**PID Ping Pong Ball Levitation**



Ryan Edley

Sara M. Johari

Mitchell Moletta

Isaac Ramirez

Zahory Ramirez

Fall 2019 MECA 482 Controls Systems

Department of Mechanical and Mechatronic Engineering and Sustainable Manufacturing

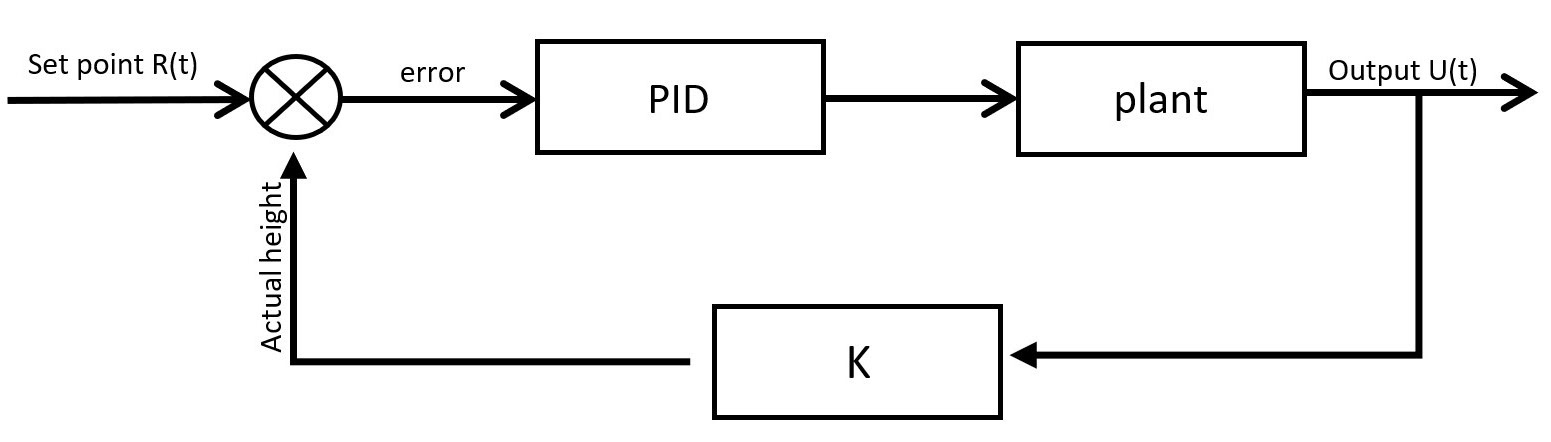
California State University, Chico

**Introduction**

In this project a Proportional-Integral-Derivative (PID) controller will be used to levitate a ping pong ball in a plastic tube to a specified height. This is only the theoretical portion of this project and no physical model will be made or used in this experiment. If this was going to be built, a DC motor with a fan blade, an optical rangefinder, and a ping pong ball in a plastic tube would be needed to conduct the experiment. These components are modeled in Solidworks and exported to Vrep. The system will be represented by a mathematical model and high level architecture shown in **Figure 1**. The mathematical model is created in Simulink and connected to Vrep through Matlab Remote API commands to show a visual representation of the working control system.

**Modeling**

In control theory one usually wishes to vary a physical variable of interest while holding all others constant. Fundamentally, feedback is one of the great ideas developed to make this desired theory feasible. Therefore, in this PID Ping Pong Ball Levitation system a feedback loop must be included. This feedback will modify the PID control loop that regulates some experimental quantity, in this case height. Appendix A



**Figure 1:** High Level Architecture of PID Ping Pong Ball Levitation

**Sensor Calibration**

The sensor needs to be calibrated by relating the voltage signal output for the corresponding distance of the ping pong ball away from the sensor. This can be done by measuring known distances and placing the ping pong ball inline from the optical range finder sensor. Once that is done the voltage at each known distance must be noted. These values are noted and then graphed as voltage versus distance using Excel. To ensure the target set point is reached by the system, the graphed data will be fitted to generate the proper calibration curve desired.

**Controller Design and Simulation**

The plan for this simulation was to have a user input a desired height of the ball. This value would then be shown in a scope where height is a variable with respect to time. The PID controller would adjust the height of the ball by changing the motor speed which increases or decreases the airflow from the fan blade. The motor transfer function receives some voltage input and outputs an angle of the motor. This angle then gets altered in a way that is directly proportional to fan speed and is sent into the ball-and-tube transfer function as a force input. The ball’s height is outputted and plotted on the scope for comparison to the user defined height.

This output height is sent over to Vrep for a visual representation of the control system in action! The block diagram can be seen in Appendix A.

**Documentation**

**Exporting Solidworks Model**

A solidworks model assembly was created to represent the tube, ball, and blower fan.  The assembly was then exported as URDF with the following characteristics:

-Base link is the tube components

            Child links:

                      Fan     -      Link type ‘Revolute’

-      mass 0.03 kg

                      Child links:

Ball

o   Link type ‘Planar’

o   mass 0.0027 kg

-Export URDF with meshes

**Vrep Initialization**

Next the Vrep program was opened. The solidworks model was imported by selecting Plugins>URDF import.

The following initializations were completed.

Orient the model by selecting ‘base\_link\_respondible’ and clicking the ‘rotate’ button (Alpha = 0; beta = 0; Gamma = 0)

Pan the Model by selecting ‘base\_link\_respondible’ and clicking the ‘translate’ button (X = 0; Y = 0; Z = 0.48)

Set the fan dynamics by selecting ‘Fan\_respondible’ and clicking ‘show dynamic property dialog’ and checking the ‘Body is dynamic’ box and unchecking the ‘Body is respondible’ box.

Select ‘Fan\_visual’ and click ‘show dynamic property dialog’ and uncheck the ‘Body is dynamic’ box and uncheck the ‘Body is respondible’ box.

This will show the fan rotates in place upon running the simulation.

Set the ball dynamics by selecting ‘Ball\_respondible’ and clicking ‘show dynamic property dialog’ and unchecking the ‘Body is dynamic’ box and unchecking the ‘Body is respondible’ box.

Select ‘Body\_visual’ and click ‘show dynamic property dialog’ and uncheck the ‘Body is dynamic’ box and uncheck the ‘Body is respondible’ box.

This will show the ball stays in place but spins with respect to the fan rotating upon running the simulation.

Next customize the main script by adding in the following command “simRemoteApi.start(19999)” before the first “function”.

Now Vrep is ready to receive commands from Matlab.

**Connecting Vrep to Matlab**

First, we copied remote API files from V-rep’s file location into a new folder that we opened in Matlab. Then we copied some general communication lines of code from the ‘simpleTest.m’ file into Matlab and Vrep which allowed communication between the two.

Steps:

Program files>V-REP3>programming>remoteApitBindings>matlab>matlab> (copy these three files into new file location)

remApi.m

remoteApiProto.m

simpleTest.m

Program files>V-REP3>programming>remoteApitBindings>lib>lib>64Bit> (copy this file into the new file location)

remoteApt.dll

Now Matlab should be communicating with Vrep. The next step is to add in Remote API functions into matlab that Vrep can understand in order to control Vrep from Matlab.

The Matlab code is shown below in Appendix.

**Remote API MATLAB Commands**

Methodology: Since the fan propeller in Vrep is just visual eye candy, air particles must be added to physically move the ball up and down. The ball’s initial position must be specified and a target height must be defined by the user. Within a while loop the simulink model is populated with initial variables and returns new variables (height of the ball). The loop continues until the ball height reaches steady state at the user defined height.

*Sim.addParticleObject* would be used to simulate airflow from the fan. The amount and velocity of particles is governed by inputs from the Simulink model. These inputs are changing continuously by the Simulink model to move the ball to its target height.

*sim.GetObjectPosition* was used to create an origin of the ball in Matlab. The output height from simulink was added to this origin for accurate ball location in Vrep.

Due to time constraints the particles were not used and instead the ball was just moved to a different fixed position with every iteration of the while loop. This method also proves that the simulink control system is working because the ball can be seen moving to new heights.

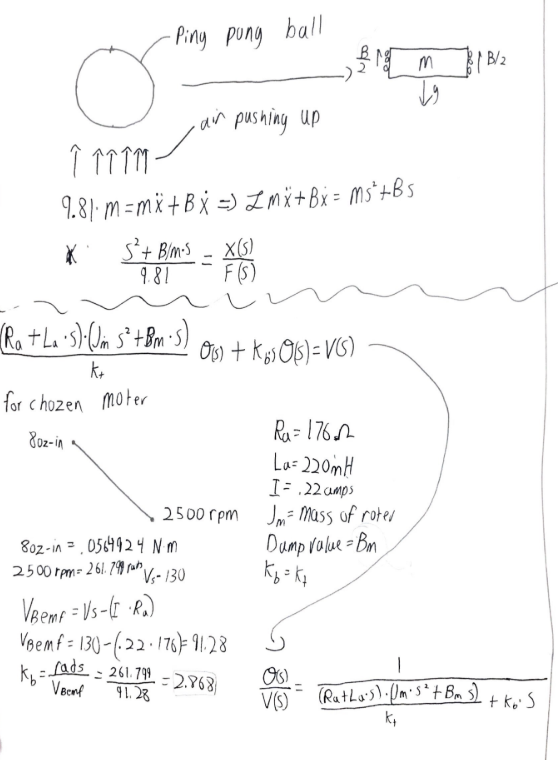
*sim.setObjectPostion* was used to visually show the ball being repositioned in vrep.

**Conclusion**

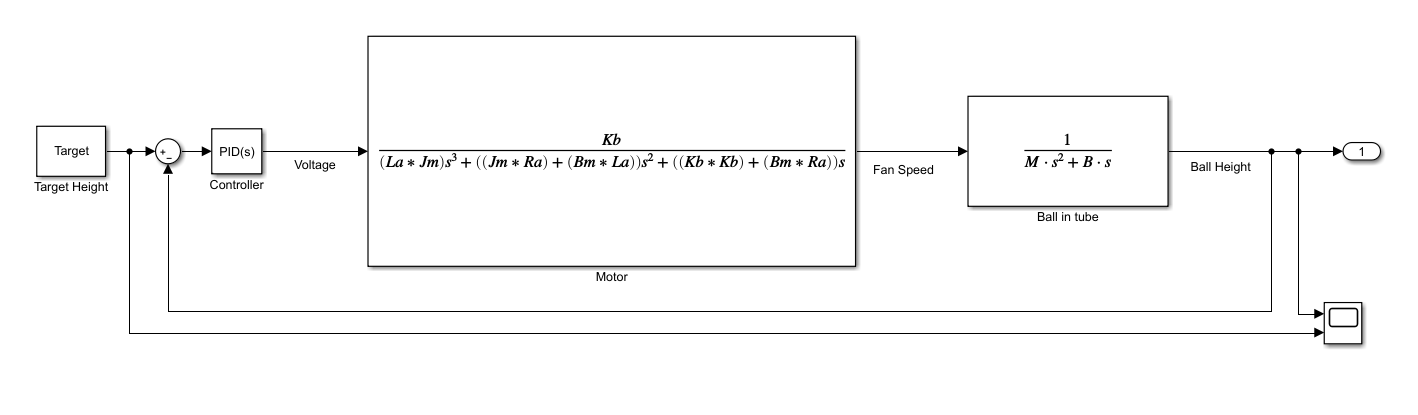
In conclusion the control system design for this project is non-functional. The transfer function for the motor seems to be correct, however we were unsure of how to convert the output motor angle theta into angular velocity. The ball-and-tube transfer function is missing the input portion where it receives angular velocity of the motor. Using the propeller dimensions the angular velocity would then be converted to airflow as an input force to the ball-and-tube transfer function where it could then output a height. If this block diagram would have worked as expected, we would have been able to simulate a rough visual representation of the ball moving vertically in the tube within VREP. Our next step would have been to go more in depth in VREP and add in variable Air Particle flow to physically move the ball with respect to motor speed. We started this project pretty late in the semester due to a switch from our previous interest in 6 DOF fin control for a rocket. Assuming this electromechanical control system would be easier, we waited too long to ask for help and got busy with finals. Unfortunately, this led to our non-functional control system.

**Appendix**

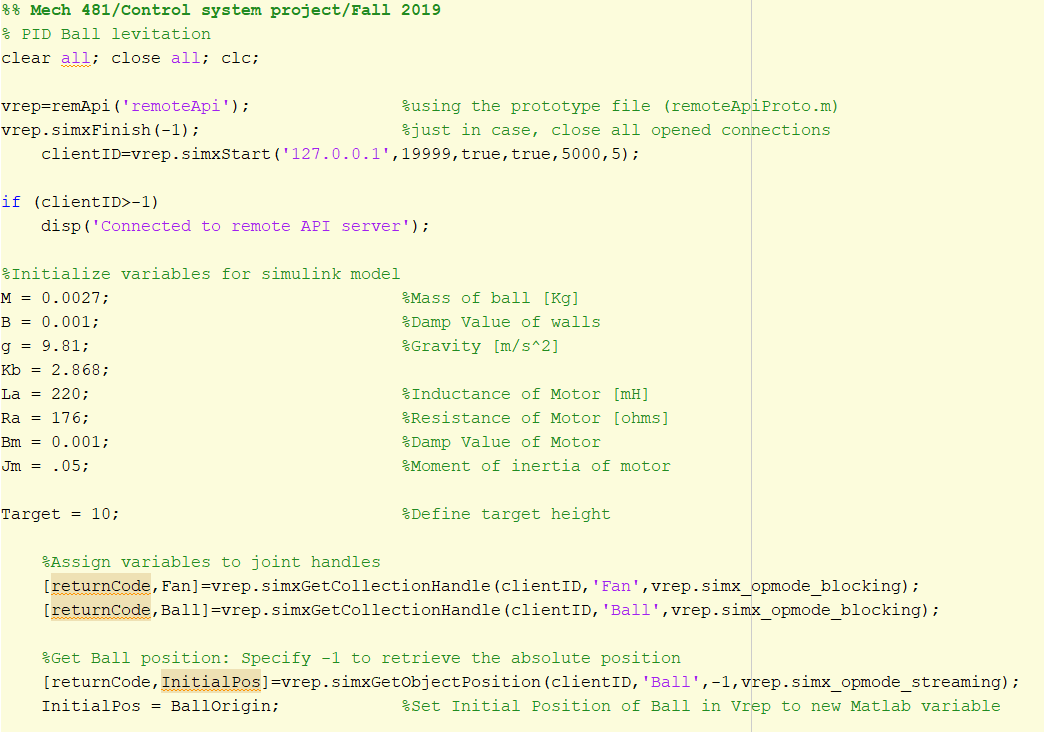
A: Transfer Function Modeling



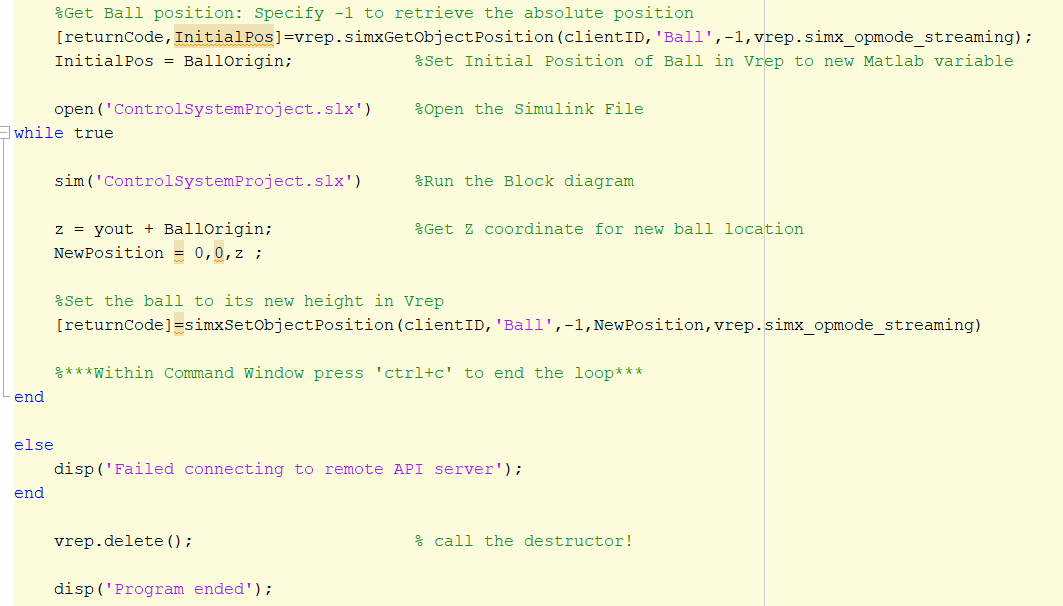
B: Simulink Block Diagram

****

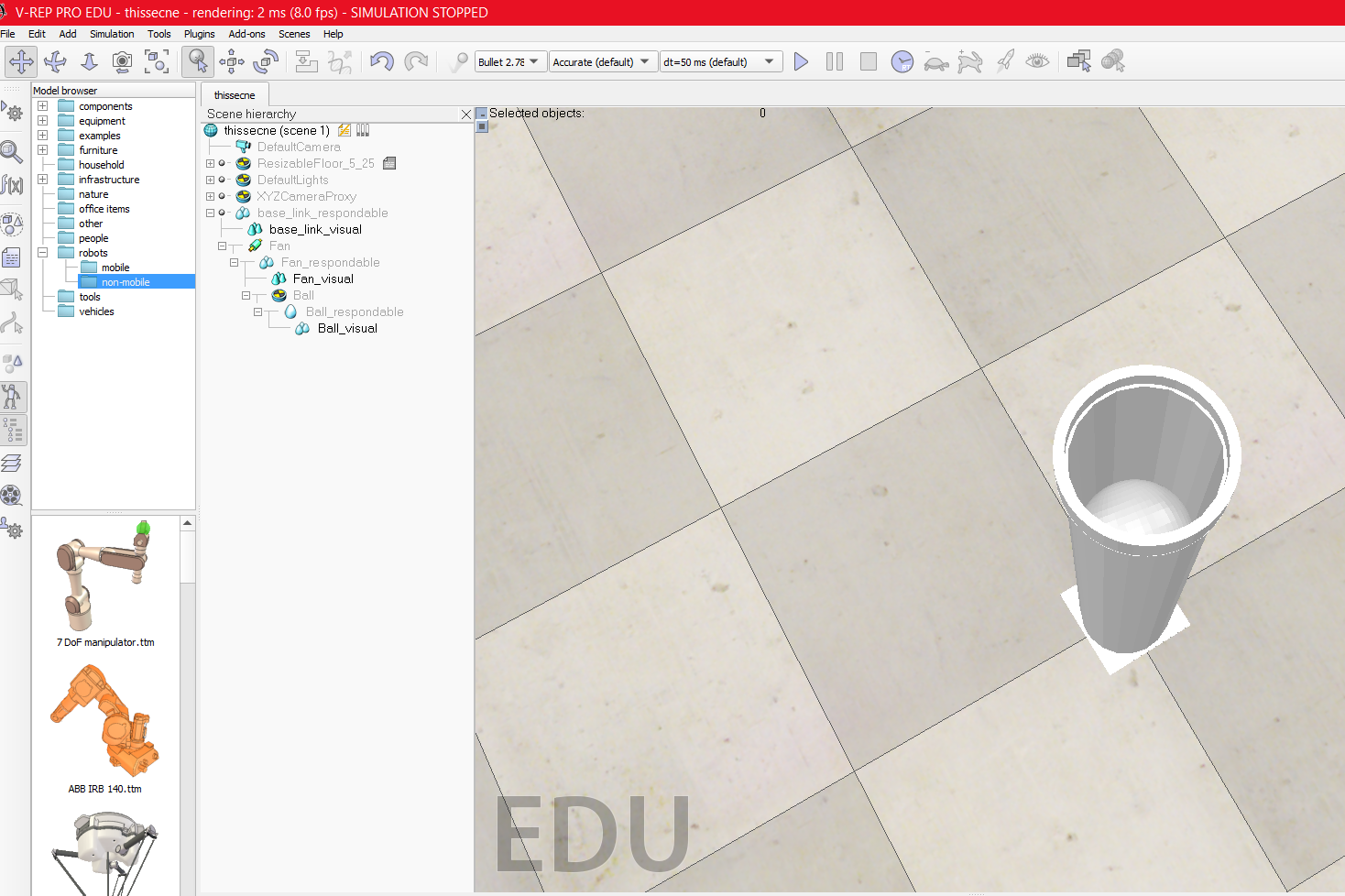
Ca: Matlab Code

****

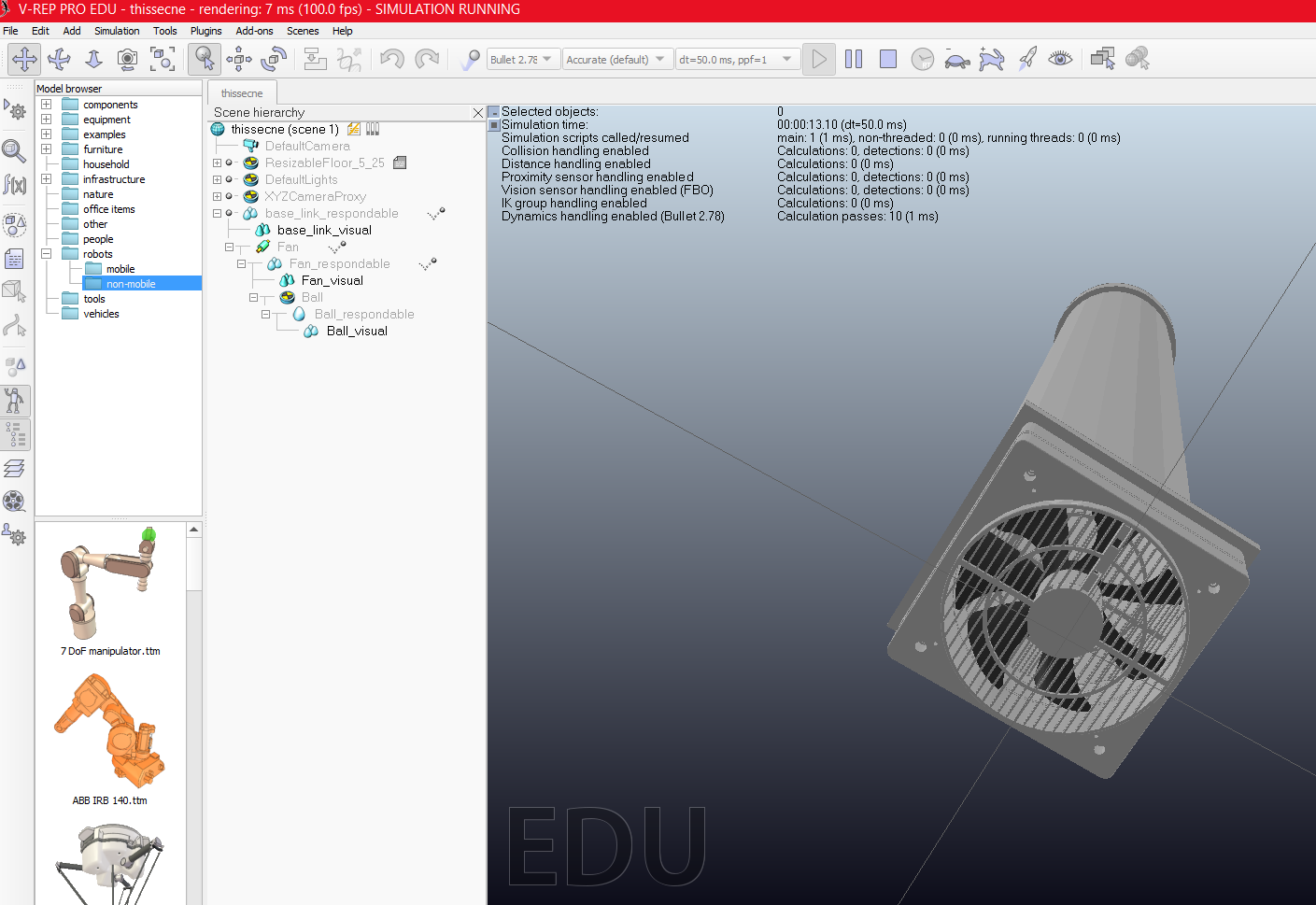
Cb: Matlab Code

****

Da: V-REP Simulation Screenshot

****

Db: V-REP Simulation Screenshot

****

**References**

1. Nise’s Control Systems Engineering

1. Ayars, Erik. “Physics 427.” *Physics 427: PID Ping-Pong Levitation*, 15 Nov. 2019, physics.csuchico.edu/ayars/427/labs/PID\_Ping-Pong.shtml.
2. Coppelia Robotics, Remote API functions (Matlab), 22 Dec. 2019, <http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsMatlab.htm>
3. Lum, Christopher, *Interacting with a Simulink Model from a Matlab Script*, 22 Dec. 2019, <https://www.youtube.com/watch?v=sF_sjFqNFUk>
4. K., Nikolai, *05: Matlab Robot Simulation with V-REP – Part 1*, <https://www.youtube.com/watch?v=piI5wYEXUms>
5. K., Nikolai, *05: Matlab Robot Simulation with V-REP – Part 2*, <https://www.youtube.com/watch?v=mal48Vd-DQY>
6. Ninja, Mechatronics, *V-rep (CoppeliaSim) matlab client, and animation of robot joints,* 22, Dec. 2019, <https://www.youtube.com/watch?v=7Z01cRw_i5E>