In week 4 lab we will implement belief propagation to track robot's location as it executes certain motion.

Unicycle robot kinematics are described as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \qquad --- \text{ Eq 1}$$

Following snippet shows how we issue (v,w) inputs to our robot.

```
velocity_msg = Twist()
velocity_msg.linear.x = 0.2
velocity_msg.angular.z = 0.2
```

Whenever we send out a Twist() message, it only acts for a certain duration (VELOCITY_TIMEOUT). The following steps illustrate this

At t=0 assume robot is at (x,y)

- we issue a Twist message with v=1, w=0

At t=0.6

- the robot is now at (x + **dx**, y+**dy**). **dx**, **dy** = computed using **dead reckoning**
  - Euler integration (simple linear interpolation)

$$\begin{matrix} x^+ \\ y^+ \\ \theta^+ \end{matrix} = \begin{matrix} x \\ y \\ \theta \end{matrix} + \begin{bmatrix} v\cos\theta\,\delta t \\ v\sin\theta\,\delta t \\ \omega\,\delta t \end{bmatrix}$$

δt = 0.6 in this case

  - Exact integration with an ODE solver

After t=0.6

- robot will stop

For the purpose of the present exercise we will consider Euler integration. With this method we can compute the odometry of the robot. Let us now perform belief propagation

Part A)

A sample project week5 is provided to you. Run the launch file '*motion_model_exercise.launch*'. Observe the following topics:

- '/odometry_tracker' : accepts **Twist**() message, relays it to robot and computes robot's Odometry by Euler integration
- '/manual_odom' : provided **Odometry data** (x,y, theta) calculated via dead reckoning

Implement a ROS node that publishes Twist() message with (v=0.11, w=0.8) to '/odometry_tracker' topic.
Plot the rosgraph and save it.

Part B)

Robot starts at (0,0,0). Apply (v,w) such that the robot reaches the point (0.5, 0). However, for this task both '/odom' and '/manual_odom' are not to be used. (HINT : Velocity timeout)
This task is akin to open loop control.

Plot the trajectory of the robot reported by '/odom'. Mention the difference between final position reported by '/odom' and point (1,0)

Note : v and w magnitude must remain within 0.12

Part C)

We will now implement belief propagation. For this we will consider the world to be a 2m x 2m arena. A 200x200 cell grid will represent the area [0,2] x [-1,1]. Thus each cell will correspond to a square of 1cm. For each grid cell we will consider 1 degree discretization of theta and we will sum all 360 values corresponding to a (x,y) cell.

- The robot starts at (0,0,0). Thus the initial belief is a 1 at cell corresponding to (0,0).
- Apply (v1,w1=0) to make the robot move in x-direction for time (t1). Read robots current pose from '/manual_odom'. Perform belief propagation, use algorithm **motion_model_velocity** from Probabilistic robotics book Ch5 (reproduced in next page).
- Apply (v2,w2) to make the robot move along an arc for time (t2). Read robots current pose from '/manual_odom'. Perform belief propagation like before.

Plot the belief space (200x200 grid) after each step above. (4 snapshots).

1: **Algorithm motion_model_velocity**(x_t, u_t, x_{t-1})**:**

2: $\mu = \frac{1}{2} \frac{(x - x')\cos\theta + (y - y')\sin\theta}{(y - y')\cos\theta - (x - x')\sin\theta}$

3: $x^* = \frac{x + x'}{2} + \mu(y - y')$

4: $y^* = \frac{y + y'}{2} + \mu(x' - x)$

5: $r^* = \sqrt{(x - x^*)^2 + (y - y^*)^2}$

6: $\Delta\theta = \text{atan2}(y' - y^*, x' - x^*) - \text{atan2}(y - y^*, x - x^*)$

7: $\hat{v} = \frac{\Delta\theta}{\Delta t} r^*$

8: $\hat{\omega} = \frac{\Delta\theta}{\Delta t}$

9: $\hat{\gamma} = \frac{\theta' - \theta}{\Delta t} - \hat{\omega}$

10: *return* $\textbf{prob}(v - \hat{v}, \alpha_1|v| + \alpha_2|\omega|) \cdot \textbf{prob}(\omega - \hat{\omega}, \alpha_3|v| + \alpha_4|\omega|)$
$\cdot \textbf{prob}(\hat{\gamma}, \alpha_5|v| + \alpha_6|\omega|)$

Week5 Lab Assignment: The objective is similar to that of Week3 Lab i.e. tracking a circular trajectory using noisy trilateration data. This time however we will have additional information in the form of noisy odometry. To combine these two odometry information we will use the EKF framework.

A sample project week5 is provided to you. Run the launch file '*filtering_exercise.launch*'. Observe the following topics are available to you:

- '/odometry_tracker' : accepts **Twist**() message, relays it to robot and computes robot's Odometry by Euler integration
- '/manual_odom' : provided **Odometry data** (x,y, theta) calculated via dead reckoning
- '/trilateration_data' : Provides distance from 3 known landmarks along with Variance information

Part A)

Obtain the matrices A,B and H for the unicycle robot model and trilateration measurement model.

Part B)

Obtain discrete versions of the matrices A and B (as D and G). Comment on your observations.

Part C)

- Consider the robot starts at (0,0,0). This is your initial state vector x.
- Consider the state noise to be Gaussian with parameters **0** and **R**
  R = diag(0.01, 0.05, 0.05)
- Consider the measurement noise to be Gaussian with parameters **0** and **Q**
  Q = diag(landmarkA variance, landmarkB variance, landmarkC variance)
- Consider the initial State covariance to be **E** = diag(0.5, 0.5, 0.5)

Apply EKF algorithm using the algorithm in Table 3.1 as reference.
For simplification, we have provided you with template code controller.py populated with helpful comments to aid you in writing the code Or you can choose to write it yourself.

- Plot the trajectory of the robot and compute the RMS of radial distance error as was done in Week3 lab.
- Plot the diagonal elements of **E** as the robot moves. You can overlay three diagonal elements in a single plot.