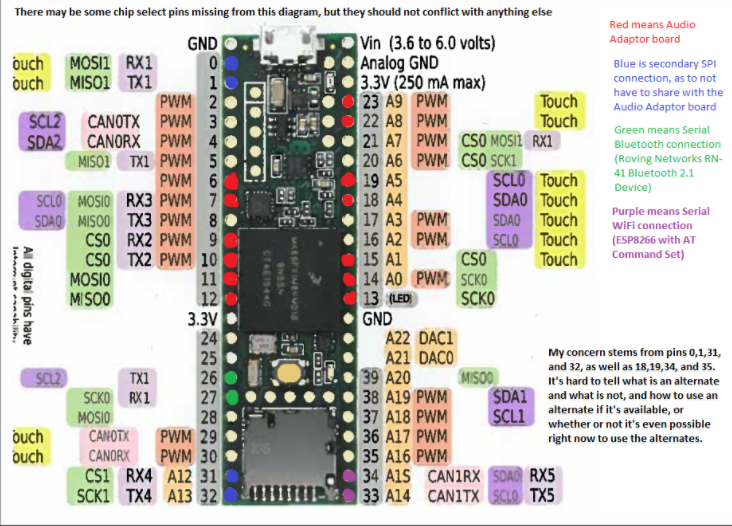
Control System with Teensy 3.6 Design Document

1. CAN BUS Communication
   1. CAN BUS Configuration on Teensy 3.6



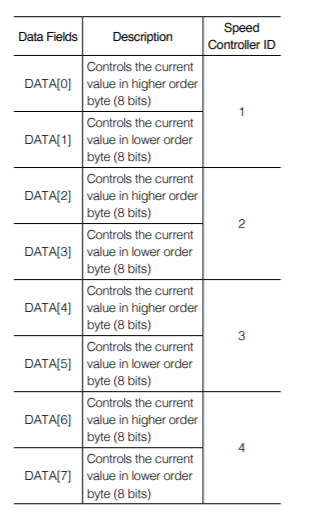
The Teensy controller has a 32 bit 180 MHz ARM Cortex-M4 processor, which is powerful enough to communicate through CAN BUS with a baud rate of 1Mbps. It has two pair CAN TX/RX ports that the user can select, which means it has a built-in CAN controller that will convert the signal (16bit output) into TX/RX form. The relevant driver is organized in the Arduino library format.

You can get the teensy loader exe here <https://www.pjrc.com/teensy/loader_win10.html>, and Teensyduino <https://www.pjrc.com/teensy/td_download.html>.

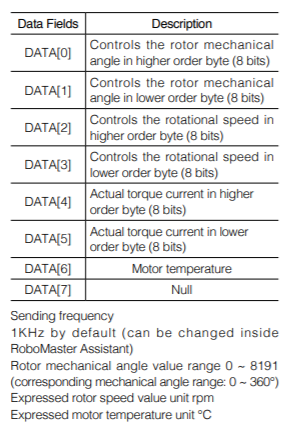
The CAN BUS library used in this document is [in this GITHUB page](https://github.com/pawelsky/FlexCAN_Library). The CAN transceiver is based on the NXP TJA1050 chip and can be purchased from AMAZON:

<https://www.amazon.com/Gikfun-TJA1050-Controller-Interface-Arduino/dp/B01GPYTAJ6/ref=sr_1_fkmr0_1?s=electronics&ie=UTF8&qid=1521493226&sr=1-1-fkmr0&keywords=NXP+TJA1050>

All control commands are generated based on DJI’s C620 Brushless DC Motor Speed Controller, which has two main CAN identifiers (0x 200 and 0x 1FF), and up to eight feedback data CAN identifiers. The detailed information is showing below:



t



*The control commands will send to the ID 0x200 to control the four driving wheels, and the data structure will contain 4 pairs of 16bits signal (8 pairs of 8bits signal), which indicate a value from -16384 ~ 0 ~ +16384 (-20A~+20A).*

*The feedback data are retrieved from ID 0x201 ~ 0x204 respectively, the information content is showing on the table.*

*(NOTE: All communication must done in 1Mbps, which is already set in the ESC hardware.)*

1. Control Logic

The timer interrupt is running in the controller with a step size of 1 millisecond, which will keep updating the signal received from the transmitter (remote control). The analog inputs are: Channel 0 ~ 3 joystick value, and the controller will be responsible for convert those values into a position or velocity command and solve for each motor’s desired speed. The meantime, our robot controller will receive the encoder/sensor reading from the motor and IMU in real-time, then use PID control to ensure an accurate speed control is achieved. A detailed code explanation will be delivered later.