# DESIGN DRIVING CONTROLER

## Problem

The navigation package on ROS will take control the AGV by sending data include “linear velocity” and “angular velocity”. Controller had to modify the motion of mobile robot base on these data, which means bring it to reality.

Our robot have two wheels connect directly to 2 motor with two free-rotating wheel – a common type of mobile robot:

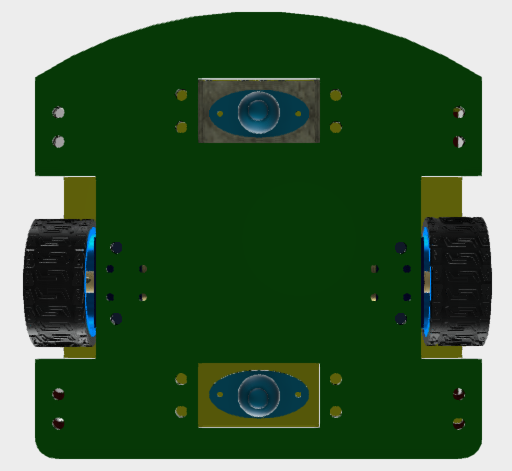
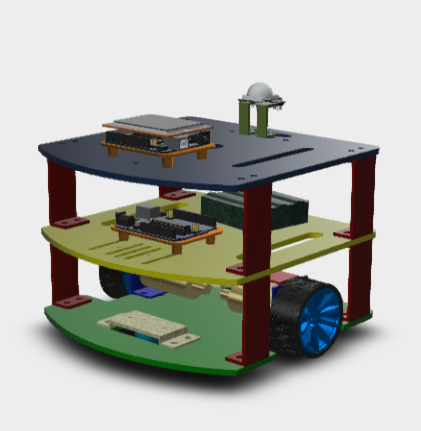


Figure: Two wheels and lead constructor

## System’s driver program

The popular method controlling this kind of robot is converting the requirement motivation vectors in to two linear velocity vectors of each wheels. This will bring the problem to simpler question: controlling velocity of motors. Here is a simple example of this method:



Figure: Example about controlling method

It is necessary building a dynamic model of the robot in order to have better imagination about motion of robot and wheels. It’s can be hard to deal with the displacement and velocities of two wheels robot. So that we prefer using a simpler model in which we can think of the mobile robot as having only one wheel with velocity (v) and specified heading (). Having the equations to translate between the unicycle model and our two wheels model.

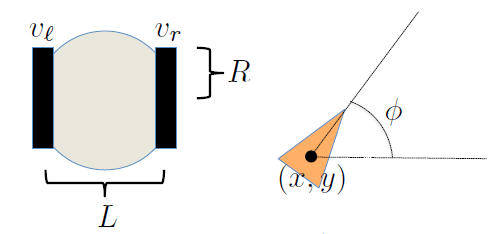


Figure: Unicycle model of mobile robot

Here:

From the unicycle model, we build the equations for translational and angular velocities. The forward velocity is calculated as average of wheels velocities:

The rotation velocity is the different of wheels velocities divide by the radius of rotation. Here, the wheels velocities are vl, vr, which is velocity of each wheels and radius is L that show the distance between two wheels.

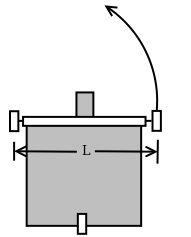


Figure: Rotation of mobile robot

Thus:

Finally, we generate the motion equation depend on those models:

Dynamic equations:

From eq. (1) and eq. (2):

(3)

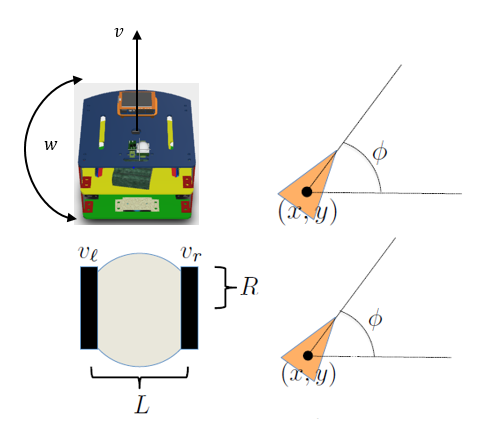


Figure: Robot’s dynamic model

We have the final equation:

Here:

### Wheels controlling

#### Convert from linear to rotation speed

Now, the requirements is taking control rotation speed and direction of each wheel follow the requirements from the previous method. From linear velocity values getting from equation (3) in previous part, we convert it to the rotation velocity by the equation:

(4)

Here:

Base on measurement and calibrations, we define the parameters of robot as:

Our block-diagram to control the mobile robot:

* Distance between two wheels : L = 207 (mm)
* Radius of wheels : R = 31 (mm)
* Encode resolution : Er = 440 (pulses/revolution)

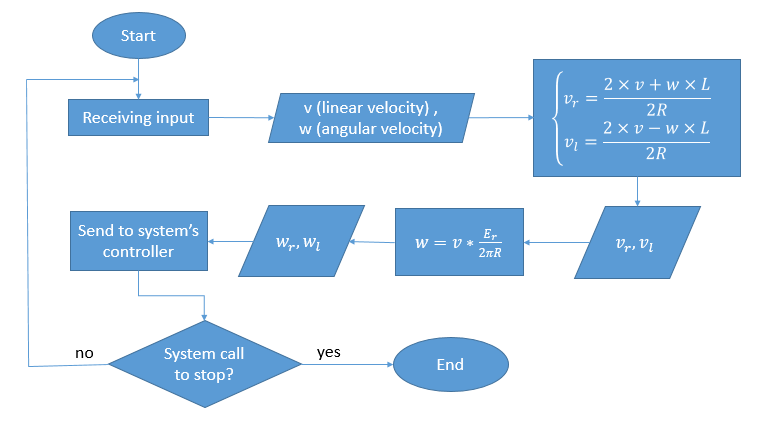


Figure: System’s driver block diagram

## System’s controller

In working environment, there are many things can affect to the working process of robot, such as friction between road and wheels, load, air resistant force, energy,… In order to keep robot performing as close as much with our expected, we have to add an extra controller. This controller will keeping receiving feedback signal from working statement, base on which it modify input signal to change the output performance. This make our controller become a “close-loop controller”, which have much more accuracy than “open-loop controller” before.

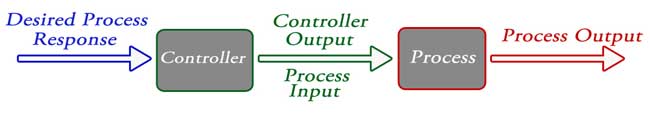


Figure: Basic open-loop system

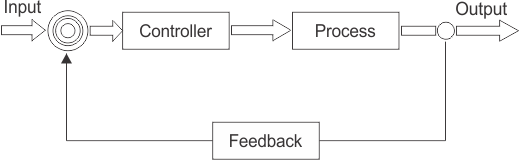


Figure: Basic closed-loop system

There are many different types of controller, but in this project, we will compare 2 common controllers to make the decision: PID controller and Fuzzy controller.

### PID controller

PID is a controller that is widely used in industrial control system. Its full-name is “proportional–integral–derivative controller”, this base on the three important parts that make the controller: Proportional part, Integral part and Derivative part.

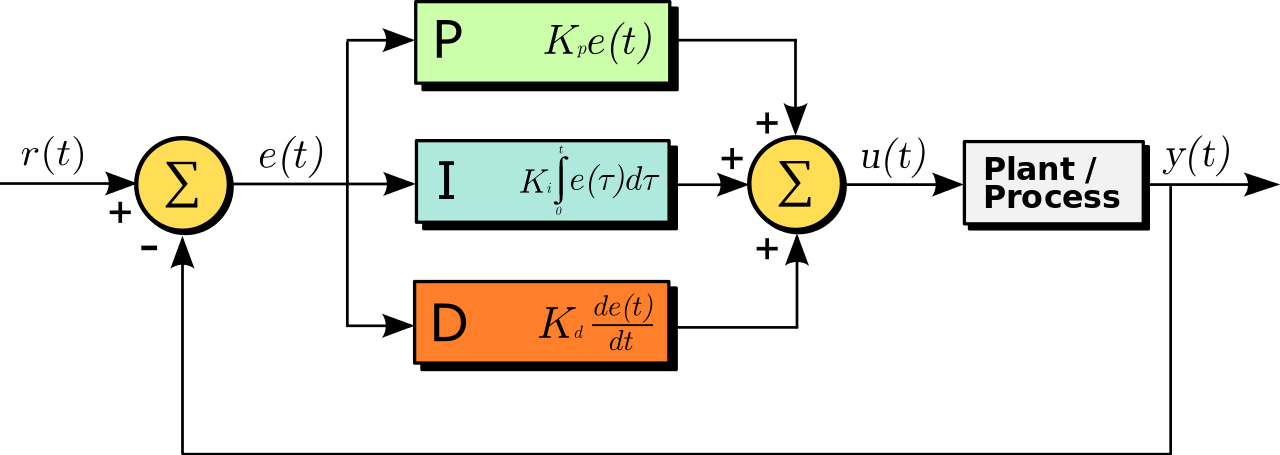


Figure: Basic PID controller’s blog diagram

#### Characteristic of PID

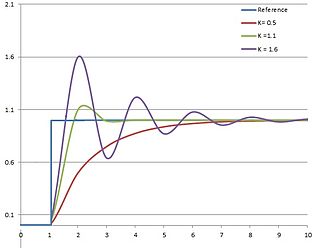
The mathematical form of PID controller:

In standard form, Ki and Kd are replace by Kp/Ti and Kp×Td:

Where:

Here are some roles of proportions, integrals, derivatives part:

+ Proportions part: The greater the value, the faster the response, so the greater the error, the larger the offset. If the scale is too high, the system will be unstable. Small size is due to small output response in large input error gases, and makes the controller less responsive, or slower response. If the gain is too low, the control action may be too small to respond to system noise.



*Figure: Proportions part of PID controller*

+ Integral part: The distribution of the integration phase (sometimes called reset) is proportional to both the error amplitude and the duration of the error. The sum of the instantaneous time error (two-digit integral) gives us the cumulative compensation that was previously calculated. Accumulation error is then multiplied by integral gain plus the output signal of the controller. The distribution amplitude of the integration phase over all adjustment effects is determined by the integral gain. The greater value of, the faster how the error is rejected. In turn, the greater the overshoot: any negative noise that is integrated throughout the transition response must be eliminated by positive integrals before reaching steady state.

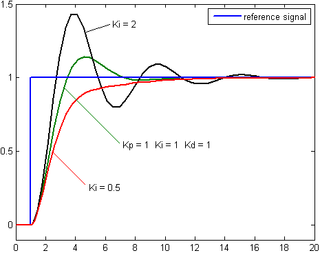
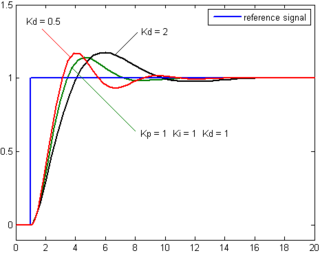


Figure: Integral part of PID controller

+ Derivative part: Derivative reduces the rate of change of the controller output and this feature is most notable for reaching the set point of the controller. Hence the derivative control is used to reduce the digital amplitude generated by the integral and to increase the stability of the composite controller. However, the derivative of a signal amplifies the noise and hence this stage is more sensitive to noise in the error and can cause the process to become unstable if the interference and derivative gain are large enough.



*Figure: Derivative part of PID controller*

#### PID turning method

*“Turning PID”* means designing the PID controller fit with your system’s requirements. PID parameters Ki, Kp, Kd can be turned by some process like “trial-and-error”, using Math Lab with already known transfer function, or “auto-tune process” in which our calculating depend on system’s reaction with control signal.

In this project, we choose “Relay-based auto tuning” as the turning method of our PID control base on its automation, quickly generating result and easily modifying. It also minimizes the possibility of operating the plant close to the stability limit.

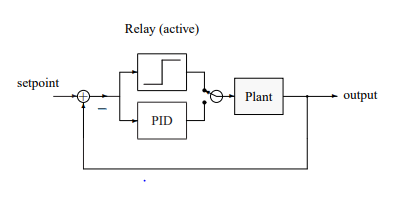


Figure: Blog diagram of relay auto-tune method

“Relay-based auto tuning” process’s key ideal is to temporarily swap a simple relay for the PID controller in the feedback loop. As it turns out, under relay feedback, most plants oscillate with a modest amplitude fortuitously at the critical frequency:

Step 1: Using a random constant input, u0 to determine the corresponding steady state output y0.

Step 2: Setting the setpoint as y0

Step 3: Running controller in relay mode, which have the input signal as:

Where:

The Input and Output signal form is like:

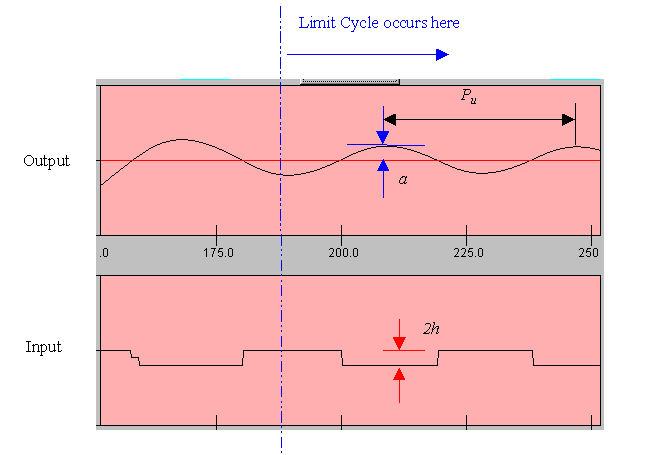


Figure: Relay-base method signal

Step 4: From the signals, define the amplitude “a” of reaction signal and ultimate period Pu. Then we can approximate the ultimate gain Ku as:

After having ultimate gain Kp and ultimate period Pu, applying them with Ziegler-Nichols tuning method to turning PID parameters:

|  |  |  |  |
| --- | --- | --- | --- |
| Specification | Kp | Ki | Kd |
| Original | 0.60×Ku | 2×Kp /Tu | Kp×Tu/8 |
| Little overshoot | 0.33×Ku | 2×Kp /Tu | Kp×Tu/3 |
| None overshoot | 0.2×Ku | 2×Kp /Tu | Kp×Tu/8 |

Because the little overshoot method is similar to none overshoot method, we just consider only two in three methods on this project: Original method and none overshoot method. After the tuning process, we get the PID parameters for each method as:

+ Original Ziegler-Nichols tuning method: Kp = 0.58; Ki = 16.58; Kd = 0.01

+ None overshoot Ziegler-Nichols tuning method: Kp = 0.26; Ki = 7.2; Kd = 0.01

Result of “Original Ziegler-Nichols tuning method” process:

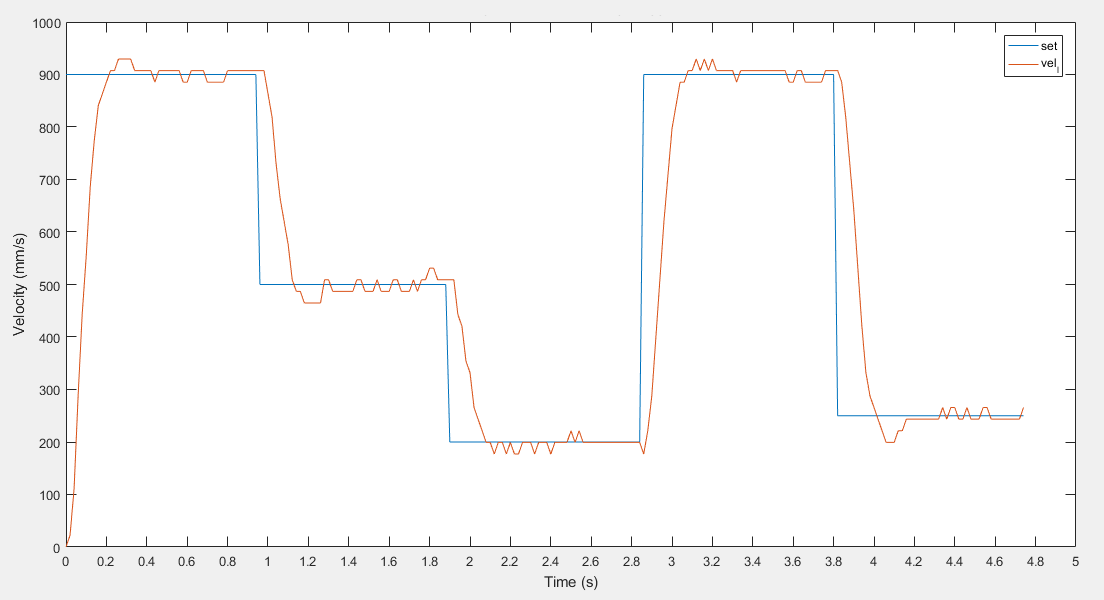


Figure: *Original Ziegler-Nichols: Left wheel’s output signal*

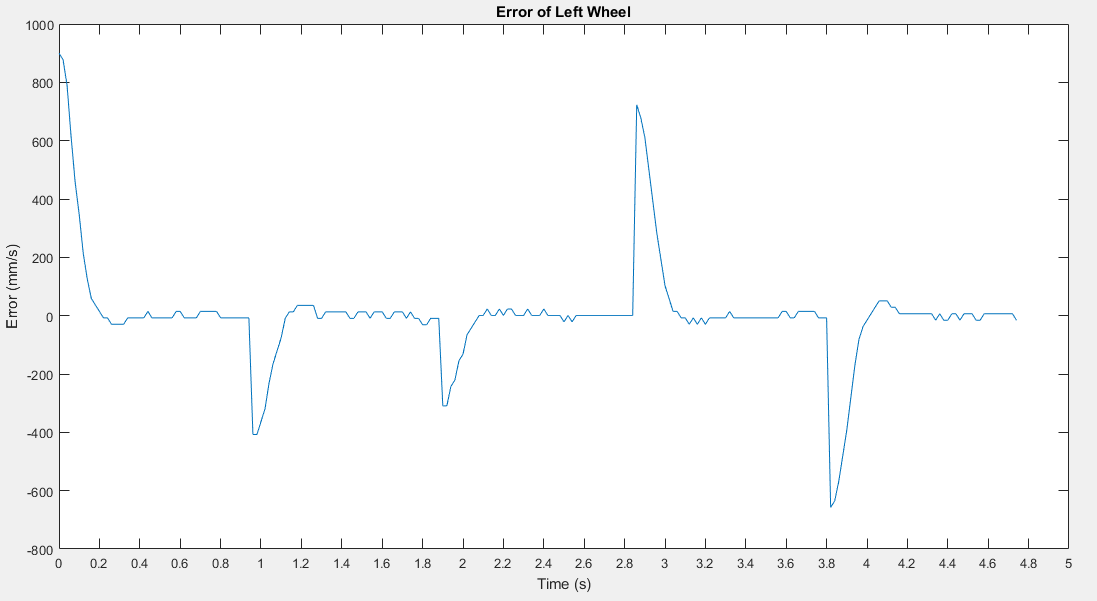


Figure: *Original Ziegler-Nichols: Left wheel’s error*

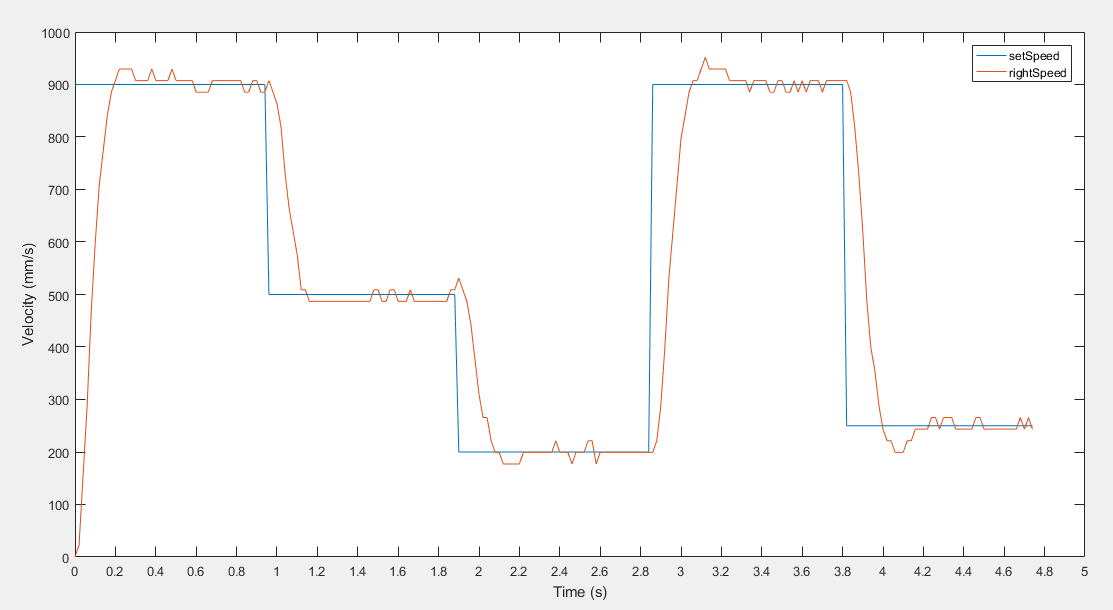


Figure: *Original Ziegler-Nichols: Right wheel’s output signal*

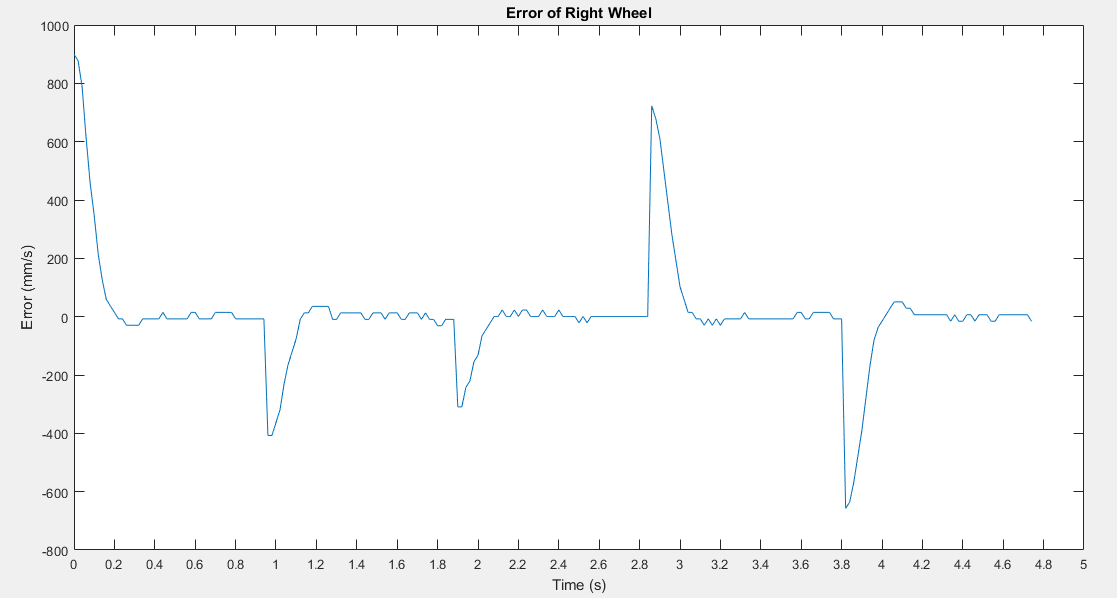
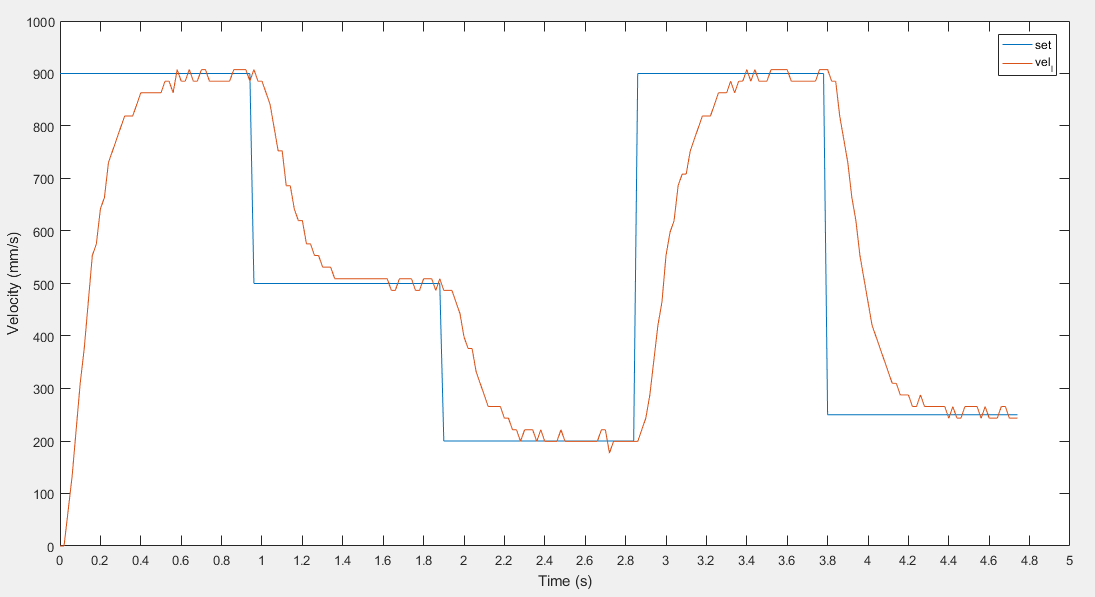
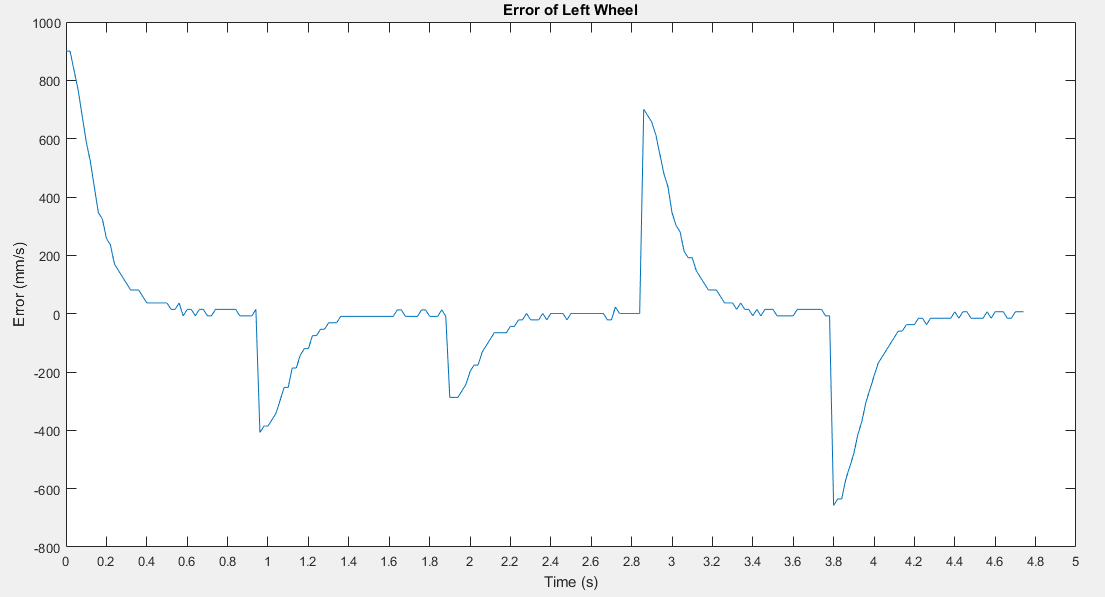


Figure: *Original Ziegler-Nichols: Left wheel’s error*

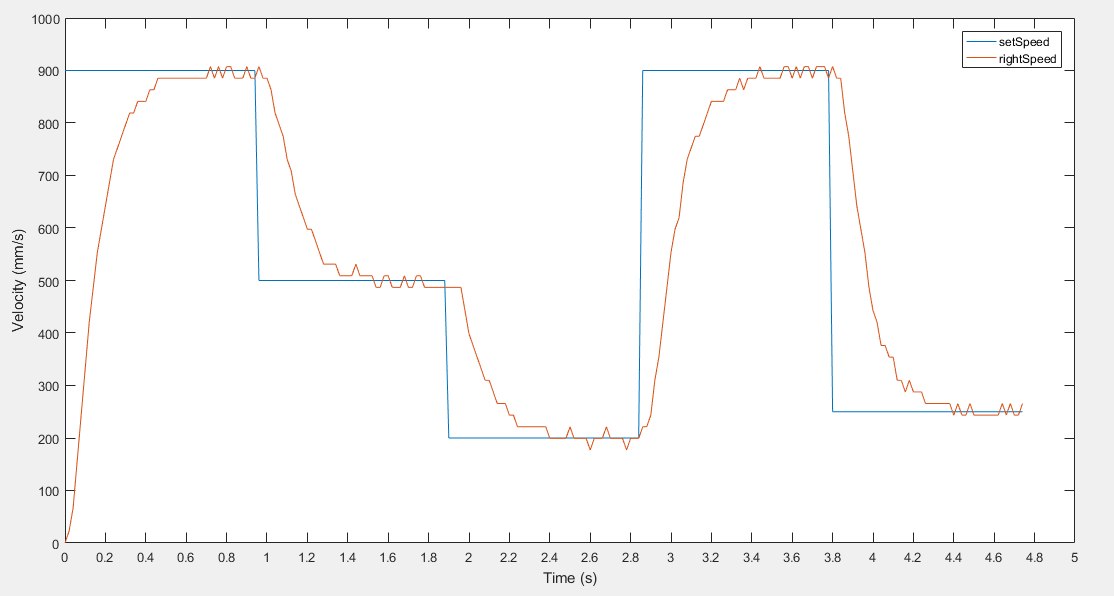
Conclusion: This type of controller really fit with the system. The overshoot is low (about 3%) when raising the set point, but when decreasing speed set point it has some over shoot (about 8-9%). Settling time is around 0.25 second, peek time is about 0.2 second.

Result of “None overshoot Ziegler-Nichols tuning method” process:

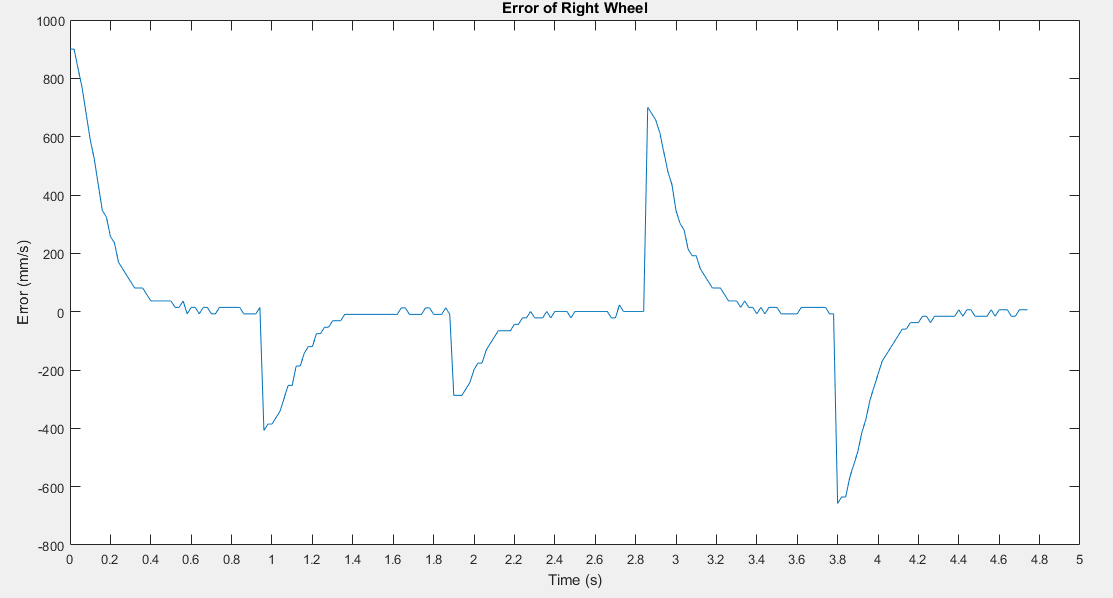
Figure*: None overshoot method: Left wheel’s output signal*



Figure*: None overshoot method: Left wheel’s error*



Figure*: None overshoot method: Right wheel’s output signal*



Figure*: None overshoot method: Right wheel’s error*

Conclusion: The overshoot signal is almost eliminated. But now the settling time and peak time are increased to more than 0.4 seconds, much slower than the *Original Ziegler-Nichols*.

### Fuzzy controller

The *Fuzzy controller* base on *Fuzzy logic,* which was first proposed by Lotfi A. Zadeh of the University of California at Berkeley. The word “*Fuzzy*” refers to the fact that the logic involved can deal with concepts that cannot be expressed as the "true" or "false" but rather as "partially true". The advantage of this logic is that the solution for the problem can be show in term that understandable with human operators, so that their experience can be applied in the design of controller. This makes it easier to mechanize tasks that are already successfully performed by engineer.

Fuzzy controller consist of an input stage, a processing stage, and an output stage. The input stage is group of inputs from sensors, switches, thumbwheels… The processing stage links data from input stage with its rules then generate a results for each case. Finally, the output stage converts the combined result back into a specific control output value.

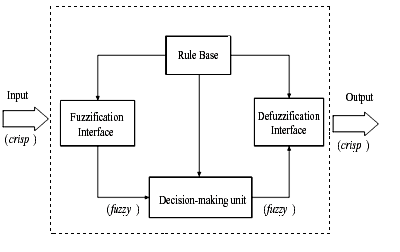


Figure: *Basic Configuration of Fuzzy logic system*

For designing Fuzzy logic control, we following these step:

1. Identification of variables: Identified the input, output and state variables
2. Fuzzy subset configuration: The universe of information is divided into number of fuzzy subsets and each subset is assigned a linguistic label.
3. Obtaining membership function: Obtain the membership function for each fuzzy subset that we get in previous step
4. Fuzzy rule base configuration: Create the fuzzy rules by assigning relationship between input stage and output stage.
5. Fuzzfication: Initiating fuzzification process.
6. Combining fuzzy outputs: By applying fuzzy approximate reasoning, locate the fuzzy output and merge them.
7. Initiating defuzzification: Initiate defuzzification process to form a crisp output.

Based on our experience about the mobile robot, we designed a fuzzy controller with one input and one output as showing bellow:

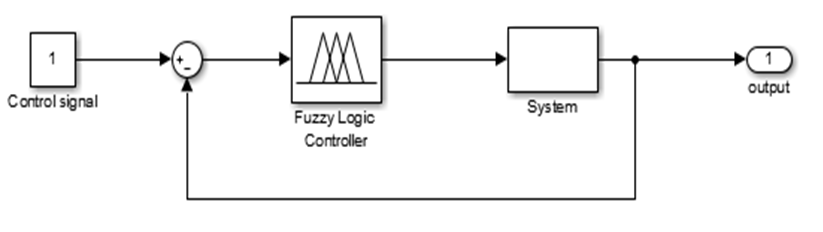


Figure: Fuzzy controller in motor system

In this Fuzzy controller, input signal is the error between controlling speed and the realize speed of motor. The output signal is extra value to add in the PWM control signal. From the feedback signal from encoder, the Fuzzy controller will modify the PWM signal to get the output reach the requirement signal.

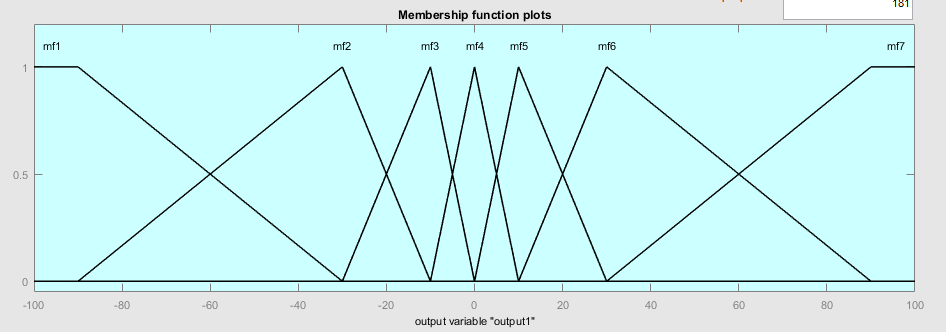


Figure: *Fuzzy controller’s output membership function*

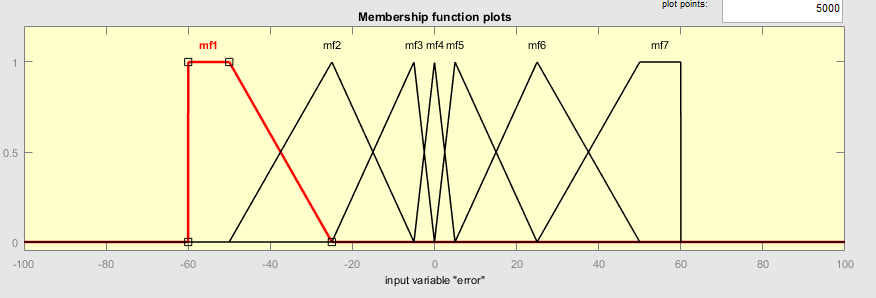


Figure: *Fuzzy controller’s input membership function*

Fuzzy rule base configuration:

* **If**  error is *negative large* **then** ouput is *add negative much*
* **If** error is *negative medium* **then** ouput is *add negative medium*
* **If** error is *negative small* **then** ouput is *add negative small*
* **If**  error is *zero* **then** ouput is *add zero*
* **If**  error is *positive small* **then** ouput is *add positive small*
* **If** error is *positive medium* **then** ouput is *add positive medium*
* **If** error is *positive large* **then** ouput is *add positive large*

Thus, the Fuzzy controller’s performance is like:

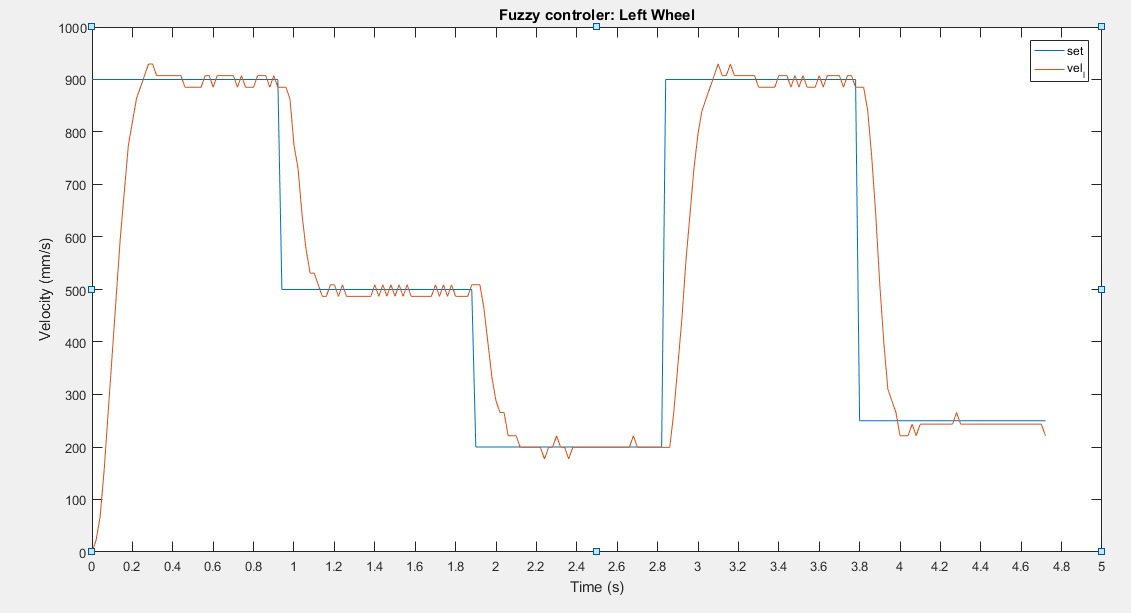


Figure: *Fuzzy controller: Left wheel’s speed*

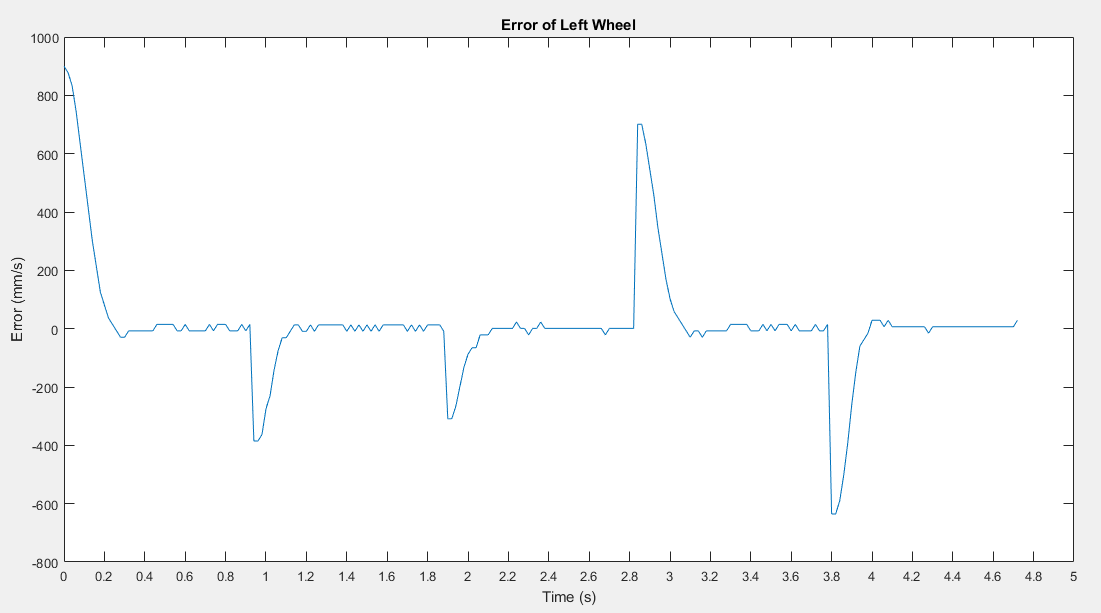


Figure: *Fuzzy controller: Left wheel’s error*

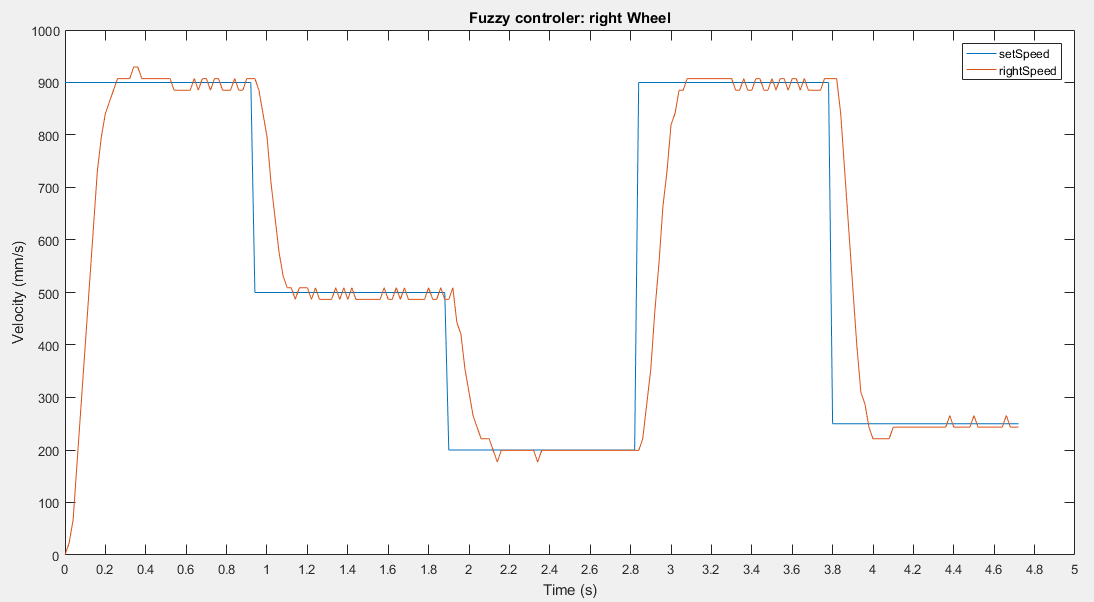


Figure: *Fuzzy controller: Right wheel’s speed*

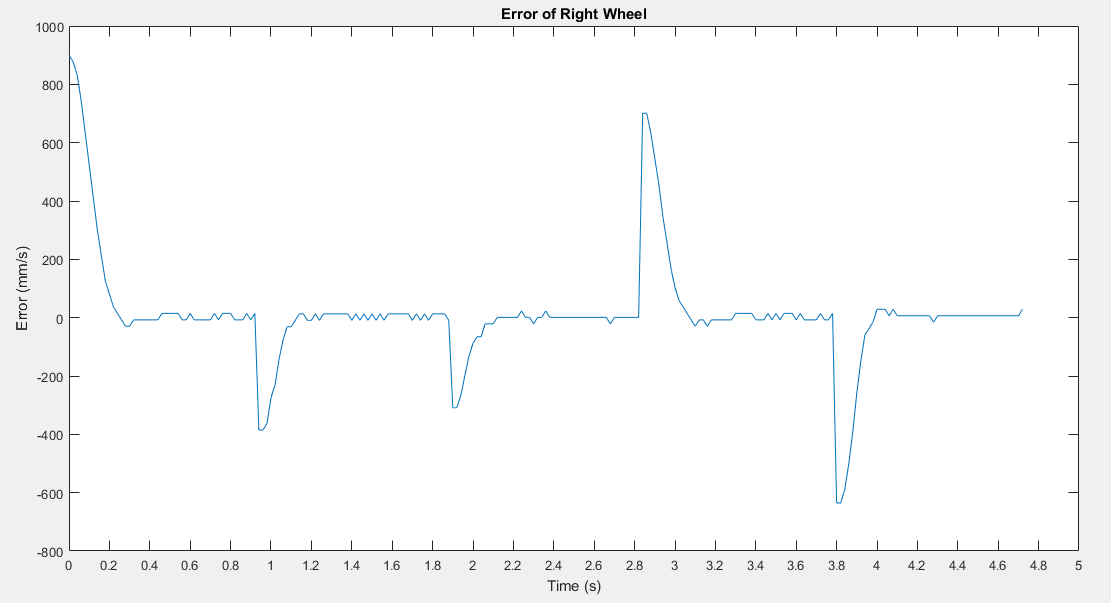


Figure: *Fuzzy controller: Right wheel’s error*

Conclusion: The Fuzzy controller have smaller overshoot than PID controller, especially in the decreasing speed process, also the stability is very good. However, the peak time and settling time (around 0.2 second) that is faster than PID’s ones.

### Self-tuning Fuzzy PID Controller

From the ideal about creating a PID self-tuning controller, we applying the Fuzzy logic into the PID Controller to design a Self-tuning Fuzzy PID Controller.

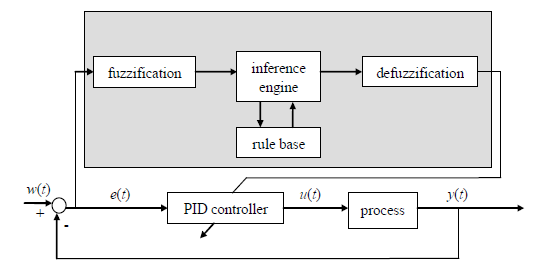


Figure: Self-tuning Fuzzy PID Controller

Basically, Our controller is a PID controller, but has the proportional parameter and integral parameter are continuously tuned by Fuzzy logic, based on feedback signal:

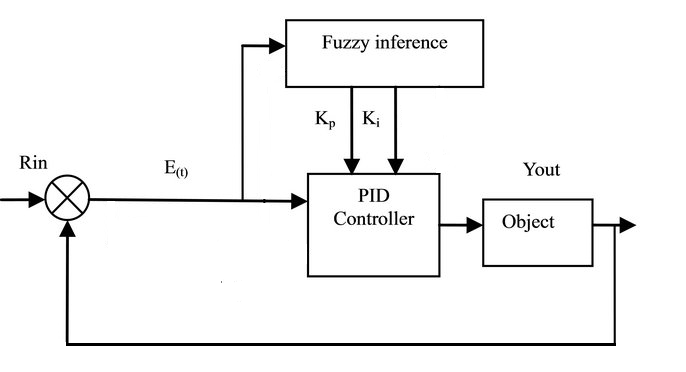


Figure: Self-tuning Fuzzy PID Controller

Based on the result of previous experience about Fuzzy and PID controller, we choose range of Fuzzy logic input and output as:

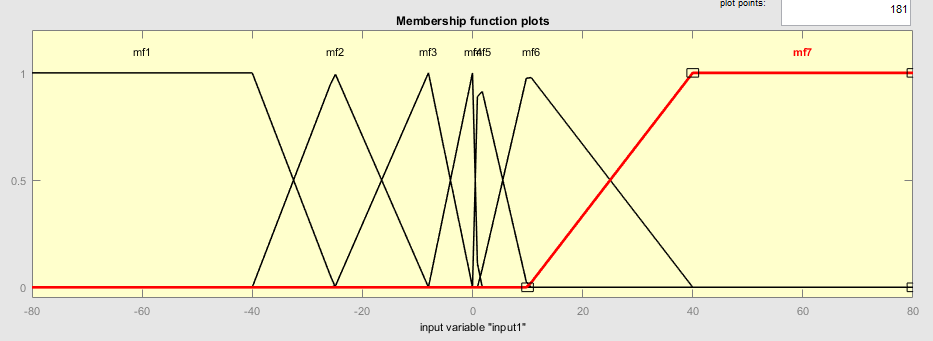


Figure: Fuzzy’s input: Error

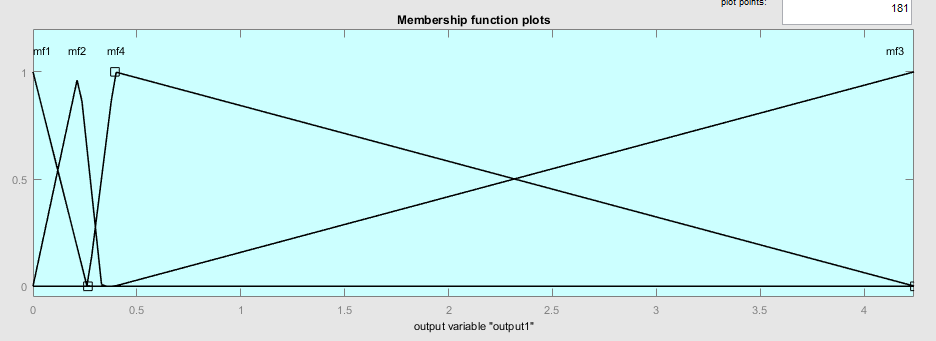
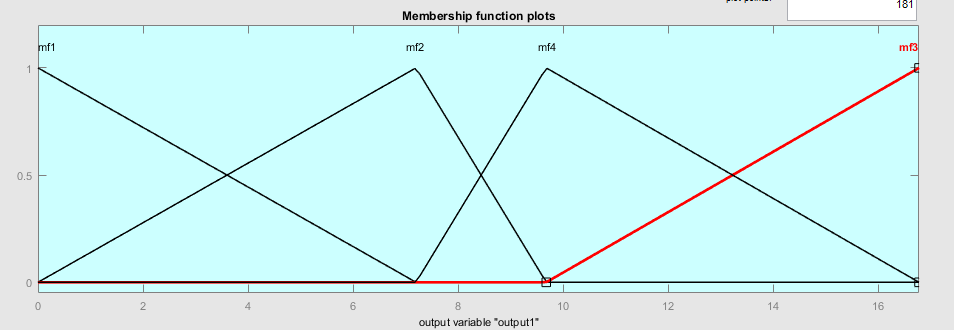
 

Figure: Fuzzy logic’s Output member function: Kp (left) and Ki (right)

Add rules:

* **If**  error is *negative large* **then** Kp is *medium* and Ki is *medium*
* **If** error is *negative medium* **then** Kp is *medium* and Ki is *medium*
* **If** error is *negative small* **then** Kp is *small* and Ki is *small*
* **If**  error is *zero* **then** Kp and Ki is *zero*
* **If**  error is *positive small* **then** Kp is *small* and Ki is *small*
* **If** error is *positive medium* **then** Kp is *medium* and Ki is *medium*
* **If** error is *positive large* **then** Kp is *large* and Ki is *large*

Applying to the mobile robot, we get the results:

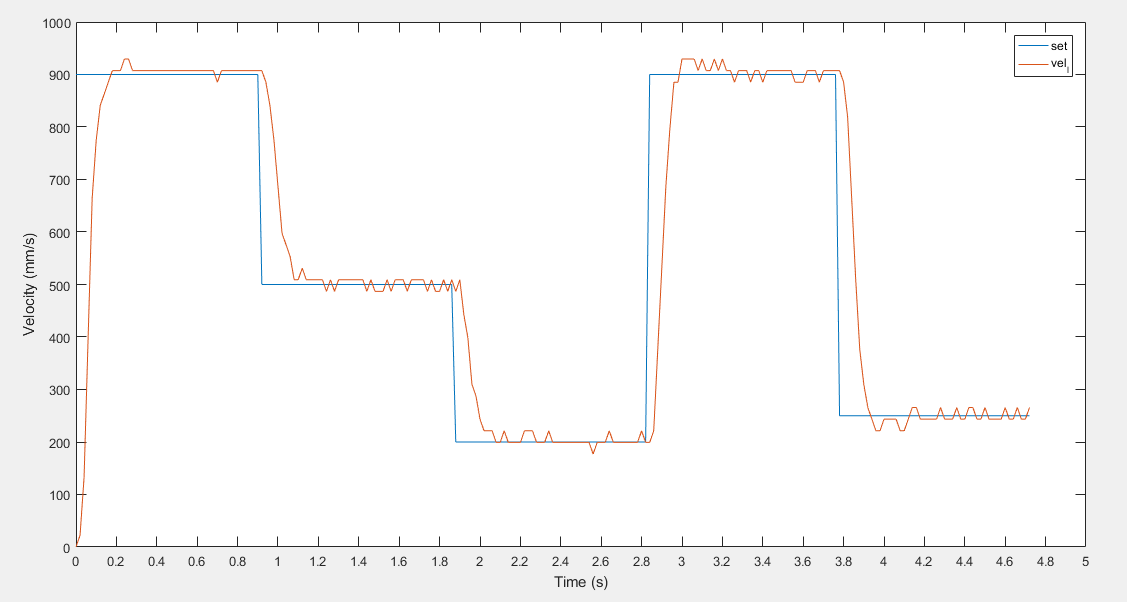


Figure: Self-tuning Fuzzy PID Controller: Left wheel’s speed

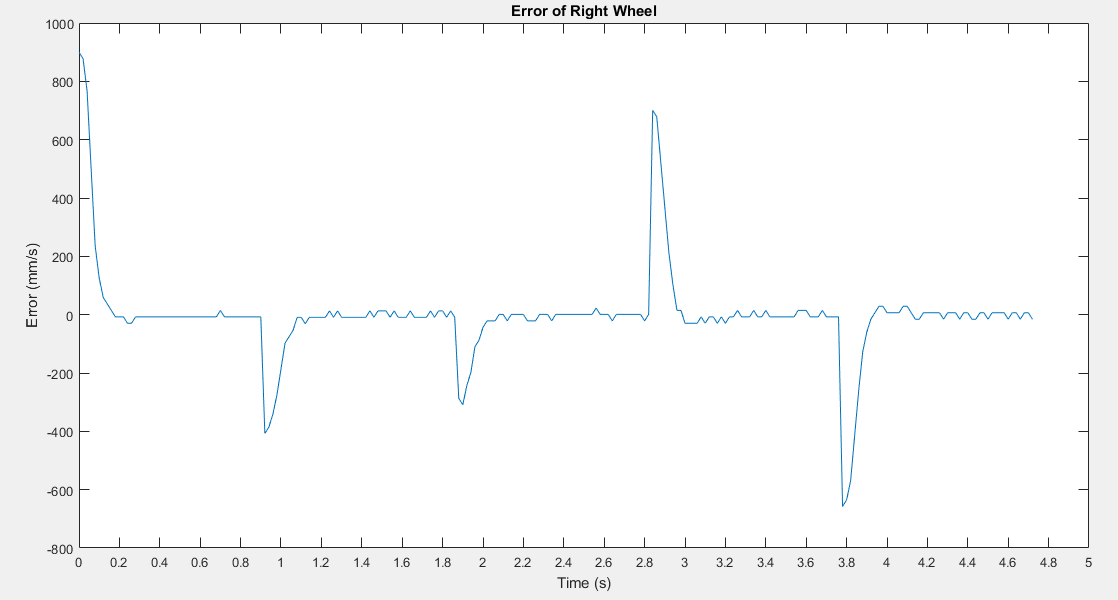


Figure: Self-tuning Fuzzy PID Controller: Left wheel’s error

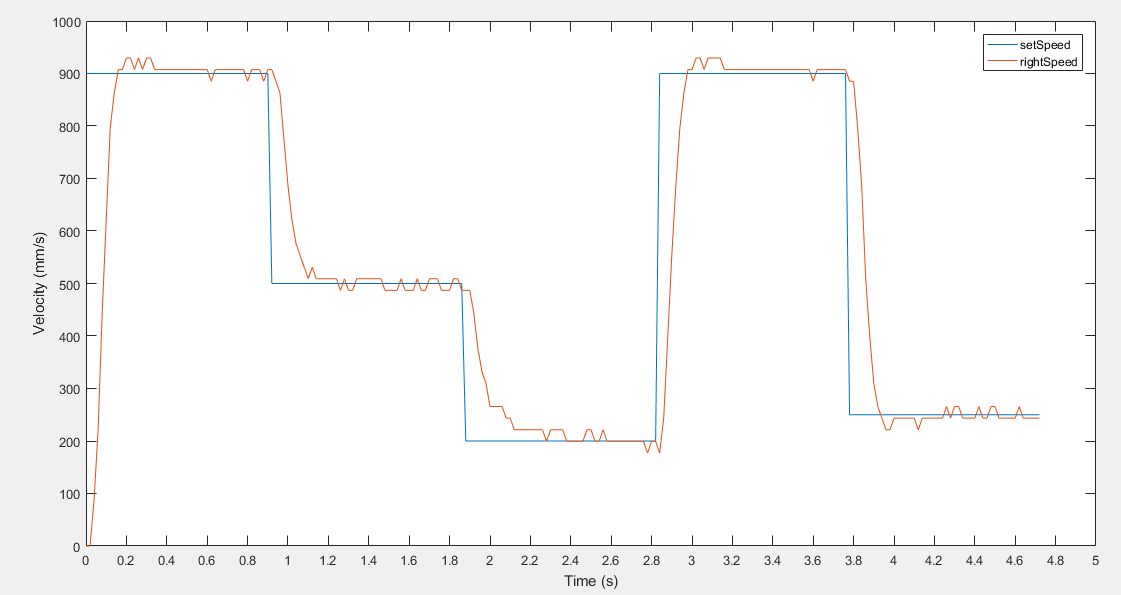


Figure: Self-tuning Fuzzy PID Controller: Right wheel’s speed

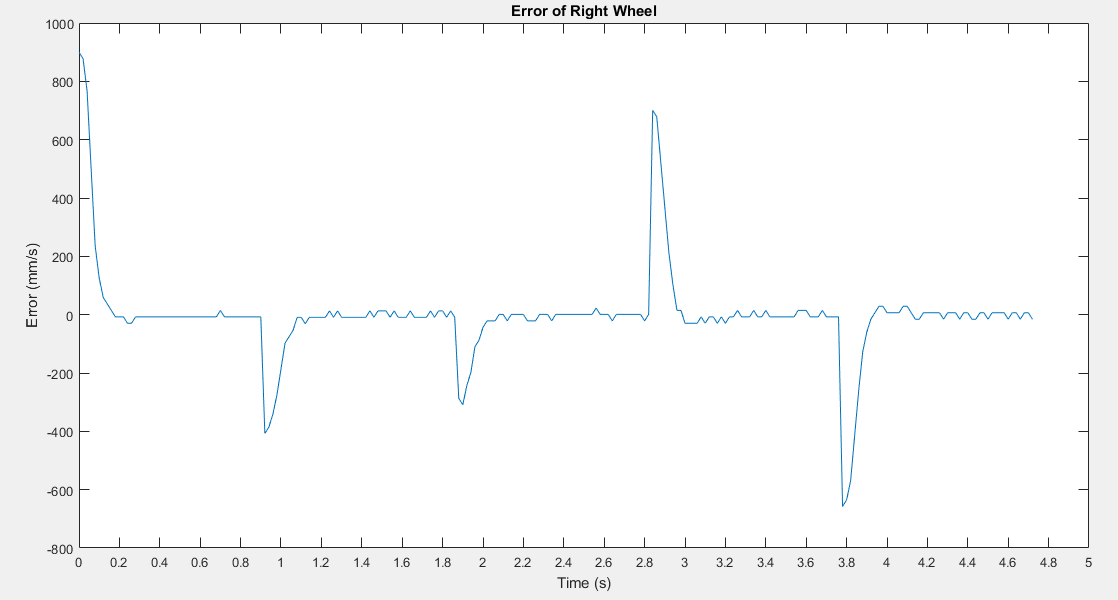


Figure: Self-tuning Fuzzy PID Controller: Right wheel’s error

Conclusion: There is a little overshoot signal (about 2%), the peak time and settling time are slightly decrease comparing with Fuzzy controller.

### Comparing controller

Finally, we will compare these controller to decide which is the best controller for our mobile robot.

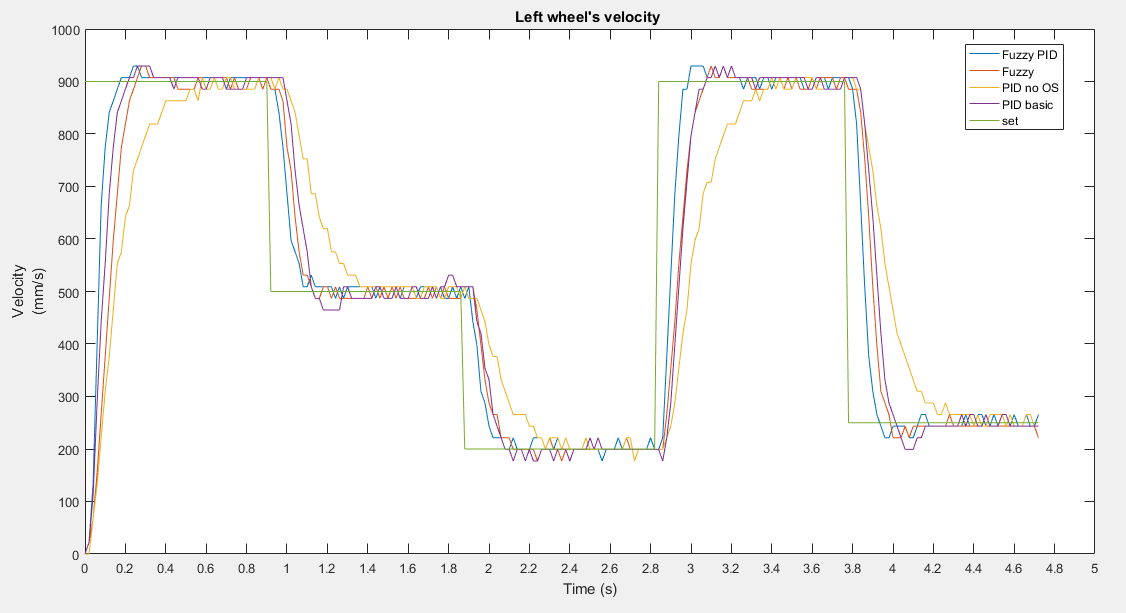


Figure: Left wheel’s velocity

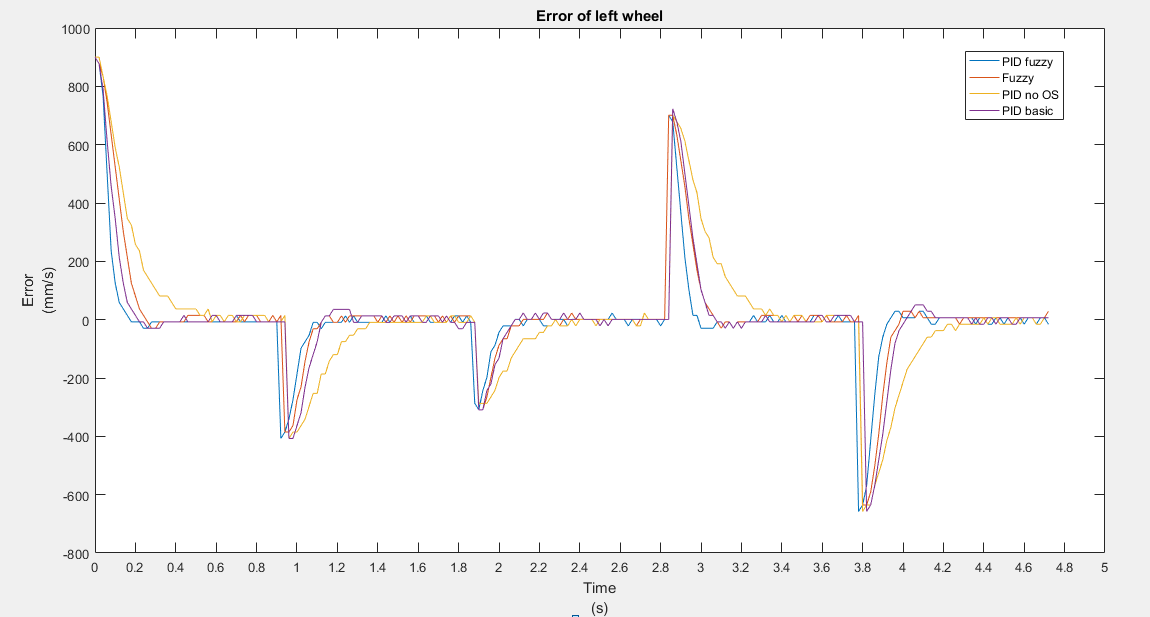


Figure: Left wheel’s error

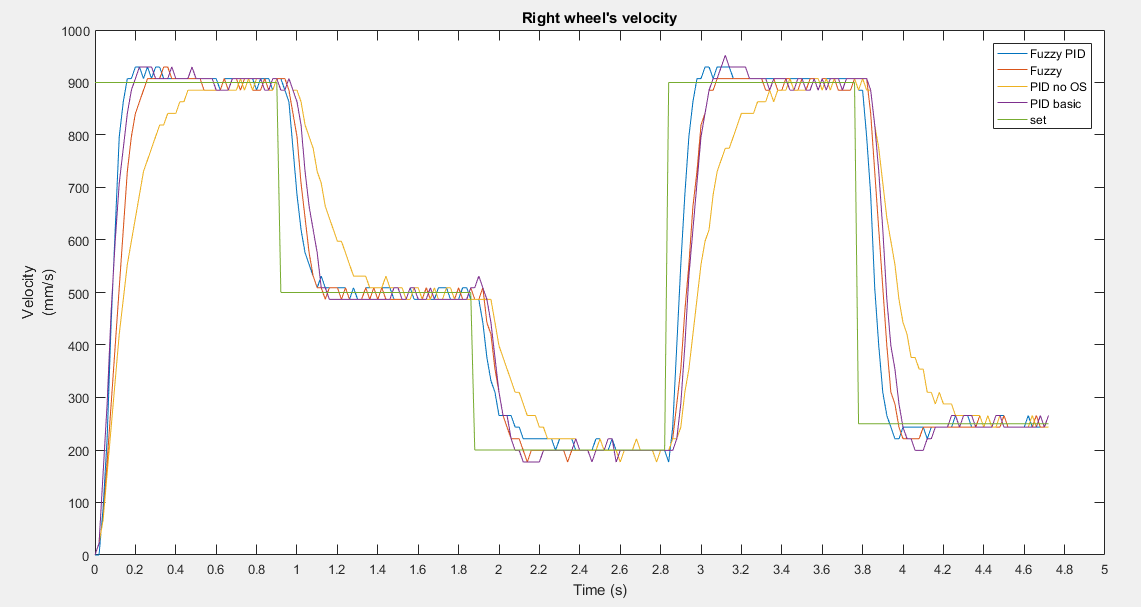


Figure: Right wheel’s velocity

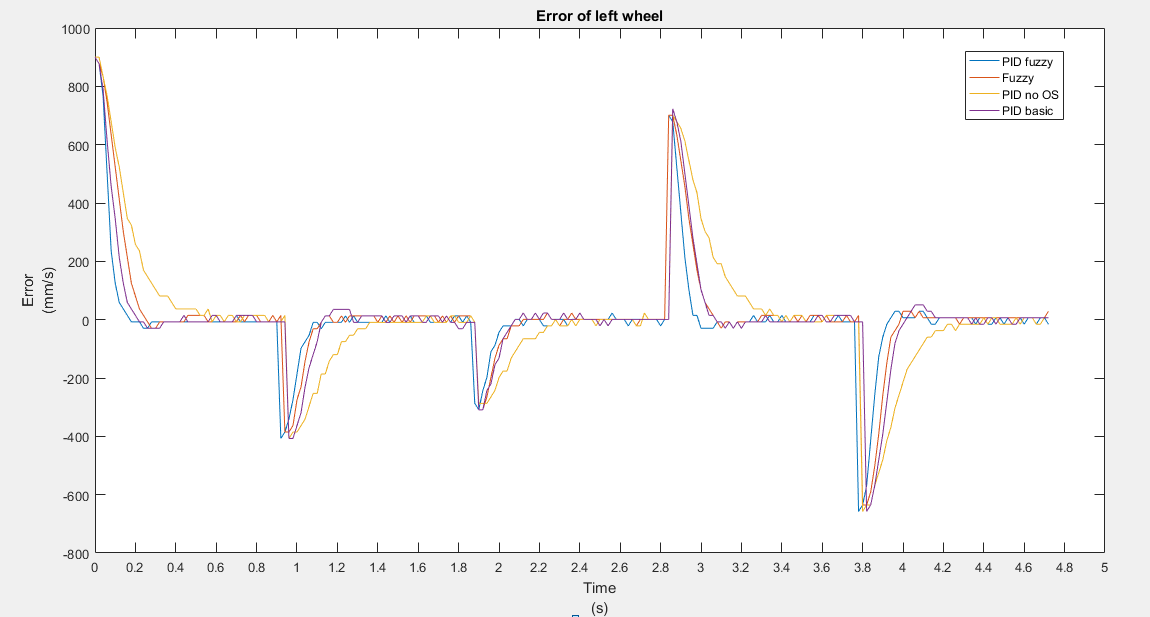


Figure: Right wheel’s error

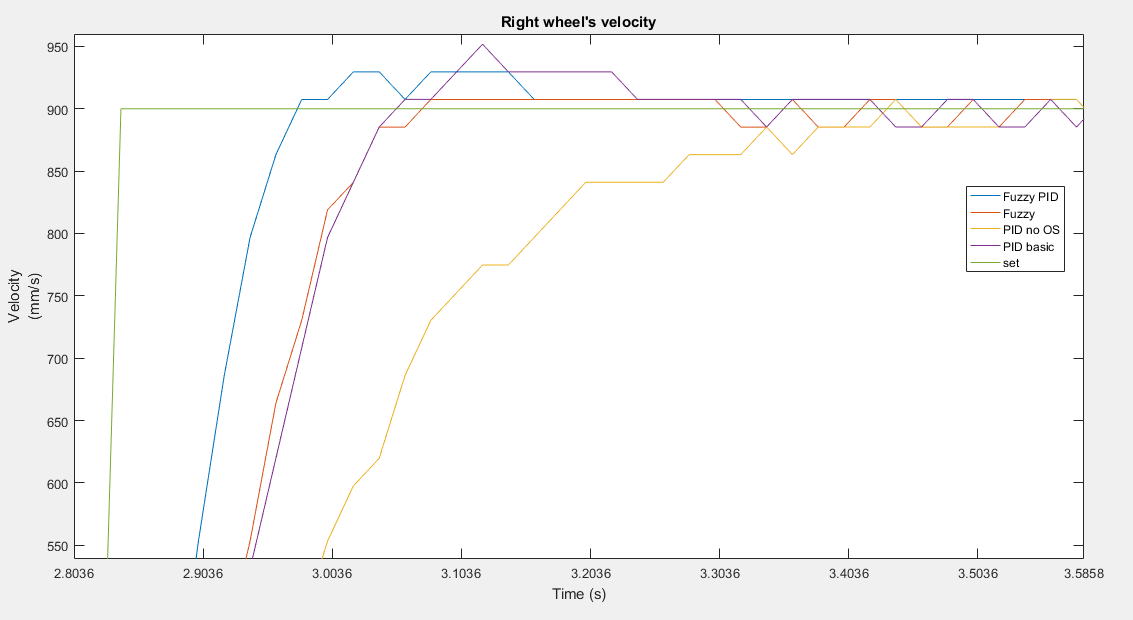


Figure: Comparing of left wheel’s speed

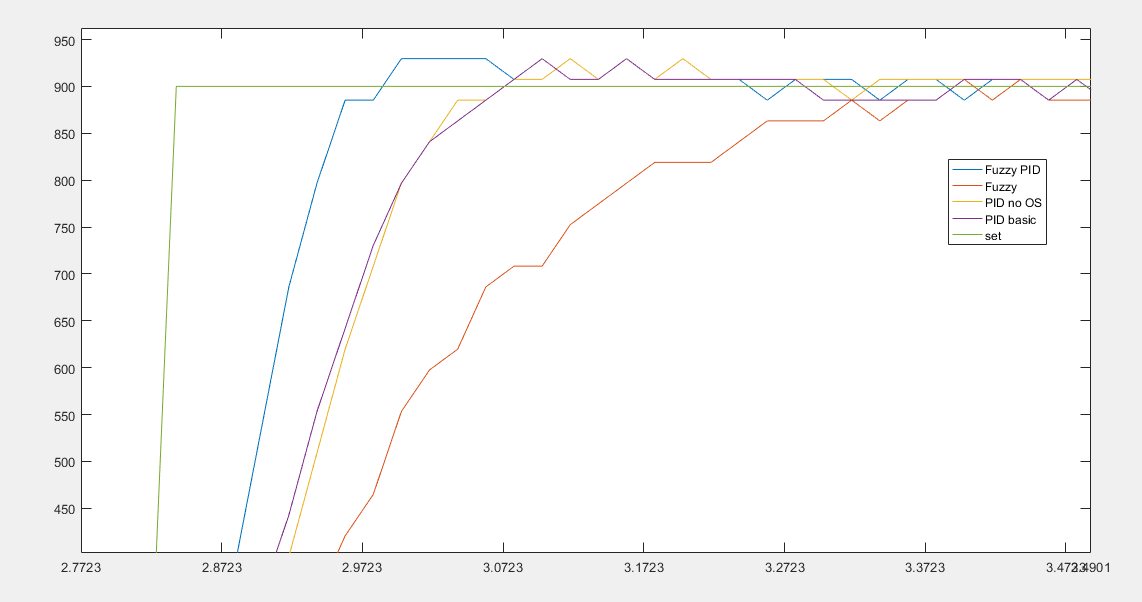


Figure: Comparing velocity of right wheel’s speed

Conclusion

Self-tuning Fuzzy PID Controller is the most suitable controller comparing with Fuzzy controller and PID controller. Following the comparison, it has the lower peak time and settling time than the other. Overshoot also less than PID, and stability are similar with Fuzzy controller. Also the wind-up problem is minimize, and the system working process is smoothly.

We decided to apply this controller to our mobile robot in this project.