```python
#!/usr/bin/env python3
#
#   demo134.py
#
#   Demonstration node to interact with the HEBIs.
#
import numpy as np
import rclpy

from rclpy.node         import Node
from sensor_msgs.msg    import JointState


#
#   Definitions
#
RATE = 100.0            # Hertz


#
#   DEMO Node Class
#
class DemoNode(Node):
    # Initialization.

    AMPS = np.array([0.4, 0.3, 0.25])
    PERIODS = np.array([1.0, 4.0, 2.0])
    WAVE_T = 6
    VMAX = 2
    AMAX = VMAX / 3

    def __init__(self, name):

        # Initialize the node, naming it as specified
        super().__init__(name)
        self.t = 0
        self.t0 = self.t
        self.mode = 0
        self.position0 = self.grabfbk()
        self.Tmove = DemoNode.splinetime(self.position0, DemoNode.wave(0)[0], np.zeros(3), DemoNode.wave(0)[1])
        # Create a temporary subscriber to grab the initial position.
        self.get_logger().info("Initial positions: %r" % self.position0)

        # Create a message and publisher to send the joint commands.
        self.cmdmsg = JointState()
        self.cmdpub = self.create_publisher(JointState, '/joint_commands', 10)

        # Wait for a connection to happen.  This isn't necessary, but
        # means we don't start until the rest of the system is ready.
        self.get_logger().info("Waiting for a /joint_commands subscriber...")
        while(not self.count_subscribers('/joint_commands')):
            pass

        # Create a subscriber to continually receive joint state messages.
        self.fbksub = self.create_subscription(
            JointState, '/joint_states', self.recvfbk, 10)

        # Create a timer to keep calculating/sending commands.
        rate       = RATE
        self.timer = self.create_timer(1/rate, self.sendcmd)
        self.get_logger().info("Sending commands with dt of %f seconds (%fHz)" %
                               (self.timer.timer_period_ns * 1e-9, rate))

    # Shutdown
    def shutdown(self):
        # No particular cleanup, just shut down the node.
        self.destroy_node()


    # Grab a single feedback - do not call this repeatedly.
    def grabfbk(self):
        # Create a temporary handler to grab the position.
        def cb(fbkmsg):
            self.grabpos   = list(fbkmsg.position)
            self.grabready = True

        # Temporarily subscribe to get just one message.
        sub = self.create_subscription(JointState, '/joint_states', cb, 1)
        self.grabready = False
        while not self.grabready:
            rclpy.spin_once(self)

        self.destroy_subscription(sub)

        # Return the values.
        return self.grabpos


    # Receive feedback - called repeatedly by incoming messages.
    def recvfbk(self, fbkmsg):
        # Just print the position (for now).
        # print(list(fbkmsg.position))
        pass
```

```
94
95      def wave(tau):
96          pos = DemoNode.AMPS * np.sin(tau * np.pi / 6 * 2 * DemoNode.PERIODS)
97          vel = np.multiply(DemoNode.AMPS, 2 * np.pi / 6 * DemoNode.PERIODS) * np.cos(tau * np.pi / 6 * 2 * DemoNode.PERIO
    DS)
98          return pos, vel
99
100     # Send a command - called repeatedly by the timer.
101     def sendcmd(self):
102         # Build up the message and publish.
103         pos = np.zeros(3)
104         vel = np.zeros(3)
105         tau = self.t - self.t0
106         if self.mode == 0:
107             init_pos = self.position0
108             pos, vel = DemoNode.spline(tau, self.Tmove, init_pos, DemoNode.wave(0)[0], np.zeros(3), DemoNode.wave(0)[1])
109         elif self.mode == 1:
110             pos, vel = DemoNode.wave(tau)
111         elif self.mode == 2:
112             pos, vel = DemoNode.spline(tau, self.Tmove, DemoNode.wave(0)[0], self.position0, DemoNode.wave(0)[1], np.zer
    os(3))
113         elif self.mode == 3:
114             pos, vel = self.position0, np.zeros(3)
115         else:
116             raise Exception('Unkown mode encountered')
117
118         if self.t - self.t0 > self.Tmove:
119             self.mode += 1
120             self.mode %= 4
121             self.t0 = self.t
122             if self.mode == 0:
123                 self.Tmove = DemoNode.splinetime(self.position0, DemoNode.wave(0)[0], np.zeros(3), DemoNode.wave(0)[1])
124             elif self.mode == 1:
125                 self.Tmove = 6
126             elif self.mode == 2:
127                 self.position0 = np.zeros(3)
128                 self.position0[1] = np.pi
129                 self.Tmove = DemoNode.splinetime(DemoNode.wave(0)[0], self.position0, DemoNode.wave(0)[1], np.zeros(3))
130             elif self.mode == 3:
131                 self.Tmove = 1
132             else:
133                 raise Exception('Unkown mode encountered')
134
135         self.get_logger().info(str(vel))
136
137         self.cmdmsg.header.stamp = self.get_clock().now().to_msg()
138         self.cmdmsg.name         = ['one', 'two', 'three']
139         self.cmdmsg.position     = list(pos)
140         self.cmdmsg.velocity     = list(vel)
141         self.cmdmsg.effort       = list(np.zeros(3))
142         self.cmdpub.publish(self.cmdmsg)
143         self.t += 0.01
144
145     def spline(t, T, p0, pf, v0, vf):
146         # Compute the parameters.
147         a = p0
148         b = v0
149         c =   3*(pf-p0)/T**2 - vf/T    - 2*v0/T
150         d = - 2*(pf-p0)/T**3 + vf/T**2 +   v0/T**2
151         # Compute the current (p,v).
152         p = a + b * t +   c * t**2 +   d * t**3
153         v =     b     + 2*c * t    + 3*d * t**2
154         return (p,v)
155
156     def splinetime(p0, pf, v0, vf):
157         m = max(1.5 * (np.linalg.norm(pf - p0) / DemoNode.VMAX + np.abs(v0) / DemoNode.AMAX + np.abs(vf) / DemoNode.AMAX
    ))
158         return max(m, 0.5)
159
160
161
162 #
163 #   Main Code
164 #
165 def main(args=None):
166     # Initialize ROS.
167     rclpy.init(args=args)
168
169     # Instantiate the DEMO node.
170     node = DemoNode('demo')
171
172     # Spin the node until interrupted.
173     rclpy.spin(node)
174
175     # Shutdown the node and ROS.
176     node.shutdown()
177     rclpy.shutdown()
178
179 if __name__ == "__main__":
180     main()
```