File: C:\Users\Terry\SkyDrive\Documents\Vex\RoboBowl\RoboBowlAutoDemo.c

```
#pragma config(I2C Usage, I2C1, i2cSensors)
#pragma config(Sensor, in1,    armPotentiometer, sensorPotentiometer)
#pragma config(Sensor, dgtl1,  rightRearBumper, sensorTouch)
#pragma config(Sensor, dgtl2,  leftRearBumper, sensorTouch)
#pragma config(Sensor, dgtl3,  sonar,          sensorSONAR cm)
#pragma config(Sensor, dgtl5,  clawSwitch,     sensorTouch)
#pragma config(Sensor, dgtl6,  rightYellow,    sensorLEDtoVCC)
#pragma config(Sensor, dgtl7,  rightGreen,     sensorLEDtoVCC)
#pragma config(Sensor, dgtl8,  rightRed,       sensorLEDtoVCC)
#pragma config(Sensor, dgtl10, leftGreen,      sensorLEDtoVCC)
#pragma config(Sensor, dgtl11, leftRed,        sensorLEDtoVCC)
#pragma config(Sensor, dgtl12, leftYellow,     sensorLEDtoVCC)
#pragma config(Sensor, I2C_1,  leftEncoder,    sensorQuadEncoderOnI2CPort,    , AutoAssign )
#pragma config(Sensor, I2C_2,  rightEncoder,   sensorQuadEncoderOnI2CPort,    , AutoAssign )
#pragma config(Motor,  port1,           leftMotor,     tmotorVex393_HBridge, openLoop, reversed, driveLeft, en
#pragma config(Motor,  port6,           clawMotor,     tmotorVex393_MC29, openLoop, reversed)
#pragma config(Motor,  port7,           armMotor,      tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port10,          rightMotor,    tmotorVex393_HBridge, openLoop, driveRight, encoderPort
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

/*
This recycler robot demonstrates a fully autonomous routine
using multiple sensor inputs to perform the same task repeatedly
without driver intervention.

1) Wait until object in gripper (detect with limit switch)
2) Grip object
3) Back up to wall square
4) Turn left 90deg
5) Back to wall square
6) Drive forward to wall (stop on US)
7) Release object
8) Back to wall square
9) Turn right 90deg
10) Back to wall square
11) Forward to start position (stop on US)

Techniques demostrated:
- PID position control on the arm
- autonomous routine sequencing
- passive monitoring of encoders with LED display
```

```
- drive wheel synchronization (drive straight)
- soft start and stop on drive moves (avoid wheel slip)
- sonar object detection
- wall squaring
- measured turns

*/

#include "Utils.c"

const int drivePower = 60;
const int turnPower = 75;
const int maxArmPower = 127;
const int clawPower = 40;
const int turn90Count = 900;

const int armLow = 4000;
const int armMid = 3000;
const int armHigh = 2000;
int armTarget = armMid;
int armPos;


// PID variables are global so they're visible in the debugger.
float dT, dE;
float iE = 0;
int pidError;
int pidPower;

const float Kp = 0.6;
const float Ti = 200.0;
const float Td = 40.0;
const float iErrZone = 30.0;

task armPID()
{
  int lastE = 0;
  long lastTime = time1[T1];

  while(true) {
    sleep(5); // make sure dT is never 0.
```

```
    long now = time1[T1];
    dT = now - lastTime;
    lastTime = now;

    // Calculate current error
    armPos = SensorValue[armPotentiometer];
    pidError = armTarget - armPos;

    // Calculate integral and differential terms
    iE = iabs(pidError) > iErrZone ? 0 : iE + pidError * dT;
    dE = (pidError - lastE) / dT;
    lastE = pidError;

    // PID formula: Standard Form
    // https://en.wikipedia.org/wiki/PID_controller#Ideal_versus_standard_PID_form
    pidPower = Kp * (pidError + iE/Ti + Td*dE);

    // Apply power to motor
    motor[armMotor] = symclip(pidPower, maxArmPower);
  }
}

int leftEnc()
{
  return nMotorEncoder[leftMotor];
}

int rightEnc()
{
  return nMotorEncoder[rightMotor];
}

task wheelMonitor()
{
  while (true) {
    int lEnc = leftEnc();
    int rEnc = rightEnc();

    static int lLastEnc = 0;
    static int rLastEnc = 0;
```

```
    int lChange = lEnc - lLastEnc;
    int rChange = rEnc - rLastEnc;

    lLastEnc = lEnc;
    rLastEnc = rEnc;

    SensorValue[leftGreen] = lChange > 0;
    SensorValue[leftRed] = lChange < 0;
    SensorValue[rightGreen] = rChange > 0;
    SensorValue[rightRed] = rChange < 0;

    sleep(50);
  }
}

void waitForObjectAndGrab()
{
  while(SensorValue[clawSwitch] == false) {}
  motor[clawMotor] = clawPower;
  sleep(1000);
  motor[clawMotor] = 0;
}

void dropObject()
{
  motor[clawMotor] = -clawPower;
  sleep(1000);
  motor[clawMotor] = 0;
}

int movePower = 0;
int moveError = 0;
int lEncStart = 0;
int rEncStart = 0;
int lMovePower = 0;
int rMovePower = 0;

int lLastChange = 0;
int rLastChange = 0;

void move()
```

```
{
  int lEnc = leftEnc();
  int rEnc = rightEnc();

  int lChange = lEnc - lEncStart;
  int rChange = rEnc - rEncStart;

  lLastChange = lChange;
  rLastChange = rChange;

  moveError = lChange - rChange;

  if(vexRT[Btn8D])
  {
    // demonstrate drive sync disabled
    lMovePower = rMovePower = movePower;

    SensorValue[leftYellow] = false;
    SensorValue[rightYellow] = false;
  }
  // How to resolve error:
  // - Set one motor to full movePower and reduce the other one.
  // - Select motor to reduce using this table:
  //          error+    error-
  // power+   left      right
  // power-   right     left
  //
  // which reduces to:
  // - if the sign of (p*e) is positive, reduce the left
  // - if the sign of (p*e) is negative, reduce the right
  else if(movePower * moveError > 0)
  {
    // Left is getting ahead:
    // subtract positive error from positive power
    // or negative error from negative power
    lMovePower = movePower - moveError;
    rMovePower = movePower;

    SensorValue[leftYellow] = true;
    SensorValue[rightYellow] = false;
  }
```

```
  else
  {
    // Right is getting ahead:
    // add negative error to positive power
    // or positive error to negative power
    lMovePower = movePower;
    rMovePower = movePower + moveError;

    SensorValue[leftYellow] = false;
    SensorValue[rightYellow] = true;
  }

  motor[leftMotor] = lMovePower;
  motor[rightMotor] = rMovePower;
}

void moveStart(int power)
{
  lEncStart = leftEnc();
  rEncStart = rightEnc();

  const int softStartSteps = 5;   // number of ramp steps
  const int softStartTime = 500;  // milliseconds
  const float powerInc = power / softStartSteps;
  const int timeInc = softStartTime / softStartSteps;

  for(int i = 0; i < softStartSteps; ++i) {
    movePower = powerInc * i;
    move();
    sleep(timeInc);
  }
  movePower = power;
}

void moveEnd()
{
  const int softEndSteps = 5;   // number of ramp steps
  const int softEndTime = 250;  // milliseconds
  const int power = movePower;
  const float powerInc = power / softEndSteps;
  const int timeInc = softEndTime / softEndSteps;
```

```
  for(int i = 0; i < softEndSteps; ++i) {
    movePower = power - powerInc * i;
    move();
    sleep(timeInc);
  }
  movePower = 0;
  motor[leftMotor] = 0;
  motor[rightMotor] = 0;
  SensorValue[leftYellow] = false;
  SensorValue[rightYellow] = false;
}

bool leftBump()
{
  return SensorValue[leftRearBumper] == true;
}

bool rightBump()
{
  return SensorValue[rightRearBumper] == true;
}

void backToWall()
{
  // Drive straight until either bumper hits
  moveStart(-drivePower);
  while(!(leftBump() || rightBump()))
  {
    move();
    sleep(50);
  }

  // continue driving until both bumpers hit
  do {
    motor[leftMotor] = leftBump() ? 0 : -drivePower;
    motor[rightMotor] = rightBump() ? 0 : -drivePower;

  } while(!(leftBump() && rightBump()));

  // Ensure both motors stopped
```

```
  moveEnd();
}

unsigned short getSonar()
{
  return SensorValue[sonar];
}

void forwardToWall()
{
  moveStart(drivePower);
  while(getSonar() > 10)
  {
    move();
    sleep(50);
  }
  moveEnd();
}

void turnLeft90()
{
  // Move only right motor forward the correct distance
  long rightPos = rightEnc();
  long rightTarget = rightPos + turn90Count;
  motor[leftMotor] = 0;
  motor[rightMotor] = turnPower;
  while(rightTarget - rightEnc() > 0) {}
  motor[rightMotor] = 0;
}

void turnRight90()
{
  // Move only left motor forward the correct distance
  long leftPos = leftEnc();
  long leftTarget = leftPos + turn90Count;
  motor[leftMotor] = turnPower;
  motor[rightMotor] = 0;
  while(leftTarget - leftEnc() > 0) {}
  motor[leftMotor] = 0;
}
```

File: C:\Users\Terry\SkyDrive\Documents\Vex\RoboBowl\RoboBowlAutoDemo.c

```
task main()
{
  startTask(armPID);
  startTask(wheelMonitor);

  while(true)
  {
    armTarget = armMid;
    waitForObjectAndGrab();
    backToWall();
    turnLeft90();
    backToWall();
    armTarget = armHigh;
    forwardToWall();
    armTarget = armLow;
    sleep(500); // wait for arm to start down
    dropObject();
    backToWall();
    turnRight90();
    backToWall();
    forwardToWall();
  }
}
```