


ME 656
Autonomous Navigation for Mobile Robots
Spring 2022

Simulation Lab 1

March 31, 2022

*“I pledge that I have abided by the
Graduate Student Code of Academic Integrity.”*

This report has been prepared by:

Aldrin Padua 

ABSTRACT

The goal of the paper was to implement SLAM procedure using two kinds of estimation methods namely, linear least squares (LLS) and linear Kalman filter (LKF), to enable the robot to localize itself while simultaneously mapping the unknown environment and its landmarks. In LLS, the estimation was implemented as a single batch where all odometry and range sensor measurements (and their respective representations) across all time steps are collected and pooled into a matrix and a vector before solving. In contrast, LKF estimation was implemented at every discrete time step (incremental). The LKF method without landmark-sensing, performed satisfactorily with mean absolute error (MAE) peak magnitude of around 0.26; very similar to the performance of the LLS method without landmark-sensing. However, the LKF method with landmarks-sensing was proven to perform superior against the counterpart LLS methods (also with landmark measurements), both having improved accuracy but the former having a MAE peak magnitude of only around 0.098, and the latter having 0.21. The accuracy of estimation in LKF was greatly influenced by the utilization of environmental features/landmarks providing a reduction in MAE peak magnitude of about 62%. On the other hand, the accuracy of the LLS method, although improved, was only able to reduce the MAE peak magnitude by about 20%. LLS method's accuracy was further enhanced during loop closure ultimately ending up with a MAE peak magnitude of around 0.146 providing an ultimate reduction in MAE peak magnitude of around 44%; still inferior compared to LKF with landmark-sensing.

INTRODUCTION

In autonomous mobile robotics, the most critical parts are localization and mapping (if environment is unknown). Localization is process of the robot being able to estimate its location in the map at specific point in time (or its trajectory), while mapping is the process of the robot recognizing areas in the previously unknown environment through position estimation of distinct features or landmarks. Both the mentioned processes can be done simultaneously, correlating one to the other to improve accuracy, in the procedure called Simultaneous Localization and Mapping (SLAM).

SLAM can be implemented using different methods. This paper explores the implementation of two of those methods namely, (1) Linear Least Squares (LLS), and (2) Linear Kalman Filter (LKF), with the intention of evaluating the merits of each technique and comparing them against each other.

For this paper, the following problem setup is used:

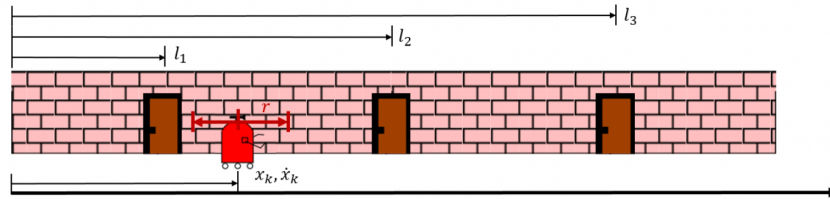


Figure 1. An illustration of the robot's workspace and range sensor capability.

where r represents the range-sensing capability of the robot, l_1 , l_2 , and l_3 are the landmark positions with respect to the starting point, and x_k and \dot{x}_k are the current position and velocity of the robot, respectively.

In addition to the workspace setup, the following assumptions are made:

- All discrete-time sensing and state estimation operates with a fixed time-step of 0.1 seconds.
- The hallway is 10m long, and the robot's sensing range is 0.5m.
- The robot moves with a constant velocity of 0.1 m/s.
- The robot starts its traversal of the hallway at $x_0 = 0$ (the robot knows this with high certainty), and drives at constant velocity until it reaches $x_f = 10$ m.
- The landmarks are located at $l_1 = 2$ m, $l_2 = 5$ m, $l_3 = 8$ m (the robot does not know this, but its range measurements, which you will simulate, will reflect this).
- The robot's odometry measurements are corrupted with zero-mean, additive Gaussian white noise with standard deviation 0.1 m/s.
- The robot's range measurements to landmarks (when the landmarks are within range) are corrupted with zero-mean, additive Gaussian white noise with standard deviation 0.01 m.

- There is no environmental process noise influencing the robot's motion – it moves in a completely deterministic manner, exactly as intended (but the robot does not know this).

The merits are evaluated and analyzed through comparing the mean absolute errors between the estimations and the true values across all 1000 trials for each technique.

THEORY AND EXPERIMENTAL PROCEDURE

I. Linear Least Squares “Batch” SLAM Without Landmarks

In the first linear least squares technique, the problem was posed as the equation $Ax = b$, where A was a constant matrix, whose rows were the correlations of the previous and the current time step positions. The vector x represented the position variables per time step. The vector b contained all the noisy odometry measurements directly corresponding to each row in the Ax vector. The problem was then constructed as

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}}_A \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_x = \underbrace{\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_b$$

The noisy odometry measurements were constructed as follows:

$$b_{k_odom} = [v_{robot} + sd_o * randn()] * delta_t$$

where b_{k_odom} was the odometry measurement for k^{th} time step, v_{robot} was the constant velocity of the robot, sd_o was the standard deviation of the odometry measurement's noise, and $randn()$ was the function to generate random number to provide variation in measurements.

where n was the total number of time steps. The batch SLAM was solved as $x = A^{-1}b$. Note that since the correlation matrix A was constant, A^{-1} needed to be computed just once across the entire 1000-trial run in the algorithm.

II. Linear Least Squares “Batch” SLAM With Landmarks

The problem and the solution for this part was essentially like that of Part I, except that the dimensions of the matrix, A , and the vectors, x and b , have expanded in accordance with the addition of the three landmarks whose positions were also unknown and were part of the x vector. The correlations of each landmark to each of the robot positions were added to the rows of matrix A , and their corresponding noisy measurements coming from the range sensors, were appended to vector b .

To mimic the noisy measurements for the range sensors, b_k , were computed in MATLAB as

$$b_{k_range} = v_{robot}(t_{lm} - t_r) + sd_r * randn()$$

where b_k was the noisy range sensor measurement of the landmark at k^{th} time step, t_{lm} was the time when the robot was expected to be aligned with the center of the landmark had it not been noise-corrupted, t_r was the current time, and sd_r was the standard deviation of the range sensor measurement's noise.

III. Linear Kalman Filter "Incremental" SLAM Without Landmarks

While the linear least squares technique solves the problem in one batch step, linear Kalman Filter provides an estimate for every discrete step in time. Kalman filter is basically a predictor-corrector technique which means that the algorithm is essentially divided into two main parts: (1) time update (predictor), and (2) measurement update (corrector).

In this part, the state vector was represented as

$$\widehat{x}_k = \begin{bmatrix} \dot{\widehat{x}}_k \\ \widehat{x}_k \end{bmatrix}$$

where k was current time step, x_k was the current position estimate, and $\dot{\widehat{x}}_k$ was the current velocity estimate.

In the algorithm implementation, the predictor problem was posed as

$$\begin{aligned} \widehat{x}_k^- &= A\widehat{x}_{k-1} + Bu_k \\ P_k^- &= AP_{k-1}A^T + Q \end{aligned}$$

where \widehat{x}_k^- represented the intermediate x value used to compute the new and actual x estimate in the corrector part, A was the previous and current estimates' correlation matrix, x_{k-1} was the previous state estimate, P_k^- was the intermediate state estimation covariance matrix used to compute for the actual value in the corrector part, P_{k-1} was the previous state estimation error covariance matrix, and Q is the process noise covariance matrix. In this paper, it was assumed that there was no external control input u_k , hence there was no B matrix as well. So essentially, the intermediate x position value in the predictor part was only

$$\widehat{x}_k^- = A\widehat{x}_{k-1} .$$

Matrix A was set to $\begin{bmatrix} 1 & 0 \\ \text{delta}_t & 1 \end{bmatrix}$, essentially declaring the that the velocity is expected to be constant in time. Matrix P was initially set to $\begin{bmatrix} 1 & 0 \\ 0 & 0.0001 \end{bmatrix}$, representing the initial degree of uncertainty about the initial values. Matrix Q was assumed to be $\begin{bmatrix} 10^{-8} & 0 \\ 0 & 0 \end{bmatrix}$.

The corrector problem was posed as

$$\begin{aligned} K_k &= P_k^- H^T (H P_k^- H^T + R)^{-1} \\ \widehat{x}_k &= \widehat{x}_k^- + K_k (z_k - H \widehat{x}_k^-) \\ P_k &= (I - K_k H) P_k^- \end{aligned}$$

where K_k was the Kalman gain, H was the measurement matrix, R was the sensor noise covariance, \widehat{x}_k was the current state estimate, z_k was the measurement vector, and P_k was the current state estimation error covariance matrix.

Since there were only odometry measurements coming in, H was set to $[1 \ 0]$ and, R was set to $(0.1)^2$ which corresponds to the variance of the odometry measurement's noise.

IV. Linear Kalman Filter "Incremental" SLAM With Landmarks

The algorithm was essentially the same with part III, except that the dimensions of vectors and matrices have changed due to the addition of the landmark measurements.

The state vector has transformed into

$$\widehat{x}_k = \begin{bmatrix} \dot{\widehat{x}}_k \\ \widehat{x}_k \\ \widehat{l}_1 \\ \widehat{l}_2 \\ \widehat{l}_3 \end{bmatrix}$$

with corresponding correlation matrix of

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \text{delta_t} & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

deeming the landmark positions to be constants.

The initial estimate for P was also adjusted to reflected uncertainty about the landmarks, with value

$$P_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0.0001 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

H and R vectors were also set to be dynamic to give way to time steps when there were not just odometry measurements coming in, but also range sensor measurements to the landmarks.

When landmarks were detected, H vector turned into matrix as follows:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & l_1 & l_2 & l_3 \end{bmatrix}$$

where l_1 , l_2 , and l_3 values were either 1 or 0, depending on which landmark were being detected.

R vector also turned into a matrix when landmarks were within range and took the form as follows:

$$R = \begin{bmatrix} sd_o^2 & 0 \\ 0 & sd_r^2 \end{bmatrix}$$

where sd_o^2 and sd_r^2 were the noise variances of the odometry and the range sensor measurements, respectively.

V. Linear Least Squares “Batch” SLAM With Landmarks (Loop Closure)

The algorithm was essentially the same with Part II except that the number of time steps has been doubled to accommodate the reversal of the robot from the 10meter-end of the hall to the 0meter-end. Also, the velocity direction has been reversed.

RESULTS AND DISCUSSION

I. Linear Least Squares “Batch” SLAM Without Landmarks

The code output shows an upward trend for the mean absolute error (MAE) of the state estimates from $t = 0\text{seconds}$ to $t = 100\text{seconds}$ across 1000 trials as shown in Figure 2.

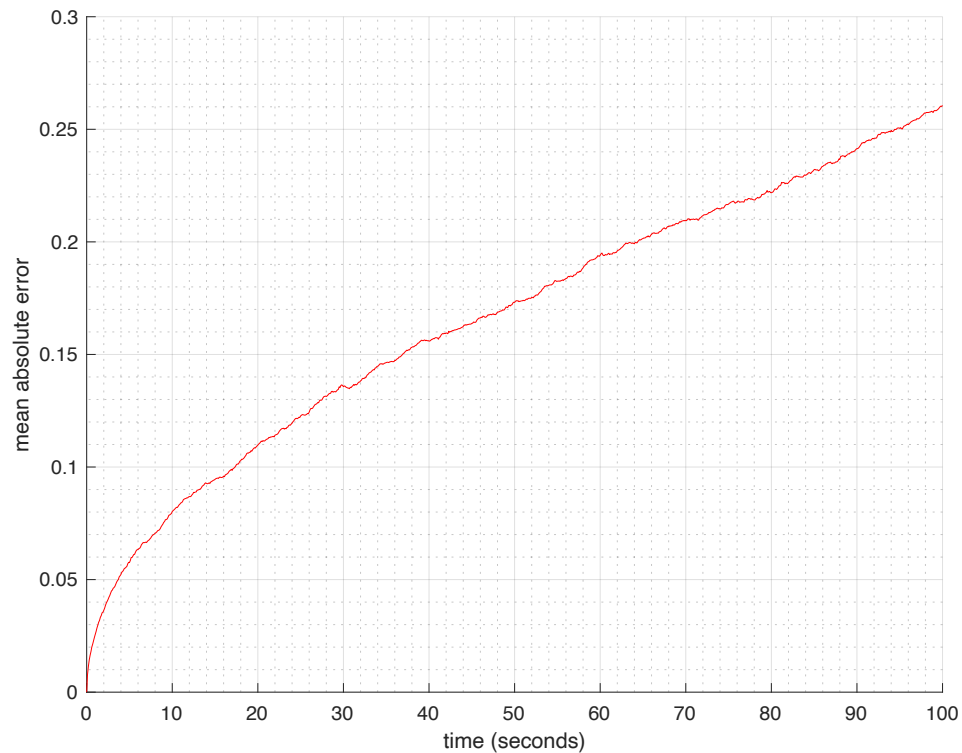


Figure 2. ‘MAE vs Time’ Plot of the output of Linear Least Squares “Batch” SLAM (without landmarks) shows an upward MAE trend during the forward march in space-time across 1000 trials.

This is an expected behavior since the measurements per time step are corrupted with noise, and that the noise is expected to have piled up by the end of the time-run. The MAE peaks at around 0.26 indicating the inaccuracy of the position estimate for 10m (end of the hall), with 0 being the minimum value reflecting the accuracy at the initial position (0m).

II. Linear Least Squares “Batch” SLAM With Landmarks

With the integration of landmark-sensing, the code output shows a still upward MAE trend, but with lesser magnitude, from $t = 0\text{seconds}$ to $t = 100\text{seconds}$ across 1000 trials as shown in Figure 3.

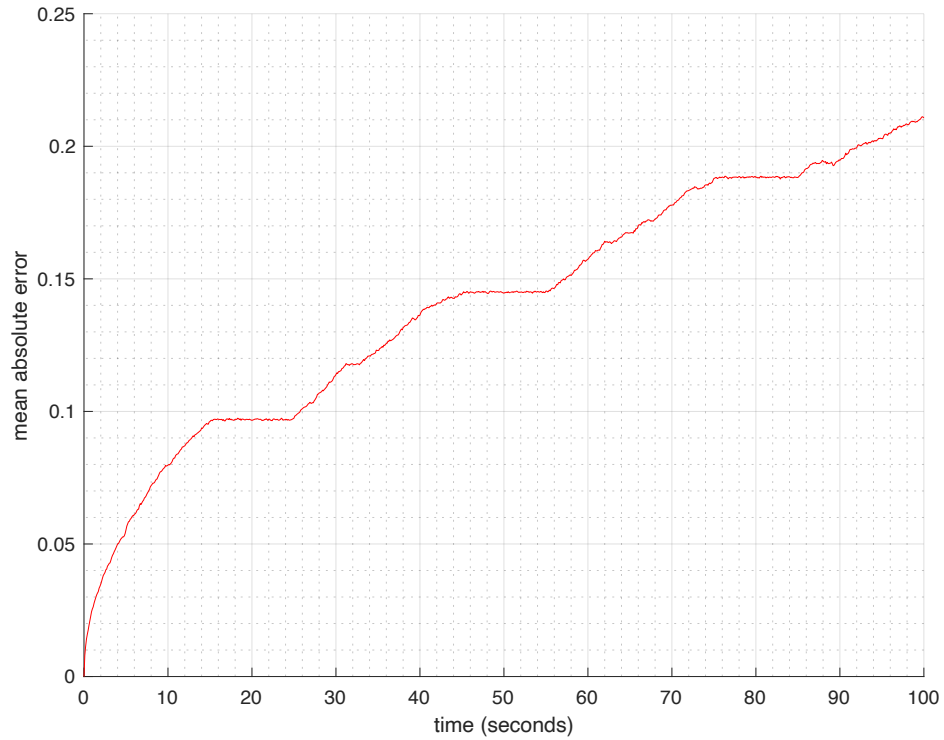


Figure 3. 'MAE vs Time' Plot of the output of Linear Least Squares "Batch" SLAM (with landmarks) shows an upward MAE trend, but with lesser magnitude compared to that of Part I, during the forward march in space-time across 1000 trials.

The MAE peaks at around 0.21 which means that the addition of landmarks and, hence, the establishment of the correlation between the robot's position estimate and those landmarks helped control the error growth. This is further proven by examining Figure 3. In the figure, the regions where the growth of error were curbed are visually represented by the almost horizontally flat segments on the curve. The mentioned segments occur at the time intervals of $t = 15 - 20 \text{ secs}$, $t = 45 - 55 \text{ secs}$, and $t = 75 - 85 \text{ secs}$. These are the intervals where the landmarks are within the range of the robot's sensor.

Intuitively, the reason for this kind of behavior roots from the fact that for the state estimates at time steps that fall within the above-mentioned intervals, we have twice the correlation reference: (1) correlation between the robot's previous and current position estimates, and (2) correlation between the robot and the landmark's current position estimates. Hence, more accuracy is achieved. However, it is important to note that, although the growth of error was curbed, the linear least squares method was only able to hold it for the time intervals where the landmarks were detectable. The error continues to grow rapidly as soon as a landmark disappears.

III. Linear Kalman Filter “Incremental” SLAM Without Landmarks

The code output also shows an upward trend for the MAE of the state estimates from $t = 0seconds$ to $t = 100seconds$ across 1000 trials, very similar to that of Part I, as shown in Figure 4.

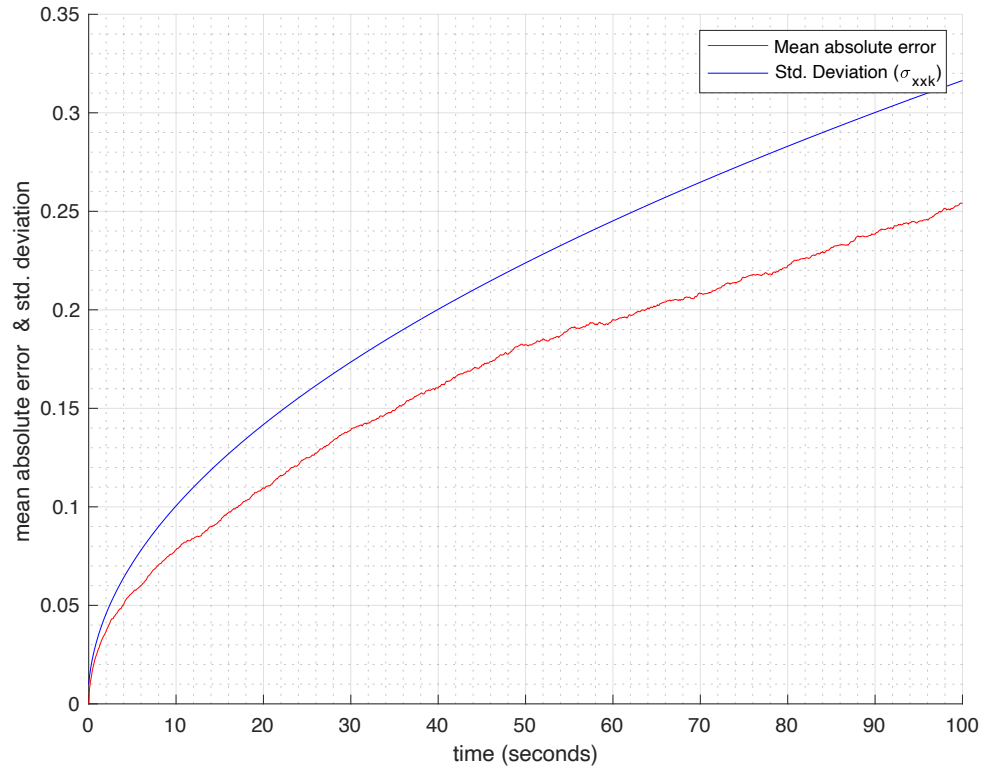


Figure 4. ‘MAE vs Time’ Plot of the output of Linear Kalman Filter “Incremental” SLAM (without landmarks) also shows an upward MAE trend during the forward march in space-time across 1000 trials.

The output is indicative of the performance of the linear Kalman filter without landmark-sensing (LKFWOL) being equally satisfactory compared to its counterpart linear least squares technique without the landmark-sensing (LLSWOL) with both having a peak MAE magnitude of around 0.26 and a very similar curve (compare Figure 2 and Figure 4).

This is expected because of the nature of LKFWOL whose state estimation is purely dependent on the current and previous states, just like in the LLSWOL. During the forward march in space-time, the degree of uncertainty about the current state only grows further since there was no correlation to other environmental factors (with certain degree of certainty) that could potentially curb the growth of error.

As the Kalman filter propagates in time, the system's trust to its own estimations decreases, thus, the error increases almost linearly with the degree of uncertainty. This is the why the predicted standard deviation of the position estimate's error, σ_{xx_k} , (indicated by the blue curve in Figure 4) also increased in time.

IV. Linear Kalman Filter “Incremental” SLAM With Landmarks

The code output shows the MAE and the σ_{xx_k} trend using linear Kalman filter with landmark-sensing as shown in Figure 5.

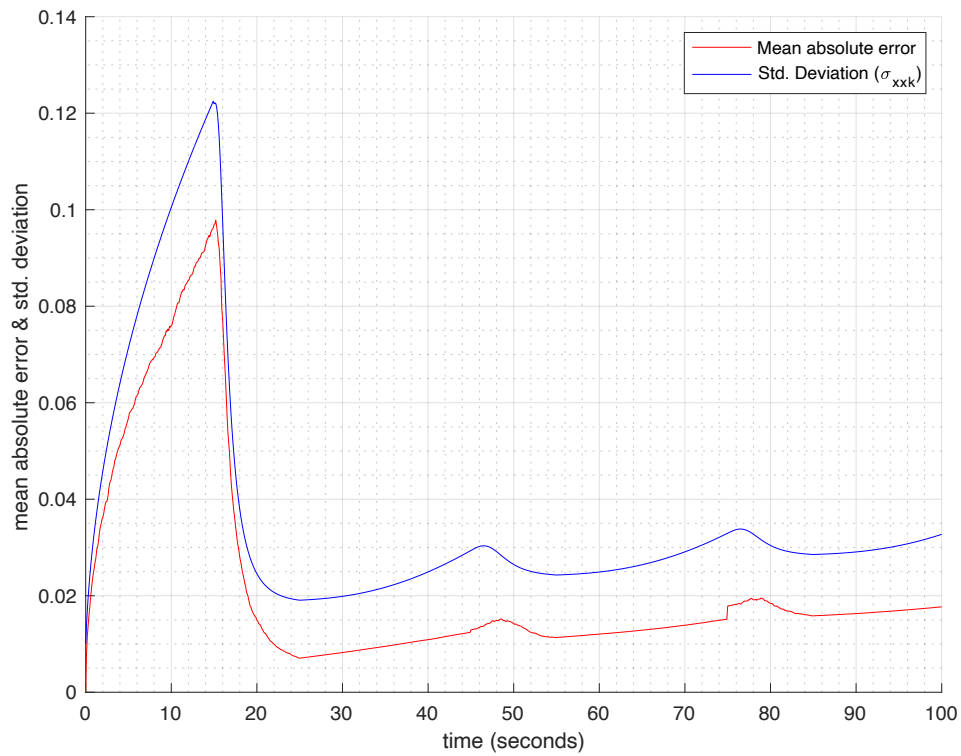


Figure 5. ‘MAE vs Time’ Plot of the output of Linear Kalman Filter “Incremental” SLAM (with landmarks) shows a greatly curbed MAE trend during the forward march in space-time across 1000 trials.

The output indicates that the addition of landmark-sensing greatly improves the performance of the linear Kalman filter during its forward evolution in space-time. Unlike the improvement that is shown in Part II for linear least squares with landmark-sensing (LLSWL) where the error growth is restrained to an almost equal magnitude at certain time intervals before it starts to grow again, the error growth using linear Kalman filter with landmark-sensing (LKFWL) is not just curbed but also, the error magnitude for state estimates after sensing the first landmark is significantly reduced

as shown in Figure 5. At every time step after the first detection of the first landmark (approximately after $t=15$ seconds), the MAE drastically reduces up until the point where it reaches its minimum value (around 0.007 at about $t=25$ seconds). The growth of the error from that point onwards is then seen to be increasing very slowly (as opposed to LLSWL), maintaining the MAE approximately below the magnitude of 0.018. It goes on until it detects another landmark and start to curb and reduce the error again. This leads to a minimal MAE by the end of the time-run with magnitude of about 0.033.

The LKFWL output proves that the growth of error occurs when the system has no environmental reference, such as a landmark, which can be correlated to the robot's current position estimate to increase the certainty about the states and potentially impact the error. This region is seen in Figure 5 as the spike in error within the intervals of $t=0$ to approximately $t=15$ seconds. However, once a landmark is detected, the degree of certainty about the position estimates significantly increases, curbing the error growth and reducing the error magnitude for the succeeding steps. The degree of certainty about the robot's state estimate can be correlated and viewed as the decline in magnitude of the standard deviation of the position estimate's error, σ_{xx_k} , represented by the blue curve in Figure 5. The LKFWL method is shown to have predicted it with much more accuracy compared to LKFWOL (Figure 4).

As mentioned above, the effects of landmark-sensing differ for the linear least squares method and the linear Kalman filter. With linear least squares method, landmark-sensing is only able to impede the growth of error and restrain it to an approximately constant magnitude at certain time intervals, whereas the linear Kalman filter can curb the growth as well as reduce the magnitude of succeeding errors. This characteristic results from the fact that in Kalman filter, the current states are correlated to the previous one from the first time-step until the last. This means that the established relationship is propagated and becomes part of the system as it evolves in time. Therefore, the moment that the system detects a landmark, the landmark estimate will be part of the system until the loop ends; the correlation does not vanish at time steps when a landmark is no longer detectable. On the contrary, in linear least squares, the correlation between the landmark and the robot's position estimate only appears at time steps where the landmarks are detectable, and then disappears again after the same landmarks fall out of range.

V. Linear Least Squares "Batch" SLAM With Landmarks (Loop Closure)

The output plot shows great improvement in performance/accuracy when the robot was made to re-observe landmarks as shown in Figure 6.

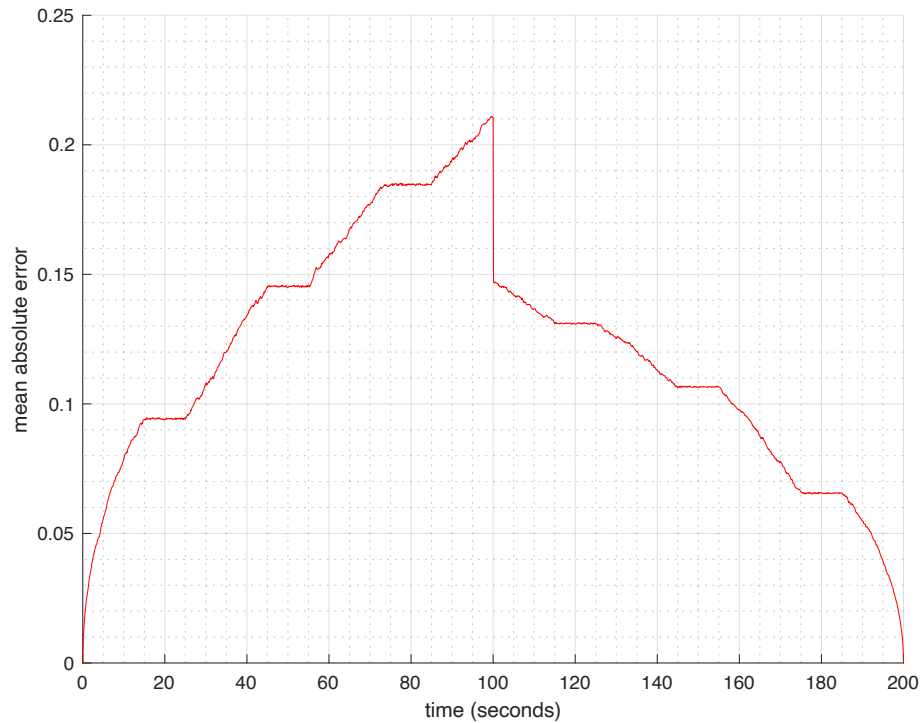


Figure 6. 'MAE vs Time' Plot of the output of Linear Least Squares "Batch" SLAM (with landmarks) shows a great improvement in estimation accuracy when the robot is made to re-observe landmarks, also known as loop-closure.

The first half of the figure is the representation of the process performed in Part II where the robot travelled forward in space and time (i.e., from the robot's initial position to the end of the hall within $t = 0 - 100 \text{ seconds}$). The second half represents the MAE trend in reverse position (with respect to the first half) where the robot is made to re-observe landmarks by travelling backward in space and forward in time (i.e., from the end of the hall down to the initial position of the robot within $t = 100.1 - 200 \text{ seconds}$). This process of re-observing landmark is known as loop-closure.

With loop-closure the accuracy of the position estimates is shown to have significantly increased with the second half turning in a peak MAE of only around 0.146, as opposed to the peak MAE of around 0.21 in the first half. This means that the re-observation of landmarks further improved the robot's ability to curb the growth of error.

Again, intuitively, the process of loop-closure can be seen as adding more references or data points to the correlation of the robot's position estimate and that of the landmarks. Thus, strengthening the least squares regression process and improving

the accuracy of results. However, this improvement in accuracy is still inferior as compared to that of the linear Kalman filter's (compare Figure 6 and Figure 5).

CONCLUSION

Linear least squares “batch” SLAM without landmarks has MAE peaking at around 0.26 as the robot marches forward in space and time. The accuracy can be improved, bringing down MAE to a peak magnitude of around 0.21, by utilizing landmark readings and correlating them to the robot’s position estimates. The least squares technique can further be enhanced, significantly improving the accuracy of estimation, bringing down the peak MAE magnitude to around 0.146 (approximately 44% improvement) through loop-closure. This is done by marching the robot backward in space and forward in time, re-estimating and re-observing the landmarks positions.

The linear Kalman filter “incremental” SLAM without landmark-sensing is seen to have similar performance with the counterpart linear least squares technique without landmark-sensing, with MAE peaking at around 0.26 for both. Without landmark sensing the system only grows more uncertain and the error only increases with time as evaluated through σ_{xx_k} . However, the linear Kalman filter proves to be superior in every aspect when integrated with landmark-sensing yielding a peak MAE magnitude of only 0.098 (approximately 62% improvement).

Also, the linear Kalman filter method with landmark-sensing cannot only curb the error growth, but also significantly reduce the magnitude of error as it evolves in time. This is as opposed to its counterpart linear least squares method where the landmark-sensing integration was only able to curb the error growth at certain intervals but not reduce its magnitude. This roots from the fact that with Kalman filter, the correlation of states is kept from start to finish and becomes part of the system, thus, the correlation between landmark estimates to that of the robot’s does not disappear when the landmark falls out of range. The opposite happens with linear least squares where the correlation between the landmark and robot’s position vanishes as soon as the landmark is no longer detectable.

REFERENCES

1. Englot, Brendan. "Overview of State Estimation and Kalman Filters." Class lecture, Autonomous Navigation for Mobile Robots, Stevens Institute of Technology, Hoboken, New Jersey, February 04, 2022.
2. Englot, Brendan. "Simultaneous Localization and Mapping (SLAM)." Class lecture, Autonomous Navigation for Mobile Robots, Stevens Institute of Technology, Hoboken, New Jersey, February 11, 2022.

APPENDIX

I. Linear Least Squares “Batch” SLAM Without Landmarks Source Code

Filename: *driver_LS_1.m*

```
clc;
clear;
close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some parameters you can use to set up your solution of SimLab 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

length_hallway = 10;
sensor_range = 0.5;

landmarks = [2; 5; 8];

num_landmarks = length(landmarks);

stdev_odometry = 0.1; % m/s
stdev_range = 0.01; % m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Robot's "Ground Truth" Trajectory when it travels the hallway
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

delta_t = 0.1; % time is discretized with a time-step of 0.1 seconds
v_robot = 0.1; % robot travels at a constant speed of 0.1 m/s

t_terminal = length_hallway/v_robot; % time that we reach the end of hall

t_vector = 0:delta_t:t_terminal; % time vector for robot's trajectory
x_vector = 0:v_robot*delta_t:length_hallway; % robot's true position in time
v_vector = ones(1,length(t_vector)).*v_robot; % robot's true velocity in
time

num_states = length(x_vector); % number of discrete-time states in trajectory

%% LEAST SQUARES SOLUTION

N = 1000; % number of trials
n = length(t_vector); % number of steps
x_pos_overall = []; % records of all positions across n-time steps and across
N trials

% populate the constant matrix A
A = [];

for j = 1:n
    if isempty(A)
        A = [1 zeros(1,n-1)];
    else
```

```

        A(j,:) = [zeros(1,j-2) -1 1 zeros(1,n-j)];
    end
end

% get the inverse of A
pinv_A = pinv(A);

% LS Loop
for i = 1:N
    x_odom = (v_robot+stdev_odometry*randn(1,1001))*delta_t;
    b = [0; x_odom(2:end)'];
    X = pinv_A*b;
    x_pos_overall(:,i) = X;
end

%% mean absolute error plot
abs_err = [];
mean_abs_err = [];
[m,n] = size(x_pos_overall);
for i = 1:n
    abs_err(:,i) = abs(x_vector' - x_pos_overall(:,i));
end

for i = 1:m
    mean_abs_err(i) = mean(abs_err(i,:));
end

line(t_vector,mean_abs_err(:),'Color','r');
grid on
grid minor
xlabel('time (seconds)');
ylabel('mean absolute error');

```

II. Linear Least Squares “Batch” SLAM With Landmarks Source Code

Filename: *driver_LS_2.m*

```

clc;
clear;
close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some parameters you can use to set up your solution of SimLab 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

length_hallway = 10;
sensor_range = 0.5;

landmarks = [2; 5; 8];

num_landmarks = length(landmarks);

```

```

stdev_odometry = 0.1; % m/s
stdev_range = 0.01; % m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Robot's "Ground Truth" Trajectory when it travels the hallway
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

delta_t = 0.1; % time is discretized with a time-step of 0.1 seconds
v_robot = 0.1; % robot travels at a constant speed of 0.1 m/s

t_terminal = length_hallway/v_robot; % time that we reach the end of hall

t_vector = 0:delta_t:t_terminal; % time vector for robot's trajectory
x_vector = 0:v_robot*delta_t:length_hallway; % robot's true position in time
v_vector = ones(1,length(t_vector)).*v_robot; % robot's true velocity in
time

num_states = length(x_vector); % number of discrete-time states in trajectory

%% LEAST SQUARES SOLUTION

l_count = 3;
N = 1000; % number of trials
n = length(t_vector); % number of steps
x_pos_overall = []; % records of all positions across n-time steps
                  % and across N trials

% populate the constant matrix A
A = [];

for j = 1:n
    if isempty(A)
        A = [1 zeros(1,n+l_count-1)];
    else
        A(j,:) = [zeros(1,j-2) -1 1 zeros(1,n+l_count-j)];
    end
end

index = 1;
for j = 1:n
    % detection of first landmark
    if t_vector(j) >= 15 && t_vector(j) <= 25
        A(n+index,:) = [zeros(1,j-1) -1 zeros(1,n-j) 1 0 0];
        index=index+1;
    end

    % detection of 2nd landmark
    if t_vector(j) >= 45 && t_vector(j) <= 55
        A(n+index,:) = [zeros(1,j-1) -1 zeros(1,n-j) 0 1 0];
        index=index+1;
    end

    % detection of 3rd landmark
    if t_vector(j) >= 75 && t_vector(j) <= 85
        A(n+index,:) = [zeros(1,j-1) -1 zeros(1,n-j) 0 0 1];
    end
end

```

```

        index=index+1;
    end

end

% get the pseudoinverse of A
pinv_A = pinv(A);

% LS Loop
for i = 1:N
    x_odom = (v_robot+stdev_odometry*randn(1,1001))*delta_t;
    b = [0 x_odom(2:end)];

    k = length(b)+1;
    for j = 1:n
        % detection of first landmark
        if t_vector(j) >= 15 && t_vector(j) <= 25
            b(k) = (v_robot*(20-t_vector(j))+stdev_range*randn());
            k = k+1;
        end

        % detection of 2nd landmark
        if t_vector(j) >= 45 && t_vector(j) <= 55
            b(k) = (v_robot*(50-t_vector(j))+stdev_range*randn());
            k = k+1;
        end

        % detection of 3rd landmark
        if t_vector(j) >= 75 && t_vector(j) <= 85
            b(k) = (v_robot*(80-t_vector(j))+stdev_range*randn());
            k = k+1;
        end

    end

    X = pinv_A*b';
    x_pos_overall(:,i) = X;
end

%% mean absolute error plot
abs_err = [];
mean_abs_err = [];
[m,n] = size(x_pos_overall);
for i = 1:n
    abs_err(:,i) = abs(x_vector' - x_pos_overall(1:end-3,i));
end

for i = 1:m-3
    mean_abs_err(i) = mean(abs_err(i,:));
end

line(t_vector,mean_abs_err(:),'Color','r');
grid on
grid minor
xlabel('time (seconds)');

```

```
ylabel('mean absolute error');
```

III. Linear Kalman Filter “Incremental” SLAM Without Landmarks Source Code

Filename: *driver_KF_1.m*

```
clc;
clear;
close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some parameters you can use to set up your solution of SimLab 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

length_hallway = 10;
sensor_range = 0.5;

landmarks = [2; 5; 8];

num_landmarks = length(landmarks);

stdev_odometry = 0.1; % m/s
stdev_range = 0.01; % m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Robot's "Ground Truth" Trajectory when it travels the hallway
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

delta_t = 0.1; % time is discretized with a time-step of 0.1 seconds
v_robot = 0.1; % robot travels at a constant speed of 0.1 m/s

t_terminal = length_hallway/v_robot; % time that we reach the end of hall

t_vector = 0:delta_t:t_terminal; % time vector for robot's trajectory
x_vector = 0:v_robot*delta_t:length_hallway; % robot's true position in time
v_vector = ones(1,length(t_vector)).*v_robot; % robot's true velocity in
time

num_states = length(x_vector); % number of discrete-time states in trajectory

N = 1000; % number of trials

%% Define A and initial states
A = [1 0;
     delta_t 1];

x0 = 0;
P0 = [1 0;
     0 0.0001];
```

```

Q = [10^(-8) 0;
      0 0];

% sensor noise covariance matrices
R = stdev_odometry^2;

x_pos_overall = []; % all x positions across all time steps per trial
stdev_xk_err = []; % std. dev. of position estimate's error across all
                  % time steps

for j = 1:N
    % x positions via odometry measurements with noise
    odom = [0];
    for i = 2:length(t_vector)
        odom(i) = v_robot+stdev_odometry*randn();
    end

    % set initial values
    x_prev = [v_robot x0]';
    P_prev= P0;

    % all positions per trial
    x_pos = [x_prev(2)];

    % for plotting stddev of position estimate's error
    if j== N
        stdev_xk_err(1) = sqrt(P_prev(2,2));
    end

    for i = 2:length(t_vector)
        % =====Time Update=====
        xk_hat_pred = A*x_prev;
        Pk_pred = A*P_prev*A' + Q;

        % =====Measurement Update=====
        H = [1 0];
        z = [odom(i)]';

        K = Pk_pred*H'*pinv(H*Pk_pred*H'+R);

        % Update prev values
        x_prev = xk_hat_pred + K*(z - H*xk_hat_pred);
        P_prev = (eye(2,2) - K*H)*Pk_pred;

        % update position values
        x_pos(i) = x_prev(2);

        % for plotting stddev of position estimate's error
        if j == N
            stdev_xk_err(i) = sqrt(P_prev(2,2));
        end
    end

    x_pos_overall(:,j) = x_pos';

```

```

end

%% mean absolute error plot
abs_err = [];
mean_abs_err = [];
[m,n] = size(x_pos_overall);
for i = 1:n
    abs_err(:,i) = abs(x_vector' - x_pos_overall(:,i));
end

for i = 1:m
    mean_abs_err(i) = mean(abs_err(i,:));
end

plt1 = line(t_vector,mean_abs_err(:),'Color','r');
hold on
plt2 = line(t_vector,stddev_xk_err,'Color','b');
hold off
grid on
grid minor
legend([plt1 plt2], 'Mean absolute error', 'Std. Deviation (\sigma_{x_k}')
xlabel('time (seconds)');
ylabel('mean absolute error & std. deviation');

```

IV. Linear Kalman Filter “Incremental” SLAM With Landmarks Source Code

Filename: *driver_KF_2.m*

```

clc;
clear;
close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some parameters you can use to set up your solution of SimLab 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

length_hallway = 10;
sensor_range = 0.5;

landmarks = [2; 5; 8];

num_landmarks = length(landmarks);

stdev_odometry = 0.1; % m/s
stdev_range = 0.01; % m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Robot's "Ground Truth" Trajectory when it travels the hallway
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

delta_t = 0.1; % time is discretized with a time-step of 0.1 seconds

```



```

v_robot = 0.1; % robot travels at a constant speed of 0.1 m/s

t_terminal = length_hallway/v_robot; % time that we reach the end of hall

t_vector = 0:delta_t:t_terminal; % time vector for robot's trajectory
x_vector = 0:v_robot*delta_t:length_hallway; % robot's true position in time
v_vector = ones(1,length(t_vector)).*v_robot; % robot's true velocity in
time

num_states = length(x_vector); % number of discrete-time states in trajectory

N = 1000; % number of trials

```

```

%% Define A and initial states

```

```

A = [1 0 0 0 0;
     delta_t 1 0 0 0;
     0 0 1 0 0;
     0 0 0 1 0;
     0 0 0 0 1];

```

```

x0 = 0;
l1 = 0;
l2 = 0;
l3 = 0;
P0 = [1 0 0 0 0;
      0 0.0001 0 0 0;
      0 0 1 0 0;
      0 0 0 1 0;
      0 0 0 0 1];

```

```

Q = [10^(-8) 0 0 0 0;
     0 0 0 0 0;
     0 0 0 0 0;
     0 0 0 0 0;
     0 0 0 0 0];

```

```

% sensor noise covariance matrices

```

```

R1 = stdev_odometry^2;
R2 = [stdev_odometry^2 0;
      0 stdev_range^2];

```

```

x_pos_overall = []; % all x positions across all time steps per trial
stddev_xk_err = []; % std. dev. of position estimate's error across all
                    % time steps

```

```

for j = 1:N
    % x position via odometry measurements with noise
    odom = [0];
    for i = 2:length(t_vector)
        odom(i) = v_robot+stdev_odometry*randn();
    end

    % set initial values

```

```

x_prev = [v_robot x0 l1 l2 l3]';
P_prev= P0;

% all positions per trial
x_pos = [x_prev(2)];

% for plotting stddev of position estimate's error
if j== N
    stddev_xk_err(1) = sqrt(P_prev(2,2));
end

for i = 2:length(t_vector)

    % =====Time Update=====
    xk_hat_pred = A*x_prev;
    Pk_pred = A*P_prev*A' + Q;

    % =====Measurement Update=====
    R = R2;

    % detection of 1st landmark
    if t_vector(i) >= 15 && t_vector(i) <= 25
        H = [1 0 0 0 0;
              0 -1 1 0 0];
        z = [odom(i) v_robot*(20-t_vector(i))+stdev_range*randn()];

    % detection of 2nd landmark
    elseif t_vector(i) >= 45 && t_vector(i) <= 55
        H = [1 0 0 0 0;
              0 -1 0 1 0];
        z = [odom(i) v_robot*(50-t_vector(i))+stdev_range*randn()];

    % detection of 3rd landmark
    elseif t_vector(i) >= 75 && t_vector(i) <= 85
        H = [1 0 0 0 0;
              0 -1 0 0 1];
        z = [odom(i) v_robot*(80-t_vector(i))+stdev_range*randn()];

    % no landmarks
    else
        H = [1 0 0 0 0];
        R = R1;
        z = [odom(i)]';

    end

    K = Pk_pred*H'*pinv(H*Pk_pred*H'+R);

    % Update prev values
    x_prev = xk_hat_pred + K*(z - H*xk_hat_pred);
    P_prev = (eye(5,5) - K*H)*Pk_pred;

    % update position values
    x_pos(i) = x_prev(2);

```

```

        % for plotting stddev of position estimate's error
        if j == N
            stddev_xk_err(i) = sqrt(P_prev(2,2));
        end

    end

    x_pos_overall(:,j) = x_pos';

end

%% mean absolute error plot
abs_err = [];
mean_abs_err = [];
[m,n] = size(x_pos_overall);
for i = 1:n
    abs_err(:,i) = abs(x_vector' - x_pos_overall(:,i));
end

for i = 1:m
    mean_abs_err(i) = mean(abs_err(i,:));
end

plt1 = line(t_vector,mean_abs_err(:),'Color','r');
hold on
plt2 = line(t_vector,stddev_xk_err,'Color','b');
hold off
grid on
grid minor
legend([plt1 plt2], 'Mean absolute error', 'Std. Deviation (\sigma_{x_k}')
xlabel('time (seconds)');
ylabel('mean absolute error & std. deviation');

```

V. Linear Least Squares “Batch” SLAM With Landmarks & Loop-Closure Source Code

Filename: *driver_LS_3.m*

```

clc;
clear;
close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some parameters you can use to set up your solution of SimLab 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

length_hallway = 10;
sensor_range = 0.5;

landmarks = [2; 5; 8];

num_landmarks = length(landmarks);

```

```

stdev_odometry = 0.1; % m/s
stdev_range = 0.01; % m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Robot's "Ground Truth" Trajectory when it travels the hallway
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

delta_t = 0.1; % time is discretized with a time-step of 0.1 seconds
v_robot = 0.1; % robot travels at a constant speed of 0.1 m/s

t_terminal = length_hallway/v_robot; % time that we reach the end of hall

t_vector = 0:delta_t:t_terminal; % time vector for robot's trajectory
x_vector = 0:v_robot*delta_t:length_hallway; % robot's true position in time
v_vector = ones(1,length(t_vector)).*v_robot; % robot's true velocity in
time

num_states = length(x_vector); % number of discrete-time states in trajectory

%% LEAST SQUARES SOLUTION; t = 0:100 seconds

l_count = 3;
N = 1000; % number of trials
n = length(t_vector); % number of steps
x_pos_overall_100 = []; % records of all positions across n-time steps
x_pos_overall_200 = []; % records of all positions across n-time steps
% and across N trials

%% populate the constant matrix A for t=0:100secs
A = [];
A_1 = [];
for j = 1:n
    if isempty(A_1)
        A_1 = [1 zeros(1,n+l_count-1)];
    else
        A_1(j,:) = [zeros(1,j-2) -1 1 zeros(1,n+l_count-j)];
    end
end
A = A_1;

A_2 = [];
index = 1;
for j = 1:n
    % detection of first landmark
    if t_vector(j) >= 15 && t_vector(j) <= 25
        A_2(index,:) = [zeros(1,j-1) -1 zeros(1,n-j) 1 0 0];
        index = index + 1;
    end

    % detection of 2nd landmark
    if t_vector(j) >= 45 && t_vector(j) <= 55
        A_2(index,:) = [zeros(1,j-1) -1 zeros(1,n-j) 0 1 0];
        index = index + 1;
    end
end

```

```

    % detection of 3rd landmark
    if t_vector(j) >= 75 && t_vector(j) <= 85
        A_2(index,:) = [zeros(1,j-1) -1 zeros(1,n-j) 0 0 1];
        index = index + 1;
    end

end

A = [A ; A_2];
pinv_A = pinv(A); % get the pseudoinverse of A for the first half

%% populate A for t = 100.1:200secs

A = [A; -A_1(2:end,:)];
A = [A; -A_2];
pinv_A_2 = pinv(A); % get the pseudoinverse of A for the second half

%% populate b vector for t=0:100secs
% LS Loop
for i = 1:N
    x_odom = (v_robot+stdev_odometry*randn(1,n))*delta_t;
    b = [0 x_odom(2:end)];

    k = length(b)+1;
    for j = 1:n
        % detection of first landmark
        if t_vector(j) >= 15 && t_vector(j) <= 25
            b(k) = (v_robot*(20-t_vector(j))+stdev_range*randn());
            k = k+1;
        end

        % detection of 2nd landmark
        if t_vector(j) >= 45 && t_vector(j) <= 55
            b(k) = (v_robot*(50-t_vector(j))+stdev_range*randn());
            k = k+1;
        end

        % detection of 3rd landmark
        if t_vector(j) >= 75 && t_vector(j) <= 85
            b(k) = (v_robot*(80-t_vector(j))+stdev_range*randn());
            k = k+1;
        end
    end

    end
    X = pinv_A*b';
    x_pos_overall_100(:,i) = X;

%% populate b vector for t=100.1:200secs
x_odom = (-v_robot+stdev_odometry*randn(1,n-1))*delta_t;
b = [b x_odom];
k = length(b)+1;
for j = 1:n
    % detection of first landmark
    if t_vector(j) >= 15 && t_vector(j) <= 25

```

```

        b(k) = (-v_robot*(20-t_vector(j))+stdev_range*randn());
        k = k+1;
    end

    % detection of 2nd landmark
    if t_vector(j) >= 45 && t_vector(j) <= 55
        b(k) = (-v_robot*(50-t_vector(j))+stdev_range*randn());
        k = k+1;
    end

    % detection of 3rd landmark
    if t_vector(j) >= 75 && t_vector(j) <= 85
        b(k) = (-v_robot*(80-t_vector(j))+stdev_range*randn());
        k = k+1;
    end
end

X = pinv_A_2*b';
x_pos_overall_200(:,i) = X;
end

x_pos_overall_100 = x_pos_overall_100(1:end-3,:);
x_pos_overall_200 = x_pos_overall_200(1:end-3,:);

%% mean absolute error plot
abs_err_100 = [];
abs_err_200 = [];
mean_abs_err_100 = [];
mean_abs_err_200 = [];
[m,n] = size(x_pos_overall_100);
for i = 1:n
    abs_err_100(:,i) = abs(x_vector' - x_pos_overall_100(1:end,i));
    abs_err_200(:,i) = abs(x_vector' - x_pos_overall_200(1:end,i));
end

for i = 1:m
    mean_abs_err_100(i) = mean(abs_err_100(i,:));
    mean_abs_err_200(i) = mean(abs_err_200(i,:));
end

end

line([t_vector t_vector(end)+t_vector(2:end)], [mean_abs_err_100
mean_abs_err_200(end-1:-1:1)], 'Color', 'r');
grid on
grid minor
xlabel('time (seconds)');
ylabel('mean absolute error');

```