```
//
// @file    bluebot.c
//
// Simple robot program using beaglebone blue
//

#include    <stdio.h>
#include    <signal.h>
#include    <getopt.h>
#include    <rc/motor.h>
#include    <rc/time.h>

#include    "arclib.h"

//
// Create our robot structure as a global
// That way all our arc routines will have easy
// access to it
//

arc_robot_t     robot ;

// &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
// MAIN PROGRAM
// &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

int  main() {
    arc_location_t  loc ;

// Configure our robot

    arc_config() ;

// Set starting location and orientation

    loc.x = 0 ; loc.y = 0 ; loc.theta = 90.0 ;

    if (arc_init(loc) == ARC_FAIL) {
                fprintf(stderr,"ERROR: failed to initialize robot. Aborting\n");
                return -1 ;
        }

// Set target location and orientation
```

```
    loc.x = 0 ; loc.y = 12.0 ; loc.theta = 90.0 ;    // Drive straight 12 inches

// Goto target location

    arc_goto(loc) ;

// Cleanup before exiting

    arc_cleanup() ;
}
```

```
//
// arclib is a library of robot routimes
// makes heavy use of strawson libcontrol
//

#include    <stdio.h>
#include    <math.h>
#include    <robotcontrol.h>
#include    <rc/math.h>

#include   "arcdefs.h"

extern  arc_robot_t  robot ;

//
// Routine to configure robot
//

int arc_config(void) {

        double  perimeter ;

// Robot configuration settings

    robot.config.sample_rate = 10.0 ;                                       // Hz
    robot.config.sample_period = 1.0 / robot.config.sample_rate ;       // sec
    robot.config.wheel_diameter = 2.70 ;                                    // inches
    robot.config.wheel_spacing = 2.85 ;                                     // inches
    robot.config.encoder_tics_per_revolution = 60.0 ;
    robot.config.motor_PID_gain.Kp = 10.0 ;                                     // Proportional gain constant
    robot.config.motor_PID_gain.Ki = 1.0 ;                                      // Integral gain constant
    robot.config.motor_PID_gain.Kd = 0.0 ;                                  // Differential gain constant

// Right motor configuration

        robot.right_motor.id = 2 ;                                                              // Blue motor numb
er
        robot.right_motor.swap_wires = ARC_NO_SWAP  ;                           // No need to swap motor wires

// Left motor configuration

        robot.left_motor.id = 3 ;                                                               // Blue motor numb
er
        robot.left_motor.swap_wires = ARC_SWAP ;                                    // Swap blk and red wires
```

```
// Config common to both motors

        perimeter = M_PI * robot.config.wheel_diameter ;
        robot.config.tics_per_inch = robot.config.encoder_tics_per_revolution / perimeter ;
        robot.config.inches_per_tic = 1.0 / robot.config.tics_per_inch ;

        return ARC_PASS ;
}

//
// Routine to initialize everything
// and set our current location and orientation
//

int arc_init(arc_location_t loc) {

        double  Kp, Ki, Kd, dt ;

// Set up the filters we need for PID on right and left motors

//      robot.right_motor.PID_filter = RC_FILTER_INITIALIZER ;
//      robot.left_motor.PID_filter = RC_FILTER_INITIALIZER ;

        Kp = robot.config.motor_PID_gain.Kp ;
        Ki = robot.config.motor_PID_gain.Ki ;
        Kd = robot.config.motor_PID_gain.Kd ;
        dt = robot.config.sample_period ;

        if(rc_filter_pid(&(robot.right_motor.PID_filter), Kp, Ki, Kd, (4.0 * dt), dt)){
                fprintf(stderr,"ERROR in arc_init(), failed to make PID right motor controller\n");
                return ARC_FAIL ;
        }

        if(rc_filter_pid(&(robot.left_motor.PID_filter), Kp, Ki, Kd, (4.0 * dt), dt)){
                fprintf(stderr,"ERROR in arc_init(), failed to make PID left motor controller\n");
                return ARC_FAIL ;
        }

//      Set our current location

        robot.current_location = loc ;          // x, y, theta
        robot.target_location = loc ;
```

```
// make sure another process instance isn't running
// if return value is -3 then a background process is running with
// higher privaledges and we couldn't kill it, in which case we should
// not continue or there may be hardware conflicts. If it returned -4
// then there was an invalid argument that needs to be fixed.

        if(rc_kill_existing_process(2.0)<-2) {
                return ARC_FAIL;
        }

// Initialize motors

        if (rc_motor_init_freq(RC_MOTOR_DEFAULT_PWM_FREQ)) {
                return ARC_FAIL;        // Set PWM frequency
        }

        if (rc_motor_init()==-1){
                fprintf(stderr,"ERROR: failed to initialize motors\n");
                return ARC_FAIL;
        }

        rc_motor_standby(1); // start with motors in standby

// Initialize enocders

        if (rc_encoder_eqep_init()==-1){
                fprintf(stderr,"ERROR: failed to initialize eqep encoders\n");
                return ARC_FAIL;
        }

// Start MPU

        if(rc_mpu_initialize_dmp(&(robot.mpu_data), robot.config.mpu_config)){
                fprintf(stderr,"ERROR: can't talk to IMU, all hope is lost\n");
                rc_led_blink(RC_LED_RED, 5, 5);
                return ARC_FAIL;
        }

// If gyro isn't calibrated, run the calibration routine

        if(!rc_mpu_is_gyro_calibrated()){
                printf("Gyro not calibrated, automatically starting calibration routine\n");
                printf("Let your MiP sit still on a firm surface\n");
                rc_mpu_calibrate_gyro_routine(robot.config.mpu_config);
```

```
        }

// Turn the red and green LEDs on

        rc_led_set(RC_LED_GREEN, ARC_ON);
        rc_led_set(RC_LED_RED, ARC_ON);

// Put the motors into freespin mode

        rc_motor_free_spin(robot.right_motor.id);
        rc_motor_free_spin(robot.left_motor.id);

// Return

        return ARC_PASS ;
}

//
// Routine to "goto" given location
//

int arc_goto(arc_location_t loc) {
        int     target_reached ;
        int     right_tics, left_tics ;
        int     error ;

//      Set our target location

        robot.target_location = loc ;

// Set flag for target reached location

        target_reached = ARC_FALSE ;
        robot.right_motor.pwm_value = 0.2 ;
        robot.left_motor.pwm_value = -0.2 ;

// Keep going until target reached

        while (target_reached == ARC_FALSE) {

// Read encoders

                right_tics = rc_encoder_read(robot.right_motor.id) ;
                left_tics = rc_encoder_read(robot.left_motor.id) ;
```

```
// Update motors

                rc_motor_set(robot.right_motor.id, robot.right_motor.pwm_value);
                rc_motor_set(robot.left_motor.id, robot.left_motor.pwm_value);

// Compute error and filter

                error = robot.right_motor.vel_in_tics - right_tics ;
                robot.right_motor.pwm_value = rc_filter_march(&robot.right_motor.PID_filter, error) ;
                error = robot.left_motor.vel_in_tics - left_tics ;
                robot.left_motor.pwm_value = rc_filter_march(&robot.left_motor.PID_filter, error) ;

// Done ... just wait

        }
        return ARC_PASS ;
}

//
// Routine to cleanup after ourselves
//

int arc_cleanup(void) {

// Clean up motors

        rc_motor_cleanup();

// Free up the memory used by the PID filters

        rc_filter_free(&(robot.right_motor.PID_filter));
        rc_filter_free(&(robot.left_motor.PID_filter));

// Turn the LEDs off and shutoff LED handlers

        rc_led_set(RC_LED_GREEN, ARC_OFF);
        rc_led_set(RC_LED_RED, ARC_OFF);
        rc_led_cleanup();

// Cleanup the encoder stuff

        rc_encoder_eqep_cleanup() ;
```

```
// Turn mpu off

        rc_mpu_power_off();

        return ARC_PASS ;
}
```

```
/**
 * @file rc_test_motors.c
 * @example    rc_test_motors
 *
 * Demonstrates use of H-bridges to drive motors with the Robotics Cape and
 * BeagleBone Blue. Instructions are printed to the screen when called.
 */


#include <stdio.h>
#include <signal.h>
#include <stdlib.h> // for atoi
#include <getopt.h>
#include <rc/motor.h>
#include <rc/time.h>

static int running = 0;

// possible modes, user selected with command line arguments
typedef enum m_mode_t{
        DISABLED,
        NORMAL,
        BRAKE,
        FREE,
        SWEEP
} m_mode_t;

// printed if some invalid argument was given
static void __print_usage(void)
{
        printf("\n");
        printf("-d {duty}   define a duty cycle from -1.0 to 1.0\n");
        printf("-b          enable motor brake function\n");
        printf("-F {freq}   set a custom pwm frequency in HZ, otherwise default 25000 is used\n");
        printf("-f          enable free spin function\n");
        printf("-s {duty}   sweep motors back and forward at duty cycle\n");
        printf("-m {motor}  specify a single motor from 1-4, otherwise all will be driven\n");
        printf("            motors will be driven equally.\n");
        printf("-h          print this help message\n");
        printf("\n");
}

// interrupt handler to catch ctrl-c
static void __signal_handler(__attribute__ ((unused)) int dummy)
```

```
{
        running=0;
        return;
}

int main(int argc, char *argv[])
{
        double duty = 0.0;
        int ch = 0; // assume all motor unless set otherwise
        int c, in;
        int freq_hz = RC_MOTOR_DEFAULT_PWM_FREQ;
        m_mode_t m_mode = DISABLED;

        // parse arguments
        opterr = 0;
        while ((c = getopt(argc, argv, "m:d:F:fbs:h")) != -1){
                switch (c){
                case 'm': // motor channel option
                        in = atoi(optarg);
                        if(in<=4 && in>=0){
                                ch = in;
                        }
                        else{
                                fprintf(stderr,"-m motor option must be from 0-4\n");
                                return -1;
                        }
                        break;
                case 'd': // duty cycle option
                        if(m_mode!=DISABLED) __print_usage();
                        duty = atof(optarg);
                        if(duty<=1 && duty >=-1){
                                m_mode = NORMAL;
                        }
                        else{
                                fprintf(stderr,"duty cycle must be from -1 to 1\n");
                                return -1;
                        }
                        break;
                case 'F': // pwm frequency option
                        freq_hz = atoi(optarg);
                        if(freq_hz<1){
                                fprintf(stderr,"PWM frequency must be >=1\n");
                                return -1;
                        }
```

```
                        break;
                case 'f':
                        if(m_mode!=DISABLED) __print_usage();
                        m_mode = FREE;
                        break;
                case 'b':
                        if(m_mode!=DISABLED) __print_usage();
                        m_mode = BRAKE;
                        break;
                case 's':
                        if(m_mode!=DISABLED) __print_usage();
                        duty = atof(optarg);
                        if(duty<=1 && duty >=-1){
                                m_mode = SWEEP;
                        }
                        else{
                                fprintf(stderr,"duty cycle must be from -1 to 1\n");
                                return -1;
                        }
                        break;
                case 'h':
                        __print_usage();
                        return -1;
                        break;
                default:
                        __print_usage();
                        return -1;
                        break;
                }
        }

        // if the user didn't give enough arguments, print usage
        if(m_mode==DISABLED){
                __print_usage();
                return -1;
        }

        // set signal handler so the loop can exit cleanly
        signal(SIGINT, __signal_handler);
        running =1;

        // initialize hardware first
        if(rc_motor_init_freq(freq_hz)) return -1;
```

```
	// decide what to do
	switch(m_mode){
	case NORMAL:
		printf("sending duty cycle %0.4f\n", duty);
		rc_motor_set(ch,duty);
		break;
	case FREE:
		printf("Free Spin Mode\n");
		rc_motor_free_spin(ch);
		break;
	case BRAKE:
		printf("Braking Mode\n");
		rc_motor_brake(ch);
		break;
	default:
		break;
	}

	// wait untill the user exits
	while(running){
		if(m_mode==SWEEP){
			duty = -duty; // toggle back and forth to sweep motors side to side
			printf("sending duty cycle %0.4f\n", duty);
			fflush(stdout);
			rc_motor_set(ch,duty);
		}

		// if not in SWEEP mode, the motors have already been set so do nothing
		rc_usleep(500000);
	}


	// final cleanup
	printf("\ncalling rc_motor_cleanup()\n");
	rc_motor_cleanup();
	return 0;
}
```

```
//
// Define our types here
//

#include <robotcontrol.h>

// Some defines that we would like to use

#define         ARC_ON                  1
#define         ARC_OFF                 0
#define         ARC_FAIL                1
#define         ARC_PASS                0
#define         ARC_SWAP                -1
#define         ARC_NO_SWAP             1
#define         ARC_PI                  3.14159
#define         M_TO_INCH               39.37
#define         INCH_TO_M               (1.0/39.37)
#define         ARC_TRUE                1
#define         ARC_FALSE               0

// ****************************************
// Structure to hold PID gains
// ****************************************

typedef struct  arc_PIDgain_t {
    double   Kp ;
    double   Ki ;
    double   Kd ;
} arc_PIDgain_t ;


// ****************************************
// Structure to hold motor data
// ****************************************

typedef struct  arc_motor_t {
        rc_filter_t             PID_filter ;                            // Filter to be used for PID control
    double          id ;                        // Motor number {1, 2, 3, 4}
    int             swap_wires ;                // -1 = swap blk and red wires, 1 don't swap
    double          pwm_value   ;               // last pwm_value assigned to motor [-1, +1]
    int             vel_in_tics ;                           // number of tics we expect to count in sample period
} arc_motor_t ;

// ****************************************
```

```
// Structure to hold location data
// ****************************************

typedef struct  arc_location_t {
    double       x ;                                        // x-coordinate (in inches)
    double       y ;                                        // y-coordinate (in inches)
    double       theta ;                                    // orientation (90 degrees for looking north)
} arc_location_t ;

// ****************************************
// Structure to hold robot config data
// ****************************************

typedef struct arc_config_t {
    double          sample_rate ;                           // Sampling frequency
    double          sample_period ;                         // Period used for updates
    double          wheel_diameter ;                    // Wheel diameter in inches
    double          wheel_spacing ;                         // Spacing between wheels in inches
    double          encoder_tics_per_revolution ;
        double                  inches_per_tic ;
        double                  tics_per_inch ;
    arc_PIDgain_t   motor_PID_gain ;                    // PID gain constants for motor
        rc_mpu_config_t mpu_config ;                        // MPU config
} arc_config_t ;

// ****************************************
// Structure which defines our robot
// ****************************************

typedef struct  arc_robot_t {
    arc_config_t    config ;                                // Struct that contains our robot configuration
    arc_motor_t     right_motor ;                           // Struct for the right motor
    arc_motor_t     left_motor ;                            // Struct for the left motor
    arc_location_t  current_location ;                  // Robot's current location
    arc_location_t  target_location ;                   // Robot's target location
    double          desired_velocity ;          // Velocity of robot we desire in in / sec
    double          actual_velocity ;           // Actual current velocity of robot in in / sec
    double          state ;                     // Robot state
        rc_mpu_data_t   mpu_data;                           // MPU data
} arc_robot_t ;
```

```
#include "arcdefs.h"

//
// Here are the routines in the arclib
//

int arc_config(void) ;
int arc_init(arc_location_t loc) ;
int arc_goto(arc_location_t loc) ;
int arc_cleanup(void) ;
```