```cpp
#include "main.h"
using namespace pros;
/* Button definition */
#define ANLG_FB ANALOG_LEFT_Y
 // Define front-back channel for arcade mode drive
#define ANLG_LR ANALOG_RIGHT_X
 // Define turning channel for arcade mode

#define BTN_ARM_UP DIGITAL_UP
 // Manually raise arm
#define BTN_ARM_DOWN DIGITAL_DOWN
 // Manually lower arm
#define BTN_MANUAL_CW DIGITAL_RIGHT
 // Manually rotate wrist clockwise
#define BTN_MANUAL_CCW DIGITAL_LEFT
 // Manually rotate wrist counterclockwise
#define BTN_FLOOR_FLIP DIGITAL_B
 // For automatic cap-flipping routine
#define BTN_TARE_ARM DIGITAL_Y
 // Sets arm position to 0

#define BTN_ARM_OTHERSIDE DIGITAL_R1
 // Toggles arm position: pole height or ground
#define BTN_ARM_AUTOBUMPUP DIGITAL_L1
 // Raises arm predetermined number of ticks
#define BTN_ARM_AUTOBUMPDOWN DIGITAL_L2
 // Lowers arm predetermined number of ticks
#define BTN_ARM_AUTODOWN DIGITAL_R2
 // Lowers arm to the ground

Controller controller (E_CONTROLLER_MASTER);                    /*
 Instantiate controller */

/* Global variables and constants */
double armSeek = -1;
 // The position arm is trying to reach. -1 when not seeking
double armPos = 0;
 // Current arm position
double armPower = 0;
 // Power sent to the arm, from 0 to 12000
double MAX_SPEED = 12000;
 // Max speed of arm (usual speed)

int colorMultiplier = 1;
 // 1 or -1. Causes mirroring of auton routines for red/blue sides

int flipStep = -1;
 // Tracks step# in a multi-step routine
int closeEnough = 10;
 // Threshold error for motor position seeking
int DEGS_FOR_COMPLETE_DOWN = 20;
 // Used instead of 0 when seeking ground. Compensates for mech issues
int BUMP_VALUE = 110;
 // Amount (ticks) by which arm goes up/down
int UPPER_ARM_THRESHOLD = 1300;
 // Highest position the arm can go

double POLE_HEIGHT_DEGS = 750;
 // Arm height to reach low pole
double ABOVE_POLE_HEIGHT_DEGS = 1200;
 // Arm height to seek when flipping cap above pole

/* Motors Instantiation */
Motor left1(8, E_MOTOR_GEARSET_18, false, E_MOTOR_ENCODER_DEGREES);
 // Set drive motors
Motor left2(2, E_MOTOR_GEARSET_18, false, E_MOTOR_ENCODER_DEGREES);
Motor left3(10, E_MOTOR_GEARSET_18, false, E_MOTOR_ENCODER_DEGREES);
Motor left4(9, E_MOTOR_GEARSET_18, false, E_MOTOR_ENCODER_DEGREES);
Motor right1(6, E_MOTOR_GEARSET_18, true, E_MOTOR_ENCODER_DEGREES);
Motor right2(4, E_MOTOR_GEARSET_18, true, E_MOTOR_ENCODER_DEGREES);
Motor right3(15, E_MOTOR_GEARSET_18, true, E_MOTOR_ENCODER_DEGREES);
Motor right4(5, E_MOTOR_GEARSET_18, true, E_MOTOR_ENCODER_DEGREES);

Motor wrist(1, E_MOTOR_GEARSET_18, true, E_MOTOR_ENCODER_DEGREES);
 // Wrist motor
Motor flipper(8, E_MOTOR_GEARSET_18, true, E_MOTOR_ENCODER_DEGREES);
 // Arm motor

/* Zero motors at beginning of the match */
void myInit(void* param){
    wrist.tare_position();
    flipper.tare_position();
}

/* Drive in arcade mode activated; once this is called, controller
 always listening for input*/
void enable_drive(void* param){
    while(true) {
        left1.move_voltage((controller.get_analog(ANLG_FB) +
         controller.get_analog(ANLG_LR))*12000/127);
        left2.move_voltage((controller.get_analog(ANLG_FB) +
         controller.get_analog(ANLG_LR))*12000/127);
        left3.move_voltage((controller.get_analog(ANLG_FB) +
         controller.get_analog(ANLG_LR))*12000/127);
        left4.move_voltage((controller.get_analog(ANLG_FB) +
         controller.get_analog(ANLG_LR))*12000/127);
        right1.move_voltage((controller.get_analog(ANLG_FB) -
         controller.get_analog(ANLG_LR))*12000/127);
        right2.move_voltage((controller.get_analog(ANLG_FB) -
         controller.get_analog(ANLG_LR))*12000/127);
```

```
        right3.move_voltage((controller.get_analog(ANLG_FB) -
         controller.get_analog(ANLG_LR))*12000/127);
        right4.move_voltage((controller.get_analog(ANLG_FB) -
         controller.get_analog(ANLG_LR))*12000/127);
        pros::delay(20);
    }
}

/* Task for arm controls */
void manual_arm(void* param){
    armSeek = -1;

    while(true){
        armPower = 0;                           // Calculated later
        if(controller.get_digital(BTN_ARM_UP)) // If arm-up button
         pressed
        {
            armPower = MAX_SPEED;               //
            if(armPos > UPPER_ARM_THRESHOLD){ //dont keep going. Gonna
             skip gears
                pros::lcd::print(0, "armPos: %f\n", armPos);
                pros::lcd::print(0, "armPos: %f\n", armPos);
                armPower = 0;
            }
            armSeek = -1;
        }
        else if(controller.get_digital(BTN_ARM_DOWN)) //manual arm
         down
        {
            armSeek = -1;
            armPower = -1*MAX_SPEED;
            if(armPos < -15){ //dont keep going. Gonna skip gears
                armPower = 0;
            }
        }
        else if(controller.get_digital(BTN_ARM_AUTOBUMPUP)){
            if(armPos + BUMP_VALUE <= UPPER_ARM_THRESHOLD){
                armSeek = armPos + BUMP_VALUE;
            }
        }
        else if(controller.get_digital(BTN_ARM_AUTOBUMPDOWN)){
            if(armPos - BUMP_VALUE >= -20){
                armSeek = armPos - BUMP_VALUE;
            }
        }
        else if(controller.get_digital(BTN_TARE_ARM)){
            //motorApexA.tare_position();
            //motorApexB.tare_position();
        }

        /*
         else if(controller.get_digital(BTN_ARM_AUTODOWN)){
         if(isFlipped == true)
         //wristSeek = 0;
         armSeek = DEGS_FOR_COMPLETE_DOWN;
         }*/
        else{ //manual mode...
            armPower = 0;
        }

        if(controller.get_digital(BTN_ARM_OTHERSIDE)){
            if(justFlipped == false){
                justFlipped = true;
                pros::lcd::print(5, "(t)justFlipped: ", justFlipped);
                if(armSeek == POLE_HEIGHT_DEGS)
                    armSeek = 100;
                else
                    armSeek = OTHER_SIDE;

            }
        }
        else{
            justFlipped = false;
            pros::lcd::print(5, "(f)justFlipped: ", justFlipped);

        }

        /*Manual controls*/
        if(controller.get_digital(BTN_MANUAL_CW))
        {
            wristSeek = -1;                 //Doesn't see a position.
            motorWrist.tare_position();        //Primary function of
             this is re-zeroing of displaced encoders
            wristPower = MANUAL_WRIST_SPEED;    //Turn the wrist
        }

        else if(controller.get_digital(BTN_MANUAL_CCW)) //A
        {
            wristSeek = -1;
            motorWrist.tare_position();
            wristPower = -1*MANUAL_WRIST_SPEED;
        }
        else if(controller.get_digital(BTN_FLOOR_FLIP)) //
        {
            flipStep = 1;
            armSeek = armPos + 400;
        }


    }
    delay(20);
}
```

```
/*Auton Vars*/
int MIN_DRIVE_VOLTAGE = 5000;
int armAutonPos = -1;
int armAutonPower = 0;

/*Auton Code*/

/*Zeros all motors on the robot*/
void tareDriveMotors(){
    left1.tare_position();
    left2.tare_position();
    left3.tare_position();
    left4.tare_position();
    right1.tare_position();
    right2.tare_position();
    right3.tare_position();
    right4.tare_position();
}
void driveMotorsAt(int driveVoltage){
    left1.move_voltage(driveVoltage);
    left2.move_voltage(driveVoltage);
    left3.move_voltage(driveVoltage);
    left4.move_voltage(driveVoltage);
    right1.move_voltage(driveVoltage);
    right2.move_voltage(driveVoltage);
    right3.move_voltage(driveVoltage);
    right4.move_voltage(driveVoltage);
}

/*positive = counterclockwise*/
void turnMotorsAt(int wheelSeek){
    tareDriveMotors();
    int wheelPos = right1.get_position();//0 to neg

    if(wheelSeek > 0){
        while(wheelPos < wheelSeek){

            int driveVoltage = (wheelSeek - wheelPos)*60;
            if(driveVoltage < MIN_DRIVE_VOLTAGE)
                driveVoltage = MIN_DRIVE_VOLTAGE;
            left1.move_voltage(-1*driveVoltage);
            left2.move_voltage(-1*driveVoltage);
            left3.move_voltage(-1*driveVoltage);
            left4.move_voltage(-1*driveVoltage);
            right1.move_voltage(driveVoltage);
            right2.move_voltage(driveVoltage);
            right3.move_voltage(driveVoltage);
            right4.move_voltage(driveVoltage);
            wheelPos = right1.get_position();
        }
        driveMotorsAt(0);
    }
    else{
        while(wheelPos > wheelSeek){

            int driveVoltage = (wheelSeek - wheelPos)*60;
            if(driveVoltage > -1*MIN_DRIVE_VOLTAGE)
                driveVoltage = -1*MIN_DRIVE_VOLTAGE;

            left1.move_voltage(-1*driveVoltage);
            left2.move_voltage(-1*driveVoltage);
            left3.move_voltage(-1*driveVoltage);
            left4.move_voltage(-1*driveVoltage);
            right1.move_voltage(driveVoltage);
            right2.move_voltage(driveVoltage);
            right3.move_voltage(driveVoltage);
            right4.move_voltage(driveVoltage);
            wheelPos = right1.get_position();
        }
        driveMotorsAt(0);
    }
    delay(1);
}

void flipCap(){
    motorWrist.tare_position();                    //In this event
     that encoders get messed up, tare.
    int position = motorWrist.get_position();            //Track
     position of wrist
    int t0 = pros::millis();                    //The time in
     milliseconds now
    //^If something gets stuck, gives up on cap flip instead of trying
     forever
    while(position < 180){                        //While we still
     haven't reached 180deg
        if(pros::millis() == t0 + 2000)               //If we've been
         trying for over 2 sec...
            break;                          //...give up.
        position = motorWrist.get_position();           //Get wrist's
         position
        int voltage = (90 - position)*150;            //Voltage /
         speed of wrist gets slower as closer to 180
        if(voltage < 3000)                    //Set minimum voltage
            voltage = 3000;
        motorWrist.move_voltage(voltage);            //Turn wrist at
         calculated voltage
    }
}
void driveForward(int wheelSeek){
    int t0 = pros::millis() + abs(wheelSeek)*3;
    tareDriveMotors();
```

```
    int wheelPos = left2.get_position();
    if(wheelSeek > 0){
        while(wheelPos < wheelSeek){
            if (pros::millis() > t0)
                break;
            pros::lcd::print(4,"Positive Seek: %f\n", wheelPos);

            wheelPos = left2.get_position();
            int driveVoltage = (wheelSeek - wheelPos)*60;

            if(driveVoltage < MIN_DRIVE_VOLTAGE)
                driveVoltage = MIN_DRIVE_VOLTAGE;

            driveMotorsAt(driveVoltage);
        }
        driveMotorsAt(0);
    }
    else{
        while(wheelPos > wheelSeek){
            if (pros::millis() > t0)
                break;
            wheelPos = left2.get_position();
            int driveVoltage = (wheelSeek - wheelPos)*60;

            if(driveVoltage > -1*MIN_DRIVE_VOLTAGE)
                driveVoltage = -1*MIN_DRIVE_VOLTAGE;

            driveMotorsAt(driveVoltage);
        }
        driveMotorsAt(0);
    }
    delay(1);
}

/*slow drive fwd*/
void driveForward(int wheelSeek, int maxVoltage){
    int t0 = pros::millis();

    tareDriveMotors();
    int wheelPos = left2.get_position();
    if(wheelSeek > 0){
        while(wheelPos < wheelSeek){
            if(pros::millis() == t0 + abs(wheelSeek) * 3)
                break;

            wheelPos = left2.get_position();
            int driveVoltage = (wheelSeek - wheelPos)*60;

            if(driveVoltage < MIN_DRIVE_VOLTAGE)
                driveVoltage = MIN_DRIVE_VOLTAGE;
            /*if(driveVoltage > maxVoltage)
                driveVoltage = maxVoltage;*/

            driveMotorsAt(driveVoltage);
        }
        driveMotorsAt(0);
    }
    else{
        while(wheelPos > wheelSeek){
            if(pros::millis() == t0 + abs(wheelSeek) * 3)
                break;
            wheelPos = left2.get_position();
            int driveVoltage = (wheelSeek - wheelPos)*60;

            if(driveVoltage > -1*MIN_DRIVE_VOLTAGE)
                driveVoltage = -1*MIN_DRIVE_VOLTAGE;

            driveMotorsAt(driveVoltage);
        }
        driveMotorsAt(0);
    }
    delay(1);
}

void run_arm(void* param){
    motorApexA.tare_position();

    while(true){
        if(armAutonSeek != -1){
            armAutonPos = motorApexA.get_position();
            pros::lcd::print(4,"armAutonPos: %f\n", armAutonPos);
            pros::lcd::print(5,"armAutonSeek: %f\n", armAutonSeek);

            armAutonPower = (armAutonSeek - armAutonPos)*60;
            if(armAutonSeek > 0){
                if(armAutonPower > 12000)
                    armAutonPower = 12000;

            }
            else if(armAutonSeek < 0){
                if(armAutonPower < -12000)
                    armAutonPower = -12000;

            }
        }
        pros::delay(20);
    }
}
```