

Using limit switches to control behavior

Using limit switches to control behavior

Limit switches are often used to control mechanisms on robots. While limit switches are simple to use, they only can sense a single position of a moving part. This makes them ideal for ensuring that movement doesn't exceed some limit but not so good at controlling the speed of the movement as it approaches the limit. For example, a rotational shoulder joint on a robot arm would best be controlled using a potentiometer or an absolute encoder, the limit switch could make sure that if the potentiometer ever failed, the limit switch would stop the robot from going too far and causing damage.

What values are provided by the limit switch



Limit switches can have "normally opened" or "normally closed" outputs. The usual way of wiring the switch is between a digital input signal connection and ground. The digital input has pull-up resistors that will make the input be high (1 value) when the switch is open, but when the switch closes the value goes to 0 since the input is now connected to ground. The switch shown here has both normally open and normally closed outputs.

Using limit switches to control behavior

Polling waiting for a switch to close

C++

```
#include "RobotTemplate.h"
#include "WPILib.h"

RobotTemplate::RobotTemplate()
{
    DigitalInput* limitSwitch;
}

void RobotTemplate::RobotInit()
{
    limitSwitch = new DigitalInput(1);
}

void RobotTemplate::operatorControl()
{
    while(limitSwitch->Get())
    {
        Wait(10);
    }
}
```

Java

```
import edu.wpi.first.wpilibj.DigitalInput;
import edu.wpi.first.wpilibj.SampleRobot;
import edu.wpi.first.wpilibj.Timer;

public class RobotTemplate extends SampleRobot {

    DigitalInput limitSwitch;

    public void robotInit() {
```

Using limit switches to control behavior

```
        limitSwitch = new DigitalInput(1);
    }

    public void operatorControl() {
        // more code here
        while (limitSwitch.get()) {
            Timer.delay(10);
        }
    }
}
```

You can write a very simple piece of code that just reads the limit switch over and over again waiting until it detects that its value transitions from 1 (opened) to 0 (closed). While this works, it's usually impractical for the program to be able to just wait for the switch to operate and not be doing anything else, like responding to joystick input. This example shows the fundamental use of the switch, but while the program is waiting, nothing else is happening.

Command-based program to operate until limit switch closed

C++

```
#include "ArmUp.h"

ArmUp::ArmUp()
{

}

void ArmUp::Initialize()
{
    arm.ArmUp();
}

void ArmUp::Execute()
{
}
```

Using limit switches to control behavior

```
void ArmUp::IsFinished()
{
    return arm.isSwitchSet();
}

void ArmUp::End()
{
    arm.ArmStop();
}

void ArmUp::Interrupted()
{
    End();
}
```

Java

```
package edu.wpi.first.wpilibj.templates.commands;

public class ArmUp extends CommandBase {
    public ArmUp() {
    }

    protected void initialize() {
        arm.armUp();
    }

    protected void execute() {
    }

    protected boolean isFinished() {
        return arm.isSwitchSet();
    }

    protected void end() {
        arm.armStop();
    }
}
```

Using limit switches to control behavior

```
    }

    protected void interrupted() {
        end();
    }
}
```

Commands call their `execute()` and `isFinished()` methods about 50 times per second, or at a rate of every 20ms. A command that will operate a motor until the limit switch is closed can read the digital input value in the `isFinished()` method and return true when the switch changes to the correct state. Then the command can stop the motor.

Remember, the mechanism (an Arm in this case) has some inertia and won't stop immediately so it's important to make sure things don't break while the arm is slowing.

Using a counter to detect the closing of the switch

C++

```
#include "WPILIB.h"
#include "Arm.h"

DigitalInput* limitSwitch;
SpeedController* armMotor;
Counter* counter;

Arm::Arm()
{
    limitSwitch = new DigitalInput(1);
    armMotor = new Victor(1);
    counter = new Counter(limitSwitch);
}

bool Arm::IsSwitchSet()
{
    return counter->Get() > 0;
}
```

Using limit switches to control behavior

```
}

void Arm::InitializeCounter()
{
    counter->Reset();
}

void Arm::ArmUp()
{
    armMotor->Set(.5);
}

void Arm::ArmDown()
{
    armMotor->Set(-0.5);
}

void Arm::ArmStop()
{
    armMotor->Set(0);
}

void InitDefaultCommand()
{
}
```

Java

```
package edu.wpi.first.wpilibj.templates.subsystems;
import edu.wpi.first.wpilibj.Counter;
import edu.wpi.first.wpilibj.DigitalInput;
import edu.wpi.first.wpilibj.SpeedController;
import edu.wpi.first.wpilibj.Victor;
import edu.wpi.first.wpilibj.command.Subsystem;
public class Arm extends Subsystem {

    DigitalInput limitSwitch = new DigitalInput(1);
```

Using limit switches to control behavior

```
SpeedController armMotor = new Victor(1);
Counter counter = new Counter(limitSwitch);

public boolean isSwitchSet() {
    return counter.get() > 0;
}

public void initializeCounter() {
    counter.reset();
}

public void armUp() {
    armMotor.set(0.5);
}

public void armDown() {
    armMotor.set(-0.5);
}

public void armStop() {
    armMotor.set(0.0);
}

protected void initDefaultCommand() {
}
}
```

It's possible that a limit switch might close then open again as a mechanism moves past the switch. If the closure is fast enough the program might not notice that the switch closed. An alternative method of catching the switch closing is use a Counter object. Since counters are implemented in hardware, it will be able to capture the closing of the fastest switches and increment it's count. Then the program can simply notice that the count has increased and take whatever steps are needed to do the operation.

Above is a subsystem that uses a counter to watch the limit switch and wait for the value to change. When it does, the counter will increment and that can be watched in a command.

Using limit switches to control behavior

Create a command that uses the counter to detect switch closing

C++

```
#include "ArmUp.h"

ArmUp::ArmUp()
{
}

void ArmUp::Initialize()
{
    arm.InitializeCounter();
    arm.ArmUp();
}

void ArmUp::Execute()
{
}

bool ArmUp::IsFinished()
{
    return arm->IsSwitchSet();
}

void ArmUp::End()
{
    arm->ArmStop();
}

void ArmUp::Interrupted()
{
    End();
}
```


Using limit switches to control behavior

Java

```
package edu.wpi.first.wpilibj.templates.commands;

public class ArmUp extends CommandBase {

    public ArmUp() {
    }

    protected void initialize() {
        arm.initializeCounter();
        arm.armUp();
    }

    protected void execute() {
    }

    protected boolean isFinished() {
        return arm.isSwitchSet();
    }

    protected void end() {
        arm.armStop();
    }

    protected void interrupted() {
        end();
    }
}
```

This command initializes the counter in the above subsystem then starts the motor moving. It then tests the counter value in the `isFinished()` method waiting for it to count the limit switch changing. When it does, the arm is stopped. By using a hardware counter, a switch that might close then open very quickly can still be caught by the program.