```c
#pragma config(UART_Usage, UART1, uartVEXLCD, baudRate19200, IOPins, None, None)
#pragma config(UART_Usage, UART2, uartNotUsed, baudRate4800, IOPins, None, None)
#pragma config(Sensor, dgtl1,  leftFly,        sensorQuadEncoder)
#pragma config(Sensor, dgtl3,  rearSonarR,     sensorSONAR_inch)
#pragma config(Sensor, dgtl5,  rearSonarL,     sensorSONAR_inch)
#pragma config(Sensor, dgtl7,  leftSonar,      sensorSONAR_inch)
#pragma config(Sensor, dgtl9,  ballSwitch,     sensorDigitalIn)
#pragma config(Sensor, dgtl11, rightFly,       sensorQuadEncoder)
#pragma config(Motor,  port1,           rightRail,     tmotorVex393_HBridge, openLoop)
#pragma config(Motor,  port2,           frontLeft,     tmotorVex393HighSpeed_MC29, openLoop)
#pragma config(Motor,  port3,           backLeft,      tmotorVex393HighSpeed_MC29, openLoop)
#pragma config(Motor,  port4,           sideIntake,    tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port5,           leftFlywheel,  tmotorVex393TurboSpeed_MC29, openLoop)
#pragma config(Motor,  port6,           hopper,        tmotorVex393TurboSpeed_MC29, openLoop)
#pragma config(Motor,  port7,           backRight,     tmotorVex393HighSpeed_MC29, openLoop, reversed)
#pragma config(Motor,  port8,           frontRight,    tmotorVex393HighSpeed_MC29, openLoop, reversed)
#pragma config(Motor,  port9,           rightFlywheel, tmotorVex393TurboSpeed_MC29, openLoop, reversed)
#pragma config(Motor,  port10,          frontIntake,   tmotorVex393_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 "Flywheel.h"

int ballCount = 0; //Running counter of balls fired

//Driver control variables
int lastIntakeButton = 0; // (1 is 5U) (2 is 5D)
int deadZone = 10; //minimum controller input before motors are turned on (prevents whining)
bool isIntake = false; // Used for the toggling behavior for intakes

//Setpoint formula = (Wanted RPM) * RPM_VAR
//Desired Ticks Per DELTA_TIME
float setPoint = 0;

//Min and Max motor speeds for Clamp Function
const float MIN_MOTOR_SPEED = 0;
const float MAX_MOTOR_SPEED = 127;
```

```c
//Milliseconds between each PID Loop execution
const float DELTA_TIME = 250;
const float DELTA_TIME_IN_SECONDS = DELTA_TIME/1000;

//To convert from Ticks Per DELTA_TIME to RPM: ticks/RPM_VAR
//To convert from RPM to Ticks Per DELTA_TIME: RPM * RPM_VAR
const float RPM_VAR = DELTA_TIME / 166;
const float SETPOINT_DELTA = 2;

//Variables required by PID controller (left)
//Derivative Formula = (currentError - lastError)/delta_time
float pGainLeft = 0;
float iGainLeft = 0;
float dGainLeft = 0;
bool PID_Running = false;
bool isAuto = false;

float derivativeLeft = 0;

//Current error in ticks from left Quad
float errorLeft = 0;

//Error from last 'check' in ticks from left Quad
float lastErrorLeft = 0;

//Total error from bot startup
float cumErrorLeft = 0;

//Motor speed output for leftFlywheel
float outputLeft = 0;

//Variables required by PID Controller (right)
float pGainRight = 0;
float iGainRight = 0;
float dGainRight = 0;
float derivativeRight = 0;

//Current error in ticks from right Quad
float errorRight = 0;

//Error from last 'check' in ticks from right Quad
float lastErrorRight = 0;

//Total error from bot startup
```

```
float cumErrorRight = 0;

//Motor speed output for rightFlywheel
float outputRight = 0;

void pre_auton(){
	bStopTasksBetweenModes = true;
}

task PID(){
	//Use SetTunings method before starting PID task
	//Initialize and reset PID variables
	bLCDBacklight = true;
	while(true){
		if(!PID_Running){
			derivativeLeft = 0;
			errorLeft = 0;
			lastErrorLeft = 0;
			cumErrorLeft = 0;
			outputLeft = 0;

			derivativeRight = 0;
			errorRight = 0;
			lastErrorRight = 0;
			cumErrorRight = 0;
			outputRight = 0;
			clearLCDLine(0);
			clearLCDLine(1);
			displayLCDNumber(0,0,0);
			resetSensors();


			clearTimer(T1);
			}else{
			while(PID_Running){
				//PID Loop
				if(	time1[T1] >= DELTA_TIME){
					computeLeft();
					computeRight();
					//writeDebugStream("%d\n", cumErrorRight);
					clearTimer(T1);
				}
			}
		}
```

```
        }
}

//Drives to the middle to shoot 4 balls
task autonomous(){
        isAuto = true;
        resetSensors();
        setTunings(2.0, 0.1, 0.01, 0);
        setTunings(2.0, 0.1, 0.01, 1);
        setPoint = 186 * RPM_VAR;
        PID_Running = true;
        startTask(PID);
        wait1Msec(1500);
        motor[frontLeft] = 45;
        motor[frontRight] = 45;
        motor[backLeft] = 45;
        motor[backRight] = -45;
        wait1Msec(2000);
        motor[frontLeft] =0;
        motor[frontRight] = 0;
        motor[backLeft] = 0;
        motor[backRight] = 0;
        motor[hopper] = -90;
        wait1Msec(2000);
        motor[frontIntake] = 127;
        motor[sideIntake] = -127;
}

task usercontrol()
{
        isAuto = false;
        resetSensors();
        //setTunings(2.38, 0.07, 0.001, 0);
        //setTunings(2.38, 0.07, 0.001, 1);
        PID_Running = false;
        setTunings(2.0, 0.1, 0.01, 0);
        setTunings(2.0, 0.1, 0.01, 1);
        startTask(PID);
        wait1Msec(50);
        PID_Running = true;


        while(true){
                //SetPoint Controls (Adam)
```

```
		if(vexRT[Btn6UXmtr2] == 1){
			PID_Running = true;
			cumErrorRatio(setPoint + (SETPOINT_DELTA * RPM_VAR));
			wait1Msec(250);
		}
		else if(vexRT[Btn6DXmtr2] == 1){
			PID_Running = true;
			cumErrorRatio(setPoint - (SETPOINT_DELTA * RPM_VAR));
			wait1Msec(250);
		}
		else if(vexRT[Btn7UXmtr2] == 1){
			PID_Running = true;
			cumErrorRatio(210 * RPM_VAR);
		}
		else if(vexRT[Btn7LXmtr2] == 1){
			PID_Running = true;
			cumErrorRatio(190 * RPM_VAR);
		}
		else if(vexRT[Btn7DXmtr2] == 1){
			PID_Running = true;
			cumErrorRatio(155 * RPM_VAR);
		}
		else if(vexRT[Btn7RXmtr2] == 1){
			PID_Running = false;
			motor[leftFlywheel] = 0;
			motor[rightFlywheel] = 0;
			setPoint = 0;
		}

		//Lift Controls
		if(vexRT[Btn8L] && vexRT[Btn8LXmtr2]){
			motor[rightRail] = -127;
		}else if(vexRT[Btn8U])
			motor[rightRail] = 127;
		else
			motor[rightRail] = 0;

		//Manual Hopper Controls (Lydia)
		if(vexRT[Btn6U] == 1){
			motor[hopper] = -127;	//Shooting balls
		}
		else
			if(vexRT[Btn6D] == 1)
			motor[hopper] = 127;
```

```
	else
		motor[hopper] = 0;

	//Intake controls (Adam)
	if(vexRT[Btn5UXmtr2] == 1){
		if(lastIntakeButton == 1 && isIntake){
			motor[frontIntake] = 0;
			motor[sideIntake] = 0;
			isIntake = false;
		}
		else{
			motor[frontIntake] = 127;
			motor[sideIntake] = -127;
			isIntake = true;
		}
		lastIntakeButton = 1;
		wait1Msec(200);
	}
	else{
		if(vexRT[Btn5DXmtr2] == 1){
			if(lastIntakeButton == 2 && isIntake){
				motor[frontIntake] = 0;
				motor[sideIntake] = 0;
				isIntake = false;
			}
			else{
				motor[frontIntake] = -127;
				motor[sideIntake] = 127;
				isIntake = true;
			}
			lastIntakeButton = 2;
			wait1Msec(200);
		}
	}

	//Drive base controls (Lydia)
	if(vexRT[Btn7L] == 1){
		mechanumDrive(true, 127);
	}else if(vexRT[Btn8R] == 1){
		mechanumDrive(false, 127);
	}
	else if(abs(vexRT[Ch3]) > deadZone || abs(vexRT[Ch2]) > deadZone)
	{
		motor[frontLeft] = vexRT[Ch3];
```

```
				motor[frontRight] = vexRT[Ch2];
				motor[backLeft] = vexRT[Ch3];
				motor[backRight] = -vexRT[Ch2];
			}
			else
			{
				stopDrive();
			}
		}
}

int clamp(int var, int min, int max){
		if(var < min)
			return min;
		else if (var > max)
			return max;
		else return var;
}

void setTunings(float pGain, float iGain, float dGain, int side){
		if(side == 0){
			pGainLeft = pGain;
			iGainLeft = iGain;
			dGainLeft = dGain;
		}
		else if(side == 1){
			pGainRight = pGain;
			iGainRight = iGain;
			dGainRight = dGain;
		}
}

void computeLeft(){
		int sensorVal = SensorValue(leftFly);
		SensorValue[leftFly] = 0;
		errorLeft = setPoint - sensorVal;
		clearLCDLine(0);
		displayLCDNumber(0,0, sensorVal / RPM_VAR);
		if((sensorVal / RPM_VAR) > (setPoint / RPM_VAR))
			cumErrorLeft += errorLeft * 1.7;
		else
			cumErrorLeft += errorLeft;
		derivativeLeft = (errorLeft - lastErrorLeft)/(DELTA_TIME_IN_SECONDS);
		lastErrorLeft = errorLeft;
```

```
        outputLeft = clamp(((pGainLeft * errorLeft) + (iGainLeft * cumErrorLeft) + (dGainLeft * derivativeLeft)),
MIN_MOTOR_SPEED, MAX_MOTOR_SPEED);
        motor[leftFlywheel] = outputLeft;
}

void computeRight(){
        int sensorVal = abs(SensorValue(rightFly));
        SensorValue[rightFly] = 0;
        errorRight = setPoint - sensorVal;
        clearLCDLine(1);
        displayLCDNumber(1,0, sensorVal / RPM_VAR);
        displayLCDNumber(1,13, setPoint / RPM_VAR);
        if((sensorVal / RPM_VAR) > (setPoint / RPM_VAR))
                cumErrorRight += errorRight * 1.7;
        else
                cumErrorRight += errorRight;
        derivativeRight = (errorRight - lastErrorRight)/(DELTA_TIME_IN_SECONDS);
        lastErrorRight = errorRight;
        outputRight = clamp((int)((pGainRight * errorRight) + (iGainRight * cumErrorRight) + (dGainRight *
derivativeRight)), MIN_MOTOR_SPEED, MAX_MOTOR_SPEED);
        motor[rightFlywheel] = outputRight;
}

void cumErrorRatio(float newSetPoint){
        if(newSetPoint != 0){
                float lastSetPoint = setPoint;
                setPoint = newSetPoint;
                if(lastSetPoint != 0){
                        cumErrorLeft = (setPoint / lastSetPoint) * cumErrorLeft;
                        cumErrorRight = (setPoint / lastSetPoint) * cumErrorRight;
                }
                if((lastSetPoint - newSetPoint) >= 50){
                        cumErrorRight += 650;
                        cumErrorLeft += 650;
                }
        }
}

void resetSensors(){
        SensorValue[leftFly] = 0;
        SensorValue[rightFly] = 0;
}

void mechanumDrive(bool left, int power){
```

```
	if(left){
		motor[frontLeft] = -power;
		motor[backLeft] = power;
		motor[frontRight] = power;
		motor[backRight] = power;
		}else{
		motor[frontLeft] = power;
		motor[backLeft] = -power;
		motor[frontRight] = -power;
		motor[backRight] = -power;
	}
}

void stopDrive(void){
	motor[frontLeft] = 0;
	motor[backLeft] = 0;
	motor[frontRight] = 0;
	motor[backRight] = 0;
}
```