**Ground Control**

publish(‘Initialization’)

while True:

if msg exists:

if msg is ‘Power Off’:

break

else:

publish(msg)

**State Machine**

msg = subscribe(**Ground Control**)

if msg is ‘Acquisition and Tracking’:

R, t = **Camera**.capture\_and\_process(msg.mission)

if R is not None: # target

miss = 0

if tracking is False: # first target in series

tracking = True

else: # ‘Tracking’

x, z = **Attitude**.kinematics\_and\_ode45(R, t, R0, t0, x0, z0)

**Motors**.write(x[7::])

[R0, t0, x0, z0] = [R, t, x, z]

else: # no target

miss += 1

if miss > msg.patience: # ‘Acquisition’, i.e., non-Tracking

tracking = False

R, t = **Attitude**.acquisition(msg.acquisition\_mode)

x, z = **Attitude**.kinematics\_and\_ode45(R, t, R0, t0, x0, z0)

**Motors**.write(x[7::])

[R0, t0, x0, z0] = [R, t, x, z]

elif msg is ‘Other’:

...

elif msg is ‘Initialization’:

tracking = False

miss = msg.patience

x0 = zeros(10, 1)

z0 = zeros(3, 1)

**Camera**

def capture():

...

return img, t

def process(img, mission):

try:

...

except Exception as e:

R = None

return R

def capture\_and\_process(mission):

img, t = **Camera**.capture()

R = **Camera**.process(img, mission)

return R, t

**Attitude**

def kinematics(dt, R, R0, z0):

...

return xdot, z

def ode45(dt, xdot, x0):

...

return x

def kinematics\_and\_ode45(R, t, R0, t0, x0, z0):

dt = t – t0

xdot, z = **Attitude**.kinematics(dt, R, R0, z0)

x = **Attitude**.ode45(dt, xdot, x0)

return x, z

**Motors**

def write(rpm):

...

# motor1\_rpm = rpm[0]

# motor2\_rpm = rpm[1]

# motor3\_rpm = rpm[2]

...

return