

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 (\mathbf{R}_ϕ) and pan (\mathbf{R}_ψ) rotations are given by:

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

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

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

$$p_x = \lambda \frac{q_x}{q_z}, \quad p_y = \lambda \frac{q_y}{q_z} \quad (3)$$

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

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

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:

$$\mathbf{x}_k = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \alpha \\ v \\ z \end{bmatrix} \quad (5)$$

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:

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

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

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

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 \mathbf{x} and covariance \mathbf{P} of the prediction step is defined as follows:

$$\mathbf{x}_{k+1|k} = f(\mathbf{x}_{k|k}) \quad (9)$$

$$\mathbf{P}_{k+1|k} = \mathbf{F}_k \mathbf{P}_{k|k} \mathbf{F}_k^T + \mathbf{G}_k \mathbf{Q}_{k|k} \mathbf{G}_k^T \quad (10)$$

3.3.2 Jacobian Matrices

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

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

$$\mathbf{G} = \begin{bmatrix} dt & 0 & 0 & 0 & 0 & 0 \\ 0 & dt & 0 & 0 & 0 & 0 \\ 0 & 0 & dt & 0 & 0 & 0 \\ 0 & 0 & 0 & dt & 0 & 0 \\ 0 & 0 & 0 & 0 & dt & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (12)$$

3.3.3 Update Step

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

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

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

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

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

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

4 Results

4.1 Camera Field of View (FOV)

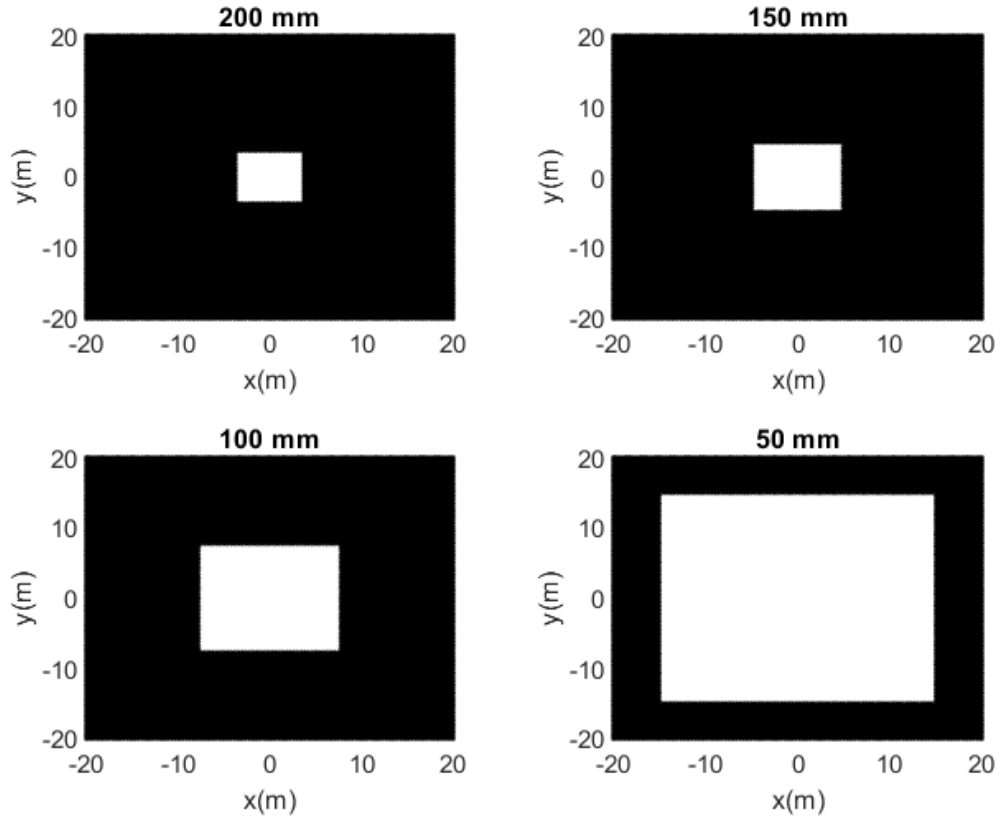


Figure 1: FOV with different focus lengths

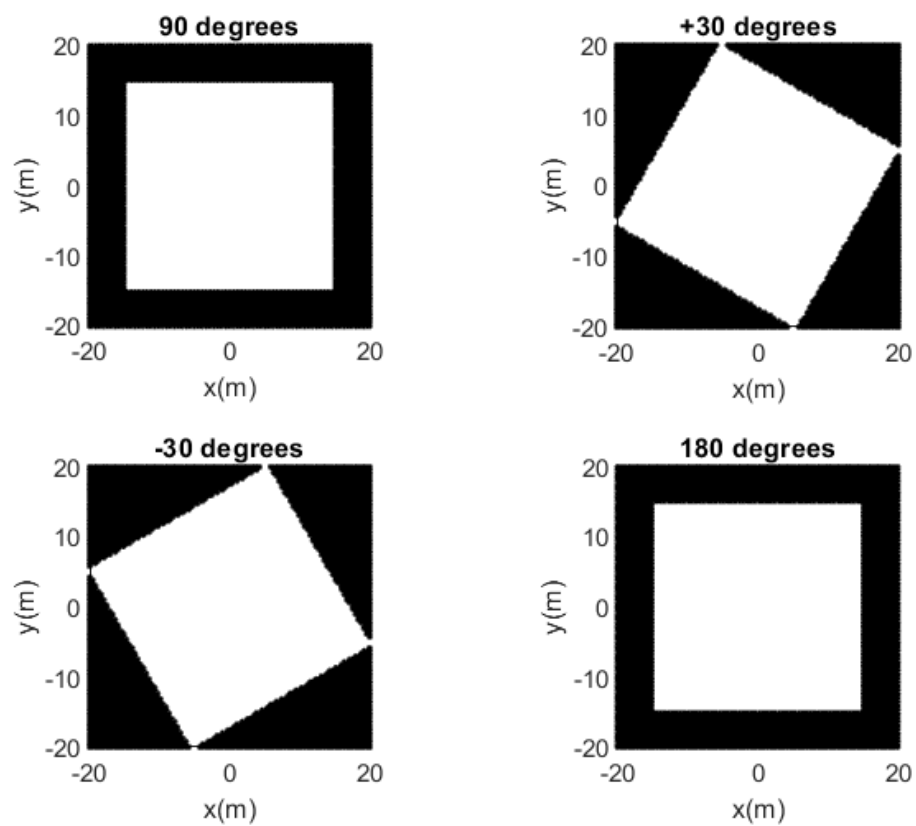


Figure 2: FOV with different pan angles

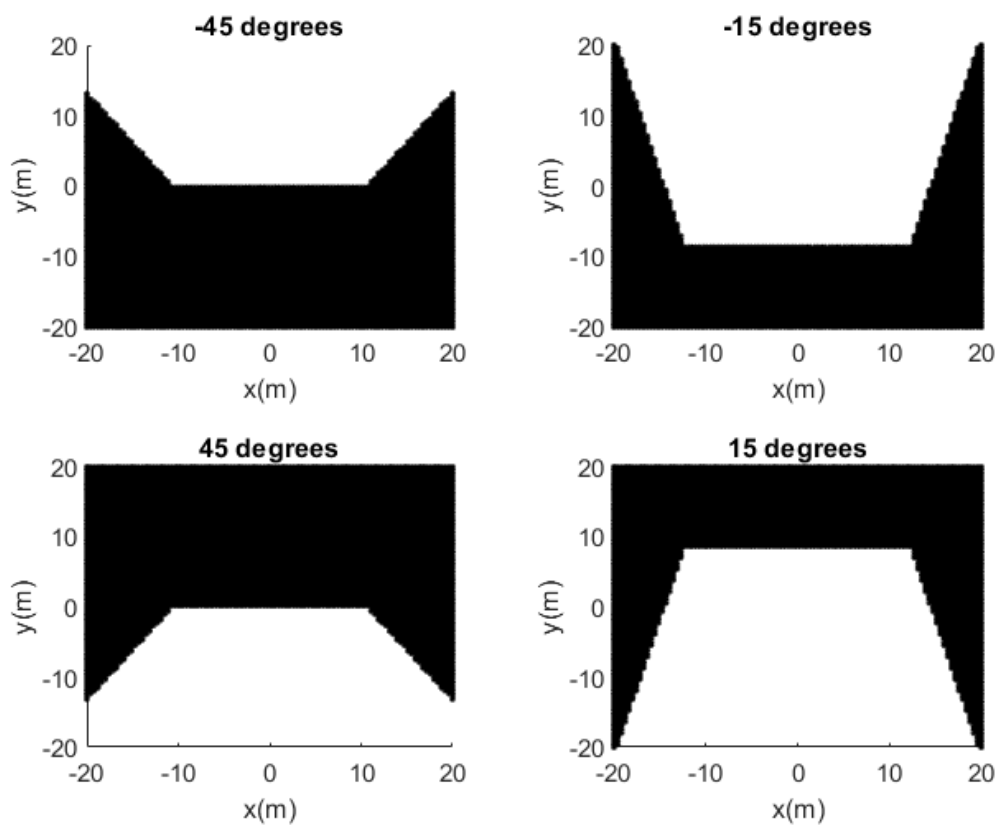


Figure 3: FOV with different tilt angles

4.2 Camera Pan and Tilt with Stationary Object

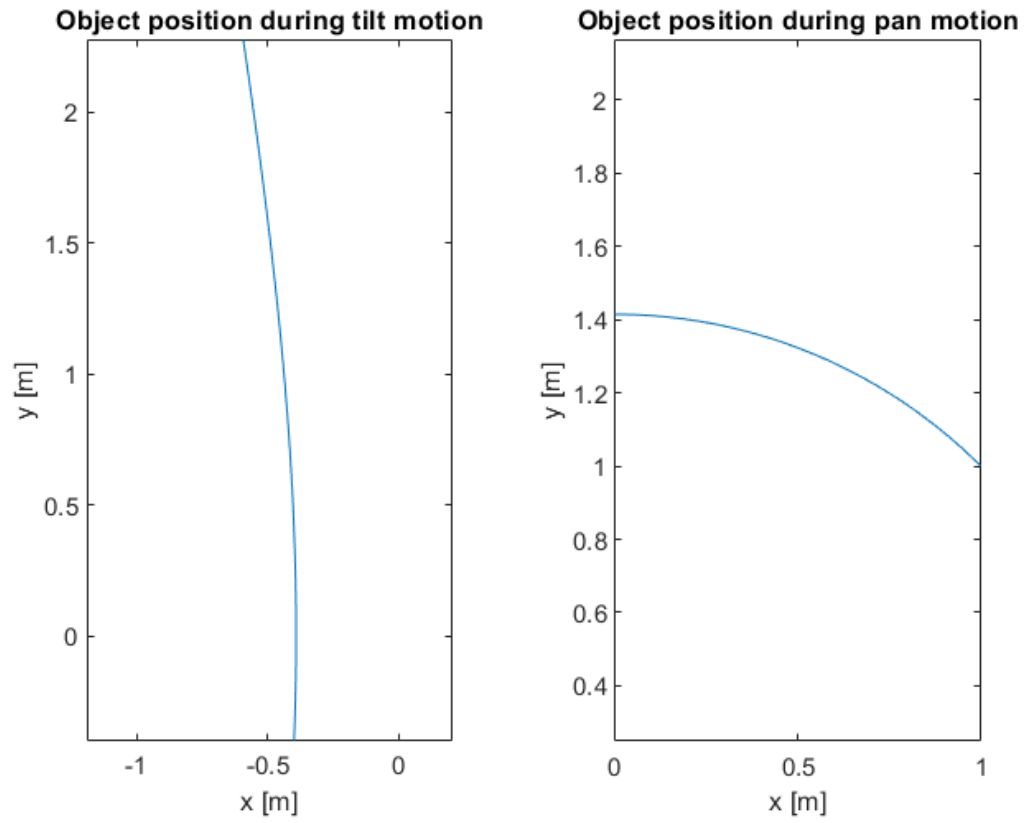
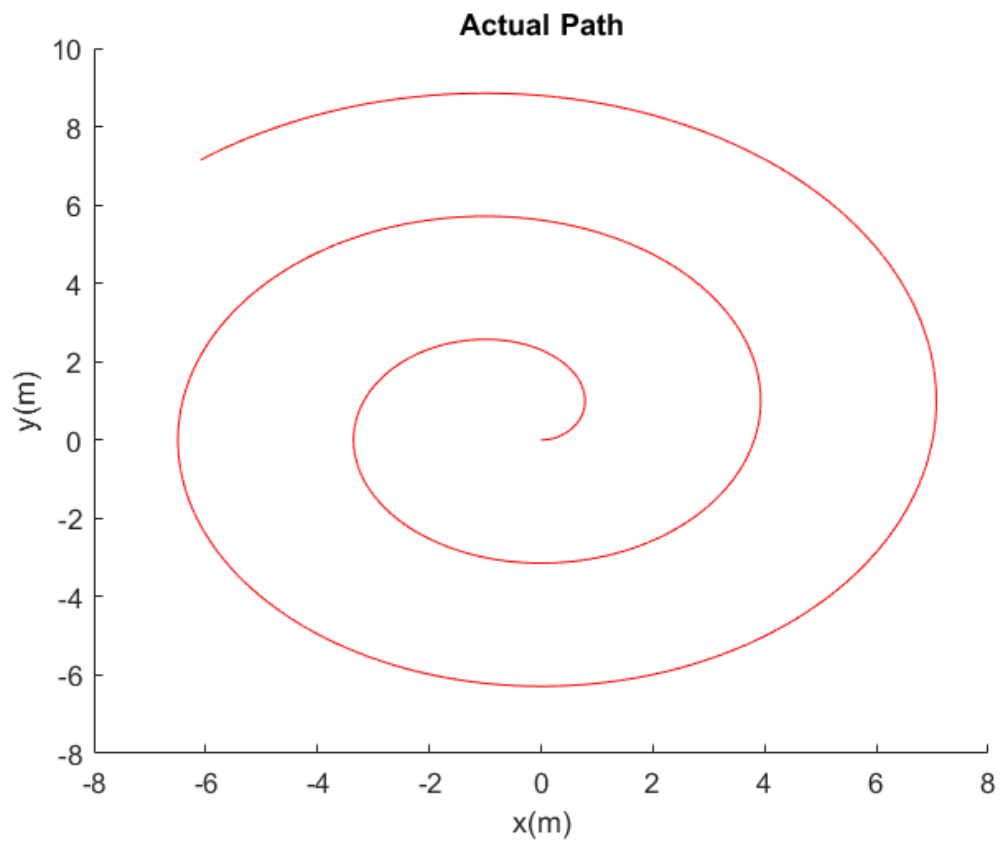
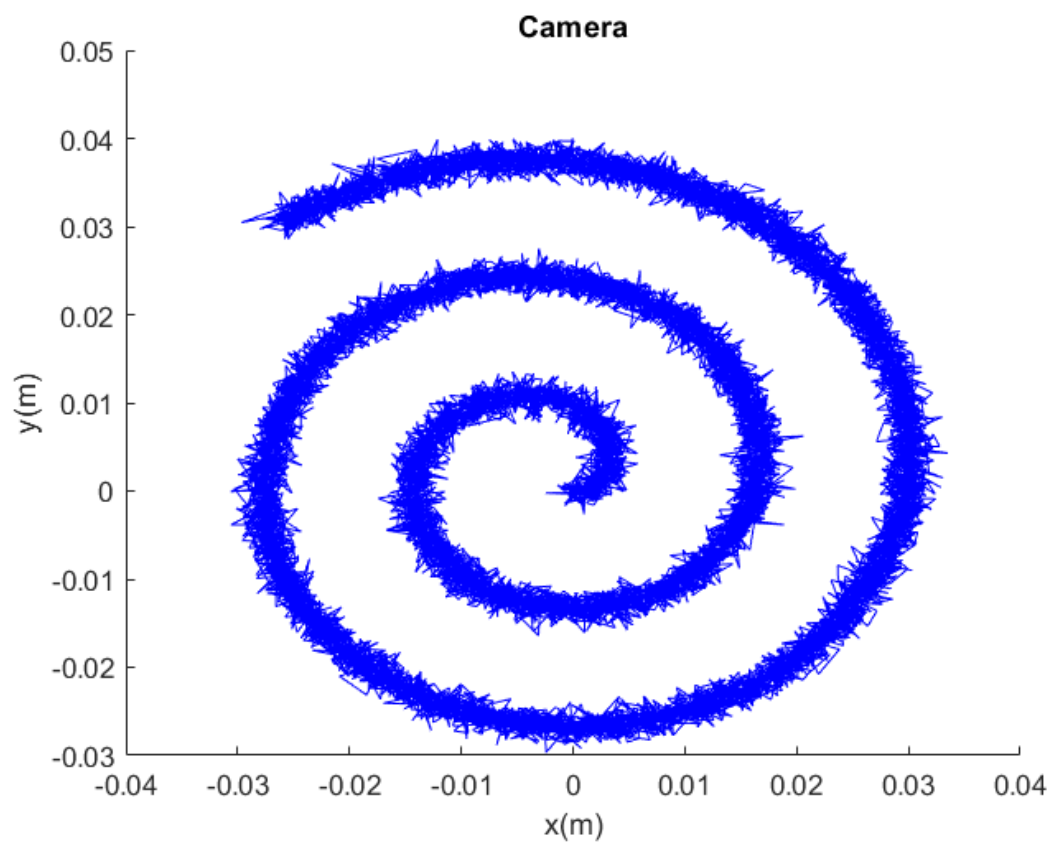
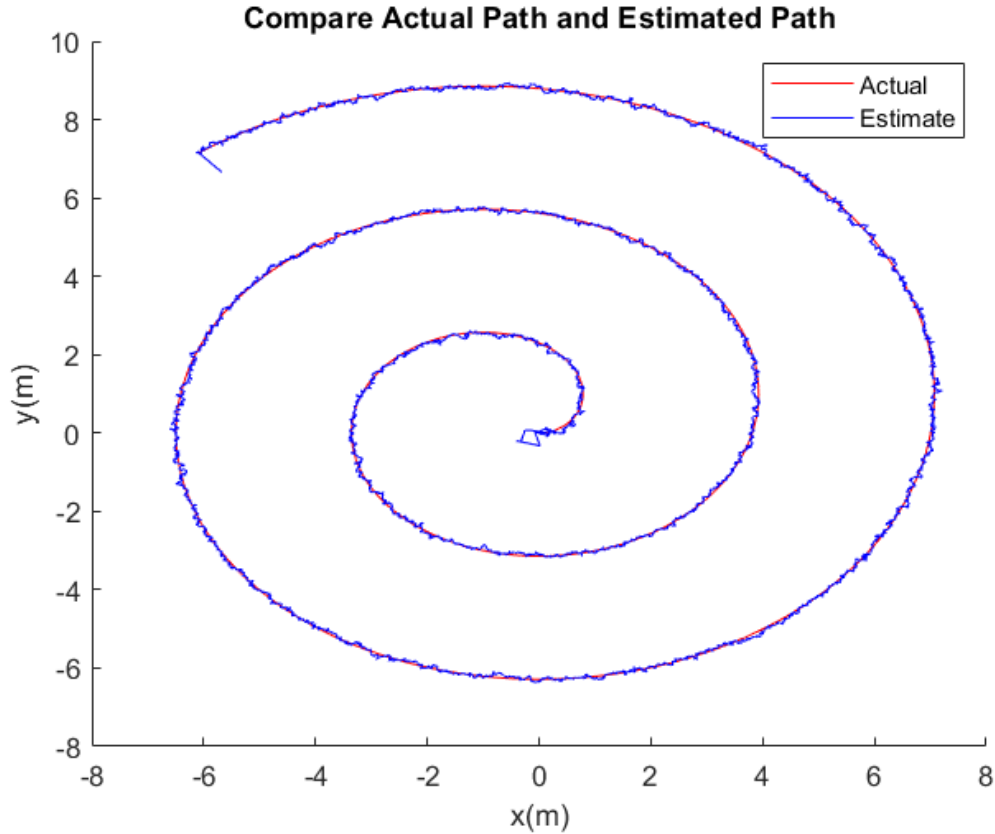


Figure 4: Trajectory of stationary object on image plane of camera as it pans and tilts

4.3 Camera Tracking







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

```

1 clear;
2 close all;
3 x = linspace(-20,20);
4 y = linspace(-20,20);
5 %% FOV FOCAL Length dependency
6 figure;
7 tiledlayout(2,2);
8 nexttile
9 hold on
10 for i = 1:length(x)
11     for j = 1:length(y)

```

```

12     meas = camera_model(0,0,0.2,[x(i);y(j)],[0,0,15]');
13     if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
14         plot(x(i),y(j),'k. ');
15     end
16 end
17 end
18 title("200 mm")
19 xlabel('x(m)')
20 ylabel('y(m)')
21 hold off
22
23 nexttile
24 hold on
25 for i = 1:length(x)
26     for j = 1:length(y)
27         meas = camera_model(0,0,0.15,[x(i);y(j)],[0,0,15]');
28         if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
29             plot(x(i),y(j),'k. ');
30         end
31     end
32 end
33 title("150 mm")
34 xlabel('x(m)')
35 ylabel('y(m)')
36 hold off
37
38 nexttile
39 hold on
40 for i = 1:length(x)
41     for j = 1:length(y)
42         meas = camera_model(0,0,0.1,[x(i);y(j)],[0,0,15]');
43         if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
44             plot(x(i),y(j),'k. ');
45         end
46     end
47 end
48 title("100 mm")
49 xlabel('x(m)')
50 ylabel('y(m)')
51 hold off
52
53 nexttile
54 hold on
55 for i = 1:length(x)
56     for j = 1:length(y)
57         meas = camera_model(0,0,0.05,[x(i);y(j)],[0,0,15]');
58         if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
59             plot(x(i),y(j),'k. ');
60         end
61     end
62 end
63 title("50 mm")
64 xlabel('x(m)')
65 ylabel('y(m)')

```

```

66 hold off
67 %sgtitle("Camera: 15 meters above ground")
68
69 %% FOV Pan angle dependency
70 figure;
71 tiledlayout(2,2);
72 nexttile
73 hold on
74 for i = 1:length(x)
75     for j = 1:length(y)
76         meas = camera_model(pi/2,0,0.1,[x(i);y(j)],[0,0,30]');
77         if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
78             plot(x(i),y(j),'k. ');
79         end
80     end
81 end
82 title("90 degrees")
83 xlabel('x(m)')
84 ylabel('y(m)')
85 axis square
86 hold off
87
88 nexttile
89 hold on
90 for i = 1:length(x)
91     for j = 1:length(y)
92         meas = camera_model(pi/6,0,0.1,[x(i);y(j)],[0,0,30]');
93         if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
94             plot(x(i),y(j),'k. ');
95         end
96     end
97 end
98 title("+30 degrees")
99 xlabel('x(m)')
100 ylabel('y(m)')
101 axis square
102 hold off
103
104 nexttile
105 hold on
106 for i = 1:length(x)
107     for j = 1:length(y)
108         meas = camera_model(-pi/6,0,0.1,[x(i);y(j)],[0,0,30]');
109         if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
110             plot(x(i),y(j),'k. ');
111         end
112     end
113 end
114 title("-30 degrees")
115 xlabel('x(m)')
116 ylabel('y(m)')
117 axis square
118 hold off
119

```

```

120 nexttile
121 hold on
122 for i = 1:length(x)
123     for j = 1:length(y)
124         meas = camera_model(pi,0,0.1,[x(i);y(j)],[0,0,30]');
125         if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
126             plot(x(i),y(j),'k. ');
127         end
128     end
129 end
130 title("180 degrees")
131 xlabel('x(m)')
132 ylabel('y(m)')
133 axis square
134 hold off
135
136 %% FOV tilt angle angle dependency
137 figure;
138 tiledlayout(2,2);
139 nexttile
140 hold on
141 for i = 1:length(x)
142     for j = 1:length(y)
143         meas = camera_model(0,-pi/4,0.05,[x(i);y(j)],[0,0,15]');
144         if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
145             plot(x(i),y(j),'k. ');
146         end
147     end
148 end
149 title("-45 degrees")
150 xlabel('x(m)')
151 ylabel('y(m)')
152 hold off
153
154 nexttile
155 hold on
156 for i = 1:length(x)
157     for j = 1:length(y)
158         meas = camera_model(0,-pi/12,0.05,[x(i);y(j)],[0,0,15]');
159         if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
160             plot(x(i),y(j),'k. ');
161         end
162     end
163 end
164 title("-15 degrees")
165 xlabel('x(m)')
166 ylabel('y(m)')
167 hold off
168
169 nexttile
170 hold on
171 for i = 1:length(x)
172     for j = 1:length(y)
173         meas = camera_model(0,pi/4,0.05,[x(i);y(j)],[0,0,15]');

```

```

174         if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
175             plot(x(i),y(j), 'k. ');
176         end
177     end
178 end
179 title("45 degrees")
180 xlabel('x(m)')
181 ylabel('y(m)')
182 hold off
183
184 nexttile
185 hold on
186 for i = 1:length(x)
187     for j = 1:length(y)
188         meas = camera_model(0,pi/12,0.05,[x(i);y(j)],[0,0,15]');
189         if(abs(meas(1)) > 0.05 || abs(meas(2)) > 0.05)
190             plot(x(i),y(j), 'k. ');
191         end
192     end
193 end
194 title("15 degrees")
195 xlabel('x(m)')
196 ylabel('y(m)')
197 hold off
198
199 %% Functions
200 function sensor_meas=camera_model(psi,phi,lambda,x_t,x_c)
201 %sensor model for the camera
202 %psi and phi are pan and tilt angles
203 %lambda is focal length of the parameter
204 %x_t is location of robot in inertial frame (must be a 3d vector)
205 %x_c is coordinates of the camera in the inertial frame (3d vector)
206 %returns sensor_meas which is a 2d vector (xp, yp)
207
208 R_phi = [1      0      0;
209          0 cos(phi) sin(phi);
210          0 -sin(phi) cos(phi)];
211 R_psi = [cos(psi) sin(psi) 0;
212          -sin(psi) cos(psi) 0;
213          0      0      1];
214 q_t = R_phi'*R_psi'*([x_t' 0]'-x_c);
215 p_t = lambda*[q_t(1)/q_t(3) q_t(2)/q_t(3)]';
216 sensor_meas = -p_t;
217 end

```

8 Appendix B: Camera Pan and Tilt Matlab Code

```

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

```

```

7  psi_array = linspace(0,pi/2,100);
8  t = linspace(0,5,100);
9  lambda = 2;
10 meas = zeros(2,100);
11 tiledlayout(1,2)
12 nexttile
13 %show camera dependence on tilt angle
14 for k = 1:100
15     psi = 0;
16     phi = phi_array(k); % constant tilt
17     meas(:,k) = camera_model(psi,phi,lambda,x_t,x_c);
18 end
19
20 plot(meas(1,:),meas(2,:))
21 xlabel("x [m]")
22 ylabel("y [m]")
23 title("Object position during tilt motion")
24 axis equal;
25 nexttile
26 %show camera dependence on pan angle
27 x_t = [1;1]; %fixed target state
28 x_c = [0;0;2]; %fixed camera position
29 phi_array = linspace(0,-pi/4,100);
30 psi_array = linspace(0,pi/4,100);
31 t = linspace(0,5,100);
32 lambda = 2;
33 meas = zeros(2,100);
34 for k = 1:100
35     phi = 0;
36     psi = psi_array(k); % constant pan
37     meas(:,k) = camera_model(psi,phi,lambda,x_t,x_c);
38 end
39 plot(meas(1,:),meas(2,:))
40 xlabel("x [m]")
41 ylabel("y [m]")
42 title("Object position during pan motion")
43 axis equal;
44
45 function sensor_meas=camera_model(psi,phi,lambda,x_t,x_c)
46 %sensor model for the camera
47 %psi and phi are pan and tilt angles
48 %lambda is focal length of the parameter
49 %x_t is location of robot in inertial frame (must be a 3d vector)
50 %x_c is coordinates of the camera in the inertial frame (3d vector)
51 %returns sensor_meas which is a 2d vector (xp, yp)
52
53 R_phi = [1      0      0;
54          0 cos(phi) sin(phi);
55          0 -sin(phi) cos(phi)];
56 R_psi = [cos(psi) sin(psi) 0;
57          -sin(psi) cos(psi) 0;
58          0      0      1];
59 q_t = R_phi'*R_psi'*([x_t' 0]'-x_c);
60 p_t = lambda*[q_t(1)/q_t(3) q_t(2)/q_t(3)]';

```

```

61 sensor_meas = -p_t;
62 end

```

9 Appendix C: Camera Tracking Matlab Code

```

1  close all;
2  clear;
3  %% Sensor Model
4  %camera parameters
5  focal = 0.085; % focal length
6  xc = [0;0;20]; %camera location
7  sx = 0.05; %image plane width
8  sy = 0.05; %image plane height
9  %sensor and process noise parameters
10 Rq = 0.000001;
11 Qsys = 0;
12 dt = 0.01;
13
14 %% Swirl Path
15 t = 0:dt:150; %setup the simulation of the car
16 v = 0.5; %
17 L = 4;
18 %simulate the motion of the car (sinusoidal path)
19 x = zeros(length(t),1);
20 y = zeros(length(t),1);
21 theta = zeros(length(t),1);
22 alpha = zeros(length(t),1);
23 pixels = zeros(length(t),2);
24
25 R = 1; % starting radius
26 for k = 1:length(t) - 1
27     grow = 1/R;
28     alpha(k) = atan(L*grow/(v));
29     x(k+1) = x(k) + v*cos(theta(k))*dt;
30     y(k+1) = y(k) + v*sin(theta(k))*dt;
31     theta(k+1) = theta(k) + (v)/L * tan(alpha(k))*dt;
32     R = R + grow*dt;
33     %generate simulated sensor measurements
34     sensor_meas = camera_model(0,0,0.085,[x(k+1);y(k+1)], [0;0;20], sx, sy)
35     ;
36     pixels(k,1) = sensor_meas(1) + randn(1,1)*sqrt(Rq);
37     pixels(k,2) = sensor_meas(2) + randn(1,1)*sqrt(Rq);
38
39
40 figure
41 hold on
42 plot(x,y, 'r')
43 title(" Actual Path")
44 xlabel('x(m)')
45 ylabel('y(m)')
46 hold off
47

```



```

48 figure
49 hold on
50 plot(pixels(1:length(t)-1,1),pixels(1:length(t)-1,2),'b')
51 title("Camera")
52 xlabel('x(m)')
53 ylabel('y(m)')
54 hold off
55
56 %% compute symbolic jacobian for camera model
57 syms phi psi xc_1 xc_2 xc_3 xt_1 xt_2 xt_3 lambda v theta alpha
58
59 R_phi = [1 0 0;
60          0 cos(phi) sin(phi);
61          0 -sin(phi) cos(phi)];
62
63
64 R_psi = [cos(psi) sin(psi) 0;
65          -sin(psi) cos(psi) 0;
66          0 0 1];
67
68 pt = R_phi'*R_psi'*([xt_1 ; xt_2; xt_3] - [xc_1; xc_2; xc_3]);
69
70 p = -lambda*[pt(1)/pt(3); pt(2)/pt(3)];
71 p_x = p(1);
72 p_y = p(2);
73
74 %compute the jacobian matrix
75 jac = jacobian([p_x, p_y],[xt_1,xt_2,xt_3, v, theta, alpha]);
76 % jac = [diff(p_x,xt_1), diff(p_x,xt_2), diff(p_x,xt_3),0,0,0;
77 %        diff(p_y,xt_1), diff(p_y,xt_2), diff(p_y,xt_3),0,0,0];
78 H_jac = matlabFunction(jac);
79
80
81 %% EKF Filter for Swirl Path
82 x0=[0;0;0;0;0;0];
83 P0=diag([10^2 10^2 10^2 10^2 10^2 10^2]);
84 xhat1p=x0;P1p = zeros(length(x0),length(x0),length(t)-1);P1p(:,:,1)=P0;
85 xhat1u=x0;P1u = zeros(length(x0),length(x0),length(t)-1);P1u(:,:,1)=P0;
86 xhat1u=x0;P1u(:,:,1)=P0;
87 Q = diag([0.000001,0.000001,0.000001,0.000001,0.000001,0.000001]); %
88     guess process noise covariance
89 R = diag([0.000001^2 0.000001^2]); %guess sensor noise covariance
90 Z = pixels'; %sensor measurements
91 for k=1:(length(t)-1)
92     %predict state
93     xhat1p(:,k+1)=predict_state_track(xhat1u(:,k),L,dt);
94     %predict covariance
95     [F,G]=getFG_carpouse_track(xhat1u(:,k),L,dt);
96     P1p(:,:,k+1) = F*P1u(:,:,k)*F' + G*Q*G';
97     %predict psi and phi values
98     psi = 0;
99     phi = 0;
100     %Kalman Gain
101     H = H_jac(focal,phi,psi,xc(1),xc(2),xc(3),xhat1p(1,k+1),xhat1p(2,k

```

```

+1),0);
101 K = P1p(:, :, k+1)*H'*inv(H*P1p(:, :, k+1)*H' + R);
102 %get sensor measurement from predicted data
103 meas = camera_model(psi, phi, focal, [xhat1p(1, k+1); xhat1p(2, k+1)], [xc
(1); xc(2); xc(3)], sx, sy);
104 %skips update step if target is outside the sensor fov
105 if isnan(meas) == false
106 xhat1u(:, k+1) = xhat1p(:, k+1) + K *(Z(:, k+1) - meas);
107 P1u(:, :, k+1) = (eye(6)-K*H)*P1p(:, :, k+1)*(eye(6)-K*H)' + K*R*K';
108 else
109 xhat1u(:, k+1) = xhat1p(:, k+1);
110 P1u(:, :, k+1) = P1p(:, :, k+1);
111 end
112 end
113 figure
114 hold on
115 plot(x, y, 'r')
116 plot(xhat1u(1, 1:length(t)), xhat1u(2, 1:length(t)), 'b')
117 title("Compare Actual Path and Estimated Path")
118 xlabel('x(m)')
119 ylabel('y(m)')
120 legend('Actual', 'Estimate')
121 hold off
122
123 %% Functions
124 function Xkp1=predict_state_track(Xk, L, dt)
125 %
126 % For object tracking
127 % discrete prediction of state
128 % This is now 6 states, since the control inputs are added on as state
129 % vectors and the z coordinate is appended onto the end as a constant
value
130
131 v=Xk(4);
132 alpha=Xk(5);
133 tk = Xk(3);
134
135 Xkp1 = Xk +...
136 dt*[v*cos(tk);
137 v*sin(tk);
138 v/L * tan(alpha);
139 0;
140 0;
141 0];
142 end
143
144
145 function [F,G]=getFG_carpose_track(X, L, dt)
146 %
147 % For object tracking
148 % get the linearized system matrixes F,G
149 %
150
151 v=X(4);

```

```

152 alpha=X(5);
153 tk = X(3);
154 F=[1 0 -dt*v*sin(tk) dt*cos(tk) 0 0;
155     0 1 dt*v*cos(tk) dt*sin(tk) 0 0;
156     0 0 1 dt*1/L * tan(alpha) dt*v/L*(sec(alpha))^2 0;
157     0 0 0 1 0 0;
158     0 0 0 0 1 0;
159     0 0 0 0 0 1];
160
161 G=[dt 0 0 0 0 0;
162     0 dt 0 0 0 0;
163     0 0 dt 0 0 0;
164     0 0 0 dt 0 0;
165     0 0 0 0 dt 0;
166     0 0 0 0 0 0]; %no noise measurements in z coordinate
167 end
168
169
170 function sensor_meas=camera_model(psi,phi,lambda,x_t,x_c,sx,sy)
171 %sensor model for the camera
172 %psi and phi are pan and tilt angles
173 %lambda is focal length of the parameter
174 %x_t is location of robot in inertial frame (must be a 3d vector)
175 %x_c is coordinates of the camera in the inertial frame (3d vector)
176 %returns sensor_meas which is a 2d vector (xp, yp)
177
178 R_phi = [1 0 0;
179           0 cos(phi) sin(phi);
180           0 -sin(phi) cos(phi)];
181 R_psi = [cos(psi) sin(psi) 0;
182           -sin(psi) cos(psi) 0;
183           0 0 1];
184 q_t = R_phi'*R_psi'*([x_t' 0]' - x_c);
185 p_t = lambda*[q_t(1)/q_t(3) q_t(2)/q_t(3)]';
186 if abs(p_t(1)) <= sx && abs(p_t(2)) <= sy
187     sensor_meas = -p_t;
188 else
189     sensor_meas = NaN;
190 end
191 end

```