#pragma config(I2C\_Usage, I2C1, i2cSensors)

#pragma config(Sensor, dgtl10, clawtouch, sensorTouch)

#pragma config(Sensor, dgtl11, frontsonarinch, sensorSONAR\_inch)

#pragma config(Sensor, I2C\_1, frontrightint, sensorQuadEncoderOnI2CPort, , AutoAssign )

#pragma config(Sensor, I2C\_2, frontleftint, sensorQuadEncoderOnI2CPort, , AutoAssign )

#pragma config(Motor, port2, frontleft, tmotorVex393HighSpeed\_MC29, openLoop, driveLeft, encoderPort, I2C\_2)

#pragma config(Motor, port3, rearleft, tmotorVex393HighSpeed\_MC29, openLoop, driveLeft)

#pragma config(Motor, port6, frontright, tmotorVex393HighSpeed\_MC29, openLoop, reversed, driveRight, encoderPort, I2C\_1)

#pragma config(Motor, port7, rearright, tmotorVex393HighSpeed\_MC29, openLoop, reversed, driveRight)

#pragma config(Motor, port9, claw, tmotorVex393HighSpeed\_MC29, openLoop, reversed)

#pragma config(DatalogSeries, 0, "front left motor power", Motors, MotorPower, port2, 100)

#pragma config(DatalogSeries, 1, "rear left motor power", Motors, MotorPower, port3, 100)

#pragma config(DatalogSeries, 2, "front right motor power", Motors, MotorPower, port6, 100)

#pragma config(DatalogSeries, 3, "rear right motor power", Motors, MotorPower, port7, 100)

#pragma config(DatalogSeries, 4, "claw motor power", Motors, MotorPower, port9, 100)

#pragma config(DatalogSeries, 5, "battery level", Properties, immediateBatteryLevel, , 100)

#pragma config(DatalogSeries, 6, "rangefinder vaue?", Sensors, Sensor, dgtl11, 100)

#pragma config(DatalogSeries, 7, "button value", Sensors, SensorBool, dgtl10, 100)

//\*!!Code automatically generated by 'ROBOTC' configuration wizard !!\*//

// 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"

//Main Smart Motor Library Background Code

#include "SmartMotorLib.c"//must be put in C:\Program Files (x86)\Robomatter Inc\ROBOTC Development Environment 4.X\Includes

int c1, c2, c3, c4, x, y, z;//master joystick

int c1s, c2s, c3s, c4s, xs, ys, zs;//sister joystick

int batteryvoltage;

int upperthreshhold = 20, lowerthreshhold = -20;

int rightencode = 0, leftencode = 0;

const short leftbutton = 1;

const short centerbutton = 2;

const short rightbutton = 4;

int autonomousselectorcount = 0, drivemodeselectorcount = 0, menudisplaycount = 0;

//\*!!Code automatically generated by 'ROBOTC' configuration wizard !!\*//

/\*---------------------------------------------------------------------------\*/

/\* \*/

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

/\* \*/

/\*---------------------------------------------------------------------------\*/

//play around with tasks http://www.robotc.net/wikiarchive/Multitasking\_tips

//declare user tasks and methods here

task record();//declare task record

task sound();

task clawcontrol();

void waitforpress();

void waitforrelease();

void clearlcd();

void statdisplay();

void driveselector();

void autonselector();

void updatemaster();

void updatesister();

void rightdrive();

void leftdrive();

void frontdrive();

void reardrive();

void driveforward();

void turn();

void tank();

void arcade();

void tilt();

void meccanum();

void drive();

void auton();

void auton1();

void auton2();

void auton3();

void auton4();

void IMEreset();

void UpdateEncoders();

//task timeout();//declare task timeout

void IMEreset(){

resetMotorEncoder(frontright);

resetMotorEncoder(frontleft);

}

void UpdateEncoders(){

rightencode = nMotorEncoder[frontright];

leftencode = nMotorEncoder[frontleft];

}

void waitforpress(){

clearTimer(T1);

while(nLCDButtons == 0 && time1[T1] < 2000){}//waits until lcd button pressed or 1 second has passed

wait1Msec(5);//allows slight delay for program to catch up

}

void waitforrelease(){

while(nLCDButtons != 0){}

wait1Msec(5);//allows slight delay for program to catch up

}

void clearlcd(){

clearLCDLine(0);

clearLCDLine(1);

}

void statdisplay(){

switch (menudisplaycount){

case 0:

clearlcd();

displayLCDCenteredString(0, "Battery Voltage");

string mainbattery;

sprintf(mainbattery, "%1.2f%c", batteryvoltage/1000, 'V');//build value

displayLCDCenteredString(1, mainbattery);//print value

if(nLCDButtons == leftbutton){

waitforrelease();

menudisplaycount = 2;

}

else if(nLCDButtons == rightbutton){

waitforrelease();

menudisplaycount++;

}

break;

case 1:

clearlcd();

displayLCDCenteredString(0, "Range Finder");

string alpha;

sprintf(alpha, "%1.2f%c", SensorValue[frontsonarinch], "in");//build value

displayLCDCenteredString(1, alpha);//print value

if(nLCDButtons == leftbutton){

waitforrelease();

menudisplaycount--;

}

else if(nLCDButtons == rightbutton){

waitforrelease();

menudisplaycount++;

}

break;

case 2:

clearlcd();

displayLCDString(0, 0, "FREncode");

displayLCDString(1, 0, "FLEncode");

displayLCDNumber(0, 9, rightencode, 0.01);

displayLCDNumber(1, 9, leftencode, 0.01);

if(nLCDButtons == leftbutton){

waitforrelease();

menudisplaycount--;

}

else if(nLCDButtons == rightbutton){

waitforrelease();

menudisplaycount = 0;

}

break;

}

}

void driveselector(){

wait1Msec(500);//pause so two options are not chosen at the same time

while (nLCDButtons != centerbutton){

switch (drivemodeselectorcount){

case 0:

clearlcd();

displayLCDCenteredString(0, "TANK DRIVE");

displayLCDCenteredString(1, "<--ENTER-->");

waitforpress();

if(nLCDButtons == leftbutton){

waitforrelease();

drivemodeselectorcount = 3;

}

else if(nLCDButtons == rightbutton){

waitforrelease();

drivemodeselectorcount++;

}

break;

case 1:

clearlcd();

displayLCDCenteredString(0, "ARCADE DRIVE");

displayLCDCenteredString(1, "<--ENTER-->");

waitforpress();

if(nLCDButtons == leftbutton){

waitforrelease();

drivemodeselectorcount--;

}

else if(nLCDButtons == rightbutton){

waitforrelease();

drivemodeselectorcount++;

}

break;

case 2:

clearlcd();

displayLCDCenteredString(0, "TILT DRIVE");

displayLCDCenteredString(1, "<--ENTER-->");

waitforpress();

if(nLCDButtons == leftbutton){

waitforrelease();

drivemodeselectorcount--;

}

else if(nLCDButtons == rightbutton){

waitforrelease();

drivemodeselectorcount++;

}

break;

case 3:

clearlcd();

displayLCDCenteredString(0, "MECCANUM DRIVE");

displayLCDCenteredString(1, "<--ENTER-->");

waitforpress();

if(nLCDButtons == leftbutton){

waitforrelease();

drivemodeselectorcount--;

}

else if(nLCDButtons == rightbutton){

waitforrelease();

drivemodeselectorcount = 0;

}

break;

}

}

}

void autonselector(){

wait1Msec(500);//pause so two options are not chosen at the same time

while (nLCDButtons != centerbutton){

switch (autonomousselectorcount){

case 0:

clearlcd();

displayLCDCenteredString(0, "Auton 1");

displayLCDCenteredString(1, "<--ENTER-->");

waitforpress();

if(nLCDButtons == leftbutton){

waitforrelease();

autonomousselectorcount = 3;

}

else if(nLCDButtons == rightbutton){

waitforrelease();

autonomousselectorcount++;

}

break;

case 1:

clearlcd();

displayLCDCenteredString(0, "Auton 2");

displayLCDCenteredString(1, "<--ENTER-->");

waitforpress();

if(nLCDButtons == leftbutton){

waitforrelease();

autonomousselectorcount--;

}

else if(nLCDButtons == rightbutton){

waitforrelease();

autonomousselectorcount++;

}

break;

case 2:

clearlcd();

displayLCDCenteredString(0, "Auton 3");

displayLCDCenteredString(1, "<--ENTER-->");

waitforpress();

if(nLCDButtons == leftbutton){

waitforrelease();

autonomousselectorcount--;

}

else if(nLCDButtons == rightbutton){

waitforrelease();

autonomousselectorcount++;

}

break;

case 3:

clearlcd();

displayLCDCenteredString(0, "Auton 4");

displayLCDCenteredString(1, "<--ENTER-->");

waitforpress();

if(nLCDButtons == leftbutton){

waitforrelease();

autonomousselectorcount--;

}

else if(nLCDButtons == rightbutton){

waitforrelease();

autonomousselectorcount = 0;

}

break;

}

}

}

void updatemaster(){

c1 = vexRT[Ch1];

c2 = vexRT[Ch2];

c3 = vexRT[Ch3];

c4 = vexRT[Ch4];

x = vexRT[AccelX];//side to side

y = vexRT[AccelY];//forward and back

z = vexRT[AccelZ];//rotation?

batteryvoltage = nImmediateBatteryLevel;//voltage in thoustandths of volts

}

void updatesister(){

c1s = vexRT[Ch1Xmtr2];

c2s = vexRT[Ch2Xmtr2];

c3s = vexRT[Ch3Xmtr2];

c4s = vexRT[Ch4Xmtr2];

xs = vexRT[AccelXXmtr2];

ys = vexRT[AccelYXmtr2];

zs = vexRT[AccelZXmtr2];

}

//SetMotor( int index, int value = 0, bool immeadiate = false )//index is port, value is speed, immediate is for traditional motor[] driving (jumpy)

void rightdrive(int power){

SetMotor(frontright,power,false);

SetMotor(rearright,power,false);

//standard jumpy functions

//motor[frontright] = power;

//motor[rearright] = power;

}

void leftdrive(int power){

SetMotor(frontleft,power,false);

SetMotor(rearleft,power,false);

//standard jumpy functions

//motor[frontleft] = power;

//motor[rearleft] = power;

}

void frontdrive(int power){

SetMotor(frontright,power,false);

SetMotor(frontleft,power,false);

//standard jumpy functions

//motor[frontright] = power;

//motor[frontleft] = power;

}

void reardrive(int power){

SetMotor(rearleft,power,false);

SetMotor(rearright,power,false);

//standard jumpy functions

//motor[rearleft] = power;

//motor[rearright] = power;

}

void driveforward(int power){

rightdrive(power);

leftdrive(power);

}

void driveforward(int power, int time){

rightdrive(power);

leftdrive(power);

wait1Msec(time);

rightdrive(0);

leftdrive(0);

}

void turn(int rightpower, int leftpower){

rightdrive(rightpower);

leftdrive(leftpower);

}

void turn(int rightpower, int leftpower, int time){

rightdrive(rightpower);

leftdrive(leftpower);

wait1Msec(time);

rightdrive(0);

leftdrive(0);

}

void tank(){

int a = c2;

int b = c3;

if(a>upperthreshhold || a < lowerthreshhold){

rightdrive(a);

}

else{

rightdrive(0);

}

if(b>upperthreshhold || b < lowerthreshhold){

leftdrive(b);

}

else{

leftdrive(0);

}

}

void arcade(){

int a = c3 - c4;

int b = c3 + c4;

if(a>upperthreshhold || a < lowerthreshhold){

rightdrive(a);

}

else{

rightdrive(0);

}

if(b>upperthreshhold || b < lowerthreshhold){

leftdrive(b);

}

else{

leftdrive(0);

}

}

void tilt(){

int a = -y-x;

int b = -y+x;

if(a>upperthreshhold || a < lowerthreshhold){

rightdrive(a);

}

else{

rightdrive(0);

}

if(b>upperthreshhold || b < lowerthreshhold){

leftdrive(b);

}

else{

leftdrive(0);

}

}

void meccanum(){

int a = c3 - c1 - c4;

int b = c3 - c1 + c4;

int c = c3 + c1 + c4;

int d = c3 + c1 - c4;

if(a>upperthreshhold || a < lowerthreshhold){

motor[frontright] = a;

}

else{

motor[frontright] = 0;

}

if(b>upperthreshhold || b < lowerthreshhold){

motor[rearright] = b;

}

else{

motor[rearright] = 0;

}

if(c>upperthreshhold || c < lowerthreshhold){

motor[frontleft] = c;

}

else{

motor[frontleft] = 0;

}

if(d>upperthreshhold || d < lowerthreshhold){

motor[rearleft] = d;

}

else{

motor[rearleft]= 0;

}

}

void auton1(){

}

void auton2(){

}

void auton3(){

}

void auton4(){

}

void drive(){

switch (drivemodeselectorcount){

case 0:

tank();

break;

case 1:

arcade();

break;

case 2:

tilt();

break;

case 3:

meccanum();

break;

}

}

void auton(){

switch (autonomousselectorcount){

case 0:

auton1();

break;

case 1:

auton2();

break;

case 2:

auton3();

break;

case 3:

auton4();

break;

}

}

void pre\_auton()

{

bLCDBacklight = true;

autonselector();

driveselector();

//startTask(timeout);

//need to start task to wait for 10 seconds and if no input just go with default code

// Set bStopTasksBetweenModes to false if you want to keep user created tasks

// running between Autonomous and Driver controlled modes. You will need to

// manage all user created tasks if set to false.

bStopTasksBetweenModes = true;

// Set bDisplayCompetitionStatusOnLcd to false if you don't want the LCD

// used by the competition include file, for example, you might want

// to display your team name on the LCD in this function.

// bDisplayCompetitionStatusOnLcd = false;

// when true displays time spent deactivated

}

task autonomous()

{

auton();

}

task usercontrol()

{

IMEreset();//resets integrated motor encoders

//clearTimer(T2);//starts a timer used for the datalog

datalogBackgroundPollingResume();//starts the polling of values in the datalog as defined in motor and sensor setup

//to use the datalog connect robot and controller then connect wireless programming terminal and click start datalog in debug window then profit.

//start and initialize the smart motor library

SmartMotorRun();

SmartMotorsInit();

//startTask(sound);

startTask(clawcontrol);

// User control code here, inside the loop

datalogClear();

while (true)

{

UpdateEncoders();

statdisplay();

updatemaster();

updatesister();

drive();

{//beep control

if(vexRT[Btn6U]){

playTone(4000, 15);

}

else if(vexRT[Btn6D]){

playTone(2000, 15);

}

else{

clearSounds();

}

}

//startTask(record);//dont put tasks in loops

}

}

task clawcontrol(){

while (true){

{//claw control

if(vexRT[Btn5U]){

//motor[claw] = 90;

SetMotor(claw, 90, true);

}

else if(vexRT[Btn5D]){

//motor[claw] = -90;

SetMotor(claw, -90, true);

}

else if(vexRT[Btn5D] == false && vexRT[Btn5U] == false && SensorValue[clawtouch] == true){

//hold

SetMotor(claw, 15, true);

}

else{

//motor[claw] = 0;

SetMotor(claw, 0, true);

}

}

}

}

task record()//initialize record //need to figure out better

{

//C1 | C2 | C3 | C4 | BATT | TIME

clearTimer(T2);//starts a timer

while (true)//I think I need to connect the controller to the robot then use the wireless kit to be able to view

{

datalogDataGroupStart();//make as set so it occupies the same row

if(time1[T2] % 100 == 0){//create a new data series every tenth of a second

//datalogAddValue(nDataSeries, nDataValue);

datalogAddValue(0, c1);//print value of channel 1 master in collum 1

datalogAddValue(1, c2);//print value of channel 2 master in collum 2

datalogAddValue(2, c3);//print value of channel 3 master in collum 3

datalogAddValue(3, c4);//print value of channel 4 master in collum 4

datalogAddValue(4, batteryvoltage);//print the battery voltage in mV in collum 5

datalogAddValue(5, time1[T2]);//print the current time in collum 6

datalogDataGroupEnd();

}

//I think I use the programming port and open debug while program is running then save -> Datalog

}

}

task sound()

{

// Simple program to cycle through the file system and play all sound files

long nCycles = 0;

long nPlaying = 0;

string sFileName;

TFileExtensionTypes nFileType;

while (true)

{

++nCycles;

for (int nFileIndex = 0; nFileIndex < 64; ++nFileIndex)

{

if (!bValidFile(nFileIndex))

continue;

nFileType = getFileType(nFileIndex);

switch (nFileType)

{

case ftSound:

case ftWAV:

break;

default:

continue;

}

while (bSoundActive)

{

++nPlaying;

wait1Msec(1);

}

wait1Msec(200);

getFileName(nFileIndex, sFileName);

playSoundFile(sFileName);

}

}

}