WRRF Workshop

December 10, 2012

Converting from Labview to C++,

Experience with the 2012 Beta C++ tools,

And other miscellaneous robot programming hints

Michael Smith
Jeremy Lee
Aaron Wang
John Best, jsbest@pacbell.net

Leland Robotics

The Switch to C++

- Team 604 had been using Labview since the cRio controller was introduced
- Difficult to read/understand/maintain
  - Various timeout errors not understood in 2011
  - Programmers allowed code to devolve into spaghetti
  - This is not necessarily an inherent problem with Labview – it's just us
- Team interest
- Adult team mentors were useless for Labview

C++ Boot Camp

- Interested students and mentors met for about 2 hours roughly once a week over several months
  - Students did a few elementary workshops on C
  - All installed the 2011 tools on their own machines
    - 50% first session, most of the rest in second session
  - All got basic “Simple Robot” drive code running on 2010 and 2011 robots
    - Required modifying template for Tank Drive and Victors
    - Each individual did code/test iteration, with help from others, as needed
  - A few moved on to program all functionality on 2010 and 2011 robots (shifting transmissions, ball kicker, arm/gripper, etc.)
  - One student simultaneously fixed bugs in 2011 Labview code for use at Calgames, installed beta code for C++, and ported Labview code to C++ (two different machines). Debugging took an extra 2 hours.
- Now have several students with ready to get started on 2012 task

Installation

- Gather together all the things you need
  - National Instruments (LabView) DVD
  - WindRiver DVDs (there will be two for 2012)
  - FRC updates for NI tools
  - FRC updates for WindRiver tools
  - FIRST is not completely consistent on where instructions and updates get put on the WEB. Look in
    - Installation disk readme files
    - *decibel.ni.com/content/community/first*
    - *first.wpi.edu/FRC/frccupdates.html*
    - *firstforge.wpi.edu/sf/projects/wpilib*
    - FIRST FRC mentor resources pages
    - Look on 604robotics.com after kickoff – we will update where things are
- Copy everything to USB sticks, and then to the hard drive of target computers. Its faster to install from the hard drive.

Installation (cont'd)

- Print and read the FRC README on the WindRiver disk
- Print and read the C/C++ Getting started manual
- Install from the NI/Labview disks
  - Crio Imaging tool
  - Driver's station (not required, but it is very nice to have driver's station on the same computer you are compiling and downloading from).
- Install the WindRiver disks
- Install the NI/Labview updates
- Install the WindRiver updates

Installation Hints

- Follow the instructions
  - Need to uninstall last year's tools first
  - Make sure everything in WindRiver disk gets installed
  - Licensing can get all goofed up if you don't follow the detailed instructions
- Watch out for firewall
  - We were successful leaving Windows firewall and antivirus active if we noticed and checked all the "allow" boxes that popped up.
  - If you mess this up, you will need to open your firewall advanced settings and allow the FRC/NI/Windriver stuff. This is the thing to look for if you can't connect to the robot after you are sure you have configured the network settings correctly (which requires manually setting the IP address to 10.xx.yy.5 or.6 (xx.yy is your team number).
  - It's probably better to disable firewall and antivirus during installation
- We were successful with Windows XP, Windows 7 32bit, Windows 7 64 bit, Macbooks running Windows under BootCamp, and (maybe) a Macbook running Windows in a virtual machine.

Other Getting Started Hints

- Only ethernet Port 1 on the cRio will work. The other is for the camera and they are not connected together.
  - New small cRio has one ethernet port; camera attached to router
- The DIP switches on the cRio should be OFF, except Console Out, which should be ON
  - We had problems reliably connecting to the cRio with the 2012 code unless Console Out was on. (likely was just module mess up – see below)
  - No DIP switches in new cRio; set via imaging tool.
- The slot assignments for the i/o modules are different for 2012
  - You MUST put the modules in the correct slots, or nothing will work. Error messages may be cryptic (likely fixed by kickoff).
  - Slot 1: analog module
  - Slot 2: digital module
  - Slot 3: solenoid module
- An Estop button is not required (good). The driver station spacebar is the new Estop button and cannot be changed (beware). Enter key is the disable button.

Debugging C++ Code

- Follow the "Getting Started with C++.." manual to configure VxWorks. It is easy to do if you follow the instructions, screen by screen. Its impossible to guess how to do it.
- The cRio must be reset to clear code from memory before loading new code.
  - Cycle the robot power, OR
  - In the Remote Systems window, right click on VxWorks6x…. , and click on "Reset Connected Target"
- Deploying code (writing to non-volatile memory in the cRio so that it will run when you power up) is done by going to the FIRST/deploy menu after building. You must undeploy old code before either downloading code for debugging or deploying new code.

C/C++ Getting Started Guide

Worcester Polytechnic Institute Robotics Resource Center

Brad Miller, Ken Streeter, Beth Finn, Jerry Morrison, Dan Jones, Ryan O'Meara, Derek White, Stephanie Hoag

Rev 3.0

October 7, 2011

Further C++ Beta Test Comments

- All of the C++ code that we tried prior to the beta test while getting ready to move from LabView to C++ worked without modification with the 2012 libraries
- There is new programming model that is described in the "WPILib Robot Programming Cookbook" that is new for this year
  - Robot commands are defined and programmed separately from the subsystems on the robot which execute the elements of the commands
  - The provides a cleaner model for complex code, particularly if developed by multiple programmers
  - The value is less clear for a typical robot implementation that is relatively simple
  - We did not implement code using this model on our robot

Elementary C

```
#include <stdio.h>

int main()
{
  printf(“hello, world\n”);
  return 0;
}
```

Variables in C

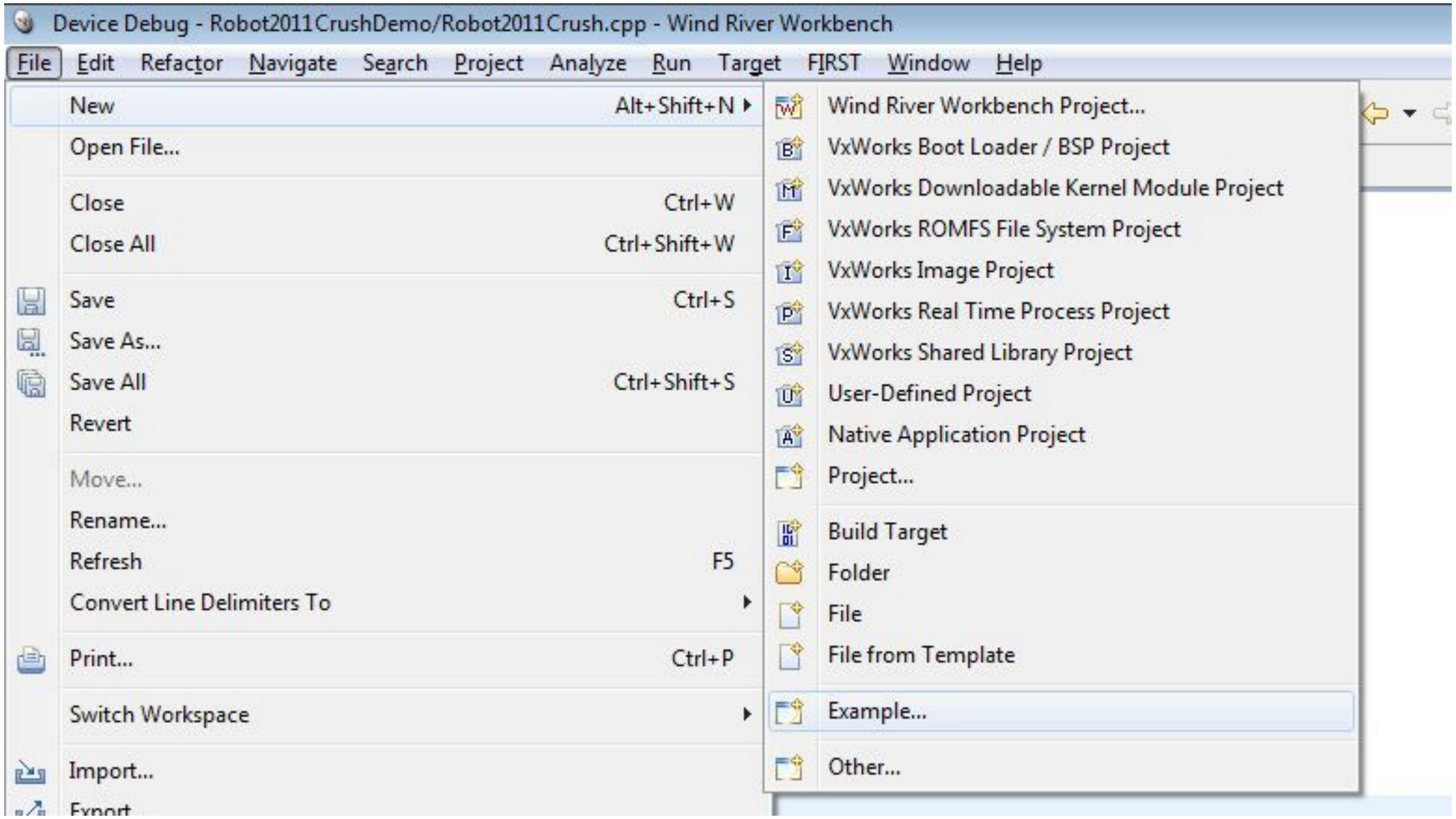
```
#include <stdio.h>

int a;

float findcirc(float dia)
{
  return 3.14159265 * dia;
}

int main()
{
  float c;

  a = 4;
  c = findcirc(a);
  printf(“circumference is %f\n, c”);
  return 0;
}
```

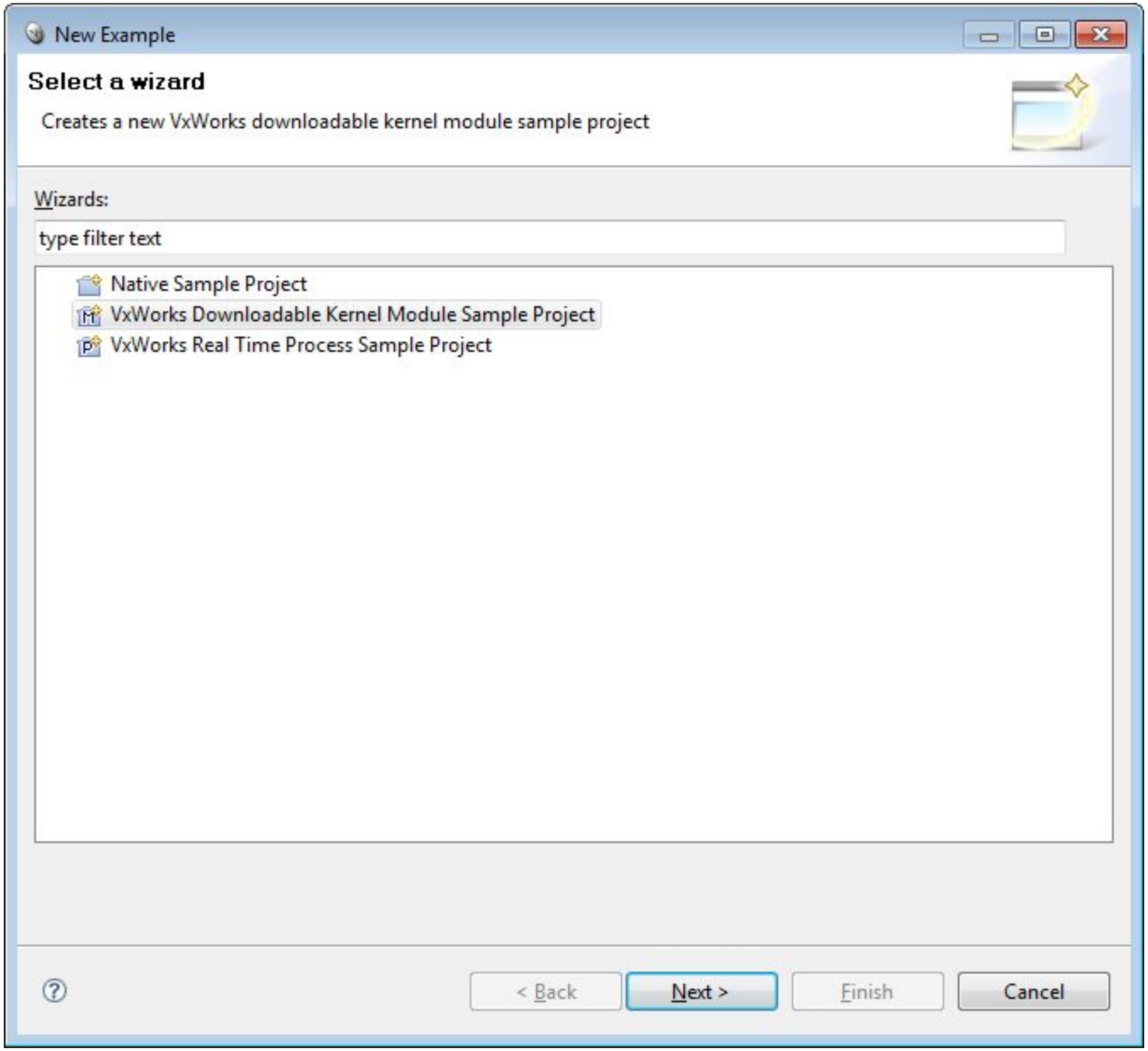
Classes in C++

- A class is a user defined type that can represent a complicated object
  - Includes multiple variables to represent the state of the object
  - Includes functions to define the behavior of the object
  - Classes can be based on a previously defined class
    - Replace stuff you don't' like
    - Add new capability

```
Class Date {
public:
  Date(int y, int m, int d);
  void add_day(int n);
  int month();
  //…
Private:
  int y, m, d;
};

Date mydate;
```

Example: SimpleRobot Modified


Device Debug - Robot2011CrushDemo/Robot2011Crush.cpp - Wind River Workbench


New Example

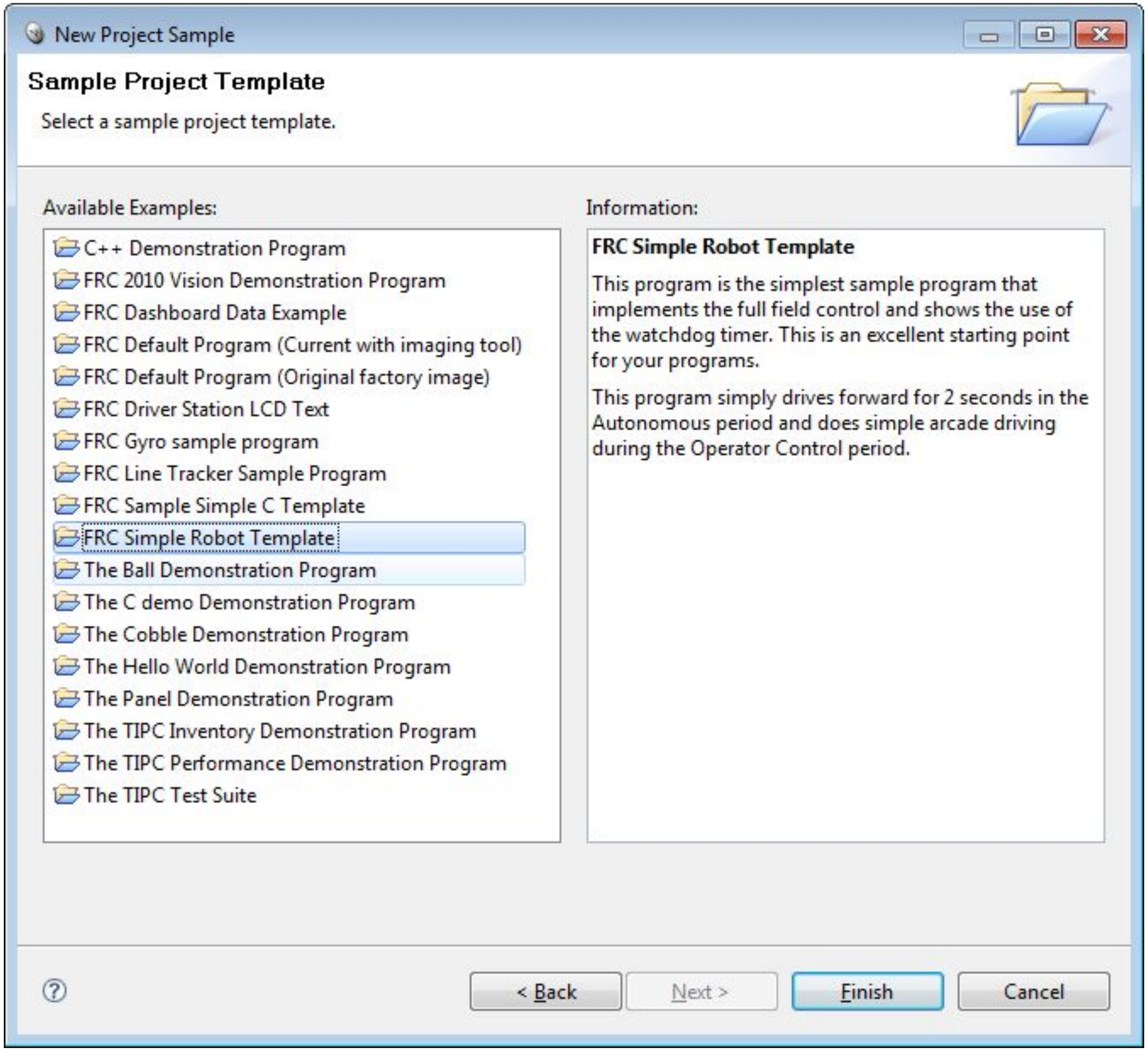
Select a wizard

Creates a new VxWorks downloadable kernel module sample project

Wizards:

type filter text

- Native Sample Project
- VxWorks Downloadable Kernel Module Sample Project
- VxWorks Real Time Process Sample Project



New Project Sample

Sample Project Template

Select a sample project template.

Available Examples:

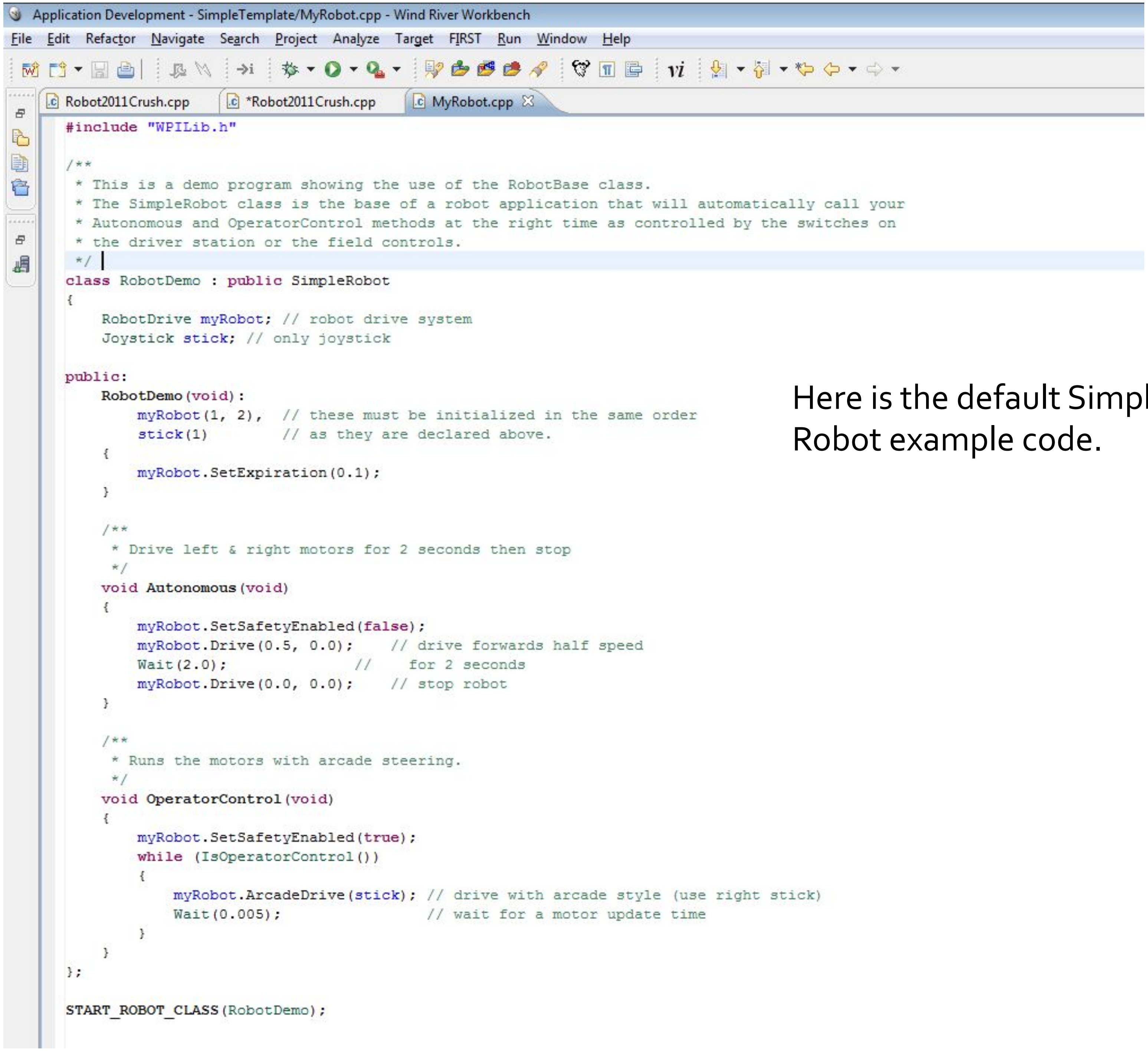
- C++ Demonstration Program
- FRC 2010 Vision Demonstration Program
- FRC Dashboard Data Example
- FRC Default Program (Current with imaging tool)
- FRC Default Program (Original factory image)
- FRC Driver Station LCD Text
- FRC Gyro sample program
- FRC Line Tracker Sample Program
- FRC Sample Simple C Template
- FRC Simple Robot Template
- The Ball Demonstration Program
- The C demo Demonstration Program
- The Cobble Demonstration Program
- The Hello World Demonstration Program
- The Panel Demonstration Program
- The TIPC Inventory Demonstration Program
- The TIPC Performance Demonstration Program
- The TIPC Test Suite

Information:

FRC Simple Robot Template

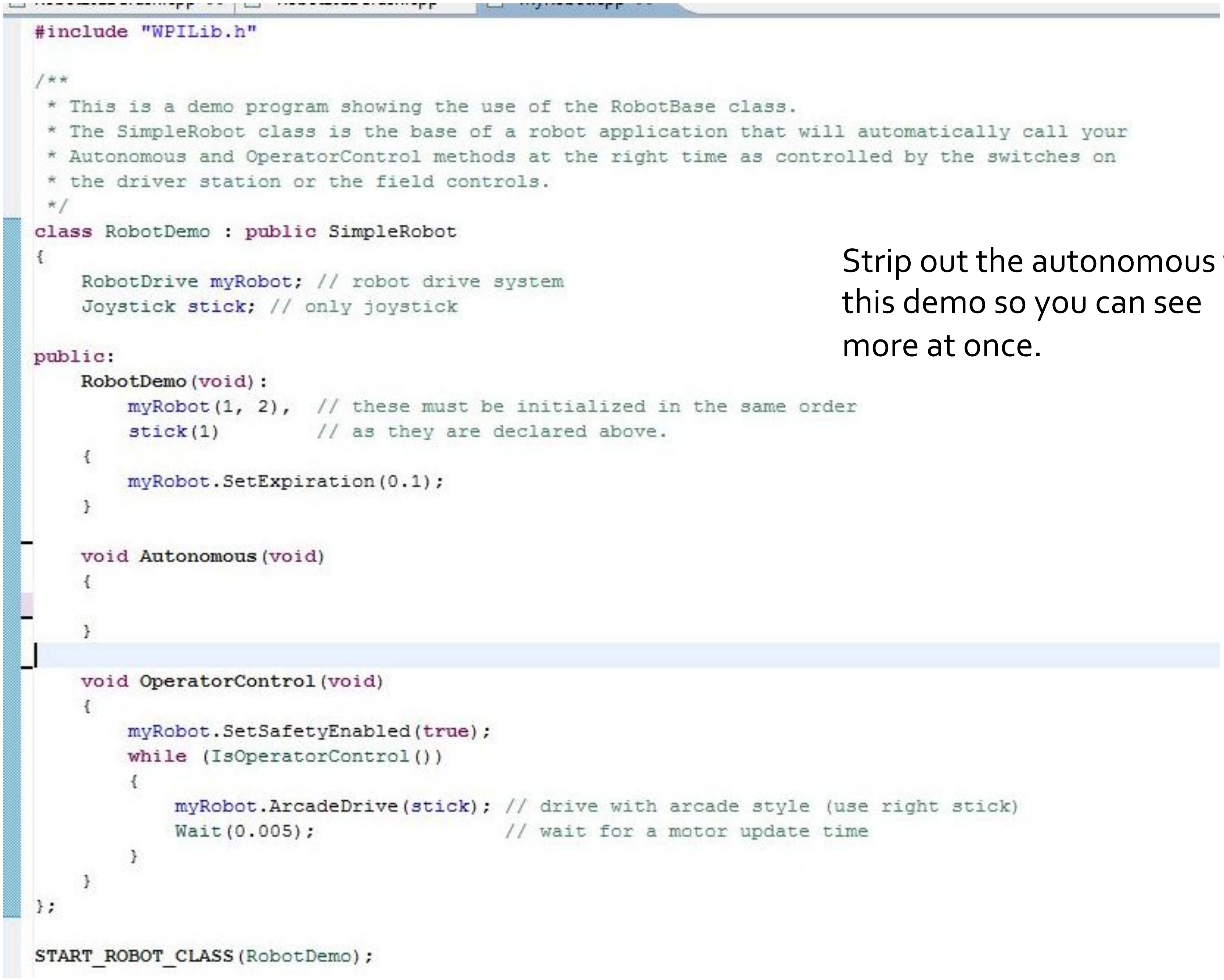
This program is the simplest sample program that implements the full field control and shows the use of the watchdog timer. This is an excellent starting point for your programs.

This program simply drives forward for 2 seconds in the Autonomous period and does simple arcade driving during the Operator Control period.


Application Development - SimpleTemplate/MyRobot.cpp - Wind River Workbench
File Edit Refactor Navigate Search Project Analyze Target FIRST Run Window Help
Robot2011Crush.cpp
*Robot2011Crush.cpp
MyRobot.cpp

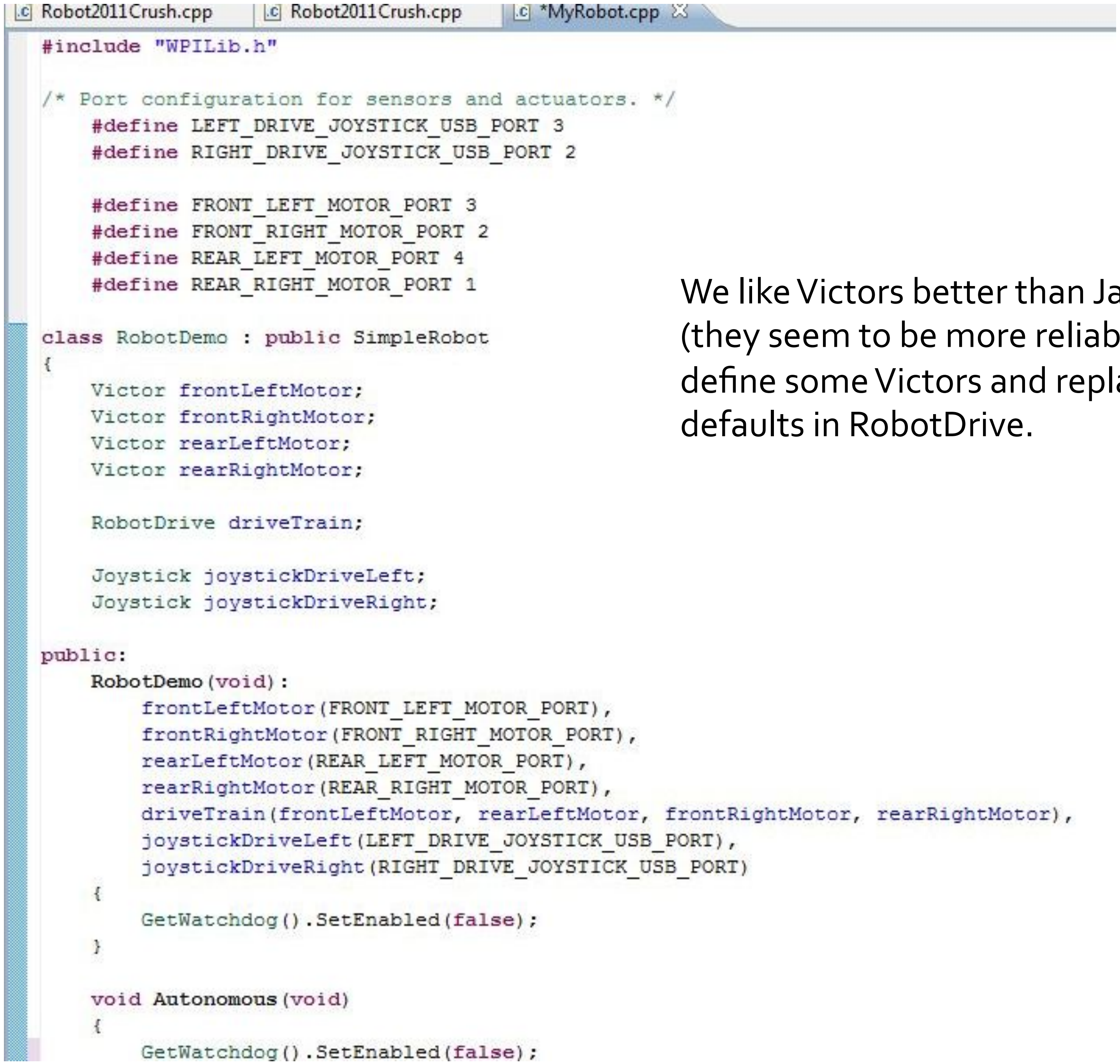
```cpp
#include "WPILib.h"

/**
 * This is a demo program showing the use of the RobotBase class.
 * The SimpleRobot class is the base of a robot application that will automatically call your
 * Autonomous and OperatorControl methods at the right time as controlled by the switches on
 * the driver station or the field controls.
 */
class RobotDemo : public SimpleRobot
{
    RobotDrive myRobot; // robot drive system
    Joystick stick; // only joystick

public:
    RobotDemo(void):
        myRobot(1, 2),  // these must be initialized in the same order
        stick(1)        // as they are declared above.
    {
        myRobot.SetExpiration(0.1);
    }

    /**
     * Drive left & right motors for 2 seconds then stop
     */
    void Autonomous(void)
    {
        myRobot.SetSafetyEnabled(false);
        myRobot.Drive(0.5, 0.0);    // drive forwards half speed
        Wait(2.0);                 //    for 2 seconds
        myRobot.Drive(0.0, 0.0);    // stop robot
    }

    /**
     * Runs the motors with arcade steering.
     */
    void OperatorControl(void)
    {
        myRobot.SetSafetyEnabled(true);
        while (IsOperatorControl())
        {
            myRobot.ArcadeDrive(stick); // drive with arcade style (use right stick)
            Wait(0.005);                // wait for a motor update time
        }
    }
};

START_ROBOT_CLASS(RobotDemo);
```

Here is the default Simple Robot example code.



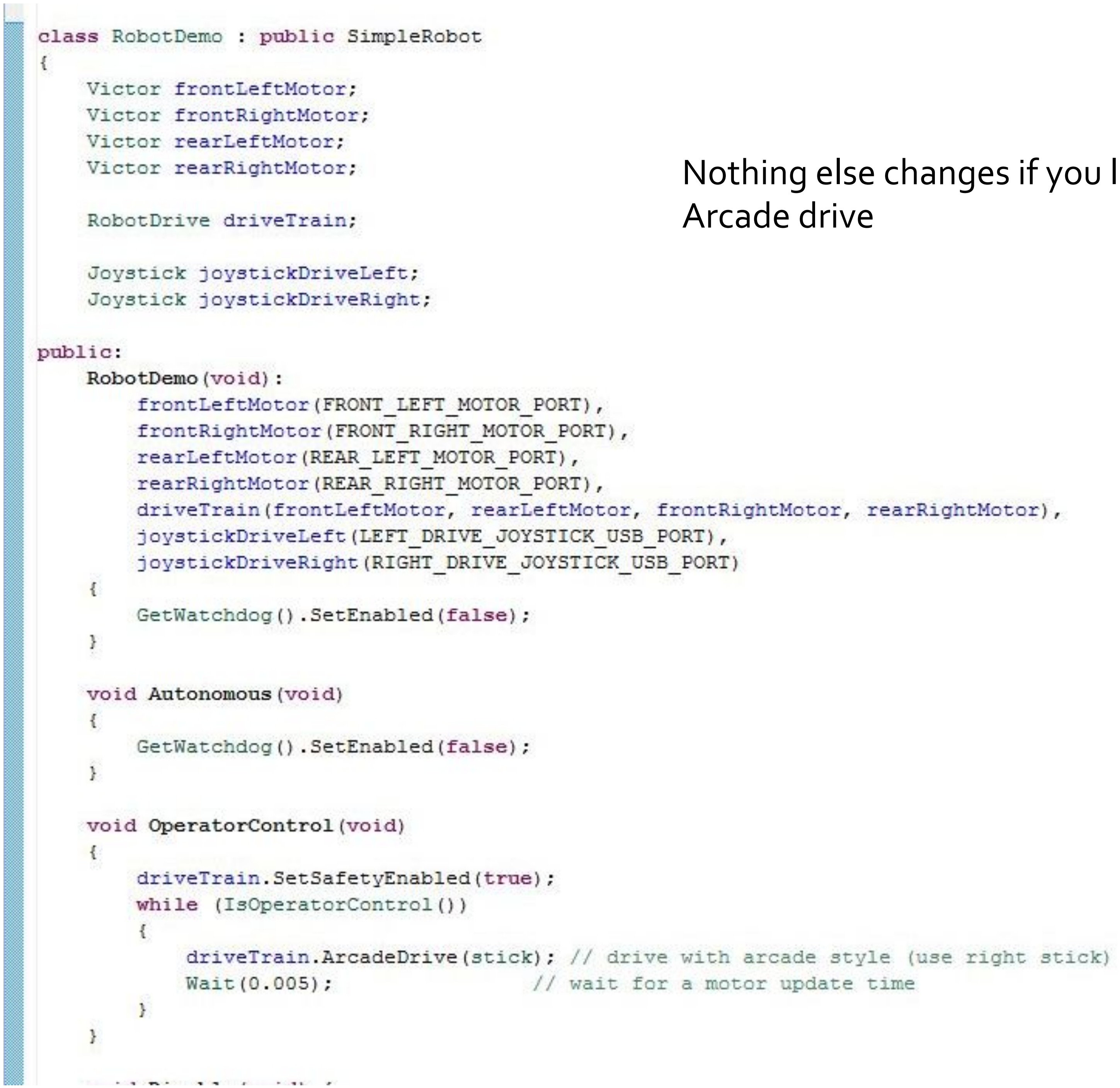
```
#include "WPILib.h"

/**
 * This is a demo program showing the use of the RobotBase class.
 * The SimpleRobot class is the base of a robot application that will automatically call your
 * Autonomous and OperatorControl methods at the right time as controlled by the switches on
 * the driver station or the field controls.
 */
class RobotDemo : public SimpleRobot
{
    RobotDrive myRobot; // robot drive system
    Joystick stick; // only joystick

public:
    RobotDemo(void):
        myRobot(1, 2),  // these must be initialized in the same order
        stick(1)        // as they are declared above.
    {
        myRobot.SetExpiration(0.1);
    }

    void Autonomous(void)
    {

    }

    void OperatorControl(void)
    {
        myRobot.SetSafetyEnabled(true);
        while (IsOperatorControl())
        {
            myRobot.ArcadeDrive(stick); // drive with arcade style (use right stick)
            Wait(0.005);                // wait for a motor update time
        }
    }
};

START_ROBOT_CLASS(RobotDemo);
```

Strip out the autonomous for this demo so you can see more at once.


Robot2011Crush.cpp | Robot2011Crush.cpp | *MyRobot.cpp

```
#include "WPILib.h"

/* Port configuration for sensors and actuators. */
    #define LEFT_DRIVE_JOYSTICK_USB_PORT 3
    #define RIGHT_DRIVE_JOYSTICK_USB_PORT 2

    #define FRONT_LEFT_MOTOR_PORT 3
    #define FRONT_RIGHT_MOTOR_PORT 2
    #define REAR_LEFT_MOTOR_PORT 4
    #define REAR_RIGHT_MOTOR_PORT 1

class RobotDemo : public SimpleRobot
{
    Victor frontLeftMotor;
    Victor frontRightMotor;
    Victor rearLeftMotor;
    Victor rearRightMotor;

    RobotDrive driveTrain;

    Joystick joystickDriveLeft;
    Joystick joystickDriveRight;

public:
    RobotDemo(void):
        frontLeftMotor(FRONT_LEFT_MOTOR_PORT),
        frontRightMotor(FRONT_RIGHT_MOTOR_PORT),
        rearLeftMotor(REAR_LEFT_MOTOR_PORT),
        rearRightMotor(REAR_RIGHT_MOTOR_PORT),
        driveTrain(frontLeftMotor, rearLeftMotor, frontRightMotor, rearRightMotor),
        joystickDriveLeft(LEFT_DRIVE_JOYSTICK_USB_PORT),
        joystickDriveRight(RIGHT_DRIVE_JOYSTICK_USB_PORT)
    {
        GetWatchdog().SetEnabled(false);
    }

    void Autonomous(void)
    {
        GetWatchdog().SetEnabled(false);
```

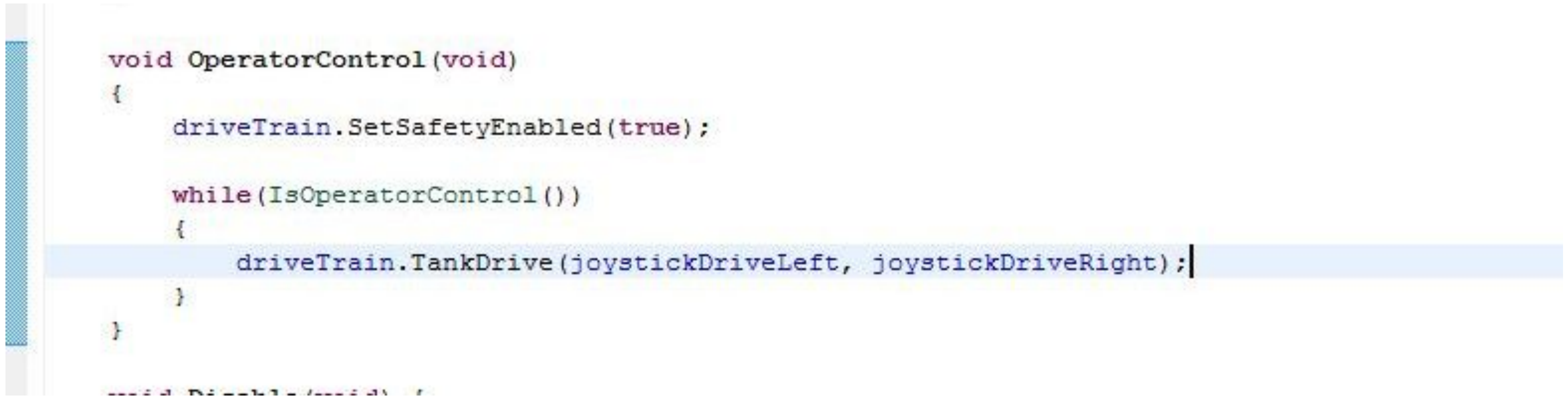
We like Victors better than Jaguars (they seem to be more reliable), so define some Victors and replace defaults in RobotDrive.



```
class RobotDemo : public SimpleRobot
{
    Victor frontLeftMotor;
    Victor frontRightMotor;
    Victor rearLeftMotor;
    Victor rearRightMotor;

    RobotDrive driveTrain;

    Joystick joystickDriveLeft;
    Joystick joystickDriveRight;

public:
    RobotDemo(void):
        frontLeftMotor(FRONT_LEFT_MOTOR_PORT),
        frontRightMotor(FRONT_RIGHT_MOTOR_PORT),
        rearLeftMotor(REAR_LEFT_MOTOR_PORT),
        rearRightMotor(REAR_RIGHT_MOTOR_PORT),
        driveTrain(frontLeftMotor, rearLeftMotor, frontRightMotor, rearRightMotor),
        joystickDriveLeft(LEFT_DRIVE_JOYSTICK_USB_PORT),
        joystickDriveRight(RIGHT_DRIVE_JOYSTICK_USB_PORT)
    {
        GetWatchdog().SetEnabled(false);
    }

    void Autonomous(void)
    {
        GetWatchdog().SetEnabled(false);
    }

    void OperatorControl(void)
    {
        driveTrain.SetSafetyEnabled(true);
        while (IsOperatorControl())
        {
            driveTrain.ArcadeDrive(stick); // drive with arcade style (use right stick)
            Wait(0.005);                // wait for a motor update time
        }
    }
```

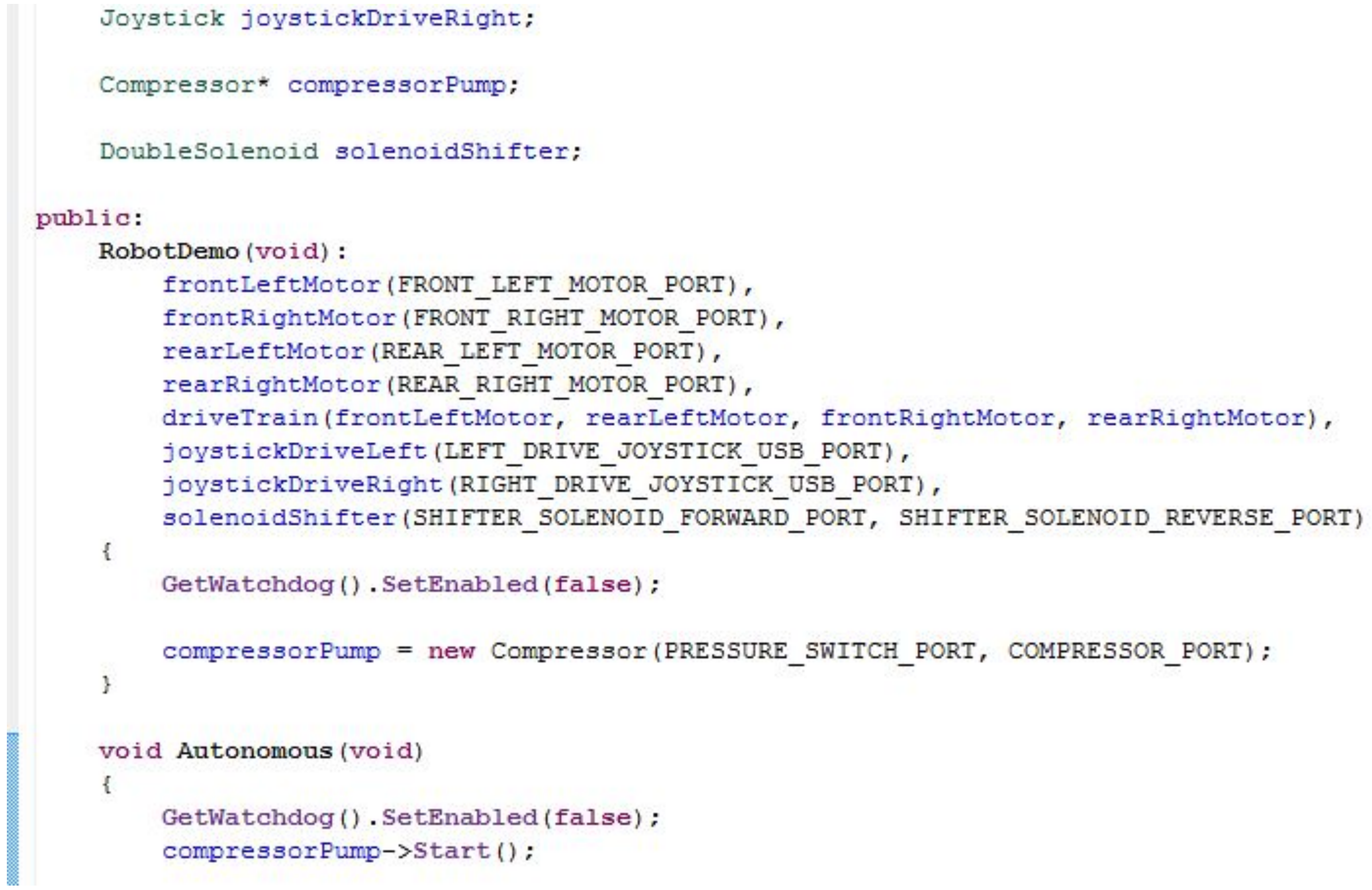
Nothing else changes if you like Arcade drive

Change one line of code to convert to Tank Drive



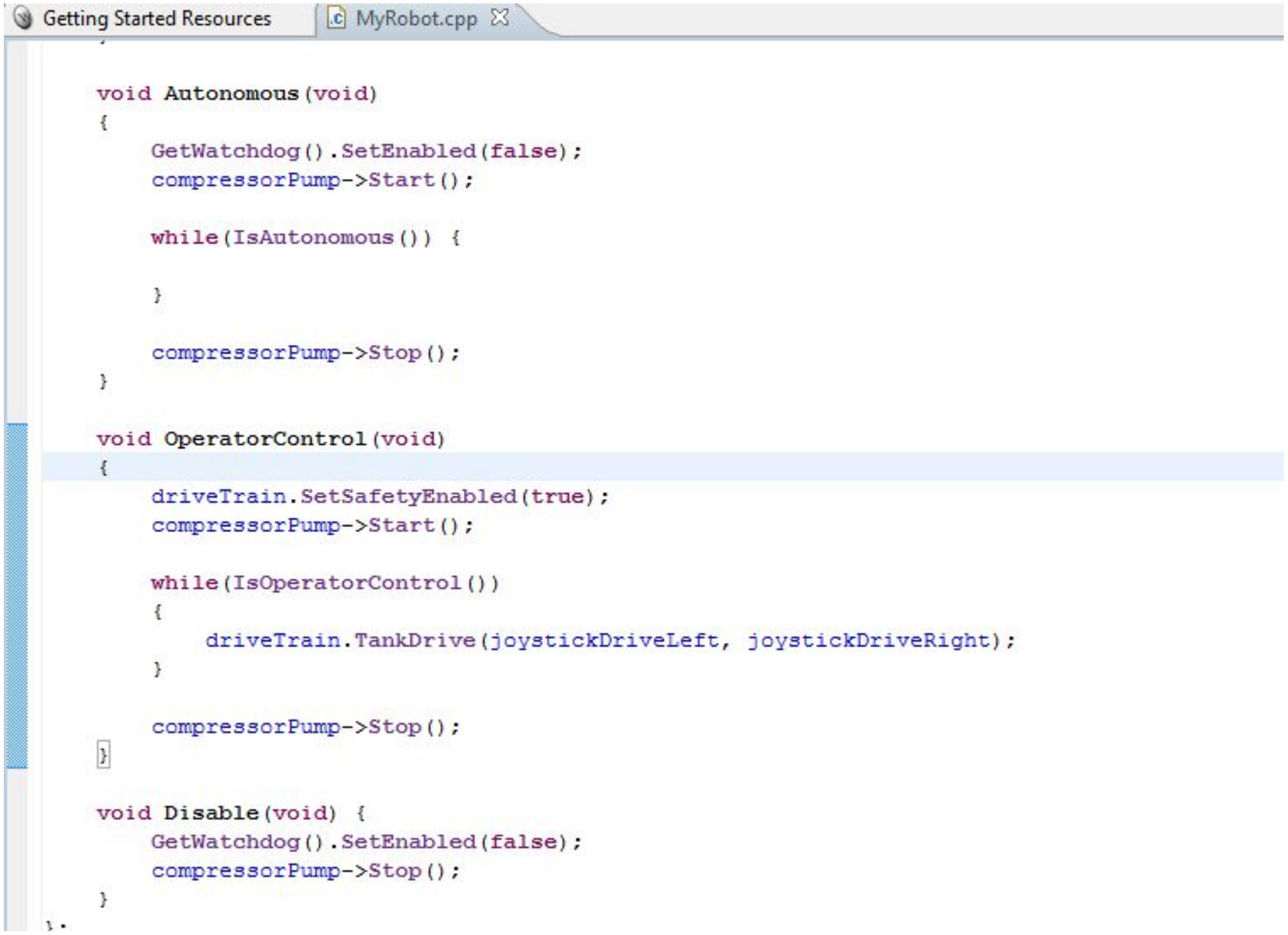
```
void OperatorControl(void)
{
    driveTrain.SetSafetyEnabled(true);

    while(IsOperatorControl())
    {
        driveTrain.TankDrive(joystickDriveLeft, joystickDriveRight);
    }
}
```

Two speed transmission required pneumatics, so define the compressor, and a solenoid valve to actuator the shifter. Initialize it in the Robot constructor

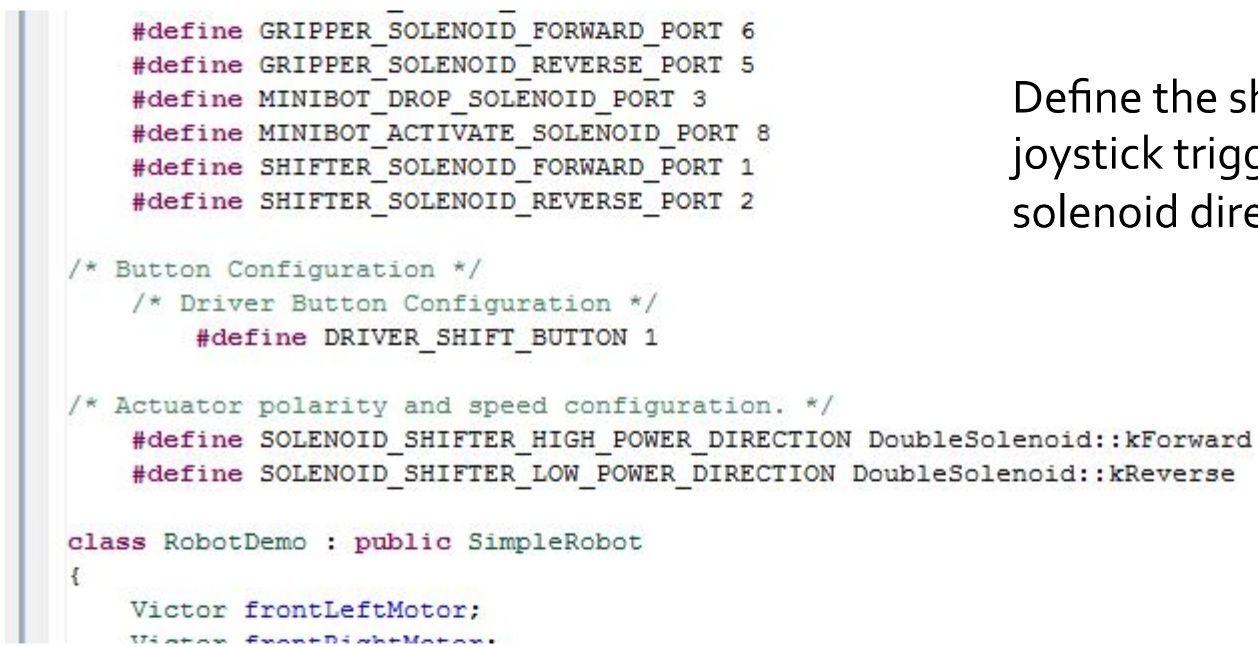


```
    Joystick joystickDriveRight;

    Compressor* compressorPump;

    DoubleSolenoid solenoidShifter;

public:
    RobotDemo(void):
        frontLeftMotor(FRONT_LEFT_MOTOR_PORT),
        frontRightMotor(FRONT_RIGHT_MOTOR_PORT),
        rearLeftMotor(REAR_LEFT_MOTOR_PORT),
        rearRightMotor(REAR_RIGHT_MOTOR_PORT),
        driveTrain(frontLeftMotor, rearLeftMotor, frontRightMotor, rearRightMotor),
        joystickDriveLeft(LEFT_DRIVE_JOYSTICK_USB_PORT),
        joystickDriveRight(RIGHT_DRIVE_JOYSTICK_USB_PORT),
        solenoidShifter(SHIFTER_SOLENOID_FORWARD_PORT, SHIFTER_SOLENOID_REVERSE_PORT)
    {
        GetWatchdog().SetEnabled(false);

        compressorPump = new Compressor(PRESSURE_SWITCH_PORT, COMPRESSOR_PORT);
    }

    void Autonomous(void)
    {
        GetWatchdog().SetEnabled(false);
        compressorPump->Start();
```

Start the compressor on enabling the robot. The library compressor object spawns a separate background task that reads the pressure switch and controls the compressor. Turn it off on entering disable mode.

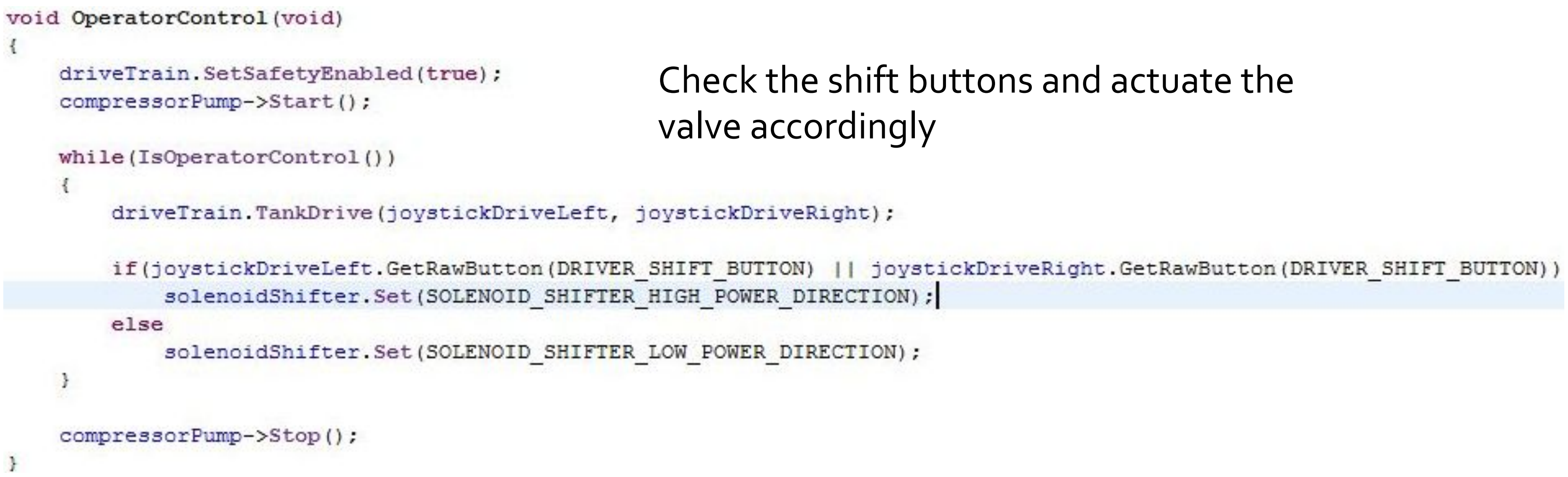


```
    void Autonomous(void)
    {
        GetWatchdog().SetEnabled(false);
        compressorPump->Start();

        while(IsAutonomous()) {

        }

        compressorPump->Stop();
    }

    void OperatorControl(void)
    {
        driveTrain.SetSafetyEnabled(true);
        compressorPump->Start();

        while(IsOperatorControl())
        {
            driveTrain.TankDrive(joystickDriveLeft, joystickDriveRight);
        }

        compressorPump->Stop();
    }

    void Disable(void) {
        GetWatchdog().SetEnabled(false);
        compressorPump->Stop();
    }
};
```



```
    #define GRIPPER_SOLENOID_FORWARD_PORT 6
    #define GRIPPER_SOLENOID_REVERSE_PORT 5
    #define MINIBOT_DROP_SOLENOID_PORT 3
    #define MINIBOT_ACTIVATE_SOLENOID_PORT 8
    #define SHIFTER_SOLENOID_FORWARD_PORT 1
    #define SHIFTER_SOLENOID_REVERSE_PORT 2

/* Button Configuration */
    /* Driver Button Configuration */
        #define DRIVER_SHIFT_BUTTON 1

/* Actuator polarity and speed configuration. */
    #define SOLENOID_SHIFTER_HIGH_POWER_DIRECTION DoubleSolenoid::kForward
    #define SOLENOID_SHIFTER_LOW_POWER_DIRECTION DoubleSolenoid::kReverse

class RobotDemo : public SimpleRobot
{
    Victor frontLeftMotor;
    Victor frontRightMotor;
```

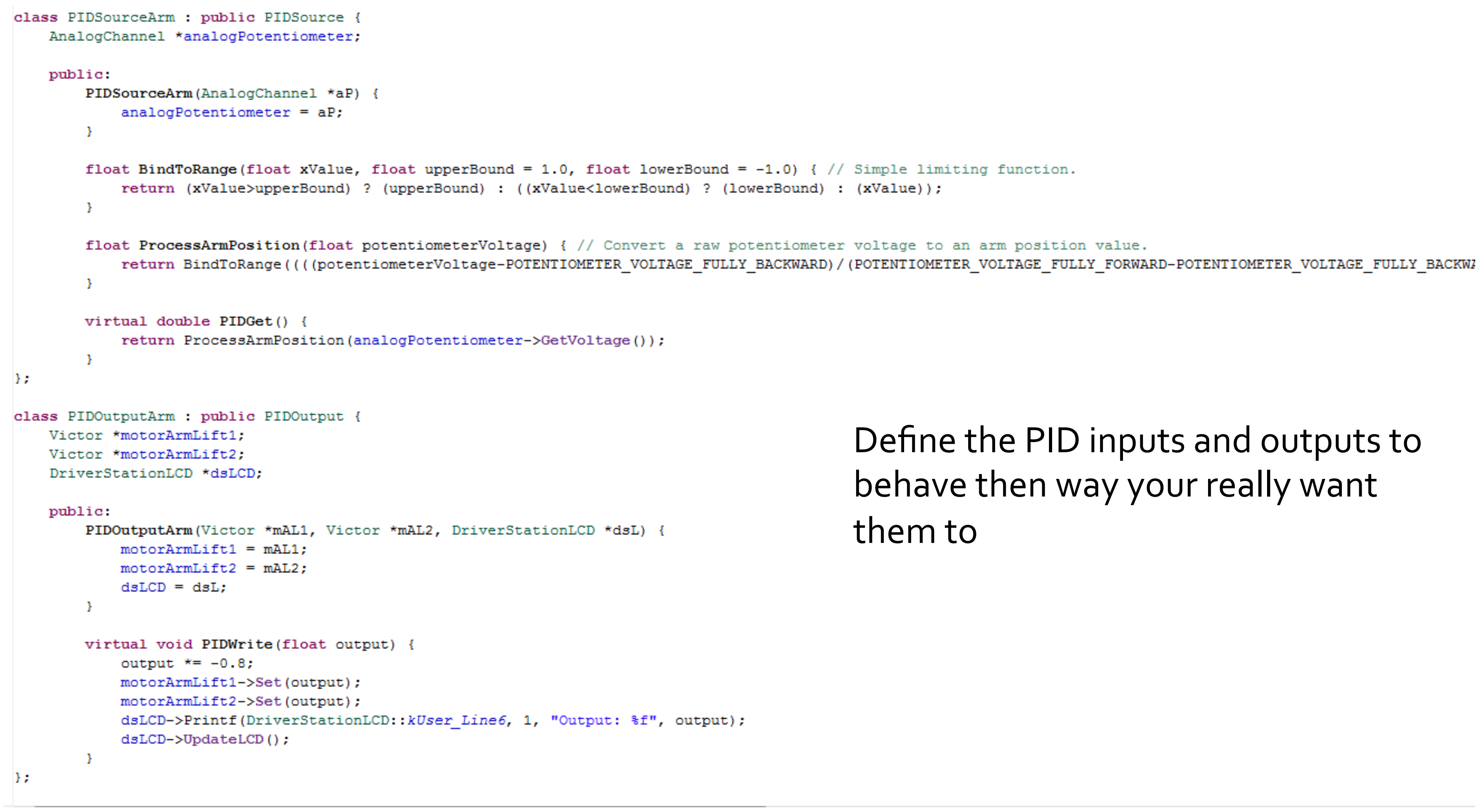
Define the shift buttons (tank drive joystick trigger), and the shifter solenoid direction



```
void OperatorControl(void)
{
    driveTrain.SetSafetyEnabled(true);
    compressorPump->Start();

    while(IsOperatorControl())
    {
        driveTrain.TankDrive(joystickDriveLeft, joystickDriveRight);

        if(joystickDriveLeft.GetRawButton(DRIVER_SHIFT_BUTTON) || joystickDriveRight.GetRawButton(DRIVER_SHIFT_BUTTON))
            solenoidShifter.Set(SOLENOID_SHIFTER_HIGH_POWER_DIRECTION);
        else
            solenoidShifter.Set(SOLENOID_SHIFTER_LOW_POWER_DIRECTION);
    }

    compressorPump->Stop();
}
```

Check the shift buttons and actuate the valve accordingly

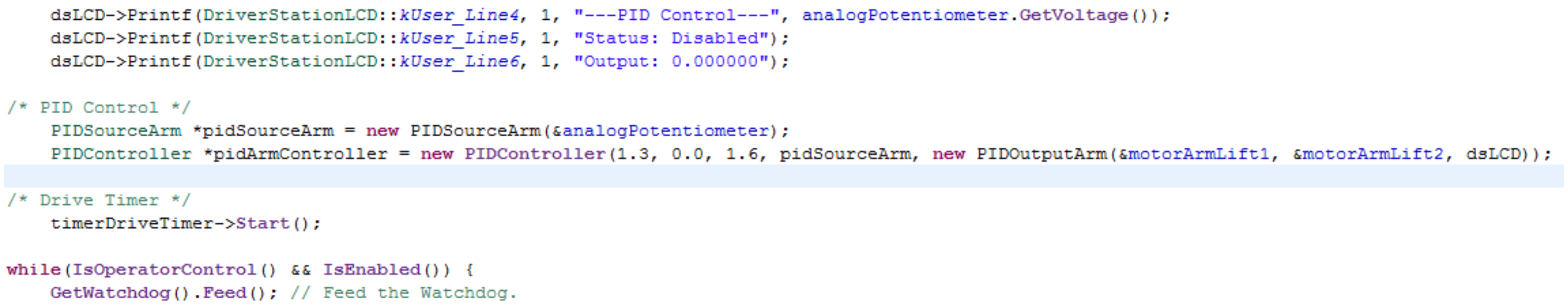
PID Arm Elevation Control



```
class PIDSourceArm : public PIDSource {
    AnalogChannel *analogPotentiometer;

    public:
        PIDSourceArm(AnalogChannel *aP) {
            analogPotentiometer = aP;
        }

        float BindToRange(float xValue, float upperBound = 1.0, float lowerBound = -1.0) { // Simple limiting function.
            return (xValue>upperBound) ? (upperBound) : ((xValue<lowerBound) ? (lowerBound) : (xValue));
        }

        float ProcessArmPosition(float potentiometerVoltage) { // Convert a raw potentiometer voltage to an arm position value.
            return BindToRange((((potentiometerVoltage-POTENTIOMETER_VOLTAGE_FULLY_BACKWARD)/(POTENTIOMETER_VOLTAGE_FULLY_FORWARD-POTENTIOMETER_VOLTAGE_FULLY_BACKW
        }

        virtual double PIDGet() {
            return ProcessArmPosition(analogPotentiometer->GetVoltage());
        }
};

class PIDOutputArm : public PIDOutput {
    Victor *motorArmLift1;
    Victor *motorArmLift2;
    DriverStationLCD *dsLCD;

    public:
        PIDOutputArm(Victor *mAL1, Victor *mAL2, DriverStationLCD *dsL) {
            motorArmLift1 = mAL1;
            motorArmLift2 = mAL2;
            dsLCD = dsL;
        }

        virtual void PIDWrite(float output) {
            output *= -0.8;
            motorArmLift1->Set(output);
            motorArmLift2->Set(output);
            dsLCD->Printf(DriverStationLCD::kUser_Line6, 1, "Output: %f", output);
            dsLCD->UpdateLCD();
        }
};
```

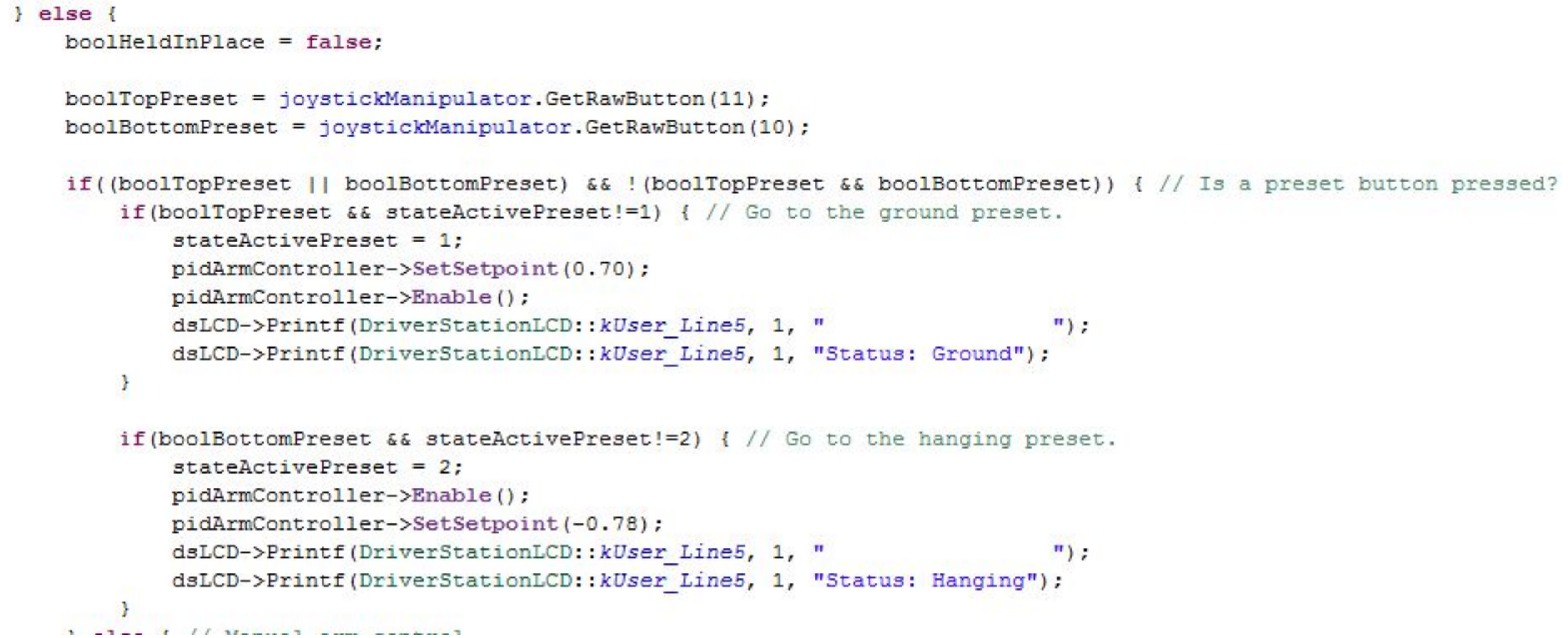
Define the PID inputs and outputs to behave then way your really want them to

Initialize the PID. The real processing takes place in a separate thread at evenly spaced time intervals

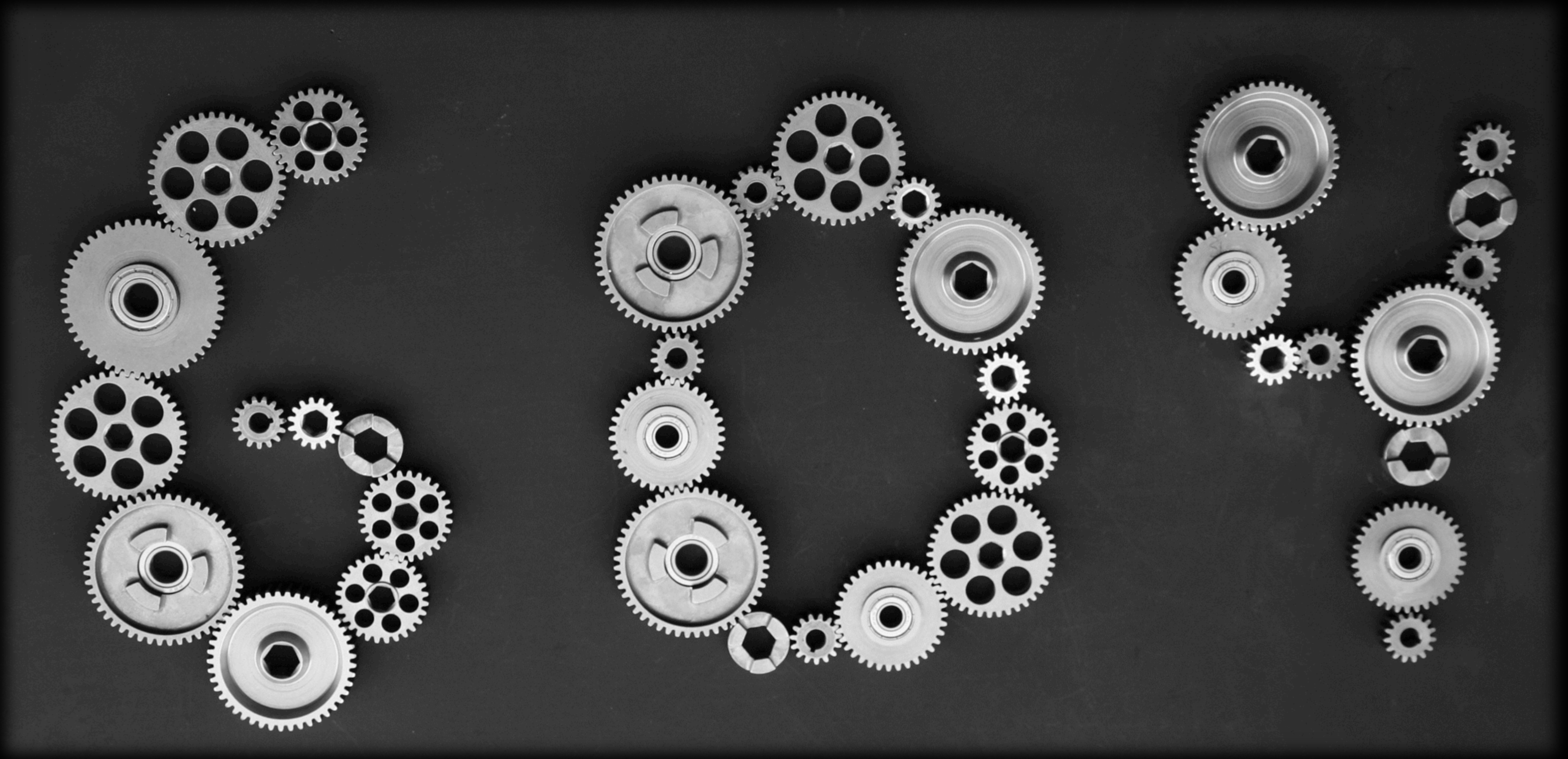


```
    dsLCD->Printf(DriverStationLCD::kUser_Line4, 1, "---PID Control---", analogPotentiometer.GetVoltage());
    dsLCD->Printf(DriverStationLCD::kUser_Line5, 1, "Status: Disabled");
    dsLCD->Printf(DriverStationLCD::kUser_Line6, 1, "Output: 0.000000");

/* PID Control */
    PIDSourceArm *pidSourceArm = new PIDSourceArm(&analogPotentiometer);
    PIDController *pidArmController = new PIDController(1.3, 0.0, 1.6, pidSourceArm, new PIDOutputArm(&motorArmLift1, &motorArmLift2, dsLCD));

/* Drive Timer */
    timerDriveTimer->Start();

while(IsOperatorControl() && IsEnabled()) {
    GetWatchdog().Feed(); // Feed the Watchdog.
```

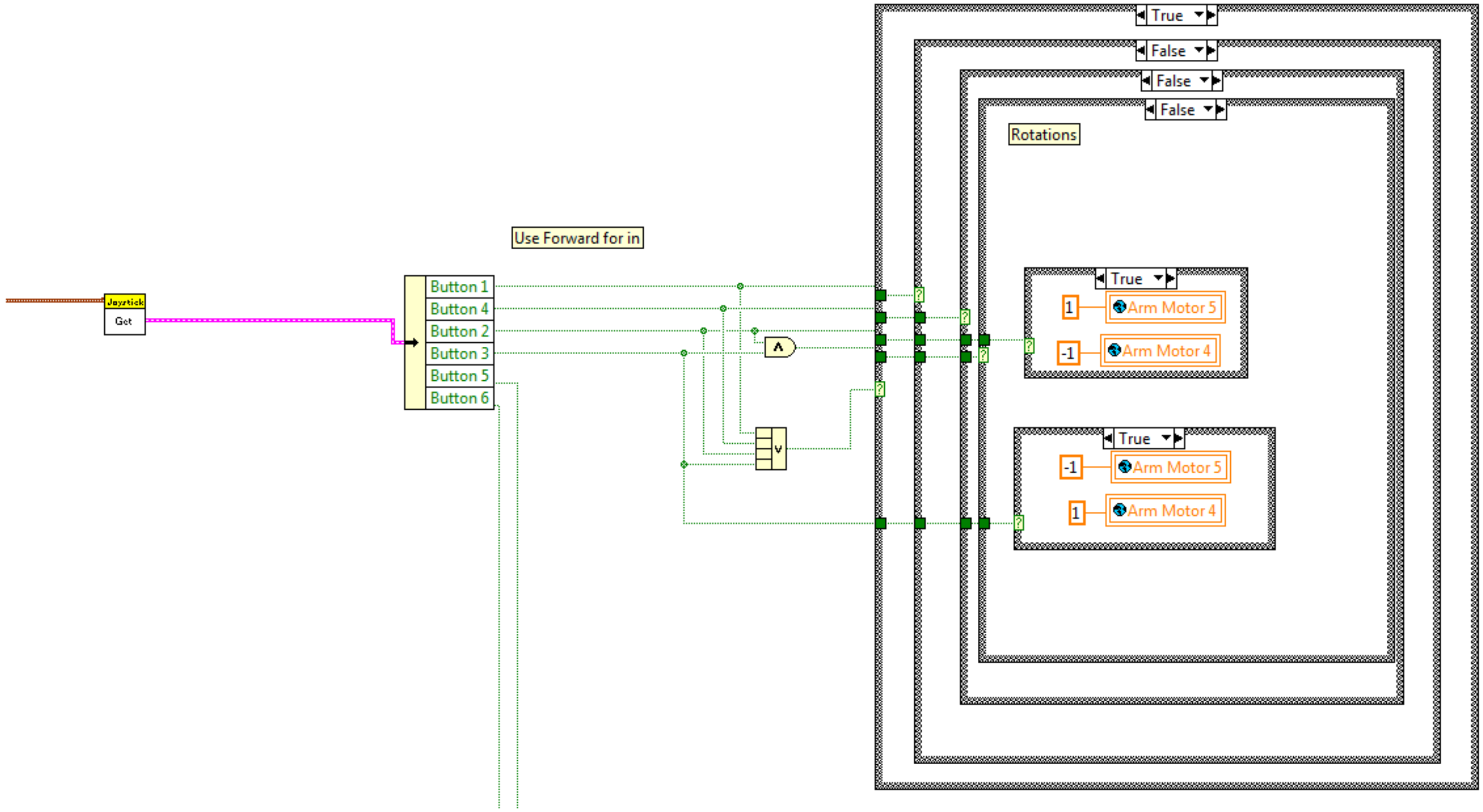
Example: moving arm to preset positions



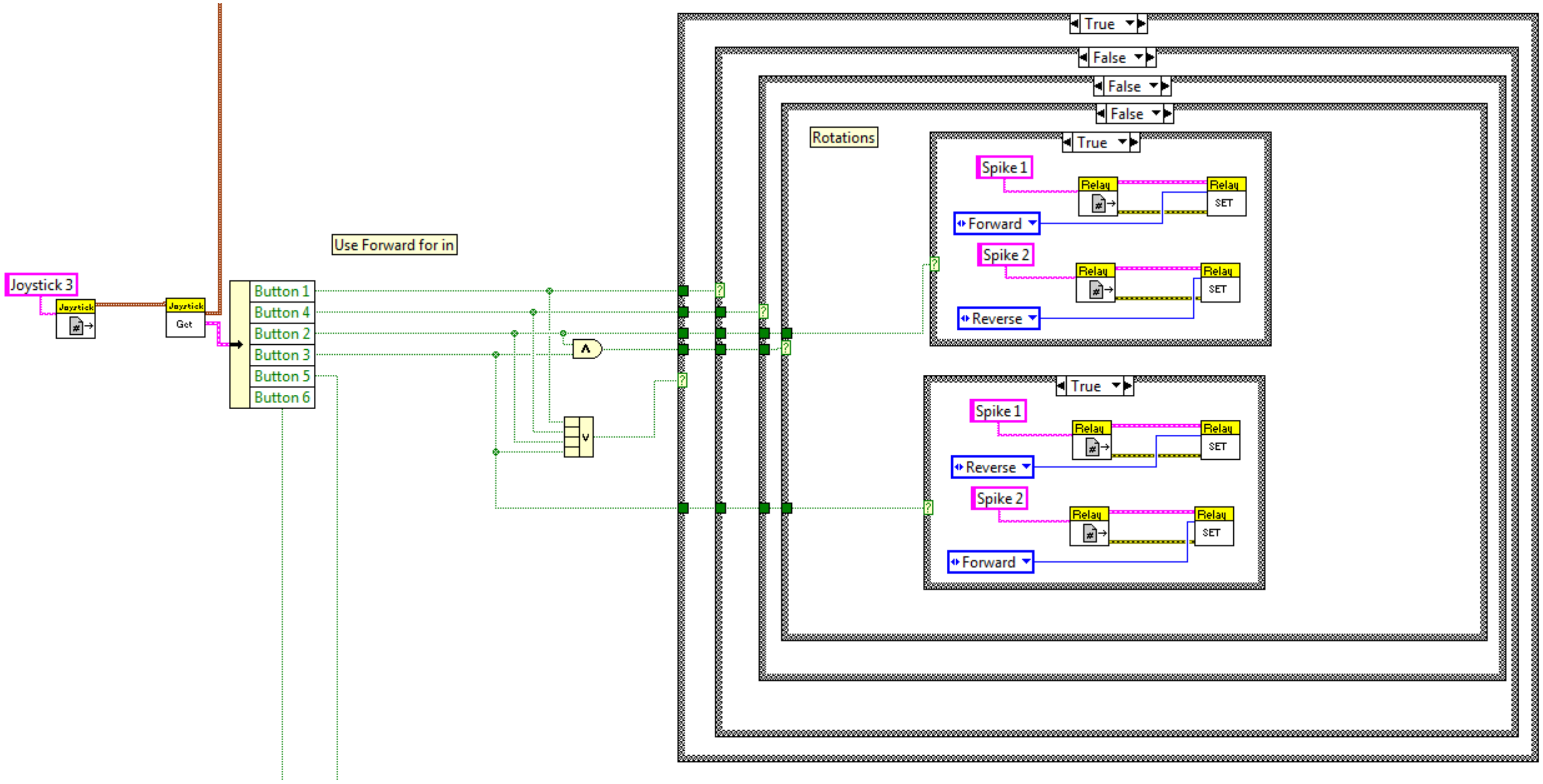
```
 } else {
     boolHeldInPlace = false;

     boolTopPreset = joystickManipulator.GetRawButton(11);
     boolBottomPreset = joystickManipulator.GetRawButton(10);

     if((boolTopPreset || boolBottomPreset) && !(boolTopPreset && boolBottomPreset)) { // Is a preset button pressed?
          if(boolTopPreset && stateActivePreset!=1) { // Go to the ground preset.
               stateActivePreset = 1;
               pidArmController->SetSetpoint(0.70);
               pidArmController->Enable();
               dsLCD->Printf(DriverStationLCD::kUser_Line5, 1, "                    ");
               dsLCD->Printf(DriverStationLCD::kUser_Line5, 1, "Status: Ground");
          }

          if(boolBottomPreset && stateActivePreset!=2) { // Go to the hanging preset.
               stateActivePreset = 2;
               pidArmController->Enable();
               pidArmController->SetSetpoint(-0.78);
               dsLCD->Printf(DriverStationLCD::kUser_Line5, 1, "                    ");
               dsLCD->Printf(DriverStationLCD::kUser_Line5, 1, "Status: Hanging");
          }
     } else { // Manual arm control
```



Labview Gripper Code

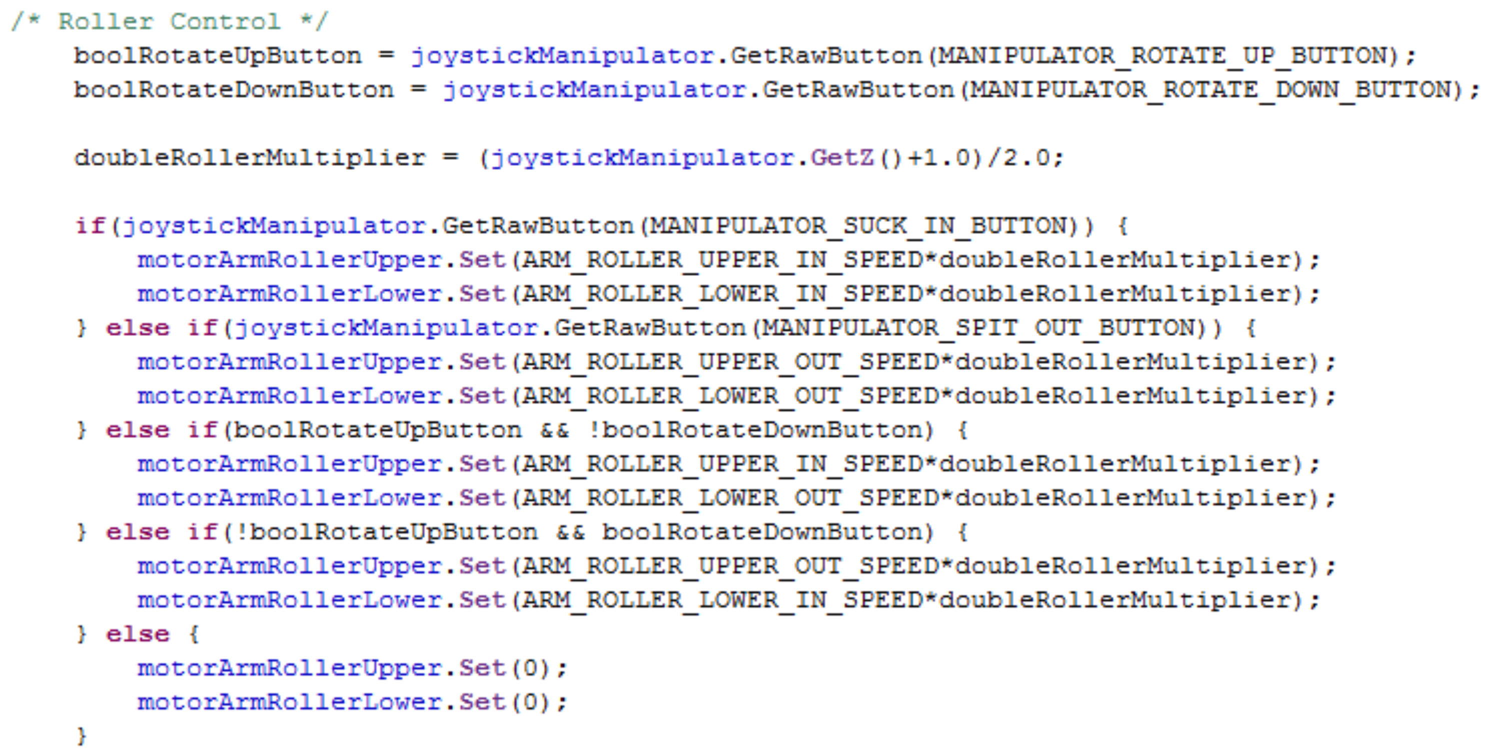


Labview Gripper Alternate



C++ Gripper Code

Sometimes words are easier to understand than pictures



```cpp
/* Roller Control */
    boolRotateUpButton = joystickManipulator.GetRawButton(MANIPULATOR_ROTATE_UP_BUTTON);
    boolRotateDownButton = joystickManipulator.GetRawButton(MANIPULATOR_ROTATE_DOWN_BUTTON);

    doubleRollerMultiplier = (joystickManipulator.GetZ()+1.0)/2.0;

    if(joystickManipulator.GetRawButton(MANIPULATOR_SUCK_IN_BUTTON)) {
        motorArmRollerUpper.Set(ARM_ROLLER_UPPER_IN_SPEED*doubleRollerMultiplier);
        motorArmRollerLower.Set(ARM_ROLLER_LOWER_IN_SPEED*doubleRollerMultiplier);
    } else if(joystickManipulator.GetRawButton(MANIPULATOR_SPIT_OUT_BUTTON)) {
        motorArmRollerUpper.Set(ARM_ROLLER_UPPER_OUT_SPEED*doubleRollerMultiplier);
        motorArmRollerLower.Set(ARM_ROLLER_LOWER_OUT_SPEED*doubleRollerMultiplier);
    } else if(boolRotateUpButton && !boolRotateDownButton) {
        motorArmRollerUpper.Set(ARM_ROLLER_UPPER_IN_SPEED*doubleRollerMultiplier);
        motorArmRollerLower.Set(ARM_ROLLER_LOWER_OUT_SPEED*doubleRollerMultiplier);
    } else if(!boolRotateUpButton && boolRotateDownButton) {
        motorArmRollerUpper.Set(ARM_ROLLER_UPPER_OUT_SPEED*doubleRollerMultiplier);
        motorArmRollerLower.Set(ARM_ROLLER_LOWER_IN_SPEED*doubleRollerMultiplier);
    } else {
        motorArmRollerUpper.Set(0);
        motorArmRollerLower.Set(0);
    }
```