Waypoint Navigation Control for a Differential Drive Rover

The following is a MATLAB live script that reads a (2D) waypoint matrix from a CSV file and sets up all the required closed loop control algorithms to navigate a **Differential Drive** *Rover* to each waypoint in which they're present in the matrix. We use the **`controllerPurePursuit`** class in MATLAB to generate the immediate trajectory of the rover in real time and the **`differentialDriveKinematics`** class in MATLAB to then convert the trajectory into inputs for our rover. The computed control commands are used to drive the simulated robot along the desired trajectory to navigate the waypoints. This is an isolated live script, not interfaced with any hardware or ROS2.

Initializing the Waypoint Matrix and Beginning Rover Pose

reading waypoint matrix from a CSV file

```
waypoints = readmatrix("waypoints.csv")
```

initialising starting pose of rover

```
roverInitialLocation = [0.00 0.00];
roverInitialOrientation = 0;
roverCurrentPose = [roverInitialLocation roverInitialOrientation]';
```

final destination of the rover (last waypoint)

```
roverDestination = waypoints(end,:);
```

Creating the Kinematic Robot Model

initialising rover object and assigning appropriate physical parameters, also setting control inputs (for the rover) as the linear and angular velocities (for sake of plotted simulation); refer `differentialDriveKinematics` documentation .

```
rover = differentialDriveKinematics;
rover.WheelRadius = 0.05;
rover.TrackWidth = 0.20;
rover.VehicleInputs = "VehicleSpeedHeadingRate";
```

Creating the Path Following Controller

initializing the pure pursuit controller and initialising its parameters (refer `controllerPurePursuit` documentation)

```
controller = controllerPurePursuit;
controller.DesiredLinearVelocity = 0.3;
controller.MaxAngularVelocity = 2.0;
controller.LookaheadDistance = 0.1;
controller.Waypoints = waypoints;
```

tying it all up together

defining a goal radius, i.e., the desired distance threshold between the robot's location and the waypoint location, within which it considers a waypoint to be reached

```
goalRadius = 0.033;
distanceToDestination = norm(roverInitialLocation - roverDestination);
```

initializing the simulation loop to run at a fixed frequency

```
sampleTime = 0.1;
vizRate = rateControl(1/sampleTime);
```

initializing vehicle frame size as it will appear in the simulation plot

```
frameSize = rover.TrackWidth/0.8;
```

the closed loop controller

```
figure % initializing figure

while(distanceToDestination > goalRadius)

    % compute the controller inputs to the rover
    [v, omega] = controller(roverCurrentPose);

    % compute the speed of each of the pair of wheels in RPM
    omega_L = (v - rover.TrackWidth*omega)/rover.WheelRadius * 60/(2*pi);
    omega_R = (v + rover.TrackWidth*omega)/rover.WheelRadius * 60/(2*pi);

    % get the rover's cartesian linear and angular velocities using
controller inputs
    vel = derivative(rover, roverCurrentPose, [v omega]);

    % update rover's pose
    roverCurrentPose = roverCurrentPose + vel*sampleTime;

    % re-compute the distance to the goal
    distanceToDestination = norm(roverCurrentPose(1:2) -
roverDestination(:));

    % update plot
    hold off

    % plot path segments each instance so that it persists while rover mesh
moves
    plot(waypoints(:,1), waypoints(:,2),"k--d")
    hold all

    % plot the path of the rover as a set of transforms
    plotTrVec = [roverCurrentPose(1:2); 0];
    plotRot = axang2quat([0 0 1 roverCurrentPose(3)]);
    plotTransforms(plotTrVec', plotRot, "MeshFilePath", "groundvehicle.stl",
"Parent", gca, "View","2D", "FrameSize", frameSize);
    light;
    xlim([0 5])
    ylim([0 5])

    waitfor(vizRate);
end
```