src\main.cpp

```
//   1/5/24
#include "main.h"
#include "lemlib/api.hpp"
#include "lemlib/chassis/chassis.hpp"
#include "pros/adi.h"
#include "pros/adi.hpp"
#include "pros/imu.hpp"
#include "pros/llemu.hpp"
#include "pros/misc.h"
#include "pros/motors.h"
#include "pros/rotation.hpp"
#include "pros/rtos.hpp"


//Motors and defining functions
pros::Controller master(pros::E_CONTROLLER_MASTER);
pros::Motor FL(12,false);
pros::Motor MidL(11,false);
pros::Motor BL(2,false);
pros::Motor FR(18,true);
pros::Motor MidR(20,true);
pros::Motor BR(9,true);
pros::Motor intake(15,false);
pros::Motor flywheel(14,false);
pros::Imu Gyro(1); // port for inertial
pros::ADIDigitalOut lift('h', LOW);
pros::ADIDigitalOut wings('g', LOW);
pros::Rotation rot(21);

//Drivetrain motor groups
pros::Motor_Group leftDb({FL,MidL, BL});
  pros::Motor_Group rightDb({FR,MidR,BR});




lemlib::Drivetrain_t drivetrain {
    &leftDb, // left drivetrain motors
    &rightDb, // right drivetrain motors
    10, // track width
    3.25, // wheel diameter
    400 // wheel rpm
    };



// Odom construct
    lemlib::OdomSensors_t sensors {
    nullptr, // vertical tracking wheel 1
    nullptr, // vertical tracking wheel 2
    nullptr, // horizontal tracking wheel 1
    nullptr, // we don't have a second tracking wheel, so we set it to nullptr
    &Gyro // inertial sensor
```

```
54      };
55
56
57  // forward/backward PID
58      lemlib::ChassisController_t lateralController {
59        //kp 9.71 kd 17 - good values
60      9.71, // kPs
61      17, // kD
62      1, // smallErrorRange
63      100, // smallErrorTimeout
64      1, // largeErrorRange
65      500, // largeErrorTimeout
66      25 // slew rate
67      };
68
69  // turning PID wow
70      lemlib::ChassisController_t angularController {
71        // Good Values kp 1.1 kd 20
72      1.1, // kP
73      20, // kD
74      1, // smallErrorRange
75      200, // smallErrorTimeout
76      1, // largeErrorRange
77      700, // largeErrorTimeout
78      25 // slew rate
79      };
80
81
82  // create the chassis
83  lemlib::Chassis chassis(drivetrain, lateralController, angularController, sensors);
84
85
86  /**
87   * A callback function for  LLEMU's center button.
88   *
89   * When this callback is fired, it will toggle line 2 of the LCD text between
90   * "I was pressed!" and nothing.
91   */
92  void on_center_button() {
93    static bool pressed = false;
94    pressed = !pressed;
95    if (pressed) {
96      pros::lcd::set_text(2, "I was pressed!");
97    } else {
98      pros::lcd::clear_line(2);
99    }
100 }
101
102
103 void screen() {
104     while (true) {
105         auto pose = chassis.getPose();
106         pros::lcd::print(0, "X: %f", pose.x);
107         pros::lcd::print(1, "Y: %f", pose.y);
108         pros::lcd::print(2, "Heading: %f", pose.theta);
109         pros::delay(20);
```

```
110     }
111 }
112 /**
113  * Runs initialization code. This occurs as soon as the program is started.
114  *
115  * All other competition modes are blocked by initialize; it is recommended
116  * to keep execution time for this mode under a few seconds.
117  */
118 void initialize() {
119   pros::lcd::initialize();
120     pros::Task screenDisplay(screen);
121     chassis.calibrate();
122   pros::lcd::set_text(1, "Hello PROS User!");
123
124
125   pros::lcd::register_btn1_cb(on_center_button);
126 }
127
128
129 /**
130  * Runs while the robot is in the disabled state of Field Management System or
131  * the VEX Competition Switch, following either autonomous or opcontrol. When
132  * the robot is enabled, this task will exit.
133  */
134 void disabled() {}
135
136
137 /**
138  * Runs after initialize(), and before autonomous when connected to the Field
139  * Management System or the VEX Competition Switch. This is intended for
140  * competition-specific initialization routines, such as an autonomous selector
141  * on the LCD.
142  *
143  * This task will exit when the robot is enabled and autonomous or opcontrol
144  * starts.
145  */
146 void competition_initialize() {}
147
148
149 /**
150  * Runs the user autonomous code. This function will be started in its own task
151  * with the default priority and stack size whenever the robot is enabled via
152  * the Field Management System or the VEX Competition Switch in the autonomous
153  * mode. Alternatively, this function may be called in initialize or opcontrol
154  * for non-competition testing purposes.
155  *
156  * If the robot is disabled or communications is lost, the autonomous task
157  * will be stopped. Re-enabling the robot will restart the task, not re-start it
158  * from where it left off.
159  */
160  /***/
161  /** auto for league games
162  void autonomous() {
163     chassis.moveTo(0,-150,600);
164     pros::delay(600);
165     chassis.moveTo(0,60,600);
```

```
}

 */


void turn(double theta){
    double x = 10000 * (cos(theta * (M_PI / 180.0) + chassis.getPose().x));
    double y = 10000 * (sin(theta * (M_PI / 180.0) + chassis.getPose().y));
    chassis.turnTo(x,y,1000);


}

void autonomous() {

    //pros::delay(600);
    //chassis.moveTo(0,60,600);
    // Match Far Auton ---------------------------------------------------------------------------


    // AWP Close ---------------------------------------------------------------------------
    chassis.moveTo(0,-30,600);
    pros::delay(600);
    chassis.moveTo(0,10,600);
    pros::delay(600);
    chassis.turnTo(-45,17.7, 600);
    pros::delay(600);
    wings.set_value(HIGH);
    pros::delay(600);
    chassis.moveTo(70,30,600);
    pros::delay(600);
    chassis.turnTo(-60,25,600);
    pros::delay(600);
    wings.set_value(LOW);
    pros::delay(600);

    /*
    wings.set_value(LOW);
    pros::delay(600);
    chassis.moveTo(-34.25,40,600);
    chassis.moveTo(-55.4,16.6,600);
    pros::delay(7000);
    */




    /*
    wings.set_value(HIGH);
    chassis.moveTo(0,10,600);
    pros::delay(600);
    chassis.turnTo(-33,17.7, 600);
    wings.set_value(LOW);
```

```
221     pros::delay(600);
222     chassis.moveTo(-34.25,40,600);
223     chassis.moveTo(-55.4,16.6,600);
224     pros::delay(7000);
225 
226     */
227 
228 
229     //Auton skills
230     //flydeck = 118;
231 
232 }
233 
234 /**
235  * Runs the operator control code. This function will be started in its own task
236  * with the default priority and stack size whenever the robot is enabled via
237  * the Field Management System or the VEX Competition Switch in the operator
238  * control mode.
239  *
240  * If no competition control is connected, this function will run immediately
241  * following initialize().
242  *
243  * If the robot is disabled or communications is lost, the
244  * operator control task will be stopped. Re-enabling the robot will restart the
245  * task, not resume it from where it left off.
246  */
247 //cata motor loop and configuration
248 void opcontrol() {
249 
250 
251 
252 
253 
254  while (true) {
255   pros::delay(20);
256   double power = master.get_analog(pros::E_CONTROLLER_ANALOG_LEFT_Y);
257   double turn = master.get_analog(pros::E_CONTROLLER_ANALOG_RIGHT_X);
258   leftDb.move(power - turn);
259   rightDb.move(power + turn);
260 
261 
262 
263 //flywheel
264 
265 
266 
267 //Lift
268     if(master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_UP)){
269         lift.set_value(HIGH);
270         flywheel = 128;
271     }
272 
273     if(master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_Y)){
274         flywheel = 128;
275     }
276 
```

```
277     if(master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_A)){
278         lift.set_value(HIGH);
279         wings.set_value(HIGH);
280         flywheel = 128;
281     }
282 
283     if(master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_B)){
284         lift.set_value(LOW);
285         wings.set_value(LOW);
286         flywheel = -0;
287     }
288 
289 
290     if(master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_DOWN)){
291         lift.set_value(LOW);
292         if(lift.set_value(LOW) == true ){
293             double angle = rot.get_angle()/100.0;
294             pros::lcd::print(0, "Angle: ", angle);
295             if (angle < 46) {
296                 flywheel.move_velocity(100);
297             }
298             else {
299                 flywheel = 0;
300             }
301 
302         }
303 
304 
305 
306 
307 
308         pros::delay(10);
309     }
310 
311 
312 //WINGS
313     if(master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_R1)){
314         wings.set_value(HIGH);
315     }
316     if(master.get_digital_new_press(pros::E_CONTROLLER_DIGITAL_L1)){
317         wings.set_value(LOW);
318 
319         pros::delay(10);
320     }
321 
322 
323   //intake motor code
324     if(master.get_digital(pros::E_CONTROLLER_DIGITAL_L2)){
325         intake = 127;
326     }
327     else if (master.get_digital(pros::E_CONTROLLER_DIGITAL_R2)) {
328        intake = -127;
329     }
330     else {
331      intake = 0;
332     }
```

```
333
334
335   }
336 }
```