Probabilistic Robotics - Assignment 4

Nedko Savov - 11404345
Athanasios Roidis - 11413441

October 10, 2017

1 Task 1

Before we start, a code problem with the code in EKF.m is addressed. In case of a "mark" tag in the log file, the code adds a state entry $(x, y, \theta) = (0, 0, 0)$ to the list of states. This effectively resets the pose of the robot after the mark and prevents from continuous transition between two mark datasets. This is why we altered the code not to include this entry.

During the prediction stage, the task is to model the current position of the robot, after performing a movement. Since no planning for future movement is required, an odometry model can be used. An odometry model is practically proven to be more accurate than a velocity model in the general case. Therefore some modifications had to be made to the given code.

First and foremost, we had to change the way the data was loaded so that the control vector u had the following form:

$$u = \begin{bmatrix} \delta_{trans} \\ \delta_{rot1} \\ \delta_{rot2} \end{bmatrix} = \begin{bmatrix} \sqrt{d_x^2 + d_y^2} \\ atan2(d_y, d_x) - \mu_{t-1,\theta} \\ \mu_{t,\theta} - \mu_{t-1,\theta} - \delta_{rot1} \end{bmatrix}, \qquad d = \begin{bmatrix} \mu_{t,x} - \mu_{t-1,x} \\ \mu_{t,y} - \mu_{t-1,y} \end{bmatrix}$$

when the robot is moving from $\mu_{t-1} = (\mu_{t-1,x}, \mu_{t-1,y}, \mu_{t-1,\theta})$ to $\mu_t = (\mu_{t,x}, \mu_{t,y}, \mu_{t,\theta})$.

The new motion function is:

$$\bar{\mu_t} = \mu_{t-1} + \begin{bmatrix} \delta_{trans} \cos(\delta_{rot1} + \mu_{t-1,\theta}) \\ \delta_{trans} \sin(\delta_{rot1} + \mu_{t-1,\theta}) \\ \delta_{rot1} + \delta_{rot2} \end{bmatrix}$$

The Jacobian of the motion function with respect to the robot pose is then given by:

$$G = \begin{bmatrix} 1 & 0 & -\delta_{trans} * sin(\delta_{rot1} + \mu_{t,\theta}) \\ 0 & 1 & \delta_{trans} * cos(\delta_{rot1} + \mu_{t,\theta}) \\ 0 & 0 & 1 \end{bmatrix};$$

The Jacobian of the motion function with respect to the motion is given by:

$$V = \begin{bmatrix} \cos(\delta_{rot1} + x_(3)) & -\delta_{trans} \sin(\delta_{rot1} + x_(3)) & 0 \\ \delta_{trans} * \sin(\delta_{rot1} + \mu_{t,\theta}) & \delta_{trans} * \cos(\delta_{rot1} + \mu_{t,\theta} & 0 \\ 0 & 1 & 1 \end{bmatrix};$$

Since the correction step is not related to the motion model, it was left untouched, as it was given.

Finally, the Q (measurement noise covariance) and M (control vector noise covariance) matrices should also be defined according to the assignment. As stated, there is a noise of 15% on range measurements and 10^o (to be converted to radians) on the bearing. This is incorporated in the Q matrix:

$$Q = \begin{bmatrix} 0.15 * range(t, l) & 0 \\ 0 & 10 * Pi/180 \end{bmatrix}$$

,where $range(t, l)$ is the distance measured to landmark l at time t. Dependency on time step requires the Q matrix to be now recalculated at each iteration over the measurements.

Since noise on the odometry data is not known, some experimentation needs to be done with the initialization of the M matrix.

The state at each time step is stored from the algorithm in history variables. We start with the default value $M = 0.15I$, given in the original implementation. The data of the first 8 marks in the full dataset is used. Using this smaller subset can be used to visualize the corrected trajectory between all the marks and also to make the task computationally feasible for the approaches discussed next. Fig. 2 shows the estimated trajectory in this case. Visibly, the new trajectory goes closer to the marks than the original trajectory constructed from the noisy odometry observations (in red). However, there may be a better choice for M and the noise is visibly affecting the estimated trajectory with the default value. This noise is also visible with the big uncertainties, note that the plotted ellipses cover 34% confidence interval (half a standard deviation) for visualization purposes. An attempt is made to find a better choice for M by using grid search. To do this, a metric of a better choice of M has to be chosen. A trajectory is better than another if the sum of the distances from the estimated positions at the marked time to the actual positions of the marks is smaller. It should be noted that in this metric large uncertainty is not being penalized. Therefore, the resulting model has to be analyzed for this. The choice of ranges for M to be done grid search one partly make up for this problem. As defined in the textbook:

$$M = \begin{bmatrix} \alpha_1\delta_{trans} + \alpha_2\delta_{rot1} + \alpha_3\delta_{rot2} & 0 & 0 \\ 0 & \alpha_4\delta_{trans} + \alpha_5\delta_{rot1} + \alpha_6\delta_{rot2} & 0 \\ 0 & 0 & \alpha_7\delta_{trans} + \alpha_8\delta_{rot1} + \alpha_9\delta_{rot2} \end{bmatrix}$$

The grid search is done over the α_1 to α_9 variables. For each of them a value is chosen from an array of K values. Since the variables are too many, some of them already need to have predetermined values. After some experimentation with small K and earlier termination, it was determined that α_8 and α_9 are mostly zero. also α_3 tends to match α_6. After making these assignments, there are 6 variables to do the grid search on. The possible choices for values of all the variables are selected to be $[0, 0.15, 0.3]$. Other combinations were tried as well, but these gave the optimal result. The largest values is 0.3, as it is reasonable to assume that the odometry metrics are not extremely noisy. The result was $\alpha = [\alpha_1, \alpha_2, \alpha_3, \alpha_4,] = [0.3, 0, 0, 0.3, 0, 0, 0.3, 0, 0]$ The estimation for the best found M is plotted on Fig. 1b. It can be seen that while passing closely to all the other landmarks, the fifth landmark is further away from the path. However, it is still within the confidence range of the estimated position. It is visible that the estimated path has the expected eight shape, which is in contrast with the measured odometry positions.

An attempt to do grid search on the u_{error}, where $M = u_{error}I$, was also made, which yielded a value, giving estimation with similar distance metric.

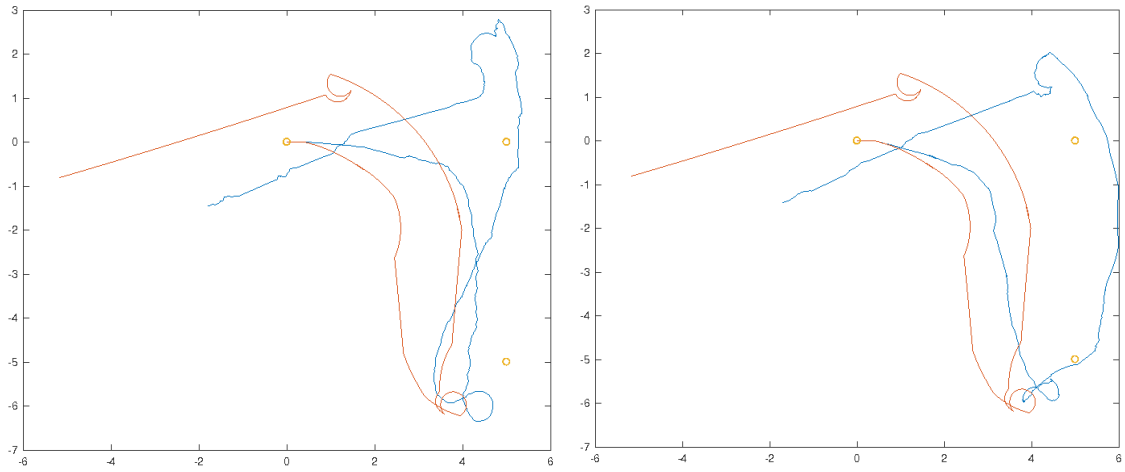
As for how M influences the model - the more the noise applied to the control vector, trajectory disturbances become more erratic. Biggest changes happen on sharper corners or larger velocities.

For comparison, on Fig. 1 are shown the estimated trajectories for the first three marks from the velocity (originally implemented) and the odometry model. The best found M found for the odometry model was used. For the velocity model, the last dimension of M is cut to fit the size of the control vector. Of course, there may be better parameters for the velocity model. However, the figure is to show the observed property of the odometry model to more easily preserve the shape of the odometry measurements even after distortion, done to consider the noise.

On Fig. 3 it is shown the estimated trajectory of the robot together with the uncertainty ellipses at equal distances. The green ellipses show the uncertainty of only using prediction (based on the fully updated covariance from the previous step) and the black ones - the same covariance after the correction step. For this choice of the best found M, the difference is very small, in some cases - barely visible. This is to be expected, as the calculated M is usually small, causing the algorithm to trust the odometry data more, leading to smaller corrections from the landmarks. However, if it were too small, the path will match very closely the odometry data, which is not the case.

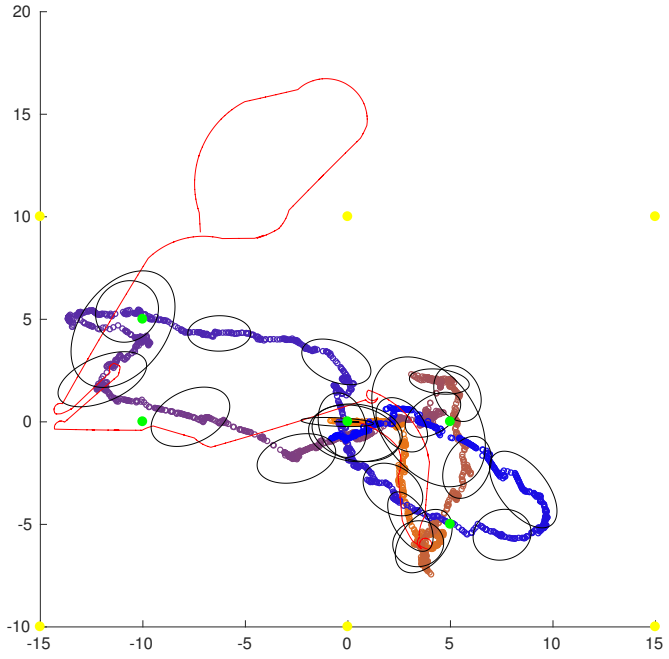
It can be seen that on the leftmost part the uncertainties are the biggest, which may be caused by the increased velocity and the dependency of this choice of M only on velocity.

When we tried the same choice of M for the velocity model, the results were also good. They are shown on Fig. 4. It can be seen that the path depicted here goes more closely to the points on



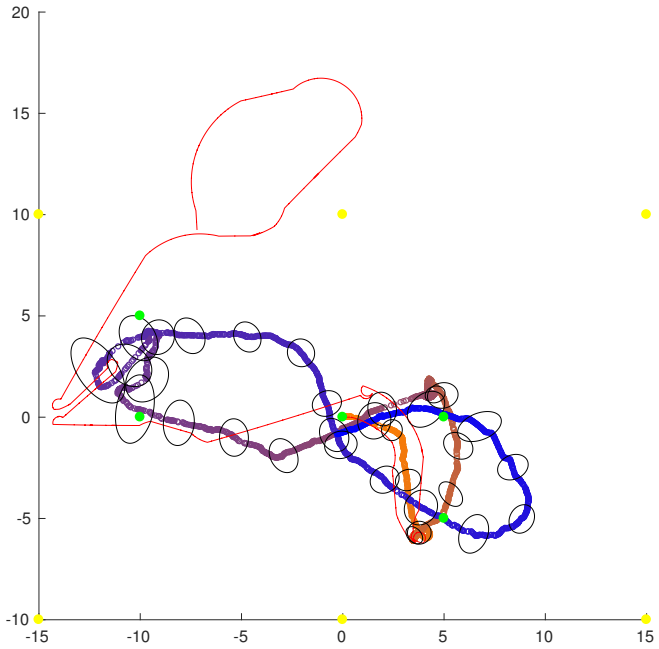
(a) Velocity model (b) Odometry model

Figure 1: Comparison of a velocity and odometry model (in blue) for the first 3 marks. For both - M is the optimal found for the odometry model (reduced for velocity model). The red line represents the odometry measurements.



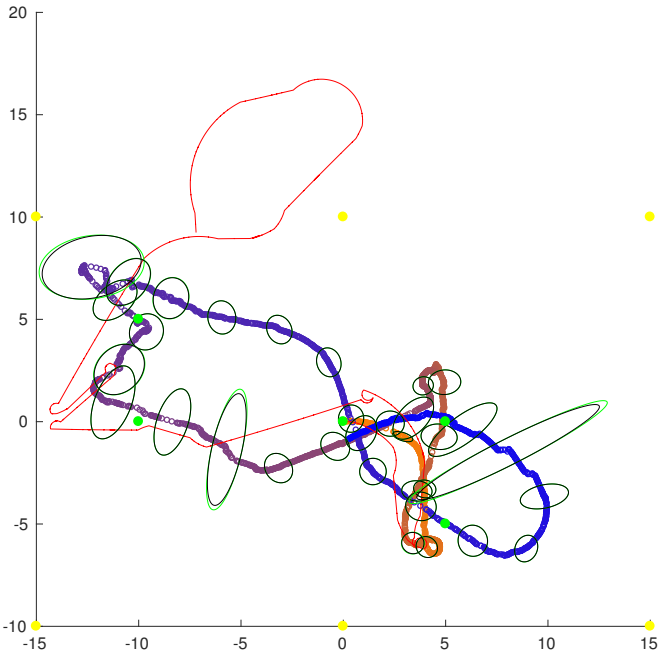
(a)

Figure 2: Trajectory(blue lines) and location uncertainties(black ellipses) on the estimated trajectory, using $M = 1.5I$. The trajectory from the odometry measurements is shown in red. Marks are shown in green, landmarks - in yellow.



(a)

Figure 3: Location uncertainties on the estimated trajectory, using the best found M. The trajectory from the odometry measurements is shown in red. Marks are shown in green, landmarks - in yellow.



(a)

Figure 4: Location uncertainties of the velocity model before(green ellipses) and after(black ellipses) correction on the estimated trajectory (orange-blue line), using best found M. The trajectory from the odometry measurements is shown in red. Marks are shown in green, landmarks - in yellow.

the left. However, the movement in the beginning is not modeled correctly, as it is seen that the first marked is missed by a lot. It can also be seen that this model's covariances can grow much bigger. This is to be expected, as the odometry model does not plan the position and can rely on the immediate odometry measurements.

The plotting of the uncertainties was done with two implementation separately - one is our own and one is the "error ellipse" function, taken from Matlab File Exchange. Both yielded the same result.

All the code is shown at the end of this document.

2 Task 2

Now the landmarks' positions are considered unknown, therefore changes have to be made to the algorithm in order to estimate their location. The implementation discussed next works with initially unknown number of landmarks. However, note that the assumption of known correspondences between a landmark and a measurement is still present. For the solution, we use the EKF SLAM algorithm, described in detail in Table 10.1

Extending the previous model, the estimated positions for the tracked landmarks so far will also be included in the state vector, besides the robot's pose:

$$\mu_t = \begin{bmatrix} x \\ y \\ \theta \\ m_{1,x} \\ m_{1,y} \\ \vdots \\ m_{L,x} \\ m_{L,y} \end{bmatrix}$$

Here $(m_{i,x}, m_{i,y})$ is the estimated position of the i-th landmark and L is the number of landmarks tracked so far. In this state vector we choose not to represent the landmark signatures, as they are unnecessary when there are known correspondences.

At this point it should be noted that we use exactly the same odometry model explained in Task 1. We need to take into account the fact that only the robot's pose (x, y, θ) must be updated at the prediction step, therefore we use the same matrix Fx as described in the book's algorithm for the calculations.

An important feature of our implementation is that the size of our state is not fixed from the start, but it is updated as new landmarks are seen by the robot. More specifically, when we have a measuremnt $z = (r, \phi)$ for a new landmark, the initial estimation of its location (l_x, l_y) is given by:

$$\begin{bmatrix} l_x \\ l_y \end{bmatrix} = \begin{bmatrix} m_{t,x} + r\cos(m_{t,\theta} + \phi) \\ m_{t,y} + r\sin(m_{t,\theta} + \phi) \end{bmatrix}$$

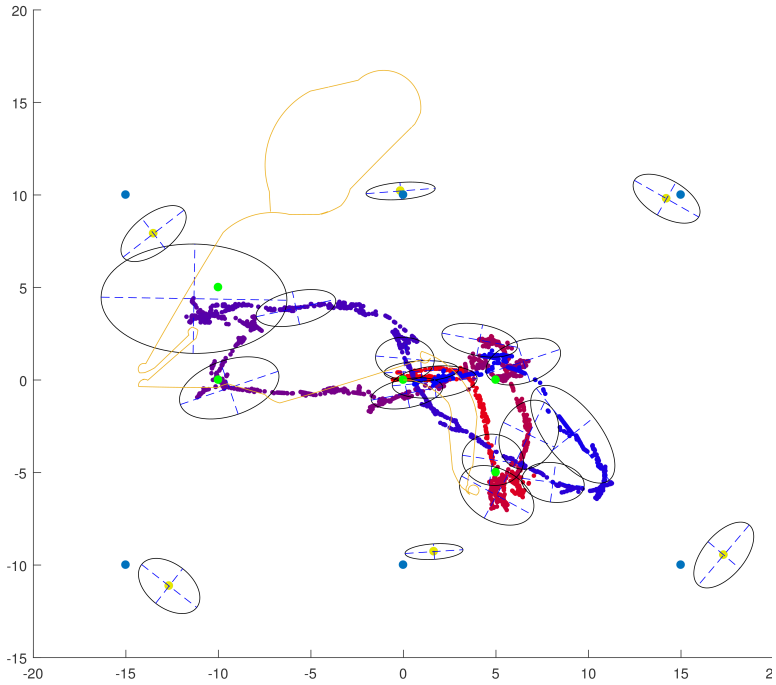
also known as the *inverse observation model*. This location is then appended to the end of the state vector m_t. But the covariance Σ of the state must also be extended by 2 rows/columns for uncertainty of the new landmark. Therefore, if M_{n+1} is a new landmark, then the new covariance is going to be

$$\Sigma = \begin{bmatrix} \Sigma_R & \Sigma_{R,M_1} & \cdots & \Sigma_{R,M_n} & \Sigma_{R,M_{n+1}} \\ \Sigma_{M_1,R} & \Sigma_{M_1} & \cdots & \Sigma_{M1,M_n} & \Sigma_{M_1,M_{n+1}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \Sigma_{M_n} & \Sigma_{M_n,M_1} & \cdots & \Sigma_{M_n} & \Sigma_{R,M_{n+1}} \\ \Sigma_{M_{n+1},R} & \Sigma_{M_{n+1},M_1} & \cdots & \Sigma_{M_{n+1},M_n} & \Sigma_{M_{n+1}} \end{bmatrix}, \quad \begin{array}{c} \Sigma_{M_{n+1}} = G_R\Sigma_R G_R^T + G_z Q G_z^T \\ \Sigma_{M_{n+1},Mi} = G_R\Sigma_{R,M_i} \\ \Sigma_{M_{n+1},R} = G_R\Sigma_R \end{array}$$

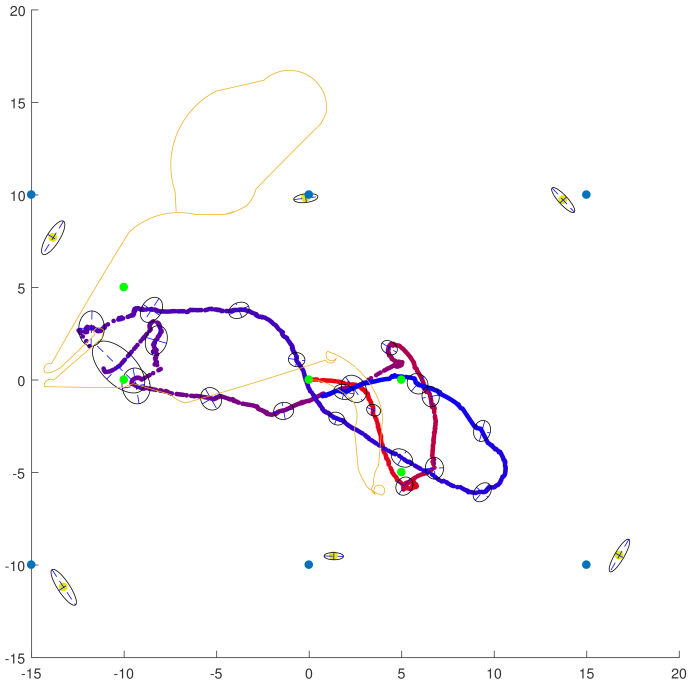
where G_R and G_z are the Jacobians of the *inverse observation model* with respect to the robot pose $(m_{t,x}, m_{t,u}, m_{t,\theta})$ and the measurement (r, ϕ) respectively:

$$G_R = \begin{bmatrix} 1 & 0 & -r\sin(m_{t,\theta} + \phi) \\ 0 & 1 & r\cos(m_{t,\theta} + \phi) \end{bmatrix}, \qquad G_z = \begin{bmatrix} \cos(m_{t,\theta} + \phi) & -r\sin(m_{t,\theta} + \phi) \\ \sin(m_{t,\theta} + \phi) & r\cos(m_{t,\theta} + \phi) \end{bmatrix}$$

On Figure 5 two simulation of the algorithm are shown until the robot reaches the 8th landmark, hence it travels on at least one 8-like path. The first one has the default given noise for the motion, $M = 0.15 * I$ and the other one uses same optimal noise from task 1. As we can see the locations of the landmarks are correctly predicted and the trajectory is roughly correctly estimated. As expected when the motion noise is smaller, which is our case when we use the optimal M, then the uncertainty also decreases. For better visualization, the second trajectory with the before and after measurement uncertainties is plotted on Fig. 6. As before, the updates seem to be small, except for some cases on the first half of the eight, where the already seen landmarks with established



(a) M = 0.15 * I



(b) Optimal M

Figure 5: Estimated trajectories and position uncertainties for EKF SLAM. The six blue points are the true landmarks, with their estimated uncertainties. Orange trajectory represents the odometry metrics, the red-blue trajectory - the estimated one, together with its uncertainties after correction.

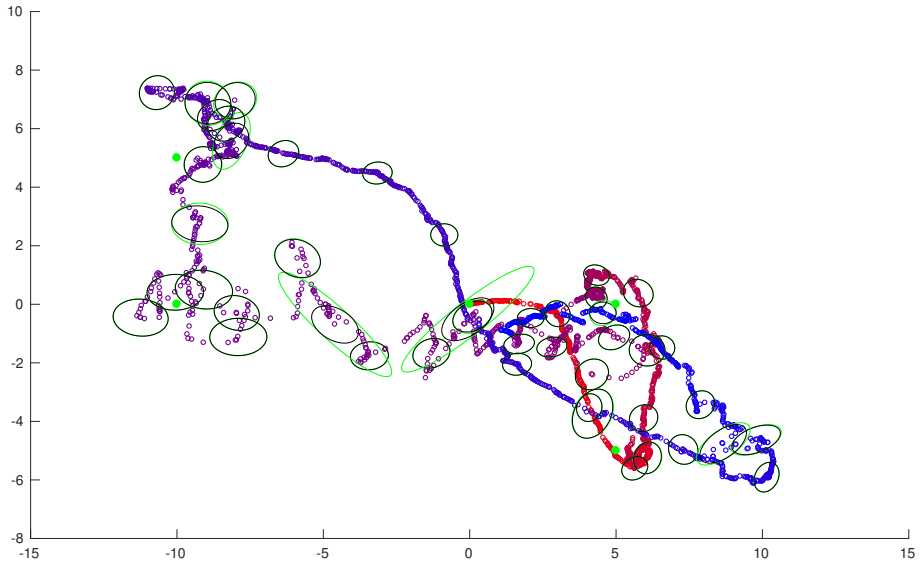
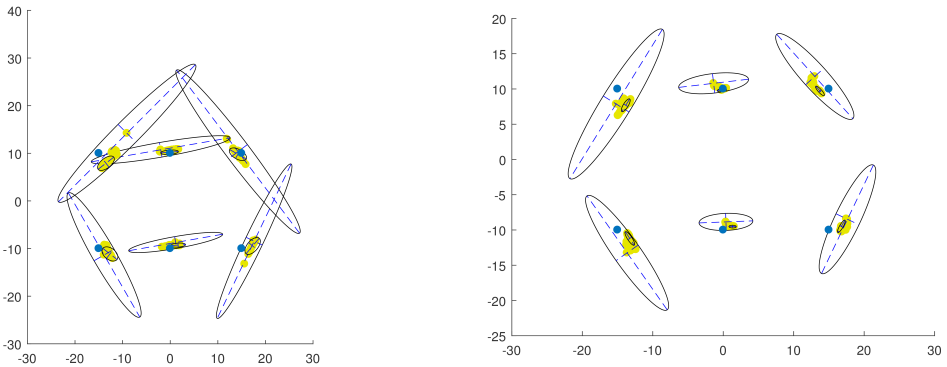


Figure 6: A close up of the trajectory (red-blue points, start is from red), estimated from EKF SLAM on the best found M, with uncertainties before (green ellipses) and after (black ellipses) correction. Red trajectory represents odometry measurements. The green points are the marks.

positions help reduce big uncertainties caused by higher velocity in one step. It can also be seen how on the second part of the eight shape the uncertainties are decreasing as a result of more certain positions of landmarks which are visible.

On Figure 7 a plot with the estimated landmark positions through time is shown. As we can see, the uncertainty of the landmarks is big when they are first seen, due to the uncertainty of the measurements, but by the end of the simulation it gets significantly smaller after multiple observations.



(a) M = 0.15 * I

(b) Optimal M

Figure 7: Landmark estimation through time. The uncertainties when a landmark is first seen (big ellipse) and at the end of the simulation (small ellipse) are displayed. The blue dots are the true positions

All the used Matlab code is shown at the end of this document.

2.1 Improvements

Many improvements can be applied to the default EKF SLAM algorithm in order to make it more efficient in more a practical setting.

One of those techniques is named the *provisional landmark list* and deals with *outliers* in the measurements. *Outliers* are essentially landmarks that are measured at unexpected locations. So instead of immediately adding them to the current state, they are added to a second list, the *provisional landmark list*. This list is like a parallel map to the original one, but its landmarks do not affect the robot pose. Once the uncertainty of a landmark in this list decreases enough, it is moved to the original map.

Another improvement to the algorithm is to keep *landmark existence probabilities*. When a landmark is seen, an internal value for that landmark is increased, signifying its existence. But if it is not seen in the position that it is expected to be, this value is decreased and if it goes beneath a threshold, then the landmark is removed.

Our implementation already contains one improvement over the default EKF SLAM algorithm. In the original EKF SLAM algorithm, when a new landmark is added its uncertainty is set to a very high value. What can be done instead, is to initialize it directly with the actual landmark uncertainty, as explained above. This reduces the numerical instabilities caused by the huge covariance.

The source code with the updated model and the plotting functions is shown below.

3 Code

3.1 Task 1

The modified function from Task 1, working with M, defined with alphas (as parameters) is given. The same function, but with modified definition of M and input parameters is used for using $M = u_{error} I$ for some given u_error.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%    Extended Kalman Filter
%    by Jrgen Sturm, Tijn Schmits, Arnoud Visser
%    April 2008
%
% Based on:
%
% Wolfram Burgard's
% http://ais.informatik.uni-freiburg.de/teaching/ss07/robotics/slides/
% --> 09.pdf
%
% Thrun's
% http://robots.stanford.edu/probabilistic-robotics/ppt/slam.ppt
%
% Dataset dlog.dat provided by Steffen Gutmann, 6.5.2004
% http://cres.usc.edu/radishrepository/view-one.php?name=comparison_of_self-localization_methods_continued
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%---------------------------------------------------------------------------- init
%
% clear;


function [x, P, xt, P_pred_hist, mark] = EKF_rot_model(logfilename, N, alpha)
    mark = [];
```

```
28      % expected user input noise
29
30  %     M = u_err*eye(3);
31
32      % expected robot location noise
33      m_err = .1;
34      Q = m_err*eye(2);
35
36      %------------------------------------------------------------------ data
            creation
37      %
38      % true robot position at t = 1
39      xt(:,1) = [0 0 0]'; dim = 3;  % x = [x y angle]'
40
41      % user input at t = 1
42      u(:,1) = [0 0]';              % u = [speed delta_angle]'
43
44      % Landmark locations
45      L2006 = [20   20 -20  -20;...
46            20 -20   20  -20];
47
48      % You also need the following information about the landmark
            positions:
49      % cyan:magenta -1500 -1000 magenta:cyan -1500 1000 magenta:green 0
            -1000 green:magenta 0 1000 yellow:magenta 1500 -1000 magenta:
            yellow 1500 1000
50      % 0 -> green 1 -> magenta 2 -> yellow 3 -> blue
51      L = [-15 -15 0 0 15 15;-10 10 -10 10 -10 10];
52      LID = [3 1 1 0 2 1;1 3 0 1 1 2];
53  %     U = M;          % user input noise (set to be equal to expected
        input noise)
54
55      angle = 0;
56
57      logfile = true;
58
59      if ~logfile
60
61          for t=2:N
62
63              % fabricate user input
64              u(2,t) = randn;
65              if abs(u(2,t)) > 0.4 %  P(steering) = 0.4
66                  u(2,t) = 0;
67              end
68              u(1,t) = .5*(1 - u(2,t)/0.4); % high delta_angle --> low
                    speed
69
70              % create noisy user input
71              un = U*randn(2,1) +u(:,t);
72
73              % calculate true robot position t+1
74              xt(:,t) =  [xt(1,t-1)+ un(1)*cos(xt(3,t-1)) ; ...
75                  xt(2,t-1)+ un(1)*sin(xt(3,t-1)) ; ...
76                  xt(3,t-1)+ un(2)];
77
78          end
79
```

```
80          %-------------------------------------------------------------------
                measurements
81          %
82          perc = .7; % percentage of Landmark measurement loss
83          for t=1:N
84              for landmark=1:size(L,2)
85                  if rand > perc
86                      % z = [distance angle]'
87                      z(:,t,landmark) = [ sqrt((L(1,landmark)-xt(1,t))^2
                            + (L(2,landmark)-xt(2,t))^2)+randn*m_err;...
88                          atan2(L(2,landmark)-xt(2,t),L(1,landmark)-xt(1,
                                t)) - xt(3,t)+randn*m_err];
89                  else
90                      z(:,t,landmark) = [0;0];
91                  end
92              end
93          end
94
95      else % logfile
96
97          fid = fopen(logfilename,'r');
98          t=0;
99          for i=1:N
100             tline = fgetl(fid);
101             [type,success] = sscanf(tline, '%s', 1);
102
103             if strcmp(type,'mark')
104                 fprintf(1,'*')
105                 mark(end+1) = t;
106                 continue
107             end
108
109             t = t + 1;
110             [xt(:,t),success] = sscanf(tline, 'obs: %*d %f %f %f', 3);
111             xt(1,t)=xt(1,t)/100; % milimeters to decimeters
112             xt(2,t)=xt(2,t)/100; % degrees to radians
113             xt(3,t)=xt(3,t)*pi/180;
114             if t > 1
115                 dx=xt(1,t)-xt(1,t-1);
116                 dy=xt(2,t)-xt(2,t-1);
117
118                 u(1,t) = sqrt (dx*dx+dy*dy); % speed
119                 u(2,t) = atan2(dy, dx) - xt(3,t-1); % diff_angle
120                 u(3,t) = xt(3,t) - xt(3,t-1) - u(2,t);
121
122                 %Add the defined noise to the clean control vectors
123 %                   u(:, t) = u(:, t) + mvnrnd([0 0 0], M)';
124             end
125             for landmark=1:6
126                 z(:,t,landmark) = [0;0];
127             end
128
129             [obs_landmarks, success,errmsg,nextindex] = sscanf(tline, '
                    obs: %*d %*f %*f %*f %d', 1);
130             for observation=1:obs_landmarks
131                 tline=tline(1,nextindex:size(tline,2));
132                 [signature, success] = sscanf(tline, ' ( %d:%d', 2);
133                 for landmark = 1:6
```

```
134                     if signature(1) == LID(1,landmark) && signature(2)
                            == LID(2,landmark)
135                         [z(:,t,landmark),success,errmsg,nextindex] =
                                sscanf(tline, ' ( %*d:%*d %f %f )', 2);
136                         z(1,t,landmark) = z(1,t,landmark) / 100; %
                                milimeters to decimeters
137                         z(2,t,landmark) = z(2,t,landmark) * pi / 180; %
                                 degrees to radians
138                     end
139                 end % for landmarks
140             end % for observations
141
142         end % for t=1:N
143         fclose(fid);
144     end % if logfile
145     N = t;
146
147     %------------------------------------------------------------------ a
            prioris
148     %
149     x_ = xt(1:3,1); % a priori x = true robot position
150     P_ =  0*eye(3); % a priori P = very certain (no error)
151     P_pred = 0*eye(3);
152
153     %
            ------------------------------------------------------------------
            EKF
154      %
155     x = zeros( dim, N );
156     P = zeros( dim, dim, N );
157     P_pred_hist = zeros( dim, dim, N );
158     I = eye(dim);
159     match = ones(1, N);
160
161     for t = 1:N
162 %------------------------------------------------------------------ prediction
163
164         %get user input
165     v = u(1,t);         % translation
166     rot1 = u(2,t);      % rot1
167     rot2 = u(3,t);  % rot2
168
169     % predicted robot position mean
170     x_ = x_ + [ v * cos(rot1 + x_(3)); v * sin(rot1 + x_(3)); rot1 +
            rot2];
171
172     % Jacobian with respect to robot location
173      g_ = [0 0 -v * sin(rot1 + x_(3));
174            0 0 v * cos(rot1 + x_(3));
175            0 0 0 ];
176
177     G = eye(dim) + g_;
178
179     % Jacobian with respect to control
180     V = [ cos(rot1 + x_(3)), -v * sin(rot1 + x_(3)) 0;
181           sin(rot1 + x_(3)), v * cos(rot1 + x_(3)) 0;
182           0, 1, 1];
183
```

```
184     M = diag([alpha(1)*v^2 + alpha(2)*(rot1)^2 + alpha(3)*rot2^2, ...
185          alpha(4)*v^2 + alpha(5)*rot1^2 + alpha(6)*rot2^2,...
186          alpha(7)*v^2 + alpha(8)*rot1^2 + alpha(9)*rot2^2]);
187
188     Rt = V * M * V';
189
190     % predicted covariance
191
192     P_ = G*P_*G' + Rt;
193     P_pred = P_;
194
195         %------------------------------------------------------------------
                correction
196         %
197         for landmark = 1:size(z,3)
198             if z(1,t,landmark) ~= 0   % if Landmark is measured
199
200                 % predicted measurement
201                 z_ = [sqrt((L(1,landmark)-x_(1))^2 + (L(2,landmark)-x_
                        (2))^2);...
202                     atan2(L(2,landmark)-x_(2),L(1,landmark)-x_(1)) - x_
                        (3)];
203
204                 % Jacobian of H with respect to location
205                 H(:,:,landmark) = [ -(L(1,landmark)-x_(1))/(L(1,
                        landmark)^2-2*L(1,landmark)*x_(1)+x_(1)^2+L(2,
                        landmark)^2-2*L(2,landmark)*x_(2)+x_(2)^2)^(1/2),
                        -(L(2,landmark)-x_(2))/(L(1,landmark)^2-2*L(1,
                        landmark)*x_(1)+x_(1)^2+L(2,landmark)^2-2*L(2,
                        landmark)*x_(2)+x_(2)^2)^(1/2),   0;
206                     (L(2,landmark)-x_(2))/(L(1,landmark)^2-2*L(1,
                            landmark)*x_(1)+x_(1)^2+L(2,landmark)^2-2*L(2,
                            landmark)*x_(2)+x_(2)^2),          -(L(1,landmark)
                            -x_(1))/(L(1,landmark)^2-2*L(1,landmark)*x_(1)+
                            x_(1)^2+L(2,landmark)^2-2*L(2,landmark)*x_(2)+
                            x_(2)^2),  -1];
207
208                 % predicted  measurement covariance
209
210                 Q = diag([0.15*z(1,t,landmark), 10*pi/180]);
211
212                 S = H(:,:,landmark)*P_*H(:,:,landmark)' + Q;
213
214                 %Kalman gain
215                 K(:,:,landmark) = P_* H(:,:,landmark)' / S;
216
217                 %innovation
218                 nu = z(:,t,landmark) - z_;
219
220                 %validation gate
221                 ro = nu'/S*nu; % From Kristensen IROS'03, section III.A
222
223                 if ro < 2
224                     %updated mean and covariance
225                     foundx(:,landmark) = x_ + K(:,:,landmark)*nu;
226                     foundP_(:,:,landmark) = (I-K(:,:,landmark)*H(:,:,
                            landmark))*P_;
227                 else
```

```
228                         %propagate known mean and covariance
229                         foundx(:,landmark) = x_;
230                         foundP_(:,:,landmark) = P_;
231                         z(:,t,landmark)=[0; 0];
232                     end
233 
234                 else
235                     %propagate known mean and covariance
236                     foundx(:,landmark) = x_;
237                     foundP_(:,:,landmark) = P_;
238                 end
239             end
240 
241             % determine mean
242             x_ = mean(foundx,2);
243             P_ = mean(foundP_,3);
244 
245             % create history
246             x(:,t) = x_;
247             P(:,:,t) = P_;
248             P_pred_hist(:,:,t) = P_pred;
249 
250         end
251     end
```

3.1.1 Grid Search

```
1  logfilename1 = {'dlog_firstmark.dat';'dlog_secondmark.dat';'
       dlog_thirdmark.dat'; 'dlog.dat'};
2  N = [758, 1159, 1435, 2951];%r51523];
3 
4  %Collect data from all datasets
5  a = [0 0.1 0.3]';
6  best_m_dist = 100000;
7  best_a = [];
8 
9  tic
10 for a1 = 1:size(a)
11     for a2 = 1:size(a)
12         for a3 = 1:size(a)
13             for a4 = 1:size(a)
14                 for a5 = 1:size(a)
15                     for a6 = 1:size(a)
16 
17                         alpha = [a(a1), a(a2), a(a5), ...
18                             a(a3), a(a4), a(a5), ...
19                             a(a3), 0, a(a6)
20                             ];
21 
22                         [x, P, xt, P_pred, mark] = EKF_rot_model(
                               logfilename1{4}, N(4), alpha);
23                         %filter only the points with positive definite
                               covariance
24 
25                         marks = [ 5, 5, 0, -10, -10; -5, 0, 0, 0, 5];
26                         mark_dist = [];
27                         for mark_i=1:size(mark, 2)
```

```
                                    mark_dist(end+1) = sqrt((x(1,mark(mark_i))-
                                        marks(1, mod(mark_i-1, 5)+1))^2 + (x(2,
                                        mark(mark_i))-marks(2, mod(mark_i-1, 5)
                                        +1))^2);
                                end

                                if best_m_dist > sum(mark_dist)
                                    best_m_dist = sum(mark_dist);
                                    best_a = alpha;
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
toc
best_a
```

3.1.2 Plotting

```
logfilename1 = {'dlog_firstmark.dat';'dlog_secondmark.dat';'
    dlog_thirdmark.dat'; 'dlog.dat'};
N = [758, 1159, 1434, 51523];

%Collect data from all datasets
alpha = [0.3 0. 0., 0.3 0. 0, 0.3 0 0];

[x, P, xt, P_pred] = EKF_rot_model(logfilename1{3}, N(3), alpha);

%filter only the points with positive definite covariance
pos_definite = [0];
for t=1:size(x,2)
    [~,p] = chol(P(1:2,1:2,t));
    pos_definite(t) = ~p;
end
pos_definite = find(pos_definite == 1);

x = x(:,pos_definite);
P = P(:, :,pos_definite);
P_pred = P_pred(:, :,pos_definite);

%Plot the estimated trajectory
figure;
plot(x(1, :), x(2,:))
hold on
%Plot the odometry measurement trajectory
plot(xt(1, :), xt(2,:))

%Equalize plot axis steps
tmpAspect=daspect();
daspect(tmpAspect([2 2 2]))

% Plot the uncertainties over a certain distance
dist = 0;
limit = 0;
old_x = x(1:2, 1);
for i=2:size(x, 2)
    dist = dist + sqrt((x(1, i)-old_x(1))^2+(x(2, i)-old_x(2))^2);
```

```
    if dist > limit
        draw_ellipse(P_pred(1:2, 1:2, i), x(1:2, i), 0.68, 'green')
        draw_ellipse(P(1:2, 1:2, i), x(1:2, i), 0.34, 'blue')

        dist = 0;
        limit = dist + 8;
    end
    old_x = x(1:2, i);
end

%Plot the marks
marks = [0, 5, 5; 0, 0, -5];
scatter(marks(1, :), marks(2, :));

% L = [-15 -15 0 0 15 15;-10 10 -10 10 -10 10];
% scatter(L(1,:), L(2,:), 'fill');
```

```
function [ radius, a ] = ellipse_parameters( C, conf )
%ELLIPSE_PARAMETERS Summary of this function goes here
%   Detailed explanation goes here
    [V, D] = eig(C);
    D = diag(D);
    [~,i] = max(D);
    D = sort(D, 'descend');

    a = atan2(V(2,i),V(1,i)); % angle of the ellipsoid
    a = rad2deg(a);
    k = chi2inv(conf,2);

    radius = 2 * sqrt(k * D); % radius of the ellipsoid
end
```

```
function [ output_args ] = draw_ellipse(C, x, conf, color)
%DRAW_ELIPSE Summary of this function goes here
%   Detailed explanation goes here
    [radius, a] = ellipse_parameters(C,conf);
    g = hgtransform;
    r = rectangle('Position', [-radius(1)/2 -radius(2)/2 radius(1)
        radius(2)], 'Curvature', [1 1] ,'Parent',g, 'EdgeColor', color)
        ;
    g.Matrix = makehgtform('translate', [x(1) x(2) 0], 'zrotate',
        deg2rad(a));

end
```

3.2 Task 2

```
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
%   Extended Kalman Filter
%   by J???rgen Sturm, Tijn Schmits, Arnoud Visser
%   April 2008
%
% Based on:
%
% Wolfram Burgard's
```

```
% http://ais.informatik.uni-freiburg.de/teaching/ss07/robotics/slides/
% --> 09.pdf
%
% Thrun's
% http://robots.stanford.edu/probabilistic-robotics/ppt/slam.ppt
%
% Dataset dlog.dat provided by Steffen Gutmann, 6.5.2004
% http://cres.usc.edu/radishrepository/view-one.php?name=comparison_of_self-localization_methods_continued
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-------------------------------------------------------------------------- init
%
clear;
logfilename1 = {'dlog_firstmark.dat';'dlog_secondmark.dat';'dlog_thirdmark.dat'; 'dlog.dat'};
N = [758, 1159, 1434, 51523];

N = N(3);
logfilename = logfilename1{3};

% expected user input noise
% u_err = .15;
% M = u_err*eye(3);
alpha = [0.3, 0, 0, 0,3, 0, 0, 0.3, 0, 0];

% expected robot location noise
m_err = .1;
Q = m_err*eye(2);

%------------------------------------------------------------------ data creation
%
% 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 0]';                % u = [speed delta_angle]'

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
```

```
61              u(2,t) = 0;
62          end
63          u(1,t) = .5*(1 - u(2,t)/0.4); % high delta_angle --> low speed
64
65          % create noisy user input
66          un = U*randn(2,1) +u(:,t);
67
68          % calculate true robot position t+1
69          xt(:,t) =  [xt(1,t-1)+ un(1)*cos(xt(3,t-1)) ; ...
70              xt(2,t-1)+ un(1)*sin(xt(3,t-1)) ; ...
71              xt(3,t-1)+ un(2)];
72
73      end
74
75      %-------------------------------------------------------------------
            measurements
76      %
77      perc = .7; % percentage of Landmark measurement loss
78      for t=1:N
79          for landmark=1:size(L,2)
80              if rand > perc
81                  % z = [distance angle]'
82                  z(:,t,landmark) = [ sqrt((L(1,landmark)-xt(1,t))^2 + (L
                        (2,landmark)-xt(2,t))^2)+randn*m_err;...
83                      atan2(L(2,landmark)-xt(2,t),L(1,landmark)-xt(1,t))
                            - xt(3,t)+randn*m_err];
84              else
85                  z(:,t,landmark) = [0;0];
86              end
87          end
88      end
89
90  else % logfile
91
92      marked = 0;
93      fid = fopen(logfilename,'r');
94      t = 0;
95      for i=1:N
96          tline = fgetl(fid);
97          [type,success] = sscanf(tline, '%s', 1);
98
99
100         if strcmp(type,'mark')
101             fprintf(1,'*')
102             marked = 1;
103             continue
104         end
105         t = t + 1;
106
107         [xt(:,t),success] = sscanf(tline, 'obs: %*d %f %f %f', 3);
108         xt(1,t)=xt(1,t)/100; % milimeters to decimeters
109         xt(2,t)=xt(2,t)/100;
110         xt(3,t)=xt(3,t)*pi/180; % degrees to radians
111         if t > 1
112             dx=xt(1,t)-xt(1,t-1);
113             dy=xt(2,t)-xt(2,t-1);
114
115             u(1,t) = sqrt (dx*dx+dy*dy); % speed
```

```
116             u(2,t) = atan2(dy, dx) - xt(3,t-1); % diff_angle
117             u(3,t) = xt(3,t) - xt(3,t-1) - u(2,t);
118         end
119         for landmark=1:6
120             z(:,t,landmark) = [0;0];
121         end
122
123         [obs_landmarks, success,errmsg,nextindex] = sscanf(tline, 'obs:
                %*d %*f %*f %*f %d', 1);
124         for observation=1:obs_landmarks
125             tline=tline(1,nextindex:size(tline,2));
126             [signature, success] = sscanf(tline, ' ( %d:%d', 2);
127             for landmark = 1:6
128                 if signature(1) == LID(1,landmark) && signature(2) ==
                        LID(2,landmark)
129                     [z(:,t,landmark),success,errmsg,nextindex] = sscanf
                            (tline, ' ( %*d:%*d %f %f )', 2);
130                     z(1,t,landmark) = z(1,t,landmark) / 100; %
                            milimeters to decimeters
131                     z(2,t,landmark) = z(2,t,landmark) * pi / 180; %
                            degrees to radians
132                 end
133             end % for landmarks
134         end % for observations
135         marked = 0;
136     end % for t=1:N
137     fclose(fid);
138 end % if logfile
139
140 N = t;
141
142 %------------------------------------------------------------------ a
        prioris
143 %
144 x_ = xt(1:3,1); % a priori x = true robot position
145 P_ =  0*eye(3); % a priori P = very certain (no error)
146
147 %
    ------------------------------------------------------------------------
        EKF
148  %
149 x = {};%zeros( dim, N );
150 P = {};%zeros( dim, dim, N );
151 P_pred_hist = {};
152 match = ones(1, N);
153
154 landmark_positions = [];
155 % N = 10;
156 for t = 1:N
157     %-------------------------------------------------------------------
            prediction
158     %
159
160     %get user input
161     v = u(1,t);           % velocity
162     rot1 = u(2,t);        % delta angle
163     rot2 = u(3,t);
164
```

```
165     Fx = zeros(3, dim);
166     Fx(1:3, 1:3) = eye(3);
167
168     % predicted robot position mean
169     x_ = x_ + Fx' *  [ v * cos(rot1 + x_(3)); v * sin(rot1 + x_(3));
            rot1 + rot2];
170
171     % Jacobian with respect to robot location
172      g_ = [0 0 -v * sin(rot1 + x_(3));
173            0 0 v * cos(rot1 + x_(3));
174            0 0 0 ];
175
176     G = eye(dim) + Fx' * g_ * Fx;
177
178     % Jacobian with respect to control
179     V = [ cos(rot1 + x_(3)), -v * sin(rot1 + x_(3)) 0;
180           sin(rot1 + x_(3)), v * cos(rot1 + x_(3)) 0;
181           0, 1, 1];
182
183     M = diag([alpha(1)*v^2 + alpha(2)*(rot1)^2 + alpha(3)*rot2^2, ...
184     alpha(4)*v^2 + alpha(5)*rot1^2 + alpha(6)*rot2^2,...
185     alpha(7)*v^2 + alpha(8)*rot1^2 + alpha(9)*rot2^2]);
186
187     Rt =     V * M * V';
188
189     % predicted covariance
190     P_ = G*P_*G' + Fx'*Rt*Fx;
191     P_pred = P_;
192     %------------------------------------------------------------------
            correction
193     %
194     for landmark = 1:size(z,3)
195         Q = diag([0.15 * z(1,t,landmark), deg2rad(10)]);
196
197         if z(1,t,landmark) ~= 0   % if Landmark is measured
198
199             %if it is a new landmark
200             if ~ismember(landmark, landmark_positions)
201                 r = z(1,t,landmark);
202                 phi = z(2,t,landmark);
203                 m_x = x_(1) + r * cos(x_(3) + phi);
204                 m_y = x_(2) + r * sin(x_(3) + phi);
205
206                 %expand
207                 x_(end + 1) = m_x;
208                 x_(end + 1) = m_y;
209
210                 landmark_positions(end + 1) = landmark;
211                 dim = dim + 2;
212
213
214                 %Expand covariance matrix
215                 P_(end+1:end+2, end+1:end+2) = 0;
216                 Gr = [ 1, 0, -r * sin(x_(3) + phi);
217                        0, 1 ,r * cos(x_(3) + phi)];
218                 Sr = P_(1:3,1:3);
219
220                 Gz = [ cos(x_(3) + phi) , -r* sin(x_(3) + phi);
```

```
                    sin(x_(3) + phi) , r * cos(x_(3) + phi)];

                P_(end-1:end, 1:3) = Gr * Sr;

                for i=4:2:size(P_,1)-2
                    P_(end-1:end, i:i+1) = Gr * P_(1:3,i:i+1);
                end
                P_(:, end-1:end) = P_(end-1:end,:)';

                P_(end-1:end, end-1:end) = Gr * Sr * Gr' + Gz * Q * Gz
                    ';



            end

            j = find(landmark_positions == landmark) - 1;
            m_x = x_(3 + 2*j + 1);
            m_y = x_(3 + 2*j + 2);

            d = [m_x - x_(1); m_y - x_(2)];

            q = d'*d;

            % predicted measurement
            z_ = [sqrt(q) ; atan2(d(2), d(1)) - x_(3)]; % z = h(x,m)

            Fxj = zeros(5, dim);
            Fxj(1:3, 1:3) = eye(3);
            Fxj(4:5, 3 + 2*j +1: 3 + 2*j + 2) = eye(2);


            % Jacobian of h wrt X and M_j
            Hit = (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;

            K = P_ * Hit' / ( Hit * P_ * Hit' + Q);
            x_ = x_ + K *( z(:,t,landmark) - z_ );
            P_ = (eye(dim) - K * Hit) * P_;


        end
    end


    % create history
    x{end + 1} = x_;
    P{end + 1} = P_;
    P_pred_hist{end+1} = P_pred;

%     stop

end
```

3.2.1 Plotting

```
EKF_SLAM_rot_model;
plot_trajectory(x,P, P_pred_hist)
```

```
figure;
for l=1:6
    plot_landmark(x,P, l)
    hold on
end
```

```
function [] = plot_landmark(X_h,P_h, l)
    % Plots a landmark
    X = [];
    Y = [];
    P = [];
    for i=1:size(X_h,2)-1
        x_t = X_h{i};
        P_ = P_h{i};
        if size(x_t, 1) >= 3 + l * 2
            x_l = x_t(3 + 2*l - 1);
            y_l = x_t(3 + 2*l);
            X(end + 1) = x_l;
            Y(end + 1) = y_l;
            P(:,:,end + 1) = P_(3 + 2*l - 1: 3 + 2*l,3 + 2*l - 1: 3 +
                2*l);
        end
    end


    pos_definite = [0];
    for t=1:size(X,2)
        [~,p] = chol(P(:,:,t));
        pos_definite(t) = ~p;
    end
    pos_definite = find(pos_definite == 1);
    %
    X = X(pos_definite);
    Y = Y(pos_definite);
    P = P(:,:,pos_definite);

    scatter(X,Y, 'filled', 'MarkerFaceColor', [0.9 0.9 0]);
%
    %Plot the uncertainties
    dist = 0;
    limit = 0;

    tmpAspect=daspect();
    daspect(tmpAspect([2 2 2]))

    draw_ellipse(P(:,:,1), [X(1); Y(1)], 0.68, 'blue');
    draw_ellipse(P(:,:,end), [X(end); Y(end)], 0.68, 'red');

end
```

```
function [ ] = plot_trajectory( x, P, P_pred_hist)
%PLOT_TRAJECTORY Summary of this function goes here
%   Detailed explanation goes here
    X = [];
    Y = [];
    for i=1:size(x,2)
        X(end + 1) = x{i}(1);
        Y(end + 1) = x{i}(2);
    end
```

```
10
11      tmpAspect=daspect();
12      daspect(tmpAspect([2 2 2]))
13
14      hold on
15      length = size(X,2);
16      start_color = [1, 0, 0];
17      end_color = [0, 0, 1];
18      colors_p = [linspace(start_color(1),end_color(1),length); linspace(
            start_color(2),end_color(2),length); linspace(start_color(3),
            end_color(3),length)]';
19      size(colors_p)
20      size(X)
21  %     plot(X, Y, 'LineWidth', 3);
22
23      scatter(X',Y', 10, colors_p, 'filled');
24
25      %Plot the uncertainties
26      if nargin > 1
27          size(x)
28          size(P)
29          pos_definite = zeros(1,size(x,2));
30          for t=1:(size(x,2))
31              [~,p] = chol(P{t}(1:2,1:2));
32              pos_definite(t) = ~p;
33          end
34          pos_definite = find(pos_definite == 1);
35
36          X = X(pos_definite);
37          Y = Y(pos_definite);
38          P = P(pos_definite);
39          P_pred_hist = P_pred_hist(pos_definite);
40
41
42          dist = 0;
43          limit = 0;
44          prev_x = [X(1) ; Y(1)];
45          for i=2:size(X, 2)
46              cur_x = [X(i); Y(i)];
47              dist = dist + sqrt(sum((cur_x - prev_x) .^ 2));
48
49              if dist > limit
50                  P_ = P{i}(1:2, 1:2);
51                  P_pred = P_pred_hist{i}(1:2, 1:2);
52
53                  draw_ellipse(P_pred, cur_x, 0.68, 'green')
54                  draw_ellipse(P_, cur_x, 0.68, 'blue')
55                  limit = limit + 0.6;
56                  dist = 0;
57              end
58              prev_x = cur_x;
59          end
60      end
61
62      marks = [0, 5, 5; 0, 0, -5];
63      scatter(marks(1, :), marks(2, :), 'filled','MarkerFaceColor', [0.9
            0.9 0]);
64
```

```
65 end
```