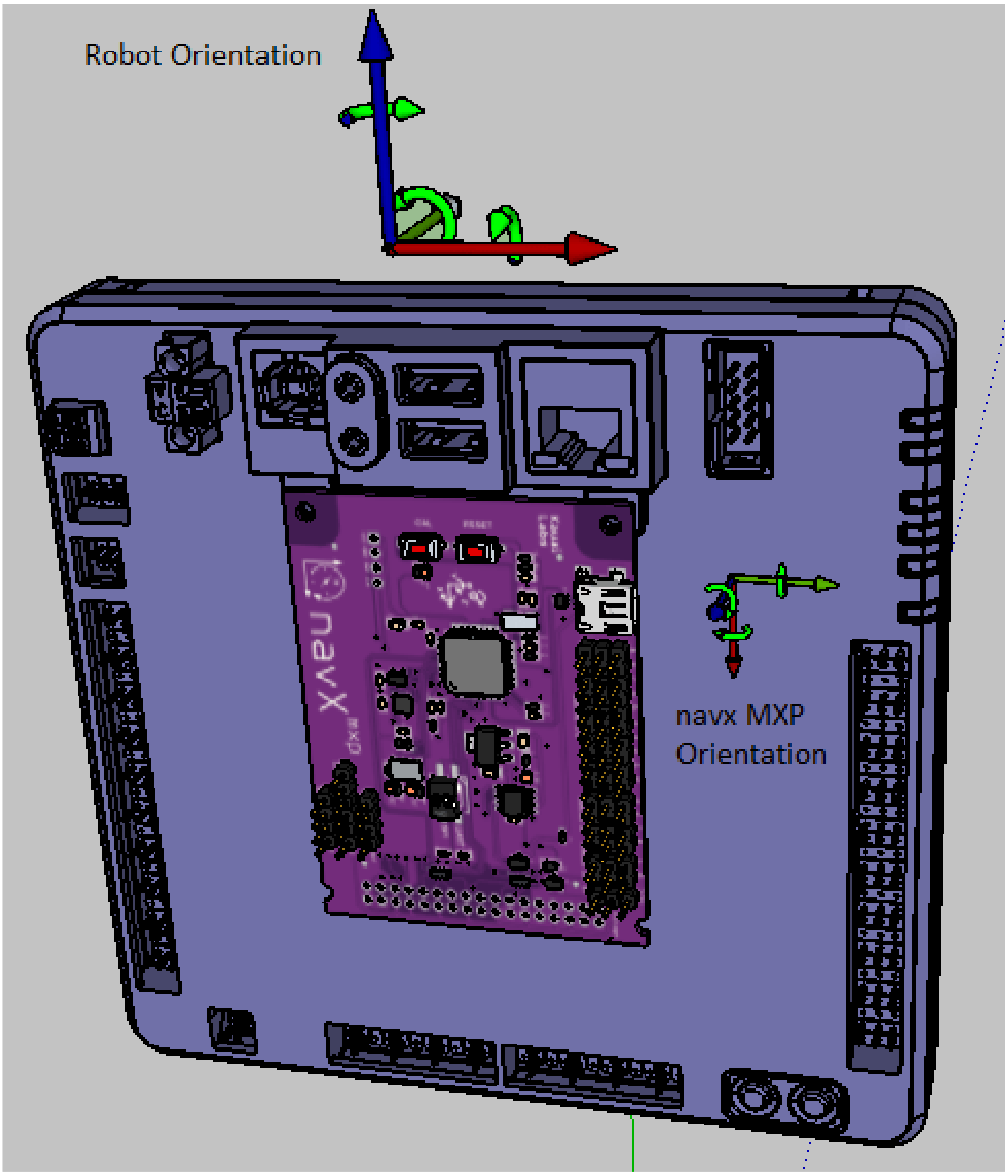

MXP Connector
Y (Roll) +
0°
X (Pitch) +
navX mxp
NATIONAL INSTRUMENTS
PWM
DIO
ANALOG IN
RELAY
SPI
POWER
STATUS
RADIO
COMM
MODE
RSL
USER
RESET
INPUT
CAN
RS-232

OmniMount

If the navX-MXP default yaw axis orientation isn't sufficient for your needs, you can use the **OmniMount** feature to re-configure the navX-MXP yaw axis, allowing high-accuracy yaw axis readings when navX-MXP is mounted horizontally, vertically, or even upside down.

Important Note: The OmniMount feature requires navX-MXP firmware version 1.1 or higher, so you may need to update your navX-MXP firmware***.***

In certain cases, the navX-MXP axes (Board Frame) may not be oriented exactly as that of the Robot (Body Frame). For instance, if the navX-MXP circuit board is plugged directly into the RoboRIO-MXP connector, and the top of the RoboRIO (the edge on which the USB connectors are mounted) is pointing up with the top side of the RoboRio pointing towards the front of the robot, the navX-MXP axes will not be oriented identically to the Robot. This configuration is shown in the diagram below:



Transforming navX-MXP Board Frame to Body Frame with OmniMount

The navX-MXP's "OmniMount" feature can transform the navX-MXP X, Y and Z axis sensor data (Board Frame) into Robot Orientation (Body Frame) by detecting which of its three axes is perpendicular to the earth's surface. [NOTE: this requires navX-MXP firmware revision 1.1.37 or later].

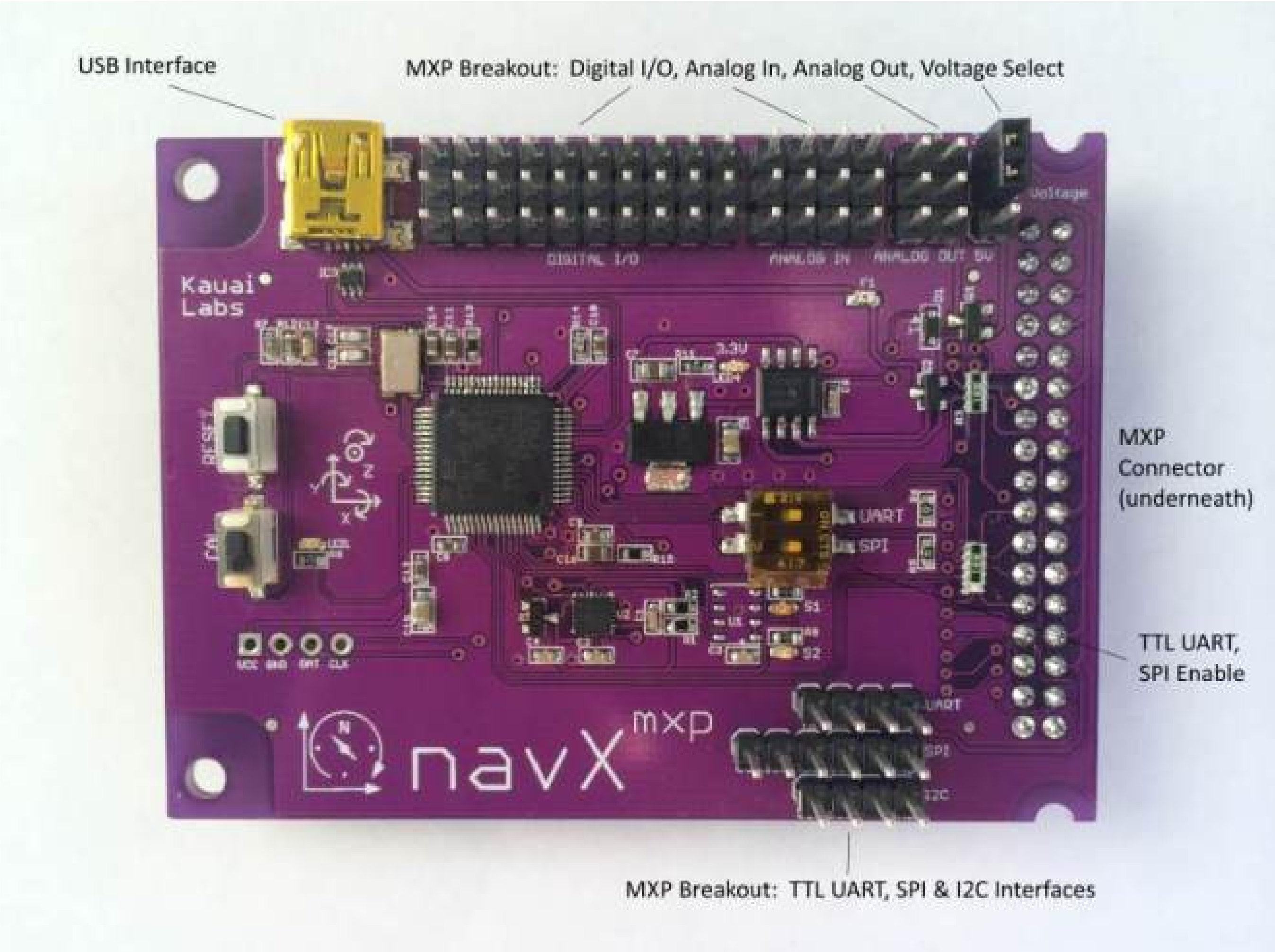
This is similar to how a modern smart phone will rotate the display based upon the phone's orientation. However unlike a smart phone, the OmniMount detection of orientation does not happen all the time – since the orientation should not change while the robot is moving. Rather, each time OmniMount configuration occurs, navX-MXP records this transformation in persistent flash memory, and will continue to perform this transformation until the transform is reconfigured.

To configure OmniMount, follow these simple steps:

- Install the navX-MXP onto your robot. ENSURE that one of the navX-MXP axes (as shown on the navX-MXP circuit board) is perpendicular to the earth's surface. This axis will become the yaw (Z) axis. Note that this axis can either be pointing away from the earth's surface, or towards the earth's surface.
- Press the 'CAL' button on the navX-MXP Circuit board AND HOLD THE BUTTON DOWN FOR AT LEAST 5 SECONDS.
- Release the 'CAL' button, and verify that the orange 'CAL' light flashes for 1 second and then turns off.
- Press the 'RESET' button on the navX-MXP circuit board, causing it to restart.
- The navX-MXP circuit board will now begin OmniMount auto-calibration. During this auto-calibration period, the orange 'CAL' LED will flash repeatedly. This process takes approximately 15 seconds, and requires two things:
    - 1. During auto-calibration, ***one of the navX-MXP axes MUST be perpendicular to the earth's surface***.
    - 2. During auto-calibration, ***navX-MXP must be held still***.
    - If either of the above conditions is not true, the 'CAL' LED will be flashing quickly, indicating an error. To resolve this error, you must ensure that conditions 1 and 2 are met, at which point the 'CAL' LED will begin flashing slowly, indicating calibration is underway.
- Once navX-MXP auto-calibration is complete, the Board Frame to Body Frame Transform will be stored persistently into navX-MXP flash memory and used until auto-calibration is run once again.

I/O Expansion

navX-MXP breaks out all usable signal pins on the National Instruments RoboRIO™ / MyRIO MXP Connector.

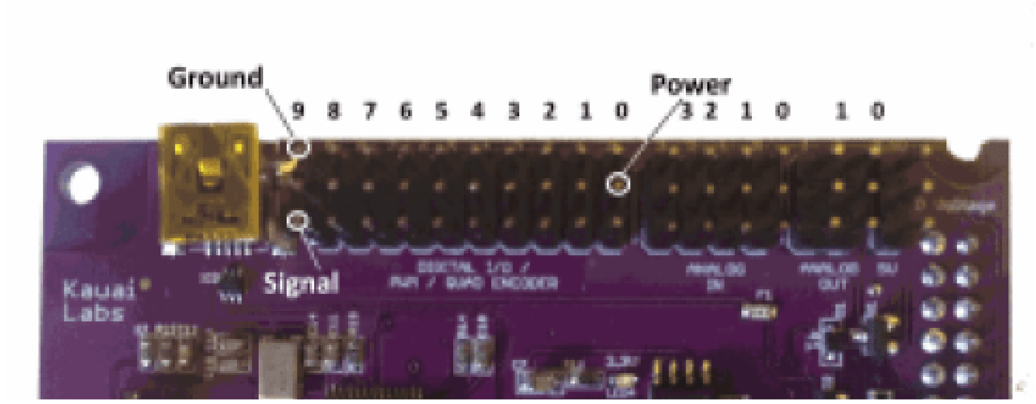

USB Interface
MXP Breakout: Digital I/O, Analog In, Analog Out, Voltage Select
MXP
Connector
(underneath)
TTL UART,
SPI Enable
MXP Breakout: TTL UART, SPI & I2C Interfaces
Kauai
Labs
navX mxp
DIGITAL I/O
ANALOG IN
ANALOG OUT
UART
SPI
I2C

DC Voltage Selection

Using the provided jumper, select the DC Voltage which will be routed to each of the connector blocks. Select from either 3.3V or 5V DC. This regulated voltage is supplied directly by the host computer (e.g., the RoboRIO).

I/O Summary

Each of the I/O pins on the MXP connector has a corresponding 3-pin connector (DC Voltage, Ground and Signal). The orientation of the Ground, Power and Signal pins for each of the Digital I/O, Analog Input and Analog Output pins is as follows:



Digital I/Os

Each of the 10 Digital I/O pins may be configured for use as either DigitalInputs or DigitalOutputs, PWM (Outputs) or Counters (Inputs).

Additionally, multiple (either 2 or 3) DigitalInputs may be used to form an QuadratureEncoder input.

DigitalInput/DigitalOutput Addressing

navX-MXP Port	MXP Pin Number	RoboRIO Channel Address
0	DIO0	10
1	DIO1	11
2	DIO2	12
3	DIO3	13
4	DIO8	18
5	DIO9	19
6	DIO10	20
7	DIO11	21

8	DIO12	22
9	DIO13	23

PWM Output Addressing

navX-MXP Port	MXP Pin Number	RoboRIO Channel Address
0	PWM0	10
1	PWM1	11
2	PWM2	12
3	PWM3	13
4	PWM4	14
5	PWM5	15
6	PWM6	16
7	PWM7	17
8	PWM8	18
9	PWM9	19

For further documentation, please see the on this topic.

NOTE: The MXP connector has 2 Digital I/O pins which are dedicated to the I2C interface. MXP Digital I/O Pin DIO14 is used for I2C SCL and DIO15 is used for I2C SDA. Since the navX MXP I2C interface is always active, these pins must not be used for any other purpose.

Analog Inputs

Each of the 4 Analog Input pins on the MXP connector has a corresponding 3-pin connector (DC Voltage, Ground and Signal). On the RoboRIO, these signals are routed to the internal Analog-to-Digital Converters (ADCs).

Analog Input Addressing

navX-MXP Port	MXP-Pin Number	RoboRIO Channel Address
0***	AI0	4
1***	AI1	5
2	AI2	6
3	AI3	7

Analog Outputs

Each of the 2 Analog Output pins on the MXP connector has a corresponding 3-pin connector (DC Voltage, Ground and Signal). On the RoboRIO, these signals are generated by the host computer’s internal Digital-to-Analog Converters (DACs).

Analog Output Addressing

navX-MXP Port	MXP Pin Number	RoboRIO Channel Address
0	AO0	0
1	AO1	1

I2C

The nav-MXP I2C connector can be used to connect the RoboRIO to an external I2C Device. The RoboRIO functions as an I2C Master. The connector provides DC Voltage, Ground, Clock (SCL) and Data (SDA).

Note that this I2C connector resides on the same I2C bus which may optionally be used to communicate between the RoboRIO and the navX-MXP's onboard processor. navX-MXP uses I2C Address 50 (0x32), so be sure that any external I2C device does not use this address.

SPI

The navX MXP SPI connector can be used to connect the RoboRIO to an external SPI device. The RoboRIO functions as an SPI Master. The connector provides DC Votage, Ground, Clock (SCK), Slave Select (SS), Master-in/Slave-out (MISO) and Master-out/Slave-in (MOSI) signals.

Note that this SPI connector resides on the same SPI bus which may optionally be used to communicate between the RoboRIO and the navX-MXP's onboard processor. navX-MXP will respond to the Slave Select signal if and only if the SPI Enable dip switch is set to the "ON" position. Thus, the SPI Enable dip switch should be set to the "OFF" position if you wish to communicate with an external device via the SPI connector.

TTL UART

The navX-MXP TTL UART connector can be used to connect the RoboRIO to an external TTL-level UART device.

NOTE: The TTL UART connector cannot be used to connect to an external RS-232 signal, since RS-232 voltages are much higher than TTL-level UART voltages. Connecting a higher-voltage RS-232 device to the TTL UART connector may subject the RoboRIO to damaging voltage levels on these pins.

Note that this TTL UART connector can be used to communicate between the RoboRIO and the navX-MXP's onboard processor (in fact, this is the default). navX-MXP will respond to the UART TX signal from the RoboRIO if and only if the UART Enable dip switch is set to the "ON" position. Thus, the UART Enable dip switch should be set to the "OFF" position if you wish to communicate with an external device via the TTL UART connector.

Alternative Installation Options

In addition to Plug-n-Play installation on the National Instruments RoboRIO™, navX-MXP's flexible design accommodates several additional installation options.

One-wire Connect via "Floppy-disk" extension cable

If mounting the navX-MXP circuit board directly into the RoboRIO's onboard MXP connector is not possible, a "Floppy-disk" extension cable can be used to place the navX-MXP circuit board up to a few feet away from the RoboRIO. This installation method supports the I/O expansion capabilities, since all MXP connector signals are carried over the extension cable.

Note that higher-speed signals such as those found on the SPI and I2C bus, and noise-sensitive analog signals like those on the Analog Input and Output pins may be negatively impacted by longer distances and electro-magnetic interference, so high quality shielded cabling and shorter distances may be called for.

These extension cables are available online at AndyMark:

Image not found

One-wire Connect via USB cable

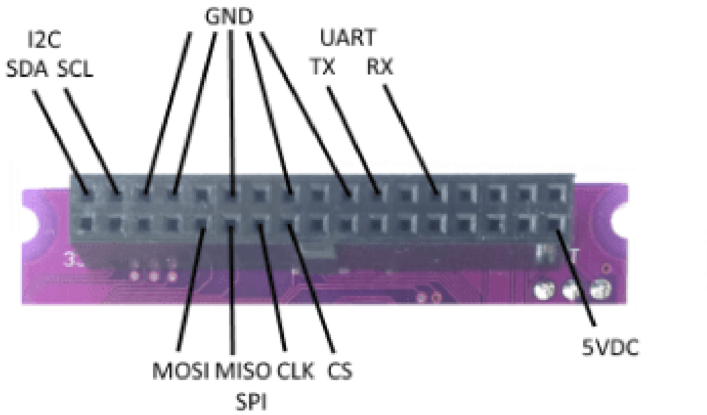
By using a USB Mini-B type (Male) to USB A type (Male) connector, navX-MXP can receive both power and also communicate with the RoboRIO.

This installation method allows the navX-MXP circuit board to be placed longer distances away from the RoboRIO than via the "Floppy-disk" extension cable method. However, this installation method does support the I/O expansion capabilities, since the MXP connector signals are not routed over the USB cable.

Low-level Connect via Power and Signal pins on MXP

Connector

If any of the “one-wire” methods described above is not desirable, you may also interface to the navX-MXP circuit board using the Power, Ground and I2C/SPI/TTL UART signals on the MXP Connector.



I2C

To use the I2C interface without directly plugging the navX-MXP circuit board into the RoboRIO MXP connector, first ensure that the navX-MXP circuit board has power (either via the USB connector, or via the +5VDC pin on the MXP connector).

Next, make sure that the digital ground from the host computer (e.g., the RoboRIO) is connected to one of the GND pins on the MXP connector.

Finally, connect the SDA and SCL pins on the host computer (e.g., the RoboRIO) to the corresponding SDA and SCL pins on the navX-MXP circuit board.

Note that the I2C bus expects that the SDA and SCL pins be pulled up with a pull-up resistor on each line. The RoboRIO internally pulls these lines high.

The I2C pins are 5V tolerant, so the host computer can use either 5V or 3.3V DC levels on these pins.

SPI

To use the SPI interface without directly plugging the navX-MXP circuit board into the RoboRIO MXP connector, first ensure that the navX-MXP circuit board has power (either via the USB connector, or via the +5VDC pin on the MXP connector).

Next, make sure that the digital ground from the host computer (e.g., the RoboRIO) is connected to one of the GND pins on the MXP connector.

Finally, connect the CS, CLK, MISO and MOSI pins on the host computer (e.g., the RoboRIO) to the corresponding CS, CLK, MISO and MOSI pins on the navX-MXP circuit board.

The SPI pins are 5V tolerant, so the host computer can use either 5V or 3.3V DC levels on these pins.

Creating an Enclosure

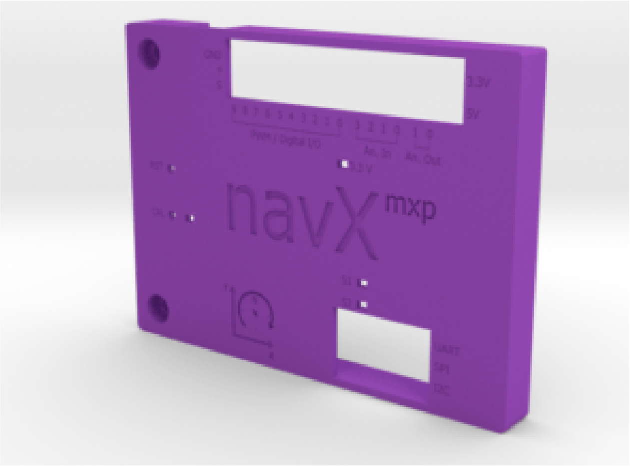
The navX-MXP circuit board contains sensitive circuitry, and should be handled carefully.

An enclosure is recommended to protect the navX-MXP circuit board from excessive handling, “swarf”, electro-static discharge (ESD) and other elements that could potentially damage the navX-MXP circuitry. The enclosure can also help prevent accidental shorts to ground which may occur on the MXP Expansion I/O pins.

Build vs. Buy

Those who prefer to print the enclosure using their personal 3D printer, an enclosure design file (in STL format) is available in the “enclosure” directory of the .

Those who prefer to purchase the enclosure can order it from (which takes approximately 2 weeks to deliver), or at the Kauai Labs store if you’re in a hurry. The price including shipping will be approximately $20, depending upon the type of material used.



Design Files

The enclosure design files include:

- navx-mxp.skp: Sketchup 3D Design File for the navX-MXP circuit board
- navx-mxp-roborio-lid_v4.skp: Sketchup 3D Design File for a lid-style enclosure for the

navX-MXP circuit board. Note that the design file scale is 1000X actual size, so will need to be scaled down by a factor of 1000 before printing.

- navx-mxp-roborio-lid_v4_scaleddown.stl: STL Format File for 3d printing the lid-style enclosure for the navX-MXP circuit board. This file contents have been scaled to their actual size.

Printing and Customizing the Enclosure

The Sketchup (.skp) files can be edited using . Then, the files can be exported to a STL format using the Sketchup STL Import/Export extension. Finally, these exported STL format files can be opened and 3d-printed using .

Securing the Enclosure

The Lid Enclosure can be secured to the RoboRIO by two #4-40 3/8? screws. This will secure not only the Lid, but will also secure the navX-MXP circuit board.

Software
Software

navX-MXP includes software which makes navX-MXP easier to understand, integrate and use with FIRST FRC and FTC robots than other navigation technologies and products available today. This software includes the following components:

- FRC RoboRIO Libraries for accessing navX-MXP from a National Instruments RoboRIO™-based robot
- An FTC Android Library for accessing navX-MXP from an Android-based FTC Robot Control Application.
- Libraries for accessing navX-MXP from Linux and Arduino.
- The navXUI, which demonstrates navX-MXP capabilities

For advanced users, several calibration/configuration tools are also available.

Note: For developers on Linux and Mac OS platforms, the latest non-Windows RoboRIO (FRC)/Android (FTC) libraries build is also available. Please note that this build does not contain any of the navX-MXP tools, but does contain the RoboRIO and Android libraries.

RoboRIO Libraries

navX-MXP libraries for use with the RoboRIO Libraries from WPI are available in each of the languages/development environments commonly used to development FIRST FRC robot applications:

- Java
- C++
- LabVIEW navX-AE

These libraries provide access to navX-MXP via SPI, I2C and USB and UART – as well as USB and I2C interfaces to navX-Micro, and USB Interfaces to VMX-pi.

*[**Update: 2/2/2019 – Version 3.1.366 is now available – which is compatible with the 2019 FRC Release (2019.1.1). For more details on installation, see the page corresponding to your chosen development language. This build also contains a firmware update recommended if you are using the navX-MXP USB interface.**]*

Android Library (FTC)

NOTE: The 2016 Version of the navX-Micro Android Library for FTC is tested with the current 2017 version of the "ftc_app" library for use with the Modern Robotics Core Device Interface Module.

The navx_ftc Android software library supports access to navX-Model devices via the I2C communication interface. Several example programs are provided, demonstrating how to use a navX-Model device in a FTC-based robot control application.

To use the library, you can download the of the libraries, or you can checkout the source code with Git. To learn more about the library, online help is available.

Getting Started

Before getting started, ensure you have installed Android Studio and the latest FTC Robot Controller Application ("ftc_app") package.

Several sample Java Robot Applications are provided. After running the setup program included in the latest build, the libraries and samples will be installed to the following location:

<HomeDirectory>\navx-micro\android

For example, if your user name is Robot, the directory name will be C:\Users\Robot\navx-micro\android.

Within this directory, the "examples" sub-directory contains several example programs. Select the example you wish to start with and copy it into your project as follows:

- Copy one or more of the example navX-Model "op modes" files from the <HomeDirectory>\navx-micro\android\examples directory into your project's

"TeamCode" top-level directory. (i.e., org.firstinspires.ftc.teamcode).

Next, several configuration changes must be made in order that the Android Studio ftc_app-based project can locate the navx_ftc library:

- Modify the op mode example file to change the following line near the top of the file to match the I2C port on the Core Device Interface Module to which you have connected the navX-Model device:

  ```
  private final int NAVX_DIM_I2C_PORT = 0; /* See the installation
  page for details on port numbering. */
  ```

- Modify your robot application's (the "TeamCode" project) build.release.gradle file repository list to add a reference the directory where the navx_ftc library is installed:

  ```
  repositories {
      flatDir {
          dirs 'libs', 'C:\\Users\\Robot\\navx-
  micro\\android\\libs'
      }
  }
  ```

- Again in the same build.release.gradle file, add the navx_ftc library to the list of libraries the ftc_app will link to – by adding this line near the bottom of the gradle build file, in the dependencies section:

  ```
  dependencies {
    ...
    compile (name:'navx_ftc-release', ext:'aar')
  }
  ```

Linux Library

A library for accessing navX-MXP (and navX-Micro) from Linux is available. This library was developed by Alexander Allen of FRC Team 900 (Zebracorns) and supports the USB interface.

The navX-MXP Linux Library is useful for integrating with video processors such as the Raspberry-PI and the Jetson TK1 and TX1.

To use the library, you can the source code with Git. Online help is also available.

Getting Started

After checking out the source code with Git into a directory on your Linux OS, compile the library using CMake.

The file Timestamp.cpp demonstrates how to integrate the library into your application; you will need to identify the Linux serial port name to use, as follows:

```
AHRS ahrs = AHRS("/dev/ttyACM0");
```

Sensor data values can be retrieved after the completion of the AHRS constructor.

Arduino Library

A library for accessing navX-MXP (and navX-Micro) from Arduino is available. This library supports the I2c and SPI interfaces.

The navX-MXP Arduino Library is useful for integrating navX-MXP into Arduino-based project.

To use the library, you can the source code with Git.

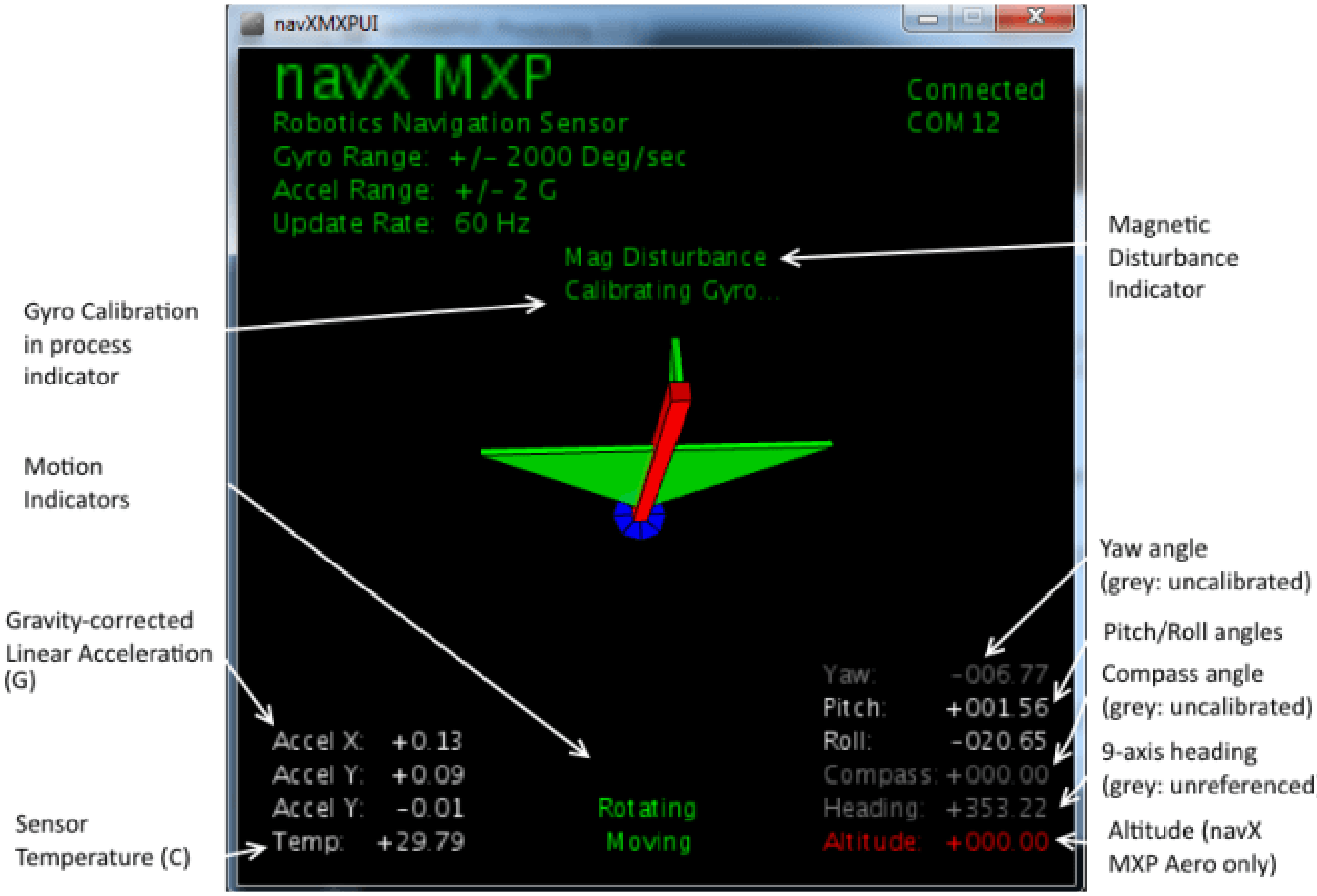
Getting Started

After checking out the source code with Git into a directory on your computer, compile using the Arduino IDE.

The file navXTestJig.ino demonstrates how to integrate the library into your application. The setup() and loop() functions in this file demonstrate how to initialize and communicate with the sensor.

navXUI

The navXUI user interface application provides a simple way to visualize the data provided by the navX-MXP.



Gyro Calibration in Progress Indicator

The Gyro Calibration in Progress Indicator is displayed during initial gyroscope calibration, which occurs immediately after power is applied to the navX-MXP. If the gyroscope calibration does not complete, the navX-MXP yaw accuracy will be adversely impacted. For more information on Gyro Calibration, please see the Gyro/Accelerometer Calibration page.

Motion Indicators

The navX-MXP provides dynamic motion indicators: (a) the “Moving” indicator and (b) the

"Rotating" indicator.

The Moving indicator is present whenever the current Gravity-corrected Linear Acceleration exceeds the "Motion Threshold".

The Rotating indicator is present whenever the change in yaw value within the last second exceeds the "Rotating Threshold". Note that the navX-MXP Gyroscope Calibration only occurs when the navX-MXP is not Rotating for a few seconds.

Gravity-corrected Linear Acceleration (G)

The navX-MXP automatically subtracts acceleration due to gravity from accelerometer data, and displays the resulting linear acceleration. These measures are in units of instantaneous G, and are in World Reference Frame.

Sensor Temperature

The Sensor Temperature indicates the die temperature of the MPU-9250 IC. Since shifts in gyro temperature can impact yaw accuracy, navX-MXP will automatically perform Gyroscope calibration whenever navX-MXP is still. See the Gyro/Accelerometer Calibration page for more details.

Magnetic Disturbance Indicator

Once the navX-MXP Magnetometer has been calibrated (see the Magnetometer Calibration page), whenever the current magnetic field diverges from the calibrated value for the earth's magnetic field, a magnetic disturbance is indicated.

Yaw Angle

The Yaw Angle is displayed in grey text if Gyro Calibration has not yet been completed. Once Gyro Calibration is complete, the Yaw Angle text color will change to white.

Pitch/Roll Angles

The Pitch/Roll Angles are always displayed in white text, since Accelerometer calibration occurs at the Kauai Labs factory.

Compass Angle

The Compass Angle displays the tilt-compensated compass heading calculated from the navX-MXP's Magnetometer combined with the tip/tilt measure from the Accelerometers.

The Compass Angle is displayed in grey text if Magnetometer Calibration has not yet been completed. Once Magnetometer Calibration is complete, the Compass Angle text color will change to white.

9-axis ("Fused") Heading

The 9-axis heading is displayed in grey text if Magnetometer Calibration has not yet been completed and/or if no undisturbed magnetic readings have occurred.

Altitude

The Altitude displays the navX-MXP's calculated current altitude, based upon the reading from the pressure sensor, the current temperature and the sea-level pressure.

The Altitude is displayed in red text if a Pressure Sensor is not installed. Pressure Sensors are only installed on the navX-MXP Aero. Valid altitude readings are displayed in white text.

Installing/Running the navXUI

- To run the navXUI, the navX-MXP must be connected to a PC running Windows via USB.
- Make sure Java 7 (version 1.7) or Java 8 (version 1.8) is installed on your computer. The 64-bit version of Java is recommended. To tell which version of java is currently "Active", open up a command window, and type this command:

```
java -version
```

- Download the and unzip the contents to your local computer.
- Run the setup.exe program, which will install the navXUI, as well as all necessary device drivers for communicating over USB with the navX-MXP, as well as some additional tools.
- Start the navXUI:

From your Start Menu, select "Kauai Labs" and then "navX-MXP" and click on the "navXUI" icon to start the navXUI.

If your computer has more than one serial port, you can select which serial port to use by clicking on the up/down arrows in the COM port selection control in the UI.

Tools

navX-MXP includes several tools for magnetometer calibration and advanced configuration. These tools run on a Windows PC and communicate via USB to navX-MXP.

NOTE: These tools are provided for use by advanced users; please carefully read the tool descriptions before using them.

Examples
Examples

Example source code for various navX-MXP (and navX-Micro) capabilities are available for both for FRC and FTC Robotics Control Systems.

FRC Examples

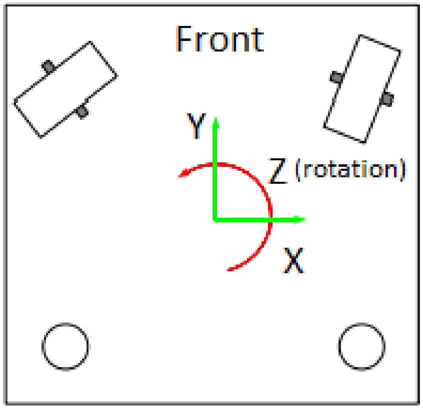
This section provides example code for several common navX-MXP (and navX-Micro) applications used by FIRST FRC teams on their robots to add sophisticated navigation capabilities. These examples are in Java, C++ and LabVIEW.

- Java/C++
  - When you run the setup program contained in the latest build, Java/C++ examples will be installed to subdirectories underneath \navx-mxp\\examples (e.g., **C:\Users\Robot**\navx-mxp**cpp**\examples).
- LabVIEW
  - When you run the setup program contained in the latest build, LabVIEW examples are installed at:
    - \vi.lib\Rock Robotics\WPI\ThirdParty\Sensors\navX
  - The LabVIEW Install Directory is typically C:\Program Files (x86)\National Instruments\LabVIEW 2016.

FTC Examples

If you are looking for FTC examples, please see the navX-Micro Examples.

Field-Oriented Drive (FRC)



An easy-to-use, highly-maneuverable drive system is at the heart of a successful robot. Omnidirectional drive systems provide motion in the Y axis (forward-backward), X-axis (strafe), and Z axis (rotating about it's center axis). Each "degree of freedom" is independent, meaning that the overall robot motion is comprised of a "mix" of motion in each of the X, Y and Z axes, control of which is easily provided with a 3-degree of freedom joystick. This resulting maneuverability is quite useful during FRC competitions to avoid other robots, pick up and place game pieces, line up for shooting to a target, etc.

Yet the driver who remains in a fixed position is now presented a new challenge: *when the driving joystick is pushed forward, the robot does not necessarily move forward with respect to the field – rather it moves forward with respect to the robot*. This forces the driver to develop the skill of "placing their head in the robot" and performing the angular transformation mentally. This skill can take quite awhile to develop meaning that rookie drivers face an uphill climb before they can be productive team contributors. Additionally, the mental energy involved in field-to-robot rotational transformations reduces the driver's cognitive ability to focus other game-related tactical tasks, as evidenced by drivers who are so intently focused on driving that their response to their teammates is diminished. Moreover, when the driver does not have a clear line of sight to the robot, the "head in the robot" becomes even more challenging.

Solving this challenge is conceptually straightforward. First, the current angle (?) of rotation between the head of the field, and the head of the robot must be measured; secondly, the joystick X/Y coordinates are transformed by ?, as shown in following pseudo-code:

```
double rcw = pJoystick->GetTwist();
double forwrd = pJoystick->GetY() * -1; /* Invert stick Y axis */
double strafe = pJoystick->GetX();

float pi = 3.1415926;

/* Adjust Joystick X/Y inputs by navX MXP yaw angle */

double gyro_degrees = ahrs->GetYaw();
float gyro_radians = gyro_degrees * pi/180;
float temp = forwrd * cos(gyro_radians) +
 strafe * sin(gyro_radians);
strafe = -forwrd * sin(gyro_radians) +
```

```
 strafe * cos(gyro_radians);
fwd = temp;

/* At this point, Joystick X/Y (strafe/forwrd) vectors have been
*/
/* rotated by the gyro angle, and can be sent to drive system */
```

The WPI Library “MecanumDrive_Cartesian()” function and the LabView “Holonomic Drive” VI, which are used in the examples below, implement the field-centric drive algorithm. The navX-MXP “Yaw” angle is provided to these library functions to specify the amount of rotation between the robot and the field.

For more details on field-centric drive algorithms, please see this which provides a wealth of helpful, well written information on implementing field-centric drive on various types of drive systems.

FRC C++ Example

[Full C++ source code on GitHub]

FRC Java Example

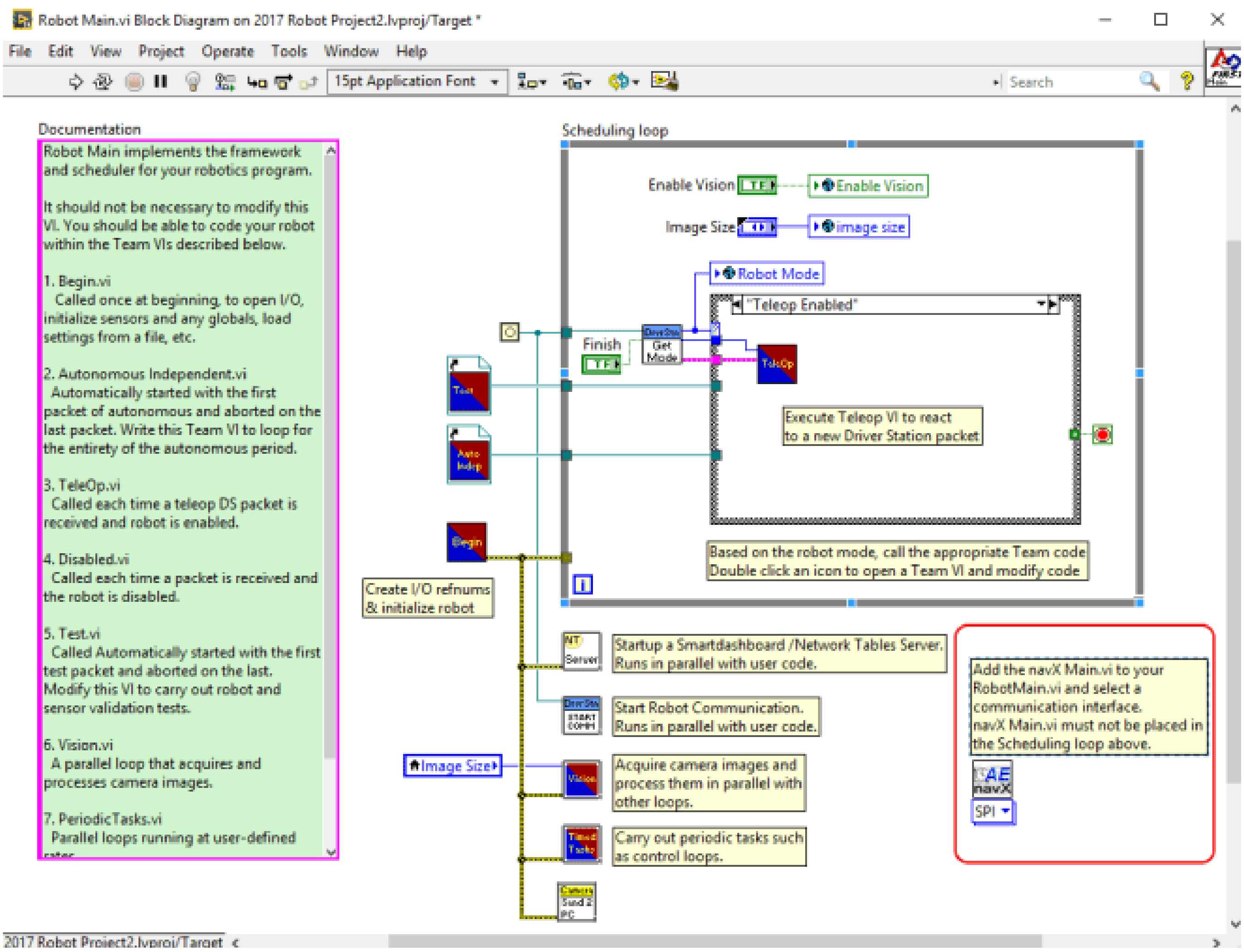
[Full Java Source code on GitHub]

FRC LabView Example

The navX-MXP FieldCentric-Drive LabView example shows how to make small modifications to the LabView “FRC RoboRIO Robot Project” using the “Mecanum Robot” configuration to implement high-accuracy Field-Centric drive.

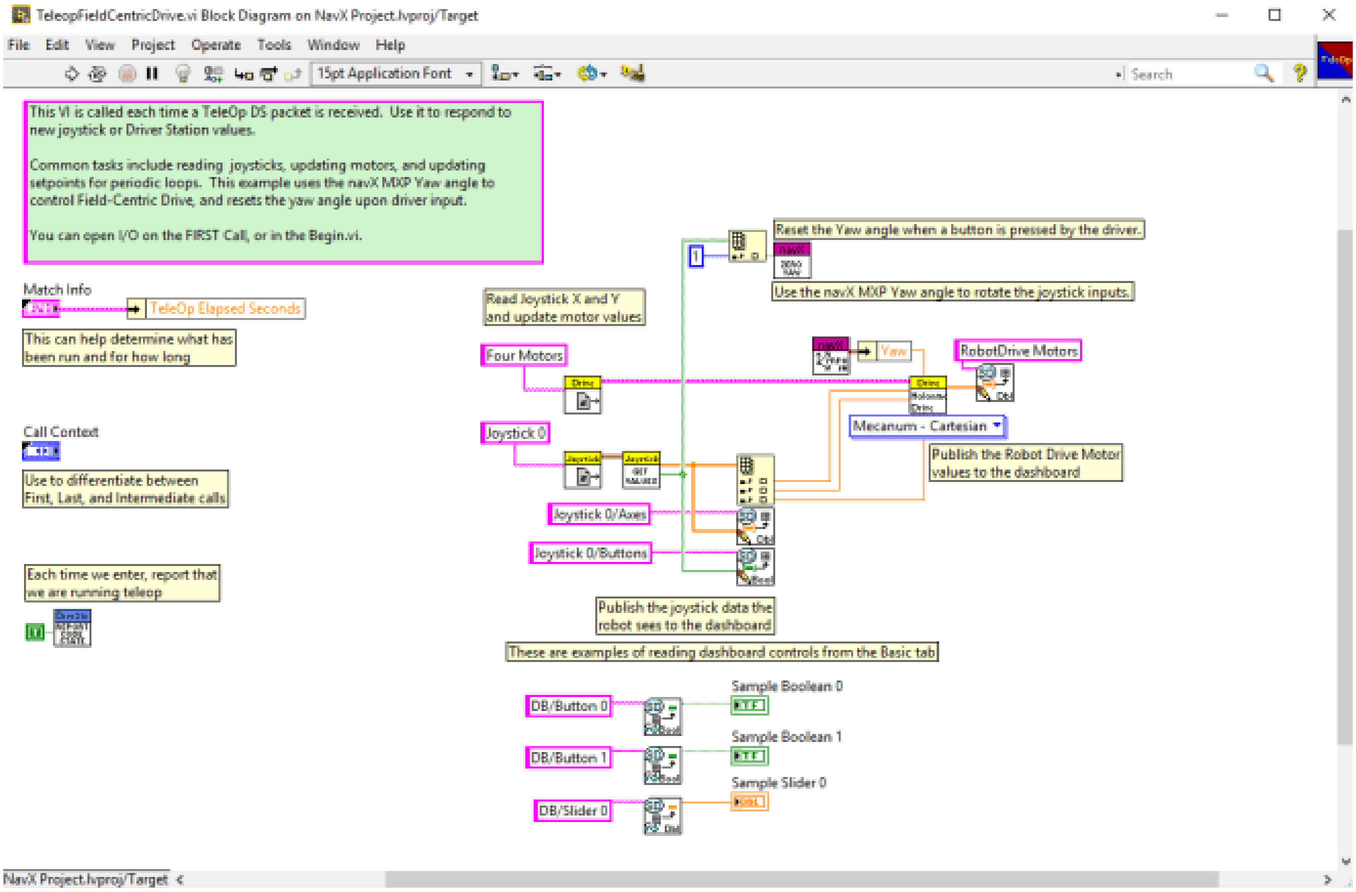
RobotMain.vi

Place the NavX main vi on the block diagram and set it up to your needs. The default sample rate is 50Hz. You may need to process faster for your situation. For the SPI, I2C and USB connections the max sample rate is 200Hz.



Teleop.vi

The Teleop.vi is modified to feed the current navX-MXP “Yaw” angle reading to the Holonomic Drive VI, which rotates the joystick X/Y coordinates by the gyro angle (and thus implements FieldCentric drive control). Additionally, if a driver joystick button is pressed, the navX-MXP “Yaw” angle is reset to zero. The navX-MXP Device TypeDef is passed to the Teleop.vi via a VI input terminal.



[Full LabVIEW Source code on Github]

Rotate to Angle (FRC)

Automatically rotating a robot to an angle using navX-MXP can be used to rotate a robot quickly and accurately to a known angle, as long as the robot drive system provides independent Z-axis rotation (the capability to “spin on a dime”). This same technique can be used to help a robot drive in a straight line.

This example code below will automatically rotate the robot to one of four angles (0, 90, 180 and 270 degrees) whenever the corresponding “rotate to preset angle” button is pressed. This rotation can occur not only when the robot is still, but also when the robot is driving. When using field-oriented control, this will cause the robot to drive in a straight line, in whatever direction is selected.

This example also includes a feature allowing the driver to “reset” the “yaw” angle. When the reset occurs, the new gyro angle will be 0 degrees. This can be useful in cases when the gyro drifts, which doesn’t typically happen during a FRC match, but can occur during long practice sessions.

The PID Controller coefficients defined in the example code will need to be tuned for your drive system.

NOTE: The examples below are for Mecanum drive systems. If you are using a tank (differential) drive system, this Java example is available.

For more details on this approach, please visit Chief Delphi, including this helpful post.

FRC C++ Example

Full C++ source code on GitHub

FRC Java Example

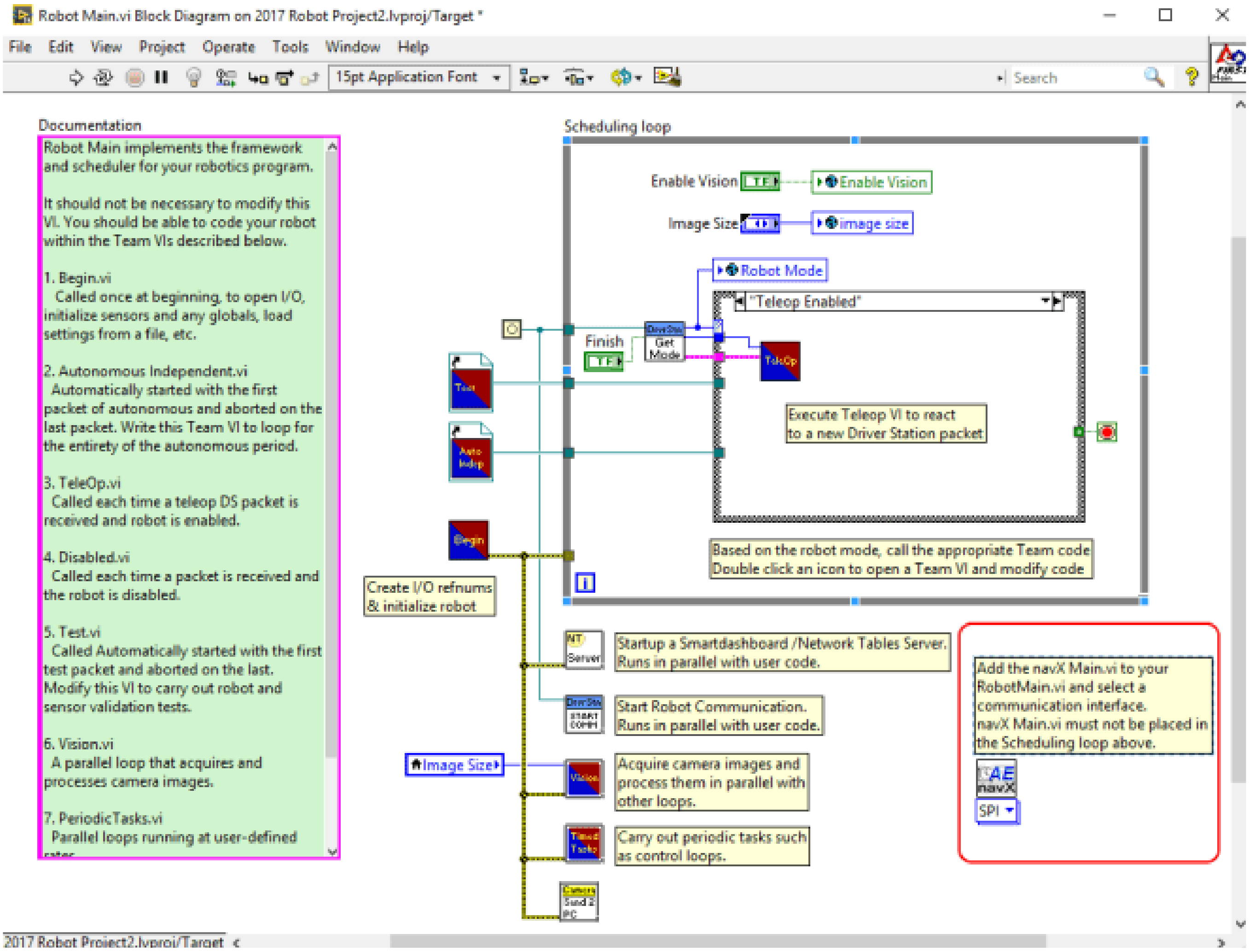
Full Java Source code on GitHub

FRC LabView Example

The navX-MXP Rotate to Angle LabView example shows how to make small modifications to the LabView “FRC RoboRIO Robot Project” using the “Mecanum Robot” configuration to rotate the robot to a given angle.

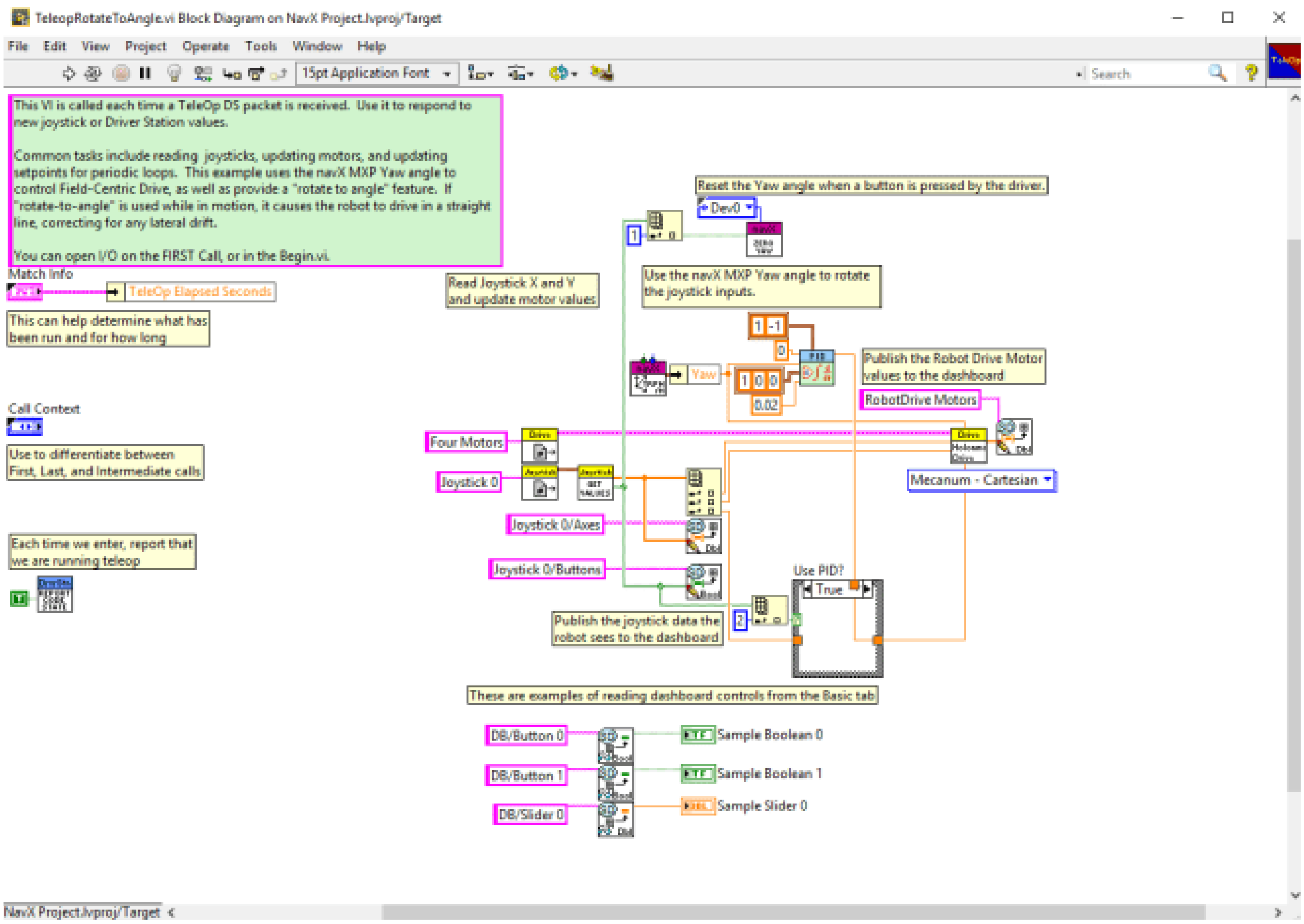
RobotMain.vi

Place the NavX main vi on the block diagram and set it up to your needs. The default sample rate is 50Hz. You may need to process faster for your situation. For the SPI, I2C and USB connections the max sample rate is 200Hz.



Teleop.vi

The Teleop.vi is modified to feed the current navX-MXP “Yaw” angle reading to the Holonomic Drive VI, which rotates the joystick X/Y coordinates by the gyro angle (and thus implements FieldCentric drive control). Additionally, if a driver joystick button is pressed, the navX-MXP “Yaw” angle is reset to zero. This example also includes a “Rotate to angle” feature, using a PID controller; note that if “Rotate to Angle is used while in motion, it causes the robot to drive in a straight line, correcting for lateral drift.



[Full LabVIEW Source code on Github](#)

Automatic Balancing (FRC)

The Automatic Balancing example demonstrates how to implement a self-balancing robot, which can be useful to help avoid a robot tipping over when driving. As an example, [FRC team 263 demonstrated the auto-balance feature effectively during the 2018 FRC Championships](#).

The basic principle used in the example is based upon measurement of the navX-MXP Pitch (rotation about the X axis) and Roll (rotation about the Y axis) angles. When these angles exceed the “off balance” threshold and until these angles fall below the “on balance” threshold, the drive system is automatically driven in the opposite direction at a magnitude proportional to the Pitch or Roll angle.

Note that this is just a starting point for automatic balancing, and will likely require a reasonable amount of tuning in order to work well with your robot. The selection of the magnitude of correction to apply to the drive motors in response to pitch/roll angle changes could be replaced by a PID controller in order to provide a tuning mechanism appropriate to the robot.

FRC C++ Example

Full C++ source code on GitHub

FRC Java Example

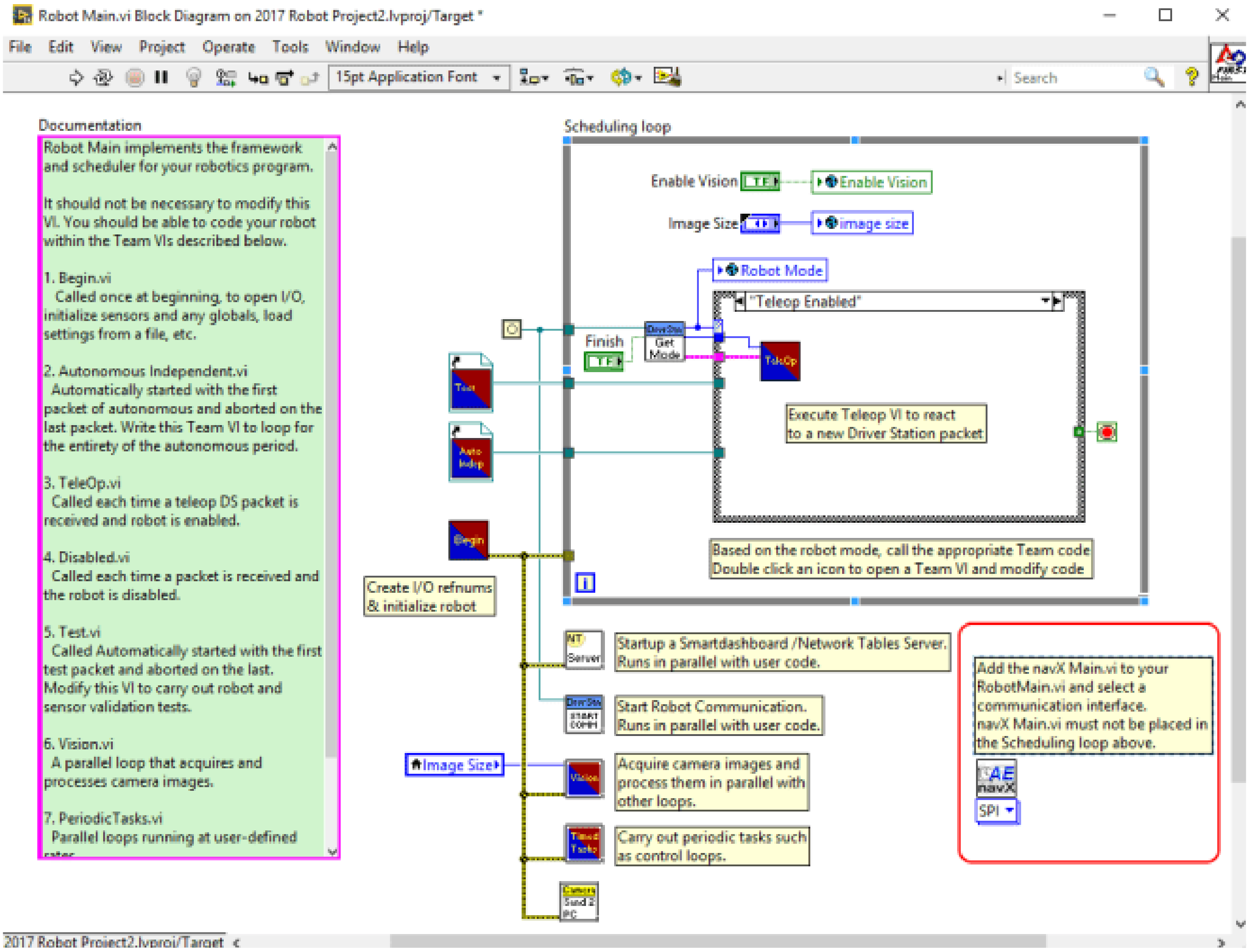
Full Java Source code on GitHub

FRC LabView Example

The navX-MXP AutoBalance LabView example shows how to make small modifications to the LabView “FRC RoboRIO Robot Project” using the “Mecanum Robot” configuration to implement high-accuracy Automatic Balancing.

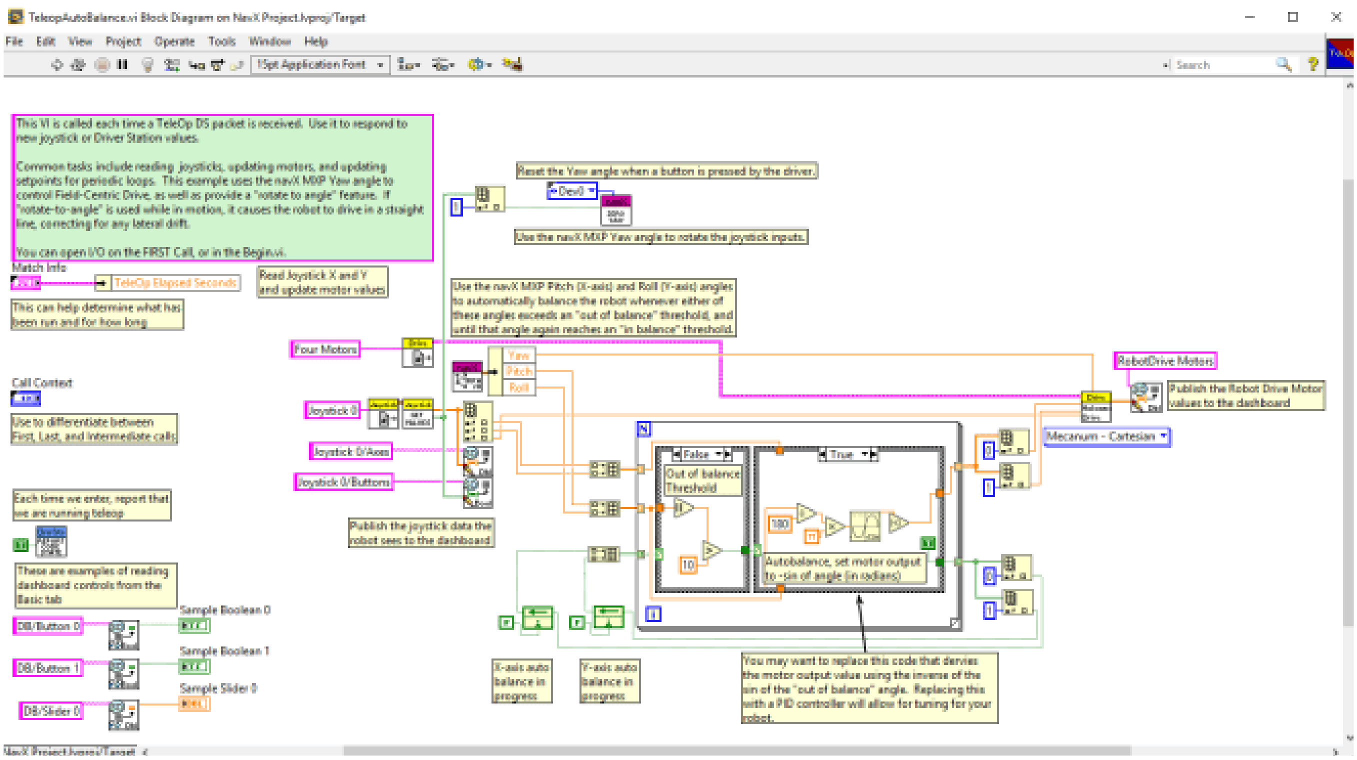
RobotMain.vi

Place the NavX main vi on the block diagram and set it up to your needs. The default sample rate is 50Hz. You may need to process faster for your situation. For the SPI, I2C and USB connections the max sample rate is 200Hz.



Teleop.vi

The Teleop.vi is modified to feed the current navX-MXP "Yaw" angle reading to the Holonomic Drive VI, which rotates the joystick X/Y coordinates by the gyro angle (and thus implements FieldCentric drive control). Additionally, if a driver joystick button is pressed, the navX-MXP "Yaw" angle is reset to zero. Finally, the navX-MXP "Pitch" (X-axis) and "Roll" (Y-axis) angles are continuously compared to a "out of balance" threshold, at which point the corresponding axis motor output value is derived from the inverse of the sin of that angle, until the time when that same angle falls below the "in balance" threshold.

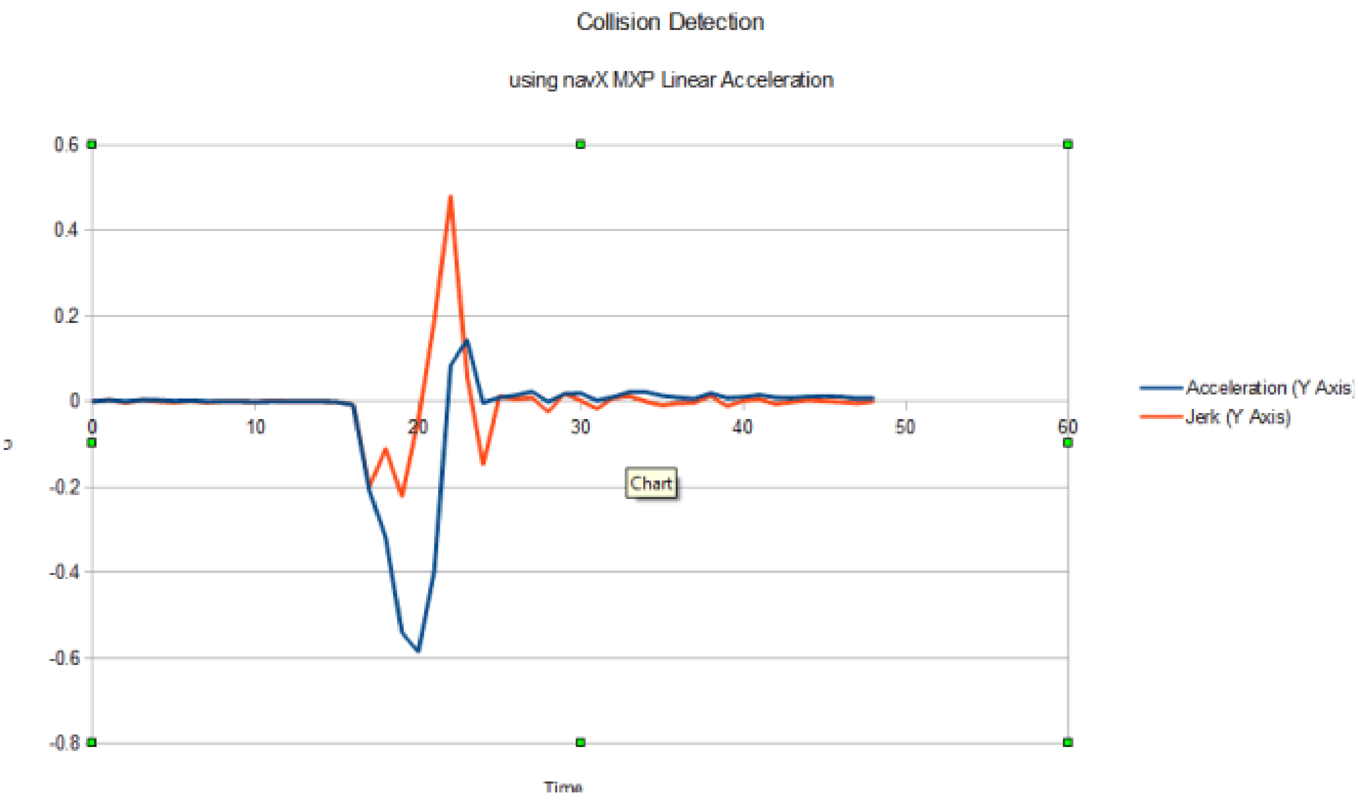


[Full LabVIEW Source code on Github]

Collision Detection (FRC)

Collision Detection is commonly used in automobiles to trigger airbag deployment, which can reduce the force of an impact and save lives during an accident. A similar technique can be used on a robot to detect when it has collided with another object.

The principle used within the Collision Detection example is the calculation of Jerk (which is defined as the change in acceleration). As shown in the graph below (taken from navX-MXP data recorded in LabVIEW of a small collision), whenever the jerk (in units of G) exceeds a threshold, a collision has occurred.



In the sample code shown below, both the X axis and the Y axis jerk are calculated, and if either exceeds a threshold, then a collision has occurred.

The “collision threshold” used in these samples will likely need to be tuned for your robot, since the amount of jerk which constitutes a collision will be dependent upon the robot mass and expected maximum velocity of either the robot, or any object which may strike the robot.

FRC C++ Example

Full C++ Source Code

FRC Java Example

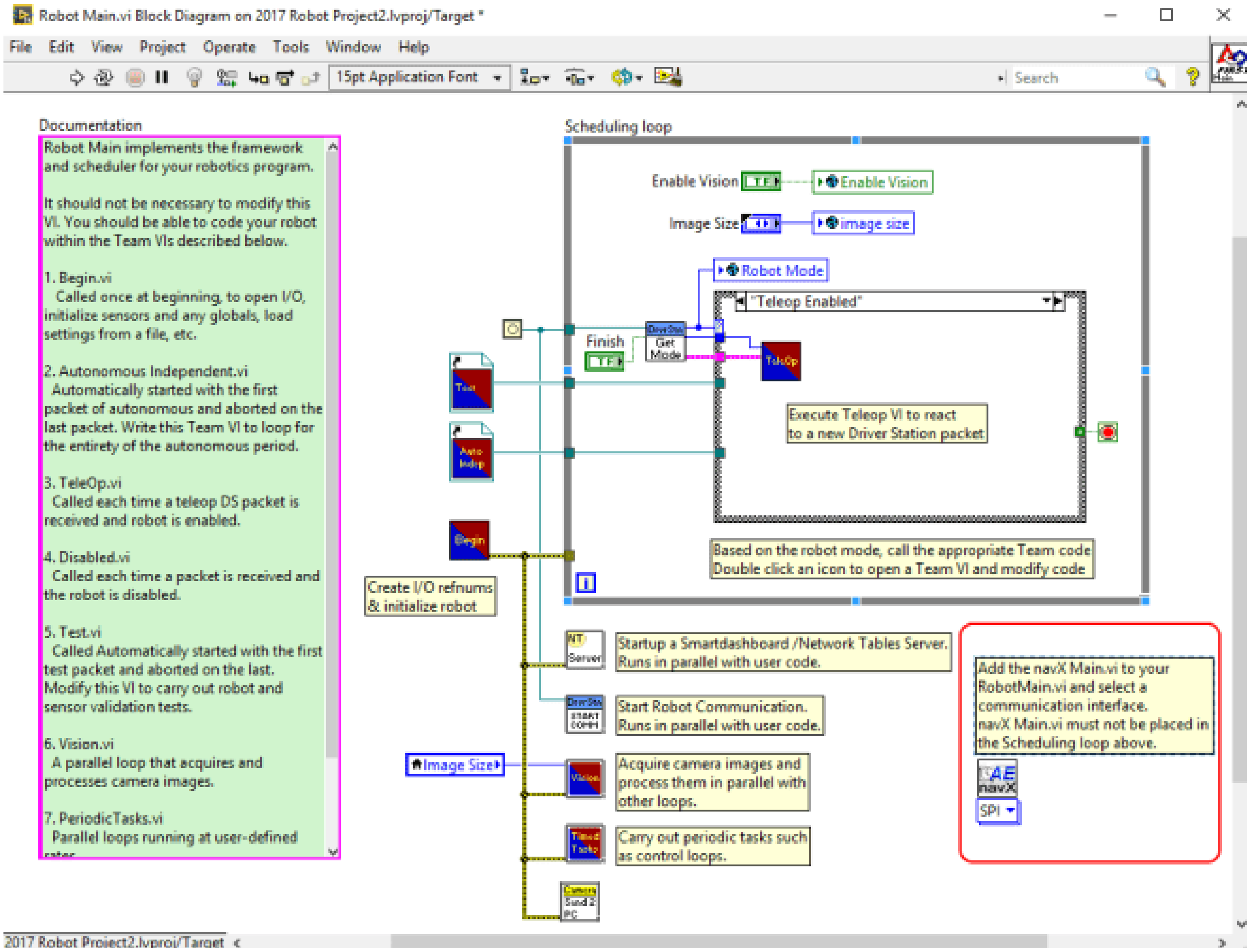
Full Java Source Code

FRC LabView Example

The navX-MXP AutoBalance LabView example shows how to make small modifications to the LabView “FRC RoboRIO Robot Project” using the “Mecanum Robot” configuration to implement collision detection.

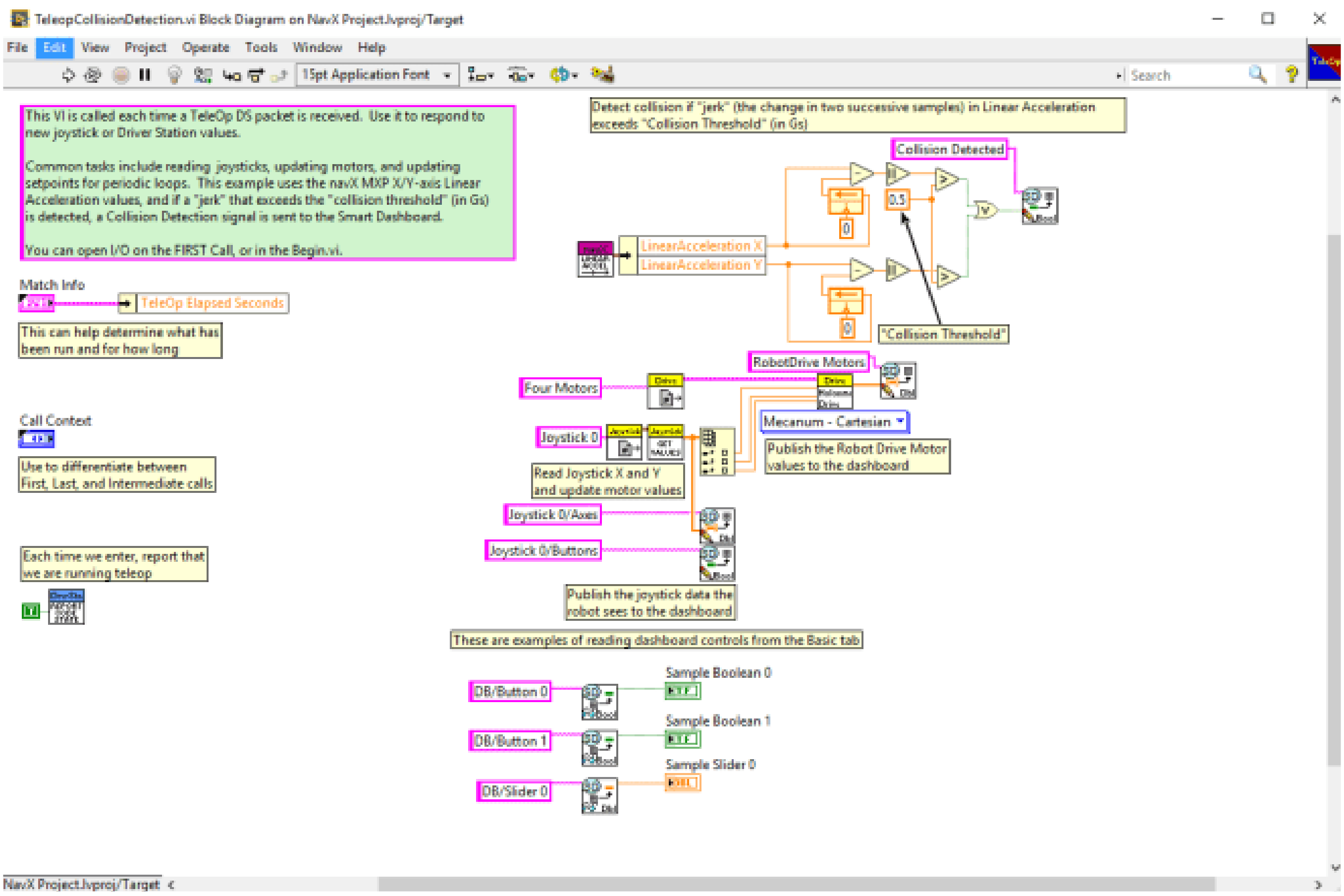
RobotMain.vi

Place the NavX main vi on the block diagram and set it up to your needs. The default sample rate is 50Hz. You may need to process faster for your situation. For the SPI, I2C and USB connections the max sample rate is 200Hz.



Teleop.vi

The Teleop.vi is modified to feed the Linear Acceleration to a threshold detector to determine if a collision has occured.



[Full LabVIEW Source code on Github](#)

Motion Detection (FRC)

Detecting motion/no-motion can be simply detected by determining if a body's linear acceleration exceeds a small threshold.

Using the data directly from accelerometers, this is not as easy as it seems, *since raw accelerometer readings contain both acceleration due to gravity as well as acceleration due to a body's motion*. One method for detecting motion with raw acceleration data is to use a [high-pass filter](#), which lets quickly-changing information through but blocks information that doesn't change frequently.

However, a more comprehensive and reliable approach is to subtract the acceleration due to gravity from the raw acceleration values. The result value is known as "world linear acceleration", representing the actual amount of acceleration due to motion, and is calculated automatically by navX MXP's motion processor. Whenever the sum of the world linear acceleration in both the X and Y axes exceeds a "motion threshold", motion is occurring.

FRC C++ Example

[Full C++ Source Code]

FRC Java Example

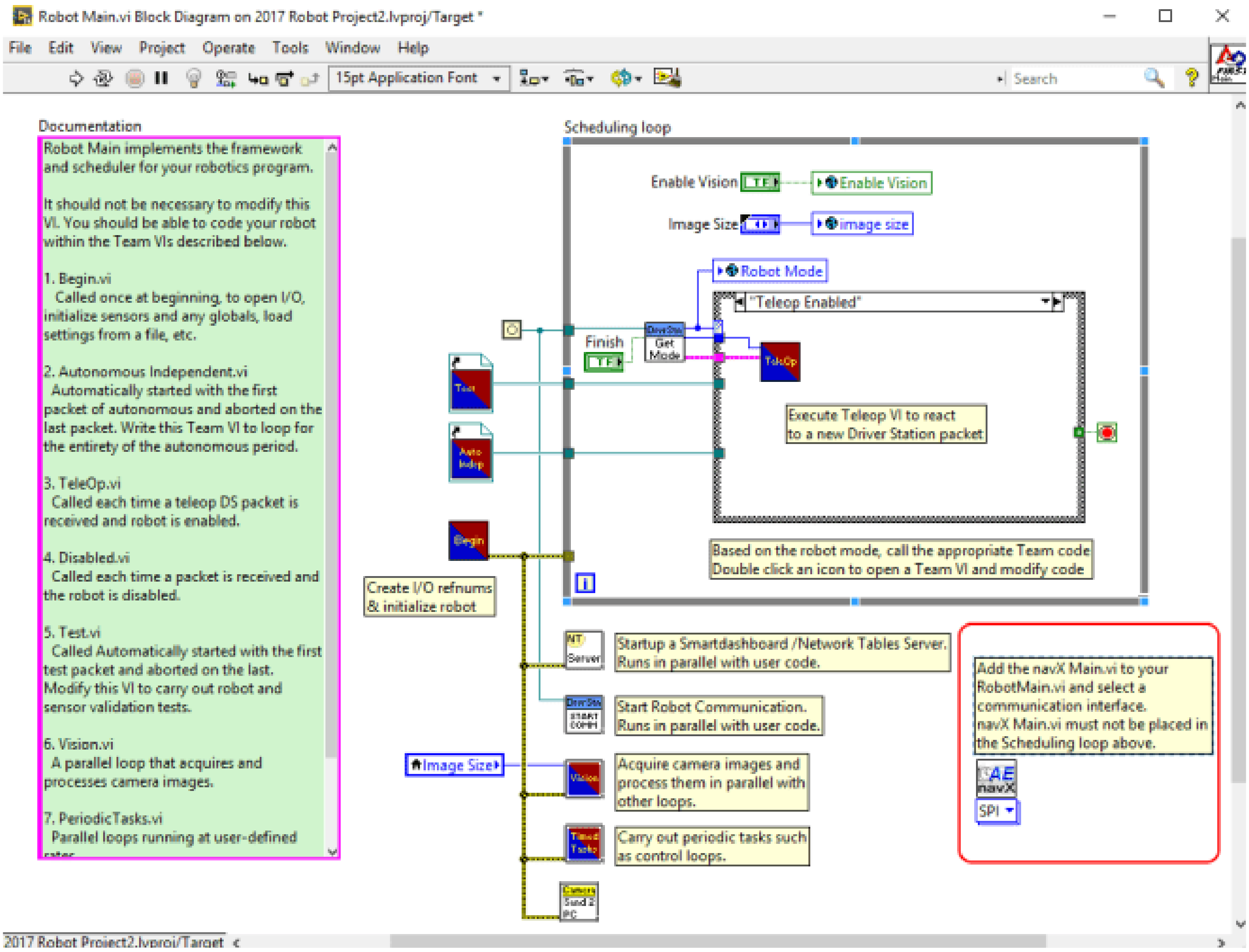
[Full Java Source Code]

FRC LabView Example

The navX-MXP AutoBalance LabView example shows how to make small modifications to the LabView “FRC RoboRIO Robot Project” using the “Mecanum Robot” configuration to detect when your robot is moving.

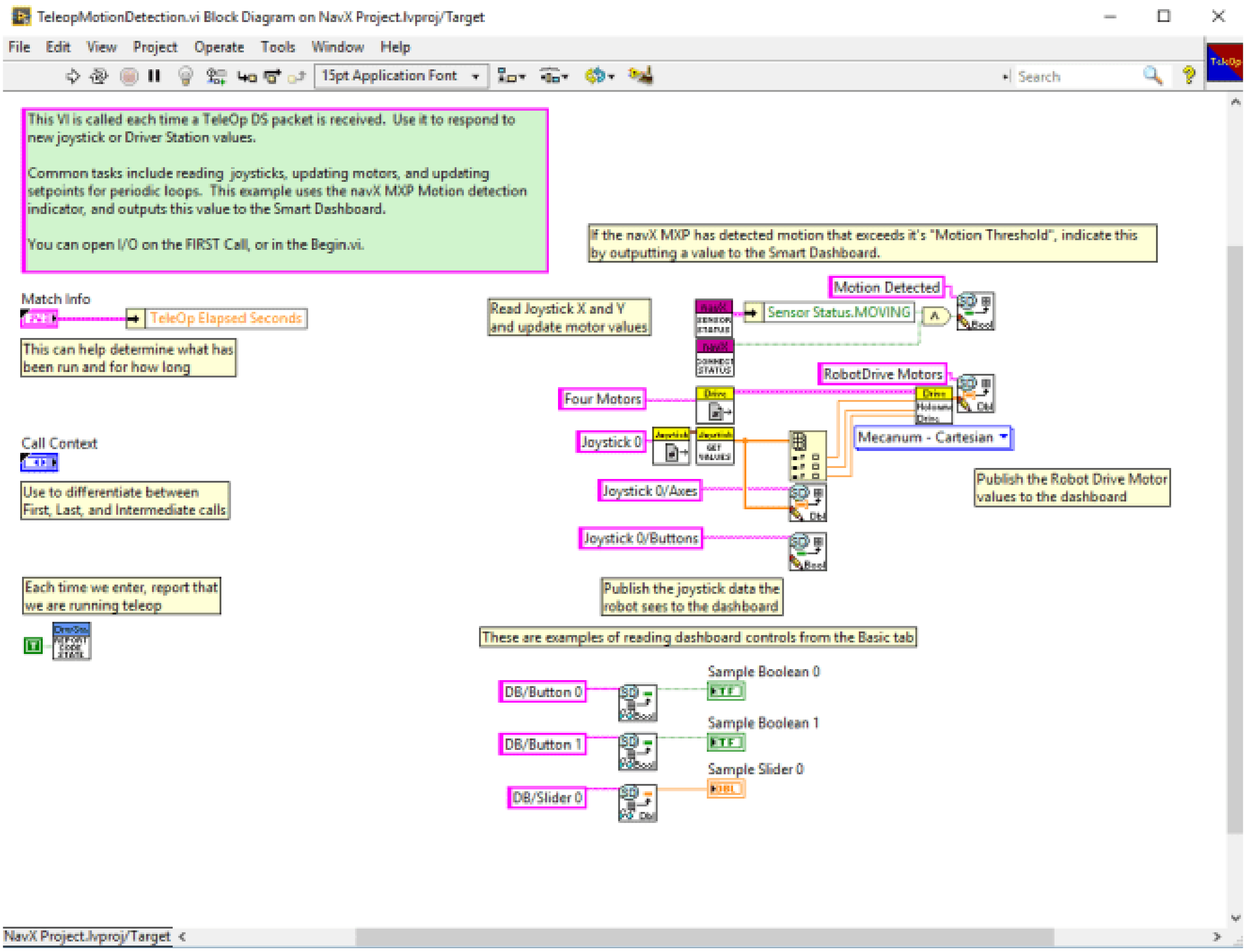
RobotMain.vi

Place the NavX main vi on the block diagram and set it up to your needs. The default sample rate is 50Hz. You may need to process faster for your situation. For the SPI, I2C and USB connections the max sample rate is 200Hz.



Teleop.vi

The Teleop.vi is modified to detect when the robot has motion.



[Full LabVIEW Source code on Github]

Data Monitor (FRC)

The Data Monitor example code demonstrates how to perform navX-MXP initialization and display all sensor values on a FIRST FRC robotics dashboard. The output data values include:

- Yaw, Pitch and Roll angles
- Compass Heading and 9-Axis Fused Heading (requires Magnetometer calibration)
- Linear Acceleration Data
- Motion Indicators
- Estimated Velocity and Displacement

- Quaternion Data
- Raw Gyro, Accelerometer and Magnetometer Data

As well, Board Information is also retrieved; this can be useful for debugging connectivity issues after initial installation of the navX-MXP sensor.

FRC C++ Example

[Full C++ source code on GitHub]

FRC Java Example

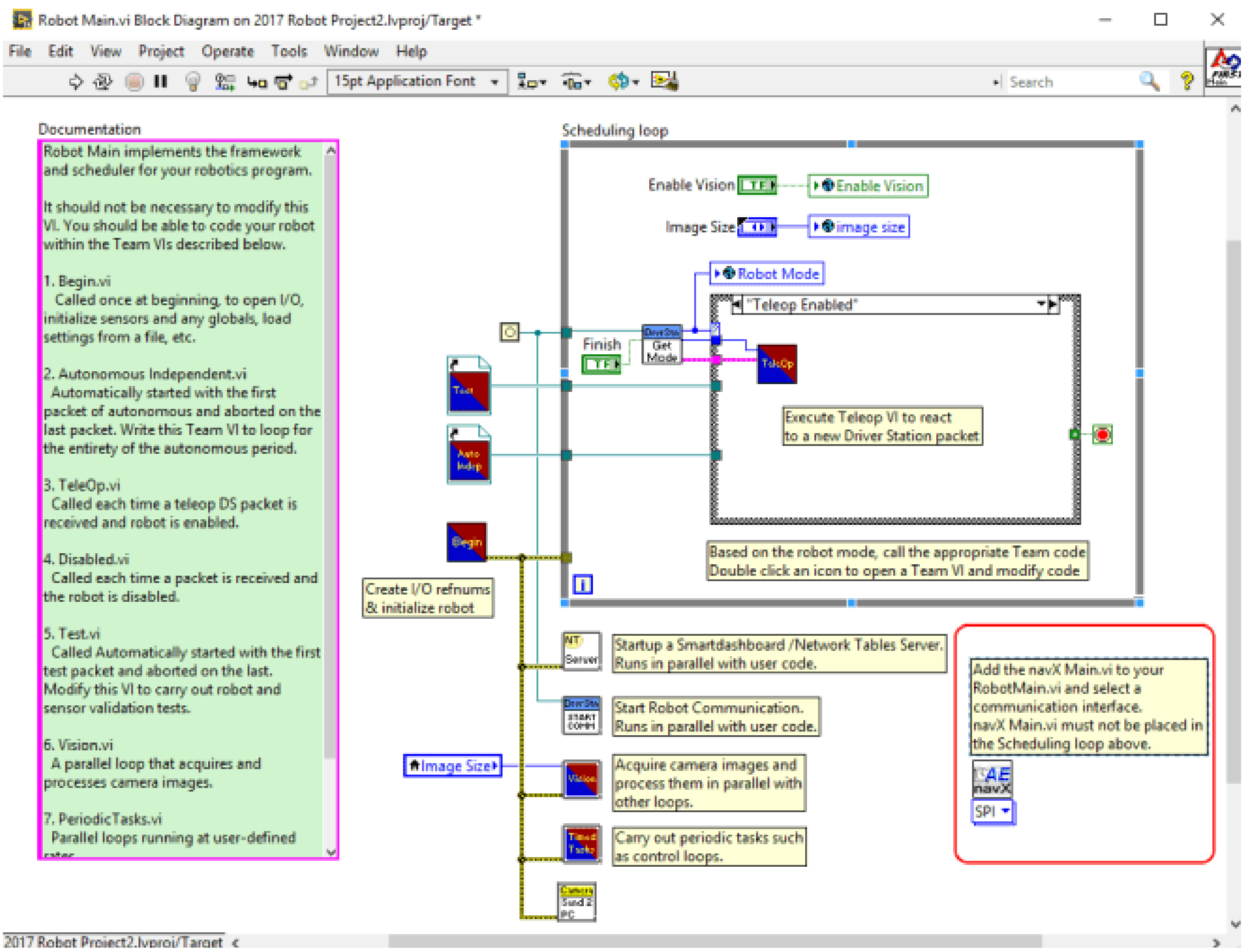
[Full Java Source code on GitHub]

FRC LabVIEW Example

The navX-MXP Test_Window.vi example shows all of the outputs from the navX through “FRC RoboRIO Robot Project”.

RobotMain.vi

Place the NavX main vi on the block diagram and set it up to your needs. The default sample rate is 50Hz. You may need to process faster for your situation. For the SPI, I2C and USB connections the max sample rate is 200Hz.



Test Window.vi

Place the Test Window.vi inside of a loop in any VI (for instance in your Teleop.vi loop) and the values will automatically update. Test Window.vi is in the navX-AE “Get” folder.

MXP I/O Expansion (FRC)

The "MXP I/O Expansion" example program demonstrates the use of the MXP I/O Expansion capabilities of the navX-MXP, including the following capabilities:

DIGITAL I/O

- Pulse-Width Modulation [PWM] (e.g., Motor Control)
- Digital Inputs (e.g., Contact Switch closure)
- Digital Outputs (e.g., Relay control)
- Quadrature Encoders (e.g., Wheel Encoder)

ANALOG I/O

- Analog Inputs (e.g., Ultrasonic Sensor)
- Analog Input Trigger (e.g., Proximity Sensor trigger)
- Analog Trigger Counter
- Analog Output (e.g., Constant-current LED, Sound)

This example also demonstrates a simple method for calculating the 'RoboRIO Channel Number' which corresponds to a given navX-MXP IO Pin number.

FRC C++ Example

[Full C++ source code on GitHub]

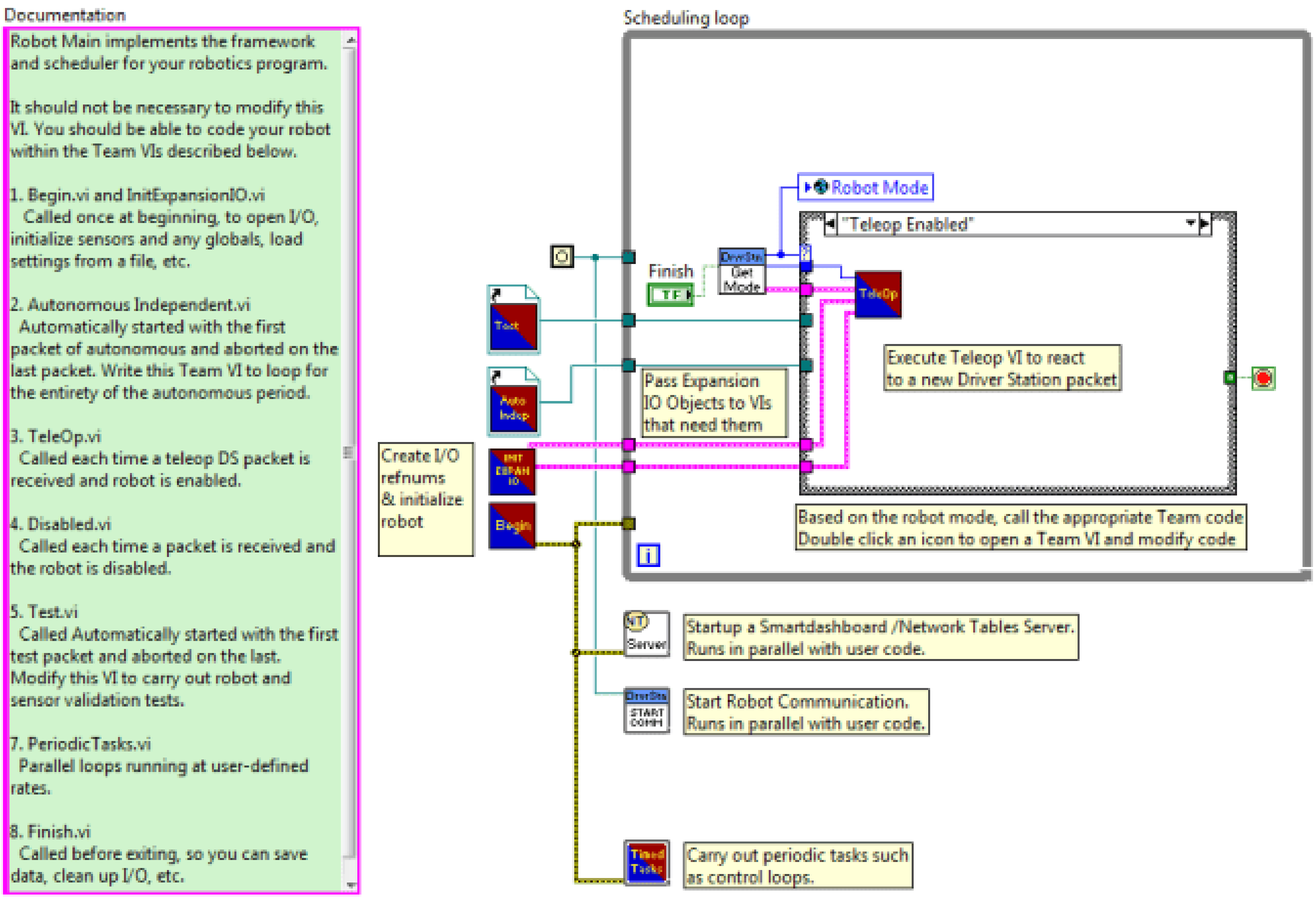
FRC Java Example

[Full Java Source code on GitHub]

FRC LabView Example

The navX-MXP FieldCentric-Drive LabView example shows how to make small modifications to the LabView “FRC RoboRIO Robot Project” using the “Mecanum Robot” configuration to implement high-accuracy Field-Centric drive.

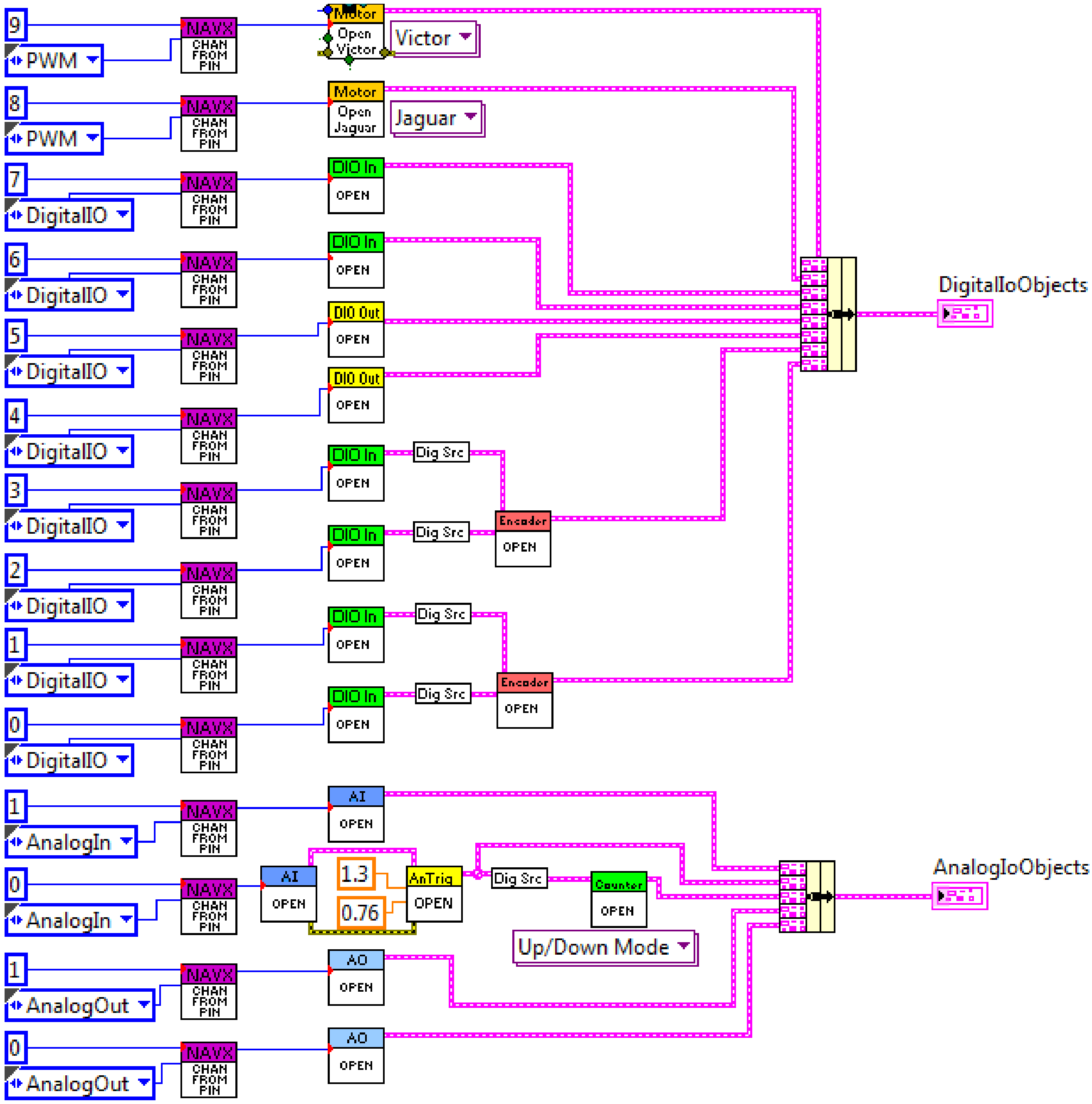
RobotMain.vi



The RobotMain.vi invokes the InitExpansionIO.vi during initialization, and routes the resulting DigitalIoObjects and AnalogIoObjects clusters to the Teleop.vi.

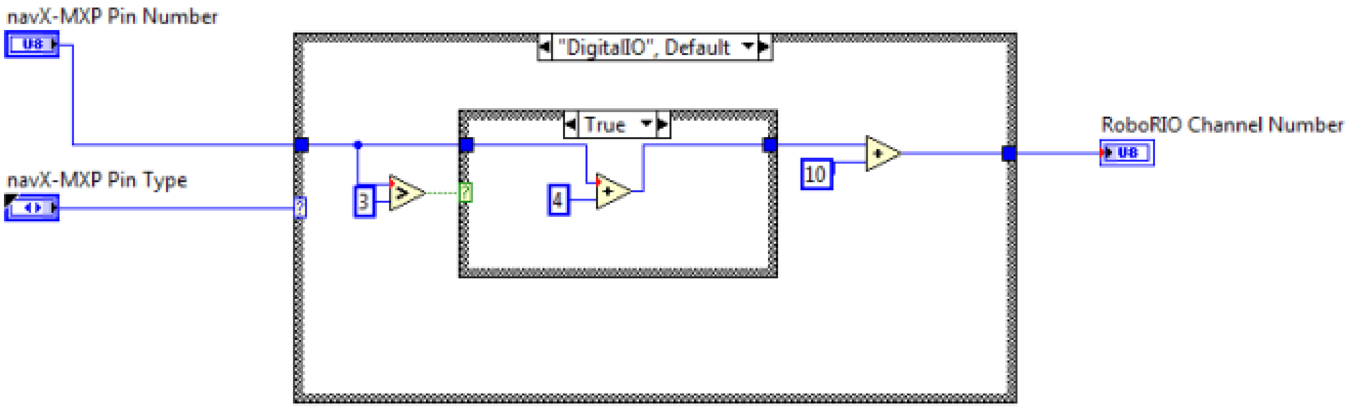
InitExpansionIO.vi

The InitExpansionIO.vi instantiates the various objects which map onto the navX-MXP Expansion IO Pins.



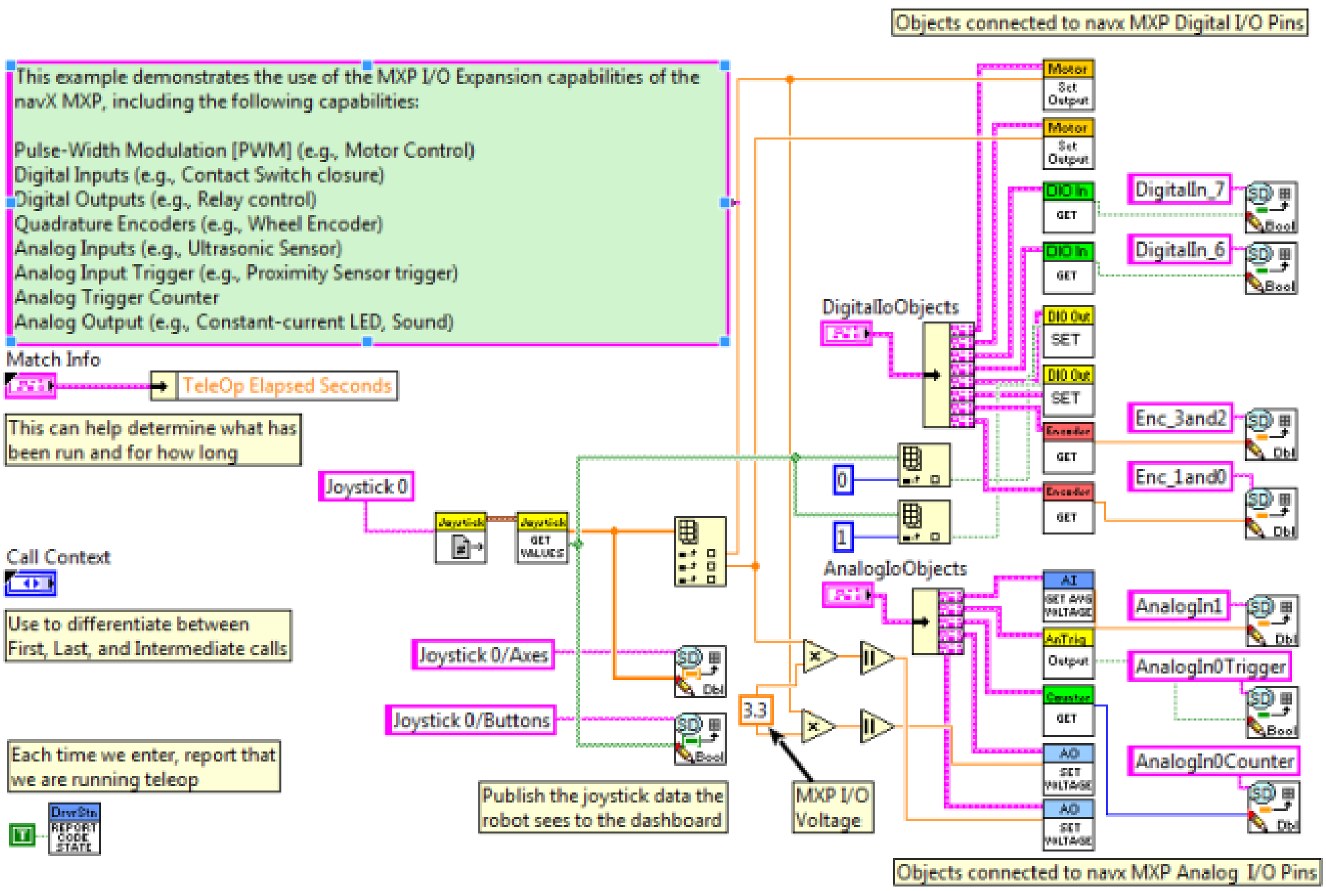
GetChannelFromNavX-MXPPin.vi

The GetChannelFromNavX-MXPPin.vi performs the translation from the navX-MXP Digital or Analog Pin number to the corresponding RoboRIO Channel Number, which is provided to the various VIs that open that particular port.



Teleop.vi

The Teleop.vi reads the Joystick inputs and programs the output pins accordingly (PWM to motor controllers, Digital Outputs and Analog Outputs). As well, values from the input pins (Digital Inputs, Encoders and Analog Inputs) is retrieved and displayed on the Smart Dashboard.



[Full LabVIEW Project on GitHub]

Guidance
Best Practices

This page summarizes the recommended best practices when integrating navX-MXP with the National Instruments RoboRIO™. Following these best practices will help ensure high reliability and consistent operation.

1) Secure the navX-MXP circuit board to the Robot Chassis

Excessive vibration will reduce the quality of navX-MXP sensor measurements. The navX-MXP circuit board should be mounted in such a way that it as firmly attached to the robot chassis.

2) Plan for RoboRIO Brownouts

The RoboRIO contains circuitry to remove power from the MXP connector when it detects an input voltage drop below a certain voltage level; this is known as a Stage 2 brownout. While brownouts do not typically occur during a FRC match (since fresh batteries are typically used at these times), during practice matches brownouts are common. If the robot drive chain draws large amounts of current, even for a short time, brownouts could potentially occur even with a FRC match.

navX-MXP maintains state information that will be reset when the navX-MXP circuit board is restarted. *Avoiding navX-MXP restarts is very important if your robot software uses the “yaw” angle.*

To avoid a navX-MXP restart when stage 2 brownouts occur, a secondary power supply for the navX-MXP circuit board should be provided. Fortunately, the RoboRIO provides just such a power supply, since its onboard USB interface is powered by a boost regulator which will provide 5V of power even when the RoboRIO input voltage (VIN) drops as low as 4.4 volts (once the RoboRIO VIN drops lower than this, the RoboRIO itself will restart).

To address this situation, simply connect a USB cable from the navX-MXP circuit board to the RoboRIO; if a brownout does occur, the navX-MXP circuit board will automatically switch to use power from the RoboRIO’s USB port.

3) Understand and Plan for Calibration

Gyro/Accelerometer Calibration is vital to achieving high-quality navX-MXP readings. Be sure to understand this process, and ensure that it completes successfully each time you use the robot.

If your robot moves during calibration, or if noticeable temperature changes occur during calibration, the calibration process may take longer than normal.

Using the navX-MXP yaw angle before calibration completes may result in errors in robot control. To avoid this situation, your robot software should verify that calibration has completed (e.g., by calling the isCalibrating() function) before using navX-MXP data.

4) If using the MXP connector, secure the navX-MXP circuit board to the RoboRio

During operation of the robot, certain actions (for instance, driving over a bump at high speed) may cause the navX-MXP circuit board to become dislodged from the MXP connector.

To avoid this case, when mounting the navx-MXP circuit board be sure to secure the navX-MXP circuit board to the RoboRio via two correctly-sized screws.

5) Protect the Sensor

navX-MXP contains sensitive circuitry. The navX-MXP circuit board should be handled carefully.

An enclosure is recommended to protect the navX-MXP circuit board from excessive handling, “swarf”, electro-static discharge (ESD) and other elements that could potentially damage navX-MXP circuitry. The enclosure can also help prevent accidental shorts to ground which may occur on the MXP Expansion I/O pins.

6) Plan for Catastrophic Sensor Failure

Any electronic component can fail. To ensure that your robot can still function during a FRC match even if such a failure does occur, your robot software should handle cases when communication with sensors such as the navX-MXP is disrupted.

An easy way to accomplish this is to use the “isConnected()” indication, and only use navX-MXP sensor data to control your robot when this is true.

Additionally, displaying whether the robot software is connected to the navX-MXP circuit board on the driver “dashboard” can help the drivers quickly detect a connection problem.

7) Provide a “Zero Yaw” feature (for Field-Oriented Drive)

The navX-MXP gyro “yaw” angle will drift over time (approximately 1 degree/minute). While this does not normally impact the robot during a FRC match, if using field-oriented drive during extended practice sessions it may be necessary to periodically “zero” the yaw. Drivers should be provided a simple way (e.g., a joystick button) with which to zero the yaw.

8) Avoid shorts on Expansion I/O pins

If a short occurs between any of the MXP Expansion I/O pins, the POWER led on the RoboRIO will turn red, and the navX-MXP circuit board will not receive power.

To protect against accidental shorts, Kauai Labs recommends a protective enclosure that at least partially encases the MXP I/O pins, such as the “lid”-style enclosure created for the navx-MXP.

9) If possible, mount the navX-MXP circuit board near the center of rotation

Since navX-MXP measures rotation, errors in the measured angles can occur if the navX-MXP circuit board is mounted at a point not near the robot center of rotation. For optimal results, the navX-MXP circuit board should be mounted at the robot’s center of rotation. If the navX-MXP circuit board cannot be mounted near the robot’s center of rotation, the offset from the center of rotation can be used to correct the yaw angle.

10) Use OmniMount if navX-MXP is not mounted horizontally

By default, the navX-MXP’s motion processing requires the unit be mounted horizontally, parallel to the earth’s surface; the yaw (Z) axis should be perpendicular to the earths surface.

If your RoboRIO is mounted vertically, you will need to enable the “OmniMount” feature in order to get reliable, accurate yaw (Z) axis readings.

11) Learn how the sensor behaves by using the navXUI

The navXUI provides insight into the key navX-MXP features, and can help debug issues you may encounter when integrating navX-MXP onto your robot. Running this user interface is highly recommended for anyone using navX-MXP. You can even run the navXUI while your robot is simultaneously communicating with the navX-MXP circuit board via the TTL UART, I2C or SPI interfaces.

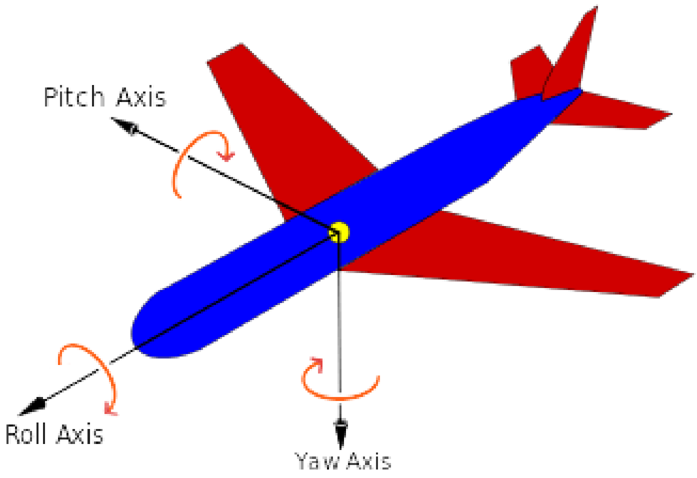
Terminology

Several terms used throughout the navX-MXP libraries and documentation may not be commonly understood and are defined herein.

Basic Terminology

A working knowledge of the following Basic Terminology is highly recommended when working with navX-MXP or any other Inertial Measurement Unit (IMU).

Pitch, Roll and Yaw



Pitch, Roll and Yaw are measures of angular rotation about an object's center of mass, and together provide a measure of "orientation" of that object with respect to an "at rest" position. When units of degrees are used, their range is from -180 to 180 degrees, where 0 degrees represents the "at rest" position of each axis.

Axis	Orientation relative to object's center of mass	Rotational Motion
X (Pitch)	Left/Right	+ Tilt Backwards
Y (Roll)	Forward/Backward	+ Roll Left
Z (Yaw)	Up/Down	+ Clockwise/ – Counter-wise

Important Note: Pitch, Roll and Yaw angles represent rotation from the "origin" (0,0,0) of a 3-axis coordinate system. navX-MXP Pitch and Roll angles are referenced to earth's gravity – so when navX-MXP is flat, Pitch and Roll angles should be 0.

The Yaw angle is different – Yaw is not referenced to anything external. When navX-MXP startup calibration completes, the Yaw angle is automatically set to 0 – thus at this point, 0 degrees represents where the "head" of the navX-MXP circuit board is pointing. The Yaw angle can be reset at any time after calibration completes if a new reference direction is desired.

Linear Acceleration

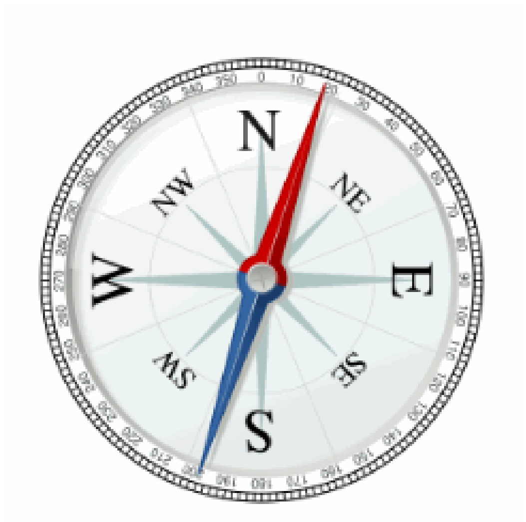
Linear Acceleration is a measure of the change in velocity in a specific direction. For example, when a car starts from a standstill (zero relative velocity) and travels in a straight line at increasing speeds, it is accelerating in the direction of travel.

Axis	Orientation	Linear motion
X	Left/Right	– Left / + Right

Y	Forward/Backward	+ Forward / – Backward
Z	Up/Down	+ Up / – Down

Because the gyroscope and accelerometer axes are aligned, navX-MXP measures linear acceleration in the same set of 3 axes used to measure Pitch, Roll and Yaw. However unlike Pitch, Roll and Yaw, acceleration measures linear motion rather than rotation, and is measured in units of G, with a range of +/- 2.0.

Compass Heading



A compass measures the earth's magnetic field and indicates the current direction (heading) relative to magnetic north (N). Compass Heading is measured in degrees and is similar to Yaw, but has a few key differences:

- Compass Heading has a range of 0-360 (where magnetic north is 0).
- Compass Heading is absolute – it is referenced to magnetic north, and thus Compass Heading does not drift over time

Important Note 1: Compass Heading relies upon being able to measure the earth's magnetic field. Since the earth's magnetic field is weak, Compass Heading may not be able to measure earth's magnetic field when the compass is near a strong magnetic field such as that generated by a motor.

Important Note 2: Magnetic North is not exactly the same as True North*. Your robot can calculate True North given a Magnetic North reading, as long as the current declination is known. Declination is a measure of the difference in angle between Magnetic North and True North, and changes depending upon your location on earth, and also changes over time at that same location. An* online calculator *is provided allowing one to calculate declination for a given earth location and date.*

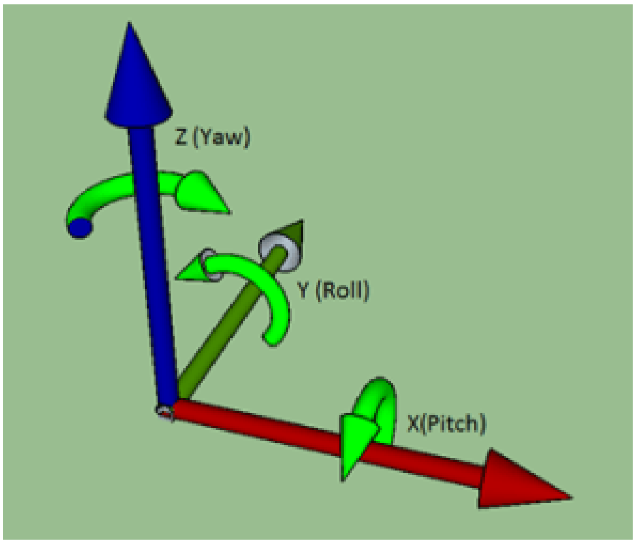
Altitude

Altitude is a measure of distance in the “up” direction from a reference; navX-MXP (Aero Edition) calculates altitude above sea-level using a pressure sensor.

navX-MXP (Aero Edition) altitude has a range of 0 to 25,000 meters.

Important Note: Altitude is calculated based upon barometric pressure. In order to accurately estimate altitude above the earth, navX-MXP should be configured with the sea-level barometric pressure in the surrounding area. This setting can be configured via the navX-MXP Advanced Configuration Tool.

3-D Coordinate System



navX-MXP 3-D Coordinate System

A 3-D Coordinate System uses one or more numbers (coordinates), often used to uniquely determine the position of a point within a space measured by that system. The origin of a 3-D coordinate system has a value of (0, 0, 0).

navX-MXP features gyroscopes, accelerometers and magnetometers which are all aligned with each other in a 3-D coordinate system. Each sensor type measures values with respect to that coordinate system, as follows:

Gyroscopes: measure rotation (as shown in the green arrows) about each axis. The coordinate system origin represents the center of the navX-MXP circuit board.

Accelerometers: measure acceleration, where the origin represents the position in space at which the previous acceleration sample was acquired.

Magnetometers: measure earth's magnetic field, where the origin represents the center of the navX-MXP circuit board.

Important Note: Because the navX-MXP Gyroscopes, Accelerometers and Magnetometers are all aligned to this 3-D Coordinate System, navX-MiXP's motion processor can also use Sensor Fusion to provide additional information and processing including Tilt Correction, "Fused Heading", a Gravity Vector, World Reference Frame-based Linear Acceleration and Quaternions, as discussed in the Motion Processing section below.

Motion Processing

Users should also have a working knowledge of the terms defined in the Motion Processing Terminology.

Tilt Correction

Without correction, the compass heading calculated by a 3-axis magnetometer will only be accurate if the magnetometers are held "flat" with respect to the earth. To ensure the compass heading is valid even in cases when the sensor is "pitched" (Pitch angle != 0) or "rolled" (Roll angle != 0), navX-MXP performs "tilt correction" fusing the reading from the magnetometers with the pitch and roll angles from the accelerometers. Once corrected, the compass heading is aligned with the navX-MXP Z axis, which ensures that the Yaw angle and the Compass Heading measure rotation similarly.

"Fused" Heading

Given the gravity-referenced orientation provided by the Yaw angle, as well as the absolute compass heading angle which has been aligned to the navX-MXP 3-D coordinate system, both angles can be fused together. As shown in the table below, over a period of several minutes this can minimize the drift inherent in the Yaw angle, as well as provide an absolute reference for the Yaw angle – as long as the magnetometer is calibrated and a valid magnetometer reading is available every minute or so.

Value	Accuracy	Update Rate	Drift
Yaw	.01 degrees	Up to 200 Hz	~1 degree/minute
Compass Heading	2 degrees	1 Hz (if not magnetically disturbed)	None
Fused Heading	2 degrees (as long as a valid magnetometer reading is received in the last minute or so)	Up to 200Hz	None (~1 degree/minute, during periods of magnetic disturbance)

Like the Compass Heading, the Fused Heading has a range from 0-360 degrees.

Important Note: If the Compass Heading is not valid, the Fused Heading origin is the same as *the Yaw angle. When valid (magnetically undisturbed) compass readings are received, the Fused Heading's origin shifts to magnetic north (0 degrees on the Compass).*

Gravity Vector

Accelerometers measure both acceleration due to gravity, as well as acceleration due to linear acceleration. This fact makes using raw accelerometer data difficult. navX-MXP's automatic accelerometer calibration determines the component of measured acceleration which corresponds to gravity, and uses this information together with gyroscope readings to calculate a gravity vector, which represents acceleration due to gravity. Pitch and Roll angles are derived from this gravity vector.

Once the gravity vector is understood, this value is then subtracted from the raw accelerometer data to yield the acceleration due to linear motion.

Velocity and Displacement

Acceleration is defined as the change in Velocity. Therefore, linear velocity can be calculated by integrating linear acceleration over time.

Velocity is defined as the change in Position, otherwise known as Displacement. Therefore, linear displacement can be calculated by integrating linear velocity over time.

Important Note: Using currently-available MEMS-based accelerometers to calculate linear velocity and displacement is subject to large amounts of error primarily due to accelerometer "noise" (a difference between the actual acceleration and the measured acceleration inherent with MEMS sensors). This noise not only accumulates, but is also squared in the case of velocity, and is cubed in the case of displacement. Therefore, the resulting estimated velocity and displacement values are not typically useful for robotic navigation. The amount of error in displacement estimation can be several feet per second. As MEMS sensors improve in the coming years and accelerometer noise is reduced by approximately 100 times its currently value, this technique will become more useful for robotics navigation.

If you would like to experiment with using the navX-MXP to calculate displacement and velocity, you can use the navXUI'*s "Experimental" button to bring up a dialog which displays the integrated velocity and displacement values calculated in real-time by the navX-MXP.*

World Reference Frame

Raw acceleration data measures acceleration along the corresponding sensor axis. This measurement occurs in a reference frame known as "Body Reference Frame". This works well as long as the navX-MXP circuit board is in it's original orientation. However as the navX-MXP circuit board rotates, the X and Y accelerometer axes no longer point "forward/back" and "left/right" with respect to the original orientation. To understand this more clearly, consider how the meaning of the term "left" changes once a robot has rotated 180 degrees? Introducing a

World Reference Frame solves this issue by providing a reference upon which to measure "leftness".

To account for this, navX-MXP's motion processing adjusts each linear acceleration value by rotating it in the opposition direction of the current yaw angle. The result is an acceleration value that represents acceleration with respect to the area in which navX-MXP operates, which is known as "World Reference Frame". This world-frame linear acceleration value is much simpler to use for tracking motion of an object, like a robot, which might rotate while it moves.

Important Note: navX-MXP Linear Acceleration values are in World Reference Frame.

Advanced

Advanced users may require knowledge of the following terminology.

Quaternions

A quaternion is a four-element vector that can be used to encode any rotation in a 3D coordinate system. This single 4-element vector value can describe not only rotation about a reference frame's origin (Pitch, Roll and Yaw) but also the rotation of that entire reference frame with respect to another. Furthermore, when Pitch, Roll and Yaw measures to perform certain calculations, it is not possible to clearly ascertain orientation when two axes are aligned with each other; this condition is referred to as "Gimbal Lock". For robotics applications, Pitch, Roll and Yaw are sufficient, however for certain aerospace applications, Quaternions may be required to handle all possible orientations.

navX-MXP uses Quaternions internally, and also provides the 4 quaternion values for use by those who might need them.

Selecting an Interface

The navX-MXP provides several methods for communicating with robotics control applications:

- MXP I2C
- MXP SPI
- USB 2.0

Streaming vs. Register-based Communication

The navX-MXP interfaces fall into two types: Streaming and Register-based.

Streaming: data is sent at regular intervals by the navX-MXP, and the host is notified when new data arrives. To support the low bitrate of the TTL UART interface, the streaming data is sent in two different formats: Processed Data and Raw data. Streaming is used over the TTL UART and USB interfaces. More details on the communication detail are available in the Serial Protocol Definition.

Register-based: communication is initiated by the host whenever new data is desired, and the host can request any data required. Register-based communication is used over the I2C and SPI interfaces. More details on the communication detail are available in the Register Protocol Definition.

Comparing the navX-MXP Communication Interfaces

Interface Type	Speed	Latency	Type	Cable distance	Max Update Rate
SPI	2 mbps	<1ms	Register-based	<1 meter	200
I2C	400 kbps	~10ms	Register-based	1 meter	200
USB	12 mbps	1ms	Streaming	6 meters	200

Recommendations

Based upon the above, the following recommendations are provided for selecting the best navX-MXP communications interface:

– If mounting the navX-MXP directly on the RoboRIO, the **SPI** interface is preferred for it's high speed and low latency.

– If mounting the navX-MXP separately from the RoboRIO using an extension cable and if MXP IO support is desired, run **SPI at a lower speed**. The I2C interface is also a reasonable option.

– If mounting the navX-MXP separately from the RoboRIO, and MXP IO support is not desired and only Processed or Raw Data (not both) is needed, **USB** may be used. This configuration is useful when using the navX-MXP magnetometer data, since it makes it possible to mount the navX-MXP far away from motors. This configuration is also useful when accessing navX-MXP data from a separate processor, such as a PC or a separate video processor. However, please note that in certain cases when other USB devices (e..g, cameras) are connected to the same RoboRIO USB bus, and are used simultaneously with navX-MXP, sometimes the communication is interrupted. For this reason, USB is not recommended on the RoboRIO, especially if you are connecting with other USB devices on the same USB bus.

Gyro/Accelerometer Calibration

Gyro/Accelerometer Calibration

navX-MXP onboard sensors require calibration in order to yield optimal results. We highly recommend taking the time to understand this calibration process – successful calibration is vital to ensure optimal performance.

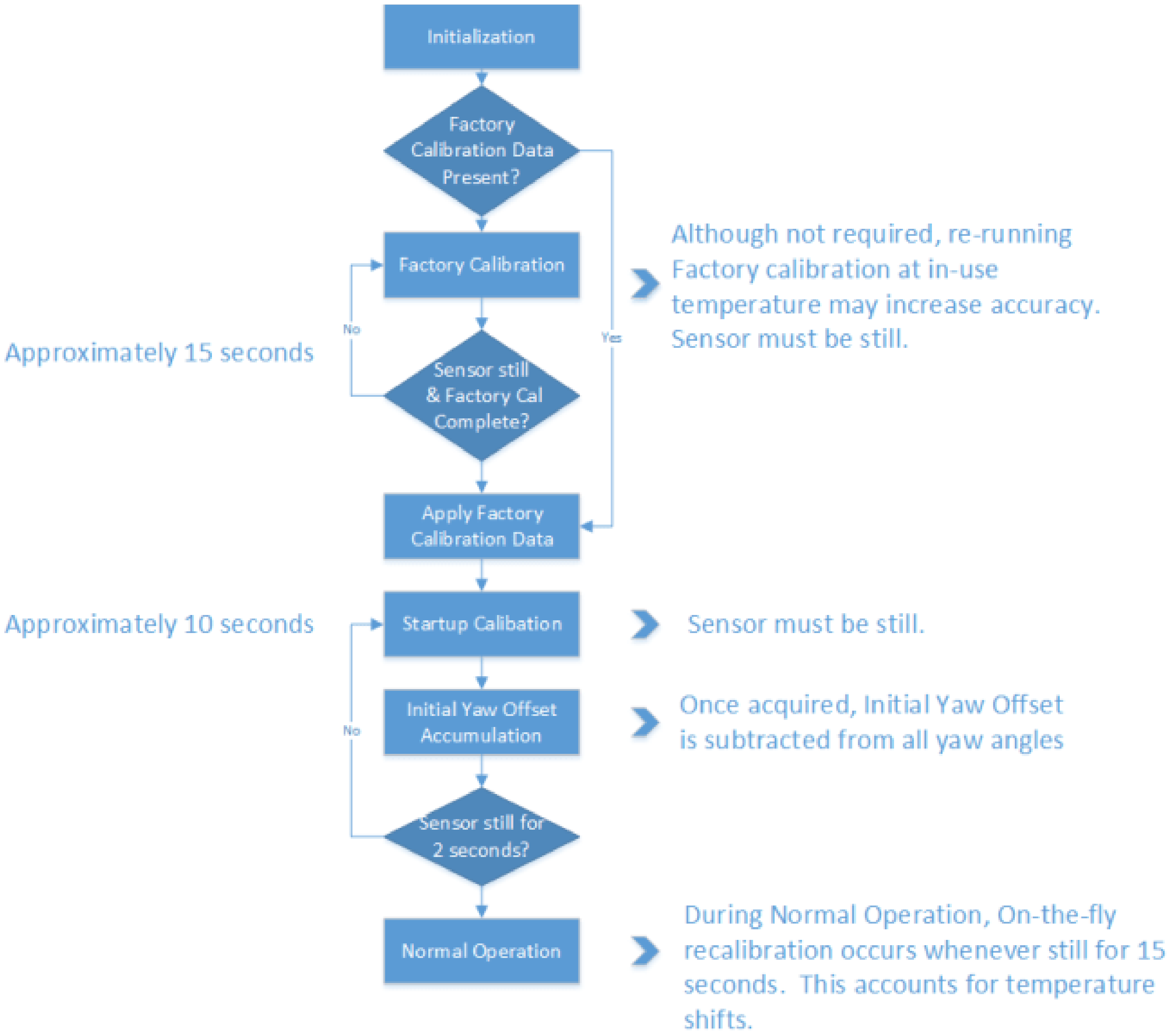
Accurate Gyroscope Calibration is crucial in order to yield valid yaw angles. Although this process occurs automatically, understanding how it works is required to obtain the best results.

If you are tempted to ignore this information, please read the section entitled "The Importance of Stillness" at the end of this page.

Calibration Process

The navX-MXP Calibration Process is comprised of three calibration phases:

- Factory Calibration
- Startup Calibration
- On-the-fly Calibration



Factory Calibration

Before navX-MXP units are shipped, the accelerometers and gyroscopes are initially calibrated at the factory; this calibration data is stored in flash memory and applied automatically to the accelerometer and gyroscope data each time the navX-MXP circuit board is powered on.

Note that the onboard gyroscopes are sensitive to temperature changes. Therefore, since the average ambient temperature at the factory (on the island of Kauai in Hawaii) may be different than in your environment, you can optionally choose to re-calibrate the gyroscope by pressing and holding the “CAL” button for at least 10 seconds. When you release the “CAL” button, ensure that the “CAL” Led flashes briefly, and then press the “RESET” button to restart navX-MXP. When navX-MXP is re-started, it will perform the Initial Gyro Calibration – the same process that occurs at our factory. *NOTE: It is very important to hold navX-MXP still, and parallel to the earth’s surface, during this Initial Gyro Calibration period.* You might consider performing this process before using your robot the first time it is used within a new environment (e.g., when you arrive at a FTC competition event).

The value of re-running Factory Calibration at the same temperature navX-MXP will be operated at is potentially increased yaw accuracy as well as faster Startup Calibration. If a significant temperature shift has occurred since the last Factory Calibration, the Startup Calibration time may take longer than normal, and it's possible that yaw accuracy will be diminished until the next On-the-fly Gyro Calibration completes.

Startup Calibration

Startup Calibration occurs each time the navX-Micro is powered on, and requires that the sensor be held still in order to complete successfully. Using the Factory Calibration as a starting point, the sensor calibrates the accelerometers and adjusts the gyroscope calibration data as well based upon current temperature conditions.

If the sensor continues to move during startup calibration, Startup Calibration will eventually timeout – and as a result, the navX-Micro yaw angle may not be as accurate as expected.

Initial Yaw Offset Calibration

Immediately after Startup Calibration, an **Initial Yaw Offset** is automatically calculated. The purpose of the Initial Yaw Offset is to ensure that whatever direction the "front" of the navX-MXP circuit board is pointed to at startup (after initial calibration is applied) will be considered "0 degrees".

Yaw Offset Calibration requires that navX-MXP be still for approximately 2 seconds after Startup Calibration completes. After approximately 2 seconds of no motion, navX-MXP will acquire the current yaw angle, and will subtract it from future yaw measurements automatically. The navX-MXP protocol and libraries provide a way to determine the yaw offset value it is currently using.

NOTE: If navX-MXP is moving during startup, this Yaw Offset Calibration may take much longer than 2 seconds, and may not be calculated at all if the sensor continues moving long enough. Therefore it is important to keep navX-Micro still until initial calibration and Initial Yaw Offset calibration completes.

On-the-fly Gyro Calibration

In addition to Startup Calibration, during normal operation navX-MXP will automatically re-calibrate the gyroscope (e.g., to account for ongoing temperature changes) during operation, whenever it detects 8 seconds of no motion. This process completes after about 7-8 more seconds, and is completely transparent to the user. Therefore each time navX-MXP is still for approximately 15 seconds, the gyroscopes are re-calibrated "on-the-fly". The purpose of On-the-fly Gyro re-calibration is to help maintain yaw accuracy when shifts in ambient temperature occur during operation.

This On-the-fly Gyro Calibration can help deal with cases where the sensor was moving during Startup Calibration, but note that the yaw is not zeroed at the completion of On-the-fly Calibration. So once again, it's important to keep the sensor still during Startup Calibration.

Runtime Yaw Zeroing

Your robot software can optionally provide the robot operator a way to reset the yaw angle to Zero at any time. Please see the documentation for the navX-MXP libraries for more details.

The importance of stillness

This is the most important takeaway from this discussion: It is very important that navX-MXP be held still during the above Initial Gyro and Initial Yaw Offset calibration periods. In support of this, navX-MXP indicates when it is calibrating; we recommend you incorporate this information into your software. Please see the discussion of the navXUI, and the navX-MXP libraries for more details on this indication.

Magnetometer Calibration Tool

Please visit navX-Sensor Support for Magnetometer Calibration Tool usage instructions.

Yaw Drift

A gyroscope measures the amount of angular rotation about a single axis. Since the gyroscope measures changes in angular rotation, rather than an absolute angle, calculation of the actual current angle of that axis is estimated via numerical integration rather than an exact measurement.

Any Inertial Measurement Unit (IMU), including navX-MXP, that integrates a signal from a gyroscope will also accumulate error over time. Accumulated error is due to several factors, including:

- Quantization noise (which occurs when an analog-to-digital converter (ADC) converts a continuous analog value to a discrete integral value)
- Scale factor error (which occurs due to manufacturing errors causing a specified scale factor [e.g., 256 bits per unit G] to be incorrect)
- Temperature instability (which occurs when a sensor is more or less sensitive to an input as temperature changes)
- Bias error (which occurs because the value the sensor reports at 'zero' is not known well enough to 'subtract' that value out during signal processing)

Over time, these errors accumulate leading to greater and greater amounts of error.

With the navX-MXP, Quantization error is minimized due to the MPU-9250's internal signal conditioning, high-resolution 16-bit Analog-to-Digital Converters (ADC), and extremely fast internal sampling (200Hz). Scale factor error is easily corrected for by factory calibration, which the navX-MXP provides. So these two noise sources are not significant in the navX-MXP.

The remaining sources of error – temperature instability and bias error – are more challenging to overcome:

- Gyro bias error is a major contributor to yaw drift error, but is inherent in modern MEMS-based gyroscopes like the MPU-9250.
- Temperature instability can cause major amounts of error, and should be managed to get the best result. To address this, the MPU-9250 automatically re-calibrates the gyro biases whenever it is still for 8 seconds, which helps manages temperature instability.

Errors in the navX-MXP Pitch and Roll values to be extremely accurate over time since gyroscope values in the pitch/roll axes can be compared to the corresponding values from the accelerometer. This is because when navX-MXP is still, the accelerometer data reflects only the linear acceleration due to gravity.

Correcting for integration error in the Yaw axis is more complicated, since the accelerometer values in this axis are the same no matter how much yaw rotation exists.

To deal with this, several different "data fusion" algorithms have been developed, including:

- Complementary filter
- Extended Kalman filter (EKF)
- Direction Cosine Matrix filter (DCM)

Note: See the References page for links to more information on these algorithms.

These algorithms combine the acceleromter and gyroscope data together to reduce errors.

The Complementary and EKF filter algorithms are designed to process 3-axis accelerometer and 3-axis gyroscope values and yield yaw/pitch/roll values. The Complementary filter is a simple approach, and works rather well, however the response time is somewhat slower than the EKF, and the accuracy is somewhat lower.

The DCM filtering approach is similarly accurate and responsive as the EKF, however it requires information from a 3-axis magnetometer as well to work correctly. Since the magnetometer on a FIRST FRC robot typically experiences significant amounts of magnetic disturbance, the DCM algorithm is not well suited for use in a Robotics Navigation Sensor.

For these reasons, the EKF is the preferred filtering algorithm to provide the highest performance IMU on a FIRST FRC robot. However, the EKF algorithm is complex and difficult to understand, making it typically beyond the capabilities of many robotics engineers. The navX-MXP circuit board uses the Invensense MPU-9250 IC, and this IC implements a proprietary algorithm which is widely believed to be an EKF (it exhibits similar accuracy to documented EKF implementations on MEMS acceleromter/gyroscope sensors).

With this processing, navX-MXP exhibits yaw drift on the order of ~1 degree per minute; yaw drift is typically much lower when navX-MXP is still.

Tips

What follows are some tips on how to deal with the yaw drift within the context of a FIRST FRC competition.

In general, the yaw will not drift significantly during a FRC match, based upon the following calculation:

> yaw drift(degrees) at end of match = yaw drift (~1 degree/minute) x match length (2.5 minute) = **~2.5 degrees**

However, during long practice matches the drift may become noticeable, and can be dealt with using the following approaches:

1) The simplest approach which is supported by the navX-MXP RoboRIO libraries is to periodically “re-zero” navX-MXP by applying an offset to the navX-MXP yaw angle. To use this approach, when the robot is in the correct orientation, a driver can press a button which causes an offset to be added so that the reported angle at that orientation is 0.

2) Even though the navX-MXP magnetometer will likely give erroneous readings once the robot motors are energized, a calibrated magnetometer can potentially provide a stable reading during the moments before a FRC competition round. The navX-MXP provides a 9-axis “fused heading” which is combined with the ~1 degree per minute of drift in the yaw angles. Using the “fused heading”, it is possible to calculate the robots absolute orientation and maintain it. With the “fused heading”, that drift will be updated w/the absolute heading from the compass whenever a compass reading which is free from magnetic disturbance is detected. Note that to be effective this requires the magnetometer to be calibrated. Once calibrated, an initial magnetometer reading undisturbed by magnetic disturbances can be acquired at the beginning of a match, before the motors are energized. If the sensor is placed far enough away from motors, it may be possible to also get an undisturbed magnetometer during a match.

Support
Support

Please visit navX-Sensor Support if you are experiencing difficulty or trouble.

In addition, some common needs are addressed:

- Instructions for updating the navX-MXP Firmware
- The navX-MXP Discussion Forum
- A “factory test” procedure which can verify the navX-MXP circuit board is functioning properly

Firmware Archive

The Firmware Archive includes past navX-MXP firmware releases. Please visit navX-Sensor Support to access the firmware archive.

Factory Test Procedure

The Factory Test Procedure verifies correct operation of the circuit board and it’s key components. Please visit navX-Sensor Support for Factory Test Procedure instructions.

Software Archive

The navX-MXP Software Archive includes past navX-MXP software releases.

NOTE: Kauai Labs strongly recommends using the latest software versions.

To download an archived navX-MXP software version, right-click on the version number and download the file to your computer, and run the setup.exe file.

2018 FRC Season Release

Version Number: 3.0.348

The cross-platform build is also available for non-Windows platforms.

2017 FRC Season Release

Version Number: 3.0.329

The cross-platform build is also available for non-Windows platforms.

2016 FRC Season Release

Version Number: 3.0.263

Change Summary

- Added new “Omnimount” capabilities
- Adds onboard integration of Acceleration and Velocity estimates
- Adds onboard “yaw reset” feature
- Fixes some reliability issues w/I2C and SPI communication

Advanced
Serial Protocol

In order to communicate sensor data to a client (e.g., a RoboRio robot controller) the navX-MXP software uses a custom protocol. This protocol defines messages sent between the navX-MXP and the client over a serial interface, and includes an error detection capability to ensure corrupted data is not used by the client.

The navX-MXP Serial protocol uses two message types, the legacy ASCII messages initially introduced in the nav6 sensor, and the modern binary messages introduced in the navX-MXP.

Source code that implements the navX-MXP ASCII and binary protocols in Java and C++ are provided to simplify adding support for the navX-MXP protocol to a software project.

Message Structure

ASCII Protocol Messages

Each navX-MXP Serial ASCII protocol message has the following structure:

Start of Message	Message ID	Message Body	Message Termination
1 byte	1 byte	length is message-type dependent	4 bytes

Binary Protocol Messages

Each navX-MXP Serial Binary protocol message has the following structure:

Start of Message	Binary Message Indicator	Binary Message Length	Message ID	Message Body	Message Termination
1 byte	1 byte	1 byte	1 byte	length is message-type dependent	4 bytes

Data Type Encoding (ASCII)

Base16 encoding is used for ASCII message elements, as follows:

Data Type	Encoding	Example
Float	(Sign)(100s)(10s)(1s).(10ths)(100ths)	'-132.96'. ' 257.38'
8-bit Integer	(HighNibble)(LowNibble)	'E9'

16-bit Integer	(HighByte,HighNibble)(HighByte,LowNibble)(LowByte,HighNibble)(LowByte,LowNibble)	'1A0F'

Data Type Encoding (Binary)

Binary encoding is used for all Binary message elements. All Binary-formatted data types that are signed are encoded as 2's complement. All multi-byte data types are in little-endian format. Certain non-standard 'packed' data types are used to increase storage efficiency.

Data Type	Range	Byte Count
Unsigned Byte	0 to 255	1
Unsigned Short	0 to 65535	2
Signed Short	-32768 to 32768	2
Signed Hundredths	-327.68 to 327.67	2
Unsigned Hundredths	0.0 to 655.35	2
Signed Thousandths	-32.768 to 32.767	2
Signed Pi Radians	-2 to 2	2
Q16.16	-32768.9999 to 32767.9999	4
Unsigned Long	0 to 4294967295	4

*Unsigned Hundredths: original value * 100 rounded to nearest integer
*Signed Hundredths: original value * 100 rounded to nearest integer
*Signed Thousandths: original value * 1000 rounded to nearest integer
*Signed Pi Radians: original value * 16384 rounded to nearest integer

Start of Message

Each message begins with "start of message" indicator (a '!' character), which indicates that the following bytes contain a message.

Binary Message Indicator

Each binary message includes a "binary message" indicator (a '#' character), which indicates that the following bytes contain a binary message.

Binary Message Length

Each Binary message contains a length value (a value from 0-255), which indicates that the number of bytes which follow in the Message Body and Message Termination.

Message ID

The Message ID indicates the type of message, which may be one of the following:

ID	Message Type	Encoding
‘y’	Yaw/Pitch/Roll/Compass Heading Update	ASCII
‘g’	Raw Data Update	ASCII
‘p’	AHRS + Position Data Update	Binary
‘S’	Stream Configuration Command	ASCII
‘s’	Stream Configuration Response	ASCII
‘I’	Integration Control Command	Binary
‘j’	Integration Control Response	Binary

Message Body

The message body differs depending upon the Message Type; the various Message Body specifications are listed below.

Message Termination

The final four bytes of each Serial protocol message contain a Base16 unsigned 8-bit checksum (encoded in 2 bytes as an ASCII 8-bit integer) followed by a carriage return and then a line feed character.

Checksum

The checksum is calculated by adding each byte of the message except the bytes within the Message Termination itself. The checksum is accumulated within an 8-bit unsigned byte.

New Line

The carriage return (0x10) and newline characters (0x13) are present at the end of the message so that when the message is displayed in a console window, a new line will be inserted in the console at the end of the message.

Message Body Definitions

Yaw/Pitch/Roll/Compass Heading Update Message

The Yaw/Pitch/Roll/Compass Heading Update message indicates the navX-MXP current orientation and heading, in units of degrees, as follows:

Byte Offset	Element	Data Type	Unit
0	Yaw	Float	Degrees (-180 to 180)

7	Pitch	Float	Degrees (-180 to 180)
14	Roll	Float	Degrees (-180 to 180)
21	Compass Heading	Float	Degrees (0 to 360)

Raw Data Update Message

The Raw Data update message communicates the raw gyro, accelerometer, magnetometer and temperature data. This data bypasses the Digital Motion Processor, and allows the individual sensors to be used directly without any intervening processing. This can allow the following types of use:

- Access to instantaneous measures of angular velocity in each of the X, Y and Z axes, provided by the tri-axial gyroscopes. Note that the accelerometer and gyroscope data has already had bias calibration applied.

- Additionally, raw magnetometer data is provided. Note that the raw magnetometer data may have already had soft/hard iron calibration applied, if the navX-MXP magnetometer calibration procedure has already been completed.

Byte Offset	Element	Data Type
0	Gyro X (15-bits, signed)	16-bit Integer
4	Gyro Y (15-bits, signed)	16-bit Integer
8	Gyro Z (15-bits, signed)	16-bit Integer
12	Acceleration X (16-bits, signed)	16-bit Integer
16	Acceleration Y (16-bits, signed)	16-bit Integer
20	Acceleration Z (16-bits, signed)	16-bit Integer
24	Magnetometer X (12 bits, signed)	16-bit Integer
28	Magnetometer Y (12 bits, signed)	16-bit Integer
32	Magnetometer Z (12 bits, signed)	16-bit Integer
36	Temperature (Centigrade degrees)	Float

Gyro Device Units: value in deg/sec * gyro full scale range

Accelerometer Device Units: value in G * accelerometer full scale range

Magnetometer Device Units: value in uTesla * .15

AHRS / Position Data Update

Byte Offset	Element	Data Type	Unit
0	Yaw	Signed Hundredths	Degrees
2	Pitch	Signed Hundredths	Degrees

4	Roll	Signed Hundredths	Degrees
6	Compass Heading	Unsigned Hundredths	Degrees
8	Altitude	Signed 16:16	Meters
12	Fused Heading	Unsigned Hundredths	Degrees
14	Linear Accel X	Signed Thousandths	G
16	Linear Accel Y	Signed Thousandths	G
18	Linear Accel Z	Signed Thousandths	G
20	Velocity X	Signed 16:16	Meters/Sec
24	Velocity Y	Signed 16:16	Meters/Sec
28	Velocity Z	Signed 16:16	Meters/Sec
32	Displacement X	Signed 16:16	Meters
36	Displacement Y	Signed 16:16	Meters
40	Displacement Z	Signed 16:16	Meters
44	Quaternion W	Signed Pi Radians	Pi Radians
46	Quaternion X	Signed Pi Radians	Pi Radians
48	Quaternion Y	Signed Pi Radians	Pi Radians
50	Quaternion Z	Signed Pi Radians	Pi Radians
52	MPU Temp	Signed Hundredths	Centigrade
54	Op. Status	Uint8	NAVX_OP_STATUS
55	Sensor Status	Uint8	NAVX_SENSOR_STATUS
56	Cal. Status	Uint8	NAVX_CAL_STATUS
57	Selftest Status	Uint8	NAVX_SELFTEST_STATUS

Stream Configuration Command

By default, navX-MXP begins transmitting YPR Updates upon power up. The Stream Configuration Command is sent in order to change the type of navX-MXP Streaming Update transmitted to the client.

Byte Offset	Element	Data Type
0	Stream Type	8-bit ASCII Character
1	Update Rate (Hz) – Valid range: 4-60	8-bit Integer

Stream Type	Description
‘y’	Yaw, Pitch, Roll & Compass Heading Update
‘g’	Gyro (Raw) Data Update
‘p’	AHRS + Position Data Update

Stream Configuration Response

Whenever a Stream Configuration Command is received, navX-MXP responds by sending a

Stream Configuration Response message, which is formatted as follows:

Byte Offset	Element	Data Type
0	Stream Type	8-bit ASCII Character
1	Gyroscope Full Scale Range (Degrees/sec)	16-bit Integer
5	Accelerometer Full Scale Range (G)	16-bit Integer
9	Update Rate (Hz)	16-bit Integer
13	Calibrated Yaw Offset (Degrees)	Float
20	Reserved	16-bit Integer
24	Reserved	16-bit Integer
28	Reserved	16-bit Integer
32	Reserved	16-bit Integer
36	Flags	16-bit Integer

Flag value	Desription
0, 1	Startup Gyro Calibration in progress
2	Startup Gyro Calibration complete

Integration Control Command

The Integration Control Command is sent in order to cause certain values being integrated on the navX-MXP to be reset to 0.

Byte Offset	Element	Data Type
0	Action	Uint8 (NAVX_INTEGRATION_CTL)
1	Parameter	Uint32

Integration Control Response

The Integration Control Response is sent in response to an Integration Control Command, confiming that certain values being integrated on the navX-MXP have been reset to 0.

Byte Offset	Element	Data Type
0	Action	Uint8 (NAVX_INTEGRATION_CTL)
1	Parameter	Uint32

Register Protocol

In addition to the streaming Serial protocol, navX-MXP may be accessed over the I2C and SPI buses, using a register-based protocol. This page documents the register-based protocol used on both the I2C and SPI bus.

Register-based protocol overview

Unlike the streaming Serial protocol, which periodically sends out updates messages whenever new data is available, the register based protocol is a “polled” interface, in that the consumer of the navX-MXP data (in this case referred to as a “bus master”) can request data from the navX-MXP at any time. At the same time, when using the register-based protocol the bus master does not know when new data is available.

To help this situation, a timestamp – which is updated whenever new data is available – is made available. Therefore, the general approach to ensure each new data sample is retrieved is to regularly (at the navX-MXP update rate) retrieve both the timestamp and the data of interest), and if the timestamp differs from the previous timestamp by the update rate as expressed in milliseconds, then the data sample just retrieved is current, and no data has been missed.

I2C Overview

The navX-MXP responds to 7-bit address 50 (0x32) on the I2C bus. If accessing the navX-MXP via the MXP I2C bus, ensure that no other device at that address is on the same bus.

The navX-MXP I2C bus operates at a speed up to 400Khz.

When accessing the navX-MXP via the I2C bus, this following pattern is used:

– The I2C bus master sends the navX-MXP I2C address. The highest bit is set to indicate the bus master intends to write to the navX-MXP. If the highest bit is clear, this indicates the bus master intends to read from the navX-MXP.

– The I2C bus master next sends the starting register address it intends to write to or read from.

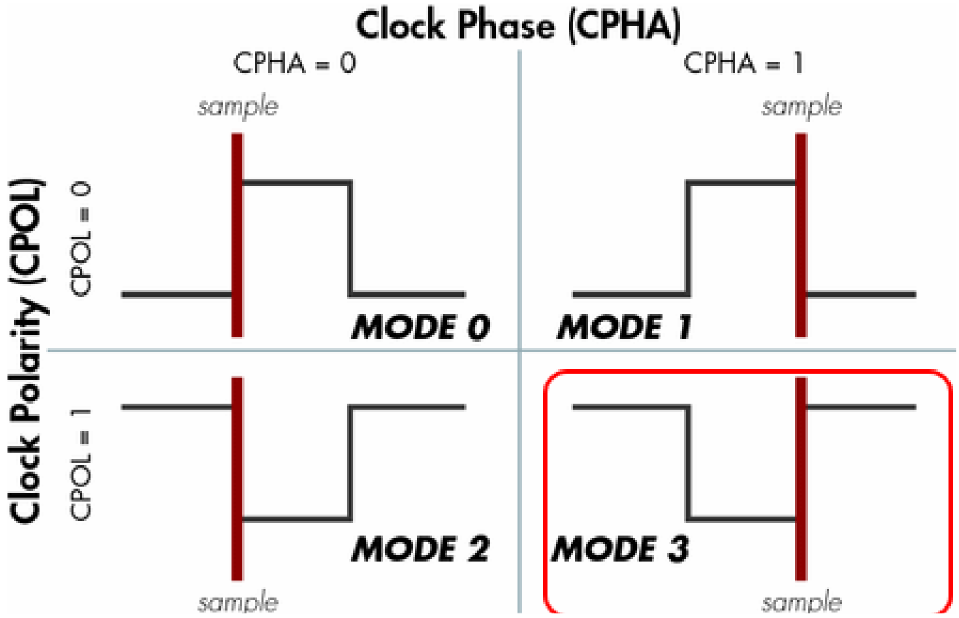
– The I2C bus master next initiates I2C bus transactions. The navX-MXP supports I2C burst mode for read operations, therefore the navX-MXP will respond with register values as long as the I2C bus master continues the transaction, and as long as the last register address has not yet been reached.

If instead the I2C bus master intends to write data to a writable navX-MXP register, the bus master should transmit the new register value immediately after sending the register address.

SPI Overview

The navX-MXP SPI data is communicated as follows:

> – Most-significant bit first – Maximum Bitrate: 2mbps – Clock Polarity/Clock Phase – Mode 3



When accessing the navX-MXP via the SPI bus, this following pattern is used:

– When the SPI bus master is not communicating with the navX-MXP, the SPI bus master must hold the chip select (CS) line high.

– The SPI bus master lowers the CS line.

– The SPI bus master next transmits the register address it intends to read from or write to. If writing, the upper bit (0x80) must be set; if this upper bit is clear, this indicates a read transaction.

– If the SPI bus master is reading, it next transmits the count of registers it wishes to read from. This count must be at least 1, and must be not exceed the maximum register address less the requested register address.

– If the SPI bus master is writing, it transmits the register value to be written to the specified register address.

– The SPI bus master finally transmits an 8-bit CRC (see CRC calculation section below) which is calculated on the register address and count values previously transmitted.

– If the SPI bus master is writing, it raises the CS line to complete the write sequence.

– If the SPI bus master is reading:

– The SPI bus master raises the CS line.

– The SPI bus master delays for 200 microseconds, giving the navX-MXP sufficient time to prepare for the upcoming SPI bus transaction.

– The SPI bus master lowers the CS line.

– The SPI bus master initiates a series of SPI bus transactions, where the number of individual 8-bit transfers is equal to the count previously specified, plus one additional transfer for a CRC value transmitted by the nav-MXP.

– The SPI bus master raises the CS line to complete the read sequence.

CRC Calculation

The SPI protocol requires use of a cylic redundancy check (CRC) allowing the detection of corrupted data transmission over the high-speed SPI bus. Each SPI protocol message must end with a byte containing the CRC value.

The SPI protocol uses a 7-bit CRC with a polynomial value of 0x91.

For example code to calculate the CRC value, please see Line 445 of the IMURegisters.h source code.

navX-MXP Register Data Types

All multi-byte registers are in little-endian format.
All registers with 'signed' data are 2's-complement.

Data Type	Range	Byte Count
Unsigned Byte	0 to 255	1
Unsigned Short	0 to 65535	2
Signed Short	-32768 to 32768	2
signed hundredths	-327.68 to 327.67	2
Unsigned Hundredths	0.0 to 655.35	2
Signed Thousandths	-32.768 to 32.767	2
Signed Pi Radians	-2 to 2	2
Q16.16	-32768.9999 to 32767.9999	4
Unsigned Long	0 to 4294967295	4

*Unsigned Hundredths: original value * 100 to rounded to nearest integer
*Signed Hundredths: original value * 100 rounded to nearest integer
*Signed Thousandths: original value * 1000 rounted to nearest integer
*Signed Pi Radians: original value * 16384 rounded to nearest integer

navX-MXP Register Map

Address (Hex)	Name	Access	Range/Data Type
0x00	WhoAmI	Read-only	50 (0x32): navX-MXP
0x01	Board Revision	Read-only	Unsigned byte
0x02	Firmware Major Version	Read-only	Unsigned byte
0x03	Firmware Minor Version	Read-only	Unsigned byte
0x04	Update Rate	Read/write	Unsigned byte (Hz)
0x05	Accel FSR	Read-only	Unsigned byte (Degrees/Sec)
0x06-0x07	Gyro FSR	Read-only	Unsigned short(G)
0x08	Operational Status	Read-only	See NAVX_OP_STATUS
0x09	Calibration Status	Read-only	See NAVX_CAL_STATUS
0x0A	Self-test Status	Read-only	See NAVX_SELFTEST _STATUS
0x0B	Capability Flags (low)	Read-only	See NAVX_CAPABILITY
0x0C	Capability Flags (high)	Read-only	“”
0x0D-0x0F	n/a	Read-only	

Open-source Hardware/Software

The navX-MXP project is completely open source, including schematics, firmware and design files for an enclosure.

These sources are available online at the navX-MXP Github Repository.

Firmware Customization

The navX-MXP was developed/debugging using the following software tools, which (with the exception of the Debugging hardware) are open-source or freely-available. The only component you may want to purchase is the inexpensive ST-LINK/V2 JTAG programmer/debugger described below.

Install Compiler

Install the free Code sourcery G++ Lite compiler for the ARM Cortex processor used in the nav-MXP.

Download URL:

https://sourcery.mentor.com/sgpp/lite/arm/portal/subscription?@template=lite

After installing, the compiler is installed into folder (32-bit Windows)

C:\Program Files\CodeSourcery\Sourcery G++ Lite

For 64-bit Windows, it is installed into:

C:\Program Files (x86)\CodeSourcery\Sourcery G++ Lite

Add the path to the “bin” director underneath the Code Sourcery G++ Lite installation directory, so that the compiler is on the path.

Install Eclipse IDE

Install the Eclipse IDE for C/C++ Developers at the following download URL:

https://www.eclipse.org/downloads/

If you already have eclipse installed w/out the C/C++ Development tools (CDT) you will need to install them, too:

CDT 8.1.2 (or later)

A URL for this software, including the CDT, is at:

https://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/junosr2

Install the Zylin embedded CDT Plugin

This is installed from within Eclipse, since it is an Eclipse Plugin. If you are unfamiliar with installingn Eclipse plugins, please visit this URL for more information on the process:

https://wiki.eclipse.org/FAQ_How_do_I_install_new_plug-ins%3F

Zylin Plugin Update URL: http://opensource.zylin.com/embeddedcdt.html

Import the project into Eclipse

Open up eclipse, and import the project which is contained in the navX-MXP stm32 directory in the Github repository.

Building

In Eclipse, select Project->Build. You might find it necessary to Project->Clean first to remove old build output files.

The output of the build will be placed in the stm32/Debug directory. The extension of the file will be .hex (Intel HEX Binary format).

You can either download this file via the ST Microelectronics DfuSe utility, or you can download it via the ST-LINK/V2 adapter (see instructions on debugging below).

In-Circuit Debugging (optional, but highly recommended)

ST-LINK/V2

ST-LINK/V2 JTAG in-circuit debugger was used, this is very inexpensive and works very well.

The ST-LINK/V2 can be purchased at www.digikey.com (among others) for approximately $40.

Additional utilities for the ST-LINK/V2 (for windows) are available on the STM website.

Connecting the ST-LINK/V2 to the navX-MXP Circuit Board

You will need to solder a 4-pin header to the navX-MXP board in order to connect debug on the navX-MXP's STM32F411 microcontroller. Then, you will need to connect 4 wires from the connector to the corresponding location on the ST-LINK/V2 connector. Instructions on how to do this can be found at the following URL:

https://www.micromouseonline.com/2011/11/05/stlink-swd-for-stm32/

Install OpenOCD

In order to interface eclipse with the ST-LINK/V2 JTAG in-circuit debugger, the OpenOCD Server is used.

OpenOCD, version 0.9.0 (windows version available at

https://www.freddiechopin.info/en/articles/34-news/92-openocd-w-wersji-080)

OpenOCD includes A gdb server that runs with the ST-LINK/V2.

- MPORTANT NOTE: THe 0.9.0 release of OpenOCD contains a bugfix; earlier releases of OpenOCD from cannot communicate correctly with the STM32F411 microcontroller used in the navX-MXP. If you are not able to acquire this release of OpenOCD, please contact support@kauailabs.com for information on how to proceed.

Configure Eclipse to run OpenOCD

Run->External Tools->External Tools Configuration…

Add a new configuration, name it "OpenOCD"

In the "main" tab, under Location, provide the path to the location of Open OCD. E.g., C:\OpenOCD\openocd-09.0\bin-x64\openocd-x64-0.9.0.exe

In the same "main" tab, in the Arguments window, enter the following:

-f C:\OpenOCD\openocd-0.9.0\scripts\interface\stlink-v2.cfg -f C:\OpenOCD\openocd-0.9.0\scripts\target\stm32f4x_stlink.cfg

To start the OpenOCD Server, Select Run->ExternalTools?->OpenOCD (where OpenOCD is the name provided earlier on the "main" tab)

Once the OpenOCD Server has started, the debug session can be started.

Starting a Debug Session

6b) To start a debug session, first create a debug configuration:

- Select Run->Debug Configurations…
- Select “Zylin Embedded Debug (Cygwin)”

Then, add a new configuration (e.g., (navX-MXP OpenOCD Debug Session”); the new configuration will be a child node of Zylin Embedded Debug (Cygwin)

On the Debugger Tab:

- Set GDB Debugger to arm-none-eabi-gdb
- Set GDB Command File to <navx-mxp-distribution_directory\stm32\gdb\nav10.script
- Select “Verbose console mode”

- NOTE: You will need to edit the nav10.script file to reference your particular directory path to the “navx-mxp-distribution-directory” you unpacked the navx mxp distribution .zip into.

Once the debug configuration is created, and the open ocd session is started, start debugging via Run->Debug

navXUI Customization

The navXUI Source Code is Open-Source and can be customized using the following instructions:

- Download and install the free . NOTE: the current navXUI code is compatible with version 3.0beta5 of the Processing development environment.

- Checkout the navX MXP source code on GitHub.

- Copy the contents of the navX-MXP source code’s ‘processing’ directory to <User Directory>\Processing directory.

- Open the Processing IDE and then open the navXUI sketch via the File->Sketchbook menu.

- Compile/Run the navXUI by selecting the Sketch->Run menu.

If your computer has more than one serial port, you will need to select the appropriate serial port (corresponding to the USB serial port navX-MXP is connected to) from the COM port selection drop-down list in the top-right of the navXUI display.

Technical References

The references on this page are provided to help students gain a deeper understanding of the algorithms, technologies and tools used within the navX-MXP and other Inertial Measurement Unit (IMU) and Attitude/Heading Reference System (AHRS) products. Additionally, links to other notable open-source works which could perhaps be adapted to work on the navX-MXP are included.

Algorithms

Complementary Filter Algorithm

Magnetometer Calibration and Tilt Compensation

Implementing Positioning Algorithms Using Accelerometers

Technology

MEMS

MEMS Gyroscopes

Magnetometers

Tools

Eagle PCB Tutorial

Other Notable Open Source IMU Projects

AeroQuad