Low Level Control System and Control GUI For High Speed Vehicle

Michael Y. Wu

A thesis submitted in partial fulfillment of the requirements of the degree of

Bachelor of Engineering (Mechatronics)

Australian Centre for Field Robotics(ACFR)

School of Aerospace, Mechanical and Mechatronic Engineering

The University of Sydney

November 2002

Declaration

I hereby declare that:

- I conducted the literature review for this thesis.
- I researched, designed and implemented the Hercules II low level control Graphic User Interfaces for both on-line and off-line applications.
- I designed and implemented a fine-tuning graphical user interface for steering, allowing automatic step, ramp and sinusoidal inputs.
- I designed a Keyboard Remote Control user interface to control HSV by a keyboard.
- I developed the accelerator and brakes' PID tuning and Velocity Controlling graphical user interfaces.
- I completed PID fine-tuning process for the steering.
- I collected and analyzed the best behavior of steering in steps, ramp and sinusoidal inputs under the ideal PID gains.
- With the assistance of Shahram, I collected and analyzed the response of the steering under different Step inputs in various velocities.
- I researched and developed a fuzzy logic controller for the Velocity control.
- I designed and developed a PID tuning user interface for path following control system.
- I researched and developed a PID control algorithm for path following control system with the assistances from Jose, Wang and Shahram.

- I designed and developed a Clone Control user interface allowing speed inputs into the path following system.
- With the assistance of Shahram, I upgraded PID control algorithm for steering, brake and accelerator.
- I assisted Chris and Shahram to correct power amplification for steering system.
- I have studied a concept design of a mechanical emergency system for the accelerator actuation system.
- With the assistance of Shahram, I designed a set of emergency procedure during the accidents.
- All results listed above have been demonstrated to my supervisor, Associated Professor Eduardo Nebot.

Michael Wu Yue

November 10, 2002

Abstract

Ever since the beginning of human's science, to create and build men-like machine always be the dream to achieve. For many years, the development of a fully autonomous vehicle comes into the focus of many potential applications in areas such as car manufacturers, defense forces or even space programs. Navigation is the main focus out of many researches and for 5 years, navigation of an autonomous vehicle has been undertaken in the heart of all robotic technologies in Australia, the Australian Centre for Field Robotics (ACFR) in the University of Sydney.

The Project is called HSV, abbreviation for High Speed Vehicle project. Since the first day, many undergraduate students have spent many hours in the creation of making it more perfect and more men-like.

"A journey of a thousand miles must begin with a single step."

Lao Tzu, Tao Te Ching

For the last few years, low level control has always be the problem for HSV to solve before moving up to a new step. Just like Lao Tzu taught us, the most important thing in front of you often to be the first thing but the most difficult thing to accomplish. Low Level Control is the basic control in the entire navigation research yet because it deals with many no-linear behaviors such as power steering, power brake or any speed-related system responses, it is still the myth waiting to be solved.

And for this year, I am the undergraduate student who takes part in the low level control of the HSV project.

In order to fully understand the system response and behavior in controlling the low level control, which involves the steering control and speed control, my task this year is to gather and analysis data of the responses of the steering in different types environments.

The efforts and approaches used to achieve this is to

- Develop a Graphical User Interface allowing fine tuning process of all three actuations.
- Gather information and analysis the results
- Seeking the relationship between any on linearity and the control of the vehicle
- Finding possible solutions towards the problem

Acknowledgements

To begin with, I would like to thank my supervisor, Professor Eduardo Nebot for giving me the chance to be a part of this interesting project and constantly pays attention to our progress in theses.

My special thanks to Jose Guivant, as he helped mainly on the software implementation in my thesis. He is a fun people to be with and always make us large.

My special thanks to Shahram Rezaei for his friendly companion when we worked together sometimes till the late evening. I found out he loved female songs very much.

My special thanks to Juan Nieto for helping me on understanding on the basic idea of path following simulation. He doesn't talk as much, but when he's talking…. "soccer".

My special thanks to Chris Misud for his time to come and help us on the electronic circus confusion. A question will rise in mine when anyone mentions his name, "Where is he?"

My special thanks to Trevor Fitzgibbons with his assistance in software and reversing the car for me constantly. He and Jeremy Randle are my savior when I am sick and tired of reversing the car back to garage.

My special thanks to Gurce Isikyildiz as he helped me on some software bugs when I were working. After work, he always be my best person to talk and complain to.

My thanks to other projects members especially Kim Jong-Hyuk, Alex Makarenko and Ralph Koch.

Content

Chapter 1

Introduction

The navigation controls of the Project HSV mainly consists of two large control systems, a high level control system and a low level control system. The high level controls the path of navigation while the low level control tends to reach the path with its best performance. Both of systems are equally important. For instance, it is like a ship where the captain on the ship tells stuff where to go and when to change the course(a high level controller), then the stuff who received the commands will change the velocity and steering angle as captain demanded(a low level controller). A ship can not last long in the deep ocean without a proper cooperation between captain and his staff and so as in the navigation control of HSV, the low level control should always cooperate with a high level control.

For the low level control, a better cooperation with high level control requires an improving performance in the response of both speed control and steering control. Different form the high level control, the difficulties of a low level control is that it deals with large amount of uncertainties or no-linear behaviors in both steering control system and speed control system.

It is set to be one of my major tasks to find these possible no-linear behaviors and responses in the steering control system. In addition, I will do in this year:

Steering Control System

- Develop a Graphical User Interface allowing fine tuning process of all three actuations.
- Online fine tuning of the steering control system
- Gather information and analysis the results
- Make correction or modification to PID control loops and steering amplifier
- Fine tuning the steering control system again

- Seeking the relationship between any no-linearity and the control of the vehicle
- Gather information and analysis the results
- Finding possible solutions towards the problem

Speed Control System

- Develop a Graphical User Interface for speed control
- Develop a Fuzzy logic controller for speed control
- Implement codes into Hyperkernel application

Path following Control System

- Research previous theses on path following
- Develop a Graphical User Interface for path following
- Develop a PID controlled path following algorithm
- Implement codes into Hercules software

Chapter 2

Low Level Control System

2.1 Introduction

The low level control is always to be one of the difficult tasks through out the whole project as it deals with many no-linear system responses under various conditions. A low level control system focuses on building a better responded system with lower overshoots and lesser steady-state-errors. It mainly consists of two systems, a steering control system and a speed control system.

The steering control system focuses on the position control of the steering wheel by feed-back PID-controlled DC motor. The speed control system is in controls on both actuators in the responses of brake and throttle systems, to control the speed of the vehicle eventually.

In order to achieve better responses of the systems, fine-tuning of the PID controllers on all actuators are necessary. In hence they produce an accurate and fast response of the vehicle.

Some of the no-linear behaviors occurred in the low level control system such as speed related steering responses or motor characteristic related steering responses are needed to be concluded and analyzed. A low level control graphical user interface is necessary for fine tuning PID controllers and gathering data for analysis proposes.

2.2 Steering Control System

The steering control system focuses on the position control of the steering wheel. The input of the control system is the desire position of the steering in counts and output of it will be the actual position of the steering in counts.

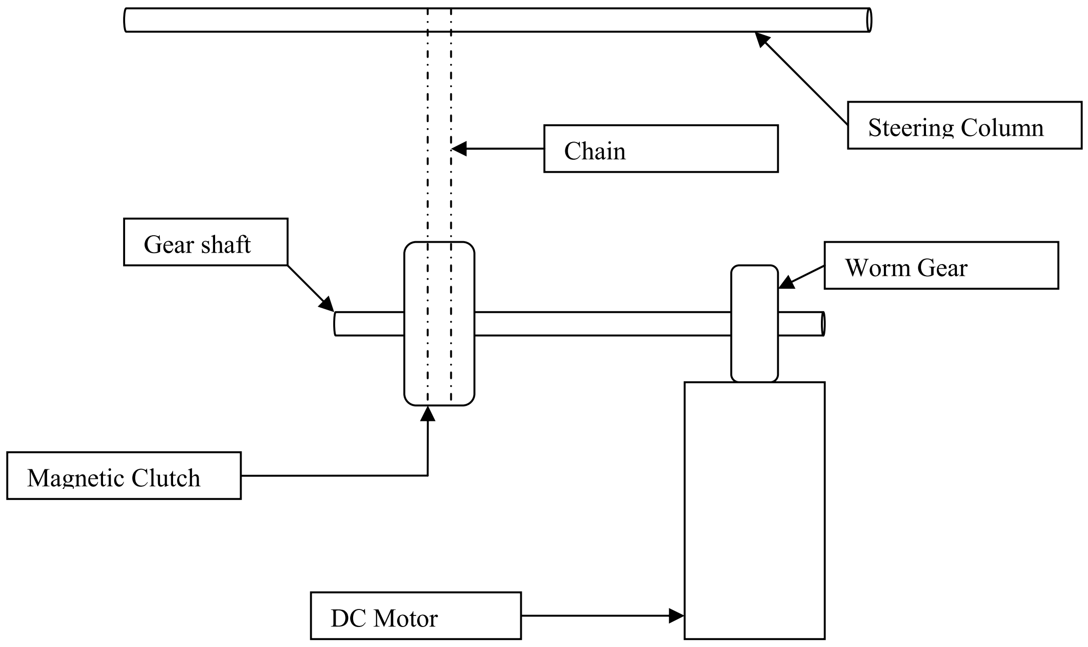
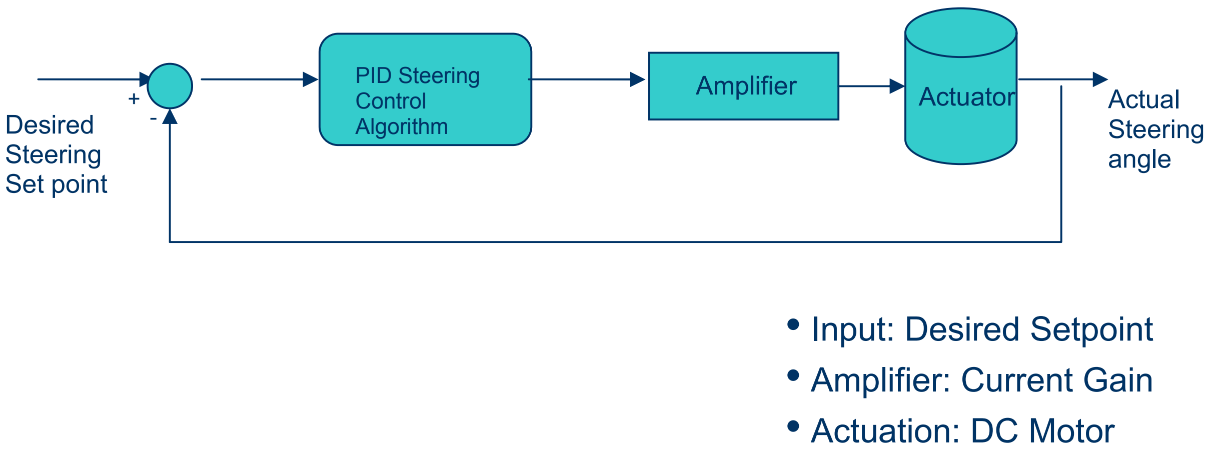


Figure 2.1: Hardware map of steering Control

As shown in the figure 2.1 above, The Hardware components of the steering control system are mainly consisting of a motor, a magnetic clutch, gears and shafts, a steering column links to the steering wheel and a chain which links the motor to the column.

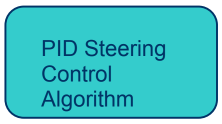
By controlling the current flows into the DC motor, we can control the torque generated by the motor and hence control the steering wheel's position.

The rest of the steering control system involves in the software and the electronic devices used. It consists of a PID Control Algorithm, a Current-gain Amplifier, a $\underline{+}$24 volt DC motor and a LVDT feedback loop(Figure 2.2).



Desired Steering Set point
+
-
PID Steering Control Algorithm
Amplifier
Actuator
Actual Steering angle
• Input: Desired Setpoint
• Amplifier: Current Gain
• Actuation: DC Motor

Figure 2.2: Steering Control System



$$m = k_p \varepsilon + k_i \int \varepsilon dt + k_d \frac{d\varepsilon}{dt} \qquad (1)$$

$$\Delta m = k_p(\varepsilon - \varepsilon_{-1}) + k_i \varepsilon + k_d(\varepsilon - \varepsilon_{-1} + \varepsilon_{-2}) \qquad (2)$$

$$m = m_{-1} + \Delta m \qquad (3)$$

Figure 2.3: Steering Control Algorithm

The PID Control Algorithm(Figure 2.3) is used to calculate the power flows into the amplifier in the range of 0 to 5 vols. Distinguishing from the steering control system, the input to the PID steering control algorithm is the error between desired and actual position and the output will the power calculated.

The power output from the PID Control Algorithm is in terms of pervious two states and current state of the steering position error. It can be tuned in respect of different PID control gains, Kp, Ki and Kd to achieve a better steering position-control response.

The integral term of error, $\int \varepsilon dt$ needs to be set down to zero each time a new desired position(set-point) of the steering is inputted into the algorithm. It is to prevent the happen of integral team related steering responses(See Chapter 4 Section 5 for detail). As indicated in David's thesis 1997, Steering's LVDT reading(Figure 2.4) from 2028 to 3093 is representing vehicle heading from -35 deg to -1 deg(left); LVDT reading from 2026 to 492 is representing vehicle heading from 1 deg to 35 deg(right).

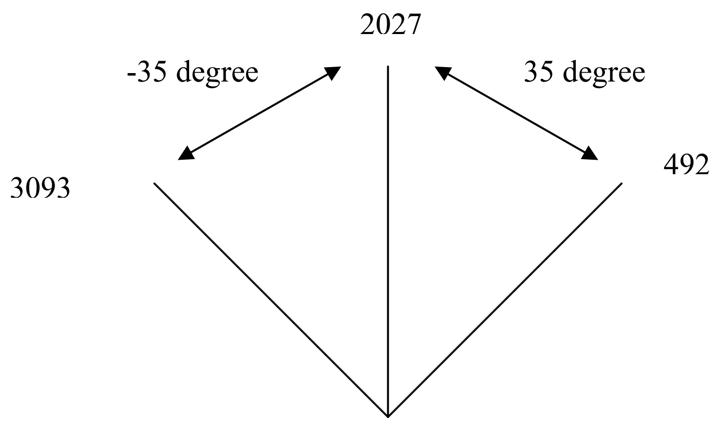


Figure 2.4: LVDT Reading Configuration

By controlling the torque to the motor generated by the power from the PID control algorithm and the amplifier, we can accurately control the position of the steering wheel in a range of 492 to 3093 counts. The LVDT reading is then feed-backed to produce next power output from PID control algorithm.

The Steering control algorithm is written down in Hyper Kernel Application previously. A PID structure is used to store PID data information in terms of set-point, gains, errors and output power to an amplifier. Three pointers are created to point to the structure in the use of three different control-loops by all three actuations. Steering Control is in one of them. Execution of the control loops take place in the main function where threads to the shared memory are in running. The calculated power output by control algorithm lies in the range between $\pm$ 2001 in counts. Then it is amplified from 0 to 5 Amp to the motor under 24 volt DC by a current gain amplifier.

However some of the unexpected steering responses are due to the LVDT noises in the feedback. Figure 2.4B shows a sample of LVDT readings at steering position remaining 2027.

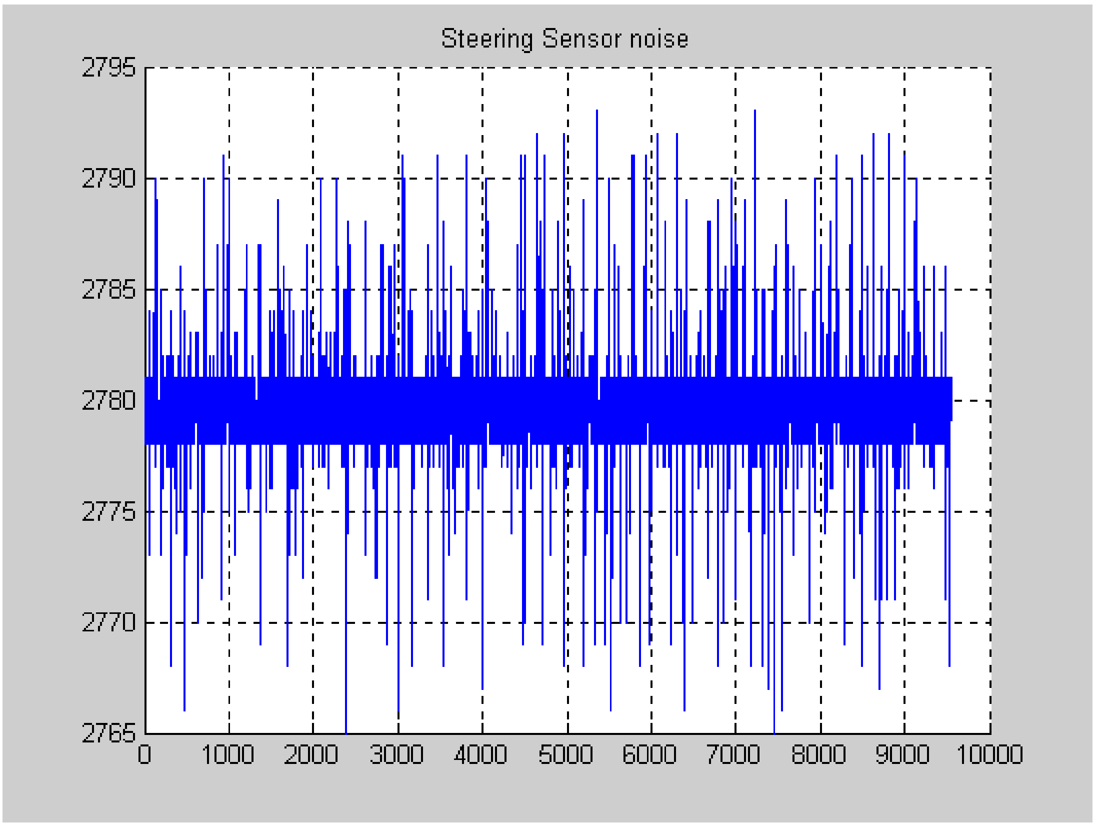


Figure 2.4B: Noise in LVDT readings of steering control system

The figure2.4B above indicates there are mass amount of noise in both direction with an average magnitude from 8 to 17 counts. In results some shape edges appear in most of the steering LDVT reading plots and since the maximum magnitude of noise is around 0.3-0.4 of a degree, and for different direction, the noise peaks varies. That may be one of the reasons we needs to have two sets of PI gains for both of the direction of turning(explained in later sections).

However, comparing with the rest of actuations, LDVT readings of the steering system have the lowest noises and considered to be much more reliable in control. Nevertheless, the reading can be further improved by technique such as median filters.

2.3 Steering Fine-tuning results

The tuning process is started to locate a set of gains to provide steering control system with the best response in terms of minimum overshoots and steady state errors.

Steering fine tuning processes are taken on the grass land near St. John College. Results are taken in shinny day to avoid variations in results caused by weather related ground conditions. During the fine tuning processes, speed of vehicle is maintained around 10 km/hr. Hercules software(See Chapter 3) is used to provide auto pilot of the steering and ease in fine-tuning.

Steering fine tuning processes are mainly focusing on the steering Reponses on step inputs. Then further results such like ramp and sinusoidal inputs are taken under the fine-tuned gain set concluded from step inputs results.

2.3.1 Step input results

In the low level control of HSV, overshoots and steady state errors are essential towards the performance of the steering control system while the system is following a steering command from a high level control algorithm. Steady state errors of the steering responses sometimes are more important in cases of a ‘slow’ high level control. For instance, during a path following process, if the steady state error in low level control is high, after 3 seconds the vehicle may have the orientation offset up to 10 degrees under a high speed. It is depending on the control interval time or sample time of the high level control system as well.

During the tuning process, it is showing that steering system under only proportional-control(only Kp) seems to have some surprisingly good performances in terms of no overshoots and faster rising time characteristics.

But the only problem of proportional controlled steering system is it shows a large steady state error in the response. Therefore introducing Ki term of control is necessary to reduce steady state errors, but in return it increases overshoots of the system response. In PID controlled steering responses, Although derivate term of gain(Kd) can eliminate a certain amount of overshoots created by integral term of gain(Ki), but at the same time steady state error and overshoot time increase.

As discussed before, in the low level control of HSV, overshoots and steady state errors are essential towards the performance of the system. As results indicated, a PI controller has a better performance than any other type of controllers.

Therefore focus of finding a better gain set has been moved from PID region to PI region. In other words, by comparing the steering response under a PID with under a PD controller, it is clear that a PI controller can bring us a better response.

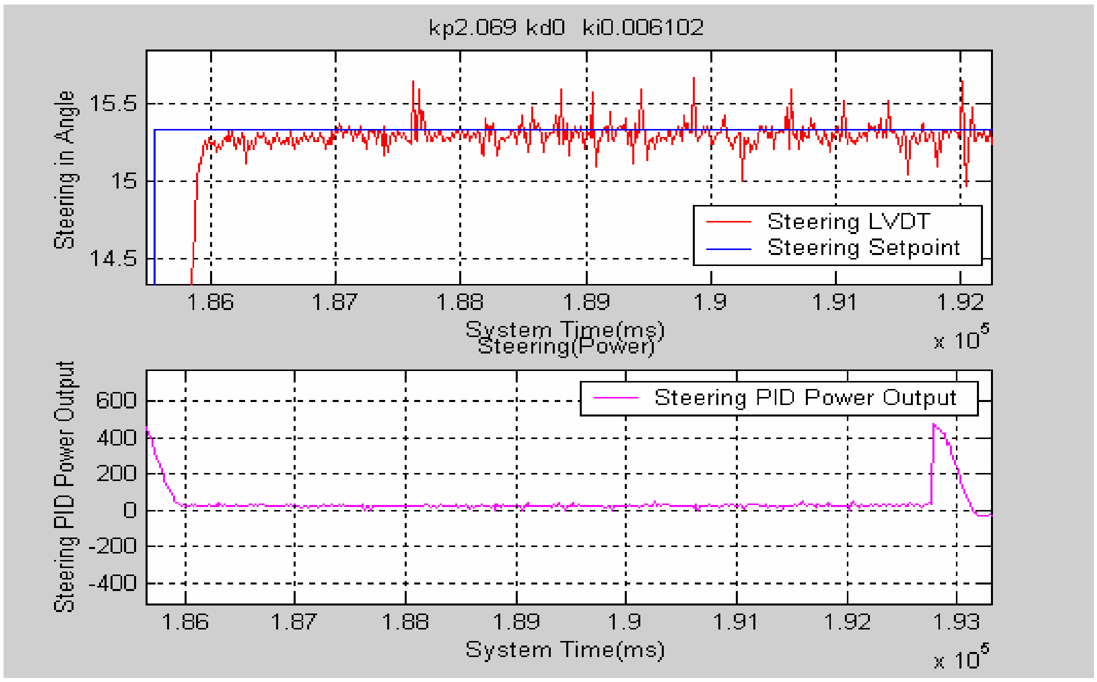


Figure 2.5: Step inputs response of the PI-controlled steering system

In general for turning the steering control system, the gains of Kp=3.276, Kd=0 and Ki = 0.006102 has been found to be one of the reliable gain sets which can have a very high accuracy in terms of minimum overshoot, overshoot time and steady state error. The steering response has an average overshoot around 1.8 degrees in 0.2 seconds under the speed of 15km/hr. The average steady state is about 0.4 degrees. In addition, a PI controlled steering system with a small integral term no less than 0.006102 can have the performance when no other types of controller can achieve. As in Figure 2.5, some particular performances of steering have been recorded down of the 0.1 steady state error in degree and zero overshoot for a small integral term.

In addition to the fine tuning process, it is found that the generally fine-tuned steering responses while turning right and left are different. Therefore two more fine-tuning processes had been taken separately to find gain sets for each side of directions vehicle turns to. While turning to the left direction, steering response in figure 2.6 shows the maximum overshoots of 1.7 degree, a minimum steady state error of 0 to 0.1 degrees at average,

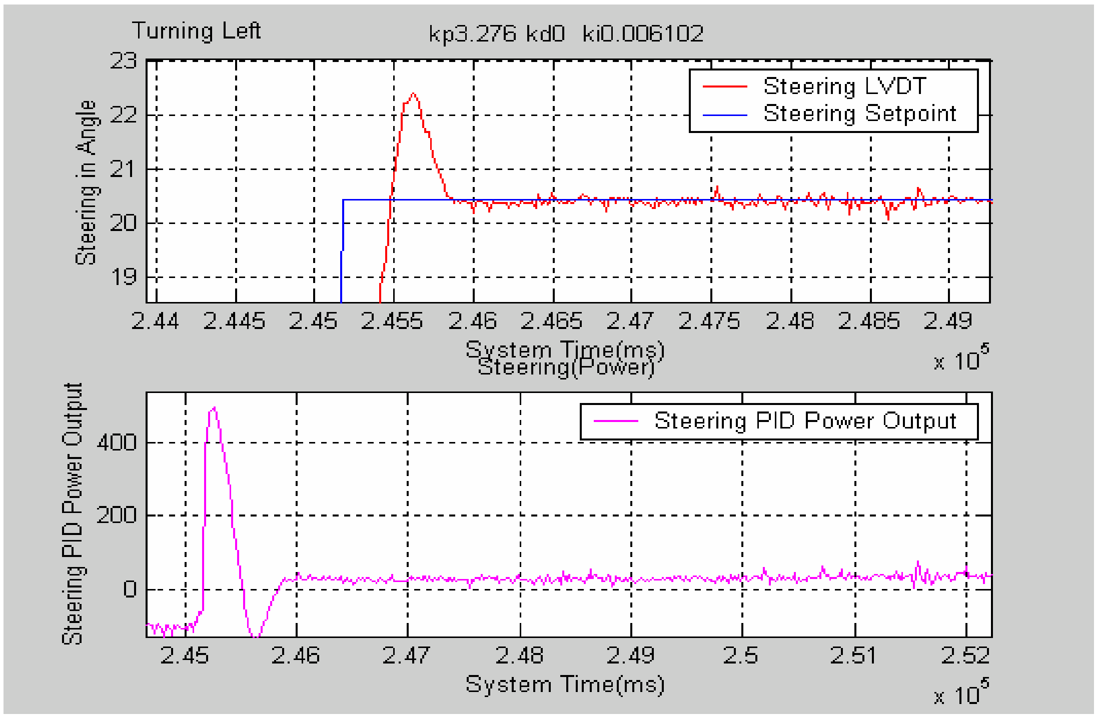


Figure 2.6: Fine tuning process(Left)

Kp = 3.276 and Ki = 0.006495 are found to be suitable for vehicle to turn to the right direction in figure 2.7. The average overshoots are 1.5 to 1.7 degrees in angle, overshoot time is about 0.2 seconds. The average steady state error is recorded as 0.1 to 0.5 degrees.

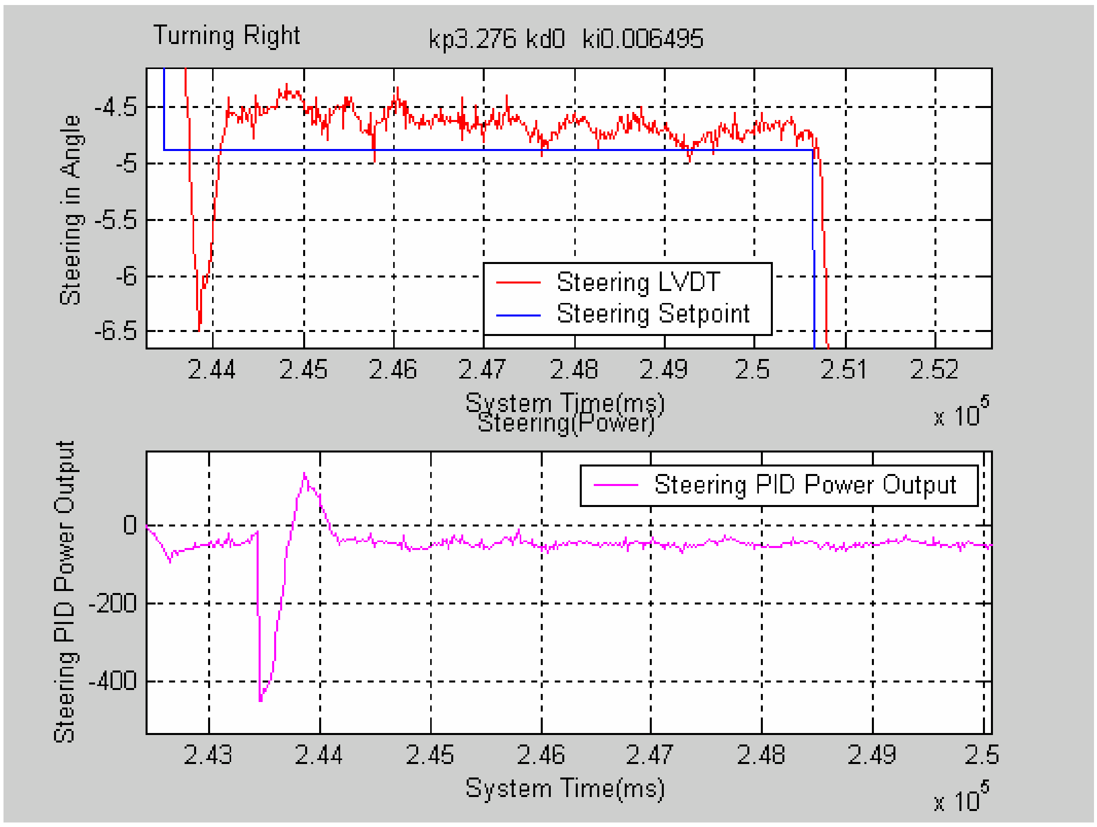


Figure 2.7: Fine tuning process(Right)

Results also indicate that decrease of Ki will be better for the left-turn application but be worse for the right-turn situation. Similar characteristics but opposite are found in the increase of Ki.

In conclusion of the PI controller, which can provide the best performance of the low level steering control system has the set of PI gains of 3.276 in Kp and 0.006102 for Ki. Due to the characteristic of motor's dead zone related steering responses, steady state error of the steering can not be set down to zero when power output to the motor is within the dead zone region(Please see Chapter 4 Section 4 for more detail).

2.3.2 Ramp input results

The figure 2.4 is a ramp input test taken as increment of one degree per 1.4 seconds shown below in figure 2.4. The result is taken as PI controller gains Kp=3.276 and

Ki=0.006102 and indicates the error between the desired position of the steering to the actual position is between 0 to 0.7 degree. The result of a power output plot(Figure 2.4) indicates the characteristic of the motor's dead zone related steering response. Error of steering can not be set to zero also because of the little-angle-increments related steering response(See Chapter 4 Section 3 for more detail).

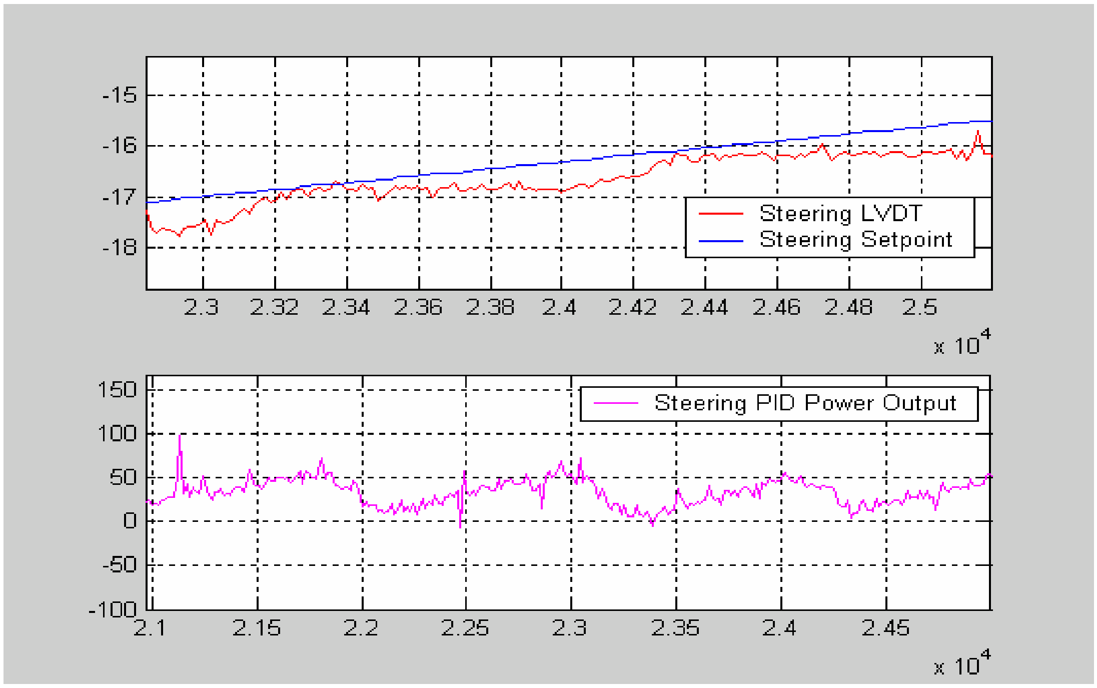


Figure 2.4: Ramp inputs response of the steering

In figure 2.4, it is also clear that from system elapsed time of 23.4 seconds to 24 seconds, although power output to the motor is increasing, steering does not move due to insufficient torque generated from these power.

This characteristic of the steering response has been categorized as motor characteristic related steering responses and it will be explained and analyzed in later chapter.

2.3.3 Sinusoidal input results

In sine wave input results(figures 2.5 and 2.6) under fine-tuned PI gains of Kp=2.069, and Ki=0.006102 under a speed of approximately 10 km/hr shows there is a increase in error between desired and actual position of the steering as steering reaches towards both of its limits.

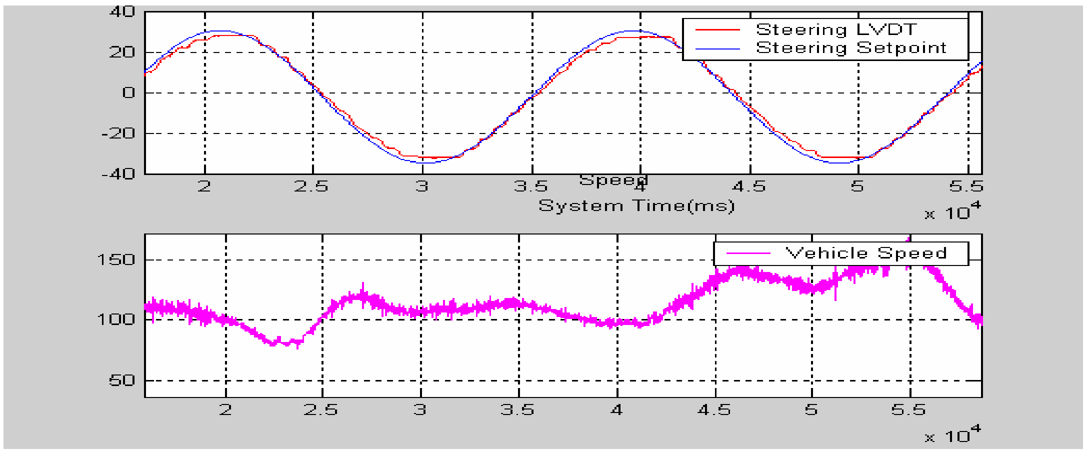


Figure 2.5: Speed vs steering in sine wave inputs

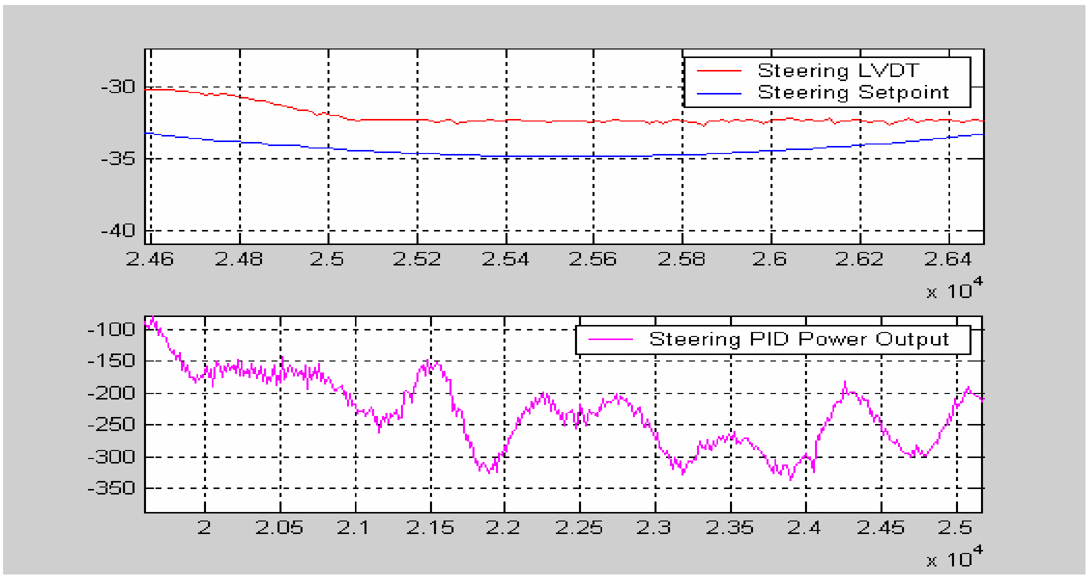
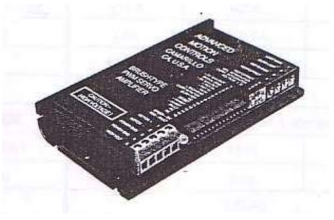


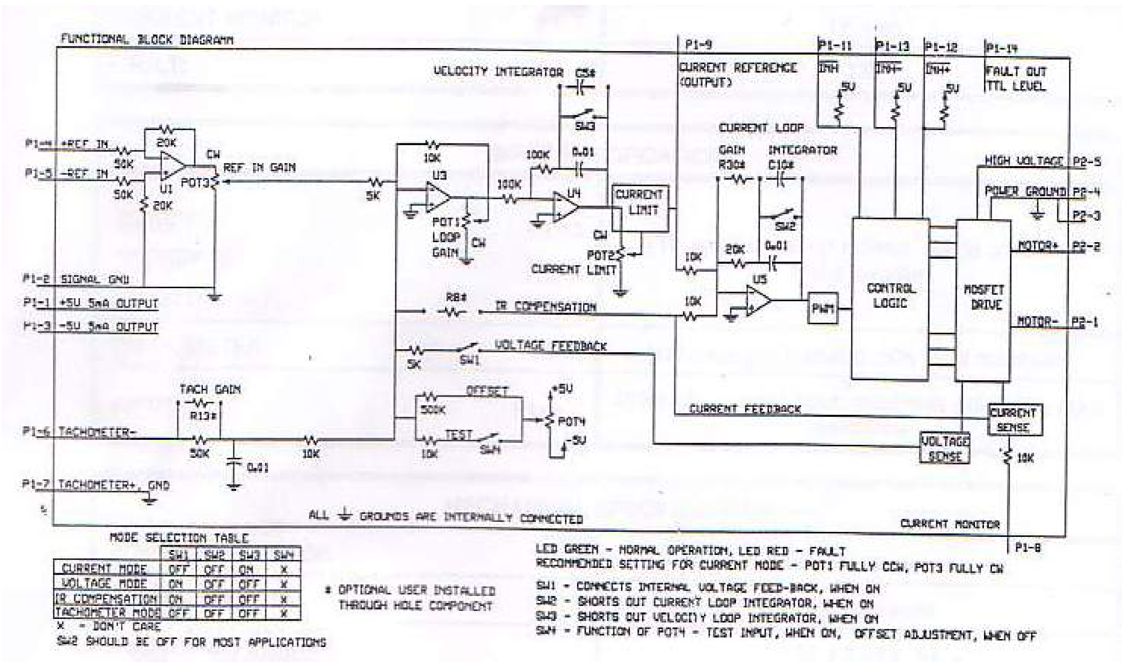
Figure 2.6: Power VS Steering in sine wave inputs

This result can be further improved by a bigger Kp gain increased from 2.069 to 3.276 as much powerful toque will force motor to turn the steering wheel near its limit.

2.4 Steering Amplifier configuration



A 25A Series PWM servo amplifier is used to drive the brush type DC motor, which controls the steering. It is protected against over-voltage, over-current, over hearing and short-circuits across motor, ground and power leads by a limitation current fuse. Single red/green LED indicates its operation status. Loop gain, current limit, input gain and offset can be adjusted using 15-turn potentiometers. The offset adjusting potentiometer can also be used as an on-board input signal for testing purposes when SW4 (DIP switch) is "On". The Configuration map of the amplifier is shown in figure 2.7.



MODE SELECTION TABLE

	SW1	SW2	SW3	SW4
CURRENT MODE	OFF	OFF	ON	X
VOLTAGE MODE	ON	OFF	OFF	X
IR COMPENSATION	ON	OFF	OFF	X
TACHOMETER MODE	OFF	OFF	OFF	X

X - DON'T CARE
SW2 SHOULD BE OFF FOR MOST APPLICATIONS

OPTIONAL USER INSTALLED THROUGH HOLE COMPONENT

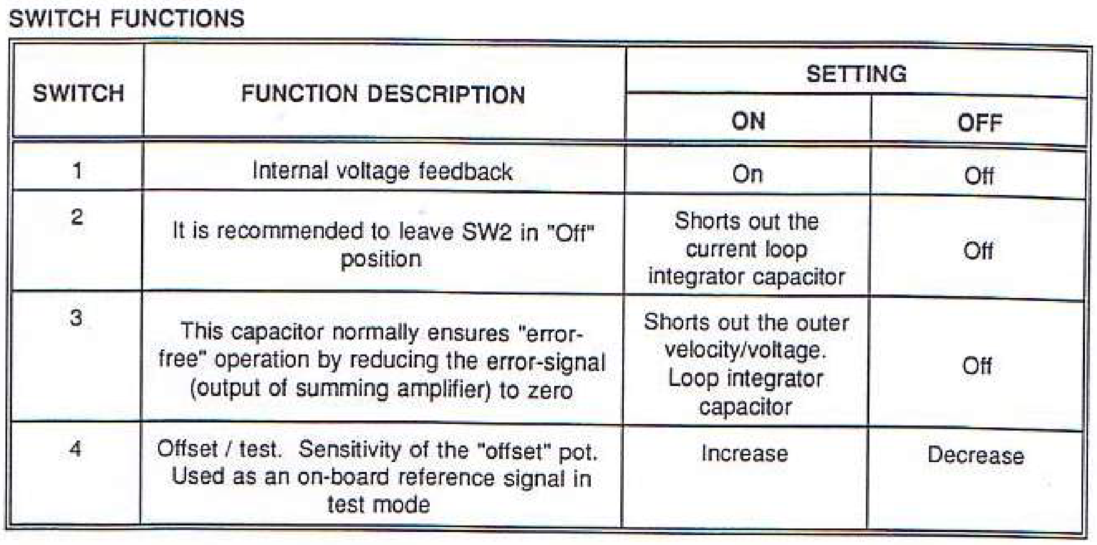
LED GREEN - NORMAL OPERATION, LED RED - FAULT
RECOMMENDED SETTING FOR CURRENT MODE - POT1 FULLY CCW, POT3 FULLY CW

SW1 - CONNECTS INTERNAL VOLTAGE FEED-BACK, WHEN ON
SW2 - SHORTS OUT CURRENT LOOP INTEGRATOR, WHEN ON
SW3 - SHORTS OUT VELOCITY LOOP INTEGRATOR, WHEN ON
SW4 - FUNCTION OF POT4 - TEST INPUT, WHEN ON, OFFSET ADJUSTMENT, WHEN OFF

Figure 2.7: Configuration map of steering amplifier

In mid 2002, the current gain factor at Pot 1 has been adjusted to produce current output range of 0 to 5A to satisfy the maximum torque output of the steering motor when output power of PID controller returns the maximum value of 2001 counts.

Before adjust any current gains or limits of the amplifier, please check switch 1 and 2 are off and switch 3 should be on and switch 4 don't care for current gain mode. The information on functionalities of these four switches is listed in figure 2.7b below:

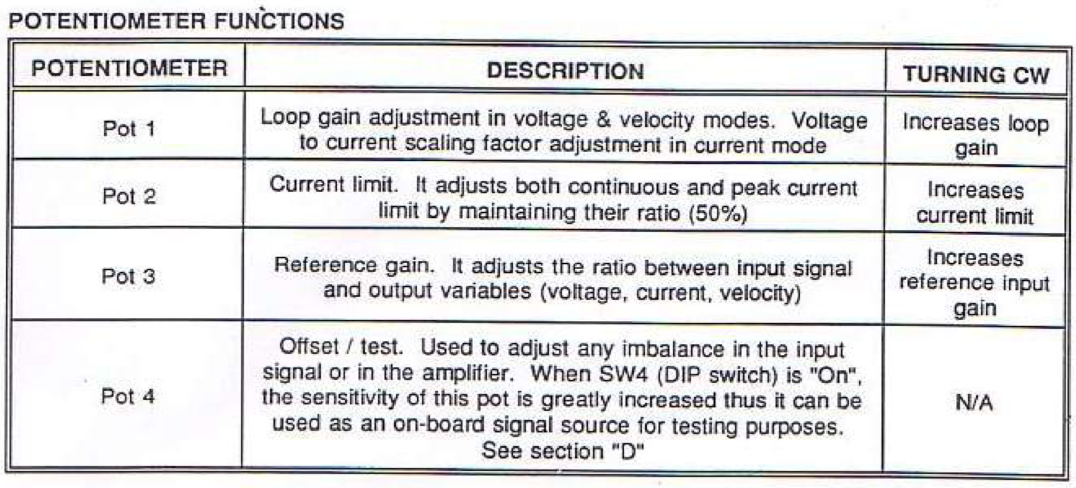


SWITCH FUNCTIONS

SWITCH	FUNCTION DESCRIPTION	SETTING	
		ON	OFF
1	Internal voltage feedback	On	Off
2	It is recommended to leave SW2 in "Off" position	Shorts out the current loop integrator capacitor	Off
3	This capacitor normally ensures "error-free" operation by reducing the error-signal (output of summing amplifier) to zero	Shorts out the outer velocity/voltage. Loop integrator capacitor	Off
4	Offset / test. Sensitivity of the "offset" pot. Used as an on-board reference signal in test mode	Increase	Decrease

Figure 2.7b functionalities of four switches

Four potentiometers are used to adjust the limits or gains in different mode as listed in figure 2.7c, only pot 1 is used to tune current gain factor. It is located in the left-center op-amp on the configuration map of the amplifier(Figure 2.7).



POTENTIOMETER FUNCTIONS

POTENTIOMETER	DESCRIPTION	TURNING CW
Pot 1	Loop gain adjustment in voltage & velocity modes. Voltage to current scaling factor adjustment in current mode	Increases loop gain
Pot 2	Current limit. It adjusts both continuous and peak current limit by maintaining their ratio (50%)	Increases current limit
Pot 3	Reference gain. It adjusts the ratio between input signal and output variables (voltage, current, velocity)	Increases reference input gain
Pot 4	Offset / test. Used to adjust any imbalance in the input signal or in the amplifier. When SW4 (DIP switch) is "On", the sensitivity of this pot is greatly increased thus it can be used as an on-board signal source for testing purposes. See section "D"	N/A

Figure 2.7c functionalities of four potentiometers

Please read through all configuration procedures available before changing the power output limits of the amplifier. If motor is suddenly out of function and it shows no current to the motor, please check the fuse that protect the amplifier and replace them under the permission of any HSV stuff.

Any more specify changes on the current gain and current limit of the amplifier needs to be discussed in meeting first. An unsuitable gain or limit will cause harm to the motor. There are certain cautions notes to be taken for the safety of amplifiers in HSV's steering testing processes.

1. Do not reverse the power supply leads
2. Use sufficient power supply capacitance
3. Do not spin the motor without power
4. Do not short the motor at a high speed

2.5 Speed Control system

Since the year 1997, speed control has become one of the focuses in low level control system development. In Mark's thesis[2] 1997, he has implemented the two methods(both simple and reversed smoothing filter) of smoothing the data collected from wheel encoder, stated the relationship between the velocity and acceleration from the filtered data. In Lawrence thesis[3] 1998, he mentioned the possibility of controlling the velocity in respect to the wheel encoder reading. In his thesis, he designed the first logical control algorithm for speed control in HSV but couldn't have the time to finish it. At that time, the logical control algorithm he was written is very similar to a fuzzy logic control algorithm.

In difference to a PID control law, a fuzzy logic or a neural fuzzy logic control law does not calculate the output according to a very precise input or the calculation process is only base on some pre-set logics.

Imaging a driver driving a car on a high way, the variation of the vehicle's speed depends on the force he steps down on the throttle paddle and the force which the driver used to step down on the brake paddle. By how much force applied to each paddle, the driver does not calculate in his mind. In the explanation from a biology point of view, the inner ear is the body's gyroscope, telling the brain at all times the head's orientation and movement in space. Fluid moves through a set of three semicircular canals and two otolith organs to constantly inform the brain as to the direction and the speed at which the head is moving. The system then directs the movement of the other parts of the driver's body to slow down or accelerate the vehicle according to current state of event happening around him. For instance, when a driver wants to maintain the vehicle slowly under 5 km/hr, he can maintain it by using the friction force generated between the tyre and ground instead of brake system in the vehicle and only applying only a little force on the throttle paddle when speed is fallen too fast. The phoneme for the driver to sense speed and making decisions to produce accordingly amount of force to decelerate or accelerate the vehicle as to be described as "the feeling of speed in control" and it can not be easily explained and copied down as codes. Certainly it does not only base on the error of speed to be simply controlled under a PID controller but decisions and rules that only can be set as logics to follow. We can reproduce them as some pre-set logics or rules which simulating decisions that a driver will make in react of controlling the vehicle's speed in reality.

Also as we know that during the speed control of a vehicle in reality, a driver will either step down on the throttle paddle or the brake paddle, and will not step down on both paddles at the same time. This rule is the logic rule in controlling the speed as well.

Therefore the autonomous control of the speed is more likely to favor a fuzzy logic controller, which determines the amount of Set-point for each of two actuators. The fuzzy logic controller has the input as the error between desired speed and actual speed from sensors. It then produces two proportional outputs as the position set-points of both PID controlled actuators.

2.5.1 Introduction

The speed control system mainly consists of a fuzzy logic controller and two PID controllers for each of two actuations.

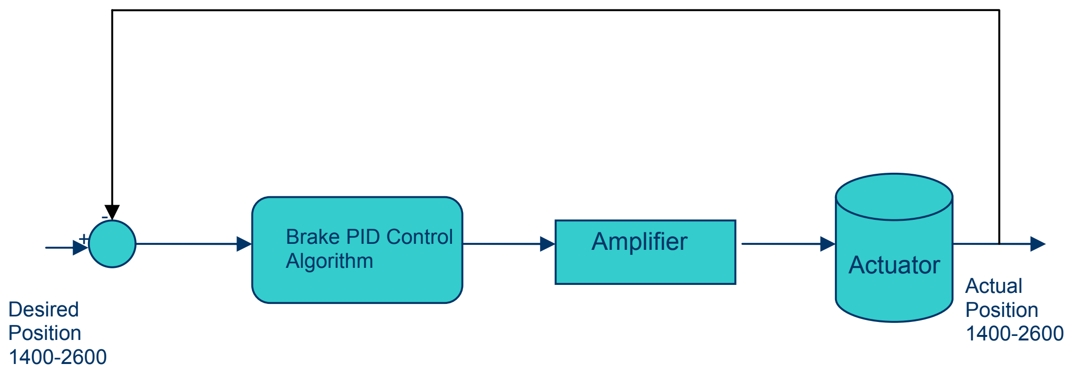
A fuzzy logic controller is used to decide which of two actuation controller should be used to control the speed at the moment. The input to the controller will be the difference in desired and actual speed, and the output from the controller will be both set-points of the two actuators.

The two PID controllers for brake and throttle(accelerator) actuation controls are in a way very similar to the steering position control. The inputs of the two PID controllers will be the error between desired and actual positions of the cylinders in terms of the retracted or expended length from the actuator housings. The outputs of the two controllers will produce the necessary power output for the next state to compensate the position errors existing in the current state.

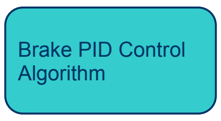
Once the two PID-controlled actuator control systems can efficiently in reduce of the error in desired position(set-point) and actual position. The error in speed will be reduced as well and hence speed control is achieved.

2.5.2 Brake Control System

Similar to the steering control system, brake control system is mainly consists of a Brake PID control algorithm, a amplifier, a linear actuator and a potential meter reads position of the retracted or extended cylinder as $\pm$8.6 vol for feedback.



+
-
Brake PID Control Algorithm
Amplifier
Actuator
Desired Position 1400-2600
Actual Position 1400-2600



Kp = 4, Ki = 0, Kd = 0

Tuning in demand

- Input: Desired Set-point
- Output: Actual position
- Amplifier: Current Gain
- Actuation: Linear actuator

Figure 2.8: Brake Control System

The PID control law used in the brake is the same as the rest. The integral term of error needs to set to zero each time a new desired position(set-point) is given. The effected position of the brake actuator is found to be in between 1400 to 2600 in counts. When the position of the brake actuator is at 1400, the cable will be pulled slightly and just stops when tension in cable reaches maximum. When actuator stops at 2600, the brake mechanism reaches it maximum capacity and its paddle can not be pressed further down. The variation of the range due to temperature changes and duration of a mechanism assumed to have small impact the effective range.

The effective range of brake actuator is used for the linearization of the fuzzy logic controller to control the speed of the vehicle and it is important in controls of a very slow speed.

Comparing with the LVDT reading from steering control system, the potential meters of both brake and throttle(Accelerator) actuators return data containing a large amount of high peak noises as shown in figure 2.9. The figure is taken when no powers are flowing into the actuator.

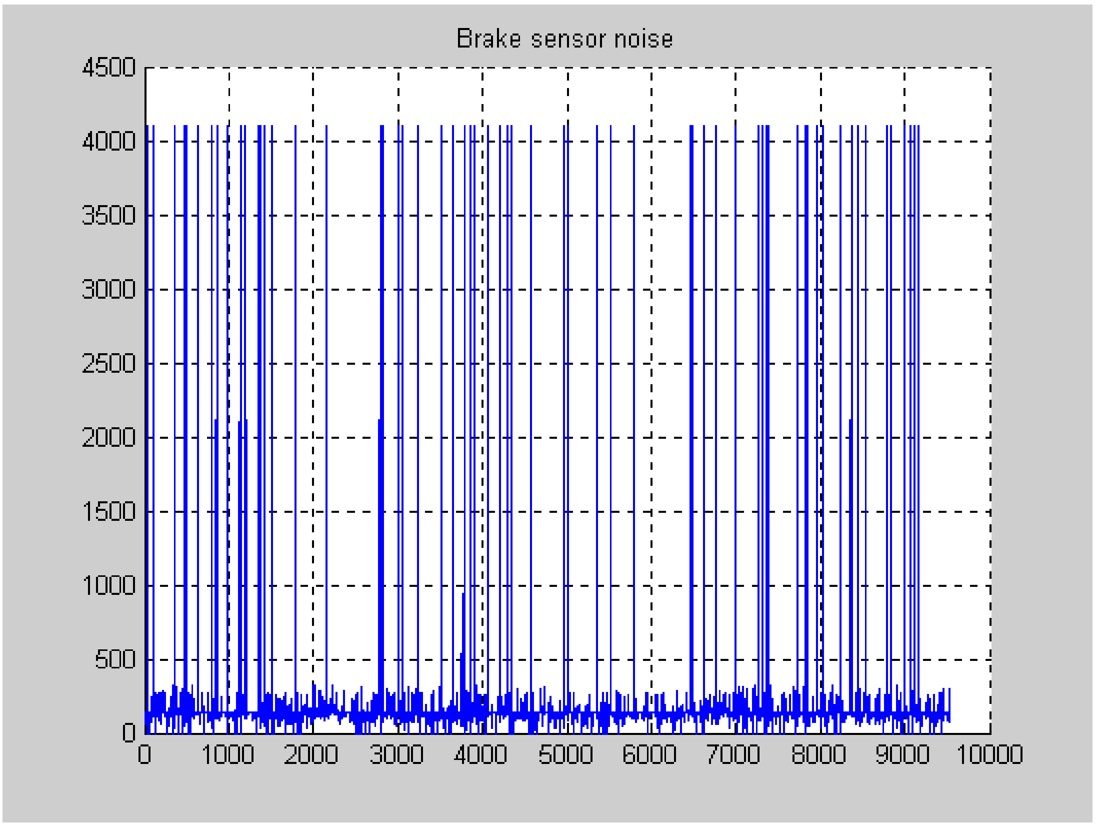


Figure 2.9 Noise from Potential Meter Reading in Brake Control system

From the figure 2.9, it is obvious that there are some very high peak noises in the positive position. Peak of noises are up to 4100, but since they are very singled-out high peaks and happens very fast, they can be eliminated by a median filter. A boundary for output power can also be added to eliminate oscillation of the brake actuator caused by noise during the control. A boundary for output power has been set as ± 50 for the brake to stay still under the noises exists as shown in figure 2.9.

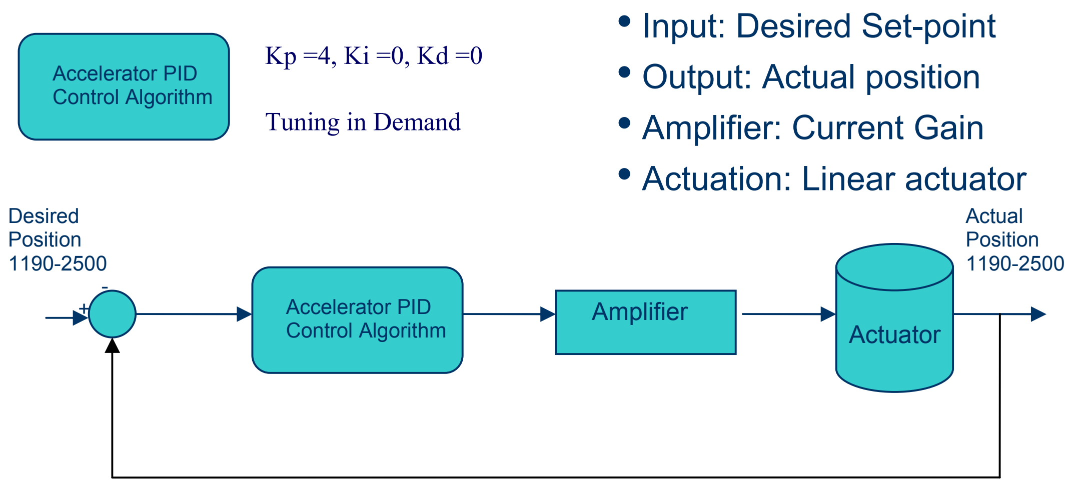
In HyperKernel Application, the maximum power output has been set to ± 1500. Currently the actuator is running under a proportional controller with Kp=4. A proper tuning of the controller is required and it is easier to be done after the signal noises are eliminated completed and the boundary of power to be re-opened again.

Once tuning process is ready, the brake actuator responses can be fine tuned to give very sensible and accurate results. The tuning process required a low level control tuning software to proceed,   e.g. Hercules 3.50(See Chapter 3 for tuning process).

2.5.3 Accelerator Control System

Similar to Brake Control System or Steering Control System, Accelerator Control System deals will the position control of the accelerator linear actuator. A cable is attached to the front end of the actuator's cylinder whereas the other end attaches to the butter fly valve which controls the amount of air flow into the engine. By retracting or extending the cylinder from actuator house, the cable is either pulled or set loosed by the amount the cylinder moved. When the cable is set loosed, the spiral pre-loaded spring on the butter fly valve will pull up the cable to close the valve and stop air flowing into engine.

Assuming that the geometric position of the cable attachment on the valve and the friction which cable is experiencing are not relevant to the accelerator control system, we can conclude the PID control strategy of accelerator actuator is just like the ones in the other two systems.



Accelerator PID Control Algorithm
Kp =4, Ki =0, Kd =0
Tuning in Demand
• Input: Desired Set-point
• Output: Actual position
• Amplifier: Current Gain
• Actuation: Linear actuator
Desired Position 1190-2500
+
-
Accelerator PID Control Algorithm
Amplifier
Actuator
Actual Position 1190-2500

Figure 3.0: Accelerator Control System

Similar to the brake control system, accelerator actuation's position also has an effective range from 1190 to 2500. But position of 2500 is not the when the accelerator actuation reaches its maximum capacity, it can be further increased during butter fly valve can not open any more. Currently under the effective range, the accelerator actuation can have a capacity from 0% to 90% of its total capacity limit. It can be changed by a further measurement.

Currently in the HyperKernel Application, the power output limit of the PID controller set to be -1000 to 1000 in counts. And similar to the brake control system, the power output has a boundary of ±70 counts to reduce noise-related oscillations. The accelerator control system responses with little overshoot under a proportional control of Kp=4.

From the figure 3.1which shows a sample of potential meter readings when there is no power out to the accelerator actuation.

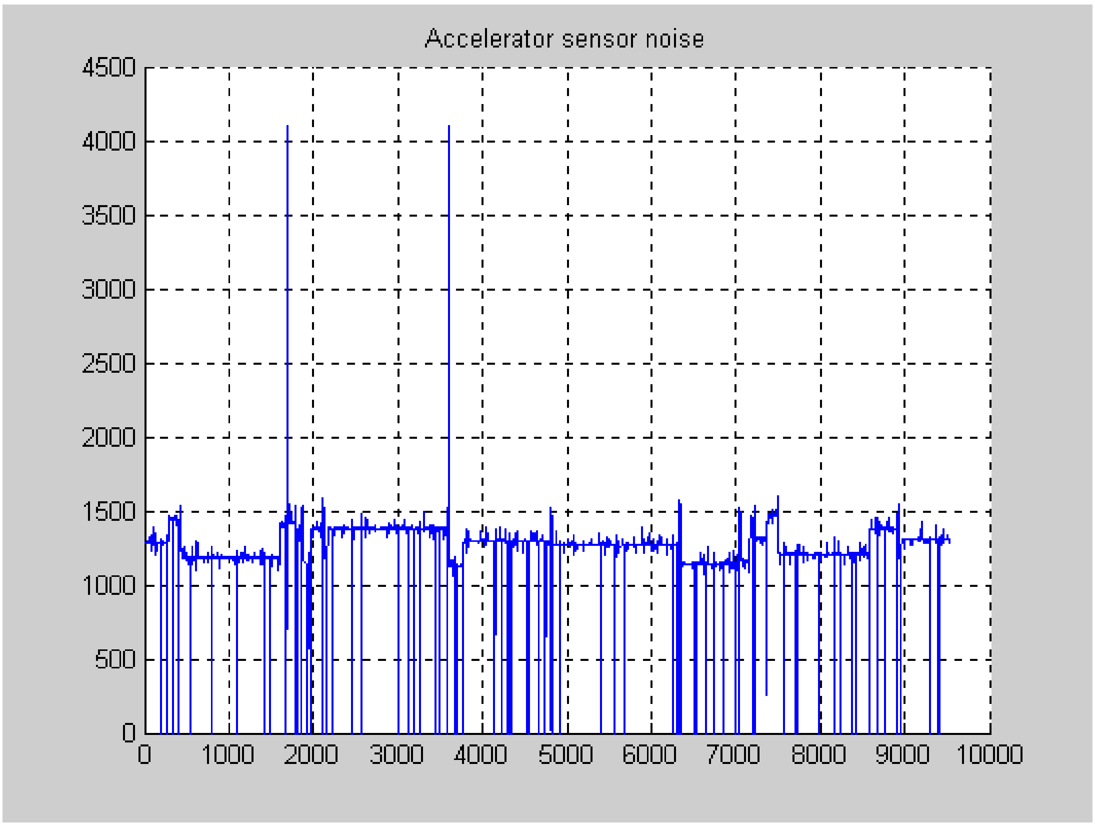


Figure 3.1: noise in potential meter readings in Accelerator control system

From the figure above, there is a large amount of noise to the negative side of the accelerator signal. Even the actuator does not move, the result shows that actuator signal is having bias or off set possibly by the damage on the power connection. Although a capacitor has been added to reduce the noise this year, problem might still occur when further damages are caused possibly in the future. Integral term of error should be turn down to zero whenever a new set-point is introduced. PID gains can be further tuned to achieve a better response of the accelerator control system.

2.5.4 Fuzzy logic Speed Control System

The fuzzy logic speed control system in figure 3.2 consists of two PID-controlled actuation systems and a fuzzy logic controller with two factors. The input of the system is the desired speed and the output will be the actual speed of the vehicle.

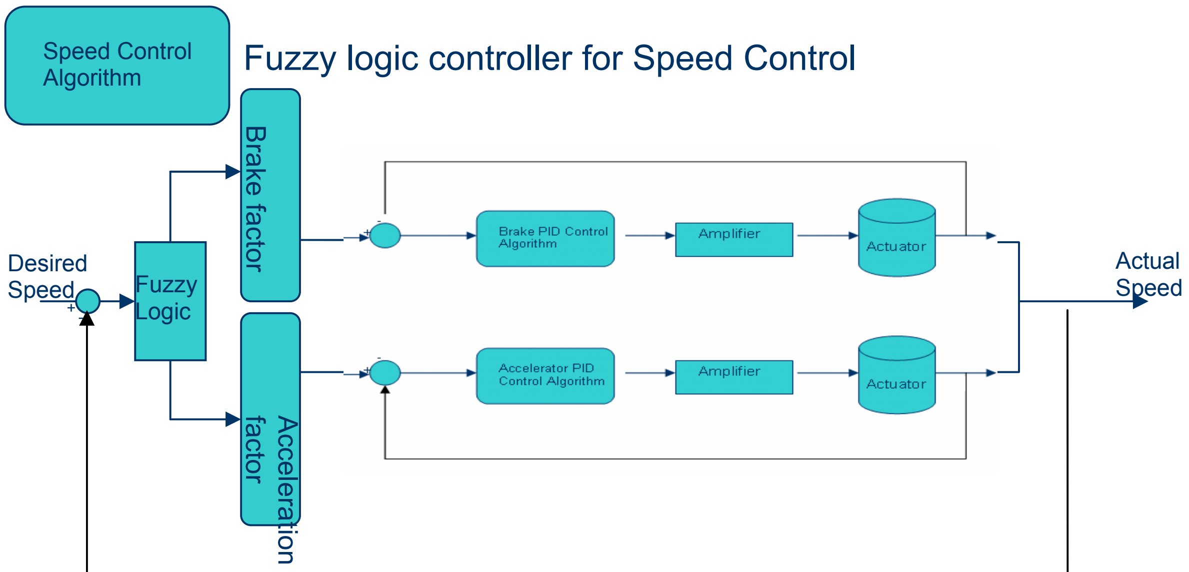


Figure 3.2 Speed Control System

The system is capable of controlling the speed according to the readings from both wheel encoder and a global positioning system(GPS).

The readings directly from wheel encoder are originally in the form of counts. It will be needed to linearized in terms of the difference in counts between previous and current states. A range is set as any value of difference bigger than the absolute value of 64000, it will be brought back into the region by plus or reduce 64-bit(64x1024) value.

It is then transformed from the position of the wheel encoder to the readings at centre of the vehicle in km/hr through the equation (2.1) given below.

$$V_{centre} = \frac{V_{encoder} \times 0.024970 \times (1 - 0.21)}{1 - \frac{\tan(\gamma) \times 0.76}{2.83}} \times \frac{3600}{1000} \tag{2.1}$$

Where γ is the steering angle in counts.

It seems that it is more convenient to use km/hr as the speed' unit in control than to use m/s, as m/s returns very small values. For example, 10km/hr is 2.7778 m/s and 5 km/hr is 1.38889 m/s respectively.

For GPS horizontal speed reading, it requires to be converted to km/hr as shown in equation (2.2).

$$V_{GPS'} = \frac{V_{GPS}}{1.994} \times \frac{3600}{1000} \tag{2.2}$$

Where $V_{GPS'}$ is the GPS horizontal speed in Km/hr and it is not transformed to the centre of the vehicle as it's position isn't fixed on the vehicle.
The speed control algorithm is implemented in HyperKernel application. Once the desired speed and actual speed are in the same units, error between them is then inserted into a Fuzzy logic controller. Error is calculated in the equation (2.3).

$$\varepsilon_{spd} = V_{desire} - V_{actual} \tag{2.3}$$

There are three conditions to be considered in terms of the speed error, they are when $\varepsilon_{spd} > 0$, $\varepsilon_{spd} < 0$ and $\varepsilon_{spd} = 0$.

When $\varepsilon_{spd} > 0$, which means the desired speed is greater than the actual speed, vehicle needs to accelerate forward to reduce the error. Therefore the desired position set-point as an input to the accelerator control system will be calculated according to the accelerator factor. The set-point of the brake control system will be set to 1400(no-brake status). The speed controller after sending a corresponding set-point to accelerator control system will wait for the next speed error input.

When $\varepsilon_{spd} < 0$, which means the vehicle is traveling too fast and needs to be decelerated, there are two more situations as pre-set logic conditions are tested. When the absolute of speed error $|\varepsilon_{spd}|$ is greater than α, where α is the maximum speed error that allows the use of the friction force on the ground to slow down the vehicle, the set-point of brake control system will be calculated due to the brake factor. On the contrary, if the absolute value of error $|\varepsilon_{spd}|$ is less or equal to α, both set-points of brake and accelerator actuations will be set back to original position where brake and accelerator's set-points are 1400(no-brake) and 1190(no-acceleration) respectively.

When $\varepsilon_{spd} = 0$, which means the desired speed is equivalent to the actual speed, then the set-point of both accelerator actuation and brake actuation will be set to original position, where set-point of brake and accelerator actuation are 1400(no-acceleration) and 1190(no-brake) respectively.

In addition, When desired speed is zero, which means the user wants the vehicle to stop at the time, accelerator actuation will be set back to its original position at 1190 and brake actuation will be set to its maximum value of 2600 for a full brake status. Because when the gear box on the left of driver seat is set to *drive* mode, a little fuel is then started to be pumped into the engine producing a pushing force and starting vehicle go forward even the butter fly valve is still remaining close that the moment. It is necessary to stop the vehicle driving forward by a full-brake command whenever the desired speed is zero.

Chapter 3

Low Level Control Software Design

3.1 Introduction to Microsoft Foundation Class

The Microsoft Foundation Classes, usually abbreviated to MFC, are a set of predefined classes upon which Windows programming with Visual C++ is built. These classes represent an object-oriented approach to Windows programming that encapsulates the windows API. The process of writing a windows program involves creating and using MFC objects, or object of classes derived from MFC. The objects created will incorporate member functions for communicating with window, or in our case, the Hyper Kernel Application.

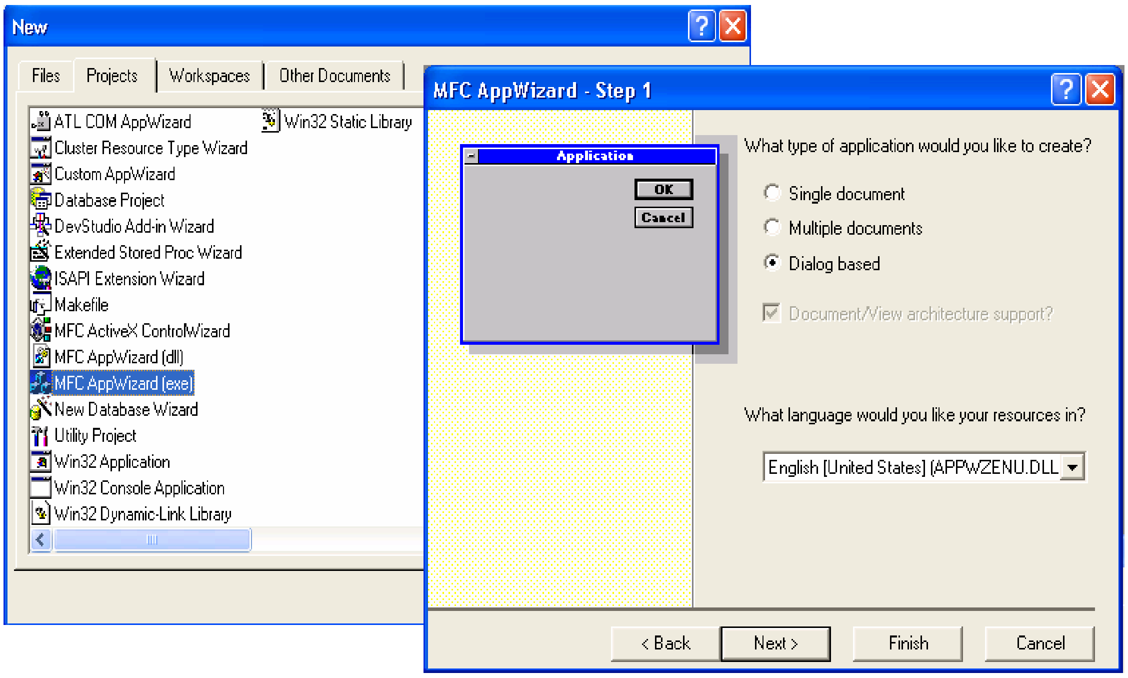
In addition to some more specialized tools, Microsoft's Visual C++ package encompasses a C/C++ compiler, a resource editor, a debugger, and the Microsoft Foundation Class Library. This class library provides a collection of C++ classes that take most of the drudgery out of writing software for windows.

The most compelling reason to use MFC is the vast amount of functionality that the classes can realize. Since the classes are targeted at the features your application needs – such as slider bars, the implementation required for multiple document windows – using MFC saves us coding time that we can spend on other features in our application.

"A journey of a thousand miles must begin with a single step."

Lao Tzu, Tao Te Ching

Let us begin with some features that lie inside the development of a MFC Application. A New MFC Application can be easily created through MFC Application Wizard in figure 3.3.



What type of application would you like to create?

( ) Single document
( ) Multiple documents
[✓ Dialog based]

[✓ Document/View architecture support?]

What language would you like your resources in?

English [United States] (APPWZENU.DLL

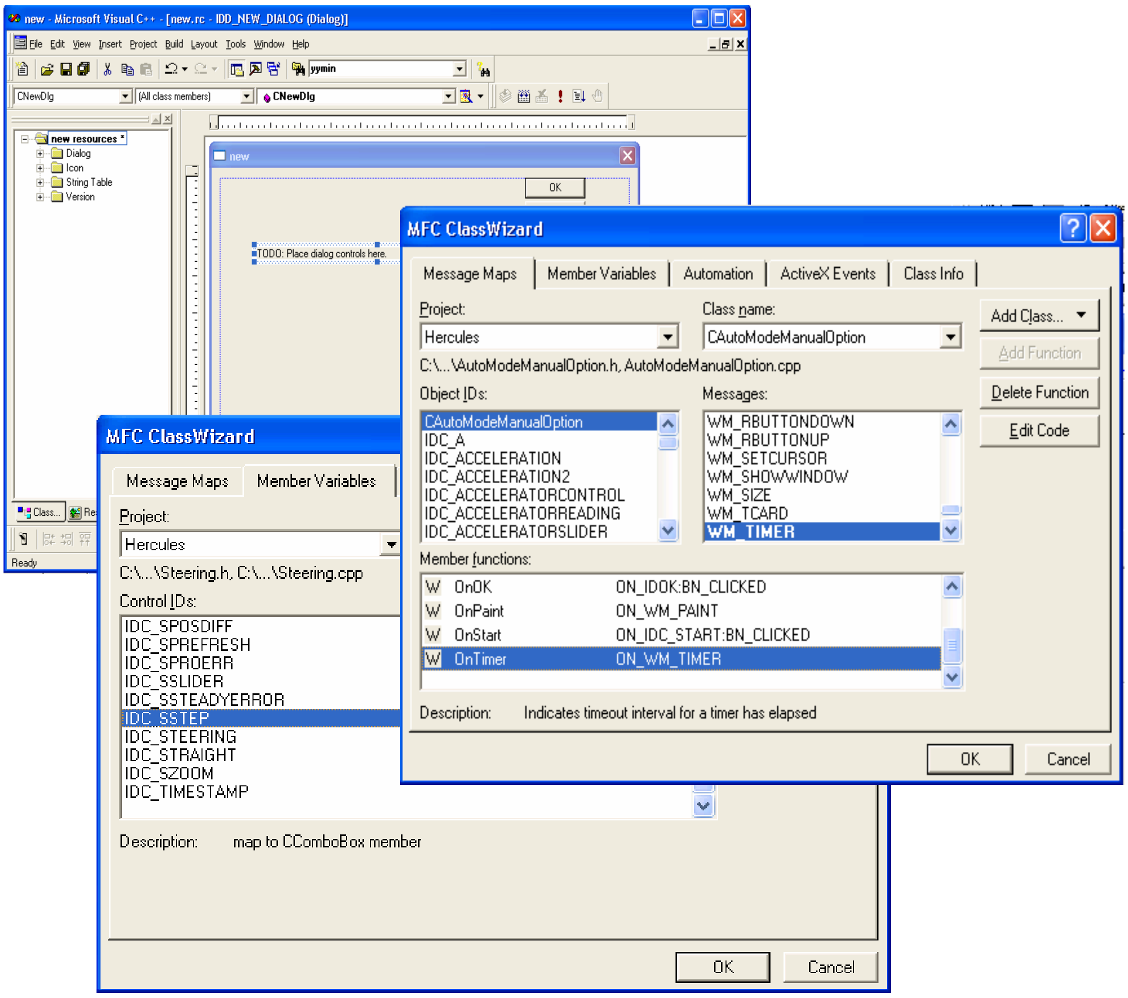
Figure 3.3: MFC AppWizard window

MFC provides tools windows such as MFC ClassWizard to provide ease in creating new member functions, member variables, events, message handlers.

An event or a message handler is a *Wnd*-member function, which handles the messages or any actions when certain events have been satisfied. You can call out MFC ClassWizard window any time by pressing Ctrl+w to set up a message handle or creating a member variable or a control member.

A member variable that defined as a control type to its control ID can use the member function built within the control object easily.

For example a defined CSlider member variable in the member variable in MFC ClassWizard(figure 3.5) as a control type can excess to all the CSlider class's member functions such as *CSlider::SetRange()* or *CSlider::SetPos()*.



Project: Hercules

Class name: CAutoModeManualOption

C:\...\AutoModeManualOption.h, AutoModeManualOption.cpp

Object IDs:
CAutoModeManualOption
IDC_A
IDC_ACCELERATION
IDC_ACCELERATION2
IDC_ACCELERATORCONTROL
IDC_ACCELERATORREADING
IDC_ACCELERATORSLIDER

Messages:
WM_RBUTTONDOWN
WM_RBUTTONUP
WM_SETCURSOR
WM_SHOWWINDOW
WM_SIZE
WM_TCARD
WM_TIMER

Member functions:

W	OnOK	ON_IDOK:BN_CLICKED
W	OnPaint	ON_WM_PAINT
W	OnStart	ON_IDC_START:BN_CLICKED
W	OnTimer	ON_WM_TIMER

Description: Indicates timeout interval for a timer has elapsed

Project: Hercules

C:\...\Steering.h, C:\...\Steering.cpp

Control IDs:
IDC_SPOSDIFF
IDC_SPREFRESH
IDC_SPROERR
IDC_SSLIDER
IDC_SSTEADYERROR
IDC_SSTEP
IDC_STEERING
IDC_STRAIGHT
IDC_SZOOM
IDC_TIMESTAMP

Description: map to CComboBox member

Figure 3.5: MFC ClassWizard window

In Graphical user interface design, the most commonly used controls are Slider controls, Progress Controls, Animation Controls, Button Controls, List Controls, Edit Controls and the Rich Edit Controls. Most of these controls will be explained as examples during the creation of the *Hercules 3.50* software.

3.2 Low Level Control GUI Hercules 3.50

3.2.1 Introduction

Hercules 3.50 is a tested-software, developed for HSV's low-level control graphical user interface. It has the abilities and functionalities to access all the low level controls in HSV, which include steering control and the speed control.

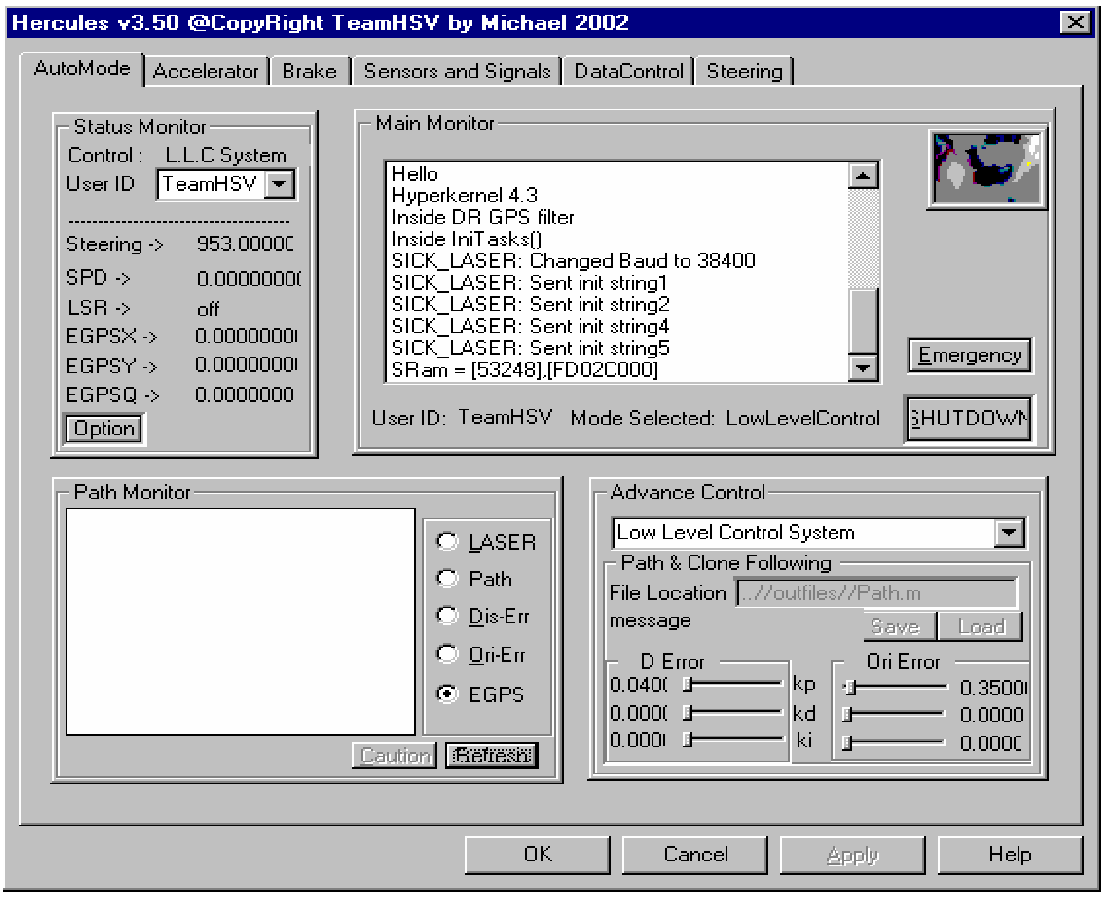
Hercules 3.50 also provides a path following tuning interface which allows user to tune the PID controller for path following on-line.

It not only allows users to control the motions of all three actuators(steering motor, throttle and brake actuators), but allows users to fine-tune the responses of these actuators through each of the corresponding PID control interfaces as well.

It has the access to HyperKernel application, shared memory and a navigation loop which allows the software to do path following and tracking for the high level control. It provides an automatic pilot system for step, ramp and sinusoidal inputs to the steering, which makes it much easier for the on-line fine-tuning process.

It provides in total 32 online plots monitoring all sensors and actuations as well as online path tracking and path following information.

In addition, it has an interface which allows user fine-tuning gains for path following and path tracking system. Besides all these, it has a built-in remote control user interface allow users to control HSV's motion by a keyboard.



Hercules v3.50 @CopyRight TeamHSV by Michael 2002

Status Monitor
Control : L.L.C System
User ID TeamHSV

```
Steering ->   953.0000C
SPD ->        0.00000000
LSR ->        off
EGPSX ->      0.00000000
EGPSY ->      0.00000000
EGPSQ ->      0.0000000
```

Main Monitor

```
Hello
Hyperkernel 4.3
Inside DR GPS filter
Inside IniTasks()
SICK_LASER: Changed Baud to 38400
SICK_LASER: Sent init string1
SICK_LASER: Sent init string2
SICK_LASER: Sent init string4
SICK_LASER: Sent init string5
SRam = [53248].[FD02C000]
```

User ID: TeamHSV Mode Selected: LowLevelControl

Path Monitor
LASER ( ) Path ( ) Dis-Err ( ) Ori-Err [✓ EGPS]

Advance Control
Low Level Control System
Path & Clone Following
File Location ..//outfiles//Path.m
message

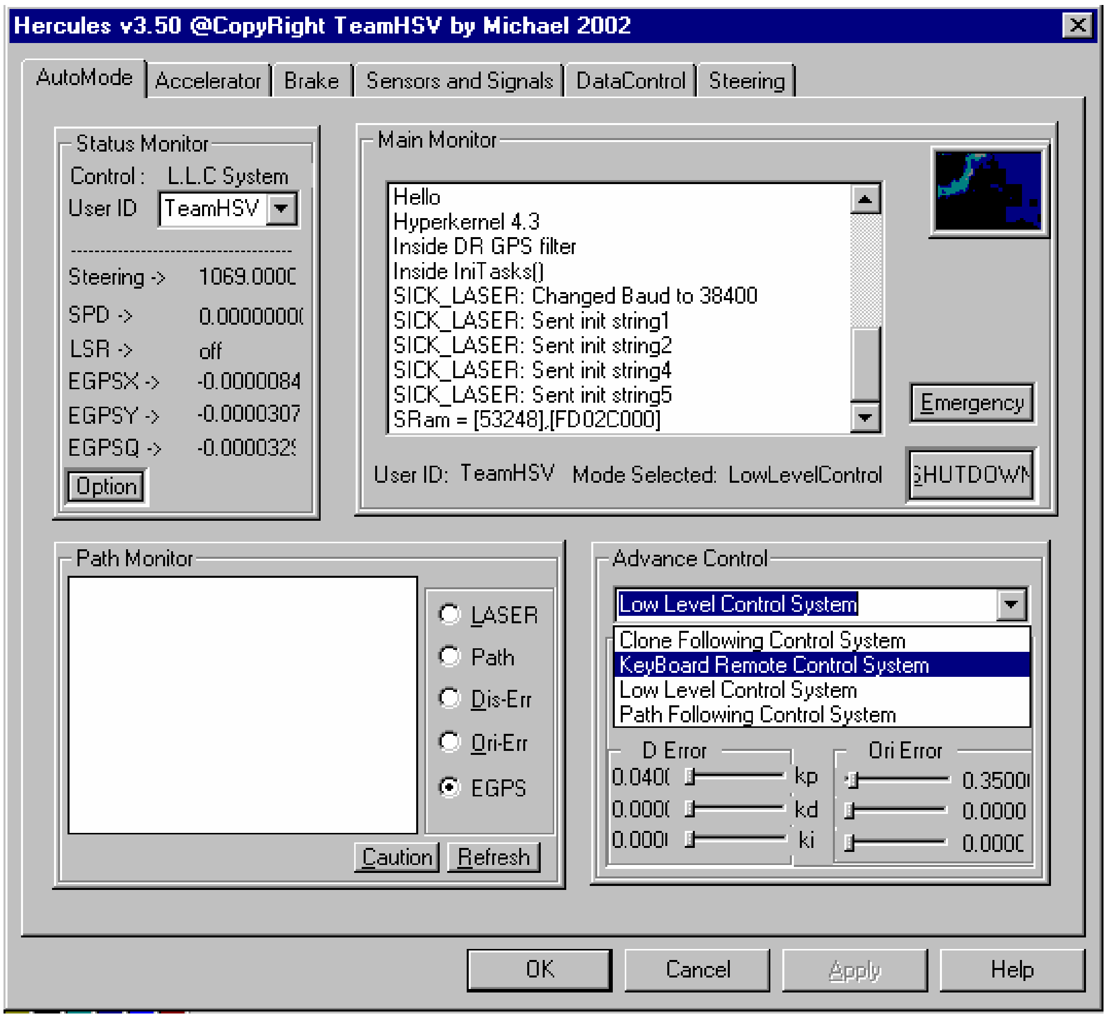
D Error			Ori Error
0.040(	kp		0.3500
0.000(	kd		0.0000
0.000	ki		0.000C

Figure 3.6: Low Level control system

Hercules 3.50 is a dialog based MFC application. It is constructed by grouping other dialog boxes logically together onto a tab-styled property sheet(figure 3.6). Each of these dialog boxes are called a property page and stored itself as a CPropertyPage object to the main sheet. Once the dialogs are created in the resource view window, they can be added into your main property sheet as an object by calling the *CPropertySheet::AddPage()* function. Then you will need to call out *CPropertySheet::DoModel()* function to active the object as a dialog box. You can also use *SetActivePage()* to set which page is active within the sheet by its page index. Given a pointer to a page, you can use *GetPageIndex()* function to find the page's index or call *GetActivePage()* to find a pointer to the currently active page object. Hercules 3.50 is a dialog box itself and before it ends its *CMyDialog::OnInitDialog()* function, it dynamically adds and actives its six CPorpertyPage Objects by calling *CPropertySheet::DoModel().* One good thing of using

CPropertySheet::DoModel() is you can keep your pages alive long enough, because the *DoModel()* call doesn’t return until the property sheet has been destroy. Each of the six CPropertyPage objects is created separately and has its own unique Graphical user interface.

3.2.2 AutoMode Control Page



Control : L.L.C System
User ID TeamHSV

Steering ->	1069.000C
SPD ->	0.0000000(
LSR ->	off
EGPSX ->	-0.0000084
EGPSY ->	-0.0000307
EGPSQ ->	-0.000032!

```
Hello
Hyperkernel 4.3
Inside DR GPS filter
Inside IniTasks()
SICK_LASER: Changed Baud to 38400
SICK_LASER: Sent init string1
SICK_LASER: Sent init string2
SICK_LASER: Sent init string4
SICK_LASER: Sent init string5
SRam = [53248],[FD02C000]
```

User ID: TeamHSV Mode Selected: LowLevelControl

Path Monitor: [✓ EGPS]

Low Level Control System
- Clone Following Control System
- KeyBoard Remote Control System
- Low Level Control System
- Path Following Control System

D Error		Ori Error
0.040(	kp	0.3500I
0.000(	kd	0.0000
0.000I	ki	0.000C

Figure 3.7 Automode main user interface

Automode CPropertyPage Object is the main user interface of all interfaces. As in figure 3.7, it mainly consists of a CList box control object, a CCombo box control object, six CSlider control objects and a few CButton control objects. These controls construct mainly four windows from the top left to down bottom. They are respectively Status monitor, Main monitor, Path monitor and Advance Control windows.

The CList box in the Main monitor is used to display debug message sent by the hyper kenel application. Once we find a pointer which pointing to the address of the CList control object, then we can send messages from hyper kenel application's debug to the address of the object through its pointer.

The CButton Control Objects are used for execute message handles whenever those buttons are pressed or checked. A message handler is a *Wnd*-member function, which handles the messages or any actions when certain events have been satisfied. You can call out MFC ClassWizard window any time by pressing Ctrl+w to set up a message handle or creating a member variable or a control member. In this case, for a button object the message handler will be executed when it is pressed or checked. There are mainly two types of buttons, ones are for pressing and the other radio ones are for checking. The ones for pressing are in rectangular shape, the radio ones are in small circular shapes. Rectangular-shaped buttons are most commonly used for executing the message handler. For pressing the *Start* or *Shutdown* button in the main monitor, it calls out the Hyper kernel start or shutdown functions defined in the HyperShared header file. The *Emergency* button is used to shutdown all the output powers to each of the steering, accelerator(throttle) and brake actuations. The message handler for this is simply triggering a bool flag. The flag is member variable called *m_Emergency* and it is accessible to all other CPropertyPage Objects. The *Refresh* button at the left-bottom Path Monitor window is used in refreshing the rectangle underneath the plots. The member function *OnPrint()* created as a message handler does not reprint the dialog when drawings on dialog are expired. It is necessary to create another message handler to *Refresh* the drawings on the dialog by calling out *UpdateWindow() Wnd*-member function. The radio button in the Path Monitor window will be explained in the Plotting section.

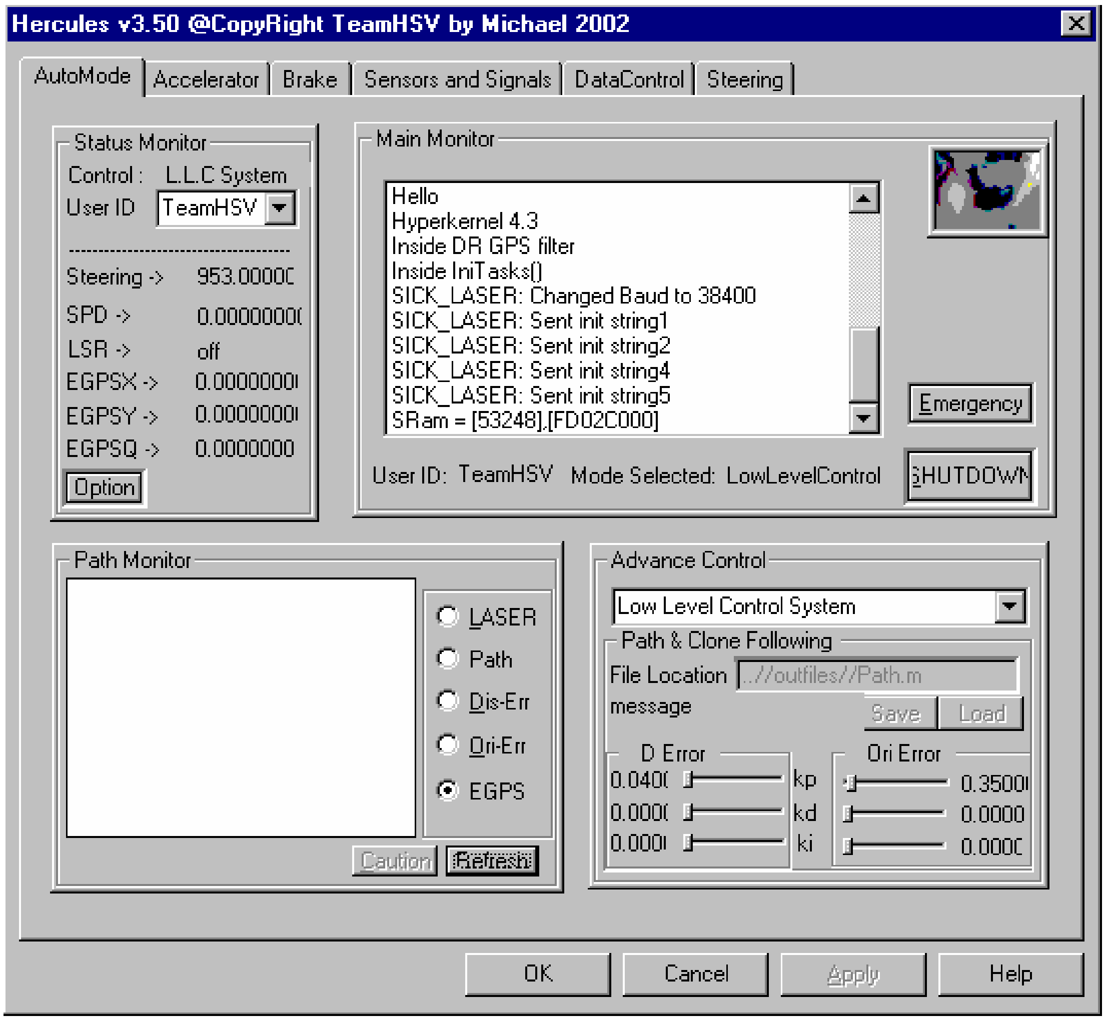
The CEdit Control Object is useful to the application when inputs from users are demanded. Unlike other control objects, CEdit does not have many member functions. In fact, the propose of using a CEdit control object is to be able to retrieve an input as a string from the edit slot. As an example, in the Advance Control Window, when Path Cloning or Path Following system is selected from the CList object, a edit slot will become enabled to allow user typing up the file path and name of which the user would like to save or load. *GetDlgItemText()* member function is called to retrieve the string from the edit slot.

The CSlider Control Objects are most commonly used in the Hercules software, as it is easy to control the actuations' positions or their PID gains by sliders. The MCF ClassWizard Window(Press Ctrl+w) will help you to define any type of member variables based on the control IDs of the control objects and its data type. For example, a member variable *m_DKpSliderCrl* is created under the control ID of *IDC_DKPSLIDER,* and the data type is selected as Control from the *Category* list box in Add Member Variable Window. After the member variable *m_DKpSliderCrl* of the CSlider Control type is created, you can use it as an object to access member function in CSlider. *CSlider::SetRange()* function is commonly used to set the minimum to maximum range of the slider. *CSlider::GetPos()* member function is used to retrieve current position of the slider. *CSlider::SetPos()* is to set a position on the slider. These member functions are the most powerful functions in CSlider Control. *CSlider::SetRange()* functions are most commonly to be called in the *OnInitDialog()* member function, as well as pre-setting the position of the sliders. The *OnInitDialog()* member function is created as a message handler, it is the first function to be called when you open the dialog. It is commonly called as the initialization function of the dialog. Together with *OnTimer()* member function which creates a run-time timer for the dialog, the position of the current slider can be retrieved every sampled time.

The CCombo box control objects in the Status window and in the Advance Control window are created to allow user to switch to a pre-set system easily. *CCombo::AddString()* and *CCombo::InsertString()* are used to add and insert text strings into the Combo box list. The index can be retrieved by calling *CCombo:: GetCurSel()* function. It is easy to firstly setting up a message handler on every time selecting or changing the present content of the Combo box from its list. Back to the software above, In the Advance Control window, the index pointing to a selected content from the list is stored in a member variable *m_ControlIndex.* This member variable plays a role as a flag of swapping control systems back and forth. There are four system to be chosen from:

Clone Following Control system, Keyboard remote control system, Low Level Control system and Path Following Control system.

3.2.3 Low Level Control System



Control : L.L.C System
User ID TeamHSV

Steering -> 953.0000C
SPD -> 0.0000000(
LSR -> off
EGPSX -> 0.0000000I
EGPSY -> 0.0000000I
EGPSQ -> 0.0000000

```
Hello
Hyperkernel 4.3
Inside DR GPS filter
Inside IniTasks()
SICK_LASER: Changed Baud to 38400
SICK_LASER: Sent init string1
SICK_LASER: Sent init string2
SICK_LASER: Sent init string4
SICK_LASER: Sent init string5
SRam = [53248].[FD02C000]
```

User ID: TeamHSV Mode Selected: LowLevelControl

[✓ EGPS]

Low Level Control System
File Location ..//outfiles//Path.m
message

D Error		Ori Error
0.040(	kp	0.3500I
0.000(	kd	0.0000
0.000I	ki	0.000C

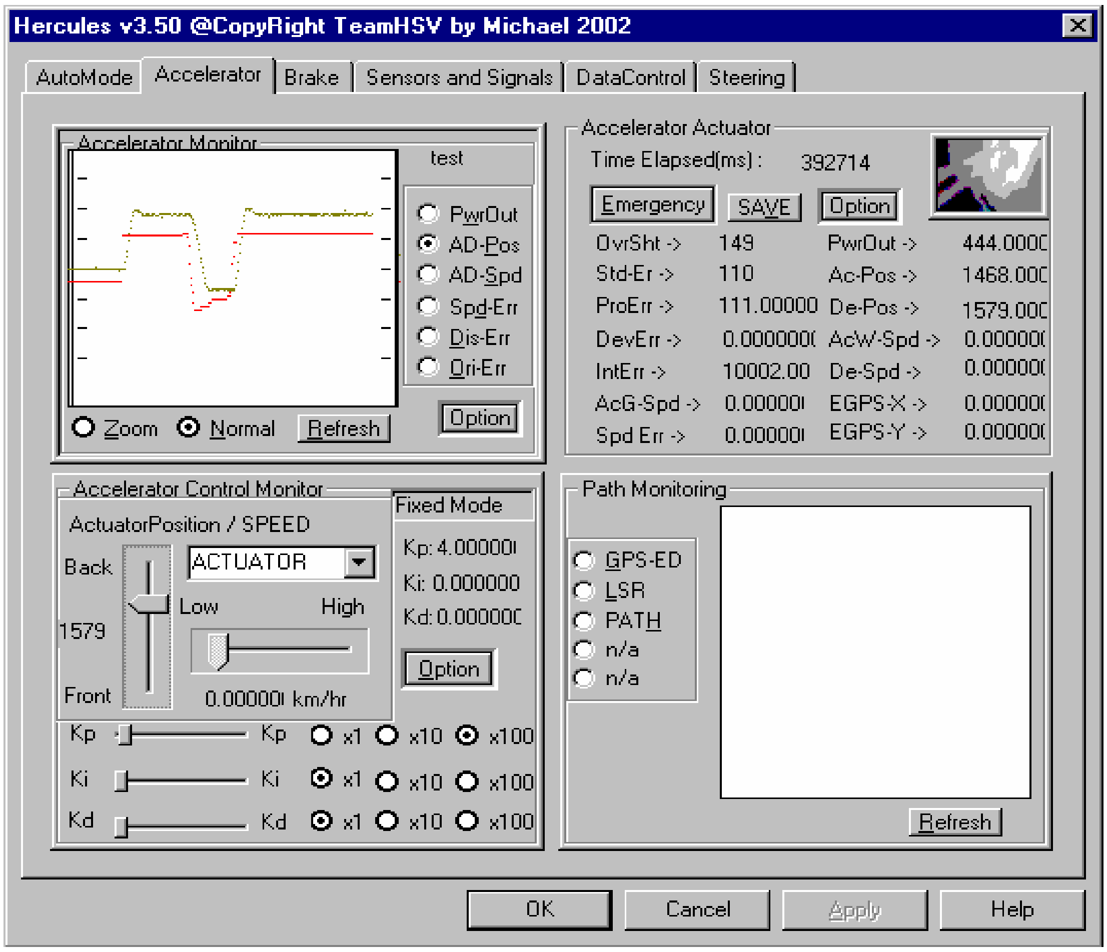
Figure 3.8 Low level Control System

Low Level Control System is built especially to fine-tune the three actuations(steering, brake and throttle actuations) through three PID control interfaces. It also includes a speed control interface by a fuzzy logic controller, which is added inside the hyperkenel application. Each of the three PID control interfaces are CPropertyPage Objects. They are

Accelerator CPropertyPage Object, Brake CPropertyPage Object and Steering CPropertyPage Object respectively.

As Low Level Control System has been selected, the CEdit Control Object, two CButton Objects for saving and loading trajectory files for Path Cloning or Path Following System, as well as six CSlider Objects for tuning the Path following System will be all disenabled by a member function called *EnableWindow()*. The function is often used to distinguish control options between the four different control systems.

3.2.3.1 Accelerator actuation and Speed Control Interfaces



Accelerator Monitor: ( ) PwrOut [✓ AD-Pos] ( ) AD-Spd ( ) Spd-Err ( ) Dis-Err ( ) Ori-Err

( ) Zoom [✓ Normal]

Time Elapsed(ms) : 392714

OvrSht ->	149	PwrOut ->	444.000C
Std-Er ->	110	Ac-Pos ->	1468.00C
ProErr ->	111.00000	De-Pos ->	1579.00C
DevErr ->	0.000000(	AcW-Spd ->	0.00000(
IntErr ->	10002.00	De-Spd ->	0.00000(
AcG-Spd ->	0.000001	EGPS-X ->	0.00000(
Spd Err ->	0.000001	EGPS-Y ->	0.00000(

ACTUATOR

1579

0.000001 km/hr

Fixed Mode
Kp: 4.000001
Ki: 0.000000
Kd: 0.00000C

Kp ( ) x1 ( ) x10 [✓ x100]
Ki [✓ x1] ( ) x10 ( ) x100
Kd [✓ x1] ( ) x10 ( ) x100

( ) GPS-ED ( ) LSR ( ) PATH ( ) n/a ( ) n/a

Figure 3.9: Accelerator actuation and speed control interfaces

Accelerator CPropertyPage Object is the interface of both Accelerator (throttle) PID fine-tuning and Fuzzy-logic Speed Control systems.

As in figure 3.9 it consists of an Accelerator Monitor, an Accelerator Actuator, an Accelerator Control Monitor and a Path Monitoring Windows.

In the Accelerator Monitor Window, there are six radio buttons in the right and two at the bottom. Radio buttons are very commonly used in the button family. It can be seen in most of the applications where selections from a group of similarity are demanded.

A *wnd* member function called *IsDlgButtonChecked()* is often used to return TRUE whenever the user check the certain Control ID which pointing the corresponding Radio button. A Control ID is a identification number representing as a pointer pointing to the address of the dialog control object. It is often in a form of "IDC_EXAMPLE".

The six radio buttons in the right hand side of the Accelerator Monitor Window(figure 3.9) are used to swapping plots among Power output, Actual-Desired actuator Position, Actual-Desired Speed, Speed error in Speed control, Distance error and Orientation error in both Path Following & Path Cloning system from top to bottom respectively. The Bottom two radio buttons are only used when Actual-Desired actuator position plot radio button on the right is checked. It provides the zooming view of the desired set point and actual potential-meter's position reading from sensor, allowing finer tuning of the PID controlled acceleration-actuator system.

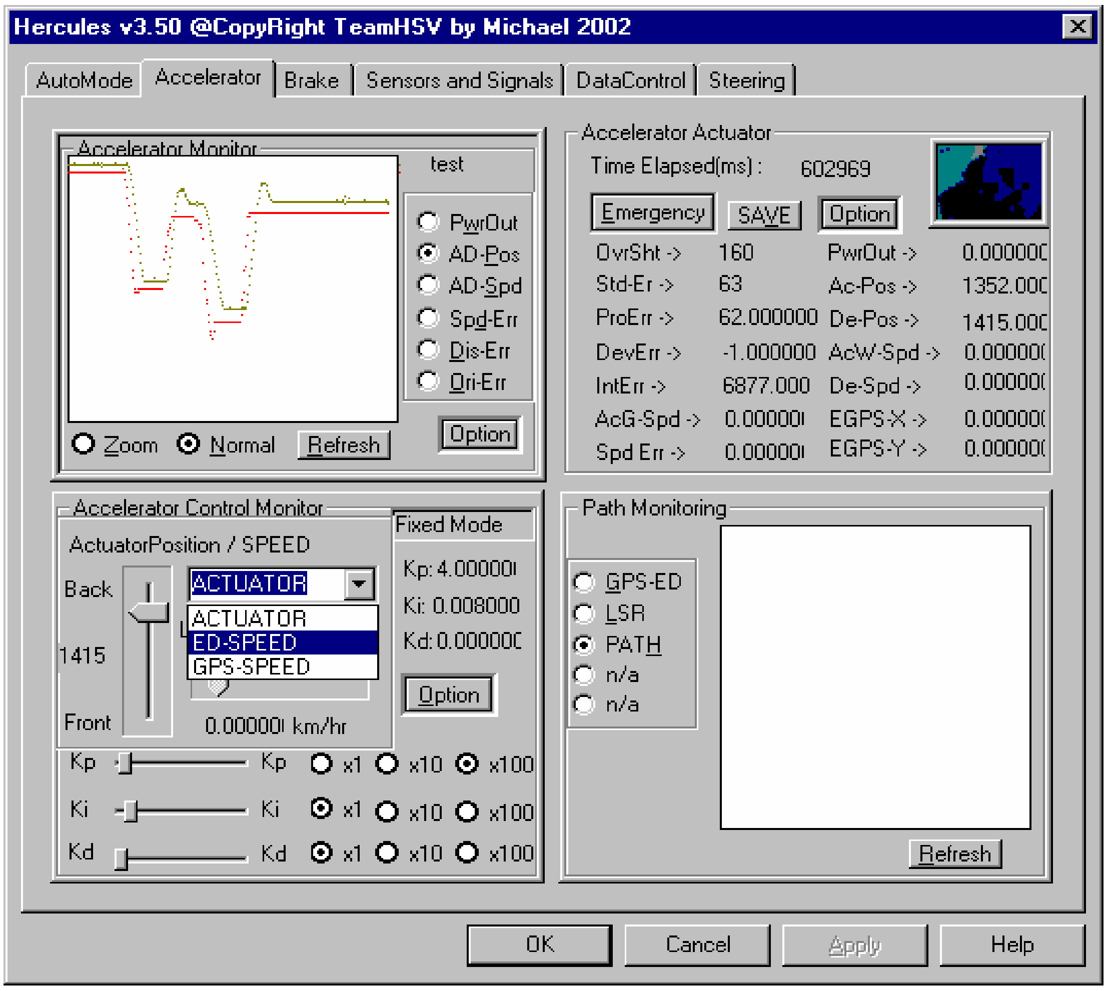
In the Acceleration Actuator Window(figure 3.9) on the right, there shows the current status of the actuator. The readings from left column to the right column are represented as:

Overshoot(counts)	Power Output(counts)
Steady State Error(counts)	Actual Actuator Position(counts)
Proportional Term Error(counts)	Desired Actuator Position(counts)
Derivative term Error(counts)	Actual Speed From Wheel Encoder(km/hr)
Integral Term Error(counts)	Desired Speed(km/hr)
Actual horizontal Speed From GPS(m/s)	Latitude From Encoder GPS(m)
Error in Desire and Actual Speed(km/hr)	Longitude From Encoder GPS(m)

Table 2 Readings arrangement in figure 3.9

In order to display various types of data onto the dialog, an overload member function *IceUpdate()* is created to accept four different types of data. Included data types are integer, long integer, double and text. Double type of data will be firstly combined into a string, then display it on the dialog. *Wnd*-member functions *SetDlgItemInt()* and *SetDlgItemText()* are used for the other types of data.

In the Accelerator Control Monitor Window(figure 3.9), there's a CCombo box object allowing user to select or change among the Actuator, Wheel encoder Speed control and GPS speed Control Interfaces.


Hercules v3.50 @CopyRight TeamHSV by Michael 2002

AutoMode | Accelerator | Brake | Sensors and Signals | DataControl | Steering

Accelerator Monitor
test
( ) PwrOut [✓ AD-Pos] ( ) AD-Spd ( ) Spd-Err ( ) Dis-Err ( ) Ori-Err
( ) Zoom [✓ Normal] Refresh Option

Accelerator Actuator
Time Elapsed(ms) : 602969
Emergency SAVE Option

OvrSht ->	160	PwrOut ->	0.00000C
Std-Er ->	63	Ac-Pos ->	1352.00C
ProErr ->	62.000000	De-Pos ->	1415.00C
DevErr ->	-1.000000	AcW-Spd ->	0.00000(
IntErr ->	6877.000	De-Spd ->	0.00000(
AcG-Spd ->	0.00000I	EGPS-X ->	0.00000(
Spd Err ->	0.00000I	EGPS-Y ->	0.00000(

Accelerator Control Monitor
ActuatorPosition / SPEED
Back
1415
Front
ACTUATOR
ACTUATOR
ED-SPEED
GPS-SPEED
0.00000I km/hr

Fixed Mode
Kp: 4.00000I
Ki: 0.008000
Kd: 0.00000C
Option

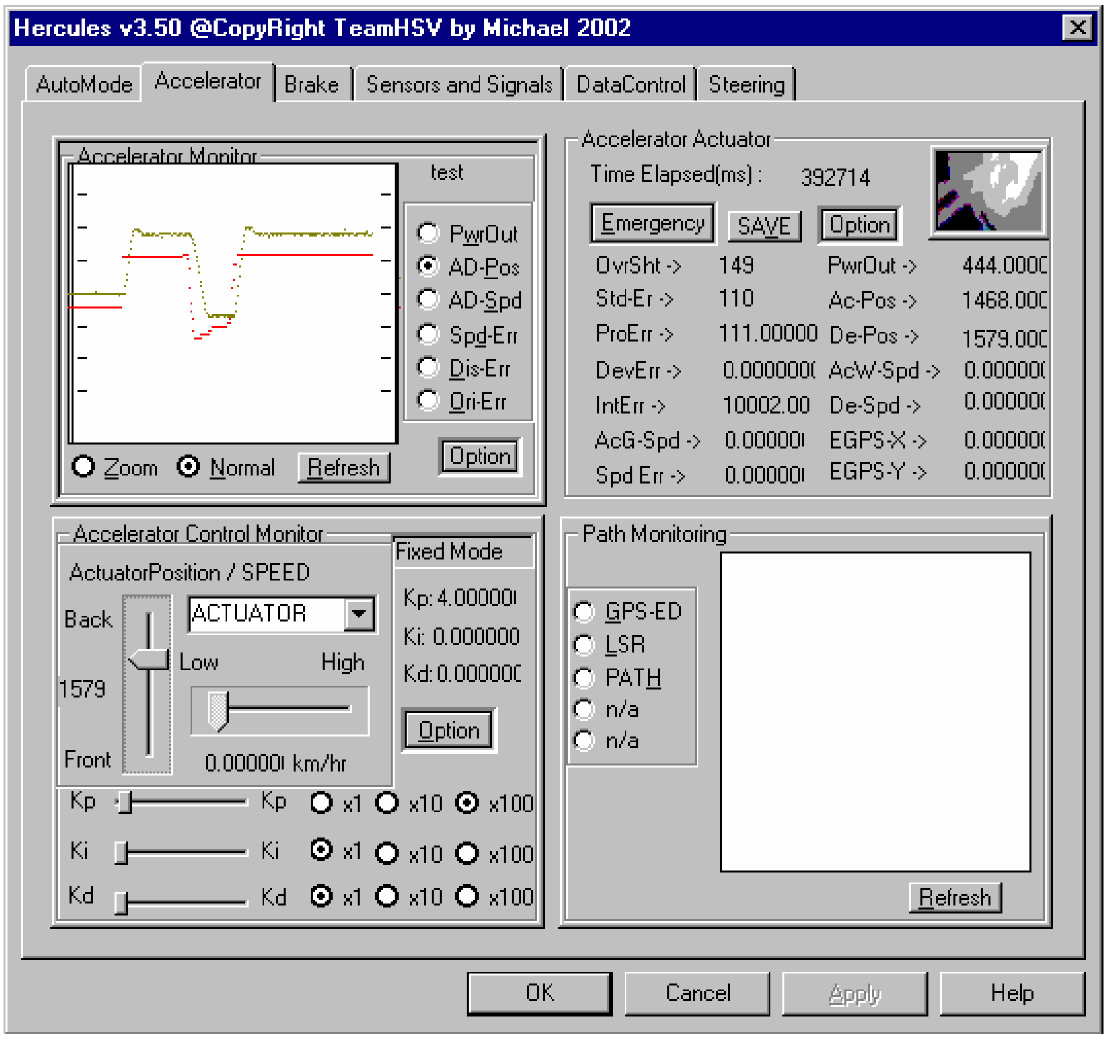
Kp ( ) x1 ( ) x10 [✓ x100]
Ki [✓ x1] ( ) x10 ( ) x100
Kd [✓ x1] ( ) x10 ( ) x100

Path Monitoring
( ) GPS-ED ( ) LSR [✓ PATH] ( ) n/a ( ) n/a
Refresh

OK Cancel Apply Help

Figure 4.0: Speed Control system interface

If the Actuator Control is selected from the Combo Box in figure 4.0, the Slider bar underneath the list box, which is used to control desired speed of the vehicle in terms of Kilo meter per hour, will be disenabled by *EnableWindow()* member function. User can only control the slider bar on the left to control the position of the linear acceleration actuator. The position of the slider bar is put in a shared-memory variable call AcceleratorSliderValue, which is defined in the SharedMemProtocol header file. Like integer variables SteeringSliderValue and BrakeSliderValue, AcceleratorSliderValue controls the set point in the structure PID_B, which is used in the hyper kenel application to calculated the output power required to retract or extend the cylinder in the acceleration linear actuator. The set point in the PID controller indicates the desired position of the cylinder exceeding from the actuator.


Hercules v3.50 @CopyRight TeamHSV by Michael 2002

AutoMode | Accelerator | Brake | Sensors and Signals | DataControl | Steering

Accelerator Monitor
test
PwrOut [✓ AD-Pos] AD-Spd Spd-Err Dis-Err Ori-Err
Zoom [✓ Normal] Refresh
Option

Accelerator Actuator
Time Elapsed(ms) : 392714
Emergency SAVE Option

OvrSht ->	149	PwrOut ->	444.000C
Std-Er ->	110	Ac-Pos ->	1468.00C
ProErr ->	111.00000	De-Pos ->	1579.00C
DevErr ->	0.000000(	AcW-Spd ->	0.00000(
IntErr ->	10002.00	De-Spd ->	0.00000(
AcG-Spd ->	0.000001	EGPS-X ->	0.00000(
Spd Err ->	0.000001	EGPS-Y ->	0.00000(

Accelerator Control Monitor
ActuatorPosition / SPEED
Back
1579
Front
ACTUATOR
Low High
0.000001 km/hr

Fixed Mode
Kp: 4.000001
Ki: 0.000000
Kd: 0.00000C
Option

Kp Kp x1 x10 [✓ x100]
Ki Ki [✓ x1] x10 x100
Kd Kd [✓ x1] x10 x100

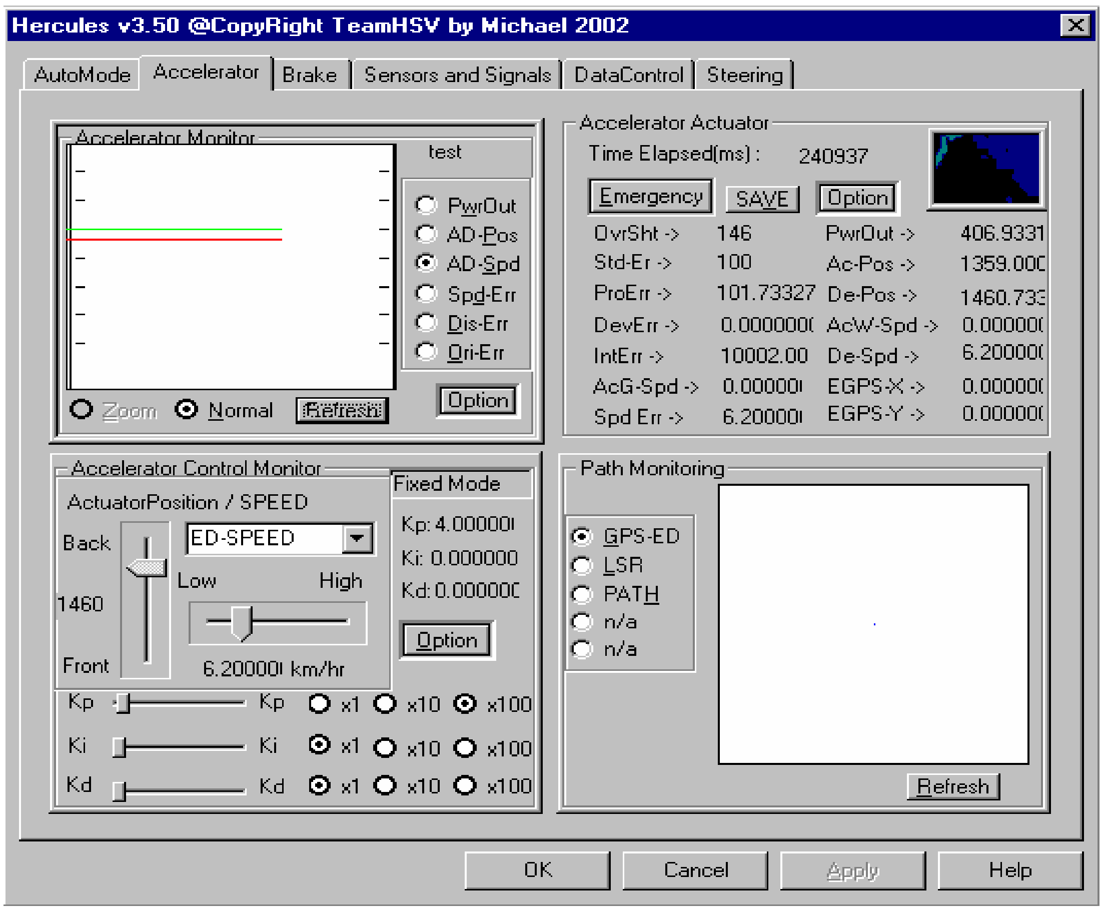
Path Monitoring
GPS-ED LSR PATH n/a n/a
Refresh

OK Cancel Apply Help

Figure 4.1: Accelerator Control Monitor PID controller

Kp, Ki and Kd are controlled in the very similar way by three sliders. There are three radio buttons following each of the three PID gain Sliders in rows in figure 4.1. These are called gain factors and they are used when regions of the gains are needed to fine tuning the actuator system. For instance, when radio button of X1 is checked for Kp, the gain Kp slider bar lies in the range of 0 to 1 with increment of 0.0001; when radio button of X10 is checked, Kp slider value lies between 0 to 10 with increment of 0.001; when radio button of X100 is checked, Kp value lies between 0 to 100 with increment of 0.01 It is done by pre-setting the gain slider range from 0 to 10000, then multiply one of the gain factors as 0.0001, 0.001 and 0.01 in the order of radio button X1, X10 and X100 as one of the button checked. There are corresponded gain readings on the right corner of this window in figure 4.1. It is often wise to pre-check some necessary radio buttons in the *OnInitDialog()* member function when dialog is created at the first instant.

If either Wheel Encoder or GPS Speed Control interfaces option is selected from the Combo list. Actuator position-control Slider will be disenabled by *EnableWindow()* and at the same time, the Desired Speed Control Slider underneath will be enabled.



Hercules v3.50 @CopyRight TeamHSV by Michael 2002

AutoMode | Accelerator | Brake | Sensors and Signals | DataControl | Steering

Accelerator Monitor

test

PwrOut [✓ AD-Spd] AD-Pos Spd-Err Dis-Err Ori-Err

Zoom [✓ Normal] Refresh Option

Accelerator Actuator

Time Elapsed(ms) : 240937

Emergency SAVE Option

OvrSht ->	146	PwrOut ->	406.9331
Std-Er ->	100	Ac-Pos ->	1359.00C
ProErr ->	101.73327	De-Pos ->	1460.733
DevErr ->	0.000000(	AcW-Spd ->	0.00000(
IntErr ->	10002.00	De-Spd ->	6.20000(
AcG-Spd ->	0.000001	EGPS-X ->	0.000000
Spd Err ->	6.200001	EGPS-Y ->	0.00000(

Accelerator Control Monitor

ActuatorPosition / SPEED

Back

1460

Front

ED-SPEED

Low High

6.200001 km/hr

Fixed Mode

Kp: 4.000001

Ki: 0.000000

Kd: 0.00000C

Option

Kp Kp x1 x10 [✓ x100]

Ki Ki [✓ x1] x10 x100

Kd Kd [✓ x1] x10 x100

Path Monitoring

[✓ GPS-ED] LSR PATH n/a n/a

Refresh

OK Cancel Apply Help

Figure 4.2 Speed Control Interface

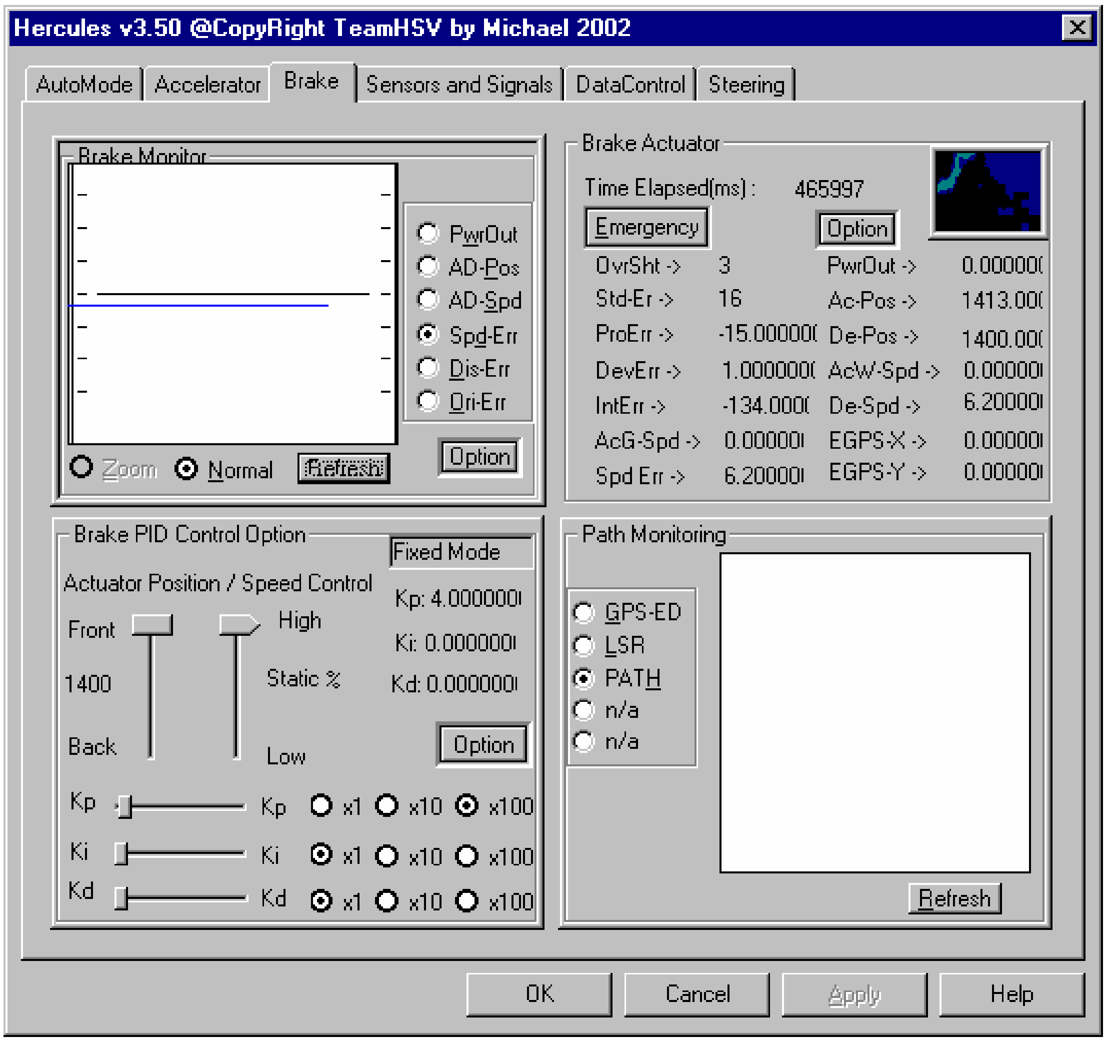
The Desired Speed Control Slider controls the desired speed sending into the Fuzzy logic controller in Hyperkernel application through the shared memory as a variable named

DesSpeed. The Shared- memory Variable DesSpeed is used in the Fuzzy Logic Control loop to compare with either wheel encoder or GPS speed reading in Kilo meter per hour to calculate a desired position set-point passing down to either one of the PID structure for brake or accelerator actuation. It is then up to each of the two PID control loops to decide the final power output sent to their actuators. At the same time, the resulted two set-points from fuzzy logic controller are stored in the shared memory and retrieved back to this Speed control interface as variables SetAccelerator and SetBrake. These two Set Points are used to set back the position of the slider which controls the set point of the actuations before. It results in two actuation sliders moves automatically according to their feed back set point of the PID actuations' control. Error of the desired speed and actual speed are also sent back through the shared memory for status reading and on-line plotting proposes.

Whenever a speed control is under going, all Slider value from both brake and acceleration actuators' Sliders will lose controls to the set points in the PID structures by the Index returned from *CCombo::GetCurSel()* member function. In another words, when Speed control is engaged, set points that sent to these two actuations' PID structures, will be fully determined by the Fuzzy Logic Controller.

3.2.3.2 Brake actuation Control Interface

The Brake Control interface(Figure 4.3) is very similar to accelerator control interface. It provides user a PID tuning system to the user for fine tuning the PID responses of the actuation.



Hercules v3.50 @CopyRight TeamHSV by Michael 2002

AutoMode | Accelerator | Brake | Sensors and Signals | DataControl | Steering

Brake Monitor

PwrOut AD-Pos AD-Spd [✓ Spd-Err] Dis-Err Ori-Err

Zoom [✓ Normal] Refresh Option

Brake Actuator

Time Elapsed(ms) : 465997

Emergency Option

OvrSht ->	3	PwrOut ->	0.00000(
Std-Er ->	16	Ac-Pos ->	1413.00(
ProErr ->	-15.00000(	De-Pos ->	1400.00(
DevErr ->	1.000000(	AcW-Spd ->	0.00000I
IntErr ->	-134.000(	De-Spd ->	6.20000I
AcG-Spd ->	0.00000I	EGPS-X ->	0.00000I
Spd Err ->	6.20000I	EGPS-Y ->	0.00000I

Brake PID Control Option

Fixed Mode

Actuator Position / Speed Control

Front High

1400 Static %

Back Low

Kp: 4.000000I

Ki: 0.000000I

Kd: 0.000000I

Option

Kp — Kp x1 x10 [✓ x100]

Ki — Ki [✓ x1] x10 x100

Kd — Kd [✓ x1] x10 x100

Path Monitoring

GPS-ED LSR [✓ PATH] n/a n/a

Refresh

OK Cancel Apply Help

Figure 4.3: Brake Control Interface

In all three of the actuations control interfaces, overshoots and steady state error are to be computerized as display in the actuator monitoring windows in the right top corner. Overshoot is calculated when actual positions of the actuators exceed the set-point and it is recorded as the maximum peak value exceeding a set-point. Each time when a new set-point appear, overshoot will be washed away. It is similar to the way steady state error is monitored. A slower timer is created to record a steady state error when previous state error is identical to the current state error.

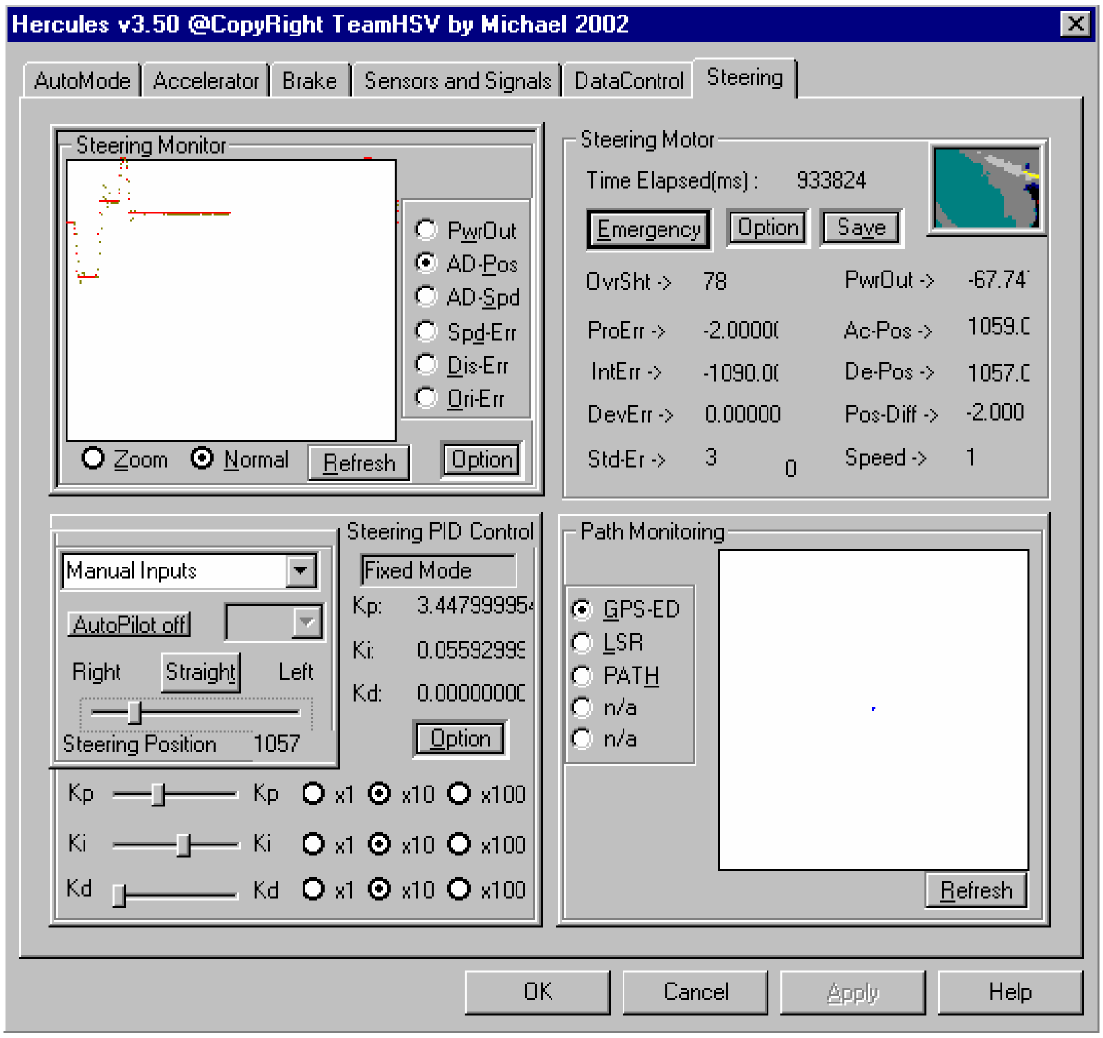
3.2.3.3 Steering actuation Control Interface

The Steering control interface is the mostly used interface through out the year.

It has similar features on the actuation position plots, speed plots and GPS plots like other *CPropertyPage* objects have. It provides the most basic manual inputs function that allow user to input a steering position by a slider located in top half of the Steering PID Control Window. The statuses of the steering actuation are displayed inside the Steering Motor Window. In order, the readings are represented as shown in table 3.

Overshoot(counts)	Power output(counts)
Proportional term of Error(counts)	Actual Position of the Steering(counts)
Integral term of Error(counts)	Desired Position of the Steering(counts)
Derivative term of Error(counts)	Steering Position error(counts)
Steady State Error(counts)	Wheel encoder Speed(km/hr)

Table 3: Steering control interface readings



PwrOut [✓ AD-Pos] AD-Spd Spd-Err Dis-Err Ori-Err

Zoom [✓ Normal]

Time Elapsed(ms) : 933824

OvrSht -> 78
ProErr -> -2.0000(
IntErr -> -1090.0(
DevErr -> 0.00000
Std-Er -> 3 0
PwrOut -> -67.74'
Ac-Pos -> 1059.C
De-Pos -> 1057.C
Pos-Diff -> -2.000
Speed -> 1

Manual Inputs

Steering Position 1057

Fixed Mode
Kp: 3.44799995
Ki: 0.05592999
Kd: 0.0000000C

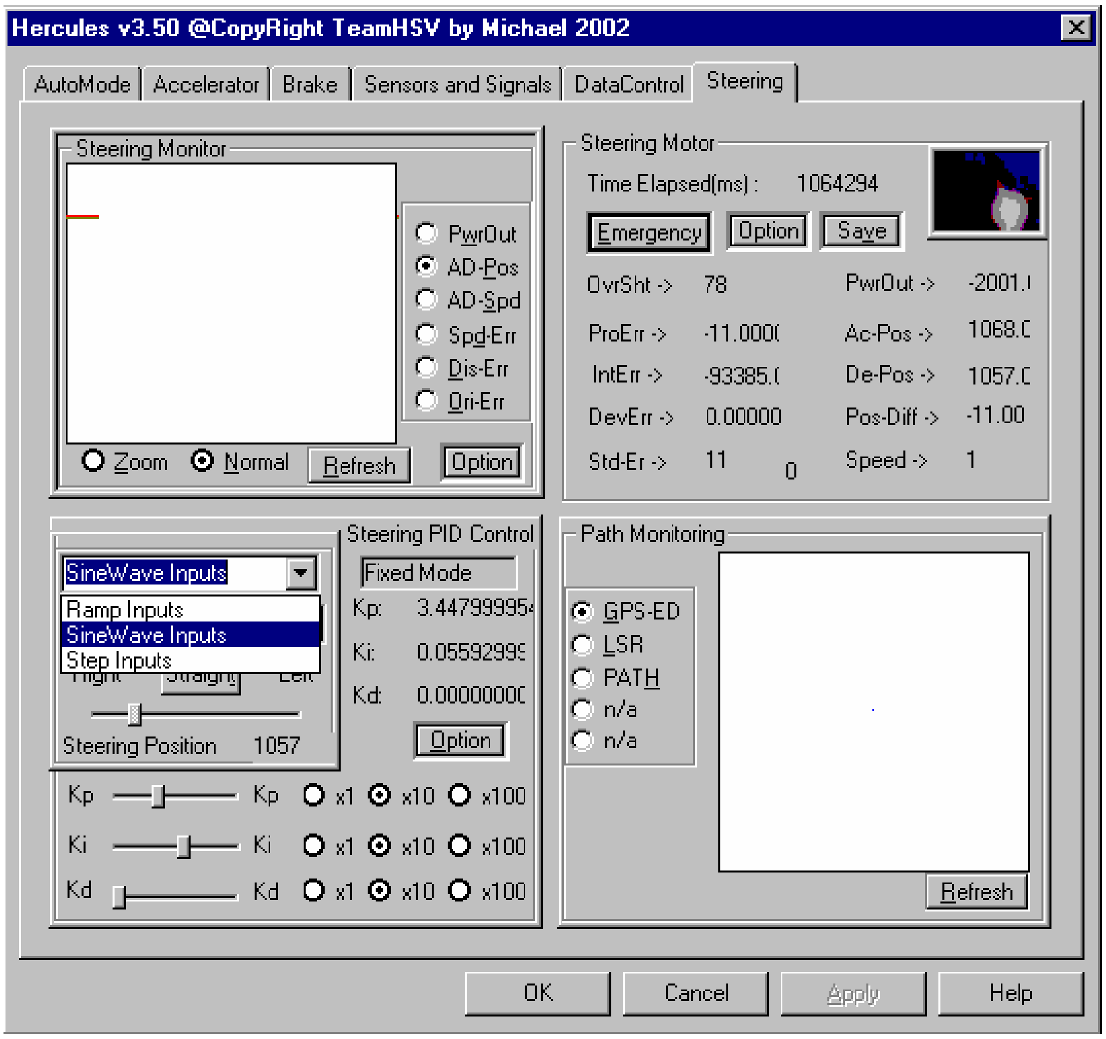
Kp x1 [✓ x10] x100
Ki x1 [✓ x10] x100
Kd x1 [✓ x10] x100

[✓ GPS-ED] LSR PATH n/a n/a

Figure 4.4 Steering Control Interface

The Combo box in the Steering control interface in figure 4.4 can be dropped down to select other control option for steering. Notice that the rest of the options are for Auto Pilot system. It includes auto steps inputs, auto sinusoidal inputs and auto ramp inputs.

The Automatic Pilot System in Steering Control interface is designed for the ease of tuning the gains while the desired set-points to the PID controller are set automatically.



Steering Monitor: PwrOut [✓ AD-Pos] AD-Spd Spd-Err Dis-Err Ori-Err
Zoom [✓ Normal]

Time Elapsed(ms) : 1064294

OvrSht ->	78	PwrOut ->	-2001.I
ProErr ->	-11.000(	Ac-Pos ->	1068.C
IntErr ->	-93385.(	De-Pos ->	1057.C
DevErr ->	0.00000	Pos-Diff ->	-11.00
Std-Er ->	11 0	Speed ->	1

SineWave Inputs
Ramp Inputs
SineWave Inputs
Step Inputs
Steering Position 1057

Fixed Mode
Kp: 3.44799995
Ki: 0.05592999
Kd: 0.0000000C

Kp: x1 [✓ x10] x100
Ki: x1 [✓ x10] x100
Kd: x1 [✓ x10] x100

Path Monitoring: [✓ GPS-ED] LSR PATH n/a n/a

Figure 4.5: auto pilot steering control system

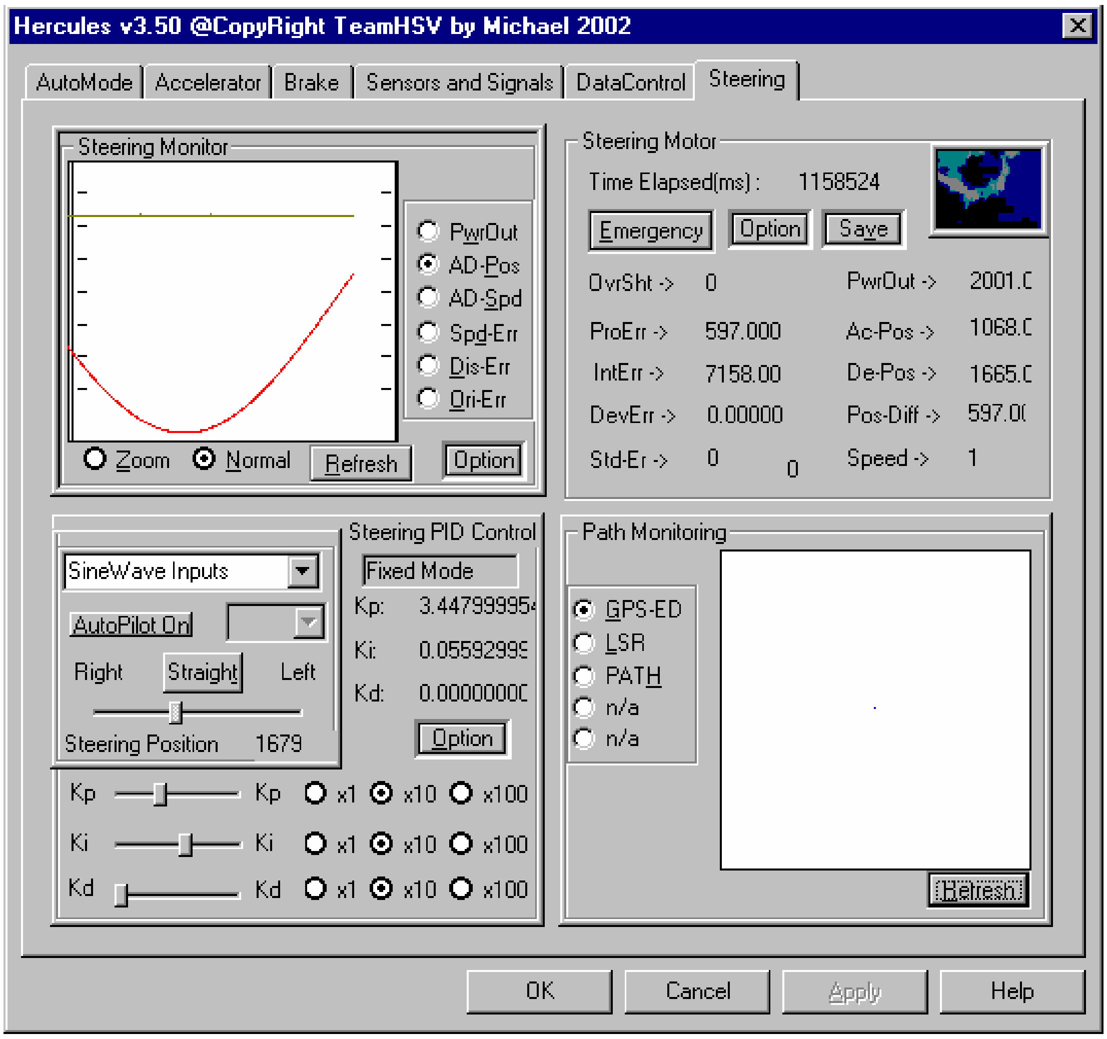
Since most of the tests demand the same set-point inputs every time. During the speed and steering experiments, the auto pilot system played a very important role on producing same increments every six second of system elapsed time. The *Save* button in the top right corner of the Steering Monitor Window allows user to save Overshoots, Steady state error and corresponding Kp, Ki, Kd gains. The matrix *.mat file will be saved in an out-files folder for the propose of offline analysis.

The matrix file will consists of data in column as shown in table 4 below.

System time(ms)	Overshoot (counts)	Steady State Error(counts)	Kp	Ki	Kd	Setpoint (counts)	LVDT Reading (counts)	Power out(counts)

Table 4: Steering data saved

In the automatic sinusoidal input mode, set-point of the steering will be generated as a continuous sine wave pattern in figure 4.6.


Hercules v3.50 @CopyRight TeamHSV by Michael 2002

AutoMode | Accelerator | Brake | Sensors and Signals | DataControl | Steering

Steering Monitor: PwrOut [✓ AD-Pos] AD-Spd Spd-Err Dis-Err Ori-Err; Zoom [✓ Normal]

Steering Motor
Time Elapsed(ms) : 1158524

OvrSht -> 0 PwrOut -> 2001.0
ProErr -> 597.000 Ac-Pos -> 1068.0
IntErr -> 7158.00 De-Pos -> 1665.0
DevErr -> 0.00000 Pos-Diff -> 597.0
Std-Er -> 0 0 Speed -> 1

SineWave Inputs
Steering Position 1679

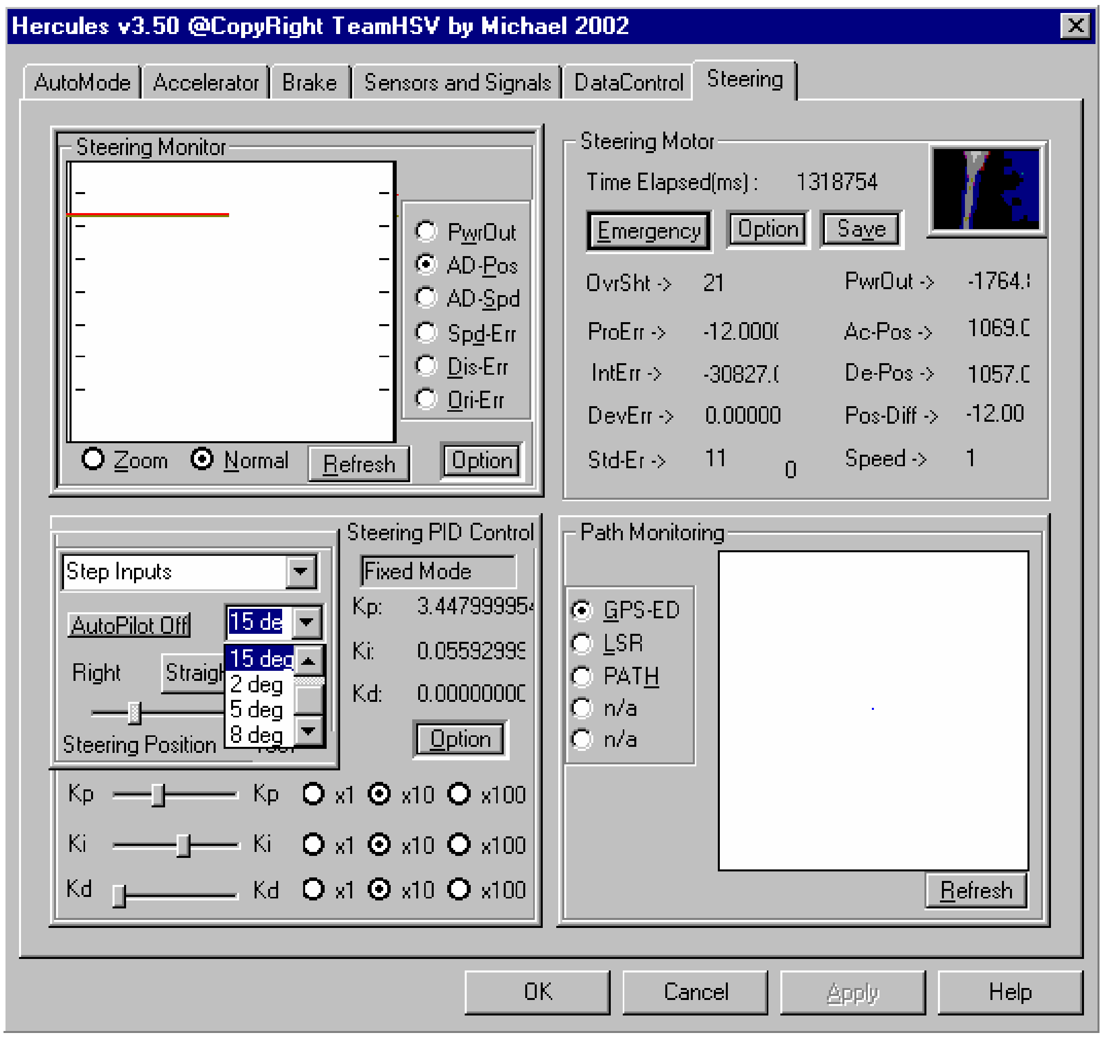
Steering PID Control
Fixed Mode
Kp: 3.44799995
Ki: 0.05592999
Kd: 0.0000000

Kp: x1 [✓ x10] x100
Ki: x1 [✓ x10] x100
Kd: x1 [✓ x10] x100

Path Monitoring: [✓ GPS-ED] LSR PATH n/a n/a

Figure 4.6: Automatic sine wave inputs

In Auto Step inputs mode in figure 4.7, user will be asked to select a step increment from the Combo list at the right. System then provides the same increment as the steering set-point every 6 seconds.



Steering Monitor: PwrOut [✓ AD-Pos] AD-Spd Spd-Err Dis-Err Ori-Err
Zoom [✓ Normal]

Time Elapsed(ms) : 1318754

OvrSht ->	21	PwrOut ->	-1764.!
ProErr ->	-12.000(	Ac-Pos ->	1069.C
IntErr ->	-30827.(	De-Pos ->	1057.C
DevErr ->	0.00000	Pos-Diff ->	-12.00
Std-Er ->	11 0	Speed ->	1

Step Inputs
AutoPilot Off
15 de
15 deg
2 deg
5 deg
8 deg
Right Straigh
Steering Position

Fixed Mode
Kp: 3.44799995‹
Ki: 0.05592999
Kd: 0.0000000C

Kp x1 [✓ x10] x100
Ki x1 [✓ x10] x100
Kd x1 [✓ x10] x100

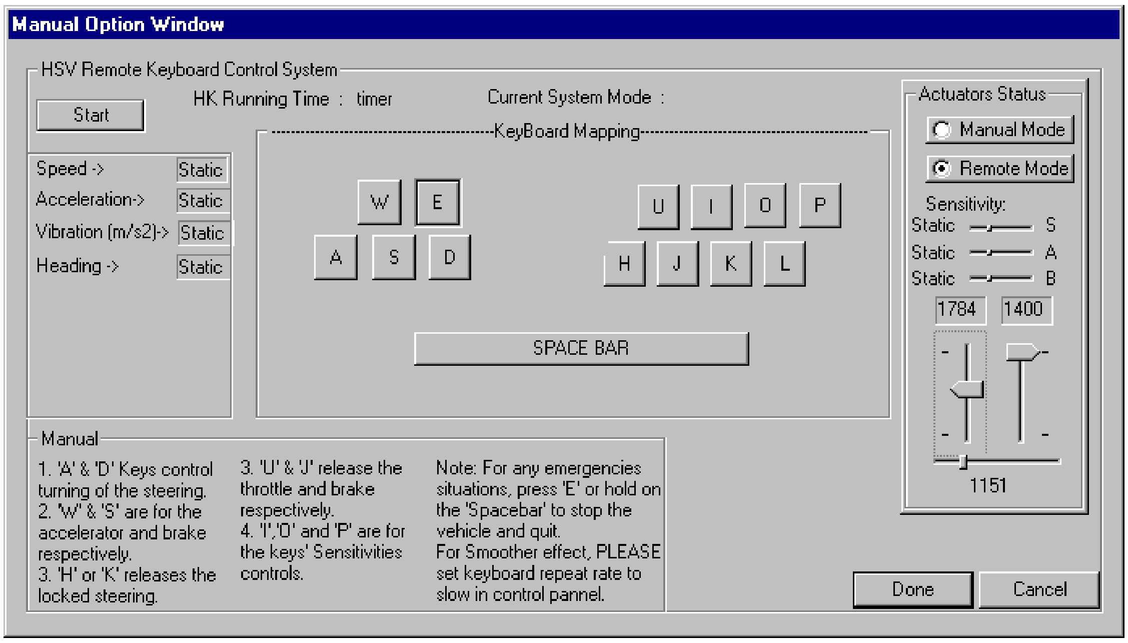
[✓ GPS-ED] LSR PATH n/a n/a

Figure 4.7 Auto step inputs mode

Note that the emergency buttons on the right top corner will immediate stop the power generated by PID control system by shutting down all the gains to zero. Unless emergency button is to be pressed again, any running experiments will not be able to be continued.

3.2.4 Keyboard Remote Control System

Keyboard Remote Control system


Manual Option Window

HSV Remote Keyboard Control System

HK Running Time : timer

Current System Mode :

Speed -> Static

Acceleration-> Static

Vibration (m/s2)-> Static

Heading -> Static

Actuators Status

( ) Manual Mode

(•) Remote Mode

Sensitivity:

Static S

Static A

Static B

1784 1400

1151

Manual

1. 'A' & 'D' Keys control turning of the steering.
2. 'W' & 'S' are for the accelerator and brake respectively.
3. 'H' or 'K' releases the locked steering.
3. 'U' & 'J' release the throttle and brake respectively.
4. 'I','O' and 'P' are for the keys' Sensitivities controls.

Note: For any emergencies situations, press 'E' or hold on the 'Spacebar' to stop the vehicle and quit.
For Smoother effect, PLEASE set keyboard repeat rate to slow in control pannel.

Figure 4.8 Keyboard Remote Control System

The keyboard Remote control system in figure 4.8 is through pressing down the Keys that correspond to the actuation set point control, to achieve the full low level controls(including speed control) of the vehicle.

It is done by using the *GetAsyncKeyState() wnd* member function that reads the asynchronous state of the a pressed key. The function takes a virtual key code as its argument and returns a value indicates its according address in the key map when the key on the keyboard is pressed. If there's nothing pressed, *GetAsyncKeyState()* returns 0 in response. Therefore detecting whether the function returns zero or not zero will be the way to control actuations through a keyboard. As user holding down a control key on the keyboard, an increment will be added towards the set-point of the actuation it tends to control. Increment will be set to zero when user releases the control key, and the actuation position will stop at where it is unless a particular key is pressed to set actuation back to the beginning simulating the spring effects in brake and throttle paddles as well as in steering wheel shaft. The increments to the set-points are known as the sensitivities in control.

For instance, if the sensitivity of a button controlling the steering motion is high, that means the increment each time the program detect a keyboard interrupt when the key is pressed will be high as well. That results a faster response of the actuation.

The sensitivities can be changed either through three sliders at the top right corner in the interface, or through pressing down three keys on the keyboard as well. Three sliders below represent the current positions of three actuations.

Keyboard Remote Control Interface (figure 4.8)provides on-board user manual and two *Emergency* for any unexpected situations.

3.2.5 Path following and Path Cloning Control System

3.2.5.1 Introduction to Path following and tracking system

Path following also known as trajectory following is to control the vehicle following a given path or a trajectory under a constant velocity. The Path will be mainly considered to be consisting of x, y position and the vehicle's orientation angle. Path tracking on the other hand involves the fully tracking of an accurate path in terms of position, orientation as well as speed. For example, a path from St. John College to ACFR building has been set, requiring the vehicle to reach the destination in 20 minutes. In this case, speed no longer to be considered as constant but as a factor or variable plotted into the path planning process. To achieve this, a very accurate path following system need to be fine tuned.

Unlike the simulations in mathlab, in the reality the speed of the vehicle hardly to be maintained as a perfect constant and slightly variation in speed will cause unwanted path overshoots and steady state errors between true path and desired path. Therefore the second stage in path following might need to involve speed control in the process. For example using a fuzzy or neural fuzzy logic speed controller to maintain and control a speed can be essential.

The Clone Following system will take the 4th variable of speed into the path following process. It will

Both of the control systems require the communication to the high level control algorithm in either mathlab or C/C++ environment. The path following control system consists of a PID controller tuning the performance of the vehicle following a given trajectory. There are two ways of getting a trajectory the vehicle is able to follow. The trajectory is either prepared from a path planning algorithm in high level control, or pre-recorded by an active navigation loop instead of making from high level control. Since the navigation loop such as encoder GPS loop filters away noises from a GPS position reading even when the vehicle can not be covered by satellites. It is more likely to record down a very smooth and reasonable path the vehicle is capable of reproducing. Because of the limitation in space the vehicle faces during a fine-tuning progress of the PID-controlled Path following system, it is wise to record down a trajectory within the limited space and reproduce it after.

3.2.5.2 Path Following Control Algorithm

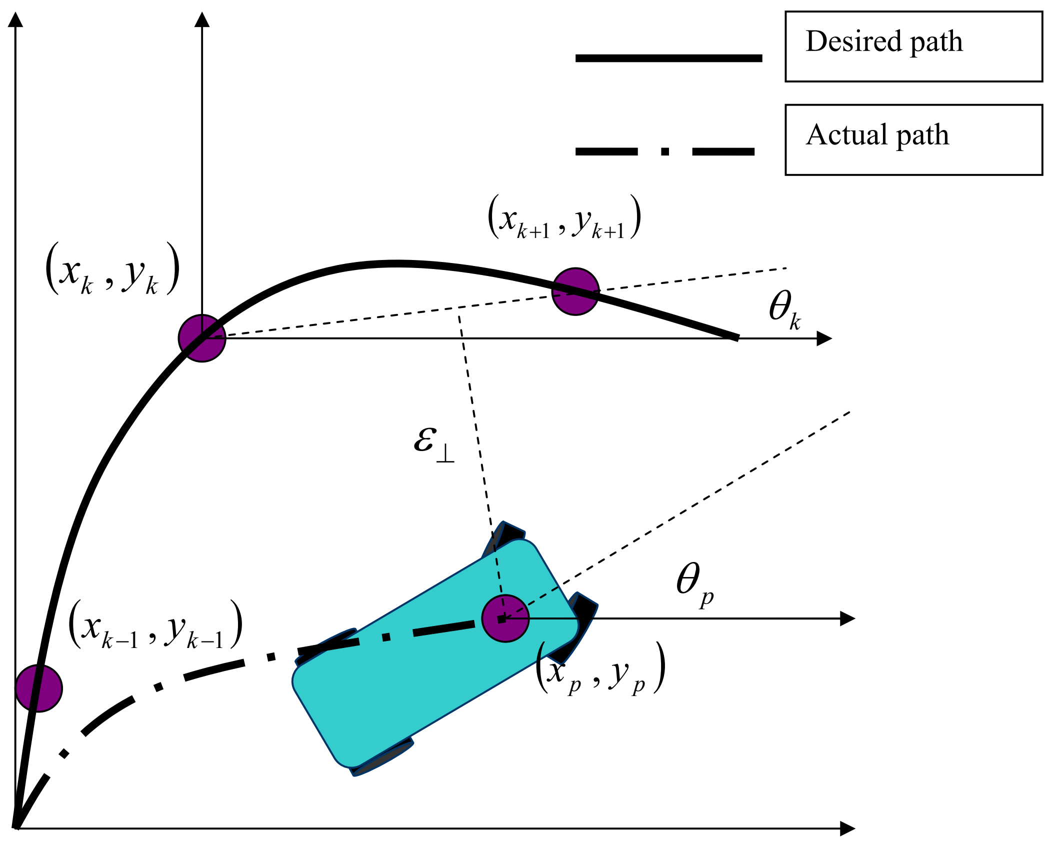


Figure 4.9: Path Following Control Algorithm

As mentioned in Matthew J. Barton's thesis 2001, if we can specify that the path or the trajectory to be followed as a series of discrete Cartesian coordinates at regular(around 0.5 m in length) intervals, it then can be linearized by using "a series of short line segments" technique. Taking (x_k, y_k) as the current position on the desired path and (x_{k-1}, y_{k-1}), (x_{k+1}, y_{k+1}) are representing the previous and next state of desired positioning the linearized path.

The line equation can be given by:

$$(\frac{y_{k+1}-y_k}{x_{k+1}-x_k})x - y + y_k - (\frac{y_{k+1}-y_k}{x_{k+1}-x_k})x_k = 0 \qquad (2.4)$$

Hence the relationship between current position on the actual path and the desired position on the desired path can be established. As shown in the figure above, if $\varepsilon_\perp$ is the error in the perpendicular distance from actual position to the desired position and (x_p, y_p) is the current position of the vehicle, the error in distance can be calculated as

$$\varepsilon_\perp = \frac{|a_k x_p + b_k y_p + c_k|}{\sqrt{a_k{}^2 + b_k{}^2}} \qquad (2.5)$$

Where

$$a_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} \qquad (2.6)$$

$$b_k = -1 \qquad (2.7)$$

$$c_k = y_k - \frac{y_{k+1}-y_k}{x_{k+1}-x_k} \times x_k \qquad (2.8)$$

If the orientation angle at the current position of the vehicle to the x axis is θ_p and orientation angle at the desired position is θ_k, the error in orientation can be calculated as,

$$\varepsilon_\theta = \theta_p - \theta_k \qquad (2.9)$$

A PID control algorithm often takes integral and derivation terms of error into the calculation of the output as showing below.

$$m = k_p\varepsilon + k_i \int \varepsilon dt + k_d \frac{d\varepsilon}{dt}$$

$$\Delta m = k_p(\varepsilon - \varepsilon_{-1}) + k_i\varepsilon + k_d(\varepsilon - \varepsilon_{-1} + \varepsilon_{-2})$$

$$m = m_{-1} + \Delta m \qquad (3.0)$$

A PID controller for the path following system should be a total of two PID control algorithm in terms of both error in distance and error in orientation.

The change in calculated steering angle $\Delta\phi_{spo\,\mathrm{int}}$ in terms of previous state of the both the errors $\varepsilon_{\perp_{-1}}$, $\varepsilon_{\theta_{-1}}$ and previous previous state of the errors $\varepsilon_{\perp_{-2}}$, $\varepsilon_{\theta_{-2}}$ and current state of the errors $\varepsilon_{\theta}, \varepsilon_{\perp}$ can be calculated as below:

$$\begin{aligned}\Delta\phi_{spo\,\mathrm{int}} = \; & [\pm k_{p\perp}(\varepsilon_{\perp} - \varepsilon_{\perp_{-1}}) \pm k_{i\perp}\varepsilon_{\perp} \pm k_{d\perp}(\varepsilon_{\perp} - \varepsilon_{\perp_{-1}} + \varepsilon_{\perp_{-2}})] \\ & + [\pm k_{p\theta}(\varepsilon_{\theta} - \varepsilon_{\theta_{-1}}) \pm k_{i\theta}\varepsilon_{\theta} \pm k_{d\theta}(\varepsilon_{\theta} - \varepsilon_{\theta_{-1}} + \varepsilon_{\theta_{-2}})]\end{aligned} \tag{3.1}$$

It is then added to the previous steering to form the current steering angle

$$\phi_{spo\,\mathrm{int}} = \phi_{spo\,\mathrm{int}_{-1}} + \Delta\phi_{spo\,\mathrm{int}}$$

Note that the sign $\pm$ in the equation are determined by which sides from the desired position the actual vehicle is on.

From the study of Matthew J. Barton's mathlab codes, it seems that the sign taken into the equation above can be calculated by two sign of coefficient matrices. One is for distance error and the other one is for the orientation error.

$$Sign\_matrix_{\perp} = \begin{bmatrix} -1 & 1 & 1 & 1 \\ -1 & -1 & 1 & -1 \end{bmatrix} \tag{3.2}$$

$$Sign\_matrix_{\theta} = \begin{bmatrix} -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \tag{3.3}$$

$$R = \frac{1}{2^{sign\ of\,\perp}} \tag{3.4}$$

$$C = \frac{2}{2^{sign\ of\Delta\theta}} - \left|\frac{\Delta\theta}{\pi}\right| \tag{3.5}$$

Where $\perp$ and $\Delta\theta$ are distance error and orientation error with its own sign

$$Sign_{\perp} = Sign\_matrix_{\perp}(R, C) \tag{3.6}$$

$$Sign_{\theta} = Sign\_matrix_{\theta}(R, C) \tag{3.7}$$

After checking if sign of the coeff. requires swapping, the signs can be put into the equation before to calculated a correct steering angle.

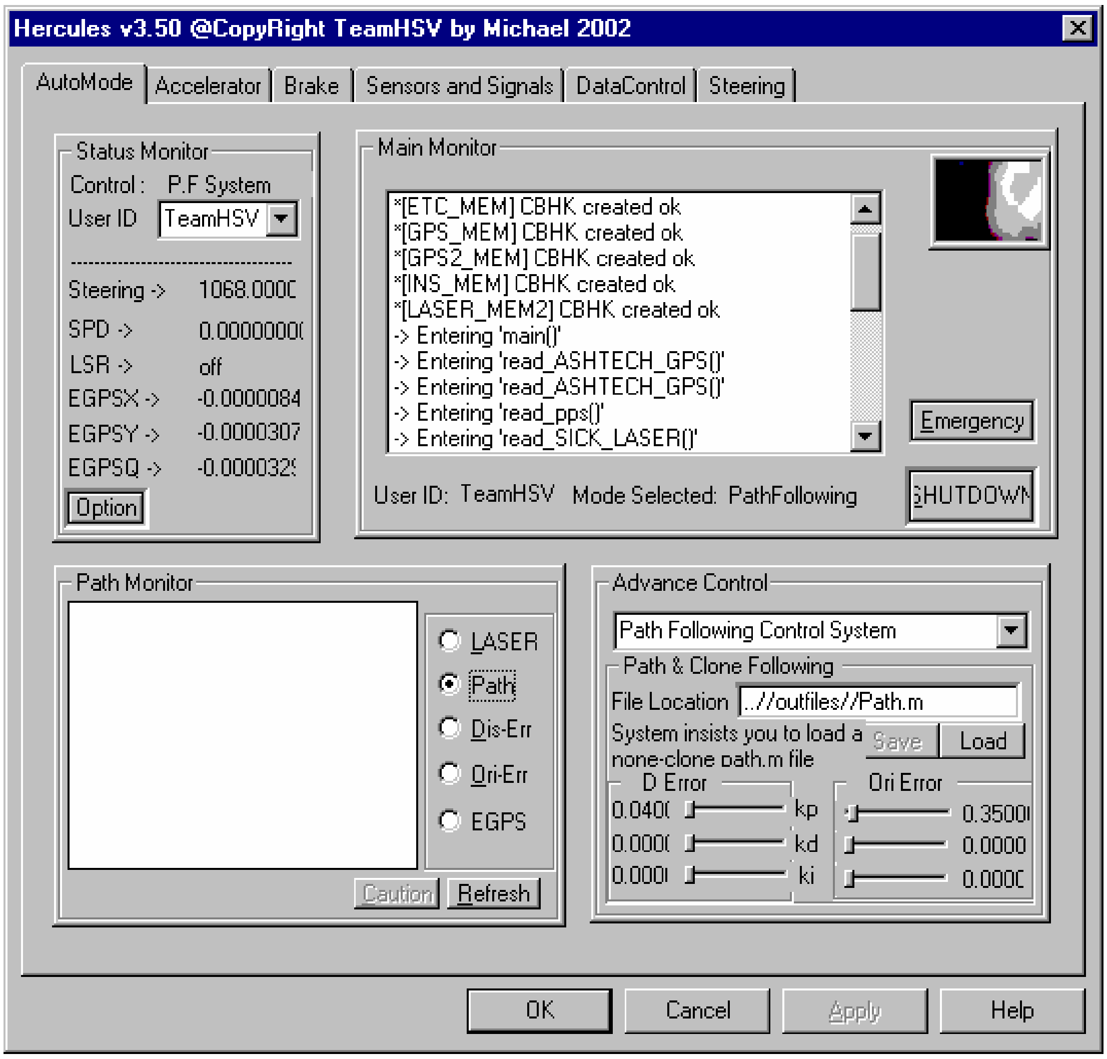
3.2.5.3 Path Following Control Interface

The Path following and Clone following control loops are implemented in the software and needed to be updated and fixed before any out door experiments. In the path following Control system interface, user needs to load a prepared matrix *.m file which only consists of 3 columns of data in table 5.

Latitude(m)	Longitude(m)	Orientation(radius)

Table 5: path following loading data

When the path following control system is engaged, the tuning tools below will be enabled to allow online tuning process. Path plot as well as distance and orientation plots will monitor the tuning progress. In a path following progress, user can swap back to any of three actuation control interfaces to observe and analysis the responses of actuations. All actuations' PID tuning tools will be disenabled by *EnableWindow()* member function during the path tuning process.



Control : P.F System
User ID TeamHSV

Steering ->	1068.000C
SPD ->	0.0000000(
LSR ->	off
EGPSX ->	-0.0000084
EGPSY ->	-0.0000307
EGPSQ ->	-0.000032!

```
*[ETC_MEM] CBHK created ok
*[GPS_MEM] CBHK created ok
*[GPS2_MEM] CBHK created ok
*[INS_MEM] CBHK created ok
*[LASER_MEM2] CBHK created ok
-> Entering 'main()'
-> Entering 'read_ASHTECH_GPS()'
-> Entering 'read_ASHTECH_GPS()'
-> Entering 'read_pps()'
-> Entering 'read_SICK_LASER()'
```

User ID: TeamHSV Mode Selected: PathFollowing

Path Following Control System

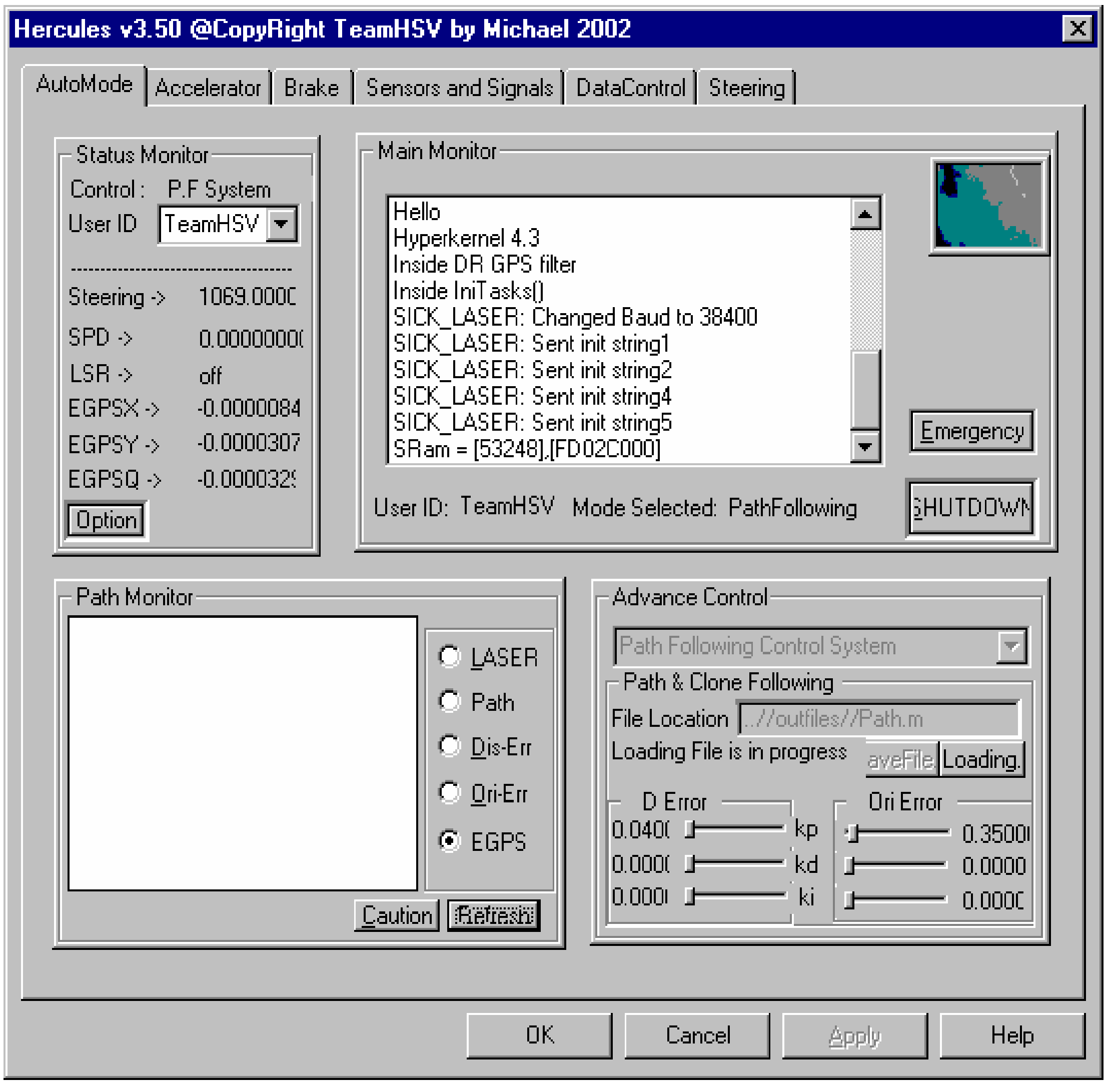
File Location ..//outfiles//Path.m

System insists you to load a none-clone path.m file

D Error		Ori Error
0.040(	kp	0.3500I
0.000(	kd	0.0000
0.000I	ki	0.000C

Figure 5.0: Path Following Control interface

User needs to specify the path file and its location in *File Location* slot. Error will be detected and message will be displayed if path file can not be found. If file is found, loading will be in progress. Data parsed from *Path.m* file will become the desired trajectory and vehicle will start to follow it through a doubled PID control loop.


Hercules v3.50 @CopyRight TeamHSV by Michael 2002

AutoMode | Accelerator | Brake | Sensors and Signals | DataControl | Steering

Status Monitor
Control : P.F System
User ID TeamHSV

Steering -> 1069.000C
SPD -> 0.0000000(
LSR -> off
EGPSX -> -0.0000084
EGPSY -> -0.0000307
EGPSQ -> -0.000032!
Option

Main Monitor
```
Hello
Hyperkernel 4.3
Inside DR GPS filter
Inside IniTasks()
SICK_LASER: Changed Baud to 38400
SICK_LASER: Sent init string1
SICK_LASER: Sent init string2
SICK_LASER: Sent init string4
SICK_LASER: Sent init string5
SRam = [53248],[FD02C000]
```
User ID: TeamHSV Mode Selected: PathFollowing
Emergency
SHUTDOWN

Path Monitor
LASER
Path
Dis-Err
Ori-Err
[✓ EGPS]
Caution
Refresh

Advance Control
Path Following Control System
Path & Clone Following
File Location ..//outfiles//Path.m
Loading File is in progress aveFile Loading.

D Error		Ori Error
0.040(	kp	0.3500(
0.000(	kd	0.0000
0.000(	ki	0.000C

OK Cancel Apply Help

Figure 5.1: path following loading in progress

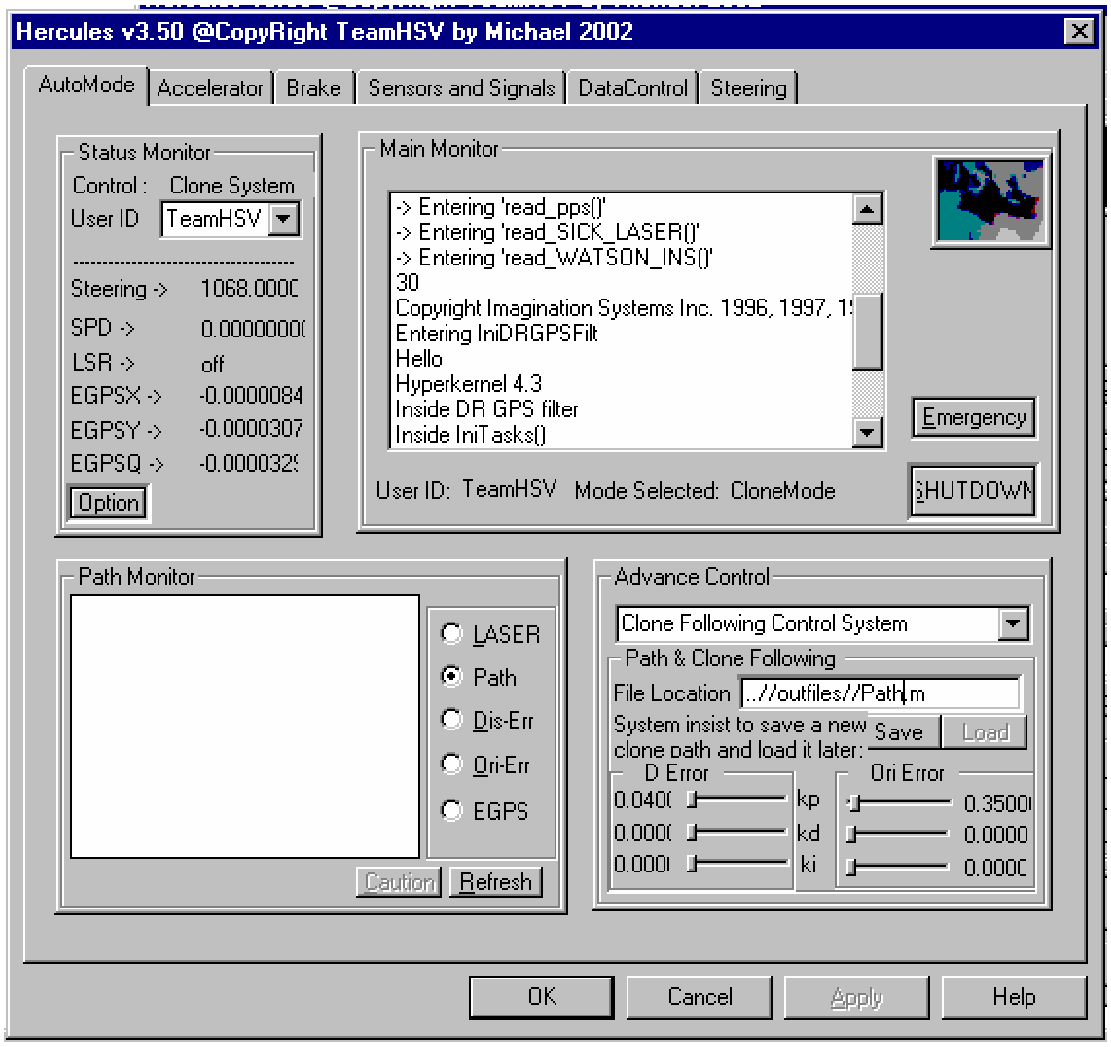
3.2.5.4 Clone Following Control Interface

The clone system allows user to save a path in terms of position as well as speed.

Latitude(m)	Longitude(m)	Orientation(radius)	Speed(m/s)

Table 6: Data saved in clone following interface

Before it starts to save the data, GPS origin needs to be to set to zero.



Hercules v3.50 @CopyRight TeamHSV by Michael 2002

AutoMode | Accelerator | Brake | Sensors and Signals | DataControl | Steering

Status Monitor

Control : Clone System

User ID TeamHSV

Steering ->	1068.000C
SPD ->	0.0000000(
LSR ->	off
EGPSX ->	-0.0000084
EGPSY ->	-0.0000307
EGPSQ ->	-0.000032!

Option

Main Monitor

```
-> Entering 'read_pps()'
-> Entering 'read_SICK_LASER()'
-> Entering 'read_WATSON_INS()'
30
Copyright Imagination Systems Inc. 1996, 1997, 1!
Entering IniDRGPSFilt
Hello
Hyperkernel 4.3
Inside DR GPS filter
Inside IniTasks()
```

User ID: TeamHSV Mode Selected: CloneMode

Emergency

SHUTDOWN

Path Monitor

LASER [✓ Path] Dis-Err Ori-Err EGPS

Caution Refresh

Advance Control

Clone Following Control System

Path & Clone Following

File Location ..//outfiles//Path.m

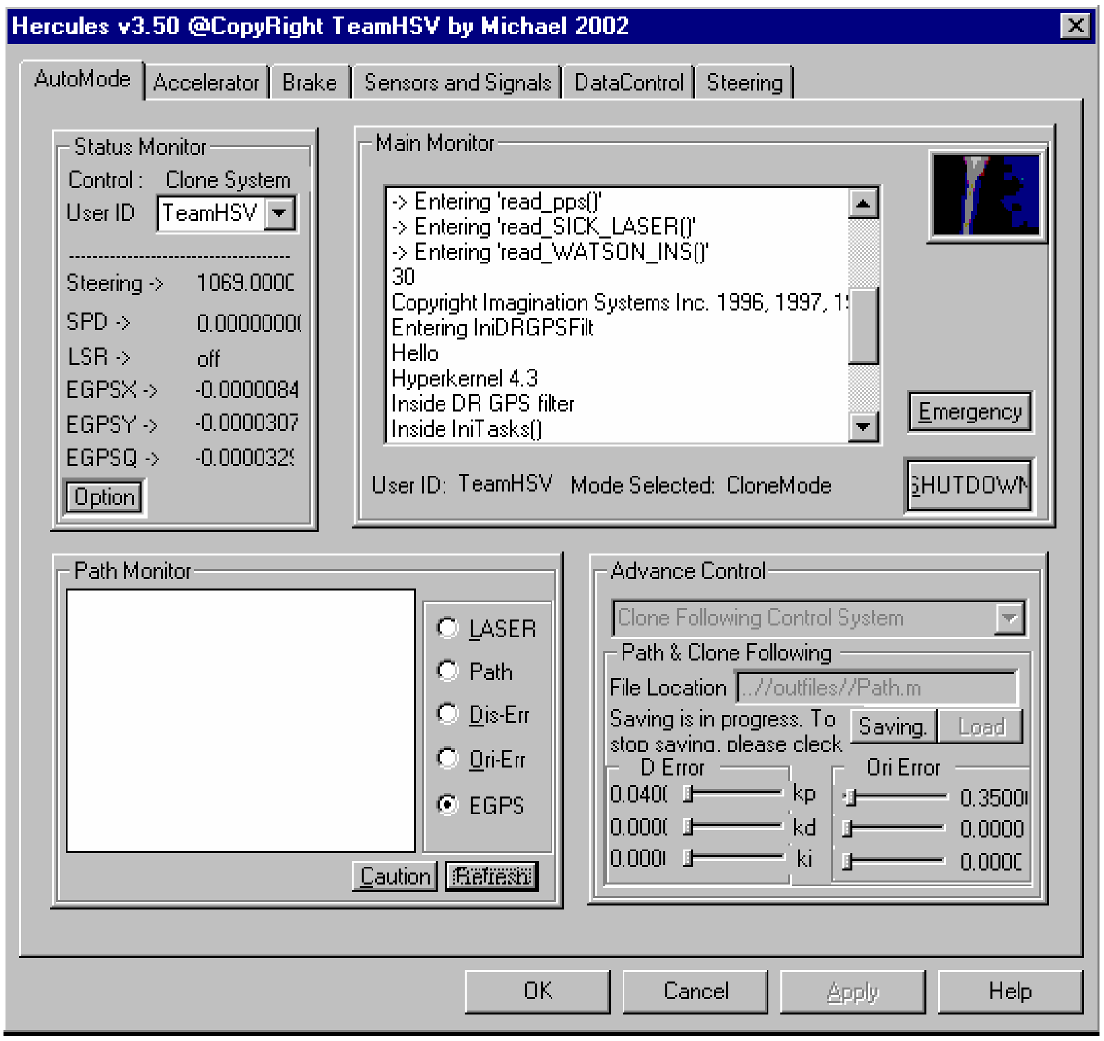
System insist to save a new clone path and load it later: Save Load

D Error		Ori Error
0.040(	kp	0.3500(
0.000(	kd	0.0000
0.000(	ki	0.000C

OK Cancel Apply Help

Figure 5.2 Clone Following control interface

Once it is saved, the car needed to return to the origin position where the path was started saving and reset to origin of the GPS to zero. The speed parsed from a prepared clone path will become the desired speed sent to a fuzzy logic controller. The software will try to control the steering motor and two actuators for speed control to follow the desired path and desired speed.


Hercules v3.50 @CopyRight TeamHSV by Michael 2002

AutoMode | Accelerator | Brake | Sensors and Signals | DataControl | Steering

Status Monitor

Control : Clone System

User ID TeamHSV

Steering -> 1069.000C

SPD -> 0.0000000(

LSR -> off

EGPSX -> -0.0000084

EGPSY -> -0.0000307

EGPSQ -> -0.000032!

Main Monitor

```
-> Entering 'read_pps()'
-> Entering 'read_SICK_LASER()'
-> Entering 'read_WATSON_INS()'
30
Copyright Imagination Systems Inc. 1996, 1997, 1!
Entering IniDRGPSFilt
Hello
Hyperkernel 4.3
Inside DR GPS filter
Inside IniTasks()
```

User ID: TeamHSV Mode Selected: CloneMode

Advance Control

Clone Following Control System

Path & Clone Following

File Location ..//outfiles//Path.m

Saving is in progress. To stop saving, please cleck

D Error		Ori Error
0.040(	kp	0.3500(
0.000(	kd	0.0000
0.000(	ki	0.000C

Figure 5.3 Clone following file saving in progress.

3.3 Summery on Hercules 3.50

Hercules is a tested program used for the tuning process in steering control system as well as speed control system. Up to the end of 2002, it is capable of doing steering control and speed control for the low level control in HSV. The keyboard remote control system is fully functional as well. The interfaces of path planning and clone system are implemented by the control algorithm hasn't been tested. Hercules 3.50 is better to work with bc_saver program which is used to save the data. Saved data is in terms of:

Time	Counts	Accele	Steer	Brake	Speed	Apwr	Spwr	Bpwr	Aset	Sset	Bset

Chapter 4

Steering response characteristic

4.1 Steering responses introduction

As indicated in previous thesis, the Steering's LVDT reading from 2028 to 3093 is representing vehicle heading from -35 deg to -1 deg(left); LVDT reading from 2026 to 492 is representing vehicle heading from 1 deg to 35 deg(right). Therefore each degree while vehicle is turning on its left-hand-side is equivalent to 43.088 units in LVDT reading. Similarly every degree on its right-hand-side is equal to 45.118 units in LVDT reading.

Experiments are taken on the grass land at St. John's college to exam and investigate the un-linear behaviors of the steering in relation with speed of the vehicle, angle increment, motor characteristic, integral term of error, directions of turning and other conditions related system response respectively.

4.2 Speed related Characteristic of steering response

4.2.1 Overshoots and Steady State Errors

From the results and analysis of steering response in all different step inputs under different vehicle speeds, it is obvious that the speed of the vehicle is related to the behaviors of the steering system.

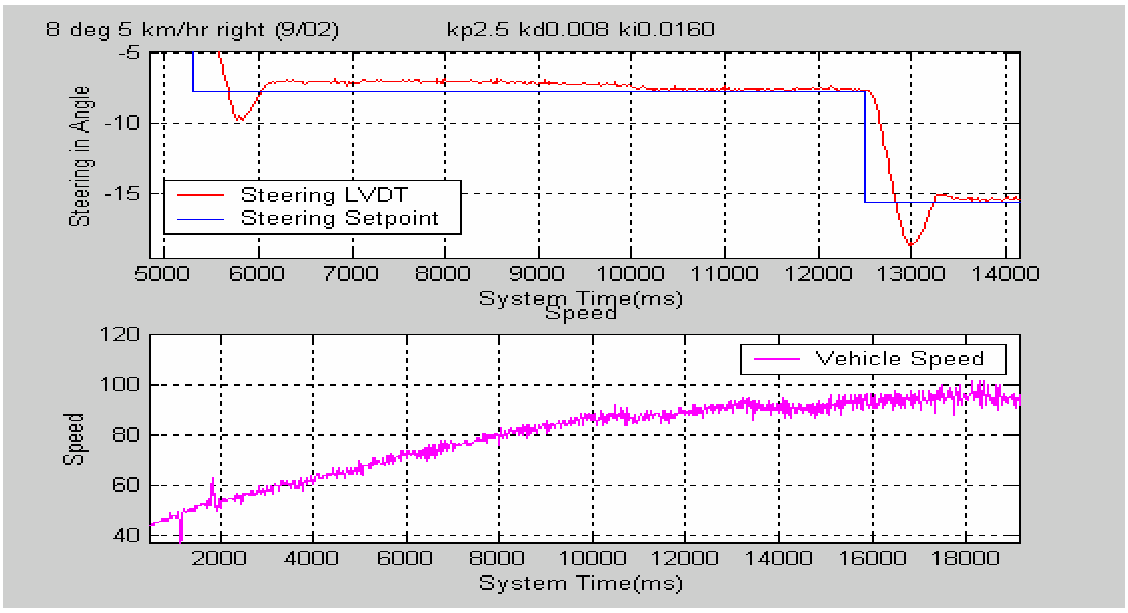


Figure 5.4: Speed related 8 degree increment

In the experiment of 8-degree-incremental step inputs while the vehicle is turning to the right direction and the speed of the vehicle varies from 70 counts to 90 counts, two steering responses(figure 5.4) are not identical to each other.

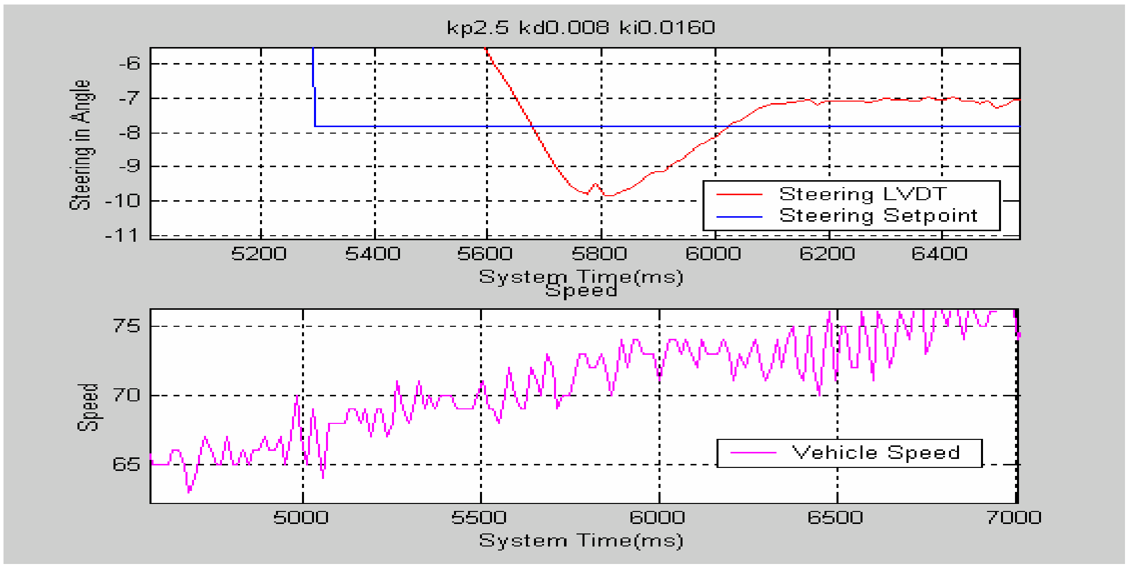


Figure 5.5: Speed related steering response at 5800ms

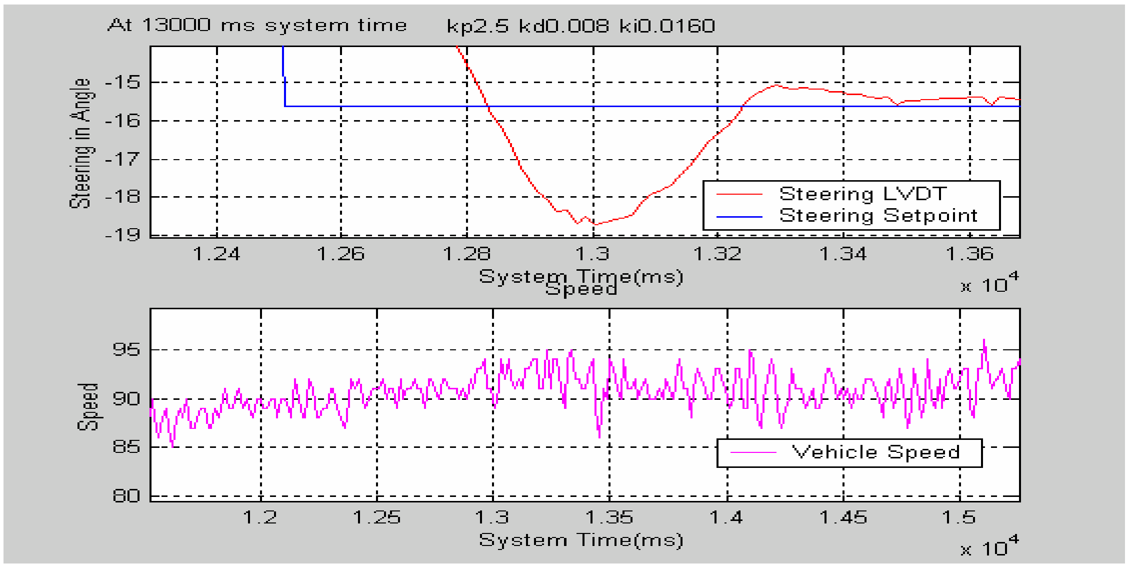


Figure 5.6: Speed related steering response at 12400ms

The step response of the steering at system elapsed time of 5800 ms(figure 5.5) under the speed of 70 counts has the overshoot of 2 degree. At system time of 12400 ms, overshoot of the steering system exceed 3 degree in angle while the speed is increased up to 90 counts in figure 5.6.

Similar response observed under the same step input increment but a higher vehicle speed. The vehicle is turning to the left.

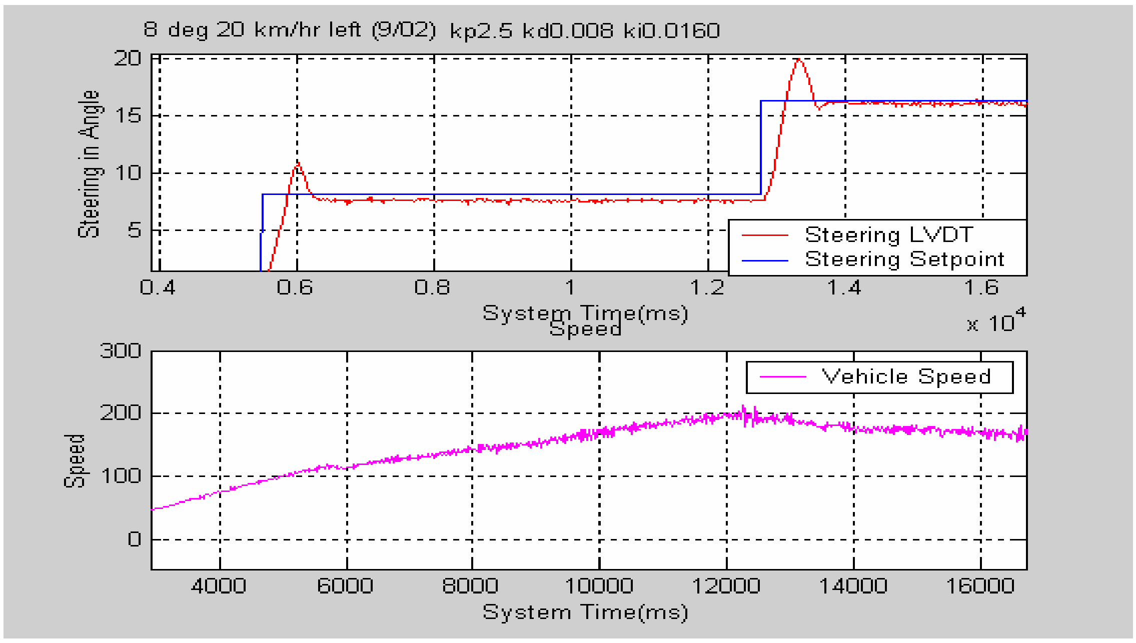


Figure 5.7: Vehicle Turning Left speed related steering responses

At system time of 6000 ms, the overshoot of the steering system is 2.5 degree as it increased up to 3.5 degree when speed is increased by 73.5 counts at 13300 ms system time.

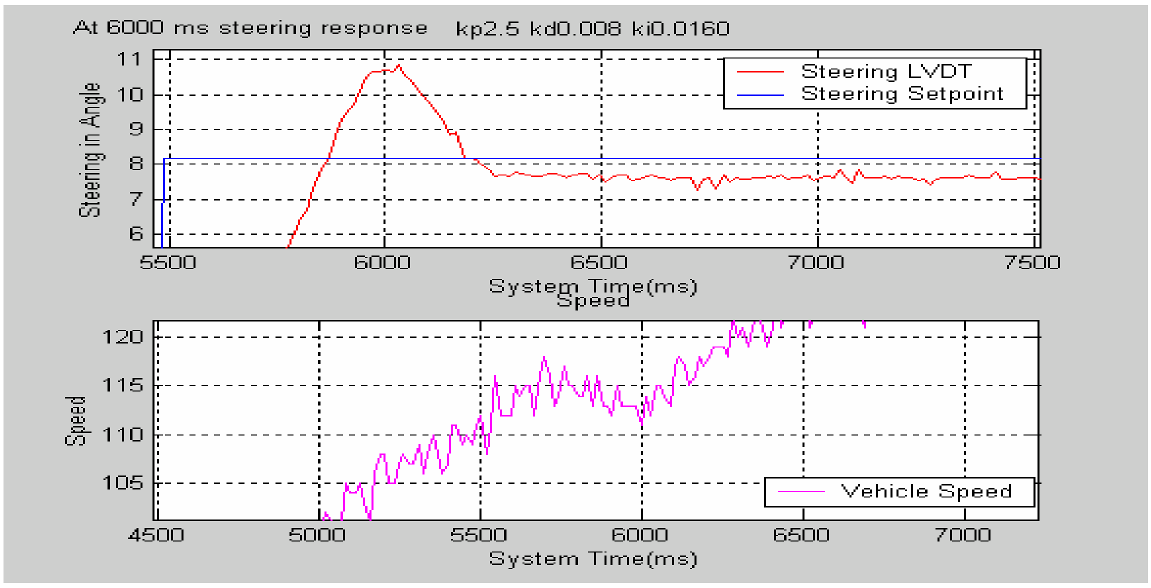


Figure 5.8: Turning Left at 6000 ms speed related steering responses

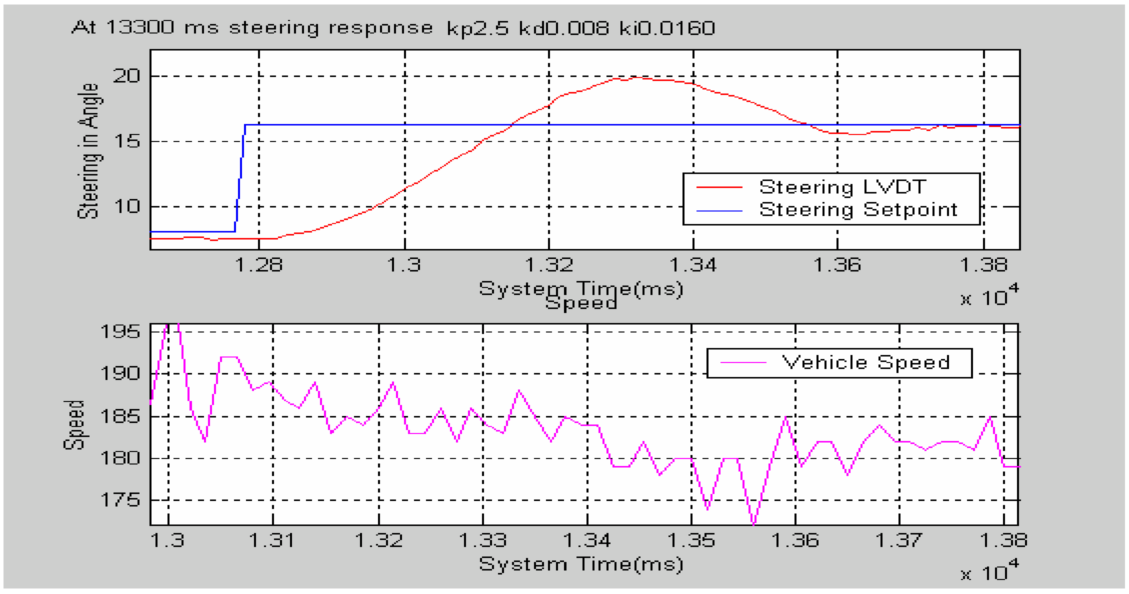


Figure 5.9: At 13300 ms Turning Left speed related steering responses

If the speed of the vehicle is reducing, it results that the overshoot of the steering response is reducing accordingly. Steady state error is relatively increased.

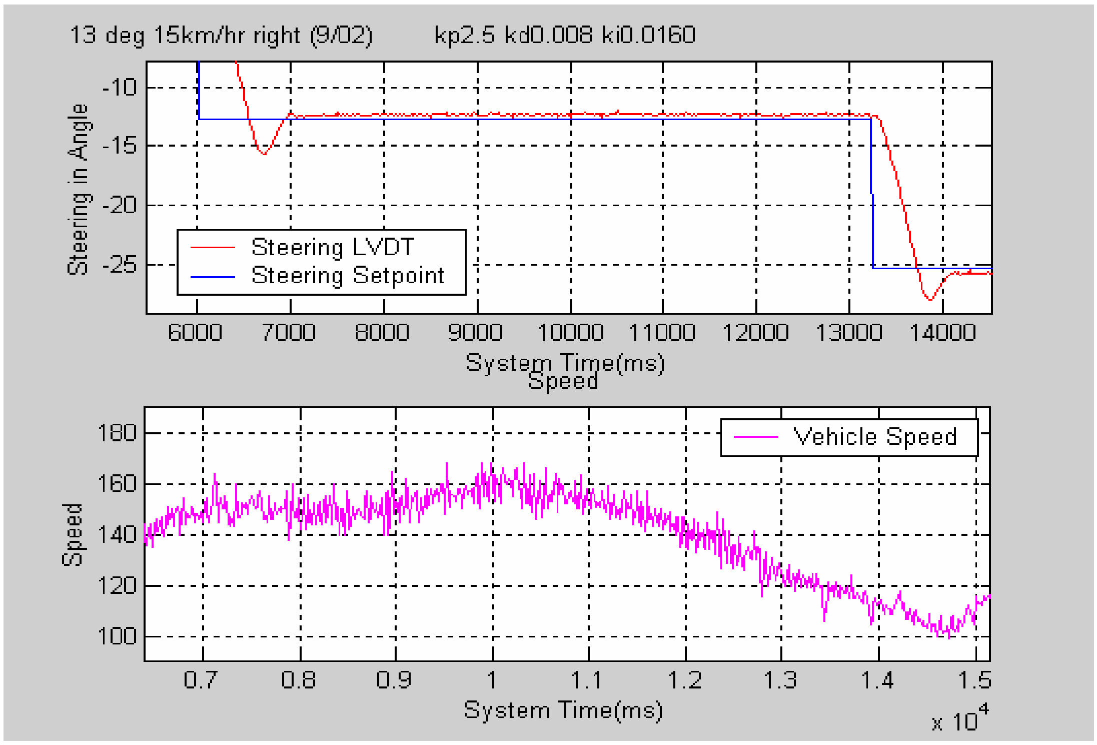


Figure 6.0: Vehicle reduces speed related steering responses

For example, in the experiment of 13-degree incremental step input while the vehicle is turning to the right direction, overshoot of the steering response reduced from 3.2 degree(Figure 6.1) to 2.6 degree(Figure 6.2) in angle. Steady state error increased from 0.25 to 0.4 degree in angle.

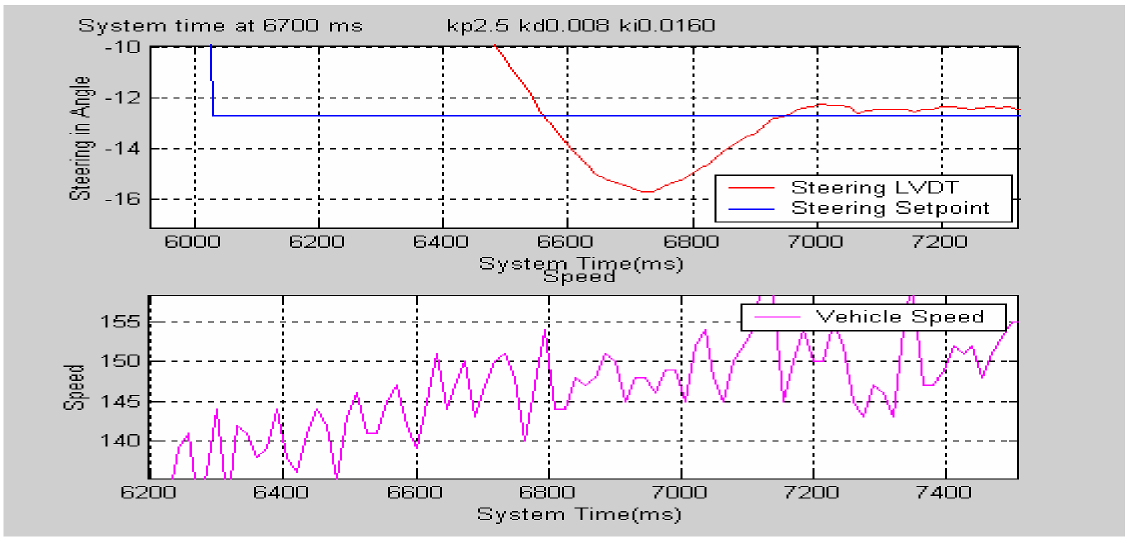


Figure 6.1: Vehicle reduces speed at 6700ms related steering responses

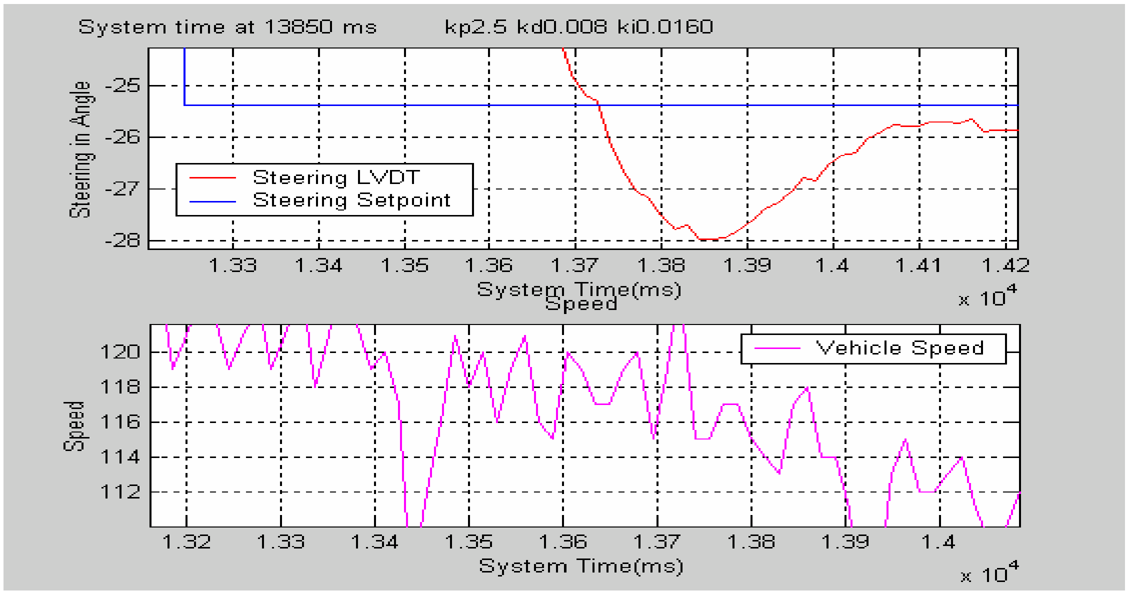


Figure 6.2: Vehicle reduces speed at 13850ms related steering responses

Similar response happens when car is turning to the left direction and speed is reducing in figure 6.3.

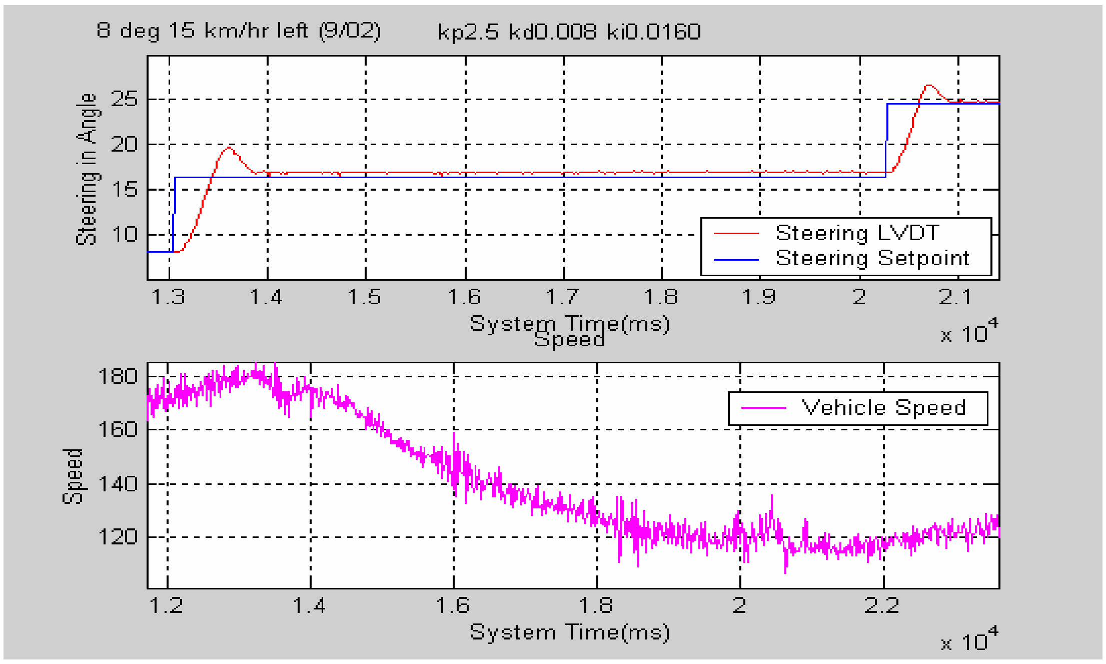


Figure 6.3: Vehicle turning left speed reducing related steering responses

An example of 8-degree incremental step input below in figure 6.4 shows the overshoot of the steering response reduces from 3.5 degree under the speed of 180 counts (figure 6.4) down to 2.2 degree in angle under the speed of 115 in counts(figure 6.5). And the steady state error reduced from 0.6 degree to 0.2 degree in angle.

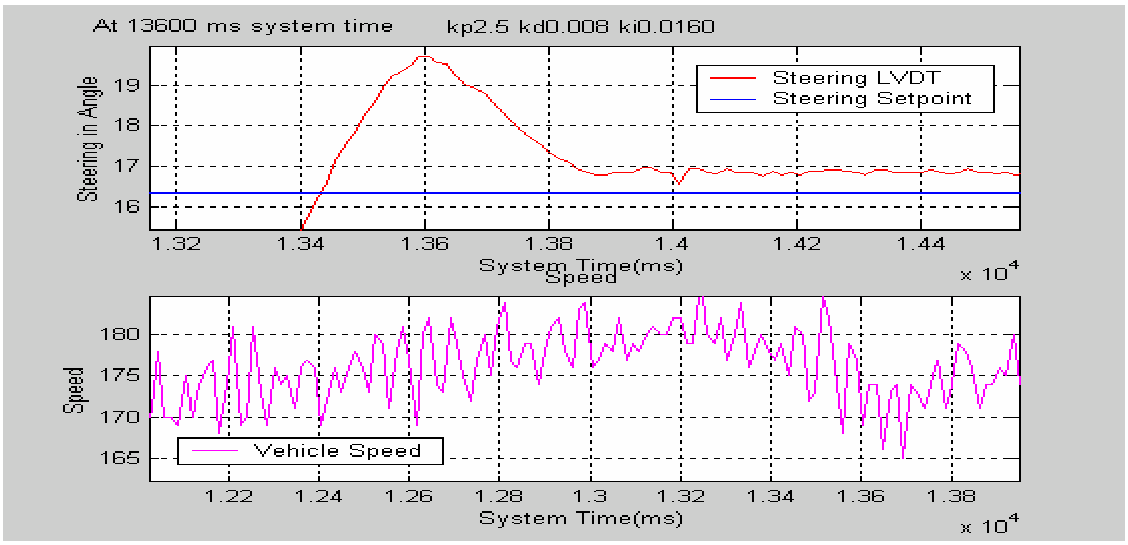


Figure 6.4: Vehicle turning left speed reducing related steering responses at 13600ms

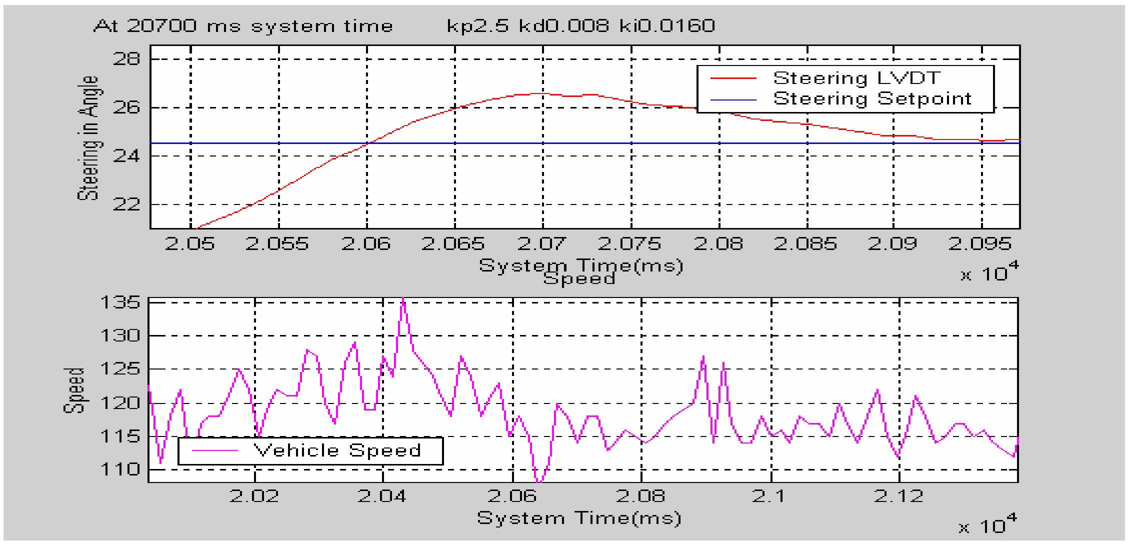


Figure 6.5: Vehicle turning left speed reducing related steering responses at 20700ms

It is very hard to produce two step-inputs under the exact same vehicle speed, however, if we can prove that the steering response identical to each other under the same vehicle speed, it will be further proving the link between various speeds and steering responses. Once we can prove that steering responses are identical to each other under one constant speed, we can compare it with any other experiences prove the bonded relationship between speed and steering.

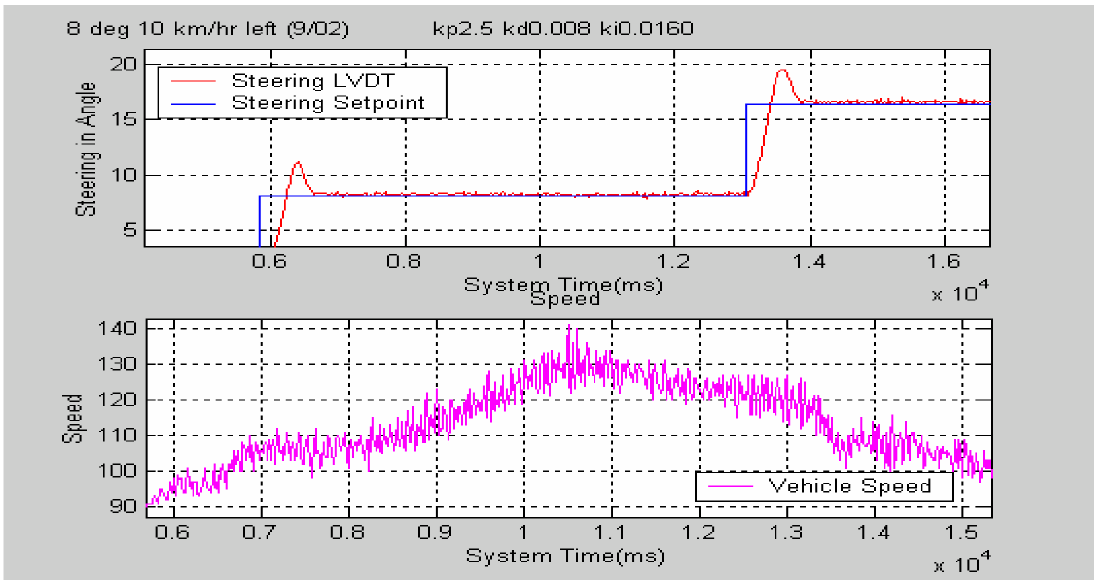


Figure 6.6: Similar speed related steering responses

As in the experiment shown in figure 6.6, an overshoot of 3 degree in angle is recorded under the speed of 100 in counts at the system time of 6400 ms(Figure 6.7), another overshoot is taken 6 seconds later which has magnitude of 3.1 degree in angle under a close speed of 110 in counts(Figure 6.8).

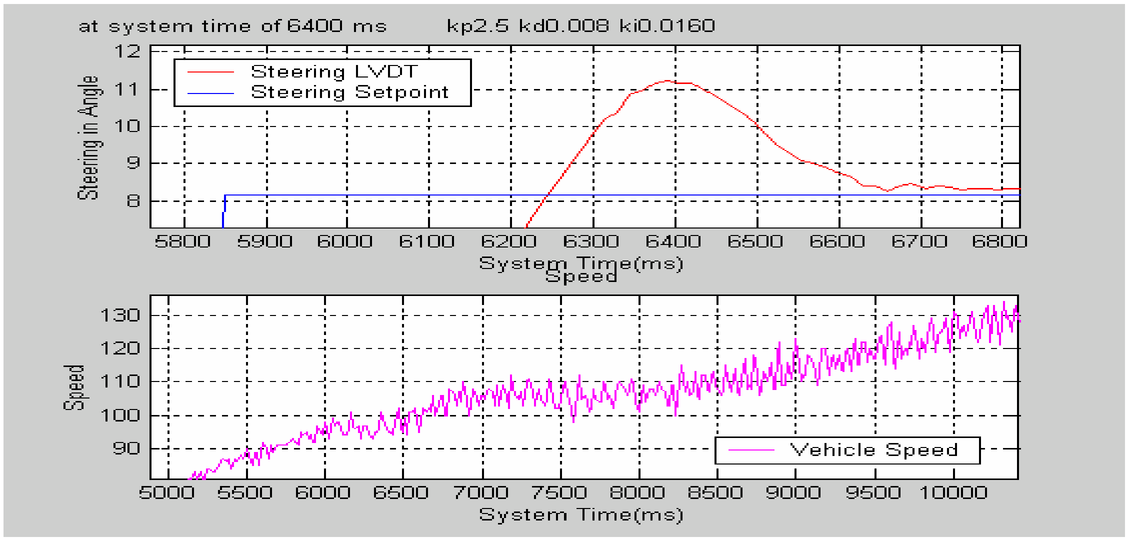


Figure 6.7: Similar speed related steering responses at 6400ms

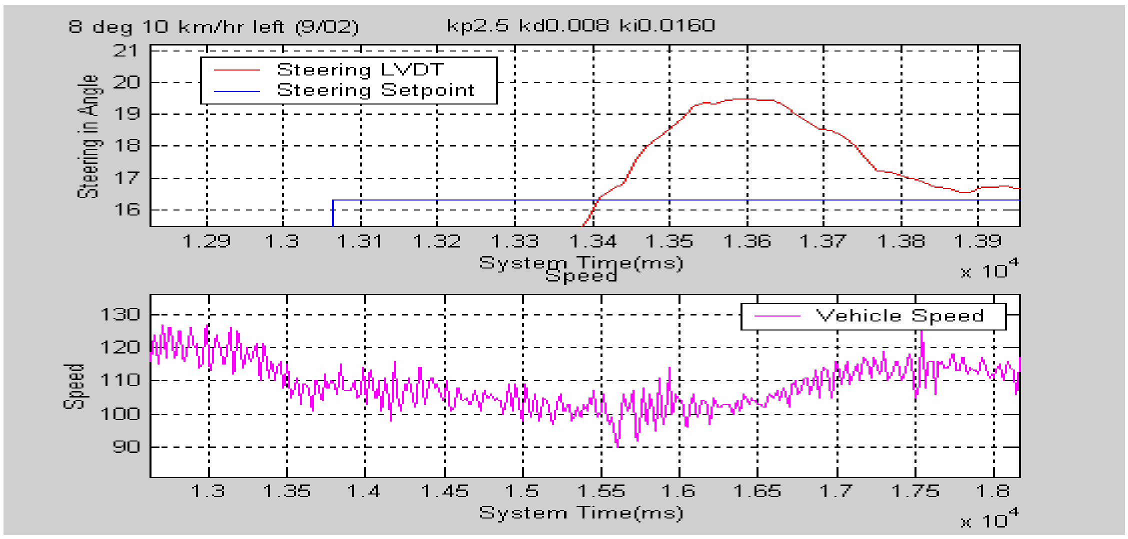


Figure 6.8: Similar speed related steering responses at 6 seconds later

Both of the responses are very close to each other as their variation in overshoot and steady state error are just 0.1 degree differences in angle. Hence we assume under the same speed and other unexpected conditions, the responses of the steering are always the same or similar in their characteristic formats.

When Speed of the vehicle keeps on increasing and over a particular spectrum, the steering response will have second overshoot or even more overshoots after the first one. It is often considered as the oscillations before the system finally settles down in the steady state.

Under the same vehicle speed, this second overshoot spectrum more likely to occur earlier in larger angle incremental inputs as the PID calculated power output is greater than that in smaller angle incremental inputs' occasions.

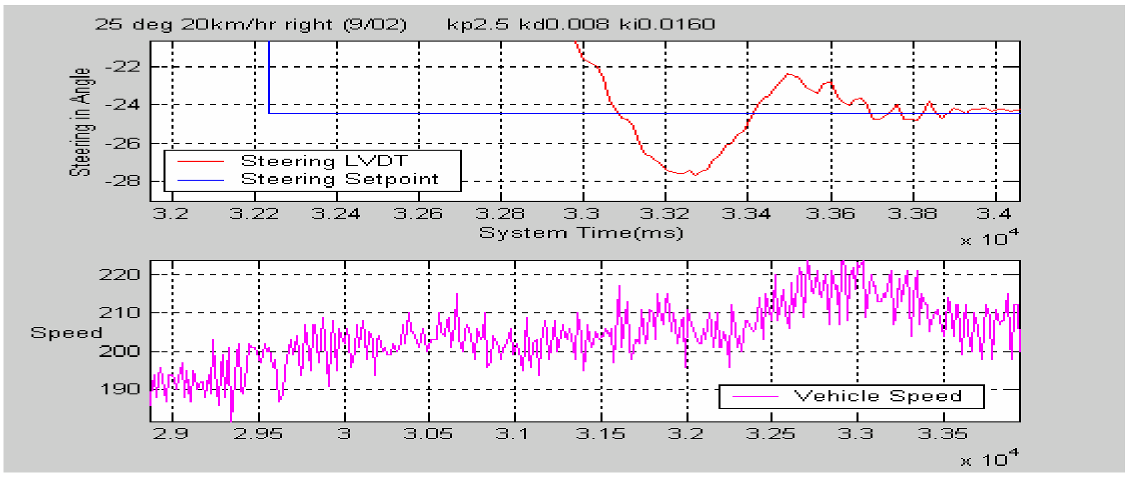


Figure 6.9: Speed related Second overshoot steering responses

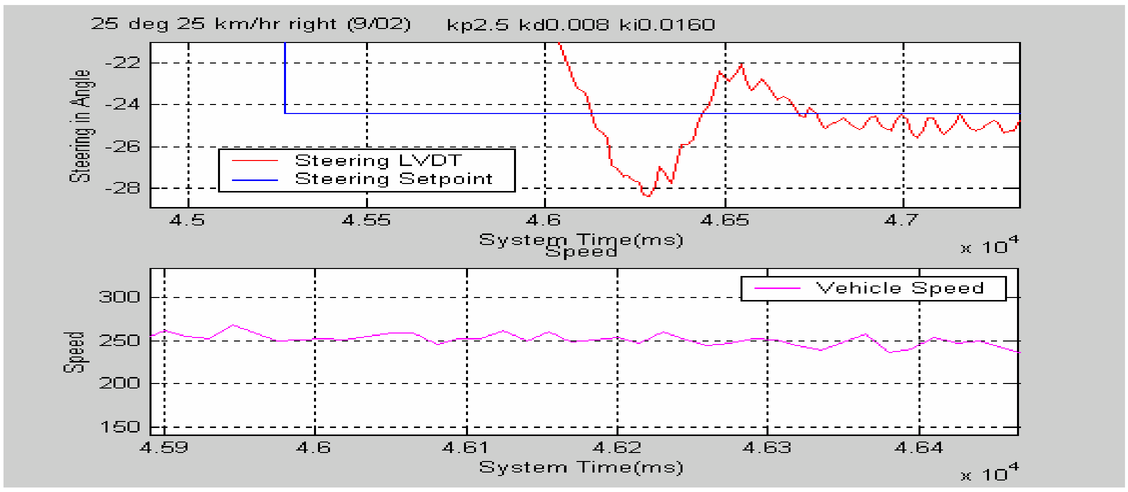


Figure 7.0: Speed related Second overshoot steering responses

The examples(in figure 6.9 and 7.0) above show the relationship between speed of the vehicle and the appearance of the second overshoots. At speed of the vehicle increases from 200 to 250 counts, the second overshoot of the steering response increases from 1.8 to 2.1 degree in angle. As mentioned before, when speed increases, the corresponding steady state error following the overshoot increases as well, for a larger speed and incremental angle inputs, the over exceeding steady state error bounds back to form a second overshoot and results in the oscillations observed above.

In compares to both power outputs related to different speed as an example of 15-degree-incremental step input experiment shown below, both power plots should be observed as identical if we assume that speed of the vehicle is not related to the steering responses. But in fact the difference between two power plots appeared on the size of the overshoot when power output tends to bounds back to zero.

Experiments show that the size of the power overshoot is in fact related to the speed of the vehicle. The bigger the speed demanded, the larger the oscillations power output will result when it tend to settle down to zero.

Because of these oscillated power outputs, steering response favor to have second or even more overshoots after the first one.

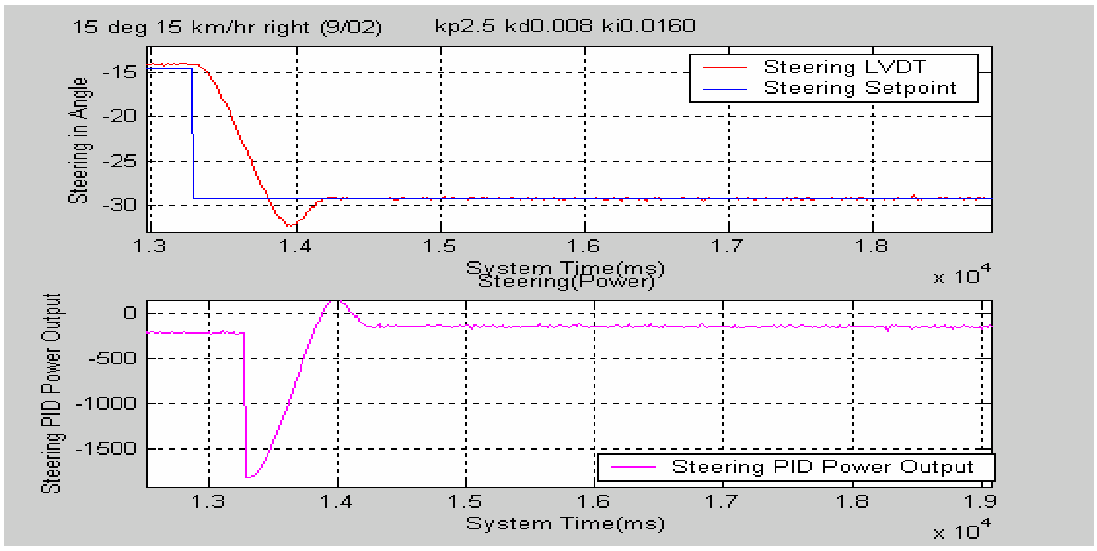


Figure 6.9: Speed related Second overshoot steering responses

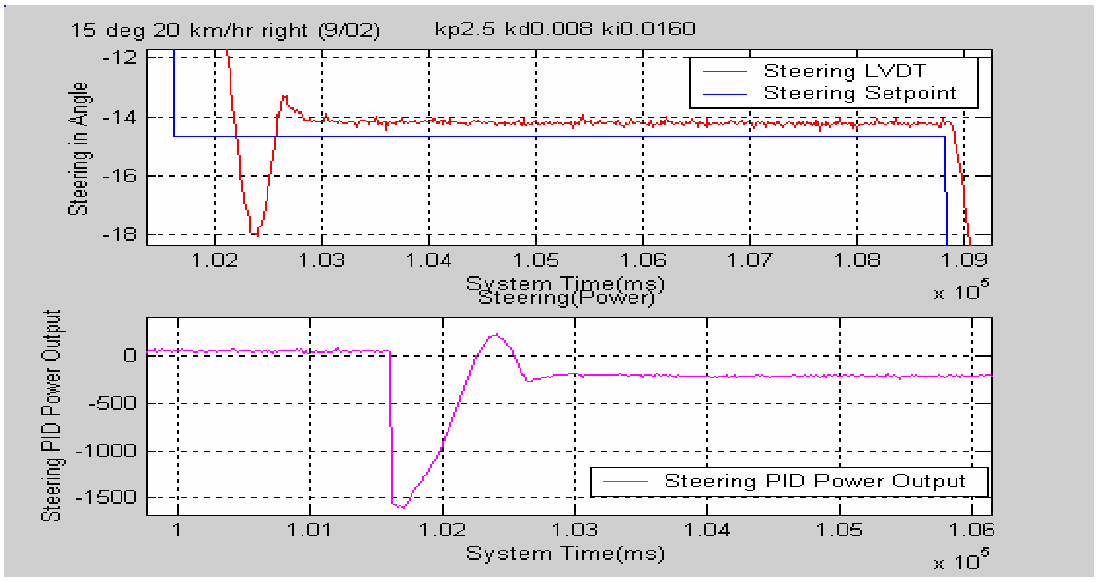


Figure 6.9: Speed related Second overshoot steering responses

In high level steering control, it is wise to avoid second overshoot appearing if it is avoidable.

Because of the more overshoots, the steering response more likely to have a huge steady state error in the result of motor dead zone(explain in later chapter), unless a 'kick' in the power output performed or a larger integral error piling up in time.

Statistic summery on speed related Overshoots and steady state errors:

4.2.2 System Rising time and Overshoot time

Speed does not seem to play a role in system rising time unlike angle increment does. But it is responsible to the change of overshooting time in result of the total system settling time.

Since various speeds change the shape of overshoot in the steering response, the time taken for the overshoot to settle down to steady state is changed accordingly. The time is considered to be the overshoot time for steering response. Statistic of the experiments shows that the faster the speed is, the longer it takes in overshoot time.

Angle Increment	Speed in Counts	Max 1st overshoot time	Max 2nd overshoot time
2	170	0.4	no second overshoot
	240	1.6	no second overshoot

Angle Increment	Speed in Counts	Max 1st overshoot time(in second)	Max 2nd overshoot time
8	130	0.31	0.17
	190	0.32	0.3
	230	0.45	no second overshoot

Angle Increment	Speed in Counts	Max 1st overshoot time	Max 2nd overshoot time
10	190	0.33	0.2
	245	0.34	0.3

Angle Increment	Speed in Counts	Max 1st overshoot time	Max 2nd overshoot time
15	115	0.38	no second overshoot
	160	0.4	no second overshoot

Angle Increment	Speed in Counts	Max 1st overshoot time	Max 2nd overshoot time
20	50	0.34	no second overshoot
	80	0.35	0.125
	160	0.35	0.15

Angle Increment	Speed in Counts	Max 1st overshoot time	Max 2nd overshoot time
25	75	0.35	no second overshoot
	140	0.4	no second overshoot
	200	0.35	0.24
	250	0.31	0.26

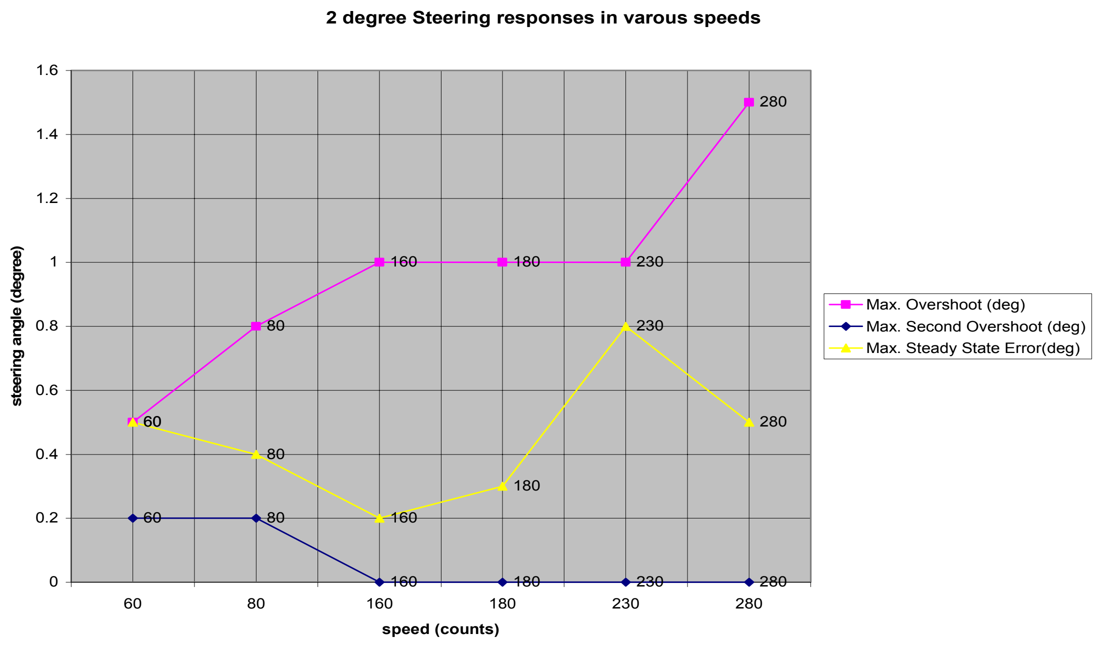
From the statistic above, it is clear to see that as speed increases, there is also an obvious increase in the overshoot time of the steering response. Therefore the total system settling time is longer and it means it will take longer for the steering system to be steady or stable.

4.2.3 Summery on Speed related Characteristics

Angel Increment	Speed Ranges(km/hr)	Wheel Encoder(offline)	Max. Overshoot (deg)	Max. Second Overshoot (deg)	Max. Steady State Error(deg)
2 deg	5	60	0.5	0.2	0.5
	10	80	0.8	0.2	0.4
	15	160	1	0	0.2
	20	180	1	0	0.3
	25	230	1	0	0.8
	30	280	1.5	0	0.5
5 deg	5	40	2.8	0	0.3
	10	90	2.9	0	0.5
	25	235	3	0	0.5
8 deg	5	50	3	0	0.3
	15	165	3.5	1	0.5
	20	220	3.8	1.5	0.5
10 deg	5	55	2.5	0	0.3
	15	170	3.5	2.1	0.5
	20	190	3.8	1.5	0.7
13 deg	10	120	3.4	0	1
	25	230	4	2.2	1.1
15 deg	5	72	2.9	0	0.05
	10	115	3	0	0.2
	15	160	3.2	0	0.5
	20	210	3.4	0.8	0.5
18 deg	5	80	3	0	1
	10	120	3.7	0.5	0.2
	15	150	3.8	0.7	0.5

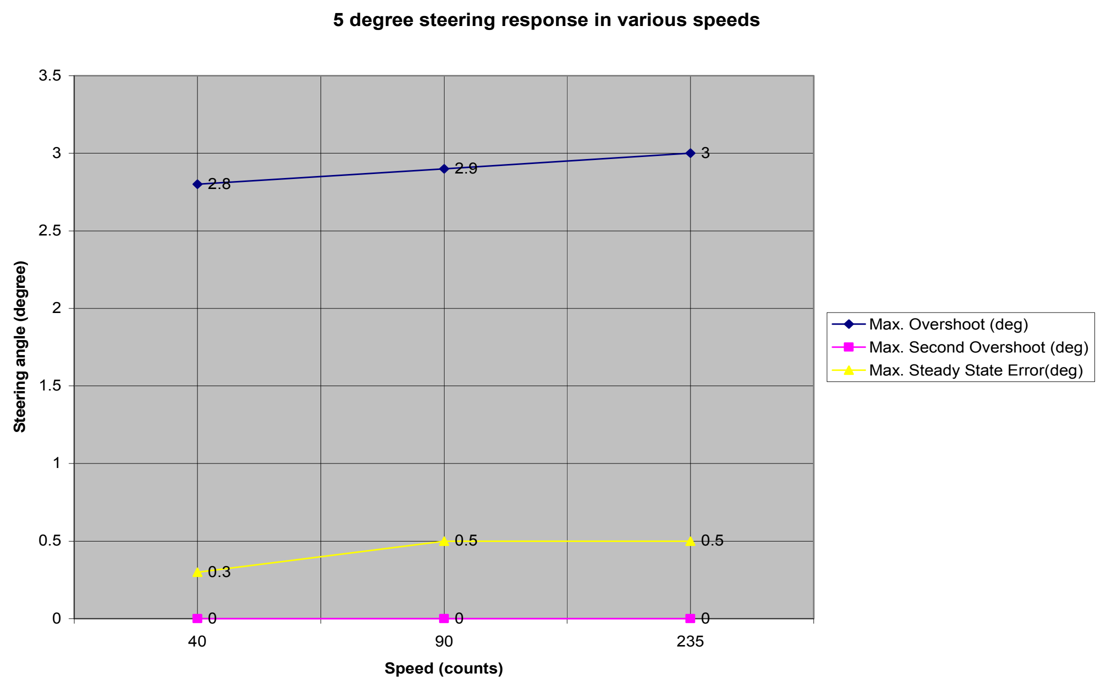
20 deg	5	50	3.2	0	0.2
	10	80	3.5	0.13	0.2
	15	160	3.6	0.25	0.4
22 deg	5	70	3	0	1
	10	95	3.3	0	0.1
	15	150	3.5	0	0.15
	20	200	3.7	1.5	0.5
25 deg	5	75	3	0	0.4
	15	140	3.2	0	0.6
	20	200	3.4	1.8	0.6
	25	250	3.9	2.1	0.4

The statistic are plotted according to its increment and test number below, with some additional characteristic and name of the mathlab files stored in the CD on the back of the thesis booklet.



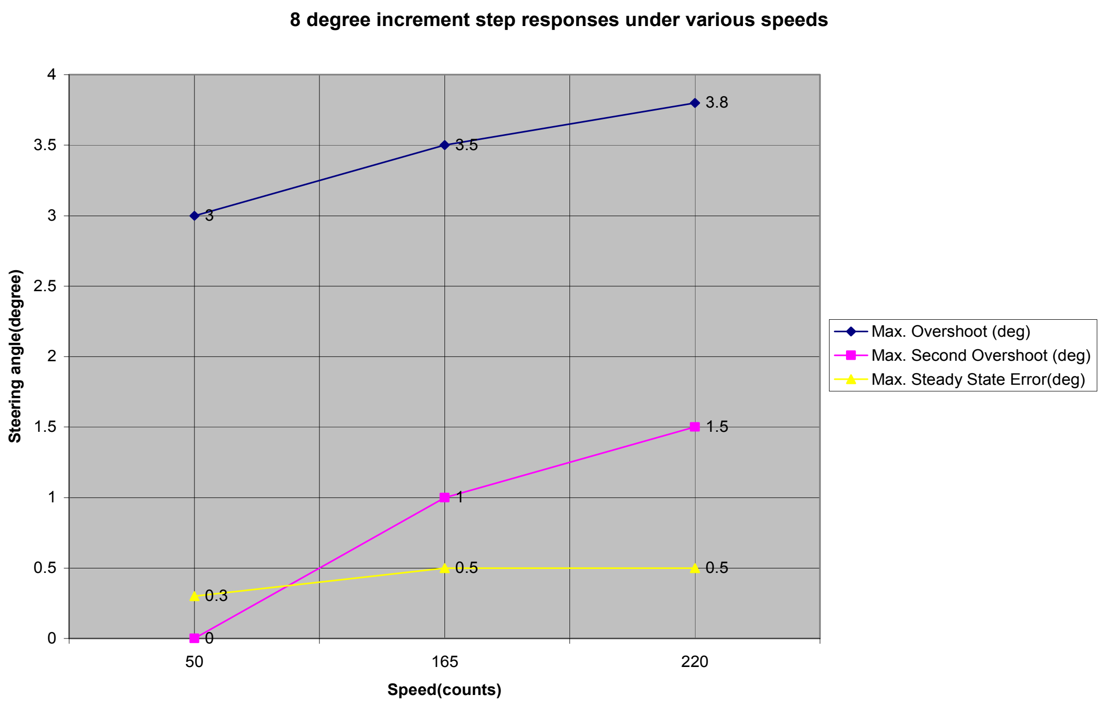
In the summery of 2 degree steering response due to different vehicle's speed, as speed is increasing the first overshoot of the system increases 0.5 degree to 1.5 degree. Second overshoot appeared only in lower speed tests, as most of them are caused by motor characteristic related or integral term related steering responses.

Tests #No.	Angel Increment	Mat File ID	Figure File ID
1	2 deg	TS11	2deg5km
		TS12	2deg10km
		TS13	2deg15km
		TS14	2deg20km
		TS15	2deg25km
		TS16	2deg30km



In the summery of 5 degree increment step response of the steering, as figure above shown, three out of five tests shows the increase in both overshoot and steady state error. Statistic shows that second overshoots of the steering response are not obvious. In addition there are some characteristics shows there are a few quick second overshoots about 0.7 degree and last less than 0.02 seconds appear when speed reaches 25km/hr.

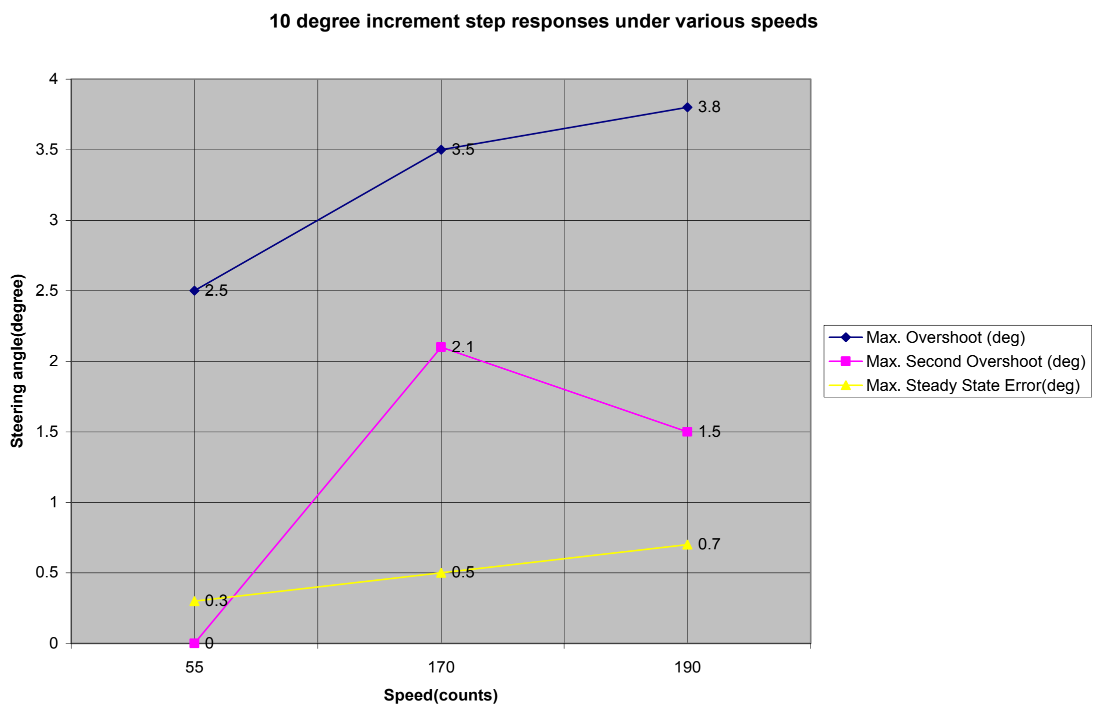
Tests #No.	Angel Increment	Mat File ID	Figure File ID
2	5 deg	TS21	5deg5km_Power, 5deg5km
		TS22	5deg10km_Left, 5deg10km_Right
		TS23	5deg15km_Left, 5deg15km_Right
		TS24	5deg20km_Left, 5deg20km_Right_Power
		TS25b	5deg25km_Left, 5deg25km_Right



In the summery of 8 degree increments step responses, some of the tests when speed reaches 15km/hr, there are some vibrations in the steering system causing plots have shape edges as noises. It is more obvious in 25km/hr test result.

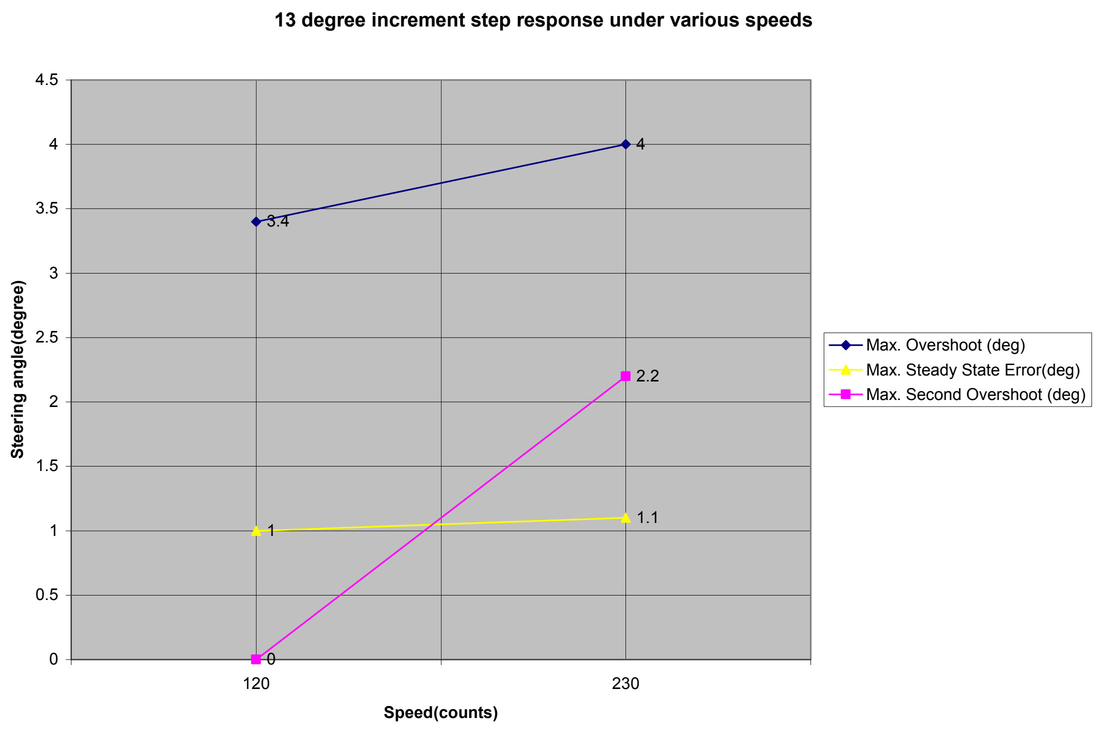
In the test of 20 km/hr some results show third overshoots' appearances.

Tests #No.	Angel Increment	Mat File ID	Figure File ID
3	8 deg	TS31b	8deg5km_Left_Speed, 8deg5km_Right
		TS32	8deg10km_Left_Speed, 8deg10km_Right
		TS33c	8deg15km_Left_Speed, 8deg15km_Right
		TS34b	8deg20km_Left, 8deg20km_Right_Speed
		TS35	8deg25km_Left_Speed, 8deg25km_Right



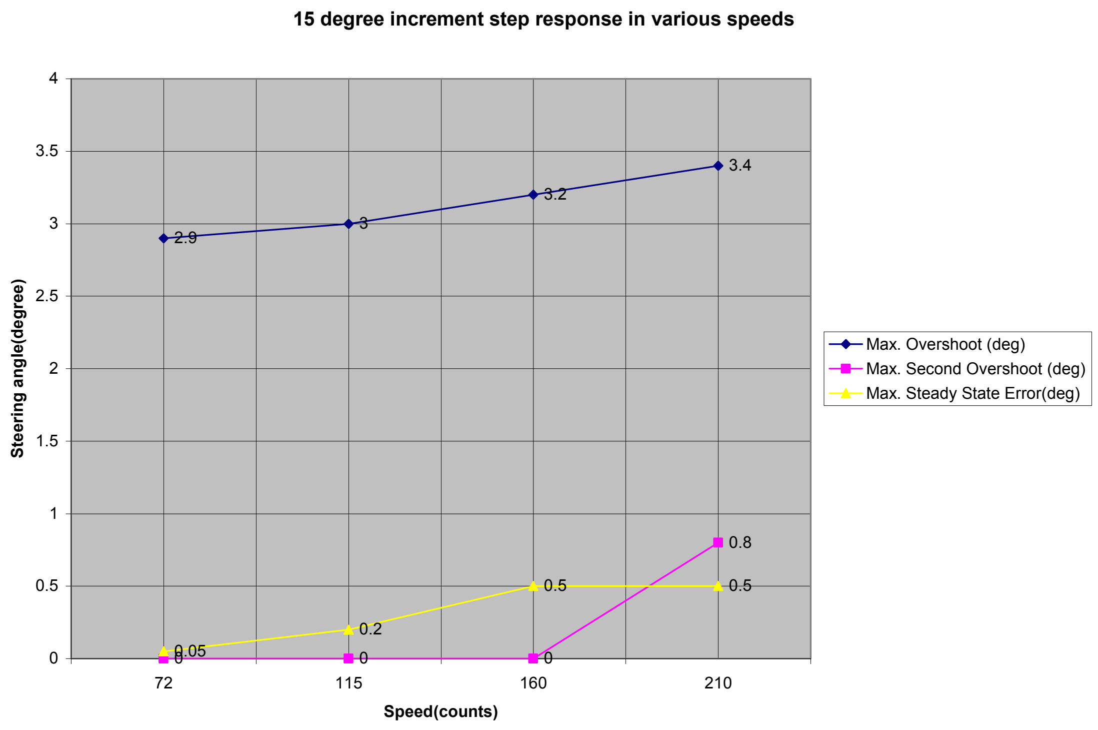
There are some obvious curve sharpness possibly caused by vibration in the test of 15 km/hr 10 degree increment step response of the steering. The second overshoot starts to appear in the 20 km/hr test. Third overshoots start to form in 25 km/hr test with 1 degree in magnitude.

Tests #No.	Angel Increment	Mat File ID	Figure File ID
4	10 deg	TS41	10deg5km_Left, 10deg5km_Right
		TS42c	10deg15km_Left_Speed, 10deg15km_Right
		TS43b	10deg15km_Left, 10deg15km_Right_Speed
		TS44	10deg20km_Left, 10deg20km_Right
		TS45	10deg25km_Left, 10deg25km_Right_Speed



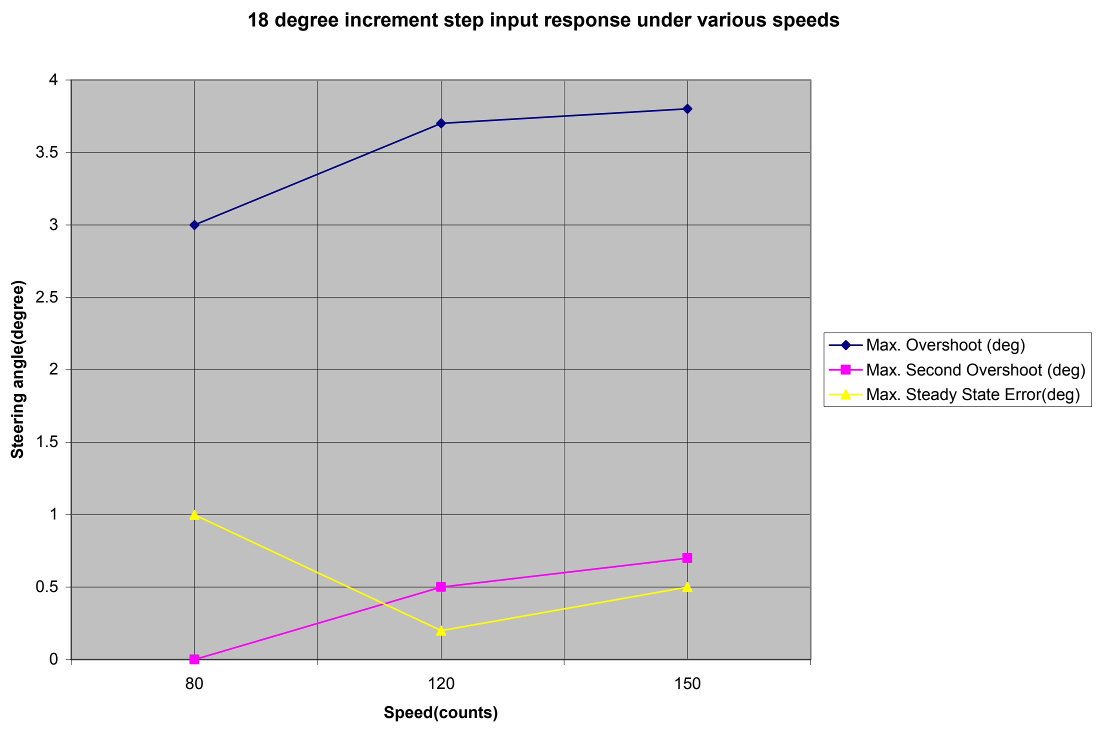
At 5 km/hr, some smallest steady state recorded as low as 0.15 in degree. Larger steady state error of up to 1 degree has been recorded in the test at 20 km/hr. some characteristic in the test at 25km/hr indicates the overshoot time is relevant to speed changes.

Tests #No.	Angel Increment	Mat File ID	Speed Ranges(km/hr)
5	13 deg	TS51	5
		TS52	10
		TS53b	15
		TS54	20
		TS55	25



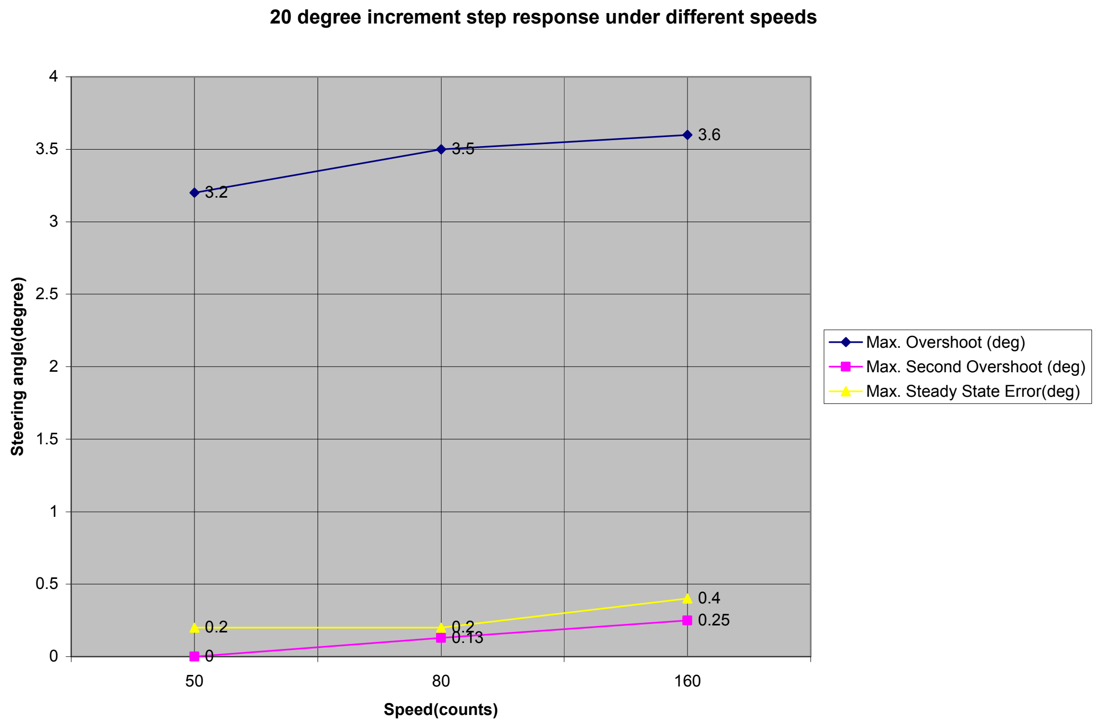
At 10 km/hr test, there are some very small second overshoot lasts only 0.1second. Some characteristic of steering shown in the test under 15 km/hr that second overshoot is relevant to speed changes. 20 km/hr test shows some extremely long rising time and some second overshoots over 0.8 degree.

Tests #No.	Angel Increment	Mat File ID	Speed Ranges(km/hr)
6	15 deg	TS01	5
		TS02b	10
		TS03b	15
		TS04c	20



In 10 km/hr test results, second overshoot of 0. 5 of a degree has been recorded, it increased up to 0.7 when speed closes to 15km/hr. in the test under 25km/hr, second overshoots jumped up to 1.7 degree, but overshoot time reduced.

Tests #No.	Angel Increment	Mat File ID	Speed Ranges(km/hr)
7	18 deg	TS61	5
		TS62	10
		TS63	15
		TS64	20

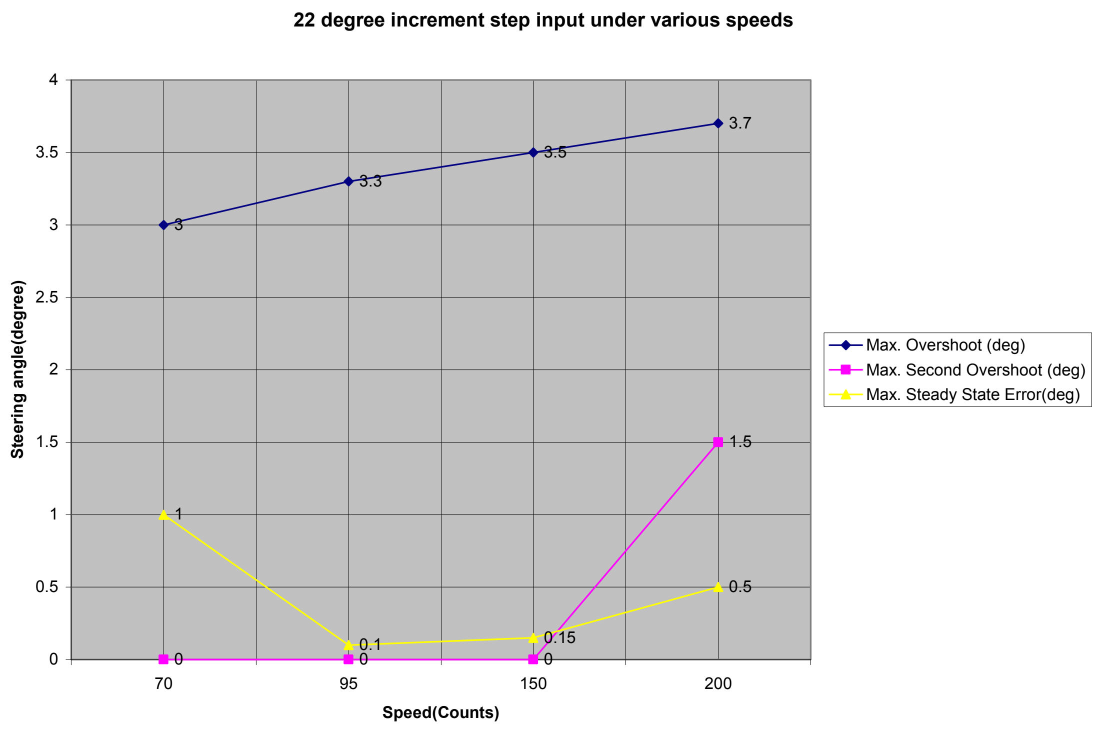


There are some steering characteristics conclusions shown longer rising time of the system caused by larger angle increments in step response.

Second overshoots started to appear under 10 km/hr.

Characteristic shown in 15 km/hr indicates maximum power output time is irrelevant to speed changes.

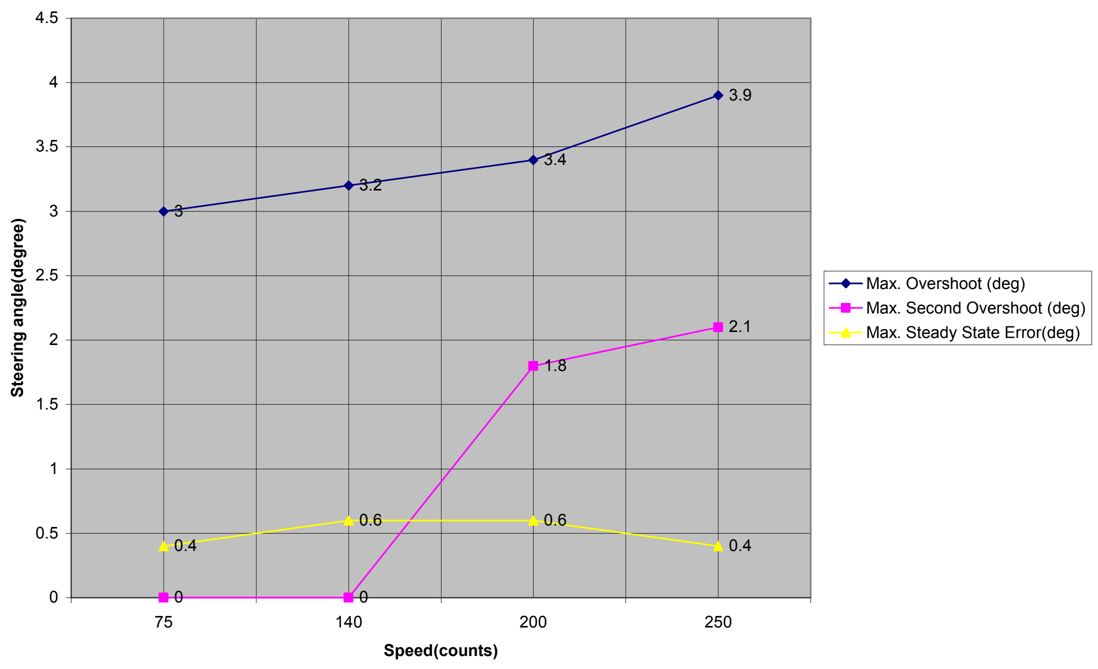
Tests #No.	Angel Increment	Mat File ID	Speed Ranges(km/hr)
8	20 deg	TS71	5
		TS72	10
		TS73b	15
		TS74b	20



Some characteristic shows that steady state error increases as more turning torque required as closer to limit under 5km/hr. It is further proven that maximum power output time is not related various speed. Tests show the characteristic of an angle increment related changes in overshoots and steady state error. Test under 20km/hr shows large second overshoot and increased steady state error.

Tests #No.	Angel Increment	Mat File ID	Speed Ranges(km/hr)
9	22 deg	TS81	5
		TS82	10
		TS84c	15
		TS83	20

25 degree increment step response under various speeds



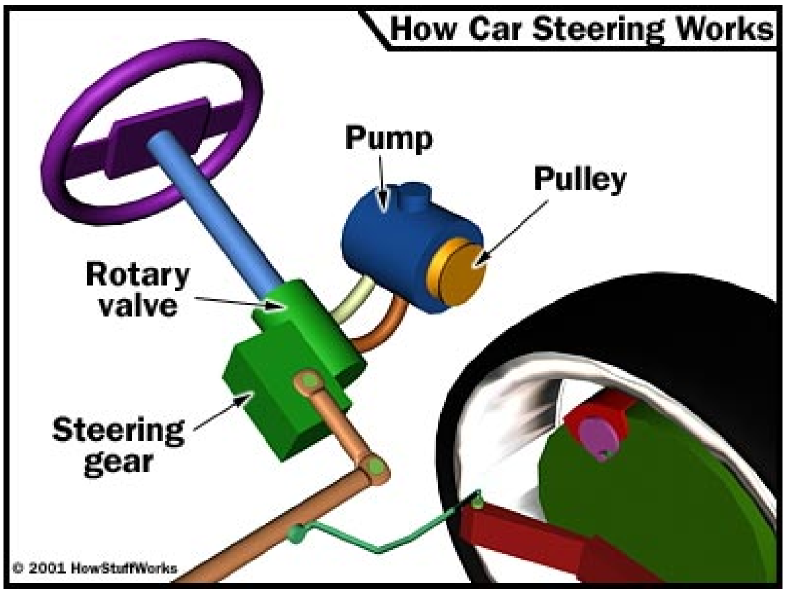
In the analysis of 15km/hr test, it is clear that the second overshoot time is related to the speed changes. In the test under 20 km/hr appears large second overhshoots.

Tests #No.	Angel Increment	Mat File ID	Speed Ranges(km/hr)
10	25 deg	TS91	5
		TS92	15
		TS93	20
		TS94b	25

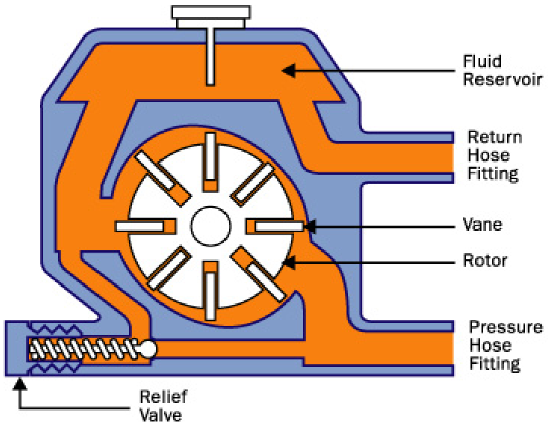
4.2.4 Speed and Power Steering

Speed related steering response may be caused by Power Steering

A power steering system consists of a rotary-vane pump, a rotary valve and a pully.

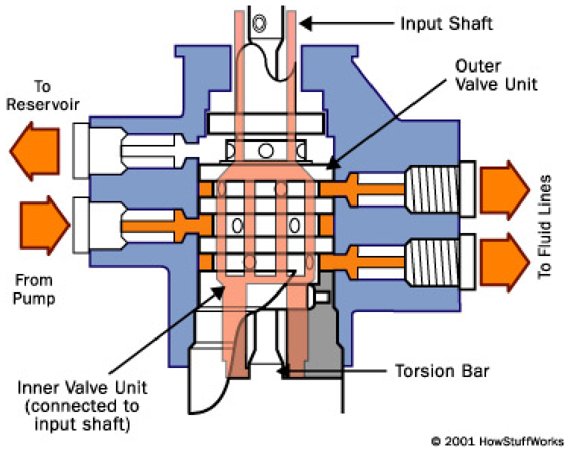


The rotary-vane pump provides hydraulic power for the steering. It is driven by the car's engine via a belt and the pulley. It contains a set of retractable vanes that spin inside an oval chamber. As the vanes spin, they pull hydraulic fluid from the return line at low pressure and force it into the outlet at high pressure. The amount of flow provided by the pump depends on the car's engine speed. The pump is to be designed that provide adequate flow when the engine is idling. As a result, the pump moves much more fluid than necessary when the engine is running at faster speeds.



The pump contains a pressure-relief valve to make sure that the pressure does not get too high, especially at high engine speeds when so much fluid is being pumped.

A power-steering system should assist the driver only when he is exerting force on the steering wheel(such as when starting a turn). When the driver is not exerting force, the system should not provide and assist. The device that senses the force on the steering wheel is called the rotary valve.



The key to the rotary valve is a torsion bar. The torsion bar is a thin rod of metal that twists when torque is applied to it. The top of the bar is connected to the steering wheel, and the bottom of the bar is connected to the pinion or worm gear(which turns the wheels), so the amount of torque in the torsion bar is equal to the amount of torque the driver(steering motor in HSV) is using to turn the wheel. The more torque the driver uses to turn he wheels, the more the bar twists.

In another word, power steering is designed by the car manufactory to amplify the force we use to turn the steering wheel. It is like a mechanical amplifier, which amplify our force of tuning the steering with respect to the speed of the engine. The gain of this mechanical amplification depends on how much fuel has been pumped from the engine into the hydraulic. Faster the vehicle speed, more tuning power will be added to the steering wheel.

Power Steering is a very un-linear issue as how much fuel is pumped into the mechanical depends on not just the speed but temperature of the engine, the fuel level left in the fuel chamber and the conditions of each of the power steering elements.

Under these circumstances, it is not too difficult to understand the behaviors of steering system responses to different speeds. As speed increases, the speed of engine increases as well. It results more fuel be pumped to the mechanical amplifier and hence cause the steering wheel much easier to turn. Because power output of the PID controller is not related to speed at all, therefore under the same step increments but fast speed, steering wheel will be overturned and causes larger overshoots and steady state error.

4.3 Angle Increments Related Steering Response

It is not difficult to see that the angle increments can also affect the responses of the steering system. For a very small angle increment such as 2 degree or 5 degree in angle, steering response are not seemed to be very stable from the figure shown below:

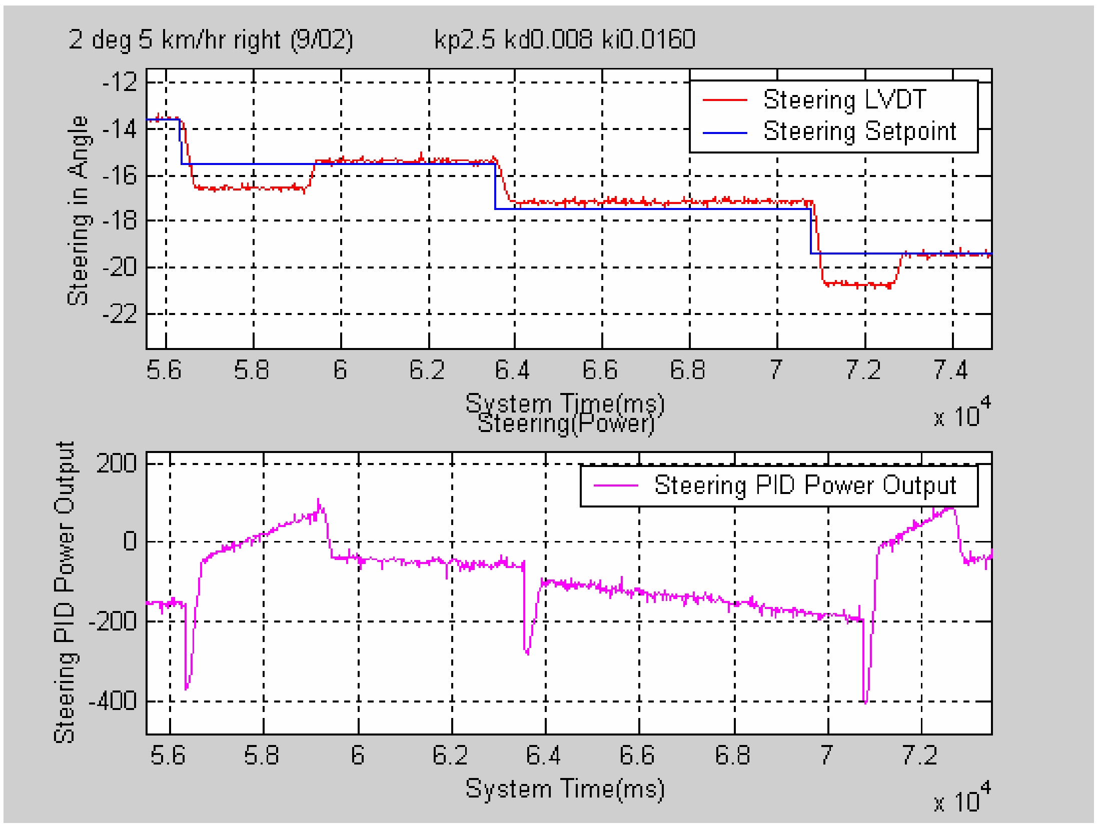


Figure 7.1 2 degree increments in angle

In the experiment (figure 7.1)of 2 degree incremental step response under the speed of 5 km/hr, the steering response seems to have a extremely long overshoot times, and each overshoot will often followed by a undershoot. In the power output plot under, it is obvious that power output experiencing an offset from zero. Unless there is a jump in the output power, there will be not movement to the steering. This is often caused by motor's dead zone. A motor's dead zone indicates the power region that motor can not turn the steering wheel. The motor still be able to rotate within the dead zone, but can not deliver enough turning torque to move the steering wheel. The Motor's dead related steering responses are often very similar to each other and they are more likely to occur in the small incremental step responses, because of the power output calculated from the desired position and actual position are often very small.

In a zooming view (figure 7.2)of the previous plot at system elapsed time of 58000 ms, when power output lies in the region of 0 to 100 (counts), the torque generated by motor is not sufficient enough to turn the steering wheel. The dead zone range from 0 to 100 is a part of the motor’s dead zone spectrum, noted as the positive spectrum.

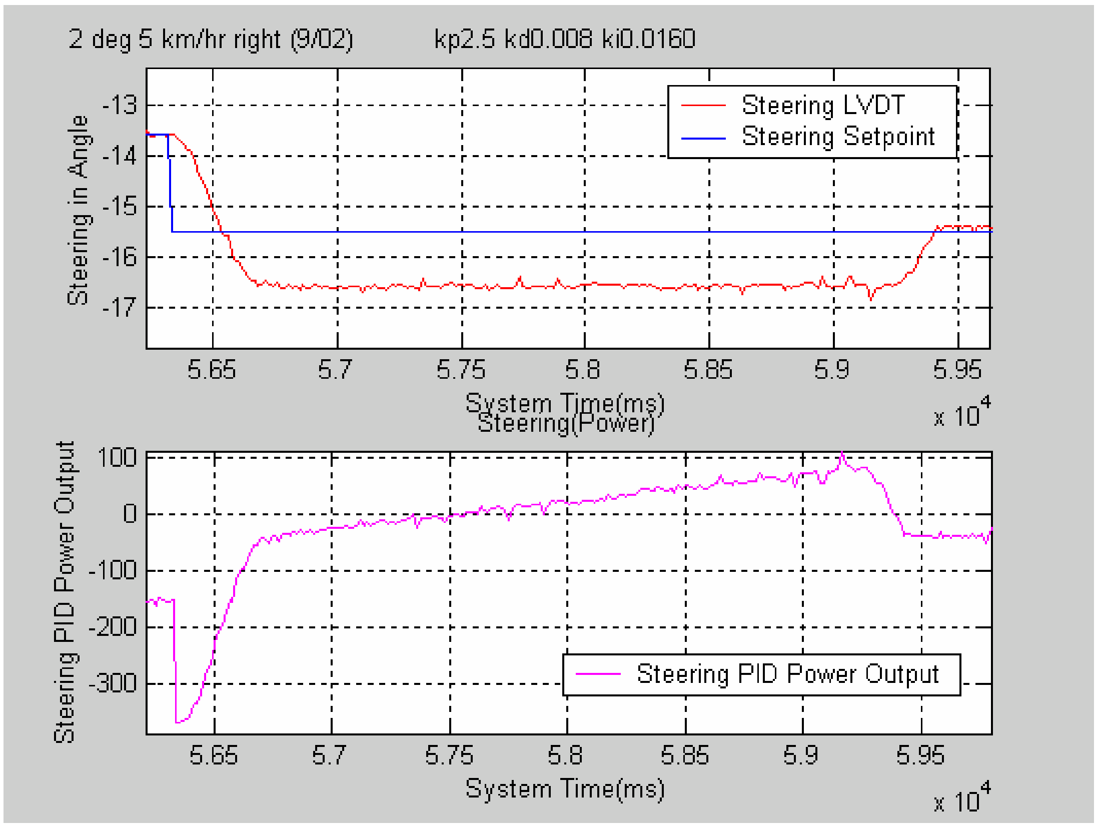


Figure 7.2: zoom at 58000ms

The power’s dead zone related steering responses however are not very obvious in the bigger angle increments step responses. But in a slightly larger increment step response, its characteristic have been transformed into another formation.

In the experiment of 5 degree incremental step response, in stead of have the characteristic of one long overshoot follows by one long undershoot and then one overshoot again, the first overshoot tends to bound back to zero as the power outputs are larger than that in the 2 degree incremental step response. However since the error between the desired and actual position of the steering are still small because of the each of 5 degree increments(figure 7.3), the actual steering will be left in the dead zone again even it tends to bound back in the first plays, unless a large-enough integral term of error piles up to exceed the motor’s dead zone.

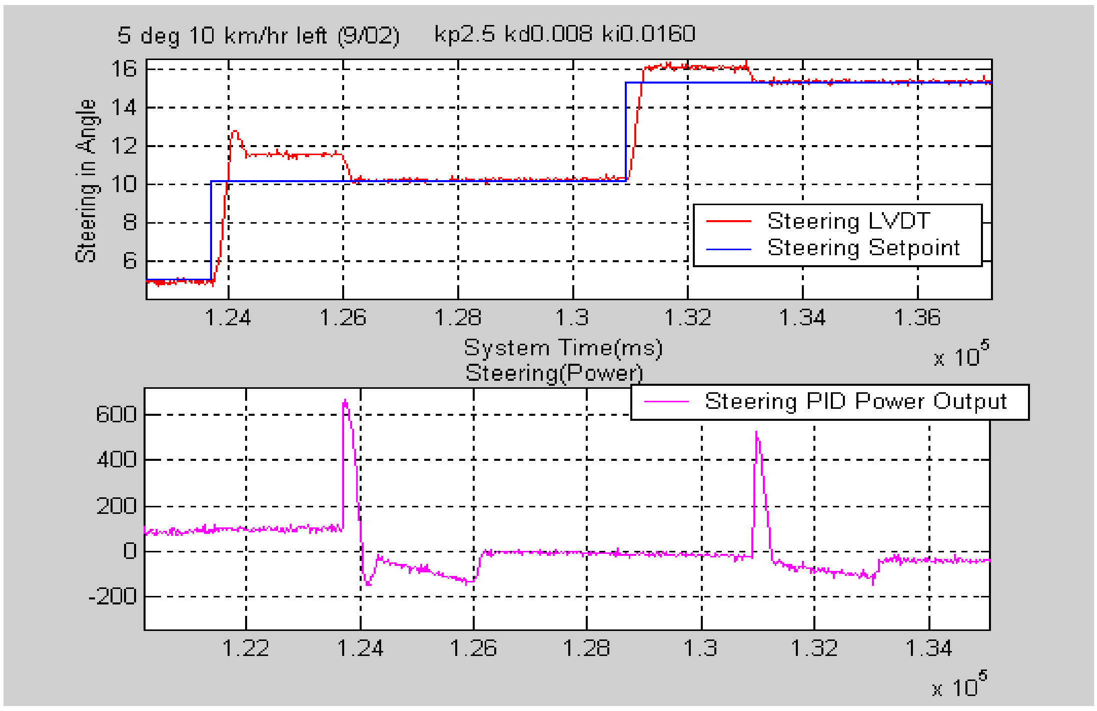
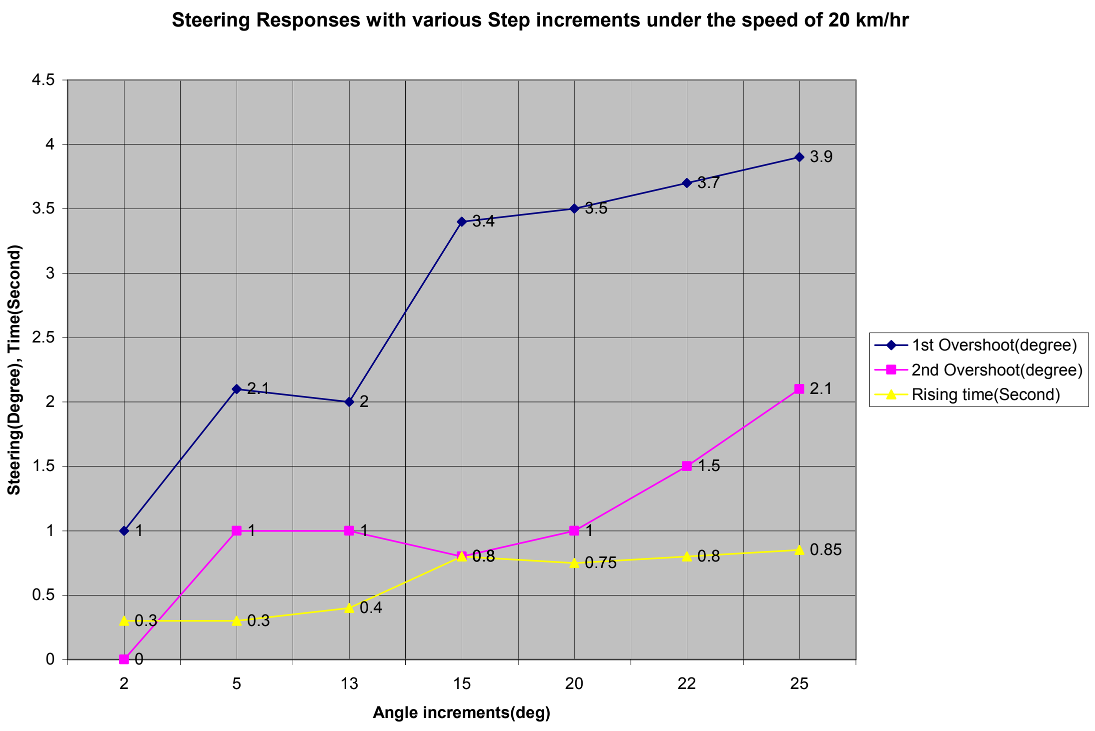


Figure 7.3: 5 degree increments

Statistic shows that a larger incremental step response often causes a larger overshoot and a longer Rising time in the steering system. And speed of the vehicle is no relevant to this characteristic. Below shows a conclusion of the steering response under the similar speed but different in step angle increments.

Step angle increment (degree) under 20km/hr	1st Overshoot(degree)	2nd Overshoot(degree)	Rising time(Second)	Steady state error(degree)
2	1	0	0.3	0.3
5	2.1	1	0.3	0.2
13	2	1	0.4	0.1
15	3.4	0.8	0.8	0.5
20	3.5	1	0.75	0.4
22	3.7	1.5	0.8	0.5
25	3.9	2.1	0.85	0.4



It may be because the torque turning the steering wheel is not in a perfectly straight line relationship with the power in counts calculated by the steering PID controller. As the error between the desired position and actual position are getting larger when increment of step response increases, a lesser output power calculated by Steering PID controller is required to maintain the same overshoot and rising time as in a smaller increment step response.

4.4 Motor Characteristic Related Steering Response

One of the most un-linear characteristics of the motor is the motor's dead zone. As it is mentioned in previous section, a motor's dead zone indicates the power region that can not provide enough torque to turn the steering wheel while the motor is still trying to rotate. It is more often to be seen in the small angle increments step-input applications.

As in the experiment of 2 degree increment step response below in figure 7.4, the motor's dead zone has its positive end of the spectrum of 100 power outputs in counts, with a dead zone time of approximately 3 seconds. A dead zone time indicates the time it takes the power output to stay inside the dead zone's region.

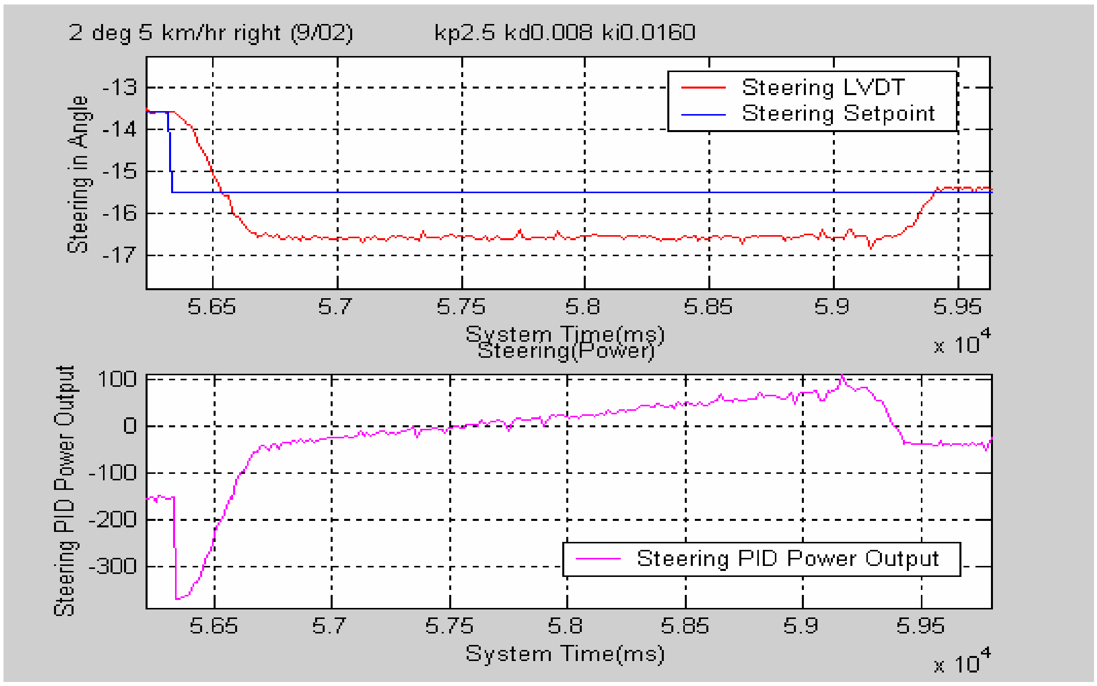


Figure 7.4: 2 degree dead zone analysis

Around another 3 seconds later, the steering response is recorded with slightly a bit different in its overshoot, dead zone region and dead zone region.

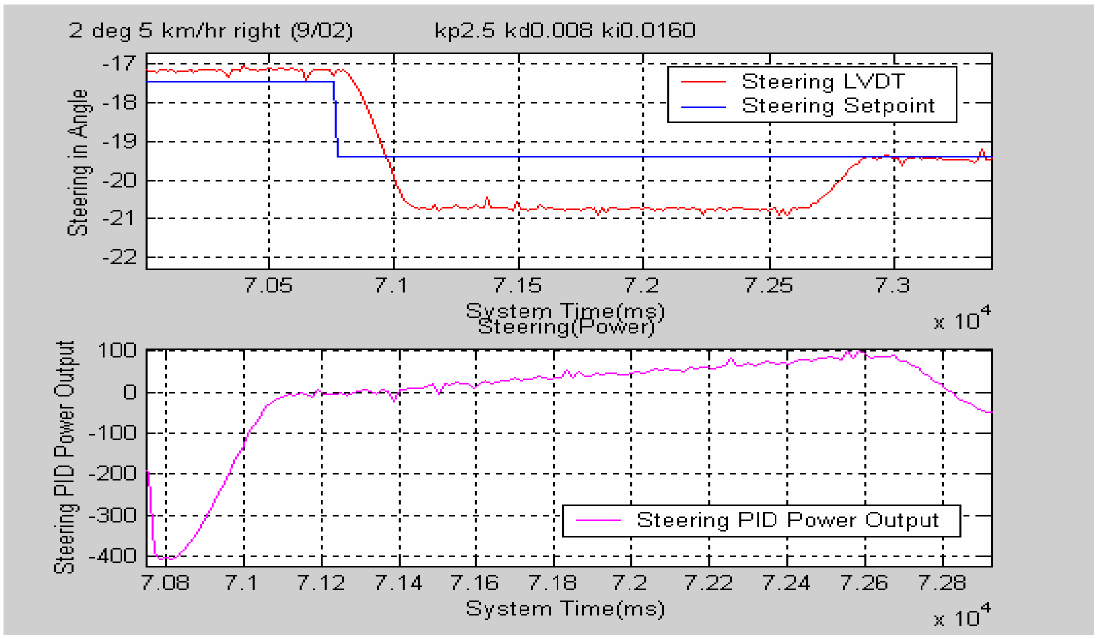


Figure 7.5: 2 degree dead zone analysis

At the system elapsed time of 72.62 second in figure 7.5, the power of the motor exceeds the dead zone as power output reaches to 90 counts. Steering starts to move again. In the same experiment, at power output reaches the end of the negative spectrum at -200 in counts.

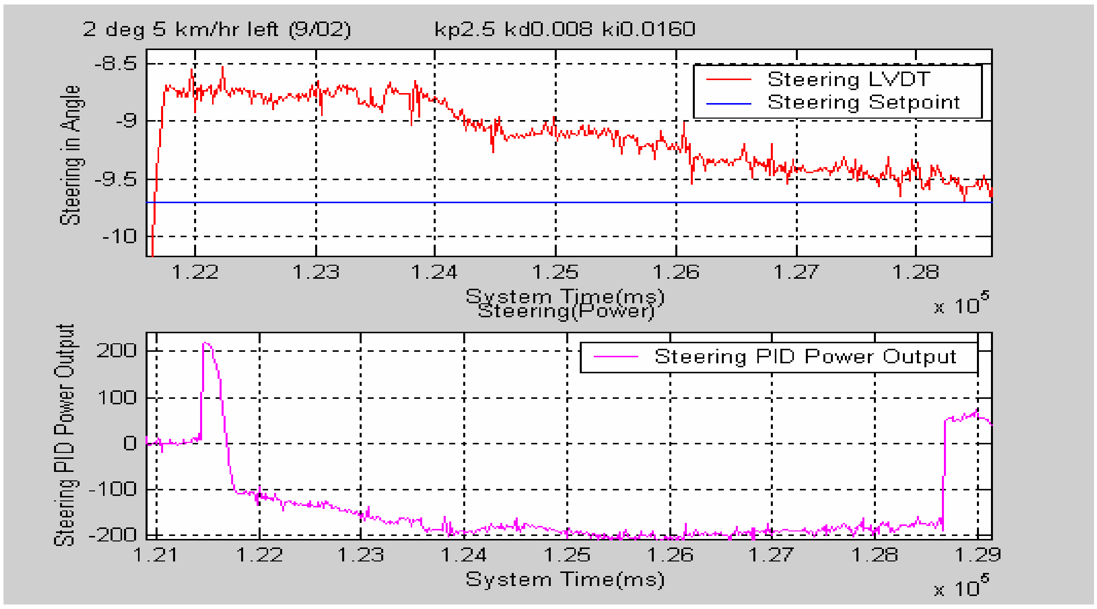


Figure 7.6: 2 degree motor dead zone analysis

In the figure 7.6, the output power doesn’t seem to return back to zero as it should be when steering wheel starts to turn again. The reason for this behavior may be caused by the integral constant error related steering response. It usually occurs when as the integral term error continuously to pile up over a long period and at constant at it’s maximum peak. When it happens, the motor will continuously sending power constantly when proportional term error and derivative term error drop down to zero.

4.5 Integral Terms Related Steering Response

An integral term related steering response often results in a continuous increase in the power output to the motor. It is usually in a form of a jump in steering suddenly appeared during the steady state. The figure shown blow shown an integral term related steering response. As it is mentioned before, an integral term related steering response often appears in a small angle increment step response. It often related to the problems occurred by motor's dead zone as it cause the power output suddenly exceeds dead zone spectrum causing a jump in the response of steering.

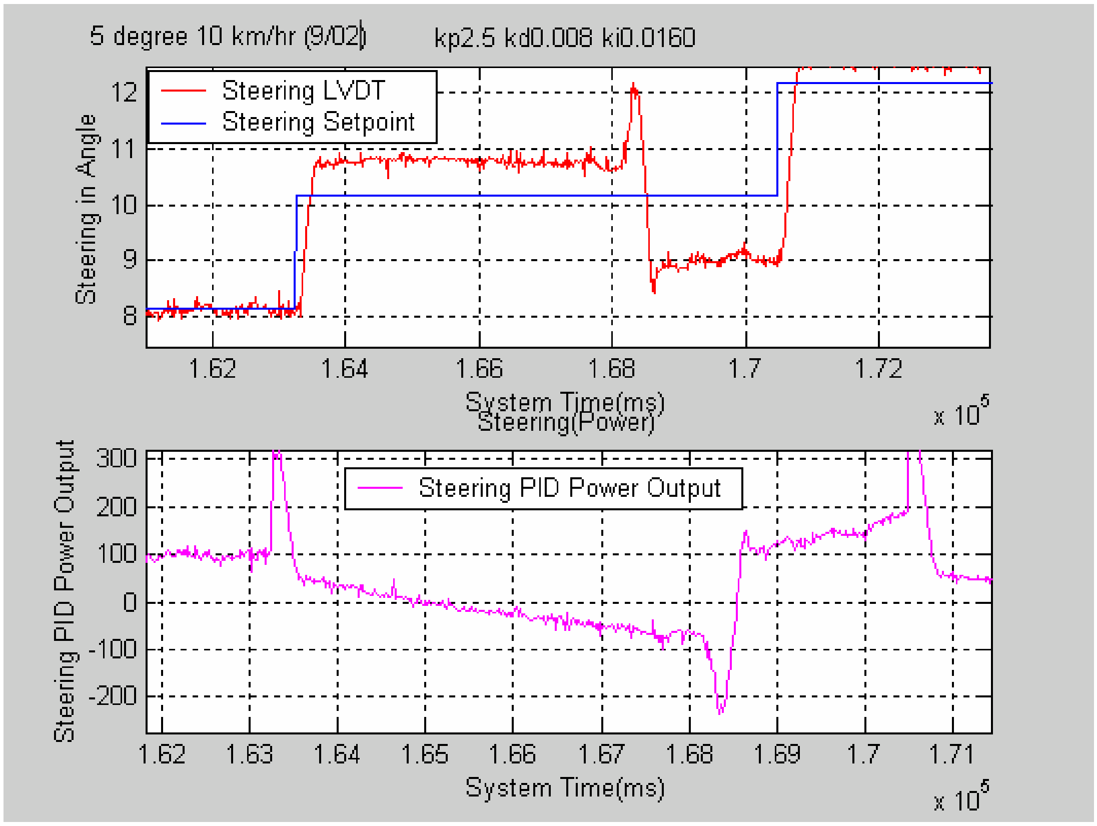


Figure 7.7: integral term related steering response

A integral term related steering response can be avoided by using a flag to set it down to zero each time a new set point is inputted into the PID controller.

Chapter 5

HSV Safety issues

5.1 Introduction

As the research of the high speed vehicle is getting more and more advanced to any other step, the requirements of more dangerous tasks or experiments are in demands. Issues on the safeties of people or their surrounding environments involved into the experiments are essential and can not be ignored.

Because the research on navigation system in HSV project involves lots of online tuning process of both steering and speed. Especially for low level control of the vehicle as most of its works need to be dealing the most on no-linear behaviors in both steering and speed control responses. Sometimes a waiting accident is often likely to occur. The question on how to improve the safety for more and more experiments ahead of the project research just becomes one of the most important issues in the project HSV.

It is everyone’s responsibility to improve the safety of our thesis working environment and passes our knowledge of safety down to next generation of young students like us.

In order to maintain and improve our safety during the thesis work, there are certain rules needed to be notified and obeyed.

5.2 Emergency procedure in HSV Running Experiments

There are rules during any run tests are experiments should be kept in mind always. They are especially important to the students that involves in the work on controlling the behavior of the vehicle.

Rules list below should be checked before any experiment started.

Hardware and Personnel:

- Have at least 2 people in the vehicle and the other person has be a member in the HSV team.
- Light up the alarm lamp on top of the vehicle
- Fasten seat belt always
- Push the emergency light to green color
- Make sure there's enough voltage in both batteries
- Double check actuations condition and attachment
- Make sure the driver seat has been adjusted to the best position for any unexpected emergency procedure.

Software:

Check all other Microsoft Windows programs are closed when only interface is running. New software always needs to be double checked with other team member before any on-running tests and experiments.

When accident happens

Driver

- Stepping down the brake paddle immediately while switching off the steering wheel clutch disengage button
- Turn off the engine

Passenger

- (Optional)Push the emergency buttons built in any User interfaces.
- Pull up the handbrake

5.3 Concept Design of an emergency system

A Concept design is undertaken to implant an emergency override system for the brake and accelerator actuation systems.

In the early stage of the HSV project, the issue of safety during the road testing is ineligible. It is more likely to have program crashes or even the electrical failures while the vehicle is still in driving automatically.

The problem is the brake paddle and butterfly throttle valve are connected to an electronic worm gear. Since worm gear can not transmit power backwards when the electricity is dead, the actuator will get stuck and stop moving. In the other word, when program or electricity fails, cylinder of the actuator can not be moved back, it is more likely to have difficulty to control the vehicle back to manual.

Similar problem has been solved for the steering control by using a electro-magnetic clutch to separate the gear attached to motor when accident happens.

Previous Thesis student have chosen an electrical trigger to disengage the actuations for brake and throttle under the control of computer. By stepping on a button-like panel, it triggers an override voltage to the actuation system to close off the butter fly valve and release the brake paddle. That eventually did not solve problem when electricity went dead.

The cables that attached to the front end of two actuators respectively need to be pulled to open the throttle valve to accelerate or Set-brake to the vehicle. Therefore they need to be released when the system emergency procedure takes place. Therefore to release the necessary cables to close the valve and release the brake paddle.

The concept idea for this design comes from how to increase the current length of the cable to release the bake paddle and close butterfly valve. All that is needed is to create the length of 5 to 6 center meter of cable traveling distance after an emergency button has been pushed.
The hardware listed below is designed to move the entire actuation system forward 6 cm in the same effect of creating extra length in cable traveling.

Briefly, there are three parts of components as in figure 8.4:

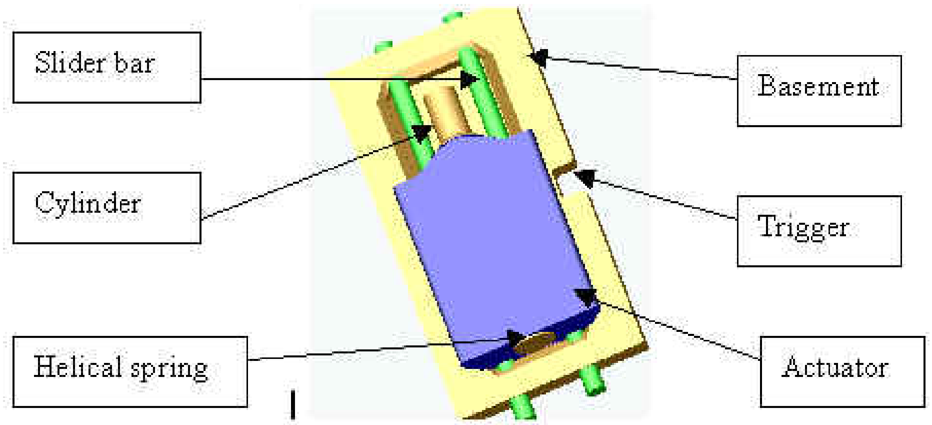


Figure 8.4: Components of emergency-buttoned actuation system

Basement plate - Consists of a thin mounting plate, 60X20X2 (length X width X thickness in cm) Gray Cast Iron. It also consists of two slider bar 0.5 in diameter, 50 in length with a clearance of slot 1 cm in depth. A mechanical switch should be built in one side of the slot as emergency button.

Actuator - Using either a new solenoid actuator or existing worm gear actuator does not matter (40X15X10), with two 0.7-0.9 diameter holes at the bottom. There should be 4 bearings built inside hole and leave still enough space to allow slider bar passing through.

Helical Compression Spring - provide 0.5 to 1 Kg to push slider forward. Fully compressed length 4cm fully expended length 10cm.

Trigger - The trigger is in terms of a button which when it is pressed, the pre-compressed spring will be release causing a force to push actuation forwards.

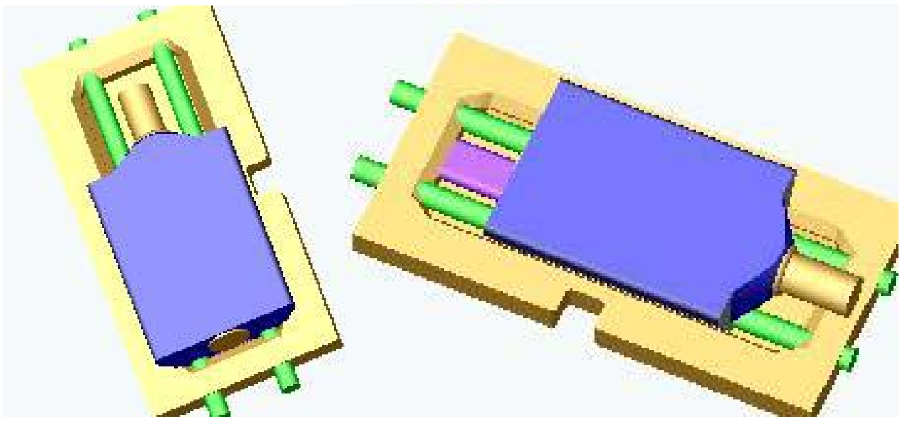


Figure 8.5: Functionality of emergency-buttoned actuation system

Functionality:

When the button is pushed, Actuator sitting on two smooth slider bar will be pushed forward 6 cm by a helical spring on the back to cover the required cable length for complete System override. It is shown as below in figure 8.5

Conclusion:

In conclusion, the design is completely free and independent from any programming logic and electrical limitation. In other words, it is independent from software and electronic devices and when accident happens, it can be the reliable break down system for the emergency procedure. The design of it has not be completed and more work or new idea are required to solve the emergency problem existing in our actuation control.

Chapter 6

Conclusion

In conclusion of Low-level control at the end of 2002, the PI steering controller has been found to have a better response than a PID controller for the steering control system. The gains are tuned as Kp=3.276, Ki=0.006102. it provides a good steering response of 1.7 degree of overshoot and 0.2 seconds in overshoot time. But motor characteristic such as dead zone related steering response is more likely to produce a big steady state error in the steering control system. Nevertheless, the best performance of the steering response is found under the same gains which has zero in overshoot and 0.2 in steady state error.

For the dead zone related steering responses along with other no-linearity related steering responses such as speed related and increment of angle related responses are to be explained briefly.

For the speed control, the algorithm is implemented into the Hyperkernal but did not have the time to accomplish the on-line task. And two position-controlled actuators are ready to be fine-tuned. The efficient ranges of the working actuations are outlined.

In the middle of the year, motor for controlling the steering has been corrected to provide the maximum current for motor.

The path following algorithm is written as well as a user interface used to fine tune the path following results.

Further more work in the development of low level control involves the speed control and path following tuning process. They can not be approached if the requirement of the safety in our working environment has been proven to be reliable.

Reference:

Kelly, A. (1995), “A feedforward control approach to the local navigation problem”, Technical report, Carnegie Mellon University, The Robotics Institute

John G. Bollinger and Neil A. Duffie, “Computer Control of Machines and Processes”, Addison-Wesley,1993

Thomas D. Gillespie, “Fundamentals of Vehicle Dynamic ”, Society of Automotive Engineers,1992

Advanced Motion Controls, “users manual for 25A8 series PWM servo amplifier”

Huosheng Hu, “LICA User Manual – A Locally intelligent Control Agent for Building Complex Control System Version 1.1”, Robotics Research Group, University of Oxford, 1994

S.R. Earle, “Braking of Road Vehicle”, Proceedings of the Institution of Mechanical Engineers, 1993

Michael J. Young, “Mastering Visual C++ 6”, text book for C++ programming

Ivor Horton, “Beginning with Visual C++ 6”, published by Wrox Press Ltd, 1998

Mike Blaszczak, “Professional MFC with Visual C++ 6”, Published by Wrox Press Ltd 2001”