```
import rospy
from geometry_msgs.msg import Twist
from kobuki_msgs.msg import BumperEvent
from kobuki_msgs.msg import WheelDropEvent
from sensor_msgs.msg import LaserScan
import math
from math import radians

class GoForward():
    def __init__(self):
        # initiliaze
        rospy.init_node('GoForward', anonymous=False)

        # What function to call when you ctrl + c
        rospy.on_shutdown(self.shutdown)

        # Create a publisher which can "talk" to TurtleBot & tell it to move
        self.cmd_vel = rospy.Publisher('cmd_vel_mux/input/navi', Twist, queue_size=10)
        #Create two different twist messages. The attributes of these will change in the callbacks.
        self.mMsg = Twist()
        self.bhitMsg = Twist()
        #Sector/Angular Velocity Table
        self.ang = {1:0, 2:.4, 3:-.4, 4:.8, 5:-.8}
        #Sector/Forward Velocity Table
        self.fwd = {1:.2, 2:.15, 3:.15, 4:.1, 5:.1}
        #Sector/Debug Message Table
        self.dbgmsg = {1:'Move Straight', 2:'Veer Left', 3:'Veer Right', 4:'Turn Left', 5:'Turn Right'}

        #Bumper, Wheel Drop, Laserscan Subscribers
        rospy.Subscriber("/mobile_base/events/bumper", BumperEvent,
self.BumperEventCallback)
        rospy.Subscriber("/mobile_base/events/wheel_drop", WheelDropEvent,
self.WheelDropEventCallback)
        rospy.Subscriber("/scan", LaserScan, self.LaserScanCallback)
        rospy.loginfo("Line 32")

        # Define the states for the state machine

        # bhit is bumper hit. The most significant bit is the left bumper, the next is the middle
bumper, the next is the right bumper, the next is the left wheel, & the least significant bit is the
right wheel. 1 Means it's been pushed | the wheels are dropped, 0 means not hit & wheels are not
dropped
        self.bhit = 0b00000
        # self.bhit will be made up of self.bumpers & self.wheels
        self.bumpers = 0b000
        self.wheels = 0b00
```

```
    self.safety = 0b00000

    self.r = rospy.Rate(5)


    # as long as you haven't ctrl + c keeping doing...
    while not rospy.is_shutdown():
        # Make the state machine here

        # Moving Forward State
        if (self.safety == 0):
            if (self.bhit == 0):
                for i in range(0, 3):
                    self.cmd_vel.publish(self.mMsg)
                    self.r.sleep()
            if (self.bhit > 0):
                self.safety = 1

        # Something Happened State
        if (self.safety == 1):
            for i in range(0, 10):
                self.cmd_vel.publish(self.bhitMsg)
                self.r.sleep()
            rospy.loginfo("Line 65")
            if (self.bhit == 0):
                self.safety = 0

    # Is this necessary?
    rospy.spin()


def BumperEventCallback(self, data):
    rospy.loginfo("Bumper Event Callback")
    # Print out what happened & change the state variable based on what happened
    if (data.state == BumperEvent.PRESSED) :
        state = "pressed"
        if (data.bumper == BumperEvent.LEFT) :
            bumper = "left"
            self.bhit = self.bhit | 0b10000
            self.bhitMsg.angular.z = -radians(60)
            self.bhitMsg.linear.x = -.1
        elif (data.bumper == BumperEvent.RIGHT) :
            bumper = "right"
```

```
            self.bhit = self.bhit | 0b00100
            self.bhitMsg.angular.z = radians(60)
            self.bhitMsg.linear.x = -.1
        else:
            bumper = "center"
            self.bhit = self.bhit | 0b01000
            self.bhitMsg.linear.x = -.25
            self.bhitMsg.angular.z = 0
    else:
        state = "released"
        if (data.bumper == BumperEvent.LEFT) :
            bumper = "left"
            self.bhit = self.bhit & 0b01111


        elif (data.bumper == BumperEvent.RIGHT) :
            bumper = "right"
            self.bhit = self.bhit & 0b11011

        else:
            bumper = "center"
            self.bhit = self.bhit & 0b10111

    rospy.loginfo("Bumper %s was %s."%(bumper, state))


def WheelDropEventCallback(self, data):
    rospy.loginfo("Wheeldrop Event Callback")
    if (data.state == WheelDropEvent.RAISED):
        state = "raised"
        if (data.wheel == WheelDropEvent.LEFT):
            self.bhit= self.bhit & 0b11101
            wheel = "left"
        else:
            self.bhit= self.bhit & 0b11110
            wheel = "right"
    else:
        state = "dropped"
        if (data.wheel == WheelDropEvent.LEFT):
            wheel = "left"
            self.bhit= self.bhit | 0b00010
            self.bhitMsg.linear.x = 0
            self.bhitMsg.angular.z = 0
        else:
            wheel = "right"
            self.bhit= self.bhit | 0b00001
```

```
            self.bhitMsg.linear.x = 0
            self.bhitMsg.angular.z = 0
        rospy.loginfo("The %s wheel was %s."%(wheel, state))

    def LaserScanCallback(self, scanmsg):
        self.averager(scanmsg)
        self.movement()

    def averager(self, laserscan):
        '''Goes through 'ranges' array in laserscan message and determines
        where obstacles are located. The class variables sect_1, sect_2,
        and sect_3 are updated as either '0' (no obstacles within 0.7 m)
        or '1' (obstacles within 0.7 m)

        Parameter laserscan is a laserscan message.'''
        entries = len(laserscan.ranges)

        totalEntries1 = 0
        totalEntries2 = 0
        totalEntries3 = 0
        totalEntries4 = 0
        totalEntries5 = 0
        toSubtract1 = 0
        toSubtract2 = 0
        toSubtract3 = 0
        toSubtract4 = 0
        toSubtract5 = 0

        for entry in range(0, (entries/5)-1):
            if not (math.isnan(laserscan.ranges[entry])):
                totalEntries1 += laserscan.ranges[entry]
            else:
                totalEntries1 += 10

        self.average1 = totalEntries1/(entries/5)


        for entry in range((entries/5), ((2*entries)/5)-1):
            if not (math.isnan(laserscan.ranges[entry])):
                totalEntries2 += laserscan.ranges[entry]
            else:
                totalEntries2 += 10
        self.average2 = totalEntries2/(entries/5)


        for entry in range((2*entries/5), (3*entries/5)-1):
```

```
            if not (math.isnan(laserscan.ranges[entry])):
                totalEntries3 += laserscan.ranges[entry]
            else:
                totalEntries3 += 10
        self.average3 = totalEntries3/(entries/5)


        for entry in range((3*entries/5), (4*entries/5)-1):

            if not (math.isnan(laserscan.ranges[entry])):
                totalEntries4 += laserscan.ranges[entry]
            else:
                totalEntries4 += 10
        self.average4 = totalEntries4/(entries/5)


        for entry in range((4*entries/5), entries):

            if not (math.isnan(laserscan.ranges[entry])):
                totalEntries5 += laserscan.ranges[entry]
            else:
                totalEntries5 += 10
        self.average5 = totalEntries5/(entries/5)

    def movement(self):
        '''Uses the information known about the obstacles to move robot.

        Parameters are class variables and are used to assign a value to
        variable sect and then  set the appropriate angular and linear
        velocities, and log messages.
        These are published and the sect variables are reset.'''
        averages = [self.average3,self.average4,self.average2,self.average5,self.average1]
        #averages = [self.average5,self.average4,self.average3,self.average2,self.average1]
        rospy.loginfo(averages)
        #for average in averages:
        #    rospy.loginfo("1: " + str(average))
        maxSector = averages.index(max(averages)) + 1
        # I think this logic is wrong cause it defaults left.
        # One way to combat this would make "1" go straight in our archetecture, and then have 2
and 3
        # be the veer states, and then 4 and 5 be the turn states
        self.mMsg.angular.z = self.ang[maxSector]
        self.mMsg.linear.x = self.fwd[maxSector]
        rospy.loginfo(self.dbgmsg[maxSector])
        self.reset_averages()
```

```
    def reset_averages(self):
        '''Resets the below variables before each new scan message is read'''
        self.average1 = 0
        self.average2 = 0
        self.average3 = 0
        self.average4 = 0
        self.average5 = 0



    def shutdown(self):
        # stop turtlebot
        rospy.loginfo("Stop TurtleBot")
        # a default Twist has linear.x of 0 & angular.z of 0.  So it'll stop TurtleBot
        self.cmd_vel.publish(Twist())
        # sleep just makes sure TurtleBot receives the stop comm& prior to shutting down the script
        rospy.sleep(1)

if __name__ == '__main__':
    #try:
    GoForward()
    #except:
     #   rospy.loginfo("GoForward node terminated.")
```