Report of Puyuma self-driving system
Lane following algorithm

Sheng-Wen Cheng , Po-Sheng Chen

August 2019

1 Introduction

Puyuma is a computer vision based self-driving system inspired by MIT duckietown project. By using Xenomai Linux real-time extension, we had created a miniature self-driving system which is suitable for doing real-time robotics research.

This report introduces the design of Puyuma self-driving system, including computer vision algorithm, data filtering and controller design.

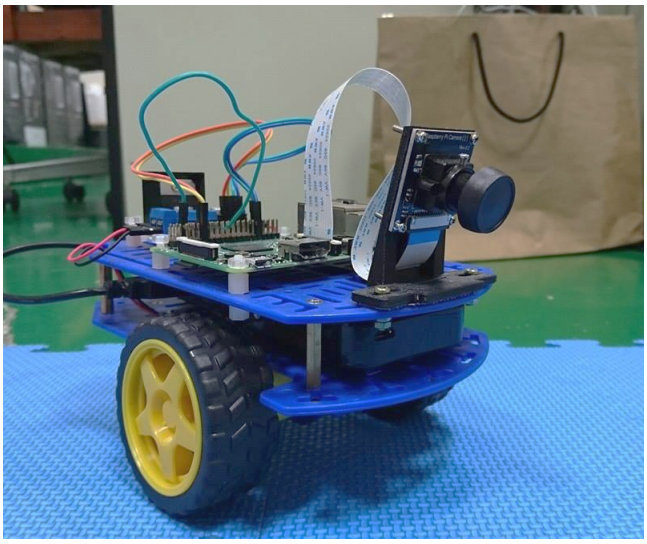

Figure 1: Puyuma self-driving system

2 Camera calibrations

Camera calubration is a early stage job before running computer vision algorithm. There are two set of parameters need to be found, intrinsic parameters and extrinsic parameters. They describe the relation between 2D image plane and 3D world.

2.1 Intrinsic calibration

Intrinsic matrix describes the 3D world to 2D image plane projection. The ideal linear camera model is consist of focal length, pixel size and principal point. Non-linear effects like lens distortion also exists but can not be modeled by linear equations, therefor it is usually solved by numerical methods.

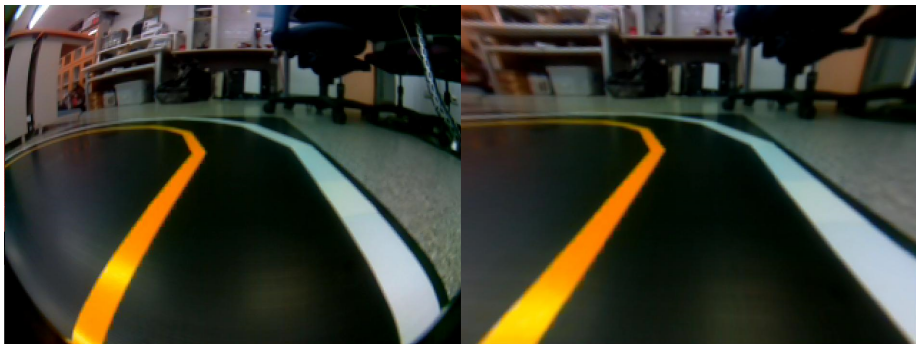

Figure 2: Camera undistortion and rectification

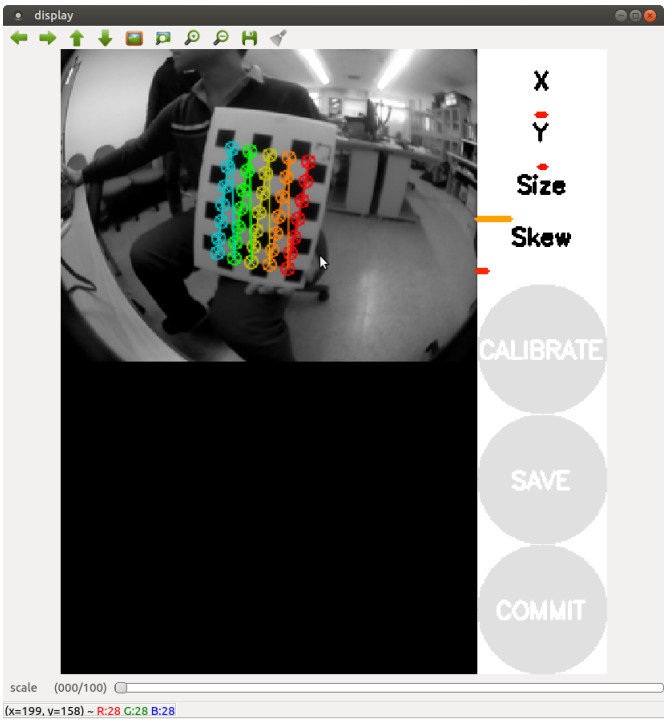

Figure 3: Intrinsic calibration

2.2 Extrinsic calibration

Extrinsic matrix describes the translation and rotation of camera with respect to the world frame. (i.e., angle and position of camera in 3D world.)

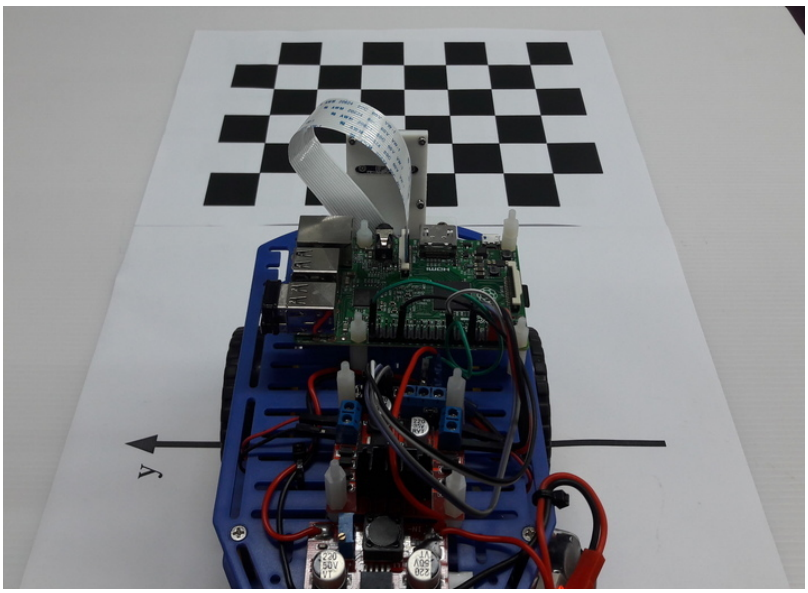

Figure 4: Extrinsic calibration

3 Lane pose estimation

3.1 System conventions

The coordination system of Puyuma self-driving system is defined as follow: x points to right, y points foward and z point upward.

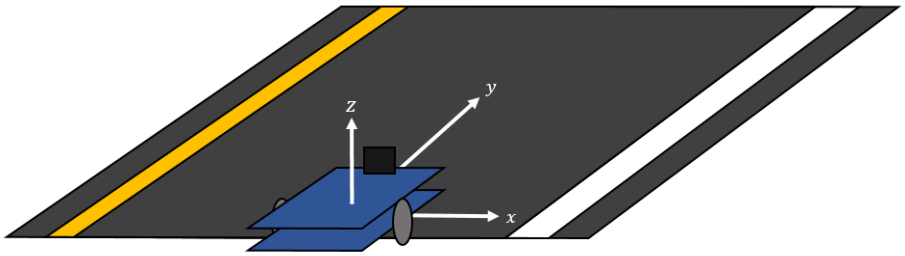

Figure 5: Lane coordination

We also assume the car always drives between yellow and white line. The yellow line is called inner segment and located to the left. The white line is called outer segment and located to the right.

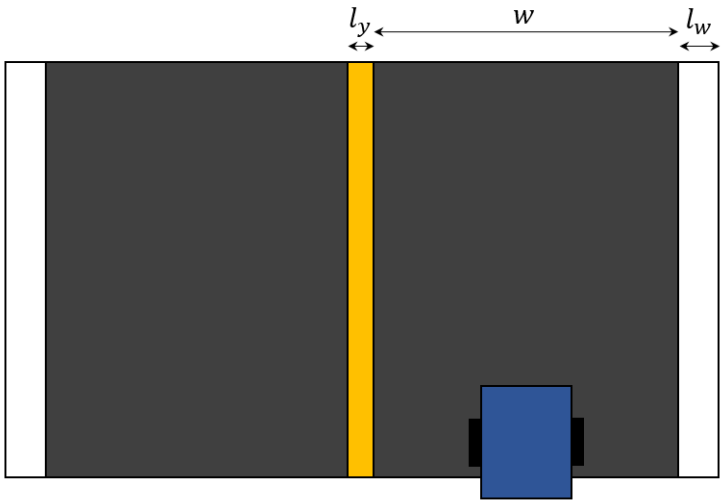

Figure 6: 2D view of lane

Now we can define the pose of the car: $\boldsymbol{d}$ is the lateral displacement and $\boldsymbol{\phi}$ is the orientation angle.

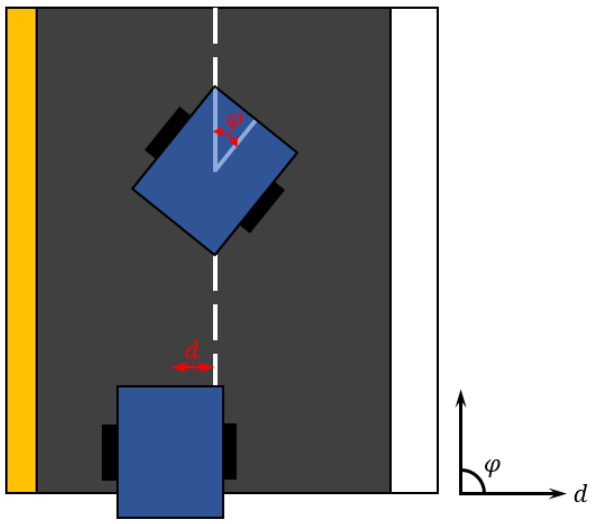

Figure 7: Lane pose

3.2 Segements detection

Before doing pose estimation, the lane segments have to be detected from image. Canny edge detection, HSV color thresholding and Hough transform are used.

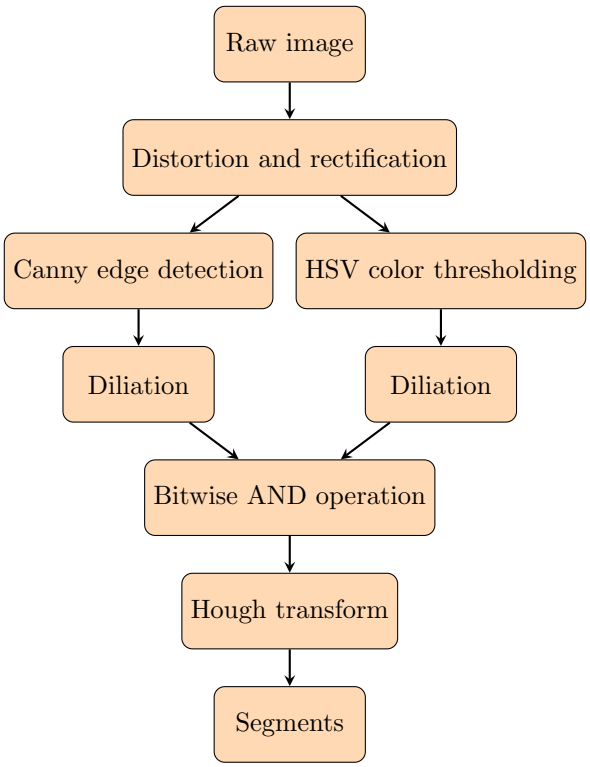

Figure 8: Lane detector workflow

The image is first transfromed from RGB color space into the HSV color space. Color thresholding method is then applied on HSV color space to find the interest color region like yellow or white.

Figure 9: Binary image of color thresholding result

After the color thresholding step, Canny edge detection is applied as a preprocessing step of Hoght transform. By using Hough transform, the line position (point-to-point) vector can be obtained for further computation.

Figure 10: Result of Canny edge detection

A bitwise AND operation of color thresholding image and Canny edge image is also used to to distinguish segements from different colors. The following figure shows the visualization result of lane segments and car pose: d and ϕ.

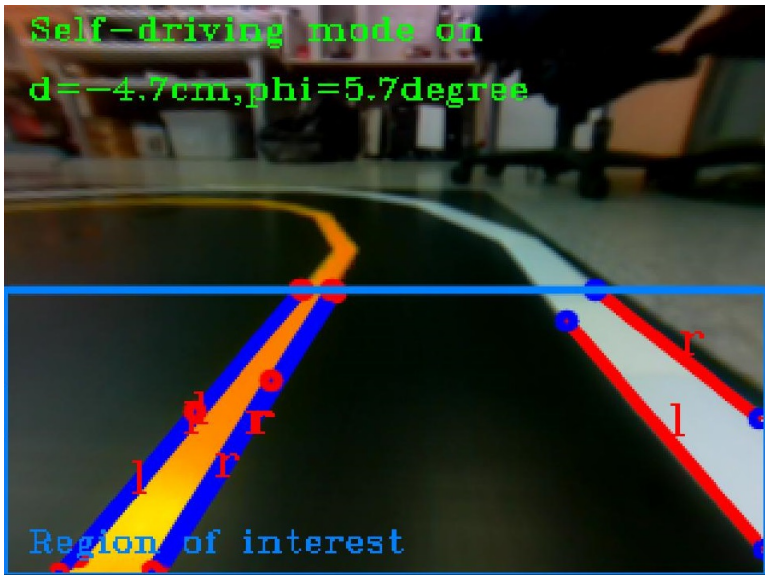

Figure 11: A visualization image of lane pose estimator result

3.3 Frame transformation

The camera is mounted in a certain distance from origin to the car. A camera-to-car frame coordinate transformation is needed for obtaining the right result.

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

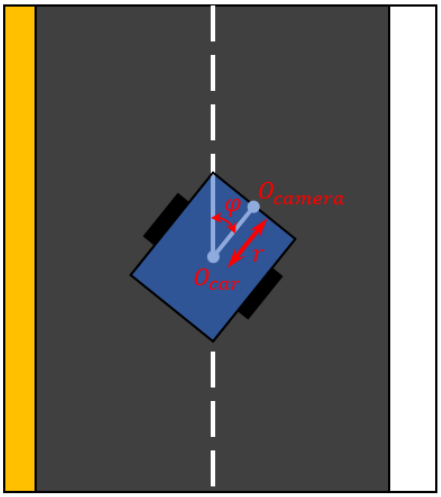

Figure 12: Frame transformation

3.4 Segment side recognition

Although the lane detector can help for finding segments, a segment side recognizing algorithm is also needed. We can determine the left or right side by testing several pixels in positive and negative direction on the lane segment normal vector direction. (using color thresholding image)

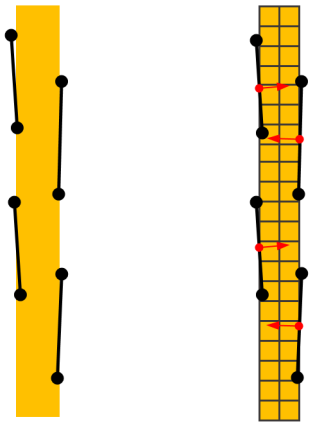

Figure 13: Segment side recognition

Algorithm 1: Segment side recognizing algorithm

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 already found the homography matrix which can help us map the image from camera's perspection to bird view perspection. This is usally called inverse perspective mapping. Inverse perspective mapping is applied before running the lane detector algorithm

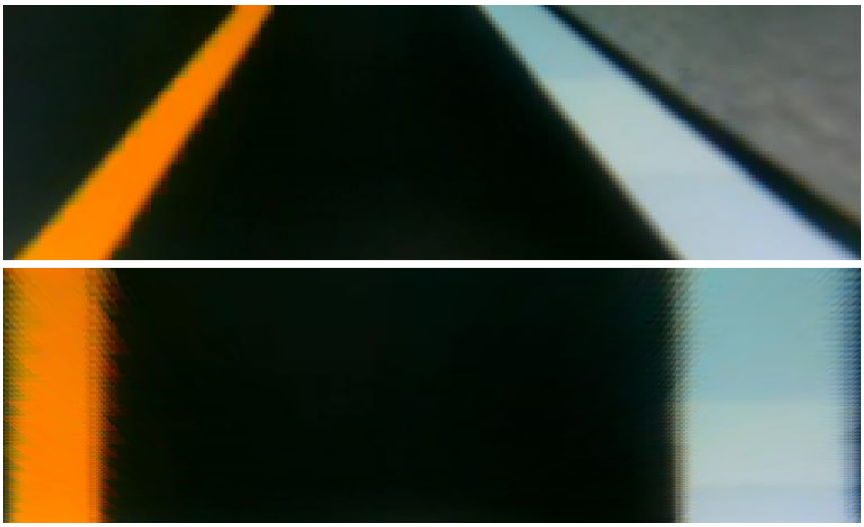

Figure 14: Ground projection

3.6 Pose estimation for single segment

Each segments detected by lane detector detects can be treated as a single pose data due to the lane geometry. These datas are put into a histogram filter for data filitering later. The process acts like a voting mechanism so is called a vote generating function.

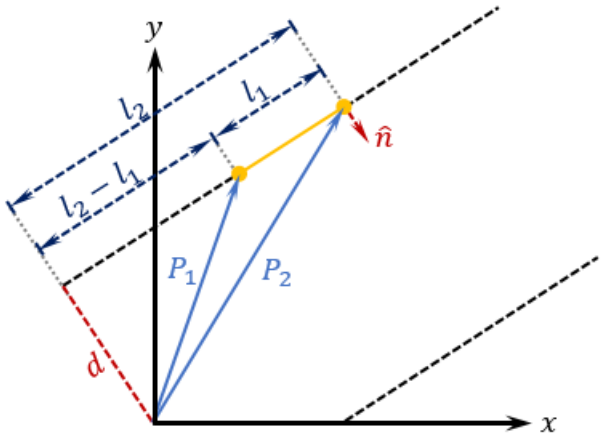

Figure 15: Lane geometry

Algorithm 2: Vote generating function

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

Segment poses generated in previous process are then collected into the histogram filter. The histogram filter is designed to find the **mode** of datas and eliminate extremes.

After the process, The **mean** of pose datas is calculated and becomes the estimation result.

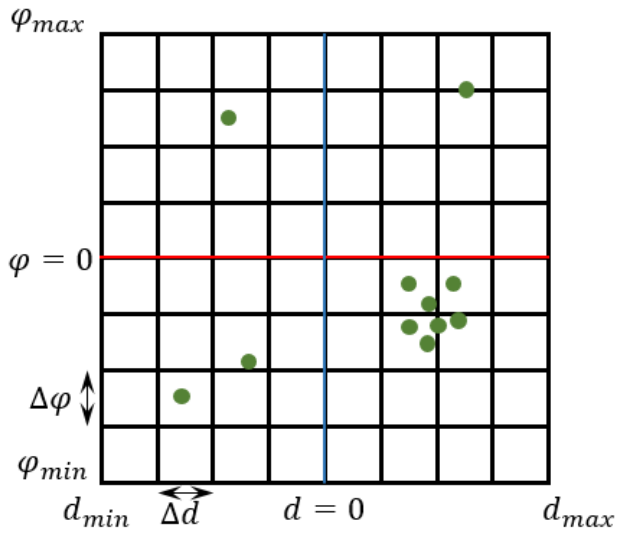

Figure 16: Histogram filter

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 design

4.1 Differential wheels

Puyuma is designed as a differential wheeled robot. There are two independent motors on the car. The car can turn by changing the speed rate between two motors. Hence it does not require any additional steering mechanism.

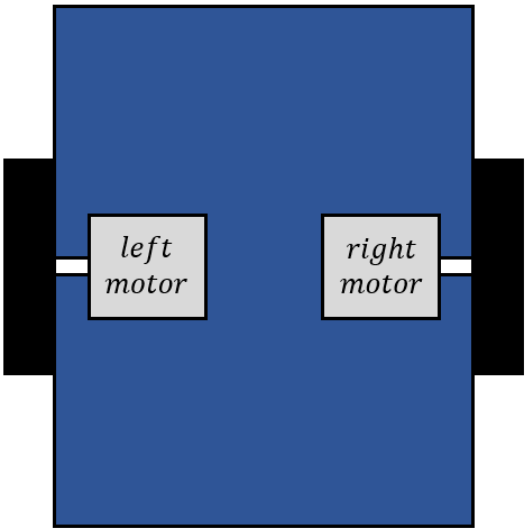

Figure 17: Differential wheeled robot

4.2 PID Controller

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

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 Puyuma is designed as a cascaded PID controller.

We designed phi controller as a lower level contoller for orientation stablizing. d controller can fix the lateral displacement error by changing phi controller's setpoint to turn left or right.

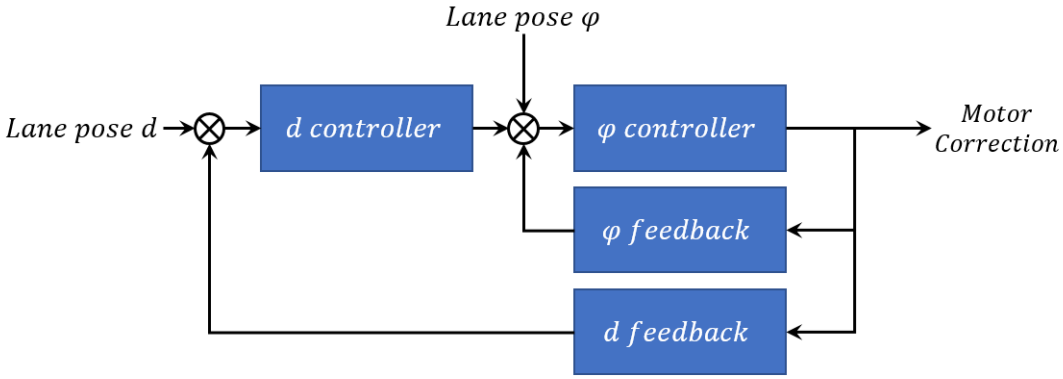

Figure 18: Control diagram

The wheel control signal (PWM) are simply the throttle value plus/minus 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