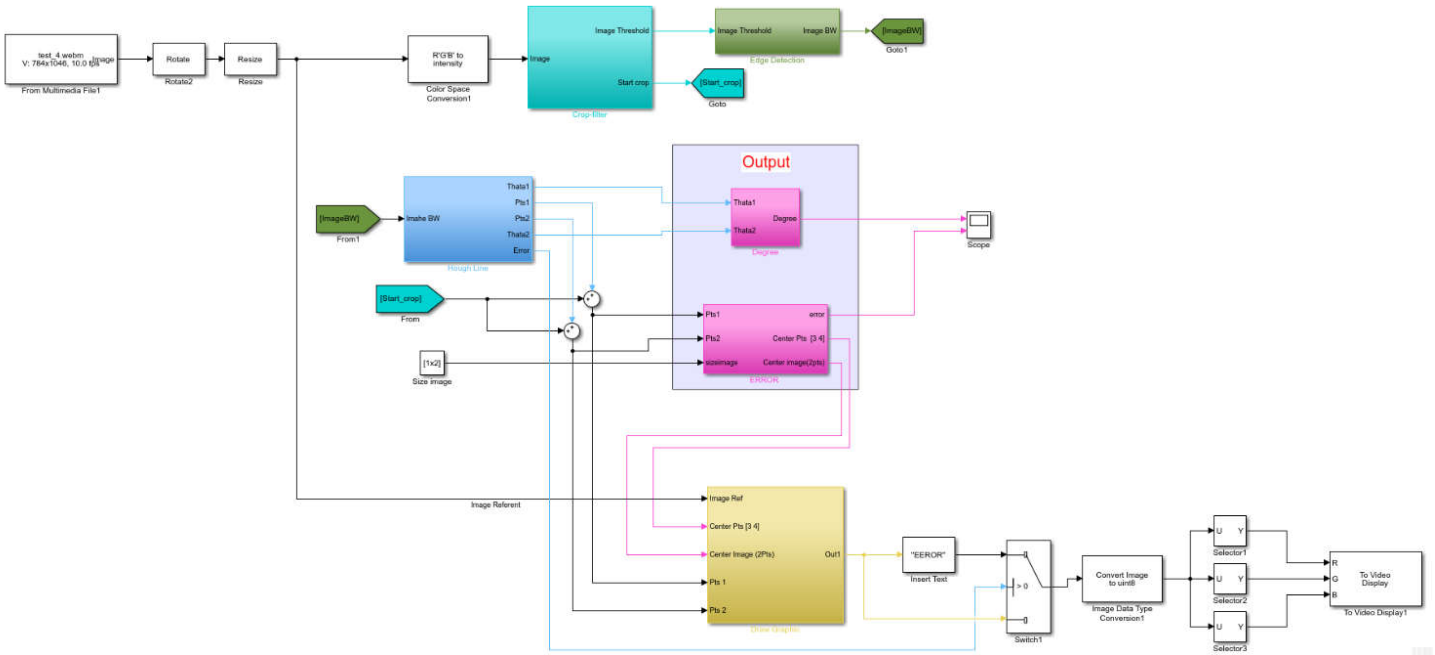
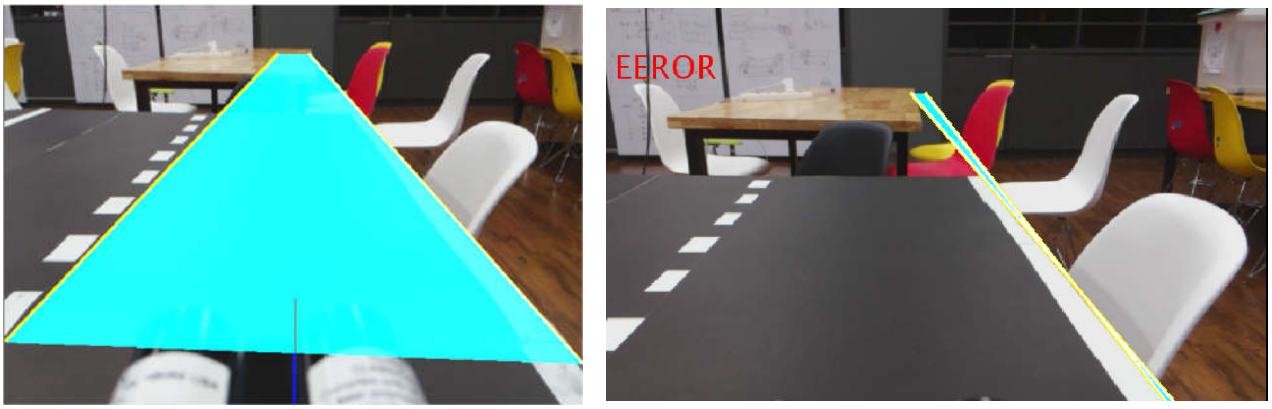
## การดำเนินงานในส่วนของ image processing

Track Lane

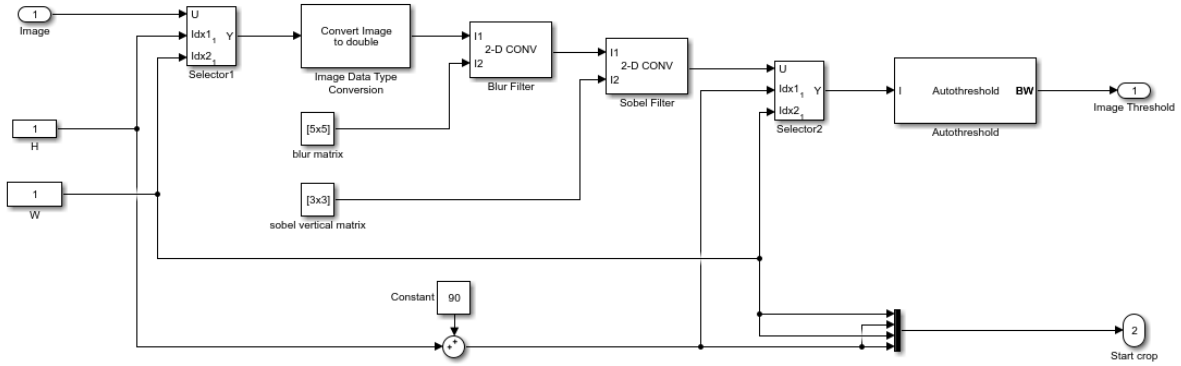



เป็นโมเดลที่ทำให้รถสามารถวิ่งไปตามเลนของถนนได้เองโดยไม่ต้องมีคนบังคับ โดยมี input เป็นสัญญาณภาพจาก Camera Module มี Output เป็นองศาการหมุนของล้อ , ระยะ error ระหว่างกลางเลนกับตรงกลางรถ และ สัญญาณ ERROR เมื่อไม่สามารถตรวจจับเลนได้



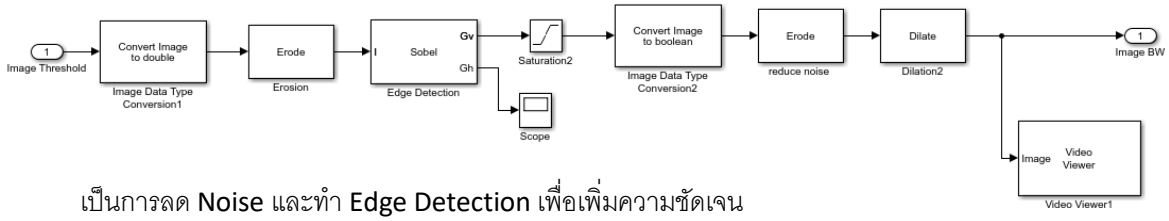

รูปภาพแสดงลักษณะของเลนที่ตรวจจับได้

**Crop-filter BLOCK**



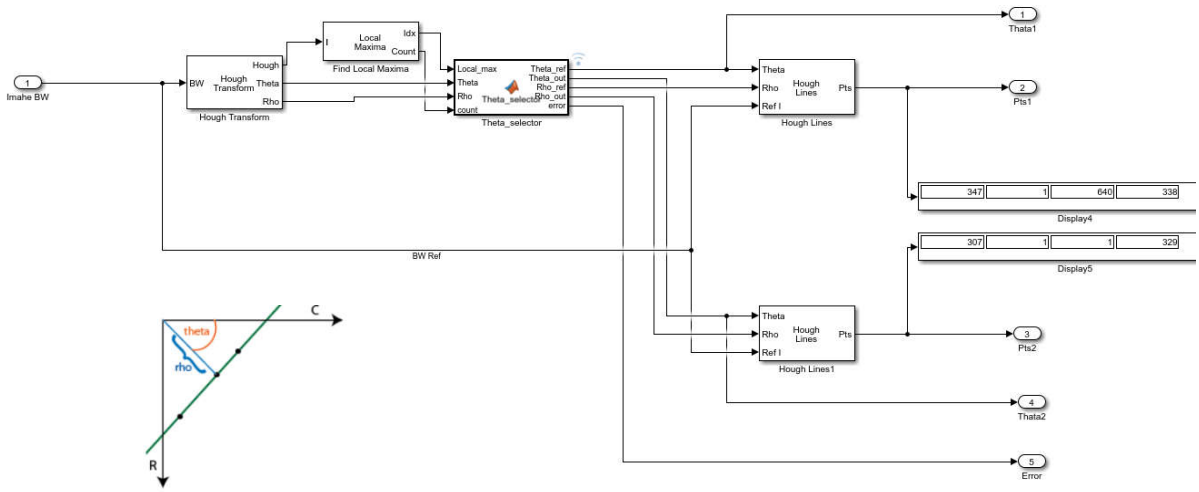

เป็นการ crop รูปเพื่อตัดส่วนที่ไม่จำเป็นในการ Tracking lane และนำส่วนนั้นไป Blur เพื่อลด Noise ออกจากนั้นนำไปหา Edge Detection โดยใช้ Sobel vertical เนื่องจากเลนที่กล้องจับได้จะมีลักษณะอยู่ในแนว vertical มี Output เป็นสัญญาณรูปขาวดำ และจุดเริ่มต้นของการ crop รูปภาพ เพื่อใช้อ้างอิงเปรียบเทียบจากรูปที่ input เข้ามา

**Edge Detection BLOCK**




เป็นการลด Noise และทำ Edge Detection เพื่อเพิ่มความชัดเจน

**Hough Line BLOCK**



เป็นการทำ Hough Transform เพื่อหาจุดเส้นต่างๆในรูปภาพ และเลือกเส้นที่ชัดเจนที่สุด โดยมี Theta_selector เป็นการเลือกองศาของเส้นที่จะสามารถเป็นเลนได้ และมี Output เป็นเส้นเลน 2 เส้น,องศาของเส้น และสถานะ Error

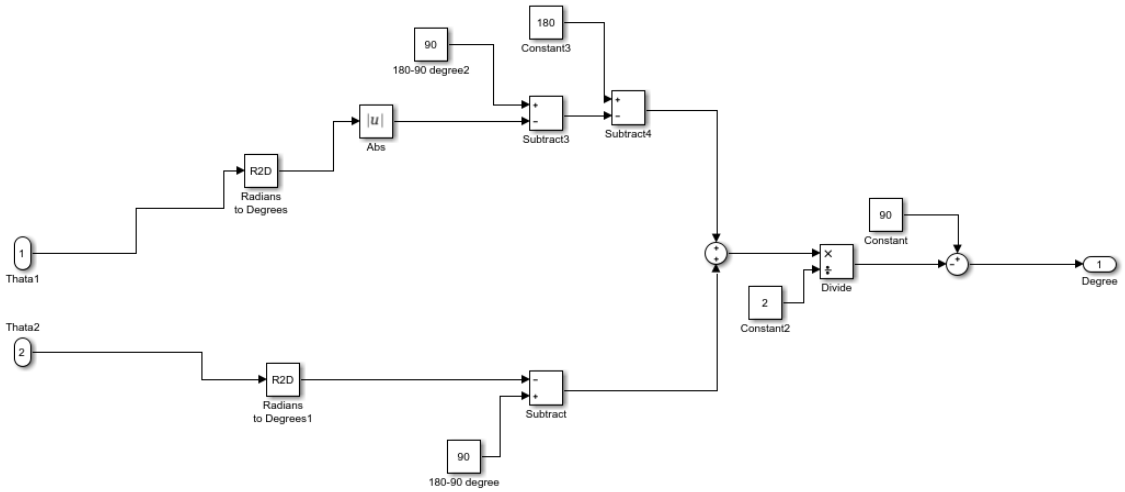
## Theta_Selection Function

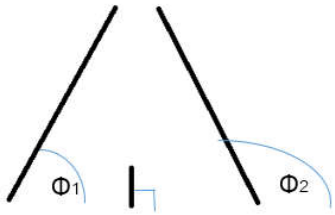
เพื่อหาเส้นที่ชัดที่สุดจาก hough line 2 เส้น โดย 2 เส้นที่ได้เป็นเส้นที่มีองศารวมกันเข้าใกล้ 0 มากที่สุด

```
function [Theta_ref,Theta_out,Rho_ref,Rho_out,error] = Theta_selector(Local_max,Theta,Rho,count)
[H,W] = size(Local_max)
error = 0;
mem = 100;
idx_out = 1;
%%
if count > 1
for x = 1:H
    if Theta(Local_max(x,1)) < 0
        Theta_ref = Theta(Local_max(x,1));
        Rho_ref = Rho(Local_max(x,2));
        break;
    else
    Theta_ref=Theta(Local_max(x,1));
    Rho_ref = Rho(Local_max(x,2));
    end
end
```

```
%% abs
for x = 1:H
A = abs(Theta(Local_max(x,1)) + Theta_ref);
    if A < mem
        mem = A;
        idx_out = x;
    end
end
Theta_out = Theta(Local_max(idx_out,1));
Rho_out = Rho(Local_max(idx_out,2));
%% error

if Theta_out > 1 | Theta_ref > 1
        error = 1;
end
if (Theta_out > 0 && Theta_ref > 0) | (Theta_out < 0 && Theta_ref < 0)
        error = 1;
end
else
    Theta_ref=Theta(Local_max(1,1));
    Rho_ref = Rho(Local_max(1,2));
    Theta_out=Theta(Local_max(1,1));
    Rho_out = Rho(Local_max(1,2));
end
```

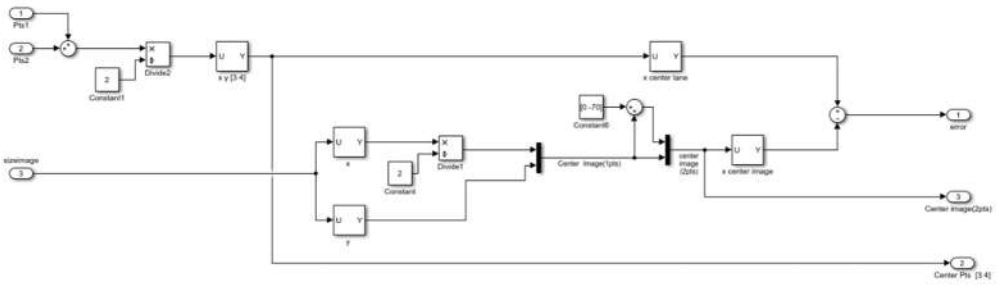
**Degree BLOCK**



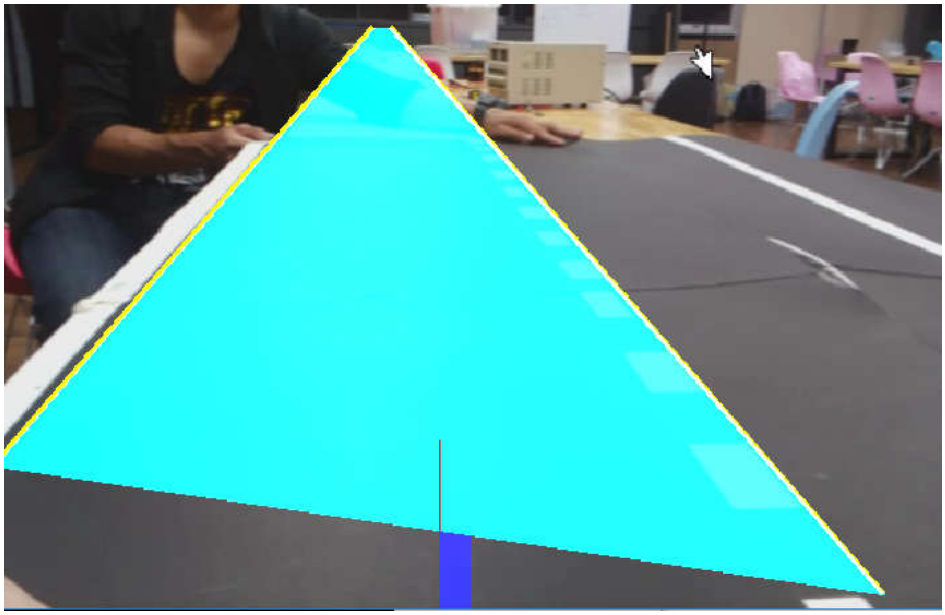




จาก Theta Selection _ Function นำองศามาแปลงเพื่อหาองศาการหมุนของล้อ เช่น หลังจากแปลง จะได้ theta1 มีขนาด 45 องศา theta2 มีขนาด 135 องศา รวมกันแล้วจะได้180 องศา นำมาหาร 2 จะ 90 องศา แล้วนำ 90 ลบ จะได้ 0 องศา คือทางตรง ล้อไม่หมุน

**ERROR BLOCK**

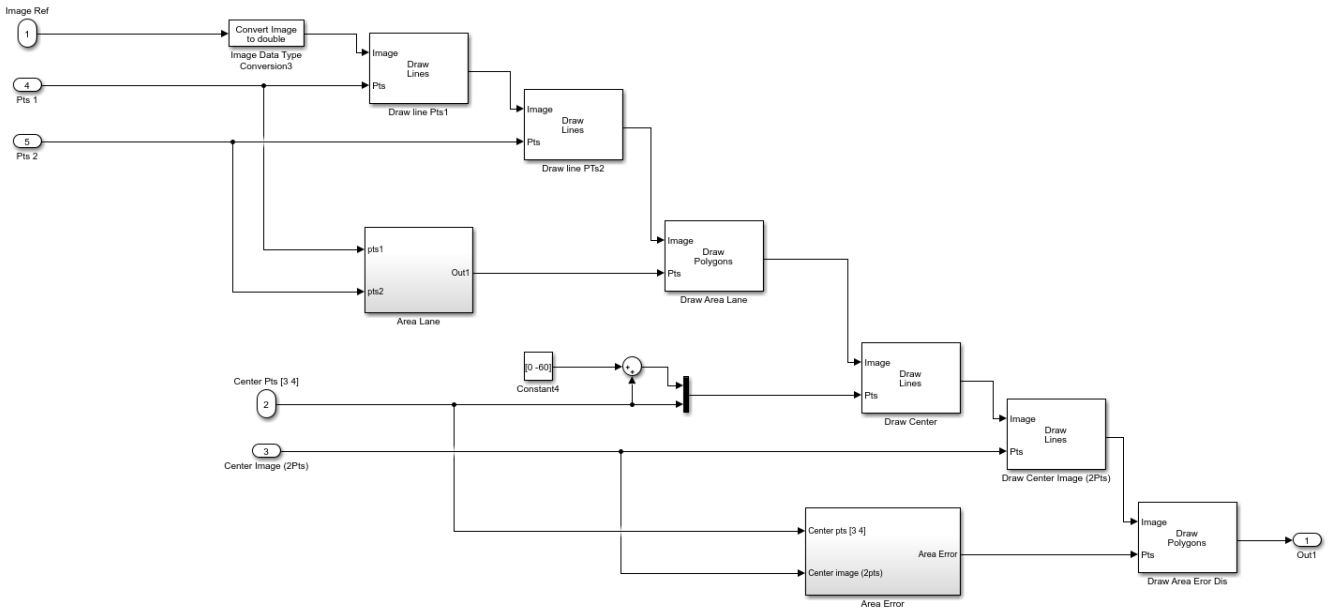


หาระยะห่างระหว่างกลางเลนกับตำแหน่งตรงกลางของรถ เพื่อให้เป็นระยะ error ว่ารถเคลื่อนที่อยู่กลางเลนหรือไม่ ถ้าค่าที่ได้เป็นบวก แสดงว่าตำแหน่งตรงกลางของรถอยู่ทางด้านขวาของกลางเลน ถ้าค่าที่ได้เป็นลบ แสดงว่าตำแหน่งตรงกลางของรถอยู่ทางด้านขวาของกลางเลน

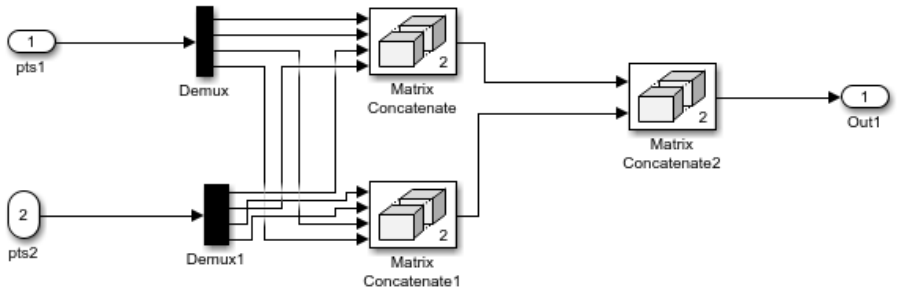


รูปภาพตัวอย่างแสดง พื้นที่สีน้ำเงิน คือระยะห่างระหว่างจุดกลางเลนกับตำแหน่งตรงกลางของรถ

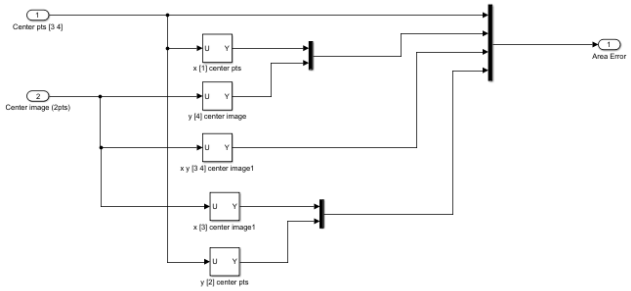
## Draw Graphic BLOCK



- **Area Lane BLOCK**



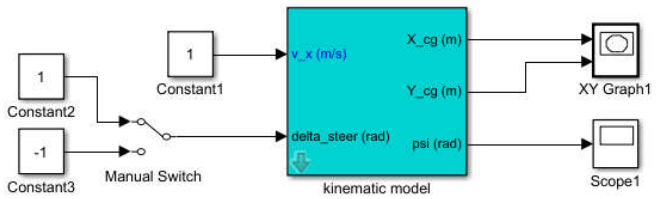

- **Area Error BLOCK**




ส่วนการวาดกราฟฟิค เพื่อแสดงผลลัพธ์ของการตรวจจับเลนเวลาทดสอบกับวิดีโอ มี Output เป็นสัญญาณรูปภาพที่วาดเส้นของเลน พื้นที่เลน  เส้นกลางเลน เส้นจุดกลางของรถ พื้นที่ระยะห่างระหว่างกลางเลนกับตรงกลางของรถ

# ขั้นตอนการดำเนินงานในส่วนของการควบคุมรถ

1. สร้าง kinematic model ของรถ เพื่อศึกษาลักษณะการเคลื่อนที่ของรถ




Kinematic model สามารถเขียนให้อยู่ในรูปของ state space ดังนี้

$$\vec{u} = \begin{bmatrix} v_x^L \\ \delta \end{bmatrix}$$

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_g^G \\ y_g^G \\ \varphi \end{bmatrix}$$

$$\dot{\vec{x}} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \cos x_3 & -\sin x_3 \\ \sin x_3 & \cos x_3 \end{bmatrix} \begin{bmatrix} u_1 \\ r_{og} \cdot \dot{x}_3 \end{bmatrix} \\ \frac{u_1}{L} \cdot u_2 \end{bmatrix}$$

$$\vec{y} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

โดยที่

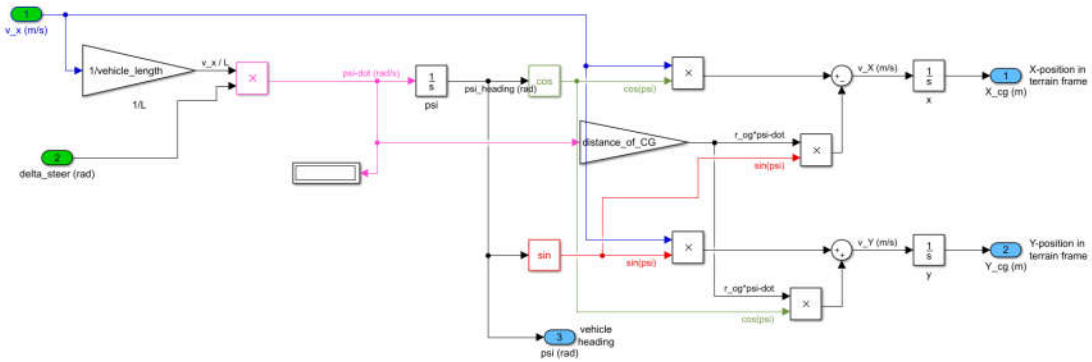
$v_x^L$ คือ ความเร็วในแนวแกน x ที่จุด o ใน local frame

$x_g^G$ คือ พิกัด x ที่จุด g ใน global frame

$y_g^G$ คือ พิกัด y ที่จุด g ใน global frame

$\varphi$ คือ orientation ของรถ ใน 2 dimensions

$r_{og}$ คือ ขนาดของเวกเตอร์ $\vec{r}_{og}$



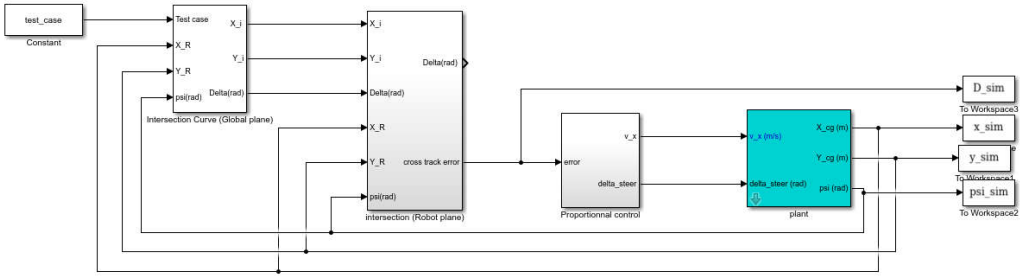

Kinematic model of 2D steering car

2. สร้างโมเดลจำลอง <u>เส้น</u> ในแบบต่างๆ เพื่อทำการทดสอบ line tracking algorithm

   1) Linear equation เพื่อจำลองสนามในทางตรง

   2) Parabola curve เพื่อจำลองสนามในทางโค้ง

   3) Arc of Circle เพื่อจำลองการเลี้ยวซ้าย หรือขวา

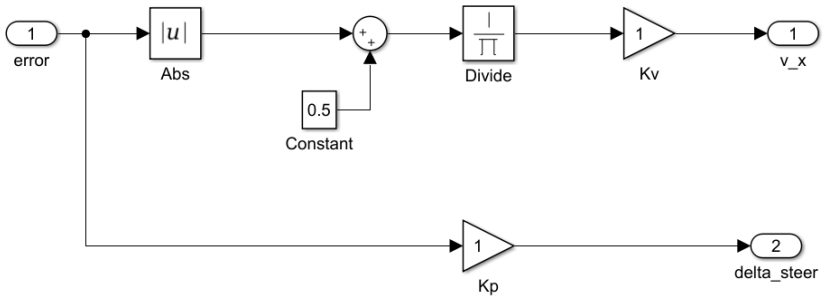



line tracking algorithm จะใช้ Proportional control ในการควบคุม plant ซึ่งรับ input เป็น cross track error โดยจะทำการควบคุมเพื่อให้ cross track error = 0 โดย cross track error สามารถหาได้จากการคำนวณทางคณิตศาตร์และเรขาคณิต

Controller นี้จะใช้สมการดังนี้

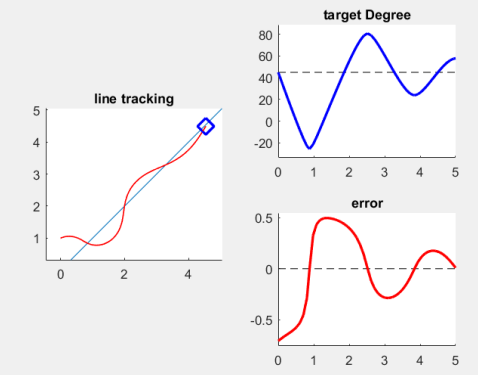
$$v_x^L = \frac{1}{e+0.5} \cdot Kv$$

$$\delta = e \cdot Kp$$

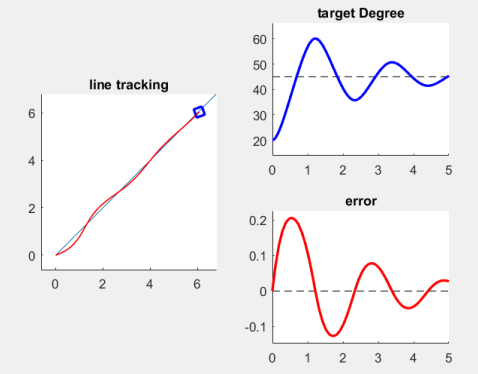


3. ทำการ test algorithm โดยเปลี่ยน characteristic ของ curve นั้นๆ ซึ่งเริ่มต้นจะให้ $Kv, Kp = 1$

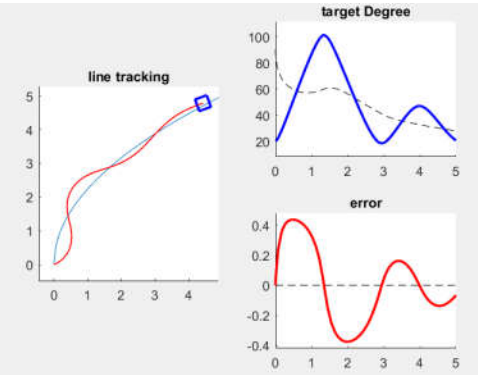
- Initial car: $(x_i, y_i) = (0,1)\,, \varphi_i = 45$
- Line: $y - y_0 = tan\theta(x - x_0)\,; x_0 = 0, y_0 = 0, \theta = 45$



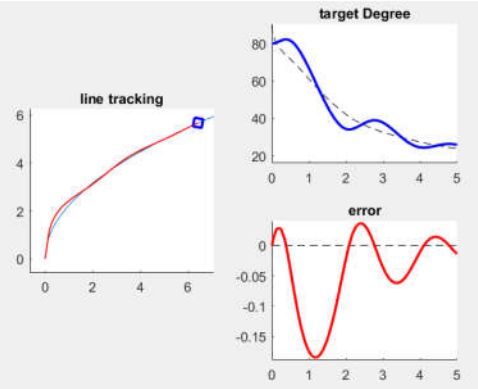

- Initial car: $(x_i, y_i) = (0,0)\,, \varphi_i = 20$
- Line: $y - y_0 = tan\theta(x - x_0)\,; x_0 = 0, y_0 = 0, \theta = 45$



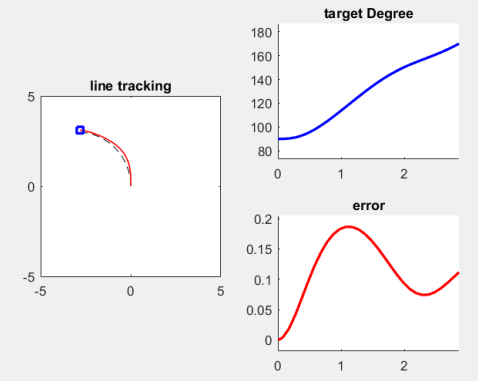

- Initial car: $(x_i, y_i) = (0,0)\,, \varphi_i = 20$
- Parabola: $y = \sqrt{\frac{x}{c}}; c = 0.2$



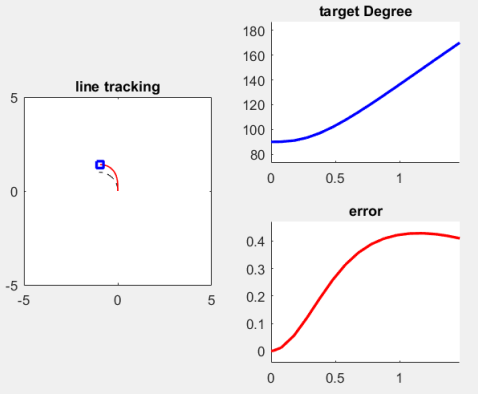
- Initial car: $(x_i, y_i) = (0,1)\,, \varphi_i = 80$
- Parabola: $y = \sqrt{\frac{x}{c}}; c = 0.2$




- Initial car: $(x_i, y_i) = (0,0)\,, \varphi_i = 90$
- Arc of Circle: $Radius = 3$

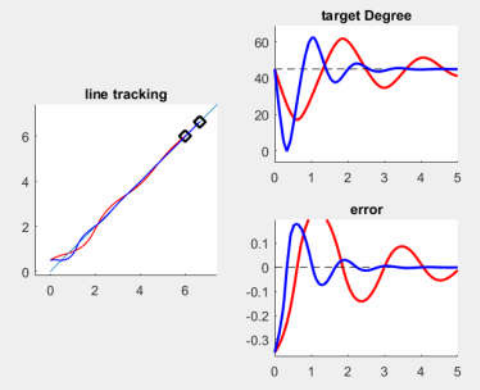



- Initial car: $(x_i, y_i) = (0,0)\,, \varphi_i = 90$
- Arc of Circle: $Radius = 1$



จากผลการทดสอบ พบว่า

I. อัลกอริทึมที่ใช้จะใช้งานได้ไม่ดีเมื่อมีค่า cross track error($e_p$) ที่มาก เพราะจะเกิดการ over shoot ของ error ทำให้ตัวรถเข้าสู่ steady state ได้ช้า ซึ่งสามารถแก้ไขได้เบื้องต้นโดยการเพิ่มค่า gain ให้มากขึ้น แต่ก็จะเกิดผลเสียเมื่อ มี $e_p$ ที่เยอะมากๆ จะทำให้รถเกิดการวิ่งเป็นวงกลมแทน

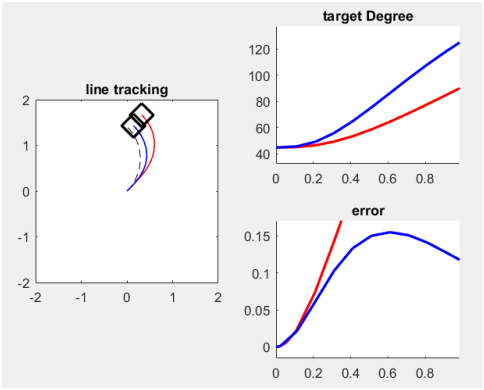



จากรูปจะสังเกตุได้ว่าเมื่อเพิ่มค่า $Kp$ มากขึ้น response จะสามารถเข้าสู่ steady state ได้เร็วขึ้น

Red: $Kp = 1$ , $Kv = 1$

Blue: $Kp = 3$ , $Kv = 1$

II. เมื่อเส้นมีการเปลี่ยน slope อย่างรวดเร็ว จะทำให้ response ของรถตอบสนองได้ไม่ทัน จะทำให้รถไม่สามารถแทร็กตามเส้น Arc of circle ที่มีรัศมีน้อยๆได้ ซึ่งสามารถแก้ไขได้เบื้องต้นโดยการเพิ่มค่า gain ให้มากขึ้น




จากรูปจะสังเกตุได้ว่าเมื่อเพิ่มค่า $Kp$ มากขึ้น response จะเร็วยิ่งขึ้นทำให้เกิด error น้อยลง

Red: $Kp = 1$ , $Kv = 1$

Blue: $Kp = 3$ , $Kv = 1$

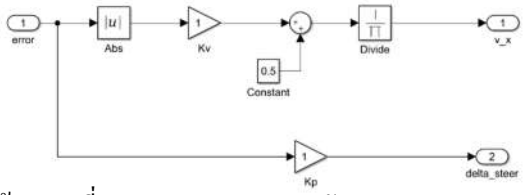
III. ความแตกต่างของ orientation ของตัวรถกับ เส้น ส่งผลให้ response เกิดการ over shoot เพิ่มขึ้น ซึ่งสามารถแก้ไขได้เบื้องต้นโดยการเพิ่มค่า $Kp$ ให้มากขึ้นเช่นกัน

ดังนั้น controller ตัวนี้สามารถแทร็กตามเส้นได้ แต่จะต้องทำการเปลี่ยนแปลงค่า $Kp$ ให้เหมาะสมตามสถานการณ์ แต่หากต้องคำนึงถึง error ที่เกิดขึ้นทำให้ controller ตัวนี้ไม่ควรนำไปใช้ เพราะมีเพียง Proportional gain ทำให้ response เกิด over shoot ที่มากเกินจะรับได้ นอกจากนี้เมื่อนำไปใช้งานจริงบนตัวรถค่า gain ที่มากจะทำให้ mechanic ของรถเกิดการ saturate ทำให้รถไม่สามารถ response ไปตาม controller ได้

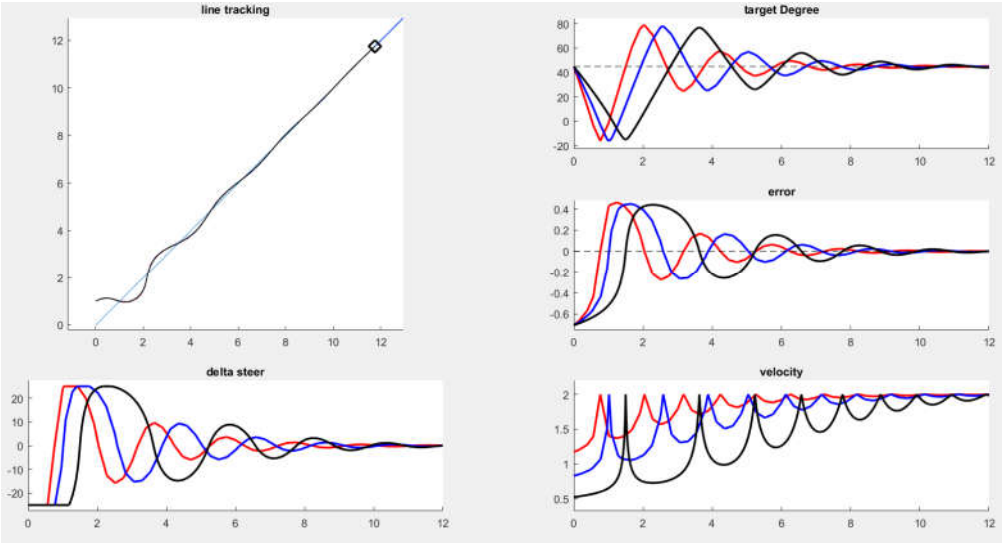
จากข้อสรุปที่ได้ทำให้มีแนวคิดที่จะเพิ่ม Derivative term เข้ามาเพื่อลดปัญหาการ over shoot และจะทำการนำผลต่างระหว่าง Orientation ของรถกับเส้น มาช่วยในการปรับปรุง controller ให้ดียิ่งขึ้น

4. พัฒนา controller เป็น

$$v_x^L = \frac{1}{(e \cdot Kv)+0.5}$$
$$\delta = e \cdot Kp$$



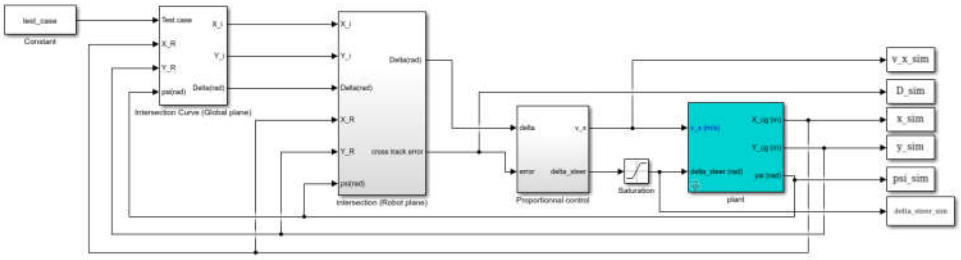

เพื่อทำการทดลอง โดยให้ $Kp$ คงที่ $Kv = 0.5,1,2$ ตามลำดับ (Red, Blue, Black)



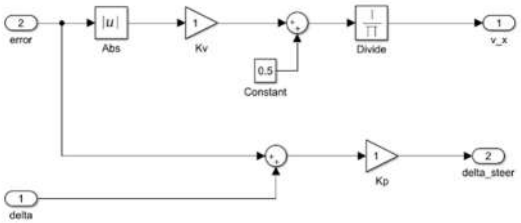

จากรูปพบว่า response ของระบบมี settling time และ %OS ใกล้เคียงกัน แต่มี peak time และ rise time ต่างกันโดยจะแปลผันตรงกับค่า $Kv$

เนื่องจาก response เกิดจาก kinematic model เพียงอย่างเดียว จึงไม่สามารถสรุปได้ว่า $Kv$ มีผลต่อ response ของระบบหรือไม่ แต่เพื่อให้ง่ายต่อการพัฒนา controller จึงประมาณว่า $Kv$ ไม่มีผลต่อ response ของระบบ

5. จากข้อ 3 ทำให้มีการนำผลต่างระหว่าง Orientation ของรถกับเส้นมาพัฒนา controller ได้เป็น




$$v_x^L = \frac{1}{(e \cdot Kv)+0.5}$$
$$\delta = (e + \Delta) \cdot Kp$$

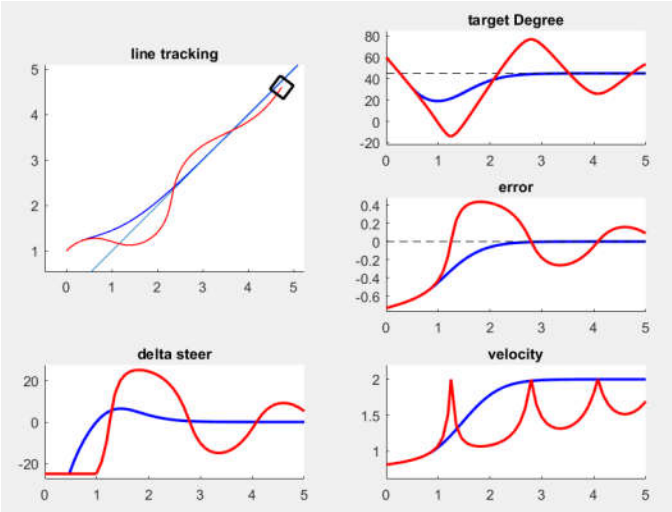


ทดสอบ controller version ใหม่ โดยการเปรียบเทียบกับ version เก่า ได้ผลดังนี้

I. Initial car: $(x_i, y_i) = (0,1)$ , $\varphi_i = 60$

Line: $y - y_0 = tan\theta(x - x_0)$ ; $x_0 = 0, y_0 = 0, \theta = 45$

$Kv, Kp = 1$



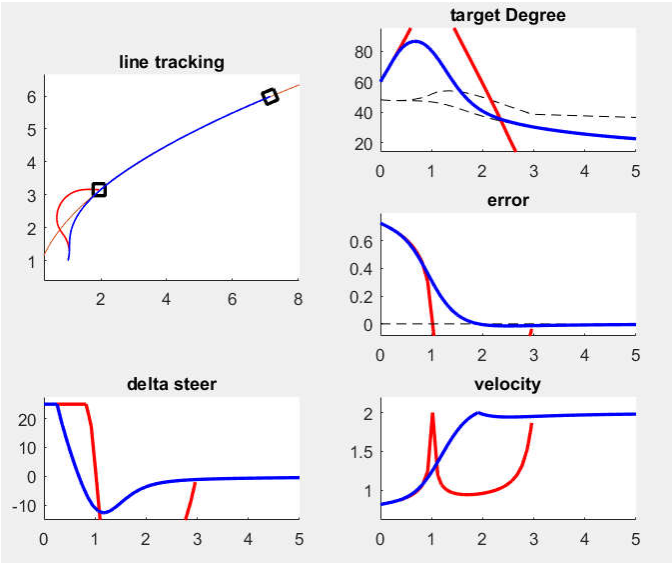

Red: Old Version

Blue: New Version

II. Initial car: $(x_i, y_i) = (1,1)$ , $\varphi_i = 60$

Parabola: $y = \sqrt{\frac{x}{c}}; c = 0.2$

$Kv, Kp = 1$

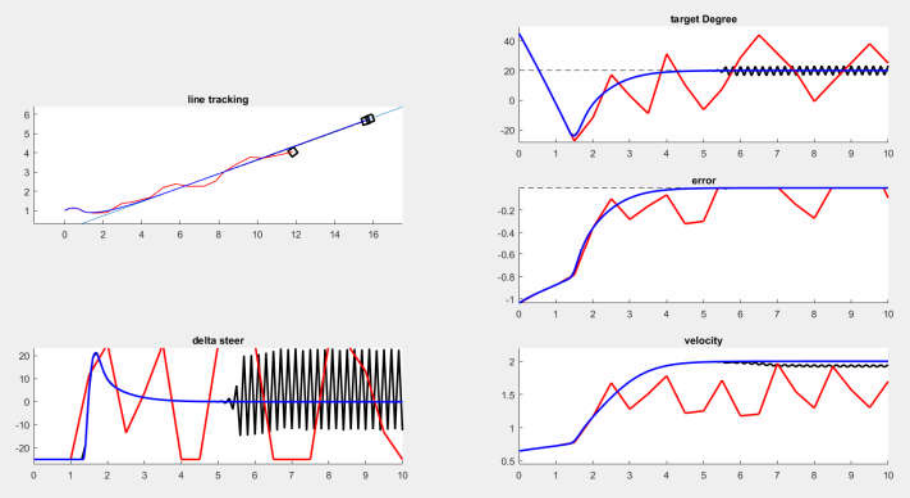



Red: Old Version

Blue: New Version

จากผลการทดลองพบว่า response มี Over shoot ต่ำลง Settling time และ Peak time เร็วขึ้น จึงสรุปได้ว่า controller new version มีประสิทธิภาพดีกว่า old version

6. ปรับการ Simulation จาก Continuous Time เป็น Discrete Time เพื่อทดสอบว่า controller ที่มีสามารถทำงานได้ใน Discrete Time



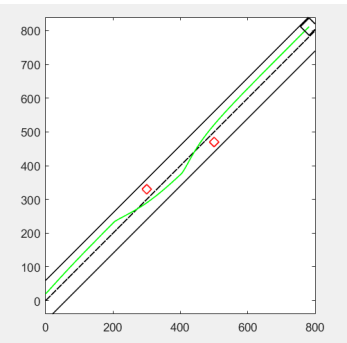

Red: Sampling time = 500 ms.

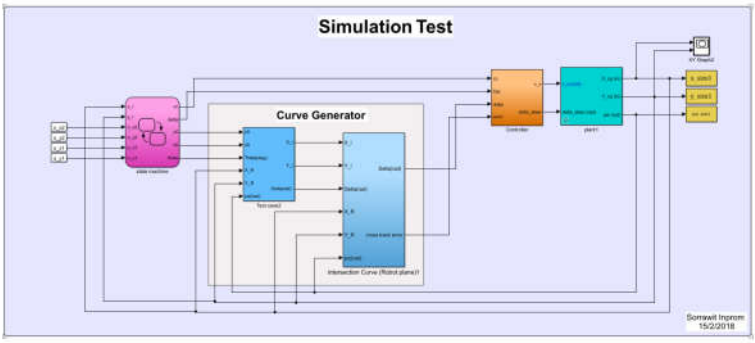
Black: Sampling time = 100 ms.

Blue: Sampling time = 50 ms.

จากผลการทดลองพบว่า Sampling time มีผลต่อการทำงานของ controller โดยถ้า Sampling time สูงเกินไปจะทำให้ระบบ Unstable หรือ Boundary Stable ซึ่งสามารถแก้ไขได้โดยการลด Sampling time ให้ต่ำลง อย่างไรก็ตาม Sampling time ที่น้อยเกินไปจะทำให้ระบบ Physical response ไม่ทัน ดังนั้นจึงจำเป็นต้องหา Sampling time ที่เหมาะสมในการทดลองแบบ Physical ต่อไป

7. นำ controller ที่ได้มาทำการ test algorithm การ avoid obstacle โดยทำการจำลอง map ขึ้นมาจาก dimension ของสนามจริง และจำลอง state flow ที่รับ input เป็น Distance จาก Lidar ซึ่งวัตถุประสงค์ที่ทำการทดลองเพื่อหา distance ที่เหมาะสมที่รถจะสามารถหลบสิ่งกีดขวางได้พอดีโดยไม่เลี้ยวหลุดนอกสนาม






8. รวม Model แต่ละส่วน และ ออกแบบ State Machine สำหรับใช้จริง จากนั้นนำรถไปทำ Physical และ Prototype testing โดยสร้างสนามจำลองขึ้นมา

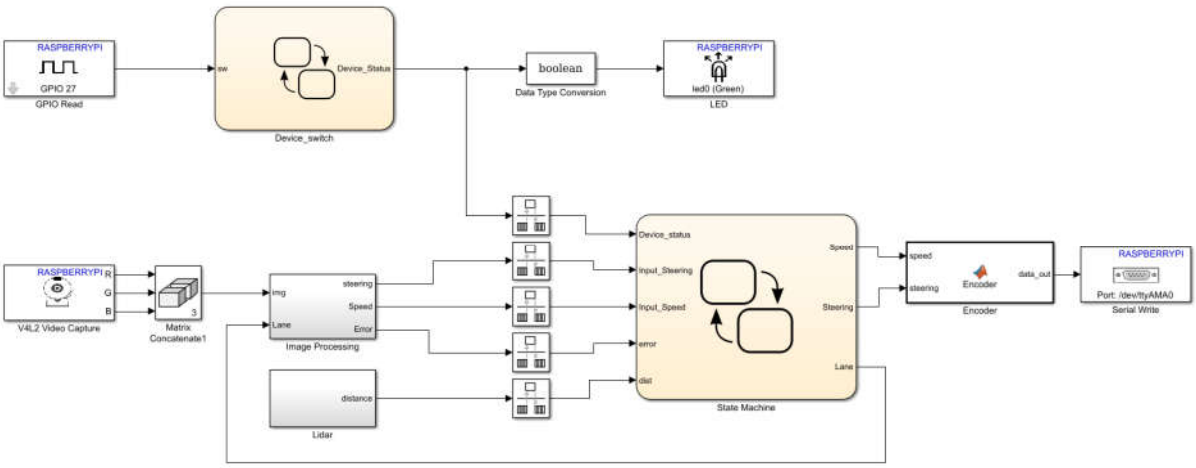
**Latest Model**

1. Raspberry Pi Board

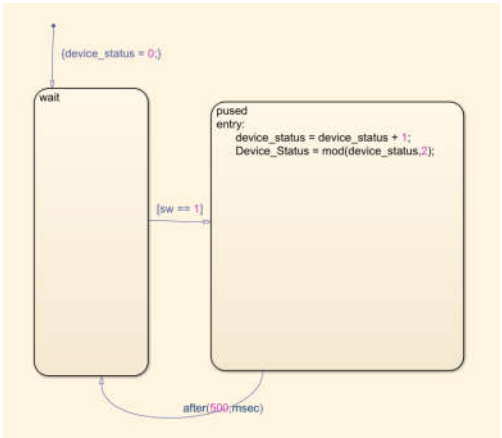
Feature (HIGH-LEVEL)

- Image Processing: Road Tracking and Find cross track error
- Lidar: Calculate Distance
- State Machine: Planning action of a car in each state and what will happen in next state
- Serial Communication: Send data to low level

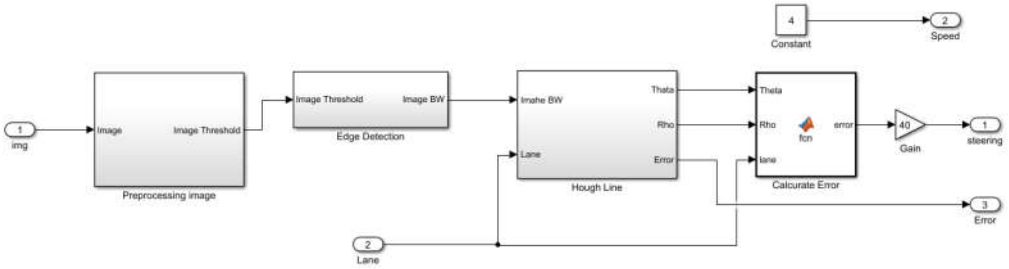
ภาพรวมของmodel



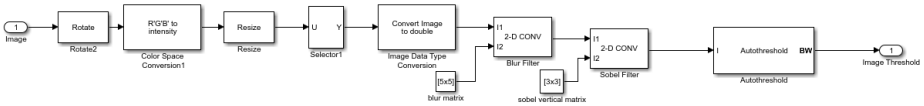

1. Device switch Chart: ทำหน้าที่รับ input จาก switch บน RPI เพื่อสร้าง signal 1/0 ส่งให้ตัว state machine



2. Image Processing: รับ input เป็นรูปภาพจาก Pi-camera และ สถานะปัจจุบันของรถว่าอยู่ lane ไหน เพื่อคำนวณ error ได้ถูกต้องตาม algorithm

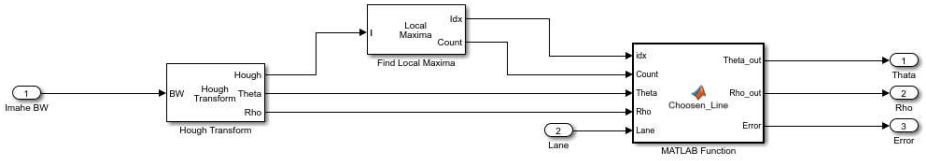



1) Preprocessing image: จัดการรูปภาพทั้งหมดก่อนนำไปเข้า edge detection




2) Edge Detection: หาขอบและแปลงเป็น binary image เพื่อใช้ในการหาเส้นต่อ


3) Hough Line: ใช้ Hough transform ในการหาเส้นภายในภาพ และทำการหา local maximum เพื่อเลือกเส้นที่เด่นที่สุดมาจากนั้นเขียนฟังก์ชัน chosen line เพื่อเลือกเส้นที่เป็นถนนจริงๆ



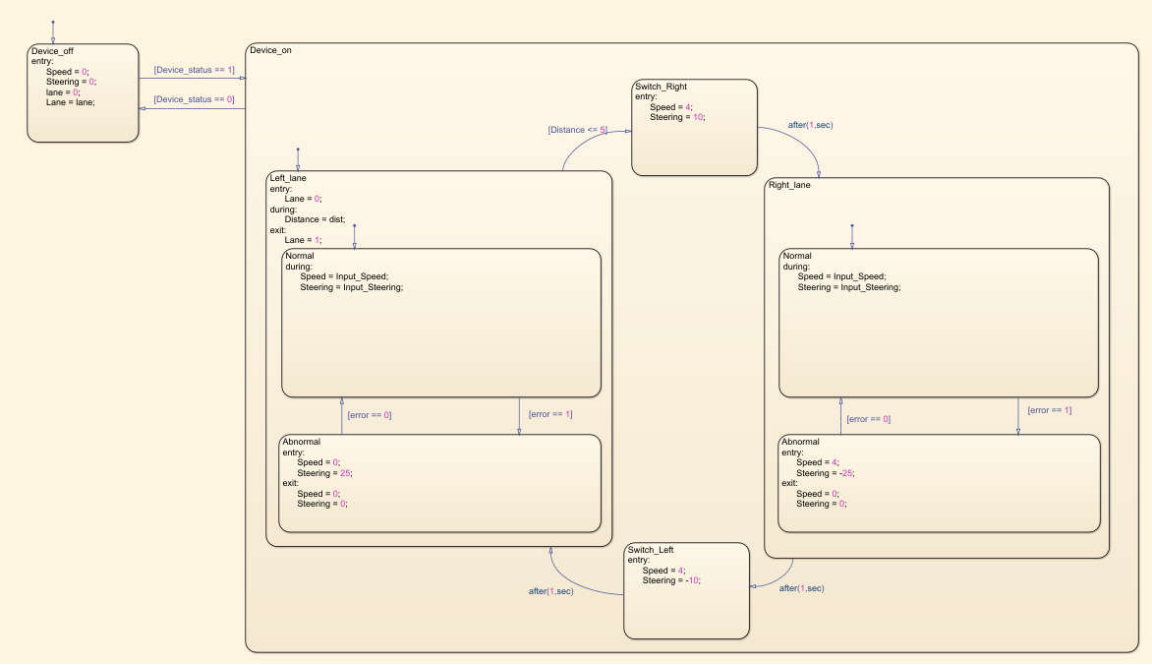

4) นำ เส้นที่ได้ไปหา cross track error ต่อไปในฟังก์ชั่น Calculate error

3. Lidar: ทำการคำนวณหาระยะห่างของวัตถุที่อยู่หน้ารถ

ตอนนี้อ่านค่า distance (cm) ได้ใน script Matlab แต่ยังติดปัญหาในการแปลงเป็น Simulink จึงทำให้ยังไม่ได้ Model ออกมา

```
mypi = raspi;  %raspberry pi connect to matlab
pause(1)

lidar_id = scanI2CBus(mypi,'i2c-1');
lidar_device = i2cdev(mypi,'i2c-1','0x62');
pause(1)

%% configure lidar
% Reset Lidar
disp("Reset Lidar ...")
writeRegister(lidar_device,0,0);
pause(1)

% Default mode
disp("Configure Lidar : Default mode ...")
writeRegister(lidar_device,2,128);
pause(0.5)
writeRegister(lidar_device,4,8);
pause(0.5)
writeRegister(lidar_device,28,0);
pause(0.5)

%% read lidar
for i = 1:100

    writeRegister(lidar_device,0,4);
    pause(0.5)

    distance_h = readRegister(lidar_device,15);
    pause(0.1)
    distance_l = readRegister(lidar_device,16);
    pause(0.1)

    disp(distance_h)
    disp(distance_l)

end
```

4. State Machine: วางแผนการทำงานของระบบทั้งหมด



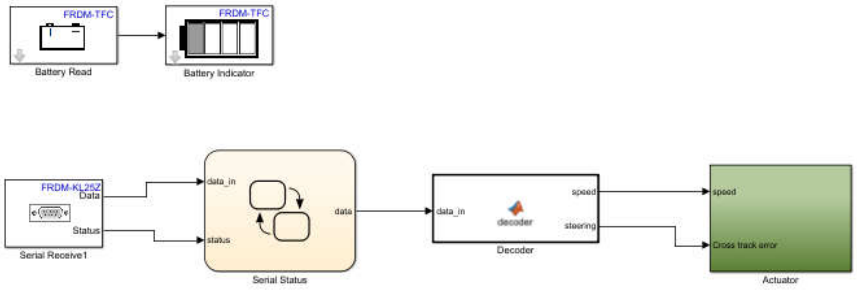
5. Function Encoder: ทำหน้าที่รับ data เข้ามาและแปลงให้อยู่ในรูป data 32-bit ประเภท uint32 1 length เพื่อส่งไปให้ Free Scale Board

2. Freescale Board

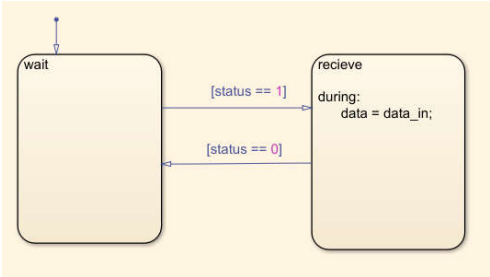
Feature(LOW-LEVEL)

- Serial Communication: Receive data from high level
- Proportional Control: Control steering angle with respect to cross track error

ภาพรวมของ model




1. Serial Status Chart รับข้อมูล และ status ที่ได้จากบล็อก serial receive เพื่อตัดสินใจจะส่งข้อมูลต่อไป




2. Decoder: แปลงข้อมูล uint32 ไปเป็น Speed กับ steering angle ให้กับ actuator ต่างๆ
3. Actuator: สั่งการทำงานให้กับ มอเตอร์และ servo