THE SMART CAR PROJECT:
A CASE STUDY IN COMPUTER-MEDIATED CONTROL

by

O. Daniel Gott

A project submitted to the
Faculty of the Graduate School of
State University of New York at Buffalo in partial
fulfillment of the requirements for the degree of
Master of Science

May 6, 2003

ABSTRACT

Currently successful execution of many human-in-the-loop manipulation tasks directly depends on the operator's skill or a programmer's knowledge of the presumed environment in which the task will be performed. Computer mediation of human inputs can augment this process to: (i) permit easy and rapid incorporation of local sensory information to augment performance, (ii) provide variable performance (precision- and power-) assist for output motions/forces, and (iii) hierarchical distribution of control and graceful degradation. Such mediated control has enormous potential to both reduce operator error and permit incorporation of greater autonomy into human/robot interaction.

We chose to examine two sets of tasks to help enhance a remote operator's performance: one involving precision in setting the forward velocity in the presence of variable loads/disturbances and the other improves safety by assisting the operator to avoid obstacles by carefully mediating the operator's joystick inputs. The overall goal is to endow a set of local reflexes that use local sensory information to override the user's input in order to enhance security, safety, and performance. In particular, we implement and evaluate the paradigm of mediated control for remotely driving a mobile robot system (a 1/10 scale remote control truck equipped with Basic Stamp 2 microprocessors) that will serve as our scaled inexpensive testbed. We discuss various aspects of the design and implementation of the Smart Car mobile platform. Beginning with the process of selection of the mechanical platform, we will discuss our motivation and reasoning for making several of the design choices necessary to create the testbed in the subsequent sections. In addition, we discuss the issues pertaining to implementation of two controllers. One for obstacle detection and the other for Wheel RPM PID control. We present the motivation and detail of the actual implementation. We use commercial off the shelf (COTS) hardware and integrated the whole system. Validation and calibration is a very critical step in the whole process. While an ad-hoc solution approach would also suffice for general demonstration purposes, we are interested in using the Smart Car as a scaled testbed. In particular, we are also interested in obtaining quantitative data and hence we spent considerable time validating our system. We first calibrate the independent subsystems, which include the transmitter/receiver, sensors, and then consider the calibration of the controller's interaction with the entire system. A test setup with an external reference tachometer was created to validate open and closed loop response of the various controllers.

Such mediated control has considerable significance and application in a wide range of applications ranging from "Smart Highway Systems" to semiautonomous exploratory rovers.

ACKNOWLEDGEMENTS

I have had the opportunity to work with a great group of knowledgeable people that made my project a pleasurable and exciting experience. Their help and support throughout my Masters Project is greatly appreciated. I would like to thank my advisor, Professor Venkat Krovi, for his help throughout the project especially during the documentation phase and always helping me work through difficulties. I would also like to thank Professor Roger Mayne for taking the time to review my project and attend my final presentation. I would like to give a special thanks to Seung-Kook Jun for his support and continued innovations with the Smart Car development. I would also like thank Rajankumar Bhatt, Pravin Nair, Chin-Pei Tang and all the members of the Automation Robotics and Mechatronics (ARM) Laboratory for all their support and providing assistance in the lab. Finally, I would like to thank Chii-Chuan Chen for finding time to help throughout my project especially in the System Lab.

CONTENTS

Chapter 1. Introduction:

The mediated control of operation of various engineering systems is of tremendous interest to many application arenas. For example, several companies in the automotive industry such as Motorola, Delphi Automotive Systems, and DaimlerChrysler are researching mediated control systems that can enhance overall performance/safety of driving [6]. The proposed mediated control systems would use a variety of sensors to monitor both the driver's actions such as steering and braking patterns as well as the environment such as road and traffic conditions. This information would then permit them to respond to adverse driving conditions far more rapidly than would be possible by the driver alone. Mediated control systems become an absolute necessity in order to integrate intelligent vehicles (a.k.a smart cars) such as these into the smart highways of the future. Similarly, considerable research activity is currently underway to create Intelligent Vehicle Highway Systems (a.k.a. Smart Highways) [8]. These Smart Highways are intended to solve traffic congestion by allowing vehicles to travel more closely, assist drivers merge into traffic, and pass slow moving vehicles [7, 8].

Military and aerospace industries also have a considerable interest in mediated control for improving the success of tasks carried out using unmanned semiautonomous vehicles. As with the automotive systems, the military and aerospace systems can benefit from sensing the environment and either communicating the data to the operator or acting on the obstacle with a preprogrammed sequence of events.

The purpose of this project was to create a mobile robot that operates in real-time with a mediated control system to increase the robots autonomy. The robot operates remotely through a transmitter, which sends a signal to a receiver that passes the signal to a microprocessor that executes a control algorithm. The microprocessor outputs control signals to drive and steer the robot. Figure 1.1 below shows the standard configuration where the command sent from the transmitter directly controls the mobile platform.

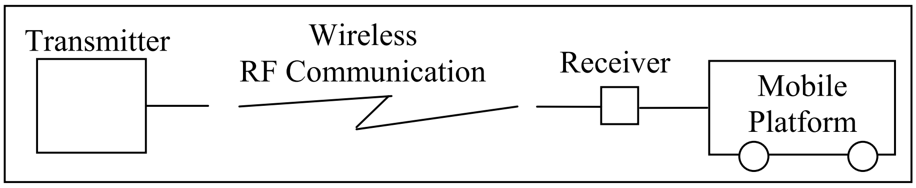


Figure 1.1: Command directly passes to mobile platform

With the configuration shown in Figure 1.1, the operator has direct control over the mobile platform, which means that the success of a task depends completely on the operator's skill. In this paper, we will investigate the configuration above with a computer inserted between the receiver and mobile platform. The microprocessor will mediate the command sent from the transmitter and the signal sent to the mobile robot. Figure 1.2 shows a diagram of real-time remote operation with microprocessor mediation.

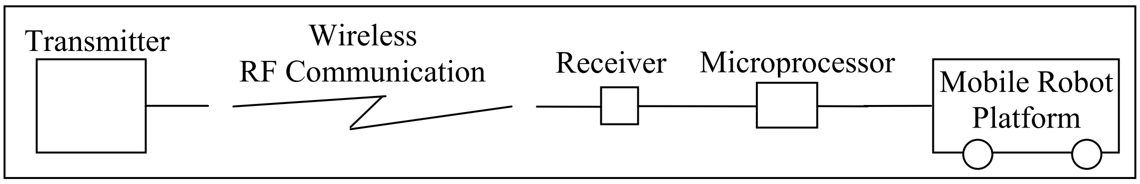


Figure 1.2: Real-time remote operation with microprocessor mediation

Using the configuration in Figure 1.2 allows for computer mediation between the command signal and the output signal sent to the mobile robot platform. Now when an operator executes a task, the microprocessor can assist by sensing the environment and help guide the vehicle through a successful task by avoiding obstacles and intercepting erroneous commands sent to the robot.

Further research is currently on-going using the configuration below.

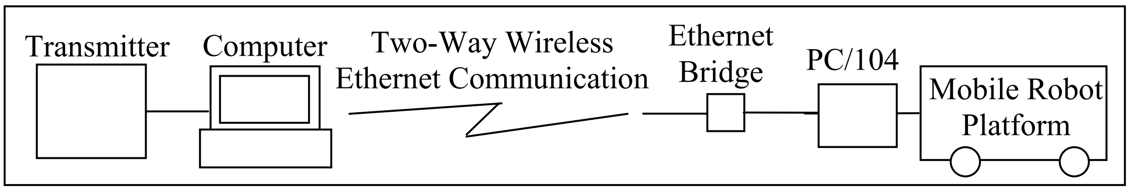


Figure 1.3: Real-time remote operation with computer mediation and two-way communication between the host PC and target PC

The configuration shown in Figure 1.3 allows two-way communication between a host computer (such as a laptop or desktop computer) and the target computer that controls the robot and senses the environment. Ultimately, burden can be taken off the target PC because the host computer can now be used to run control algorithms, log data, and supply a graphical user interface (GUI). Reducing the burden on the target PC is useful to save power, increase real-time sensing speed, and reduce weight because fewer batteries will be necessary to run the target PC.

Therefore, in this paper we will demonstrate the implementation of mediated control on a scaled testbed that will be created using relatively inexpensive commercial off the shelf (COTS) components such as our RC truck, microcontroller, sensors, and actuators. Although the additional control features assist the operator in many ways, the tradeoff lies in the fact that we take some of the control away from the operator. For example, if the mediated control system prevents the operator from steering into an object to the side of the vehicle while this is clearly the better choice when faced with a head-on collision. We will use this scaled testbed to study some of these issues in greater detail without having to build costly full-scale prototypes.

Chapter 2. Design of the Smart Car Mobile Platform

In this chapter, we will discuss various aspects of the design and implementation of the Smart Car mobile platform. Beginning with the process of selection of the mechanical platform, we will discuss our motivation and reasoning for making several of the design choices necessary to create the testbed in the subsequent sections.

2.1 Robot Platform Selection

To begin the project we needed to decide whether to design a mobile platform or buy an existing platform that we would mount microprocessors, actuators, and sensors. For this project, we planned to design a mobile robot that would be rugged enough for the outdoors, yet still have the capability to maneuver in tight areas. We concluded that the focus of the project was not to build the platform, but to interface microprocessors with the drive motors and steering motors to allow mediated control of the mobile platform. This conclusion led to our decision to buy a remote control (RC) truck. The truck must be able to withstand years of use in a rugged environment as well as the ability to easily modify components, add sensors and actuators, and mount microprocessors and/or a PC/104.

The most important design requirement was that we must be able to easily mount microcontrollers and/or a PC such as a PC/104 and the vehicle must be able to support and maneuver with the additional weight of a PC, microcontrollers, sensors, and actuators. The remote control truck should have the option to change suspension spring rate by being able to replace the stock springs with springs of various stiffness to optimize the vehicles response to rough terrain and handling at high speeds. In addition, the shocks should have the option to change suspension damping, possibly by changing oil viscosity in oil- filled shock that will allow tuning of the suspensions response. A desired aspect of the remote control vehicle is easy accessibility to the suspension components allowing active suspension control using actuators. It is necessary that the major electronic components of the remote control truck be separate to permit us to insert a microcontroller between them such as between the receiver and ESC and steering servomotors. The power source for the vehicle must be wireless allowing remote operation and the source must be reusable to be cost effective. Another requirement for the mobile platform is to be able to mount brackets, sensors, and actuators to the vehicle. Some materials such as carbon composite material may not be easily drilled and tapped and may crack. It is necessary that we be able to buy replacement components such as differential gears, ball joints, bearings, etc. The transmitter and receiver would preferably be able to control additional actuators beyond the necessary drive and steering motors. For example, a six-channel transmitter with a corresponding receiver has the potential to control four additional actuators for robotic arms, active suspension, etc. The vehicle would ideally have the option to steer both the front and rear wheels separately allowing for tight turning radius and movement in a diagonal direction when both the front and rear wheels are turned in the same direction.

Figure 2.1 is a picture of the Tamiya TXT-1 RC truck that we chose, which has all of the features listed above and has proven to be a very versatile and durable robot platform for our scaled testbed.

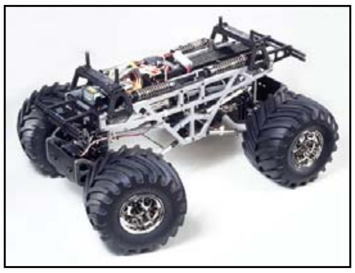


Figure 2.1: TXT-1 with the cover off

2.2 Microcontroller

The intention was to keep the microcontroller in this project small and light enough to mount on the robot platform with an adequate processing speed capable of reading sensors while acting in real-time to mediate the input command. To allow us to modify programs during the debugging stage and to optimize the control code we need an electronically erasable programmable read only memory (EEPROM). The capacity of the EEPROM and random access memory (RAM) must be large enough to run a control algorithm such as a proportional integral derivative (PID) controller.

Because of our familiarity and ease of programming/debugging, we chose the Basic Stamp 2 (BS2) microcontroller mounted on the Board of Education (BOE) programming board, which are both produced by Parallax Inc. The BS2 is very lightweight, only requires a 6V source, is inexpensive, has a 20 MHz processor, 2KB EEPROM, and 32 bytes of RAM which give the microcontroller sufficient performance for our application. To program the BS2 we use a modified version of BASIC programming language developed by Parallax called PBASIC. Once programmed the BS2 allows relatively easy interfaces such as asynchronous and synchronous serial and I^{2}C/SPI. The microcontroller can be used as an intelligent subsystem to handle sensors and actuators and can be seen as a smart peripheral device when communicating with a high a PC/104.

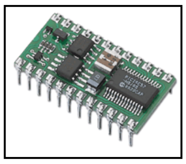


Figure 2.2: BASIC Stamp 2 microcontroller

2.3 Communication Medium

This paper will focus on the configuration shown in Figure 1.2 where the command signal sent from the transmitter to the receiver is 75 MHz frequency modulation (FM) and the microprocessor transmits data to a desktop computer via wireless 900 MHz RF modem.

Wireless communication does not require line-of-sight to operate like infrared (IR) signals and this was one factor in our selection. Another design requirement was that the data transfer rate must be sufficiently fast to operate a remote vehicle in real-time. Although two-way wireless RF communication is possible using the SuperSCREAMER RF data modem that we used in our project, in this paper we focus on one-way data transfer. Our goal was to implement wireless communication with a range of operation near 100 feet, which does not appear to be a problem because most transmitter/receiver pairs that we have considered allow ranges up to 300 feet.

To retain the mobility of our Smart Car while using the data acquisition software we decided to use a wireless RF data modem by Ewave, Inc. The SuperSCREAMER is a drop-in replacement for a serial cable that supports five protocols including PBASIC, which is optimized to enable wireless debugging and programming of the Parallax Basic Stamp microcontrollers. The SuperSCREAMER has an operating range of 300 feet, which is more than sufficient for the Smart Car. The wireless capability significantly speeds the edit-compile-debug cycle for applications in mobile robotics, inaccessible locations, or hazardous environments.

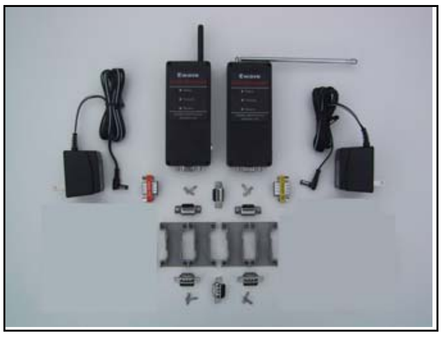


Figure 2.3: SuperSCREAMER RF data modem developers package

2.4 Sensors
To mediate control of the robot we need to be able to read data for wheel speed and obstacle detection. The sensors for wheel speed must be able to read rpm at full speed yet detect small enough movement for position measurement if desired. The obstacle detection sensors must be able to detect an obstacle within a distance allowing the necessary time to act on the problem. We created composite sensors such as the obstacle detection array of infrared rangers and encoders that use photoreflectors to detect wheel movement. We only need to use simple sensors in the initial stages of experimentation to demonstrate the concept of mediated control. One of the Sharp GP2D02 infrared rangers that we chose to create our obstacle detection array is shown in Figure 2.4.

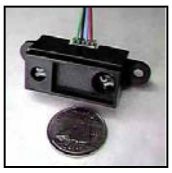


Figure 2.4: Close-up of the IR ranger

Figure 2.5 shows the Hamamatsu-P5587 photoreflector that we used to create a composite encoder, which detects black and white stripes on our encoder strip mounted inside the wheel. The encoder design is required to be able to read RPM at the vehicles full speed, which is approximately 1000 RPM. All of the sensors must be robust and work in rough terrain.

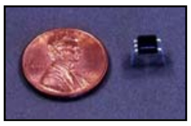


Figure 2.5: Photoreflector used in the design of the encoder

2.5 Interface Board Design

To begin studying the mediation of command signal, we mounted two Basic Stamp 2 microcontrollers on a metal base plate and designed an interface board to protect the BS2's form high current surges. We used a standard 7.2V RC battery to power the BS2's retaining the mobility of the RC truck while keeping the project cost effective by using the rechargeable batteries. The picture in Figure 2.6 shown below was taken during the early stages of the Smart Car development.

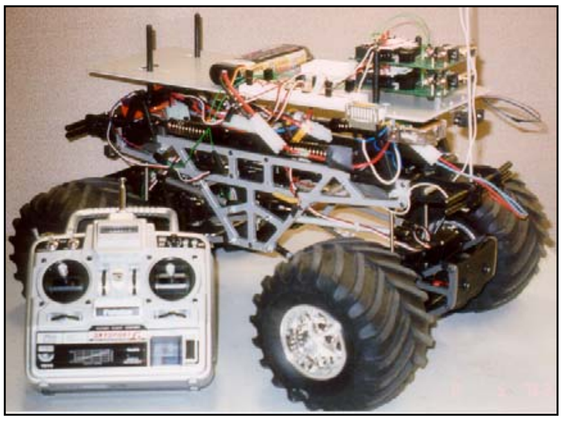


Figure 2.6: TXT-1 with BASIC Stamp 2 microcontrollers

Since we are working with two difference power sources, we decided to create an interface board to protect the microcontroller from high current surges. We achieved the current protection using optical isolators with photo diodes to separate the receiver battery power circuit and the BS2 power circuit. The two circuits are separated using optical isolators because differences in the receiver ground and the BS2 ground may cause high current damage to the microprocessor and other IC chips [9]. After testing the interface board using a breadboard, we found that the voltage sent from the optical isolator between the receiver and BS2 was too low for the Basic Stamp to read changes in state of the pins from high to low. To amplify the voltage to a readable quantity we used LM358 Op-Amps. The signals sent from the Basic Stamp 2 do not require amplification to operate the electronic speed control (ESC) and the steering servomotors. Figure 2.7 shows the circuit for the interface board including the receiver circuit and the BS2 circuits with the photoreflectors and infrared rangers interfaced. Notice that the receiver signal is passed to the BS2 while isolating the receiver current from the BS2.

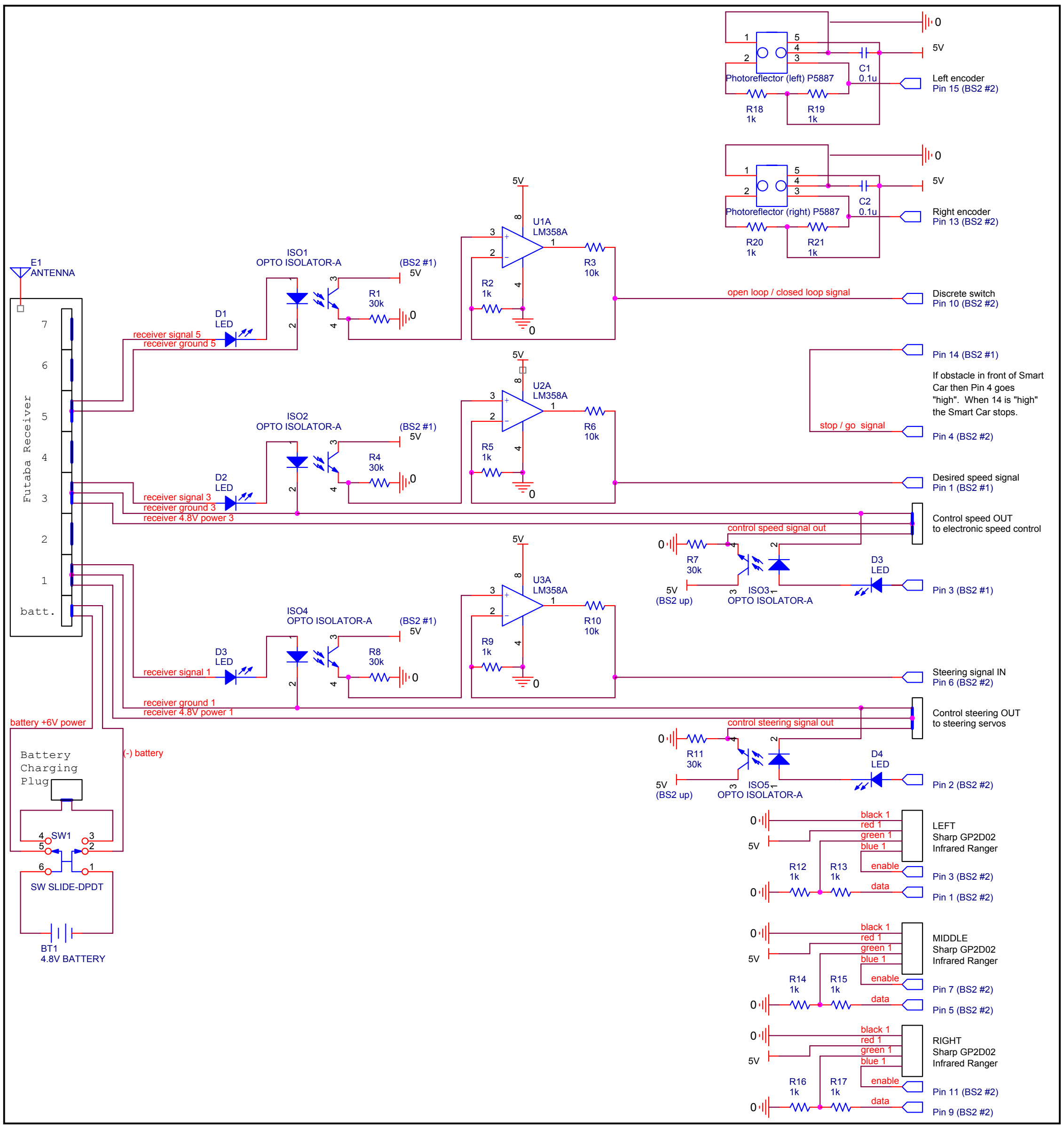


Figure 2.7: Circuit diagram for the Smart Car

The interface board that we designed for current protection to the microcontrollers is shown below in Figure 2.8. This interface board is more robust than our first designs, which used breadboards and the next step to would be to create a printed circuit board (PCB).

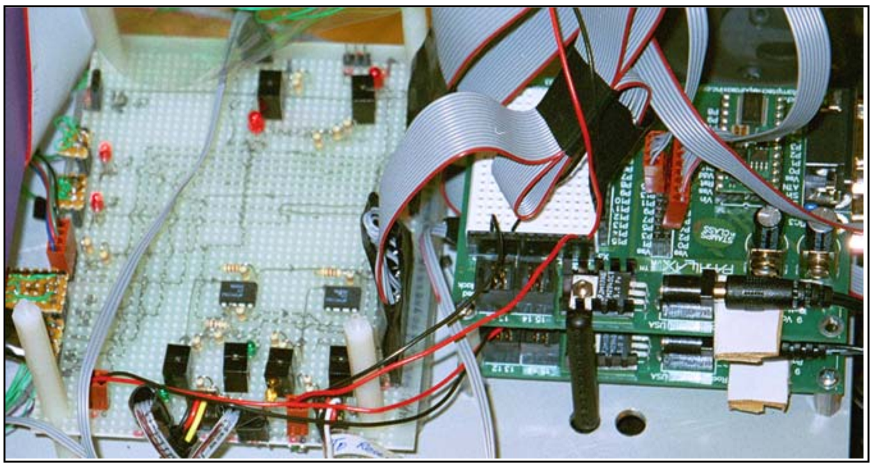


Figure 2.8: Current protection circuit using optical isolators

2.6 Obstacle Detection Array

To demonstrate mediated control we designed a composite sensor that we call an "obstacle detection array" which is interfaced with one of the Basic Stamp 2 microcontrollers. The obstacle detection array is composed of three infrared rangers shown in Figure 2.9 and the interface shown in Figure 2.10 is easily implemented on the Smart Car [9]. Figure 2.11 shows how we mounted the infrared rangers to detect obstacles in front of the vehicle as well as obstacles to the left and right sides of the vehicle. The Sharp GP2D02 infrared ranger not only detects obstacles but also can be used to measure distances up to 80cm.

Figure 2.9: Close-up of the IR ranger

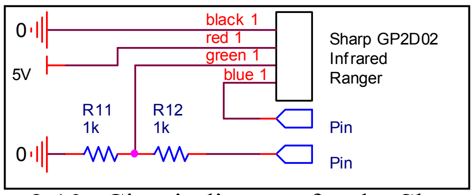


Figure 2.10: Circuit diagram for the Sharp GP2D02 infrared ranger

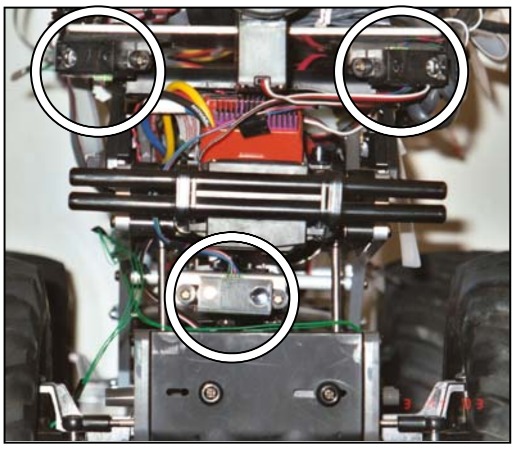


Figure 2.11: Obstacle Detection Array of IR rangers

Using the obstacle detection array we can stop the Smart Car when an obstacle is in the vehicles forward path and prevent the operator from steering into obstacles to the left and right side. This idea has several practical applications in automotive and military industries such as preventing rear end collisions, collision caused by blind spots especially with large vehicles like busses, tractor-trailers, and military tanks [6, 7, 8]. Due to the nature of the size of military tanks, most use two people to maneuver through

rough terrain; one person at the controls and the other as the lookout. Using an obstacle detection system such as this may reduce operator error allowing the driver to pay attention to other complicated tasks such as determining whether an approaching vehicle is friend or foe.

2.7 Encoder Design

To facilitate the operator in driving the Smart Car at the precise speed chosen by moving the joystick on the Futaba radio control we decided to design a feedback controller. The digital controller nearly eliminates the need for the operator to compensate for changes in speed by adjusting the joystick position when the Smart Car is subjected to various loadings such as rough terrain, hills, towing, low battery power, etc.

To implement a feedback controller we needed to measure the actual wheel speed of the Smart Car. Using the small photoreflector shown in Figure 2.12 to read an encoder strip with black and white stripes mounted inside the wheel we were able to measure the vehicles actual wheel speed. Without augmentation, the RC truck will lose speed when put through a loaded condition even though the operator may keep the joystick at a constant position, which is why we need to measure the actual wheel speed allowing the controller to reduce the error in desired and measured wheel speed. The Hamamatsu photoreflector is relatively easy to interface and requires only one pin as shown in Figure 2.13.

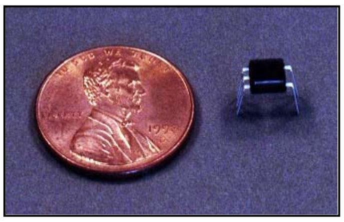


Figure 2.12: Photoreflector used in the design of the encoder

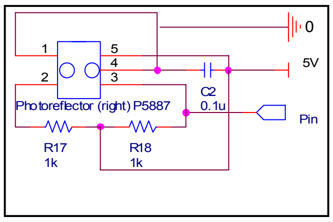


Figure 2.13: Hamamatsu-P5587 Circuit

We mounted the entire photoreflector circuit to the axle of the Smart Car between the wheel and the axle as shown in Figure 2.14 and Figure 2.15. Encoders are often mounted on the drive motors to increase resolution however; the Smart Car has open differential axles, which required us to measure the actual wheel movement. The open differentials allow one wheel to spin at a different rate than the other wheel even though both wheels are on the same axle, therefore the individual wheel speeds cannot be determined by measuring the drive motor RPM.

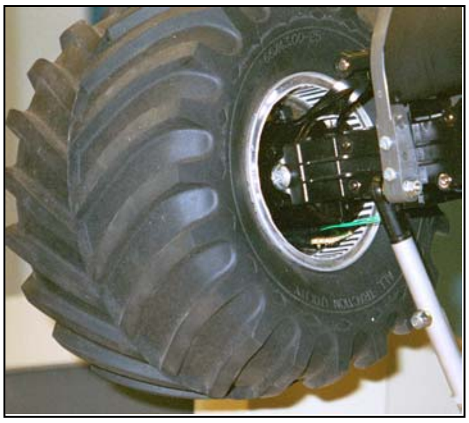


Figure 2.14: Hand built encoder Installed on the Smart Car

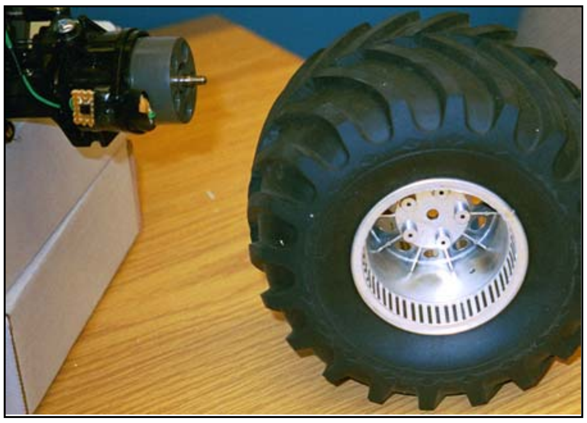


Figure 2.15: Encoder mounted on the axle with encoder strip inside wheel

When a black stripe from the encoder strip passes by the photoreflector, a signal is sent to the microcontroller changing the pin state to "high" until the wheel rotates further causing a white stripe to be detected. By measuring the time that the pin on the BS2 is "high" we can determine the actual wheel speed of the Smart Car. Figures 2.16 and 2.17 below show the shape of our encoder signal at slow and fast RPM.

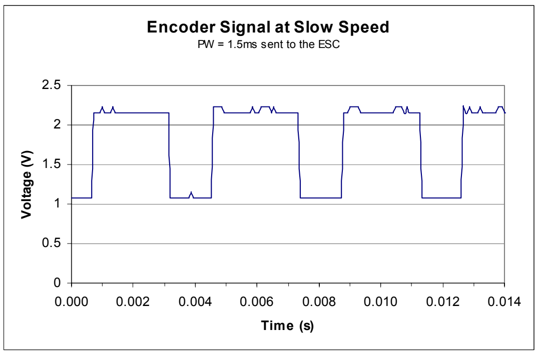


Figure 2.16: Encoder signal at slow speed

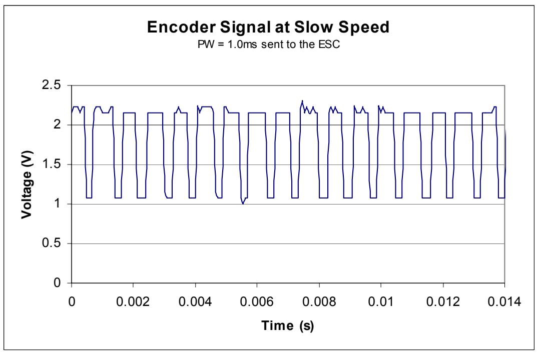


Figure 2.17: Encoder signal at fast speed

2.8 Specific Architecture of the Smart Car

The specific architecture of the Smart Car is similar to the block diagram shown in Figure 1.1 where the command signal is transmitted from the Futaba radio control to the receiver, which passes the signal to the steering servomotors and electronic speed control (ESC).

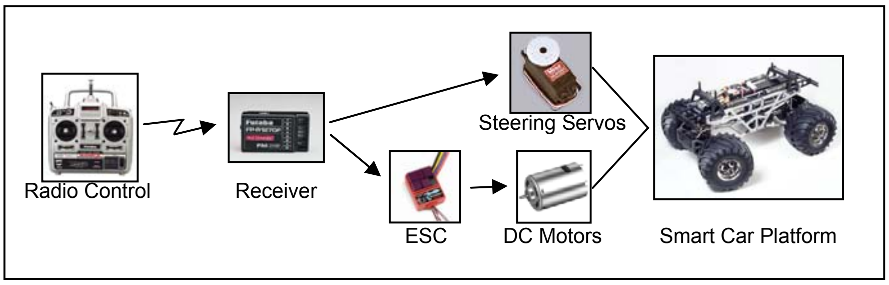


Figure 2.18: Smart Car Architecture without Augmentation

In this project, we have augmented the architecture shown above in Figure 2.18 with two microprocessors, obstacle detection array, encoders, and wireless RF data modem to transfer data to a desktop computer. A block diagram showing the Smart Car architecture with augmentation is shown below in Figure 2.19.

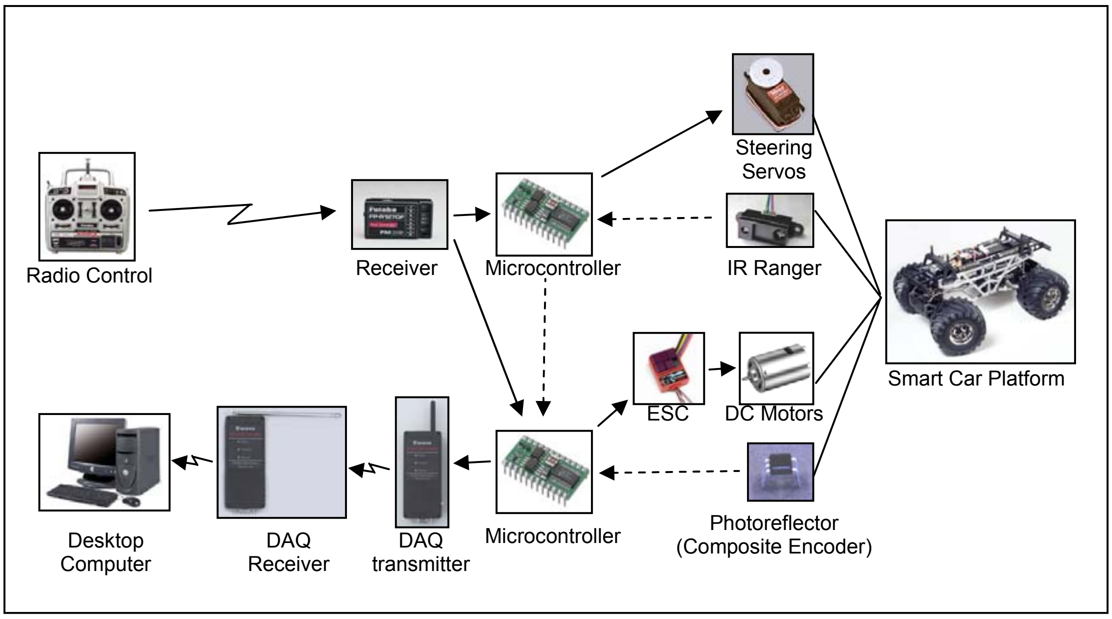


Figure 2.19: Smart Car Architecture with Augmentation

The Smart Car architecture allows the command signal sent from an operator to be modified prior to steering and driving the vehicle based on the various sensor feedback. The wireless data modem can be used to transfer data in real time for plotting or logging while driving the Smart Car and in addition, the RF modem can be used to program the Basic Stamps from remote locations.

2.9 Objective Summary Table/Outline

The table below summarizes our objectives, goals, and the corresponding sections, which describe the milestone.

Table 2.1: Objective summary

Milestone	Objective	Goal	Sections
Robot Platform	Select robust robot platform	Durable enough to drive outdoors on grass or dirt	2.1
	Assemble robot platform and interface with microprocessor	Be able to drive robot using microprocessor	2.5
Signal Analysis	Analyze transmitter signal	Reproduce transmitter signal	4.1
Obstacle Detection	Detect obstacles	Detect obstacle within 2.5 feet	2.6
Sensor Design	Accurate measurement of wheel speed	Maximum error margin of 10%	2.7
Controller Design	Controller design to improve system response to a disturbance	Stable with zero steady-state error	3.2 – 3.3
Overall Success	Demonstration of remote real-time mediated control	Detect obstacles and control wheel RPM in real-time while being given input commands	4.3 – 4.6

Chapter 3. Implementation:

In this chapter, we discuss the issues pertaining to implementation of two controllers. One for obstacle detection and the other for Wheel RPM PID control. We present the motivation and detail of the actual implementation.

3.1 Program Design and Development

Since the Smart Car needs to act in real-time to mediate the input command, the programs must be as concise as possible in order to keep the processing time low. The programs written for this project will have two main objectives that will demonstrate mediated control of the mobile robot:

i. Obstacle detection

   This portion of the project requires writing a program such that when the Smart Car detects obstacles, the vehicle stops, and only allows the operator to reverse before proceeding. In addition, the Smart Car detects obstacles on the sides of the vehicle and prevents it from steering into the obstacle while allowing the operator to continue moving forward or reverse. In open loop mode, the operator is given full control without mediation.

ii. Wheel RPM control using a PID controller

   The Smart Car must track the wheel RPM input or setpoint received by the transmitter, which has been calibrated using no load, even when subjected to loaded conditions. There must be improvement in system response using the control algorithm when compared with the open loop system. Under loaded conditions, the Smart Car must track wheel RPM more closely than without the controller. In open loop mode, the operator is given full control without mediation.

3.2 Obstacle Detection Implementation

The obstacle detection feature of the Smart Car mediates the command signals sent to both the DC drive motors and the steering servomotors. One of the Smart Car microcontrollers is used to steer using the obstacle detection algorithm and the other is used to drive the vehicle with the PID controller for wheel RPM. When the obstacle detection array senses an obstacle within 2.5 feet in front of the vehicle, the smart peripheral device sends a variable to the microcontroller where the program determines if the obstacle is too close. If the obstacle detection algorithm calculates that the obstacle is too close, then the BS2 signals the second BS2 to stop the Smart Car by limiting the maximum pulse width to the neutral position causing the Smart Car to coast to a stop and thus preventing a collision. Since the Smart Car coasts to a stop and the IR ranger maximum range is 2.6 ft., there is a maximum practical speed limit that we can drive the vehicle and we have concluded that a sensor with a much larger distance sensing range would be more practical. When the obstacle detection array senses that the vehicle is moving toward an obstacle, the program only allows the Smart Car to turn the opposite direction or continue driving straight forward or in reverse. When the infrared rangers on the side of the vehicle sense an object, the program sets the maximum pulse width, minimum pulse width, or both depending on where obstacles are sensed. Limiting the pulse width sent to the steering servomotors prevents the operator from mistakenly steering the Smart Car into an obstruction.

The steering servomotors require an update rate around 18ms however; the time to read all three Sharp GPD2D02 IR rangers requires much more time. Therefore, if all three IR rangers are read before the servos are updated; the steering rate is extremely slow. To retain similar steering rate, we send a control signal to the steering servos before checking each of the three IR rangers however, a decreased obstacle detection sampling rate, around 54ms resulted.

An additional feature available to the operator is the option to drive the Smart Car in open loop mode giving the driver full control. Open loop mode may be useful when maneuvering in tight spaces where the closed loop mode may inhibit the operator from positioning the vehicle close to obstacles as intended. Shifting between open and closed loop mode can be done at any time, even while the Smart Car is in motion, by moving a discrete toggle switch on the Futaba radio control. Every 54ms the microcontroller determines the mode that the operator has selected by reading the width of the pulses sent from the transmitter to the receiver, which depends on the position of the discrete switch.

3.3 PID Control Theory

The Smart Car uses a PID controller to regulate the wheel speed, which allows the operator to drive the vehicle at precise speeds set by the position of the joystick on the Futaba radio control [1 – 4]. The PID controller requires the feedback of actual wheel speed, which is obtained from the composite encoder; then the control algorithm compares the desired speed set by the operator with the actual measured wheel speed to calculate the error. The PID program acts on the error and calculates a new control output to compensate for the error in wheel speed bringing the Smart Car to the desired wheel RPM set by the joystick position approaching zero steady-state error.

A brief review of PID control theory from *Dynamic Modeling and Control of Engineering Systems* by Shearer, Kulakowski, and Gardner for continuous and digital systems will follow [1, 2]. The signal produced by a proportional (P) controller is proportional to the control error.

$$u(t) = K_P e(t) \tag{3.1}$$

where K_P is the proportional gain. The transfer function for the P controller is

$$\mathbf{T}_C(s) = \frac{\mathbf{R}(s)}{\mathbf{E}(s)} = K_P \tag{3.2}$$

The steady-state performance and speed of response of a system with a P controller improve with increasing proportional gain K_p. However, increasing the controller gain may decrease system stability.

The steady-state performance can also be improved by adding an integral action to the control algorithm. The ideal proportional-integral (PI) controller produces a control signal defined by the equation

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau \tag{3.3}$$

where K_I is the integral gain. The ideal controller transfer function is

$$\mathbf{T}_C(s) = K_P + \frac{K_I}{s} \tag{3.4}$$

Adding integral control, although improving the steady-state performance, may lead to oscillatory response (that is, reduced system stability), which is usually undesirable.

The stability of a system can be improved by adding a derivative action to the control algorithm. The control signal produced by an ideal proportional-derivative (PD) controller is

$$u(t) = K_P e(t) + K_D \frac{d\left[e(t)\right]}{dt} \tag{3.5}$$

where K_D is the derivative gain. The controller transfer function is

$$\mathbf{T}_C(s) = K_P + K_D s \tag{3.6}$$

The derivative action provides an anticipatory effect that results in a damping of the system response. By increasing the stability margin in this way, it becomes possible to use greater loop gain, thus improving speed of response and reducing steady-state error.

All three control actions are incorporated in a proportional-integral-derivative (PID) control algorithm [1]. The control signal generated by an ideal PID controller is [2, 3]

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{d\left[e(t)\right]}{dt} \tag{3.7}$$

The transfer function of the ideal PID controller is

$$\mathbf{T}_C(s) = K_P + \frac{K_I}{s} + K_D s \tag{3.8}$$

The ideal controllers in Equations (3.5) to (3.8) are not physically realizable [2, 3]. All real controllers have a transfer function incorporating at least a fast first-order lag term

$(1+\tau_{pr}s)$ in the denominator [1]. Inclusion of this lag term becomes important in order to program the controllers for digital computer simulation.

Determining optimal adjustments of the control gains K_P, K_I, and K_D is one of the basic problems faced by control engineers. The Ziegler-Nichols tuning rules are one of the simplest procedures developed to select the PID gains. There are two versions of this method: one is based on the process step response and the other on characteristics of sustained oscillations of the system under P control at the stability limit. The first method, based on a delay-lag model of the process, can be applied if the process step response data are available in the form shown below in Figure 3.18.

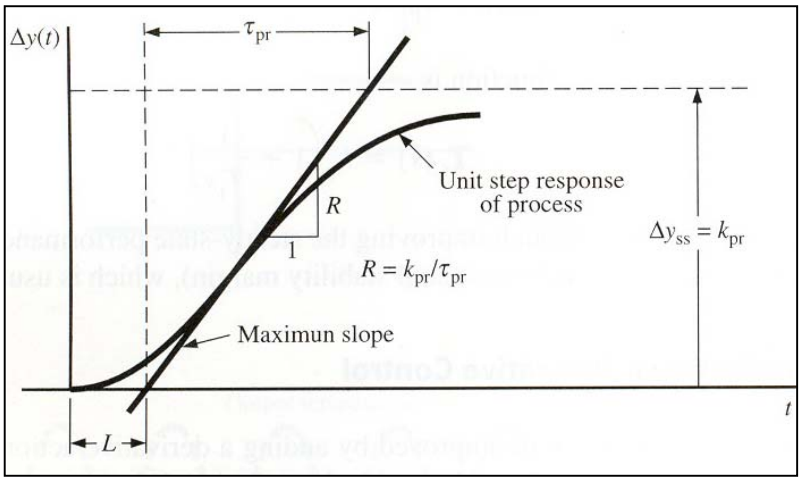
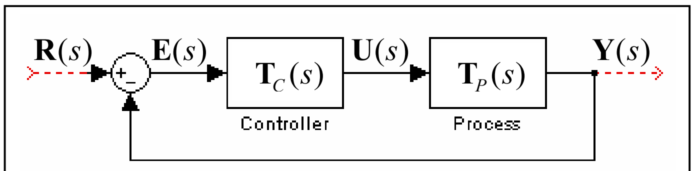


Figure 3.1: Process response graph used to determine PID Gains using Ziegler Nichols method [1]

The transfer function block diagram for the system shown in Figure 3.1 is shown below in Figure 3.2.



$$\mathbf{T}_C(s) = K_P + \frac{K_I}{s} + K_D s$$

$$\mathbf{T}_P(s) = K_{pr} \frac{e^{-Ls}}{\tau_{pr}s + 1}$$

$$\mathbf{T}_P(0) = K_{pr}$$

Figure 3.2: Transfer function block diagram for the system [1]

The controller parameters are calculated using the values of slope R and delay time L of the unit step response as follows [1]:

i. For P controller

$$K_P = \frac{1}{RL} \tag{3.9}$$

ii. For PI controller

$$K_P = \frac{0.9}{RL} \tag{3.10}$$

$$K_I = \frac{1}{3.3L} \tag{3.11}$$

iii. For PID controller

$$K_P = \frac{1.2}{RL} \tag{3.12}$$

$$K_I = \frac{1}{2L} \tag{3.13}$$

$$K_D = 0.5L \tag{3.14}$$

In the other method, the gain of the P controller in a closed loop system test, shown in Figure 3.3, is increased until a stability limit is reached with a test controller gain K_u. The control parameters are then calculated based on this critical value of gain K_u and the resulting period of sustained oscillations P_u using the following relations [1]:

i. For P controller

$$K_P = \frac{0.5}{K_u} \tag{3.15}$$

ii. For PI controller

$$K_P = 0.45K_u \tag{3.16}$$

$$K_I = \frac{1}{0.83P_u} \tag{3.17}$$

iii. For PID controller

$$K_P = 0.6K_u \tag{3.18}$$

$$K_I = 2P_u \tag{3.19}$$

$$K_D = 0.125P_u \tag{3.20}$$

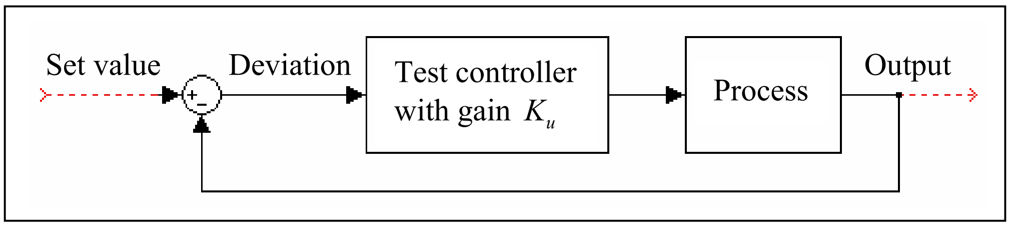


Figure 3.3: Block diagram of process with controller [1]

It should be emphasized that the rules of Ziegler and Nichols were developed empirically, and that the control parameters provided by these rules are not optimal. However, they do give a good starting point from which further tuning can be performed to obtain satisfactory system performance.

The continuous PID control law was given by Equation (3.7). In a discrete time system, the integral and derivative terms are approximated by using discretized models [2, 3, 4]. Various discretization methods can be used to generate different versions of the digital PID algorithm. The simplest method for approximating an integral is a rectangular (staircase) backward difference approximation described as

$$\int_0^t e(t_D)dt_D \approx \sum_{i=0}^{k-1} e(i)T \tag{3.21}$$

where T is the sampling time [2]. The derivative of the control error is approximated by a backward difference quotient,

$$\frac{d\left[e(t)\right]}{dt} \approx \frac{e(k)-e(k-1)}{T} \tag{3.22}$$

Substituting the approximating expressions of Equations (3.12) and (3.22) into Equation (3.7) and replacing $u(t)$ and $e(t)$ with $u(k)$ and $e(k)$, respectively, yields

$$u(k) = K_P e(k) + K_I \sum_{i=0}^{k-1} e(i) + K_D\left[e(k)-e(k-1)\right] \tag{3.23}$$

where K_P, K_I, and K_D are the digital proportional, integral, and derivative gains.

Equation (3.23) is referred to as a position PID algorithm. Another, very widely used form of digital PID is a velocity algorithm. To derive the velocity algorithm, first consider the control signal defined by Equation (3.23) at the $(k-1)T$ instant of time:

$$u(k-1) = K_P e(k-1) + K_I \sum_{i=0}^{k-2} e(i) + K_D\left[e(k-1)-e(k-2)\right] \tag{3.24}$$

Subtracting Equation (3.24) from Equation (3.23) yields

$$\begin{aligned}\Delta u(k) = u(k) - u(k-1) = K_P\left[e(k) - e(k-1)\right] + \\ K_I e(k-1) + K_D\left[e(k) - 2e(k-1) + e(k-2)\right]\end{aligned} \tag{3.25}$$

Now solving for $u(k)$ we get

$$\begin{aligned}u(k) = u(k-1) + \left\{K_P\left[e(k) - e(k-1)\right] + \right. \\ \left. K_I e(k-1) + K_D\left[e(k) - 2e(k-1) + e(k-2)\right]\right\}\end{aligned} \tag{3.26}$$

The velocity algorithm is usually preferred to the to the position algorithm because it is computationally simpler, it is safer because in case of controller failure, the control signal remains unchanged, $\Delta u(k) = 0$, and it better handles "wind-up" [2]. A control error wind-up occurs within the controller after a control actuator such as a DC motor, reaches the maximum output. When this happens, the error signal to the controller persists and the integrator output continues to increase, producing a "wind-up" phenomenon [2].

The controller gains K_P, K_I, and K_D are selected to meet specified process performance requirements and must be adjusted according to the process transient and steady state characteristics. A set of tuning rules, derived from the Ziegler-Nichols rules for analog controllers can be used to adjust the controller gains based on the process step response. The Ziegler-Nichols rules for a digital PID controller are [2]

$$K_P = \frac{1.2}{R(L+T)} \tag{3.27}$$

$$K_I = \frac{0.6T}{R(L+0.5T)^2} \tag{3.28}$$

$$K_D = \frac{0.5}{RT} \tag{3.29}$$

The values of R and L are determined from the step response curve, as shown in Figure 3.1. It should be stressed that, just as in the case of analog controllers, the Ziegler-Nichols rules do not guarantee optimal settings for the digital controller gains. In most cases, however, the values obtained using Equations (3.27) to (3.29) provide a good starting point for further fine tuning of the controller gains based on the system on-line performance.

The values of the PID gains of a digital controller are also dependent on the sampling time T as can be seen by Equations (3.27) to (3.29). The initial magnitude of a control signal produced by a digital PID controller in response to a step change of an input signal is greater if the sampling time is smaller for most control algorithms. Thus, selecting the sampling time for a digital control system is a compromise between performance and cost including the control effort [2, 4].

We chose to use the Ziegler-Nichols method due to the nonlinearities in the Smart Car system model. By calibrating the Smart Car and collecting the required step response data, we were able to successfully apply the Ziegler-Nichols method to calculate the PID gains for control of wheel RPM, which can be seen in Section 4.6.

3.4 PID Controller Implementation

To demonstrate mediated control by assisting an operator with a task involving precision we designed and implemented a PID control system to manage the wheel speed of the Smart Car. The general block diagram for the Smart Car digital PID control system with the continuous drive system process is shown below.

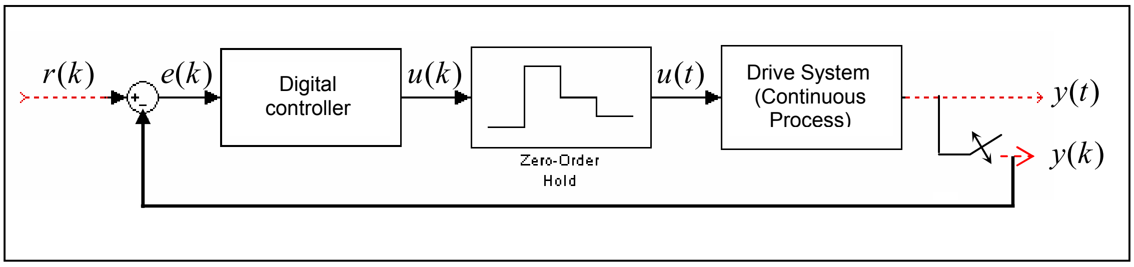


Figure 3.4: Block diagram of the Smart Car digital control system

We used a Basic Stamp 2 as our digital microcontroller, which interfaces with our composite encoders and Novak electronic motor speed controller that doubles as a zero order hold device. The ESC uses PWM to drive the DC drive motors and holds the output constant while the digital controller calculates the updated control pulse width to send to the ESC.

To control the wheel speed of the Smart Car we needed to measure the actual wheel speed and compare it with the desired wheel speed which the operator adjusts using the joystick on the radio control. Because we calibrated the Smart Car wheel speed with the pulse width sent from radio control under a no-load situation we are able to determine the desired wheel speed at any instant. The composite encoder is also calibrated so that by measuring the width of the pulses generated by the photoreflector we can determine the actual wheel speed. The program then calculates the error

$$Err = (Desired\_wheel\_speed - Measured\_wheel\_speed) \tag{3.30}$$

which is needed to determine the portion of proportional, integral, and derivative control that will be added or subtracted from the reference signal. To calculate the proportional control the following equation was programmed as

$$P = (Err * Kp) \tag{3.31}$$

The integral control was calculated using an algorithm which adds the cumulative error for T_i measurements and then finds the average error (E_i) before performing the following calculations

$$\begin{aligned} Ei &= Ei * Ki \\ I &= I + Ei \end{aligned} \tag{3.32}$$

that first multiplies the average error by the integral gain (K_i), which is then added to the portion of integral control that will adjust the reference signal. The derivative calculation is programmed as

$$D = (Err - LastErr) * Kd \tag{3.33}$$

which obtains the difference of the current error and the last error and multiplies by the derivative gain (K_d).

Since we are controlling the speed of the wheel (not the position) the reference signal *Desired_wheel_speed* is adjusted by adding the cumulative sum to the current sum of Equations (3.31), (3.32), and (3.33) which are named *LastDrive* and *Drive* respectively. The "new" desired wheel speed was programmed as follows

$$\begin{aligned} &Desired\_wheel\_speed = Desired\_wheel\_speed + Last\_Drive + Drive \\ &Last\_Drive = Last\_Drive + Drive \end{aligned} \tag{3.34}$$

Now the calibration curves are used again to calculate the width of the pulses in the pulse width coded modulation (PWCM) pulse train that the Basic Stamp 2 microcontroller sends to the ESC to drive the Smart Car. The PID wheel speed controller runs continuously while in closed loop mode tracking the desired wheel speed as the operator drives compensating for changes in load that the vehicle undergoes.

The program that includes the PID algorithm has some additional features, which were required for our Smart Car application. Since pulse width calibration curves are used to determine the signal output to the ESC, we first needed to check whether the Futaba radio control was on or off. If the radio control is off, the pulse width is zero, which corresponds to a wheel speed of 965 RPM using the calibration curves. In the program, we assume that if the pulse width is less than 1ms that the desired wheel speed is zero RPM. Another issue was that too much time was required to measure data and perform the PID calculation for the Smart Car to run smoothly because the update rate was 40ms compared with the standard 18ms. In order to overcome this problem we decided to alternate between measuring the wheel speed and performing the PID calculations. Now the control algorithm first measures the wheel speed, outputs the previous control signal, calculates the new control signal, and outputs the new control signal before repeating the cycle. Alternating cycles also allowed for a lower minimum wheel speed for which the PID controller will properly operate. Another timing issue was encountered because when the Smart Car wheels moves less than approximately 200 RPM, the time required to measure the width of the photoreflector pulse forces the program to output signals at a rate that causes the vehicle to run roughly. Below approximately 200 RPM, the program defaults to open loop mode.

Another trick that we use to ensure that the ESC is updated approximately every 20ms is by reading the pulse sent from the Futaba radio control each cycle, which forces the microcontroller to wait until a pulse arrives. This means that if we are able to perform all measurements and calculations in less than the time span (18ms) between pulses in the pulse train we are guaranteed that the pulses will be output at the same rate as the radio control sends the reference signal. We ran into trouble when we started alternating the wheel speed measurement and the PID calculations because the time to measure the wheel speed and output the previous control pulse was very little time compared the time required to complete the PID loop. Using alternating algorithm the time span on the output pulse train was greater than 30ms; again causing the Smart Car not to run smoothly. To fix the pulse update rate, we inserted a variable time delay in the in the wheel speed measurement loop. The time delay is required to be variable because the pulse widths generated by the composite encoders vary with the speed of the wheels.

The PID program also reads the state of a pin set by the microcontroller running the obstacle detection program to determine whether to drive the vehicle forward or when an obstacle is detected in front of the vehicle, override the PID program and set the Smart Car in neutral.

As with the obstacle detection program, when open loop is selected using the discrete toggle switch the operator is given full control and no mediation will occur to the wheel speed. Shifting between open and closed loop mode can be done at any time, even while the Smart Car is in motion, by moving a discrete toggle switch on the Futaba radio control. Every 54ms the microcontroller determines the mode that the operator has selected by reading the width of the pulses sent from the transmitter to the receiver, which depends on the position of the discrete switch.

The PID controller for the Smart Car wheel RPM allows the operator to precisely vary the speed of the vehicle while having the assurance that the controller will track the desired speed even when subjected to varying load conditions allowing the driver to concentrate on other things. Using the PID algorithm, the operator is no longer required to manually adjust the joystick position to compensate for the vehicle change in speed due to an external disturbance.

Chapter 4. Validation:

As shown in Chapter 2, we used commercial off the shelf (COTS) hardware and integrated the whole system. Validation and calibration is a very critical step in the whole process. While an ad-hoc solution approach would also suffice for general demonstration purposes, we are interested in using the Smart Car as a scaled testbed. In particular, we are also interested in obtaining quantitative data and hence we spent considerable time validating our system. We begin with calibrating the independent subsystems, which include the transmitter/receiver, sensors, and then consider the calibration of the controllers interaction with the entire system. A test setup with an external reference tachometer was created to validate open and closed loop response of the various controllers.

4.1 PWCM Transmitter/Receiver Signals

To mediate the Smart Car commands we used the BASIC Stamp 2 microcontroller inserted between the receiver and actuators, which required us to analyze the signals used to control the original RC truck. The analysis shows that the transmitter and receiver signals are pulse width coded modulation (PWCM) signals that we now read into the BS2, execute a control algorithm, and generate the proper control signal to send to the actuators. Figure 4.1 below shows the transmitter signal from the six channel Futaba radio control, which merges the command signals into a pulse train of the appropriate pulse widths that the receiver separates into six individual pulse width coded modulation (PWCM) signals for the corresponding channels.

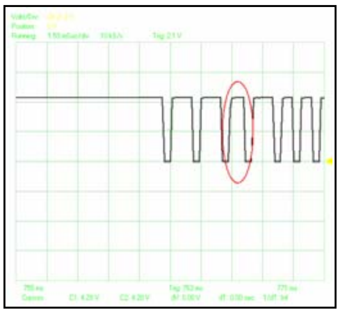


Figure 4.1: Six-channel transmitter signal with channel three circled

Each PWCM signal sent from the receiver contains only data for a single channel, which can be seen below.

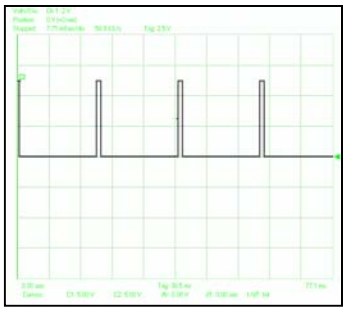


Figure 4.2: PWCM signal for a single channel sent from the receiver

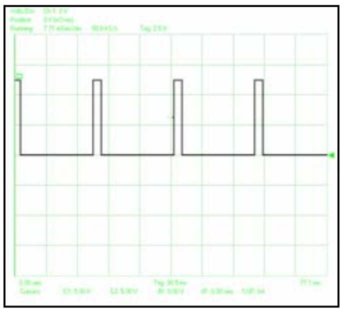


Figure 4.3: PWCM signal for a single channel sent from the receiver

The standard time-span between pulses for pulse width coded modulation (PWCM) to drive servomotors and electronic speed controls used for RC cars is approximately 18ms. The time-span output by the Futaba receiver is approximately 18ms, the pulse width is the standard range from 1ms to 2ms, and the magnitude of each pulse is approximately 5V.

The BS2 has a built in command to simplify the measurement and creation of pulses, which is described on pages 241 and 243 in the Basic Stamp Programming Manual Version 2.0c. Using the built in commands, the pulse train can be generalized to output variable pulse widths.

To compare the transmitter/receiver signals with the signals generated using the BS2 PULSOUT command we recorded the pulses width for channels 1 – 4 at three stick positions and compared them with the BS2 output. Figure 4.4 shows two things: (i) the pulse width varies linearly with the Futaba radio control stick position and (ii) the BASIC stamp 2 can duplicate the PWCM signal. The stick position on the Futaba radio control was calibrated to determine the corresponding pulse widths read by the BS2. The ideal relationship should be

$$Pulsewidth = \left(2\times10^{-6}\right) JoystickPosition + \left(1\times10^{-5}\right)$$

however, the actual pulse widths corresponding to joystick positions are shown below.

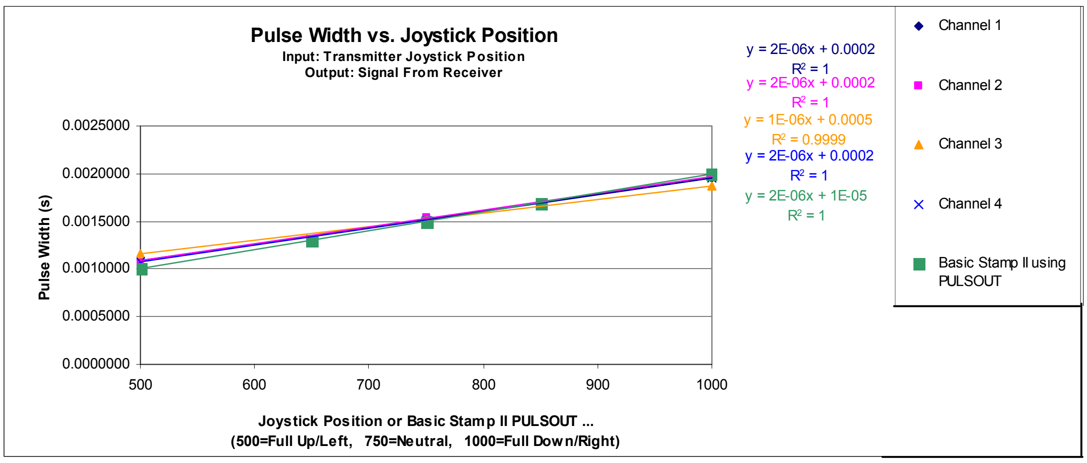


Figure 4.4: Pulse width comparison between transmitter/receiver and BS2

4.2 Obstacle Detection Calibration

The calibration curve for the center infrared ranger of the obstacle detection array is shown below in Figure 4.5. This calibration curve is used to determine the upper limit on the distance when microcontroller running the obstacle detection program signals the second microcontroller to override the PID algorithm and set the Smart Car in neutral.

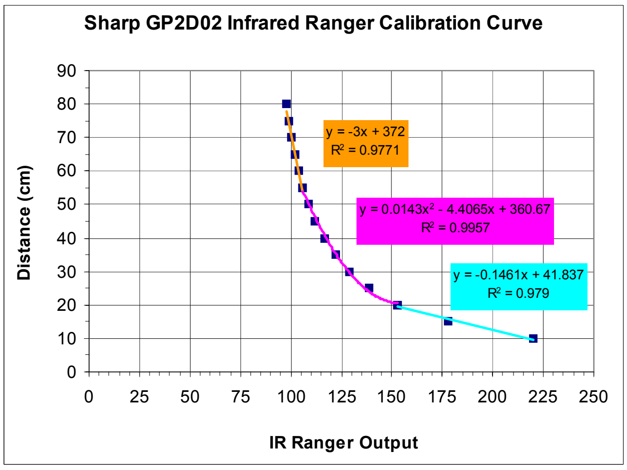


Figure 4.5: Sharp GP2D02 infrared ranger calibration curve

4.3 Calibration Necessary to Implement PID Control of Wheel RPM

The calibration curves used in the PID algorithm described in Section 3.4 are shown below in Figure 4.6 and Figure 4.7. To obtain these calibration curves, we ran the smart car with no load and pulse trains with pulse widths ranging from 1.5ms to 1ms. We fit the curve with linear calibration curves because the maximum value that the Basic Stamp 2 can handle is 65535 and signed variables from -32767 to 32767. To prevent scaling the data and since a second order equation would produce numbers greater than the BS2 can handle, we fit the nonlinear curve with first order equations.

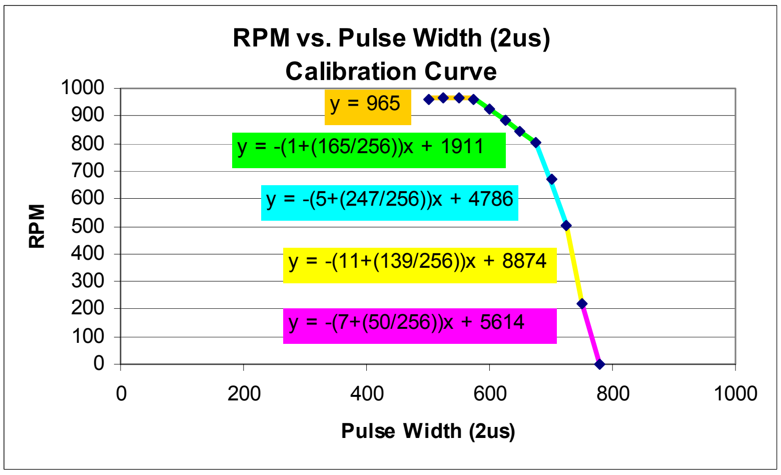


Figure 4.6: Calibration curve to calculate desired RPM from the pulse width sent from the Futaba radio control

The curve in Figure 4.6 is used to determine the desired wheel speed by measuring the pulse widths of the reference signal pulse train sent from the Futaba radio control. The curve shown in Figure 4.7 is used in the PID algorithm to determine the actual wheel speed by measuring the width of the pulse generated by the composite encoder.

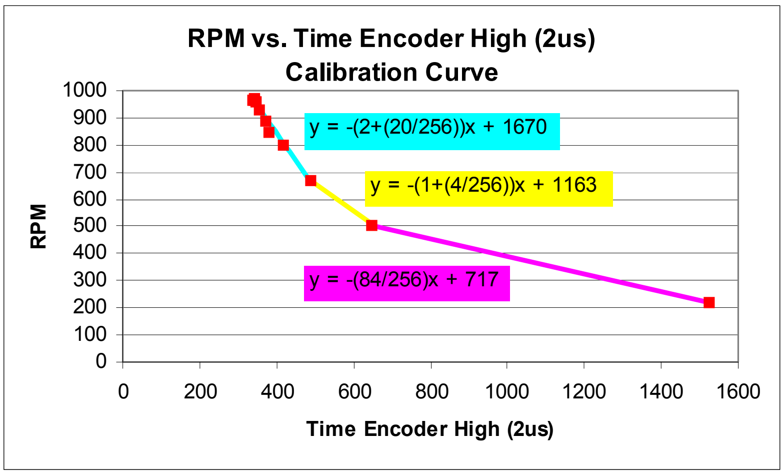


Figure 4.7: Calibration curve to calculate measured RPM from the encoder signal

4.4 Validation of Obstacle Detection

The Smart Car stops when the obstacle detection array senses an obstruction within 2.5 ft. in front the vehicle and restricts the operator from steering into an obstacle sensed on the sides of the Smart Car. When an obstacle is sensed in the path of the vehicle, the Smart Car mediates the command signals by stopping the vehicle and only allows the operator to reverse. If obstacles are sensed on the side of the vehicle the Smart Car mediates the command signals and allows the smart car to move forward or reverse but restricts the driver from steering into the obstacle. The obstacle detection array will perform with multiple obstacles in the path of the vehicle. For example, with an obstacle in front of the vehicle and to the side of the Smart Car, the operator will only be able to reverse and steer in the opposite direction of the side obstacle.

4.5 Controller Test Setup

The test setup to collect data for calibration, Ziegler-Nichols calculations, and validation of the digital PID controller is shown in Figure 4.8. The tachometer is connected to one front wheel and a torsional damper is connected to one rear wheel.

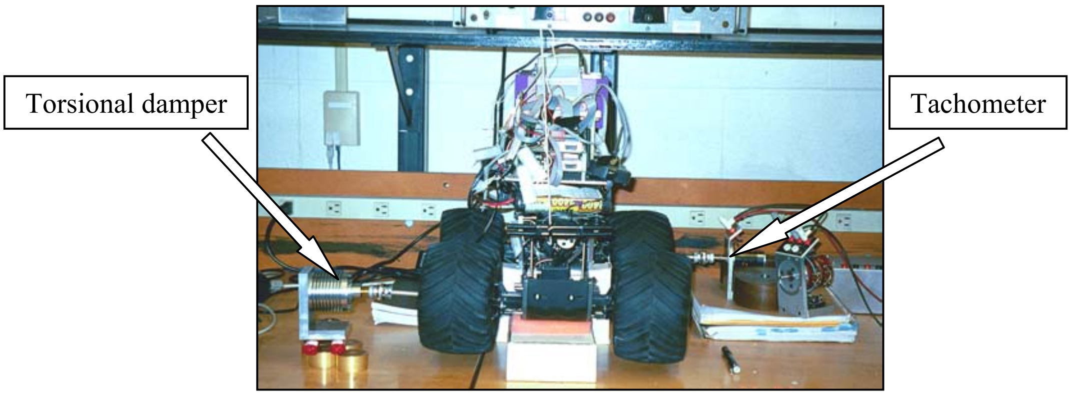


Figure 4.8: Setup for PID testing

The tachometer is used as an absolute reference while the torsional damper can produce an external disturbance to validate the digital wheel speed PID control system. We used LabVIEW VirtualBench Scope data acquisition to record the tachometer data while simultaneously recording the RPM from the encoder on the Smart Car using StampDAQ by Parallax, Inc. The use of StampDAQ was convenient because it allowed us to output variables rather than signals that would require post processing. For example, to calibrate the Smart Car wheel RPM we wanted to record only the width of the encoder pulses and not the entire signal. Using StampDAQ, we just used serial communication to record only the program variable containing the width of the pulses. In addition, to validate the wheel RPM calibration we can output the wheel RPM calculated using the BS2 microcontroller. This test setup allowed us to calibrate the Smart Car encoder and verify the results during step responses and responses to the external disturbance when in open loop and closed loop.

The diagram shown in Figure 4.9 describes how we used the Smart Car's open differentials to induce an external step increase in load using the torsional damper allowing us to test the performance of the PID controller.

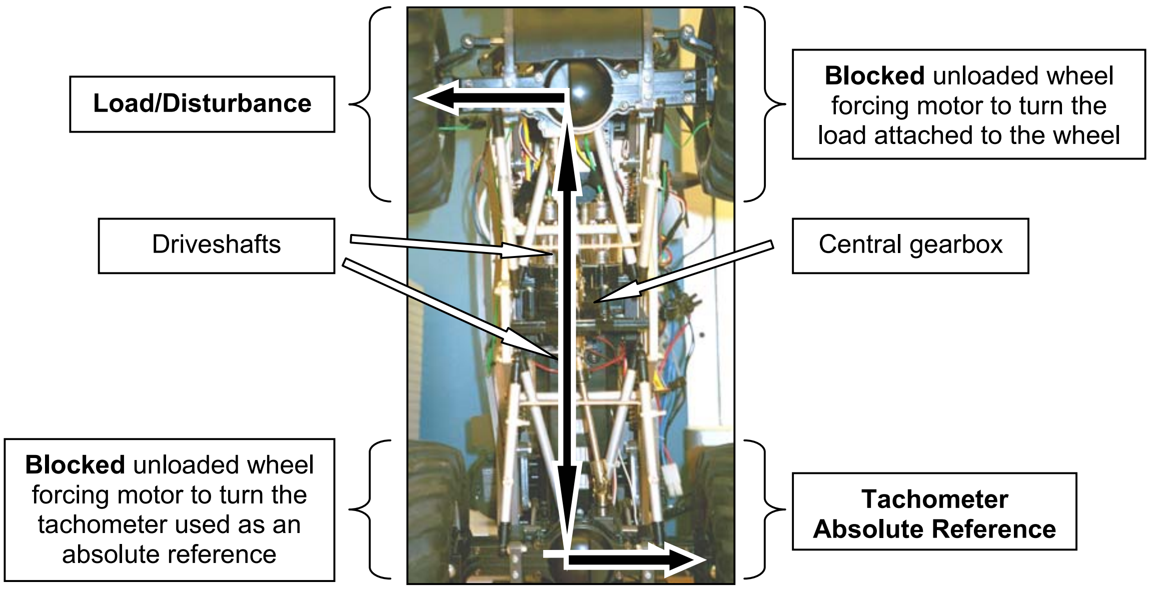

Load/Disturbance
Blocked unloaded wheel forcing motor to turn the load attached to the wheel
Driveshafts
Central gearbox
Blocked unloaded wheel forcing motor to turn the tachometer used as an absolute reference
Tachometer Absolute Reference

Figure 4.9: Power/torque flow diagram

The open differentials are design so that while driving Smart Car with the wheels lifted off the ground and with one wheel is held stationary, the opposite wheel on the same axle spins freely and the Smart Car does not "feel" the load. Therefore, to ensure that during testing the full power/torque is directed to the wheel with the tachometer, we blocked the wheel on the opposite side of the tachometer on the same axle. Since the central gearbox transmits power to the front and rear differential through the same shaft, when a load is imposed on the rear wheels of the Smart Car, the corresponding reduction in RPM can be measured with the front wheels. This flow of torque gives us an easy way to measure wheel RPM on the front wheels using the tachometer as an absolute reference while switching between no-load and loaded conditions on the rear wheels. To measure the wheel speed during a no-load situation, block the rear wheel with the load/disturbance attached to the axle and to induce a step increase in load, quickly stop the rear wheel with no-load on the opposite side, which transfers the power to the wheel with the external disturbance.

4.6 Control of wheel RPM using a digital PID controller

Before we could implement a PID controller for the wheel speed of the Smart Car, we needed to verify that the Smart Car was calibrated properly and that the encoder would sufficiently follow the absolute tachometer reference. Figure 4.10 shows that the encoder tracks the absolute reference through a full cycle from start to stop including two step inputs. In addition, the desired RPM is plotted to show where the step inputs occur and that while still in open loop under no-load the actual RPM measured using our composite encoder matches the desired RPM at steady state.

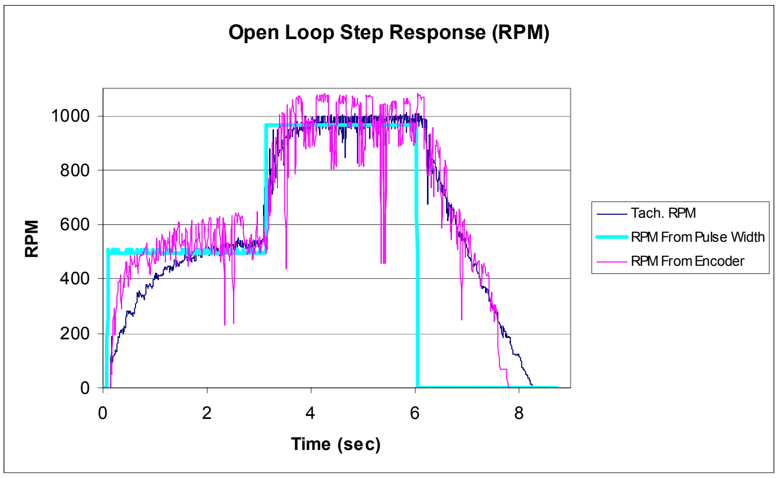


Figure 4.10: Full start/stop cycle with actual tachometer RPM, calculated RPM and the desired RPM

To select PID gains using one of the Ziegler and Nichols methods [1, 2], we recorded data while the vehicle ran in steady state before applying a step input as in the second step response of Figure 4.10. Figure 4.11 is a close up of the second step response and shows one of several step responses that we analyzed using the Ziegler-Nichols method to find a good starting point for our PID gains.

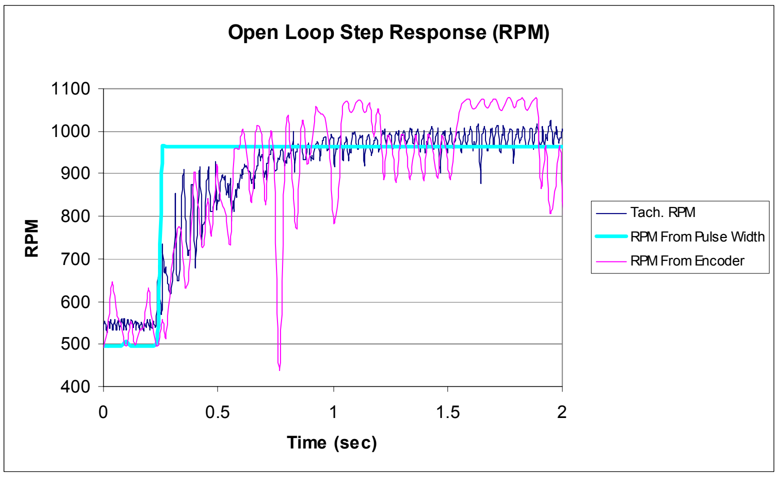


Figure 4.11: Step response with actual tachometer RPM, calculated RPM using the encoder, and the desired RPM

The actual RPM data with the Ziegler-Nichols guidelines is shown in Figure 4.12.

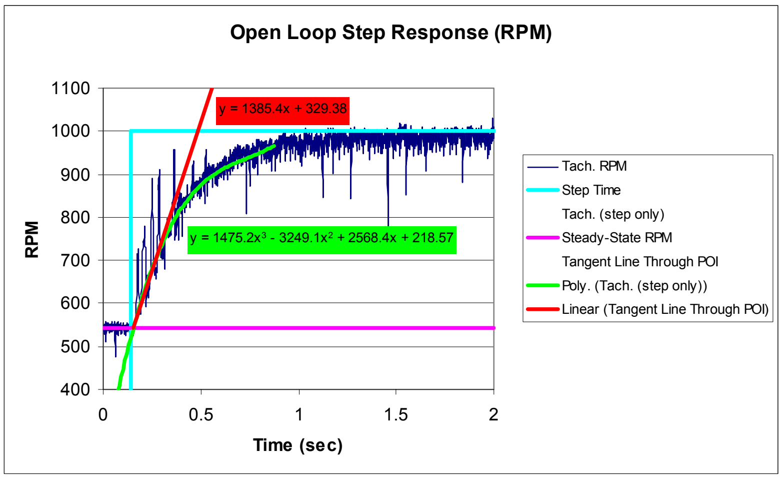


Figure 4.12: Ziegler-Nichols method needed to determine digital PID gains

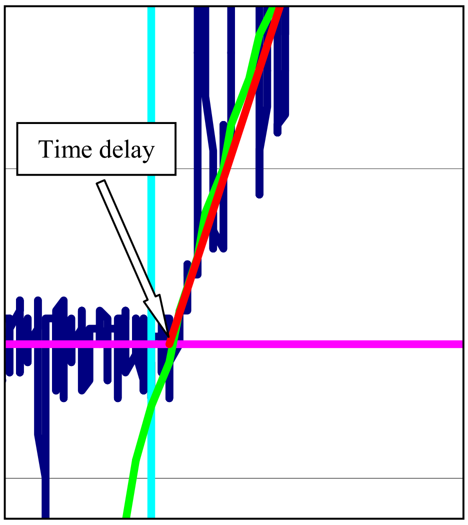


Figure 4.13: Close up of Ziegler-Nichols guidelines

Referring to Section 3.3 the slope (R) is determined from the tangent line that passes through the point of inflection of the RPM step response data. We determined the time delay (L) shown in Figure 4.13 as the difference from the time of the step input and the point where the moving average of the data intersects the initial steady state RPM. Our sampling time (T) is 40ms even though we output a control signal every 20ms because we only read the encoder every-other-time. Now we are able to plug R, L, and T into the empirical Ziegler-Nichols equations for a digital PID controller Equations (3.27) through (3.29) to estimate the proportional, integral, and derivative gains. We obtained our initial estimated values of the PID gains by averaging the calculations of six different step responses which were, $K_P = 0.035$, $K_I = 0.022$, and $K_D = 0.027$.

The estimated PID gains were very close to the final values that we chose by manually adjusting to optimize the response the external load caused by the torsional damper. The integral gain was a little high causing the Smart Car to oscillate during steady state. We increased the proportional gain to decrease the rise time, however values over approximately 0.042 with PID controller and 0.100 with a P controller cause the RPM to slightly overshoot. We found that the following gains work well with the corresponding controllers:

i. For P controller: $K_P = 0.1$
ii. For PI controller: $K_P = 0.042$, $K_D = 0.027$
iii. For PID controller: $K_P = 0.042$, $K_I = 0.006$, $K_D = 0.027$

The Smart Car open loop response to an external disturbance as described in Section 4.5 is shown below in Figure 4.14 [1 – 4]. The RPM from the encoder decreases with the tachometer absolute reference while the desired RPM stay constant at approximately 700 RPM.

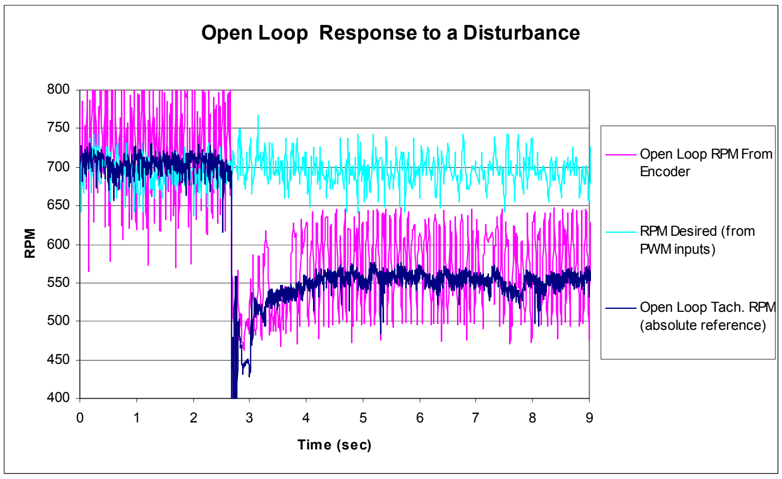


Figure 4.14: Open loop system response to a disturbance (step increase in load)

Using closed loop P controller with a proportional gain of 0.1 gives a system response as shown in Figure 4.15, which shows that the system does not overshoot in response to the disturbance however there is some steady-state error.

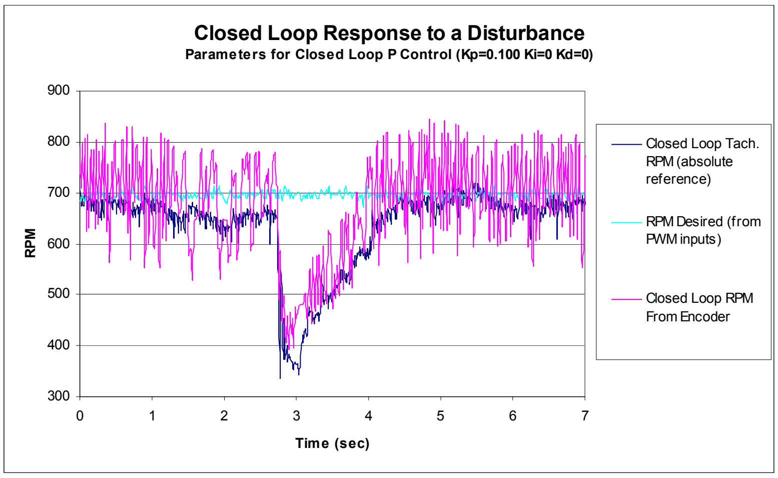


Figure 4.15: closed loop system response using proportional control

The PD controller did not produce as smooth a response as the P controller. Control theory states that derivative control amplifies noise and due to our inexpensive composite encoder, we are suggesting that the response is actually more sluggish and less desirable in this particular case [3, 4].

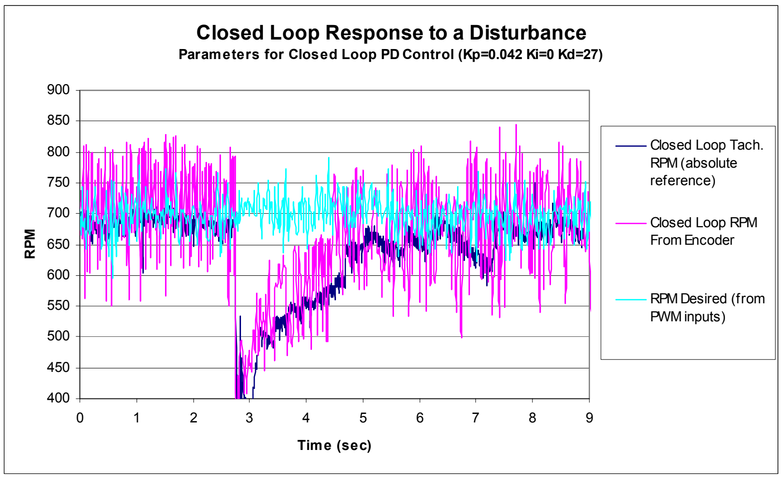


Figure 4.16: Closed loop system response using proportional-derivative control

The PID control of the Smart Car wheel RPM is most desirable because the rise time is close to that of the system with only P control and has approximately zero steady-state error [3, 4]. Figure 4.16 shows the Smart Car wheel RPM response to a step increase in load using a digital PID controller.

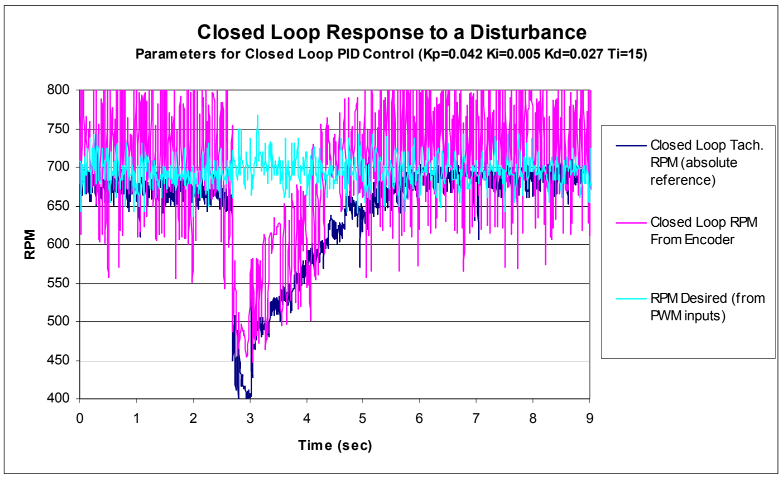


Figure 4.17: Closed loop system response using proportional-integral-derivative control

The advantage of having a feedback control system is evident in Figures 4.15 to 4.17. The control system responds to external disturbances by sensing a change in RPM and adjusts the output to bring the actual RPM back to the desired RPM without requiring any operator input. The operator can set the speed and let the controller precisely track the RPM while the operator is free to drive other actuators. The controller is constantly running while in closed loop mode which means the operator does not have to tell the Smart Car what the setpoint is because the controller tracks the command signal set with the joystick position that can be varied as desired.

Chapter 5. Discussion/Future Work:

We have successfully demonstrated the idea of mediate control, which takes burden off the operator and assists the operator in executing tasks. The success of a task carried out using the Smart Car is no longer fully dependent on the operator's skill and awareness. For certain tasks, the operator has the option to drive the vehicle in open loop mode by switching the discrete switch on the Futaba radio control. The Smart Car detects obstacles and prevents collisions without totally removing control from the operator. The operator can still drive the vehicle away from the obstruction and full control will resume when the operator successfully moves beyond the obstacle. The digital PID controller precisely tracks the desired wheel speed set by the operator allowing the operator to focus attention on more complicated tasks such as possibly controlling robotic arms.

In addition, we have demonstrated that relatively high-level control systems can be implemented using inexpensive commercial off the shelf (COTS) components. Future work on the Smart Car is now open to improving the PID control performance using microcontrollers with faster processors and encoders that are designed for precise wheel speed control. We have already determined that we can easily mount an encoder wheel inside the axle housing and mount the encoder sensor to the axle housing with the only intrusion being a hole just large enough for the encoder sensor, which can be sealed with silicone caulk or hot glue. This encoder design should be much more robust to dirt, water, vibration, and shock. The performance of the obstacle detection array can be greatly improved by simply replacing the infrared rangers with ultrasonic rangers that have a greater distance sensing range. Care should be taken when selecting sensors to ensure that the required time to retrieve data is small to keep the processing time low and sampling rate high especially when working with actuators that use PWCM because the required update rates are usually less than 20ms.

The performance of the PID control algorithm can be improved using filtering techniques such as a weighted moving average to filter the encoder signals. In addition, the framework for feedback control has been validated, which will allow more sophisticated controllers to be implemented such as nonlinear sliding mode control, nonlinear adaptive control, or adaptive cruise control to be used in conjunction with the obstacle detection algorithm. The scaled testbed that we designed will facilitate future development of mechatronics systems and control techniques to be implemented at a cost well below that of a full-scale prototype [8]. Nonlinear system modeling and system identification techniques used to create a virtual prototype system model of the Smart Car can be verified by comparing the actual system response with the virtual prototype models. Development time can be greatly reduce with the addition of a PC/104, xPC Target software and Real-Time Workshop because C-code can be generated from the block diagram virtual prototype models and sent via wireless Ethernet to the target PC on the Smart Car to allow immediate testing of the control system.

Currently, research is being done using a PC/104 as the main processor and the allowing the Basic Stamp 2 microcontrollers to operate as subsystems. The subsystem design will allow the subsystems to control repetitive tasks such as the PID control of wheel speed and constantly checking for obstacles. The PC/104 will run Mathworks xPC Target and have the ability to communicate via wireless Ethernet with a host computer. This architecture has several advantages such as rapid controller prototyping using Mathworks Real-Time Workshop to generate C-code that is generated control systems using the convenient Simulink block diagramming. The target computer on the Smart Car can run independently or communicate with the host computer to take burden off the PC/104, which is advantageous when designing systems that do not require large amounts of power. Robot rovers for planetary research on Mars are one example of a vehicle that could take advantage of low power consumption and the ability to communicate with a host computer.

The automotive market is one of the largest industries that are researching mediated control due to the many peripheral devices to distract the average driver. Companies such as Motorola, Delphi Automotive Systems, and DaimlerChrysler are researching mediated control systems that can enhance overall performance/safety of driving [6]. The proposed mediated control systems would use a variety of sensors to monitor both the driver's actions such as steering and braking patterns as well as the environment such as road and traffic conditions [6 – 8]. This information would then permit them to respond to adverse driving conditions far more rapidly than would be possible by the driver alone.

Appendices:

Appendix A. In-Depth Description of the Mobile Platform:

To begin the project we needed to select a mobile platform to mount our microprocessor, sensors, and in the future, we would like to add actuators. We chose a 1/10th scale remote control monster truck, the Tamiya TXT-1, as our platform. One of the reasons we chose the TXT-1 over the Tamiya Clodbuster was because we found through online research that the Clodbuster was phasing out. Once companies sell the Clodbusters in inventory, there will be no option to buy new Clodbusters.

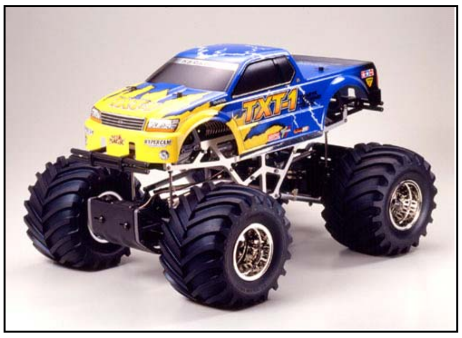


Figure A.1: Tamiya TXT-1 used as the mobile platform for the Smart Car

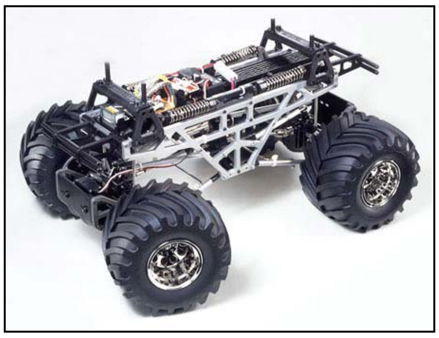


Figure A.2: TXT-1 with the cover off

This particular vehicle is extremely durable with several metal parts such as the aluminum ladder style frame, aluminum suspension and steering components, and metal gears in the differentials and gearbox. Four-wheel-steering (4WS) allows relatively tight turning radius increasing the vehicles maneuverability, which is needed when completing complicated tasks. There is one steering servomotor located directly between the front wheels and one between the rear wheels, which is a much stronger design compared with 4WS vehicles designed with one central steering servo. By using two separate servomotors to steer the TXT-1, each servo can either be controlled together or controlled separately allowing a diagonal path of travel when both the front and rear wheels are turned in the same direction.

As you can see from Figure A.2, this particular remote control truck has the potential to easily mount a base plate for the control electronics. The large 4.5-inch wide and 6.5-inch tall tires can have foam inserts installed to compensate for the added weight of the microcontrollers and/or PC/104. The aluminum frame lends itself to be easily drilled and tapped to mount additional brackets and actuators and the frame is perpendicular to the ground, which also makes adding brackets easier by reducing the need for complicated angled brackets. In addition to a metal frame, Figure A.3 shows the interior of the front and rear open differentials that have metal gears to withstand the high torque applications for several years.

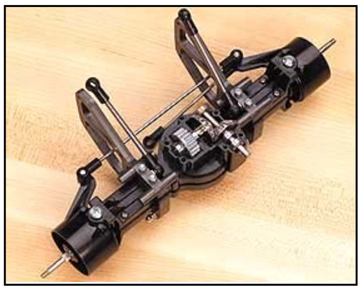


Figure A.3: Differential with top cover removed

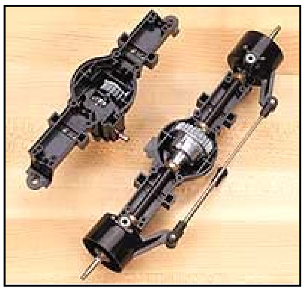


Figure A.4: Open differential split open displaying metal planetary gears

Most of the parts on the TXT-1 including the metal differential gears are replaceable which is necessary when selecting platform, which will be used for years to come. A second photo shows the open differential split in half displaying the metal planetary gears.

Figure A.5 below shows many more aluminum parts including the stabilizer bars and suspension components. The steering has dual adjustable tie rods, which allows for camber and caster/toe-in adjustments, which is sometime useful for optimum handling characteristics. Camber is how the tires sit vertically and caster is how the tires point inward as viewed from above. The four link solid axle is adjustable and the front and rear stabilizer bars reduce chassis roll. The suspension links are made of high quality aluminum tubing with plastic ball cup ends for fully articulated movement.

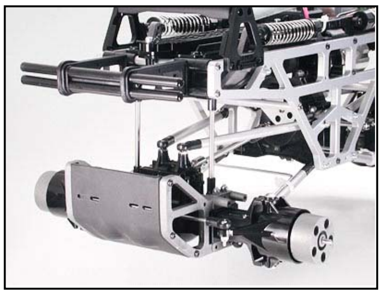


Figure A.5: TXT-1 durable metal components

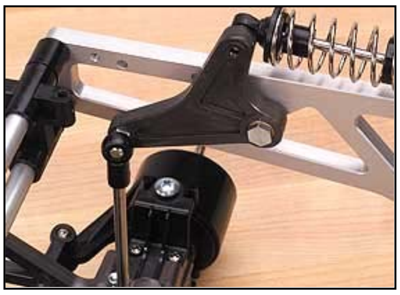


Figure A.6: TXT-1 cantilever suspension

The suspension consists of a cantilever push rod design with high volume oil damped coil over shocks. The cantilever design allows the shocks to provide long travel to enable the axles to clear the most extreme angles. The suspension setup lends itself to be easily actuated for projects with active suspension compared to the limited space of vertical shock designs where the suspension components are not so accessible. Figure A.6 shows a close-up.

The oil-filled shocks allow for damping changes by increasing or decreasing the oil viscosity to tune the suspension response and can be easily modeled compared with friction-damped shocks, which are much less reliable. With these coil over shocks we have the option to change suspension spring rate by being able to replace the stock springs with springs of various stiffness to optimize the vehicles response to rough terrain and handling at high speeds. Being able to tune the suspension damping and spring rate was an important feature for robotics application because we needed to be able to compensate for the additional weight of the microcontrollers, PC/104, sensors, and actuators.

Although the TXT-1 weighs 11 lbs. stock plus the additional components, the truck has a lot of torque, but does not go very fast (less 10mph stock). The picture in Figure A.7 shows the dual high-torque Mabuchi RS- 540SH's DC drive motors mounted on the center gearbox with the metal front and rear drive shafts attached. The center gearbox has additional bracing that acts a heat sink and an integral stress member for the chassis.

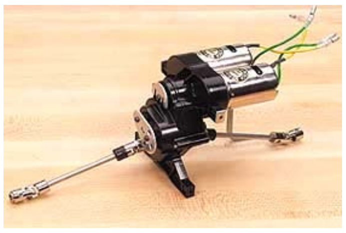


Figure A.7: Dual motors mounted on the center gearbox

For our robotics application, a remote control vehicle with high torque output is important to compensate for the weight of the extra components added to the vehicle where as high speed is not critical.

The transmitter shown below in Figure A.8 is a six channel Futaba Skyport 6 with two joysticks that move left, right, forward, and backward thus four control signals can be sent from these two joysticks. In addition, the transmitter has a dial that rotates clockwise and counterclockwise to send commands and the last option for control is through a toggle switch, which can only send on/off signals.

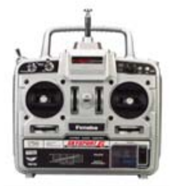


Figure A.8: Six channel Futaba radio control

Figure A.9: Trainer cord for the Futaba Skyport 6 radio control

The transmitter uses a 75 MHz crystal to meet the United States Federal Communications Commission (FCC) standard for ground vehicles. For a list of frequencies that are legal in the U.S. see http://rcvehicles.about.com/library/eih/bleih_freq.htm. The Futaba transmitter has many additional features such as an input/output port on the rear of the transmitter. The port is called a *trainer port* because it was designed to be used for training a novice remote control airplane operator. If two transmitters are connected using the trainer cord shown in Figure A.9 and the trainee makes an error while flying an airplane the trainer using a separate transmitter can take over to recover the expensive airplane. The trainer port gives the option to interface the transmitter with a computer allowing computer control while using the transmitter to send signals to the smart vehicle.

The Futaba transmitter also has a servo reversing capability, which allows the user to reverse the signal sent from the transmitter. For example, if it is convenient to mount a servomotor in a position such that the servomotor moves left when a joystick moves to the right, you can use the servo-reversing switch to correct the problem. The transmitter also has mechanical trims for channels 1 – 4 which is an easy way to fine tune the center position of a servomotor when the control stick is at neutral. The transmitter has dual rates for two channels, which allow you to control the sensitivity on the channel. Along with the above features, the transmitter has adjustable travel volume (ATV) on channel 3 that allows you to set endpoints for servo travel so that you can limit how far an actuator moves in either direction.

We decided to select an electronic speed control (ESC) to replace the mechanical speed control (MSC) because the ESC has 32 discrete steps in speed resolution compared to the 3 steps used in the MSC. In addition, the ESC has reverse with 32 discrete steps whereas the MSC only has one step in reverse. *Note: Not all electronic speed controls have reverse.* (We chose the popular Novak Super Rooster because many hobbyists have successfully used the Super Rooster with the TXT-1. The Super Rooster ESC can be used with dual motors and allows for high performance motor upgrades in the future.

Figure A.10: Novak ESC

Figure A.11: DuraTrax charger

Figure A.12: Hitec servo

We chose a DuraTrax IntelliPeak AC/DC Deluxe Pulse Charger to charge and discharge the nickel-metal-hydride and nickel-cadmium batteries.

This IntelliPeak charger has peak charging technology for nickel-metal hydride batteries, and discharging and cycling features not commonly found in other chargers. "Negative Delta V" peak detection for NiCd's, plus advanced "Zero Delta V" peak detection technology specifically for NiMH's, provides full charges without overcharging. The DuraTrax charger auto-selects the peak detection method by identifying cell chemistry and auto-identifies the quantity of cells being charged. In addition, the charger comes with a detachable, 12 volt 7 amp AC power supply with built-in cooling fan. The charger also has the option to use an 11-15V DC input power supply such as a standard car battery. This feature may be nice for charging batteries in the field where an AC power outlet may not be available. Finally, twin built-in miniature fans keep the charger cool, increasing efficiency and lifespan.

The TXT-1 did not come with steering servomotors so after reading reviews of the truck in many hobby magazines and online websites we decided to buy two metal-geared, coreless, high-torque servos from Hitec. In a conventional servo, the motor has a steel core armature wrapped in wire that spins inside the magnets. In a coreless design, the armature uses a thin wire mesh that forms a cup that spins around the outside of the magnet eliminating the heavy steel core. A coreless motor does not have magnets as standard servo motors do, so they have a smoother, more constant, and stronger action. Regular servomotors have either three or five magnets (poles) which when the armature is between these, the servomotor is at its weakest. The HS-945MG Hitec servomotors we chose are not only high torque but also high speed that is useful when steering while driving fast. The specifications for the servomotors are:

- Torque: 122 oz-in at 4.8V (153 oz-in at 6V)
- Transit time: 0.16 sec/60 deg. at 4.8V (0.12 sec/60 deg. at 6V)

Appendix B. Real-Time Remote Operation:

The Smart Car operates in real-time, which requires processing data sensed from the environment in a short time allowing the microcontroller to determine the proper signal to output. In addition, the Smart Car operates remotely where the design requirements are lightweight and low power consumption, which contradicts the real-time control requirements. Microcontrollers and PC's with fast processors, high capacity hard drives or solid state memory modules, and RAM with large amounts of space are desired for real-time control, however they usually weigh more and require more power than lower performance microcontrollers. Sections B.1 and B.2 describe the microcontroller that we are using including wireless data transfer accessories and data acquisition software that can be used to record and plot data in real-time.

B.1 BASIC Stamp 2

We chose to use the BASIC Stamp 2 microcontroller by Parallax, Inc., which is very light weight, only requires a 6V source, is inexpensive, and has sufficient performance for our application. The BS2 has a 20 MHz processor, 2KB EEPROM, and 32 bytes of RAM. To program the BS2 we use a modified version of BASIC programming language developed by Parallax called PBASIC. Shown below is the Board of Education from Parallax, which is our platform for prototyping the Smart Car embedded controller.

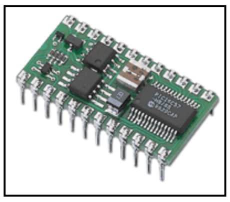


Figure B.1: BASIC Stamp 2 microcontroller

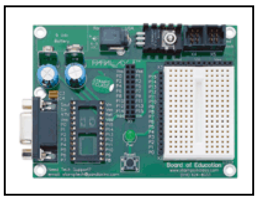
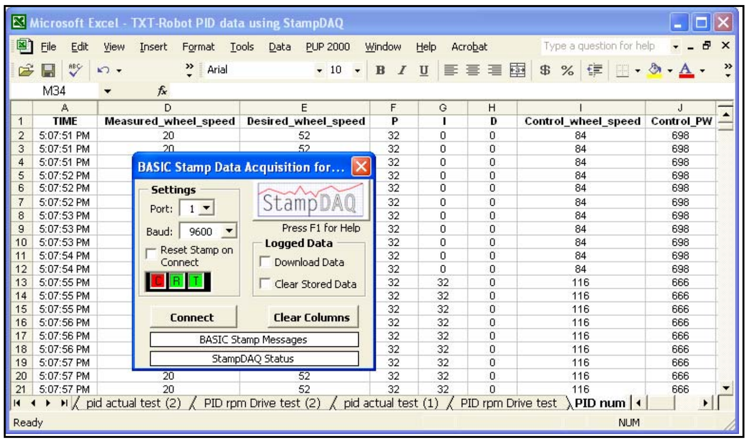


Figure B.2: Board of Education microcontroller platform

The PBASIC code is transferred to the EEPROM via RS-232 serial communication. The serial port is on the lower left side of the board in Figure B.2, which connects to a PC for programming and debugging the BASIC Stamp 2. Parallax has developed software called BASIC Stamp Windows Editor where the code is written, edited, and debugged. The BASIC Stamp Editor combined with the PBASIC code allows programmers to use a debugging window to output variables, text, or almost anything within the code, which greatly helps beginner and intermediate programmers debug and optimize their code.

B.2 Data Acquisition Software

The data acquisition software called StampDAQ can be downloaded free at Paralax.com and allows the BASIC Stamp to be interfaced with a desktop using serial communication. StampDAQ is an add-in for Microsoft Excel that acquires up to 10 channels of data from the BASIC Stamp and records the data in real-time through a serial port at the selected baud rate from 300 to 56000. StampDAQ provides easy spreadsheet analysis of data collected in the field, laboratory analysis of sensors and real-time equipment monitoring. A BASIC Stamp connected to any sensor and the serial port of a PC can now send data directly into Excel in real-time or controlled data dumps.


Microsoft Excel - TXT-Robot PID data using StampDAQ

M34

	A	D	E	F	G	H	I	J
1	TIME	Measured_wheel_speed	Desired_wheel_speed	P	I	D	Control_wheel_speed	Control_PW
2	5:07:51 PM	20	52	32	0	0	84	698
3	5:07:51 PM	20	52	32	0	0	84	698
4	5:07:51 PM			32	0	0	84	698
5	5:07:52 PM			32	0	0	84	698
6	5:07:52 PM			32	0	0	84	698
7	5:07:52 PM			32	0	0	84	698
8	5:07:53 PM			32	0	0	84	698
9	5:07:53 PM			32	0	0	84	698
10	5:07:53 PM			32	0	0	84	698
11	5:07:54 PM			32	0	0	84	698
12	5:07:54 PM			32	0	0	84	698
13	5:07:55 PM			32	32	0	116	666
14	5:07:55 PM			32	32	0	116	666
15	5:07:55 PM			32	32	0	116	666
16	5:07:56 PM			32	32	0	116	666
17	5:07:56 PM			32	32	0	116	666
18	5:07:56 PM			32	32	0	116	666
19	5:07:57 PM			32	32	0	116	666
20	5:07:57 PM	20	52	32	32	0	116	666
21	5:07:57 PM	20	52	32	32	0	116	666

BASIC Stamp Data Acquisition for...

Settings
Port: 1
Baud: 9600
Reset Stamp on Connect

Press F1 for Help

Logged Data
Download Data
Clear Stored Data

Figure B.3: Screen capture of the StampDAQ software add-in for Excel

Appendix C. Basic Stamp 2 Source Code:

C.2 Obstacle Detection Program

```
'{$STAMP BS2}
'=====================================================================================
'File: PID rpm control.BS2
'
'This program implements PID control on the Smart Car to control wheel rpm.  This program
'was written following the basic format of program 6.1 in the text called Industrial
'Control Version 1.1 written by Parallax Inc.
'
'Written by: Dan Gott
'Date: 3/8/03
'=====================================================================================
'
'--[Variables]------------------------------------------------------------------------
'
'** PID CONTROL SETTING ******
Kp              CON 42        'Proportional Gain Setting    (1000=Gain of 1)
Ki              CON 6         'Integral gain constant       (1000=Gain of 1)
Kd              CON 27        'Derivative gain constant     (1000=Gain of 1)
                              'See the equation where "Drive" is calculated for an
                              'explanation.
Ti              CON 10        'CANNOT be > 15 because IntCount is only size "Nib"
                              'Interal Reset time
                              'The average integral error is calculated every Ti error
                              'readings and a new
                              'integral gain (I) is calculated and will be used for the
                              'next Ti times
'
'
'*****************************
NeutralPulseWidth      CON 780        'This is the center pulse width for neutral (units
of 2us)
NeutralLowerLimit      CON 770        'This is the lower limit of of the deadband due to
                                      'encoder limitations and noise
```

```
                                               'which ever is greater (units of 2us)
NeutralUpperLimit      CON 790        'This is the upper limit of of the deadband due to
                                      'encoder limitations and noise which ever is greater
                                      '(units of 2us)
PulseWidth             VAR Word       'Pulse width measured from receiver
Desired_wheel_speed    VAR Word       'The desired RPM is based on the rpm under "normal"
                                      'no load driving.  The actual RPM will vary
                                      'depending on the load imposed on the TXT-Robot.
                                      'For example, at half throttle the TXT-Robot may go
                                      '5mph on flat "normal" conditions, but at half
                                      'throttle going up hill the TXT-Robot may actually
                               be
                                      'going 4mph.  Therefore the Desired_RPM will vary
                                      'from actual RPM.
Last_Drive             VAR Word       'This variable stores the previous desired wheel
                                      'speed
EncTime                VAR Word       'Time the encoder is HIGH in 2us units
                                      'The variable "EncTime" is read from the encoder
                                      'output "0 for black" and "1 for white"
Measured_wheel_speed   VAR Word       'Calculated wheel speed
Drive                  VAR Word       'Amount of total drive
Err                    VAR Word       'Amount of error present
Ei                     VAR Word       'Cumulative error for integral calculations
P                      VAR Word       'Amount of Proportional drive
I                      VAR Word       'Amount of Integral Drive
D                      VAR Word       'Amount of Derivative Drive

LastErr                VAR Word       'Holds last RPM error for derivative drive
IntCount               VAR Nib        'Variable for counting cycles for integral drive
BitCounter             VAR Bit        'Used to create a 0-1-0-1 counter to alternate
                                      'between pulsing-in "EncTime" and the PID algorithm
'
'--[Initialize]------------------------------------------------------------------------
'
DirL = %00001000       'Make pins 7-0 either inputs=0 or outputs=1
DirH = %00000000       'Make pins 15-8 either inputs=0 or outputs=1

'** Configure StampDAQ *******
'sPin   CON     16     'Serial Pin - P16, Programming port
'Baud   CON     6      'Baud mode for a rate of 9600, 8-N-1
'                      'BS2P, BS2SX use 240 for 9600, 8-N-1
'
'Pause 1000                           'Allow data communications to stabilize
'SEROUT sPin,Baud,[CR]                'Send a lone CR to ensure StampDAQ buffer is ready
'SEROUT sPin,Baud,[CR,"LABEL,TIME, Time Encoder High, Measured_wheel_speed, Error
'(Measured-Desired), P, I, D, NEW Desired_wheel_speed, Control Pulse Width",CR]
'Label Excel columns
'SEROUT sPin,Baud,["CLEARDATA",CR]    'Clear all data columns (A-J) in Excel
'*****************************
'
'
Desired_wheel_speed = 0        'Set the initial value of Desired_wheel_speed to zero
LastErr = 0                    'Reset Last Error to zero
IntCount = 0                   'Reset IntCount to zero
BitCounter = 0                 'Reset BitCounter to zero
Ei = 0                         'Clear cumulative error
P=0                            'Reset P
I=0                            'Reset I
D=0                            'Reset D
'
'
'--[Main]------------------------------------------------------------------------------
'


Main:
  Getdata:
    PULSIN 1,1,PulseWidth                      'Check pulse width sent from receiver
  CheckTransmitter:
    If PulseWidth < 500  Then SetNeutral       'This subroutine is needed because when the
                                               'transmitter is OFF a "zero" is pulsed in.
                                               'Therefore WITHOUT "CheckTransmitter" the
                                               'BASIC Stamp sends a pulse to the
                                               'electronicspeed controller and the truck
                                               'goes full speed forward.  This is needed
                                               'because when the tranmitter is off it is
                                               'NOT sending a neutral pulse width of 750
                                               '(in 2us units).

'  CheckNeutral:                               'This creates a deadband
'    If NeutralLowerLimit < PulseWidth Then CheckNeutralUpperLimit    NeutralReturn:
  CheckIfOpenLoopMode:
    IF IN10 = 1 Then Check_If_Obstacle       'If pin 10 is high then the Smart Car will
                                             'drive in open loop mode
  Alternate_looping:
    If PulseWidth > 750 Then Check_If_Obstacle       'This is determined by the maximum
                                                     'allowable time to pulse-in the
                                                     'variable "EncTime".  Another limit
                                                     'will be needed for the PID algorithm
                                                     'to work while driving in reverse.
                                                     'Used to create a 0-1-0-1 counter to
                                                     'alternate between pulsing-in
    BitCounter=BitCounter+1                          '"EncTime" and the PID algorithm.
    If BitCounter=1 Then CalculateMeasured_wheel_speed

  CalculateDesired_wheel_speed:
    Check_PulseWidth:
      If PulseWidth > 750 Then DWS_Curve_1
      If PulseWidth > 724 Then DWS_Curve_2
      If PulseWidth > 675 Then DWS_Curve_3
      If PulseWidth > 575 Then DWS_Curve_4

    DWS_Curve_5:                      'Desired_wheel_speed is calculated from the
      Desired_wheel_speed = 965       'calibration curve in the form of y=mx+b where x in
    Goto Calc_Drive                   'this case is the variable called "PulseWidth"
                                      'pulsed in from the receiver

    DWS_Curve_4:
      Desired_wheel_speed = 1911-(PulseWidth*/$01A5)
    Goto Calc_Drive

    DWS_Curve_3:
      Desired_wheel_speed = 4786-(PulseWidth*/$05E7)
    Goto Calc_Drive

    DWS_Curve_2:
      Desired_wheel_speed = 8874-(PulseWidth*/$0B8B)
    Goto Calc_Drive

    DWS_Curve_1:
      Desired_wheel_speed = 5614-(PulseWidth*/$0732)
    Goto Calc_Drive

'++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  '********* Calculate Drive Using PID Control
  Calc_Drive:

    '********** Calculate Error
    ErrorCalc:
      Err = (Desired_wheel_speed - Measured_wheel_speed)    'Calculate RPM error
'      If PulseWidth > NeutralPulseWidth Then Reverse_Direction
      Reverse_Direction_Return:
      IF Err =0 Then SetError 'IF ABS Err < 20 Then SetError        'This may improve the
                                                                    'steady state error by
                                                                    'preventing
                                                                    'oscillation


    '*********** Proportional Drive
    PropCalc:
      P = (Err * Kp)                         'Prop err = Err * Kp.

    '********** Integral Drive - Sign Adjusted
    IntCalc:
      Ei = Ei + Err                          'Accumulate err each time
      IntCount = IntCount + 1                'Add to counter for reset time
        IF IntCount < Ti Then IntDone        'Not at reset count? -- done
          IF Ei.bit15=0 Then Ei_positive
          IF Ei.bit15=1 Then Ei_negative

          Ei_positive:
            Ei=Ei/Ti                         'Find average error over time
          Goto Avg_Ei_Done

          Ei_negative:
            Ei= -((ABS Ei)/Ti)               'Find average error over time
          Goto Avg_Ei_Done

          Avg_Ei_Done:
            Ei = Ei * Ki                     'Int err = (integral err) * Ki
          I = I + Ei                        'Add error to total int.
          IntCount = 0                      'Reset int. counter and accumulator
          Ei = 0
      IntDone:

    '*********** DERIVATIVE DRIVE
    DerivCalc:
        D = (Err-LastErr) * Kd              'Calculate amount of derivative drive
                                            'based on the difference of last error
   SetError:
        LastErr = Err                       'Store current error for next deriv calc

  '********* Drive the TXT-Robot

  Calc_Control_PW:
    Drive = P + I + D
    IF Drive.bit15=0 Then Drive_positive
    IF Drive.bit15=1 Then Drive_negative

    Drive_positive:                         'Calculate total POSITIVE drive.
      Drive = (P + I + D)/1000              'Then divide by 1000 so that each gain, Kp,
    Goto DriveDone                          'Ki, and Kd are are in thousandths.


    Drive_negative:                         'Calculate total NEGATIVE drive.
      Drive = -(ABS (P + I + D)/1000)       'Then divide by 1000 so that each gain, Kp,
    Goto DriveDone                          'Ki, and Kd are are in thousandths.

    DriveDone:
       Desired_wheel_speed = Desired_wheel_speed + Last_Drive + Drive
      Last_Drive = Last_Drive + Drive

    CheckDesired_wheel_speed:
'      IF Desired_wheel_speed.bit15=0 Then Desired_wheel_speed_POSITIVE
'      IF Desired_wheel_speed.bit15=1 Then Desired_wheel_speed_NEGATIVE

    Desired_wheel_speed_POSITIVE:                          'Using the
        If Desired_wheel_speed < 218 Then PW_Curve_1p      'Desired_wheel_speed
        If Desired_wheel_speed < 507 Then PW_Curve_2p      'PulseWidth allows us to
        If Desired_wheel_speed < 802 Then PW_Curve_3p      'calculate the pulse width
        If Desired_wheel_speed < 965 Then PW_Curve_4p      'that will control the robot
      PW_Curve_5p:
        PulseWidth = 500
      Goto Control_PW_Done

      PW_Curve_4p:
        PulseWidth = (1917-Desired_wheel_speed)*/$009B
      Goto Control_PW_Done

      PW_Curve_3p:
        PulseWidth = (4825-Desired_wheel_speed)*/$002B
      Goto Control_PW_Done

      PW_Curve_2p:
        PulseWidth = (8948-Desired_wheel_speed)*/$0016
      Goto Control_PW_Done

      PW_Curve_1p:
        PulseWidth = (5554-Desired_wheel_speed)*/$0024
      Goto Control_PW_Done

'     Desired_wheel_speed_NEGATIVE:                        'This is only needed if
'         If Desired_wheel_speed < 218 Then PW_Curve_1n    'Desired_wheel_speed is
'         If Desired_wheel_speed < 507 Then PW_Curve_2n    'negative
'         If Desired_wheel_speed < 802 Then PW_Curve_3n
'         If Desired_wheel_speed < 965 Then PW_Curve_4n
'
'       PW_Curve_5n:
'         PulseWidth = 500
'       Goto Control_PW_Done
'
'       PW_Curve_4n:
'         PulseWidth = -(ABS (1917-Desired_wheel_speed)*/$009B)
'       Goto Control_PW_Done
'
'       PW_Curve_3n:
'         PulseWidth = -(ABS (4825-Desired_wheel_speed)*/$002B)
'       Goto Control_PW_Done
'
'       PW_Curve_2n:
'         PulseWidth = -(ABS (8948-Desired_wheel_speed)*/$0016)
'      Goto Control_PW_Done
'
'      PW_Curve_1n:
'        PulseWidth = -(ABS (5554-Desired_wheel_speed)*/$0024)
'      Goto Control_PW_Done

    Control_PW_Done:

      PulseWidth = PulseWidth MAX NeutralPulseWidth  'This should actual be 1000 in order
                                                     'for the PID to work in REVERSE

      PulseWidth = PulseWidth MIN 500                'This should actual be 500 in order
                                                     'for the PID to work in REVERSE
  Check_If_Obstacle:
    IF IN14 = 0 Then Drive_Robot                     'If pin 14 is LOW then allow forward
                                                     'motion of Smart Car
    PulseWidth = PulseWidth MIN NeutralPulseWidth    'If pin 14 is HIGH then only allow
                                                     'reverse motion of Smart Car
  Drive_Robot:
     PULSOUT 3,PulseWidth

'    SEROUT sPin,Baud,["DATA,TIME,", SDEC EncTime,",", SDEC Measured_wheel_speed,",",
'SDEC Err,",", SDEC P,",", SDEC I,",", SDEC D,",", SDEC Desired_wheel_speed,",", SDEC
'PulseWidth,CR]

Goto Main
End
'
'--[Subroutines]-------------------------------------------------------------------
'
CalculateMeasured_wheel_speed:
  PULSIN 12,1,EncTime                'This is actually Pin 13 on the LOWER BS2 connected
                                     'to Pin 12 on the UPPER BS2.  Measures the time the
                                     'photoreflector sees the black stripe in 2us units.

  Calc_wheel_speed:
    Check_EncTime:
      If EncTime > 2185 Then Check_If_Obstacle       'Measured_wheel_speed CANNOT be
                                                     'Negative
      If EncTime > 647 Then MWS_Curve_1
      If EncTime > 485 Then MWS_Curve_2

    MWS_Curve_3:
      Measured_wheel_speed = 1670-(EncTime*/$0214)   'Measured wheel speed in RPM
      PAUSE 4                                        'This pause helps keep the delay
    Goto CheckDesired_wheel_speed                    'around 20ms between outgoing pulses

    MWS_Curve_2:
      Measured_wheel_speed = 1163-(EncTime*/$0104)   'Measured wheel speed in RPM
      PAUSE 4                                        'This pause helps keep the delay
    Goto CheckDesired_wheel_speed                    'around 20ms between outgoing pulses

    MWS_Curve_1:
      Measured_wheel_speed = 717-(EncTime*/$0054)    'Measured wheel speed in RPM
      PAUSE 2                                        'This pause helps keep the delay
Goto CheckDesired_wheel_speed                        'around 20ms between outgoing pulses

SetNeutral:
  PulseWidth = NeutralPulseWidth
Goto NeutralReturn

CheckNeutralUpperLimit:
  If NeutralUpperLimit > PulseWidth Then SetNeutral  'This creates a deadband
Goto NeutralReturn

Reverse_Direction:                                   'These are summed because in reverse
                                                     'the Measured_wheel_speed should be
                                                     'negative.  I am assuming that the
Err = (Desired_wheel_speed + Measured_wheel_speed)   'robot is going backwards when the
Goto Reverse_Direction_Return                        'pulse width is > NeutralPulseWidth
'
'----------------------------------------------------------------------------------
```

C.2 Obstacle Detection Program

```
'{$STAMP BS2}
'==================================================================================
'File: Obstacle Avoidance.BS2
'
'This program uses Sharp GP2D02 infrared rangers to detect obstacles and
```

```
'stop the Smart Car only allowing the operator to reverse.
'
'
'Written by: Dan Gott
'Date: 3/25/03
'=========================================================================================
'
'--[Variables]----------------------------------------------------------------------------
'
NeutralPulseWidth             CON 770        'This is the center pulse width for neutral
                                             ' (units of 2us)
k                             VAR Byte       'Index variable
DistL                         VAR Byte       'Left infrared ranger distance (raw data not
                                             'calibrated)
DistR                         VAR Byte       'Right infrared ranger distance (raw data
                                             'not calibrated)
DistM                         VAR Byte       'Middle infrared ranger distance (raw data
not calibrated)
SteeringPulseWidth            VAR Word       'Pulse width sent to the Basic Stamp 2
Switch                        VAR Word       'Pulse width sent to choose to run the
                                             'system in open loop or closed loop mode
                                             'If the pulse width is LESS than
                                             '"NeutralPulseWidth" the system will be
                                             'closed loop.  If the pulse width is GREATER
                                             'than "NeutralPulseWidth" the system will be
                                             'open loop
'
'--[Initialize]---------------------------------------------------------------------------
'
DirL = %01000000
DirH = %01000000
'
'
LOW 3                                        'Set pin 3 low initially
LOW 4                                        'Set pin 4 low initially
LOW 7                                        'Set pin 7 low initially
LOW 11                                       'Set pin 11 low initially
'
'--[Main]---------------------------------------------------------------------------------
Main:
  GOSUB  SteerSmartCar

  CheckSwitch:
    PULSIN 10,1,Switch        'Pin 13 on the UPPER BS2 is connected to pin 10 on the
                              'LOWER BS2
    IF Switch > NeutralPulseWidth Then OpenLoop
    LOW 8                     'Make pin 8 LOW to operate in closed loop mode
                              'Pin 10 on the UPPER BS2 is connected to pin 8 on the
                              'LOWER BS2

  CheckInfraredRangers:
    MiddleInfraredRanger:
      HIGH 7
      PAUSE 3
      LOW 7
      FOR k=1 TO 70
        PAUSE 1
        IF IN5=1 THEN Check_DistM
      NEXT
    Check_DistM:
      SHIFTIN 5,7,2,[distm\8]
      PAUSE 1
    IF DistM >= 90 THEN StopSmartCar

  GOSUB  SteerSmartCar

    LeftInfraredRanger:
      HIGH 3
      PAUSE 3
      LOW 3
      FOR k=1 TO 70
        PAUSE 1
        IF IN1=1 THEN Check_DistL
      NEXT
    Check_DistL:
      SHIFTIN 1,3,2,[distl\8]
      PAUSE 1

  GOSUB  SteerSmartCar

    RightInfraredRanger:
      HIGH 11
      PAUSE 3
      LOW 11
      FOR k=1 TO 70
        PAUSE 1
        IF IN9=1 THEN Check_DistR
      NEXT
    Check_DistR:
      SHIFTIN 9,11,2,[distr\8]
      PAUSE 1

  LOW 4                         'Make pin 4 LOW to signal that there is NO obstacle in
                                'front of the Smart Car
                                'Pin 14 on the UPPER BS2 is connected to pin 4 on the
                                'LOWER BS2
GOTO Main

END

'--[Subroutines]-------------------------------------------------------------------------
'
OpenLoop:
  HIGH 8                        'Make pin 8 HIGH to operate in closed loop mode
GOTO Main                       'Replace GOTO Main with GOTO CheckInfraredRangers to allow
                                'the obstacle detection and steering
                                'mediation to work while in OPEN LOOP mode.

StopSmartCar:
  HIGH 4                        'Make pin 4 HIGH to signal that there IS an obstacle in
GOTO Main                       'front of the Smart Car

SteerSmartCar:
    PULSIN 6,1,SteeringPulseWidth      'Pin 5 on the UPPER BS2 is connected to pin 6 on the
                                       'LOWER BS2

  CheckLimitLeftSteering:
    IF DistL >= 90 THEN LimitSteeringLeft

  CheckLimitRightSteering:
    IF DistR >= 100 THEN LimitSteeringRight

  MediateSteering:
    PULSOUT 2,SteeringPulseWidth       'Pin 7 on the UPPER BS2 is connected to pin 2 on the
                                       'LOWER BS2 Pulse out the same signal being sent by
                                       'the operator.  There is no mediated control
RETURN                                 'for steering commands.

LimitSteeringLeft:
  SteeringPulseWidth = SteeringPulseWidth MIN NeutralPulseWidth
Goto CheckLimitRightSteering

LimitSteeringRight:
  SteeringPulseWidth = SteeringPulseWidth MAX NeutralPulseWidth
Goto MediateSteering
```

REFERENCES

1 J. Lowen Shearer, Bohdan T. Kulakowski, John F. Gardner, *Dynamic Modeling and Control of Engineering Systems*, (2nd ed.), Prentice Hall, Upper Saddle River, New Jersey, 1997, pp. 290 – 294.

2 J. Lowen Shearer, Bohdan T. Kulakowski, John F. Gardner, *Dynamic Modeling and Control of Engineering Systems*, (2nd ed.), Prentice Hall, Upper Saddle River, New Jersey, 1997, pp. 336 – 338.

3 Benjamin C. Kuo, *Automatic Control Systems*, (7th ed.), Prentice Hall, Upper Saddle River, New Jersey, 1995.

4 Benjamin C. Kuo, *Digital Control Systems*, (2nd ed.), Oxford University Press, Inc., New York, 1992.

5 Jean-Jacques E. Slotine, Weiping Li, *Applied Nonlinear Control*, Prentice Hall, Upper Saddle River, New Jersey, 1991.

6 Willie D. Jones, "Building Safer Cars", *IEEE Spectrum*, January 2002, pp. 82 – 85.

7 Richard Bishop, "A Survey of Intelligent Vehicle Applications Worldwide", *Proceedings of the IEEE Intelligent Vehicles Symposium 2000*, October 3 – 5, 2000, pp. 25 – 30.

8 L. Verhoeff, D.J. Verburg, H.A. Lupker, L.J.J. Kusters, "VEHIL: A Full-Scale Test Method for Intelligent Transport Systems, Vehicles and Subsystems", *Proceedings of the IEEE Intelligent Vehicles Symposium 2000*, October 3 – 5, 2000, pp. 369 – 375.

9 Michael B. Histand, David G. Alciatore, *Introduction to Mechatronics and Measurement Systems*, McGraw-Hill Companies, 1999.