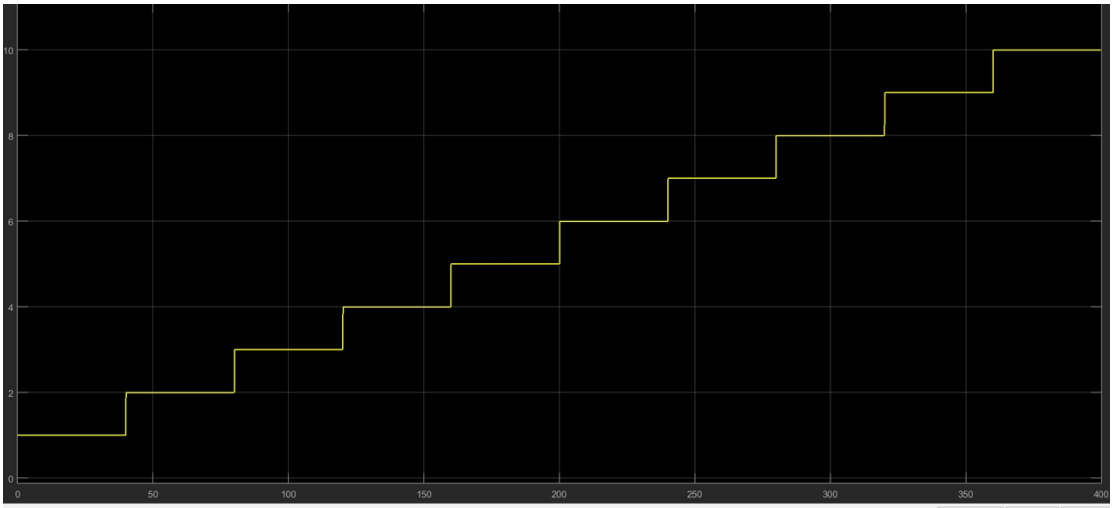
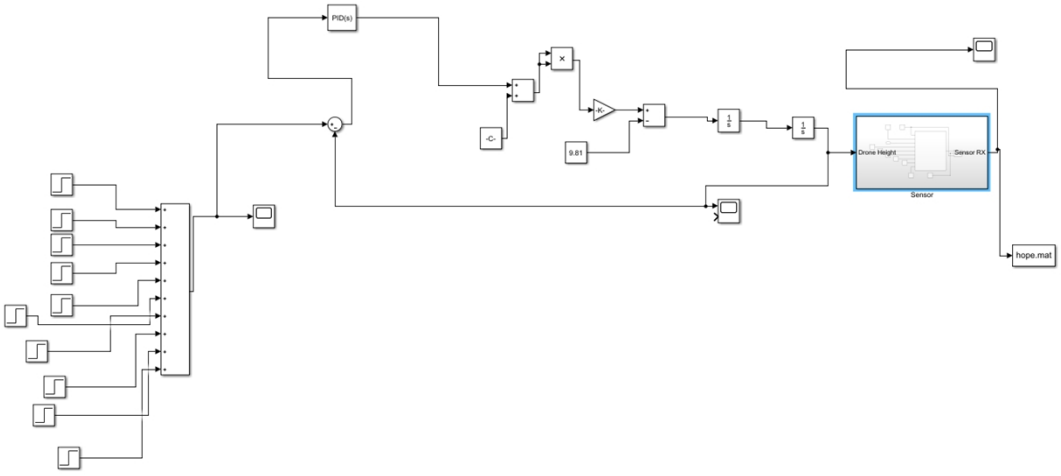
USAGE: please run sensor_model.slx first to generate ‘hope.mat’ first, and then use ‘code.m’ to process ‘hope.mat’ as its input.

REPORT

This is an implementation of the second project from the two options, namely, the ee113 and ee141 combined project. We first hook up the sensor module provided with the controller we designed in ee141, then we give the whole system a combination of step inputs. The scope of the inputs we use is shown below:



The input is added by another step every 40 seconds. So our drone will move to an altitude and stay there for 40 seconds to let the sensor take record of the signal. We then export the output from the sensor into a file which we named ‘hope.mat’, and the time and signal are stored in 2D array. The overall simulink design is shown below. The to file block is on the lower right corner.



After we get the 2D array of corresponding times and signal values (from ‘hope.mat’), we use the average filter technology we learnt in ee113 to filter the input, namely, de-noise the input.

```
c1=load('hope.mat')
Time=c1.ans(1,:)
amplitude=c1.ans(2,:)
%for windowSize=[1 5 10 50 100]
windowSize = 1000;
b = (1/windowSize)*ones(1,windowSize);
a = 1;
y = filter(b,a,amplitude);
[val, idx] = max(y);
```

Here, we choose the window size to be 1000, after testing window size of 1,5,10,50,100 and 1000. And after passing through the filter, we will get an array y that contains all filtered signal. Now from the equation below we know that the power of the signal will only reach the maximum when the drone is closest to the position of the source.

$P_{r,l} = \frac{k_l P_t}{d_l^{\alpha}}$ .....(1)

Where $\alpha = 2$ in the above equation. If we click into the sensor module, we will see that the distance is calculated as below

```
% Drone-Sensor Distance
d = sqrt((x_d+x_s)^2 + (y_d-y_s)^2);
```

We know that our drone is moving along the line x=0, thus x_d+x_s will not change. Since y_s is also a constant, the distance will reache its minimum when y_d – y_s reaches its minimum, which is when y_d=y_s. And this point corresponds to the maximum of the power of signal. Thus we can find the maximum power from the filtered array y and calculate y_source based on the time elapsed.

```
y = filter(b,a,amplitude);
[val, idx] = max(y);
%TF = islocalmax(y);
%plot(Time,y,Time(TF),y(TF),'r*')
plot(Time,y,Time(idx),y(idx),'r*')
ysource=1+idx/(2000*40);
ysource=floor(ysource);
```

Here, we will get the value of ysource to be 4. After sanity check, we find out that this is the same value used in the sensor module. Now we want to calculate xsource. To calculate xsource, we only need a ratio between two distances.

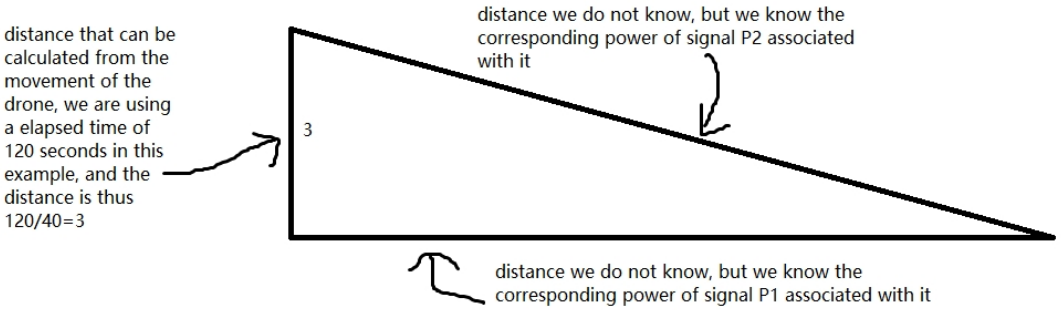


Figure 1. the overall structure to calculate the distance

Thus we can use the ratio between the power of signals to deduce the angle of the triangle, and then use arccos and tan function to find out the distance xsource, which is implemented as x_d+x_s in the sensor module.

```
sum=0;
start=(ysource-1)*40+10;
for i=[start]% 80 120 160 200]
    for j=i*2000:10:i*2000+2000*5
        sum=sum+y(j)^2
    end
end
p1=sum/1000;
```

Above is the way we calculate the power of the signal. We will take samples while the drone is at a certain altitude over a time period of 5 seconds, which corresponds to 5*2000 samples. Here we make the step size to be 10 so that the computation is less intense. From here, we will be able to get the average power p1 corresponding to the lower side of the triangle in figure 1. Similarly, we can calculate the power p2 that is associated with the upper right side of the triangle in figure 1. After that, we will be able to obtain the ratio.

```
ratio=p1/p2;

xsource= 3/(tan(acos(sqrt(1/ratio))));
xsource=floor(xsource);
```

The ratio of the power of the signal is the ratio of the squared ratio of the two sides of the triangle in Figure 1. (this can be seen from equation (1) we mentioned earlier) To visualize, in our case the ratio of power is 1.9895, and the ratio of distance is thus sqrt(1.9895). With this ratio of distance, we can thus calculate x_source.

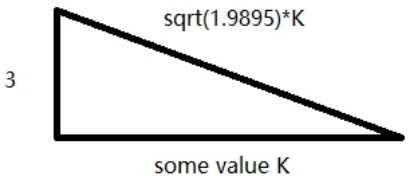


Figure 2. A visualization of the triangle with ratio

The x_source, which is K in figure 2, is calculated to be 3.01588. And to sanity check, we looked into the implementation of the sensor module and find out x_source is implemented as x_s+x_d=2+1=3. Which is very close to the value 3.01588 we obtained. Finally, we display our results according to the spec

```
disp(['The estimated value of ysource is ' int2str(ysource)]);
disp(['The ysource value fed into the sensor module is ' int2str(4)]);
disp(['The estimated value of xsource is ' int2str(xsource)]);
disp(['The xsource value fed into the sensor module is x_source+x_drone, which is 2+1=3']);
```

Answer to open-ended question: I used average filter to de-noise. I have tried different values of the window size and 1000 works the best. I have also tried exponential filter to de-noise, but I think average filter performs better because it takes into account a larger range of neighboring points.