Vertical Take-Off and Landing (VTOL) Aircraft Control
MECA 482 Control System Design Project

Francisco Biordi, Cameron Schindler, Jacob Olsen
Section 3323 and 5182
MECA 482
Professor H. Sinan Bank

Introduction

This report was created to document a senior-level control systems engineering semester project. This project involves designing a control system for vertical take-off and landing (VTOL) aircraft. A VTOL-class aircraft that can hover mid-air as well as take-off and land vertically. A control system is required to successfully operate an aircraft with VTOL capabilities due to the unusual designs of most rendering it unable to be flown manually. The VTOL aircraft control system involves the control and positioning of a shaft mounted to support in a way that allows for rotation in the plane normal to the support and coincident with the shaft. A propeller is mounted to one end of the shaft and the other end is fixed to the support. The support is placed normal to the surface of the earth so it is parallel with the gravity vector. A counterweight could potentially be added to the other end of the shaft to improve the stability of the system due to the inertial effects of the mass and the less force required by the motor. The control of the propeller thrust is the main method for orienting the aircraft body in the case of this model. The idea of VTOL aircraft has been around since the early 1900s but the first good examples were not created until after World War II. The most common type of VTOL aircraft is the helicopter which is used both in military and civilian applications. There are a handful of jets that are capable of VTOL operation such as the Bell Boeing V-22 Osprey and the different variants of the Harrier jet.

A few different papers were referenced online to complete this project because we had to figure out how to derive the transfer function for the system, then interface MATLAB with V-REP to demonstrate the functionality of the system. A list of the specific papers can be found in the references section of this paper. Some of the models used in V-REP were made using SolidWorks and then adapted to be compatible with V-REP.

Modeling

First, a high-level simple model of the system was derived to determine the system's general response. Then the more accurate transfer function was derived using MATLAB with the "tf" function. The basic idea behind this system is that the propeller will overcome the force of gravity and stabilize at the desired angle. The initial calculations took some time to solve because we did not account for the system needing a dynamic impulse to arrive at the correct position and instead were calculating the holding force required.

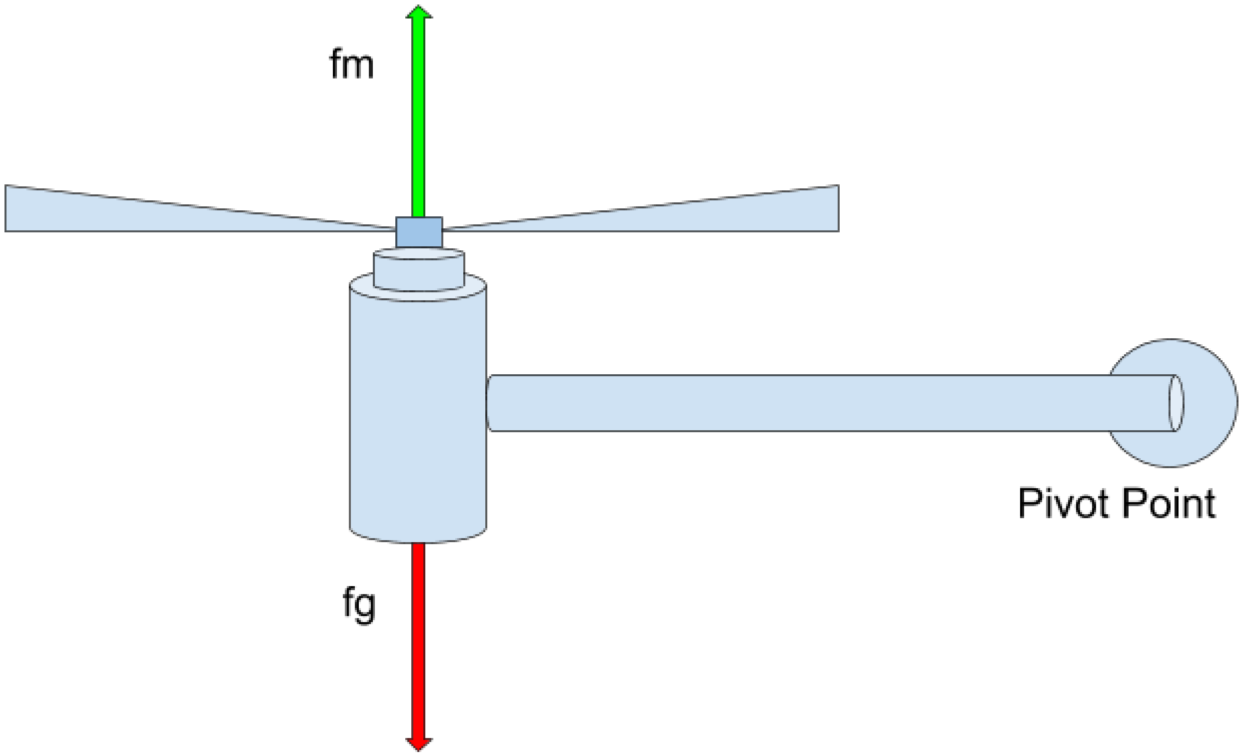

Figure 1: The force diagram of the system.

Sensor Calibration

This system will utilize a rotary potentiometer as an encoder to determine its position relative to the Earth. Rotary potentiometers are passive electrical components that are able to vary their resistance value as the shaft of the potentiometer rotates. This resistance value can be read from the potentiometer by a microcontroller such as an Arduino when wired to the analog inputs.

Controller Design and Simulations

The controller was first designed at a very high level using block diagrams to lay out the flow of the entire system. This was then converted to individual component choices for the theoretical model and to determine the capability of each. This system requires a propeller-based thruster to change the angle of the arm relative to the support stand that is attached to the ground. The controller was first derived by hand and then implemented with MATLAB.

We already had a machine with MATLAB installed on it but it was an older version that did not include Simulink. Because of this, we downloaded the trial of the most recent version of MATLAB on another machine in order to use Simulink. The machine that was used to complete the project had a limited amount of RAM available which proved to be a challenge when performing more resource-demanding simulations.

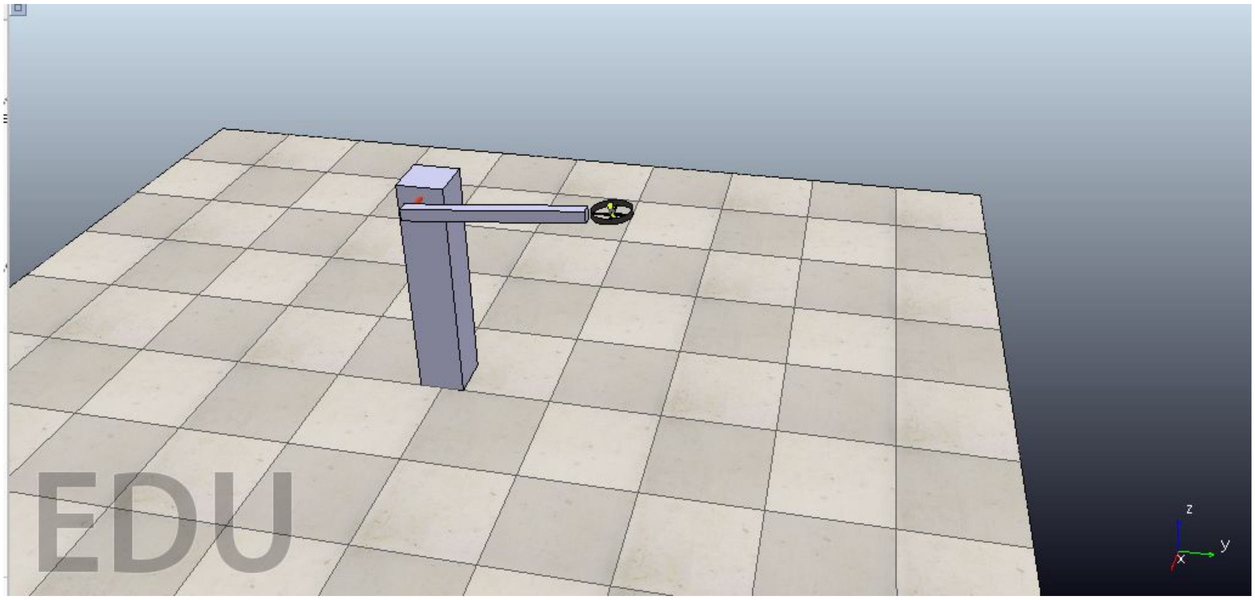

Figure 2: The V-REP implementation of the VTOL system in motion.

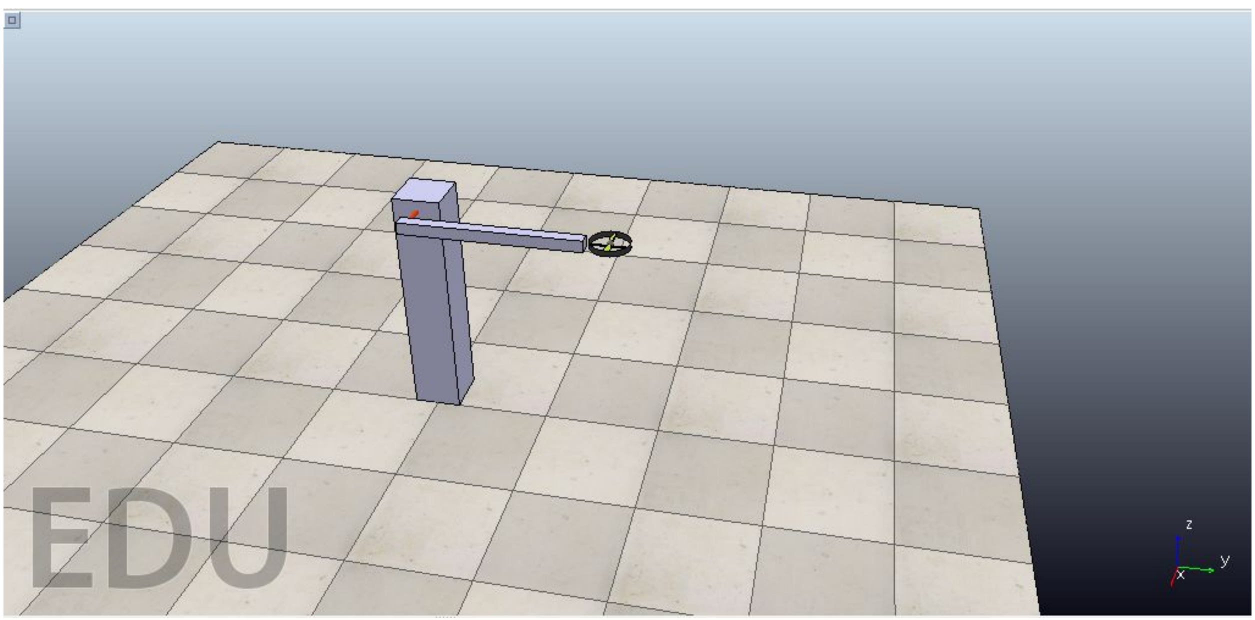

Figure 3: The V-REP implementation of the VTOL at rest.

Non-threaded child script (propeller)

```
function sysCall_init()
    propeller=sim.getObjectHandle('propeller')
    propellerRespondable=sim.getObjectHandle('propeller_respondable')
    propellerJoint=sim.getObjectHandle('propeller_joint')
    type=sim.particle_roughspheres+sim.particle_respondable1to4+sim.particle_respondable5to8+
        sim.particle_cyclic+sim.particle_ignoresgravity
    simulateParticles=sim.getScriptSimulationParameter(sim.handle_self,'simulateParticles')
    particleCountPerSecond=sim.getScriptSimulationParameter(sim.handle_self,'particleCountPer
    particleDensity=sim.getScriptSimulationParameter(sim.handle_self,'particleDensity')
    particleVelocity=sim.getScriptSimulationParameter(sim.handle_self,'particleVelocity')
    particleScatteringAngle=sim.getScriptSimulationParameter(sim.handle_self,'particleScatter
    particleLifeTime=sim.getScriptSimulationParameter(sim.handle_self,'particleLifeTime')
    maxParticleCount=sim.getScriptSimulationParameter(sim.handle_self,'maxParticleCount')


    if (sim.getScriptSimulationParameter(sim.handle_self,'particlesAreVisible')==false) then
        type=type+sim.particle_invisible
    end
    maxParticleDeviation=math.tan(particleScatteringAngle*0.5*math.pi/180)*particleVelocity
    particleObject=nil
    is=0 -- previous size factor
    notFullParticles=0
    params={2,1,0.2,3,0.4}
end

function sysCall_cleanup()

end

function sysCall_actuation()

    forcey=sim.getIntegerSignal('ForceY_Send')
    forcez=sim.getIntegerSignal('ForceZ_Send')
    sim.addForceAndTorque(propellerRespondable,{0,forcey,forcez},{0,0,0})
end
```

Figure 4: The code for the V-REP model of the propeller.

Initially, the built-in propeller used particles to provide the force. However, in order to simplify the model we opted to focus on the force it was emitting. The function used to generate force does not take into account the orientation of the propeller with respect to the world. As a result of this, we had to manually take into account what the correct direction for the force would be. Using the angle of the beam we were able to calculate the Force in the Y and Z directions in Matlab and send the values to V-REP using the signal feature. This technique allows us to have accurate control of the force emitted in V-REP using Matlab.

Controller Implementation

While we would have liked to have built a physical implementation of this system but we ran out of time because every member of the group had other projects also due before this one. We have ideas of how we would do it if we had time to construct the system. This would also provide a good learning experience because we would most likely see some small things that we did not think about but still have a small effect on the system.

Appendix A: Simulation Code

```
% Make sure to have the server side running in CoppeliaSim:
% in a child script of a CoppeliaSim scene, add following command
% to be executed just once, at simulation start:
%
% simRemoteApi.start(19999)
%
% then start simulation, and run this program.
function simpleTest()
   disp('Program started');
   sim=remApi('remoteApi'); % using the prototype file (remoteApiProto.m)
   sim.simxFinish(-1); % just in case, close all opened connections
   clientID=sim.simxStart('127.0.0.1',19999,true,true,5000,5); %initialize a connection

   if (clientID>-1)
      disp('Connected to remote API server');

      % Now try to retrieve data in a blocking fashion (i.e. a service call):
      [res,objs]=sim.simxGetObjects(clientID,sim.sim_handle_all,sim.simx_opmode_blocking);
      if (res==sim.simx_return_ok)
         fprintf('Number of objects in the scene: %d\n',length(objs));
      else
         fprintf('Remote API function call returned with error code: %d\n',res);
      end
      pause(2);

      % Now retrieve streaming data (i.e. in a non-blocking fashion):

      [res, Joint] = sim.simxGetObjectHandle(clientID, 'Revolute_joint',
sim.simx_opmode_blocking); %Obtain Handle information of the Revolute Joint to gain the
relative angle information

      GoalAngle = 0; %Angle at which we want the bar to be held at

      Force = 0; %Starting Force Amount

      MaxForce = 5; %Max Force the propeller can "emit"

      while(true)
      [res, Angle] = sim.simxGetJointPosition(clientID, Joint, sim.simx_opmode_streaming);
%Obtain Relative Angle
```

```
        Angle = Angle*180/pi; %Convert Angle from Rad to Degrees


        if Angle > GoalAngle %%If the angle of the beam is above the angle of the goal, the force
of the propeller is decreased
            if Force<MaxForce
                Force = Force - 0.005;
                Status = 0;
            end
        elseif Angle < GoalAngle %%If the angle of the beam is below the angle of the goal, the
force of the propeller is increased
            if Force<MaxForce
                Force = Force + 0.005;
                Status = 1;
            end
        else Angle == GoalAngle %%If the angle of the beam is equal to the angle of the goal, the
force of the propeller is held constant
            Force = Force;
            Status = 2;
        end




        ForceY = abs(cosd(90-Angle)*Force); %%Transforms the force required into the
components to send to VREP
        ForceZ = abs(sind(90-Angle)*Force);

        sim.simxSetIntegerSignal(clientID, 'ForceY_Send', ForceY, sim.simx_opmode_oneshot);
%%Send the Force to VREP
        sim.simxSetIntegerSignal(clientID, 'ForceZ_Send', ForceZ, sim.simx_opmode_oneshot);


        DispArray = [ForceY ForceZ Force Angle Status GoalAngle]; %%Display Important Data in
the Matlab Command Window

        disp(DispArray);
        pause(0.1);
        end
```

```
        % Now send some data to CoppeliaSim in a non-blocking fashion:
        sim.simxAddStatusbarMessage(clientID,'Clean up time!',sim.simx_opmode_oneshot);

        % Before closing the connection to CoppeliaSim, make sure that the last command sent
out had time to arrive. You can guarantee this with (for example):
        sim.simxGetPingTime(clientID);

        % Now close the connection to CoppeliaSim:
        sim.simxFinish(clientID);
    else
        disp('Failed connecting to remote API server');
    end
    sim.delete(); % call the destructor!

    disp('Program ended');
end
```

References

[1] MECA 482 Lab Proposal, Vertical Take-Off and Landing Experimental Setup, Retrieved Oct. 18, 2019, from
BlackBoardLearn class section 198-MECA482-02-3323

[2] Wikipedia, VTOL, Retrieved Oct. 18, 2019, from
https://en.wikipedia.org/wiki/VTOL

[3] ResearchGate, Modelling and Control of a Convertible VTOL Aircraft, Retrieved Oct. 20, 2019, from
https://www.researchgate.net/publication/224700285_Modelling_and_Control_of_a_Convertible_VTOL_Aircraft

[4] National Aeronautics and Space Administration (NASA), Propeller Thrust, Retrieved Dec. 4, 2019, from https://www.grc.nasa.gov/WWW/K-12/airplane/propth.html

[5] Coppelia Robotics, Coppeliasim/V-REP User Manual, Retrieved Dec. 9, 2019, from
http://coppeliarobotics.com/helpFiles/index.html

[6] Coppelia Robotics Forum, “How to use MATLAB with V-REP”, Retrieved Dec. 20, 2019, from
http://forum.coppeliarobotics.com/viewtopic.php?t=3325

[7] Coppelia Robotics Help Files, Remote API Functions (MATLAB), Retrieved Dec. 20, 2019, from http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsMatlab.htm