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.

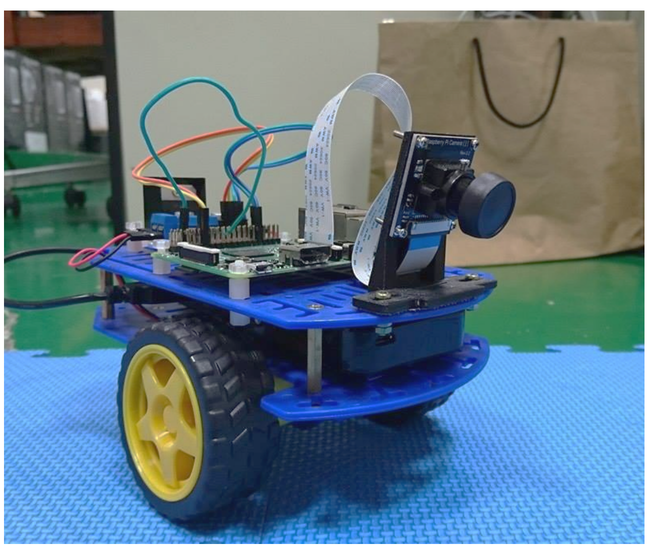

Figure 1: Xenobot

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 parameters

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.

2.2 Extrinsic parameters

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

3 Lane pose estimation

3.1 Lane pose and system conventions

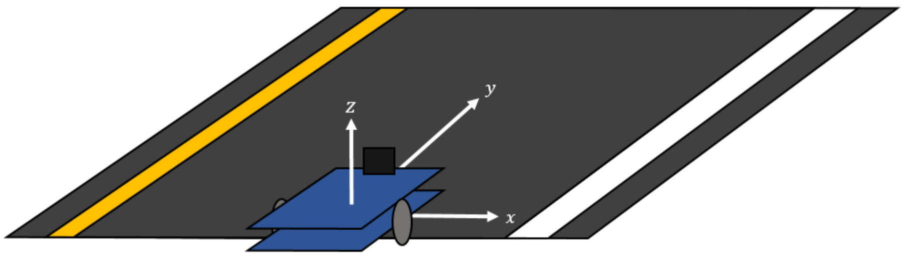

Figure 2: Lane coordination

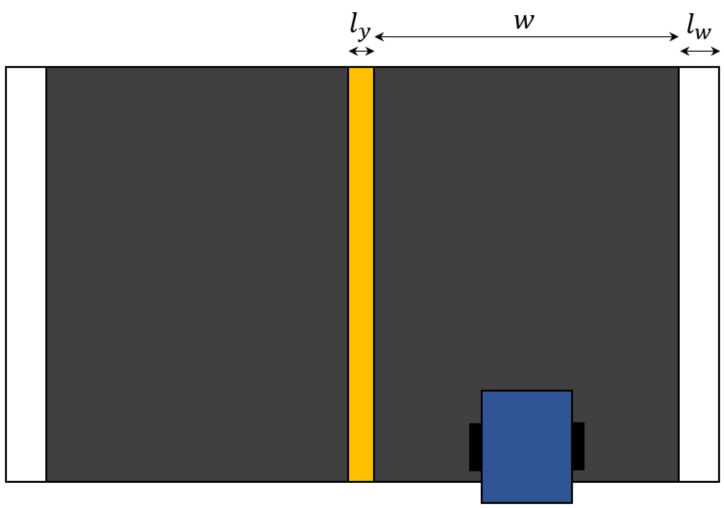

Figure 3: 2D view of lane

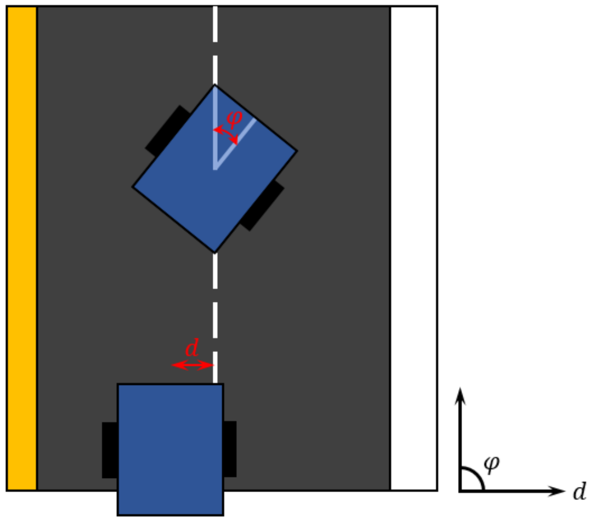

Figure 4: Lane pose

3.2 Segements detection

The first step for lane pose estimation is to extract the segments from the image, the process to do this is to threshold the specific color we want, and apply canny edge detector and Hough transform so we can find out the location of those segments

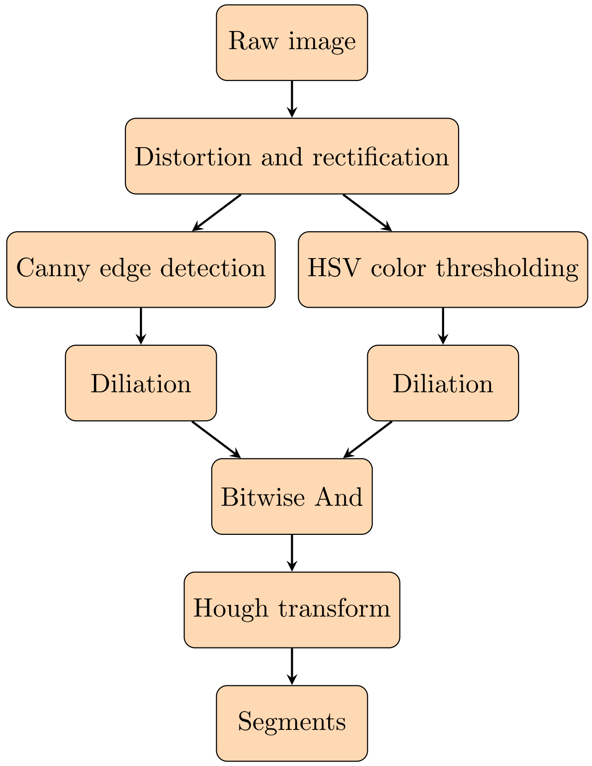

Figure 5: Work flow of lane detector

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.

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

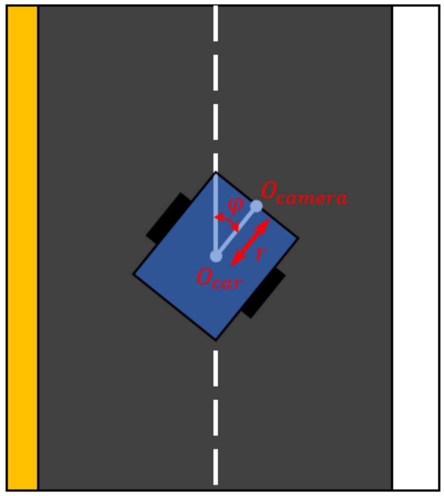

Figure 6: Frame transformation

3.4 Segment side recognition

Canny edge detection and Hough transform find the lane segments for us. However, we still don't know the segments is on which side of the lane mark.

To determine the side, we can read multiple pixels in both positive and negative direction of the segment normal vector on color thresholded image.

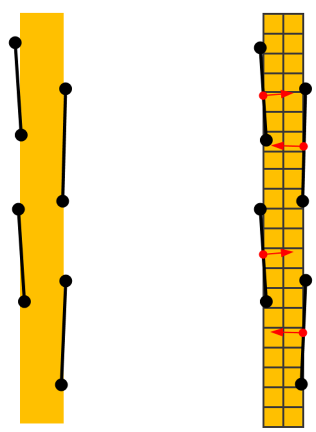

Figure 7: Lane segments

Algorithm 1: Segment side recognition

Data: segment, accumulator threshold, color binarization image

Result: side (left or right)

```
P⃗1 = (x1, y1)
P⃗2 = (x2, y2)
P⃗ = (P⃗1 + P⃗2)/2
t⃗ = (P⃗2 − P⃗1) / ‖P⃗2 − P⃗1‖
n⃗ = (−y_t, x_t)
for i < pixel count do
    x ← ⌈x_p + x_n · i⌉
    y ← ⌈y_p + y_n · i⌉
    if I(x, y) = I_max then
        left += 1
    x ← ⌊x_p − x_n · i⌋
    y ← ⌊y_p − y_n · i⌋
    if I(x, y) = I_max then
        right += 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 Segment pose estimation

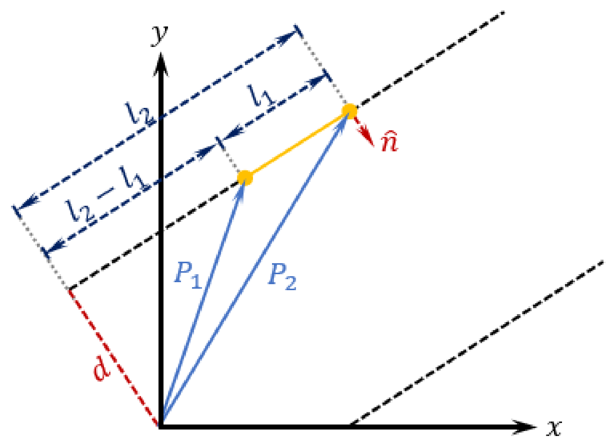

Figure 8: Lane geometry

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
$\vec{j} = (r \cdot \sin\phi, r \cdot \cos\phi)$
$\vec{P_1'} = \vec{P_1} + \vec{k} - \vec{j}$
$\vec{P_2'} = \vec{P_2} + \vec{k} - \vec{j}$
$d_1 = \vec{P_1} \cdot \vec{n}$
$d_2 = \vec{P_2} \cdot \vec{n}$
$d_i = (d_1 + d_2)/2$

Figure 9: Vote generation

3.6 Histogram filter

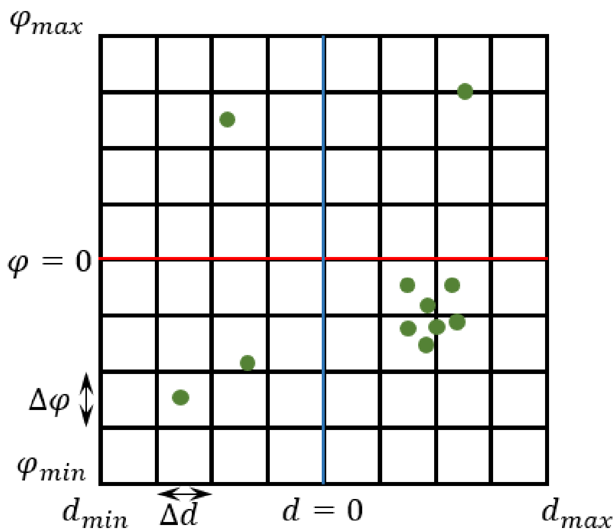

Figure 10: Histogram filter

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
$\phi_{mean} \leftarrow \frac{\phi_{mean}}{N_\phi}$
$d_{mean} \leftarrow \frac{d_{mean}}{N_d}$

Figure 11: Histogram filter

4 Control system

4.1 Differential wheels

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.

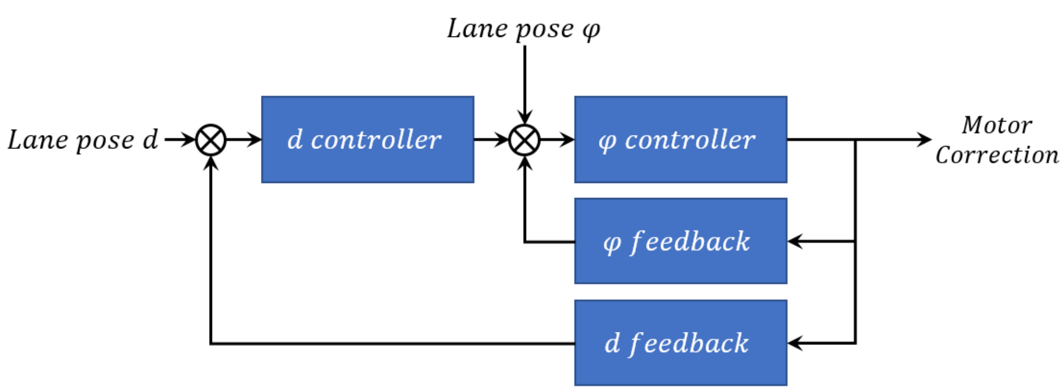

Figure 12: Control diagram

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