The SensorPro

An Electronics Project for FTC

2015

By: Michael Huang, Team 9048 Philobots, Westlake High School, Austin, TX

Sponsors: Westlake High School Robotics

ABSTRACT

The Sensor Pro Board is an extension board that allows NXT users to efficiently use sensors that are not offered by Lego. It consists of a SuperPro Board, an Arduino Nano, a built-in Gyro, and pins that allow for additional sensor attachment.

Before the Sensor Pro Board, many FTC teams had to make their own extension setup, often using less efficient ways to send and receive data, and being limited by the Super Pro Board’s lack of I2C pins. The goal of this project was to ensure the most efficient data transfers and compatibility with I2C sensors.

The SuperPro board is functional and is currently attached to the 9048 PhiloBot’s robot. The robot gets all of its angle measurements from the gyro sensor built in the board, and uses some additional sonar sensors to detect distance, and for this year’s game, the center goal position.

INTRODUCTION

FTC rules only allow Lego certified sensors to be connected directly to NXT brick. However, there are great advantages to use other kinds of sensors and devices.

- It is better to use smart sensors like Invensense MPU6050, which are often more powerful yet less expensive than a Lego Certified sensors like the HT Gyroscope.
- The NXT only has four ports for motor controllers, servo controllers, and sensors, which usually leaves two ports at the most for sensors, or a max of two sensors (each lego sensor requires one port). There are times when it is useful to have more than two sensors from which to collect data.
- High Technic sensor multiplexer can expand from one port to four ports, but only uses lego sensors, and costs about $50.
- Many teams have used the HT SuperPro prototyping board to attach analog sensors, simple digital sensors, LEDs, etc. The SuperPro itself connects directly to a Lego NXT sensor port. However, there are no communication ports such as I2C exposed, so smart sensors or devices, or even Lego certified sensors that use I2C ports, are not supported.

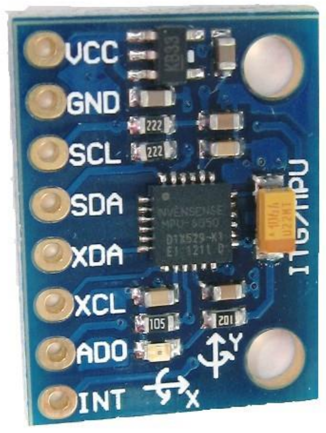
HT SuperPro prototyping board- A0-A4: analog input, O0-O1: analog output, B0-B7: digital Input/Output (bits of 0/1), S0-S3, WR, RD: strobe controls

The SuperPro has a port to connect directly to the NXT through an NXT cable.

Arduino Nano(Microprocessor)- A0-A7: analog input/output, D0-D13: digital input/output

The Arduino Nano is programmable by computer via a mini-USB cable.

D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
ICSP
ARDUINO.CC
TX
RX
PWR
L
RESET
USA
2011
ARDUINO NANO V3.1
GRAVITECH.US
1
D13 3V3 REF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN



MPU 6050(Gyro Sensor)- SCL, SDA: I2C communication

The MPU 6050 can detect angles in pitch, roll, and yaw(three dimensions), as well as acceleration. Unlike the Lego gyro sensor, the MPU is constantly callibrating, reducing angle drift significantly.

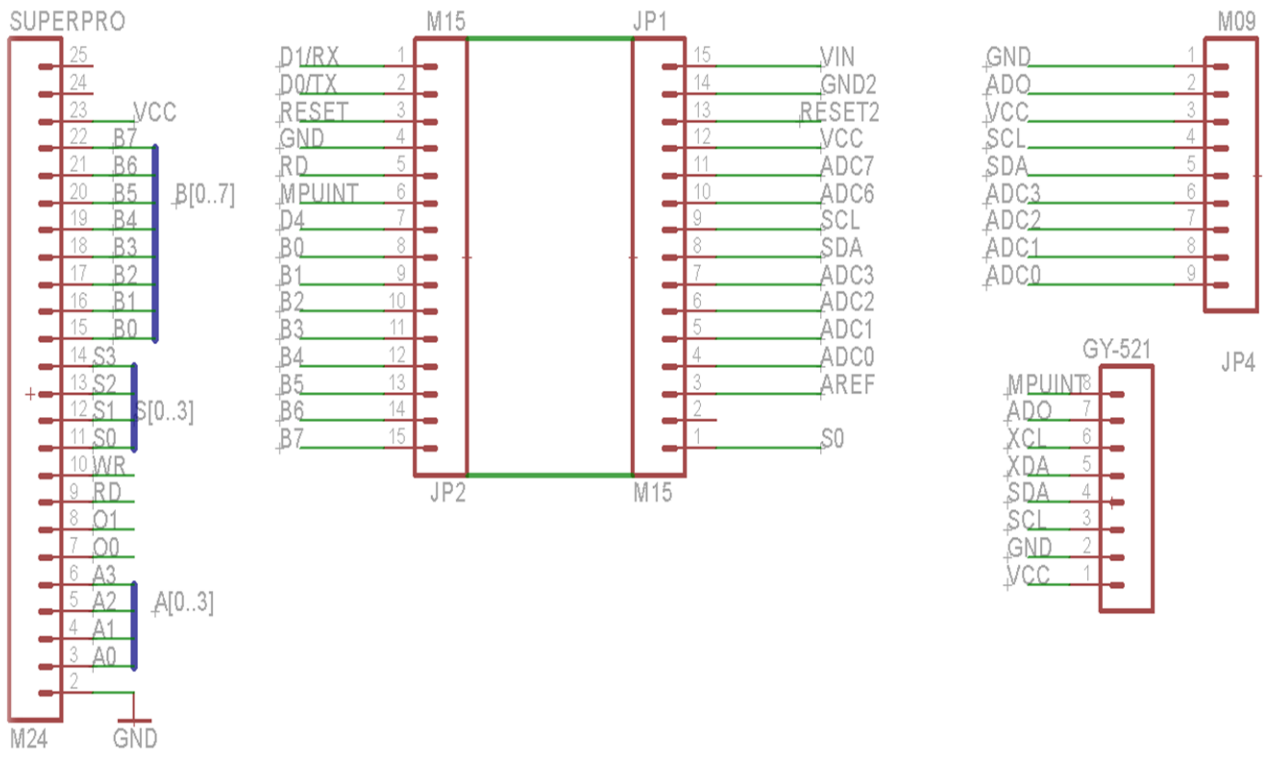
hc-sr04(Sonar Sensor)- Trig: Triggers the initial sonar signal if set to high for more than 10us, Echo: Echo outputs a pulse after the sonar bounces back from objects, duration of high level of the pulse equal to total travel time of the sonar signal, from which distance to object is calculated.

The hc-sr04 has a much more accurate and larger range than the HT sonar sensor.

NEW APPROACH

Pins B0-B7 on the superpro will act as a parallel bus enabling reading/writing 1 byte at a time from/to Arduino into/from the NXT brick (each pin is a bit, eight makes a byte), while the Arduino continuously aggregates data from sensors. A strobe signal (RD pin) is wired from the SuperPro to the interrupt pin (D2) on the Arduino, which signals a read request to the Arduino. The strobe signal (RD) is automatically triggered by superpro when a read request is sent by the NXT. After a read request, the Arduino would return sensor data in an interupt handler. Another strobe control (pin S0) is wired to a digital pin (D13) on the Arduino; setting this pin to low would indicate an error in data transit and resets communication. Each interrupt/communication session transfers a number of bytes of real sensor data (called data frames, see below in the software overview section) and one parity byte to detect possible transfer errors at the end. An error, even though it is rare, is detected by the parity byte. The parity byte is the exclusive or (XOR) of all data bytes, and is sent at the end of each frame. Errors are detected when the parity byte calculated on the NXT does not match the parity byte received from the Arduino. The communication between the Arduino and SuperPro is controlled by the hardware per byte level because the NXT can only read one byte at a time from the SuperPro.

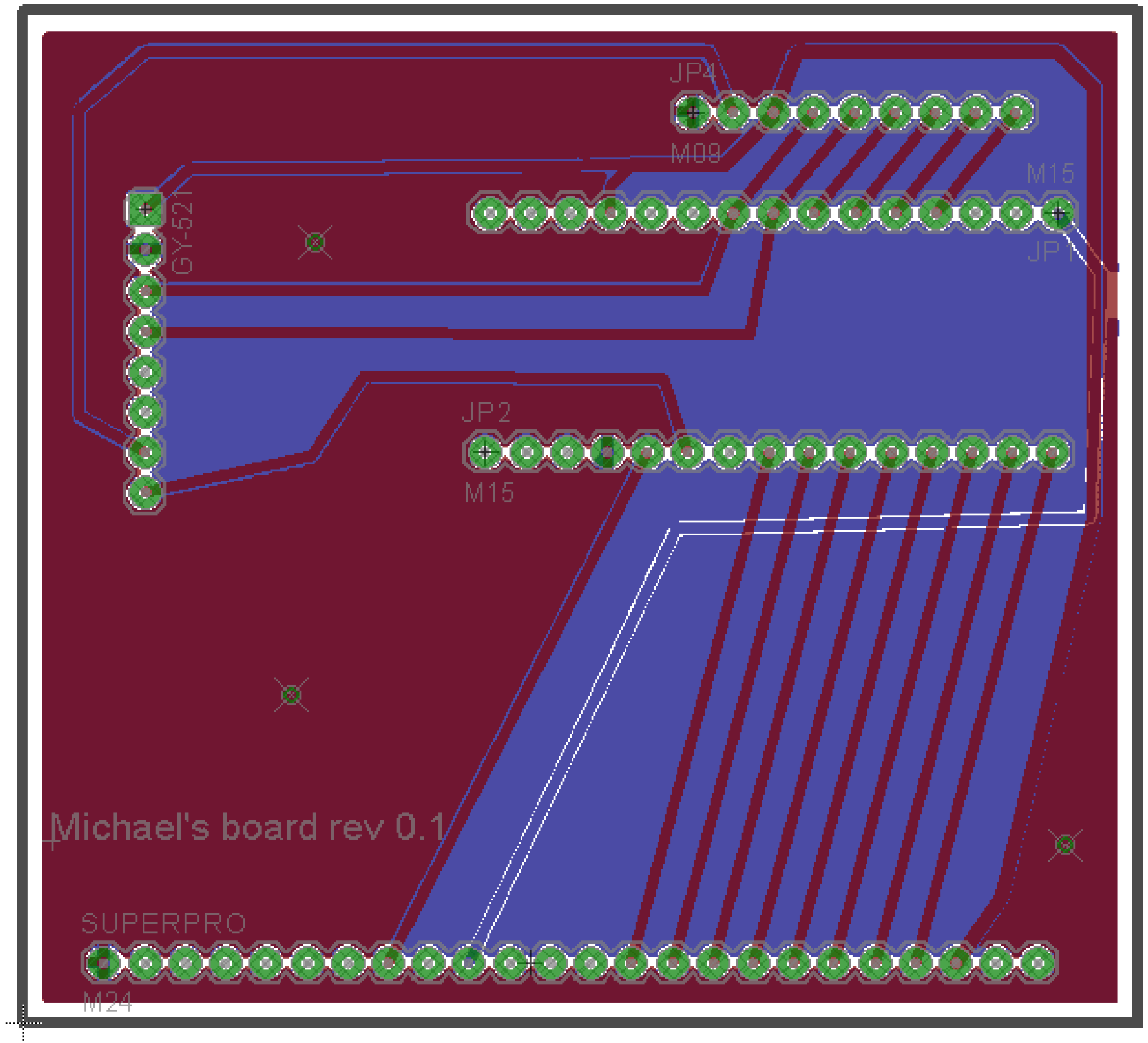
THE BOARD



Schematic of the custom board.

To minimize the risks of developing a board from the ground up, we decided to create a version using an Arduino as the main controller because I was already familiar with the controller through Chap Research (see chapreseach.com). We selected the Arduino Nano because it is inexpensive and many others have used it with the GY-521 (an MPU6050 break-out board) successfully. This saved us a lot of time during testing and making the Arduino and MPU work together.

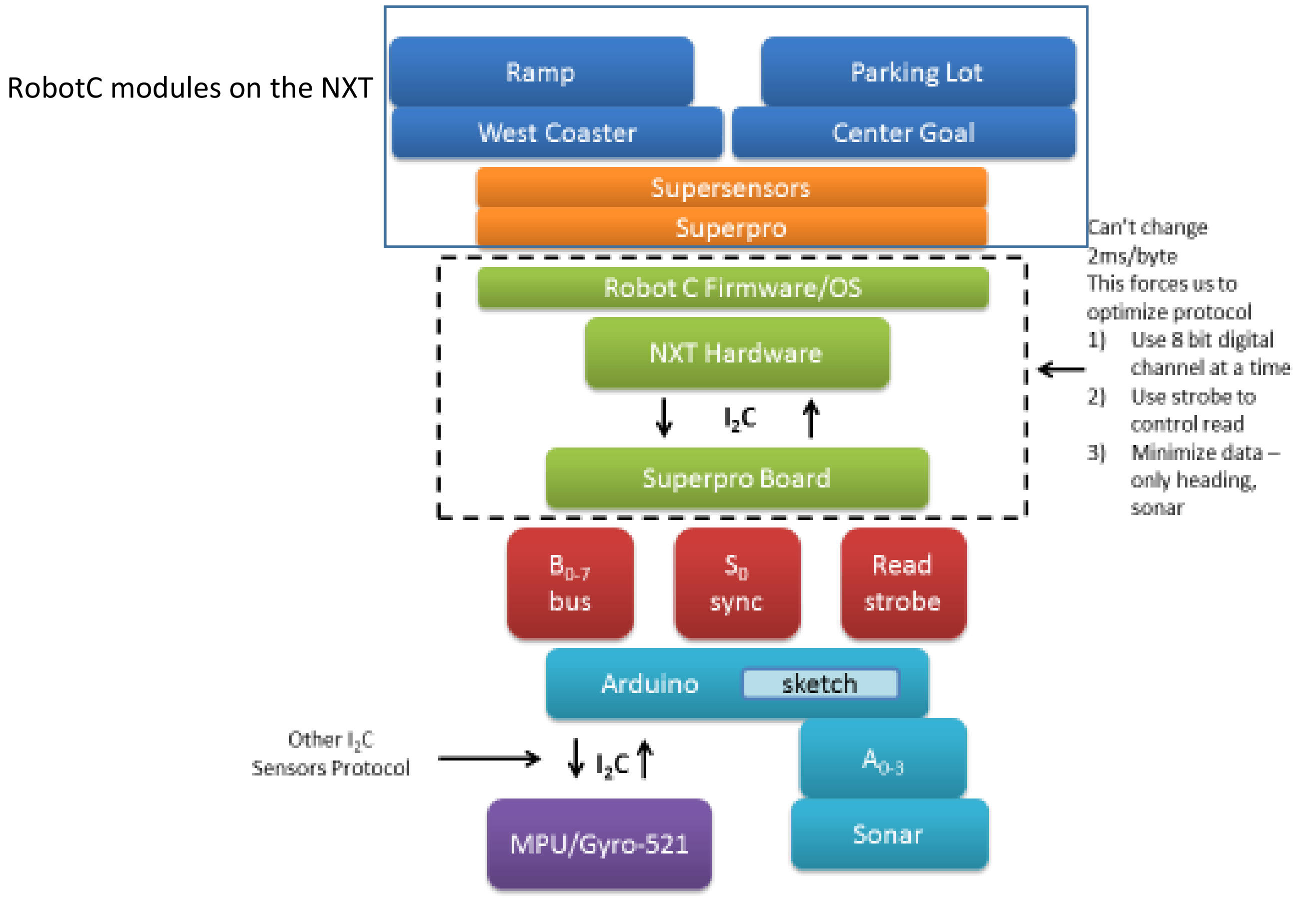
As a result, the board design is very simple. We created vias for pins of the Arduino Nano (in the middle), SuperPro Board (on the left) and GY-521 (at the lower right). Also, 4 analog pins (ADC0-3) and IC2 pins (SCL, SDA) are exposed as headers (upper right corner) so that we can plug in other analog sensors or IC2 sensors as needed (in 9048's case, an extra set of sonar sensors).



Layout of the Board

To simplify the layout, we arranged the SuperPro and Arduino Nano in such a way that most of the connection lines are parallel without crossing each other. In this way, we kept most of the connections on the top layer. Only a few lines (VCC, A0, AD0) are on the bottom layer. This was not optimized for space, which we may need to do for production. One of our mentors aided in the manufacture of one prototype version shown in the picture above at home by an etching method. We tested and verified that it worked well, then looked for professional manufacturing. We contacted Advanced Circuits, a PCB manufacturing company. They sponsor us, and manufactured our board at no cost.

SOFTWARE OVERVIEW


RobotC modules on the NXT
Ramp
Parking Lot
West Coaster
Center Goal
Supersensors
Superpro
Robot C Firmware/OS
NXT Hardware
I2C
Superpro Board
Can't change 2ms/byte This forces us to optimize protocol
1) Use 8 bit digital channel at a time
2) Use strobe to control read
3) Minimize data – only heading, sonar
B0-7 bus
S0 sync
Read strobe
Arduino
sketch
Other I2C Sensors Protocol
I2C
A0-3
MPU/Gyro-521
Sonar

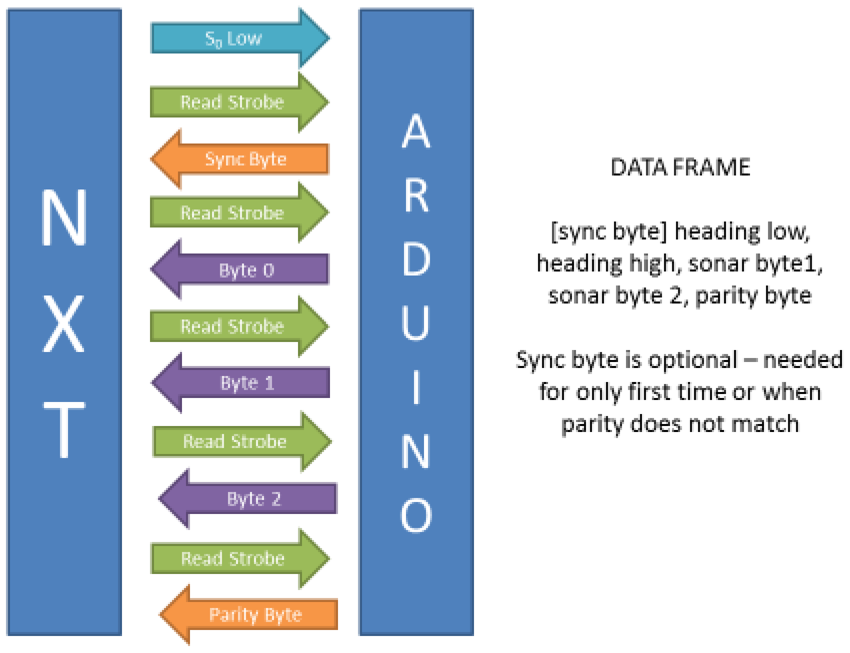
Highlevel Software Architecture

Shown in the above diagram, our software can be characterized by its layered design. Each box in the diagrahm represents separate building blocks in the overall structure. Each block has defined interfaces for modules above its layer. At the highest level, RobotC modules on the NXT communicate through the hardware and firmare to the Arduino sketch on the Arduino Nano.

RobotC Modules

- SuperPro- This module is a modified SuperPro drive to enable the strobe signal, since the standard drive code does not enable the strobe. It manages the communication between the NXT and the SuperPro at the byte level, providing APIs setStrobe(), ReadIO() for upper layers module to enable strobe, and read a byte respectively.
- Supersensors- This module handles the data communication protocal. It runs a loop to read data frames (defined in picture below) from the SuperPro, checks the parity byte for each frame, and re-syncs communication in case of a parity mismatch. It also provides APIs for upper layer modules to read the heading, and sonar detected distances.

- WestCoaster- This module manages our west coast drive system, controlling robot movement. It provides high level commands such as moveStraightWithMPU, turnWithMPU for our autonomous programs.
- Center Goal- This module contains methods that involve the sonar sensor, such as determine goal position.
- Ramp- This module uses methods from Westcoaster and defines some of its own to score ball in autonomous. It is the code that executes our ramp autonomous strategy.
- Parking Lot-This module uses methods from Westcoaster and Center Goal to determine the goal position, and manuever to knock down the kickstand. This is the execution code for the parking lot autonomous.


NXT
ARDUINO
S0 Low
Read Strobe
Sync Byte
Read Strobe
Byte 0
Read Strobe
Byte 1
Read Strobe
Byte 2
Read Strobe
Parity Byte
DATA FRAME
[sync byte] heading low, heading high, sonar byte1, sonar byte 2, parity byte
Sync byte is optional – needed for only first time or when parity does not match

Data Frame Structure

Arduino sketch

The Arduino sketch programming model is similar to RobotC in that both are C-like languages, and both have a main loop. Arduino differs in that it is much more flexible and has a community that has already developed libraries for a wide variety of sensors. These libraries were instrumental to the SensorPro, as we used them to quickly integrate the sensors into our existing code.

We used a MPU6050 example from the I2CDevLib to run the initial tests for the gyro sensor. After the wiring was verified, we modified the example code to fit our needs. The main modifications are the following:

- As mentioned earlier, the SuperPro RD pin is wired to the Arduino D2 digital pin, which is the Arduino’s external interrupt 0. We attached an interrupt routine to the interrupt 0, in which we simply set a flag (nxtIsReading) to indicate the NXT’s reading request.

- The MPU6050 also has an interrupt pin to indicate data that is ready for transfer. We wired this interrupt to the Arduino pin D3 (external interrupt 1). Similar to the interrupt 0, the interrupt routine for this pin also sets up a flag (mpuInterrupt) to indicate that the MPU data is ready for reading.
- In the main loop, the two interrupt flags are checked; if nxtIsReading is set, a subroutine is called to send data through the parallel bus to the NXT in the form of a data frame (see diagram below). For the case when mpuInterrupt is set, the sample code is re-used to read the data from the MPU and store it in holding arrays.

One thing to note here is that interrupt routines are very fast, and all data processing happens in the main loop. Therefore, data synchronization for multithreading is not needed. Also, the interrupt 0 has a higher priority than the interrupt 1. Because of this, the NXT's request is always handled as a higher priority.

When the HC-SR04 sonar sensors was added, a library, called NewPing Lib provided example code that was used to test the sensors and their wiring. Then, like before, part of the example code is integrated with the existing program that handled the MPU and NXT readings. The main touch points in this case are as following:

- Interrupts were needed again to detect the rising and falling edges of the "Echo" signal (see above) from the sonar sensors. The NewPing lib had a beta version with support for interrupts, which was not turned on. Minimal modifications were made to the lib and tested to enable this feature. The results indicated that it worked as expected.
- As all the external interrupt pins were used for MPU and NXT, we had to use software interrupts called pin change interrupts. For this, we found another library called PinChangeInt. We simply modified some of the NewPing beta code to call the APIs defined by PinChangeInt. It worked as expected.

RESULTS

The Sensorpro currently functions as expected on our robot, providing data from the gyro and sonar sensors to the lego NXT. The NXT uses the data from the gyro to move straight by correcting any angular drift while the robot is moving straight, as well as by controlling turning accurately. Using the sonar sensor data, the NXT can determine the center field structure orientation and decide which path to take to knock down the kickstand. Without the board, our autonomous program would not be able to function as well.

The custom board also accomodates greater wiring flexibilities for our robot. We were able to leave more ports open for motors and servos (only one port was needed for all sensors). The cables used to wire the industrial sensors to the board do not have the commercial length limits the NXT cables had.

The total cost for the board is around $75 (including $50 for SuperPro), much less than the $193 that Hitechnic mulitplexer, sonars, and gyro set would cost.

FUTURE PLANS

A major vision for the SensorPro board is to become a product that other FTC teams could use as well to bring cheaper and more efficient sensors onto the FTC playing field. In order for this to happen, the SensorPro board should be further generalized, allowing for a wider range of custom sensors to be used. Also, an easy interface for custom sensors should be developed, and a public library of code should be made accessible for quick installation and coding on the NXT (or other Robot controllers).

However, as next year FTC is making major control system changes (no more NXT), the future for this version of the board is uncertain. The elimination of the SuperPro board and the NXT in our board would greatly increase the rate of data transfer, as the NXT can only read one byte at a time from the SuperPro. However, the board's role may already be taken with FTC's introduction of the Advanced Sensor Module - the replacement for the SuperPro board. Ultimately, further research into next year's control components will be needed to know for certain the future of the SensorPro board. Regardless, the basic concept of aggregating sensor data on a hub separate from the main Robot controller may still be a valuable idea worth exploring.

REFERENCES AND SPONSORS

Advanced Circuits: http://www.4pcb.com/

We are fortunate to have Advanced Circuits to manufacture the boards for free. We greatly appreciate their support.

WESTA: http://westaaustin.org

I2C Device Library: https://github.com/jrowberg/i2cdevlib

NewPing lib: https://code.google.com/p/arduino-new-ping/

PinChangeInt lib: https://github.com/GreyGnome/PinChangeInt