Final Project

Rayne Milner
EBS 298k - Sensors and Actuators

December 31, 2019

1 Summary

Detailed in this report is a comprehensive solution to real world problem. The ultimate goal is to traverse an orchard using simulated sensors, and with a LIDAR sensor, detect the diameters of trees in this orchard. The project was successful in implementing a extended kalman filter to traverse the orchard block, and imaging the trees. However, work needs to be done on the computer vision algorithms to accurately detect the tree sizes.

2 Discussion of Code

The main script for this assignment was FinalProject.m this script called all sub-functions and scripts. Robot odometery and LIDAR data was simulated using supplied functions (further discussion in Section 2.3.1).

2.1 Computing Optimal Path

Once the way-points, representing the two ends and the midpoint of each row in the orchard, were generated based on the dimensions of the field, a cost-matrix is produced that gives the "cost" of travelling from point to point in the way-point array. The costs were produced with trial and error, loosely relying on the distance between points and a few logical principles (e.g. if the robot has started down a row, then there should be no cost associated with completing that row). A genetic algorithm then solves the travelling salesman problem to find a locally optimal path to traverse the orchard.

2.2 Creating Path

Once the optimal traversal through the way-points is computed. Then a path is created, using only straight line and circular arc segments (circular and omega turns only). This path is computed such that the vehicle with the given turning radius and dimensions may follow it.

2.3 Pure Pursuit

A simple pure-pursuit algorithm was used to follow the produced path. The main parameter of this function are the "look ahead distance" which determines haw far ahead in the path the robot aims for. As expected, increasing the look-ahead distance makes the path smoother, however, this also results in cutting-corners and other undesirable effects. A look-ahead distance of 2m seemed to produce good results.

2.4 Odometery and Filtering

The first step to use an extended Kalman filter is to find the covariance of noise present in the odometery and in the sensors. In a real situation this would be experimentally found by comparing the true values (in this case, of position) to the estimated or sensed values.

In this simulation, the true and noisy values are supplied by simulated odometery and sensor functions unknown to the user. In order to get the values of covariance, a Bayesian approach is taken.

2.4.1 robotodo.p

The robot odometery was simulated using the hidden function robotodo.p. However, this function produced bad data and had to be modified. The given function was modified to give correct results: on line 48 the limits of the for loop was changed to $t = 0 : dT : DT - dT$. Line 77 was changed to $dth = qk(3) - q0(3);$. In order to successfully run the program, the user must put robot_odo.m in the path and remove robot_odo.p

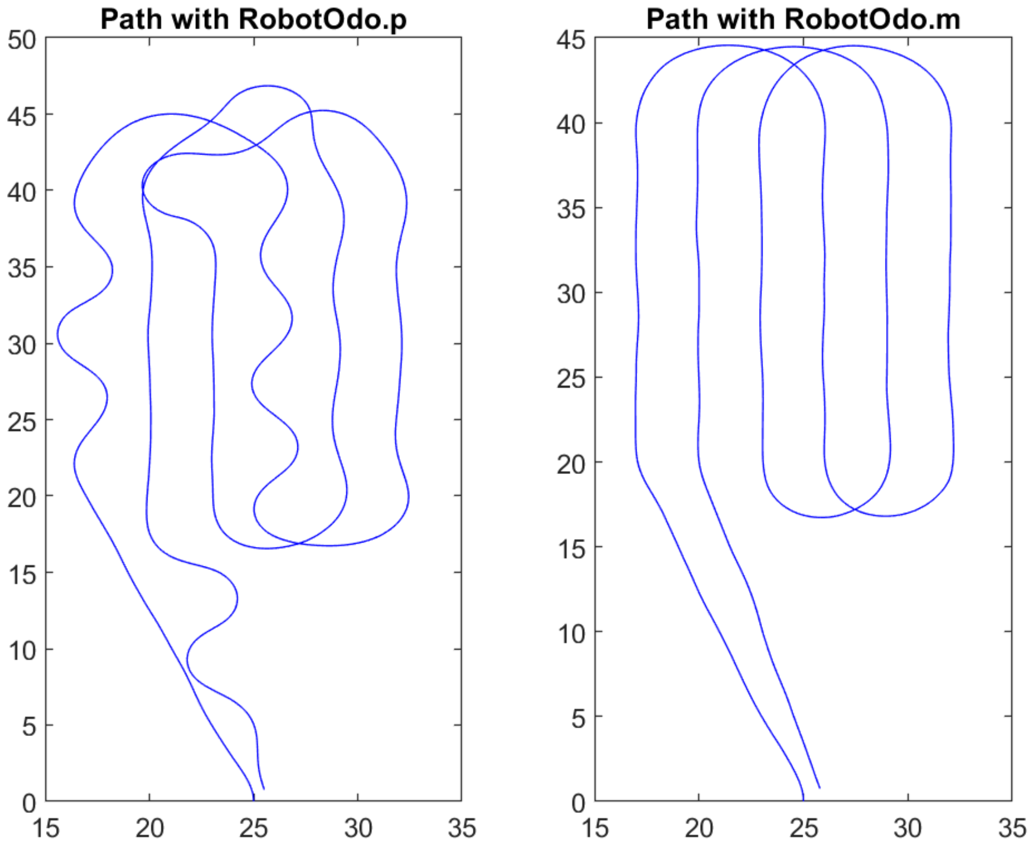


Figure 1: Comparison of supplied function with modified function.

2.5 Tree Scanning

Once the navigational elements are simulated, the program performs its main function, computing the diameter of tree sizes in the orchard. This is performed using a simulated noisy LIDAR. By iterating through the path points previously calculated and performing a 90 degree scan at each point, an occupancy grid is updated which contains the location of each tree in the orchard. A built-in Matlab Function to find circles is applied to this data and, finally, the data is processed to output a location of each tree and it's estimated diameter.

3 Covariance Matrices

The covariance matrices are as follows:

$$\sigma^2_{odo} = \begin{bmatrix} 0.0025 & 0 \\ 0 & 1.14 * 10^{-4} \end{bmatrix}$$

$$\sigma_{GPS}^2 = \begin{bmatrix} 8.8084 * 10^{-4} & 0 & 0 \\ 0 & 9.5809 * 10^{-4} & 0 \\ 0 & 0 & 4.0491 * 10^{-4} \end{bmatrix}$$

4 Analysis

4.1 Path planning

The high-level planning presented here involves creating a array of nodes based on the known geography of the orchard, computing a optimal path through the nodes and then generating a series of closely spaced path points for the robot to follow.

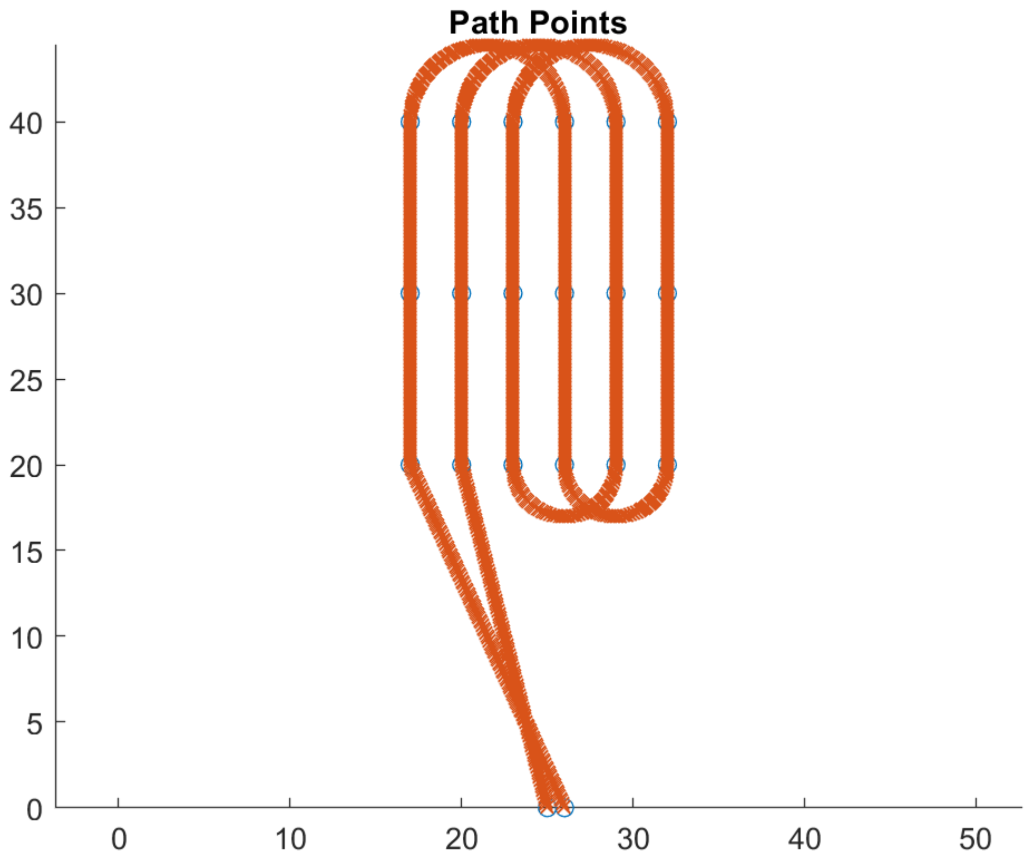


Figure 2: Array of path points generated to navigate the block.

4.2 EKF and PurePursuit

The lower level navigation relies on an extended kalman filter and a purePursuit controller. The loop that performs the kinematic calculations has several different sections that are executed at different rates. The EKF portion of this loop is only performed every one second, when GPS data becomes available. While the odometery and controller continues to be executed at every control interval. We can see that the EKF and controller works very well to follow the path, deviating from ground truth by only a few centimeters.

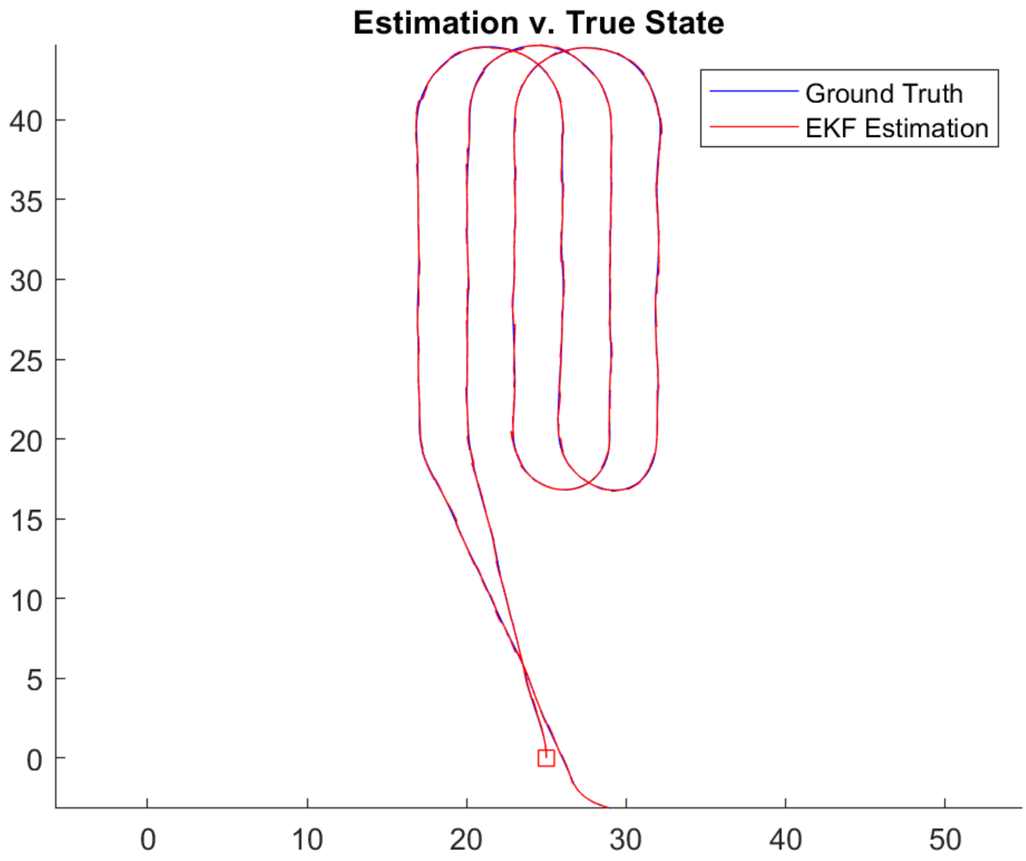


Figure 3: Trace of true pose and estimated pose

The behavior of the Kalman Filter is shown below. The estimate of pose can be seen updating every time the GPS updates, after which the odometery integrates without GPS data. During the period without GPS data the odometery becomes more erroneous until the EKF is updated again.

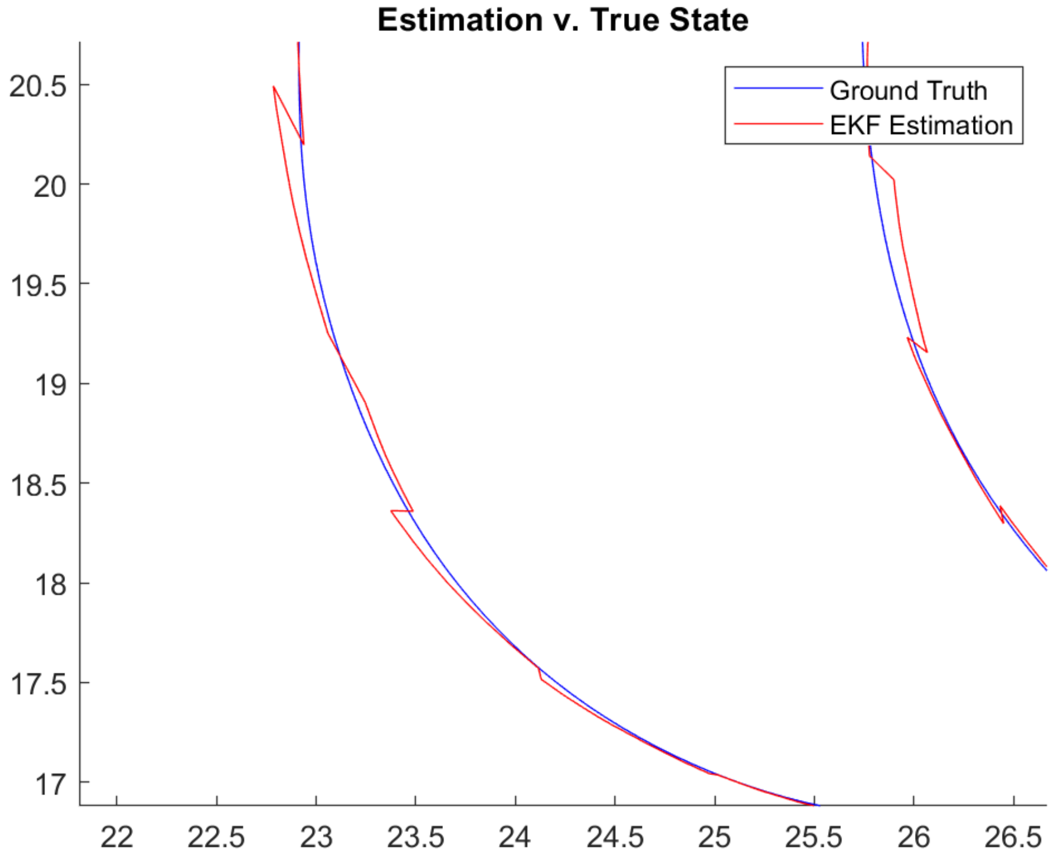


Figure 4: Enlarged image showing how estimate of pose updates

4.3 Tree Detection

The tree detection algorithm runs after the kinematic calculations are completed. The LIDAR data is collected from a series of points saved during kinematic stage. The noisy LIDAR data is filtered using a median filter with a range of three pixels (meaning this only filters outliers that are one pixel in size). The median filter as not tested with simulated spike-type noise.

The LIDAR iteratively creates the occupancy grid for the orchard. This occupancy grid is then post processed to detect edges, and converted into a black and white image. finally the function imfindcircles is run on the processed image to find the locations of the trees. The output of the LIDAR data and image processing is shown below.

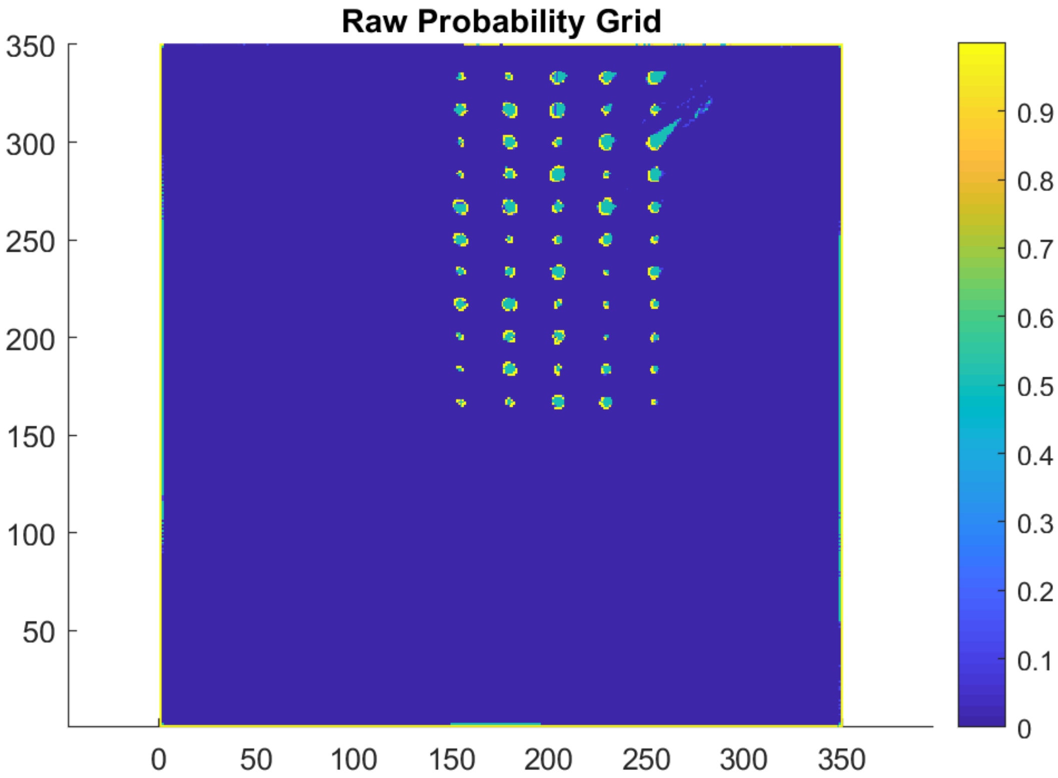
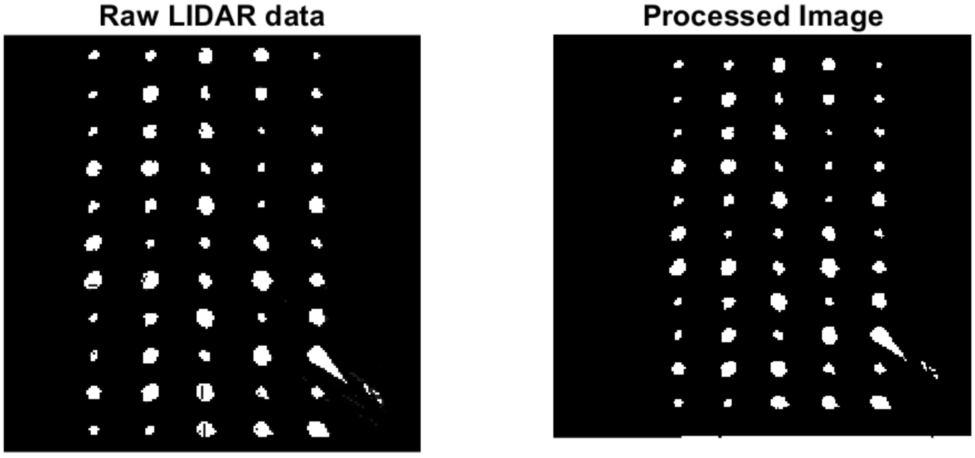


Figure 5: Probability of occupancy for each pixel, obtained from LIDAR detection on bitmap



Raw LIDAR data

Processed Image

Figure 6: Comparison of raw LIDAR data to Processed image

Finally, below, we see how the circle algorithm detects the trees. The algorithm has trouble detecting circles of small radius. This could possibly be improved by increasing the density of the grid.

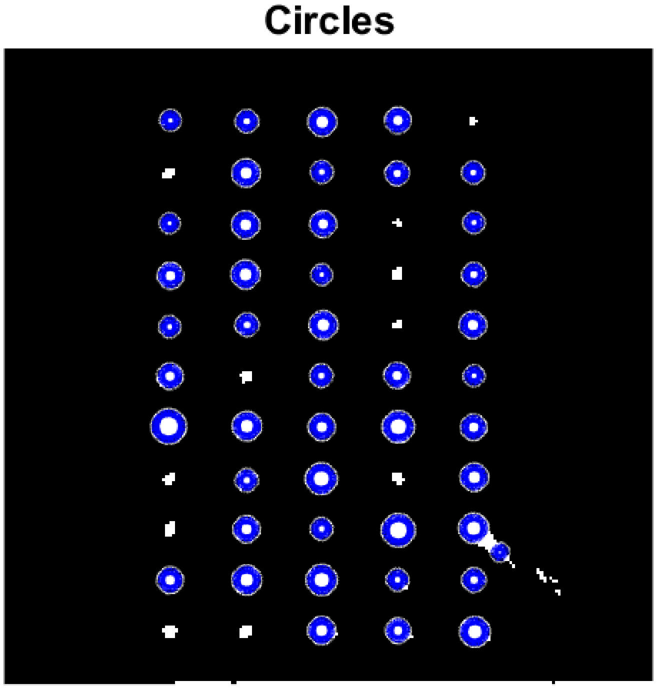
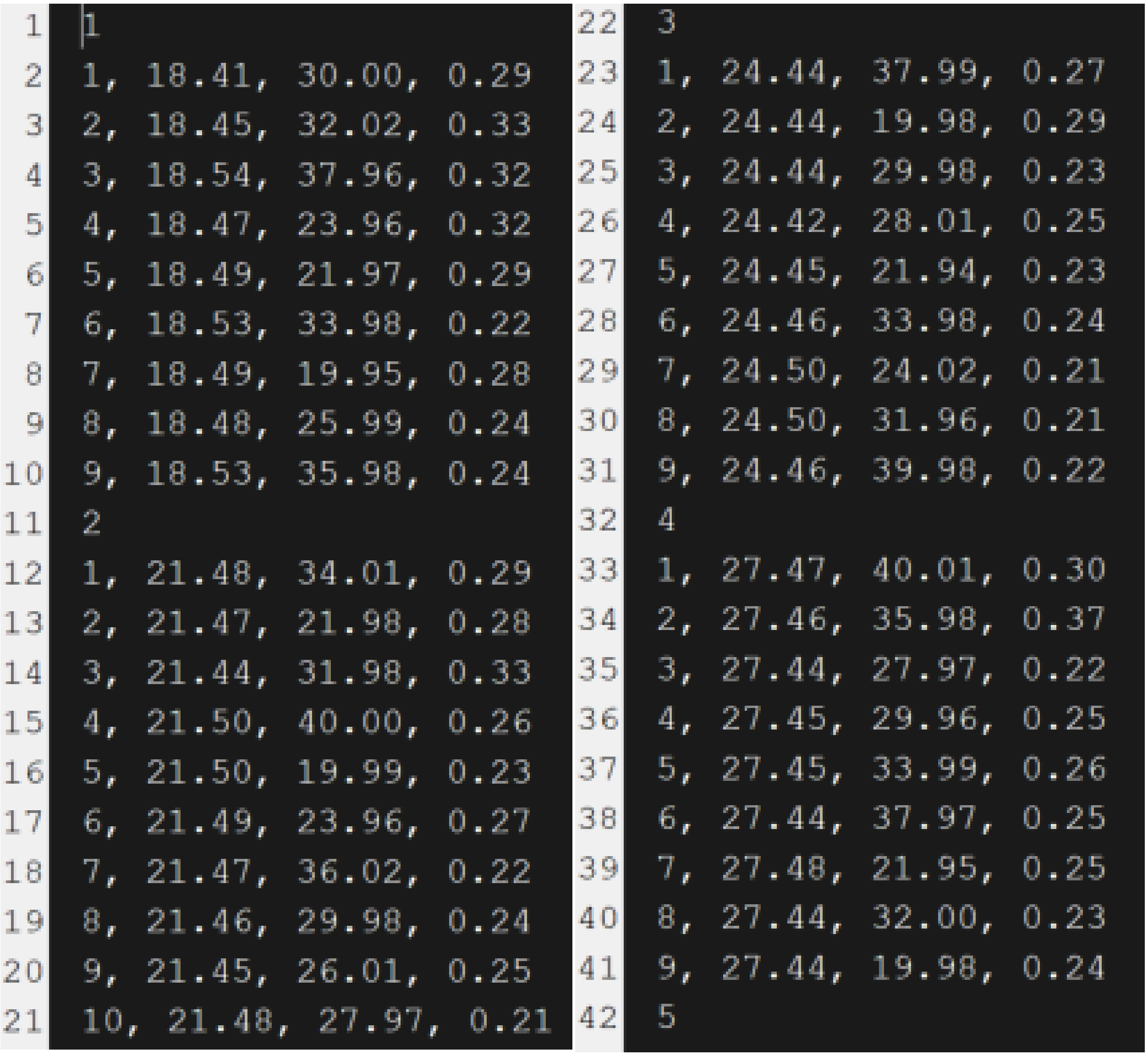


Figure 7: Circle detection plot

4.3.1 Output File

The script outputs as text file formatted according to problem specifications. This file shows the estimated x and y coordinates of the trees in each row, as well as the estimated radius of the tree. A sample image of the text file is shown below



```
1
1, 18.41, 30.00, 0.29
2, 18.45, 32.02, 0.33
3, 18.54, 37.96, 0.32
4, 18.47, 23.96, 0.32
5, 18.49, 21.97, 0.29
6, 18.53, 33.98, 0.22
7, 18.49, 19.95, 0.28
8, 18.48, 25.99, 0.24
9, 18.53, 35.98, 0.24
2
1, 21.48, 34.01, 0.29
2, 21.47, 21.98, 0.28
3, 21.44, 31.98, 0.33
4, 21.50, 40.00, 0.26
5, 21.50, 19.99, 0.23
6, 21.49, 23.96, 0.27
7, 21.47, 36.02, 0.22
8, 21.46, 29.98, 0.24
9, 21.45, 26.01, 0.25
10, 21.48, 27.97, 0.21
3
1, 24.44, 37.99, 0.27
2, 24.44, 19.98, 0.29
3, 24.44, 29.98, 0.23
4, 24.42, 28.01, 0.25
5, 24.45, 21.94, 0.23
6, 24.46, 33.98, 0.24
7, 24.50, 24.02, 0.21
8, 24.50, 31.96, 0.21
9, 24.46, 39.98, 0.22
4
1, 27.47, 40.01, 0.30
2, 27.46, 35.98, 0.37
3, 27.44, 27.97, 0.22
4, 27.45, 29.96, 0.25
5, 27.45, 33.99, 0.26
6, 27.44, 37.97, 0.25
7, 27.48, 21.95, 0.25
8, 27.44, 32.00, 0.23
9, 27.44, 19.98, 0.24
5
```

Figure 8: Text file output

4.4 Error Analysis

Below we see histogram plots of the error in both radius and distance. The error statistics for radius (shown on the image) show that the estimation is consistently under estimating the radius of the tree by 0.11m meter on average. This is significant considering the radius of the trees are somewhere between 0.2m and 0.5m. However, we can see that the standard deviation of error is an order of magnitude lower, 0.03m, meaning that the algorithm is more precise than it is accurate. We could conceivably subtract the offset from our estimation of radius and thereby consistently estimate the radius of the tree.

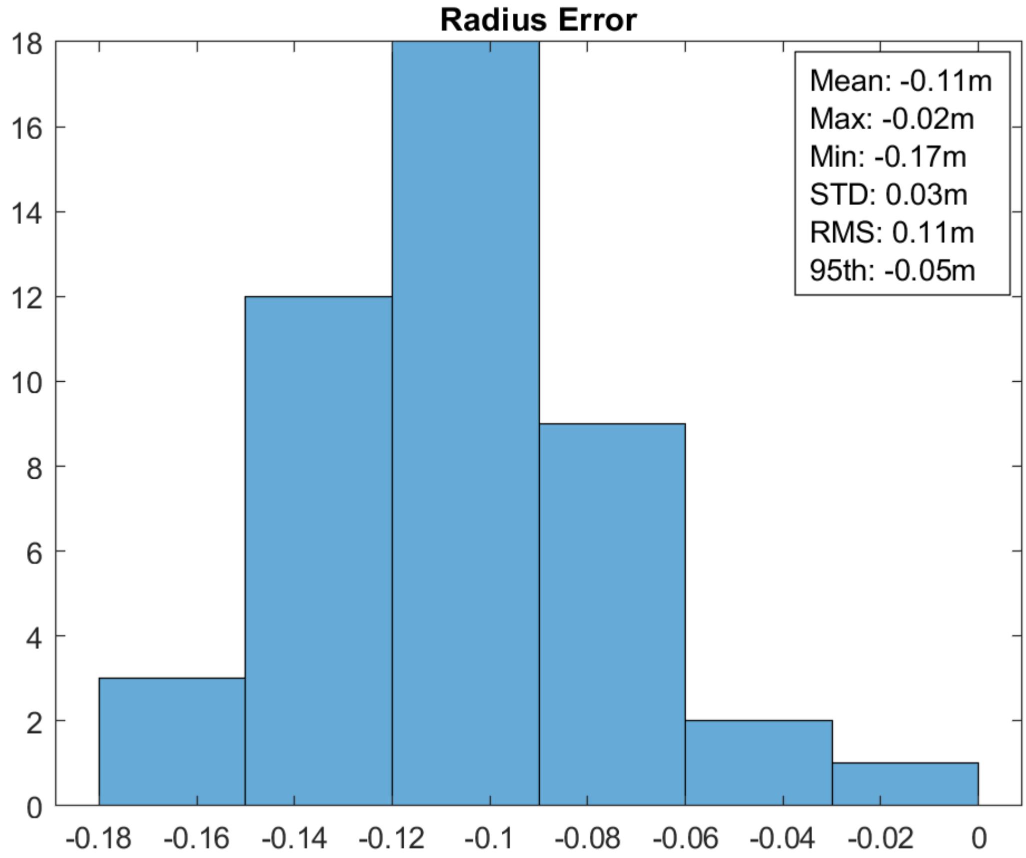


Figure 9: Error in radius

Looking at the histogram for distance error, we see that the algorithm is both precise and accurate, both mean and STD are on the order of 0.05m.

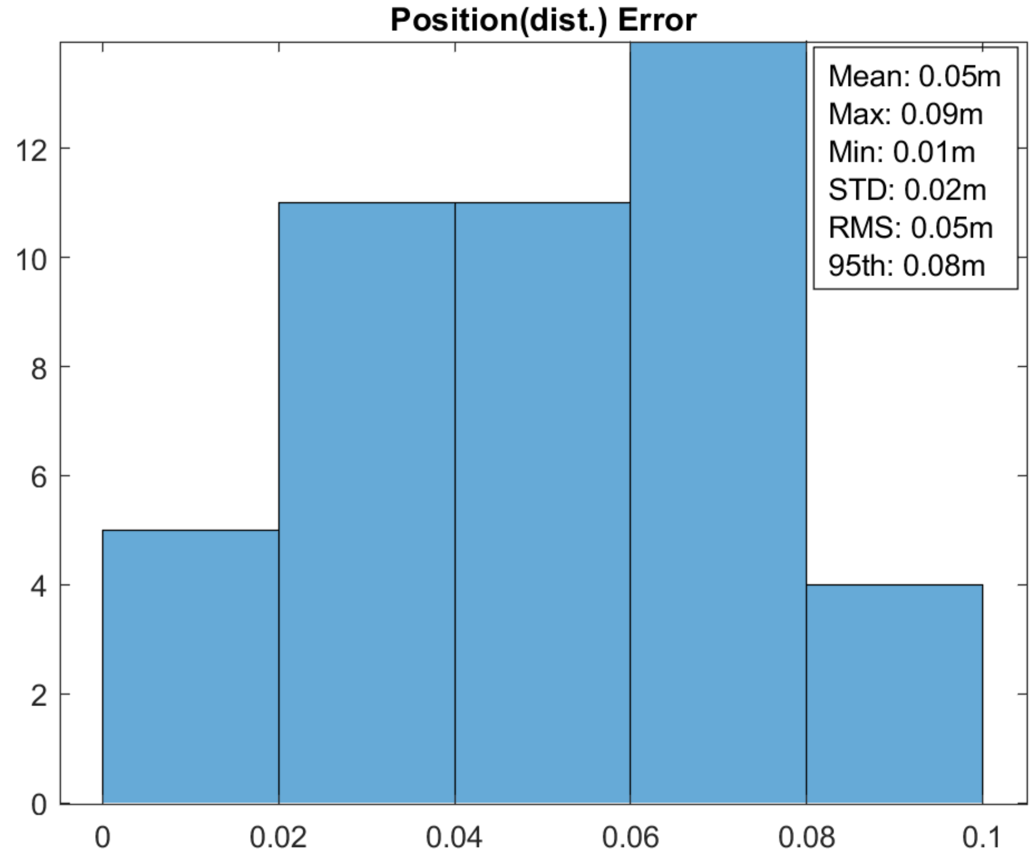


Figure 10: Error in distance

5 Appendix A: Matlab Code

Table of Contents

FinalProject.m

This script performs the orchard navigation using an EKF for state esimation. This script produces a series of points that represent poses of the robot, which are later fed to another script that performs tree localization

```
clearvars;

%Declare global variables.
global bitmap points rangeMax occuGrid probGrid dT DT DTgps occuGrid
 rangeMax bitpos trees_percieved;
```

Generate nursery

replace the following line with some other script call for testing purposes.

```
generateNursery(); %creates a random nursery, saving tree locations to
 bitmap global variable
bitmapinput = bitmap; % saves bitmap to a input variable for laser
 scanner
```

Variable Set-up

```
%simulation parameters
dT = 0.01; %Euler integration timestep
DT = 0.1; %Controller sampling timestep
DTgps = 1; %sampling time of GPS
T = 230; %simulation time
```

```
steps = T/DT; %number of steps in simulation
gpssamplerate = DTgps/DT; %ratio of gps sampling to controler interval

% Pre-computed variance values
varianceOdo = [0.0025 1.1420e-04]; %varaince in noise from odometery
 (distance, theta)
varianceGPS = [8.8084e-04 9.5809e-04 4.0491e-04]; %variance in noise
 for x from gps (x, y, theta)

%Space variables
Xmax = 50; Ymax = 50; %physical dimensions of space (m)
Xmax_nurs = 42; % xmax in the generateNursery script
Ymax_nurs = Xmax_nurs; % xmax in the generateNursery script
R = 350; C = 350; %rows and columns; discretization of physical space
 in a grid
q_true = zeros(5, steps); % state vectors over N time steps
q = zeros(5, steps); % state vectors over N time steps for kinematic
 model
u = zeros(2, steps); % input vectors over N time steps
e = zeros(1, steps); %cross track error
odo = zeros(3, steps); %odometery values over N time steps
occuGrid = ones(R, C); %initialize as odds of one for each pixel
probGrid = 0.0* ones(R, C); %initialize as empty

%Variance computation arrays
x_diff_gps = zeros(1, steps); %holds difference b/w gps data and q
 true for x
y_diff_gps = zeros(1, steps); %holds difference b/w gps data and q
 true for y
theta_diff_gps = zeros(1, steps); %holds difference b/w gps data and q
 true for angle
d_diff = zeros(1, steps); %holds difference b/w odometery data and q
 true for distance
theta_diff = zeros(1, steps); %holds difference b/w odometery data and
 q true for theta

%input and state constraints
Vmax = 4; %velocity max
gmax = 55*(pi/180); %turning angle max

%vehicle parameters
l = 3; %m, distance between axis
Rmin = l/tan(gmax); %minimum turning radius
width = 2; %width of vehicle

%Feild parameters
rowWidth = 3;%spacing between tree rows
Krows = 5; % number of rows
N = Krows+1; % k plus 1 (so that robot encircles orchard)
len = N*rowWidth; %transverse to rows
RL = 20; %lateral length
nTrees = 11; %number of trees in the row
Treespacing = RL/(nTrees-1); %spacing between trees down the row
nStart = [17;20]; %one half row width to the left of the SW tree
```

```
trees_percieved = zeros((Krows*nTrees),3); %arry that holds the values
 of trees percieved for error calculation

%Path variables
nd = zeros(2,3*N+2); %nd for path
x = nd(1,:); %x coordinates of node points
y = nd(2,:); %y coordinates of node points
spacing = 0.1; %m, spacing between points on the path

%initial pose of the robot
q_true(:,1) = [25,0,pi/2,0,0];
q(:,1) = [25,0,pi/2,0,0];
odo(:,1) = [25,0,pi/2];

%laser scanner variables
points = [q_true(1,1), q_true(2,1), q_true(3,1)];
rangeMax = 50; angleSpan = pi; angleStep = angleSpan/720;

%Non-Ideal effects
delta1 = 0*pi/180; delta2 = 0*pi/180; s = 0.0; %slip and skid
tau_g = 0.1; %time-lag for turning angle
tau_v = 0.2; %time-lag for velocity

%create constraint vectors
Qmax(1) = inf; Qmax(2)=inf; Qmax(3) = inf; %state constraint
Qmax(4) = gmax; Qmax(5) = Vmax;%state constraint
Qmin = -Qmax; %minimum constraints.
Umax = [gmax Vmax]'; %input constraint
Umin= -Umax;%input constraint

%Pursuit parameters
Ld = 2.0; %lookahead distance

%Cost Matrix
DMAT = zeros(2*N+1,2*N+1);

%Kalman Filter parameters
Ks = 1.1; %steering angle gain
tauFilter = 0.5; %time constant of low pass filter

wTr = eye(3,3); %3x3 Matrix, transforamtion to the origin
```

Figure Setup

```
close all;

figure(1);hold on; axis equal; title('Path Points');

figure(2); hold on; axis equal; title('Estimation v. True State');
```

Build Tractor

```
Tractor = BuildTractor();
```

```
T1 = transl2(q_true(1,1),q_true(2,1))*trot2(q_true(3,1)); %intial
 position

figure (2)
plotTractor(Tractor,T1,'r'); %plots the tractor it the initial
 position
```

Compute Node Coordinates

```
nd(:,1) = q_true(1:2,1); %start node
nd(:,3*N+2) = q_true(1:2,1)+[1;0]; %end node

for i = 2:N+1 %bottom row
        nd(:,i) = [nStart(1)+(i-2)*rowWidth nStart(2)+0];
end
for i = N+2:2*N+1 %middle row
        nd(:,i) = [nStart(1)+(i-(N+2))*rowWidth nStart(2)+RL/2];
end
for i = 2*N+2:3*N+1 %top row
        nd(:,i) = [nStart(1)+(i-(2*N+2))*rowWidth nStart(2)+RL];
end
figure (1)
plot(nd(1,:),nd(2,:),'o'); %plot nd
```

Generate Cost Matrix

```
%non headland costs
for i = 2:N+1 %bottom row
    for j = N+2:2*N+1 %middle row
        if (j-i) == N %if they belong to same row
            DMAT(i,j) = 0; %we make this negative in order to "reward"
 the algorithm
            DMAT(j,i) = 0;
        else %if they belong to different rows
            DMAT(i,j) = 10^12;
            DMAT(j,i) = 10^12;
        end
    end

    for j = 2*N+2:3*N+1 %top row
        DMAT(i,j) = 10^12;% we dont want to go from bottom to top
        DMAT(j,i) = 10^12;
    end
end

for i = N+2:2*N+1 %middle row
    for j = 2*N+2:3*N+1 %top row
       if (j-i) == N %if they belong to same row
            DMAT(i,j) = 0; %we make this negative in order to "reward"
 the algorithm
            DMAT(j,i) = 0;
        else %if they belong to different rows
            DMAT(i,j) = 10^12;
```

```
            DMAT(j,i) = 10^12;
        end
    end
end

% Headland Turning Costs
for i=2:N %bottom row
    for j=i+1:N+1 %node to the right
        d = abs(i-j); %distance between nodes

        if 2*Rmin > d*width % an omega turn
            %cost to do a turn is the length of the maneuver
            DMAT(i,j) = ((3*pi*Rmin-2*Rmin*acos(1-(2*Rmin+d*width)^2/
(8*Rmin^2)))); %length of omega turn
        else %Pi turn
            DMAT(i,j) = ((d*width+(pi-2)*Rmin)); %length of pi turn
        end
     %symmetry conditions for top row
     DMAT(j,i)=DMAT(i,j);
     DMAT(i+2*N, j+2*N) = DMAT(i,j);
     DMAT(j+2*N, i+2*N)= DMAT(i,j);
    end
end

% Start and End nodes
for i=2:3*N+1

    if (i>1 && i<N+1)
        DMAT(1,i) = -100;
        DMAT(3*N+2,i) = -100;
    else
        DMAT(1,i) =  abs(x(1)-x(i)) + abs(y(1)-y(i)); %manhattan
 distance
        DMAT(3*N+2,i) = abs(x(2*N+2)-x(i)) + abs(y(2*N+2)-
y(i)); %manhattan distance
    end

    DMAT(i,1) = DMAT(1,i); % cost matrix symmetry
    DMAT(i,3*N+2) = DMAT(3*N+2,i); % cost matrix symmetry
end

%cost between start and end nd
DMAT(1,3*N+2) = 10^12;
DMAT(3*N+2, 1) = 10^12; % cost matrix symmetry
```

Compute Optimal Path

```
XY = [x' y']; t = cputime;
resultStruct = tspof_ga('xy', XY , 'DMAT',
 DMAT, 'SHOWRESULT',false, 'SHOWWAITBAR', false, 'SHOWPROG', false);
E = cputime - t; %time required to compute it.
rt = [1 resultStruct.optRoute 3*N+2]; % extract node sequence
resultStruct.minDist %print computed minimum distance
```

Create route for robot to follow

```
pathroute = nd(:,rt(1)); %the path the pursuit controller will follow

for i = 1:length(rt)
    pathroute1(:,i) = nd(:,rt(i)); %this route is just the path
 throught he nodes
end

% The following loop creates the actual path that follow the route
 given by
% the optimiser. This loop determines if omega turns, pi turns or
 stright lines are
%are to be made based on the index of the nodes and then creates the
 path
%between them

for i = 2:length(rt) %for every index in route

    distance = ((pathroute(1,end)-nd(1,rt(i)))^2+(pathroute(2,end)-
nd(2,rt(i)))^2)^(1/2); %euclid distance
    c = nd(:,rt(i))-pathroute(:,end); %vector from current to next
 node
    angledifference = atan2(c(2),c(1)); %angle between the nodes

    %series of booleans that return true if the current or last node
 is on
    %the top or the bottom
    isBottom = and(rt(i)>1,rt(i)<N+2); %true if current node is on
 bottom
    lastIsBottom = and(rt(i-1)>1,rt(i-1)<N+2); %true if last node is
 on bottom
    isTop = and(rt(i)>2*N+1,rt(i)<3*N+2); %true if current node is on
 top
    lastIsTop = and(rt(i-1)>2*N+1,rt(i-1)<3*N+2); %true if lsat node
 is on top

    %returns true if the current maneuver is a turn
    turn = or(and(isBottom,lastIsBottom),and(isTop,lastIsTop));

    if turn
        if (distance > 2*Rmin) %pi turn
            if isTop %if the nodes are on the top of the feild
                if c(1)>0 %if the turn is in positive direction
                    pathadd = createPath([pathroute(:,end);
((pi/2))],'circle',distance,pi,spacing);
                else %c(1)<0, turn is in negative direction
                    pathadd = createPath([pathroute(:,end);
((pi/2))],'circleback',distance,pi,spacing);
                end
            else %isBottom %if the nodes are on the bottom of the
 feild
                if c(1)>0 %if the turn is in positive direction
```

```
                    pathadd = createPath([pathroute(:,end);(-
(pi/2))],'circleback',distance,pi,spacing);
                else %c(1)<0 %if the turn is in negative direction
                    pathadd = createPath([pathroute(:,end);(-
(pi/2))],'circle',distance,pi,spacing);
                end
            end
            pathroute = [pathroute, pathadd.']; %adds the turn to the
 path

        else %omega turn
            gturn = acos(1 - (2*Rmin+distance)^2/(8*Rmin^2));
            aturn = (pi-gturn)/2;

            if isTop %if the nodes are on the top of the feild
                if c(1)>0 %if the turn is in positive direction
                    pathadd = createPath([pathroute(:,end);
(pi/2)],'circleback',Rmin*2,aturn,spacing);
                    pathroute = [pathroute, pathadd.'];
                    pathadd = createPath([pathroute(:,end);
((aturn)+pi/2)],'circle',Rmin*2,2*pi-gturn,spacing);
                    pathroute = [pathroute, pathadd.'];
                    pathadd = createPath([pathroute(:,end);(pi/2+aturn
+gturn)],'circleback',Rmin*2,aturn,spacing);
                    pathroute = [pathroute, pathadd.'];

                else %c(1)<0, turn is in negative direction
                    pathadd = createPath([pathroute(:,end);
(pi/2)],'circle',Rmin*2,aturn,spacing);
                    pathroute = [pathroute, pathadd.'];
                    pathadd = createPath([pathroute(:,end);(-
(aturn)+pi/2)],'circleback',Rmin*2,2*pi-gturn,spacing);
                    pathroute = [pathroute, pathadd.'];
                    pathadd = createPath([pathroute(:,end);(pi/2-
aturn-gturn)],'circle',Rmin*2,aturn,spacing);
                    pathroute = [pathroute, pathadd.'];

                end
            else %isBottom %if the nodes are on the bottom of the
 feild
                if c(1)>0 %if the turn is in positive direction
                    pathadd = createPath([pathroute(:,end);(-
pi/2)],'circle',Rmin*2,aturn,spacing);
                    pathroute = [pathroute, pathadd.'];
                    pathadd = createPath([pathroute(:,end);-
((aturn)+pi/2)],'circleback',Rmin*2,2*pi-gturn,spacing);
                    pathroute = [pathroute, pathadd.'];
                    pathadd = createPath([pathroute(:,end);-
(pi/2+aturn+gturn)],'circle',Rmin*2,aturn,spacing);
                    pathroute = [pathroute, pathadd.'];

                else %c(1)<0 %if the turn is in negative direction
                    pathadd = createPath([pathroute(:,end);(-
pi/2)],'circleback',Rmin*2,aturn,spacing);
```

```
                    pathroute = [pathroute, pathadd.'];
                    pathadd = createPath([pathroute(:,end);((aturn)-
pi/2)],'circle',Rmin*2,2*pi-gturn,spacing);
                    pathroute = [pathroute, pathadd.'];
                    pathadd = createPath([pathroute(:,end);(-
pi/2+aturn+gturn)],'circleback',Rmin*2,aturn,spacing);
                    pathroute = [pathroute, pathadd.'];
                end
            end

        end

    else %just creates a stright line to the next node
        pathadd =
 createPath([pathroute(:,end);angledifference],'line',distance,0,spacing);
        pathroute = [pathroute, pathadd.'];
    end

end

figure(1)
plot(pathroute(1,:),pathroute(2,:),'-x')
```

Begin odometery and filtering

```
V = diag(varianceOdo); %odometery variance matrix, calculated
 previously
W = diag(varianceGPS); %GPS variance matrix, calculated previously
Hx = eye(3); %jacobian, here just idetiy matrix
Hw = eye(3);%jacobian, here just idetiy matrix
P = zeros(3); % a priori assumption of covariance matrix (zero b/c we
 know initial state perfectly)

k=1; %counter for time
%%Navigation Open Loop Controller
for t=0:DT:T-2*DT

    %Jacobians for linearization
    Fx = [1 0 -1*odo(1,k)*sin(odo(3,k)); 0, 1, odo(1,k)*cos(odo(3,k));
 0 0 1];
    Fv = [cos(odo(3,k)) 0; sin(odo(3,k)) 0; 0 1];

    if (q_true(2,k) < 1 && wrapTo2Pi(q_true(3,k)) > 1.2*pi) %makes
 sure the robot doesn't leave 1st quadrant
        u(2,k) = 0;
    else
        u(2,k) = 1; %desired velocity
    end

    dist = ((pathroute(1,:)-q_true(1,k)).^2 + (pathroute(2,:)-
q_true(2,k)).^2).^(1/2); %calculates distance to each point in path
    [Mc,Ic] = min(dist);
```

```
    %current path is the set of path points that is passed to the
    %purepursuit controller
    currentpath = pathroute(:,fix(max((min(u(2,k),Vmax)*t/
spacing-1/spacing),1)):fix(min(min(u(2,k),Vmax)*t/spacing+1*Ld/
spacing,length(pathroute)))).';

    %calls purePursuit controller using estimate of position
    [kappa,error] = purePursuitController(odo(:,k),l,Ld,currentpath);

    u(1,k) = kappa; %desired steering angle

    if mod(k-1,gpssamplerate) == 0 %every get GPS data and perform EKF
 calculations

        [x_n, y_n, theta_n] =
 GPS_CompassNoisy(q_true(1,k),q_true(2,k),q_true(3,k)); %retrieves GPS
 data

        %calls kinematic model, which returns true state and noisy
 odometry
        [q_true_next, odo_next] = robot_odo(q_true(:,k), u(:,k), Umin,
 Umax, Qmin, Qmax, l, tau_g, tau_v);

        % Performs calculation to find covaraiance matricies
        x_diff_q = q_true_next(1)-q_true(1,k); %difference in x
 position since last step
        y_diff_q = q_true_next(2)-q_true(2,k); %difference in y
 position since last step
        distance_diff_q = sqrt((q_true_next(1)-
q_true(1,k))^2+(q_true_next(2)-q_true(2,k))^2); %difference in
 euclidian distance since last step
        angle_diff_q = q_true_next(3) - q_true(3,k); %difference in
 pointing angle since last step

        % Following arrays store differnce between true and odo/gps
 data
        d_diff(k) = distance_diff_q - odo_next(1); %difference b/w
 odometery and q true for distance
        theta_diff(k) = angle_diff_q - odo_next(2); %difference b/w
 odometery and q true for angle
        x_diff_gps(k) = q_true(1,k) - x_n; % difference b/w gps and q
 true for x value
        y_diff_gps(k) = q_true(2,k) - y_n; % difference b/w gps and q
 true for y value
        theta_diff_gps(k) = q_true(3,k) - theta_n; %differnce b/w gps
 and q true for theta

        % EKF prediction step

        %update state estimate
        odo(:,k+1) = odo(:,k) +
 [odo_next(1)*cos(odo(3,k)+odo_next(2));
 odo_next(1)*sin(odo(3,k)+odo_next(2)); odo_next(2)];
```

```
        P = Fx*P*Fx' + Fv*V*Fv'; %update estimate of state uncertainty
        q_true(:, k+1) = q_true_next; %update true position

        % Compute Kalman Gain
        S = Hx*P*Hx'+Hw*W*Hw'; %innovation covariance
        K = P*Hx'/S; %compute kalman gain

        %Compute innovation
        [x_n_next, y_n_next, theta_n_next] =
 GPS_CompassNoisy(q_true(1,k+1),q_true(2,k+1),q_true(3,k+1)); %calls
 GPS data for next step
        v_innov = [x_n_next; y_n_next; theta_n_next] - odo(:,k
+1); %calculates innovation

        %Update Step
        odo(:,k+1) = odo(:,k+1)+ K*v_innov; %update state estimation
        P = P - K*Hx*P; %update covariance matrix

    else %no GPS data for this step, integrate odometery and update P

        if mod(k-2,10) == 0 %step after GPS data is taken, saves point
 for tree localization
            qcurrent = [odo(1,k), odo(2,k), odo(3,k)]; %current true
 position

            if (odo(1,k) > 0 && odo(2,k) > 0) %if x and y values are
 positive
                points = vertcat(points,qcurrent); %adds point to
 points array (to pass to tree localization script)
            end

        end

        [q_true_next, odo_next] = robot_odo(q_true(:,k), u(:,k), Umin,
 Umax, Qmin, Qmax, l, tau_g, tau_v); %calls kinematic model
        odo(:,k+1) = odo(:,k) +
 [odo_next(1)*cos(odo(3,k)+odo_next(2));
 odo_next(1)*sin(odo(3,k)+odo_next(2)); odo_next(2)]; %updates state
 estimation
        P = Fx*P*Fx' + Fv*V*Fv'; %update estimate of state uncertainty
        q_true(:, k+1) = q_true_next; %updates tue pose
    end


    if (t == T-DT) || (t > T-DT)
        T1 = transl2(q_true(1,k),q_true(2,k))*trot2(q_true(3,k));
        figure(2)
        plotTractor(Tractor,T1,'b');
    end

    k = k+1;

    if (k == steps+1)
        k = k-1;
```

```
    end

end
```

Find Covariance

```
%prints variance for both odometery and gps data
var_d = var(d_diff);
var_theta = var(theta_diff);
var_x_gps = var(x_diff_gps);
var_y_gps = var(y_diff_gps);
var_theta_gps = var(theta_diff_gps);
```

Plot path

```
figure(2)
plot(q_true(1,:), q_true(2,:),'b');
plot(odo(1,:), odo(2,:),'r');
legend('Ground Truth','EKF Estimation')
```

Begin Tree Scanning Section.

This section performs the laser scanning and image processesing to localize the trees in the orchard and estimate thier diameters

```
uiwait(msgbox('Begin scanning, Paused until you press OK'));
```

Figure Set-up

```
close all;

figure(1); hold on; axis equal; title('Raw Probability Grid');

figure(2); hold on; axis equal; title('Processed Data');

figure(3); hold on; axis equal; title('Circles');
```

Laser Scanning

```
for k=1:length(points) % iterates through every point saved in EKF
 script

    Tl = SE2([points(k,1) points(k,2) points(k,3)]);
    p = laserScannerNoisy(angleSpan, angleStep, rangeMax, Tl.T,
 bitmapinput, Xmax, Ymax);

    % Median filter for laser scanner data
    pmed(1) = p(1,2);
    for i=2:length(p)-1
    A = [p(i-1,2),p(i,2),p(i+1,2)]; %three point window
        pmed(i) = median(A); %takes the median of points in window
```

```
    end
    pmed(i+1) = p(i+1,2);
    pmed = pmed';
    p(:,2) = pmed;

    for i=1:length(p) %for each point that the scanner passes through
        angle = p(i,1); range = p(i,2);
        % handle infinite range
        if(isinf(range))
            range = rangeMax+1;
        end
        %updates occupancy grid for each point scanned by the laser
        n = updateLaserBeamGrid(angle, range, Tl.T, R, C, Xmax, Ymax);
    end
end

for i = 1:R %for each column
    for j = 1:C %for each row
        %computes probability from odds for each point in grid
        probGrid(i,j) = (occuGrid(i,j)/(1+occuGrid(i,j)));
    end
end

figure(1)
imagesc(probGrid) %plots the probability grid
colorbar;
```

Image processing

```
figure(2)
se = strel('disk',1,0);
bitpos = imclose(occuGrid,se);
BW = imbinarize(bitpos);
subplot(1,2,1)
imshow(occuGrid);
title('Raw LIDAR data')
subplot(1,2,2)
imshow(BW);
title('Processed Image')

figure(3)
[centers, radii, metric] = imfindcircles(BW,[1 20]);
imshow(BW)
hold on
viscircles(centers, radii,'EdgeColor','b');
```

Group trees by row, calculate center and diameter of tree.

this code has not been generalized to rows size K, this only works for nurseries of Krows = 5

```
row1 = []; row2 = []; row3 = []; row4 = []; row5 = [];
```

```
%This takes the tree positions from generate nursery and ocmputes the
 locations in the bitmap
for i = 1:Krows
    [~,yC(i)] =
 XYtoIJ(3*(i-1)+18.5,-2*(i-1)+18,Xmax_nurs,Ymax_nurs,R,C);
end

for i = 1:nTrees
    [xC(i),~] =
 XYtoIJ(3*(i-1)+18.5,-2*(i-1)+22,Xmax_nurs,Ymax_nurs,R,C);
end

minX = min(xC); maxX = max(xC); minY = min(yC); maxY = max(yC);
searchrange_i = 6; %range of pixels around the predetermined locations
 where
searchrange_j = 20;%the algorithm searches for centers

%This block determines which row the tree belongs to in the nursery
 and
%will ignore any circles found in areas outside of the predetermined
 rows
%and columns
for i = 1:length(radii)
    if centers(i,1) < yC(1) + searchrange_i && centers(i,1) > yC(1) -
 searchrange_i
        if centers(i,2) < maxX+searchrange_j && centers(i,2) > minX-
searchrange_j
            row1 = [row1 i];
        end
    end
    if centers(i,1) < yC(2) + searchrange_i && centers(i,1) > yC(2) -
 searchrange_i
        if centers(i,2) < maxX+searchrange_j && centers(i,2) > minX-
searchrange_j
            row2 = [row2 i];
        end
    end
    if centers(i,1) < yC(3) + searchrange_i && centers(i,1) > yC(3) -
 searchrange_i
        if centers(i,2) < maxX+searchrange_j && centers(i,2) > minX-
searchrange_j
            row3 = [row3 i];
        end
    end
    if centers(i,1) < yC(4) + searchrange_i && centers(i,1) > yC(4) -
 searchrange_i
        if centers(i,2) < maxX+searchrange_j && centers(i,2) > minX-
searchrange_j
            row4 = [row4 i];
        end
    end
    if centers(i,1) < yC(5) + searchrange_i && centers(i,1) > yC(5) -
 searchrange_i
```

```
        if centers(i,2) < maxX+searchrange_j && centers(i,2) > minX-
searchrange_j
            row5 = [row5 i];
        end
    end
end

index_t = 1;
%Following block computes the centers and radii of each tree
for i=1:length(row1)
    [x1(i),y1(i)] =
 IJtoXY(centers(row1(i),2),centers(row1(i),1),Xmax_nurs,Ymax_nurs,R,C);
    c1(i) = radii(row1(i))/10;
    y1(i) = Ymax_nurs-y1(i);
    trees_percieved(index_t,:) = [x1(i) y1(i) c1(i)];
    index_t = index_t+1;
end

for i=1:length(row2)
    [x2(i),y2(i)] =
 IJtoXY(centers(row2(i),2),centers(row2(i),1),Xmax_nurs,Ymax_nurs,R,C);
    c2(i) = radii(row2(i))/10;
    y2(i) = Ymax_nurs-y2(i);
    trees_percieved(index_t,:) = [x2(i) y2(i) c2(i)];
    index_t = index_t+1;
end

for i=1:length(row3)
    [x3(i),y3(i)] =
 IJtoXY(centers(row3(i),2),centers(row3(i),1),Xmax_nurs,Ymax_nurs,R,C);
    c3(i) = radii(row3(i))/10;
    y3(i) = Ymax_nurs-y3(i);
    trees_percieved(index_t,:) = [x3(i) y3(i) c3(i)];
    index_t = index_t+1;
end

for i=1:length(row4)
    [x4(i),y4(i)] =
 IJtoXY(centers(row4(i),2),centers(row4(i),1),Xmax_nurs,Ymax_nurs,R,C);
    c4(i) = radii(row4(i))/10;
    y4(i) = Ymax_nurs-y4(i);
    trees_percieved(index_t,:) = [x4(i) y4(i) c4(i)];
    index_t = index_t+1;
end

for i=1:length(row5)
    [x5(i),y5(i)] =
 IJtoXY(centers(row5(i),2),centers(row5(i),1),Xmax_nurs,Ymax_nurs,R,C);
    c5(i) = radii(row5(i))/10;
    y5(i) = Ymax_nurs-y5(i);
    trees_percieved(index_t,:) = [x5(i) y5(i) c5(i)];
    index_t = index_t+1;
end
```

Output to text file

```
fileID = fopen('finaloutput.txt','w');

fprintf(fileID,'1 \n');
for i = 1:length(row1)
    fprintf(fileID,'%d, %.2f, %.2f, %.2f\n',i,x1(i),y1(i),c1(i));
end

fprintf(fileID,'2 \n');
for i = 1:length(row2)
    fprintf(fileID,'%d, %.2f, %.2f, %.2f\n',i,x2(i),y2(i),c2(i));
end

fprintf(fileID,'3 \n');
for i = 1:length(row3)
    fprintf(fileID,'%d, %.2f, %.2f, %.2f\n',i,x3(i),y3(i),c3(i));
end

fprintf(fileID,'4 \n');
for i = 1:length(row4)
    fprintf(fileID,'%d, %.2f, %.2f, %.2f\n',i,x4(i),y4(i),c4(i));
end

fprintf(fileID,'5 \n');
for i = 1:length(row5)
    fprintf(fileID,'%d, %.2f, %.2f, %.2f\n',i,x5(i),y5(i),c5(i));
end
```

Published with MATLAB® R2019a