**Camera Control Problem**

Chuanwei Wu

*Abstract* – In this project, I am asked to design a camera controller, so the camera will be able to catch the target and following the target. The motion of target will be recorded and stored for the further analysis. The trajectory of the target is plotted to compare with the input trajectory. The pan and tilt angle change during the test is also plotted as result shown in the following part of the paper. The controller designed in this project is user friendly as the controller can catch any input trajectory if the initial point of the trajectory is at the middle of the workspace, aka (50,50).

1. **INTRODUCTION AND THEORY**

The camera monitor is widely used and has shown promise for sensor and robotics application. In this project, one camera is used to monitor and capture the trajectory of the target. The motion of the target is pre-determined but not known by the monitor system. The trajectory of the target is generated by using the *formula 1*,

. *Formula 1*

is denoted as the input linear velocity of the target and is denoted as the input angular velocity.

The camera monitor is able to pan and tilt angles, which can be utilized to follow the target in the workspace. The pan and tilt angle are represented by the Euler angles know as yaw () and roll () angles, respectively, and illustrated by the *figure 1*.

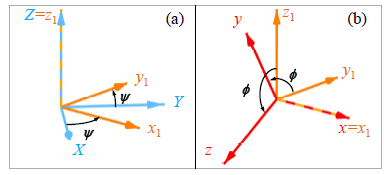


Figure 1 The illustration of pan and tilt angle is shown in the subfigure a and subfigure b.

As defined, the yaw and roll Euler angles are defined by two successive right-hand rotations: a rotation by an angle about the Z-axis, leading to the intermediate frame in *figure 1.a*, followed by a rotation by an angle about the x1-axis, leading to the camera-fixed frame in *figure 1.b*.

The current time location will be transformed into the camera based 3D frame by using the *formula 2*,

. *Formula 2*

is the position of the camera monitor with respect to the inertial frame, in this project, this parameter is set to be . and are calculated based on the given *formula 3*,

. *Formula 3*

The virtual image plane will be transformed from the camera based 3D frame by using the *formula 4*,

*Formula 4*

is the focal length of the camera, in this project, this parameter is set to be .

The only input of the monitor system is the frame obtained by the camera, while this information is calculated from the previous formulas. This piece of information will be used to calculate backward to the real location of the target in the inertial frame. Under the calculation described before, the vector of location of the target in the workspace will be recorded. By using *formula 5*,

. *Formula 5*

The location changed in the workspace of the target is considered to be the velocity of the target. In the constant time interval the velocity is treated as a constant value to simplify the problem. The target velocity in the workspace can also be found by calculating the target position change in the virtual image plane. The calculation is done according to the *formula 6*,

*Formula 6*

in the previous formula is the image Jacobian matrix, as shown in the *formula 7*,

*Formula 7*

By finding the velocity of the target in the workspace, we can tell where the camera should be pointing at for the next time stamp position. Using this piece of information, the pan and tilt angle of the camera will be found and the difference between where we want the camera to be and where the camera is at currently can be found also. These four numbers make up the information, defined as. Given the formula 8, we can calculate the camera control parameter.

*Formula 8*

In the previous formula, A and B are defined accordingly as shown,

, where b1 and b2 are two constant motor parameters, setting to be 100­­­°/ (Vs2).

Take the kinematic constraint into consideration, the camera state and control must also obey inequality constraints that reflect physical bounds imposed by the instrumentation. The pan and tilt angles are constrained to the ranges, and the pan and tilt angular velocities are bounded by the constants, respectively. Also both element in the vector u is bound between -1 and 1.

By setting the camera size a and b, we should always check if the target is in the virtual image plane. To accomplish this task, we can calculate S(t), the projection of pT on the workspace. If the position of the target X­T is in the range of S(t), then we can conclude that the target image will always be in the virtual image plane.

In this project, we assume the camera measurements are noise free and the sampling interval is a known finite constant. The camera is also assumed to capture target position and velocity in the same time, and all the information will be stored in the field of view.

1. **Numerical Methods**

In the beginning of the code, the length of test is set to be 1000, which indicating the time period will be tested on. Time interval is set to be one second for easier calculation in the code. The workspace in my design is set to be a 100\*100 square 2-D place, and the camera is set to be at the center of the room (50,50) and 3m over the bottom. The focal length of the camera is set to be 3m, while the initial angular position of the camera is set to be (0,π), indicating that the camera is pointing downwards without pan angle changed yet.

The simulation route is also set. The first route is set to be a straight line. If the controller is designed properly, the pan angle of the straight line should be keeping constant while the tilt angle should be decreasing. The second route is set to be a half circle. If the controller is designed properly, the pan angle of the half circle should be linearly decreasing while the tilt angle should be keeping constant. The third route is set to be a spiral out curve. If the controller is designed properly, the pan angle of the spiral out curve should be periodically, similar to but not exactly the sinusoidal wave, while the tilt angle should be keeping increasing. The result figures is shown in the result section, which indicates that the assumption is met.

After all the initial setup is done, the virtual image plane information will be captured by the camera and stored. We can use the formulas in the introduction and theory section to figure out where the target is in the workspace. Compare the two successive target position, we can have the velocity of the target at certain time. At this part, we can also use the *formula 6* and *formula 7* to find the same information. In this project, both methods are conducted, and they both works fine with the given task.

As the velocity of the target is obtained, we can estimate the next position of the target and tell the camera to move to that point.

To accomplish this, we have to use controller function to calculate out the value of u by using the *Formula 8*. In the controller function, based on the estimated target position, the needed pan and tilt angle will be calculated. If the output of the function, u, is directly delivered to the system to calculate the motion of the camera, the result will be pretty bad as the result of over step. The result figure will have a lot of zigzag behavior in the curve. To avoid this, I used PID controller idea to smooth the camera motion.

1. **Results**

Three simulation is done by using the designed controller.

The first simulation is a straight line. The reason of choosing a straight line as simulation is that it is easy to test if the controller is behaving properly. In the *figure 2*, we can tell that the pan angle is not changing while the tilt angle is decreasing linearly in the beginning part and decrease rate is decaying as the target moving away from the middle of the workspace. All these fulfill the expectation, which is indicating the robustness of the controller. The designed trajectory of the simulation is matching the actual measured trajectory in *figure 3*, which is also indicating the soundness of the controller and the related algorithm.

|  |  |
| --- | --- |
| C:\Users\cit-labs\AppData\Local\Microsoft\Windows\INetCache\Content.Word\straight line.jpg  Figure 2 The pan and tilt angle change based on the straight line trajectory. | C:\Users\cit-labs\AppData\Local\Microsoft\Windows\INetCache\Content.Word\straight line_in.jpg  Figure 3 The designed trajectory compared to the measured trajectory of a straight line. |

The second simulation is a half circle. The reason of choosing a half circle as simulation is that it is easy to test if the controller is behaving properly. In the *figure 4*, we can tell that the pan angle is decreasing linearly while the tilt angle is keeping constant. All these fulfill the expectation, which is indicating the robustness of the controller. The designed trajectory of the simulation is matching the actual measured trajectory in *figure 5*, which is also indicating the soundness of the controller and the related algorithm.

|  |  |
| --- | --- |
| C:\Users\cit-labs\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Circle.jpg  Figure The pan and tilt angle change based on the half circle trajectory. | C:\Users\cit-labs\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Circle_in.jpg  Figure The designed trajectory compared to the measured trajectory of half circle. |

The third simulation is a spiral out curve. In the *figure 6*, we can tell that the pan angle behaves periodically, similar to but not exactly the sinusoidal wave while the tilt angle is decreasing quicker in the beginning part and decrease rate is decaying as the target moving away from the middle of the workspace. All these fulfill the expectation, which is indicating the robustness of the controller. The designed trajectory of the simulation is matching the actual measured trajectory in *figure 5*, which is also indicating the soundness of the controller and the related algorithm.

|  |  |
| --- | --- |
| C:\Users\cit-labs\AppData\Local\Microsoft\Windows\INetCache\Content.Word\spiral.jpg  Figure The pan and tilt angle change based on the spiral out curve trajectory. | C:\Users\cit-labs\AppData\Local\Microsoft\Windows\INetCache\Content.Word\spiral_in.jpg  Figure The designed trajectory compared to the measured trajectory of spiral out curve. |

1. **Conclusions**

In this project, the camera controller is designed. By using this controller, the camera is able to follow the target in the workspace and record its trajectory. For all the input trajectories, the camera is able to catch the target, which is indicating the robustness of the algorithm and the controller. The target is assumed to be in the camera frame, which may not be case in the real world application. In the future edition of the controller, the camera should be able to scan the workspace first, so it can find the target itself. To simplify the project, noise is not existing in this project. In the real world, how to get rid of the noise is actually a large part of the camera function. So it might be more considerate if the noise can be taken into consideration for the future edition of the controller.

**Appendix**

Main code

clc

clear

close all

testlength=1000;

deltat=1;

Xc=[50;50;3];

s=zeros(4,testlength);

% s(:,1)=[5/4\*pi;pi/2+0.0424;0;0];

s(:,1)=[0;pi;0;0];

lmd=3;

%

Xt(1,:)=linspace(50,80,testlength);

Xt(2,:)=linspace(50,80,testlength);

x=linspace(-1\*pi\*1,0,testlength);

r=20;

Xt(1,:)=sin(x(1:testlength)).\*r+50;

Xt(2,:)=cos(x(1:testlength)).\*r+50;

x=linspace(-1\*pi\*2,pi\*2,testlength);

r=linspace(0,10,testlength);

Xt(1,:)=sin(x(1:testlength)).\*r(1:testlength)+50;

Xt(2,:)=cos(x(1:testlength)).\*r(1:testlength)+50;

plot(Xt(1,2:end),Xt(2,2:end),'r.','MarkerSize',19)

Xt\_est=zeros(2,testlength);

Xt\_est(:,1)=[50;50];

dXt=[0;0];

u=[0;0];

hold on

xlim([0,100])

ylim([0,100])

for i=1:testlength

Rphi=[1 0 0;

0 cos(s(2,i)) sin(s(2,i));

0 -sin(s(2,i)) cos(s(2,i))];

Rpsi=[cos(s(1,i)) sin(s(1,i)) 0;

-sin(s(1,i)) cos(s(1,i)) 0;

0 0 1];

Qt(:,i)=Rphi\*Rpsi\*([Xt(:,i);0]-Xc);

Pt(:,i)=lmd\*[Qt(1,i)/Qt(3,i);Qt(2,i)/Qt(3,i)];

temp=Rpsi'\*(Rphi'\*[Pt(:,i);lmd]);

t=-3/temp(3);

Xt(:,i)=[temp(1)\*t+50;temp(2)\*t+50];

if i>=2

dXt=Xt(:,i)-Xt(:,i-1);

% H=[-lmd/Qt(3,i) 0 Pt(1,i)/Qt(3,i) Pt(1,i)\*Pt(2,i)/lmd -(lmd^2+Pt(1,i)^2)/lmd Pt(2,i);

% 0 -lmd/Qt(3,i) Pt(2,i)/Qt(3,i) (lmd^2+Pt(1,i)^2)/lmd -Pt(1,i)\*Pt(2,i)/lmd -Pt(1,i)];

% dPt(:,i)=H\*cat(1,cat(2,Rphi'\*Rpsi',zeros(3)),cat(2,zeros(3),-Rphi'))\*[dXt(:,i);0;s(3,i);0;s(4,i)];

end

Xt\_est(:,i+1)=dXt+Xt(:,i);

u=cam\_controller\_1(Xt\_est(:,i+1),s(:,i),u);%,u,deltat);

s(3,i+1)=s(3,i)+(100/180\*pi)\*u(1);

% s(3,i+1)=min(s(3,i+1),(100/180\*pi));

s(4,i+1)=s(4,i)+(100/180\*pi)\*u(2);

% s(4,i+1)=min(s(4,i+1),(100/180\*pi));

s(1,i+1)=s(1,i)+s(3,i);

if s(1,i+1)>2\*pi

s(1,i+1)=s(1,i+1)-2\*pi;

end

s(2,i+1)=s(2,i)+s(4,i);

end

plot(Xt\_est(1,2:end),Xt\_est(2,2:end),'k--','Linewidth',2)

set(gca,'FontSize',28)

legend('Input trajectory','Detected trajectory')

figure

alf=30;

D=zeros(testlength-1,testlength);

for i=1:testlength-1

D(i,i)=-1;

D(i,i+1)=1;

end

A=zeros(2\*testlength-1,testlength);

A(1:testlength,:)=eye(testlength);

A(testlength+1:end,:)=alf\*D;

b=[s(1,1:testlength)';zeros(testlength-1,1)];

y\_corr\_1=pinv(A)\*b;

plot(1:820,y\_corr\_1(81:900)\*180/pi,'k--')

xlim([1,820])

ylim([0,360])

set(gca,'FontSize',28)

hold on

plot(1:40:820,y\_corr\_1(81:40:900)\*180/pi,'bs','MarkerSize',9)

alf=30;

D=zeros(testlength-1,testlength);

for i=1:testlength-1

D(i,i)=-1;

D(i,i+1)=1;

end

A=zeros(2\*testlength-1,testlength);

A(1:testlength,:)=eye(testlength);

A(testlength+1:end,:)=alf\*D;

b=[s(2,1:testlength)';zeros(testlength-1,1)];

y\_corr\_2=pinv(A)\*b;

plot(1:testlength,y\_corr\_2(1:testlength)\*180/pi,'k-.')

plot(1,s(2,1)\*180/pi,'b\*','MarkerSize',9)

plot(40:40:testlength,y\_corr\_2(40:40:testlength)\*180/pi,'b\*','MarkerSize',9)

legend('Pan angle','Sample pan angle points','Tilt angle','Sample tilt angle points')

Controller function

function u=cam\_controller\_1(Xt,s\_old,u\_old)

if Xt(1)>=50&&Xt(2)>=50

angle=3/2\*pi+atan((Xt(2)-50)/(Xt(1)-50));

elseif Xt(1)>=50&&Xt(2)<50

angle=3/2\*pi+atan((Xt(2)-50)/(Xt(1)-50));

elseif Xt(1)<50&&Xt(2)>=50

angle=pi/2+atan((Xt(2)-50)/(Xt(1)-50));

elseif Xt(1)<50&&Xt(2)<50

angle=pi/2+atan((Xt(2)-50)/(Xt(1)-50));

end

s(1)=angle;

s(2)=pi/2+atan(3/sqrt((Xt(1)-50)^2+(Xt(2)-50)^2));

s(3)=(s(1)-s\_old(1));

s(4)=(s(2)-s\_old(2));

u(1)=(s(3)-s\_old(3))/(100/180\*pi);

u(2)=(s(4)-s\_old(4))/(100/180\*pi);

du(1)=u(1)-u\_old(1);

du(2)=u(2)-u\_old(2);

u(1)=0.29\*u\_old(1)+du(1);

u(2)=0.29\*u\_old(2)+du(2);

u(1)=max(min(u(1),1),-1);

u(2)=max(min(u(2),1),-1);

end

**Reference**

[1] S. Ferrari, *Midterm Project: Camera Control Problem*