Using Timers

Make a Radar Gun

Your task is to make a working **radar gun** that can determine the speed of an *oncoming* object using your sonar sensor.

Remember:

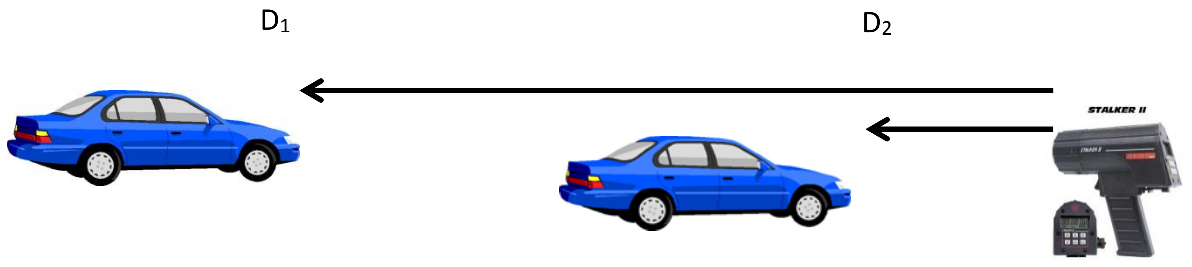
$$\text{Speed} = \frac{\text{Distance}}{\text{Time}}$$

$$v = \frac{d}{t}$$

v = speed
d = distance travelled
t = time taken

To determine how far the car moved you will have to collect **2 separate distances from the sonar sensor.** You can achieve this in a similar way you did in your "Area Finder Assignment" using button presses from the touch sensor.

The total distance = D_1 - D_2



The **total time** is the *time* it took to go from **distance 1** to **distance 2**

Use a timer for this (see next page) – clear timer at **D_1** and then collect the timer reading at **D_2**

Once you have the **Distance** the object went and the **time** is was travelling for, you can calculate the **speed** of the object. Pretty cool.

Make sure you display the distance travelled, time, and speed on the NXT screen
Hand in Code/Algorithm/Marking Sheet for this assignment

$$\text{Speed} = \frac{\text{Distance}}{\text{Time}}$$

Your robot is equipped with four **Timers**, T1 through T4, which you can think of as Time Sensors, or if you prefer, programmable stopwatches.

Using the Timers is pretty straightforward: you reset a timer with the **`ClearTimer()`** command, and it immediately starts counting time.

Then, when you want to find out how long it's been since then, you just use the **`time1[`***TimerName***`]`** command. It gives you the value of the timer in the same way that the **`vexRT[`***ChannelNumber***`]`** command gives you the value of a joystick or button input. The **`time1[`***TimerName***`]`** command gives you the amount of time since the last reset, in milliseconds.

```
ClearTimer(TimerName);

while(time1[TimerName] < 5000)
...
```

Reminder
Timers should be reset when you are ready to start counting.

`time1[TimerName]` represents the timer value in milliseconds since the last reset. It is shown here being used to make a while loop run until 5 seconds have elapsed.

Helpful Hints

The Timer function has limitations. The default **`time1[`***TimerName***`]`** *can only count to around 30 seconds (see description below). Since we need to count to 2 minutes, we will need to use the* **`time10[`***TimerName***`]`** *version instead. Keep this in mind as we move on to the next lesson, where you will use the Timer to control the length of the radio controlled behavior.*

Timer Read Command	Units	Maximum Length of time	Example
`time1[TimerName]`	milliseconds (ms) = 1/1000ths of a second	32767ms ≈ 32.8 seconds	30000 = 30 seconds
`time10[TimerName]`	centiseconds (cs) = 1/100ths of a second	32767cs ≈ 328 seconds ≈5 ½ minutes	12000 = 120 seconds = 2 minutes
`time100[TimerName]`	deciseconds (ds) = 1/10ths of a second	32767ds ≈ 3277 seconds ≈54 ½ minutes	18000 = 1800 seconds = 30 minutes

Why time1 can only count to 30 seconds:

No Timer value can be read if the number value it would produce is more than about 30000 (specifically, 32767 or 2^{15-1}). This is because 32767 is the largest number ROBOTC can fit in a standard integer variable. Other larger variable types do exist, but they require special handling.

Since the default time1[TimerName] command reads in milliseconds, this poses a problem for us. Two minutes is 120 seconds, or 120,000 milliseconds. 120,000 is greater than 32767 and therefore cannot be expressed using the time1 command. The time10 command must be used instead.

Timers Using Timers *(cont.)*

3. Timers should always be reset before use. They begin counting immediately after they are reset. Add the `ClearTimer();` command just before the robot enters the Radio Control while() loop.

```
task main()
{

    bIfiAutonomousMode = false;
    bMotorReflected[port2] = 1;

    ClearTimer(T1);
    while(1 == 1)
    {
      motor[port3] = vexRT[Ch3];
      motor[port2] = vexRT[Ch2];
    }
}
```

3. ***Add this code***
Clear the timer T1 so that it starts counting from the beginning of the radio control period.

4. Change the (condition) of the `while()` loop to check the Timer. The `while()` loop should run while the Timer value is still below (less than, <) 5 seconds.

```
task main()
{

    bIfiAutonomousMode = false;
    bMotorReflected[port2] = 1;

    ClearTimer(T1);
    while(time10[T1] < 500)
```

4a. ***Modify this code***
Replace the "infinite" `(1==1)` condition with the more appropriate "while" condition `(time10[T1] < 500)`. This allows the `while()` loop to continue repeating as long as the accumulated time in T1 is less than 500 hundredths of a second (5 seconds).