FINAL PROJECT – Due June 13
EBS 221: Agricultural Robotics

A tree nursery has a field block with *K* rows of trees aligned in a North-South direction. Each row is 20 m long and spaced 3 m apart. The southern midpoint of the western row is located at 20 m, 20 m. The trees vary in age, indicated by trunk diameters. Some trees may be absent from the row due to being sold or lost to disease or other damage. You need to program a mobile robot equipped with GPS, a digital compass, and a 2D laser scanner to traverse the field block and *count the number of trees in each row*. Additionally, you need to *estimate the diameter of* each tree's trunk so that management can determine which trees are ready for the market.

MODULES (see Appendix)

1) A Matlab function named *robot_odo.p,* provided by the instructor, models a mobile robot as a bicycle model with first-order closed-loop dynamics for steering and speed. The function integrates the model over time DT with an internal integration interval dt (global variables) and returns the new state as a vector [x y θ γ v], along with a noisy estimate of the odometry, represented by the vector $\boldsymbol{\delta}_k + \boldsymbol{v}= [\delta_d+v_d \; \delta_\theta+v_\theta]$.
2) The Matlab function *LaserScannerNoisy.p,* provided by the instructor, returns range measurements $\boldsymbol{l}_k$ and noise, $\boldsymbol{n}$. The noise consists of a white Gaussian component and spike-type noise (resulting from laser reflections). When a reading is out of range, it returns Inf.
3) The MATLAB function, *GPS_CompassNoisy.p,* is provided by the instructor. This function accepts the true pose of your robot as an argument and returns a noisy pose, $\boldsymbol{x}_k + \boldsymbol{w}$, simulating the behavior of real sensors. Additionally, the GPS and compass provide new measurements every second. However, the control system functions at a rate of DT = 10 ms, which means that you should call this function once every 100 control intervals.
4) The Matlab function, *generateNursery.m* generates a 2D bitmap with trees ("real world"/ground truth). This function calls *draw_disc.m*, which is also provided. Your robot is not allowed to access this bitmap directly, but ONLY through its LiDAR sensor. The bitmap is provided for testing, and you are encouraged to test your code with your own noisy test bitmaps (see EVALUATION).

TASKS:

1) Estimate the odometry error covariance matrix (2x2), and the GPS and compass error covariance matrix (3x3). You can use the true states to do so.
2) Implement an EKF filter for robot state estimation using the odometry model described in class for prediction and the GPS-compass sensor. Additionally, develop a filter for the laser. This is necessary because laser measurements can contain spikes, so it is recommended to apply a median filter to the range data before feeding it to the EKF.

3) Develop a Matlab function that updates an occupancy grid with R rows and C columns, covering a physical area defined by [0-Xmax, 0-Ymax]. This function should use the measurements [angle range] from a 2D LiDAR and the pose of the LiDAR (transform matrix Tl). You may utilize the function updateLaserBeamBitmap(angle, range, Tl, R, C, Xmax, Ymax) provided on Canvas. Rename it (for example, to updateLaserBeamGrid(angle, range, Tl, R, C, Xmax, Ymax)) and integrate the update equations for all pixels covered by the LiDAR sensor.
4) Traverse the orchard block (access the bitmap only through your sensors) and build an occupancy grid.
5) Using the grid, detect and localize trees, and estimate their diameters.
6) Write a file with your output (see format below).
7) Write a script to produce your estimate of the perceived environment (bitmaps) from the output file.
8) Write a script to calculate error histograms of tree positions and diameters between a test bitmap (ground truth synthetic environment) and the output of task 5 (perceived environment). Extract descriptive statistics from histograms and report them: mean, standard deviation, RMS, min, max, 95th percentile.

OUTPUT

After the robot finishes its travel, the output should be a file that looks like this: Row number followed by index of each tree detected, tree coordinates (x, y) and diameter (d). E.g.:

1
1, x1, y1, d1
2, x2, y2, d2
3, x3, y3, d3
N, x_N, y_N, d_N

2
1, x1, y1, d1
2, x2, y2, d2
3, x3, y3, d3
M, x_M, y_M, d_M

K
1, x1, y1, d1
2, x2, y2, d2
3, x3, y3, d3
M, x_M, y_M, d_M

ROBOT

You have an Ackermann-steered vehicle with a wheelbase of L = 3 m, a width of 2 m, and a maximum steering angle of 55 degrees. The steering lag is 0.1 seconds, and the velocity lag is also 0.1 seconds. The robot is initially positioned at (0,0), facing 90 degrees North. You can set your own robot speed.
The 2D lidar has a maximum range of 20 meters, a scanning angle of 180 degrees, and an angular resolution of 0.125 degrees. You must use the true pose ONLY for vehicle motion and sensor simulation. All of your control, sensing, and processing algorithms must rely ONLY on noisy data from the odometry, lidar, and GPS-compass sensors.

EVALUATION
To evaluate your code, I will use it with my own nursery grids. **Beware**: Tall grass or twigs near the ground may manifest themselves as speckles (isolated pixels) of noise in the grid that I will use! Simulate this to test your algorithm's robustness. Run several simulations to make sure the software works even when trees are missing.

APPENDIX

```
function [q_true_next, odo] = robot_odo(q_true, u, umin, umax,Qmin,
Qmax, L, tau_gamma, tau_v)
% model of a vehicle with closed loop steering and velocity control
% the combined effect of steering/vehicle inertia and control is
% a first order system for steering (tau_phi) and velocity (tau_v)
% state is
% q(1) -> x
% q(2) -> y
% q(3) -> theta (orientation in world frame)
% q(4) -> gamma (steering angle)
% q(5) -> linear velocity

% inputs are:
% u(1) -> desired steering angle
% u(2) -> desired linear velocity
%
% the model returns the next state q1 and noisy odometry, i.e.,
% distance traveled in DT in odo(1) and angle change in DT in odo(2).

function p = laserScannerNoisy(angleSpan, angleStep, rangeMax, Tl, map,
Xmax, Ymax)
% function assumes a laser scanner with a pose in world coordinates
% defined by Tl (from q_true). It shoots rays from
% -angleSpan/2 to +angleSpan/2 with step angleStep.
% Given a map (occupancy grid with obstacles) with origin at (0,0) and
% upper corner (Xmax, Ymax), the result is ray angles (p(:,1)) and
% noisy ranges (p(:,2)).

function [ x_n, y_n, theta_n ] = GPS_CompassNoisy( x, y, theta )
% function accepts as arguments the true pose of the robot (from
q_true) and returns a noisy measurement of the pose. Angle is in
radians.
```