{Learn, Create, Innovate};

DC Motor Sim

Activity 1

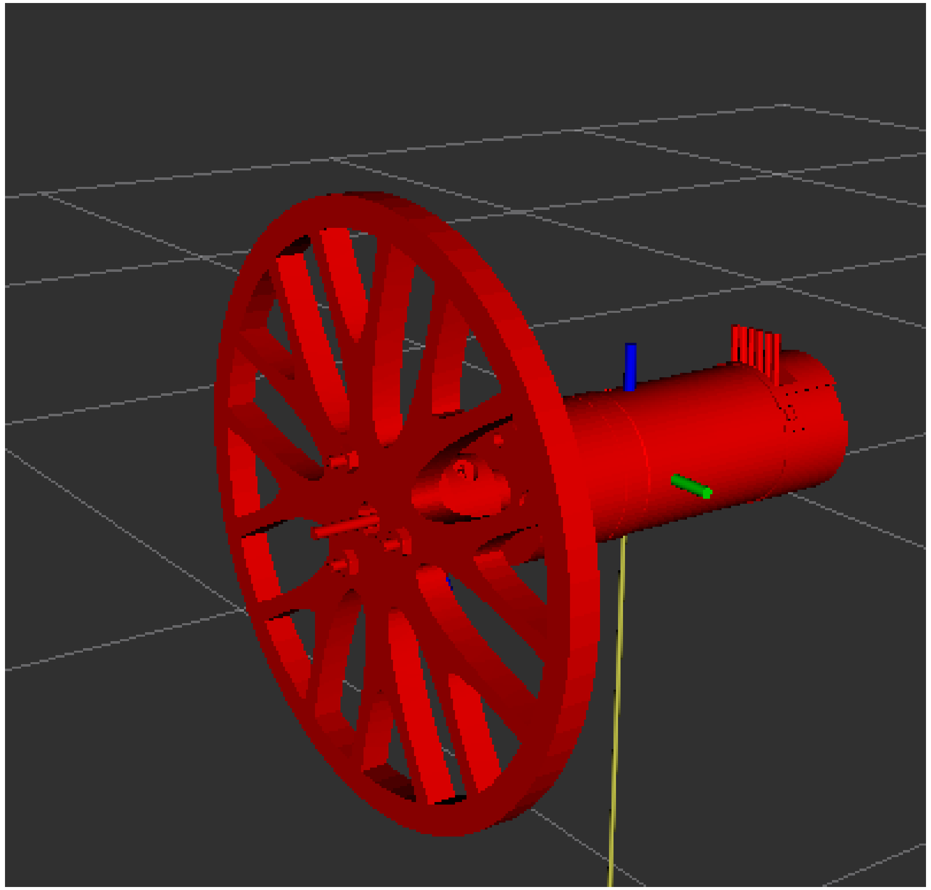
Introduction

Introduction

- This activity simulates a DC motor using its dynamical model.

$$\begin{cases} \dfrac{di}{dt} = -\dfrac{R}{L}i - \dfrac{k_1}{L}\omega + \dfrac{1}{L}u \\ \dfrac{d\omega}{dt} = \dfrac{k_2}{J}i - \dfrac{b}{J}\omega - \dfrac{1}{J}m \end{cases}$$

Where, i is the current, ω is the angular speed, J inertia, m is the mass u is the input voltage, R is the internal resistance, L is the internal inductance and k_1, k_2 are the electromagnetic constants.



Motor Dynamical System

- Download the ROS Package template “motor_sim” from GitHub and copy it to your workspace
    - Folder Activities››Activity 1››motor_sim

- Give executable permission to the files in script folder

```
$ cd ~/catkin_ws
$ sudo chmod +x src/markers/scripts/*
```

```
src/motor_sim/
├── CMakeLists.txt
├── launch
│   └── motor_sim.launch
├── models
│   ├── dcMotor.stl
│   └── MCR2_1000_1_1_Wheel_Coupler_2.stl
├── package.xml
├── rviz
│   └── motor.rviz
├── scripts
│   ├── motor_JointPub.py
│   ├── motor_sim.py
│   └── set_point_generator.py
├── src
└── urdf
    └── dc_motor.urdf
```

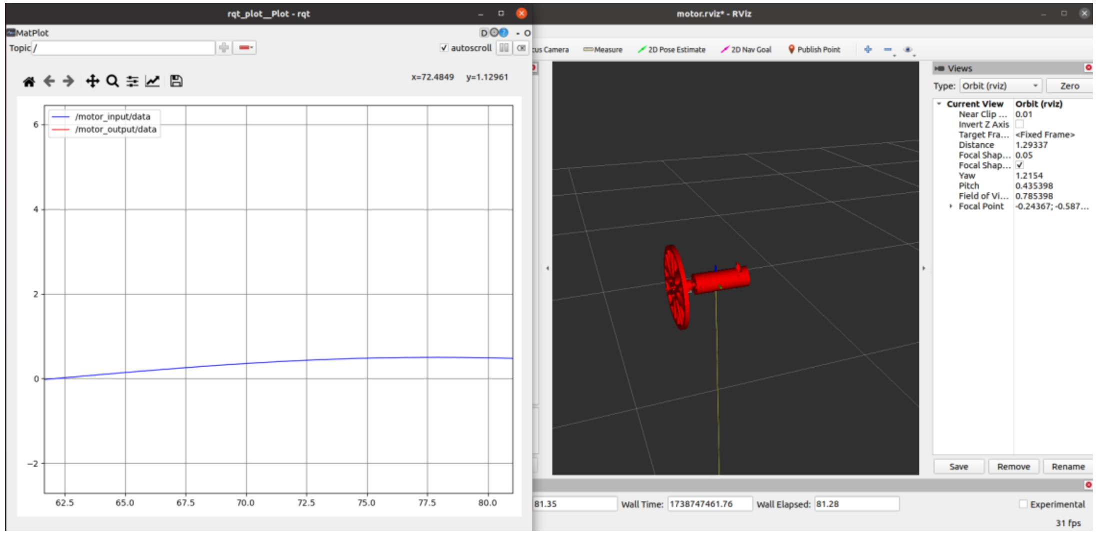
Motor Dynamical System

- Build the program using “catkin_make”

```
$ catkin_make
$ source devel/setup.bash
```

- Launch the “motor_sim.launch”

```
$ roslaunch motor_sim motor_sim.launch
```



- The package, generates three nodes.
  - “motor_sim.py” node.
  - “set_point_generator.py” node.
  - “motor_JointPub.py” node.
- For this activity, the student must develop the **“motor_sim”** node.

motor_JointPub

set_point_generator

motor_sim

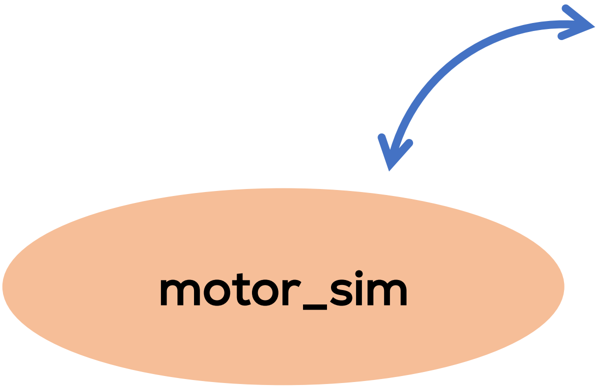
Motor Dynamical System

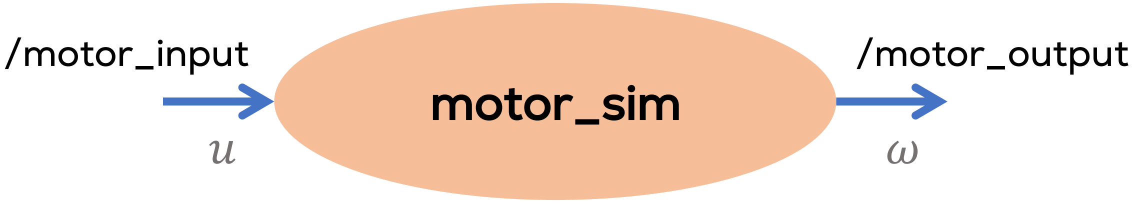
"motor_sim.py" node:

- Node to be completed by the student.
- This node must simulate the dynamical behaviour of a DC motor based on the dynamical model previously shown.

- This node must be subscribed to a topic called "/motor_input" representing the input voltage u of the model and publish a topic "/motor_output" representing the output angular speed of the motor ω.

$$\begin{cases} \dfrac{di}{dt} = -\dfrac{R}{L}i - \dfrac{k_1}{L}\omega + \dfrac{1}{L}u \\ \dfrac{d\omega}{dt} = \dfrac{k_2}{J}i - \dfrac{b}{J}\omega - \dfrac{1}{J}m \end{cases}$$






```
System input message (/motor_input):
std_msgs/Float32
float32 data            #Input to the system

System output message (/motor_output):
std_msgs/Float32
float32 data            #Output of the system
```

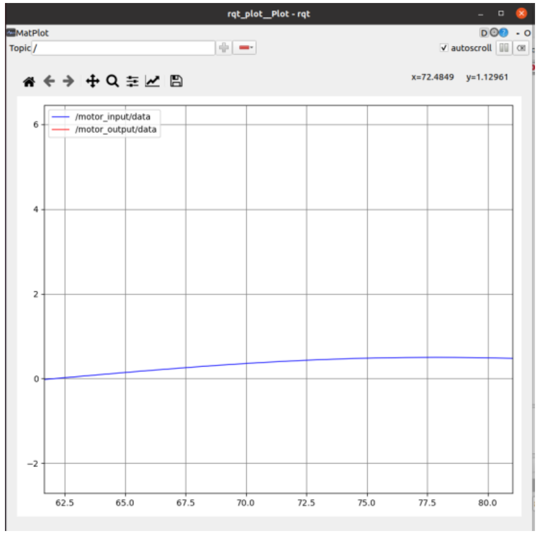
Motor Dynamical System

"set_point_generator.py" node:

- Not to be completed by the student.
- The node generates a signal representing a variable voltage u.
- This voltage can be used as an input signal for the motor simulation.
- The voltage is published on a topic called "/motor_input".

- The parameters of the signal are amplitude and ferequency and can be controlled using the parameters on the launch file.

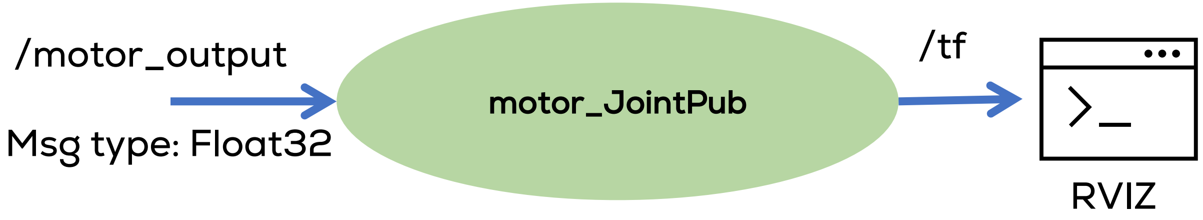
```
<param name = "setpoint_Amplitude" value = "0.5" />
<param name = "setpoint_Freq" value = "0.1" />
```

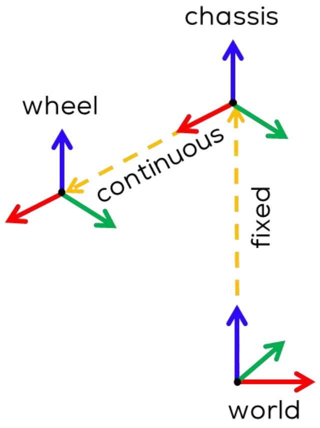


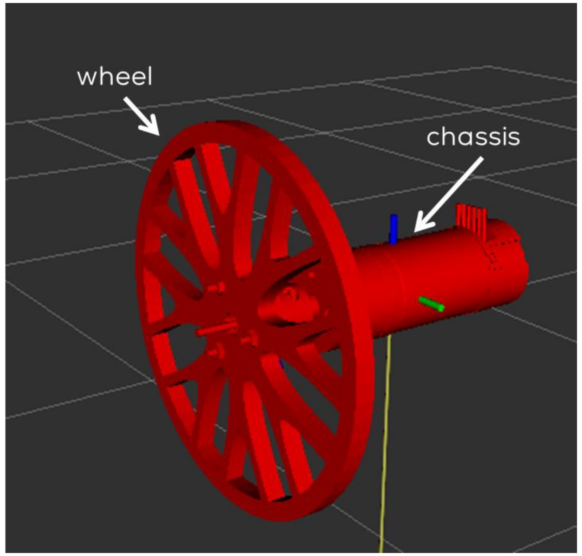
Motor Dynamical System

"motor_JointPub.py" node:

- This node must NOT be modified by the student.
- The node subscribes to a "Float32" message, in a topic "/motor_output".
- The node outputs the transforms and and markers required to visualise the motor in RVIZ.
- The node uses URDF files to generate the transforms required.
- The node rotates the motor joint according to the angular velocity ω published by the motor simulation.







Motor Dynamical System

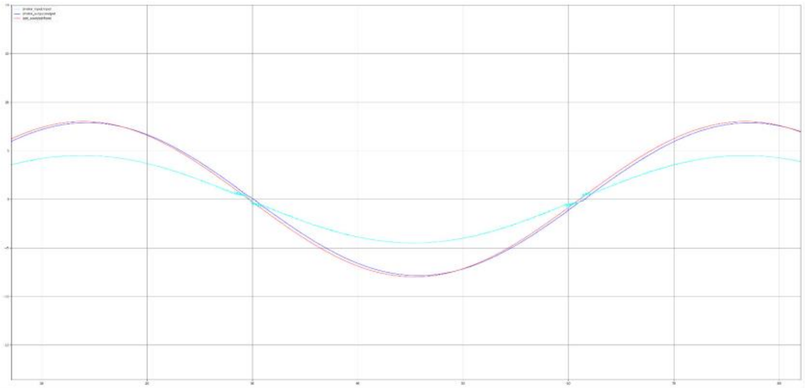
```
import rospy
import numpy as np
from std_msgs.msg import Float32

#Initial conditions
omega = 0.0
current = 0.0

# Setup Variables to be used
first = True
start_time = 0.0
current_time = 0.0
last_time = 0.0

# Declare the input Message
motorInput = Float32()

# Declare the  process output message
motorOutput = Float32()


#Define the callback functions
def input_callback(msg):
    global motorInput
    motorInput = msg
```

```
 #Stop Condition
def stop():
  #Setup the stop message (can be the same as the control
message)
    print("Stopping")
    total_time = rospy.get_time()-start_time
    rospy.loginfo("Total Simulation Time = %f" % total_time)

if __name__=='__main__':
    #Initialise and Setup node
    rospy.init_node("Motor_Sim")


    #Declare Variables/Parameters to be used
    sample_time = rospy.get_param("~motor_sampleTime",0.01)

    #Motor Parameters
    R = rospy.get_param("~motor_R",6.0)
    L = rospy.get_param("~motor_L",0.3)
    k1 = rospy.get_param("~motor_k1",0.04)
    k2 = rospy.get_param("~motor_k2",0.04)
    J = rospy.get_param("~motor_J",0.00008)
    b = rospy.get_param("~motor_b",0.00025)
    m = rospy.get_param("~motor_m",0.00)
```

Motor Dynamical System

```python
    # Configure the Node
    loop_rate = rospy.Rate(rospy.get_param("~motor_simRate",200))
    rospy.on_shutdown(stop)

    # Setup the Subscribers
    rospy.Subscriber("/motor_input",Float32,input_callback)

    #Setup de publishers
    motor_pub = rospy.Publisher("/motor_output", Float32,
queue_size=1)

    print("The Motor is Running")
    try:
        #Run the node
        while not rospy.is_shutdown():
            if first == True:
                start_time = rospy.get_time()
                last_time = rospy.get_time()
                current_time = rospy.get_time()
                first = False
    #System
            else:
            #Define sampling time
                current_time = rospy.get_time()
                dt = current_time - last_time

                #Dynamical System Simulation
                if dt >= sample_time:
                    #Motor governing equations
                    current += (-(R/L)*current-(k1/L)*omega+(1/L)*motorInput.data)*dt
                    omega += ((k2/J)*current-(b/J)*omega-(1/J)*m)*dt

                    #Message to publish
                    motorOutput.data = omega

                    #Publish message
                    motor_pub.publish(motorOutput)

                    #Get the previous time
                    last_time = rospy.get_time()


            #Wait and repeat
            loop_rate.sleep()

    except rospy.ROSInterruptException:
        pass #Initialise and Setup node
```

Motor Dynamical System



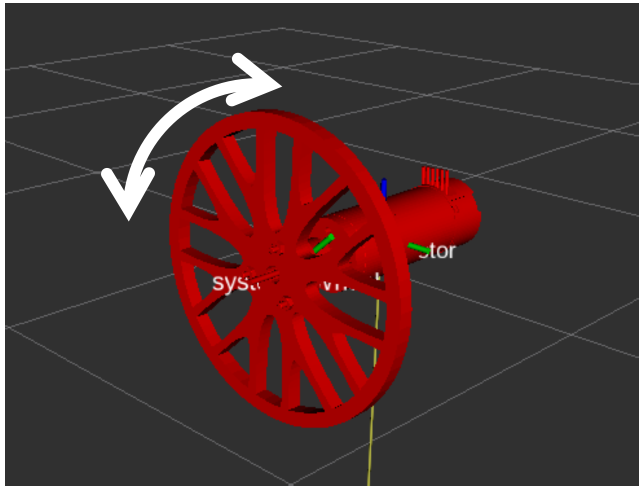
- Save and recompile the project.
- Build the program using “catkin_make”

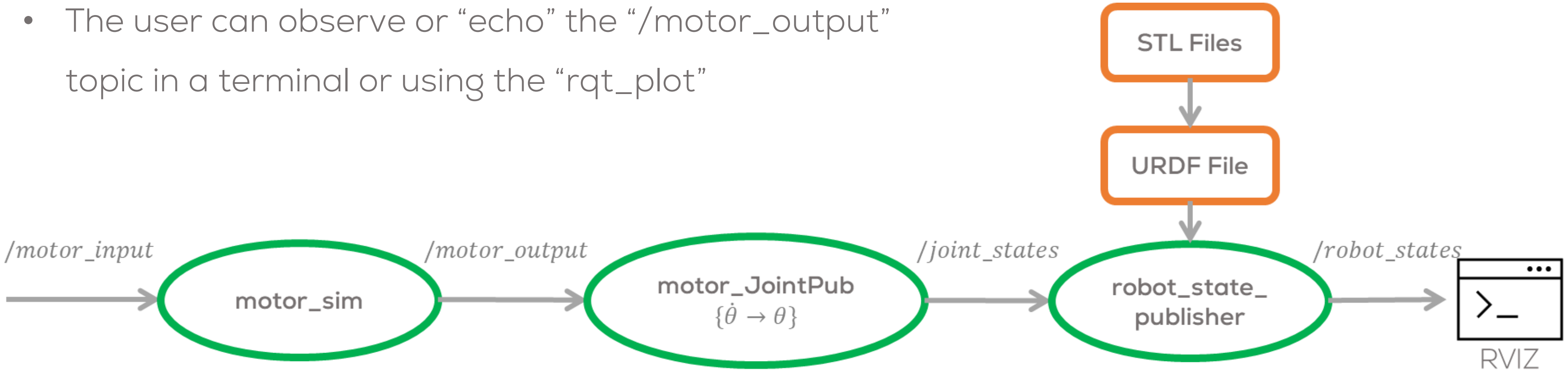
```
$ catkin_make
$ source devel/setup.bash
```

- Launch the “motor_sim.launch”

```
$ roslaunch motor_sim motor_sim.launch
```

- The user can observe or “echo” the “/motor_output” topic in a terminal or using the “rqt_plot”





Expected results

- The motor should spawn in the RVIZ world.
- The wheel must be able to rotate according to the system's dynamics.
- Different inputs to the system must be tested.