Implementation of the Unscented Kalman Filter with Mobile Robots

Andrew Wilson

Abstract— **This paper describes the implementation of the unscented Kalman filtering algorithm on a robot dataset. The data is drawn from experiments run at the Autonomous Space Robotics Lab at the University of Toronto. A fleet of five robots were run on a course with 15 coded markers. The robots' odometry data was captured as well as sensor readings using a monocular camera. We examine one robot's odometry and measurement data to construct a path using the filtering algorithm. The resulting path is then compared to a dead-reckoning path based on the robot's odometry and to the groundtruth path as recorded by an overhead Vicon motion capture system. The unscented Kalman filter successfully reconstructs the path of the robot with relatively small error compared to the dead-reckoning path.**

I. INTRODUCTION

The unscented Kalman filter is an extension of the Kalman filter and extended Kalman filtering algorithms. Although the extended Kalman filter was created to handle non-linear functions, for particularly non-linear problems, the extended filter can have very poor performance. The unscented version uses a method of discrete sampling, called an unscented transform, to sample a small set of points in the distribution. These points, known as sigma points, are then propagated through the non-linear function. This technique tends to have better performance than the extended filter for highly non-linear functions since the mean and variance are captured more accurately.

This report will cover the modeling of the robot and its sensors, the implementation of the Kalman filter algorithm, and analysis of the algorithm's performance on the UTIAS dataset. The dataset used in this implementation is the MRSLAM_Dataset4 for Robot3.

II. ROBOT MODEL

The fleet of robots used in the dataset collection were five iRobot Create platforms [1]. These are two-wheeled, differential drive robot platforms. Each robot has an on-board monocular camera for landmark sensing.

The coordinate system used in the dataset can be seen in Figure 1. The angle, $\theta = 0$, aligns the world frame with the robot's body frame. The monocular camera is fixed, pointing along the x-axis of the robot's body frame.

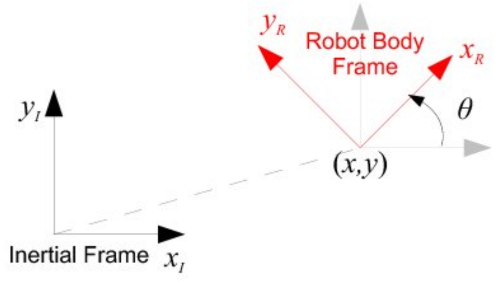

Fig. 1. Robot body frame and world frame coordinate systems.

A. Motion Model

To implement the filtering algorithm, we must create a motion model for the robot. The robot records the forward velocity commands, v, (along the x-axis) and the angular velocity commands, ω, (about the z-axis). There are a variety of model choices for the differential drive robot. One can use a model which accounts for the turning radius of the robot; however, for this implementation, we will use a simplified kinematic motion model for the robot.

At a given point in time, we will assume that the robot's body velocity is linear in the direction of heading. This results in the following equations of motion:

$$\begin{aligned}\dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega\end{aligned}$$

Although this choice of model is a larger approximation than taking the arc of the turning radius into account, this model does not lead to singular values that can arise when using the more sophisticated model. With small time steps, the error between models is minimal. The simplified model is still nonlinear which can be accommodated with the unscented Kalman filter.

B. Measurement Model

The second model that has to be created is the measurement model. This will also be a nonlinear model which takes the robot's current location and the known location of the landmark and returns an expected measurement from the robot. This measurement will then be compared to the actual measurement made by the robot in the filtering algorithm. The robot records the range, r, and bearing, ϕ, and landmark number using its monocular camera.

Given the position of the landmark, (x_ℓ, y_ℓ), that the robot measures and the current state of the robot in the world, (x_r, y_r, θ_r), we can determine the expected observation of the robot. The equations for the measurement are given as follows:

$$r = \sqrt{(x_\ell - x_r)^2 + (y_\ell - y_r)^2}$$
$$\phi = \arctan\left(\frac{y_\ell - y_r}{x_\ell - x_r}\right) - \theta_r$$

This set of nonlinear equations will be used in the measurement update step of the unscented Kalman filter.

III. Unscented Kalman Filter

With the motion model and measurement model equations, we can proceed to implement the unscented Kalman filter algorithm. The algorithm uses a set of sigma points to capture the evolution of the robot through the nonlinear model and measurement updates. If the number of states is given by n, the number of sigma points used is $2n + 1$. Each dimension of the state requires two sigma points, equally spaced on each side of the mean, to propagate through the nonlinear model, with one additional sigma point at the mean.

For this system, we will assume that the both the model and measurement noise are additive and Gaussian. This allows us to just use the state vector in the unscented Kalman filter. A more sophisticated version of the unscented Kalman filter uses an augmented state vector to more accurately represent the noise for robot localization tasks. Details of the augmented approach as well as full details of the unscented filter used in this implementation can be found in [2].

A. Prediction Step

The algorithm begins by defining a set of sigma points such that:

$$\chi^{[0]} = \mu$$
$$\chi^{[i]} = \mu + \left(\sqrt{(n+\lambda)\Sigma)_i}\right)$$
$$\chi^{[i]} = \mu + \left(\sqrt{(n-\lambda)\Sigma)_i}\right)$$

where the number of states, $n = 3$ for this system. The result is 7 sigma points. The parameter, λ, is a scaling parameter which is a function of two parameters, α and β, which determines how far the sigma points are from the mean. We will examine choices of α and β in the results section.

The square root of the covariance and parameter matrix is accomplished by using the Cholesky factorization. This provides a triangular matrix, the columns of which can be used as the matrix square root of each sigma point [3]. If the argument of this square root becomes ill conditioned or singular, the Cholesky factorization will fail, and the algorithm will halt. More sophisticated numerical techniques are available, but not implemented in this version of the algorithm.

After computing the sigma points, the points are then passed through the nonlinear motion model to obtain noise-free predicted sigma values, χ_t^*. A set of weights are calculated from the scaling parameters to compute a single mean and covariance from the sigma values. In this step, a covariance matrix, R_t is added to the predicted covariance, adding the model noise into the filter. At this point, new sigma values can be determined using the predicted mean and covariance. These sigma points capture the overall uncertainty as a result of the prediction step. If there is no additional measurement available, these sigma points become the filtered sigma points for the next timestep.

B. Measurement Update

When a measurement is available, the measurement update proceeds in a similar fashion. The predicted sigma points and the landmark data are passed through the nonlinear measurement model to obtain a predicted observation sigma points, Z_t. The same weights are then used to compute a single predicted observation, $\hat{z}_t$ from the observation sigma points. The measurement variance is then computed with additive measurement covariance, Q_t.

Following the computation of the measurement covariance, the Kalman gain, K_t, is computed and the updated estimate of the robot position, μ and variance, Σ, are returned.

IV. Results

The main purpose of the paper is to highlight the results of the unscented Kalman filter implementation. To this end, we will examine the differences between the dead-recking path and the filtered path, and compare both to the known path of the robot as recorded by the Vicon system. We will also examine the effect of noise parameter choices as well as the scaling parameter choices in the unscented Kalman filter algorithm.

In the implementation of the filters, the discrete odometry data and groudtruth data is interpolated to the first order and then sampled using a fixed timestep of 0.02s. In all of the results, 400 seconds of data is shown, beginning at $t = 1288971880.0$.

The initial mean and covariance of the robot was determined using the groundtruth information. The initial mean was set to the groundtruth location at t_0, and the initial variance was set to the variance of landmark 9 as an estimate of the variance of the Vicon camera system.

A. Dead-reckoning vs. Filtered Path

The most explicit advantage of the unscented Kalman filter algorithm is seen when compared to the dead-reckoning path of the robot. Figure 2 shows the path obtained from the robot's odometry data alone. The predicted path is shown in blue, while the groundtruth is green. The 15 black dots are the landmarks measured by the robot's camera. It is clear that the orientation of the robot is the primary error in the odometry. The robot starts at approximately $(1, -5)$ in the world, and almost instantly, begins to head in the wrong direction. The length of the first few paths are approximately correct between turns, however, the turning angle error is very large.

This large error in the robot's dead reckoning ability is partially due to the simplification of the motion model. Since we are assuming linear trajectories in every timestep, the angular velocity of the robot will not be exactly updated at

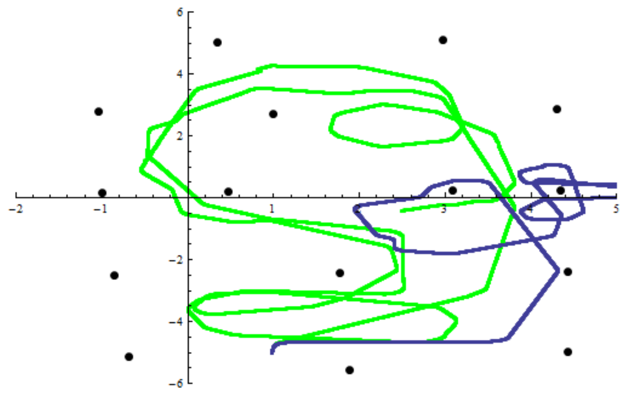

Fig. 2. Dead-reckoning robot path (blue) vs. groundtruth path (green).

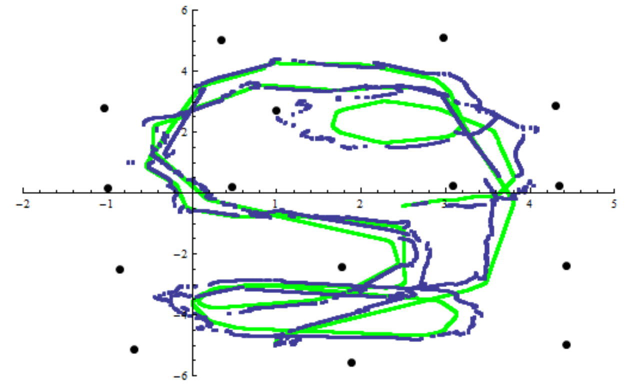

Fig. 3. Filtered path (blue) vs. groundtruth path (green).

each timestep. If we were relying on dead-reckoning alone, the accuracy of the model would have to be much greater.

Fortunately, with measurement updates provided by the robot, the unscented Kalman filter can be used to determine the robot's position using both the model and measurement data. Figure 3 shows the resulting path from the filter in blue.

Although the filtered path performs significantly better than the dead-reckoning path, the filtered path is not exact. Significant deviations from the groundtruth path can be seen, usually where the robot is taking sharp turns. This deviation is due to inaccuracies in the motion model as well as a lack of measurement updates.

B. Noise Covariances

The covariance of the predicted robot position and the measurement updates relies on the amount of uncertainty or noise that is assumed in the model and measurements. These parameters are given by the covariance matrices, R_t and Q_t. Modifying these variances affects how the measurements and predicted states are updated with the Kalman gain.

For this implementation of the unscented Kalman filter, the range of the variance of the noise matrices was limited to approximately an order of magnitude. Increasing or deceasing the relative variance too much caused the algorithm to fail, since the Cholesky factorization can not be computed if the variance matrix is very close to zero [3]. To test the impact of the noise values on the filter, two tests were conducted.

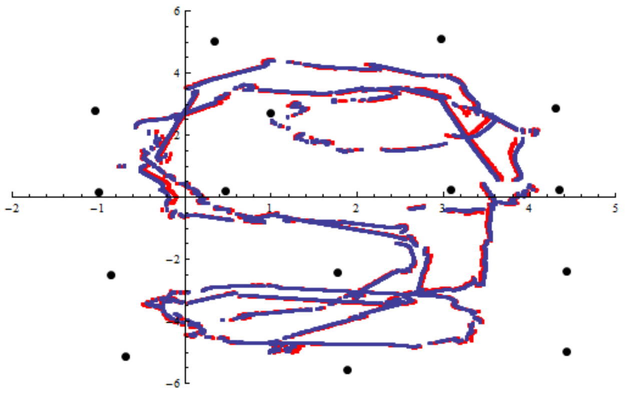

Fig. 4. Filtered path with greater (blue) and less (red) measurement variance.

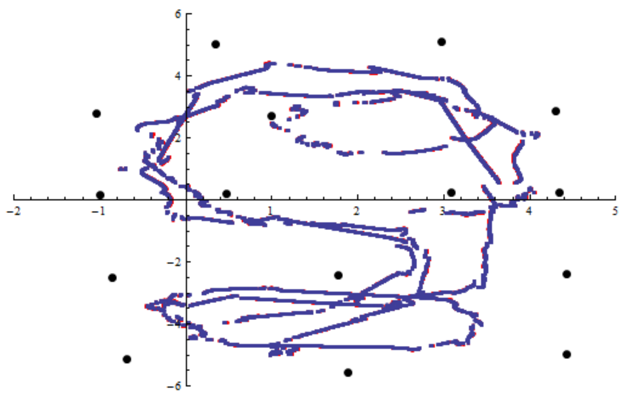

Fig. 5. Filtered path with varying spread parameters.

Figure 4 shows the comparison between noise models. The red path corresponds to less relative noise in the measurement model with $R_t = 0.002\ id(3)$ and $Q_t = 0.00006\ id(2)$ where $id(n)$ is the $n \times n$ identity matrix. The blue path is greater measurement noise with $R_t = 0.008\ id(3)$ and $Q_t = 0.00009\ id(2)$. In both cases, the filtered path is very similar; however, the red path is slightly less smooth than the blue path since the path is skewed more toward the discrete measurement updates.

C. Spread Parameters

The two parameters in the unscented Kalman filter, α and β, were also tested to determine the impact on the filtered path. In the previous tests, we set the parameters to commonly used values, $\alpha = 0.01$ and $\beta = 0$. To test the impact on the filter, we increased the spread with $\alpha = 0.05$ and $\beta = 1$. Figure 5 shows the result of this test. The path with the standard values is in red and the increased values in blue. For this system, it appears that the small parameter change does not have an appreciable effect on the filter. Increasing the values any further caused the filter to fail with variances that approached zero.

This result may be due to the structure of the motion and measurement models. The noise distribution may be captured fairly accurately by the sigma points close to the mean and increasing the spread does not help to capture additional information on the nonlinear system. For different systems or implementations, the spread parameters would be expected to have a measurable effect on the filtering algorithm's performance.

V. Conclusion

Overall, the unscented Kalman filter has been shown to be much more successful at predicting the path of a mobile robot than odometry alone. The resulting path from the filtering algorithm has a relatively bounded error throughout the path, despite errors in the motion and measurement models.

Because of the simplified version of the unscented Kalman filter, the valid noise covariances that the filter could handle were limited. Decreasing the measurement noise did cause the updates to rely more heavily on the measurements; however, the difference in overall path is minimal. An implementation of the more sophisticated localization version may increase the accuracy of the model and range of covariances that can be successfully implemented with the filter.

The spread parameters were also investigated, but the limited range of the parameters led to little change in the overall predicted path of the robot.

Given the non-linearities of the motion and measurement models, the unscented Kalman filter is provides a decent representation of the actual path of the robot. The computation time of the algorithm was 11.8 sec to filter 400 sec of data; therefore, the algorithm could easily be implemented in a real-time environment. With more sophisticated numerical techniques, the covariance matrices could most likely be further tuned to provide even better tracking of the robot trajectory.

References

[1] Leung K Y K, Halpern Y, Barfoot T D, and Liu H H T. The UTIAS Multi-Robot Cooperative Localization and Mapping Dataset. International Journal of Robotics Research, 30(8):969974, July 2011.

[2] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics. The MIT Press, Aug. 2005.

[3] J. Ko, D. Klein, D. Fox, and D. Hahnel, GP-UKF: Unscented Kalman Flters with Gaussian Process Prediction and Observation Models, in Proc. (IROS), 2007.

Unscented Kalman Filter - Dataset #1

■ Data Import & Processing

```
odomdata = Drop[Import["Robot3_Odometry.dat"], {1, 5}];
knownpose =  Drop[Import["Robot3_Groundtruth.dat"], {1, 4}];
measuredata = Drop[Import["Robot3_Measurement.dat"], {1, 4}];
landmarkdata = Drop[Import["Landmark_Groundtruth.dat"], {1, 4}];
odomdata1 = odomdata - Join[{ConstantArray[odomdata[[1, 1]], Length[odomdata]]}ᵀ,
     ConstantArray[0, {Length[odomdata], 2}], 2];
knownpose1 = knownpose - Join[{ConstantArray[odomdata[[1, 1]], Length[knownpose]]}ᵀ,
     ConstantArray[0, {Length[knownpose], 3}], 2];
measuredata1 = measuredata - Join[{ConstantArray[odomdata[[1, 1]], Length[measuredata]]}ᵀ,
     ConstantArray[0, {Length[measuredata], 3}], 2];
odominterp = {Interpolation[Thread[{odomdata1[[All, 1]], odomdata1[[All, 2]]}],
      InterpolationOrder → 1][s], Interpolation[
      Thread[{odomdata1[[All, 1]], odomdata1[[All, 3]]}], InterpolationOrder → 1][s]};
knowninterp = {Interpolation[Thread[{knownpose1[[All, 1]], knownpose1[[All, 2]]}],
      InterpolationOrder → 1][s], Interpolation[
      Thread[{knownpose1[[All, 1]], knownpose1[[All, 3]]}], InterpolationOrder → 1][s],
    Interpolation[Thread[{knownpose1[[All, 1]], knownpose1[[All, 4]]}],
      InterpolationOrder → 1][s]};
groundtruth[t_] := knowninterp /. s → t
modeldata[t_] := odominterp /. s → t
barcodes = Drop[Import["Barcodes.dat"], {1, 4}];
subject[barcode_] :=  barcodes[[Position[barcodes[[All, 2]], barcode][[1, 1]], 1]];
```

■ Algorithm Parameters

```
n = 3; k = 0; α = 0.01; B = 0; kj = -1;
λ = α^2 (n + k) - n;
γ = Sqrt[n + λ];

t0 = 50;
tf = 450;
dt = 0.02;

μ[Round[t0 / dt]] = groundtruth[t0];
Σ[Round[t0 / dt]] = {{0.00004077, 0, 0}, {0, 0.00008785, 0}, {0, 0, 0.00001}};

R[t_] := {{0.00009, 0, 0}, {0, 0.00009, 0}, {0, 0, 0.00009}};
Q[t_] := {{0.008, 0}, {0, 0.008}};

(* Turn the measurement update step on or off *)
measureupdate = 1;
```

■ Filtering Algorithm

```
(* Calculate sigma point weights *)
ωm[i_]  := Piecewise[{{λ / (n + λ), i == 0}, {1 / (2 (n + λ)), i ≠ 0}}];
ωc[i_] := Piecewise[{{λ / (n + λ) + (1 - α^2 + B), i == 0}, {1 / (2 (n + λ)), i ≠ 0}}];

(* Create h(x) measurement model function *)
```

```
h[landmark_] := {Sqrt[(ConstantArray[landmark[[1]], 2 n + 1] - Sigmabar[k][[All, 1]])^2 +
       (ConstantArray[landmark[[2]], 2 n + 1] - Sigmabar[k][[All, 2]])^2],
     ArcTan[ConstantArray[landmark[[1]], 2 n + 1] - Sigmabar[k][[All, 1]], ConstantArray[
         landmark[[2]], 2 n + 1] - Sigmabar[k][[All, 2]]] - Sigmabar[k][[All, 3]]}ᵀ;

(* Determine the first measurement data point after the starting time, t0 *)
j = 1;
While[measuredata1[[j, 1]] ≤ t0,
  j += 1;
 ];

(* Begin timestepping from t0 *)
For[k = Round[t0 / dt] + 1, k < tf / dt, k++,

  (* Calculate noise-free sigma points -
   using Cholesky factorization to provide a matrix square root *)
  Msqrt[k - 1] = CholeskyDecomposition[(n + λ) Σ[k - 1]];
  Sigma[k - 1] = {μ[k - 1], μ[k - 1] + Msqrt[k - 1][[All, 1]], μ[k - 1] - Msqrt[k - 1][[All, 1]],
     μ[k - 1] + Msqrt[k - 1][[All, 2]], μ[k - 1] - Msqrt[k - 1][[All, 2]],
     μ[k - 1] + Msqrt[k - 1][[All, 3]], μ[k - 1] - Msqrt[k - 1][[All, 3]]} // Chop;

  Sigmastar[k] =
   {modeldata[k dt][[1]] Cos[Sigma[k - 1][[All, 3]]], modeldata[k dt][[1]] Sin[Sigma[k - 1][[
         All, 3]]], modeldata[k dt][[2]] ConstantArray[1, 2 n + 1]}ᵀ dt + Sigma[k - 1];

  μbar[k] = Sum[ωm[i] Sigmastar[k][[i + 1]], {i, 0, 2 n}];

  (* Need to handle orientation angle roll-over.  Maps points to [-π,π] *)
  If[μbar[k][[3]] ≥ Pi,
   μbar[k] = Append[μbar[k][[1 ;; 2]], μbar[k][[3]] - 2 Pi];
   Sigmastar[k] = Join[Sigmastar[k][[All, 1 ;; 2]],
     Sigmastar[k][[All, 3]] - {ConstantArray[2 Pi, 2 n + 1]}ᵀ, 2];
  ];
  If[μbar[k][[3]] ≤ -Pi,
   μbar[k] = Append[μbar[k][[1 ;; 2]], μbar[k][[3]] + 2 Pi];
   Sigmastar[k] = Join[Sigmastar[k][[All, 1 ;; 2]],
     Sigmastar[k][[All, 3]] + {ConstantArray[2 Pi, 2 n + 1]}ᵀ, 2];
  ];

  Σbar[k] =
   Sum[ωc[i] Outer[Times, Sigmastar[k][[i + 1]] - μbar[k], Sigmastar[k][[i + 1]] - μbar[k]],
     {i, 0, 2 n}] + R[k];

  (* Calculate predicted sigma points with variance information *)
  Msqrtbar[k] = CholeskyDecomposition[(n + λ) Σbar[k]];
  Sigmabar[k] = {μbar[k], μbar[k] + Msqrtbar[k][[All, 1]], μbar[k] - Msqrtbar[k][[All, 1]],
     μbar[k] + Msqrtbar[k][[All, 2]], μbar[k] - Msqrtbar[k][[All, 2]],
     μbar[k] + Msqrtbar[k][[All, 3]], μbar[k] - Msqrtbar[k][[All, 3]]} // Chop;

  (* If no measurement update, this becomes mean and variance for next timestep *)
  Σ[k] = Σbar[k];
  μ[k] = μbar[k];
```

```
(* Measurement update routine *)
If[measureupdate == 1,
 While[measuredata1[[j, 1]] < k dt,

   (* Filter out measurements of other robots *)
   If[subject[measuredata1[[j, 2]]] ≥ 6 && subject[measuredata1[[j, 2]]] ≤ 20,

     If[k == kj,

      (* In case of two measurements available over one timestep,
      need to recompute sigma points without model update *)
      Msqrt[k] = CholeskyDecomposition[(n + λ) Σ[k]];
      Sigma[k] = {μ[k], μ[k] + Msqrt[k][[All, 1]],
         μ[k] - Msqrt[k][[All, 1]], μ[k] + Msqrt[k][[All, 2]], μ[k] - Msqrt[k][[All, 2]],
         μ[k] + Msqrt[k][[All, 3]], μ[k] - Msqrt[k][[All, 3]]} // Chop;

      Sigmastar[k] = {modeldata[k dt][[1]] Cos[Sigma[k - 1][[All, 3]]],
           modeldata[k dt][[1]] Sin[Sigma[k - 1][[All, 3]]],
           modeldata[k dt][[2]] ConstantArray[1, 2 n + 1]}ᵀ dt + Sigma[k];

      μbar[k] = Sum[ωm[i] Sigmastar[k][[i + 1]], {i, 0, 2 n}];

      If[μbar[k][[3]] ≥ Pi,
       μbar[k] = Append[μbar[k][[1 ;; 2]], μbar[k][[3]] - 2 Pi];
       Sigmastar[k] = Join[Sigmastar[k][[All, 1 ;; 2]],
         Sigmastar[k][[All, 3]] - {ConstantArray[2 Pi, 2 n + 1]}ᵀ, 2];
      ];
      If[μbar[k][[3]] ≤ -Pi,
       μbar[k] = Append[μbar[k][[1 ;; 2]], μbar[k][[3]] + 2 Pi];
       Sigmastar[k] = Join[Sigmastar[k][[All, 1 ;; 2]],
         Sigmastar[k][[All, 3]] + {ConstantArray[2 Pi, 2 n + 1]}ᵀ, 2];
      ];

      Σbar[k] = Sum[ωc[i] Outer[Times, Sigmastar[k][[i + 1]] - μbar[k],
           Sigmastar[k][[i + 1]] - μbar[k]], {i, 0, 2 n}] + R[k];

      Msqrtbar[k] = CholeskyDecomposition[(n + λ) Σbar[k]];
      Sigmabar[k] = {μbar[k], μbar[k] + Msqrtbar[k][[All, 1]], μbar[k] - Msqrtbar[k][[
           All, 1]], μbar[k] + Msqrtbar[k][[All, 2]], μbar[k] - Msqrtbar[k][[All, 2]],
         μbar[k] + Msqrtbar[k][[All, 3]], μbar[k] - Msqrtbar[k][[All, 3]]} // Chop;
     ];

     (* Calculate predicted observation using landmark data *)
     z[k] = measuredata1[[j, 3 ;; 4]];
     landmark[k] = landmarkdata[[subject[measuredata1[[j, 2]]] - 5, 2 ;; 3]];

     Zbar[k] = h[landmark[k]];

     zhat[k] = Sum[ωm[i] Zbar[k][[i + 1]], {i, 0, 2 n}];

     (* In case of predicted measurement angle roll-over.  Maps points to [-π,π] *)
```

```
          If[zhat[k][[2]] ≥ Pi,
           zhat[k] = {zhat[k][[1]], zhat[k][[2]] - 2 Pi};
           Zbar[k] =
            Join[{Zbar[k][[All, 1]]}ᵀ, Zbar[k][[All, 2]] - {ConstantArray[2 Pi, 2 n + 1]}ᵀ, 2];
          ];
          If[zhat[k][[2]] ≤ -Pi,
           zhat[k] = {zhat[k][[1]], zhat[k][[2]] + 2 Pi};
           Zbar[k] =
            Join[{Zbar[k][[All, 1]]}ᵀ, Zbar[k][[All, 2]] + {ConstantArray[2 Pi, 2 n + 1]}ᵀ, 2];
          ];

          S[k] = Sum[ωc[i] Outer[Times, Zbar[k][[i + 1]] - zhat[k], Zbar[k][[i + 1]] - zhat[k]],
              {i, 0, 2 n}] + Q[k];

          Σbarxz[k] = Sum[ωc[i] Outer[Times,
               Sigmabar[k][[i + 1]] - μbar[k], Zbar[k][[i + 1]] - zhat[k]], {i, 0, 2 n}];

          (* Calculate Kalman gain *)
          K[k] = Σbarxz[k].Inverse[S[k]];

          (* update mean and variance with observation based on Kalman gain *)
          μ[k] = μbar[k] + K[k].(z[k] - zhat[k]);
          Σ[k] = Σbar[k] - K[k].S[k].K[k]ᵀ;

          j = j + 1;
          kj = k;
          ,
          j = j + 1;
         ];
       ];
     ];
   ];
```

■ **Robot Animation - Plots last 1000 timesteps of filtered path and groundtruth**

```
Animate[Show[ListPlot[Table[groundtruth[t][[1 ;; 2]], {t, m dt - 1000 dt, m dt, 5 dt}],
    PlotStyle → Red, PlotRange → {{-2, 5}, {-6, 6}}],
   ListPlot[Table[μ[k][[1 ;; 2]], {k, m - 1000, m, 5}]],
   Graphics[{PointSize[Large], Table[Point[landmarkdata[[i, 2 ;; 3]]], {i, 1, 15}]}]],
  {m, Round[t0 / dt] + 1001, tf / dt - 1, 50}, AnimationRate → 400]
```

■ **Plot of total filtered path vs. groundtruth path**

```
Show[ListPlot[Table[groundtruth[t][[1 ;; 2]], {t, t0, tf, dt}],
   PlotStyle → Green, PlotRange → {{-2, 5}, {-6, 6}}],
  ListPlot[Table[μ[k][[1 ;; 2]], {k, Round[t0 / dt], tf / dt - 1, 1}]],
  Graphics[{PointSize[Large], Table[Point[landmarkdata[[i, 2 ;; 3]]], {i, 1, 15}]}]]
```