ESE 650, Spring 2019 Assignment 3

Due March 15, 2019

1 Part 1 (30 Points)

1. Suppose that the pose of a moving robot with respect to the world frame is given by the following function of time t:

$$T(t) = \begin{bmatrix} \cos\frac{t\pi}{3} & 0 & -\sin\frac{t\pi}{3} & t \\ 0 & 1 & 0 & 0 \\ \sin\frac{t\pi}{3} & 0 & \cos\frac{t\pi}{3} & 2t \\ 0 & 0 & 0 & 1 \end{bmatrix} \in SE(3)$$

   (a) Find the axis-angle representations of the robot orientation at time $t = 1$.

   (b) Find the quaternion representations of the robot orientation at time $t = 1$ and of the inverse of this orientation.

   (c) Compute the linear and the angular velocity with respect to the world frame at time $t = 1$.

   (d) Compute the linear and the angular velocity with respect to the robot frame at time $t = 1$.

   (e) Compute the coordinates of the point $p_W = (9, 0, 0)$ in the robot frame (i.e., find p_R in the robot frame) at time $t = 1$.

2. Consider the discrete-time nonlinear time-invariant system:

$$\begin{aligned} x_{t+1} &= -0.1x_t + \cos(x_t) + w_t, & w_t &\sim \mathcal{N}(0, 1) \\ z_t &= x_t^2 + v_t, & v_t &\sim \mathcal{N}(0, 0.5) \end{aligned}$$

   with a prior distribution $x_0 \sim \mathcal{N}(0, 1)$.

   (a) Derive the extended Kalman filter equations by linearizing around the nominal state $x_t^{nom} \equiv 1$.

   (b) Derive the unscented Kalman filter equations around the nominal state $x_t^{nom} \equiv 1$.

3. Consider a vehicle equipped with a laser scanner, whose dynamics can be modeled by

$$\begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v_c \cos(\phi) \\ v_c \sin(\phi) \\ v_c \tan(\alpha) \end{bmatrix} \tag{1}$$

   Now, if we translate our model to the laser point, let us call this point (x_v, y_v), we have

$$\begin{bmatrix} x_v \\ y_v \end{bmatrix} = \begin{bmatrix} x_c + a\cos\phi - b\sin\phi \\ y_c + a\sin\phi + b\cos\phi \end{bmatrix} \tag{2}$$

   The velocity v_e is measured with an encoder located in the back left wheel. This velocity is translated to the center of the axle by

$$v_c = \frac{v_e}{1 - \tan(\alpha)\frac{H}{L}} \tag{3}$$

Finally the discrete model in global coordinates can be approximated with the following set of equations:

$$\begin{bmatrix} x_v(t+\Delta T) \\ y_v(t+\Delta T) \\ \phi(t+\Delta T) \end{bmatrix} = \begin{bmatrix} x_v(t) + \Delta T(v_c \cos(\phi) - \frac{v_c}{L}\tan(\alpha)(a\sin(\phi) + b\cos(\phi)) \\ y_v(t) + \Delta T(v_c \sin(\phi) + \frac{v_c}{L}\tan(\alpha)(a\cos(\phi) - b\sin(\phi)) \\ \phi(t) + \Delta T \frac{v_c}{L}\tan(\alpha) \end{bmatrix} \tag{4}$$

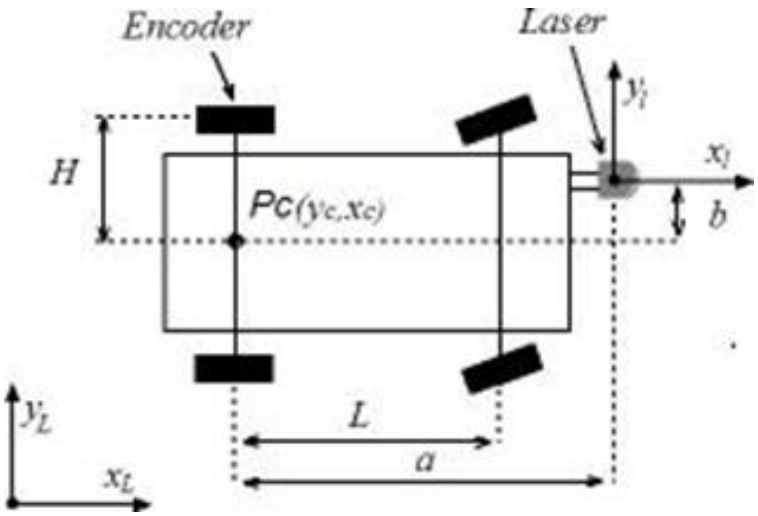

Figure 1: Vehicle kinematics

Furthermore, we have an observational model of the form

$$\begin{bmatrix} z_r \\ z_\beta \end{bmatrix} = h(x) = \begin{bmatrix} \sqrt{(x_L - x_v)^2 + (y_L - y_v)^2} \\ \arctan\left(\frac{y_L - y_v}{x_L - x_v}\right) - \phi + \frac{\pi}{2} \end{bmatrix} \tag{5}$$

where (x_L, y_L) are the coordinates of a landmark.

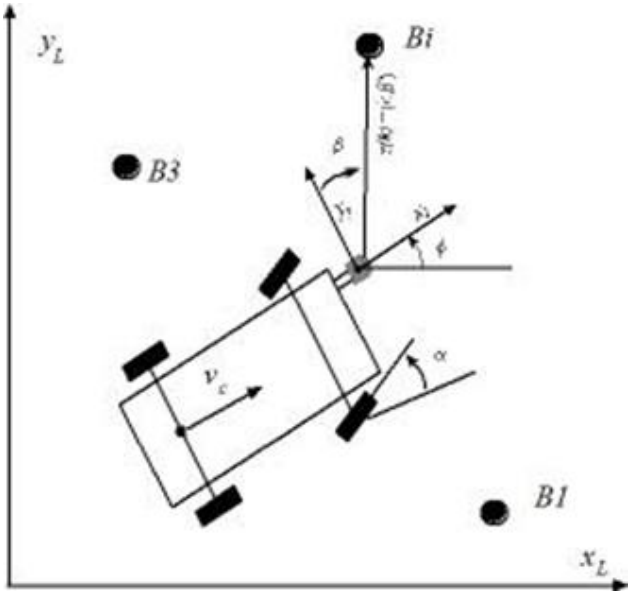

Figure 2: Vehicle and Laser sensor

Figure 3: SLAM data collection vehicle

(a) Compute the motion model Jacobian, where (x_v, y_v, ϕ_v) are the state variables.

(b) Compute the measurement model Jacobian, where $(x_v, y_v, \phi_v, x_L, y_L)$ are the state variables (assume a single landmark).

2 Part 2 - EKF SLAM (70 Points)

In this problem we will be implementing a full SLAM system based on an extended Kalman filter. The data we will be using was taken from a pickup truck equipped with a wheel encoder, GPS, and LIDAR scanner. The dataset we are using consists of this truck making several loops around a park with trees. See 3 for a picture of the data collection vehicle and the environment. The vehicle model is the same as in question 1.3 above; the specific parameters corresponding to the vehicle dimensions and sensor placement are provided in the code.

Our map will consist of the centers of the trees in the park as point features in 2D, and our vehicle state will be the usual 2D vehicle state $[x\ y\ \phi]$. To incorporate the map itself within our estimation, we need to include them as well within our EKF's state vector. Thus, for a map that currently includes n tree landmarks, the full state vector $\boldsymbol{x}$ will be given as the $3 + 2n$ dimensional vector

$$\boldsymbol{x} = \begin{bmatrix} x & y & \phi & \boldsymbol{f}_1^T & \dots & \boldsymbol{f}_n^T \end{bmatrix}^T \tag{6}$$

where $\boldsymbol{f}_i = [x\ y]^T$ is the 2D position estimate of the ith tree center feature.

From each laser scan, we begin by extracting tree features from the environment by detecting appropriately shaped clusters of laser measurements, estimating the trunk diameter, and finally estimating the range and bearing to the trunk center from the truck's position. This has already been implemented for you as `extract_trees` within the `tree_extraction` package.

Note that as these real measurements aren't time synchronized perfectly or even coming in at the same frequency, we won't have simple propagate-update iterations, but rather must process odometry/GPS/laser

Algorithm 1 : EKF SLAM

```
while filter running do
    e ← next event to process
    if e is an odometry event then
        Perform EKF propagation with e
    else if e is a GPS measurement then
        Perform an EKF update with e
    else if e is a laser scan then
        Extract tree range, bearing measurements {z} from e
        Perform an EKF update with {z}
    end if
end while
```

events as they become available to us. High level pseudocode of the entire SLAM algorithm can be written as in Algorithm 1.

After extracting a measurement $\boldsymbol{z}_j = (r, b)$ of a tree trunk in the environment from a laser scan, we do not know a priori *which* tree in our state vector this tree corresponds to. The measurement Jacobians and hence EKF update will take a different form depending on whether measurement $\boldsymbol{z}_j$ is an observation of $\boldsymbol{f}_1$ or $\boldsymbol{f}_6$, so before we perform the EKF update we must first decide which measurements j correspond to which landmarks i. Furthermore, there is the possibility that the measurement corresponds to a tree that currently is not even in our map, requiring us to initialize a new landmark and expand our state vector. This is known as the problem of *data association*.

We have included several plotting utilities that should aid your development. They are contained in the `slam_utils` package and should run by simply calling `init_plot` and then `do_plot` with your current `ekf_state`. Laser measurements are displayed *green* if they are associated with an existing mapped landmark, *blue* if they correspond to initialization of a new landmark, or *red* if they were thrown out and not used. The plotting code requires `PyQt5` and `pyqtgraph`; `matplotlib` has a simpler interface but is too slow for real-time visualization as done here. The plot will auto-scale as the map expands, or you can manually zoom in and pan with the mouse. Plotting the covariance ellipse for every estimated landmark is useful but takes a significant amount of time; this can be disabled by setting `filter_params["plot_map_covariances"] = False`.

2.1 Odometry Propagation

We will begin by implementing the propagation steps of the EKF. The motion model and Jacobians were given or derived in question 1.3 above. On completing both subsections below, you should be able to run the code and see a moving (though very inaccurate) estimate of the truck's position, along with a quickly growing 3σ covariance ellipse.

2.1.1 Vehicle motion model

Implement the vehicle motion model and its Jacobian in the function `motion_model`

2.1.2 EKF Propagation

Implement the EKF odometry propagation in the function `odom_predict`

2.2 GPS Update

Implement the GPS update equations in the function `gps_update`. GPS measures position directly, so the measurement equation is simply $h(\boldsymbol{x}) = [x\ y]^T$. After completing this, you should have a much improved filter; on each GPS measurement, the estimate should jump back towards the true value and the covariance ellipse should shrink significantly.

Due to the nature of GPS there are a few measurements within the given data that are erroneous and should be thrown out. In an EKF update, the innovation or residual $\boldsymbol{r}$ and residual covariance $\boldsymbol{S}$ are useful

for determining if a measurement is consistent with your current state estimate. Let the *Mahalanobis distance* of this residual be given as the scalar quantity

$$d(\boldsymbol{r}, \boldsymbol{S}) = r^T \boldsymbol{S}^{-1} r \tag{7}$$

Because the residual is normally distributed with covariance $\boldsymbol{S}$, d follows a known distribution, a χ^2 distribution with m degrees of freedom, where m is the dimension of $\boldsymbol{r}$ (2 in this case). Thus, we can use this known distribution to exclude measurements that are extremely unlikely given our model and estimate. Excluding measurements with probability less than 0.001 corresponds to throwing out any measurements with $d(\boldsymbol{r}, \boldsymbol{S}) > \texttt{chi2inv}(0.999, 2)$ where `chi2inv`(.999,2) is the inverse of the $\chi^2(2)$ cdf at $p = 0.999$, or about 13.8.

2.3 LIDAR update

Finally we will complete the implementation by incorporating the laser measurements. The measurement model and Jacobians were given or derived in question 1.3 above.

2.3.1 Range and bearing measurement model

Implement the range and bearing measurement model given above, as well as its Jacobian that you derived, in the function `laser_measurement_model`.

2.3.2 Landmark initialization

Implement the function `initialize_landmark` that initializes a new landmark in the state vector from a measurement $\boldsymbol{z}$. You'll need to use the inverse of the measurement function, $h^{-1}(\boldsymbol{z})$, and don't forget you also need to expand the covariance matrix $\boldsymbol{P}$ to match the newly expanded $\boldsymbol{x}$.

2.3.3 Data association

Implement the function `compute_data_association` that computes the measurement data association as discussed above. Given an input state $(\boldsymbol{x}, \boldsymbol{P})$ and set of measurements $\{z\}$, the function should return an array `assoc` such that `assoc[j] == i` if measurement j was determined to be associated with landmark i. Furthermore there are two special cases: if `assoc[j] == -1`, measurement j was determined to be an observation of a new landmark requiring initialization; if `assoc[j] == -2`, measurement j was determined to be too ambiguous to use and should be completely thrown out.

For given measurements $\boldsymbol{z}_k$ and landmarks $\boldsymbol{f}_j$, the Mahalanobis distance between $\boldsymbol{z}_k$ and the expected measurement of $\boldsymbol{f}_j$, $h(\boldsymbol{x}, \boldsymbol{f}_j)$, is a useful metric to determine how likely it is that the measurement is an observation of that feature. The residual covariance $\boldsymbol{S}$ that you will use in this calculation is computed the same as it is in the EKF update step, and the Mahalanobis distance then has the same form $d = \boldsymbol{r}^T \boldsymbol{S}^{-1} \boldsymbol{r}$. Recall from the above discussion of this metric that d follows a known χ^2 distribution.

You may find the function `slam_utils.solve_cost_matrix_heuristic` useful here. Given a matrix of costs $\boldsymbol{M}$, where m_{ij} is thought of as the cost of assigning some entity i to some entity j, this function attempts to find a minimum total cost matching such that each i is assigned to some j. It returns a list of matching pairs (i, j). This function is heuristic and suboptimal but runs efficiently and should be good enough for this use case; the optimal method known as the Hungarian or Munkres algorithm can be used instead.

You will probably need to disregard ambiguous measurements, both in the sense of those that could be matched to two known landmarks with similar cost, and in the sense of those that are on the border between being matched to a known landmark and being initialized as a new landmark. The known distribution of d will be useful here.

2.3.4 Laser update

Finally, implement the EKF update from tree detections in the function `laser_update`. After implementing the laser updates, it's important that you verify that the filter functions correctly with or without GPS updates enabled.

2.4 Miscellaneous advice

1. Any time you modify the covariance matrix $\boldsymbol{P}$, numerical issues may cause it to become slightly non-symmetric. It is thus useful to assign $\boldsymbol{P} \leftarrow \frac{1}{2}(\boldsymbol{P} + \boldsymbol{P}^T)$ after each such update, or call the function `slam_utils.make_symmetric`.

2. Similar to the above, small numerical issues may cause the eigenvalues of $\boldsymbol{P}$ to become slightly negative over time. If you observe this happening (and only if), it may be remedied by adding a small constant times the identity to $\boldsymbol{P}$ to increase its eigenvalues by the same amount: $\boldsymbol{P} \leftarrow \boldsymbol{P} + \kappa \boldsymbol{I}$ for small κ. See the function `slam_utils.plot_covariance` where we set $\boldsymbol{P} \leftarrow \boldsymbol{P} + \left(\lambda_{min}(\boldsymbol{P}) + 10^{-6}\right)\boldsymbol{I}$ if necessary in order to take the Cholesky decomposition.

3. Due to the periodic nature of SO(2) and the angle ϕ, it is important to project the angle back into the range $[-\pi, \pi)$ after each operation. The function `slam_utils.clamp_angle` can do this for you.