Xenobot technical report
Lane following

Shengwen Cheng , Po-Sheng Chen

January 2017

1 Introduction

Xenobot is a computer vision based self-driving system inspired by MIT duckietown project. We re-implement our own system for real-time robotics research and may add more new features in the future.

The algorithm we're using is a modified version of MIT duckietown, so you can find many similarities between two projects. This report is focus on how our lane following algorithm works.

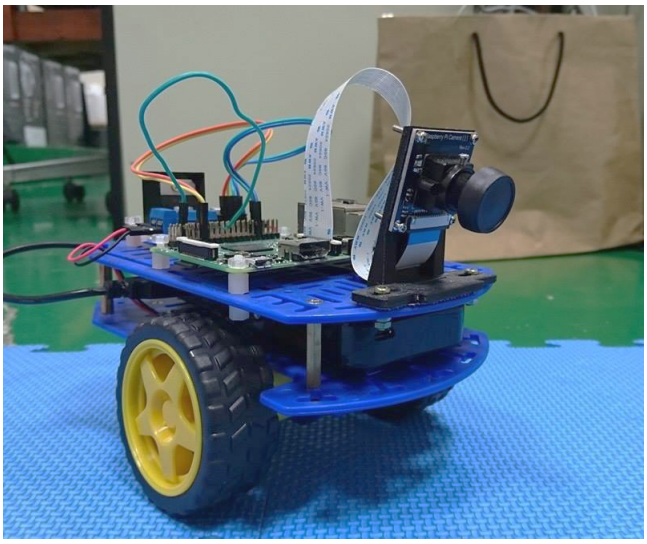

Figure 1: Xenobot

2 Camera calibrations

Camera calibration is a early stage mission before doing the computer vision, they're two set of parameters need to be found, intrinsic parameters and extrinsic parameter, by estimate them we can know the relation between 2D image plane and 3D world.

2.1 Intrinsic calibration

The intrinsic matrix describes the projection from 3D world into 2D image plane. The ideal linear model is consist of focal length, pixel size and principal point. The nonlinear effect like lens distortion are also important however can not described in the linear camera model and usually solved by numerical methods.

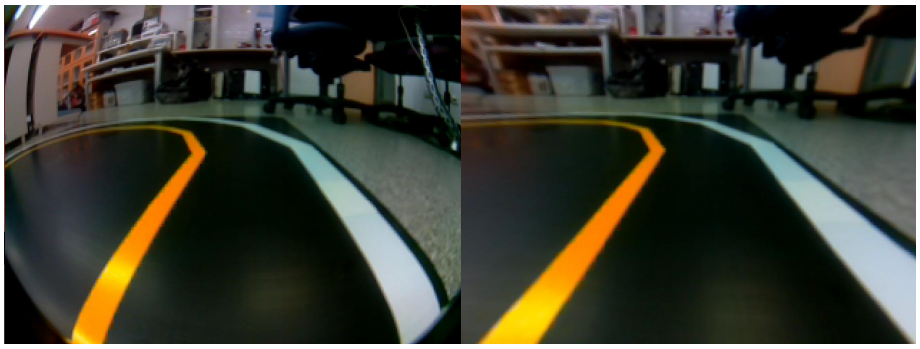

Figure 2: Camera undistortion

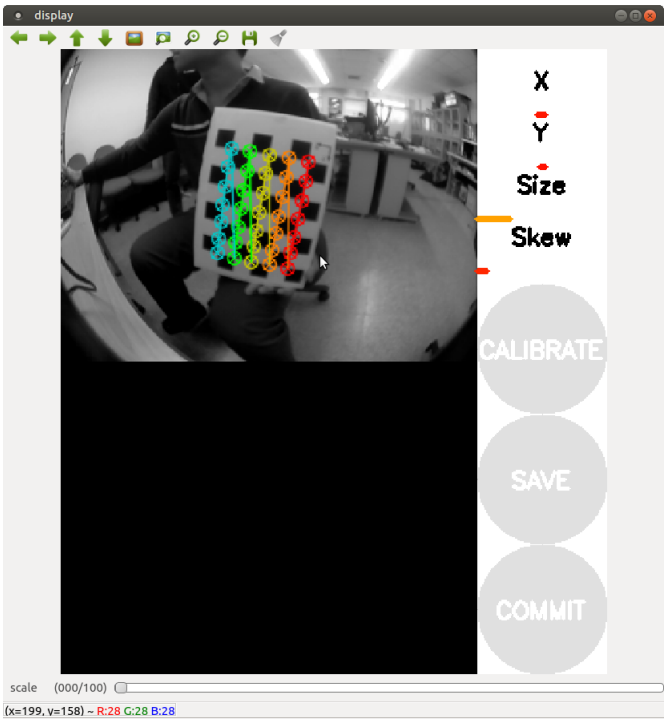

Figure 3: Intrinsic calibration

2.2 Extrinsic calibration

The extrinsic matrix describes the translation and rotation of camera with respect to the world frame, which means the angle and position of camera taking pictures in 3D world.

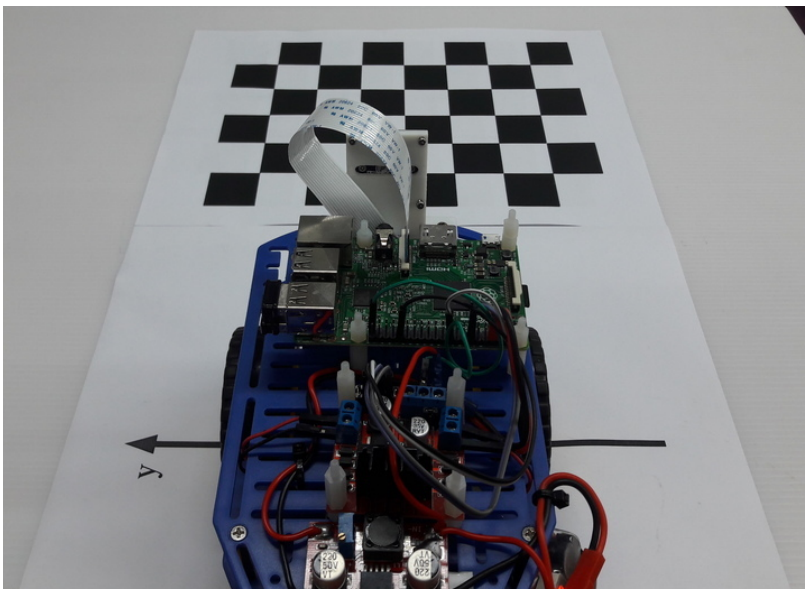

Figure 4: Extrinsic calibration

3 Lane pose estimation

3.1 System conventions

We define x axis point to the right, y axis point foward and z axis point upward as Xenobot's coordinate system.

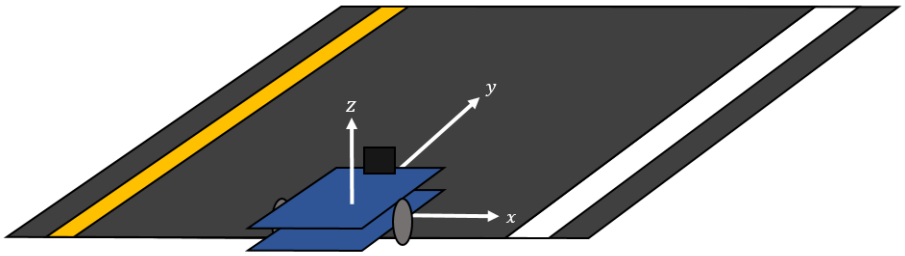

Figure 5: Lane coordination

Another assumption is that the self-driving car runs between yellow line and white line. The yellow line is at the left side and the white line is at the right side.

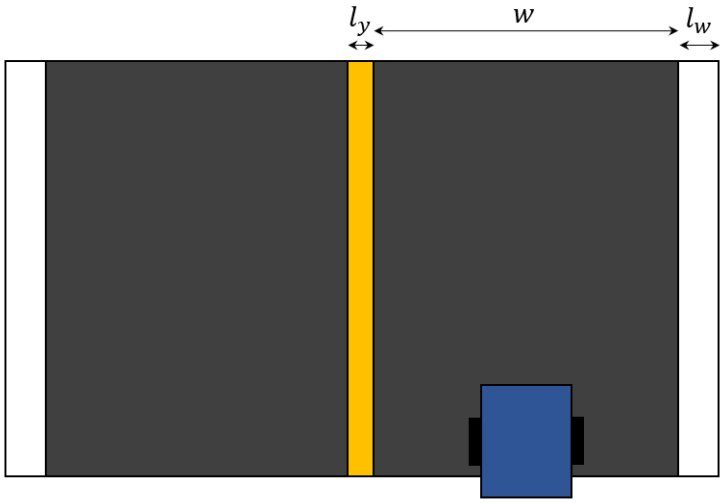

Figure 6: 2D view of lane

Now define the pose with respect to the lane, $\boldsymbol{d}$ is the lateral displacement and $\boldsymbol{\phi}$ is the orientation angle.

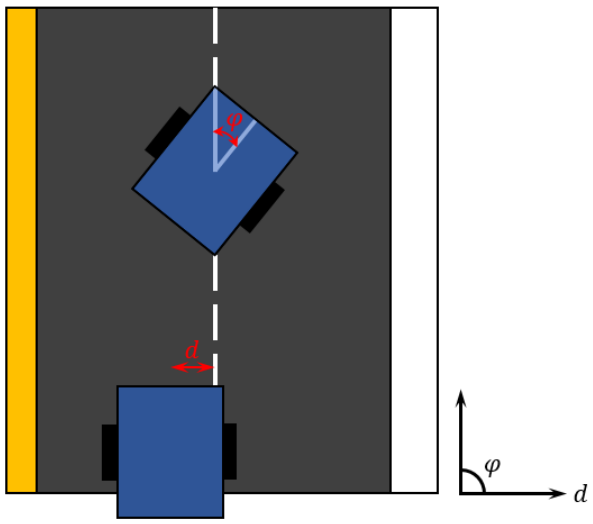

Figure 7: Lane pose

3.2 Segements detection

Before the pose estimation, first have to do is detect the lane segments from the image. The lane detector can be achieved by using Canny edge detection, color thresholding method and Hough transform.

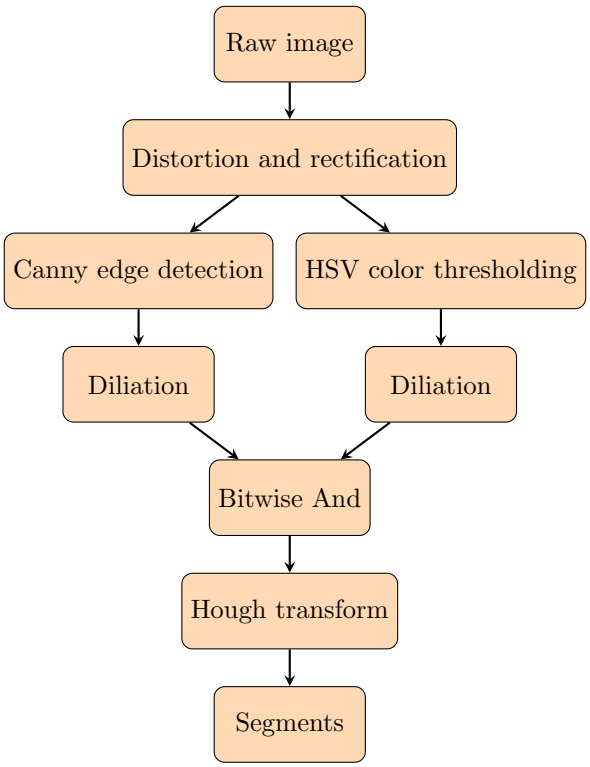

Figure 8: Workflow of lane detector

First, transform the image from RGB color space into the HSV color space then use the color thresholding method to find the color like yellow or white in the image.

Figure 9: Binary color image of color thresholding result

Next, use Canny edge detection to detect the edges in the image. This is a preparation step for Hough transform.

Figure 10: Result of Canny edge detection

Finally, use the bitwise and operation on color thresholding image and Canny edge image then apply the Hough transform comes out the segments. The following image visualizes the segment locations and also shows the values of d and ϕ.

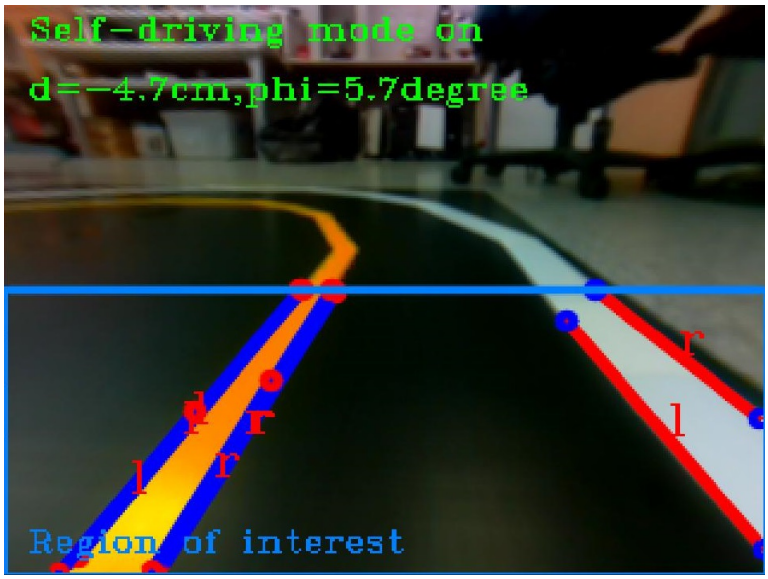

Figure 11: Segments visualization (The lane detector only works at the botom ROI(Region of Interest) part because there is much possible to be the lane)

3.3 Frame transformation

The camera on the car is mounted in a distance from the origin. In order to know the real lateral displacement of the car, a transformation is need for converting camera frame into car frame.

$$\begin{pmatrix} x_{car} \\ y_{car} \end{pmatrix} = \begin{pmatrix} x_{camera} - r \cdot \sin\phi \\ y_{camera} - r \cdot \cos\phi \end{pmatrix}$$

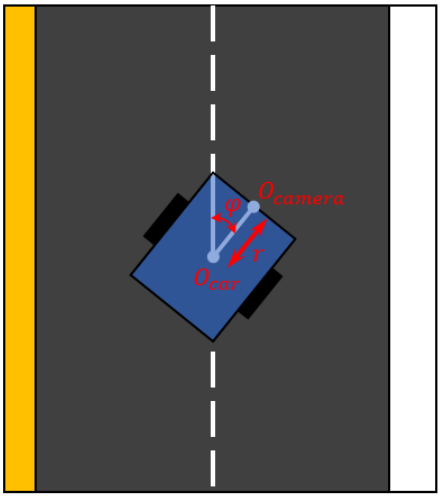

Figure 12: Frame transformation

3.4 Segment side recognition

Although the lane detector can find the segments, we still don't know the side of the lane mark that the segments is on.

Segment side can be determined by reading multiple pixels in both positive and negative direction of the segment normal vector on color thresholded image.

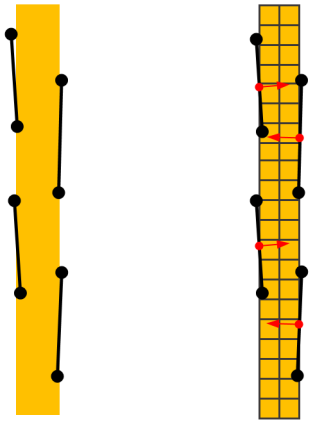

Figure 13: Segment side recognition

Algorithm 1: Segment side recognition

Data: segment, accumulator threshold, color binarization image
Result: side (left or right)

$\vec{P_1} = (x_1, y_1)$
$\vec{P_2} = (x_2, y_2)$
$\vec{P} = (\vec{P_1} + \vec{P_2})/2$
$\vec{t} = \frac{\vec{P_2} - \vec{P_1}}{\left\| \vec{P_2} - \vec{P_1} \right\|}$
$\vec{n} = (-y_t, x_t)$
$left \leftarrow 0$, $right \leftarrow 0$
for $i < pixel\ count$ **do**
    $x \leftarrow \lceil x_p + x_n \cdot i \rceil$
    $y \leftarrow \lceil y_p + y_n \cdot i \rceil$
    if $I(x, y) = I_{max}$ **then**
        $left \mathrel{+}= 1$
    $x \leftarrow \lfloor x_p - x_n \cdot i \rfloor$
    $y \leftarrow \lfloor y_p - y_n \cdot i \rfloor$
    if $I(x, y) = I_{max}$ **then**
        $right \mathrel{+}= 1$
end
if $left > threshold$ & $right < threshold$ **then**
    return is_left
else if $right > threshold$ & $left < threshold$ **then**
    return is_right
else
    return unknown side

3.5 Ground projection

During the calibration stage, we found the homography matrix that map the image from car's view to the road upper view. This is usally called a bird eye view or a inverse perspective mapping image.

We'll apply this transformation before further steps.

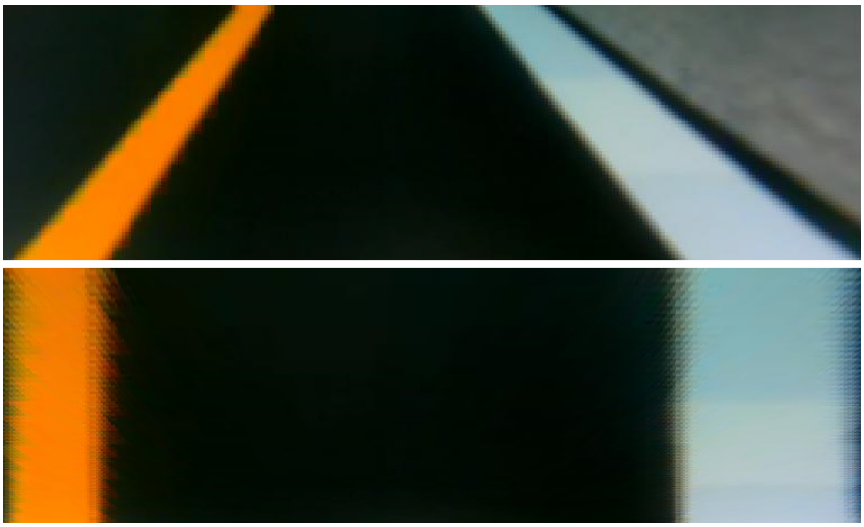

Figure 14: Ground projection

3.6 Pose estimation for single segment

Each of the segment that lane detector detects can be calculated to be a single pose data due to the lane geometry. We will put these datas into the histogram filter for data filitering later. The process is similar to vote so we call it a vote generating function.

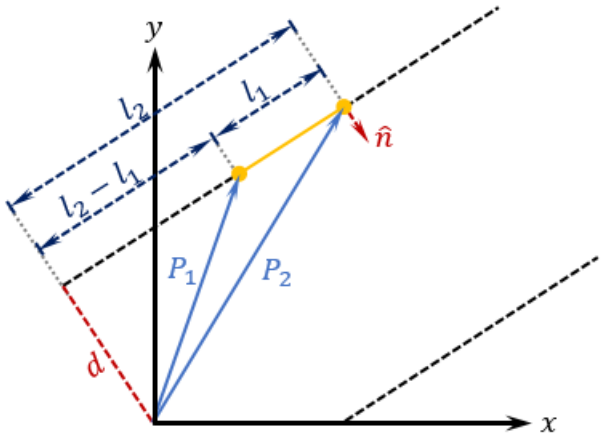

Figure 15: Lane geometry

Algorithm 2: Generate vote

Data: segment
Result: pose d_i and ϕ_i

$\vec{P_1} = (x_1, y_1)$
$\vec{P_2} = (x_2, y_2)$
$\vec{t} = \frac{\vec{P_2} - \vec{P_1}}{\left\| \vec{P_2} - \vec{P_1} \right\|}$
$\vec{n} = (-y_t, x_t)$
$\phi_i = \arctan(\frac{y_t}{x_t}) - \pi/2$
if *segment color = white* **then**
    if *edge side = right* **then**
        $\vec{k} = (\frac{w}{2} + l_w) \cdot \vec{n}$
    else
        $\vec{k} = (\frac{w}{2}) \cdot \vec{n}$
    end
else if *segment color = yellow* **then**
    if *edge side = left* **then**
        $\vec{k} = (-\frac{w}{2} - l_y) \cdot \vec{n}$
    else
        $\vec{k} = (-\frac{w}{2}) \cdot \vec{n}$
    end
$\vec{j} = (r \cdot \sin\phi, r \cdot \cos\phi)$
$\vec{P_1'} = \vec{P_1} + \vec{k} - \vec{j}$
$\vec{P_2'} = \vec{P_2} + \vec{k} - \vec{j}$
$d_1 = \vec{P_1} \cdot \vec{n}$
$d_2 = \vec{P_2} \cdot \vec{n}$
$d_i = (d_1 + d_2)/2$

3.7 Histogram filter

The segment poses generated in previous process will then be voted into the histogram filter. The histogram filter will find out the **mode** of these datas and eliminate the data out of the mode with certain range.

After that, calculate the **mean** of the rest of the datas comes out a much accurate pose information.

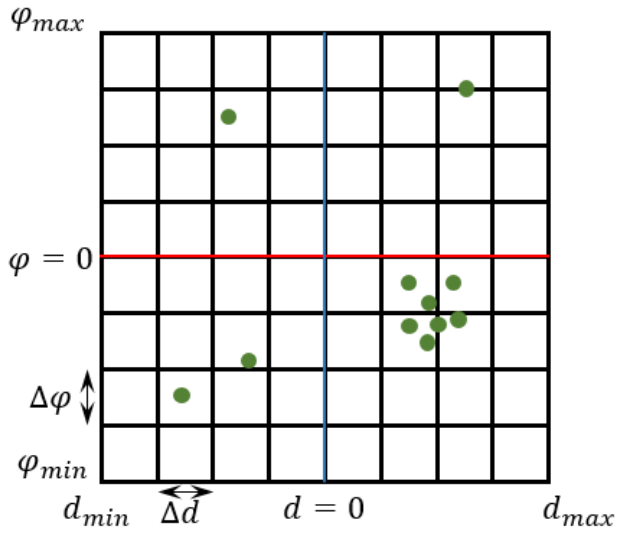

Figure 16: Histogram filter

Algorithm 3: Histogram filter

Data: segment
Result: filtered pose d and ϕ

for ***all*** *segments* **do**
  $(\phi_i, d_i) \leftarrow generate\_vote(segment)$
  $I \leftarrow round(\frac{\phi_i - \phi_{min}}{\Delta\phi})$
  $J \leftarrow round(\frac{d_i - d_{min}}{\Delta d})$
  $histogram(I, J) \mathrel{+}= 1$
end
$(I_{highest}, J_{highest}) \leftarrow find\_highest\_vote()$
$\phi_{histogram} \leftarrow I_{highest} \cdot \Delta\phi + \phi_{min}$
$d_{histogram} \leftarrow J_{highest} \cdot \Delta d + d_{min}$
$\phi_{mean} \leftarrow 0\ ,\ d_{mean} \leftarrow 0$
$N_\phi \leftarrow 0\ ,\ N_d \leftarrow 0$
for ***all*** (ϕ_i, d_i) **do**
  if $\phi_i \in [\phi_{histogram} - \frac{\Delta\phi}{2}, \phi_{histogram} + \frac{\Delta\phi}{2}]$ **then**
    $\phi_{mean} \mathrel{+}= \phi_i$
    $N_\phi \mathrel{+}= 1$
  if $d_i \in [d_{histogram} - \frac{\Delta d}{2}, d_{histogram} + \frac{\Delta d}{2}]$ **then**
    $d_{mean} \mathrel{+}= d_i$
    $N_d \mathrel{+}= 1$
end
$\phi_{mean} \leftarrow \frac{\phi_{mean}}{N_\phi}$
$d_{mean} \leftarrow \frac{d_{mean}}{N_d}$

4 Control system

4.1 Differential wheels

Xenobot is a differential wheeled robot. There are two independent motors on the car. The car can be turn by changing the speed rate between two motors, hence it does not require any additional steering mechanics.

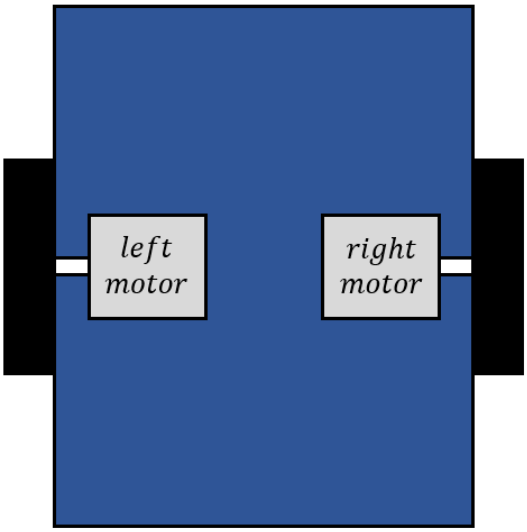

Figure 17: Differential wheeled robot

4.2 PID Controller

Xenobot use the well known algorithm **"PID controller"** to fix the orientation and lateral displacement.

The equation of PID controller in continuous time is given as:

$$e(t) = setpoint(t) - x(t)$$

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

and for discreted time:

$$e[t] = setpoint[t] - x[t]$$

$$u[t] = K_p e[t] + K_i \sum_{0}^{t} e[t]\Delta t + K_d \frac{e[t] - e[t-1]}{\Delta t}$$

4.3 Pose control

The pose controller of Xenobot is a cascaded PID controller.

The phi controller is a low level contoller for lane orientation stablizing, and d controller fix the lateral displacement by changing the setpoint of phi controller to turn left or right.

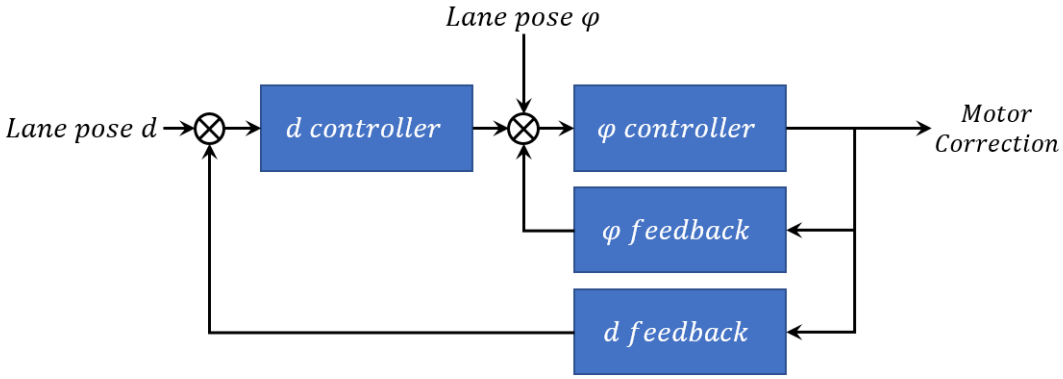

Figure 18: Control diagram

The wheel control signal (PWM) is simply the throttle value plus the correction value:

```
pwm_left = THROTTLE_BASE - pwm_correction
pwm_right = THROTTLE_BASE + pwm_correction
```

5 References

[1] Lane Filter by Liam Paull, MIT CSAIL