```
#pragma config(Sensor, in1,    expander,       sensorAnalog)
#pragma config(Sensor, dgtl1,  armQuad,        sensorQuadEncoder)
#pragma config(Sensor, dgtl3,  gateSwitch,     sensorDigitalIn)
#pragma config(Sensor, dgtl4,  armSwitch,      sensorDigitalIn)
#pragma config(Sensor, dgtl5,  gateQuad,       sensorQuadEncoder)
#pragma config(Sensor, dgtl7,  leftSonar,      sensorSONAR_cm)
#pragma config(Sensor, dgtl9,  backLeftSonar,  sensorSONAR_cm)
#pragma config(Sensor, dgtl11, backRightSonar, sensorSONAR_cm)
#pragma config(Motor,  port2,           yOne,          tmotorVex393HighSpeed_MC29, openLoop, reversed)
#pragma config(Motor,  port3,           yTwo,          tmotorVex393HighSpeed_MC29, openLoop)
#pragma config(Motor,  port4,           yThree,        tmotorVex393HighSpeed_MC29, openLoop, reversed)
#pragma config(Motor,  port5,           gate,          tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port6,           frontRight,    tmotorVex393HighSpeed_MC29, openLoop, reversed)
#pragma config(Motor,  port7,           backRight,     tmotorVex393HighSpeed_MC29, openLoop, reversed)
#pragma config(Motor,  port8,           backLeft,      tmotorVex393HighSpeed_MC29, openLoop)
#pragma config(Motor,  port9,           frontLeft,     tmotorVex393HighSpeed_MC29, openLoop)
#pragma config(Motor,  port10,          intake,        tmotorVex393HighSpeed_HBridge, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

#pragma platform(VEX)

#pragma competitionControl(Competition)
#pragma autonomousDuration(15)
#pragma userControlDuration(105)

#include "Vex_Competition_Includes.c"
#include "catapult.c"

//Sonar Values for autonomous movement
const int SONIC_STOP_SIDEWAYS = 31;
const int SONIC_STOP_FB = 15;
const int SONIC_STOP_AIM_LEFT = 47;
const int SONIC_STOP_AIM_RIGHT = 53;

//Align left side of robot with back wall using SONIC_STOP_FB value
void alignBackLeftFB(){
	wait1Msec(50);
		while(SensorValue[backLeftSonar] > SONIC_STOP_FB){
			drive(true,true,FULL_POWER / 4);
		}
	stopAllDrive();
}

//Align right side of robot with back wall using SONIC_STOP_FB value
```

```
void alignBackRightFB(){
	wait1Msec(50);
		while(SensorValue[backRightSonar] > SONIC_STOP_FB){
			drive(false,true,FULL_POWER / 4);
		}

	stopAllDrive();
}

//Move robot left until it is a set distance away from the side wall
void alignLeftSide(bool goingLeft){
	if(goingLeft){

		wait1Msec(50);
		while(SensorValue[leftSonar] > SONIC_STOP_SIDEWAYS){
			mechanumDrive(true);
		}
		stopAllDrive();

		}else{
		wait1Msec(50);
		while(SensorValue[leftSonar] < SONIC_STOP_SIDEWAYS){
			mechanumDrive(false);
		}
		stopAllDrive();
	}
}

//Align left side of robot with back wall using SONIC_STOP_AIM value
void alignBackLeftAim(bool backwards){
	if(backwards){

		wait1Msec(50);
		while(SensorValue[backLeftSonar] > SONIC_STOP_AIM_LEFT){
			drive(true,true,FULL_POWER / 4);
		}
		stopAllDrive();

		}else{
		wait1Msec(50);
		while(SensorValue[backLeftSonar] < SONIC_STOP_AIM_LEFT){
			drive(true,false,FULL_POWER / 4);
		}
		stopAllDrive();
```

```
        }
}

//Align right side of robot with back wall using SONIC_STOP_AIM value
void alignBackRightAim(bool backwards){
        if(backwards){

                wait1Msec(50);
                while(SensorValue[backRightSonar] > SONIC_STOP_AIM_RIGHT){
                        drive(false,true,FULL_POWER / 4);
                }
                stopAllDrive();

                }else{
                wait1Msec(50);
                while(SensorValue[backRightSonar] < SONIC_STOP_AIM_RIGHT){
                        drive(false,false,FULL_POWER / 4);
                }
                stopAllDrive();
        }
}

task autonomous(){
        //Tell firing routine this is a skills run
        skills = true;

        //Set arm and gate to initial positions
        setGate(GATE_OPEN);
        setArm(ARM_LOAD);

        //Zero Sensors
        resetSensors();

        //Turn intake on
        motor[intake] = -127;

        //Start gate and arm monitoring processes
        startTask(armPosition);
        startTask(gatePosition);

        //Tweak Powers for the longer distance
        midPower = 61;
        mechanumPower = 127;
```

```
	//Shoot 32 balls
	ballCount = 0;
	while(ballCount < 1){
		setArm(ARM_FIRE);
		setDistance(MID);
	}

	//Make robot parallel with back wall
	alignBackLeftFB();

	alignBackRightFB();

	alignBackLeftFB();

	alignBackRightFB();

	//Drive forward to avoid ball triangle
	drive(true,false,127);
	drive(false,false,127);
	wait1Msec(200);
	stopAllDrive();

	//Drive left initially blind to get sonar in range
	mechanumDrive(true);
	wait1Msec(1750);

	//Drive left using sonar
	alignLeftSide(true);

	//Lower strafe speed to get more accurate
	mechanumPower = mechanumPower - 92;

	//Face robot towards left net using multiple trials to avoid sonar error
	alignLeftSide(false);

	alignLeftSide(true);

	alignLeftSide(false);

	alignLeftSide(true);

	alignLeftSide(false);

	alignBackLeftAim(true);
```

```
		alignBackRightAim(false);

		alignBackLeftAim(true);

		alignLeftSide(false);

		alignLeftSide(true);

		alignLeftSide(false);

		alignLeftSide(true);

		alignLeftSide(false);

		alignBackRightAim(true);

		alignBackLeftAim(false);

		alignBackRightAim(true);

		alignBackLeftAim(false);

		alignBackRightAim(true);

		//Lower firing power because it is a little closer to the net on the left side
		midPower -= 9;

		//Fire another 32 balls
		ballCount = 0;
		while(ballCount < 32){
			setArm(ARM_FIRE);
			setDistance(MID);
		}
}

task usercontrol(){
		//Tell firing routine to go back to normal
		skills = false;

		//SAME AS MAIN DRIVER LOOP
		setGate(GATE_OPEN);
		setArm(ARM_FIRE);
```

```
	override = true;

	//Reset speeds so it drives normally
	midPower = 50;
	mechanumPower = FULL_POWER;

	startTask(armPosition);
	startTask(gatePosition);

	motor[intake] = -127;

	while(true){
		//Firing Controls
		if(vexRT[Btn7U] == 1){
			setDistance(CORNER);
			setArm(ARM_FIRE);
			}else{
			if(vexRT[Btn7L] == 1){
				setDistance(TILE);
				setArm(ARM_FIRE);
				}else{
				if(vexRT[Btn7D] == 1){
					setDistance(MID);
					setArm(ARM_FIRE);
					}else{
					if(vexRT[Btn7R] == 1){
						setDistance(SHORT);
						setArm(ARM_FIRE);
					}
				}
			}
		}

		if(vexRT[Btn8R] == true){
			setGate(GATE_OPEN);
		}

		if(vexRT[Btn6D] == true){
			rotate(false);
		}
		else{
			if(vexRT[Btn5D] == true){
				rotate(true);
			}
```

```
                        else{
                                if(vexRT[Btn6U] == true){
                                        mechanumDrive(false);
                                }
                                else{
                                        if(vexRT[Btn5U] == true){
                                                mechanumDrive(true);
                                        }
                                        else{
                                                tankDrive();
                                        }
                                }
                        }
                }
        }
}
```