1.

Given the following:

$$q = \begin{bmatrix} x_r \\ y_r \\ \theta \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \qquad u = \begin{bmatrix} v_f \\ \omega \end{bmatrix}$$

$$q[k+1] = f(q,u,v) = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} q_1[k] + T(u_1[k] + v_1[k]) cos q_3[k] \\ q_2[k] + T(u_1[k] + v_1[k]) sin q_3[k] \\ q_3[k] + T(u_2[k] + v_2[k]) \end{bmatrix}$$

$$y[k] = h(x[k]) + w[k] = \begin{bmatrix} q_1[k] \\ q_2[k] \end{bmatrix} + w[k]$$

a) The linearized approximation of the system as a function of the current state can be generally written as:

$$q[k+1] \approx F(q[k], u[k])q[k] + G(q[k])u[k] + \Gamma(q[k])v[k]$$

The matrices F, G, and Γ can be found by taking the Jacobian of q[k+1] = f(q, u, v) with respect to q, u, and v, respectively. These derivations are given below:

$$F(q[k], u[k]) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}} = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} & \frac{\partial f_1}{\partial q_3} \\ \frac{\partial f_2}{\partial q_1} & \frac{\partial f_2}{\partial q_2} & \frac{\partial f_2}{\partial q_3} \\ \frac{\partial f_3}{\partial q_1} & \frac{\partial f_3}{\partial q_2} & \frac{\partial f_3}{\partial q_3} \end{bmatrix}_{\substack{q=\hat{q}[k|k] \\ u=u[k] \\ v=0}} = \begin{bmatrix} 1 & 0 & -T(u_1[k]) sin\hat{q}_3[k|k] \\ 0 & 1 & T(u_1[k]) cos\hat{q}_3[k|k] \\ 0 & 0 & 1 \end{bmatrix}$$

$$G(q[k]) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} \\ \frac{\partial f_3}{\partial u_1} & \frac{\partial f_3}{\partial u_2} \end{bmatrix}_{\substack{q=\hat{q}[k|k] \\ u=u[k]}} = \begin{bmatrix} T cos\hat{q}_3[k|k] & 0 \\ T sin\hat{q}_3[k|k] & 0 \\ 0 & T \end{bmatrix}$$

$$\Gamma(q[k]) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{v}} = \begin{bmatrix} \frac{\partial f_1}{\partial v_1} & \frac{\partial f_1}{\partial v_2} \\ \frac{\partial f_2}{\partial v_1} & \frac{\partial f_2}{\partial v_2} \\ \frac{\partial f_3}{\partial v_1} & \frac{\partial f_3}{\partial v_2} \end{bmatrix}_{\substack{q=\hat{q}[k|k] \\ u=u[k]}} = \begin{bmatrix} T cos\hat{q}_3[k|k] & 0 \\ T sin\hat{q}_3[k|k] & 0 \\ 0 & T \end{bmatrix}$$

This results in the linear approximation:

$$q[k+1] \approx \begin{bmatrix} 1 & 0 & -T(u_1[k])sin\hat{q}_3[k|k] \\ 0 & 1 & T(u_1[k])cos\hat{q}_3[k|k] \\ 0 & 0 & 1 \end{bmatrix} q[k] + \begin{bmatrix} Tcos\hat{q}_3[k|k] & 0 \\ Tsin\hat{q}_3[k|k] & 0 \\ 0 & T \end{bmatrix} u[k] + \begin{bmatrix} Tcos\hat{q}_3[k|k] & 0 \\ Tsin\hat{q}_3[k|k] & 0 \\ 0 & T \end{bmatrix} v[k]$$

b) The general form of the system can be solved for the process noise v[k] to yield the following equation:

$$\Gamma v[k] \approx q[k+1] - Fq[k] - Gu[k]$$

Since the ground truth measurements provide the perfect robot states for the time duration of the experiment and the noiseless inputs are given, the above equation can be computed at each time step to find what the process noise was at that step. In MATLAB, v[k] was solved for directly using the left matrix divide (\ ) operator since the matrix Γ is not square (and therefore does not actually have an inverse). This made the equation for the process noise:

$$v[k] \approx \Gamma \ \backslash \left(q[k+1] - Fq[k] - Gu[k]\right)$$

Computing the process noise at each time step and using MATLAB to compute the covariance of the process noise measurements results in the process noise covariance matrix:

$$V = \begin{bmatrix} 0.2591 & 0.0010 \\ 0.0010 & 0.0625 \end{bmatrix}$$

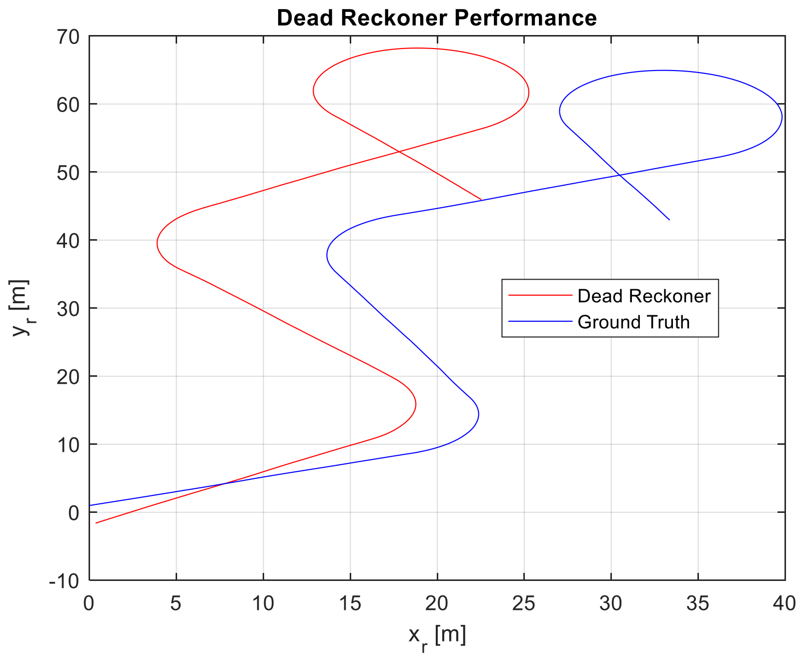
The sensor noise could be computed more directly. The equation for y[k] can be solved for the sensor noise w[k] as follows:

$$w[k] = y[k] - \begin{bmatrix} q_1[k] \\ q_2[k] \end{bmatrix}$$

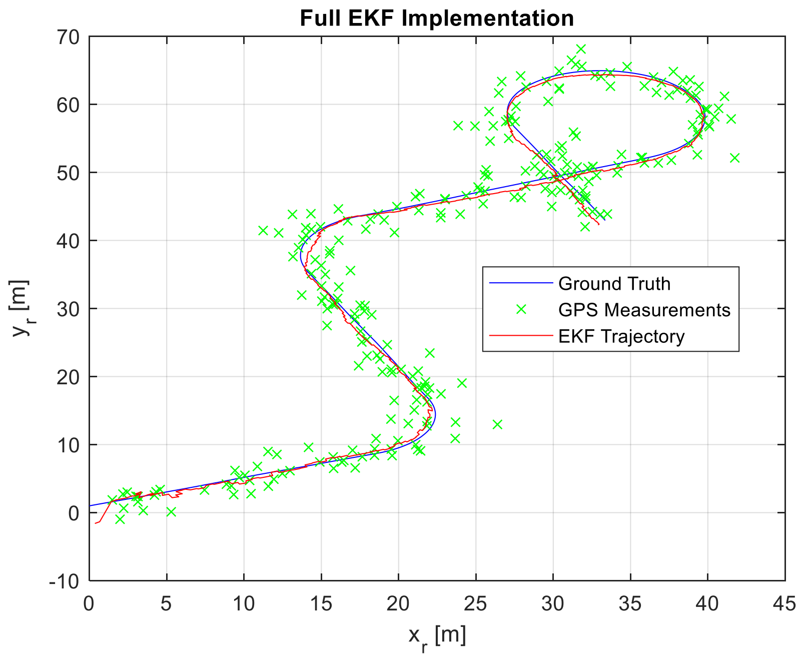
Since y contains the noisy GPS measurements and the state q is assumed to be perfect, their difference yields the sensor noise at each time step. Computing the covariance of these noise values in the same way as before yields the sensor noise covariance matrix:

$$W = \begin{bmatrix} 0.0188 & 6.3247 * 10^{-4} \\ 6.3247 * 10^{-4} & 0.0214 \end{bmatrix}$$

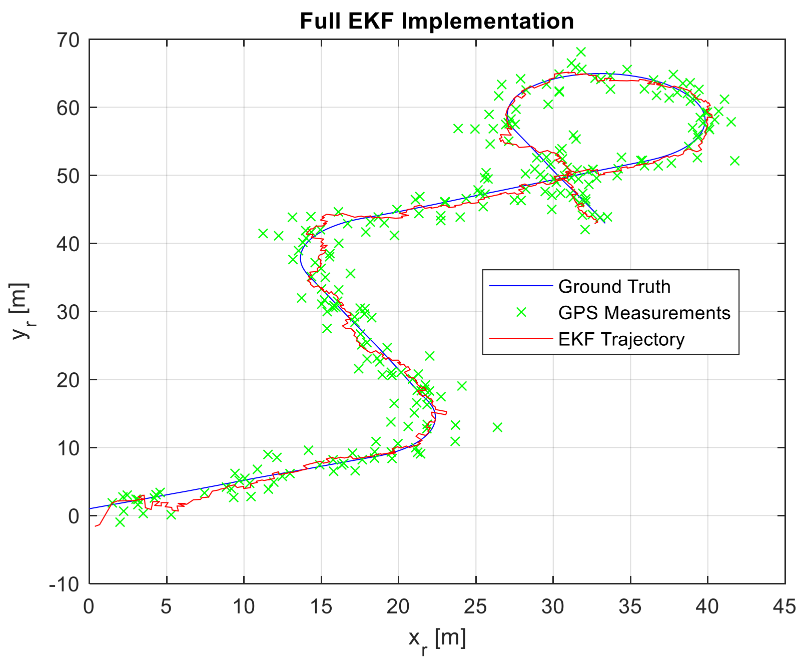
c) i) The performance of the dead reckoner is depicted below:



ii) The results of the full EKF implementation are given below:



iii) The EKF was run again, but this time the sensor noise covariance matrix was scaled down by a factor of 100. The results of this are given below:



As is visually evident between this plot and the previous one, this plot is much more jagged and "jumpy" than the previous one. This is because scaling down the matrix W is akin to saying the GPS measurements are more trustworthy/accurate. Thus, when the GPS measurements are received by the system and they don't correspond with the current state, the update step in the EKF causes the state to snap closer to the GPS measurement since it is believed that the GPS measurements are very close to where the state should actually be. This results in the EKF trajectory jumping around as each new measurement (which usually does not line up perfectly with the ground truth) is received.

2. In creating the particle, three different parameters were tuned: the number of particles, the value of the matrix V, and the value of the matrix W. As a starting point, a point cloud consisting of 50 particles was created since 50 would show the “cloud” nature of the filter while not being too computationally intensive for debugging. The V and W matrices were set to identity as an initial guess.

Once the code was working, the V and W matrices were tuned to provide reasonable results. With V and W set to the identity matrix, the point cloud followed the general shape of the ground truth trajectory (turn left and then right) but failed to find the global location. To help the cloud converge to the actual location of the robot, the diagonal values of W were scaled down so that the measurements from the beacons would have a larger effect on the cloud (i.e., the beacon measurements were perceived as more accurate, so the cloud would consider them more heavily). Scaling all of W by a constant value was not providing enough improvement, so the beacon measurements throughout the simulation were visualized as circles with their centers at the respective beacon. Doing this showed that the measurements from the right beacon (beacon 2) appeared to be slightly more accurate to the ground truth, so the bottom right element of W was scaled down more than the top right.

Scaling W helped the cloud find and stick to the ground truth trajectory, but it still took about half of the run to find the ground truth. To help the cloud explore more at the start to find the ground truth, the values in V were increased to increase the spread of the point cloud. The diagonal elements in V were hand tuned through an iterative process until a reasonable performance was achieved.

With V and W tuned, the number of particles was tuned. The number of particles was increased to values larger than 50 (e.g., 60, 100, 500, 1000), but this led to a decrease in performance. The number of particles was also decreased, but this similarly led to a decrease in performance. As a result, the number of particles was kept at 50.

The final parameters are listed below:

$$num_{particles} = 50 \ ; \ V = \begin{bmatrix} 1.26 & 0 \\ 0 & 1.75 \end{bmatrix} \ ; \ W = \begin{bmatrix} 0.75 & 0 \\ 0 & 0.50 \end{bmatrix}$$

The final choice made was how to choose a single estimate. Originally, the best, single estimate was chosen as the weighted average of all the points in the point cloud. While this gave good estimates for most of the run (when the cloud was unimodal), at the very beginning (when the cloud was still spread out and multimodal), it resulted in the estimate jumping around before finally settling down near the ground truth. To fix this, for the first 3 iterations of the particle filter, the single estimate was set to be the highest weighted particle, as is commonly done for multimodal distributions. This resulted in more stable behavior at the start.