```
#pragma config(Sensor, in1,    lift,           sensorPotentiometer)
#pragma config(Sensor, in2,    flip,           sensorPotentiometer)
#pragma config(Sensor, in3,    autonDirection, sensorPotentiometer)
#pragma config(Sensor, in8,    gyro,           sensorGyro)
#pragma config(Sensor, dgtl1,  ,              sensorQuadEncoder)
#pragma config(Sensor, dgtl3,  ,              sensorQuadEncoder)
#pragma config(Sensor, dgtl5,  ,              sensorQuadEncoder)
#pragma config(Sensor, dgtl10, jump,          sensorDigitalIn)
#pragma config(Sensor, dgtl11, resetSwitch,   sensorTouch)
#pragma config(Sensor, dgtl12, catapultSwitch, sensorTouch)
#pragma config(Motor,  port1,           ballIntake,    tmotorVex393_HE
#pragma config(Motor,  port2,           leftDrive,     tmotorVex393_MC
#pragma config(Motor,  port3,           leftLift,      tmotorVex393_MC
#pragma config(Motor,  port4,           leftExtra,     tmotorVex393_MC
#pragma config(Motor,  port5,           rightExtra,    tmotorVex393_MC
#pragma config(Motor,  port6,           leftCatapult,  tmotorVex393_MC
#pragma config(Motor,  port7,           rightCatapult, tmotorVex393_MC
#pragma config(Motor,  port8,           rightLift,     tmotorVex393_MC
#pragma config(Motor,  port9,           rightDrive,    tmotorVex393_MC
#pragma config(Motor,  port10,          flipper,       tmotorVex393_HE
//*!!Code automatically generated by 'ROBOTC' configuration wizard

/*---------------------------------------------------------------------
/*
/*        Description: Competition template for VEX EDR
/*
/*---------------------------------------------------------------------

// This code is for the VEX cortex platform
#pragma platform(VEX2)

// Select Download method as "competition"
#pragma competitionControl(Competition)

//Main competition background code...do not modify!
#include "Vex_Competition_Includes.c"

/*---------------------------------------------------------------------
/*                          Pre-Autonomous Functions
/*
/*  You may want to perform some actions before the competition starts
/*  Do them in the following function.  You must return from this func
/*  or the autonomous and usercontrol tasks will not be started.  This
/*  function is only called once after the cortex has been powered on
/*  not every time that the robot is disabled.
/*---------------------------------------------------------------------
//Constants
int catapultConstant = 17;
int catapultDownVal = 104;
```

```
int flipFullUpVal = 1500;
int flipUpVal = 1260;
int flipDownVal = 650;
int flipHalf = 900;
int leftMaxSpeed = 127;
int rightMaxSpeed = 127;
int gyroOffset = 200;
//Variables
int rightDriveVal;
int leftDriveVal;
int liftEnable = 1;
int flipperEnable = 1;
int driveEnable = 1;
int catapultEnable = 1;
int autoLiftVal;
bool flipperUp;
int maxLift = 0;
bool flipperUpOk = true;

task autoLift ()
{
  liftEnable = 0;

  while(!liftEnable && SensorValue[lift] > autoLiftVal)
    motor[leftLift] = motor[rightLift] = 127;

  liftEnable = 1;
}

task autoFlip ()
{
  flipperEnable = 0;
  if(flipperUp)
  {
    while(!flipperEnable && SensorValue[flip] > flipDownVal)
      motor[flipper] = -127;
  }
  else
  {
    while(!flipperEnable && SensorValue[flip] < flipFullUpVal && flipp
      motor[flipper] = 127;
  }
  flipperEnable = 1;
}


task autoLaunch()
{
  catapultEnable = 0;
```

```
  while(!catapultEnable)
  {
    motor[leftCatapult] = motor[rightCatapult] = 127;
    waitUntil(SensorValue[catapultSwitch] == 1);
    wait1Msec(200);
    waitUntil(SensorValue[catapultSwitch] == 0);
    resetMotorEncoder(leftCatapult);

    //here is the parts that can get changed
    waitUntil(getMotorEncoder(leftCatapult) /*here is the changeable p
    motor[leftCatapult] = motor[rightCatapult] = catapultConstant;

    //exits this part (very important)
    catapultEnable = 1;
  }

}
void pre_auton(){}


task displayVal()
{
  SensorValue[gyro] = 0;
  while(true)
  {

    long a = getMotorEncoder(leftDrive);
    long b = getMotorEncoder(rightDrive);
    long c = SensorValue[gyro];
    writeDebugStream("Left Drive: %d", a);
    writeDebugStreamLine("Right Drive: %d", b);
    writeDebugStreamLine("Test: %d", c);
    if(SensorValue[resetSwitch])
    {
      resetMotorEncoder(rightDrive);
      resetMotorEncoder(leftDrive);
      resetMotorEncoder(leftCatapult);
      SensorValue[gyro] = 0;
    }
    wait1Msec(2);
    clearDebugStream();

  }
}
task driveTest()
{
  int resetPressed = 0;
  while (true)
  {
```

```
    if(SensorValue[resetSwitch] && !resetPressed && driveEnable)
    {
      driveEnable = false;
      while(getMotorEncoder(rightDrive)<1500 && !driveEnable)
      {
        motor[leftDrive] = motor[leftExtra] = leftMaxSpeed;
        motor[rightDrive] = motor[rightExtra] = rightMaxSpeed;
      }
      driveEnable = true;
      resetPressed = 1;
    }

  }
}
/*---------------------------------------------------------------------
/*
/*                                Autonomous Task
/*
/*  This task is used to control your robot during the autonomous phas
/*  a VEX Competition.
/*
/*  You must modify the code to add your own robot specific commands h
/*---------------------------------------------------------------------
void turn(int degree)
{
  SensorValue[gyro] = 0;
  int gyroTarget = (930/90) * degree;
  if (gyroTarget > 0)
  {
    motor[leftDrive]  =  80;
    motor[rightDrive] = -60;
    waitUntil(-SensorValue[gyro] > gyroTarget - gyroOffset);
    motor[leftDrive]  = -127;
    motor[rightDrive] =  127;
    wait1Msec(30);
  }
  else if (gyroTarget < 0)
  {
    motor[leftDrive]  = -80;
    motor[rightDrive] =  60;
    waitUntil(-SensorValue[gyro] < gyroTarget + gyroOffset);
    motor[leftDrive]  =  127;
    motor[rightDrive] = -127;
    wait1Msec(30);
  }
  motor[rightDrive] = motor[leftDrive] = 0;
  wait1Msec(70);
}
void move(int distance)
```

```
{
  resetMotorEncoder(leftDrive);
  resetMotorEncoder(rightDrive);
  if(distance > 0)
  {
    motor[leftDrive] = motor[leftExtra] = leftMaxSpeed;
    motor[rightDrive]= motor[rightExtra] = rightMaxSpeed;
    waitUntil((getMotorEncoder(leftDrive)+getMotorEncoder(rightDrive))
  }
  else
  {
    motor[leftDrive] = motor[leftExtra] = -leftMaxSpeed;
    motor[rightDrive]= motor[rightExtra] = -rightMaxSpeed;
    waitUntil((getMotorEncoder(leftDrive)+getMotorEncoder(rightDrive))
  }
  motor[leftDrive] = motor[leftExtra] = motor[rightDrive] = motor[righ
}
task autonomous()
{
  int direction;
  if(SensorValue[autonDirection] < 1850)
  {
    direction = 1;
  }
  else
  {
    direction = -1;
  }
  if(SensorValue[jump] == 0)
  {
    /*startTask(autoLaunch);
    waitUntil(SensorValue[catapultSwitch] == 1);
    wait1Msec(200);
    turn(-130*direction);
    move(200);
    wait1Msec(50);*/
    motor[ballIntake] = 127;
    move(-1670);
    wait1Msec(1300);
    motor[ballIntake] = 0;
    motor[flipper] = -127;
    waitUntil(SensorValue[flip] < flipDownVal);
    motor[flipper] = 0;
    turn(-100*direction);
    move(285);
    motor[flipper] = 127;
    wait1Msec(500);
    move(60);
    waitUntil(SensorValue[flip] > flipUpVal);
```

```
    motor[flipper] = 0;
    turn(-20*direction);
    move(-2300);
  }
  else if (SensorValue[jump] == 1)
  {
    motor[ballIntake] = 70;
    move(-3000);
    wait1Msec(1600);
    move(2900);
    wait1Msec(300);
    turn(-90*direction);
    wait1Msec(300);

    int offset = 300;
    move(offset);
    wait1Msec(300);

    motor[flipper] = -127;
    waitUntil(SensorValue[flip] < 650);
    //wait1Msec(1400);
    motor[flipper] = 0;
    startTask(autoLaunch);
    wait1Msec(1000);
    motor[flipper] = 127;
    waitUntil(SensorValue[flip] > flipUpVal);
    //wait1Msec(900);
    motor[flipper] = 0;

    move(3000-offset);
    wait1Msec(300);
    move(-3180);

    turn(-90*direction);
    /*if(direction == 1)
    {
      motor[leftDrive] = -60;
      motor[rightDrive] = 60;
      waitUntil(getMotorEncoder(rightDrive) > 180);
    }
    else
    {
      motor[leftDrive] =  60;
      motor[rightDrive] =-60;
      waitUntil(getMotorEncoder(rightDrive) < -180);
    }*/
    //move(0);
    move(5560);
    resetMotorEncoder(rightDrive);
```

```
    resetMotorEncoder(leftDrive);
    liftEnable = flipperEnable = driveEnable = catapultEnable = 1;
  }
}

/*---------------------------------------------------------------------
/*
/*                              User Control Task
/*
/*  This task is used to control your robot during the user control ph
/*  a VEX Competition.
/*
/*  You must modify the code to add your own robot specific commands h
/*---------------------------------------------------------------------



task usercontrol()
{
  //INITIAL TASKS
  startTask(displayVal);
  //startTask(driveTest);
  //startTask(autoLaunch);

  // User control code here, inside the loop
  int buttonPressed;
  int buttonToggleState;
  int liftPressed;
  int liftToggleState;

  flipperEnable = driveEnable = liftEnable = 1;

  int D8pressed = 50; //50 is just a random number that is not 1 or 0
  int R8pressed = 50;
  int L8pressed = 50;
  int R7pressed = 50;
  int intakeVal;
  int driveRatio;

  while (true)
  {
    //TOGGLE CONTROLS

    if (vexRT[Btn8U])
    {
      if (!buttonPressed)
      {
        // change the toggle state
        buttonToggleState = 1 - buttonToggleState;
```

```
        // Note the button is pressed
        buttonPressed = 1;

      }
    }
    else
    {
      // the button is not pressed
      buttonPressed = 0;
    }


    // Now do something with our toggle flag
    if (buttonToggleState)
      intakeVal = 127;
    else
      intakeVal = 0;

    if(vexRT[Btn7U])
    {
      intakeVal = -127;
    }
    motor[ballIntake] = intakeVal;


    if (vexRT[Btn7D])
    {
      if (!liftPressed)
      {
        // change the toggle state
        liftToggleState = 1 - liftToggleState;

        // Note the button is pressed
        liftPressed = 1;

      }
    }
    else
    {
      // the button is not pressed
      liftPressed = 0;
    }

    if (liftToggleState)
      maxLift = 2300;
    else
      maxLift = 0;
```

```
    //if low heigh toggle is on, flipper is disabled at a certain heig
    if(SensorValue[lift] < 2580 && maxLift == 2300)
      flipperUpOk = false;
    else
      flipperUpOk = true;
    //MANUAL CONTROL

    if(vexRT[Btn6U])
      driveRatio = 2.85;
    else
      driveRatio = 1;
    leftDriveVal  = vexRT[Ch3] + vexRT[Ch4];
    rightDriveVal = vexRT[Ch3] - vexRT[Ch4];


    motor[rightDrive] = motor[rightExtra] = rightDriveVal / driveRatio
    motor[leftDrive] = motor[leftExtra] = leftDriveVal / driveRatio;
    if(liftEnable && SensorValue[lift] > maxLift)
      motor[rightLift] = motor[leftLift] = vexRT[Ch2];

    //Corrects if too high
    if(SensorValue[lift] < maxLift)
    {
      liftEnable = 1;
      motor[rightLift] = motor[leftLift] = -30;
    }

    if(catapultEnable)
      motor[rightCatapult] = motor[leftCatapult] = vexRT[Btn6D] * (127

    if(flipperEnable)
    {
      //motor[flipper] = (vexRT[Btn5U]-vexRT[Btn5D]) * 127;
      if(vexRT[Btn5U] && flipperUpOk)
        motor[flipper] = 20;
      else if(vexRT[Btn5D])
        motor[flipper] = -127;
      else
        motor[flipper] = 0;
    }


    //Motor Enable Override
    if(vexRT[Btn7L])
      liftEnable = flipperEnable = driveEnable = catapultEnable = 1;

    //is the flipper up or down?
    if(SensorValue[flip] > flipHalf)
      flipperUp = true;
```

```
    else
      flipperUp = false;

    //Automation Summons
    //High Lift: 1250
    //Low Lift:  1760
    //low flip:  655
    //mid flip:  1325
    //high flip: 1980

    //summons automatic task if button is currently pressed but was nc
    if(vexRT[Btn8D] && !D8pressed && liftEnable && SensorValue[lift] >
    {
      autoLiftVal = 1760;
      startTask(autoLift);
    }

    if(vexRT[Btn8R] && !R8pressed && liftEnable && SensorValue[lift] >
    {
      autoLiftVal = 1250;
      startTask(autoLift);
    }

    if(vexRT[Btn8L] && !L8pressed && flipperEnable)
      startTask(autoFlip);

    if(vexRT[Btn7R] && !R7pressed && catapultEnable)
      startTask(autoLaunch);

    D8pressed = vexRT[Btn8D];
    R8pressed = vexRT[Btn8R];
    L8pressed = vexRT[Btn8L];
    R7pressed = vexRT[Btn7R];
  }

}
```