Derek Xu
UID: 704751588

1)

Notice the domain and range of the tangent and the arctangent function:

$$tan: [0, 2\pi] \rightarrow [-\infty, \infty]$$

$$tan^{-1}: [-\infty, \infty] \rightarrow [0, 2\pi]$$

Given the following equation:

$$\beta = \tan^{-1}\left(\frac{l_r}{l_r + l_f}\tan(u)\right)$$

Let us perform some algebraic manipulation:

$$\tan(\beta) = \frac{l_r}{l_r + l_f}\tan(u)$$

$$u = \tan^{-1}\left(\frac{l_r + l_f}{l_r}\tan(\beta)\right)$$

We can conclude that if we bound β in range $[0, 2\pi]$, we achieve the following:

$$\frac{l_r + l_f}{l_r}\tan(\beta) \in [-\infty, \infty]$$

This results in the following assertion being true:

$$u = \tan^{-1}\left(\frac{l_r + l_f}{l_r}\tan(\beta)\right) \in [0, 2\pi]$$

Thus, we have proved that for any desired $\beta \in [0, 2\pi]$, $\exists\, u \in [0, 2\pi]$

2)

Given the following equations:

$$\dot{x} = v\, cos(\psi + \beta)$$

$$\dot{y} = v\, sin(\psi + \beta)$$

$$\dot{\psi} = \frac{v}{l_r}\, sin(\beta)$$

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\psi}\end{bmatrix} = f(x, y, \psi, \beta) = f_1(x, y, \psi) + f_2(\beta)$$

Let us perform linearization:

$$\begin{bmatrix}\dot{x} + \dot{\Delta x}\\ \dot{y} + \dot{\Delta y}\\ \dot{\psi} + \dot{\Delta\psi}\end{bmatrix} = f(x + \Delta x, y + \Delta y, \psi + \Delta\psi, \beta + \Delta\beta)$$

Derek Xu
UID: 704751588

$$\begin{bmatrix} \dot{x} + \dot{\Delta x} \\ \dot{y} + \dot{\Delta y} \\ \dot{\psi} + \dot{\Delta \psi} \end{bmatrix} \approx f(x, y, \psi, \beta) + \frac{\partial f}{\partial x, y, \psi} \Delta_{x,y,\psi} + \frac{\partial f}{\partial \beta} \Delta\beta$$

$$\dot{\Delta}_{x,y,\psi} = \frac{\partial f}{\partial x, y, \psi} \Delta_{x,y,\psi} + \frac{\partial f}{\partial \beta} \Delta\beta$$

$$\frac{\partial f}{\partial x, y, \psi} = \begin{bmatrix} 0 & 0 & -v\ sin(\psi_r + \beta_r) \\ 0 & 0 & v\ cos(\psi_r + \beta_r) \\ 0 & 0 & 0 \end{bmatrix}, \qquad \frac{\partial f}{\partial \beta} = \begin{bmatrix} -v\ sin(\psi_r + \beta_r) \\ v\ cos(\psi_r + \beta_r) \\ \frac{v}{l_r} \cos(\beta_r) \end{bmatrix}$$

$$\therefore \begin{bmatrix} \dot{\Delta x} \\ \dot{\Delta y} \\ \dot{\Delta \psi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -v\ sin(\psi_r + \beta_r) \\ 0 & 0 & v\ cos(\psi_r + \beta_r) \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \psi \end{bmatrix} + \begin{bmatrix} -v\ sin(\psi_r + \beta_r) \\ v\ cos(\psi_r + \beta_r) \\ \frac{v}{l_r} \cos(\beta_r) \end{bmatrix} \Delta\beta$$

$$\begin{bmatrix} \dot{\Delta x} \\ \dot{\Delta y} \\ \dot{\Delta \psi} \end{bmatrix} = \begin{bmatrix} -v\ sin(\psi_r + \beta_r) & -v\ sin(\psi_r + \beta_r) \\ v\ cos(\psi_r + \beta_r) & v\ cos(\psi_r + \beta_r) \\ 0 & \frac{v}{l_r} \cos(\beta_r) \end{bmatrix} \begin{bmatrix} \Delta\psi \\ \Delta\beta \end{bmatrix}$$

This is the linearized equation. I define values for ψ_r and β_r in the next part.

3)

Given the following definitions:

$$x = a + bt$$

$$y = c + dt$$

$$a, b, c, d \in \mathbb{R}$$

We define the following references:

$$\beta_r = 0$$

$$\psi_r = \tan^{-1}\left(\frac{d}{b}\right)$$

Notice the following simplifications:

$$\begin{bmatrix} \dot{\Delta x} \\ \dot{\Delta y} \\ \dot{\Delta \psi} \end{bmatrix} = \begin{bmatrix} -v\ sin\left(\tan^{-1}\left(\frac{d}{b}\right)\right) & -v\ sin\left(\tan^{-1}\left(\frac{d}{b}\right)\right) \\ v\ cos\left(\tan^{-1}\left(\frac{d}{b}\right)\right) & v\ cos\left(\tan^{-1}\left(\frac{d}{b}\right)\right) \\ 0 & \frac{v}{l_r} \cos(0) \end{bmatrix} \begin{bmatrix} \Delta\psi \\ \Delta\beta \end{bmatrix}$$

Notice the following properties (either way these values become constants):

$$-v\ sin\left(\tan^{-1}\left(\frac{d}{b}\right)\right) = -v\ \frac{d}{v} = -d, \qquad v\ cos\left(\tan^{-1}\left(\frac{d}{b}\right)\right) = v\ \frac{b}{v} = b$$

Derek Xu
UID: 704751588

$$\therefore \begin{bmatrix} \dot{\Delta x} \\ \dot{\Delta y} \\ \dot{\Delta \psi} \end{bmatrix} = \begin{bmatrix} -d & -d \\ b & b \\ 0 & \frac{v}{l_r} \end{bmatrix} \begin{bmatrix} \Delta\psi \\ \Delta\beta \end{bmatrix}$$

Now notice the following:

$$\dot{\Delta\psi} = \frac{v}{l_r}\Delta\beta \rightarrow \ \Delta\psi = \int \frac{\mathrm{v}}{\mathrm{l_r}}\Delta\beta \, dt$$

$$\dot{\Delta x} = -d(\Delta\psi + \Delta\beta), \qquad \dot{\Delta y} = b(\Delta\psi + \Delta\beta)$$

$$\Delta x = -d\int \left(\int \frac{\mathrm{v}}{\mathrm{l_r}}\Delta\beta \, dt + \Delta\beta\right) dt\,, \qquad \Delta \mathrm{y} = b\int \left(\int \frac{\mathrm{v}}{\mathrm{l_r}}\Delta\beta \, dt + \Delta\beta\right) dt$$

$$y = y_r + \Delta y, \qquad x = x_r + \Delta x, \qquad \beta = \beta_r + \Delta\beta, \qquad \psi = \psi_r + \Delta\psi$$

We can see here that x, y, and ψ become solutions to β and v.

Proof of Time Invariance:

$$\begin{bmatrix} \dot{\Delta x}(t) \\ \dot{\Delta y}(t) \\ \dot{\Delta \psi}(t) \end{bmatrix} = \begin{bmatrix} -d & -d \\ b & b \\ 0 & \frac{v}{l_r} \end{bmatrix} \begin{bmatrix} \Delta\psi(t) \\ \Delta\beta(t) \end{bmatrix}$$

Notice that all matrix elements are constants, thus it is trivial to show:

$$\begin{bmatrix} \dot{\Delta x}(t+\tau) \\ \dot{\Delta y}(t+\tau) \\ \dot{\Delta \psi}(t+\tau) \end{bmatrix} = \begin{bmatrix} -d & -d \\ b & b \\ 0 & \frac{v}{l_r} \end{bmatrix} \begin{bmatrix} \Delta\psi(t+\tau) \\ \Delta\beta(t+\tau) \end{bmatrix}$$

$\therefore$ System is Time Invariant

4.

Let us explore the case of Δy:

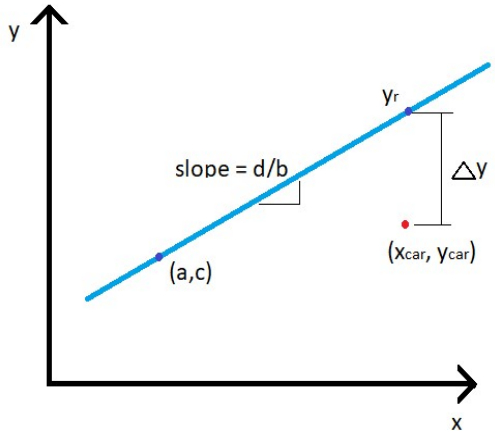
We must assign Δy to be some value we would like to minimize (some error).

Recall, in our linearized model we have stated that $y_r + \Delta y = y$.

The question now is what our y_r refers to. We can easily set y_r to be the **trajectory** as a function of t ($y_r = c + dt$). But the following issue arises. If the initial state of our car, x_o, y_o, ψ_o, is different than the initial state of the trajectory, we will never be able to meet the trajectory point. This is because given that Δy refers to the y-displacement of our car from the trajectory at a point in time, t, and our car is initially displaced away from the trajectory, for the car to perfectly follow the trajectory, it must drive distance, $l_{disp} + l_{traj}$, where l_{dist} is the initial displacement from the initial trajectory, and l_{traj} which is the displacement the trajectory is at from the initial displacement. If $l_{dist} + l_{traj} > l_{traj}$, our car will never be able to meet the trajectory as the trajectory is moving at velocity $d^2 + b^2$, and our car can travel only at the velocity $d^2 + b^2$, thus the car will never be able to compensate its initial displacement from the trajectory, and the steady state error for both Δy and Δx will never be 0. In the case, where we only track Δy, we will achieve a line parallel to the trajectory **not on the trajectory**.

Derek Xu
UID: 704751588

Thus our y_r should refer to the **path of the trajectory**. In this case, $y_r = \frac{d}{b}(x_{car} - a) + c$, in other words, y_r is simply where the car *should* be at, given the current value of x for the car. In this case, we can minimize Δy because it is no long relative to the trajectory, rather the path of the trajectory. We can now achieve a steady state error of 0, for both Δy and Δx. Furthermore, we can do this by only using a controller on Δy. We can see this in the following graph where the blue line refers to the path of the trajectory, and the red dot refers to the location of the car:



Formally, we have the following definitions:

$$\Delta y = y_r - y_{car} = \frac{d}{b}(x_{car} - a) + c - y_{car}$$

Now we have one other issue: y_r and y_{car} are not constant are varying with time in our system.

We must be able to track the locations of both x_{car} and y_{car} to make this system work.

Recall, in our linearized model:

$$\begin{bmatrix} \Delta\dot{x} \\ \Delta\dot{y} \\ \Delta\dot{\psi} \end{bmatrix} = \begin{bmatrix} -d & -d \\ b & b \\ 0 & \frac{v}{l_r} \end{bmatrix} \begin{bmatrix} \Delta\psi \\ \Delta\beta \end{bmatrix}$$

Let us obtain the transfer function $\frac{\Delta Y}{\Delta B}$, by applying Laplace to our linear model:

$$\Delta\dot{\psi} = \frac{v}{l_r}\Delta\beta$$

$$s\Delta\Psi - \psi(0) = \frac{v}{l_r}\Delta B$$

$$\Delta\Psi = \frac{v}{s\, l_r}\Delta B + \frac{\psi(0)}{s}$$

$$\Delta\dot{y} = b(\Delta\psi + \Delta\beta)$$

$$s\Delta Y - y(0) = b(\Delta\Psi + \Delta B)$$

Derek Xu
UID: 704751588

$$\Delta \mathrm{Y} = \frac{b\left(\frac{v}{s\, l_r}\Delta \mathrm{B} + \frac{\psi(0)}{\mathrm{s}} + \Delta B\right) + y(0)}{s}$$

$$\Delta \dot{x} = -d(\Delta\psi + \Delta\beta)$$

$$\mathrm{s}\Delta \mathrm{X} - x(0) = -d(\Delta\Psi + \Delta B)$$

$$\Delta \mathrm{X} = \frac{-d\left(\frac{v}{s\, l_r}\Delta \mathrm{B} + \frac{\psi(0)}{\mathrm{s}} + \Delta B\right) + y(0)}{s}$$

We must now realize what this ΔX and ΔY refers to. The solution to the above expressions is **different** from the Δy we had expressed earlier. This ΔY is describes the change in y relative to y_r, given an input of Δβ. Let us thus call the Δy we are minimizing, $\Delta \mathrm{y}_{\mathrm{err}}$.

However, this ΔX and ΔY gives us enough information to form our x_r and y_r, with the following relation:

$$x_{car} = x(0) + \int_0^t \Delta x\, dt$$

$$y_{car} = y(0) + \int_0^t \Delta y\, dt$$

Let us convert this to Laplace domain:

$$x_{car} = \frac{x(0) + \Delta X}{s}$$

$$y_{car} = \frac{y(0) + \Delta Y}{s}$$

Recall:

$$\Delta y_{err} = \frac{d}{b}(x_{car} - a) + c - y_{car}$$

Plugging in our values for Δx and Δy:

$$\Delta Y_{err} = \frac{d}{b}\left(\frac{x(0) + \Delta X}{s} - a\right) + c - \frac{y(0) + \Delta Y}{s}$$

Plugging in our values for $\Delta\beta$:

$$\Delta Y_{err} = \frac{x(0)\, d}{b\, s} + \frac{-d^2\left(\frac{v}{s\, l_r}\Delta \mathrm{B} + \frac{\psi(0)}{\mathrm{s}} + \Delta B\right) + d\, y(0)}{b\, s^2} - \frac{da}{b} + c - \frac{y(0)}{s} + \frac{b\left(\frac{v}{s\, l_r}\Delta \mathrm{B} + \frac{\psi(0)}{\mathrm{s}} + \Delta B\right) + y(0)}{s^2}$$

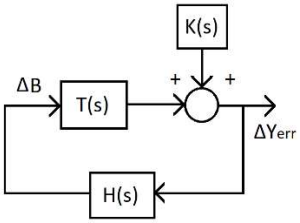
We rewrite our solution:

Derek Xu
UID: 704751588

$$\Delta Y_{err} = \frac{K_0}{s^3} + \frac{K_1}{s^2} + \frac{K_2}{s} + K_3 + \left(\frac{b\left(\frac{v}{s\, l_r} + 1\right)}{s^2} - \frac{d^2\left(\frac{v}{s\, l_r} + 1\right)}{b\, s^2} \right) \Delta B$$

$$\Delta Y_{err} = K(s) + T(s)\Delta B$$

Let us now draw our block diagram:



We achieve the following transfer function:

$$\Delta B = H(s)\, \Delta \mathrm{Y_{err}}$$

$$\Delta Y_{err} = K(s) + T(s)\, \Delta B$$

$$\Delta Y_{err} = K(s) + T(s)H(s)\, \Delta \mathrm{Y_{err}}$$

$$\Delta \mathrm{Y_{err}} = \frac{K(s)}{1 - T(s)H(s)}$$

We would like to achieve $\lim_{t\to\infty} \Delta y_{err}(t) = 0$.

Let us use a PD controller, and see if the above condition can be satisfied:

$$H(s) = K_p + K_d s$$

Let us apply final value theorem:

$$\lim_{t\to\infty} \Delta y_{err}(t) = \lim_{s\to 0} s\, \Delta Y_{err} = \lim_{s\to 0} \frac{sK(s)}{1 - T(s)H(s)}$$

$$\lim_{s\to 0} \frac{sK(s)}{1 - T(s)H(s)} = \lim_{s\to 0} \frac{s\left(\frac{K_0}{s^3} + \frac{K_1}{s^2} + \frac{K_2}{s} + K_3\right)}{1 - \left(K_p + K_d s\right)\left(\frac{b\left(\frac{v}{s\, l_r} + 1\right)}{s^2} - \frac{d^2\left(\frac{v}{s\, l_r} + 1\right)}{b\, s^2} \right)}$$

$$= \lim_{s\to 0} \frac{\left(\frac{a_0 s^2 + a_1 s + a_2}{s^2}\right)}{1 - \frac{\left(K_p + K_d s\right)(b_0 s + b_1)}{s^3}}$$

$$= \lim_{s\to 0} \frac{s(a_0 s^2 + a_1 s + a_2)}{s^3 - \left(K_p + K_d s\right)(b_0 s + b_1)}$$

Derek Xu
UID: 704751588

$$= 0$$

$$\therefore \lim_{t\to\infty} y_{er}\ (t) | (H(s) = K_p + K_d s) = 0$$

A simple PD controller on Δy_{err} would work!

Notice, we can even use just a straight P controller here. As seen by the final value theorem expression K_d could be set to 0 and the system will still converge to 0 error (given we set $K_p > 0$… obviously).

Notice, there are additional problems that arise from the nonlinear properties of the original system that we linearized. These properties will be discussed in later sections. Because we do not have standardized ways of dealing with the nonlinear systems, it is sufficient to show here that our PD Δy_{err} controller will satisfy the linearized system.

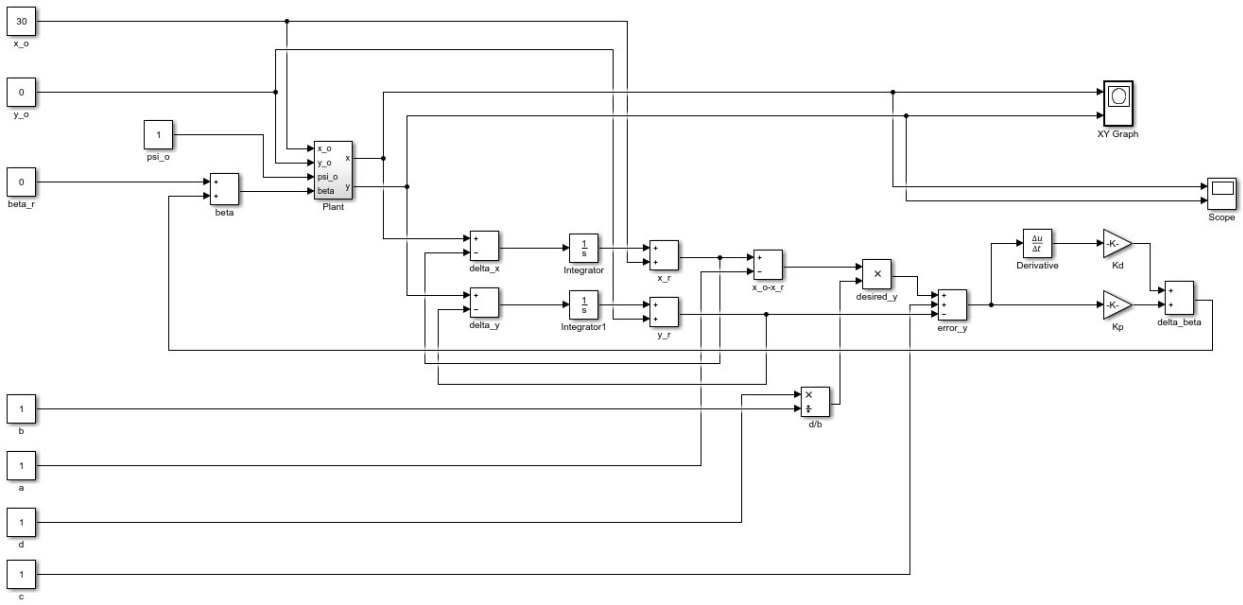
Thought experiment: we eliminate Δy_{err} with PD control, but what about Δx_{err}?

We had set Δy_{err} to be the error in y at $x = x_{car}$. In other words, we can assume that the car is at the proper location relative to the path. Thus, if $\Delta y_{err} = 0$, we are on the path at **both** y_{car} and x_{car}!

5.

The following shows a closed loop system in Simulink (we chose $K_p = 0.001, K_d = 0.01$):

Whole System (with controller):

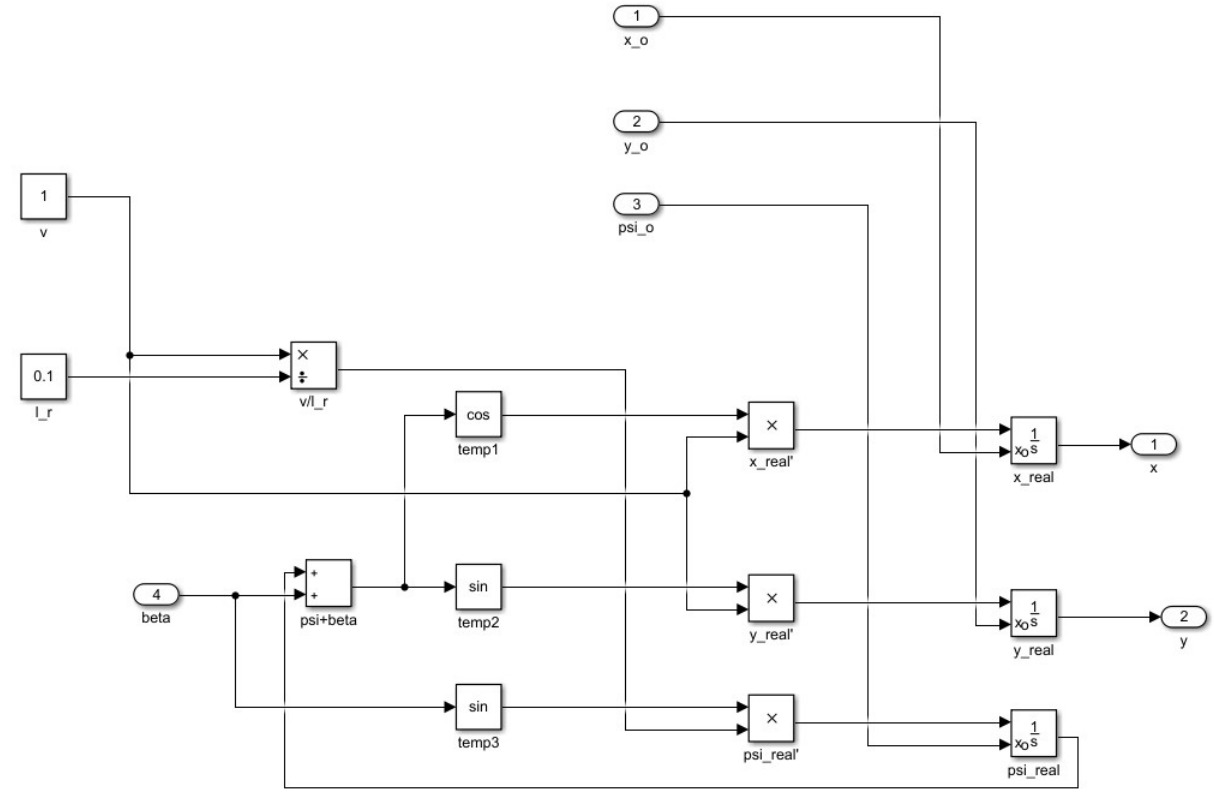


NOTE: x_r and y_r in this block diagram is mislabeled. x_r really refers to x_{car} and y_r really refers to y_{car}. This note applies to *all* block diagrams in the report.

After talking to professor and TA, our controller seems to be unconventional from the standard controller answer for this problem. If we set our controller to track lines where d can be zero, we can avoid the extra step of finding complex reference points to compute Δy_{err}. Our controller is more complex, but more robust.

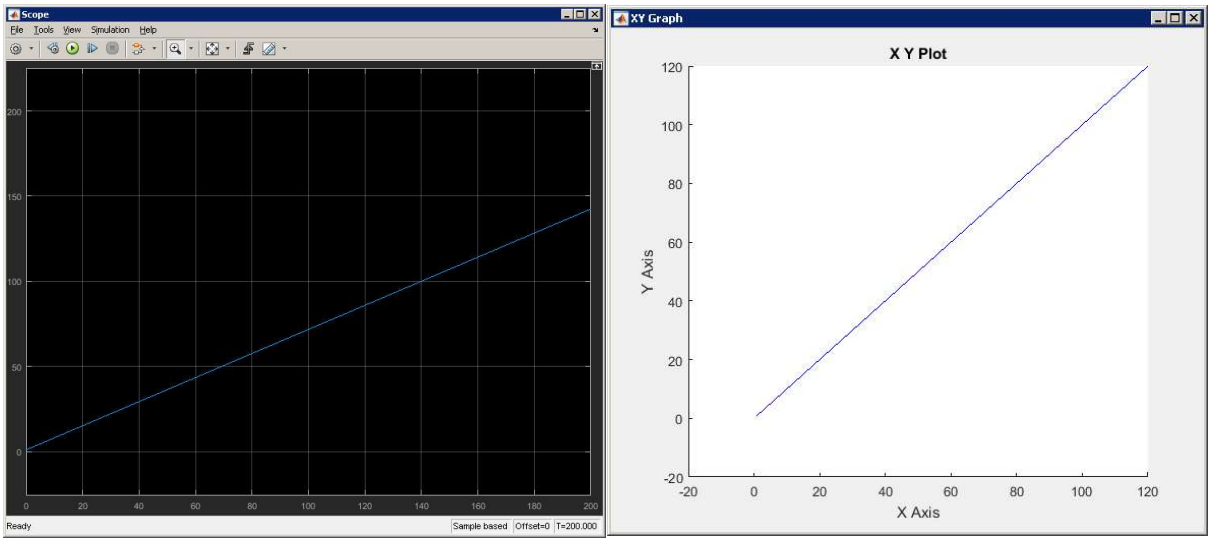
Derek Xu
UID: 704751588

Plant:



Given the values: $\boldsymbol{a = 1, b = 1, c = 1, d = 1}$ and initial conditions: $x(0) = 1, y(0) = 1, \psi(0) = 0.785$.

We get the following plot for $x = f_1(t)$ and $y = f_2(t)$:



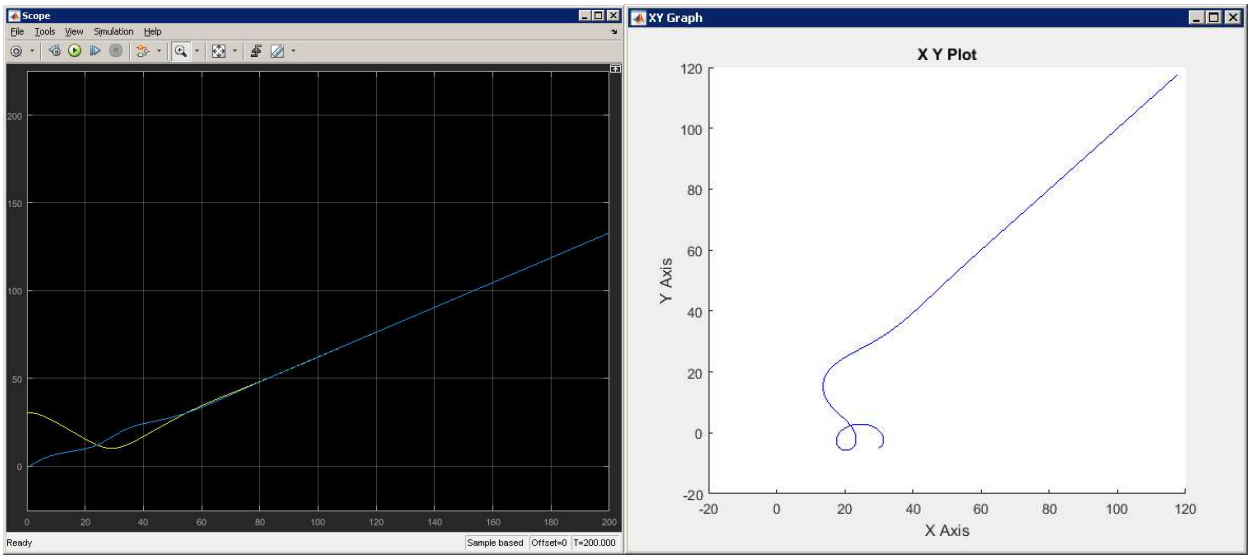
This is as expected: the x and the y values increase at the same rate starting from the same point (1,1). Note, the left plot may be hard to see as Simulink plotted the blue line on top of the yellow line, but basically x and y location increase at the same rate (which tracks the reference trajectory path). We can also see the physical trajectory of the car follows the curve: $x = 1 + t, y = 1 + t$, in the XY plot. Our

Derek Xu
UID: 704751588

system should act as the linear model here as it does not deviate from the steady state, thus our controller is functioning in its "ideal" state.

Given the values: $\boldsymbol{a = 1, b = 1, c = 1, d = 1}$ and initial conditions: $x(0) = 30, y(0) = -1, \psi(0) = 1.2$

We get the following plot for $x = f_1(t)$ and $y = f_2(t)$:

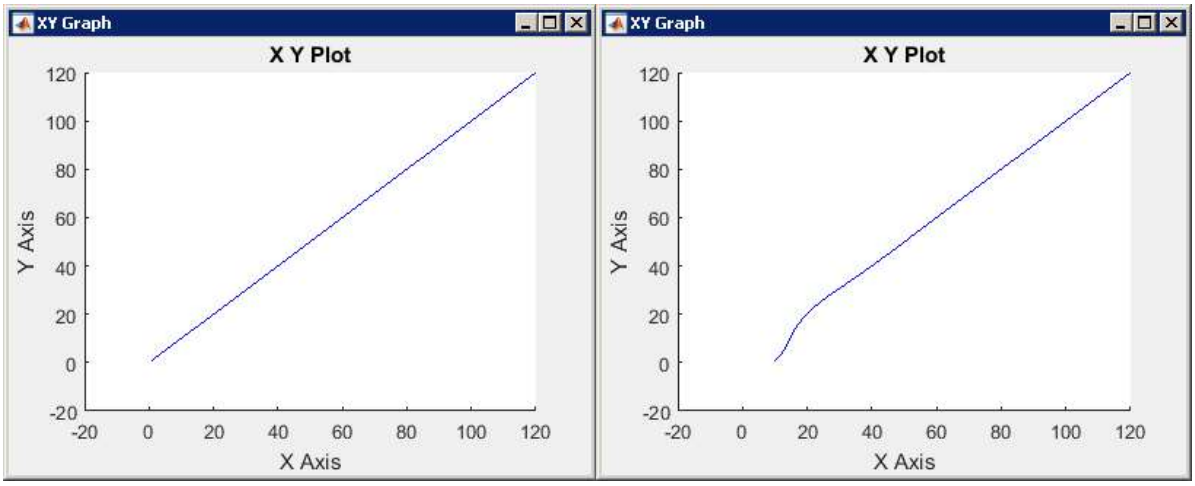


This is as expected: given that the initial conditions are not on the trajectory, our controller will change beta through the controller such that we asymptote to the correct trajectory. Note, our controller does a loop here. This is due to the nonlinearities of the system. We assume that larger Δy_{err} would correspond with larger $\Delta\beta$. This is true in our linearized model, but in the nonlinear system, x and y values depend on a sinusoid of β, thus if $\Delta\beta > \pi/2$ the car will turn over 90°, which will possibly cause the car to spin in loops. But still recognize the controller is working: we are adjusting β to minimize Δy_{err}, as can be seen in the car's trajectory after the loop. Please see part 6 for car trajectories with no loops.

6.

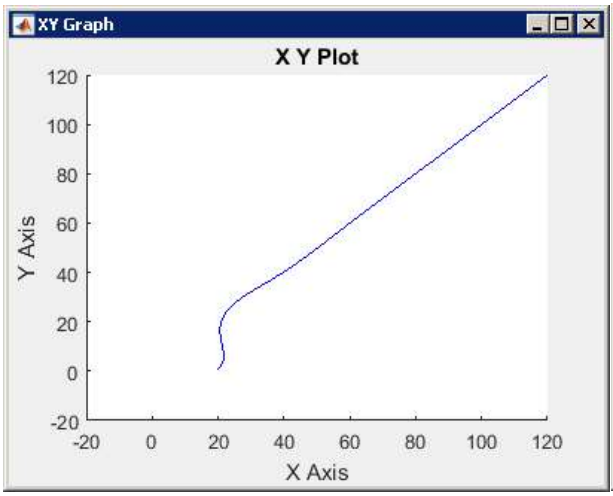
Let us vary values for $\boldsymbol{x(0)}$ (I plot all the expected graph at the end):

$x(0) = 1$ $x(0) = 10$

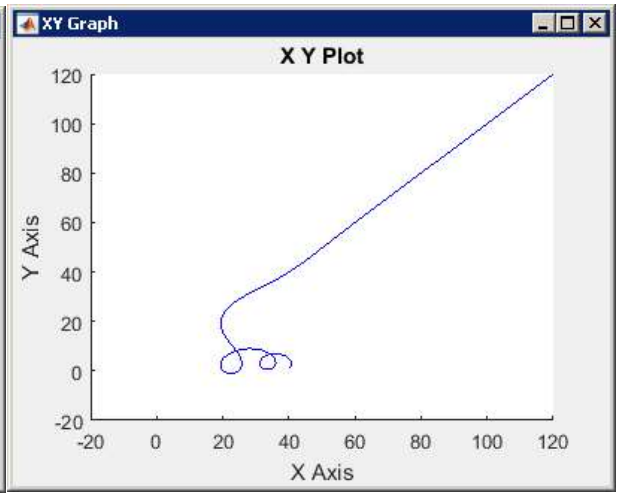


Derek Xu
UID: 704751588

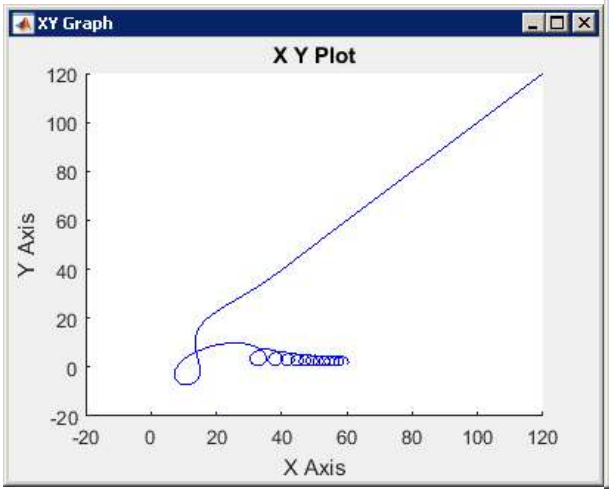
$x(0) = 20$



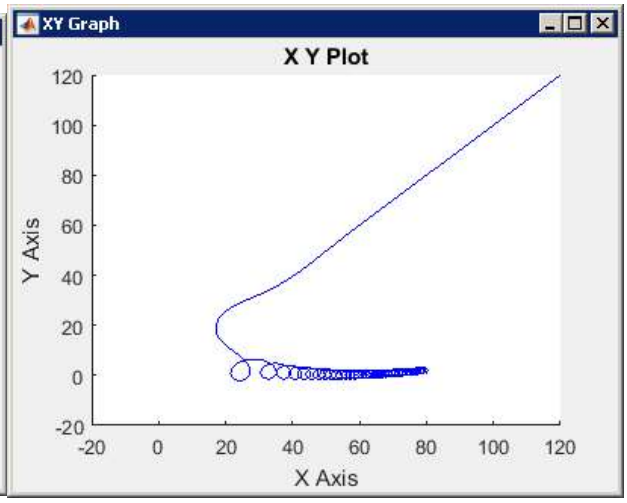
$x(0) = 40$



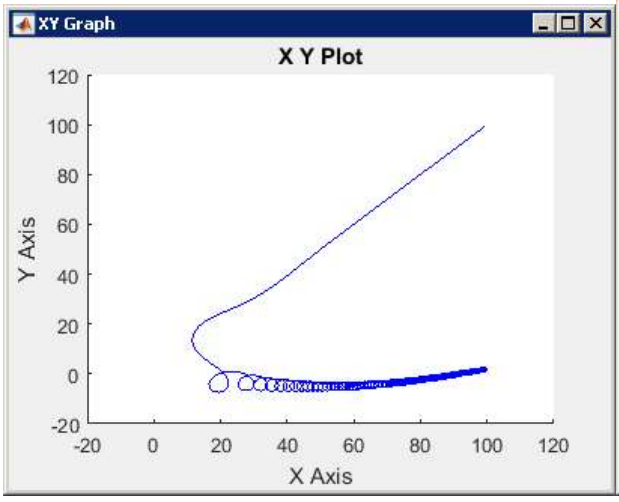
$x(0) = 60$



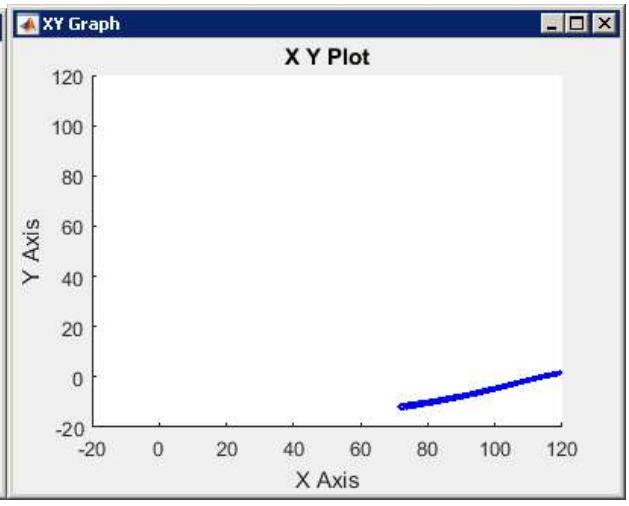
$x(0) = 80$



$x(0) = 100$



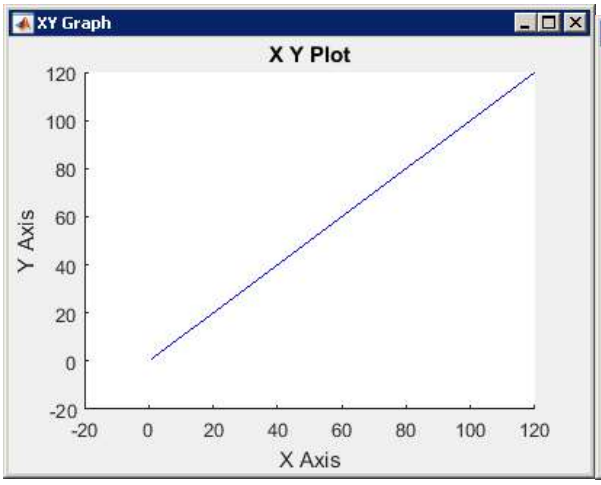
$x(0) = 120$



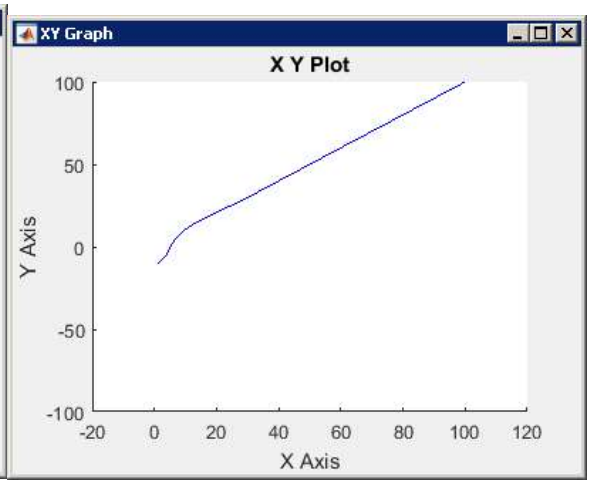
Derek Xu
UID: 704751588

Let us vary values for $y(0)$:

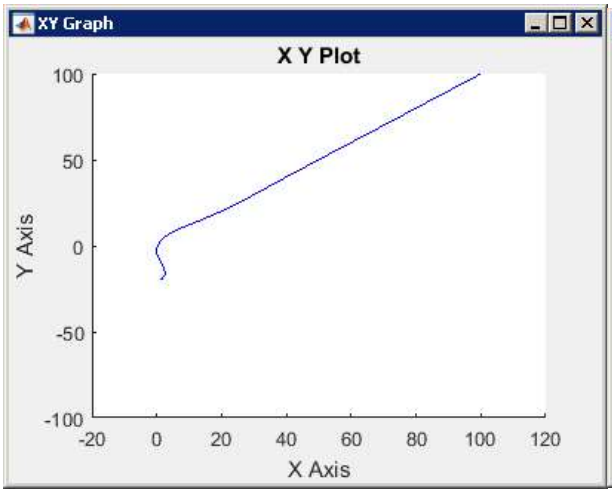
$y(0) = 1$



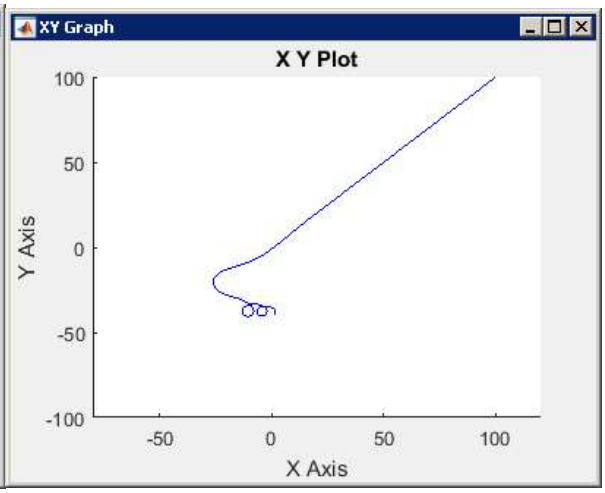
$y(0) = -10$



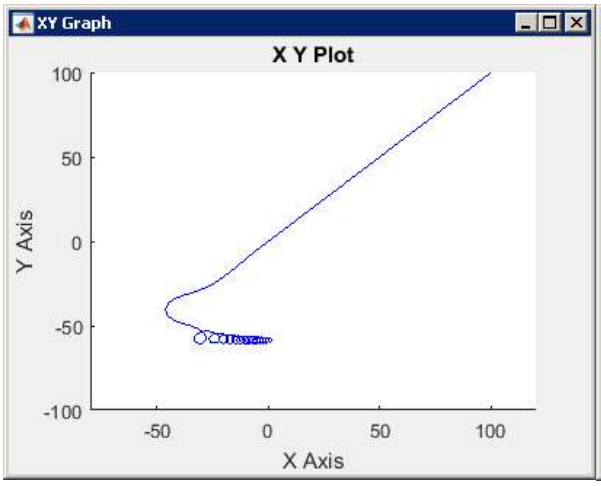
$y(0) = -20$



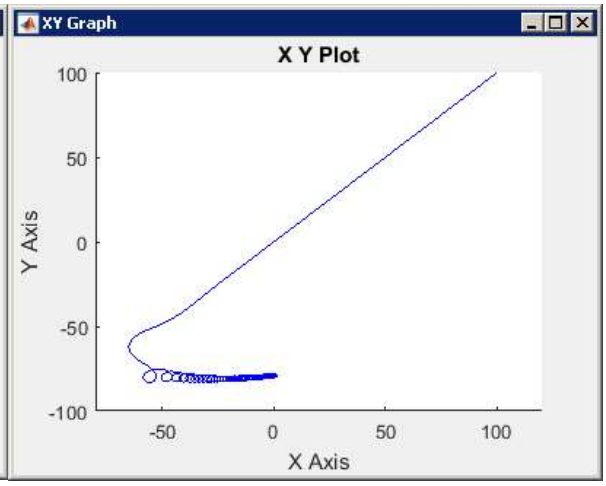
$y(0) = -40$



$y(0) = -60$

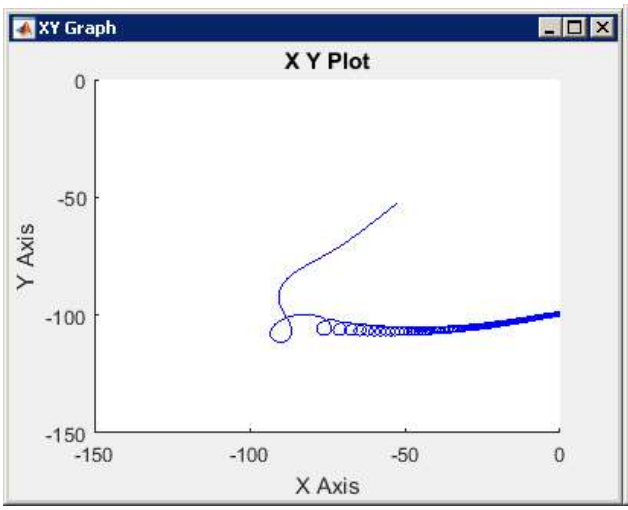


$y(0) = -80$

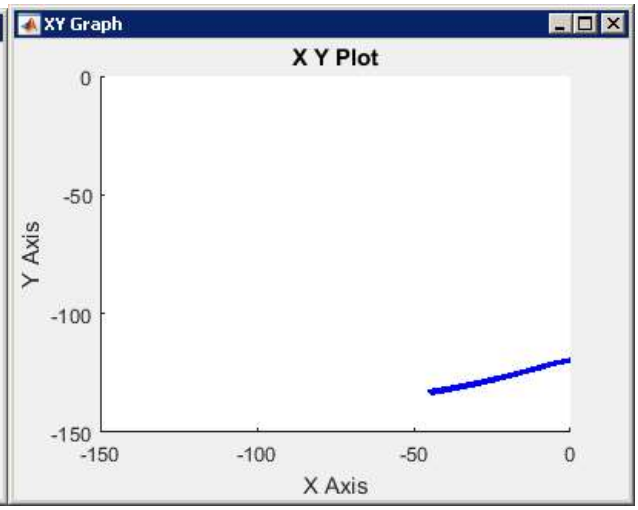


Derek Xu
UID: 704751588

$y(0) = -100$

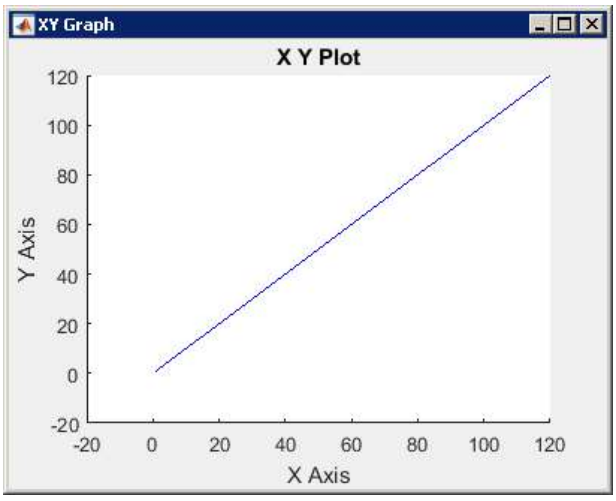


$y(0) = -120$

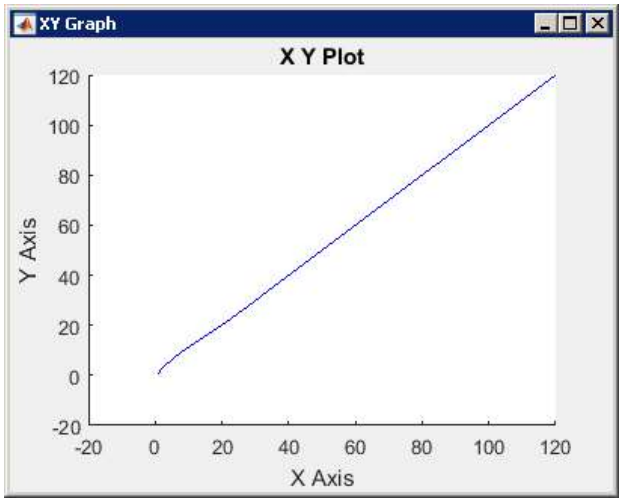


Let us vary values for $\psi(0)$:

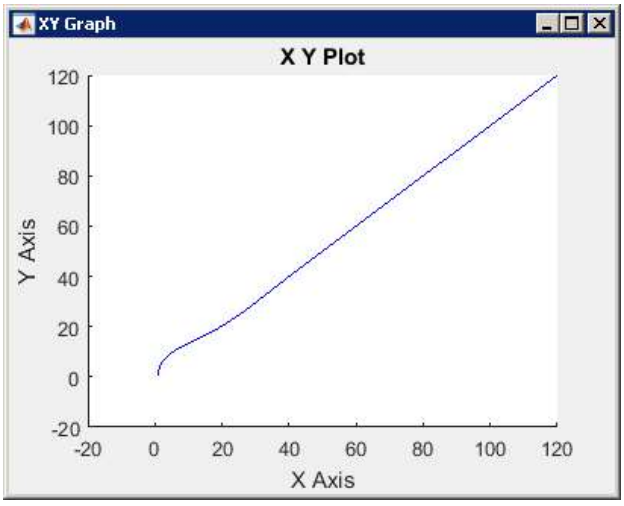
$\psi(0) = 0.785$



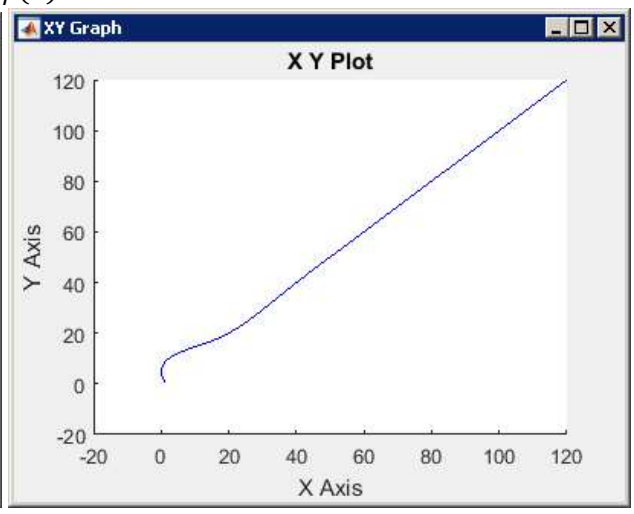
$\psi(0) = 1$



$\psi(0) = 1.5$

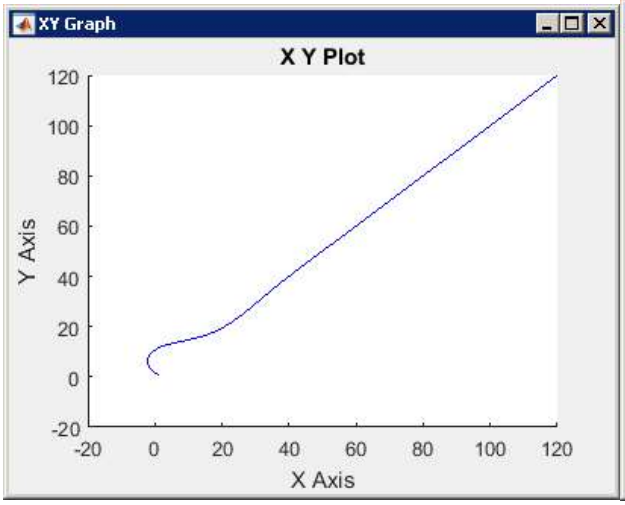


$\psi(0) = 2$

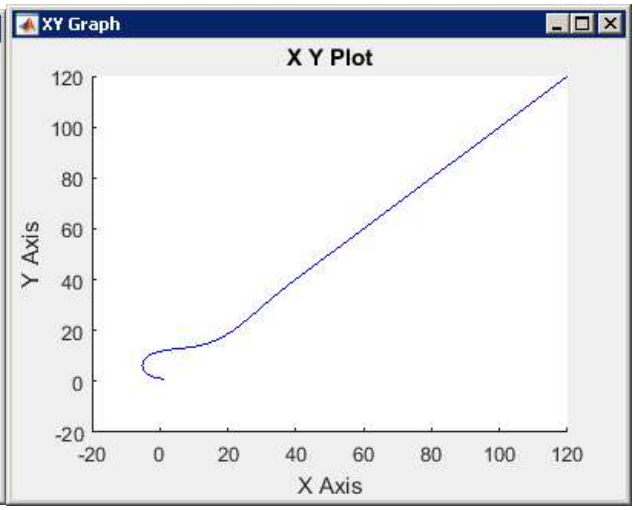


Derek Xu
UID: 704751588

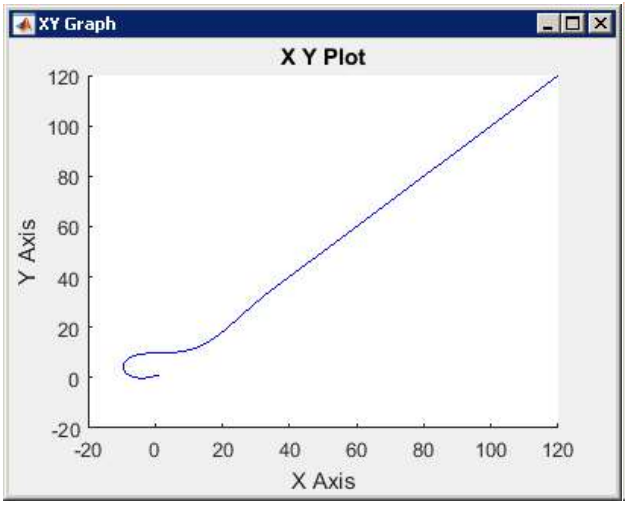
$\psi(0) = 2.5$



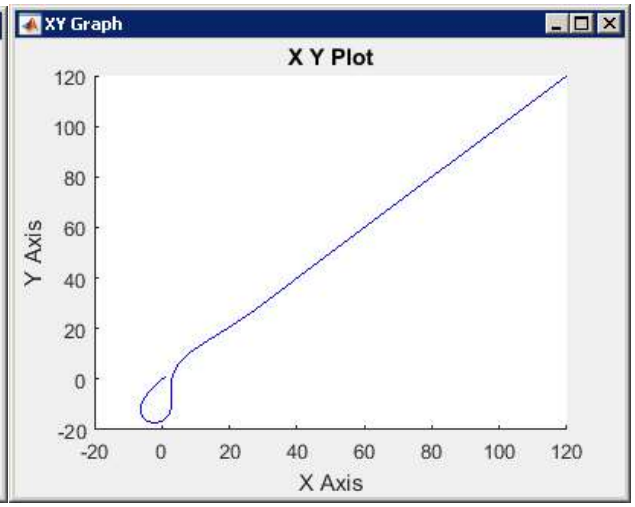
$\psi(0) = 3$



$\psi(0) = 3.5$



$\psi(0) = 4$



Comments

For x and y:

We see here that, as discussed in part 5, the car will start **spinning** in circles overcompensating for Δy_{err} in its $\Delta\beta$ term. Again, this is due to the **nonlinearities in the system** specifically applied to **sinusoids**. There is a curious case though: despite turning in loops, the car's trajectory will always land on the trajectory path. This is a neat result of more complex dynamics that cannot be explained by just the linear model. A possible hypothesis (though it needs more testing) is that the car overcompensates for Δy_{err} but that overcompensation loops around once the $\Delta\beta$ term reaches a factor of 2π, and the car continues to compensate normally; in which case, for the long run, we are still decreasing Δy_{err}. In this case, the reduction of Δy_{err} will cause the reduction of overcompensation of $\Delta\beta$, allowing the car to unloop itself once it reaches that state.

Another curious case: as the car is put farther off from the trajectory the longer time it takes to reach the trajectory. In fact, the $x(0) = 120$ and $y(0) = -120$ cases may seem that the car is going nowhere but

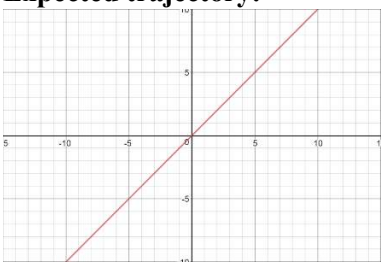
Derek Xu
UID: 704751588

more thoroughly testing will show that *no matter how far* (in the scope of our current software), *given enough time,* **the nonlinear model will self-correct**, and the car will follow the reference trajectory path. This is an unexpected, though advantageous, consequence of our nonlinear system. Note, I have only tested this for $x(0) = 2000$. It is possible that larger values will cause unbounded trajectories (though this is highly unlikely); the simulation for $x(0) = 2000$ initial condition took half an hour to complete, so such testing is outside the scope of our current software.

For ψ:

We stay within the bounds of our linear model here, thus as seen in testing, we always converge to the reference trajectory with no awkward or unexpected behaviors. We physically tested the maximum offset for $\psi(0)$ by making the car face opposite to the trajectory in the last case, and the car still corrected itself in a suitable manner.

Expected trajectory:



7.

We fix our initial conditions at the following places:

$$x(0) = 2, y(0) = 1, \psi\,(0) = 0.785$$

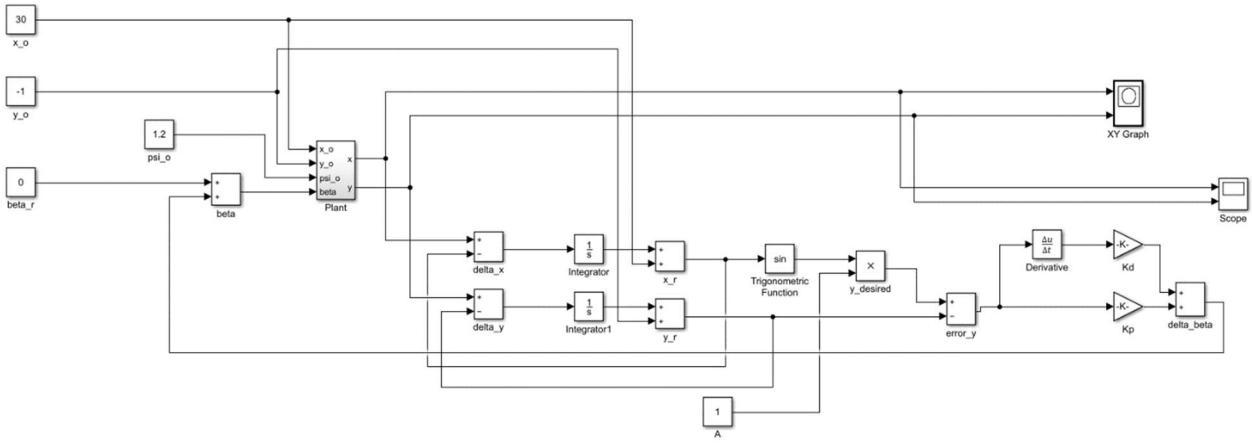
Tracking a **sinusoid**:

$$x = t$$

$$y = \mathrm{A}\,\sin(t)$$

Note: we alter our computation of, Δy, such that it is the difference between the path of the trajectory at location x_{car} and y_{car}: $\Delta y = \mathrm{y_{path}}(x_{car}) - y_{car}$

The following shows our new Simulink model:

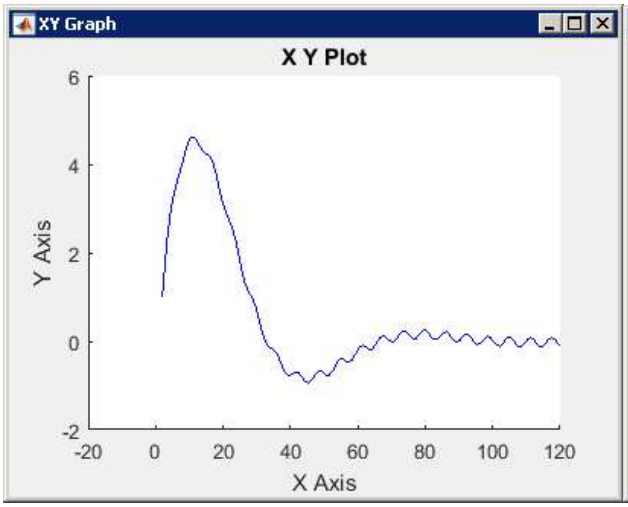


Derek Xu
UID: 704751588

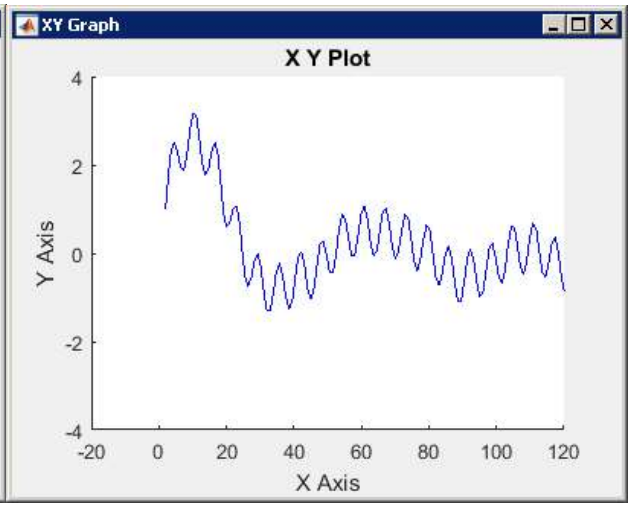
NOTE: It may look as if the **controller has changed** in this diagram, but this is **not true**! What we are changing is **not the controller** but rather the extra step of computation required to get **the reference point**. As confirmed by professor in class, doing this is OK, as we are not changing the controller, but rather the computation of the reference point!

Let us vary values for A (I plot all the expected graphs at the end):

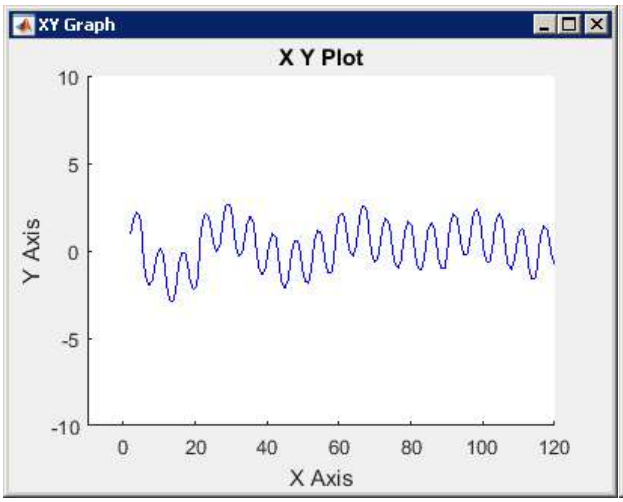
$A = 1$



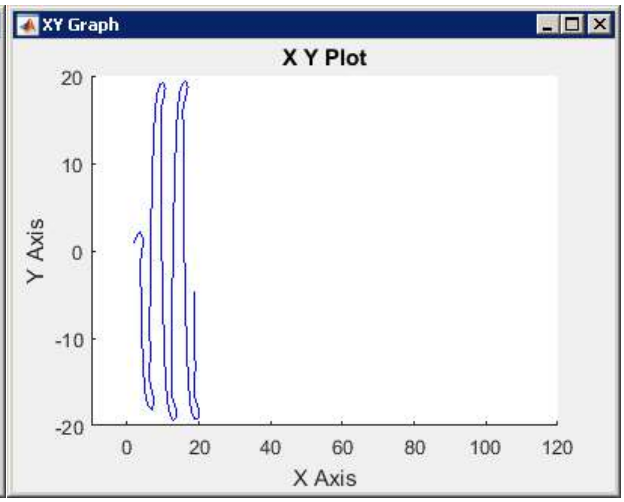
$A = 5$



$A = 10$



$A = 15$



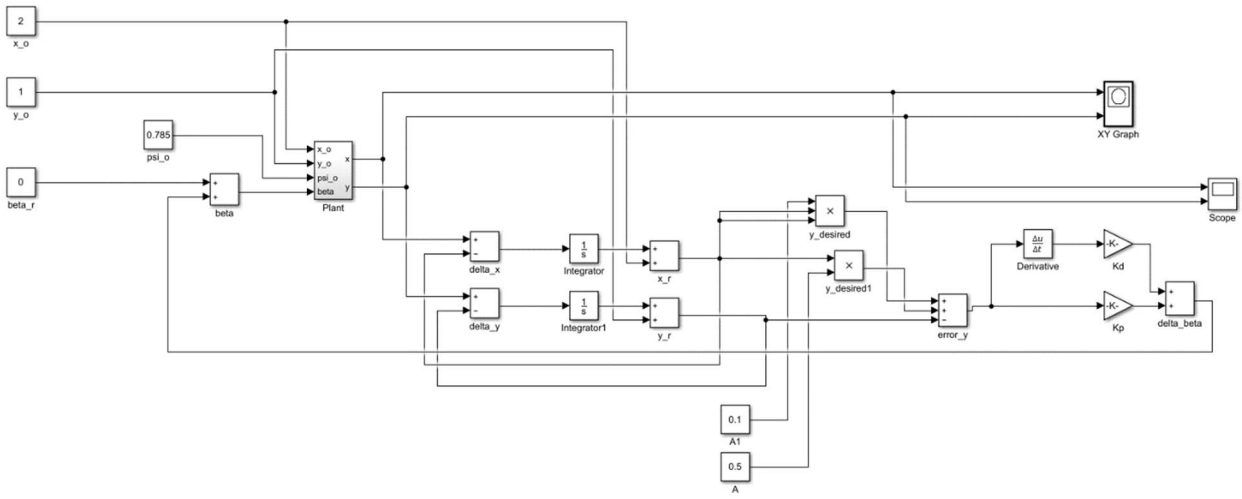
Tracking a **polynomial**:

$$x = t$$

$$y = At^2 + 0.5t$$

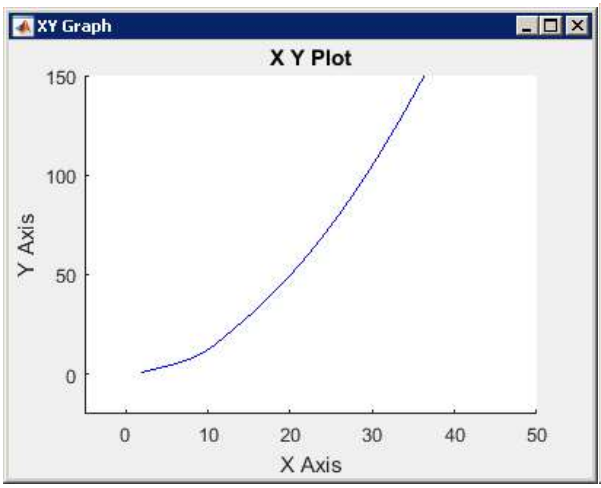
The following shows our new Simulink model:

Derek Xu
UID: 704751588

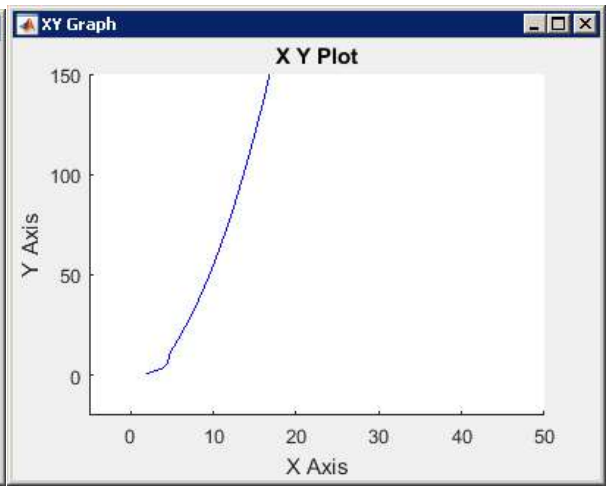


Let us vary values for A:

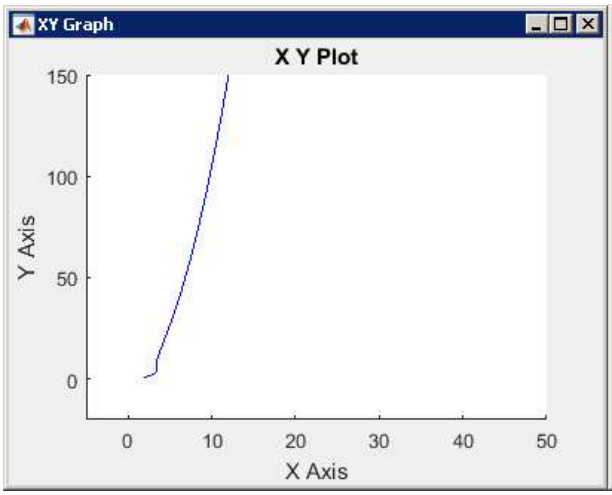
$A = 0.1$



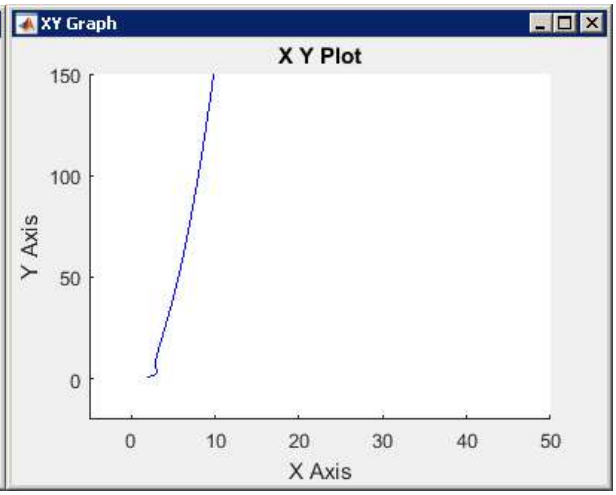
$A = 0.5$



$A = 1$



$A = 1.5$



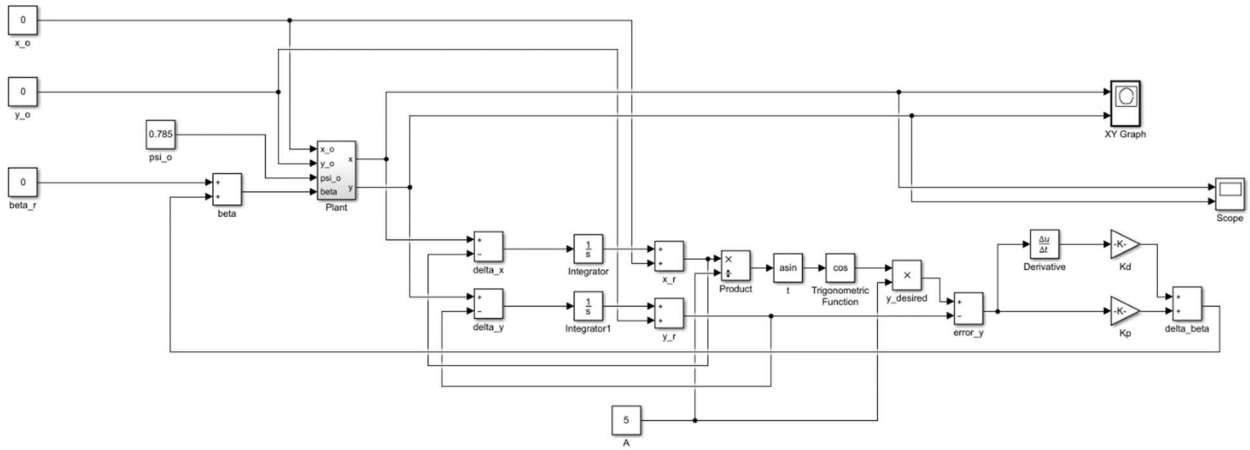
Derek Xu
UID: 704751588

Tracking a **circle**:

$$x = A\sin(t)$$

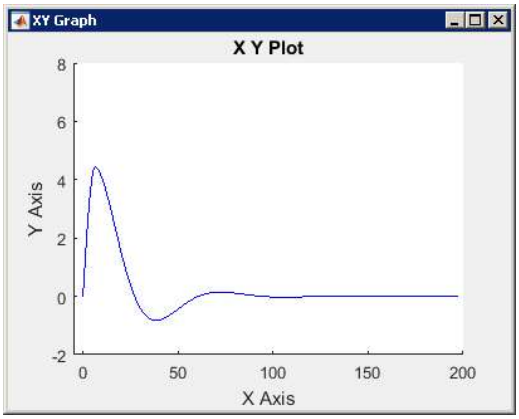
$$y = A\cos(t)$$

The following shows our new Simulink model:



Let us vary values for A:

$A = 5$



Comments:

For sinusoids:

Our controller does a **bad job** at tracking all types of sinusoids. This is expected as sinusoid trajectories undermine many of the assumptions we had made for our linearized system. In addition to all the other issues, talked about in part 6 and 7, our K_p value may also be of interest. We track sinusoids with higher amplitudes (relatively) better than those with low amplitudes. This is possibly because we cannot correct our y location fast enough to track the sinusoid. This will be later expanded upon in part 8.

For polynomials:

Our controller **does a reasonably well** job at tracking all types of polynomials. This is as expected as polynomials resemble a linear line (as we tested in part 3) more than that of a sinusoid. Our PID constants

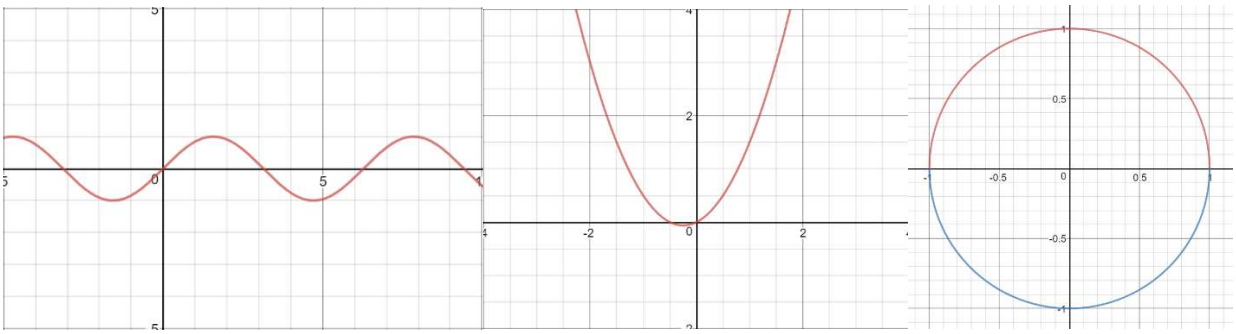
Derek Xu
UID: 704751588

work well here as well. As we are not expected to prove the existence of a solution to the curve with these references, this is a sufficient explanation.

For circles:

Our controller **does not track** circles. Circles are nonlinear with respect to both x and y. We do not even have a controller to track x, thus it is extremely hard to track a circular motion with just our current controller. The reason our controller veers of the circle and to the right is because of these nonlinearities. As y is a sinusoidal function, the controller has no way of telling which x it should be at. Contrary to our solution to problem 4, our y value can correspond to multiple x's at factors of 2π and vice-versa, thus our controller fails to track this trajectory.

Expected trajectories:



For the sake of clarity I will just plot the $A = 1$ case for each of the cases, altering A just changes the amplitude of the plots:

8.

We may explicitly find the range of possible values for the poles and their effects by evaluating the root locus of our explicitly derived expression of the linearized model:

$$\Delta \mathrm{Y}_{\mathrm{err}} = \frac{K(s)}{1 - T(s)H(s)}$$

However, as mentioned in professor's office hours this is *not necessary*. Due to many of the errors in our system, arising from the **nonlinearities** of the curves we track, the root locus of the linearized model will fail to consider many of the sources of error of our controller. Thus, we shall instead explore how altering values of K_p and K_d (moving the poles) will affect our trajectory for the various curves we try to track, through intuitive explanation and fundamental understanding of what these constants do.

Thus, in general the following is true: altering K_p will make the controller follow the trajectory more aggressively but also will cause it to (possibly) over-compensate and approach steady state slower, it will also (possibly) incur more steady state error. Altering K_d will make the controller follow the trajectory faster. Note, this is in addition to our previous discussions of nonlinearities. If we make either constant too big, the car will start turning in loops.

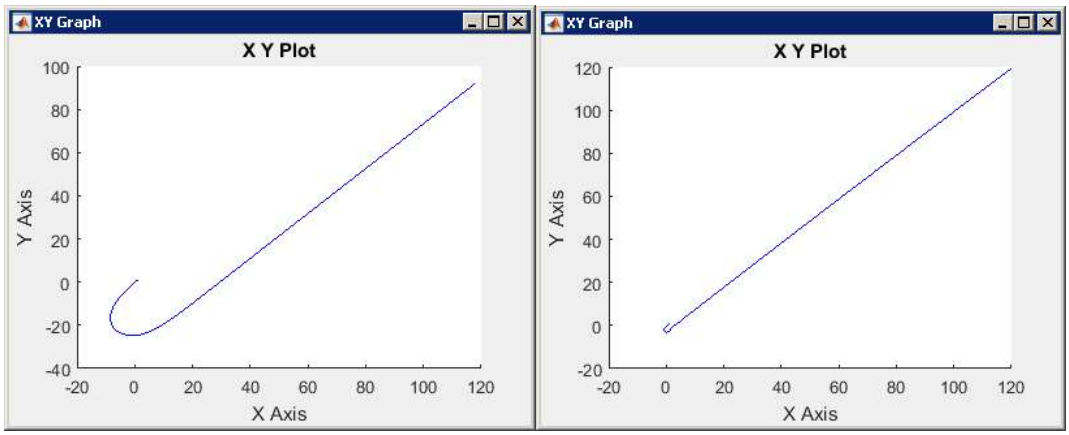
Regarding linear trajectory:

We fix all the initial conditions and constants to be the same as problem 4's first setup except: $\psi(0) = 4$

We alter one of the constants as shown below:

Derek Xu
UID: 704751588

$K_p = 0.00001$ $K_d = 0.1$

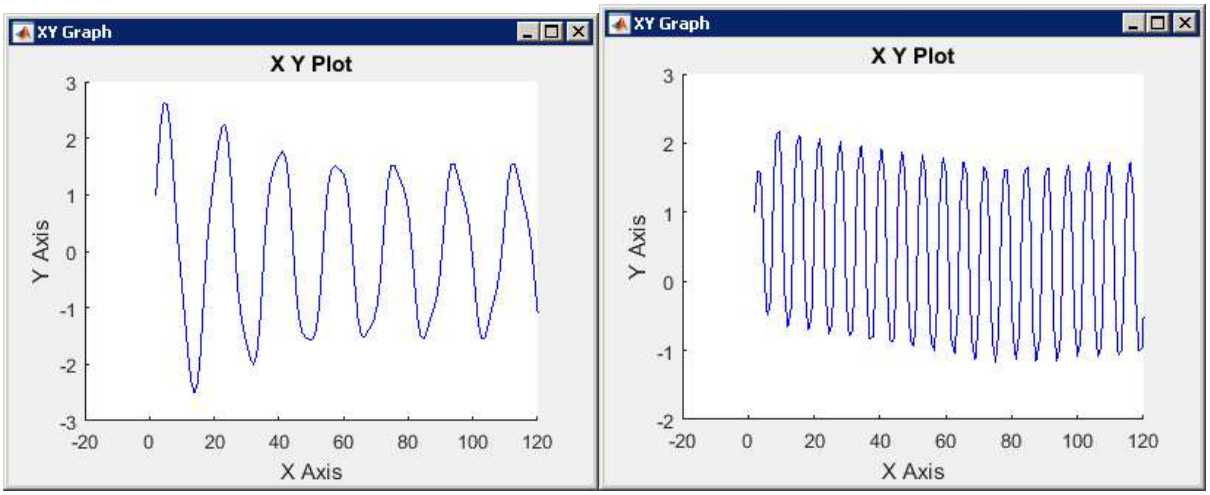


Regarding sinusoidal trajectory:

We fix all the initial conditions to be the same as problem 5's first sinusoid setup except: $A = 1$

We alter one of the constants as shown below:

$K_p = 0.01$ $K_d = 0.1$



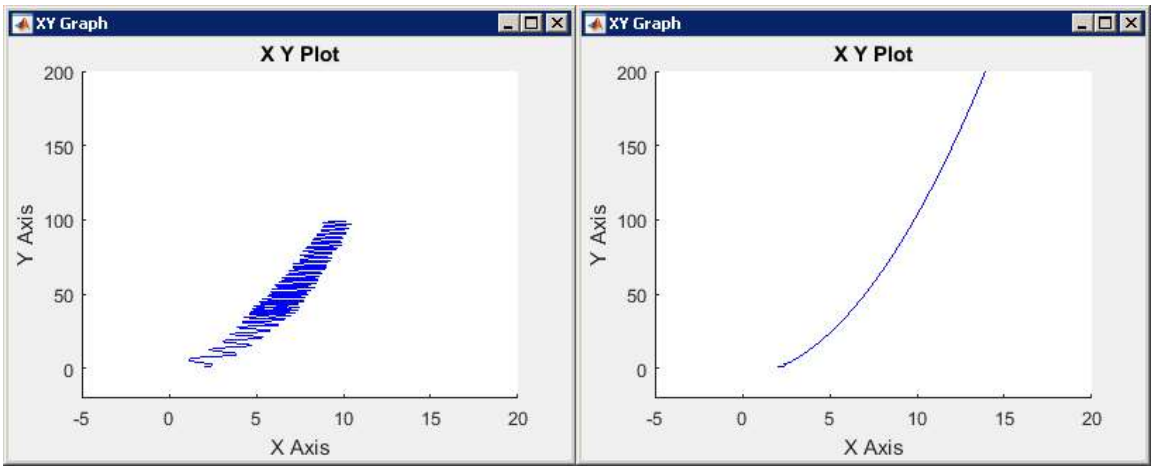
Regarding polynomial trajectory:

We fix all the initial conditions to be the same as problem 5's first sinusoid setup except: $A = 1$

We alter one of the constants as shown below:

$K_p = 0.01$ $K_d = 0.1$

Derek Xu
UID: 704751588

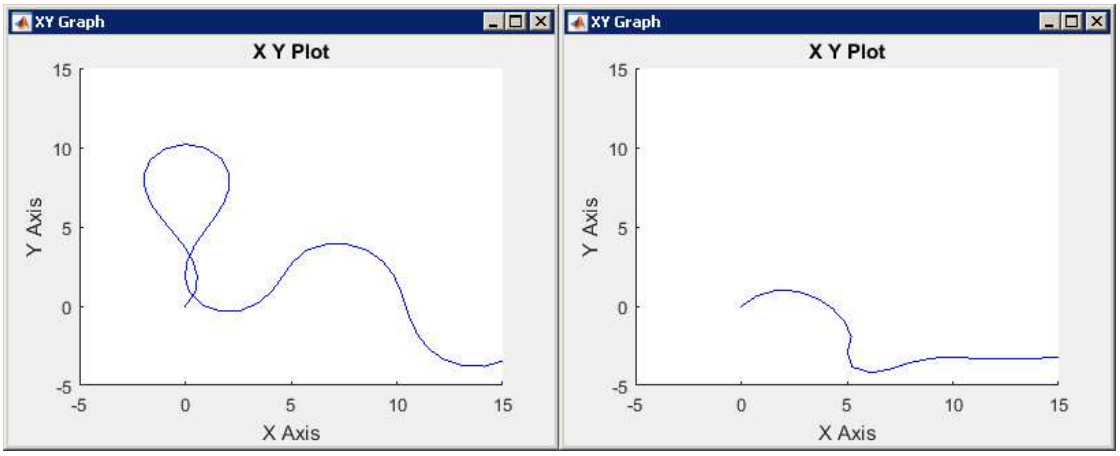


Regarding circular trajectory:

We fix all the initial conditions to be the same as problem 5's first sinusoid setup except: $A = 5$

We alter one of the constants as shown below:

$K_p = 0.01$ $K_d = 0.1$



Comments (for more detail on what K_p and K_d do refer above):

Regarding Linear:

We decreased K_p, and, as expected, our controller tracked the curve a lot less aggressively. We will converge to it slower, but we will be more smooth in our convergence. In other words had our convergence contain loops from overcompensation before, we will not have them now.

We increased K_d, and, as expected, we converged to the correct trajectory path a lot faster.

Altering the poles did allow us to robustly track lines.

Regarding Sinusoidal:

We increased K_p, and our controller now tracks at the wrong frequency.

We increased K_d, and, as expected, we converged to the trajectory path as in part 7 a lot faster.

Derek Xu
UID: 704751588

Altering the poles did not allow us to robustly track sinusoids.

Regarding Polynomial:

We increased K_p, and, as expected, our controller tracked the curve a lot less aggressively and we incurred more steady state error.

We increased K_d, and, as expected, we converged to the correct trajectory path a lot faster.

Altering the poles did allow us to robustly track polynomials.

Regarding Circular:

We increased K_p, and, as expected, our controller tracked the curve a lot less aggressively, but not aggressively enough to track the circle.

We increased K_d, and, as expected, we converged to a trajectory path a lot faster.

Altering the poles did not allow us to robustly track circles.

Thought Experiment: What about K_I (Integrator term in PID)? We did not use K_I, but what it does is reduce steady state error… However as seen both in our linear model and our simulations, we don't run into cases where adding K_I would be an adequate justification… It may be worth experimenting with for the sinusoid and circle cases, but such would require more advanced control theory, outside the scope of this class, to capture the dynamics of the system, and intuition says using PID on Δx_{err} first may be more advantageous.