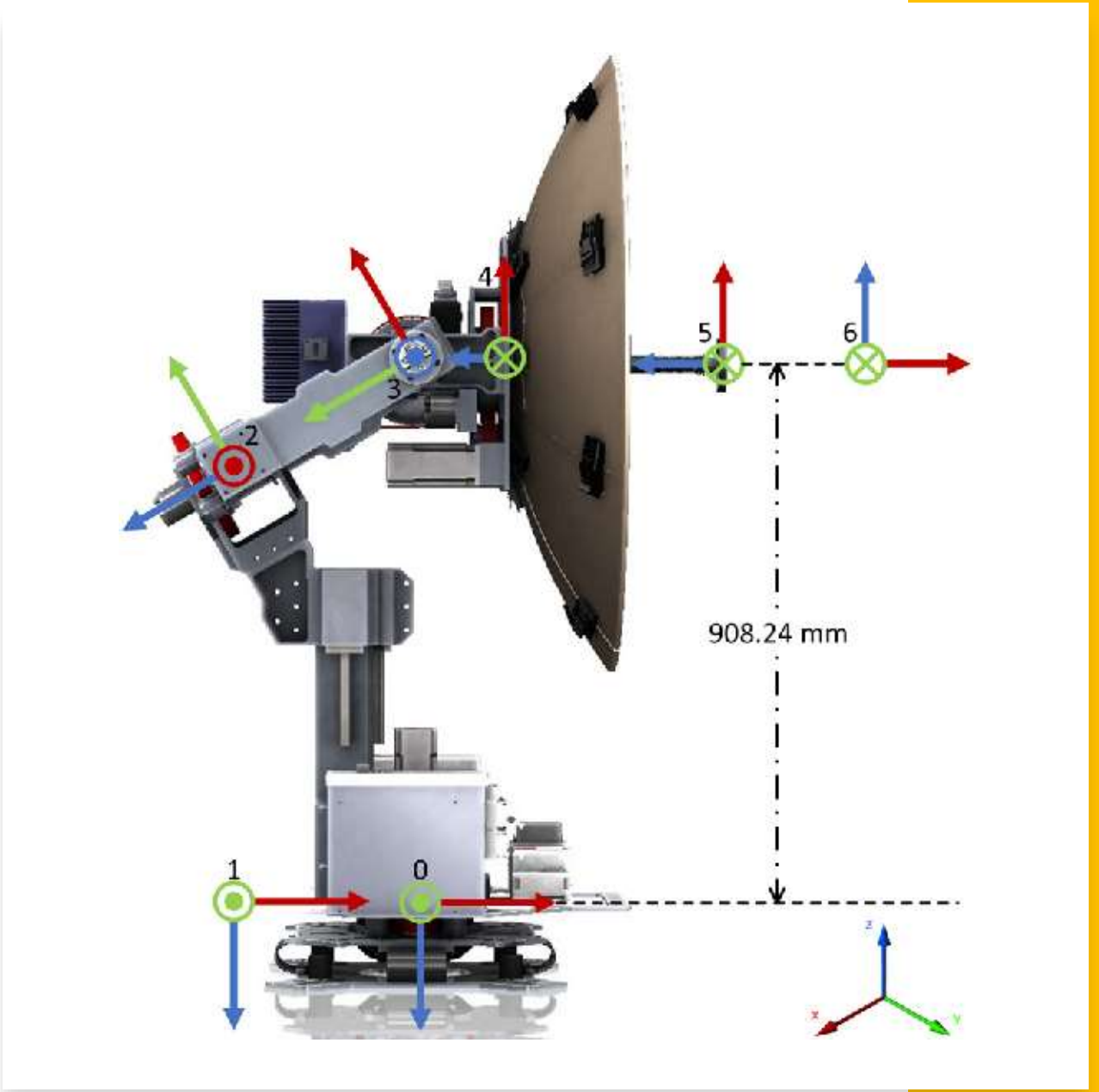
Pan-Tilt Rainforcement Learning

What is Pan-Tilt Satcom



Pan-Tilt

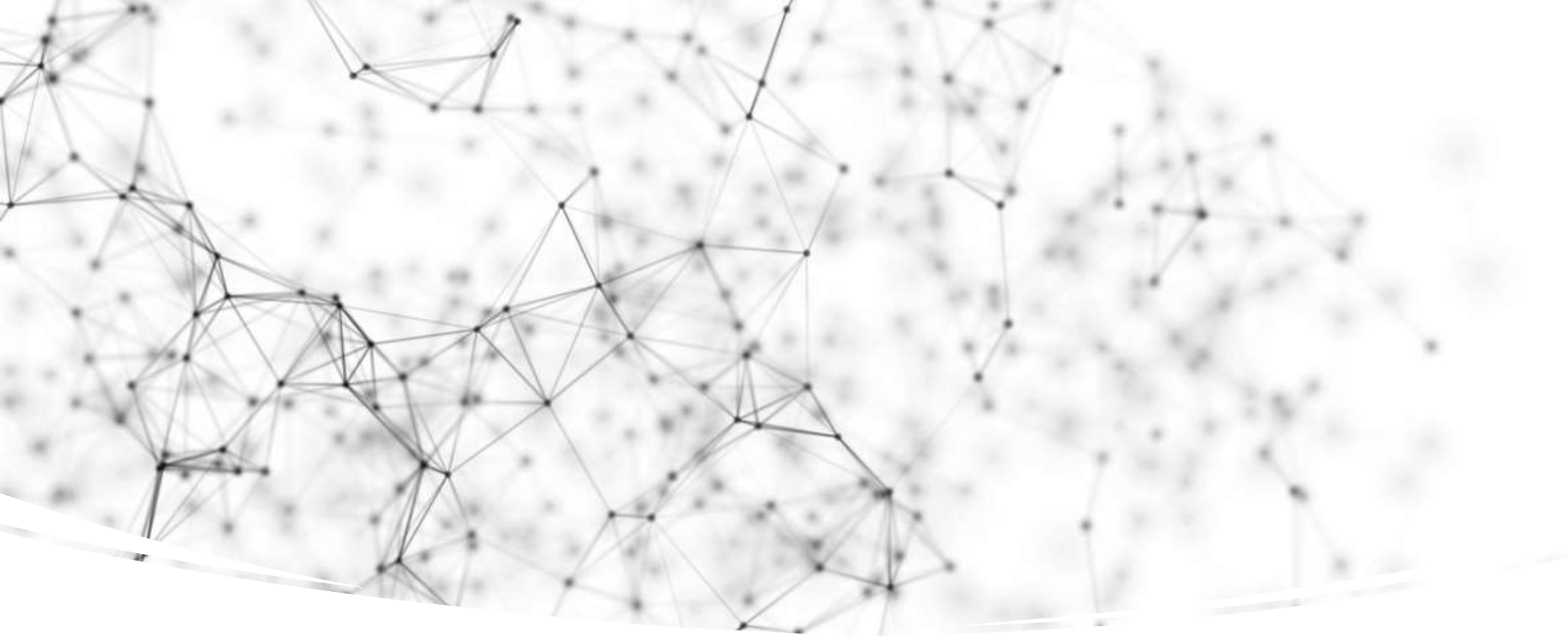
- A pan-tilt system, also known as a pan-tilt unit or PTU, is a mechanism used to rotate and tilt an object or camera in a controlled manner. It is commonly used in various applications such as surveillance systems, robotics, photography, and video production.

Satcom

- SATCOM stands for "Satellite Communication," which refers to the use of artificial satellites to facilitate communication between various points on Earth.

What is the problem ?

- The problem I am focusing on is control of the system. So satcom system’s main purpose is get face to face connection between satellite and antenna and keep that connection over time even system moves.
- It is important to know that system knows where to go but what system do not know is speed.



Code Section

- Connections
- Environment
- Model

Connections

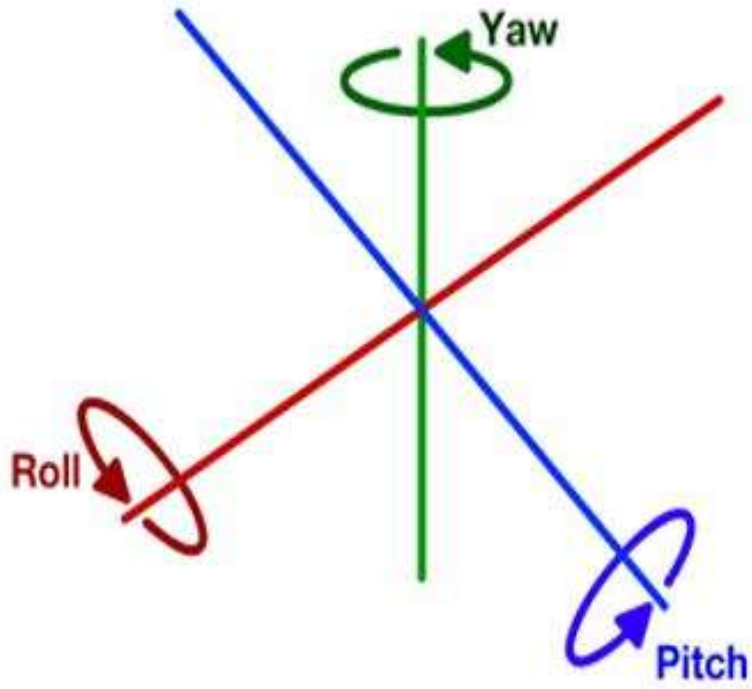
```
from Phidget22.Phidget import *
from Phidget22.Net import *
from Phidget22.Devices.BLDCMotor import *
import time



class MotorController:
    def __init__(self):
        #p0 pitch
        #p1 heading
        Net.enableServerDiscovery(PhidgetServerType.PHIDGETSERVER_DEVI

        self.bldcMotor0 = BLDCMotor()
        self.bldcMotor0.setHubPort(1)
        self.bldcMotor0.setIsRemote(True)

        self.bldcMotor1 = BLDCMotor()
        self.bldcMotor1.setHubPort(0)
        self.bldcMotor1.setIsRemote(True)

        self.bldcMotor0.openWaitForAttachment(5000)
        self.bldcMotor1.openWaitForAttachment(5000)



    def setMotorVelocities(self, p0, p1):

        if p0>1:
            p0=1
        if p0<-1:
            p0=-1
        if p1>1:
            p1=1
        if p1<-1:
            p1=1


        self.bldcMotor0.setTargetVelocity(p0)
        self.bldcMotor1.setTargetVelocity(p1)

    def stop(self):
        self.bldcMotor0.setTargetVelocity(0)
        self.bldcMotor1.setTargetVelocity(0)


motorcontroller=MotorController()

motorcontroller.setMotorVelocities(0,0.01)


time.sleep(10)

motorcontroller.stop()
```

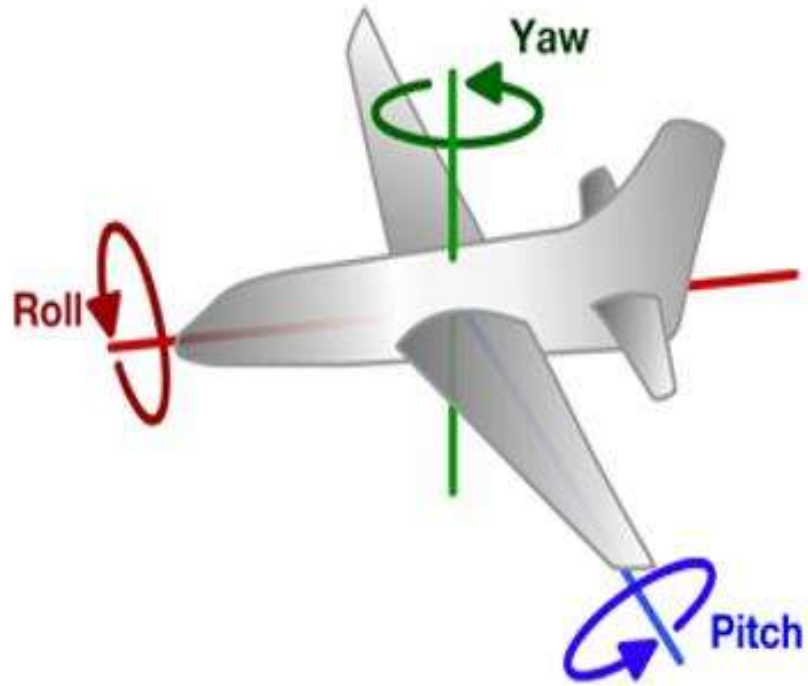
Connections Motors

Connections Imu Cencor

```
        self.t = threading.Thread(target=self.run_on_thread)
        self.t.start()
        self.heading=None
        self.roll=None
        self.pitch=None
        self.attachment = 0


    def onAlgorithmData(self,spatial0, quaternion, timestamp):
        # IMU
        eulerAngles = spatial0.getEulerAngles()

        self.pitch = eulerAngles.pitch
        self.roll = eulerAngles.roll
        self.heading = eulerAngles.heading

        pitch_rad = math.radians(self.pitch)
        yaw_rad = math.radians(self.roll)
        heading_rad = math.radians(self.heading)


    def run_on_thread(self):
        Net.enableServerDiscovery(PhidgetServerType.PHIDGETSERVER_DEVICEREMOTE)

        # Create a new spatial object
        spatial0 = Spatial()

        spatial0.setHubPort(2)

        spatial0.setIsRemote(False)

        spatial0.setOnAlgorithmDataHandler(self.onAlgorithmData)

        spatial0.setOnAttachHandler(self.onAttachHandler)
        spatial0.setOnDetachHandler(self.onDetachHandler)

        spatial0.openWaitForAttachment(5000)

        spatial0.setDataInterval(100)

        print("Spatial detached")

        # Close the spatial object
       # spatial0.close()
    def onAttachHandler(self, magnetometer):
        self.attachment = 1

    def onDetachHandler(self, magnetometer):
        self.attachment = 0
        self.angle = None

    def convert_to_pan_tilt(self,pitch, yaw, heading):
        # Convert degrees to radians
        pitch_rad = math.radians(self.pitch)
        yaw_rad = math.radians(self.roll)
        heading_rad = math.radians(self.heading)

        # Calculate pan angle
        pan = math.atan2(math.sin(yaw_rad - heading_rad), math.cos(yaw_rad - heading_rad))

        # Calculate tilt angle
        tilt = math.asin(-math.sin(pitch_rad) * math.cos(yaw_rad - heading_rad))

        # Convert radians back to degrees
        pan_deg = math.degrees(pan)
        tilt_deg = math.degrees(tilt)
```

Cencor Data





Environment

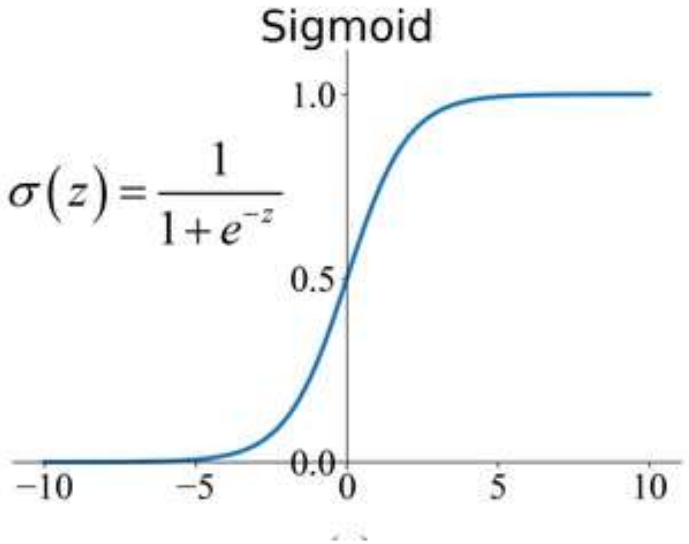
Part of Environment

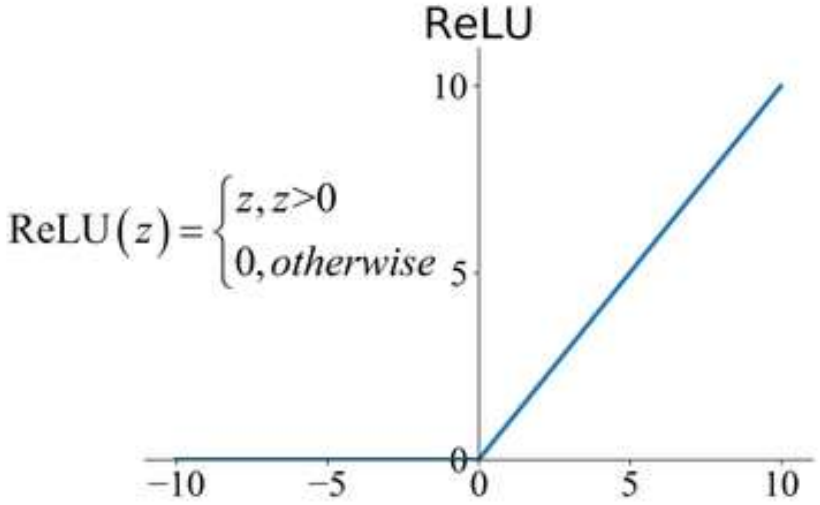
```
    def distance(self):
        distance_heading=abs(self.heading_-self.heading)
        distance_roll=abs(self.roll_-self.roll)
        return distance_heading,distance_roll

    def action(self,p0,p1,pitch,roll,heading):

        self.pitch=pitch
        self.roll=roll
        self.heading=heading

        self.p0=p0
        self.p1=p1

        self.direction()
        self.setMotorVelocities(p0, p1)
        self.state()

    def state2(self,pitch,roll,heading):

        self.pitch=pitch
        self.roll=roll
        self.heading=heading

        err_heading=abs(self.ref_heading-self.heading)
        err_roll=abs(self.ref_roll-self.roll)

        state=np.array([self.p0, self.p1, err_roll, err_heading]).reshape(1, 4)
        return state

    def isaliave(self):
        if self.attachment==1:
            return True
        else:
            return False



env=Env()
```

Model

```
model = Sequential()
model.add(Dense(100, activation='relu', input_shape=(4,)))
model.add(Dense(200, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(24, activation='relu'))
model.add(Dense(2, activation='sigmoid'))

my_model = model
my_model.compile(loss='mse', optimizer=Adam(lr=0.001))
```

Activation Functions





Training

```
while True:

    for i in range(1000):

        pitch,roll,heading=onAlgorithmData(spatial0, 0,0)
        #print("collecting"+str(i))
        if attachment==1:
            pitch,roll,heading=onAlgorithmData(spatial0, 0,0)
            state=env.state2(pitch,roll,heading)
            if random.uniform(0,1) <= epsilon:
                p0 = round(random.random()*0.5, 2)
                p1 = round(random.random(), 2)
                epsilon = epsilon*epsilon_decay
            else:
                print("predicting  "+str(i)+"   "+"ep "+str(episode))
                result = my_model.predict(state)
                p0 = round(result[0][0]*0.5, 2)
                p1 = round(result[0][1], 2)
            time.sleep(0.1)
            env.action(p0, p1,pitch,roll,heading)
            print("p0  "+str(p0)+"   "+"p1 "+str(p1))
            pitch,roll,heading=onAlgorithmData(spatial0, 0,0)
            state1=env.state2(pitch,roll,heading)
           #d0, d1, p0_, p1_, p0, p1
           #multiply 1000 for high resolution
            d0=abs(state[0][2]-state1[0][2])*1000
            d1=abs(state[0][3]-state1[0][3])*1000
            p0_=state[0][0]
            p1_=state[0][1]
            p0 =state1[0][0]
            p1 =state1[0][1]
            print(str(state1[0][2])+"  "+str(state1[0][3]))
            memory.append((d0, d1, p0_, p1_, p0, p1))
        else:
            err=1
            env.action(0, 0,pitch,roll,heading)
            env.stop()
            break

    if err==1:
        break
    # stop system
    env.action(0, 0,pitch,roll,heading)
    # train
    episode=episode+1
    for i in range(1000):
        sample = memory[i]
        state=(np.array([sample[0], sample[1], sample[2], sample[3]])).reshape(1, 4)
        y_head=(np.array([sample[4], sample[5]])).reshape(1,2)
        print("training  "+str(i))
        my_model.fit(state, y_head, verbose=1)



    if episode==5:
        env.stop()
        print("done")
        break

my_model.save('my_model_actor_low.h5')
```