

Jiroskop (Gyroscope) sensörü ivmeölçere (accelerometer) benzer ama aralarında büyük bir fark vardır:

İvmeölçer cihazın ivmesini ölçerken, jiroskop 3 koordinatın (X, Y, Z) dönüş hızını ölçmektedir.

İvmeölçerde tek bir koordinat üzerinden ivme ölçülürken jiroskopta üç koordinata göre dönüş hızı ölçümü yapılır. Bunu yaparken Vector3 isimli bir sınıftan yararlanılır. Bu sınıf X, Y, Z koordinatlarına sahip olan nesneyi üzerinde 3D işlemler yaptırmak üzere hazırlar. Örnekleme gerekirse, **Vector3.Zero** X, Y ve Z koordinatlarına "0" değerini atar.

AppBar içerisinde bulunan On/Off düğmesi ile jiroskop özelliğini açıp kapatabiliyoruz. On/Off düğmesinin bağlı olduğu **ApplicationBarIconButton\_Click** event handler'ı içerisinde bu işlemin nasıl gerçekleştirildiğini görebilirsiniz.

On/Off düğmesini kullanarak jiroskopu aktifleştirdiğimizde, değerlerde değişiklik olma anında tetiklenecek **gyroscope\_CurrentValueChanged** aksiyonunu da hazırlamış oluyoruz. Bu aksiyonun içinde **isDataValid** değişkenine true/false (Gyroscope'dan değerlerin gelme anında problem çıkmadığı sürece true değeri gelecektir) değeri atanıyor.

Önemli olan, yani kullanıcıya geri bildirimde bulunacak kısım ise **timer\_Tick** event'idir. Bu event sayfanın çalışma anında (Constructor içinde tanımlaması yapılmıştır) Timer nesnesi ile devreye alınır. Belli periyotlarda (tanımlandığı yerde 30 milisaniye olarak belirtilmiştir) jiroskopdan aldığı bilgiyi ekrana yazdırır.

## C#

```
using System;
```

```
using System.Windows;
```

```
using Microsoft.Phone.Controls;
```

```
using Microsoft.Devices.Sensors;
```

```
using Microsoft.Xna.Framework;
```

```
using System.Windows.Threading;
```

```
namespace sdkRawSensorDataCS
```

```
{
```

```
    public partial class Gyroscope Page : Phone Application Page
```

```
    {
```

```
        Gyroscope gyroscope;
```

```
        DispatcherTimer timer;
```

```
        Vector3 currentRotationRate = Vector3.Zero;
```

```
        Vector3 cumulativeRotation = Vector3.Zero;
```

```
        DateTimeOffset lastUpdateTime = DateTimeOffset.MinValue;
```

```
        bool isDataValid;
```

```

// Constructor
public GyroscopePage()
{
    InitializeComponent();
    Application.Current.Host.Settings.EnableFrameRateCounter = false;
    if (!Gyroscope.IsSupported)
    {
        // Yine öncelikle destek var mı diye bakıyoruz:

        statusTextBlock.Text = "Araç su terazisi özelliğini desteklememektedir";
        ApplicationBar.IsVisible = false;
    }
    else
    {
        // destek varsa yine timerımızı başlatıyoruz

        timer = new DispatcherTimer();
        timer.Interval = TimeSpan.FromMilliseconds(60);
        timer.Tick += new EventHandler(timer_Tick);
    }
}

private void ApplicationBarIconButton_Click(object sender, EventArgs e)
{
    if (gyroscope != null && gyroscope.IsDataValid)
    {
        // Nesne hazırsa ve kullanılıyorsa durdurmak için burayı kullanıyoruz

        gyroscope.Stop();
        timer.Stop();
        statusTextBlock.Text = "gyroscope durduruldu.";
    }
    else
    {
        if (gyroscope == null)
        {
            // ilk buton clicklendiğinde yine instance alıyoruz
            gyroscope = new Gyroscope();
        }
    }
}

```

```

//Altaki işlemler önceki örneklerle birebir aynı:
gyroscope.TimeBetweenUpdates = TimeSpan.FromMilliseconds(20);
timeBetweenUpdatesTextBlock.Text = "Güncellenme aralığı: " +
gyroscope.TimeBetweenUpdates.TotalMilliseconds + " ms";
gyroscope.CurrentValueChanged +=
newEventHandler<SensorReadingEventArgs<GyroscopeReading>>(gyroscope_CurrentValueChanged)
;

}
try
{
    statusTextBlock.Text = "Gyroscope başlatıldı.";
    gyroscope.Start();
    timer.Start();
}
catch (InvalidOperationException)
{
    statusTextBlock.Text = "Hata oluştu.";
}
}
}

```

```

void gyroscope_CurrentValueChanged(object sender,
SensorReadingEventArgs<GyroscopeReading> e)
{
    //data alabildiğimizi teyit ediyoruz

    isValid = gyroscope.IsValid;
    if (lastUpdateTime.Equals(DateTimeOffset.MinValue))
    {
        //ilk defa bu evente geliyorsak öncelikle sadece güncelleme zamanını alıyoruz
        lastUpdateTime = e.SensorReading.Timestamp;
    }
    else
    {
        // sensordan bilgileri alıp ilgili değişkene atıyoruz. yine burada vector olarak bu bilgiyi okumak
        bizim işimizi kolaylaştırıyor.
    }
}

```

```

currentRotationRate = e.SensorReading.RotationRate;

// işlem başladıktan sonraki hangi zamanda güncelleme yapıldığı bilgisini alıyoruz
TimeSpan timeSinceLastUpdate = e.SensorReading.Timestamp - lastUpdateTime;

// kümülatif dönüş değerini hesaplıyoruz
cumulativeRotation += currentRotationRate * (float)(timeSinceLastUpdate.TotalSeconds);

//son güncellemeyi tekrar set ediyoruz. Timestamp son okunma tarihini almamızı sağlayan bir
zaman niteliğidir.
lastUpdateTime = e.SensorReading.Timestamp;
}
}
void timer_Tick(object sender, EventArgs e)
{
    if (isDataValid)
    {
        statusTextBlock.Text = "Gyroscope değerleri okuyor.";
    }

    //okumuş olduğum bilgileri yine ekrandaki textblock ve vektörlere basarak görsel olarak
    kullanıcıya gösteriyoruz

```

```

currentXTextBlock.Text = currentRotationRate.X.ToString("0.000");
currentYTextBlock.Text = currentRotationRate.Y.ToString("0.000");
currentZTextBlock.Text = currentRotationRate.Z.ToString("0.000");
cumulativeXTextBlock.Text = MathHelper.ToDegrees(cumulativeRotation.X).ToString("0.00");
cumulativeYTextBlock.Text = MathHelper.ToDegrees(cumulativeRotation.Y).ToString("0.00");
cumulativeZTextBlock.Text = MathHelper.ToDegrees(cumulativeRotation.Z).ToString("0.00");
double centerX = cumulativeGrid.ActualWidth / 2.0;
double centerY = cumulativeGrid.ActualHeight / 2.0;
currentXLine.X2 = centerX + currentRotationRate.X * 100;
currentYLine.X2 = centerX + currentRotationRate.Y * 100;
currentZLine.X2 = centerX + currentRotationRate.Z * 100;
cumulativeXLine.X2 = centerX - centerY * Math.Sin(cumulativeRotation.X);
cumulativeXLine.Y2 = centerY - centerY * Math.Cos(cumulativeRotation.X);
cumulativeYLine.X2 = centerX - centerY * Math.Sin(cumulativeRotation.Y);
cumulativeYLine.Y2 = centerY - centerY * Math.Cos(cumulativeRotation.Y);

```

```

        cumulativeZLine.X2 = centerX - centerY * Math.Sin(cumulativeRotation.Z);
        cumulativeZLine.Y2 = centerY - centerY * Math.Cos(cumulativeRotation.Z);
    }
}
}

```

## Xaml

```

<phone:PhoneApplicationPage
x:Class="sdkRawSensorDataCS.GyroscopePage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
FontFamily="{StaticResource PhoneFontFamilyNormal}"
FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
mc:Ignorable="d" d:DesignHeight="696" d:DesignWidth="480"
shell:SystemTray.IsVisible="True">
<!--İçeride kullanılacak bütün kontrol ve tasarım öğeleri bu gridin içerisinde barındırılacak -->
<Grid x:Name="LayoutRoot" Background="Transparent">
<!-- Grid tanımlaması yaparken düzenli bir yapı kurabilmek için satır tanımlamaları yapılabilir. -->
<Grid.RowDefinitions>
<!-- Auto seçildiğinde bu satırın içerisinde bulunan diğer öğelere göre yüksekliğini otomatik olarak ayarlıyor -->
<RowDefinition Height="Auto"/>
<!-- * sembolü ile Grid içerisinde bulunan diğer satırlara bakılarak bu satırın sayfanın geri kalanına yayılması sağlanıyor -->
<RowDefinition Height="*/>
</Grid.RowDefinitions>
<!-- StackPanel ile içerisine eklenen öğelerin dikey veya yatay olarak dizilimi sağlanabilir. -->
<!-- Grid.Row=0 diyerek LayoutRoot Grid'inin ilk satırına bu StackPanel'in yerleştirilmesi sağlanıyor -->

```

<!-- Margin özelliği ile StackPanel'in içinde bulunduğu Grid satırındaki çerçeveden kendisinin uzaklaştırılması sağlanıyor. Bu sayede gridin başladığı yere yapışık olmak yerine ara biraz mesafe bırakılarak kullanıcı deneyimi tasarımına uygun hale getiriliyor -->

```
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
```

<!-- TextBlock metinsel değerler girilebilen bir kontroldür -->

<!-- Style içerisine yazılan StaticResource ataması ile farklı bir yerde tanımlanan tasarım özellikleri çağırılıp bu TextBlock'a uygulanmaktadır. StaticResource tasarım atamalarında kullanılan genel yöntemlerden birisidir. HTML ve CSS bilginize var ise, bunu CSS olarak değerlendirebilirsiniz. -->

```
<TextBlock x:Name="ApplicationTitle" Text="SENSOR APPLICATION" Style="{StaticResource PhoneTextNormalStyle}"/>
```

```
<TextBlock x:Name="PageTitle" Text="gyroscope" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
```

```
</StackPanel>
```

<!-- İç içe grid veya stackpanel kullanılabilir. LayoutRoot gridinin ilk satırına daha önce stackpanel eklemiştiğimiz. Şimdi ikinci satırına içerikleri ekleyeceğimiz yeni bir grid atıyoruz. -->

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
```

```
<StackPanel Orientation="Vertical">
```

```
<StackPanel Orientation="Vertical">
```

<!-- Gyroscope desteklenmiyor, başlatılamadı, durduruldu, başlatıldı vb. bilgilerin yazdırılacağı alan -->

```
<TextBlock Height="30" Name="statusTextBlock" Text="status: " VerticalAlignment="Top" />
```

```
<TextBlock Height="30" Name="timeBetweenUpdatesTextBlock" Text="time between updates: " VerticalAlignment="Top"/>
```

```
</StackPanel>
```

```
<TextBlock Text="current rotational velocity (rads/sec)"></TextBlock>
```

```
<Grid>
```

```
<TextBlock Height="30" HorizontalAlignment="Left" Name="currentXTextBlock" Text="X: 1.0" VerticalAlignment="Top" Foreground="Red" FontSize="28" FontWeight="Bold"/>
```

```
<TextBlock Height="30" HorizontalAlignment="Center" Name="currentYTextBlock" Text="Y: 1.0" VerticalAlignment="Top" Foreground="Green" FontSize="28" FontWeight="Bold"/>
```

```
<TextBlock Height="30" HorizontalAlignment="Right" Name="currentZTextBlock" Text="Z: 1.0" VerticalAlignment="Top" Foreground="Blue" FontSize="28" FontWeight="Bold"/>
```

```
</Grid>
```

```
<Grid Height="140">
```

<!-- çizgi çizmek için kullandığımız kontrol. Çizginin kalınlığını ve rengini Stroke ve StrokeThickness özellikleriyle tanımlayabiliyoruz. -->

```
<Line x:Name="currentXLine" X1="240" Y1="40" X2="240" Y2="40" Stroke="Red"
```

```

StrokeThickness="14"></Line>
<Line x:Name="currentYLine" X1="240" Y1="70" X2="240" Y2="70" Stroke="Green"
StrokeThickness="14"></Line>
<Line x:Name="currentZLine" X1="240" Y1="100" X2="240" Y2="100" Stroke="Blue"
StrokeThickness="14"></Line>

</Grid>
<TextBlock Text="cumulative rotation (degrees)"></TextBlock>
<Grid>
<TextBlock Height="30" HorizontalAlignment="Left" Name="cumulativeXTextBlock" Text="X: 1.0"
VerticalAlignment="Top" Foreground="Red" FontSize="28" FontWeight="Bold"/>
<TextBlock Height="30" HorizontalAlignment="Center" Name="cumulativeYTextBlock" Text="Y: 1.0"
VerticalAlignment="Top" Foreground="Green" FontSize="28" FontWeight="Bold"/>
<TextBlock Height="30" HorizontalAlignment="Right" Name="cumulativeZTextBlock" Text="Z: 1.0"
VerticalAlignment="Top" Foreground="Blue" FontSize="28" FontWeight="Bold"/>
</Grid>
<Grid Height="200" Name="cumulativeGrid">
<Line x:Name="cumulativeXLine" X1="240" Y1="100" X2="240" Y2="0" Stroke="Red"
StrokeThickness="14"></Line>
<Line x:Name="cumulativeYLine" X1="240" Y1="100" X2="240" Y2="0" Stroke="Green"
StrokeThickness="14"></Line>
<Line x:Name="cumulativeZLine" X1="240" Y1="100" X2="240" Y2="0" Stroke="Blue"
StrokeThickness="14"></Line>
</Grid>
</StackPanel>
</Grid>
</Grid>
<!-- ApplicationBar ile içeride bulunan öğeler ile ilgili tekil veya çoğul işlemler yapabileceğimiz ayar
tuşları barındırabilir veya uygulamanın kendisiyle ilgili genel ayarlara yönlendirmeler yapabiliriz. -->
<phone:PhoneApplicationPage.ApplicationBar>
<shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
<!-- Gyroscope özelliğini açıp kapatmak için arka taraftaki C# kodlarını çalıştıracak Button. -->
<shell:ApplicationBarIconButton IconUri="/ApplicationIcon.png" Text="on/off"
Click="ApplicationBarIconButton_Click"/>
</shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
</phone:PhoneApplicationPage>

```