# You’ve Been Promoted

**Category**: “Anomaly Review Bored”

## Description

This challenge consists of a remote TCP service for which we are not given any source or binary. When connecting to the given address (e.g., through Netcat), we are greeted with the following information:

$ nc management.quals2023-kah5Aiv9.satellitesabove.me 5300  
Send me commands to get the spacecraft under control and the spacecraft despun  
You must run for 3600 seconds  
Make sure the magnitude of the spacecraft angular velocity vector is less than 0.001 (rad/s)  
Make sure each reaction wheel has a spin rate that is between -20 and 20 (rad/s)  
  
Reaction wheels accept torque commands in N-m  
Reaction wheel commands are valid between [-0.2, 0.2] N-m  
Available reaction wheels:  
- Wheel\_X: aligned with body X axis  
- Wheel\_Y: aligned with body Y axis  
- Wheel\_Z: aligned with body Z axis  
  
Magnetic Torquer Bars (MTB) accept commands in magnetic dipole (A-m^2)  
MTB dipole commands are valid between [-1000.0, 1000.0] (A-m^2)  
Available MTB  
- MTB\_X: aligned with body X axis  
- MTB\_Y: aligned with body Y axis  
- MTB\_Z: aligned with body Z axis  
  
Actuator commands are formatted as:  
Wheel\_X, Wheel\_Y, Wheel\_Z, MTB\_X, MTB\_Y, MTB\_Z  
  
Sensor:Time (sec), AngV\_X (rad/s), AngV\_Y (rad/s)), AngV\_Z(rad/s), WheelX(rad/s), WheelY(rad/s), WheelZ(rad/s), magX (T), magY(T), magZ(T)  
0.0,0.1,0.1,-0.2,314.1592653589793,-471.23889803846896,282.7433388230814,-3.210377245457677e-05,-1.1355247439189624e-05,-2.263494595823975e-05  
Enter actuator command: nan,nan,nan,nan,nan,nan  
Array item nan is not finite.  
Expected format of array input is 'X1,X2,X3,....,XN'

The remote server is asking us to help controlling a spinning spacecraft through three-axis stabilization. The spacecraft is in fact equipped with 3 reaction wheels (RW) and 3 magnetic torque bars (MTB), each mounted on a different axis.

Each second of time we receive sensor readings for the current angular speed of the spacecraft on each axis (in rad/s), the current angular speed of each RW in (rad/s), and the current magnetic field strenght (in Tesla) measured by the spacecraft on each axis.

To perform three-axis stabilization, each second of time we can apply a chosen torque (between -0.2Nm and +0.2Nm) to each RW and a chosen magnetic dipole (between -1000Am2 and +1000Am2) to each MTB. We have 3600 seconds to get the spacecraft’s angular velocity within -0.001 and 0.001 rad/s on all 3 axes and the angular velocity of all reaction wheels within -20 and +20 rad/s. If, at the end of the 3600th second, all requested values are found within range, the spacecraft will be considered stabilized.

## Solution

The system that is being emulated by the server seems to be a standard three-axis stabilization problem:

* Torque can be applied to reaction wheels to make them accelerate and spin either clockwise or counterclockwise to contrast the spacecraft spin on each axis.
* Magnetic torque bars can be powered with a positive or negative current to provide the whole spacecraft with positive or negative torque perpendicular to the magnetic field.

To reach our goal, we can control the spin of the spacecraft itself through the RWs, and the spin of the RWs through the MTBs.

We implemented our solution using the [simple-pid](https://pypi.org/project/simple-pid/) Python package to model 6 PID controllers (one for each RW and MTB) as follows:

* 3 PIDs (one per axis) each taking the negated angular velocity of the spacecraft on the corresponding axis as input. The produced response, limited between -0.2 and +0.2, is then provided as the torque to apply to the RWs (one per axis).
* 3 PIDs (one per axis) each taking one negated component (on the corresponding axis) of the cross product *W⨯B*, where *W* is the vector *(WheelX, WheelY, WheelZ) [rad/s]*, and *B* is the magnetic field vector *(magX, magY, magZ) [T]*. The produced response, limited between -1000 and +1000, is then provided as the elecric dipole to apply to the MTBs (one per axis).

Theoretically speaking, determining the right magnetic dipole to control the MTBs is not simple, because we ideally want them to produce a torque that counters the spin of the RWs, but the only torque the MTBs can generate is perpendicular to the magnetic field felt by the spaceship at any given time. The equation to calculate the applied torque given a magnetic dipole (*M*) is *T = M⨯B*. It is however not possible to simply invert the equation and find *M* given *T* and *B*, as the matrix used for the cross-product (*[(0,Bz,-By), (-Bz,0,Bx), (By,-Bx,0)]*) is non-invertible.

Knowing the above, although seemingly nonsensical at first glance (even just dimensionally speaking), the intuitive reasoning we followed to come up with *W⨯B* was as follows:

1. The torque we want to apply on each axis needs to be opposite in sign and directly proportional in modulus to the angular velocity of the RWs.
2. The torque we can apply is perpendicular to both the magnetic field and the applied magnetic dipole (*T = M⨯B*).
3. Therefore the magnetic dipole vector to apply needs to be proportional in modulus and opposite in direction to *W⨯B*.

After figuring out the above, the rest was a matter of trial and error and manual tuning. Running multiple simulations, we ended up with the following PID parameters:

* *(Kp, Ki, Kd) = (5, 0.5, 0.1)* for the 3 RW PIDs
* *(Kp, Ki, Kd) = (105, 104, 5•104)* for the 2 MTB PIDs for the x-axis and z-axis
* *(Kp, Ki, Kd) = (3•105, 105, 105)* for the MTB PID for the y-axis.

While doing this, we also noticed that stabilizing both the spacecraft’s angular velocity and the RWs’ angular velocity was harder than expected, because PID responses were in turn altering other PIDs inputs. In other words, reducing the angular velocity of the spaceship along one axis means increasing the angular velocity of the RW for that axis, and reducing the angular velocity of a RW for one axis through MTBs could mean altering the angular velocity of the spacecraft on any axis.

In order to get over this issue, we manually defined some time windows within which we would generate a response for the torque to apply to the RWs. Outside these time windows, the response would just be 0 on all axes. We ended up applying torque to RWs only between t=500s and t=999s seconds, and then from t=2500s onwards.

## Solution script

#!/usr/bin/env python3  
#  
# @mebeim - 2023-04-02  
#  
  
from pwn import \*  
from simple\_pid import PID  
import numpy as np  
  
TIME = 0  
  
def faketime():  
 global TIME  
 return TIME  
  
pwx = PID(5, .5, .1, setpoint=0, sample\_time=1, output\_limits=(-0.2, 0.2))  
pwy = PID(5, .5, .1, setpoint=0, sample\_time=1, output\_limits=(-0.2, 0.2))  
pwz = PID(5, .5, .1, setpoint=0, sample\_time=1, output\_limits=(-0.2, 0.2))  
  
pwx.time\_fn = faketime  
pwy.time\_fn = faketime  
pwz.time\_fn = faketime  
  
pmx = PID(1e5, 1e4, 5e4, setpoint=0, sample\_time=1, output\_limits=(-1000, 1000))  
pmy = PID(3e4, 1e5, 1e5, setpoint=0, sample\_time=1, output\_limits=(-1000, 1000))  
pmz = PID(1e5, 1e4, 5e4, setpoint=0, sample\_time=1, output\_limits=(-1000, 1000))  
  
pmx.time\_fn = faketime  
pmy.time\_fn = faketime  
pmz.time\_fn = faketime  
  
def read\_sensors(r):  
 r.recvuntil(b'Sensor:')  
 r.recvline()  
 return tuple(map(float, r.recvline().decode().split(',')))  
  
VSLOTS = [range(500, 1000), range(2500, 9999)]  
WSLOTS = [range(0, 9999)]  
  
def react(t, vx, vy, vz, wx, wy, wz, bx, by, bz):  
 global TIME  
 t = int(t)  
 TIME = t  
  
 adjv = any(t in rng for rng in VSLOTS)  
 adjw = any(t in rng for rng in WSLOTS)  
  
 if adjv:  
 rwx = pwx(-vx, dt=1)  
 rwy = pwy(-vy, dt=1)  
 rwz = pwz(-vz, dt=1)  
 else:  
 rwx = rwy = rwz = 0  
  
 if adjw:  
 b = np.array([bx, by, bz], dtype='float64')  
 w = np.array([wx, wy, wz], dtype='float64')  
 rmx, rmy, rmz = np.cross(w, b)  
  
 rmx = pmx(-rmx, dt=1)  
 rmy = pmy(-rmy, dt=1)  
 rmz = pmz(-rmz, dt=1)  
 else:  
 rmx = rmy = rmz = 0  
  
 return rwx, rwy, rwz, rmx, rmy, rmz  
  
  
r = remote('management.quals2023-kah5Aiv9.satellitesabove.me', 5300)  
r.sendlineafter(b'please:\n', b'ticket{golf324482oscar4:GKUkXwTflQTacpeZCTn70CIbqdHDTYJ-pN58Nvss2iwjqrR1rYUPjuuaYtF7MP8UTA}')  
  
for t in range(3601):  
 data = read\_sensors(r)  
 t, \*v = data[:4]  
 w = data[4:4+3]  
 m = data[4+3:4+3+3]  
  
 vmag = (v[0]\*\*2 + v[1]\*\*2 + v[2]\*\*2)\*\*0.5  
  
 resp = react(\*data)  
 rw = resp[:3]  
 rmag = resp[3:]  
  
 log.info('<- Sensors : t=%4.0f, |v|=%10.04f, v=(%10.04f, %10.04f, %10.04f), w=(%10.04f, %10.04f, %10.04f), m=(%10.2e, %10.2e, %10.2e)', t, vmag, \*v, \*w, \*m)  
 log.info('-> Response: w=(%10.04f, %10.04f, %10.04f), mag=(%10.04f, %10.04f, %10.04f)', \*rw, \*rmag)  
  
 r.sendline(', '.join(map('{:.30f}'.format, resp)).encode())  
  
r.interactive()