```
#pragma config(Sensor, port2,  ArmBottomBumper, sensorVexIQ_Touch)
#pragma config(Sensor, port3,  Main_Gyro,       sensorVexIQ_Gyro)
#pragma config(Sensor, port4,  LED,             sensorVexIQ_LED)
#pragma config(Sensor, port8,  BallColor,       sensorVexIQ_ColorHue)
#pragma config(Sensor, port12, AutonStart,      sensorVexIQ_Touch)
#pragma config(Motor,  motor1,          Left,          tmotorVexIQ, PIDControl,
#pragma config(Motor,  motor5,          Intake,        tmotorVexIQ, PIDControl,
#pragma config(Motor,  motor6,          Right,         tmotorVexIQ, PIDControl,
#pragma config(Motor,  motor9,          CubeClaw,      tmotorVexIQ, PIDControl,
#pragma config(Motor,  motor10,         ArmRight,      tmotorVexIQ, PIDControl,
#pragma config(Motor,  motor11,         ArmLeft,       tmotorVexIQ, PIDControl,
//*!!Code automatically generated by 'ROBOTC' configuration wizard

//#pragma config(Sensor, port8,  CrossColor,     sensorVexIQ_ColorGrayscale)
//#pragma config(Sensor, port12, BallDetect,     sensorVexIQ_Touch)
//*!!Code automatically generated by 'ROBOTC' configuration wizard on 13/07/2019

//VIQC_SquaredAway_CompCode
/*************************************PROGRAMMER NOTES*************************
//
/*****************************************************************************

//////////////////////////////////////////////////////////////////////////////
//                                                  Variables
//////////////////////////////////////////////////////////////////////////////
bool ProgramPersmissionToStart = true;
bool MaxClawBrake = false;
bool IndexArmPressed; // defines the variable that check whether the controller
bool intakeStarted; // defines the variable that waits until the intake button i
bool AutonSwitchActive;
bool AutonSwitched = false;
bool AutonDisplayInt = false;
bool AutonPermissionToStart = false;
bool AutonLocked = false;
bool AutonFinished = false;
bool PlaceSequenceFinished = false;

#define   DATALOG_SERIES_0    0
#define   DATALOG_SERIES_1    1
#define   DATALOG_SERIES_2    2
#define   DATALOG_SERIES_3    3
#define   DATALOG_SERIES_4    4
#define   DATALOG_SERIES_5    5
#define   DATALOG_SERIES_6    6
#define   DATALOG_SERIES_7    7
#define   DATALOG_SERIES_8    8
#define   diameter 63.661977236758134307553505349006
#define   DriveWidth 19.5 //cm  B (base line distance)
#define   ticksPerRev 960
#define   IntakeSpeed 100 //72.5
#define   Height0 0 //Floor                                          \\ \.
#define   Height1 -300 //Travel Height                                 \\ \.
#define   Height2 -500 //Low Platform Place                                 > Arm H
#define   Height3 -700 //Low Platform Position / Top Platform Place // /.
#define   Height4 -1000 //Top Platfrom Position                     // /.
//#define  Height5 -1000; //
#define   FWD_DIR 0
```

```
#define   RIGHT_DIR 90
#define   BACK_DIR 180
#define   LEFT_DIR 270
#define   INCHINMM 25.40
#define   FOORINMM 304.8
#define   FIELDX 2438.4
#define   FIELDY 1219.2
#define   RobotCenterOffset 101.6
#define   LowerBracketHSV 14
#define   UpperBracketHSV 40
#define   MaxAutons 4

struct ODOMCOORDINATE {
  int pX;
  int pY;
  int pA;
};
struct ODOMCOORDINATE LStart;
struct ODOMCOORDINATE RStart;
struct ODOMCOORDINATE point1;
struct ODOMCOORDINATE point2;
struct ODOMCOORDINATE point3;
struct ODOMCOORDINATE point4;
struct ODOMCOORDINATE point5;
struct ODOMCOORDINATE point6;

long OdometryAngle;
long gyroValue;
long gyroError;

int AutonSelectLastState;
int AutonProgramSelector;
int PickupBonusSequenceState; // defines the variable that is used to tell what
int PlaceBonusSequenceState; // defines the variable that is used to tell what s
int   global_1 = 0;
int   global_2 = 0;
int   global_3 = 0;
int   global_4 = 0;
int   global_5 = 0;
int   global_6 = 0;
int   global_7 = 0;
int   global_8 = 0;
int ArmPresetValue = 0; //The preset number that tells the preset code how high
//int SpeedLeft = 0;
//int SpeedRight = 0;
//int Heading = 0;
//int HeadingStraight;

float x;
float y;
float LeftDriveTraveled;
float RightDriveTraveled;

/////////////////////////////////////////////////////////////////////////////////
//                                                 Bool Functions
/////////////////////////////////////////////////////////////////////////////////
```

```
bool BatteryWarning(int MinimumVoltage = 6500, int WarningVoltage = 7000, int Sa
  if (nImmediateBatteryLevel<MinimumVoltage) {
    delay(WarningDelay);
    playRepetitiveSound(soundCarAlarm4, 100);
    setTouchLEDColor(LED, colorRed);
    return true;
    if (nImmediateBatteryLevel<5500) {
      ProgramPersmissionToStart = false;
      stopAllTasks();
      stopAllMotors();
    };
    } else if (nImmediateBatteryLevel<WarningVoltage) {
    delay(WarningDelay);
    playSound(soundCarAlarm2);
    setTouchLEDColor(LED, colorOrange);
    return false;
    }else if (nImmediateBatteryLevel>SafeVoltage) {
    delay(WarningDelay);
    playRepetitiveSound(soundWrongWay, 100);
    setTouchLEDColor(LED, colorRed);
    return false;
    } else {
    return false;
  };
};

bool Turn(int goHere, float speed){
  return false;
  if(goHere < 0){
    goHere += 360;
  }
  int currentLoc = getGyroHeading(Main_Gyro);
  int toGo = currentLoc - goHere;
  if((toGo <= 180 && toGo >=0) || (toGo < 0 && toGo <= -180)){
    repeatUntil(getGyroHeading(Main Gyro) == goHere || getGyroHeading(Main Gyro)
      setMotor(Left, speed);
      setMotor(Right, -speed);
      return false;
    }
  }
  else if((toGo > 180 && toGo >= 0) || (toGo < 0 && toGo > -180)){
    repeatUntil(getGyroHeading(Main_Gyro) == goHere || getGyroHeading(Main_Gyro)
      setMotor(Left, -speed);
      setMotor(Right, speed);
      return false;
    }
  }
  stopMultipleMotors(Left, Right);
  return true;
}

bool TurnDegrees (float varTurnDegrees) { // turn PID function that returns true
  static bool InProgressTask; // defines the static bool that keeps the functior
  if (!InProgressTask) { // starts code while it hasn't completed the turn
    resetGyro(Main_Gyro); // resets the gyro
    if (varTurnDegrees>0) { // checks whether the setpoint (where we want to be)
      setMotorSpeed(Left, 50); // turns Right
```

```
      setMotorSpeed(Right, -50);
      } else {
      setMotorSpeed(Left, -50); // turns Left
      setMotorSpeed(Right, 50);
    };
  };
  InProgressTask = true; // stops the above code
  if ((getGyroDegrees(Main_Gyro)>varTurnDegrees && varTurnDegrees>0) || (getGyro
    InProgressTask = false; // starts above code again if it has overshot
    setMotorSpeed(Left, 0); // stops both motors
    setMotorSpeed(Right, 0);
    return true; // returns true if it has finished so that the sequence can mov

  };
  return false; // returns false if it has not finished so that the sequence wai
};

void driveDistance(float distance) { // function that converts mm into rotations
  moveMotorTarget(Left, (distance/200*360*2.66666666666666666666666666666666666
  moveMotorTarget(Right, (distance/200*360*2.66666666666666666666666666666666666
};

void moveMotorTargetMM (float WheelMotor, float distance2, int speed2 = 80) {
  moveMotorTarget(WheelMotor,(distance2/200*360*2.66666666666666666666666666666
}
////////////////////////////////////////////////////////////////////////////////
//                                                    General Functions
////////////////////////////////////////////////////////////////////////////////

void SwapDrive () {
  driveDistance(100);
};

void ArmHeightMove() { //moves the arm to the defined positions (height 0, Heigh
  switch(ArmPresetValue) {  //moves the arm by reading the variable (ArmPresetVa
  case 0://Floor
    setMotorTarget(ArmLeft, Height0, 100);
    setMotorTarget(ArmRight, Height0, 100);
    setTouchLEDColor(LED,colorOrange);
    break;

  case 1://
    setMotorTarget(ArmLeft, Height1, 100);
    setMotorTarget(ArmRight, Height1, 100);
    setTouchLEDColor(LED,colorOrange);
    break;

  case 2://
    setMotorTarget(ArmLeft, Height2, 100);
    setMotorTarget(ArmRight, Height2, 100);
    setTouchLEDColor(LED,colorOrange);
    break;

  case 3://Static Height / Moving height for balance
    setMotorTarget(ArmLeft, Height3, 100);
    setMotorTarget(ArmRight, Height3, 100);
    setTouchLEDColor(LED,colorOrange);
```

```
    break;

  case 4://
    setMotorTarget(ArmLeft, Height4, 100);
    setMotorTarget(ArmRight, Height4, 100);
    setTouchLEDColor(LED,colorOrange);
    break;

  };
};

void displayControl (int delayforscroll = 2000) {
  if (!AutonSwitched) {
    //Display Code
    displayTextLine(0, "Arm Height = %d", ((getMotorEncoder(ArmLeft)) + (getMotc
    displayVariableValues(1,ArmPresetValue); // displays the preset value for th
    displayTextLine(2, "Gyro = %d", getGyroDegrees(Main_Gyro)); // displays the
    displayVariableValues(4, PickupBonusSequenceState);
    displayVariableValues(5, PlaceBonusSequenceState);
    displayTextLine(3,"Battery (MV) = %d", nImmediateBatteryLevel);
    delay(delayforscroll);
    displayTextLine(0, "Arm Height = %d", ((getMotorEncoder(ArmLeft)) + (getMotc
    displayVariableValues(line1,x);
    displayVariableValues(line2,y);
    displayVariableValues(line3,LeftDriveTraveled);
    displayVariableValues(line4,RightDriveTraveled);
    displayVariableValues(line5, getGyroDegreesFloat(Main_Gyro));
    delay(delayforscroll);
    displayTextLine(0, "Arm Height = %d", ((getMotorEncoder(ArmLeft)) + (getMotc
    displayVariableValues(line1, DriveWidth);
    displayVariableValues(line2, diameter);
    displayVariableValues(line3, OdometryAngle);
    displayVariableValues(line4, getGyroRate(Main_Gyro));
    displayVariableValues(line5, getGyroDegreesFloat(Main Gyro));
    delay(delayforscroll);
  };
}



void GrabCube () {
  setMotorSpeed(CubeClaw, -100);
  delay(500);
};

void ReleaseCube() {
  setMotorSpeed(CubeClaw, 100);
  delay(500);
};

void RobotReset (int resetdelay = 100) {
  playSound(soundHeadlightsOn);
  delay(resetdelay);
  ReleaseCube();
  ArmPresetValue = 0;
  ArmHeightMove();
  setMotorTarget(Intake, 0, 75);
```

```
};

void GyroCustomCalibration(int count = 30) {
  startGyroCalibration( Main_Gyro, gyroCalibrateSamples512 );
  // delay so calibrate flag can be set internally to the gyro
  wait1Msec(100);

  // wait for calibration to finish or 2 seconds, whichever is longer
  while( getGyroCalibrationFlag(Main Gyro) || (count-- > 0) ) {
    wait1Msec(100);
  } resetGyro(Main_Gyro);
};

//////////////////////////////////////////////////////
//                Pickup cube Sequence              //
//////////////////////////////////////////////////////

void PickupBonusSequence () {
  static int LastState;
  bool P1;

  if (PickupBonusSequenceState !=LastState) {
    P1 = true;
    resetTimer(T1);
    LastState = PickupBonusSequenceState;
  }
  else {
    P1 = false;
  };
  switch (PickupBonusSequenceState){
  case 1:
    if (getJoystickValue(BtnFDown)==1) {
      PickupBonusSequenceState = 2;
    };
    break;

  case 2:
    if (P1) {
      driveDistance(200);
      //debugging for accuracy
      //delay(10000);
      //
      delay(500);
    };
    if (getMotorZeroVelocity(Left)) {
      PickupBonusSequenceState = 1;
    };
    break;

  case 3:
    if (P1) {
      driveDistance(-100);
      delay(400);
      ArmPresetValue=2;
      ArmHeightMove();
      delay(800);
```

```
    };
    if (getMotorZeroVelocity(ArmLeft) || (getTimerValue(T1)>3000)) {
      PickupBonusSequenceState = 4;
    };
    break;

  case 4:
    if (P1) {
      driveDistance(-600);
      delay(800);
      ArmPresetValue=0;
      ArmHeightMove();
      delay(100);
    };
    if(getTimerValue(T1)>1500) {
      ArmHeightMove();
    };
    if (getMotorZeroVelocity(Left) || (getTimerValue(T1)>3000)) {
      PickupBonusSequenceState = 1;
    };
    break;

    /*  case 5:
    if (TurnDegrees(90.0)) {
    PickupBonusSequenceState =1;
    };
    break;
    */
  default: PickupBonusSequenceState = 1;
  };
};

//////////////////////////////////////////////////////
//                Place cube Sequence               //
//////////////////////////////////////////////////////

void PlaceBonusSequence () {
  static int LastState2;
  bool P2;

  if (PlaceBonusSequenceState !=LastState2) {
    P2 = true;
    resetTimer(T1);
    LastState2 = PlaceBonusSequenceState;
  }
  else {
    P2 = false;
  };
  switch (PlaceBonusSequenceState){
  case 1:
    if (getJoystickValue(BtnFUp)==1) {
      PlaceBonusSequenceState = 2;
    };
    break;

  case 2:
    if (P2) {
```

```
      ArmPresetValue=3;
      ArmHeightMove();
      delay(100);
      GrabCube();
    };
    if (getMotorZeroVelocity(ArmLeft) || (getTimerValue(T1)>3000)) {
      PlaceBonusSequenceState = 3;
    };
    break;

  case 3:
    if (P2) {
      driveDistance(220);
      delay(1000);
      ArmPresetValue=2;
      ArmHeightMove();
      delay(400);
    };
    if(getTimerValue(T1)>1500) {
      ArmHeightMove();
    };
    if (getMotorZeroVelocity(Left) || (getTimerValue(T1)>3000)) {
      PlaceBonusSequenceState = 4;
    };
    break;

  case 4:
    if (P2) {
      ReleaseCube();
      delay(400);
      driveDistance(-200);
    };
    if (getMotorZeroVelocity(Left) || (getTimerValue(T1)>3000)) {
      PlaceBonusSequenceState = 5;
    };
    break;

  case 5:
    if (P2) {
      delay(800);
      ArmPresetValue=0;
      ArmHeightMove();
      delay(800);
    };
    if (getMotorZeroVelocity(ArmLeft) || (getTimerValue(T1)>3000)) {
      PlaceBonusSequenceState = 1;
      PlaceSequenceFinished = true;
    };
    break;

  default: PlaceBonusSequenceState = 1;
  };
};

void ArmReset() { // resets the arm if the bottom bumper is pressed
  if (getBumperValue(ArmBottomBumper)==1) {
    resetMotorEncoder(ArmLeft);
```

```
    resetMotorEncoder(ArmRight);
    setTouchLEDColor(LED,colorBlue);
    setTouchLEDColor(LED,colorNone);
  }
}

void ResetOdometry(){
  resetMotorEncoder(Left);
  resetMotorEncoder(Right);
  x = 0;
  y = 0;
};

void SwitchToAutonSkills (int DelayforAutonSwitchGo = 3000) {
  if (!AutonSwitched) {
    if (getTouchLEDValue(LED) && getBumperValue(AutonStart)) {
      AutonSwitchActive = true;
      setTouchLEDColor(LED, colorBlue);
      delay(1000);
      AutonSwitched = true;
      ProgramPersmissionToStart = false;
      setTouchLEDColor(LED, colorDarkBlue);
      } else {
      AutonSwitchActive = false;
    };
  };
}

void DataCollection2 () /* ignore values on bottom of screen only graph values a
Exp1 ArmPresetValue (fix negs)
Black = ArmPresetValue
Exp2 Motors
Drk-Green = Right
Purple = Left
Lime-Green = Intake
Maroon = CubeClaw

Exp3 Gyro Readings
blue = with drift
yellow = drift
red = Main Gyro with out Drift
*/
{
  int loops = 0;
  datalogClear();
  while(true)
  {
    global_1 = getGyroHeading(Main_Gyro); //series 1
    global_2 = gyroValue; // 2
    global_3 = gyroError; // 3
    global_4 = getColorHue(BallColor); //4
    global_5 = getMotorSpeed(Right); //5
    global_6 = getMotorSpeed(Left); //6
    global_7 = getMotorSpeed(Intake); //7
    global_8 = getMotorSpeed(CubeClaw); //8

    datalogDataGroupStart();
```

```
    datalogAddValue( DATALOG_SERIES_0, global_1 );
    datalogAddValue( DATALOG_SERIES_1, global_2 );
    datalogAddValue( DATALOG_SERIES_2, global_3 );
    datalogAddValue( DATALOG_SERIES_3, global_4 );
    datalogAddValue( DATALOG_SERIES_4, global_5 );
    datalogAddValue( DATALOG_SERIES_5, global_6 );
    datalogAddValue( DATALOG_SERIES_6, global_7 );
    datalogAddValue( DATALOG_SERIES_7, global_8 );
    datalogDataGroupEnd();

    wait1Msec(10);
    datalogAddValueWithTimeStamp( DATALOG_SERIES_3, global_3++ );
    wait1Msec(10);
    datalogAddValueWithTimeStamp( DATALOG_SERIES_3, global_3++ );

    // Repeat sequence every 360 loops
    if(loops++ == 360)
      loops = 0;

    // loop delay
    wait1Msec(10);
  }
}

/////////////////////////////////////////////////////////////////////////////////////////
//                                                    Tasks
/////////////////////////////////////////////////////////////////////////////////////////
task AutonSelectControl () {
  delay(2000);
  displayTextLine(5, "Controls Active");
  while (true) {
    if (0>AutonProgramSelector ){
      AutonProgramSelector = 0;
    };
    if (4<AutonProgramSelector) {
      AutonProgramSelector = 0;
    };

    if (!AutonLocked && !AutonFinished) {
      displayTextLine(1, "Auton Option=%d", AutonProgramSelector);
      if ((getTouchLEDValue(LED) && !AutonLocked) && AutonSelectLastState == 0)
        AutonSelectLastState = getTouchLEDValue(LED);
        AutonProgramSelector +=1;
        } else if (getBumperValue(AutonStart)) {
        AutonLocked = true;
      }
      else {
        AutonSelectLastState = getTouchLEDValue(LED);
      };
      } else {
      displayTextLine(1, "Controls Locked");
      displayTextLine(2, "Auton Selected=%d", AutonProgramSelector);
      displayTextLine(3, "Press LED to Start Auton");
      if (AutonLocked && getTouchLEDValue(LED)) {
        AutonPermissionToStart = true;
      }
      if (AutonPermissionToStart && getBumperValue(AutonStart)) {
```

```
        AutonPermissionToStart = false;
      }
    }
  };
}

task odometry () { // odometry task

  setMotorEncoderUnits(encoderCounts);
  while(true)
  {
    OdometryAngle = (LeftDriveTraveled - RightDriveTraveled)  / DriveWidth;
    LeftDriveTraveled = (((3.14159 * diameter) / ticksPerRev) * (getMotorEncoder
    RightDriveTraveled = (((3.14159 * diameter) / ticksPerRev) * (getMotorEncode
    x = ((LeftDriveTraveled + RightDriveTraveled) / 2) * cos(gyroValue);
    y = ((LeftDriveTraveled + RightDriveTraveled) / 2) * sin(gyroValue);
    delay(100);
  };
};


task gyroTask()
{
  long rate;
  long angle, lastAngle;
  // Change sensitivity, this allows the rate reading to be higher
  setGyroSensitivity(Main_Gyro, gyroNormalSensitivity);
  //Reset the gyro sensor to remove any previous data.
  resetGyro(Main_Gyro);
  wait1Msec(1000);
  repeat (forever) {
    rate = getGyroRate(Main_Gyro);
    angle = getGyroHeading(Main_Gyro);
    // If big rate then ignore gyro changes
    if( abs( rate ) < 2 )
    {
      if( angle != lastAngle )
        gyroError += lastAngle - angle;
    }
    lastAngle = angle;
    gyroValue = angle + gyroError;
    wait1Msec(10);
  }
}
/*
task keepStraight(){
while(true) {
HeadingStraight=0;
if(gyroValue<-2){HeadingStraight=-6;}
if(gyroValue>2){HeadingStraight=6;}
wait1Msec(100);
setMotorSpeed(Left, SpeedLeft-HeadingStraight);
setMotorSpeed(Right,SpeedRight +HeadingStraight);
}}*/

/*task datacollection()
/* ignore values on bottom of screen only graph values are valid /
```

```
Exp1 ArmPresetValue (fix negs)
Black = ArmPresetValue
Exp2 Motors
Drk-Green = Right
Purple = Left
Lime-Green = Intake
Maroon = CubeClaw

Exp3 Gyro Readings
blue = with drift
yellow = drift
red = Main_Gyro with out Drift
*/
/*
{
int loops = 0;
datalogClear();
while(true)
{
global_1 = getGyroHeading(Main_Gyro); //series 1
global_2 = gyroValue; // 2
global_3 = gyroError; // 3
global_4 = getColorHue(BallColor); //4
global_5 = getMotorSpeed(Right); //5
global_6 = getMotorSpeed(Left); //6
global_7 = getMotorSpeed(Intake); //7
global_8 = getMotorSpeed(CubeClaw); //8

datalogDataGroupStart();
datalogAddValue( DATALOG_SERIES_0, global_1 );
datalogAddValue( DATALOG_SERIES_1, global_2 );
datalogAddValue( DATALOG_SERIES_2, global_3 );
datalogAddValue( DATALOG_SERIES_3, global_4 );
datalogAddValue( DATALOG SERIES 4, global 5 );
datalogAddValue( DATALOG_SERIES_5, global_6 );
datalogAddValue( DATALOG_SERIES_6, global_7 );
datalogAddValue( DATALOG_SERIES_7, global_8 );
datalogDataGroupEnd();

wait1Msec(10);
datalogAddValueWithTimeStamp( DATALOG SERIES 3, global 3++ );
wait1Msec(10);
datalogAddValueWithTimeStamp( DATALOG_SERIES_3, global_3++ );

// Repeat sequence every 360 loops
if(loops++ == 360)
loops = 0;

// loop delay
wait1Msec(10);
}
}
*/

task Functions(){
  while(true){
    BatteryWarning();
```

```
    ArmReset();
    displayControl();
    SwitchToAutonSkills();
    DataCollection2();
  }
}
/////////////////////////////////////////////////////////////////////////////
//                                                    Main Task
/////////////////////////////////////////////////////////////////////////////

task main() { // main program code
  //SET MOTORS TO BRAKE MODE
  setMotorBrakeMode(ArmRight, motorHold);
  setMotorBrakeMode(ArmLeft, motorHold);
  setMotorBrakeMode(Right, motorHold);
  setMotorBrakeMode(Left, motorHold);
  setMotorBrakeMode(CubeClaw, motorHold);
  setMotorBrakeMode(Intake, motorHold);
  //RESET MOTOR ENCODERS
  resetMotorEncoder(ArmLeft);  //Resets Left Arm Motor Encoder to 0
  resetMotorEncoder(ArmRight); //Resets Right Arm Motor Encoder to 0
  resetMotorEncoder(Left);   //Resets Left Arm Motor Encoder to 0
  resetMotorEncoder(Right); //Resets Right Arm Motor Encoder to 0
  resetMotorEncoder(CubeClaw);   //Resets Left Arm Motor Encoder to 0
  resetMotorEncoder(Intake); //Resets Right Arm Motor Encoder to 0
  intakeStarted = false; // sets the variable that starts the intake to false
  resetTimer(timer2);
  GyroCustomCalibration(30);
  delay(10);
  startTask(Functions);
  //startTask();
  startTask(odometry);
  startTask(gyroTask);
  playSound(soundGasFillup);
  delay(100);

  LStart.pX = 635;
  LStart.pY = RobotCenterOffset;
  LStart.pA = FWD_DIR;

  RStart.pX = FIELDX-635;
  RStart.pY = RobotCenterOffset;
  RStart.pA = FWD_DIR;

  point1.pX = 0;
  point1.pY = 100;
  point1.pA = FWD_DIR;

  point2.pX = 0;
  point2.pY = 100;
  point2.pA = FWD_DIR+45;

  point3.pX = 0;
  point3.pY = 0;
  point3.pA = FWD_DIR+45;

  point4.pX = 0;
```

```
  point4.pY = 0;
  point4.pA = RIGHT_DIR;

  point5.pX = 0;
  point5.pY = 50;
  point5.pA = FWD_DIR;

  point6.pX = 0;
  point6.pY = 0;
  point6.pA = FWD DIR;
  while (true) {
    PickupBonusSequence();
    PlaceBonusSequence();
    while(/*timer2 < 90*/ProgramPersmissionToStart) //while the program is runni
    {
      //PickupBonusSequence();
      //PlaceBonusSequence();

      //datalogging
      /*
      datalogDataGroupStart();
      datalogAddValue( 2, nImmediateBatteryLevel);
      datalogAddValue( 6, getGyroHeading(Main_Gyro));
      datalogAddValue( 7, gyroValue);
      datalogAddValue( 8, gyroError);
      datalogDataGroupEnd();
      */

      //Arm Code
      if (getJoystickValue(BtnLDown)==1 && IndexArmPressed==false) {  //gets the
        ArmPresetValue = (ArmPresetValue-1);
        ArmHeightMove();
        IndexArmPressed=true;
      }
      else {
        if (getJoystickValue(BtnLUp)==1 && IndexArmPressed==false) {
          ArmPresetValue = (ArmPresetValue+1);
          ArmHeightMove();
          IndexArmPressed=true;
        }
      };
      if (getJoystickValue(BtnLDown)==0 && getJoystickValue(BtnLUp)==0) {
        IndexArmPressed=false;
      };

      if (0>ArmPresetValue){
        ArmPresetValue = 0;
      }
      else {
        if (4<ArmPresetValue) {
          ArmPresetValue = 4;
        };
      }

      // Intake Code
      if (getMotorEncoder(Intake)>=(990*2.666666666666666666666666666666666666666
        resetMotorEncoder(Intake);
```

```
    };

    if (!intakeStarted && getJoystickValue(BtnEDown)) {
      intakeStarted = true;
      resetTimer(timer2);
      setMotorSpeed(Intake, IntakeSpeed);
      } else if (getJoystickValue(BtnEDown) && (getColorHue(BallColor)<20)){
      setMotorSpeed(Intake, (IntakeSpeed*0.75));
      } else if (getJoystickValue(BtnEDown)) {
      setMotorSpeed(Intake, IntakeSpeed);
      } else if (getJoystickValue(BtnEUp)) {
      setMotorSpeed(Intake,  -IntakeSpeed);
      }else if (!getJoystickValue(BtnEDown)){
      setMotorSpeed(Intake, 0);
    };

    //Claw Code
    if (getJoystickValue(BtnRUp) && !MaxClawBrake) {
      setMotorSpeed(CubeClaw, 100);
      } else if (getJoystickValue(BtnRDown) && !MaxClawBrake) {
      setMotorSpeed(CubeClaw, -100);
      } else {
      setMotorSpeed(CubeClaw, 0);
    };

    //DriveCode
    if (PickupBonusSequenceState ==1 && PlaceBonusSequenceState ==1) {
      if(abs(getJoystickValue(ChA))>25 || abs(getJoystickValue(ChD))>25) {  //if
        setMotorSpeed(Left, getJoystickValue(ChA)); //set the value of the motor
        setMotorSpeed(Right, getJoystickValue(ChD)); //set the value of the motc
      }
      else { setMotorSpeed(Left, 0);  // if nothing is happening on the controll
        setMotorSpeed(Right, 0); // if nothing is happening on the controller se
      };
    };
    ///////////////EODC////////////
  };                              ///
  ////////////////////////////////

  //////////////SOAS/////////////
  while (!ProgramPersmissionToStart && AutonSwitched) {
    PickupBonusSequence();
    PlaceBonusSequence();
    while (!AutonDisplayInt) {
      displayTextLine(0, "Auton Switched");
      displayClearTextLine(1);
      displayClearTextLine(2);
      displayClearTextLine(3);
      displayClearTextLine(4);
      displayClearTextLine(5);
      startTask(AutonSelectControl);
      AutonDisplayInt = true;
    };


    while (AutonPermissionToStart) {
      PickupBonusSequence();
```

```
      PlaceBonusSequence();
      switch (AutonProgramSelector) {
        ////////////////////////////////////////////////////////////////////////

      case 0:
        waitUntil(AutonPermissionToStart);
        setTouchLEDColor(LED,colorGreen);
        displayTextLine(4, "Drive 100 (TEST)");
        // Start of Auton Option 0 (Null/Test)
        driveDistance(100);
        delay(3000);
        driveDistance(-100);
        delay(1000);
        ////////////////////////////////////
        setTouchLEDColor(LED,colorViolet);
        AutonPermissionToStart = false;
        AutonFinished = true;
        break;

      case 1:
        waitUntil(AutonPermissionToStart);
        setTouchLEDColor(LED,colorGreen);
        displayTextLine(4, "Right Auton");
        // Start of Auton Option 1
        ////////////////////Stage 1//////////////////////
        ReleaseCube();
        //Swing Turning
        moveMotorTargetMM(Left,138);
        delay(1000);
        moveMotorTargetMM(Right,235,50);
        delay(1200);
        GrabCube();
        delay(250);
        driveDistance(85);
        delay(1200);
        ArmPresetValue = 2;
        ArmHeightMove();
        moveMotorTargetMM(Left,97);
        delay(1000);
        //Forward to Sequence Preset
        driveDistance(180);
        delay(1000);
        //Sequence
        //PlaceBonusSequenceState = 2;
        //Sequence Alternative for Extended Run (Stage 2)
        //2
        ArmPresetValue=3;
        ArmHeightMove();
        delay(100);
        GrabCube();
        //3
        driveDistance(220);
        delay(1000);
        ArmPresetValue=2;
        ArmHeightMove();
        delay(400);
        //4
```

```
    ReleaseCube();
    delay(400);
    driveDistance(-210);
    //5
    delay(800);
    ArmPresetValue=0;
    ArmHeightMove();
    delay(800);
    //End of Sequence Alternative
    ////////////////////Stage 2//////////////////////
    delay(100);
    //waitUntil(PlaceBonusSequenceState==1);
    moveMotorTargetMM(Right,367,75);
    delay(1200);
    driveDistance(205);
    delay(1000);
    moveMotorTargetMM(Right,100,75);
    delay(500);
    driveDistance(70);
    GrabCube();
    delay(500);
    ArmPresetValue = 1;
    ArmHeightMove();
    delay(1000);
    driveDistance(195);
    delay(500);
    // Turn to Middle Tower
    moveMotorTargetMM(Right,260);
    delay(750);
    //arm up to max
    ArmPresetValue = 4;
    ArmHeightMove();
    delay(750);
    //drive to tower
    driveDistance(250);
    delay(850);
    ArmPresetValue = 3;
    ArmHeightMove();
    delay(200);
    ReleaseCube();
    delay(200);
    driveDistance(-200);
    delay(500);
    ArmPresetValue = 0;
    ArmHeightMove();
    ////////////////////////////////////////
    delay(1000);
    setTouchLEDColor(LED,colorViolet);
    //RobotReset();
    AutonPermissionToStart = false;
    AutonFinished = true;
    break;

  case 2:
    waitUntil(AutonPermissionToStart);
    setTouchLEDColor(LED,colorGreen);
    displayTextLine(4, "Left Auton");
```

```
        // Start of Auton Option 2

        ////////////////////////////////////////
        delay(1000);
        setTouchLEDColor(LED,colorViolet);
        //RobotReset();
        AutonPermissionToStart = false;
        AutonFinished = true;
        break;


        ///end of auton entries
        ////////////////////////////////////////////////////////////////////////
      default: AutonProgramSelector = 0;
      };
    }
    ////////////EOAS//////////////
  };                              ///
  ////////////////////////////////

  ////////////////////////Reset////////////////////////////////
  while(!ProgramPersmissionToStart && !AutonSwitched) {          ///
    displayTextLine(0, "EOF Please Press LED to Reset");         ///
    displayClearTextLine(1);                                     ///
    displayClearTextLine(2);                                     ///
    displayClearTextLine(3);                                     ///
    displayClearTextLine(4);                                     ///
    displayClearTextLine(5);                                     ///
    if (getTouchLEDValue(LED)) {                                 ///
      ProgramPersmissionToStart = true;                          ///
    }                                                            ///
  };                                                             ///
  ////////////////////////////////////////////////////////////////

  //////////////EOTMWL////////
};                           ///
////////////////////////////////

//////////////EOTM//////////
};                         ///
////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//                                          End of Code
//                                  Last Known Compile: 19/08/2019, 1548
//                                          By Joseph Greening
//                                      3428B VIQC Squared Away
////////////////////////////////////////////////////////////////////////////////
```