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 generate control inputs to navigate a **Differential Drive Rover** to each waypoint in which they're present in the matrix, and then publish them to **ROS2** as a publisher to a topic. A ROS2 node running on the ground station publishes the live detected position of the rover (by running ArUco detection on a live stream from a camera facing downwards on the field). This live position of the rover is then fed to a **`controllerPurePursuit`** object 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, as well as published to a topic on a local ROS2 network to which a RPi3B+ onboard the rover is connected and subscribed. It is then interfaced with an Arduino Uno Rev3 through a wired serial USB connection to accomplish accurate control on the RPM of the encoder DC motors.

clearing up workspace

```
clear, clc
```

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 = 0.350;
rover.TrackWidth = 0.225;
rover.VehicleInputs = "VehicleSpeedHeadingRate";
```

Creating the Path Following Controller

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

```
controller = controllerPurePursuit;
controller.DesiredLinearVelocity = 2.00;
controller.MaxAngularVelocity = 10.0;
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 = 0.005;
vizRate = rateControl(1/sampleTime);
```

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

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

initializing a ROS2 node to fetch rover's live position and publish wheel RPMs

```
matNode = ros2node("/mat_node");
```

initializing a ROS2 publisher and the Twist message object

```
groundpub = ros2publisher(matNode,'/motor_command','serial_motor_demo_msgs/
MotorCommand');
twist = ros2message(groundpub);
```

initializing a ROS2 subscriber and rover_pose message object

```
groundsub = ros2subscriber(matNode, '/rover_pose', 'serial_motor_demo_msgs/
RoverPose');
```

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;
    omega_R = (v + rover.TrackWidth*omega)/rover.WheelRadius;

    % transmit the controller inputs to the rover
    twist.is_pwm = false; twist.mot_1_req_rad_sec = single(omega_L);
twist.mot_2_req_rad_sec = single(omega_R);
    send(groundpub, twist);

    % 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;
    [roverCamPose,status,statustext] = receive(groundsub);
    roverCurrentPose(1) = roverCamPose.rover_x;
    roverCurrentPose(2) = roverCamPose.rover_y;
    % roverCurrentPose(3) = roverCurrentPose(3)+vel(3)*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 1])
    ylim([0 1])

    waitfor(vizRate);
end
```