CS 287, Fall 2015 Optional Extra Credit Problems Extended Kalman Filtering and Gaussian Belief Space Planning through Sequential Convex Programming [40pts]

Deliverable: pdf write-up by Friday December 4th, 11:59pm, submitted through Gradescope. Late days cannot be used for this extra-credit assignment. **Your pdf should be one page per subquestion, for a total of four pages. Thanks!**

Please refer to the class webpage for the homework policy. Various starter files are provided on the course website: `www.cs.berkeley.edu/~pabbeel/cs287-fa15/`.

---

1. Extended Kalman Filter (EKF) [10pt]

In this question you get to implement an extended Kalman Filter (EKF) for state estimation for nonlinear dynamics and observation models.

Notes: Let $\mathbf{x} \in \mathbb{R}^{\texttt{xDim}}$ be the system state, $\mathbf{u} \in \mathbb{R}^{\texttt{uDim}}$ denote the control input applied to the system, and $\mathbf{z} \in \mathbb{R}^{\texttt{zDim}}$ be the vector of observations obtained about the system state using sensors. We are given a discrete-time stochastic dynamics model that describes how the system state evolves and an observation model that relates the obtained observations to the state, given here in state-transition notation:

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t, \mathbf{q}_t), \qquad \mathbf{q}_t \sim \mathcal{N}(\mathbf{0}, Q), \tag{1}$$

$$\mathbf{z}_t = \mathbf{h}(\mathbf{x}_t, \mathbf{r}_t), \qquad \mathbf{r}_t \sim \mathcal{N}(\mathbf{0}, R). \tag{2}$$

Given the current state estimate $\hat{\mathbf{x}}_t$, control input $\mathbf{u}_t$, covariance Σ_t, and observation $\mathbf{z}_{t+1}$, the EKF update equations are given by:

$$\hat{\mathbf{x}}_{t+1} = \mathbf{f}(\hat{\mathbf{x}}_t, \mathbf{u}_t, \mathbf{0}) + K_t(\mathbf{z}_{t+1} - \mathbf{h}(\mathbf{f}(\hat{\mathbf{x}}_t, \mathbf{u}_t, \mathbf{0}), \mathbf{0})), \tag{3a}$$

$$\Sigma_{t+1} = (I - K_t H_t)\Sigma^-_{t+1} \tag{3b}$$

where

$$A_t = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\hat{\mathbf{x}}_t, \mathbf{u}_t, \mathbf{0}), \qquad M_t = \frac{\partial \mathbf{f}}{\partial \mathbf{q}}(\hat{\mathbf{x}}_t, \mathbf{u}_t, \mathbf{0}), \tag{3c}$$

$$H_t = \frac{\partial \mathbf{h}}{\partial \mathbf{x}}(\mathbf{f}(\hat{\mathbf{x}}_t, \mathbf{u}_t, \mathbf{0}), \mathbf{0}), \qquad N_t = \frac{\partial \mathbf{h}}{\partial \mathbf{r}}(\mathbf{f}(\hat{\mathbf{x}}_t, \mathbf{u}_t, \mathbf{0}), \mathbf{0}), \tag{3d}$$

$$\Sigma^-_{t+1} = A_t \Sigma_t A_t^T + M_t Q M_t^T, \quad K_t = \Sigma^-_{t+1} H_t^T (H_t \Sigma^-_{t+1} H_t^T + N_t R N_t^T)^{-1}. \tag{3e}$$

Q. See `ekf.m` in the q1_starter folder and look for `YOUR_CODE_HERE` for the parts that you need to fill in. Use the routine provided in `numerical_jac.m` to compute the required Jacobians (Eqs. (3c), (3d)). The script `test_ekf.m` considers a two-dimensional nonlinear system defined by:

$$\mathbf{x}_{t+1}[1] = 0.1 * (\mathbf{x}_t[1])^2 - 2 * \mathbf{x}_t[1] + 20 + \mathbf{q}_t[1];$$
$$\mathbf{x}_{t+1}[2] = \mathbf{x}_t[1] + 0.3 * \mathbf{x}_t[2] - 3 + 3 * \mathbf{q}_t[2]$$

and

$$\mathbf{z}_t[1] = \mathbf{x}_t^T \mathbf{x}_t + \sin(5 * \mathbf{r}_t[1]);$$
$$\mathbf{z}_t[2] = 3 * (\mathbf{x}_t[2])^2 / \mathbf{x}_t[1] + \mathbf{r}_t[2],$$

where dynamics noise $\mathbf{q}_t$ and the measurement noise $\mathbf{r}_t$ are assumed to be drawn from a Gaussian distribution with zero mean and specified variances $Q = \left[\begin{smallmatrix} 2 & 0 \\ 0 & 2 \end{smallmatrix}\right]$ and $R = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 10 \end{smallmatrix}\right]$, respectively.

For verifying correctness, starting from an initial state $\mathbf{x}_0 \sim \mathcal{N}(\left[\begin{smallmatrix} 10 \\ 10 \end{smallmatrix}\right], \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right])$, a random trajectory $\mathcal{X}_{1:50}$ was generated and corresponding observations $\mathcal{Z}_{1:50}$ were collected. These have been provided to you in the form of files `X.mat` and `Z.mat`, respectively.

The script `test_ekf.m` plots the ground truth value of the state $\mathcal{X}_{1:50}$ along each dimension and also plots the state estimate given by the EKF (blue dotted line) and plots 3-standard deviations of the variance corresponding to each dimension. It also prints out the state estimate $\hat{\mathbf{x}}_{50}$ and covariance Σ_{50} at the final time step to help you verify correctness of your implementation.

Deliverable: Submit the results plot, which is saved to a png file by the script, and the mean $\hat{\mathbf{x}}_{50}$ and covariance Σ_{50} at the final time step.

2. Gaussian Belief Space Planning (BSP)

In this question you get to implement a belief space planning algorithm by building on the trajectory optimization framework introduced in PS#3.

We consider a Gaussian parameterization of the probability distribution over the state, also known as the *belief*. Specifically, the belief state $\mathbf{b}_t = \begin{bmatrix} \hat{\mathbf{x}}_t \\ \text{vec}[\sqrt{\Sigma_t}] \end{bmatrix}$ is a vector comprised of the mean state $\hat{\mathbf{x}}_t$ and the columns of the square root $\sqrt{\Sigma_t}$ of the covariance Σ_t of a Gaussian distribution $\mathcal{N}(\hat{\mathbf{x}}_t, \Sigma_t)$. We consider the square root $\sqrt{\Sigma_t}$ to enforce positive semidefiniteness of the covariance matrix. Two things to note here:

(a) The square root of a matrix is not unique and we use the principal square root, which is uniquely defined for a positive definite matrix, in our implementation for numerical reasons.

(b) We also include the lower (or equivalently, upper) triangular entries of the symmetric matrix $\sqrt{\Sigma_t}$ to eliminate redundancy in the belief state.

We assume that the initial belief $\mathbf{b}_1 = \begin{bmatrix} \hat{\mathbf{x}}_1 \\ \text{vec}[\sqrt{\Sigma_1}] \end{bmatrix}$ is given. Given a current belief $\mathbf{b}_t$, a control input $\mathbf{u}_t$, and an observation $\mathbf{z}_{t+1}$, the evolution of the belief state $\mathbf{b}_{t+1} = \mathbf{g}(\mathbf{b}_t, \mathbf{u}_t)$ can be described using a Kalman filter (such as an extended Kalman filter) and is a stochastic process. However, we eliminate the stochasticity from the belief dynamics by assuming that the maximum likelihood observation is obtained at each time step, i.e., $\mathbf{z}_{t+1} = \mathbf{h}(\mathbf{f}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t, \mathbf{0}), \mathbf{0})$ [1]. Adapting the EKF update equations from Eq. (3), the deterministic belief dynamics are now given by:

$$\mathbf{b}_{t+1} = \mathbf{g}(\mathbf{b}_t, \mathbf{u}_t) = \begin{bmatrix} \hat{\mathbf{x}}_{t+1} \\ \text{vec}\left[\sqrt{(I - K_t H_t)\Sigma_{t+1}^-}\right] \end{bmatrix}, \quad \text{where} \tag{4a}$$

$$\hat{\mathbf{x}}_{t+1} = \mathbf{f}(\hat{\mathbf{x}}_t, \mathbf{u}_t, \mathbf{0}), \qquad \Sigma_{t+1}^- = A_t\sqrt{\Sigma_t}(A_t\sqrt{\Sigma_t})^T + M_t Q M_t^T, \tag{4b}$$

where Jacobian matrices A_t, M_t, H_t, N_t, and Kalman gain matrices K_t are computed using Eqs. (3c) - (3e). This makes the belief dynamics suitable for trajectory optimization using sequential quadratic programming (SQP). Note that the maximum likelihood observation assumption is only used for trajectory optimization but observations have to be taken into account during execution (refer to the note at the end of Q 2(i)).

The objective is to compute a locally optimal trajectory in the (Gaussian) belief space to minimize uncertainty during execution. For notational convenience, we concatenate the belief states and control inputs for all time steps $1 \leq t \leq T$ to form $\mathcal{B} = [\mathbf{b}_1 \dots \mathbf{b}_T]$ and $\mathcal{U} = [\mathbf{u}_1 \dots \mathbf{u}_{T-1}]$ that parameterize a belief space trajectory such that $\mathbf{b}_{t+1} = \mathbf{g}(\mathbf{b}_t, \mathbf{u}_t)$. We will solve the following constrained nonlinear optimization problem:

$$\min_{\mathcal{B},\mathcal{U}} \quad C(\mathcal{B},\mathcal{U}) = \alpha_{\text{final\_belief}} \,\text{tr}[\Sigma_T] + \sum_{t=1}^{T-1} (\alpha_{\text{belief}} \,\text{tr}[\Sigma_t] + \alpha_{\text{control}} \mathbf{u}_t^T \mathbf{u}_t) \tag{5a}$$

$$\text{s. t.}_{\forall t \in \{1,\dots,T-1\}} \quad \mathbf{b}_{t+1} = \mathbf{g}(\mathbf{b}_t, \mathbf{u}_t), \quad \triangleright \text{ belief dynamics constraint (nonlinear)} \tag{5b}$$

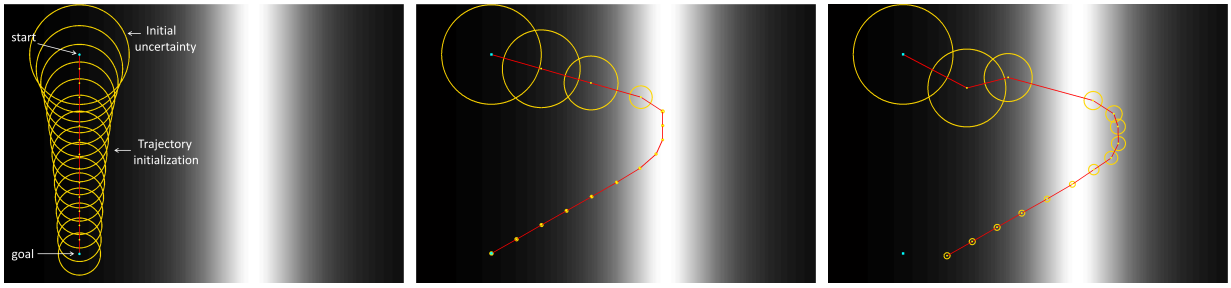
$$\hat{\mathbf{x}}_T = \mathbf{x}_{\text{goal}}, \quad \triangleright \text{ robot reaches goal at time T} \tag{5c}$$

$$\mathbf{x}_{\min} \leq \hat{\mathbf{x}}_t \leq \mathbf{x}_{\max}, \quad \triangleright \text{ bounds on state} \tag{5d}$$

$$\mathbf{u}_{\min} \leq \mathbf{u}_t \leq \mathbf{u}_{\max}, \quad \triangleright \text{ control input bounds} \tag{5e}$$

where the cost function $C(\mathcal{B},\mathcal{U})$ encodes the objective of minimizing uncertainty (minimizing the trace of the covariance $\text{tr}[\Sigma_t] = \text{tr}[\sqrt{\Sigma_t}^T \sqrt{\Sigma_t}]$) while penalizing the control effort, and α_{control}, α_{belief}, and $\alpha_{\text{final\_belief}}$ are user-defined weights.

Q 2(i)[15pt] `belief_opt_penalty_sqp.m` in the q2_starter folder contains an incomplete implementation of the penalty SQP method for belief space trajectory optimization. Look for `YOUR_CODE_HERE` for the parts



(a) (b) (c)

Figure 1: Point robot moving in a 2D light-dark domain adapted from Platt et al. [1]. (a) The optimization is initialized with a straight line trajectory (shown in red) from the start to the goal. The beliefs along the initialized trajectory are shown in yellow. (b) Sample solution obtained using trajectory optimization (included for reference). (c) Simulation of open-loop execution of control computed using belief space planning. Since the maximum likelihood observation assumption is not valid during execution, the trajectory deviates from the planned trajectory (see `simulate_bsp_trajectory.m` for code).

that you need to fill in. In particular – (i) complete the SQP outer loop, (ii) compute the merit function for a given belief trajectory and set of controls, and (iii) add the cost function and constraints given in Eq. (5) to the `cvx` problem definition.

Some things to keep in mind:

(a) Directly add constraints given in Eqs. (5c), (5d), and (5e) to the `cvx` problem.

(b) In the penalty SQP formulation, the merit function that is minimized is of the form:

$$\texttt{merit} = C(\mathcal{B},\mathcal{U}) + \sum_{t=1}^{T-1} \texttt{penalty\_coeff} * |\text{violation of belief dynamics constraint Eq. (5b)}|, \qquad (6)$$

where the constraint at each time step t is linearized around the current solution $(\mathbf{b}_{t+1}^{(k)}, \mathbf{b}_t^{(k)}, \mathbf{u}_t^{(k)})$ at the k^{th} SQP iteration.

(c) Add in trust region constraints on the optimization variables of the form $\|\mathbf{b}_t - \mathbf{b}_t^{(k)}\|_2 < \texttt{trust\_box\_size}$ and $\|\mathbf{u}_t - \mathbf{u}_t^{(k)}\|_2 < \texttt{trust\_box\_size}$ to ensure that the optimization progresses only within bounds of the region where the locally convex approximation holds.

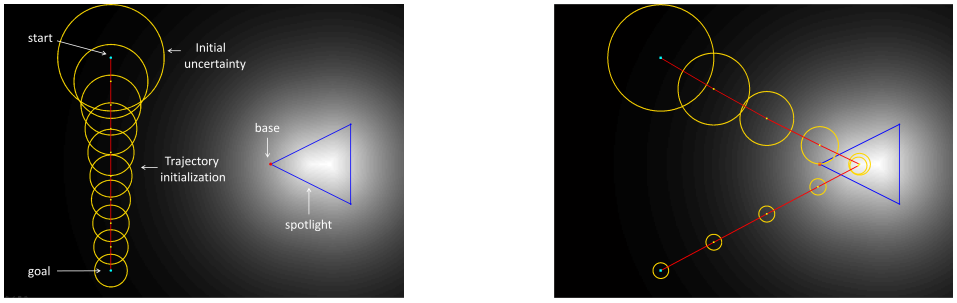
To test your implementation, look at the `test_bsp_light_dark.m` file that considers an adaptation of the light-dark planning example from Platt et al. [1]. The robot's ability to localize itself depends upon the amount of light present at its actual position. The light, i.e., observation noise, varies as a quadratic function of the x-coordinate. Fig. 1 shows the light-dark domain considered in this example. The intensity in the figure illustrates the magnitude of the light over the domain.

Depending upon the start and goal positions, and the light configuration, the robot may need to move towards the light to localize itself before coming back to the goal to reduce uncertainty, as shown in Fig. 1(b).

The script `test_bsp_light_dark.m` sets up five test cases by varying the initial beliefs and goal positions. Fig. 1(b) shows the optimized trajectory for the first test case and is included for reference so that you can verify correctness of implementation. Also provided is the cost of the optimized belief trajectory $C(\mathcal{B},\mathcal{U})$ for each of the five test cases.

Deliverable: Submit the optimized belief space trajectories for the five cases, which are saved to png files by the script, and the reported expected cost for the optimized trajectories.

Note: In Q 2(i), we computed a locally optimal trajectory in belief space, which assumes that maximum likelihood observations are obtained. However, this assumption does not hold during execution as observations have to accounted for. Fig. 1(c) shows the means and covariances obtained by open-loop execution of the controls computed by the trajectory optimization, which deviates significantly from the planned trajectory. To account for this deviation, one needs to re-plan at every time step in a model predictive control (MPC) fashion, or design a feedback controller around the locally-optimal trajectory.



(a) (b)

Figure 2: Point robot moving in a 2D domain lit up by a triangular spotlight (shown in blue), which can rotate around the base (red square). The default orientation of the spotlight ($\theta = 0$) is shown. (a) The optimization is initialized with a straight line trajectory (shown in red) from the start to the goal. The beliefs along the initialized trajectory are shown in yellow. (b) Sample solution obtained using trajectory optimization (included for reference) for the default orientation.

For your reference, sample code to simulate execution is provided in `simulate_bsp_trajectory.m`. In particular, this function takes as input the initial belief and a set of controls, and at each time step, propagates the belief $\mathbf{b}_t$ based on the applied control input $\mathbf{u}_t$ and observation $\mathbf{z}_{t+1}$. Note that, unlike the planning phase, the belief dynamics has to incorporate observations obtained during executions. For simulating observations, we also maintain the (hidden) true state $\mathbf{x}_{\text{true}}$, which is initially sampled from the initial belief, and is propagated using the stochastic dynamics function given in Eq. (1).

Q 2(ii)[5pt] In this question, we consider a variation of the light-dark example from Q 2(i) in which we have a triangular spotlight, as shown in Fig. 2. The intensity in the figure illustrates the magnitude of the light over the domain. Since trajectory optimization requires continuous gradients to progress, we approximate the light intensity given out by the spotlight as a function of the signed distance to the boundary of the spotlight (computed by the provided function in `signed_distance_spotlight.m`). The spotlight can also rotate about its base. Just like in the light-dark domain, the robot's ability to localize itself depends upon the amount of light present at its actual position.

The script `test_bsp_spotlight.m` sets up three test cases by changing the orientation of the spotlight ($\theta = \{0, \pi/2, \pi\}$). Fig. 2(b) shows the optimized trajectory for the first test case and is included for reference so that you can verify correctness of implementation. In all three cases, the optimal trajectory moves the robot to the spotlight for accurate localization before leading the robot back to the goal. Also provided is the cost of the optimized belief trajectory $C(\mathcal{B},\mathcal{U})$ for each of the three test cases.

Deliverable: Submit the optimized belief space trajectories for the three cases, which are saved to png files by the script, and the reported expected cost for the optimized trajectories.

Q 2(iii)[10pt] In this question, we now consider the problem of simultaneously planning the motion of the robot and the spotlight. This often arises in tracking applications or when one needs to simultaneously plan motions for the robot and mobile sensors. In this case, ideally, the spotlight should track the motion of the robot to minimize overall uncertainty. This is readily accomplished using our belief space planning framework by making a few minor modifications.

To this end, modify the `test_bsp_spotlight.m` script to do the following:

(a) Augment the state to now also include the spotlight orientation θ, i.e., $\mathbf{x} = [x, y, \theta]^T$, where $[x, y]^T$ is the position of the robot. Define the dynamics and observation functions as:

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t, \mathbf{q}_t) = \mathbf{x}_t + \mathbf{u}_t * \texttt{dT} + 0.01 * \mathbf{q}_t, \qquad \mathbf{q}_t \sim \mathcal{N}(\mathbf{0}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}),$$

$$\mathbf{z}_t = \mathbf{h}(\mathbf{x}_t, \mathbf{r}_t) \quad = \begin{bmatrix} x_t + \texttt{intensity} * \mathbf{r}_t[1] \\ y_t + \texttt{intensity} * \mathbf{r}_t[2] \\ \theta_t \end{bmatrix}, \qquad \mathbf{r}_t \sim \mathcal{N}(\mathbf{0}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}),$$

where the observation noise scaling factor `intensity` is a function of the spotlight orientation θ_t.

(b) Initialize the optimization with a straight line trajectory from the start to the goal for the robot. For the spotlight, initialize the optimization with 0 controls, i.e., $\mathbf{u}_t[3] = 0 \; \forall t \in \{1, \ldots, T-1\}$.

(c) The goal constraint (Eq. (5c)) is only on the robot position, i.e, $\mathbf{x}_T[1:2] = \texttt{goal}$. There is no constraint on the spotlight orientation at the final time step.

(d) Setup the state bounds as $\mathbf{x}_{\min} = [-4, -3, -2\pi]^T$ and $\mathbf{x}_{\max} = [4, 3, 2\pi]^T$, and the control input bounds as $\mathbf{u}_{\min} = [-1, -1, -\pi/3]^T$ and $\mathbf{u}_{\max} = [1, 1, \pi/3]^T$.

(e) Set the `display_spotlight_trajectory` flag to 1 to plot the spotlight at each time step along the trajectory. The color varies from red ($t = 1$) to blue ($t = T$), to indicate the rotation of the spotlight across time steps. The intermediate spotlight configurations are also drawn using dashed line segments.

Deliverable: Compute belief space plans for different initial beliefs $\mathbf{b}_1 \sim \mathcal{N}(\begin{bmatrix} -3 \\ 2 \\ \theta_1 \end{bmatrix}, I), \; \theta_1 = \{0, \pi/2, 2\pi\}$. Submit the optimized trajectories, which are saved as png files by the script.

References

[1] R. Platt, R. Tedrake, L. Kaelbling, T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. *Robotics: Science and Systems*, 2010.