```
#include "main.h"
#include "portdef.h"
#include "chassis.h"
#include "roller.h"
#include "lift.h"
#include "tray.h"

// ALL OTHER MODULE INCLUDES HERE


/**
 * Runs the operator control code. This function will be started in its own task
 * with the default priority and stack size whenever the robot is enabled via
 * the Field Management System or the VEX Competition Switch in the operator
 * control mode.
 *
 * If no competition control is connected, this function will run immediately
 * following initialize().
 *
 * If the robot is disabled or communications is lost, the
 * operator control task will be stopped. Re-enabling the robot will restart the
 * task, not resume it from where it left off.
 */
void opcontrol() {
    pros::Controller master(pros::E_CONTROLLER_MASTER);

    int left = 0;                   // left motor speed control
    int right = 0;              // right motor speed control

    double scaling = 1.0;

    extern int selection;

    bool autoRun = true;

    while (true) {

        if (DRIVE_MODE == 1) {
            // We want to do X-Drive TANK control
            int rightX = master.get_analog(ANALOG_RIGHT_X);
          int rightY = master.get_analog(ANALOG_RIGHT_Y);
            int leftX = master.get_analog(ANALOG_LEFT_X);
          int leftY = master.get_analog(ANALOG_LEFT_Y);

            if(abs(rightX) < DEAD_STICK) { rightX = 0; }
            if(abs(rightY) < DEAD_STICK) { rightY = 0; }
            if(abs(leftX) < DEAD_STICK) { leftX = 0; }
            if(abs(leftY) < DEAD_STICK) { leftY = 0; }


            setIndividualMotor((rightY - average(rightX, leftX)),
                                              (leftY + average(rightX, leftX)),
                                              (rightY + average(rightX, leftX)),
                                            (leftY - average(rightX, leftX)));
        }
        else if (DRIVE_MODE == 2) {
            // We want to do X-Drive ARCADE control
            int rightX = master.get_analog(ANALOG_RIGHT_X);
          int rightY = master.get_analog(ANALOG_RIGHT_Y);
            int leftX = master.get_analog(ANALOG_LEFT_X);
          int leftY = master.get_analog(ANALOG_LEFT_Y);

            if(abs(rightX) < DEAD_STICK) { rightX = 0; }
            if(abs(rightY) < DEAD_STICK) { rightY = 0; }
            if(abs(leftX) < DEAD_STICK) { leftX = 0; }
            if(abs(leftY) < DEAD_STICK) { leftY = 0; }

```

```
68            setIndividualMotor((rightY - leftX - rightX),
69                                               (rightY + leftX + rightX),
70                                               (rightY - leftX + rightX),
71                                               (rightY + leftX - rightX));
72          }
73          else if (DRIVE_MODE == 3) {
74              // We are wanting to do standard ARCADE control
75          left = master.get_analog(ANALOG_LEFT_Y);
76            right = master.get_analog(ANALOG_LEFT_X);
77
78              // implemenet dead stick control
79              if(abs(left) < DEAD_STICK) { left = 0; }
80              if(abs(right) < DEAD_STICK) { right = 0; }
81
82               chassisSetOpcontrol(left + right, left - right);
83      }
84          else if (DRIVE_MODE == 4) {
85         // we are wanting to do standard TANK Control
86               left = master.get_analog(ANALOG_LEFT_Y);
87               right = master.get_analog(ANALOG_RIGHT_Y);
88
89               // implemenet dead stick control
90               if(abs(left) < DEAD_STICK) { left = 0; }
91               if(abs(right) < DEAD_STICK) { right = 0; }
92
93        /*
94         if(DEBUG_ON) {
95                   std::cout << "Scaling: " << scaling ;
96                   std::cout << " Left: " << left ;
97               }
98               */
99
100        // lets do JOY stick scaling as well
101              left = (left * scaling);
102              right = (right * scaling);
103       /*
104              if(DEBUG_ON) {
105                 std::cout << " Left Scaled: " << left << "\n" ;
106              }
107              */
108         chassisSetOpcontrol(left, right);
109     }
110         else if (DRIVE_MODE == 5) {    // CURRENTLY DECOMMISSIONED UNTIL FURTHER DEVELOPMENT
111             int leftX;
112           int leftY;
113
114             if (master.get_digital(DIGITAL_UP)) {
115                 leftY = 200;
116             }
117             else if (master.get_digital(DIGITAL_DOWN)) {
118                 leftY = -200;
119             }
120             else {
121                 leftY = 0;
122             }
123
124             if (master.get_digital(DIGITAL_LEFT)) {
125                 leftX = 200;
126             }
127             else if (master.get_digital(DIGITAL_RIGHT)) {
128                 leftX = -200;
129             }
130             else {
131                 leftX = 0;
132             }
133
134           int rightX = master.get_analog(ANALOG_RIGHT_X);
135
136             if(abs(rightX) < DEAD_STICK) { rightX = 0; }
137             if(abs(leftX) < DEAD_STICK) { leftX = 0; }
```

```
138             if(abs(leftY) < DEAD_STICK) { leftY = 0; }
139 
140             setIndividualMotor((rightX - leftX - rightX),
141                                                (rightX + leftX + rightX),
142                                                (rightX - leftX + rightX),
143                                                (rightX + leftX - rightX));
144 
145         }
146 
147         if (master.get_digital(DIGITAL_R1)) {
148             rollerForward(600);
149         }
150         else if (master.get_digital(DIGITAL_R2)) {
151             rollerBackward(600);
152         }
153         else {
154             rollerStop(0);
155         }
156 
157         if (master.get_digital(DIGITAL_L1)) {
158             liftRaiseManual(100);
159         }
160         else if (master.get_digital(DIGITAL_L2)) {
161             liftRaiseManual(-90);
162         }
163         else {
164             liftLock();
165         }
166 
167         if (master.get_digital(DIGITAL_A)) {
168             trayForward(50);
169         }
170         else if (master.get_digital(DIGITAL_B)) {
171             trayBackward(50);
172         }
173         else {
174             trayLock();
175         }
176 
177         pros::delay(20);
178     }
179 }
180 
```