Task 2

Hovering the drone by controlling it from MATLAB

**Task 2A: Setting up the remote API**

1. Initially, the remote API was established by connecting MATLAB and Vrep to a particular port number. However, after the latest update of VREP, the ZeroMQ (ZMQ) API was released, which provides an easier way to establish the connection between MATLAB and VREP.
2. With this, we can use the remote API functions used in the CoppeliaSim Scripts rather than the legacy remote API functions.
3. As done in the normal remote API, the files RemoteAPIClient.m, RemoteAPIObject.m and cbor.m are added to the working directory.
4. The following code is then added to a new file in the same directory.

fprintf('Program started\n')

client = RemoteAPIClient();

sim = client.getObject('sim');

sim.startSimulation();

##Additional Code

sim.stopSimulation();

1. The above code is for MATLAB and VREP to run in an asynchronous mode i.e. MATLAB and VREP run independently and send commands to the other without waiting for the other to complete some task.
2. In the synchronous mode, MATLAB and VREP proceed after the other send a command to continue. Thus, they wait for the other to complete a particular task and then proceed.

fprintf('Program started\n')

client = RemoteAPIClient();

sim = client.getObject('sim');

sim.startSimulation();

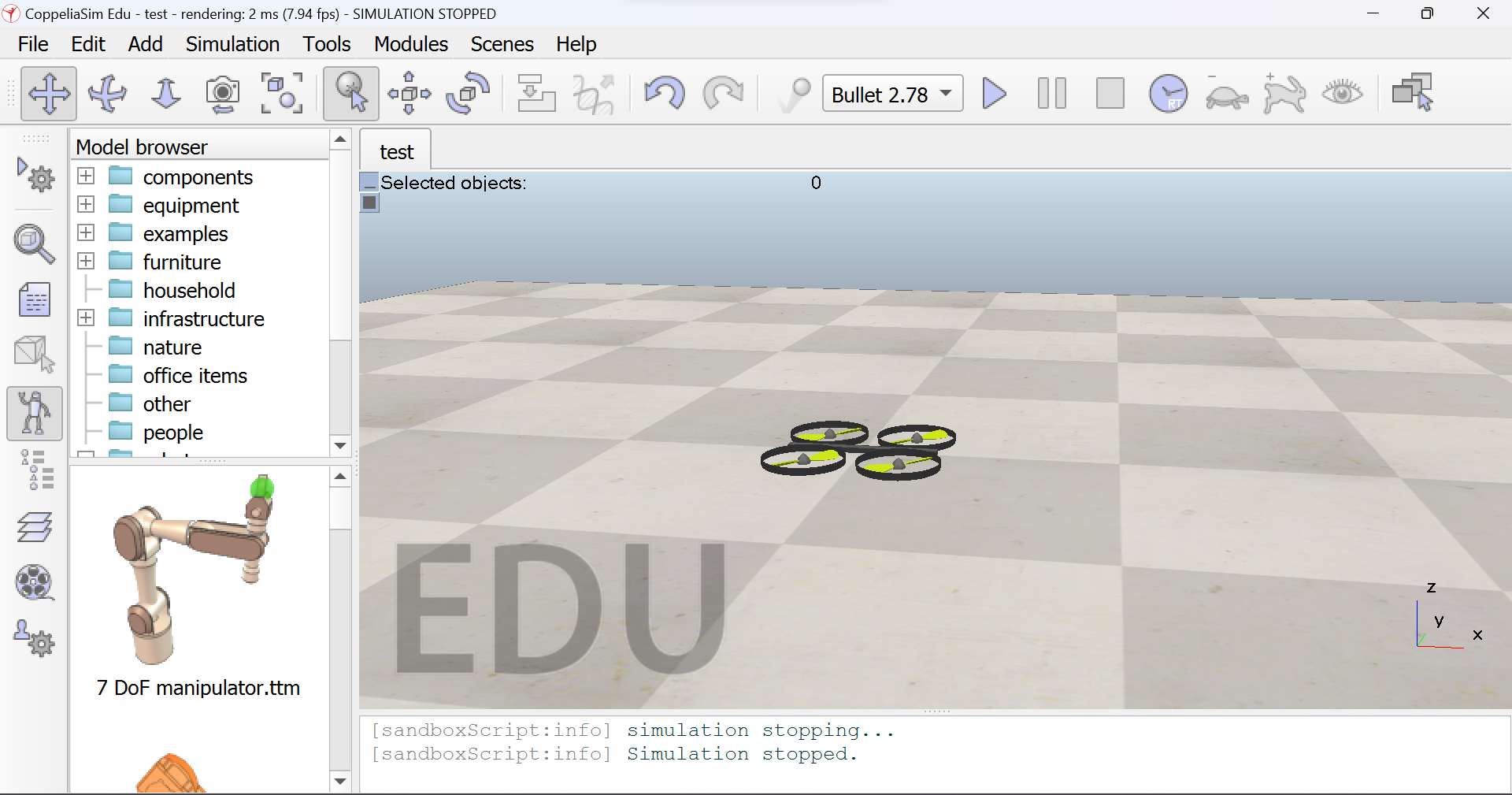
client.setStepping(true);

##Additional Code

client.step();(Proceeds to the next step)

sim.stopSimulation();

**Task 2B: Creating a PID controller for the drone**



* Getting the required object handles from vrep to matlab

targetObj=sim.getObject('/target');

d=sim.getObject('/Quadcopter/base');

heli=sim.getObject('/Quadcopter');

* Designing the altitude control with PID parameters

targetPos=sim.getObjectPosition(targetObj,sim.handle\_world);

targetPos=cell2mat(targetPos(3));

p=sim.getObjectPosition(d,sim.handle\_world);

This part of the code gets the z-axis position of the drone and the required target with respect to the real world

vel=sim.getVelocity(heli);

l=cell2mat(vel(1,3));

e=(targetPos-pos);

angvel=7.02+kp\*e+kd\*(e-laste)+vparam\*l;

laste=e;

Here we set the angular velocity of each propeller, thus maintaining a constant altitude. The parameters kp,kd were decided based on trial and error in the simulation in order to give the best control. Vparam was additionally added to provide a smoother reach to the required point. The value 7.02 was decided based on the weight of the given drone in order for it to maintain a constant altitude.

* Designing roll pitch and yaw

First, the target position was observed. Based on the difference between the actual and target position a reference roll/pitch was set. Taking this reference, a negative feedback loop of the controller was designed in order to reduce the error. Thus the following parameters were given for roll.

roll=0.01\*(cell2mat(tar(2))+5\*(cell2mat(tar(2))-prevy));

prevy=cell2mat(tar(2));

rollE=roll-cell2mat(m(1));

rollCof=1.5\*rollE+6\*(rollE-prevRollE)+0.09\*cell2mat(vel(2));

prevRollE=rollE;

cell2mat(m(1));

Similarly for pitch control

pitch=-0.01\*(cell2mat(tar(1))+5\*(cell2mat(tar(1))-prevx));

prevx=cell2mat(tar(1));

pitchE=pitch-cell2mat(m(2));

pitchCof=1.5\*pitchE+6\*(pitchE-prevPitchE)-0.09\*cell2mat(vel(1));

prevPitchE=pitchE;

For yaw control

rotCorr=cell2mat(m(3))\*0.1+2\*(cell2mat(m(3))-prevEuler);

prevEuler=cell2mat(m(3));

* Deciding angular velocities of each of the propeller

Thrust:

Equal value is added to each of the propeller which enables the drone to lift.

Pitch and roll:

The pitch and roll velocity for two propellers is added and the other two subtracted. The propellers are adjacent to each other.

Yaw:

The yaw velocity for two opposite propellers is added and the other two subtracted.

angvel1=angvel-pitchCof+rollCof+rotCorr;

angvel2=angvel+pitchCof+rollCof-rotCorr;

angvel3=angvel+pitchCof-rollCof+rotCorr;

angvel4=angvel-pitchCof-rollCof-rotCorr;

* Setting the forces,thrust,velocities

The commands for these are set from inside CoppelliaSim since this increases the time response of the simulation.

For sending the calculated values above we use the following script function.

sim.callScriptFunction('setVel@/Quadcopter',sim.scripttype\_childscript,angvel1,angvel2,angvel3,angvel4);

* Code in the script

function sysCall\_init()

d=sim.getObject('./base')

propellerHandles={}

jointHandles={}

for i=1,4,1 do

propellerHandles[i]=sim.getObject('./propeller['..(i-1)..']/respondable')

jointHandles[i]=sim.getObject('./propeller['..(i-1)..']/joint')

end

heli=sim.getObject('.')

end

function setVel(angvel1,angvel2,angvel3,angvel4)

handlePropeller(1,angvel1)

handlePropeller(2,angvel2)

handlePropeller(3,angvel3)

handlePropeller(4,angvel4)

end

function handlePropeller(index,angvel)

propellerRespondable=propellerHandles[index]

propellerJoint=jointHandles[index]

propeller=sim.getObjectParent(propellerRespondable)

local t=sim.getSimulationTime()

sim.setJointPosition(propellerJoint,t\*10)

ts=sim.getSimulationTimeStep()

m=sim.getObjectMatrix(propeller,sim.handle\_world)

-- Apply a reactive force onto the body:

totalExertedForce=angvel\*angvel\*0.0257

force={0,0,totalExertedForce}

m[4]=0

m[8]=0

m[12]=0

force=sim.multiplyVector(m,force)

local rotDir=1-math.mod(index,2)\*2

torque={0,0,rotDir\*0.02\*angvel}

torque=sim.multiplyVector(m,torque)

sim.addForceAndTorque(propellerRespondable,force,torque)

end

In the function handle propeller, the total thrust produced by each propeller in given by the equation totalExertedForce=angvel\*angvel\*0.0257

The forces and torques are then transformed from the frame of the drone to the real world and then are applied using the sim.addForceAndTorque function.

**Task 2C: Making the drone follow a predefined path**

Since we have a target object for the drone to follow, the required trajectory is set on the target.

Code:

t=sim.getSimulationTime();

sim.setObjectPosition(targetObj,sim.handle\_world,{r\*cos(0.1\*t) r\*sin(0.1\*t) 1})

**Final Code in MATLAB:**

fprintf('Program started\n')

client = RemoteAPIClient();

sim = client.getObject('sim');

%Handles

targetObj=sim.getObject('/target');

d=sim.getObject('/Quadcopter/base');

heli=sim.getObject('/Quadcopter');

sim.setObjectOrientation(targetObj,sim.handle\_world,{0 0 0});

kp=2;

kd=10;

laste=0;

vparam=-2;

prevy=0;

prevRollE=0;

prevx=0;

prevPitchE=0;

prevEuler=0;

pos=sim.getObjectPosition(d,sim.handle\_world);

client.setStepping(true);

sim.startSimulation();

r=0.5;

while sim.getSimulationTime()==0 || sim.getSimulationState()==sim.simulation\_stopped

pause(0.01);

client.step();

end

while true

t=sim.getSimulationTime();

sim.setObjectPosition(targetObj,sim.handle\_world,{r\*cos(0.1\*t) r\*sin(0.1\*t) 1});

%Altitude Control

targetPos=sim.getObjectPosition(targetObj,sim.handle\_world);

targetPos=cell2mat(targetPos(3));

p=sim.getObjectPosition(d,sim.handle\_world);

pos=p(3);

pos=cell2mat(pos);

vel=sim.getVelocity(heli);

l=cell2mat(vel(1,3));

e=(targetPos-pos);

angvel=7.02+kp\*e+kd\*(e-laste)+vparam\*l;

laste=e;

tar=sim.getObjectPosition(d,targetObj);

m=sim.getObjectOrientation(d,targetObj);

%roll

roll=0.01\*(cell2mat(tar(2))+5\*(cell2mat(tar(2))-prevy));

prevy=cell2mat(tar(2));

rollE=roll-cell2mat(m(1));

rollCof=1.5\*rollE+6\*(rollE-prevRollE)+0.09\*cell2mat(vel(2));

prevRollE=rollE;

cell2mat(m(1));

%pitch

pitch=-0.01\*(cell2mat(tar(1))+5\*(cell2mat(tar(1))-prevx));

prevx=cell2mat(tar(1));

pitchE=pitch-cell2mat(m(2));

pitchCof=1.5\*pitchE+6\*(pitchE-prevPitchE)-0.09\*cell2mat(vel(1));

prevPitchE=pitchE;

%yaw

rotCorr=cell2mat(m(3))\*0.1+2\*(cell2mat(m(3))-prevEuler);

prevEuler=cell2mat(m(3));

angvel1=angvel-pitchCof+rollCof+rotCorr;

angvel2=angvel+pitchCof+rollCof-rotCorr;

angvel3=angvel+pitchCof-rollCof+rotCorr;

angvel4=angvel-pitchCof-rollCof-rotCorr;

sim.callScriptFunction('setVel@/Quadcopter',sim.scripttype\_childscript,angvel1,angvel2,angvel3,angvel4);

client.step(); % triggers next simulation step

end

sim.stopSimulation();