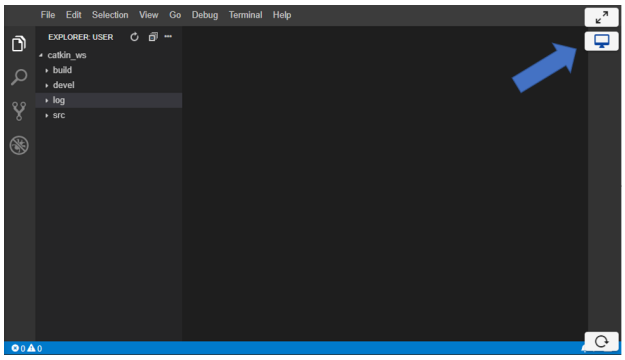
# Lab 1: ROS Basics

ME 597: Autonomous Systems

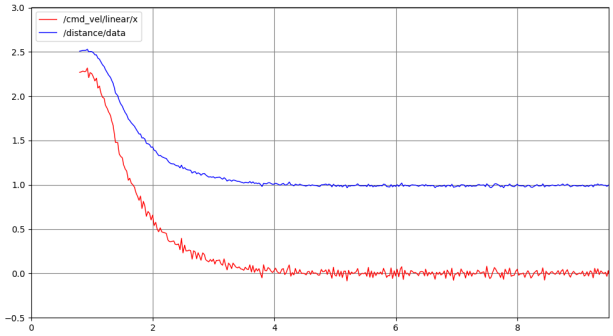
At this point you have setup an account, however, “The Construct” allows you to do more than that. This lab will help you to: 1) explore the components of the platform, and 2) understand the basic functionality of the Robot Operating System (ROS).
ROS is a framework that allows to create robot applications without worrying about underlying hardware, thus users usually do not have to implement from scratch drivers or services. ROS provides an abstraction layer, and a set of tools to visualize and debug robot data. It also provides support for languages such as C++ and Python, however, in this course we will only use Python as main language. The graph based communication architecture of ROS is composed by the following elements:

- Nodes
- Messages
- Subscribers
- Publishers
- Services
- Actions

Throughout the ROS Basics Lab, you will go through the basics of ROS such as the file system. In addition to that you will be able to interact with robot simulations, which provide a very good approximation to the real world. These simulations are mainly built with the open-source 3D Robot Simulator Gazebo.
There are two tools that can help you debug the code and tune the parameters. The first one is `rqt_plot`. It provides a GUI plugin visualizing numeric values. Before you launch the `rqt_plot` node, you need to click on the graphical interface icon. A new window will pop out.




(a) graphical interface icon




(b) rqt_plot example

The second one is `rosbag`. You can use it to record ROS topics and play them back, which enable us to reproduce behavior in the running system. What’s more, we will replay the ROS bag you created to evaluate the performance of your code

## I. Assignment

1) Once you have logged into ”The Construct”, follow the start the “ROS Basics in 5 Days” course. It can be found in the Courses Catalog. Follow the instructions and implement the solutions as required.

2) Stop the robot one meter from the wall

- Go through Unit 1 to 4 of ROS basics in 5 days and finish the exercises.
- Use the environment of Unit 4: Understand ROS Topics: Subscribers and Messages
- Create a ROS package named `assigment_1` with dependencies `rospy` and `std\_msgs` in `~/catkin_ws/src` and compile it.
- Create three directories `/launch`, `/scripts`, and `/bags` in the package.
- Create a script called `controller.py` in `/scripts` and implement the following:
  a) Create a Subscriber that reads from the `/kobuki/laser/scan` topic. This is the topic where the laser publishes its data.
  b) Create two Publishers. One writes into the `/cmd_vel` topic to move the robot. Another writes into the `/distance` topic. This latter has to be a `Float32` topic to be created within the `controller.py` script.
  c) Set `rate = rospy.Rate(30)` and use `rate.sleep()` in a loop to maintain a particular rate for the subscriber callback function. Remember that the `rate` function defines the execution speed of your node. In this particular case, the configured frequency is $30Hz$, which means that the controller node will be scheduled for execution every $33.33ms$. You can tune this parameter, setting it too high will cause an unnecessary load, while setting it to a low value can make your controller to respond very slow.
  d) In the callback function, get the distance to the wall, for simplicity just consider the measurement in the middle of the array ( `msg.ranges[360]` ), calculate the error to the target position, use a PID controller to control the velocity, and then publish the velocity (`/cmd_vel`) and position (`/distance`).
- A launch file called `run_mission.launch` will be provided. Copy it into the `/launch` folder. This file will:
  a) Launch the main script `controller.py`
  b) Run `rqt_plot`. This node should subscribe to `/cmd_vel/linear/x` and `/distance`. We use it to monitor the robot's behavior in a real-time plot.
  c) Create, and save a rosbag to record `/cmd_vel` and `/distance`. The output file will be placed in `assignment_1/bags`.
- Launch `run_mission.launch`. See the response in `rqt_plot`. Try to make the settling time as small as possible by tuning the controller.
- After every attempt **reset** the simulation to move the robot back to its initial position. During the tuning it is possible that you drive the robot out of bounds, which usually results in a robot unable to move. This can be solved by **reloading** the simulation as shown in Fig. 2
- After obtaining the desired result in `rqt_plot` use the recorded bagfile to reproduce the movement of the robot. To do that we have provided a launch file named `run_replay.launch`, this file will run `rqt_plot` and it will replay a bagfile. In order to make this work you need to change the name of the bagfile obtained in previous step to `rosbag_controller.bag` so that the launch file can find it.
- Verify that the plot obtained after running `run_replay.launch` matches the one obtained after running `run_mission.launch`.


Fig. 2: The Construct simulation

3) Husky Robot Challenge
   - Go through Unit 5 to 9 of ROS basics in 5 days to understand ROS services and actions and finish the exercises. Solution to Project - Help TurtleBot Robot Get Out of the Maze will be provided. Use it as a reference to solve the Husky Robot Challenge
   - You should follow the instructions on the Construct and finish this challenge.
   - Note that Husky uses a different Lidar from the Turtlebot, thus you need to subscribe to a different node.
   - Encapsulate your topic subscribers, and publishers inside a python class.

## II. Deliverables

1) Verify you have completed all the graded and non-graded activities of the course.
2) Make sure the folder structure of your project matches the one enforced by the course, e.g. follow the naming proposed by the instructions. This will ease the grading.
3) Download the workspace with the packages you have created throughout the course. This can be done by right-clicking on the `catkin_ws` folder and selecting the download option.

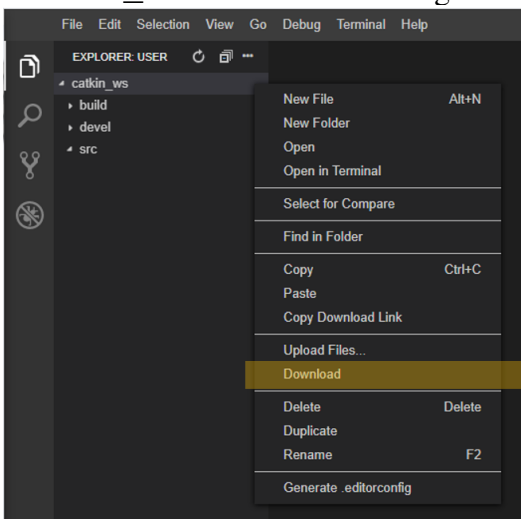



Fig. 3: The Construct IDE screenshot

4) Please add enough comments to your code. Since no other document is required to grade your activity, commenting your code has a lot of impact on your grade.
5) Upload the `catkin_ws` compressed folder into Gradescope.

## III. RUBRIC

- 40 pts - Linear Controller Performance.
  - 15 pts - Overshoot $< 10\%$ and Settling Time for 5% tolerance $< 4s$.
  - 15 pts - PID controller is used correctly.
  - 10 pts - Correct usage of ROS elements.
- 40 pts - EXAM Husky Robot Challenge.
  - 15 pts - The robot gets out of maze successfully.
  - 15 pts - Services and actions are used correctly.
  - 10 pts - Correct usage of python classes.
- 20 pts - Comments and workspace structure.