```python
import RPi.GPIO as GPIO
import time
import signal
import numpy

from threads import thrd

class Sonar():

    def __init__(self,trigPin,echoPin,bufferLen=5):

        GPIO.setmode(GPIO.BCM)

        self.echoPin=echoPin
        self.trigPin=trigPin

        GPIO.setup(self.trigPin,GPIO.OUT)
        GPIO.setup(self.echoPin,GPIO.IN)

        self.distance=100
        self.distanceBuffer=[100]*bufferLen
        self.avgDistance=100

        self.velocity=1e-5
        self.velocityBuffer=[1e-5]*bufferLen
        self.avgVelocity=1e-5

        self.initialTime=time.time()
        self.timeArray=[time.time()-self.initialTime]*bufferLen

        self.Tcollision=100
        self.Treaction=0.5
        self.Tstop=1
        self.Tmargin=0.5
        self.Tsafe=100

    def __del__(self):
        GPIO.cleanup()


    def measureDistance(self):

        time.sleep(0.05)    # Wait a bit to avoid interference from previous measurement

        def triggerSonar():
            GPIO.output(self.trigPin,False)
            time.sleep(2e-6)    # 2 microseconds
            GPIO.output(self.trigPin,True)
            time.sleep(1e-5)     # 10 microseconds
            GPIO.output(self.trigPin,False)
        thrdTriggerSonar=thrd(triggerSonar)
        thrdTriggerSonar.start()

    #   while GPIO.input(self.echoPin)==0:  # Overwrite pulseStart until pulse is detected
    #       pulseStart=time.time()-self.initialTime
            # Performing rolling average over the buffers to reduce noise-related errors

    #   while GPIO.input(self.echoPin)==1:  # Overwrite pulseEnd until pulse has ended
    #       pulseEnd=time.time()-self.initialTime
```

```
60
61          GPIO.wait_for_edge(self.echoPin,GPIO.RISING,timeout=100)
62          pulseStart=time.time()-self.initialTime
63          GPIO.wait_for_edge(self.echoPin,GPIO.FALLING,timeout=100)
64          pulseEnd=time.time()-self.initialTime
65
66          try:
67
68              pulseDuration=pulseEnd-pulseStart
69
70              sonarDistance=(pulseDuration/2.0)*340
71
72              if sonarDistance<4: # Sensor not accurate for higher values
73                  self.distance=sonarDistance
74
75                  # Update buffer
76                  for b in range(len(self.distanceBuffer)-1,0,-1):    # Shift positio
    n of the old values
77                      self.distanceBuffer[b]=self.distanceBuffer[b-1]
78                  self.distanceBuffer[0]=self.distance    # Include latest measuremen
    t
79
80                  # Update filtered distance
81                  self.avgDistance=numpy.mean(self.distanceBuffer)
82
83                  # Update time array
84                  for t in range(len(self.timeArray)-1,0,-1):
85                      self.timeArray[t]=self.timeArray[t-1]
86                  self.timeArray[0]=(pulseEnd+pulseStart)/2
87
88                  return self.distance
89
90          except:
91              print "Error reading the distance. Trying again"
92
93
94      def computeVelocity(self):
95
96          try:    # To avoid divisions by 0 from throwing an error
97
98              # Backward differences with a three-data-points stencil
99              self.velocity=(2*self.distanceBuffer[0]-self.distanceBuffer[1]-self.dis
    tanceBuffer[2])/(2*self.timeArray[0]-self.timeArray[1]-self.timeArray[2])
100
101         except:
102             pass
103
104         else:
105             for v in range(len(self.velocityBuffer)-1,0,-1):
106                 self.velocityBuffer[v]=self.velocityBuffer[v-1]
107             self.velocityBuffer[0]=self.velocity
108
109             self.avgVelocity=numpy.mean(self.velocityBuffer)
110
111             return self.avgVelocity
112
113
114     def calculateCollision(self):
115
116         self.Tcollision=self.avgDistance/self.avgVelocity
117         self.Tsafe=self.Tcollision-self.Treaction-self.Tstop-self.Tmargin
118
119         return self.Tsafe
```