Lab 1 – Part I supplementary: Differential drive in MATLAB Robotics System Toolbox

1. **Introduction**

The propose of this document is to illustrate the properties of robotics system toolbox and serve as a guide to differential drive system implementation.

As what we have learn in lecture, a differential wheeled robot is a mobile [robot](https://en.wikipedia.org/wiki/Robot) whose movement is based on two separately driven [wheels](https://en.wikipedia.org/wiki/Wheel) placed on either side of the robot body. It can thus change its direction by varying the relative rate of rotation of its wheels and hence does not require an additional steering motion. It is commonly used in robotics industry and the robot, turtlebot, which we will use in our lab is also driven by differential model.

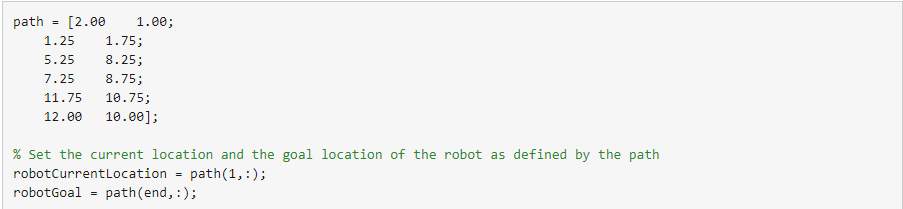
**2. Differential drive model and motion control**

The “Introduction” mentioned the basic concept of the differential drive model. In the following sections, we will demonstrate how to use robotics system toolbox to setup a model and implement the motion control of a robot.

2.1 Robot model used in MATLAB

1)Define Waypoints

Define a set of waypoints for the desired path for the robot



Assume an initial robot orientation (the robot orientation is the angle between the robot heading and the positive X-axis, measured counterclockwise).

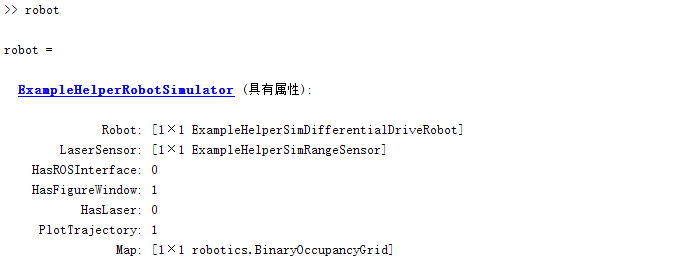


Define the current pose for the robot [x y theta]

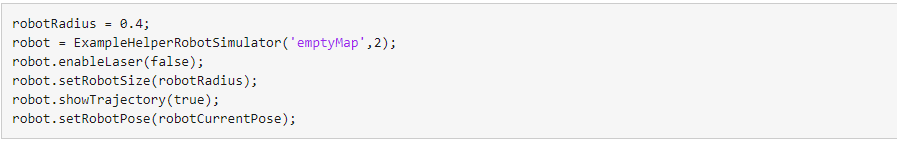


2)Initialize the Robot Simulator

A simple robot simulator is used in this example that updates and returns the pose of the differential drive robot for given control inputs. An external simulator or a physical robot will require a localization mechanism to provide an updated pose of the robot.



Initialize the robot simulator and assign an initial pose. The simulated robot has kinematic equations for the motion of a two-wheeled differential drive robot. The inputs to this simulated robot are linear and angular velocities. It also has plotting capabilities to display the robot's current location and draw the trajectory of the robot.



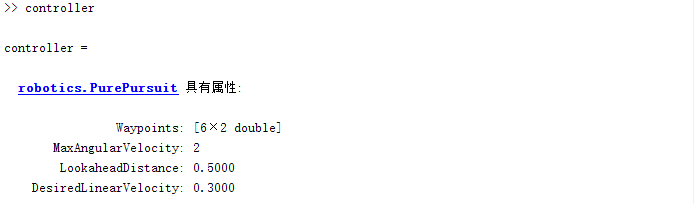
Visualize the desired path



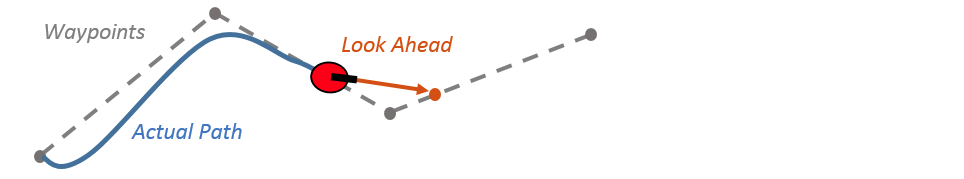
1. Path following controller

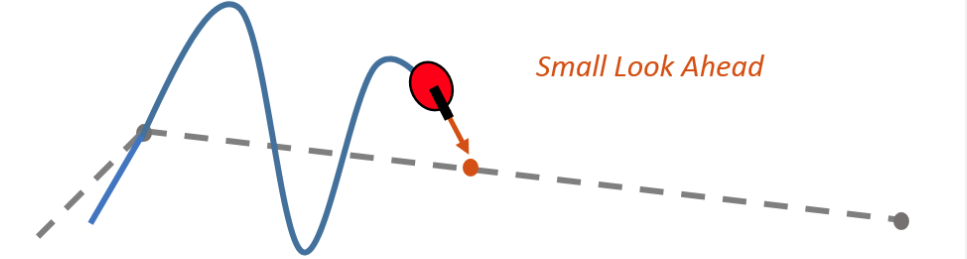
Based on the path defined above and a robot motion model, you need a path following controller to drive the robot along the path. Create the path following controller using the [robotics.PurePursuit](https://ww2.mathworks.cn/help/robotics/ref/robotics.purepursuit-system-object.html) object.

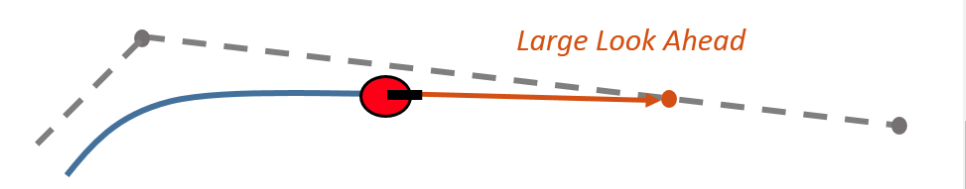




PurePursuit is a path tracking algorithm. It computes the angular velocity command that moves the robot from its current position to reach some look-ahead point in front of the robot. The linear velocity is assumed constant, hence you can change the linear velocity of the robot at any point. The algorithm then moves the look-ahead point on the path based on the current position of the robot until the last point of the path. You can think of this as the robot constantly chasing a point in front of it. The property LookAheadDistance decides how far the look-ahead point is placed.The PurePursuit class ([robotics.PurePursuit](https://ww2.mathworks.cn/help/robotics/ref/robotics.purepursuit-system-object.html)) is not a traditional controller, but acts as a tracking algorithm for path following purposes. The LookAheadDistance property is the main tuning property for the PurePursuit controller. The look ahead distance is how far along the path the robot should look from the current location to compute the angular velocity commands.  The figure below shows the robot and the look-ahead point. As displayed in this image, note that the actual path does not match the direct line between waypoints.







Use the path defined above to set the desired waypoints for the controller



Set the path following controller parameters. The desired linear velocity is set to 0.3 meters/second for this example.



The maximum angular velocity acts as a saturation limit for rotational velocity, which is set at 2 radians/second for this example.



As a general rule, the lookahead distance should be larger than the desired linear velocity for a smooth path. The robot might cut corners when the lookahead distance is large. In contrast, a small lookahead distance can result in an unstable path following behavior. A value of 0.5 m was chosen for this example.

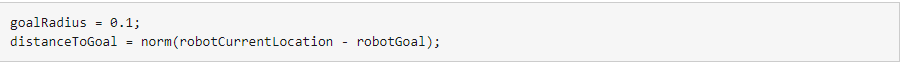


Using the Path Following Controller, Drive the Robot over the Desired Waypoints

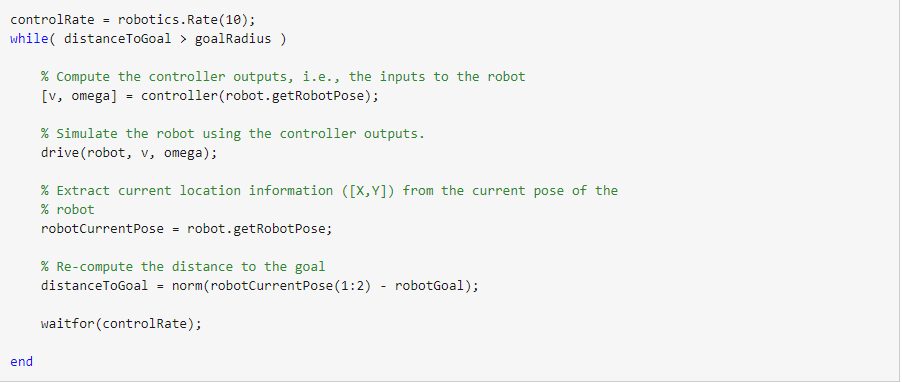
The path following controller provides input control signals for the robot, which the robot uses to drive itself along the desired path.

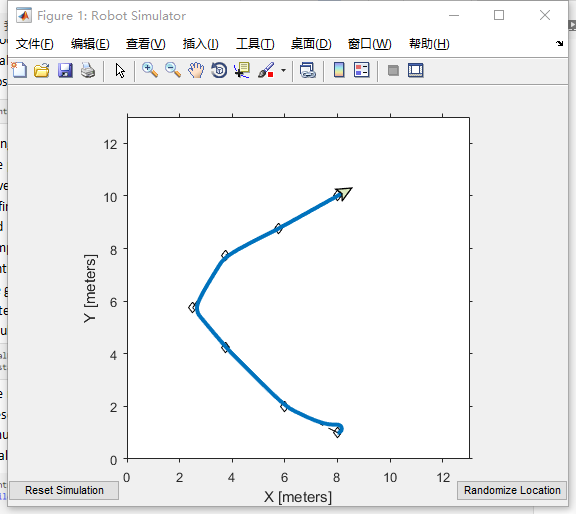
Define a goal radius, which is the desired distance threshold between the robot's final location and the goal location. Once the robot is within this distance from the goal, it will stop. Also, you compute the current distance between the robot location and the goal location. This distance is continuously checked against the goal radius and the robot stops when this distance is less than the goal radius.

Note that too small value of the goal radius may cause the robot to miss the goal, which may result in an unexpected behavior near the goal.



The [robotics.PurePursuit](https://ww2.mathworks.cn/help/robotics/ref/robotics.purepursuit-system-object.html) object computes control commands for the robot. Drive the robot using these control commands until it reaches within the goal radius. If you are using an external simulator or a physical robot, then the controller outputs should be applied to the robot and a localization system may be required to update the pose of the robot. The controller runs at 10 Hz





The simulated robot has reached the goal location using the path following controller along the desired path. Close simulation.

