{Learn, Create, Innovate};

DC Motor Sim

Mini challenge 1

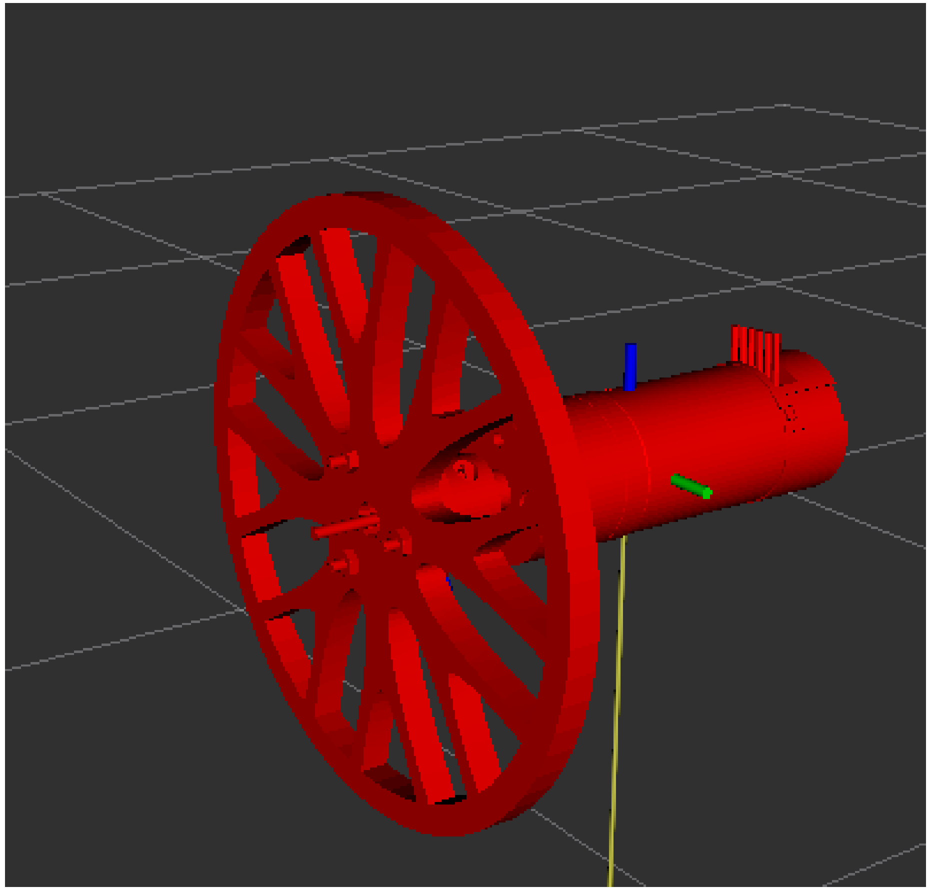
Manchester Robotics

Introduction

Introduction

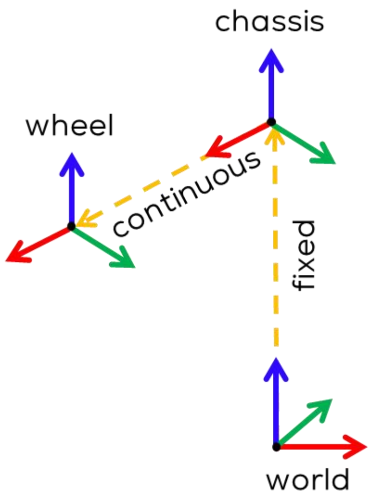
This mini-challenge is intended for the student to review the concepts introduced in the previous sessions.

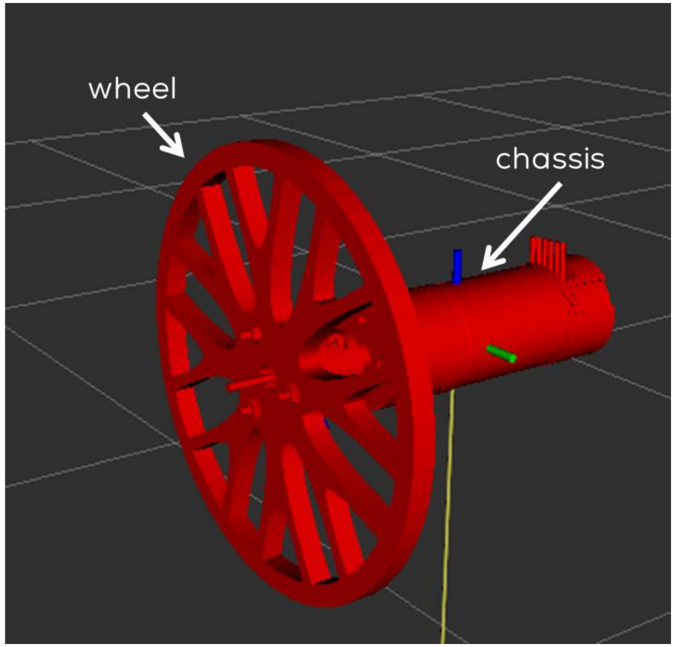
- This activity consists of visualising the dynamical behaviour of a DC Motor using URDF files and joint state publishers in RVIZ.
- This activity employs a simple dynamical system simulation to dictate the motor's state behaviour.



Motor modelling

- The student must model their own DC motor using any CAD package (or use the files provided by MCR2).
  - The model must contain a robot chassis and a wheel to be attached to the motor shaft as shown in the figure.
- The student must use a URDF description file to describe the links and joints of the motor model.
- Three links must be defined: “world”, “chassis”, and ”wheel”.
- The motor “chassis” must be fixed to a “world” frame.
- The “wheel” must be attached to the chassis using a “continuous” joint.





Motor dynamical model

- MCR2 provides the motor dynamical simulation in the package called “motor_sim.py”
- The node can be launched to be tested using the launch file “motor_sim.launch”.

```
roslaunch motor_sim motor_sim2.launch
```

- The user can publish to the “/motor_input” topic form another terminal as follows

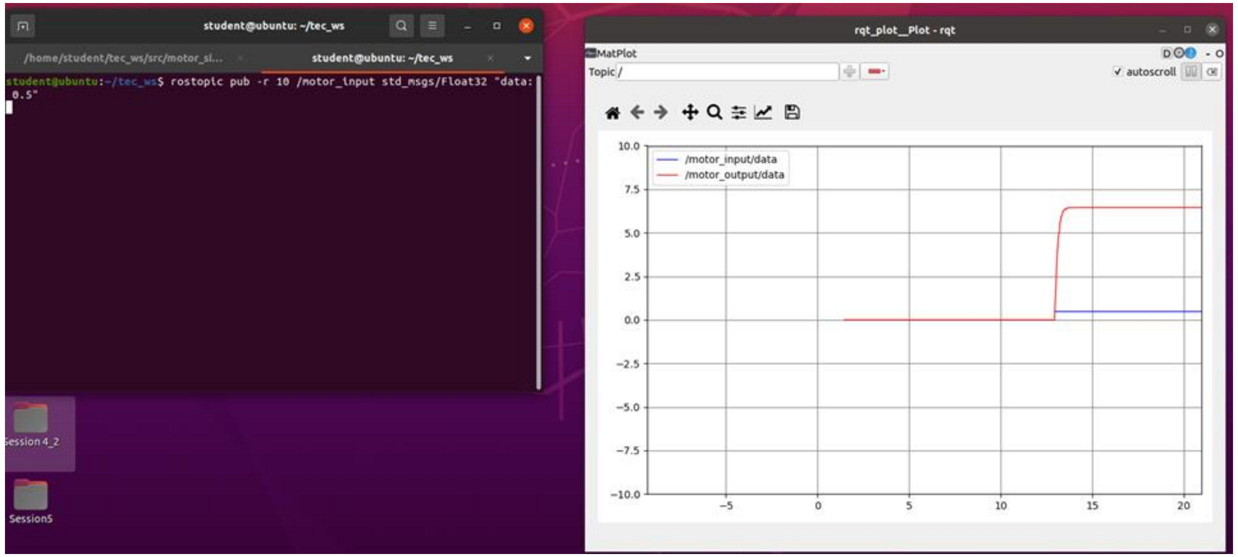
```
rostopic pub -r 10 /motor_input std_msgs/Float32 "data: 0.5"
```

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

- The input and output messages for the “motor_sim” node are “std_msgs/Float32”

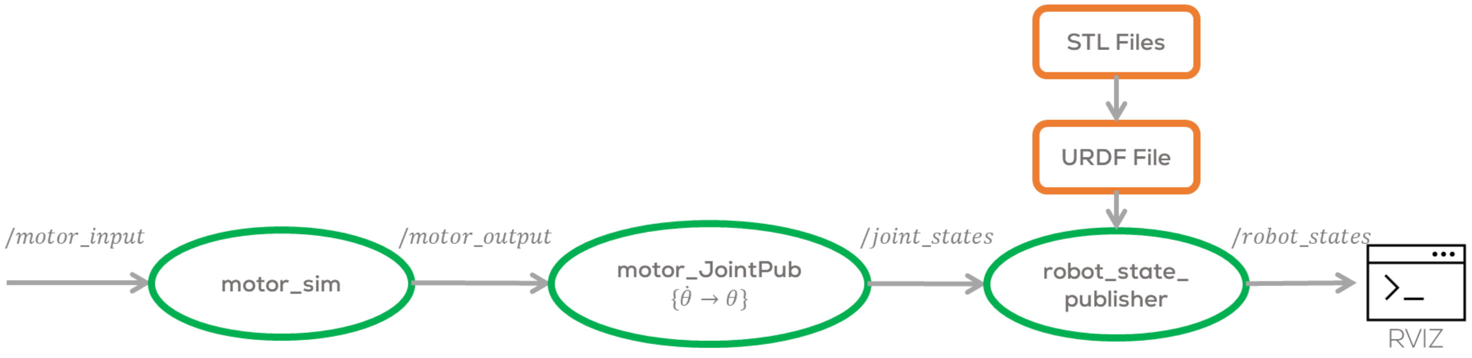
```
System input message (/motor_input):
std_msgs/Float32
float32 data            #Input to the system

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



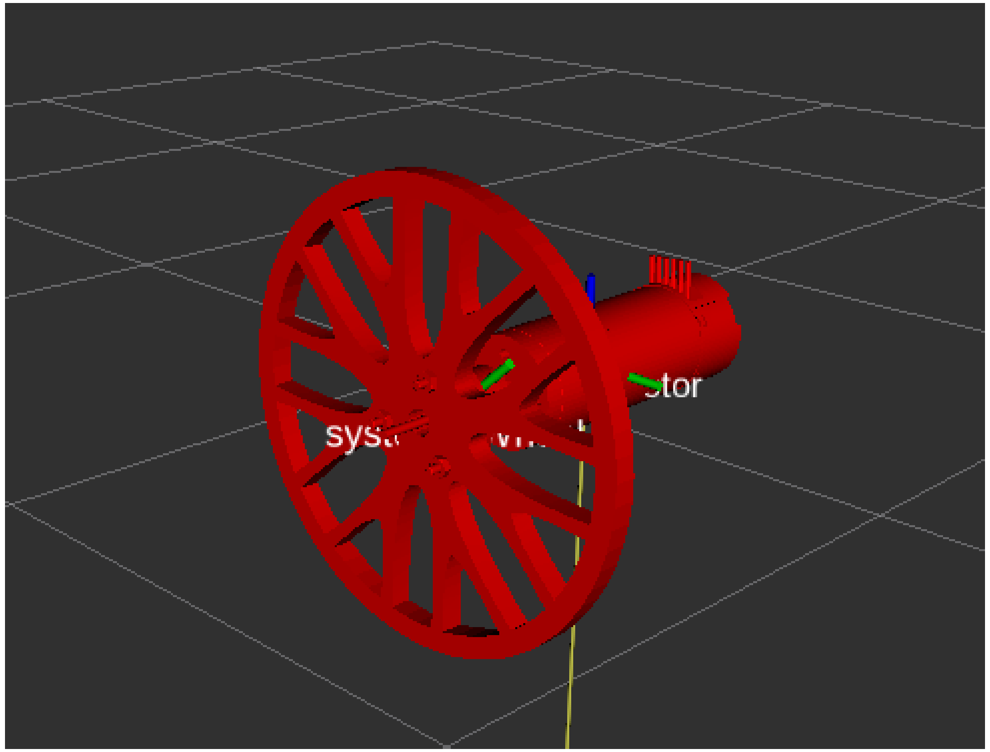
Motor modelling

- The student must develop their joint state publisher for the “continuous” joint.
- The joint state publisher must read the speed of the motor from the topic “/motor_output”.
- The student must transform the motor speed into the angular position of the joint before publishing the information to the joint.
- The user must publish the required information to move the joint (angle in radians).



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.



Mini challenge

Part 2

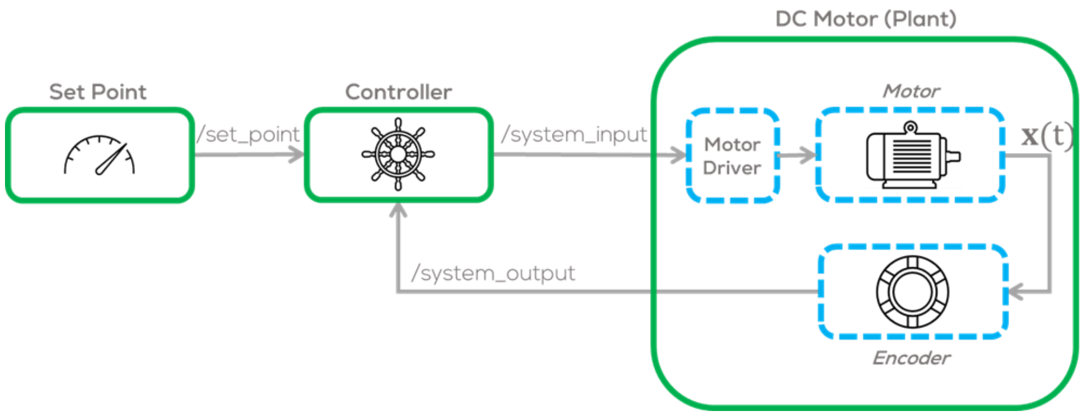
{Learn, Create, Innovate};

Manchester Robotics

Controller

Introduction

- The activity consists of creating a “controller” node and a “setPoint generator” node for the DC motor previously defined.
- The “controller node” must control the speed of the motor.
- The controller can be “P”, “PI” or “PID” controller (other controllers can be accepted upon agreement with the professor.).



Controller node

Controller Node

1. Make a "/controller" node to generate a control input to the "/motor_sim" node.
2. The node must publish in the "/motor_input" topic and subscribe to the "/motor_output" and "/set_point" topics.
3. The output of the controller "/motor_input" must be bounded between in the interval -1 to 1 i.e., $u(k) \in [-1,1]$.
4. The message for the "/set_point" topic must be a "std_msgs/Float32" message.
6. The control node, must use a parameters, for all the required tunning variables.
7. The sampling time and rate can be the same as the "/motor_sim" node 0.01 s for the sampling time and rate of ~200Hz.
8. <u>It is strictly forbidden to use any other python library, other than NumPy. The controller must be made without using any predefined online controllers or libraries.</u>

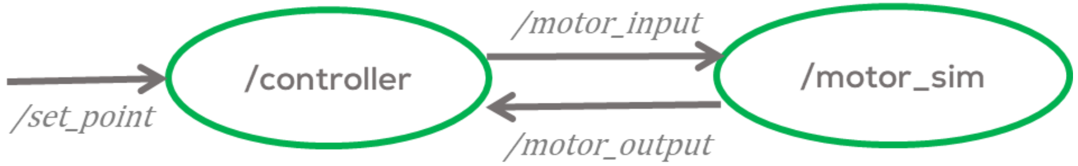
Controller node

Hints

Discrete PID controller:

$$u(k) = K_p e(k) + K_i T_s \sum_{n=0}^{k} e(n) + K_d \frac{e(k) - e(k-1)}{T_s}$$

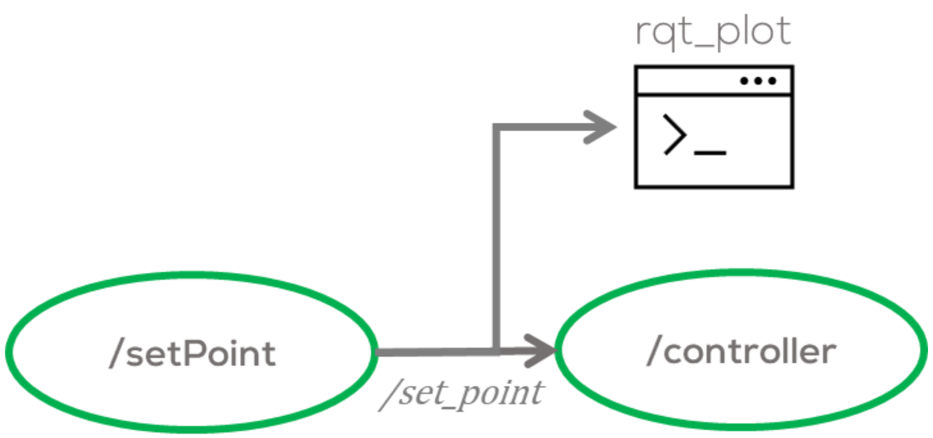
where $u(k), e(k)$ are the controller output and error at time step k, such that time $t = kT_s$ where T_s is the sampling time. K_p, K_i, K_d are the proportional, integral and derivative gains, respectively. More information here.



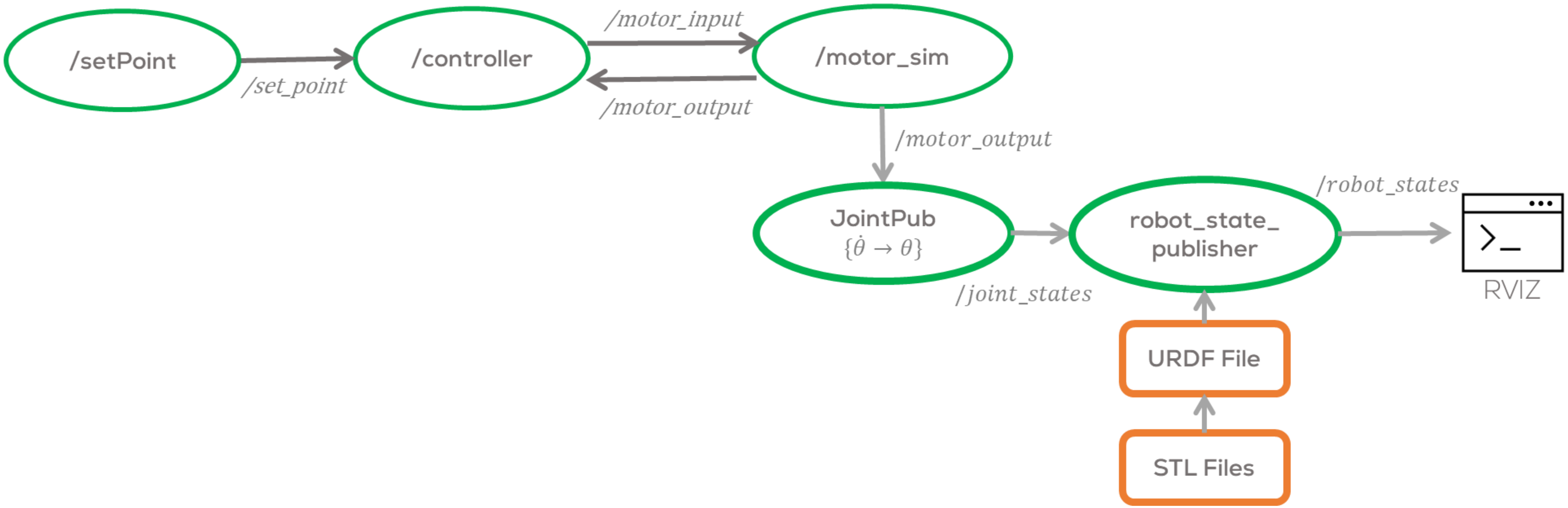
Set Point Generator

Instructions

- Make a new node or use and modify the previously developed "/setPoint" node (Mini challenge 1) to generate a Set Point signal.
  - The output of the renamed "/setPoint" node must publish into the previously defined topic "/set_point" with the appropriate message.
  - The set point generator can be a sinusoidal signal, square signal, etc.
  - As before It is forbidden to use any libraries, except from NumPy for this exercise.
- Make the necessary plots to analyse the system in rqt_plot



Expected Result



Expected Result

Launch File and Plotting

- Use the ROS tool “rqt_plot” to plot the “/motor_input”, “/motor_output”, and “set_point” signals.
- Make a Launch file to execute all the nodes at the same time.

Output (blue) Following the Set Point (red) and control signal (cyan)