

```

// sensors.hpp: Header file for hardware abstraction of sensors
// Copyright (C) 2017 Ethan Wells
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program. If not, see <http://www.gnu.org/licenses/>.

#ifdef SENSORS_HPP
#define SENSORS_HPP

#include "API.h"

/** The namespace containing all information, functions, objects, relating to
 * sensors */
namespace sensors {
    /** A 2-wire quadrature encoder */
    struct quad_t {
        /** The encoder struct used by other member functions */
        Encoder enc;
        /** The ports the encoder is connected to, in order of top, then bottom,
         * when
         * the removable cover is facing up */
        unsigned char ports[2];
        /** The relative zero from which the encoder's value will be returned. Can
         * be added to returned value to produce a true value for the encoder */
        long zero;
        /** Whether or not the encoder is inverted */
        bool inverted;
        /** Reset the value to zero */
        void reset(void);
        /** Returns the relative value of the encoder. If added to the encoder's
         * zero, produces an absolute value of the encoder */
        long value(void);
        /** The pid requested value of the encoder */
        float request;
        /** The initialization function for the encoder. Call in initialize() */
        void init(void);
    };
}

```

```

    /** Constructs the encoder object. Make sure init is also called */
    quad_t(unsigned char port1, unsigned char port2, bool inverted);
}; // struct quad_t

/** Class for gyro objects */
class gyro_t {
public:
    /** The gyro struct used in funtions */
    Gyro gyro;
    /** The port the gyro is plugged into */
    unsigned char port;
    /** The relative zero of the gyro, such that you can add it to the returned
     * value to obtain an absolute value */
    long zero;
    /** Resets the value to 0 */
    void reset(void);
    /** Returns the current value of the gyro, relative to the zero */
    long value(void);
    /** The pid requested value of the gyro */
    float request;
    /** Initialization funtion for the gyro, call in initialize() */
    void init(void);
    /** Class constructor, but it must not be forgotten to call init() */
    gyro_t(unsigned char port, unsigned int calibration);

private:
    /** The calibration, a temporary placement between construction and
     * initialization */
    int calibration;
}; // struct gyro_t

/** Class for potentiometers */
struct pot_t {
    /** The port that the pot is plugged in to */
    unsigned char port;
    /** The relative zero, that can be added to the returned value() to find the
     * absolute value */
    long zero;
    /** Whether or not the potentiometer's value should be inverted */
    bool inverted;
    /** Resets the value to 0 */
    void reset(void);
    /** Returns the relative value of the potentiometer */
    long value(void);
    /** The pid requested value of the pot */
    float request;
};

```

```

    /** The initialization funtion for the potentiometer, which must be called
    * in
    * initialize() */
    void init(void);
    /** The class constructor for a potentiometer, also be sure to init() */
    pot_t(unsigned char port, bool inverted);
}; // pot_t

/** Class for ultrasonic sensors */
struct sonic_t {
    /** The Ultrasonic struct that is referenced in member funtions */
    Ultrasonic sonic;
    /** The two ports the sensor is plugged in to, in order of the echo (aka
    * orange) cable, then the ping (aka yellow) cable */
    unsigned char ports[2];
    /** The value of the ultrasonic sensor */
    long value(void);
    /** Initializes the sensor. Call in initialize() */
    void init(void);
    /** Class constructor, but init() must also be called */
    sonic_t(unsigned char port1, unsigned char port2);
}; // sonic_t

/** Class for buttons */
struct button_t {
    /** the port that the button is plugged in to */
    unsigned char port;
    /** Whether or not the button's value should be inverted */
    bool inverted;
    /** Returns true if the button is pressed */
    bool value(void);
    /** Initializes the button. Call in initialize() */
    void init(void);
    /** Class constructor, but init() must also be called */
    button_t(unsigned char port, bool inverted);
}; // button_t

/** Initializes the sensor subsystem, calls all the funtions that need to be
* called in initialize(). Call in initialize() */
void init(void);

/** Resets the important sensors */
void reset(void);

/** left quad encoder on the drive */
extern quad_t left;

```

```
    /** right quad encoder on the drive */  
    extern quad_t right;  
    /** potentiometer on the lift */  
    extern pot_t lift;  
    /** gyro on the drive */  
    extern gyro_t gyro;  
} // namespace sensors  
  
#endif /* end of include guard: SENSORS_HPP */
```