Milestone 1: Vehicle model identification

1.1: Identify model based on simulated response to step input

In first part of the assignment, the goal is to simulate the robot behavior. That will permit us to create data, while controlling a few parameters. Simulation and collection of the data happens in a few steps, as follows.

1.1.1: Select the parameters for the step response and the vehicle

RVC robot (robotics vision control) needs first to be initialized, some starting parameters that would define the desired circumstances are set according to desires of the operator. Those values can later be adapted upon requirements. In this case, the linear and angular velocities are the controlling parameters which we define as an input below:

```
%% 1.1 Select the parameters for the step response and the vehicle
% Initialization
startup_rvc

% Selected rvc parameters
v = 1.0;                        % Forward velocity [m/s] - used as an input step
gamma = 0.0;                    % Steer angle [rad] - the car moves in a straight line when steer angle is set to zero
n = 100;                        % Number of observations []
dt = 0.1;                       % Time interval [s] - defines how fast the values are updated
```

Figure 1. Selection of the robot parameters

1.1.2: Creating a robot instance - Vehicle

After defining the desired control pattern, the vehicle can be created. Some starting parameters and limitations are implemented in the code to define the object. The values of noise errors that were created on the odometry and gyroscope sensors were provided by the mentor. These parameters and limitations are given as follows:

```
%% 1.2 Creating a robot instance - Vehicle
% Vehicle object
X = [0; 0; 0];                  % Initial position for the start(x,y,theta) [m,m,rad]
Q = diag([0.005,0.005].^2);     % Odemetry noise(distance,angle) [m,rad] - values given by mentor
vehicle = Bicycle("covar",Q,'x0',X, 'accelmax',1, 'speedmax',4, 'steermax',1, 'dt',dt); % Creating the object, setting the maximum values
vehicle.V_IMU = 0.01;           % Create noise on gyroscope measurements [rad]
```

Figure 2. Creation of the object vehicle

Next step is generating a space (set of arrays) where we can save the results of the movement of the vehicle over every timestep. Also, the range inside which the vehicle will move, needs to be created:

```
% Initialize collections - creating arrays for the gyroscope, robot and
% time for recording the data in the for-loop
time = zeros(1,n);                          % Timestamps [s]
theta_gyro = zeros(1,n);                    % Measurements returned by gyroscope [rad]
theta_robot = zeros(1,n);                   % Actual angle of robot [m]

% Define the range where the vehicle will moves (x,y)
figure('Name','The vehicle movement')
xlim([-2,10])                               % X limitations [m]
ylim([-2,5])                                % Y limitations [m]
```

Figure 3. Creation of the space for the results and the range where the vehicle will move

1.1.3: Simulate the vehicle movement and record the sensors measurements

The vehicle movement receives a step input with the help of the *vehicle.step()* command. Where the constant linear and angular velocities are given and defined in the beginning. The result of each step is saved, to be plotted hereafter. The velocities are calculated in relation to time between the current velocity and the previous velocity by utilizing the for-loop.

In the for loop the index starts from 2. The time is computed from time(i-1) and to have the first measurement point index should be equal to two. The first measurement point in array corresponds to one, as Matlab array starts at one.

```
%% 1.3 Simulate the vehicle movement and record the sensors mesurements
valueOfD = zeros(1,n);                      % Displacement array [m] - to record linear displacement between two time points
valueOfO = zeros(1,n);                      % Distortion array [rad] - to record distortion between two time points
for i=2:n
    odom = vehicle.step(v, gamma);          % Simulate a step for the vehicle(speed,steer angle) [m/s ,rad]

    theta_gyro(i) = vehicle.get_IMU();      % Get measured robot orientation [rad]
    time(i) = time(i-1) + dt;               % The current time [s] - in relation to the previous time

    % Plot results and prepare for next loop
    vehicle.plot();

    % Inserting the data into the arrays
    ds = odom(1);                           % Driven distance [m]
    dth = odom(2);                          % Driven distortion [rad]
    valueOfD(i) = ds;                       % Displacement [m]
    valueOfO(i) = dth;                      % Distortion [rad]
end
```

Figure 4. The movement of the vehicle

1.1.4: Results and the conclusion:

After saving all the data, the linear and angular velocities can be calculated. The result is later plotted on the same figure.

```
%% 2. Display the resulted measured step response
%% 2.1 Calculate the velocity from the displacement
v_final = valueOfD/dt;                      % Linear velocity [m/s]
w_final = valueOfO/dt;                      % Angular velocity [rad/s]

%% 2.2 Plot the result
figure('Name','Velocity')
subplot(2,1,1);
plot(time, v_final)
title('Linear velocity')
xlabel('time [s]')
ylabel ('speed [m/s]')

subplot(2,1,2);
plot(time, w_final)
title('Angular velocity')
xlabel('time [s]')
ylabel ('omega [rad/s]')
```

Figure 5. Calculation and plotting of the velocities

That gives the following results:

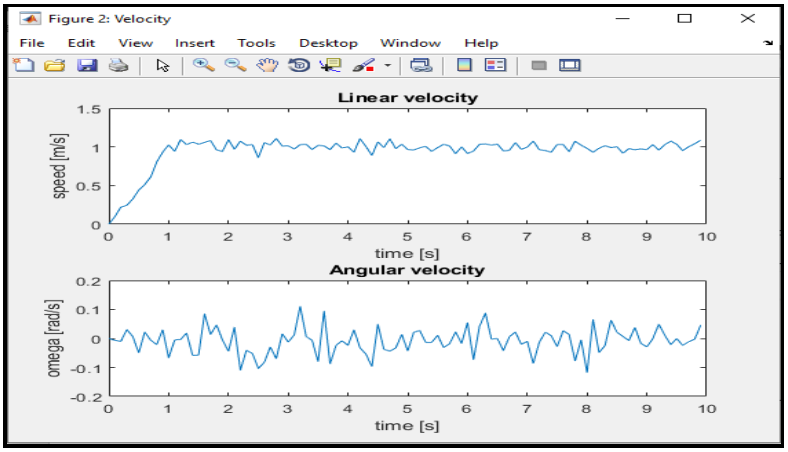

Figure 6. The linear and angular velocity of the vehicle with step input applied to the linear velocity.

There are a few important points to make based on the generated graphs. First, we can notice the presence of the noise on both sensors. For the linear velocity, there is almost a linear increase until the desired value (v = 1m/s) is reached. The desired value is constituted by the applied step of 1.0 m/s to the linear velocity. After the response evens out and stays constant with some fluctuations due to implemented sensor noise.

As no step input is applied to the angular velocity (gamma = 0 rad/s), there will be no change observed in the step response of angular velocity. Hence, the omega stays within the range close to zero. Since there is a certain noise, which is caused by odometry and gyroscope sensors, the values still alternate around zero. The values for noise on the sensors simulate the noises of the sensors in the real time due to the changes in the environment. Later, the noise can be smoothed out by using smoothing or averaging. This principle introduces a low pass filter in the frequency domain.

To see how the angular velocity corresponds to the step input, there should be also an input provided to gamma. If steer angle is zero, then the vehicle moves in the straight line. The change in the angle, will lead to the change in angular velocity. As expected, the step input to the steering angle constitutes to the step response of the angular velocity.

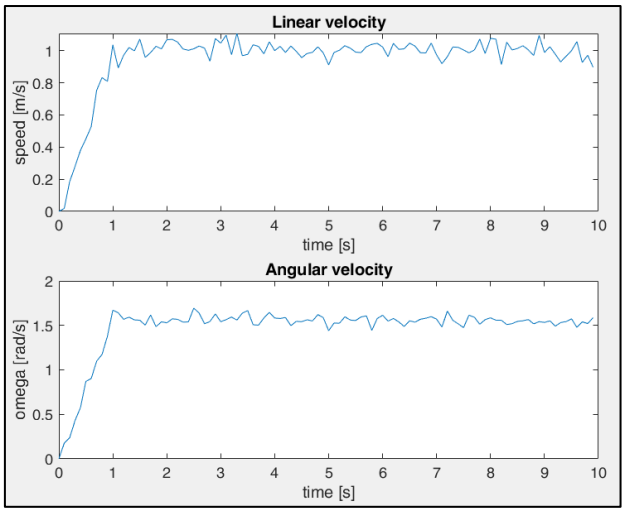

Figure 7. The linear and angular velocity of the vehicle with step input applied to velocity and to gamma.

1.2: Identify model based on simulated response to step input

In this part of the assignment, data of the vehicle has been provided. The goal is to calculate the controlling parameters, namely linear and the angular velocities. There are nine set of data provided, where for two sets of data the linear and angular velocity will be computed.

1.2.1: Data set 1: 0.1 v-step and 0 w-step:

1.2.1.1: Load the data

First, the data needs to be inserted as follows.

```
%% 1 Data input 1: 0.1 v-step and 0 w-step
%% A. Load the response of the first data set (0.1 v-step and 0 w-step)
load('v_step_0.1_w_step_0_enc.mat');
whos -file v_step_0.1_w_step_0_enc.mat
```

Figure 8. Loading the data files.

The provided data set contains all_enc_left/right, all_time,v_step and w_step. With 'whos 'function we can peek the variables of the loaded file.

1.2.1.2: Define the velocities

First step is to define the angular velocity (in m/s) of each wheel at every timestep. Therefore, we need to convert every tick per second to radians for each timestep. Next, we can calculate the linear velocity of each wheel by multiplying the angular velocity by radius of the wheel.

Now that the velocities of the wheels are known, calculating the forward velocity and angular velocity can happen as follow. The assumption made here is that the encoder calculates ticks per rad rather than tick per degree. Otherwise the conversion factor (pi/180) should be applied.

```
%% B. Define the velocities
%% B.1 Define the parameters of each wheel
% Global constants
ticks_to_rad = 0.001533981;                          % A constant from tick to rad [rad/ticks]
B = 0.16;                                            % Distance between wheels [m]
R = 0.033;                                           % Wheel radius [m]

%% B2. Calculation of the velocities
%% B2.1 Find the ticks of each wheel per time interval
for i=1:length(all_enc_left)-1
    time(i) = all_time(i+1)-all_time(i);             % Time interval [s] - time between to consequentive time points
    vel_left(i)=((all_enc_left(i+1)-all_enc_left(i))*ticks_to_rad)/time(i);    % Angular velocity of the left wheel [m/s]
    vel_right(i)=((all_enc_right(i+1)-all_enc_right(i))*ticks_to_rad)/time(i); % Angular velocity of the right wheel [m/s]
end

%% B2.2 Calculate the linear and angular velocities
%The rotational velocity of the robot is (w1-w2)/B and the forward velocity is the average between the two linear velocities
v_calc_left = vel_left*R;                            % Linear velocity of left wheel [rad*m/s]
v_calc_right = vel_right*R;                          % Linear velocity of right wheel [rad*m/s]
v_result = (v_calc_right + v_calc_left)/2;           % Linear velocity of the robot [m/s]
w_result = (v_calc_right - v_calc_left)/B;           % Angular velocity of the robot [rad/s]
```

Figure 9. Calculation of the velocities from the encoder values.

1.2.1.3: Plotting the results

Finally, the results can be plotted.

```
%% C. Plot the response to 0.1 v-step and 0 w-step
figure('Name','The velocitys of the vehicle')
subplot(2,1,1)
plot(all_time(1:length(all_time)-1),w_result)
title('Angular Velocity')
xlabel('time [s]')
ylabel ('omega [rad/s]')
grid on
grid minor

subplot(2,1,2)
plot(all_time(1:length(all_time)-1),v_result)
title('Forward Velocity')
xlabel('time [s]')
ylabel ('speed [m/s]')
grid on
grid minor
```

Figure 10. Plotting the computed velocities.

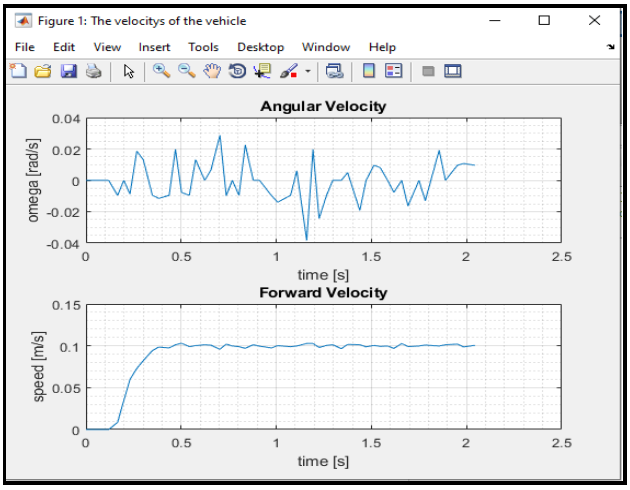

Figure 11. Plots of the angular velocity and the forward velocity.

The results let us see, that the forward velocity approaches the 0.1 m/s and the angular velocity is also approaching 0 rad/s. This is explained by the fact that the loaded file implements step on the forward velocity and no step on the angular velocity. There is a certain noise present on both measurements, that are caused by the noise of the odometry and gyroscope sensors.

1.2.1.3: Determine the system from the unit step response

As the input in this data set is applies to linear velocity, the graph of the forward kinematic will provide us with the information needed to find the transfer function. Looking at the step response, we choose to approach this system by the first order. Now looking at the plot, certain parameters can be concluded such as time delay. The steady gain and the 63.2% of the input can be calculated. Then we calculate the time constant. When these parameters are known, we can define the system which is the product of the transfer function and the input.

```
%% D. Determine the System Parameters from the step response and plot the resaults
%% D.1 Parameters (Concluded from the plot)
delay = 0.119;                                  % Time delay [s]
K = 0.1005/v_step;                              % Steady state gain [] - change in output/ change in input
el_output = (63.2/100)*v_step;                  % Output of one elapsed time constant [m/s] (= 0.0632 m/s)
t = 0.24;                                       % Time [s] - at which 63.2% of output is reached
tau = t - delay;                                % Time constant [s]

%% D.2 Define the transfer function
s = tf('s');                                    % Enabling 's'
G = K*exp(-delay*s)/(tau*s+1);                  % Transfer function of first order - output/ input
sys = v_step*G;                                 % Output of system [m/s]
```

Figure 12. Determining the transfer function.

Then the results are plotted:

```
%% D.3 Plotting the resaults
figure('Name','Step response')
step(sys)
title('Approximated Transfer Function for Velocity')
grid on
grid minor
xlabel('time [s]')
ylabel ('speed [m/s]')
```

Figure 13. Plotting results.

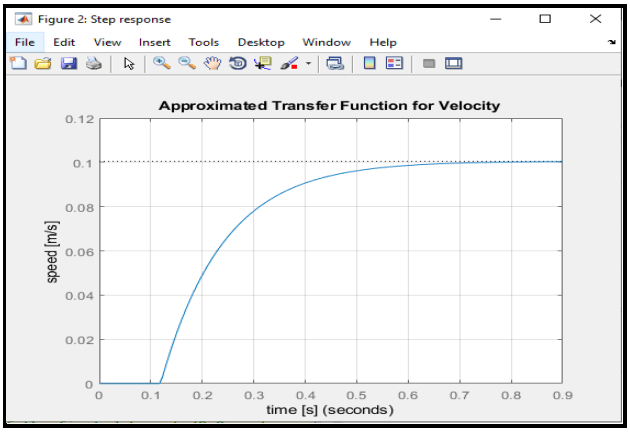

Figure 14. Calculated transfer function.

As seen in the results, the step response does approach the 0.1 m/s velocity.

1.2.1: Data set 2: 0 v-step and 2.0 w-step:

The process of the data for this data set is the same as explained here above. Only in this data set the step input is an angular velocity (see MATLAB code). To avoid the repetition, only the results of the calculation of this data set will be given in figures here under.

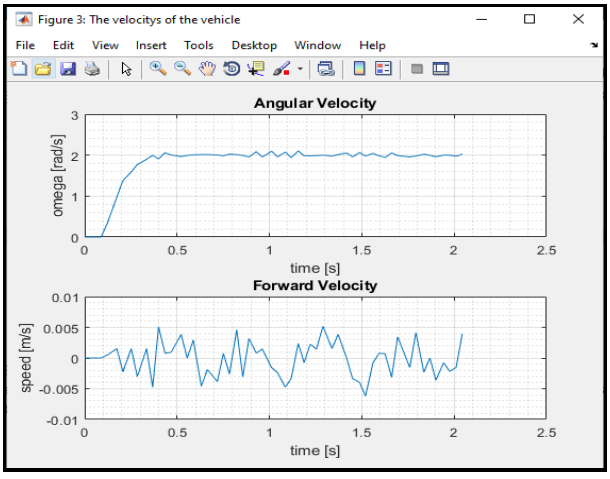

Figure 15. Step applied to the angular velocity.

As seen in the plot, the forward velocity is fluctuating around zero and the angular velocity does settle down around the 2.0 rad/s. After some calculations, the step response of the system can be given as follows:

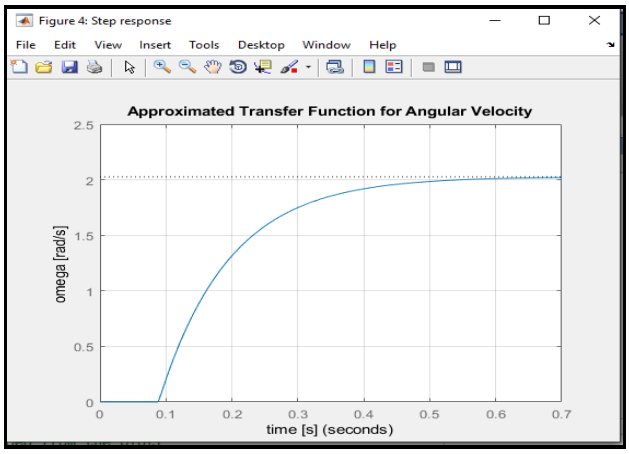

Figure 16. Step response with the computed transfer function.

The results show us that the angular velocity does approach 2.0 rad/s.