Extended Kalman Filter and Pinhole Camera Model for Path Tracking and Sensor Modeling

Angela Voo

October 2024

1 Introduction

Tracking and state estimation are essential in autonomous navigation, where sensor-based measurements are used to predict and update vehicle positions. We chose to explore aspects of the pan-tilt camera looking from above modeled using the pinhole camera model. We also craft a simulation to track a car from above with a camera.

2 Problem

2.1 Camera Model

Using the pinhole camera model, we explore the field of view (FOV) dependency on focal length, pan angle, and tilt angle. We also see how the trajectory of a stationary object would look on the image plane from the pan or tilt of the camera.

2.2 Tracking

In the tracking scenario, we have a fixed camera from above looking down on an Ackerman-steered car with wheel-base of 4 m driving in a swirl trajectory at 0.5 m/s. The camera is positioned 20 m above the ground with a focal length of 85 mm and an image plane with width and height of 50 mm. The camera has a sensor noise of 0.000001 m^2.

3 Method

3.1 Pinhole Camera Model

The camera has pan (ψ) and tilt (ϕ) angles. The transformation matrices for tilt ($\boldsymbol{R}_\phi$) and pan ($\boldsymbol{R}_\psi$) rotations are given by:

$$\boldsymbol{R}_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix} \tag{1}$$

$$\boldsymbol{R}_\psi = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

The projected image coordinates p_x and p_y of the car state $\boldsymbol{x}_t$ onto the virtual image plane are:

$$p_x = \lambda \frac{q_x}{q_z}, \quad p_y = \lambda \frac{q_x}{q_z} \tag{3}$$

where λ is the camera focal length, $\boldsymbol{x}_c$ is the position of the camera (pinhole) in the inertial frame and the components of $\boldsymbol{q}_t$ are defined as:

$$\boldsymbol{q}_t \triangleq \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} = \boldsymbol{R}_\phi^T \boldsymbol{R}_\psi^T([\boldsymbol{x}_t^T 0]^T - \boldsymbol{x}_c] \tag{4}$$

These coordinates are valid only if the projected point lies within the image plane boundaries.

3.2 Car Model

The car motion follows a swirl path, controlled by a constant speed v and a changing steering angle α. The vehicle's state at any time step k is represented by the state vector:

$$\boldsymbol{x}_k = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \alpha \\ v \\ z \end{bmatrix} \tag{5}$$

where x and y are the position coordinates, θ is the orientation angle, v is the linear velocity input, α is the steering angle input, and z represents a constant height. Below are the equations of motion that describe the car using the Ackerman steering kinematic model:

$$x_{k+1} = x_k + v\cos(\theta_k)dt \tag{6}$$

$$y_{k+1} = y_k + v\sin(\theta_k)dt \tag{7}$$

$$\theta_{k+1} = \theta_k + \frac{v}{L}\tan(\alpha_k)dt \tag{8}$$

where L is the vehicle's wheelbase.

3.3 Extended Kalman Filter

Camera sensor data is simulated using the pinhole camera model. To track the car, we implement an Extended Kalman Filter to estimate the state of the car based on the simulated camera sensor data, taking into account the camera sensor noise.

3.3.1 Prediction Step

The EKF predicts the next state based on the car motion model f as defined in Section 3.2. The state $\boldsymbol{x}$ and covariance $\boldsymbol{P}$ of the prediction step is defined as follows:

$$\boldsymbol{x}_{k+1|k} = f(\boldsymbol{x}_{k|k}) \tag{9}$$

$$\boldsymbol{P}_{k+1|k} = \boldsymbol{F}_k \boldsymbol{P}_{k|k} \boldsymbol{F}_k^T + \boldsymbol{G}_k \boldsymbol{Q}_{k|k} \boldsymbol{G}_k^T \tag{10}$$

3.3.2 Jacobian Matrices

The linearization of the prediction function requires the computation of the Jacobians $\boldsymbol{F}$ and $\boldsymbol{G}$, where $\boldsymbol{F}$ is the Jacobian of the motion model with respect to the state, and $\boldsymbol{G}$ is with respect to process noise.

$$
\boldsymbol{F} = \begin{bmatrix} 1 & 0 & -dt\, v\sin(\theta) & dt\cos(\theta) & 0 & 0 \\ 0 & 1 & dt\, v\cos(\theta) & dt\sin(\theta) & 0 & 0 \\ 0 & 0 & 1 & \frac{dt}{L}\tan(\alpha) & dt\frac{v}{L}\sec^2(\alpha) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}
$$

$$
\boldsymbol{G} = \begin{bmatrix} dt & 0 & 0 & 0 & 0 & 0 \\ 0 & dt & 0 & 0 & 0 & 0 \\ 0 & 0 & dt & 0 & 0 & 0 \\ 0 & 0 & 0 & dt & 0 & 0 \\ 0 & 0 & 0 & 0 & dt & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{12}
$$

3.3.3 Update Step

In the update step, the EKF incorporates the camera measurements to correct the predicted state. Using the Jacobian $\boldsymbol{H}$, derived from the partial derivatives of the pinhole projection equations p_x and p_y, the Kalman Gain $\boldsymbol{K}$ is computed as:

$$
\boldsymbol{K} = \boldsymbol{P}_{k+1|k}\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}_{k+1|k}\boldsymbol{H}^T + \boldsymbol{R})^{-1} \tag{13}
$$

The state $\boldsymbol{x}$ and covariance $\boldsymbol{P}$ of the update step is defined as follows:

$$
\boldsymbol{x}_{k+1|k+1} = \boldsymbol{x}_{k+1|k} + \boldsymbol{K}_{k+1}(\boldsymbol{z}_{k+1} - h(\boldsymbol{x}_{k+1|k}) \tag{14}
$$

$$
\boldsymbol{P}_{k+1|k+1} = (I - \boldsymbol{K}_{k+1}\boldsymbol{H})\boldsymbol{P}_{k+1|k}(I - \boldsymbol{K}_{k+1}\boldsymbol{H})^T + \boldsymbol{K}_{k+1}\boldsymbol{R}_{k+1}\boldsymbol{K}_{k+1}^T \tag{15}
$$

where $\boldsymbol{z}_{k+1}$ is the camera sensor measurements of the update step, h is the measurement function representing the pinhole projection, and $\boldsymbol{R}$ is the sensor noise.

4 Results

4.1 Camera Field of View (FOV)

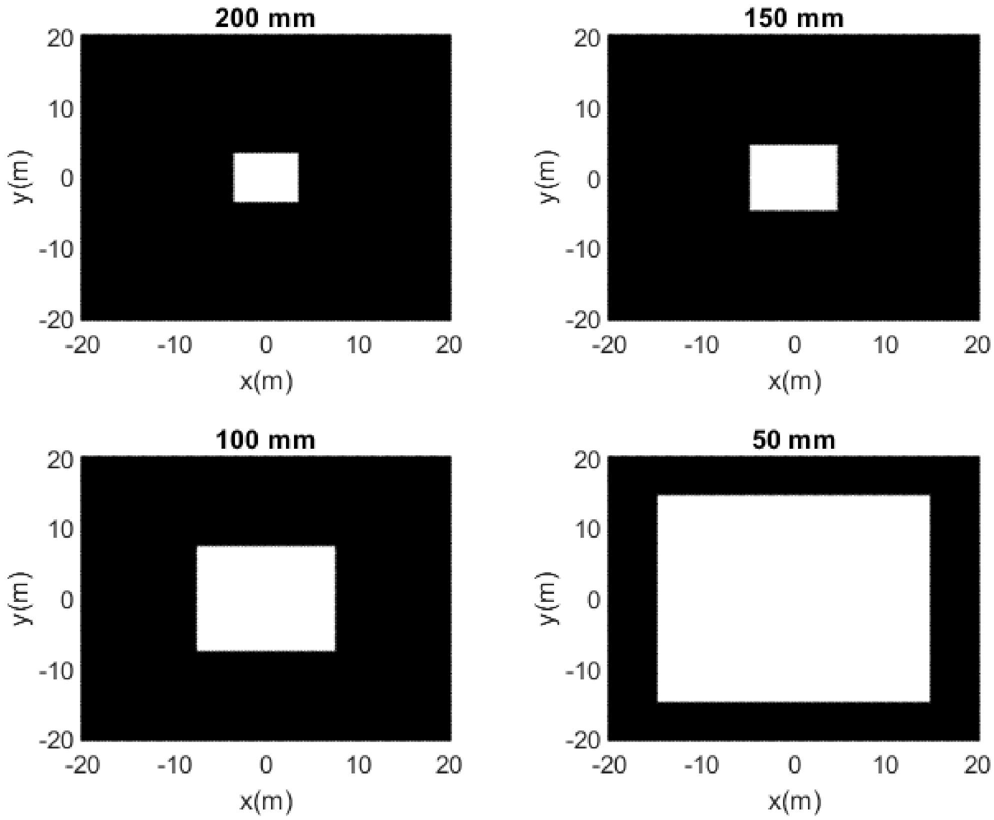


Figure 1: FOV with different focus lengths

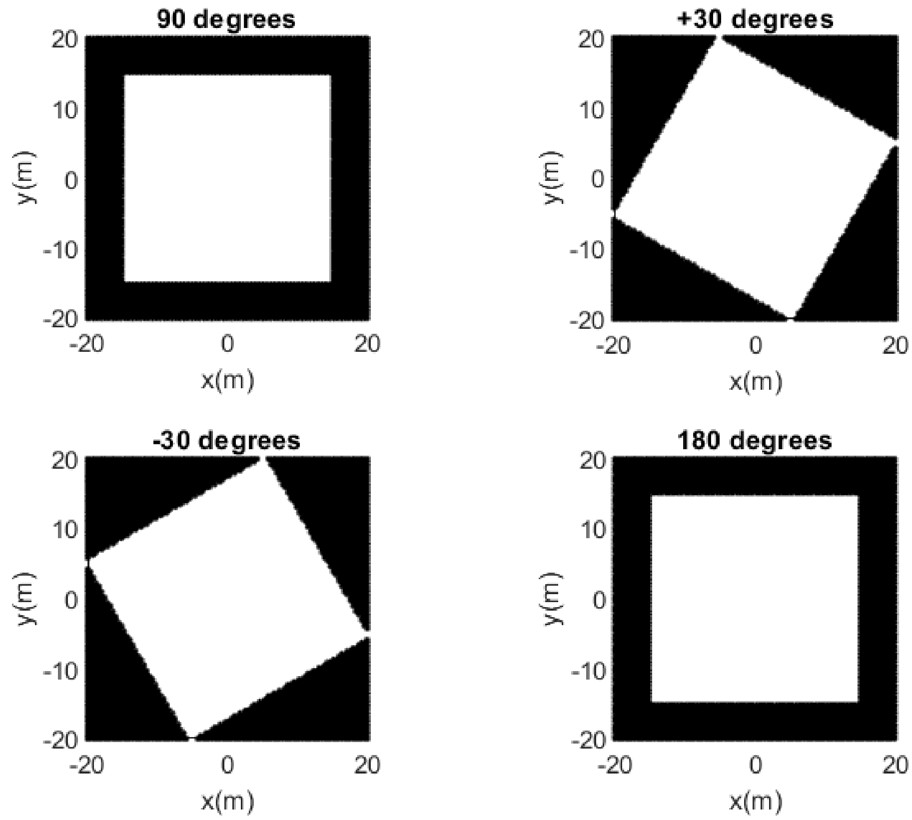


Figure 2: FOV with different pan angles

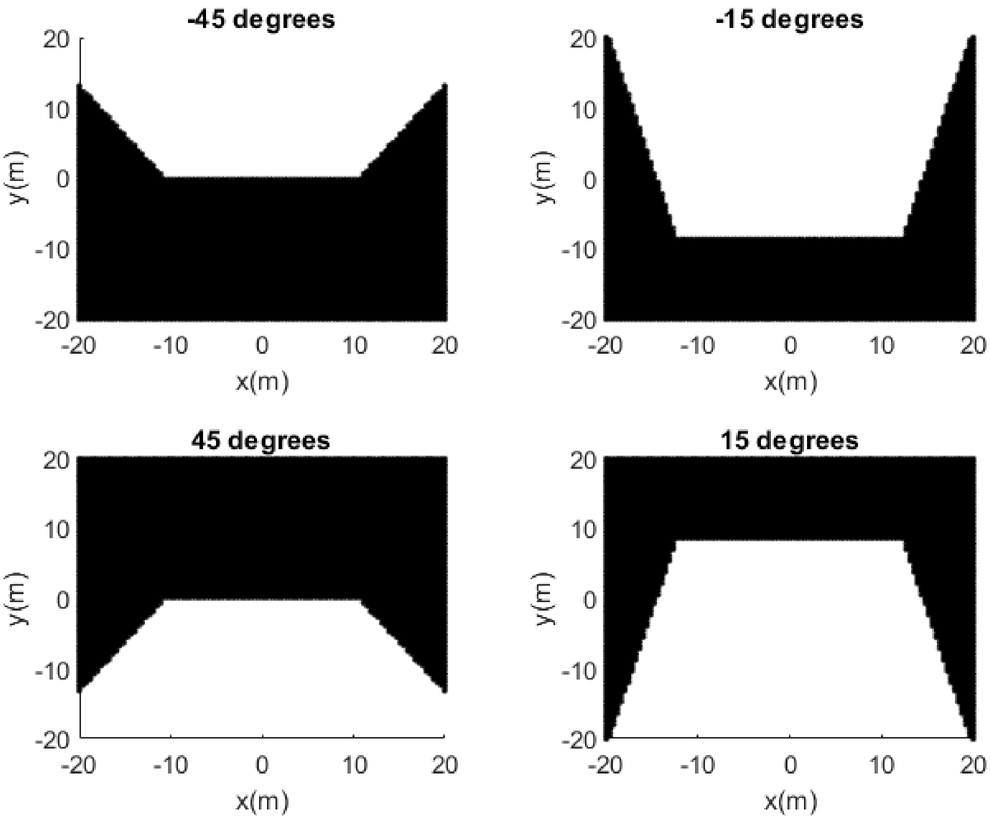


Figure 3: FOV with different tilt angles

4.2 Camera Pan and Tilt with Stationary Object

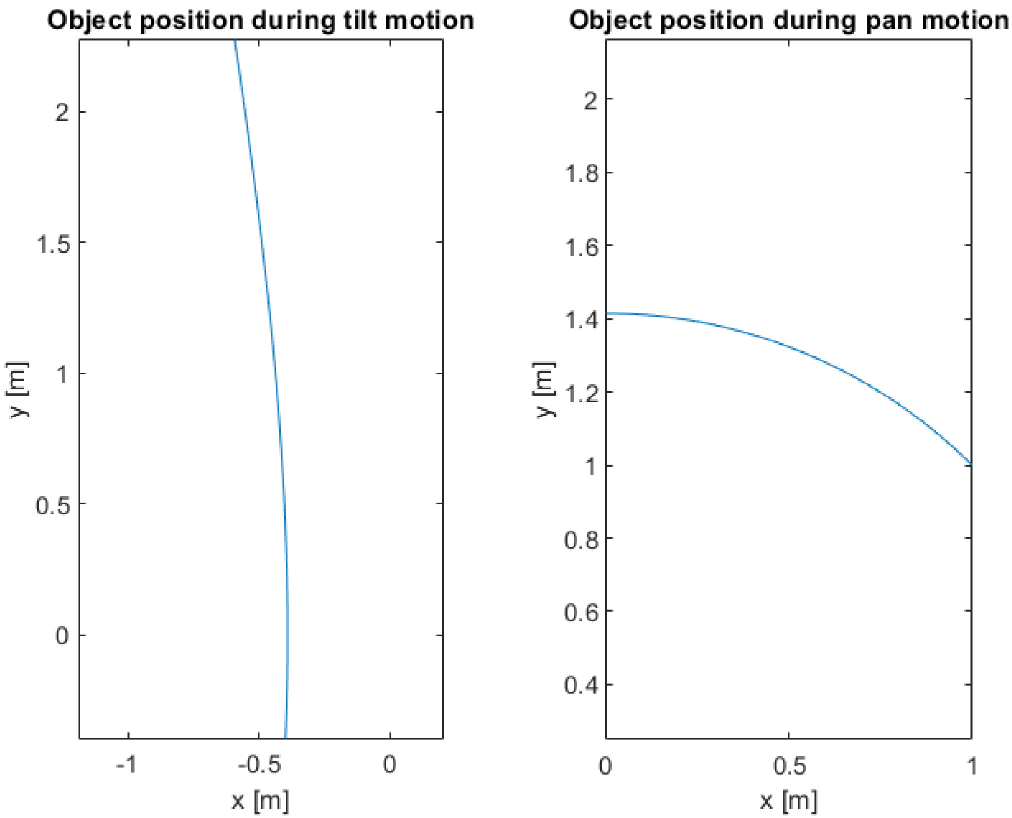
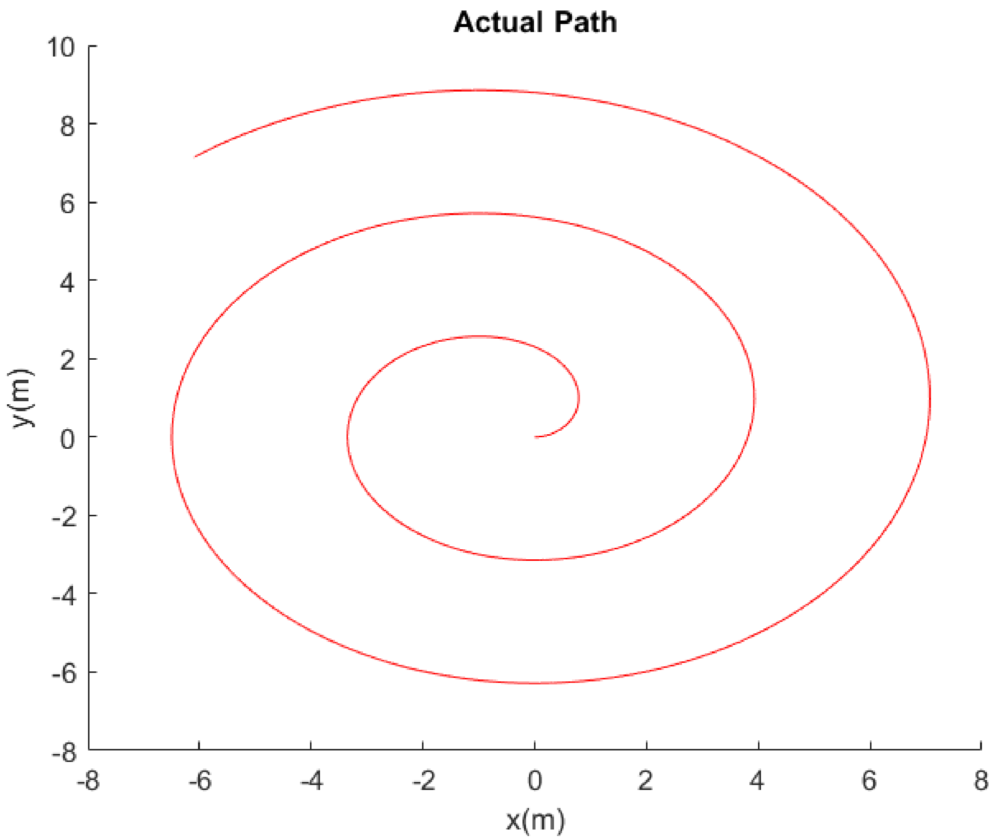
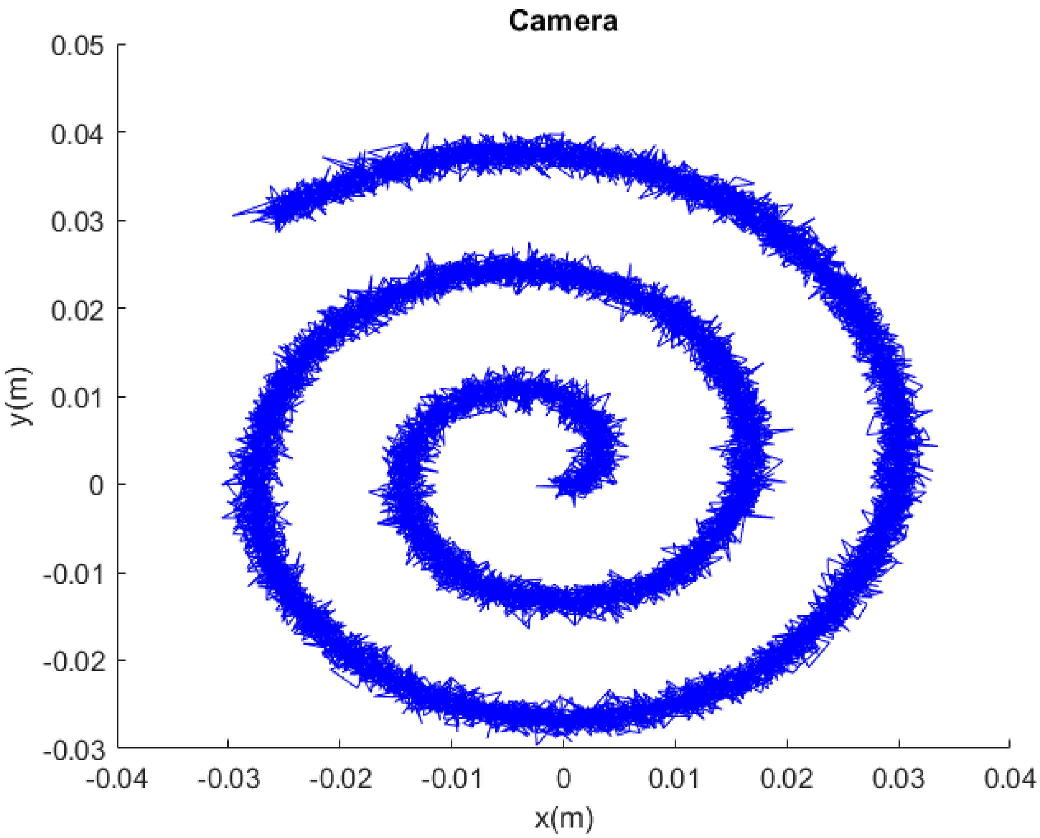
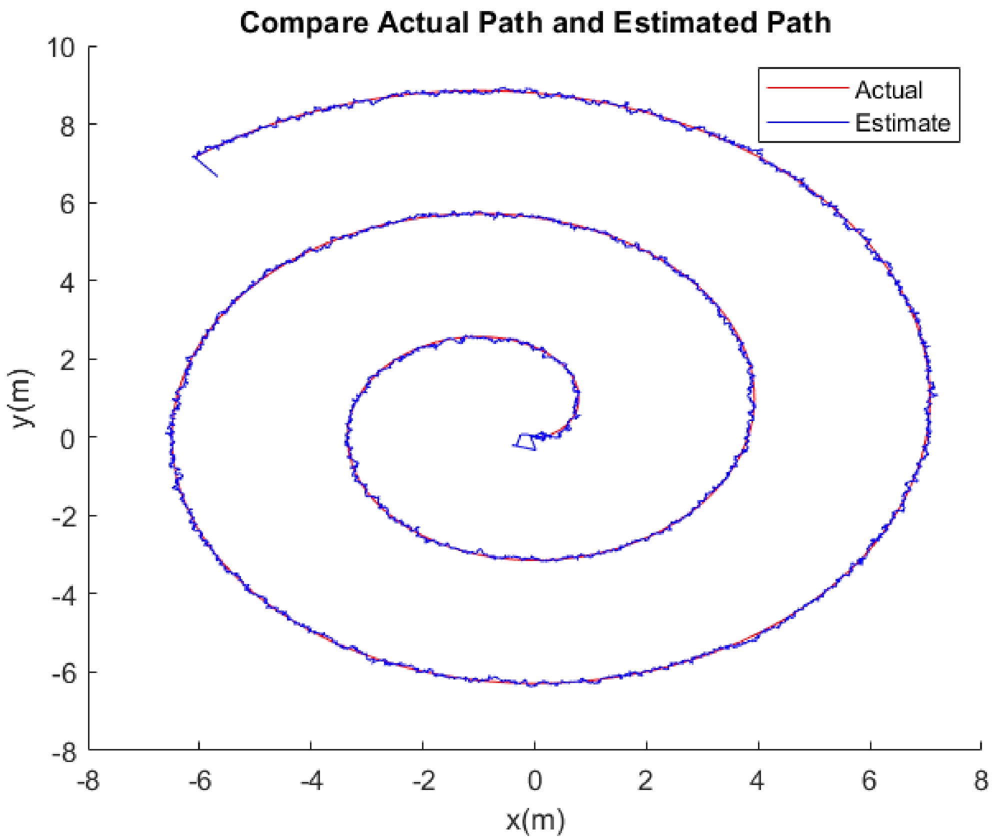


Figure 4: Trajectory of stationary object on image plane of camera as it pans and tilts

4.3 Camera Tracking




Camera
0.05
0.04
0.03
0.02
0.01
0
-0.01
-0.02
-0.03
y(m)
-0.04
-0.03
-0.02
-0.01
0
0.01
0.02
0.03
0.04
x(m)



5 Discussion

In the above figures 1-3, you can see how the focus length, pan, and tilt angles affect the camera FOV. In the camera tracking figures, you can see that although there is non-negligible noise that the camera senses, the estimated path can estimate the actual path very well.

6 Conclusion

Some future work would be to have the camera move so that the target object (car) would stay in the middle of the image plane, and track the camera state (pan and tilt angle).

7 Appendix A: Camera FOV Matlab Code

```
clear;
close all;
x = linspace(-20,20);
y = linspace(-20,20);
%% FOV FOCAL Length dependency
figure;
tiledlayout(2,2);
nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
```

```
        meas = camera_model(0,0,0.2,[x(i);y(j)],[0,0,15]');
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("200 mm")
xlabel('x(m)')
ylabel('y(m)')
hold off

nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
        meas = camera_model(0,0,0.15,[x(i);y(j)],[0,0,15]');
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("150 mm")
xlabel('x(m)')
ylabel('y(m)')
hold off

nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
        meas = camera_model(0,0,0.1,[x(i);y(j)],[0,0,15]');
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("100 mm")
xlabel('x(m)')
ylabel('y(m)')
hold off

nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
        meas = camera_model(0,0,0.05,[x(i);y(j)],[0,0,15]');
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("50 mm")
xlabel('x(m)')
ylabel('y(m)')
```

```
hold off
%sgtitle("Camera: 15 meters above ground")

%% FOV Pan angle dependency
figure;
tiledlayout(2,2);
nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
        meas = camera_model(pi/2,0,0.1,[x(i);y(j)],[0,0,30]');
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("90 degrees")
xlabel('x(m)')
ylabel('y(m)')
axis square
hold off

nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
        meas = camera_model(pi/6,0,0.1,[x(i);y(j)],[0,0,30]');
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("+30 degrees")
xlabel('x(m)')
ylabel('y(m)')
axis square
hold off

nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
        meas = camera_model(-pi/6,0,0.1,[x(i);y(j)],[0,0,30]');
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("-30 degrees")
xlabel('x(m)')
ylabel('y(m)')
axis square
hold off

```

```
nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
        meas = camera_model(pi,0,0.1,[x(i);y(j)],[0,0,30]');
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("180 degrees")
xlabel('x(m)')
ylabel('y(m)')
axis square
hold off

%% FOV tilt angle angle dependency
figure;
tiledlayout(2,2);
nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
        meas = camera_model(0,-pi/4,0.05,[x(i);y(j)],[0,0,15]');
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("-45 degrees")
xlabel('x(m)')
ylabel('y(m)')
hold off

nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
        meas = camera_model(0,-pi/12,0.05,[x(i);y(j)],[0,0,15]');
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("-15 degrees")
xlabel('x(m)')
ylabel('y(m)')
hold off

nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
        meas = camera_model(0,pi/4,0.05,[x(i);y(j)],[0,0,15]');
```

```
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("45 degrees")
xlabel('x(m)')
ylabel('y(m)')
hold off

nexttile
hold on
for i = 1:length(x)
    for j = 1:length(y)
        meas = camera_model(0,pi/12,0.05,[x(i);y(j)],[0,0,15]');
        if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
            plot(x(i),y(j),'k.')
        end
    end
end
title("15 degrees")
xlabel('x(m)')
ylabel('y(m)')
hold off

%% Functions
function sensor_meas=camera_model(psi,phi,lambda,x_t,x_c)
%sensor model for the camera
%psi and phi are pan and tilt angles
%lambda is focal length of the parameter
%x_t is location of robot in inertial frame (must be a 3d vector)
%x_c is coordinates of the camera in the inertial frame (3d vector)
%returns sensor_meas which is a 2d vector (xp, yp)

R_phi = [1     0          0;
         0  cos(phi) sin(phi);
         0 -sin(phi) cos(phi)];
R_psi = [cos(psi) sin(psi) 0;
         -sin(psi) cos(psi) 0;
         0          0        1];
q_t = R_phi'*R_psi'*([x_t' 0]' - x_c);
p_t = lambda*[q_t(1)/q_t(3) q_t(2)/q_t(3)]';
sensor_meas = -p_t;
end
```

8 Appendix B: Camera Pan and Tilt Matlab Code

```
close all;
clear;
%Modeling of the sensor pan and tilt
x_t = [-1;-1]; %fixed target state
x_c = [0;0;5]; %fixed camera position
phi_array = linspace(0,pi/3,100);
```

```
psi_array = linspace(0,pi/2,100);
t = linspace(0,5,100);
lambda = 2;
meas = zeros(2,100);
tiledlayout(1,2)
nexttile
%show camera dependence on tilt angle
for k = 1:100
    psi = 0;
    phi = phi_array(k); % constant tilt
    meas(:,k) = camera_model(psi,phi,lambda,x_t,x_c);
end

plot(meas(1,:),meas(2,:))
xlabel("x [m]")
ylabel("y [m]")
title("Object position during tilt motion")
axis equal;
nexttile
%show camera dependence on pan angle
x_t = [1;1]; %fixed target state
x_c = [0;0;2]; %fixed camera position
phi_array = linspace(0,-pi/4,100);
psi_array = linspace(0,pi/4,100);
t = linspace(0,5,100);
lambda = 2;
meas = zeros(2,100);
for k = 1:100
    phi = 0;
    psi = psi_array(k); % constant pan
    meas(:,k) = camera_model(psi,phi,lambda,x_t,x_c);
end
plot(meas(1,:),meas(2,:))
xlabel("x [m]")
ylabel("y [m]")
title("Object position during pan motion")
axis equal;

function sensor_meas=camera_model(psi,phi,lambda,x_t,x_c)
%sensor model for the camera
%psi and phi are pan and tilt angles
%lambda is focal length of the parameter
%x_t is location of robot in inertial frame (must be a 3d vector)
%x_c is coordinates of the camera in the inertial frame (3d vector)
%returns sensor_meas which is a 2d vector (xp, yp)

R_phi = [1     0          0;
         0  cos(phi)  sin(phi);
         0 -sin(phi)  cos(phi)];
R_psi = [cos(psi) sin(psi) 0;
        -sin(psi) cos(psi) 0;
         0        0        1];
q_t = R_phi'*R_psi'*([x_t' 0]' - x_c);
p_t = lambda*[q_t(1)/q_t(3) q_t(2)/q_t(3)]';
```

```
sensor_meas = -p_t;
end
```

9 Appendix C: Camera Tracking Matlab Code

```
close all;
clear;
%% Sensor Model
%camera parameters
focal = 0.085; % focal length
xc = [0;0;20]; %camera location
sx = 0.05; %image plane width
sy = 0.05; %image plane height
%sensor and process noise parameters
Rq = 0.000001;
Qsys = 0;
dt = 0.01;

%% Swirl Path
t = 0:dt:150; %setup the simulation of the car
v = 0.5; %
L = 4;
%simulate the motion of the car (sinusoidal path)
x = zeros(length(t),1);
y = zeros(length(t),1);
theta = zeros(length(t),1);
alpha = zeros(length(t),1);
pixels = zeros(length(t),2);

R = 1; % starting radius
for k = 1:length(t) - 1
    grow = 1/R;
    alpha(k) = atan(L*grow/(v));
    x(k+1) = x(k) + v*cos(theta(k))*dt;
    y(k+1) = y(k) + v*sin(theta(k))*dt;
    theta(k+1) = theta(k) + (v)/L * tan(alpha(k))*dt;
    R = R + grow*dt;
    %generate simulated sensor measurements
    sensor_meas = camera_model(0,0,0.085,[x(k+1);y(k+1)],[0;0;20],sx,sy)
        ;
    pixels(k,1) = sensor_meas(1) + randn(1,1)*sqrt(Rq);
    pixels(k,2) = sensor_meas(2) + randn(1,1)*sqrt(Rq);
end


figure
hold on
plot(x,y,'r')
title("Actual Path")
xlabel('x(m)')
ylabel('y(m)')
hold off

```

```matlab
figure
hold on
plot(pixels(1:length(t)-1,1),pixels(1:length(t)-1,2),'b')
title("Camera")
xlabel('x(m)')
ylabel('y(m)')
hold off

%% compute symbolic jacobian for camera model
syms phi psi xc_1 xc_2 xc_3 xt_1 xt_2 xt_3 lambda v theta alpha

R_phi = [1 0 0;
         0 cos(phi) sin(phi);
         0 -sin(phi) cos(phi)];


R_psi = [cos(psi) sin(psi) 0;
        -sin(psi) cos(psi) 0;
         0 0 1];

pt = R_phi'*R_psi'*([xt_1 ; xt_2; xt_3] - [xc_1; xc_2; xc_3]);

p = -lambda*[pt(1)/pt(3); pt(2)/pt(3)];
p_x = p(1);
p_y = p(2);

%compute the jacobian matrix
jac = jacobian([p_x, p_y],[xt_1,xt_2,xt_3, v, theta, alpha]);
% jac = [diff(p_x,xt_1), diff(p_x,xt_2), diff(p_x,xt_3),0,0,0;
%       diff(p_y,xt_1), diff(p_y,xt_2), diff(p_y,xt_3),0,0,0];
H_jac = matlabFunction(jac);


%% EKF Filter for Swirl Path
x0=[0;0;0;0;0;0];
P0=diag([10^2 10^2 10^2 10^2 10^2 10^2]);
xhat1p=x0;P1p = zeros(length(x0),length(x0),length(t)-1);P1p(:,:,1)=P0;
xhat1u=x0;P1u = zeros(length(x0),length(x0),length(t)-1);P1u(:,:,1)=P0;
xhat1u=x0;P1u(:,:,1)=P0;
Q = diag([0.000001,0.000001,0.000001,0.0000001,0.0000001,0.000001]); % guess process noise covariance
R = diag([0.000001^2 0.000001^2]); %guess sensor noise covariance
Z = pixels'; %sensor measurements
for k=1:(length(t)-1)
    %predict state
    xhat1p(:,k+1)=predict_state_track(xhat1u(:,k),L,dt);
    %predict covariance
    [F,G]=getFG_carpose_track(xhat1u(:,k),L,dt);
    P1p(:,:,k+1) = F*P1u(:,:,k)*F' + G*Q*G';
    %predict psi and phi values
    psi = 0;
    phi = 0;
    %Kalman Gain
    H = H_jac(focal,phi,psi,xc(1),xc(2),xc(3),xhat1p(1,k+1),xhat1p(2,k
```

```
        +1),0);
    K = P1p(:,:,k+1)*H'*inv(H*P1p(:,:,k+1)*H' + R);
    %get sensor measurement from predicted data
    meas = camera_model(psi,phi,focal,[xhat1p(1,k+1);xhat1p(2,k+1)],[xc(1);xc(2);xc(3)],sx,sy);
    %skips update step if target is outside the sensor fov
    if isnan(meas) == false
        xhat1u(:,k+1) = xhat1p(:,k+1) + K *(Z(:,k+1) - meas);
        P1u(:,:,k+1) = (eye(6)-K*H)*P1p(:,:,k+1)*(eye(6)-K*H)' + K*R*K';
    else
        xhat1u(:,k+1) = xhat1p(:,k+1);
        P1u(:,:,k+1) = P1p(:,:,k+1);
    end
end
figure
hold on
plot(x,y,'r')
plot(xhat1u(1,1:length(t)),xhat1u(2,1:length(t)),'b')
title("Compare Actual Path and Estimated Path")
xlabel('x(m)')
ylabel('y(m)')
legend('Actual','Estimate')
hold off

%% Functions
function Xkp1=predict_state_track(Xk,L,dt)
%
%   For object tracking
%   discrete prediction of state
% This is now 6 states, since the control inputs are added on as state
% vectors and the z coordinate is appended onto the end as a constant value

v=Xk(4);
alpha=Xk(5);
tk = Xk(3);

Xkp1 = Xk +...
    dt*[v*cos(tk);
        v*sin(tk);
        v/L * tan(alpha);
        0;
        0;
        0];
end


function [F,G]=getFG_carpose_track(X,L,dt)
%
%   For object tracking
%   get the linearized system matrixes F,G
%

v=X(4);
```

```
alpha=X(5);
tk = X(3);
F=[1 0 -dt*v*sin(tk) dt*cos(tk) 0 0;
   0 1 dt*v*cos(tk) dt*sin(tk) 0  0;
   0 0 1 dt*1/L * tan(alpha) dt*v/L*(sec(alpha))^2 0;
   0 0 0 1 0 0;
   0 0 0 0 1 0;
   0 0 0 0 0 1];

G=[dt 0 0 0 0 0;
   0 dt 0 0 0 0;
   0  0 dt 0 0 0;
   0  0  0 dt 0 0;
   0  0  0 0  dt 0;
   0  0  0 0  0 0]; %no noise measurements in z coordinate
end


function sensor_meas=camera_model(psi,phi,lambda,x_t,x_c,sx,sy)
%sensor model for the camera
%psi and phi are pan and tilt angles
%lambda is focal length of the parameter
%x_t is location of robot in inertial frame (must be a 3d vector)
%x_c is coordinates of the camera in the inertial frame (3d vector)
%returns sensor_meas which is a 2d vector (xp, yp)

R_phi = [1     0            0;
         0  cos(phi) sin(phi);
         0 -sin(phi) cos(phi)];
R_psi = [cos(psi) sin(psi) 0;
         -sin(psi) cos(psi) 0;
         0          0        1];
q_t = R_phi'*R_psi'*([x_t' 0]' - x_c);
p_t = lambda*[q_t(1)/q_t(3) q_t(2)/q_t(3)]';
if abs(p_t(1)) <= sx && abs(p_t(2)) <= sy
    sensor_meas = -p_t;
else
    sensor_meas = NaN;
end
end
```