

Technical Overview of Convergence Laboratory

Craig Robinson, Scott Graham
Convergence Laboratory
University Of Illinois

May 25, 2004

1 Introduction

This document is intended to give a technical overview of the workings of the Convergence Laboratory in CSL. The work documented is an 'as is' synopsis of the lab in the Summer of 2004. The first section is a general overview which leads onto an indepth discussion of each of the system components and their interfaces with the rest of the system. Where possible we have attempted to give insight into why and how the things work and what factors should be taken into account when attempting to modify the system.

2 Overview

The goal of the convergence laboratory is to study the architecture required to effectively combine the concepts of computation, communication and control to allow the technology to proliferate. As a way of investigating this concept we have established a testbed of autonomous vehicles which are observed by ceiling mounted CCD cameras. After some computation the information from these cameras provides a position and orientation of the model cars running on the track below. This information is fed to a dataserver which acts as an intermediate point for sharing information with the cars, a scheduler and any other services requiring position data.

There are several different high level control algorithms implemented on the system. In some cases a separate scheduler service plans trajectories for each of the cars so as to ensure the cars get to a desired location without a collision. In other cases the car is set to follow other cars, evade collisions or simply follow a predetermined trajectory.

In either case the movement of each car is controlled via a RF transmitter connected to a dedicated laptop. The function of the laptop is to communicate with the higher levels of the system and process the desired (and actual) location of the car so as to produce a control signal which is sent to the transmitter through a serial port connection.

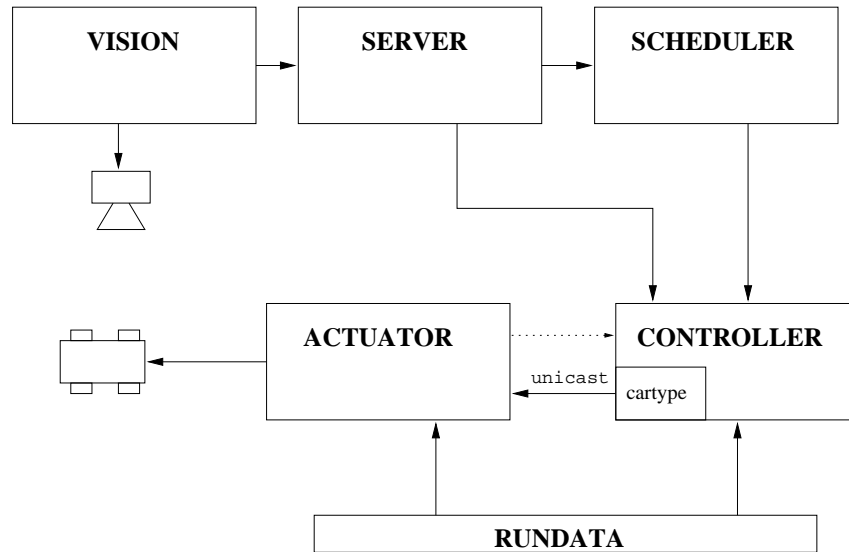


Figure 1: System software schematic.

3 Vision

The vision system is comprised of two cameras mounted on the ceiling above the track. They each cover opposite ends of the track with a small section of overlap. The cameras are digital but immediately convert their images into standard TV signal to be transmitted to the PC, where a *MATROX* frame grabber snatches images 10 times per second. The double conversion of the video signal reduces the quality of the images. The cameras are able to take much faster frame rates but the 10Hz has found to be sufficient. We have purchased a development and application licence from *MATROX* and in order to run the machine must have a 'key' inserted into the parallel port of each machine.

3.1 Vision Server

The processes running on the vision server are conceptually simple, but very complex to implement. There are *many* factors that come into play and the system is (thankfully) now sufficiently robustly so as to deal with many issues. In particular, the following should be noted:

RGB vs. HSV

It was found that the colours that were being used on the cars were not easily thresholded using the Red, Blue and Green colour scale. So, the Hue, Saturation and Value system is. This removes a large number of lighting problems related to brightness that

typically plagued the system.

Choosing colours

The colours chosen for the rooftops were not randomly assigned. From an initial sampling of *many* colours, a small group of eight were chosen that could be easily identified by the vision system. 'Easily' means that the range of Hue, Saturation and Value that uniquely defines the colour was fairly small, and hence a particular pixel is easily identified as being a certain colour. As a result, searching for a particular colour becomes a process of searching for values of H, S and V that fall within the colours limits. However, certain colours change their identifying properties depending on the position on the track and it may not be possible to recognise them all the time - in all the positions. Hence, the system was built such that an individual car can be identified *without* identifying all the colours on its roof.

Roof colour Layout

In order to orient the direction of the car, two colours (Pink and orange) were always placed as the center two colour patches. This leaves 6 colours to use in the remaining 4 spots. In order to choose good layouts, the concept of 'Hamming distance' was used. Simply put, this approach attempts to separate a group of signals from each other by the largest *hamming distance*. Taking an explanation from the web "Hamming distance is a measure of the difference between two messages, each consisting of a finite string of characters, expressed by the number of characters that need to be changed to obtain one from the other. E.g., 0101 and 0110 has a Hamming distance of two whereas "Butter" and "ladder" are four characters apart". Of the 360 combinations possible using the remaining 6 colours, those with the largest Hamming distance were chosen to give a set of approximately 18 possible roof layouts. Configurations where two adjacent cars might represent another car between them as well as personal preference (Scott has suspicions about the purple) was then used to reduce this number to a set of 15 colour layouts currently used on the cars, even if two color patches on each car could not be determined.

Process for Identifying a car

The process for identifying a car is rather complicated as a result of the many special cases that must be deal with. Here is a general approach:

- Scan through the array of pixels and create a set of three arrays - one for Hue, Saturation and Value.
- Use a bit masking operation on each of the three arrays that checks to see if a particular pixel lies within range of H,S and V for each colour. If it does it is assigned a 1, otherwise a zero
- Do a bitwise 'and' operation on the three arrays to produce a single array with areas of 1's that represent *blobs* of an individual colour.

- Increases the size of each blob and see if any intersect as they increase in size. Join any that do, and then shrink them to their original size.
- Compute the center of each of the blobs and create a new data structure that has blob colour and position
- Find adjacent pink and orange blobs and join their center points. Knowing that pink is always to the left of orange allows us to compute the orientation of the car as well as its center point.
- Look in a small pizza around the center of the car and find all the blob centers that fall within a slice. If more than one center lies in a particular slice, simply ignore one of the blobs and use the other.
- Compare the arrangement of these blobs to the known arrangement of the car roofs and identify which car is which. This is done by checking a row vector of the observed colours against the rows of an array of the known roof colours. (A column represents the position on the roof, the row the car number and the value the colour.) When the *first* match of at least two squares is obtained the collective blobs are identified as a car and given its number

3.2 Vision Software

There are two important pieces of software that run on the vision servers, `bounce` and `tracker`. `Bounce` is part of the real time control algorithm and simply replies to any message that requests the local time. `Bounce` is discussed later in Section 11. `Tracker` performs all the steps above and is written using many of the vendor provided *mil* functions (Matoix Image Library) in Visual C++. When running it will display the image that the camera sees as well as a dot at the center of each color patch and the car number it has identified. This information is sent to the `server` program running elsewhere. Information regarding the Hue, Saturation and Luminance for each color is loaded from a vision version of the `rundata` file.

4 Data Server

The job of the `server` piece of software is to provide information regarding position and orientation to anyone that requests it. It is fed an update whenever the vision server has processed a new frame. In the interim it has a simple Kalman filter to predict the position of each vehicle. It then sends out a global piece of information (UDP) containing the position, and orientation of each car. This is used by the cars as well as the scheduler. In addition the server can reply to a request to provide information to each individual car.

The output printed to the screen initially shows what connections it has established (such as with the `scheduler`). It then creates a 'node' for each vehicle which is observed by the vision system. When an individual controller requests its position the number of the car will be printed on the screen. Occasionally some error may require `server` to be restarted in which case it is advisable to restart `scheduler` as well.

5 Scheduler

The scheduler program allows you to control what algorithm the cars will be running. It obtains information regarding where the server software is running from the `rundata` file and then loads a menu of options of what can be run. Here is some general information:

- The sting printed to the screen reading ‘Sent handshake string and waiting to Exhale’ followed bay ‘Whew’ indicates it has connected to the server. However, if `server` is started after `scheduler`, pressing 3 to reset allows them to connect.
- Occasionally there is a problem loading all of the points used in the default trajectories option. Check that the scheduler receives all of the points when using this option.
- Using option 5 (Arvind’s Scheduler) will run the traffic simulator. However, just be aware of the the positions of the vehicles before starting as they will all head directly towards their starting bin at run time. Also, there is a timer after a run has completed before the next one begins....make sure you have stopped the system or run something else before the time expires!
- If ‘stop’ does not work for some reason, the secret command ‘666’ will cause everything to be killed. The origins of this ‘666’ term are unkown but are believed to be a relic of Scott’s pagan past life.

6 Controller

This section introduces the controller part of the system. It is important to realize that the controller is totally separable from the actuator described in Section 7. This means that the the control for any vehicle can be done *anywhere* and the desired control commands are sent to a laptop which communicates (or actuates) with the car. This allows a single computer to be the controller for any number of cars. However, in most cases (and for simplicity) the controller and actuator for a particular car are placed on a single laptop. Once a control action has been computed, the instruction is sent to the actuator. (Which may or may not be on the same machine.) The controller is comprised of several programs all called by one main loop program called `car`.

car This is the main loop type program that is used to call the other functions and programs. It contains no logic other than initiating the car and loading configuration files. In particular, if the car is restarting, a series of restart files are loaded. These files include information regarding what the absolute start time of the previous run was, how many iterations had been done until the car was stopped, the trajectory that the car should be following, and the state of the car when it was stopped. Using this restart information the controller is able to compute where the car should be and what it should be doing.

Cartype This is the realtime controller for each car and must be discussed more.

pathtype This program might better be known as the ‘interpolator’ as it’s function is to take the aperiodic position information in terms of waypoints¹ and produce a series of periodic waypoints as required by the controller. The set of waypoints can be provided in a multitude of ways (e.g. in a file or from the scheduler) and as a result may not even be spaced at regular intervals in time or space. The output of the program is a rolling horizon of the next few (usually 6) waypoints based on a linear interpolation of where the car is supposed to be in time *and* space based on the future way point that is provided. As an example, if the set of waypoints provided to `pathtype` were the vertices of a hexagon, the output would gradually trace a dotted line along the perimeter of the hexagon.

planner This program is used to do collision avoidance. It checks the current location of the car and its future path to ensure this does not collide with any other vehicles path. It is able to do this using the broadcast message sent by the dataserver (to all the cars) which contains information about all of the other cars direction and position².

plantSE This program estimates the current state of the car using a very simple Kalman filter. The function is conceptually simple but is complicated by having to deal with cases of lost or delayed (sometimes out of order) packages of position information.

7 Actuators

This section deals how a car is told to move. The first section details the software involve in the process as well as the machines on which it is run, and the second deals with how this affects the cars themselves. (A full description of the cars is given in section 9.)For this section it is important to note that an individual car is controlled by a dedicated laptop computer and we may often refer to a laptop (car) and speak synonymously of the car (laptop). In some cases I refer to the ‘car laptop’ and you are advised that this is *not* equivalent to a ‘laptop car’. ☺

7.1 Software

The software has typically been written and compiled on other machines and transferred to the car laptop. Hence, the laptops do not have any of the required header or compile files. All the software is located in the `\home\car` directory. Within this there is a directory for the C++ programs (`\convergence`) and the middleware (`\etherware`). You are required to be root in order to access the serial port which is needed to run the following programs:

rc This program allows the laptop to simply control the car through the microcontroller and RF antenna attached to the serial port. It is intended to enable debugging and calibrating of the cars and send simple forward, reverse or turn signals. Capital

¹A waypoint is typically defined as a position in x and y co-ordinates along with a time

²This is a separate communication to the unicast information sent to each individual car

letters between A and W move the car forward and back at increasing speeds (L is stop), and the same lower case letters (up to u) turn the car to the left and right. *NOTE:* No other programs that need to use the serial port should be running. If the car starts and stops with out moving very far, check that the actuator program is not running.

actuator This is a passive program that simply sits and waits for instructions to be received over either the wireless or LAN network from the scheduler program running on another machine. Its duty is to relay instructions to the serial port and will conflict with `rc` if it is running.

Power needs to be supplied to all of the RF controllers and the microcontroller boxes by turning on the power strip in the middle of all of the machines. It is possible to turn on individual machines - but it hurts your fingers. ☹

8 IP Addresses

The following IP addresses are in use:

Component	IP Address	Description
Car Laptop	192.168.1.1xx	xx stands for the car number
Data Server	192.168.1.2	Chosen for convenience
Vision Server	192.168.1.20x	x is 1 or 2
Scheduler	192.168.1.10x	Use x=9 for convenience
Wireless Network	10.0.0.xx	xx for anyone wireless
PDA's	192.168.1.5x	All PDA's are wireless

9 Cars

The cars that are currently used in the lab have been modified a lot from their original state. The schematic structure of the car is shown in Figure 9. In hindsight they are not the best choice for this application but have been slowly adapted to suit. In particular, the following adjustments have been made:

- The original worm gear provided with the car was powerful but *very* slow. Hence, it has been replaced by a 'High Power' gear box. The specs are to be found on some of the old boxes under the track. The 'High Speed' gear box ratio was not used as the motor is not powerful enough. There are two possible gear ratio settings with in the High Power range. Some cars have the Green-Red gears, while others the Yellow-Orange.
- The original motors have been replaced as they burnt out easily. These motors cost about \$5 and are from Slot and Wing. There are two different variations but the only real consequence is a slightly faster car. The motor that is currently used is not strong enough to have a faster gearbox.

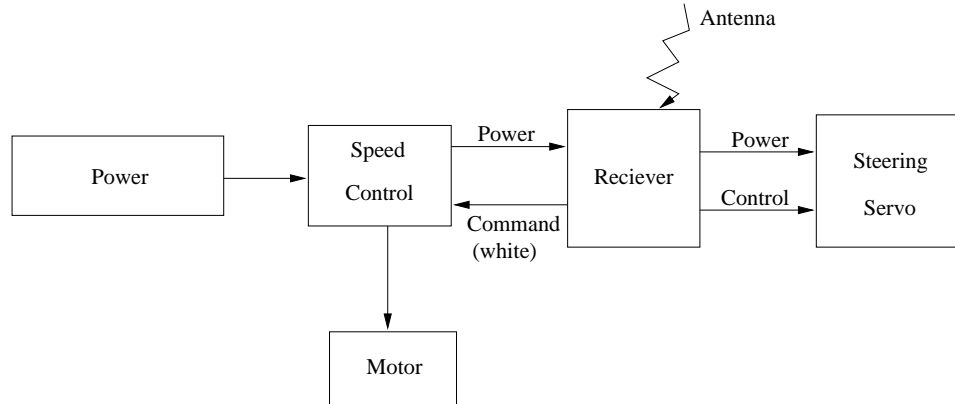


Figure 2: Schematic of car.

A box with all the extra car parts is located under the track. A small circular sticker on the side of the car gives the frequenc of the receiver on the car. This should match the number on a similar sticker on the bottom left hand side of the back of the RF transmitter.

The speed controller on the car has two tunable knobs, 'Neutral' and 'Proportional'. The Neutral adjusts the control point at which the car does not move (it shifts the neutral point forward and back), and the Proportional adjusts the how fast (in reverse and forwards) the car can go (It stretches it out).

9.1 Power

The power is supplied using a bank of (usually) 5 1.2³ Volt NiMH (Nickle Metal Hydroes) or Nicad batteries. The NiMH have two distinct advantages in that they do not develop a memory and they hold more voltage with respect to their size. The car needs at least 6 Volts to operate properly.

The batteries are charged using the AC//DC chargers using the *slow* setting and the 'J' type connector. Change the setting by pressing the *start* button and set the actual current using the knob to be about 250 milliamps. Do not use the fast charging unless it is really needed. Charging takes around 3hrs and can be repeated. Once completed they do loose some charge sitting in the charger, so remove them and place *in the charged box!*

9.2 RF Transmitter

The 6 channel RF transmitter is located under the track and is connected to the controller laptop though the 'training port' on the back of the unit. To do this the transmitter must be put in automatic mode by flipping the switch on the top left corner. The switch

³Not 1.5 Volts as common with size AA bateries

is an addition that replaces the functionality of the ‘trainer’ button. Flip the switch towards you to gain manual control. However, not all the cars are wired the same way and in some cases the left joystick will do all the control.

The transmitter has been adapted to be powered by a 10 Volt DC power source which plugs into the back of the console. Be carefull so as to ensure that it gets 10Volts by using ‘Blue’ chargers. The wiring inside works - but has not been tested so do not leave it on over night.

A small circular sticker on the side of the car gives the frequenct of the receiver on the car. This should match the number on a similar sticker on the bottom left hand side of the back of the RF transmitter. If the aerials are not fully extended or they are touching the ground or other objects, the cars tend to jerk and shake more.

10 Start up

The system has been designed such that each unit is independent of the others. This promotes robustness and should allow the the system to be configured in any number of ways. However, it has been found that it is better to start the system in particular order:

1. *Vision Servers:* These should be started by running the `tracker` and `bounce` programs on the desktop of the vision server (Section 3.1. `tracker` is very computationally intensive and should not be left running for a long period when not in use. The `\bounce` program is used to calibrate timing within the system. Please refer to Section 3.2 for more on these programs.
2. *actuator:* This is run on each individual laptop. When it starts it immediatly reads the `rundata` file and gathers information about which car it will be controlling, which set of calibration data to use, the IP address of the machine running server and the IP address of the machine running scheduler.
3. *Data Server:* This machine runs the `server` software and is the source for providing information on what cars are visible on the track. When the program is started it should indicate what cars can be seen. Once running it will continuously scroll a list of the cars that can be seen. Refer to section 4 for more on this. Information for each car regarding where the `server` software is running is stored in `rundata` file on each controller.
4. *Scheduler:* This is also setup to run on another machine and runs the `scheduler` program discussed in section 5. If it starts correctly the car will have a small twitch and remain still (provided its controller software is running). As with the `server` software, information for each car regarding where the `scheduler` software is running is stored in `rundata` file on each controller. The printed lines ‘Sent handshake string and waiting to Exhale’ followed by ‘Whew’ indicates it has connected to the server. If `server` is started after `scheduler` press 3 to reset and connect to the server.

11 General Software

This section deals with a bunch of software that I don't really know where to put at the moment.

bounce A very simple program that runs on any machine and is used to calibrate times on separate machines. When bounce receives a ping message from another machine it appends its local time (and any delay that it might have had before replying) and returns the message to the origin machine. There is different code for bounce on windows and linux machines.

translate This program initializes the ping that bounce replies to. It creates a packet of information that has its current time and IP address and sends it out and then waits for a reply. Once a reply has been received it records its own time and looks at the received message. It then computes the time taken for the package to do the round trip, and using half this time it computes the offset between its local clock and the clock of the other machine. The output printed to the screen looks like this:

210	4	10893482	10895224	o8230	r2
Machine ID	Counter	Origin Time	Dest. Time	Offset	Round Trip

The value of the offset is then saved into a `offset[machineID].dat` file and a history of all the pings are stored in the `cp[machineID].dat` file. Information regarding which machines to ping and their ID's are stored in the 'rundata' (or similarly named) files.

There is an interesting discrepancy regarding the output of this program when communicating between a linux and windows machine. Windows uses a cycle time of 10ms where as linux uses 1ms. As a result, the bound that you get on a return from bounce has a possible error of up to 10ms. As a result, if you plot the offset over time you get a sawtooth type form as the 10ms sampling increases or reduces in effect.

12 Thoughts to be included

1. Details on the configuration files on each of the cars, data servers etc
2. default trajectory files

13 Trouble Shooting

There are many small things that might go wrong with the system. These are a few things to consider:

1. Batteries do run flat! If the car is moving slowly, or not moving at all check battery.

2. Check that the power for the RF transmitter and the microcontroller are turned on! Sometimes the connections are unplugged, or a switch has been bumped. In particular check that the manual control switch is set correctly (section 9.2).