```
#pragma config(UART_Usage, UART1, uartUserControl, baudRate1200, IOPins, None, N
#pragma config(I2C_Usage, I2C1, i2cSensors)
#pragma config(Sensor, in1,    PowerExpander,  sensorAnalog)
#pragma config(Sensor, in2,    Gyro,           sensorGyro)
#pragma config(Sensor, in8,    potentSpeed,    sensorPotentiometer)
#pragma config(Sensor, dgtl1,  LauncherEncoder, sensorQuadEncoder)
#pragma config(Sensor, dgtl3,  Trans,          sensorDigitalOut)
#pragma config(Sensor, dgtl4,  Brakes,         sensorDigitalOut)
#pragma config(Sensor, dgtl5,  IntakeT,        sensorTouch)
#pragma config(Sensor, dgtl6,  UptakeT,        sensorTouch)
#pragma config(Sensor, dgtl7,  RightDrive,     sensorQuadEncoder)
#pragma config(Sensor, dgtl9,  LeftDrive,      sensorQuadEncoder)
#pragma config(Sensor, dgtl11, PSITest,        sensorDigitalOut)
#pragma config(Sensor, dgtl12, PSITest2,       sensorTouch)
#pragma config(Sensor, I2C_1,  ,               sensorQuadEncoderOnI2CPort,    ,
#pragma config(Motor,  port2,           LeftFMotor,    tmotorVex393_MC29, openLo
#pragma config(Motor,  port3,           RightFMotor,   tmotorVex393_MC29, openLo
#pragma config(Motor,  port4,           Launcher1,     tmotorVex393_MC29, openLo
#pragma config(Motor,  port5,           Launcher2,     tmotorVex393_MC29, openLo
#pragma config(Motor,  port6,           Uptake,        tmotorVex393_MC29, openLo
#pragma config(Motor,  port7,           Intake,        tmotorVex393TurboSpeed_MC
#pragma config(Motor,  port8,           RightBMotor,   tmotorVex393_MC29, openLo
#pragma config(Motor,  port9,           LeftBMotor,    tmotorVex393_MC29, openLo
//*!!Code automatically generated by 'ROBOTC' configuration wizard

#pragma platform(VEX2)

//Competition Control and Duration Settings
#pragma competitionControl(Competition)
#pragma autonomousDuration(20)
#pragma userControlDuration(120)

#include "Vex_Competition_Includes.c"    //OFFICAL VEX COMPETITION STRUCTURE COD
#include "lcdLib.c"                      //JPearman's LCDLibary used for 2nd LCD
#include "SmartMotorLib.c"               //JPearman's SmartMotor Libary to help

const short leftButton = 1;     //Declare short names for Buttons on LCD
const short centerButton = 2;
const short rightButton = 4;
float powerExpanderVoltage = 0; //Declare variable to keep track of PowerExpande
string str;     //Strings for LCD Screens
string str2;    //  |
string str3;    //  |
string str4;    //  |
string str5;    // \|/
float PowerExpanderCurrent;
float BackupCurrent;
int count = 0;
int color = 0;
int style = 0;
int AdvancedCheck = 0;
float countspeed = 40.0; //Speed Variable for Autonomous Launcher
int BatteriesReady = 0;
int PSIReady = 0;
int BackupTone = 0;
int ExpanderTone = 0;
float X1;
float Y1;
float OutputY;
```

```
float OutputX;
float SecondOutputX;
float drive speed = 1.0;
int PreAutonOver = 0;
int Auton = 0;
float pidRequestedValue = 0;
int ToggleTrans = 0;
int PreAutonLCD = 1;
float motorPower = 0;
float LauncherSpeed = 0;
int shiftedUptake = 80; //Hard Estimate: 60 Soft: 80
bool skills = false;

void ClearLCDs() { //Function used to Clear all of our LCD Screens
  clearLCDLine(0);  //Clear Line 0 of LCD1
  clearLCDLine(1);  //Clear Line 1 of LCD1
  vexLcdClearLine(0);  //Clear Line 0 of LCD2
  vexLcdClearLine(1);  //Clear Line 1 of LCD2
}

void Launcher(int power) { //Function for easy launcher power adjustment.
  motor[Launcher1] = power;  //Set motor Launcher1 to power var
  motor[Launcher2] = power;  //Set motor Launcher2 to power var
}

float RPSEncoder; //Variable to see speed
task readEncoder()  //Launcher PID Task & function
{ //Calculates Rotations Per Second (RPS) of our flywheel
  motor[Launcher1] = 0; //Clear Launcher Motors
  motor[Launcher2] = 0; //Clear Launcher Motors
  long lastSysTime = nSysTime; //Variables for differnce in time
  int deltaSysTime;      //Initalize deltaSysTime as a int var
  long lastEncoder = SensorValue[LauncherEncoder]; // Variables for difference i
  int deltaEncoder;     //Initalize deltaEncoder as a int var
  wait1Msec(20);  //Wait 20 Milliseconds
  while(1 == 1) { //Runs forever
    deltaSysTime = nSysTime - lastSysTime; //Update difference in time
    lastSysTime = nSysTime; //set lastSysTime to nSysTime
    deltaEncoder = SensorValue[LauncherEncoder] - lastEncoder; //Update differer
    lastEncoder = SensorValue[LauncherEncoder];
    RPSEncoder = (deltaEncoder / 360.0) * (1000.0/deltaSysTime); //Calculates Ve
    wait1Msec(20); //wait 20 Milliseconds
  }
}

float pre_error;     //  |  Define All PID Variables
float integral;      //  |
float epsilon = 0;   //  |
float dt = 1;        //  |
float Max;           //  |
float Min;           //  |
float Max2;          //  |
float Min2;          //  |
float Ki;            //  |
float Kp;            //  |
float Kd;            //  |
float output;        //  |
float error;         //  |
string KpString;     //  |
```

```
string KiString;     //  |
string KdString;     //  |
string ErrorString; //  |
string StableString;//  |
bool turbo = false; // \|/

float PidFunction(float target position, float current position)
{
  Max = 127; //Set Motor Max
  Min = -127; //Set Motor Min
  Max2 = 1200; //Set Integral Max
  Min2 = 0; //Set Integral Min
  float stabilizer = 1.0; //Variable to Stabilize the Error Rate of Change
  float derivative; //Rate of change of launcher velocity
  bool shoot = false; //Shoot Variable to boost speed for Fast Shooting
  error = target_position - current_position; //Calculate Error
  if(target_position <= 34.0) { //Allows us to have different levels of agressic
    stabilizer = 1.0;
    Kp = 4.75; //Values for Close and Mid Shooting
    if(vexRT[Btn5U] == 1 && SensorValue[UptakeT] == 1) { //If the driver wants t
      shoot = true; //Boost
    }
    if(vexRT[Btn5U] == 0) { //When we are no longer shooting
      shoot = false; //Stop Boost
    }
    if(shoot || turbo) { //If we are shooting
      Ki = 1.8; //Temporarly boost speed to fire quickly
      Kd = 1.0;
      } else { //If not
      Ki = 0.3; //Restabilize
      Kd = 0.15;
    }
    } else {
    if(!turbo && SensorValue[UptakeT] == 0) { //Regular values
      Kp = 3.5; //Values for Far shooting
      Ki = 0.6;
      Kd = 0.5;
      stabilizer = 2.5;
      }else{
      if(error > 0.5 && SensorValue[UptakeT] == 1 && !turbo) { //If we want to s
        Ki = error * 0.1; //Adaptive Integral to stabilize and help with shootir
        playTone(20, 20); //Notify us we are using this code
        } else if (error < 0.0 && SensorValue[UptakeT] == 1){ //If flywheel is t
        Ki = 0.01; //Slow it down
        } else {
        Ki = 0.2;   //Orignal Var
      }
      Kp = 2.0; //Keep same other variables
      Kd = 0.1;
      stabilizer = 1.6;
    }
  }
  if(skills && SensorValue[Trans] == 1) { //Skills PID Values to shoot with 8 mc
    Kp = 2.5;
    Ki = 0.05; //Low Integral to keep it very constant
    Kd = 0.75;
    stabilizer = 1.0;
  }
  if(abs(error) > 5) { //Limit error to minimize possible destruction on the lau
```

```
    if(error > 0) {
      error = 5; //Set Max error to 5
    }
    if(error < 0) {
      error = -5; //Set Min error to -5
    }
  }
  if(abs(error) > epsilon) {
    integral = integral + (error*stabilizer); //Calculate Integral
  }
  if(integral > Max2) { //Maximize the integral to limit long overshoots
    integral = Max2; //Set Integral max to 1200
  }
  if(integral < Min2) {
    integral = Min2; //Set Integral min to 0
  }
  derivative = (error-pre_error)/dt; //Calculate Derrivative
  if(target_position != 0) {
    output = (Kp*error) + (Ki*integral) + (Kd*derivative); //Calculate motor Out
  }
  if(output>Max) { //Maximize output to limit PWM overflow of underflow errors
    output = Max; //Set Motor max to 127;
  }
  if(output<Min) {
    output = Min; //Set Motor min to -127
  }
  if(target_position == 0) { //Limit motor destruction on slowdown
    output = output - 0.5;
    if(output < 50) { //Cut off motors at 50 Power
      output = 0;
    }
  }
  if(SensorValue[UptakeT] == 1){
    KpString = Kp;                                          //Print Debug Variable
    KiString = Ki;                                          //       |
    KdString = Kd;                                          //       |
    ErrorString = error;                                    //       |
    StableString = stabilizer;                              //       |
    writeDebugStreamLine("Error = " , ErrorString);         //       |
    writeDebugStreamLine("Kp = ", KpString);                //       |
    writeDebugStreamLine("Ki = ", KiString);                //       |
    writeDebugStreamLine("Kd = ", KdString);                //       |
    writeDebugStreamLine("stabilizer = " , StableString); //     \|/
  }
  return output; //return motor power
}


task LCDScreens() {  //Task that Controls LCD Screens
  vexLcdInit(UART1);  //Initalize LCD Screen in UART Port 1 as LCD2
  vexLcdBacklight(1); //Turn LCD2 Backlight on
  while(1==1){ //Run Loop Forever
    powerExpanderVoltage = SensorValue[PowerExpander] / 280.0; //Set to PowerExp
    sprintf(str2, "%5.2fV", powerExpanderVoltage ); //Set String to powerExpande
    sprintf(str3, "%1.2f%c", nImmediateBatteryLevel/1000.0,'V'); //Set string to
    if(PreAutonLCD == 1){ //During PreAuton, LCD Screen 2 displays Battery Volta
      vexLcdSetAt( 0, 0, "Primary:   "); //Display Primary Battery Voltage on LC
      vexLcdSetAt( 0, 11, str3 );
      vexLcdSetAt( 1, 0, "Launcher: ");  //Display Launcher Battery Voltage on I
```

```
      vexLcdSetAt( 1, 10, str2 );
      } else if(PreAutonLCD == 0){ //If NOT in PreAuton
      if(Auton == 1){  //If in Autonomous
        sprintf(str4, "Current: %3.3f", RPSEncoder );  //Display Current RPS on
        vexLcdSetAt(0, 0, str4);
        sprintf(str5, "Desired: %3.1f", LauncherSpeed ); //Display Desired RPS c
        vexLcdSetAt(1, 0, str5);
        } else {  //Else (if in usercontrol)
        sprintf(str4, "Current: %3.3f", RPSEncoder ); //Display Current RPS on I
        vexLcdSetAt(0, 0, str4);
        sprintf(str5, "Desired: %3.1f", pidRequestedValue ); //Display Desired F
        vexLcdSetAt(1, 0, str5);
      }
      //Display Battery Voltage on LCD Screen 1
      displayLCDString(0, 0, "Primary:   ");  //Display Primary Battery Voltage
      displayLCDString(0, 11, str3);
      displayLCDString(1, 0, "Launcher: ");   //Display Launcher Battery Voltage
      displayLCDString(1, 10, str2);
    }
  }
  wait1Msec(20);      //Wait 20 Milliseconds
}

int motorshift = 1; //Initialize motorshift as int and set to 1
task count250() {  //Task that counts 250 Milliseconds
  wait1Msec(250);  //wait 250 Milliseconds
  motorshift = 1;  //Set motorshift to 1
}

task shiftplus() { //Task to allow for smooth shifting
  motorshift = 0;  //Set motorshift var to 0
  startTask(count250); //Runs for 250 miliseconds
  while(motorshift == 0) {   //While motorshift var equals 0
    if(motorPower > 30){
      Launcher(abs(motorPower));          //Set Launcher to motorPower var
      motor[RightFMotor] = -abs(motorPower);//Set RightFMotor to Negative the ab
      motor[RightBMotor] = -abs(motorPower);//Set RightBMotor to Negative the ab
      motor[LeftFMotor] = -abs(motorPower);//Set LeftFMotor to Negative the absc
      motor[LeftBMotor] = -abs(motorPower);//Set LeftBMotor to Negative the absc
      } else {            //Spins motors slowly if Launcher is not spinning
      Launcher(30);       //Set Launcher to 30 MotorPower
      motor[RightFMotor] = -30; //Set RightFMotor to -30
      motor[RightBMotor] = -30; //Set RightBMotor to -30
      motor[LeftFMotor] = -30;  //Set LeftFMotor to -30
      motor[LeftBMotor] = -30;  //Set LeftBMotor to -30
    }
  }
  stopTask(count250);       //Stop Task count250
  stopTask(shiftplus);      //Stop Task shiftplus
}

task AccelControl() {   //Task that Actively Controls the Acceleration of our Dr
  while(1==1){  //Run Forever
    if(abs(OutputY) < abs(Y1) && abs(Y1) > 25 && abs(OutputY) < 25){ //IF Y1 gre
      if(Y1 > 0){ //IF Y1 is greater than 0
        OutputY = 25;  //Set OutputY to 25
        } else { //Else
        OutputY = -25; //Set OutputY to -25
      }
```

```
    }
    if(OutputY < Y1 && abs(Y1) > 25){ //ForwardAccel
      OutputY = OutputY + 0.5;  //Add 0.5 to OutputY
      } else if(abs(Y1) < 25 && OutputY < 20){  //Decel
      OutputY = OutputY + 0.8;  //Add 0.8 to OutputY
    }
    if(OutputY > Y1 && abs(Y1) > 25){ //Reverse Accel
      OutputY = OutputY - 0.5;  //Subtract 0.5 from OutputY
      } else if(abs(Y1) < 25 && OutputY > 20){ //Reverse Decel
      OutputY = OutputY - 0.8;  //Subtract 0.8 from OutputY
    }
    if(abs(OutputY) < 25 && abs(Y1) < 25 ){  //IF Y1 within threshold
      OutputY = 0;    //Set OutputY to 0
    }
    if(abs(OutputX) < abs(X1) && abs(X1) > 25 && abs(OutputX) < 25){  //IF X1 gr
      if(X1 > 0){  //IF X1 is greater than 0
        OutputX = 25; //Set OutputX to 25
        } else {
        OutputX = -25; //Set OutputX to -25
      }
    }
    if(OutputX < X1 && abs(X1) > 25){ //ForwardAccel
      OutputX = OutputX + 0.8;  //Add 0.8 to OutputX
      } else if(abs(X1) < 25 && OutputX < 20){  //Decel
      OutputX = OutputX + 0.8;  //Add 0.8 to OutputX
    }
    if(OutputX > X1 && abs(X1) > 25){ //Reverse Accel
      OutputX = OutputX - 0.8;  //Subtract 0.8 from OutputX
      } else if(abs(X1) < 25 && OutputX > 20){ //Decel
      OutputX = OutputX - 0.8;  //Subtract 0.8 from OutputX
    }
    if(abs(OutputX) < 25 && abs(X1) < 25 ){  //IF X1 within thresehold
      OutputX = 0;        //Set OutputX to 0
    }
    wait1Msec(1);  //Wait 1 Millisecond
  }
}

task BatteryPSICheck()  {     //This Task is written to Alert us if one of batte
  //NOTE:  Variables in this task are also used back in PreAuton, where it is re
  while(BatteriesReady == 0){   //While Batteries are not ready run the followir
    PowerExpanderCurrent = SensorValue[PowerExpander];  //Set value of PowerExpa
    BackupCurrent = BackupBatteryLevel/1000.0;          //Set value of BackupBat
    BackupTone = 0;                                     //Set variable for check
    ExpanderTone = 0;                                   //Set variable for check
    if(BackupCurrent < 3){                        //If BackupBattery Voltage less
      BackupTone = 1;                               //BackupTone is set to 1
    }
    if(PowerExpanderCurrent < 500){               //If PowerExpander Voltage less
      ExpanderTone = 1;                             //ExpanderTone is set to 1
    }
    if(BackupTone == 0 && ExpanderTone == 0){     //If BackupTone AND ExpanderTone
      BatteriesReady = 1;                           //Batteries Are Ready, end whi
      } else if(BackupTone == 1){                 //Else if BackupTone equals 1
      playTone(100, 100);                           //Play a Tone on Speaker
      wait1Msec(400);                               //Wait 100 Milliseconds
      } else if(ExpanderTone == 1) {              //Else if ExpanderTone equals 1
      playTone(420, 62.5);                          //Play a Tone on Speaker
      wait1Msec(1250);                              //Wait 1250 Milliseconds
```

```
    }
  }
  //This Part of the Task is written to check if our Air has been turned on befc
  while(PSIReady == 0) {            //While The Air is not on run the following:
    SensorValue[PSITest] = 1;         //Test Piston is turned on
    wait1Msec(50);                    //Wait 50 Milliseconds
    if(SensorValue[PSITest2] == 1) {  //If Piston pressed Button
      PSIReady = 1;                     //Air Is on
      SensorValue[PSITest] = 0;         //Turn Test Piston off
      } else if(PSIReady == 0) {        //Else if Air is off
      playTone(420, 62.5);              //Play Tone on Speaker for 420 Millisecc
      wait1Msec(1250);                  //wait 1250 Milliseconds
    }
  }
}

int EjectVar = 1;      //Defines EjectVar as a Global Variable
task EjectCount() {    //This task is written to wait 200 milliseconds before set
  wait1Msec(200);      //wait 200 Milliseconds
  EjectVar = 1;        //Set EjectVar to 1
}

int stallcheck = 0;  //Initalize stallcheck as a int and set it to 0
int triggered = 0;   //Initalize triggered as a int and set it to 0
void SmartIntake(){  //Smart, Automatic Ball Loading Function
  if(EjectVar == 1){  //if EjectVar equals 1
    if(SensorValue[IntakeT] == 1 && SensorValue[UptakeT] == 1) { //IF Intake anc
      SetMotor(Intake, 0); //Turn off Intake
      motor[Uptake] = 0;   //Turn off Uptake
      triggered = 0;       //Set triggered var to 0
    }
    if(SensorValue[UptakeT] == 0 && SensorValue[IntakeT] == 1) { //IF Uptake Ser
      motor[Uptake] = 80;  //Run Uptake at 80 motorpower
      triggered = 0;       //Set triggered var to 0
    }
    if(SensorValue[IntakeT] == 0 && SensorValue[UptakeT] == 0) { //IF Uptake anc
      if(triggered < 1) {  //If triggered var is less than 1
        triggered++;       //Add 1 to triggered var
      }
    }
    if(triggered == 1) {  //If triggered var equals 1
      motor[Uptake] = 0; //Turn off Uptake
      SetMotor(Intake, 127);  //Run Intake at max power
    }
    if(SensorValue[UptakeT] == 1) { //If Uptake Sensor equals 1
      triggered = 0;  //set triggered to 0
      motor[Uptake] = 0;  //turn off Uptake
    }
    if((SmartMotorGetCurrent(Intake) > 0.9) && SensorValue[IntakeT] == 0){ //If
      stallcheck = stallcheck + 1; //Add 1 to value of stallcheck var
      AdvancedCheck = 0; //Set AdvancedCheck var to 0
      } else { //else
      if(AdvancedCheck < 3){ //If AdvancedCheck is less than 3
        AdvancedCheck = AdvancedCheck + 1; //Add 1 to AdvancedCheck
        } else { //else
        stallcheck = 0; //Set stallcheck var to 0
      }
    }
    if(stallcheck >= 40){  //IF stallcheck is greater than or equal to 40
```

```
      if(SensorValue[UptakeT] == 1){ //If Uptake Sensor is 0
        SetMotor(Intake, 0);    //Turn Intake off
        } else {  //else
        SetMotor(Intake, -127); //Run Intake at Max Reverse Power
        motor[Uptake] = 0; //Turn Uptake off
        EjectVar = 0;      //Set EjectVar to 0
        startTask(EjectCount); //Start EjectCount Task
      }
      } else if(stallcheck >= 10 && SensorValue[UptakeT] == 0){ //if stallcheck
      motor[Uptake] = 80; //Run Uptake at 80 Power
    }
  }
}


int StartSmartIntake = 0;  //Initalize StartSmartIntake as int equal to 0
int OverrideInAuton = 0;   //Initalize OverrideInAuton as int equal to 0
task AutonDrive(){         //Task for Driving in Auton
  startTask(AccelControl); //Start AccelControl Task for Drive
  while(1==1){  //Run Forever
    if(OverrideInAuton == 0){  //If OverrideInAuton equals 0
      motor[RightFMotor] = OutputY - (OutputX/1.5);  //  | Accelerated Motor Cor
      motor[RightBMotor] = OutputY - (OutputX/1.5);  //  |
      motor[LeftFMotor] = OutputY + (OutputX/1.5);   //  |
      motor[LeftBMotor] = OutputY + (OutputX/1.5);   // \|/
    }
    if(StartSmartIntake == 1){  //IF StartSmartIntake equals 1
      SmartIntake();            //Run SmartIntake Function
    }
    wait1Msec(20);              //Wait 20 Milliseconds
  }
}

//Wait for Press
void waitForPress(){ //Function that waits until a LCD Button is Pressed
  while(nLCDButtons == 0){} //While all LCD Buttons equal 0
  wait1Msec(5); //wait 5 Milliseconds
}

//Wait for Release
void waitForRelease(){ //Function that waits until LCD Button is Released
  while(nLCDButtons != 0){} //While ALL LCD Buttons NOT equal to 0
  wait1Msec(5); //wait 5 Milliseconds
}

void pre_auton(){ //Starts Pre autonomous function
  startTask(BatteryPSICheck); //Starts Error handling Task
  startTask(LCDScreens); //Starts primary and secondary LCD Screens
  SensorValue[Trans] = 0; //4 motor launcher
  SensorValue[Brakes] = 0; //Disengage brakes
  PreAutonLCD = 2; //modify LCD output
  bStopTasksBetweenModes = false; //Set to false to keep Smart motor running
  SmartMotorsInit(); //Initilize smart motors
  SmartMotorSetSlewRate(Intake, 255); //set up control for intake
  SmartMotorRun(); //start smart motor
  bLCDBacklight = true; //Turn on Backlight of 1st Screen
  vexLcdBacklight(1); //Turn on Backlight of 2nd Screen
  ClearLCDs(); //clear the LCDs
  wait1Msec(625); //Wait for 625 Milliseconds
```

```
  if(BatteriesReady == 0){  //If all batteries are NOT plugged in
    while(BatteriesReady == 0){ //While Batteries are NOT plugged in
      if(bLCDBacklight == true){ //IF 1st Backlight is on
        vexLcdBacklight(0);  //Turn off Backlight of 1st Screen
        bLCDBacklight = false;//Turn off Backlight of 2nd Screen
        } else {  //else
        vexLcdBacklight(1);    //Turn on Backlight of 1st Screen
        bLCDBacklight = true;  //Turn on Backlight of 2nd Screen
      }
      displayLCDCenteredString(0, "Check Batteries");  //Display "Check Batterie
      vexLcdSet(0, "Check Batteries");                 //Display "Check Batterie
      if(BackupTone == 1){                   //If Backup Tone is 1
        displayLCDCenteredString(1, "9V Backup"); //Display "9V Backup" on LCD1
        vexLcdSet(1, "9V Backup");                //Display "9V Backup" on LCD2
      }
      if(ExpanderTone == 1){                      //If Expander Tone is 1
        displayLCDCenteredString(1, "Power Expander");//Display "Power Expander"
        vexLcdSet(1, "Power Expander");              //Display "Power Expander"
      }
      wait1Msec(100); //Wait 100 Milliseconds inbetween loops
    }
    ClearLCDs();             //Clear both LCD Screens
    vexLcdBacklight(1);      //Turn on Backlight of 1st Screen
    bLCDBacklight = true;    //Turn on Backlight of 2nd Screen
  }
  if(PSIReady == 0){         //IF PSIReady is 0
    while(PSIReady == 0){      //While PSIReady is 0
      if(bLCDBacklight == true){ //IF 1st Backlight is on
        vexLcdBacklight(0);  //Turn off Backlight of 1st Screen
        bLCDBacklight = false;//Turn off Backlight of 2nd Screen
        } else {  //else
        vexLcdBacklight(1);    //Turn on Backlight of 1st Screen
        bLCDBacklight = true;  //Turn on Backlight of 2nd Screen
      }
      displayLCDCenteredString(0, "Turn On Air!"); //Display "Turn on Air!" on I
      vexLcdSet(0, "Turn On Air!");                //Display "Turn on Air!" on I
      if(PSIReady == 0) {    //IF PSIReady is 0
        displayLCDCenteredString(1, "Check PSI");  //Display "CheckPSI" on LCD1
        vexLcdSet(1, "Check PSI");                 //Display "CheckPSI" on LCD2
      }
      wait1Msec(100); //wait for 100 Milliseconds
    }
    ClearLCDs();   //Clear all LCD Screens
    vexLcdBacklight(1);   //Turn on Backlight of 1st Screen
    bLCDBacklight = true; //Turn on Backlight of 2nd Screen
  }
  //Preload Loading Assist Code
  int isAble = 0;              //Define int isAble as 0
  while(isAble == 0) {         //While isAble is 0
    displayLCDCenteredString(0, "Load Balls");  //Display "Load Balls" on LCD1
    displayLCDString(1, 1, "Up");               //Display "Up" on LCD1
    displayLCDString(1, 12, "Down");            //Display "Down" on LCD1
    waitForPress();                  //Wait until a LCD Button is pressed
    if(nLCDButtons == rightButton) { //If Right LCD Button is pressed
      motor[Uptake] = -127;        //Set Uptake to max Speed Reverse
      SetMotor(Intake, -127);      //Set Intake to max Speed Reverse
      waitForRelease();            //Wait until LCD Button is Released
      motor[Uptake] = 0;           //Turn off Uptake Motor
      SetMotor(Intake, 0);         //Turn off Intake Motor
```

```
    }
    if(nLCDButtons == leftButton) { //If Right LCD Button is pressed
      motor[Uptake] = 127;         //Set Uptake to Max Speed
      SetMotor(Intake, 127);       //Set Intake to Max Speed
      waitForRelease();            //Wait until LCD Button is Released
      motor[Uptake] = 0;           //Turn off Uptake Motor
      SetMotor(Intake, 0);         //Turn off Intake Motor
    }
    if(nLCDButtons == centerButton) { //If Center LCD Button is pressed
      waitForRelease();            //Wait until LCD Button is Released
      isAble = 1;                  //Set isAble to 1, ending the loop
    }
  }
  ClearLCDs();           //Clear all LCD Screens
  PreAutonLCD = 1;       //Set PreAutonLCD to 1
  bool ReadyToGo = false;    //Declare Boolean Variable ReadyToGo as False
  while(ReadyToGo == false){  //While NOT ReadyToGo
    while(nLCDButtons != centerButton)  //While LCD Center Button not pressed
    {
      //Switch case that allows the user to choose between 4 different options
      switch(count){
      case 0:
        displayLCDCenteredString(0, "Longshot CS-6");       //Display first choi
        displayLCDCenteredString(1, "<     Enter    >");
        waitForPress();                    //wait until a LCD Button is pressed
        if(nLCDButtons == leftButton){     //IF Left LCD button pressed
          waitForRelease();                //wait until LCD button is released
          count = 3;                       //set count var to 3
          } else if(nLCDButtons == rightButton) { //else if Right LCD Button is
          waitForRelease();                //wait until LCD button is released
          count++;                         //Set count var to count + 1
        }
        break;
      case 1:
        displayLCDCenteredString(0, "Side Mid");           //Display second choi
        displayLCDCenteredString(1, "<     Enter    >");
        waitForPress();                    //wait until a LCD Button is pressed
        if(nLCDButtons == leftButton){     //IF Left LCD button pressed
          waitForRelease();                //wait until LCD button is released
          count--;                         //set count var to count - 1
          } else if(nLCDButtons == rightButton) { //else if Right LCD Button is
          waitForRelease();                //wait until LCD button is released
          count++;                         //Set count var to count + 1
        }
        break;
      case 2:
        displayLCDCenteredString(0, "Back Mid");        //Display third choice
        displayLCDCenteredString(1, "<     Enter    >");
        waitForPress();                    //wait until a LCD Button is pressed
        if(nLCDButtons == leftButton){     //IF Left LCD button pressed
          waitForRelease();                //wait until LCD button is released
          count--;                         //set count var to count - 1
          } else if(nLCDButtons == rightButton) { //else if Right LCD Button is
          waitForRelease();                //wait until LCD button is released
          count++;                         //Set count var to count + 1
        }
        break;
      case 3:
        displayLCDCenteredString(0, "Program Skills");  //Display 4th Choice
```

```
        displayLCDCenteredString(1, "<     Enter    >");
        waitForPress();
        //Increment or decrement "count" based on button press
        waitForPress();                       //wait until a LCD Button is pressed
        if(nLCDButtons == leftButton){        //IF Left LCD button pressed
          waitForRelease();                   //wait until LCD button is released
          count--;                            //set count var to count - 1
          } else if(nLCDButtons == rightButton) { //else if Right LCD Button is
          waitForRelease();                   //wait until LCD button is released
          count = 0;                          //Set count var to 0
        }
        break;
      default:
        count = 0;
        break;
      }
    }
    waitForRelease();
    if(count == 1 || count == 2) {
      while(nLCDButtons != centerButton)
      {
        //Switch case that allows the user to choose from 4 different options
        switch(color){
        case 0:
          displayLCDCenteredString(0, "Red Alliance");          //Display first
          displayLCDCenteredString(1, "<     Enter    >");
          waitForPress();                      //wait until a LCD Button is press
          if(nLCDButtons == leftButton){       //IF Left LCD button pressed
            waitForRelease();                  //wait until LCD button is release
            color++;                           //set color var to color + 1
            } else if(nLCDButtons == rightButton) { //else if Right LCD Button i
            waitForRelease();                  //wait until LCD button is release
            color++;                           //Set color var to color + 1
          }
          break;
        case 1:
          displayLCDCenteredString(0, "Blue Alliance");          //Display secor
          displayLCDCenteredString(1, "<     Enter    >");
          waitForPress();                      //wait until a LCD Button is press
          if(nLCDButtons == leftButton){       //IF Left LCD button pressed
            waitForRelease();                  //wait until LCD button is release
            color--;                           //set color var to color - 1
            } else if(nLCDButtons == rightButton) { //else if Right LCD Button i
            waitForRelease();                  //wait until LCD button is release
            color--;                           //Set color var to color - 1
          }
          break;
        default:
          color = 0;
          break;
        }
      }
    }
    waitForRelease();                     //Wait until LCD button is released
    if(count == 1 || count == 2) {        //If either driving Auton is selected
      while(nLCDButtons != centerButton)//While center LCD button is NOT pressed
      {
        //Switch case that allows the user to choose from 4 different options
        switch(style){
```

```
        case 0:
          displayLCDCenteredString(0, "Block Middle");          //Display first
          displayLCDCenteredString(1, "<     Enter    >");
          waitForPress();                       //wait until a LCD Button is press
          if(nLCDButtons == leftButton){        //IF Left LCD button pressed
            waitForRelease();                   //wait until LCD button is release
            style++;                            //set style var to style + 1
            } else if(nLCDButtons == rightButton) { //else if Right LCD Button i
            waitForRelease();                   //wait until LCD button is release
            style++;                            //set style var to style + 1
          }
          break;
        case 1:
          displayLCDCenteredString(0, "Get Stack");          //Display second ch
          displayLCDCenteredString(1, "<     Enter    >");
          waitForPress();                       //wait until a LCD Button is press
          if(nLCDButtons == leftButton){        //IF Left LCD button pressed
            waitForRelease();                   //wait until LCD button is release
            style--;                            //set style var to style - 1
            } else if(nLCDButtons == rightButton) { //else if Right LCD Button i
            waitForRelease();                   //wait until LCD button is release
            style--;                            //set style var to style - 1
          }
          break;
        default:
          style = 0;
          break;
        }
      }
    }
    if(count == 0 || count == 3) {     //If Programming Skills or Longshot Auton
      string strlcd = "";  //Define strlcd as blank string
      if(count == 0) {     //IF Longshot Auton is Selected
        countspeed = 39.0; //Default Longshot RPS
      }
      if(count == 3) {        //If Programming Skills Auton is Selected
        countspeed = 33.0; //Default Programming Skills RPS
      }
      while(nLCDButtons != centerButton)  //While Center LCD button is not press
      {
        sprintf(strlcd, "Power: %2.1f", countspeed); //set strlcd string to curr
        displayLCDCenteredString(0, strlcd);         //Display strlcd String on
        displayLCDCenteredString(1, "<     Enter    >");
        waitForPress();                      //wait until a LCD Button is pressed
        if(nLCDButtons == leftButton){       //IF Left LCD button pressed
          waitForRelease();                  //wait until LCD button is released
          countspeed = countspeed - 0.5;     //subtract 0.5 from countspeed var
          } else if(nLCDButtons == rightButton) { //else if Right LCD Button is
          waitForRelease();                  //wait until LCD button is released
          countspeed = countspeed + 0.5;     //add 0.5 to countspeed var
        }
      }
    }
    waitForRelease();       //wait until LCD button is released
    ClearLCDs();            //Clear All LCD Screens
    while(nLCDButtons != leftButton && nLCDButtons != rightButton) { //While Lef
      displayLCDCenteredString(0, "Ready To Go?");      //Display "Ready To Go?"
      displayLCDString(1, 0, "No");                     //Display "No" on LCD1
      displayLCDString(1, 13, "Yes");                   //Display "Yes" on LCD1
```

```
      waitForPress();                       //wait until a LCD Button is pressed
      if(nLCDButtons == leftButton){        //If Left LCD Button is pressed
        ReadyToGo = false;                  //set ReadyToGo to false
        } else if(nLCDButtons == rightButton) { //Else if Right LCD Button is pr
        ReadyToGo = true;                   //set ReadyToGo to true
      }
    }
    waitForRelease();         //wait until LCD Buttons are Released
  }
  stopTask(LCDScreens); //Stop LCD Screen Tasks
  ClearLCDs(); //Clear All LCD Screens
  displayLCDCenteredString(0, "Initializing");  //Display "Initializing" on LCD
  displayLCDCenteredString(1, "GyroScope");     //Display "GyroScope" on LCD
  SensorType[ in2 ] = sensorNone;  //Clear Out GyroScope
  wait1Msec(2000); //Allow Time for Calibration
  SensorType[ in2 ] = sensorGyro; //Initialize Gyro
  for(int k = 0; k < 1000; k++){  //For Loop that makes one fantastic start up S
    playTone(k, 1); //play frequency equal to k on Speaker for 1 Millisecond
    wait1Msec(1); //wait 1 Millisecond
    k = k + 1;    //set k to k + 1
  }
}

task LauncherStart(){                                          //Start Auton Launcher
  while(1 == 1) {                                              //Run Forever
    motorPower = PidFunction(LauncherSpeed, RPSEncoder);  //motorPower Var set t
    Launcher(abs(motorPower));                                 //Launcher set to motc
    wait1Msec(20);                                             //wait 20 Milliseconds
  }
}

bool shoot = false;
task LauncherStart2() { //start Launcher with 8 motors
  while(1==1) {  //Run Forever
    motorPower = PidFunction(LauncherSpeed, RPSEncoder); //Set motorPower to PII
    Launcher(abs(motorPower));          //Set Launcher to motorPower var
    motor[RightFMotor] = -abs(motorPower);//Set RightFMotor to Negative the absc
    motor[RightBMotor] = -abs(motorPower);//Set RightBMotor to Negative the absc
    motor[LeftFMotor] = -abs(motorPower);//Set LeftFMotor to Negative the absolu
    motor[LeftBMotor] = -abs(motorPower);//Set LeftBMotor to Negative the absolu
    if(shoot) {    //If shoot is true
      SetMotor(Intake, 80); //Set Intake to 80 Power
      motor[Uptake] = 70;   //Set Uptake to 70 Power
    }
    wait1Msec(20);        //wait 20 Milliseconds
  }
}

void MotorControl(int AutonSpeed) {  //Motor Controlling function for Autonomous
  motor[RightFMotor] =  AutonSpeed * color;  //Set RightFMotor to AutonSpeed * C
  motor[LeftFMotor] =  AutonSpeed  * color;  //Set LeftFMotor to AutonSpeed * Cc
}

task autonomous()
{
  stopTask(BatteryPSICheck); //Make sure BatteryPSICheck is not running
  ClearLCDs(); //Run LCD Clearing Function
  if(color == 1){ //IF color Equals 1
    color = 1;  //Sets color to Positive for Blue Side
```

```
    } else {
      color = -1; //Sets color to Negative for Red Side
    }
    startTask(LCDScreens);  //Start LCD Screen Task
    PreAutonOver = 1;       //Set PreAutonOver to 1
    PreAutonLCD = 0;        //Set PreAutonLCD to 0
    Auton = 1;              //Set Auton to 1
    startTask(readEncoder); //Start PID Task
    if(count == 0) { //IF Count Equals 0 run Back Shooting Auton
      color = 1; //Makes Sure Color Is Not Set to 0
      OverrideInAuton = 1; //Overrides AutonAccelControl Task
      countspeed = 37.5;   //Sets the Desired RPS for Auton
      if(SensorValue[Trans] != 1) { //Checks to make sure Transmission is in Launc
        SensorValue[Trans] = 1;     //Shifts DriveMotors to Launcher
      }
      startTask(shiftplus);         //Starts Smooth Shifting Task
      startTask(LauncherStart2);    //Starts 8 Motor Launcher PID
      LauncherSpeed = 0;            //Sets Launcher Speed to 0
      wait1Msec(250);               //Wait 250 Seconds to Initialize
      while(LauncherSpeed < countspeed) {  //While LauncherSpeed is less than cour
        LauncherSpeed += 1.0;  //Add 1 to LauncherSpeed
        wait1Msec(20); //Wait 20 Milliseconds
      }
      LauncherSpeed = countspeed;  //Set LauncherSpeed to countspeed
      while(RPSEncoder < LauncherSpeed) { //Wait Until the Launcher is up to speed
        wait1Msec(1); //Wait 1 Milliseconds
      }
      wait1Msec(50); //Wait 50 Milliseconds
      SetMotor(Intake, 80); //Run Intake to 80
      motor[Uptake] = 70; //Set Uptake to 70
      shoot = true;  //set shoot to true
      turbo = true;  //set turbo to true
      wait1Msec(6000); //wait 6000 Milliseconds
      stopTask(LCDScreens); //Stop LCDScreens Task
      ClearLCDs();  //Clear All LCDs
      while(LauncherSpeed > 10){  //While LauncherSpeed greater than 10
        LauncherSpeed = LauncherSpeed - 2.0;  //Subtract 2 from LauncherSpeed
        wait1Msec(10); //wait 10 Milliseconds
      }
      LauncherSpeed = 0.0;  //Set Launcher to 0
      shoot = false;  //set shoot to false
      wait1Msec(100000);  //wait for Auton to end
    }
    ////////////////////////////////////////////////////////////////////////////
    if(count == 1){ //Autonomous for the squares closer to the opposite goals
      SmartMotorRun(); //Start running Smart Motor
      startTask(AutonDrive); //Start Task for Autonomous Driving
      startTask(LauncherStart); //Start Launcher control for 4 motors
      LauncherSpeed = 23.5; //Set the launcher speed
      SensorValue[LeftDrive] = 0; //Reset the Left Drive encoder
      SensorValue[RightDrive] = 0; //Reset the Right Drive encoder
      while(abs(SensorValue[LeftDrive]) < 1000){ //While the left side has not tra
        Y1 = 127; //Drive forward
      } //When the robot has driven far enough
      Y1 = 0; //Stop driving
      while(abs(SensorValue[LeftDrive]) < 1400){ //Wait for deacceleration
        wait1Msec(1); //Wait 1 millisecond
      }
      OutputY = 0; //Force stop the motors
```

```
    SensorValue[Gyro] = 0; //Reset the Gyro
    while(abs(SensorValue[Gyro]) < 70){ //Turn 7 degrees dependent on color
      OverrideInAuton = 1; //Override Auton Drive
      if(color == -1){ //Red
        motor[RightFMotor] = 50; //Drive with the right side only
        motor[RightBMotor] = 50;
        X1 = 30 * color; //Utilize Auton Drive for Deacceleration
        } else { //Blue
        motor[LeftFMotor] = 50; //Drive with the left side only
        motor[LeftBMotor] = 50;
        X1 = 30 * color; //Utilize Auton Drive for Deacceleration
      }
    }
    MotorControl(0); //Stop the Drive Motors
    X1 = 0; //Stop the Drive
    OutputX = 0; //Stop the Drive
    OverrideInAuton = 0; //Return control to Auton Drive
    wait1Msec(100); //Wait 100 Milliseconds
    int FireOneTime = 0; //Fireing Control
    while(FireOneTime < 800){ //While we have not shot all of the balls
      motor[Uptake] = 127; //Run Uptake
      motor[Intake] = 127; //Run Intake
      turbo = true; //Enable Turbo
      wait1Msec(1); //Wait 1 Millisecond
      FireOneTime = FireOneTime + 1; //Increase FireOneTime Variable by one
    }
    turbo = false; //Stop the use of Turbo
    if(style == 0){ //If we want to Block
      SensorValue[Trans] = 1; //Shift Out to Protect Drive Motors
      for(int w = 25; w > 1; w--){ //While the launcher speed is Greater than 1
        LauncherSpeed = w; //Set Launcherspeed to the decreasing Variable
        wait1Msec(50); //Wait to not kill the launcher
      }
      LauncherSpeed = 0; //Turn Launcher Off
      wait1Msec(20000); //Wait Until End of Auton
      } else {
      Y1 = -100;  //Set Y1 to -100
      wait1Msec(950); //wait 950 Milliseconds
      Y1 = 0;  //Set Y1 to 0
      SensorValue[Gyro] = 0;  //Reset Gyro to 0
      while(abs(SensorValue[Gyro]) < 300){  //While Gyro less than 30 Degrees
        OverrideInAuton = 1;     //Override AutonDrive Task
        motor[RightFMotor] = -60 * color; //Turn at 60 Power
        motor[LeftFMotor] = 60 * color;   //Turn at 60 Power
      }
      motor[Intake] = 127; //Run Intake at Max Speed
      OverrideInAuton = 0; //Give Control back to AutonDrive Task
      wait1Msec(250);  //Wait 250 Milliseconds
      Y1 = -50; //Set Y1 to -50
      SensorValue[LeftDrive] = 0;  //Set Left Quad to 0
      waitUntil(SmartMotorGetCurrent(Intake) > 0.9);  //Wait until the current i
      OutputY = 0;  //Set OutputY to 0
      Y1 = 0;       //Set Y1 to 0
      motor[Intake] = 127;  //Set Intake to Max Speed
      motor[Uptake] = 127;  //Set Uptake to Max Speed
      waitUntil(SensorValue[UptakeT] == 1);  //Wait until UptakeT is pressed
      motor[Uptake] = -127; //Run Uptake MAX Speed in Reverse
      Y1 = 80;      //Set Y1 to 80;
      wait1Msec(250);  //Wait 250 Milliseconds
```

```
      motor[Uptake] = 0;  //Turn off Uptake
      Y1 = -80;    //Set Y1 to -80
      waitUntil(OutputY < 0);  //wait Until OutputY is less than 0
      motor[Uptake] = 0; //Turn off Uptake
      waitUntil(SmartMotorGetCurrent(Intake) > 0.9);  //Wait until the current i
      OutputY = 0;  //Set OutputY to 0
      StartSmartIntake = 1;  //Turn on SmartIntake
      wait1Msec(2000);   //Wait 2000 Milliseconds
      Y1 = 80;           //Set Y1 to 80
      if(SensorValue[LeftDrive] > 0){   //IF LeftQuad greater than 0
        waitUntil(SensorValue[LeftDrive] < 0);  //wait until its less than
        } else {  //else
        waitUntil(SensorValue[LeftDrive] > 0);  //wait until its greater than
      }
      OutputY = 0;  //set OutputY to 0
      Y1 = 0;       //set Y1 to 0
      SensorValue[Gyro] = 0;    //Clear Gyro Sensor
      while(abs(SensorValue[Gyro]) < 300){  //While Gyro is less than 30 degrees
        OverrideInAuton = 1;  //Override AutonDrive Task
        motor[RightFMotor] = 60 * color;  //Turn Depending on alliance color
        motor[RightBMotor] = 60 * color;  //      |
        motor[LeftFMotor] = -60 * color;  //      |
        motor[LeftBMotor] = -60 * color;  //      |
        X1 = 30 * color;                  //     \|/
      }
      wait1Msec(1000000); //Wait until Autonomous mode is disabled
    }
    ////////////////////////////////////////////////////////////////////////////
    if(count == 2){ //Autonomous for the squares farther away from the opposite
      SmartMotorRun(); //Start Smart Motor
      startTask(AutonDrive); //Start Autonomous Driving Control
      startTask(LauncherStart); //Start Launcher Control
      LauncherSpeed = 31.0; //Set Launcher speed to 31.0 RPS
      SensorValue[LeftDrive] = 0; //Clear Left Drive Sensor
      SensorValue[RightDrive] = 0; //Clear Right Drive Sensor
      while(abs(SensorValue[LeftDrive]) < 1000){ //While the robot has driven le
        Y1 = 127; //Set desired speed to 127
      }
      Y1 = 0; //Set desired speed to 0
      while(abs(SensorValue[LeftDrive]) < 1400){ //Wait for the robot to coast t
        wait1Msec(1); //Wait 1 millisecond
      }
      OutputY = 0; //Stop the robot
      SensorValue[Gyro] = 0; //Clear the Gyroscope
      while(abs(SensorValue[Gyro]) < 70){ //Turn 7 Degrees depeding on starting
        OverrideInAuton = 1; //Override Autonomous Driving
        if(color == -1){ //Red
          motor[RightFMotor] = 50; //Set the Right side to 50
          motor[RightBMotor] = 50;
          X1 = 30 * color; //Set X1 for Auton Drive
          } else { //Blue
          motor[LeftFMotor] = 50; //Set the Left side to 50
          motor[LeftBMotor] = 50;
          X1 = 30 * color; //Set X1 for Auton Drive
        }
      }
      MotorControl(0); //Stop Motor
      X1 = 0; //Zero X1
      OutputX = 0; //Zero OutputX
```

```
      OverrideInAuton = 0; //Return control to Auton Drive
      wait1Msec(100); //Wait 100 Milliseconds
      int FireOneTime = 0; //Create FireOneTime
      while(FireOneTime < 800){ //Wait for FireOneTime to reach 800
        motor[Uptake] = 127; //Set Uptake to 127
        motor[Intake] = 127; //Set Intake to 127
        turbo = true; //Enable Turbo
        wait1Msec(1); //Wait 1 Millisecond
        FireOneTime = FireOneTime + 1; //Increase FireOnTimw by one
      }
      SensorValue[LeftDrive] = 0; //Zero Left Drive Sensor
      turbo = false; //Disable Turbo
      if(style == 0){ //If we want to block
        SensorValue[Trans] = 1; //Shift out
        wait1Msec(20000); //Wait 20000 Milliseconds
        } else {
        Y1 = -100; //Set Y1 to -100
        motor[Intake] = 127; //Turn on Intake
        motor[Uptake] = 127; //Turn on Uptake
        waitUntil(SensorValue[UptakeT] == 1); //Wait for a ball at the top of Up
        Y1 = 0; //Stop driving
        OutputY = 0; //Stop Driving
        motor[Uptake] = -127; //Reverse Uptake
        wait1Msec(750); //Wait 750 Milliseconds
        motor[Uptake] = 127; //Run Uptake Forward
        waitUntil(SensorValue[UptakeT] == 1); //Wait for a Ball at the top of th
        motor[Uptake] = 0; //Stop Uptake
        Y1 = 100; //Start Driving
        while(abs(SensorValue[LeftDrive]) > 300){ //Drive forawrd
          wait1Msec(1); //Wait 1 Millisecond
        }
        Y1 = 0; //Stop Driving
        while(abs(SensorValue[LeftDrive]) > 0){ //Wait for the robot to stop
          wait1Msec(1); //Wait 1 Millisecond
        }
        OutputY = 0; //Stop Driving
        FireOneTime = 0; //Clear Fire Variable
        while(FireOneTime < 800){ //Wait for the Robot to Fire
          motor[Uptake] = 127; //Start Uptake
          motor[Intake] = 127; //Start Intake
          turbo = true; //Enable Turbo
          wait1Msec(1); //Wait 1 Millisecond
          FireOneTime = FireOneTime + 1; //Increase FireOneTime by one
        }
        wait1Msec(20000); //Wait 20 Seconds for Disable
      }
    }
  }
  ////////////////////////////////////////////////////////////////////////////////

  if(count == 3){ //Programming Skills
    stopTask(LCDScreens);
    displayLCDCenteredString(1, "he ever loved.");
    displayLCDCenteredString(0, "The only woman");
    vexLcdSetAt(0, 3, "His Horse");
    vexLcdSetAt(1, 6, "Tyrone.");
    while(1==1){
      playTone(random[7000] , 5);
      wait1Msec(5);
```

```
    }
    waitUntil(1==0); //WaitForever
  }
  ////////////////////////////////////////////////////////////////////////////
  waitUntil(1==0);
}

task usercontrol() {
  turbo = false;  //Sets PID Turbo to False
  stopTask(BatteryPSICheck);//Stops BatteryPSICheck Task
  stopTask(AutonDrive);     //Stops AutonDrive Task
  stopTask(readEncoder);    //Stops readEncoder Task
  stopTask(autonomous);     //Stops autonomous Task
  stopTask(LauncherStart2); //Stops LauncherStart2 Task
  stopTask(LauncherStart);  //Stops LauncherStart Task
  startTask(AccelControl);  //Starts AccelControl Task
  startTask(readEncoder);   //Starts readEncoder Task (Launcher PID)
  startTask(LCDScreens);    //Starts LCDSCreens Task
  PreAutonLCD = 0;
  PreAutonOver = 1;
  Auton = 0;
  pidRequestedValue = 0;
  SensorValue[Trans] = 0; //Starts Transmisson In Field Mode
  byte toggleup = 0;
  byte toggledown = 0;
  string mainBattery;
  byte launcherDirection = 0;
  EjectVar = 1;
  if(count == 3){
    skills = true;
    SensorValue[Trans] = 1;
    startTask(shiftplus);
    pidRequestedValue = 30.0;
    } else {
    skills = false;
  }
  if(SensorValue[potentSpeed] <= 1365) {
    launcherDirection = 0;
  }
  if(SensorValue[potentSpeed] > 1365 && SensorValue[potentSpeed] < 2730) {
    launcherDirection = 1;
  }
  if(SensorValue[potentSpeed] >= 2730) {
    launcherDirection = 3;
  }
  byte launcherToggle = 0; //Set LauncherToggle to 0
  while(true){
    //drive code: tank controls
    if(abs(vexRT[Ch1]) > 25){ //if joystick not in Deadzone
      X1 = vexRT[Ch1]; //set to joystick
      } else {
      X1 = 0; //set to 0 if in deadzone
    }
    if(abs(vexRT[Ch3]) > 25){ //if joystick not in Deadzone
      Y1 = vexRT[Ch3]; //set to joystick
      } else {
      Y1 = 0; //set to 0 if in deadzone
    }
    if(abs(vexRT[Ch1Xmtr2]) > 25) { //If joystick not in Deadzone
```

```
      SecondOutputX = vexRT[Ch1Xmtr2]; //set to joystick
      } else {
      SecondOutputX = 0; //Set to 0 if in deadzone
    }
    if(vexRT[Btn8R] == 1) { //If Btn8R pressed
      drive_speed = 1.0; //Set Drive Speed Var to Max
    }
    if(vexRT[Btn8D] == 1) { //If Btn8D pressed
      drive speed = 0.5; //Set Drive Speed Var to half
    }
    if(SensorValue[Trans] == 0) {   //IF Transmission shifted into drive motors
      motor[RightFMotor] = (OutputY * drive_speed) - (OutputX) - SecondOutputX;
      motor[RightBMotor] = (OutputY * drive_speed) - (OutputX/1.5) - SecondOutpu
      motor[LeftFMotor] = (OutputY * drive_speed) + (OutputX) + SecondOutputX;
      motor[LeftBMotor] = (OutputY * drive_speed) + (OutputX/1.5) + SecondOutput
    }
    if(vexRT[Btn6UXmtr2] == 1 && toggleup == 0) {     //If 2nd Controller Btn 6U
      if(toggleup == 0) {                               //If toggleup equals 0
        if(pidRequestedValue == 0) {                      //If PIDRequested  equ
          pidRequestedValue = 15;                           //set PIDRequested t
        } else {                                          //else
          if(pidRequestedValue < 50 && pidRequestedValue >= 15) {  //If PIDReque
            pidRequestedValue += 0.5;                         //PidRequested equ
          }
        }
      }
      toggleup = 1;                                     //toggleup equals 1.  Th
    }                                                   //increasing excessively
    if(vexRT[Btn6UXmtr2] == 0) {      //If 2nd Controller Btn 6U equals 0
      toggleup = 0;                     //toggleup equals 0.  This waits for the
    }                                   //running the IF Statement above again.
    if(vexRT[Btn6DXmtr2] == 1 && toggledown == 0) { //If Btn 6D on the Partner F
      if(toggledown == 0) { //If it has not already been pressed
        if(pidRequestedValue == 15) { //if the value is 15 RPS
          pidRequestedValue = 0; //Set pidRequestedValue to Zero
        } else {
          if(pidRequestedValue <= 50 && pidRequestedValue > 15) { //If it is bet
            pidRequestedValue -= 0.5; //Subtract half an RPS
          }
        }
      }
      toggledown = 1; //Signify that the Button has been pressed
    }
    if(vexRT[Btn6DXmtr2] == 0) { //When the button is released
      toggledown = 0; //Signify that the button has been released
    }
    //18Inch Range
    if(vexRT[Btn8RXmtr2] == 1 || vexRT[Btn7D] == 1) { //If BTN 7U or 2nd BTN 8R
      pidRequestedValue = 25.5;  //Sets pidRequestedValue to Shoot 18 Inches frc
    }
    //Mid Range
    if(vexRT[Btn8UXmtr2] == 1 || vexRT[Btn7L] == 1) { //If BTN 7L or 2nd BTN 8U
      pidRequestedValue = 29.5;  //Sets pidRequestedValue to Shoot from MidFielc
    }
    //Long Range
    if(vexRT[Btn8LXmtr2] == 1 || vexRT[Btn7U] == 1) { //If BTN 7U or 2nd BTN 8L
      pidRequestedValue = 37.0;  //Sets pidRequestedValue to Shoot from Startinc
    }
    //BarShot
```

```
    if(vexRT[Btn7RXmtr2] == 1) {  //If 2nd BTN 7R
      pidRequestedValue = 28.0;  //Sets pidRequestedValue to Shoot from the Bar
    }
    if(vexRT[Btn7R] == 1 || vexRT[Btn8DXmtr2] == 1) { //Quick button to start ar
      if(launcherToggle == 0) { //If the button has not been pushed
        if(pidRequestedValue == 0) { //If the current launcher speed is 0
          launcherDirection = 1; //Set Variable to Increase the Launcher speed
        }
        if(pidRequestedValue > 0) { //If the launcher is spinning
          launcherDirection = 2; //Set Variable to Decrease the Launcher Speed
        }
      }
      launcherToggle = 1; //Signify that the button was pushed
    }
    if(vexRT[Btn7R] == 0 && vexRT[Btn8DXmtr2] == 0) { //When the button was rele
      launcherToggle = 0; //Signify that the button was released
    }
    if((pidRequestedValue <= 0 && launcherDirection == 2) || (pidRequestedValue
      //Multiple test statments to figure out when to stop accelerating
      launcherDirection = 0;
      if(pidRequestedValue < 0) { //If pidRequestedValue is less than zero
        pidRequestedValue = 0; //Set it to zero
      }
      if(pidRequestedValue > 24 && pidRequestedValue < 27) { //If within thresho
        pidRequestedValue = 25.5; //Set it to 25.5
      }
    }
    if(launcherDirection == 1 || launcherDirection == 3) {
      pidRequestedValue += 0.2;  //Increase PIDRequested by .2
    }
    if(launcherDirection == 2) {
      pidRequestedValue -= 0.2;  //Decrease PIDRequested by .2
    }
    if(vexRT[Btn5D] == 1 || vexRT[Btn5U] == 1 || vexRT[Btn6D] == 1 || vexRT[Btn6
      //OVERRIDES SmartIntake Function With Manual Control
      if(vexRT[Btn6U] == 1) {       //IF Btn 6U is Pressed
        if(SensorValue[UptakeT] == 0) { //IF UptakeT is NOT Pressed
          SetMotor(Intake, 0);          //Sets Intake Motor to 0
          } else {                        //else
          SetMotor(Intake, 127);        //Sets Intake Motor to MAX Power
        }
        } else {                    //else
        if(vexRT[Btn6D] == 1) {     //IF Btn6D is Pressed
          SetMotor(Intake, -127);   //Sets Intake Motor to Reverse MAX Power
          } else {                    //else
          SetMotor(Intake, 127);    //Sets Intake Motor to MAX Power
        }
      }
      if(vexRT[Btn5U] == 1 || vexRT[Btn7LXmtr2] == 1 || (vexRT[Btn7DXmtr2] == 1
        if(SensorValue[Trans] == 0) {     //IF Trans NOT Shifted
          motor[Uptake] = 127;            //Set Uptake to Max Power
          } else {                          //else
          motor[Uptake] = shiftedUptake;  //Set Uptake to shiftedUptake Speed
        }
        } else {                        //else
        if(vexRT[Btn5D] == 1) {   //IF Btn5D is pressed
          motor[Uptake] = -127;   //Set Uptake to MAX Reverse Power
          } else {                  //else
          motor[Uptake] = 0;      //Turn Off Uptake
```

```
        }
      }
      } else {   //else (If No Override button is pressed)
      SmartIntake();  // Run Function "SmartIntake"
    }
    if(((vexRT[Btn5DXmtr2] == 1) || (vexRT[Btn5UXmtr2] == 1)) && ToggleTrans ==
      if(ToggleTrans == 0) { //If the button has not been pushed
        startTask(shiftplus); //start smooth shift
        if(SensorValue[Trans] == 1) { //If trans is 1
          SensorValue[Trans] = 0; //Change to 0
          } else {
          if(SensorValue[Trans] == 0) { //If trans is 0
            SensorValue[Trans] = 1; //Changes it to 1
          }
        }
      }
      ToggleTrans = 1; //Signify that the button has been pushed
    }
    if((vexRT[Btn5DXmtr2] == 0) && (vexRT[Btn5UXmtr2] == 0)) { //If both are rel
      ToggleTrans = 0; //Signify that the button has been released
    }
    motorPower = PidFunction(pidRequestedValue ,RPSEncoder); //Use PID
    Launcher(abs(motorPower)); //Send values to Launcher
    if(SensorValue[Trans] == 1) { //If we are shifted
      motor[RightFMotor] = -abs(motorPower); //Send motorPower to all drive motc
      motor[RightBMotor] = -abs(motorPower);
      motor[LeftFMotor] = -abs(motorPower);
      motor[LeftBMotor] = -abs(motorPower);
    }
    wait1Msec(15); //Wait 15 Milliseconds before Rerunning the loop
  }
  playTone(100 ,100);
  wait1Msec(100000);
  UserControlCodePlaceholderForTesting();
  AutonomousCodePlaceholderForTesting();
  CompilerShortener();
}
```