```python
def transition_model(self, us, dt):
    """
    Unicycle model dynamics.
    Inputs:
        us: np.array[M,2] - zero-order hold control input for each particle.
        dt: float         - duration of discrete time step.
    Output:
        g: np.array[M,3] - result of belief mean for each particle
                           propagated according to the system dynamics with
                           control u for dt seconds.
    """
    ########## Code starts here ##########
    # TODO: Compute g.

    # We don't use numpy.where here as arrays are not lazy-evaluated.

    U, X = us.T, self.xs.T
    n = self.M      # num of particles
    V_all, om_all = U       # All of shape (n, )
    x_all, y_all, th_all = X

    # First we need to split up the particles depending on |om|
    # to use either the normal formulae or after applying l'Hopitals
    idx = np.linspace(0, n, n, endpoint=False, dtype=np.int)
    cond = np.absolute(om_all) > EPSILON_OMEGA

    i1 = idx[cond]
    n1 = i1.shape[0]

    # Preallocate output
    x_til = np.zeros(n)
    y_til = np.zeros(n)
    th_til = np.zeros(n)

    # Normal case
    V, om = V_all[i1], om_all[i1]
    x, y, th = x_all[i1], y_all[i1], th_all[i1]
    # We preserve particle ordering to appease the validator
    th_til[i1]   = th + om*dt
    x_til[i1]    = x + V/om * (np.sin(th+om*dt) - np.sin(th))
    y_til[i1]    = y - V/om * (np.cos(th+om*dt) - np.cos(th))

    # l'Hopital's case
    i2 = idx[~cond]
    V, om = V_all[i2], om_all[i2]
    x, y, th = x_all[i2], y_all[i2], th_all[i2]
    th_til[i2]   = th + om*dt
    x_til[i2]    = x + V*dt*np.cos(th)
    y_til[i2]    = y + V*dt*np.sin(th)

    g = np.column_stack([x_til, y_til, th_til])

    ########## Code ends here ##########
    return g
```