Solutions for Assignment 4

Luca Simonetto - 11413522
Fabrizio Ambrogi - 11403640
Probabilistic Robotics

October 9, 2017

Localization

The first task in this assignment is to visualize the uncertainty in the robot's localization after running the Extended Kalman Filter (EKF) on the provided data: to achieve this, the predicted mean of the robot's position at each timestep has been plotted, along with the respective covariance ellipse of confidence. As a 15% noise in measurements and 10° noise in bearings has to be assumed, the matrix Q of the EKF algorithm has been updated at each timestep from the fact that the range measurement depends on the actual readings.

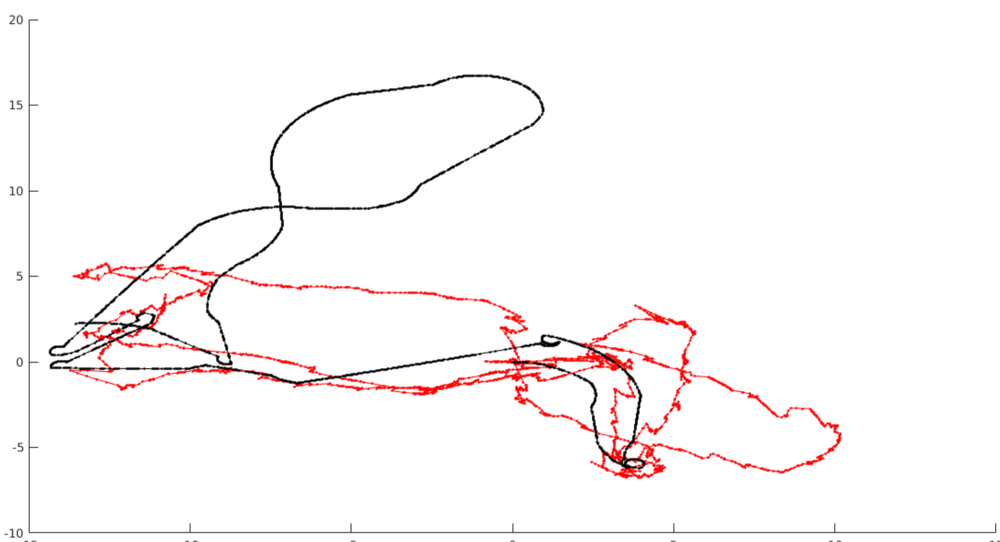

Figure 1: Robot predicted (red) and prior (black) belief on the location for the first circuit dataset (EKF)

Figure 1 shows a comparison between the predicted path of the robot before (black) and after (red) running

the EKF algorithm, indicating how the readings are updated as new measurements are analyzed. It can be seen how the underlying trajectory is corrected when the robot reaches the left side and fails to compute the correct orientation.
It has anyway to be noted how, while the readings are quite smooth, the estimate is jittery and presents spikes which are unlikely part of the actual movement of the robot.

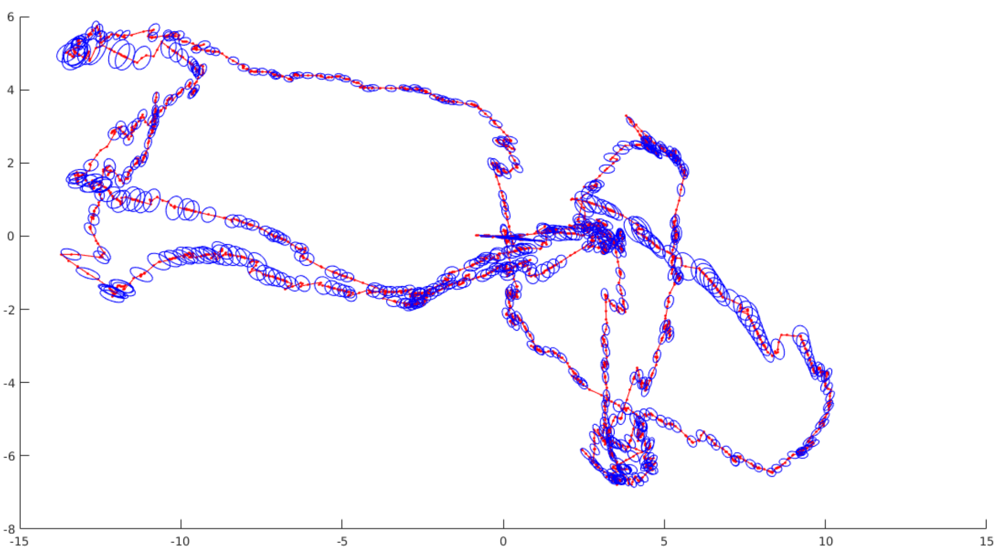

Figure 2: Robot's uncertainty on the first circuit dataset (EKF)

Figure 2 incorporates the estimates of the robot position at every timestep with uncertainty ellipses. It is possible to see how a smooth trajectory could easily be creating by taking points from the ellipses instead of just the center of the estimate.
The path is indicated in red, the red dots are the robot positions and the blue ellipses indicate the prediction covariance. The ellipses are drawn every 5 timesteps and cover 10% of the standard deviation of the predicted values, in order to improve readability..

Each dataset has been tested and the plots of the results can be found in the Appendix at the end of the report.

EKF-SLAM

Starting from the provided datasets, the Simultaneous Localization And Mapping (SLAM) algorithm has been implemented and applied, in order to allow plotting and reasoning on the results.

Implementation

In order to follow the provided pseudocode for the algorithm's implementation, the state of the world at each time step includes the landmarks position and so does the covariance matrix. This results in a state vector x with $3 + 2 \cdot L$ elements (where L is the total number of landmarks): the first three values indicate the robot's current position and bearing, and the remaining $2 \cdot L$ indicate the predicted x and y coordinates position for each landmark (set to 0 if still not encountered).

Similarly, the covariance Σ has been defined as a $3 + 2 \cdot L \times 3 + 2 \cdot L$ matrix, where each 2×2 block in the diagonal indicates the covariance regarding specific landmark coordinates.

In order to be able to better compare the results of the EKF algorithm, the provided implementation incorporates an odometry model for the pose update instead of a velocity model: this brought considerable improvements in the results as shown later in this section.

Another improvement made to the algorithm has been the inclusion of the innovation/validation calculations for the $\bar{\mu}$ and $\bar{\Sigma}$ updates, resulting in a much more stable trajectory prediction, allowing to easily find a value for the M matrix, as discussed later.

Initialization

During initialization, the position of each landmark has been set to [0,0] (being the center of the coordinates system), while the covariance values have been set to a big value to express the complete ignorance about their position. When a landmark is first encountered, it's position is set to the mean of the new estimate, and when a landmark exits the view range of the robot its position is not updated.

Matlab code

In this section, code snippets worth discussing are presented and analyzed. The full code can be found in the Appendix at the end of the report. The presented code contains both the odometry model and the innovation/validation calculations.

Initialization: the covariance Σ is initialized with all zeros apart from the diagonal starting from the fourth position, where all values are set to 10^9 to indicate the robot's ignorance regarding the landmarks positions. The mean of the estimated state of the world at each timestep is then set as the measured robot state followed by the estimated landmark x and y coordinates.

```
% ---- initialization
Sigma = zeros(3 + 2*NK, 3 + 2*NK, N);
Sigma(4:end, 4:end, 1) = eye(2*NK)*10^9;

mu = [xt; zeros(2 * NK,N)];

for i=1:NK
    mu(3 + i*2-1, 1) = mu(1, 1) + z(1, 1, i)*cos(z(2, 1, i) + mu(3, 1));
    mu(3 + i*2, 1) = mu(2, 1) + z(1, 1, i)*sin(z(2, 1, i) + mu(3, 1));
end
```

State prediction: the algorithm starts, iterating through every timestep in order to estimate the robot's position given the perfect movement. At each timestep, the matrix Fx indicated in Table 10.1 of the course book is calculated, along with the prediction of the current state $\hat{\mu}$.

```
% ---- state prediction

% odometry model
rot1 = u(1,t);
trans = u(2,t);
rot2 = u(3,t);

x = mu(1:3, t-1);

Fx = [eye(3), zeros(3, 2*NK)];

% odometry model prediction
mu_ = mu(:, t-1) + Fx' * [trans * cos(x(3)+rot1);...
```

```
trans * sin(x(3)+rot1);...
rot1 + rot2];
```

Uncertainty prediction: the algorithm then progresses with the estimation of the covariance, given by the operations applied to the previous estimate. The Scaling matrix M has been set to the identity matrix multiplied by 10^{-2} for reasons later explained. This can be interpreted as having uncertainty in the location at both timesteps, followed by a perfect movement between them. The last line of the snippet then calculates the predicted Σ matrix, by applying sandwich operation with the calculated Jacobians.

```
% ---- uncertainty prediction

% Jacobian with respect to robot location
G = eye(2*NK + 3) + Fx' * [...
0, 0, -trans * sin(x(3)+rot1);...
0, 0, trans * cos(x(3)+rot1);...
0, 0, 0] * Fx;

Sigma_ = G * Sigma(:,:,t-1) * G';

% Jacobian with respect to control
M = eye(3) * 10^-2;

V = [-trans*cos(mu_(3)+rot1), cos(mu_(3)+rot1), 0;...
trans*sin(mu_(3)+rot1),  sin(mu_(3)+rot1), 0;...
1,                0,             1];

R = V'*M*V;

Sigma_ = Sigma_ + Fx' * R * Fx;
```

Correction: for each measured landmark a correction to the prediction is applied: first, if the landmark has never been seen, it's position is determined from the current robot's position and the current observation of the landmark, then the Q matrix is calculated containing the noise values to be assumed in the calculations. The remaining matrices are calculated, including the precision matrix S to be used for the Kalman gain calculation, the innovation ν and validation ρ. Having determined all the necessary matrices, the predicted $\hat{\mu}$ and $\hat{\Sigma}$ are updated with the new values, if ρ is < 2.

```
% ---- correction
for landmark = 1:size(z,3)
  if z(1, t, landmark) ≠ 0
    % if landmark has never been measured
    if mu_(3 +2*(landmark-1) + 1) == 0 && mu_(3 +2*(landmark-1) + 1) == 0
      mu_(3 +2*(landmark-1) + 1) = mu_(1) + z(1, t, landmark)*cos(z(2, t, landmark) + mu_(3));
      mu_(3 +2*(landmark-1) + 2) = mu_(2) + z(1, t, landmark)*sin(z(2, t, landmark) + mu_(3));
    end

    % noise in readings/angle
    Q = diag([.15*z(1, t, landmark), .10]+10^-9);

    d = [mu_(3 +2*(landmark-1) + 1) - mu_(1); mu_(3 +2*(landmark-1) + 2) - mu_(2)];
    q = d'*d + 10^-9;

    z_ = [sqrt(q); atan2(d(2), d(1)) - mu_(3)];

    Fxj = createF(landmark, NK);

    H = 1/q * [-sqrt(q)*d(1), -sqrt(q) * d(2), 0, sqrt(q)*d(1), sqrt(q) * d(2);
    d(2), -d(1), -q, -d(2), d(1)] * Fxj;

    % precision matrix
    S = H * Sigma_ * H' + Q;

```

```
    % Kalman gain
    K = Sigma_ * H' / S;

    % innovation
    nu = z(:,t,landmark) - z_;

    % validation gate
    ro = nu'/S*nu;

    if ro < 2
      %updated mean and covariance
      mu_ = mu_ + K*nu;
      Sigma_ = (eye(size(mu_, 1))-K*H)*Sigma_;
    end
  end
end
```

Final mu and sigma: when every landmark has been analyzed, the corrected mean and covariance are saved as the final values for that timestep.

```
% ---- final mu and sigma
mu(:,t) = mu_;
Sigma(:,:,t) = Sigma_;
```

M matrix

The provided datasets assume implicit noise that cannot be modeled from the algorithm using pre-defined values, as noisy recordings and measurements are the only provided data. Instead, to get an output that is as close as possible with the correct real world values, a tuning phase of the M matrix has been done: this allows to account for the dataset noise, and paired with the fact that the robot had to travel through specific locations, resulted in a final trajectory that is close to ground truth.

The path followed by the robot required it to pass through three or five locations (depending on the dataset), in which an external stimuli has been given.

Figure 3 shows the position of each marker, over which the robot should have traveled: this allows to take the output of the EKF-SLAM algorithm and fine-tune the internal parameters. The leftmost two markers have been reached only in the full dataset.

[Note: the fine-tuning phase used the full dataset, as the former trajectories are not long enough to pass through all the markers and show the complete "8" shape]

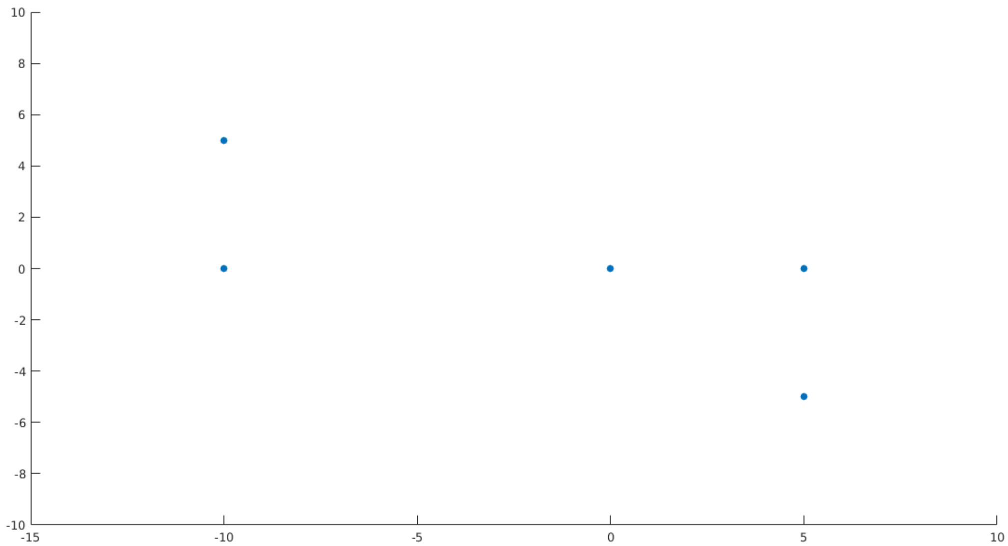

Figure 3: Positions of the markers over which the robot trajectory should lie.

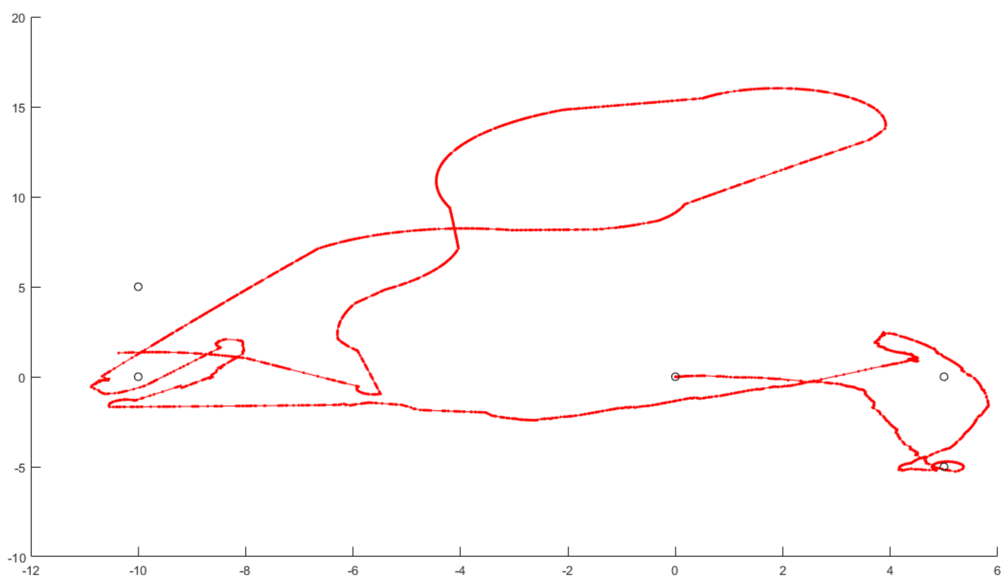

Figure 4: Output of the EKF-SLAM algorithm on the first circuit dataset with a value of 10^{-4} for M.

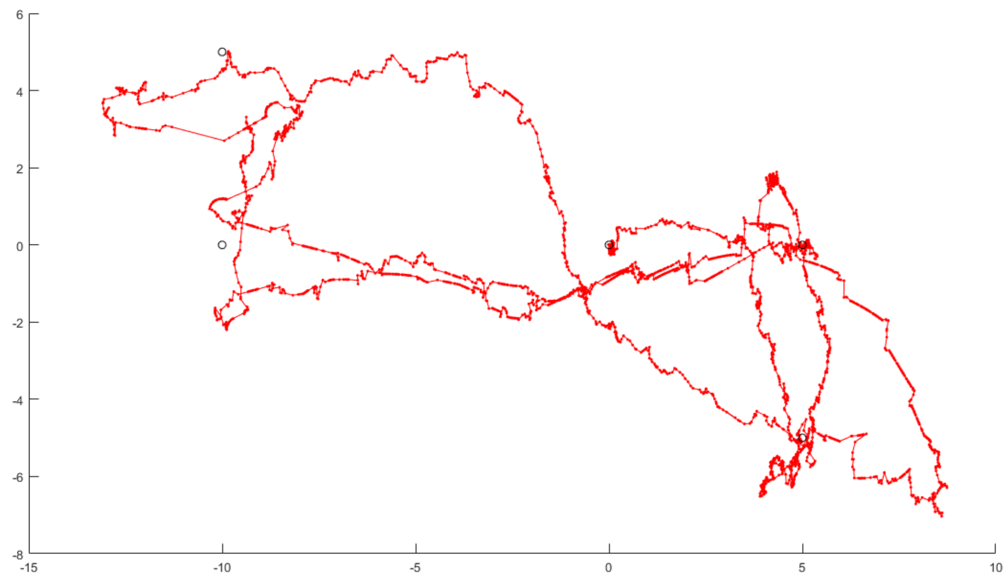

Figure 5: Output of the EKF-SLAM algorithm on the first circuit dataset with a value of 10^{-2} for M.

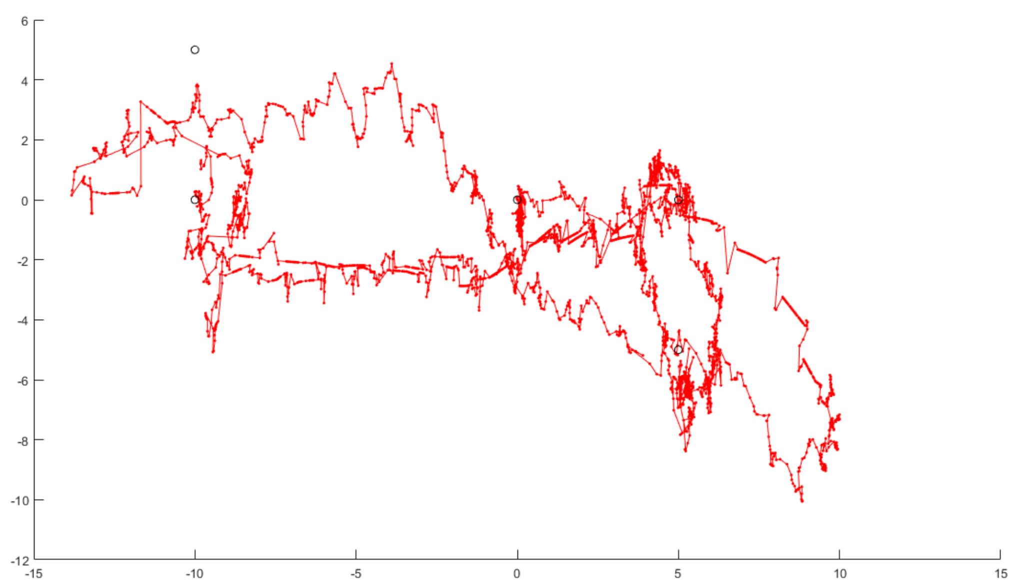

Figure 6: Output of the EKF-SLAM algorithm on the first circuit dataset with a value of 10^{-1} for M.

Figure 4, 5 and 6 show the different results that are obtained by using different values for the M matrix.
The first picture indicates the results when using a value of 10^{-4}, which gives a nice and smooth trajectory, that however deranges outside of the filed.
The second one shows the improvements made when fine-tuning the value to 10^{-2}. Even though the trajectory becomes less clean we can clearly see how it follows the correct commands, to create an "8" with the corners of before.
The third shows how going with an even higher movement error matrix doesn't give any improvements and actually creates an ugly, jittering trajectory that even misses one of the corners.
It can be seen that setting a correct value for M results in a way better path.

Odometry vs Velocity Model

A big problem that was encountered in the implementation of the pseudocode from the book comes from the movement model. Specifically, in the given algorithm the velocity model was used.
This model gave more than decent results in estimating the trajectory with the right M (Figure 7), arguably even better than with the odometry model. However the covariance matrices calculated were incredibly unstable and often resulted non positive definite.

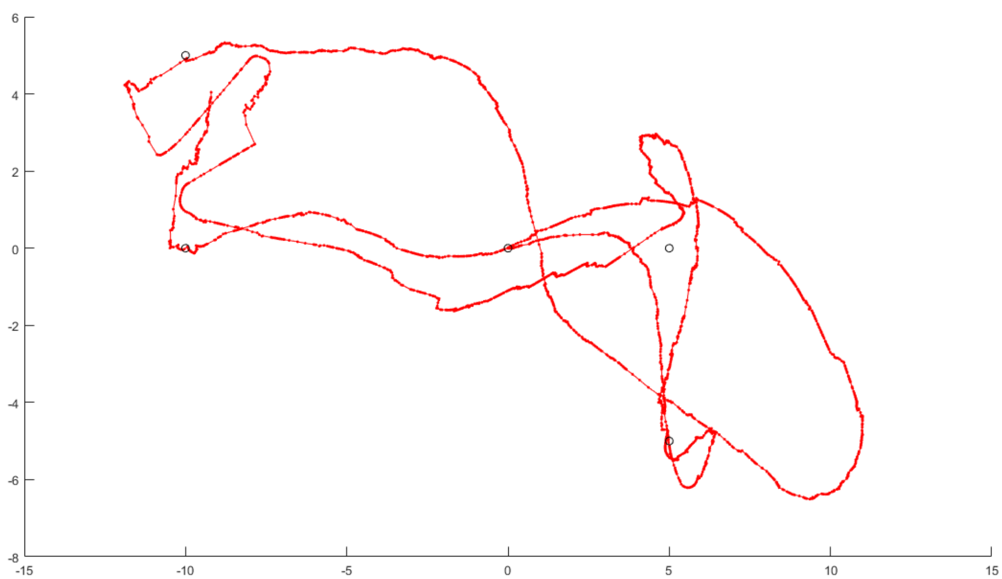

Figure 7: Full trajectory with velocity model and a value of 10^{-3} for M.

Switching to the odometry model the results where way more consistent and less unstable, giving a more correct looking output.

Final results

Having discussed the inclusions made to the implementation, outputs of each datasets can be presented:

Figure 8 shows the difference between the prior belief on the position and the predicted one after running the EKF-SLAM algorithm. The resulting trajectory corrects the predicted one and the tuning of the M matrix allows it to pass through the markers as required by the problem definition.

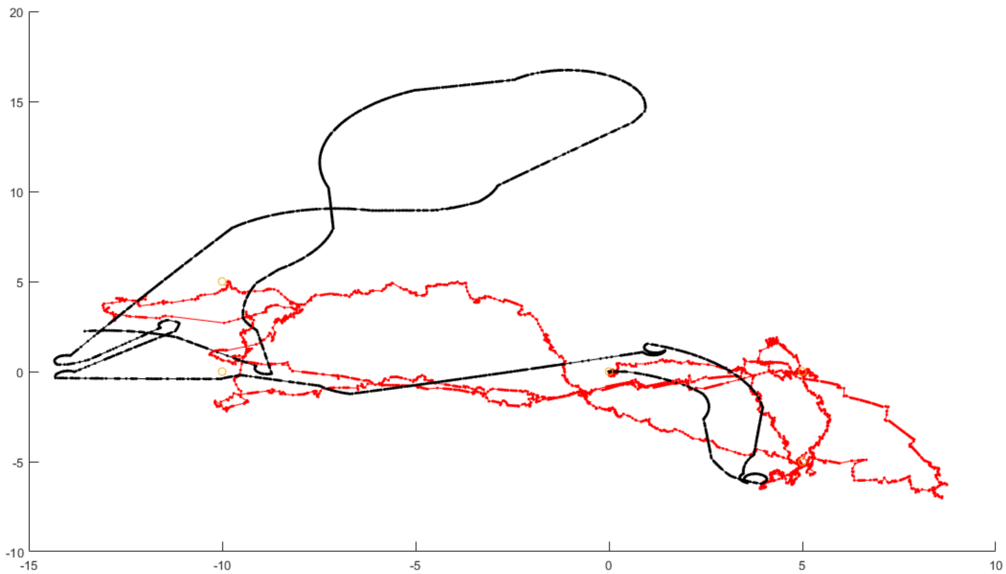

Figure 8: Robot predicted (red) and prior (black) belief on the location for the first circuit dataset (EKF-SLAM)

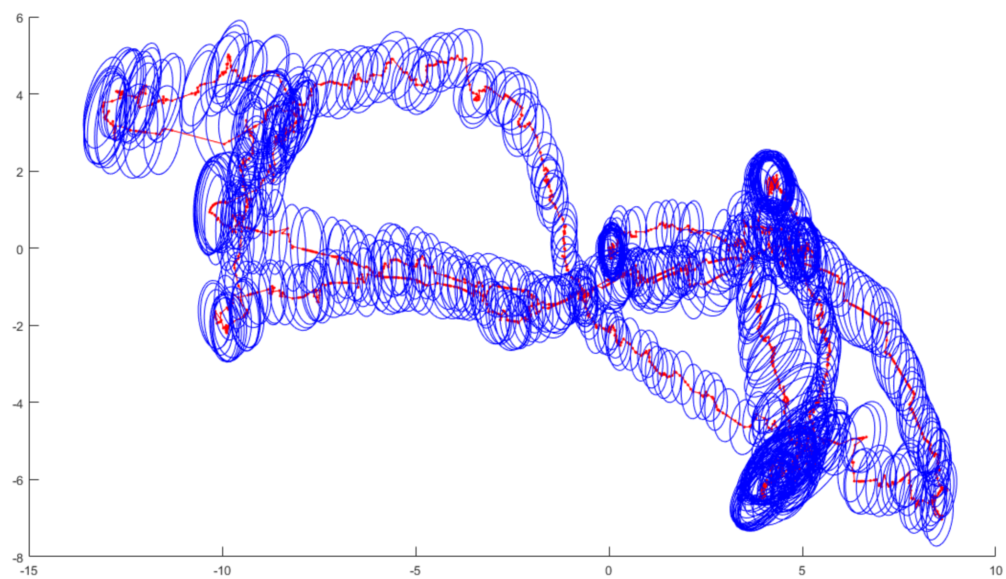

Figure 9: Robot's position and uncertainty on the first circuit dataset (EKF-SLAM)

Figure 9 represents the position estimates, surrounded by their covariance matrix. The covariance grows with movement and shrinks when landmarks enter the sight of view, confirming the coordinates.

In figure 10 shows the estimated map of the field, with the 6 landmarks computed position and confidence, compared with their real position.

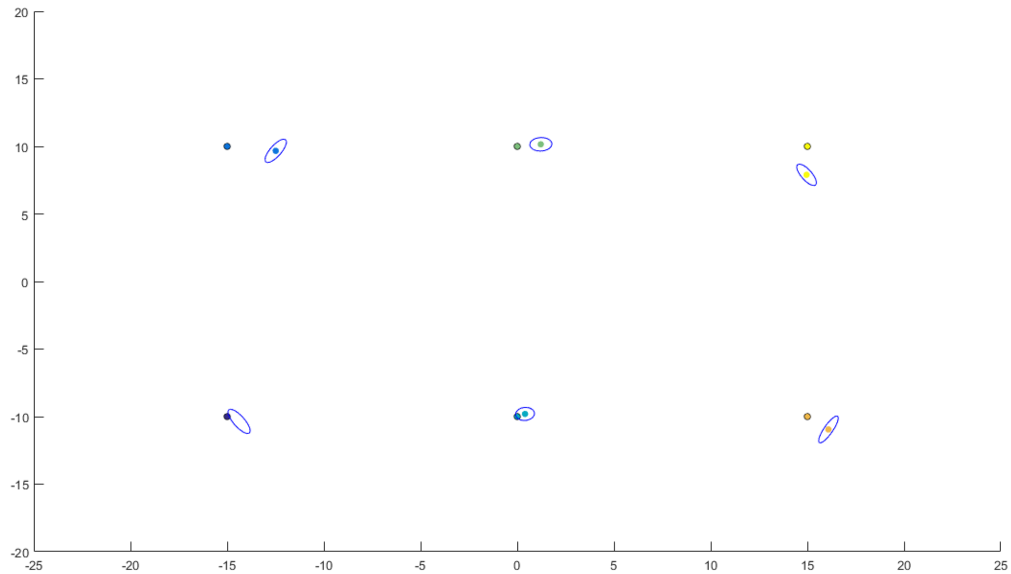

Figure 10: Landmarks real positions and the ones estimated by the EKF-SLAM algorithm after a full circuit

The outcome is surprisingly close and this truly demonstrates the power of EKF-SLAM, capable of correcting the movement measurements and create a map of the landmarks at the same time.

Possible Research and Improvement

Since the real position of the landmarks is known, an interesting test could be to center there their Prior and set its variance to different finite values, to simulate different degrees of certainty. This instead of the flat normal with mean 0 and (close to) infinite variance, which act as an uninformative Prior.

In this way it can be tested how previous explorations of a zone could be used as a base for future ones and to which degree this will influence them.

Appendix

Plots of the execution of EKF on the remaining datasets

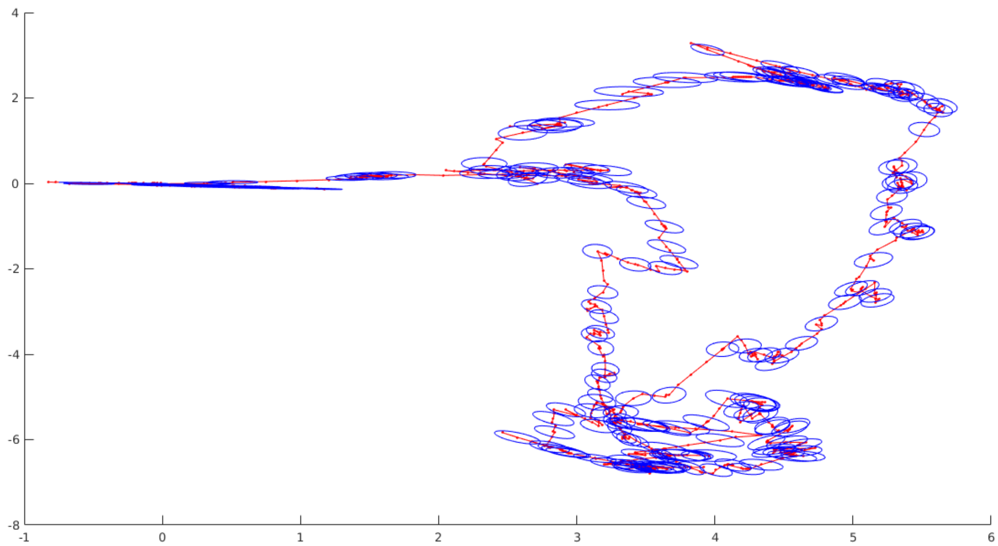

Figure 11: Robot's uncertainty on the second dataset

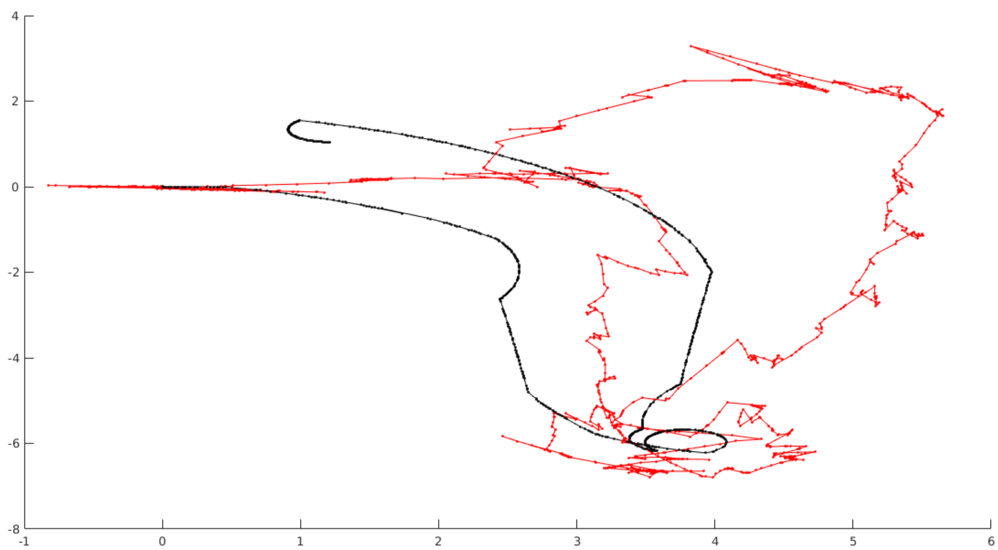

Figure 12: Robot predicted (red) and prior (black) prior belief on the location for the second dataset (EKF)

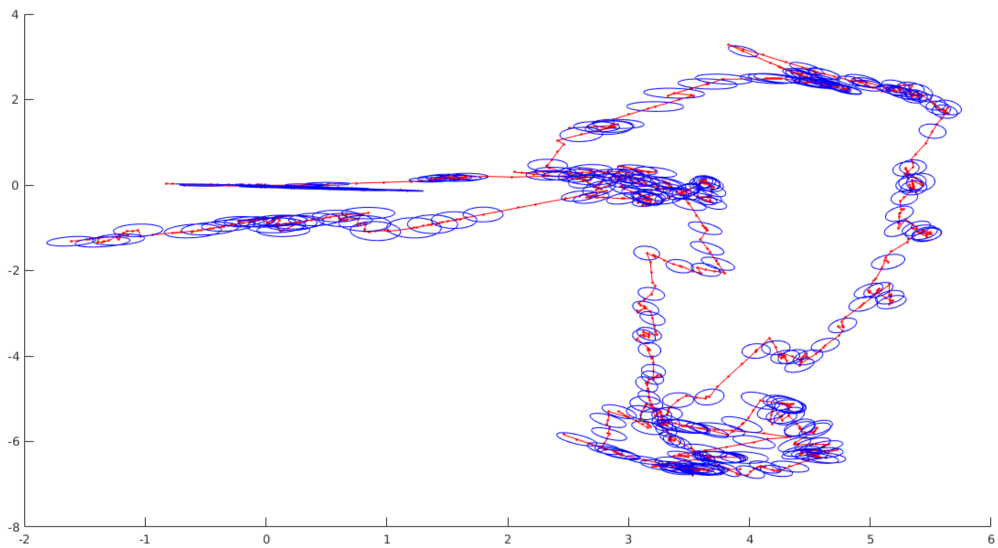

Figure 13: Robot's uncertainty on the third dataset (EKF)

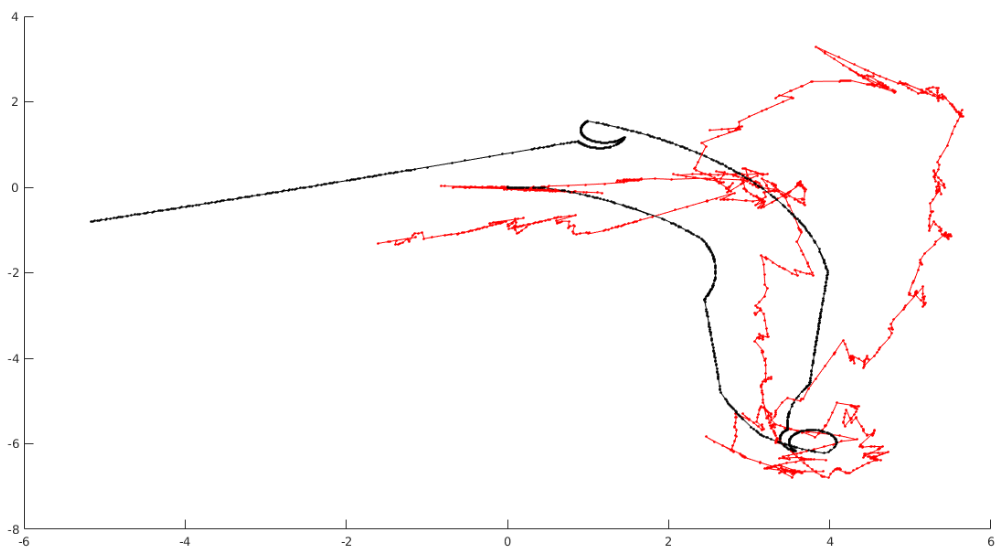

Figure 14: Robot predicted (red) and prior (black) prior belief on the location for the third dataset (EKF)

Plots of the execution of EKF-SLAM on the remaining datasets

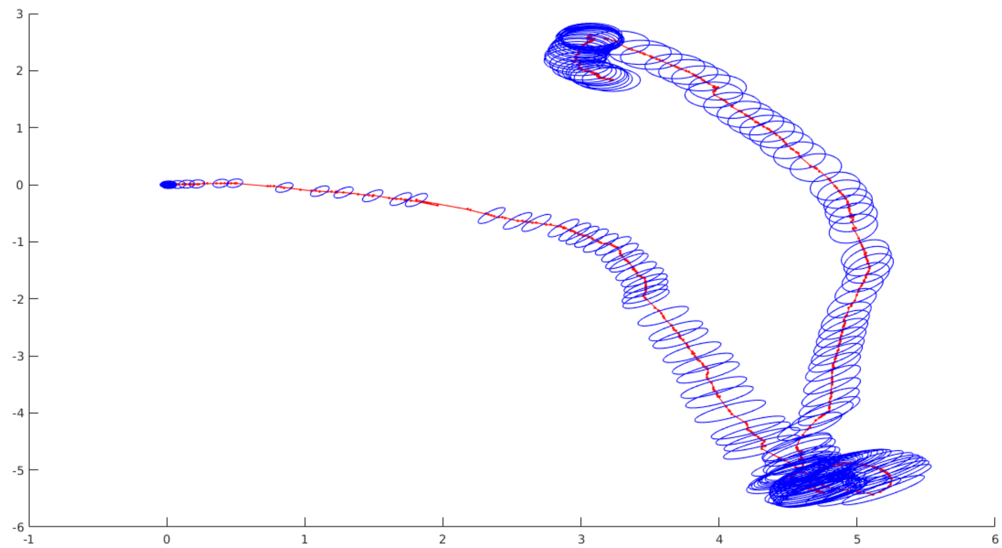

Figure 15: Robot's uncertainty on the second dataset (EKF-SLAM)

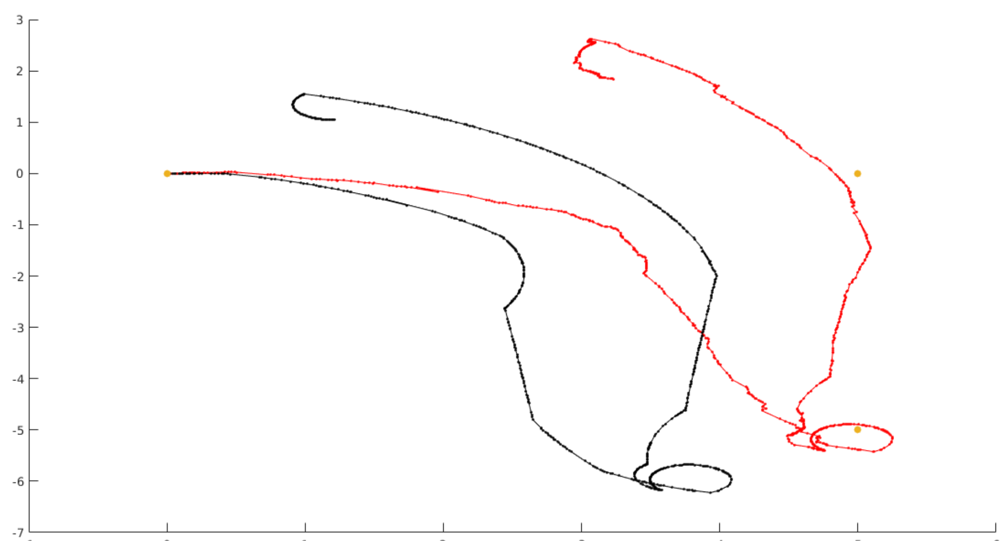

Figure 16: Robot predicted (red) and prior (black) prior belief on the location for the full dataset (EKF-SLAM)

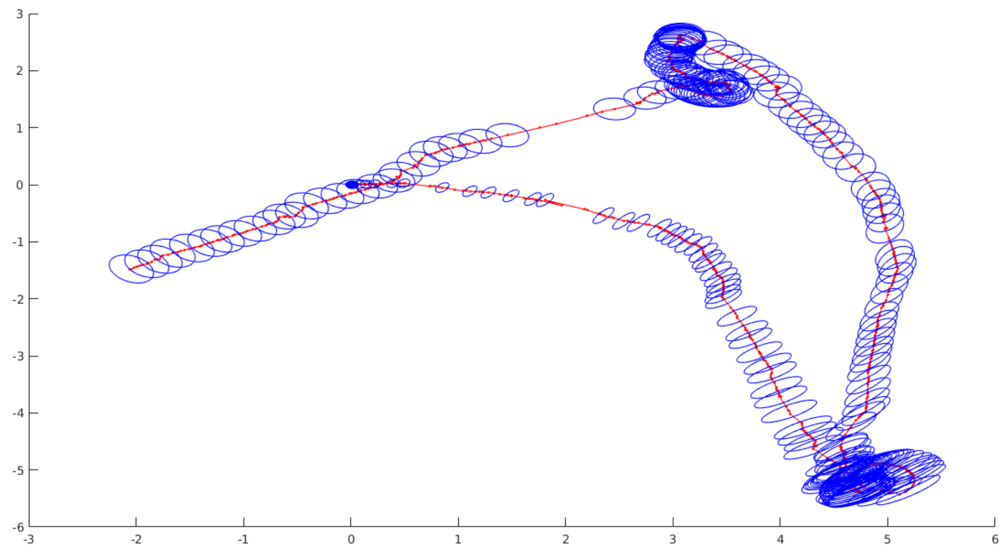

Figure 17: Robot's uncertainty on the third dataset (EKF-SLAM)

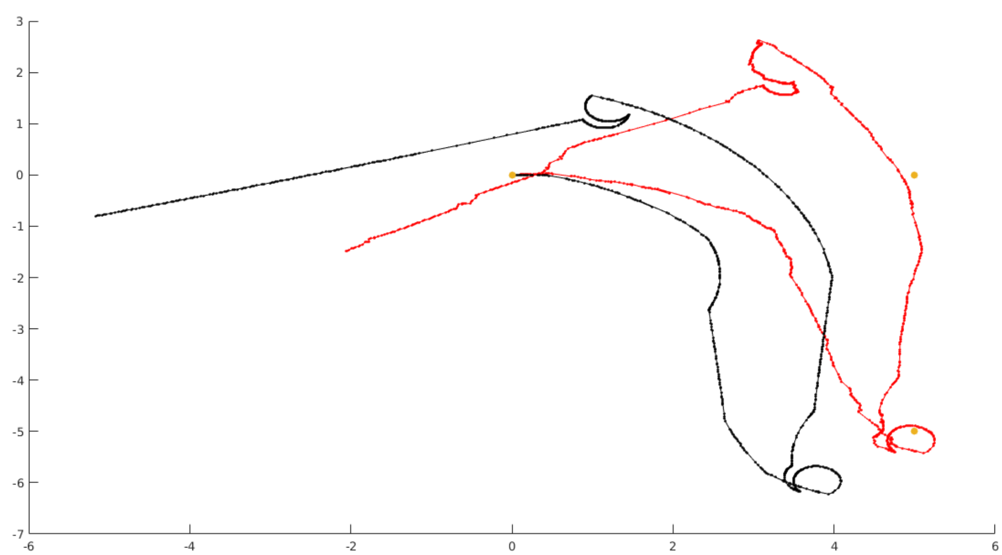

Figure 18: Robot predicted (red) and prior (black) prior belief on the location for the third dataset (EKF-SLAM)

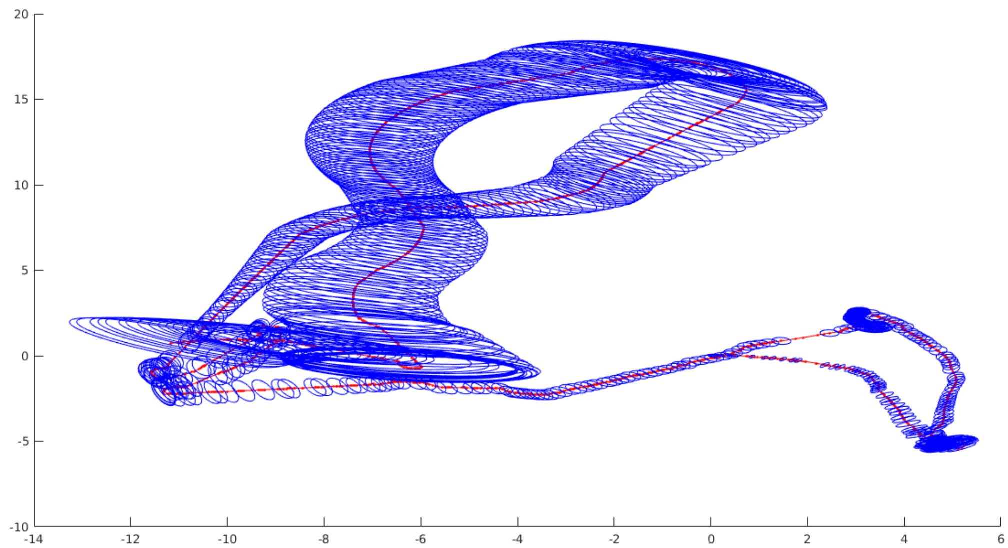

Figure 19: Robot's uncertainty on the full dataset (EKF-SLAM)

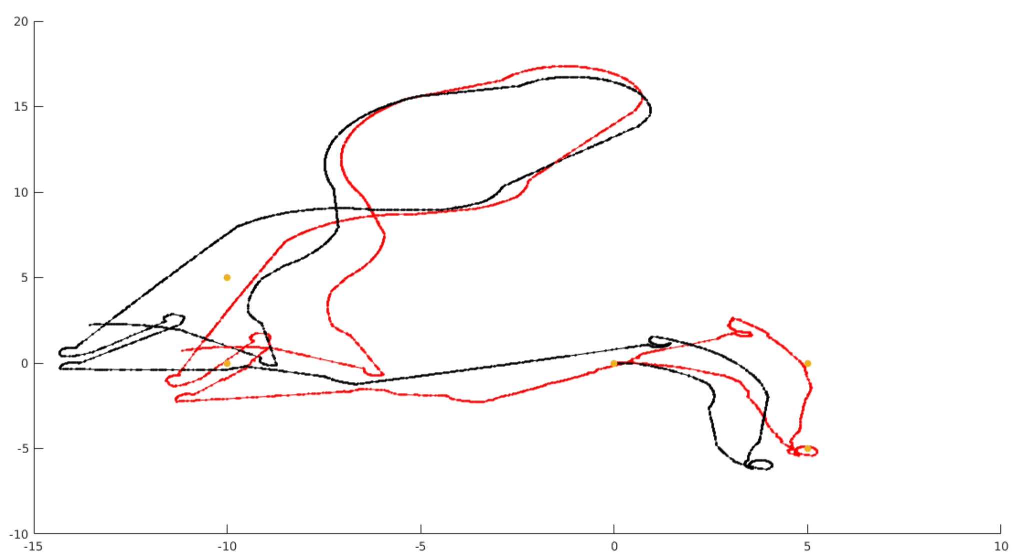

Figure 20: Robot predicted (red) and prior (black) prior belief on the location for the full dataset (EKF-SLAM)

EKF SLAM implementation

```
clear;
close all;

% N is number of observations in dlog.dat

% logfilename = 'dlog_firstmark.dat'; N = 758;
% logfilename = 'dlog_secondmark.dat'; N = 1159;
% logfilename = 'dlog_thirdmark.dat'; N = 1434;
 logfilename = 'dlog.dat'; N = 3500;

% ------ data creation
% expected user input noise
u_err = .15;
M = u_err*eye(2);

% true robot position at t = 1
xt(:,1) = [0 0 0]'; dim = 3;  % x = [x y angle]'

% user input at t = 1
%u(:,1) = [0 0]';                % u = [speed Δ_angle]'
u(:,1) = [0 0 0]';       % = [rot1 trans rot2]'

% Landmark locations
L2006 = [20  20 -20 -20;...
     20 -20  20 -20];

% You also need the following information about the landmark positions:
% cyan:magenta -1500 -1000 magenta:cyan -1500 1000 magenta:green 0 -1000 green:magenta 0 ... 1000 yellow:magenta 1500 -1000 magenta:yellow 1500 1000
% 0 -> green 1 -> magenta 2 -> yellow 3 -> blue
L = [-15 -15 0 0 15 15;-10 10 -10 10 -10 10];
LID = [3 1 1 0 2 1;1 3 0 1 1 2];
U = M;          % user input noise (set to be equal to expected input noise)

angle = 0;

logfile = true;

if ¬logfile

    for t=2:N

        % fabricate user input
        u(2,t) = randn;
        if abs(u(2,t)) > 0.4 %  P(steering) = 0.4
            u(2,t) = 0;
        end
        u(1,t) = .5*(1 - u(2,t)/0.4); % high Δ_angle --> low speed

        % create noisy user input
        un = U*randn(2,1) +u(:,t);

        % calculate true robot position t+1
        xt(:,t) =  [xt(1,t-1)+ un(1)*cos(xt(3,t-1)) ; ...
            xt(2,t-1)+ un(1)*sin(xt(3,t-1)) ; ...
            xt(3,t-1)+ un(2)];

    end

    %------------------------------------------------------------- measurements
    %
    perc = .7; % percentage of Landmark measurement loss
    t = 1;
    for i=1:N
```

```
        for landmark=1:size(L,2)
            if rand > perc
                % z = [distance angle]'
                z(:,t,landmark) = [ sqrt((L(1,landmark)-xt(1,t))^2 + ...
                    (L(2,landmark)-xt(2,t))^2)+randn*m_err;...
                    atan2(L(2,landmark)-xt(2,t),L(1,landmark)-xt(1,t)) - xt(3,t)+randn*m_err];
            else
                z(:,t,landmark) = [0;0];
            end
        end
        t = t + 1;
    end

else % logfile

    fid = fopen(logfilename,'r');
    t = 0;
    for i=1:N
        tline = fgetl(fid);
        [type,success] = sscanf(tline, '%s', 1);
        if strcmp(type,'mark')
            fprintf(1,'*')
            continue
        end
        t = t + 1;
        [xt(:,t),success] = sscanf(tline, 'obs: %*d %f %f %f', 3);
        xt(1,t)=xt(1,t)/100; % milimeters to decimeters
        xt(2,t)=xt(2,t)/100;
        xt(3,t)=xt(3,t)*pi/180; % degrees to radians
        if t > 1
            dx = xt(1,t)-xt(1,t-1);
            dy = xt(2,t)-xt(2,t-1);

%             u(2,t) = xt(3, t)- xt(3, t-1); % diff_angle
%             u(1,t) = sqrt(dx*dx + dy*dy); % speed
            u(1,t) = atan2(dy, dx) - xt(3,t-1);
            u(2,t) = sqrt(dx*dx + dy*dy);
            u(3,t) = xt(3,t) - xt(3,t-1) - u(1,t);
        end
        for landmark=1:6
            z(:,t,landmark) = [0;0];
        end

        [obs_landmarks, success,errmsg,nextindex] = sscanf(tline, 'obs: %*d %*f %*f %*f ...
            %d', 1);
        for observation=1:obs_landmarks
            tline=tline(1,nextindex:size(tline,2));
            [signature, success] = sscanf(tline, ' ( %d:%d', 2);
            for landmark = 1:6
                if signature(1) == LID(1,landmark) && signature(2) == LID(2,landmark)
                    [z(:,t,landmark),success,errmsg,nextindex] = sscanf(tline, ' ( %*d:%*d ...
                        %f %f )', 2);
                    z(1,t,landmark) = z(1,t,landmark) / 100; % milimeters to decimeters
                    z(2,t,landmark) = z(2,t,landmark) * pi / 180; % degrees to radians
                end
            end % for landmarks
        end % for observations
    end % for t=1:N
    fclose(fid);
end % if logfile

N = t;
NK = 6; % number of landmarks

% ---------
% EKF SLAM
% ---------

```

```
% ----- initialization
Sigma = zeros(3 + 2*NK, 3 + 2*NK, N);
Sigma(4:end, 4:end, 1) = eye(2*NK)*10^9;

mu = [xt; zeros(2 * NK,N)];

for i=1:NK
    mu(3 + i*2-1, 1) = mu(1, 1) + z(1, 1, i)*cos(z(2, 1, i) + mu(3, 1));
    mu(3 + i*2, 1) = mu(2, 1) + z(1, 1, i)*sin(z(2, 1, i) + mu(3, 1));
end

for t = 2:N
    % ----- state prediction

    % old velocity model
    %get user input
    %v = u(1,t); % velocity
    %omega = u(2,t) + 10^-10;   % Δ angle

    % odometry model
    rot1 = u(1,t);
    trans = u(2,t);
    rot2 = u(3,t);

    x = mu(1:3, t-1);

    Fx = [eye(3), zeros(3, 2*NK)];

    % old velocity model prediction
%     mu_ = mu(:, t-1) + Fx' * [-v/omega * sin(x(3)) + v/omega * sin(x(3)+omega);...
%                               v/omega * cos(x(3)) - v/omega * cos(x(3)+omega);...
%                               omega];

    % odometry model prediction
     mu_ = mu(:, t-1) + Fx' * [trans * cos(x(3)+rot1);...
                             trans * sin(x(3)+rot1);...
                             rot1 + rot2];

    % ----- uncertainty prediction

    % Jacobian with respect to robot location
    G = eye(2*NK + 3) + Fx' * [...
        0, 0, -trans * sin(x(3)+rot1);...
        0, 0, trans * cos(x(3)+rot1);...
        0, 0, 0] * Fx;

    Sigma_ = G * Sigma(:,:,t-1) * G';

    % Jacobian with respect to control
    M = eye(3) * 10^-2;
%     M = eye(3) * 10^-9;

    V = [-trans*cos(mu_(3)+rot1), cos(mu_(3)+rot1), 0;...
        trans*sin(mu_(3)+rot1),  sin(mu_(3)+rot1), 0;...
        1,                0,              1];

    R = V'*M*V;

    Sigma_ = Sigma_ + Fx' * R * Fx;

    % ----- correction
    for landmark = 1:size(z,3)
        if z(1, t, landmark) ≠ 0
            % if landmark has never been measured
            if mu_(3 +2*(landmark-1) + 1) == 0 && mu_(3 +2*(landmark-1) + 1) == 0
                mu_(3 +2*(landmark-1) + 1) = mu_(1) + z(1, t, landmark)*cos(z(2, t, ...
                    landmark) + mu_(3));
```

```
                mu_(3 +2*(landmark-1) + 2) = mu_(2) + z(1, t, landmark)*sin(z(2, t, ...
                    landmark) + mu_(3));
            end

            % noise in readings/angle
            Q = diag([.15*z(1, t, landmark), .10]+10^-9);

            d = [mu_(3 +2*(landmark-1) + 1) - mu_(1); mu_(3 +2*(landmark-1) + 2) - mu_(2)];
            q = d'*d + 10^-9;

            z_ = [sqrt(q); atan2(d(2), d(1)) - mu_(3)];

            Fxj = createF(landmark, NK);

            H = 1/q * [-sqrt(q)*d(1), -sqrt(q) * d(2), 0, sqrt(q)*d(1), sqrt(q) * d(2);
                d(2), -d(1), -q, -d(2), d(1)] * Fxj;

            % precision matrix
            S = H * Sigma_ * H' + Q;

            % Kalman gain
            K = Sigma_ * H' / S;

            % innovation
            nu = z(:,t,landmark) - z_;

            % validation gate
            ro = nu'/S*nu;

            if ro < 2
                %updated mean and covariance
                mu_ = mu_ + K*nu;
                Sigma_ = (eye(size(mu_, 1))-K*H)*Sigma_;
            end

            % old update
%             mu_ = mu_ + K * (z(:,t,landmark) - z_);
%             Sigma_ = (eye(2*NK+3) - K*H)*Sigma_;
        end
    end

    % ---- final mu and sigma
    mu(:,t) = mu_;
    Sigma(:,:,t) = Sigma_;
end

markers = [-10, -10, 0, 5, 5; 0, 5, 0, 0, -5];

% % ------plot trajectory and markers
hold on;
% scatter(L(1,:),L(2,:), 10, 'b');
plot(mu(1, :), mu(2, :), 'r')
hold on
scatter(mu(1, :), mu(2, :), 5, 'r', 'filled');
% xlim([-15, 15]);
% ylim([-10, 10]);
scatter(markers(1, 3:end), markers(2, 3:end), 'blue', 'filled');

% % ------plot mu vs xt
hold on;
% scatter(L(1,:),L(2,:), 10, 'b');
plot(mu(1, :), mu(2, :), 'r')
plot(xt(1, :), xt(2, :), 'k')
hold on
scatter(mu(1, :), mu(2, :), 5, 'r', 'filled');
scatter(xt(1, :), xt(2, :), 5, 'k', 'filled');
% xlim([-15, 15]);
% ylim([-10, 10]);
```

```
scatter(markers(1, 3:end), markers(2, 3:end), 'filled');

% ----- plot of the markers positions
scatter(markers(1, :), markers(2, :), 'filled');
xlim([-15, 10]);
ylim([-10, 10]);

% -------plot robot path with covariances
figure();
hold on;
% scatter(L(1,:),L(2,:), 10, 'b');
plot(mu(1, :), mu(2, :), 'r')
scatter(mu(1, :), mu(2, :), 5, 'r', 'filled');
for i=1:5:size(mu, 2)
    h = plot_gaussian_ellipsoid(mu(1:2, i), Sigma(1:2, 1:2, i), 1);
    set(h,'color','b');
end


% -------dynamical plot of the predicted landmarks positions
% figure();
% for i = 1:10:N
%     clf
%     hold on;
%     scatter(mu(1, 1:i), mu(2, 1:i), 10, 'filled', 'black');
%     for j=1:NK
%         scatter(mu(3+j*2-1, i), mu(3+j*2, i), 25, j, 'filled');
%         scatter(L(1,j),L(2,j), 25, j, 'filled', 'MarkerEdgeColor', 'black');
%         h = plot_gaussian_ellipsoid(mu(3+j*2-1:3+j*2, i), Sigma(3+j*2-1:3+j*2, ...
    3+j*2-1:3+j*2, i));
%         set(h,'color','b');
%     end
%     xlim([-25, 25]);
%     ylim([-20, 20]);
%     drawnow
%     pause(0.01)
% end

% ------- plot of the final predicted landmarks positions
% figure();
% for j=1:NK
%     scatter(mu(3+j*2-1, end), mu(3+j*2, end), 25, j, 'filled');
%     scatter(L(1,j),L(2,j), 25, j, 'filled', 'MarkerEdgeColor', 'black');
%     h = plot_gaussian_ellipsoid(mu(3+j*2-1:3+j*2, end), Sigma(3+j*2-1:3+j*2, ...
    3+j*2-1:3+j*2, end));
%     set(h,'color','b');
% end
% xlim([-25, 25]);
% ylim([-20, 20]);

function F = createF(j, N)
    F = zeros(5, 2*N + 3);
    F(1,1) = 1;
    F(2,2) = 1;
    F(3,3) = 1;

    F(4,(2*j)+2) = 1;
    F(5,(2*j)+3) = 1;
end
```