CAN NODE SYSTEM

Arduino DroneCAN

A library to send CAN messages compatible with Ardupilot and PX4 UAV autopilots.

About

We made a library to make DroneCAN development as simple as possible. the Arduino DroneCAN repository allows you to get started with Ardupilot/PX4 compatible CAN messages and functionality straight out of the box using Beyond Robotix CAN node hardware.

This repository is still in development and the API is subject to change in future versions

Github Repository:

GitHub - BeyondRobotix/Arduino-DroneCAN
GitHub

The Arduino DroneCAN project reduces using DroneCAN down to creating an object, initialising the object, and one method to call in your loop function. Any messages on the bus can be accessed in "onTransferReceived" and messages can be sent in the loop() function.

```
#include <Arduino.h>
#include <dronecan.h>

static bool shouldAcceptTransfer(const CanardInstance *ins,
                                 uint64_t *out_data_type_signature,
                                 uint16_t data_type_id,
                                 CanardTransferType transfer_type,
                                 uint8_t source_node_id);
static void onTransferReceived(CanardInstance *ins, CanardRxTransfer *transfer);

DroneCAN dronecan;

void setup()
{
    // Do your usual Arduino sensor initialisation here
    dronecan.init(onTransferReceived, shouldAcceptTransfer);
}

void loop()
{
    // Read your sensor data, and compose dronecan messages here
    dronecan.cycle();
}

void onTransferReceived(CanardInstance *ins, CanardRxTransfer *transfer)
{
    // describes what to do with received DroneCAN messages
}

bool shouldAcceptTransfer(const CanardInstance *ins,
                          uint64_t *out_data_type_signature,
                          uint16_t data_type_id,
                          CanardTransferType transfer_type,
                          uint8_t source_node_id)
{
    // can be used to filter what is passed to onTransferReceived
}
```

onTransferRecieved and shouldAcceptTransfer are required to be in this format to be compatible with Canard, the underlying DroneCAN infrastructure. Even if you don't end up using them. They could be defined in another file or the declaration could be avioded if they were placed before setup().

Installation

There are a few ways of working with Arduino Code, we recommend the following steps for seamless integration with the project.

1. Install Visual Studio Code https://code.visualstudio.com/download ↗

2. Install the PlatformIO extension https://platformio.org/install/ide?install=vscode ↗
3. Download the code
    a. Clone the repository (note, getting the sub modules is important!)

    ```
    git clone --recurse-submodules https://github.com/BeyondRobotix/Arduino-
    DroneCAN.git
    ```

    b. Or, https://github.com/BeyondRobotix/Arduino-DroneCAN/releases/latest ↗ - downloading the zip of the released source code
4. Connect your STLINK to your CAN node
5. Press upload!

Detailed Example

DroneCAN code can be daunting, even with the assistance of Arduino DroneCAN. We've provided an example in /src/main.cpp which reads sensor data, sends this data over CAN and also reads in some data on the CAN bus. We'll break down the loop() function below:

```
void loop()
{
    const uint32_t now = millis();

    // send our battery message at 10Hz
    if (now - looptime > 100)
    {
        looptime = millis();

        // collect MCU core temperature data
        int32_t vref = __LL_ADC_CALC_VREFANALOG_VOLTAGE(analogRead(AVREF),
LL_ADC_RESOLUTION_12B);
        int32_t cpu_temp = __LL_ADC_CALC_TEMPERATURE(vref, analogRead(ATEMP),
LL_ADC_RESOLUTION_12B);

        // construct dronecan packet
        uavcan_equipment_power_BatteryInfo pkt {};
        pkt.temperature = cpu_temp;

        // boilerplate to send a message
        uint8_t buffer[UAVCAN_EQUIPMENT_POWER_BATTERYINFO_MAX_SIZE];
        uint32_t len = uavcan_equipment_power_BatteryInfo_encode(&pkt, buffer);
        static uint8_t transfer_id;
        canardBroadcast(&dronecan.canard,
                        UAVCAN_EQUIPMENT_POWER_BATTERYINFO_SIGNATURE,
                        UAVCAN_EQUIPMENT_POWER_BATTERYINFO_ID,
                        &transfer_id,
                        CANARD_TRANSFER_PRIORITY_LOW,
                        buffer,
                        len);
    }

    dronecan.cycle();
    IWatchdog.reload();
}
```

The following code sets us up to run our if statement at 10Hz, checking if 100ms has passed since our last call. This is because we want to only send our CAN message at 10Hz and we want to call our "dronecan.cycle()" function to be called as much as possible to ensure CAN messages are send and received in a timely manner. We want to avoid delay() as much as possible!

```
const uint32_t now = millis();

    // send our battery message at 10Hz
    if (now - looptime > 100)
    {
        looptime = millis();
```

Next, we want to read in our sensor value. This could be from anything, a current monitor, position sensor.. in this example, we read in the temperature of our STM32 processor.

Next, we initialise our DroneCAN battery message packet and we assign one of its attributes a value from when we read in our sensor. We've used a battery message, since Ardupilot supports 8 of these by default, Mission planner can display information from any of these 8 and Ardupilot logs all battery instances. There may be messages suitable for your application, but be aware, Ardupilot does not support many DroneCAN messages.

```
uavcan_equipment_power_BatteryInfo pkt {};
pkt.temperature = cpu_temp;
```

Finally, we perform the boilerplate required to pack the message and hand it to Canard. As you can see, this process is specific to a Battery Info packet and would need changing to the appropriate equivalent for other message types.

```
uint8_t buffer[UAVCAN_EQUIPMENT_POWER_BATTERYINFO_MAX_SIZE];
uint32_t len = uavcan_equipment_power_BatteryInfo_encode(&pkt, buffer);
static uint8_t transfer_id;
canardBroadcast(&dronecan.canard,
        UAVCAN_EQUIPMENT_POWER_BATTERYINFO_SIGNATURE,
        UAVCAN_EQUIPMENT_POWER_BATTERYINFO_ID,
        &transfer_id,
        CANARD_TRANSFER_PRIORITY_LOW,
        buffer,
        len);
```

At the end of the loop() we call our dronecan.cycle() method, which is required, and we also reset our watchdog timer.

Previous
Micro Node

Next
AP Periph

Last updated 1 month ago