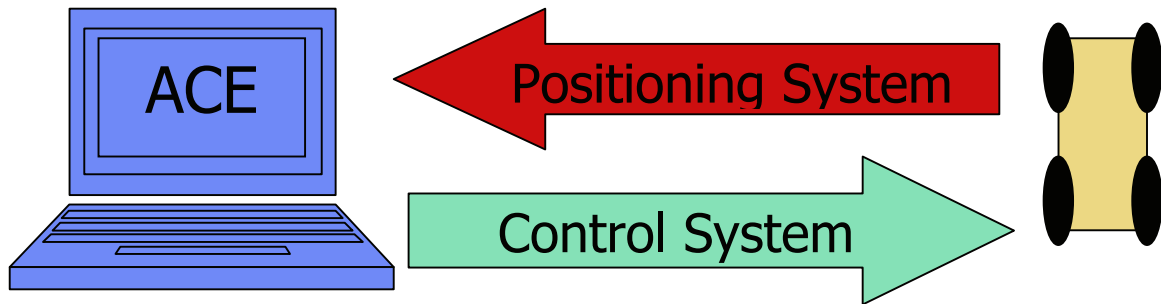


A Platform and Algorithm for High Speed Autonomous Control of a 1/10th Scale Model Car

Mark Woodward, Stanford University

Abstract This paper describes an effort towards creating a software/hardware platform for fairly testing driver algorithms designed to pilot 4-wheeled car like vehicles. Various approaches to a positioning system and control system are described in the paper, along with the optimal solutions to both: interfacing with a standard model car radio transmitter for the control system and a rotating laser-based system for the positioning system. The software architecture is also presented, including the key interfaces: Car and Driver. In addition, a simple driver algorithm is presented, which is based on a waypoint algorithm used in many racing video games.

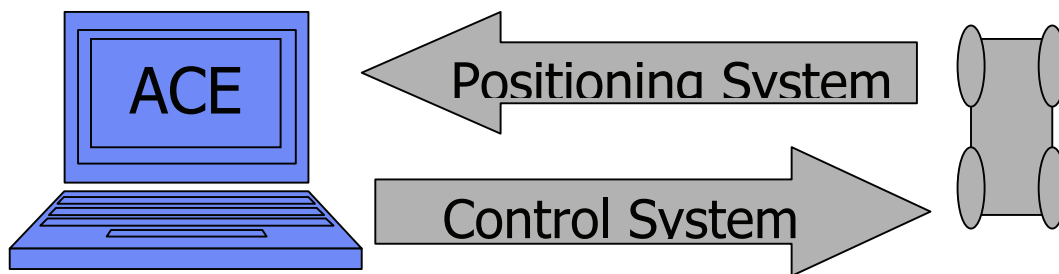


INTRODUCTION:

Control of 4-wheeled non-holonomic vehicles is an area with a rich research history; advances in electronics and artificial intelligence algorithms continue to push the area forward. In this paper, I describe a flexible platform I have developed to serve as a test-bed for future research in the area. I also explain a simple driver algorithm implemented to test the system. The scenario envisioned for the system is to test out various driver algorithms for piloting a model car around a two-dimensional racetrack. Before beginning, I made two simplifying assumptions: 1) The driver will have a priori knowledge of obstacles defining the track, 2) The driver will be supplied with absolute positioning information at regular intervals during the race. Some constraints apply to the assumption of positioning data; an absolute position must be supplied: 1) at a rate no slower than 10Hz, 2) to a precision no less accurate than 6 inches, 3) be able to represent an area no less than 10,000ft², and 4) be able to track an object moving within that space at 20mph. The project work involved creating a software user interface, called ACE (Autonomous Car Editor), for initiating and reviewing trial runs, creating a hardware positioning system to provide the position information about the car, and creating a hardware control system allowing the car to be controlled from software.

The paper describes in turn the various options, and final solutions, to 1) the ACE architecture, 2) the positioning system, and 3) the control system. This is followed by a description of the driver algorithm implemented to test out the system. Finally, some related work is described.

ACE - Autonomous Car Editor

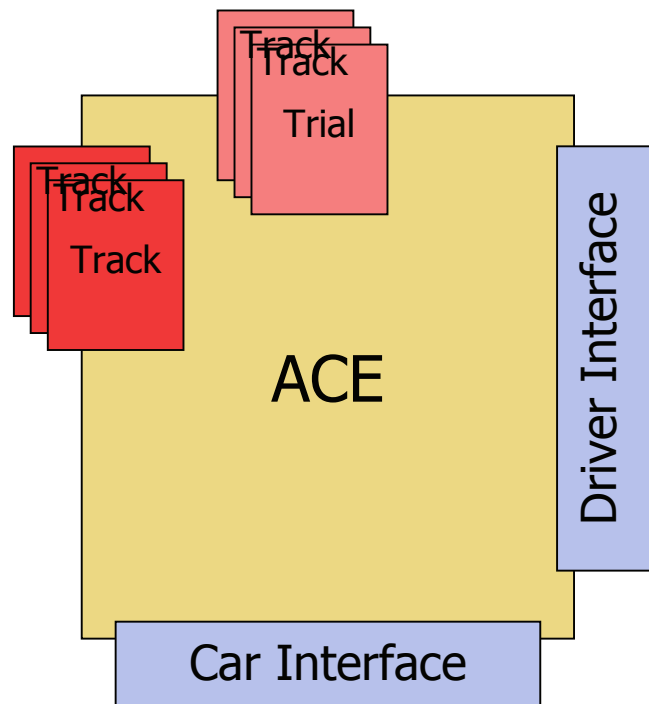


Overview of ACE:

ACE is a software application written in Java that provides an easy-to-use environment for testing different driver algorithms. A driver algorithm is a software component that will employ the ACE interfaces to pilot a car around a track. A user can graphically design a track by placing and manipulating waypoints, wayregions, and obstacles. Internally, obstacles are represented as undirected cycles (nodes, connected to the next node and the previous node by an arc, forming a loop) plus a point not on the cycle to designate the obstacle area. Points making up the obstacle can either be created by selecting a menu item or captured from the current car's position in the field. This second option allows a user to define a track by physically walking the car around the obstacles on that track.

In addition to defining a track, the interface allows a user to record and play back instances of a driver piloting the car around a track. I call these recorded instances trials.

A trial is started by selecting a "Start Trial" menu item, resulting in an event being sent to the driver, who then begins to execute the optimal path. The trial ends when the user selects the "Stop Trial" menu item. Trials and tracks are stored as XML files on the local filesystem, with, respectively, ".trl" and ".trk" extensions. A track can have many trials, but a trial can be associated with only one track. Once a trial has been recorded, it can be loaded and played, like a movie, at any time. Having the recorded trial data will be useful for future performance analysis.



ACE Architecture:

Two important software interfaces in the ACE architecture are the Driver Interface and the Car Interface. Through the Car interface, control signals, such as wheel position and pedal position, are sent and car position events are thrown. Thus, as long as a software component implements the car interface, ACE can use it. In the current implementation, two objects implement the Car interface. The first is the obvious object

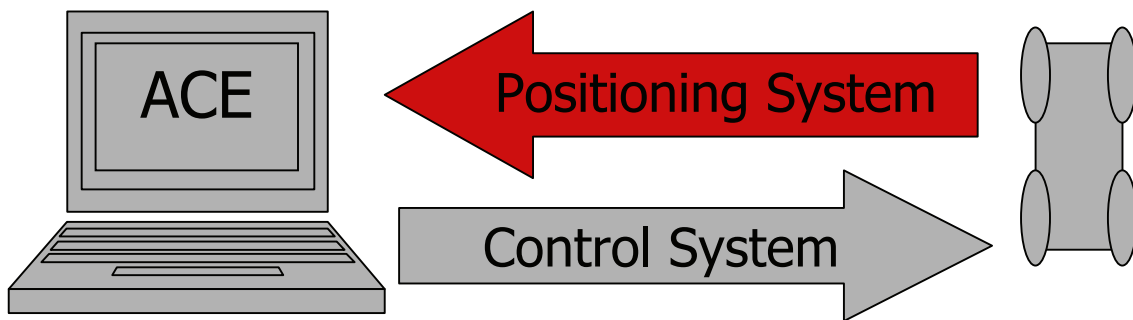
that interfaces with the hardware control and positioning systems. The second is a less orthodox use of the interface. Instead of implementing a separate trial playback viewer, I have an object that reads the trial file and through the Car interface generates wheel, pedal, and position events at the appropriate times. By doing things this way, I was able to use the same display code for displaying an actual trial as for displaying a playback. Another useful implementation of the Car interface would be a software simulator. This way a driver could be tested in simulation, before running on hardware.

Through the Driver interface, track changes and trial signals are sent. The implementing Car interface object is also sent through the Driver Interface. This gives the driver a direct hook into the low-level control and signals of a Car. Higher-level abstractions could also be built on top of the Car interface (for example, providing curve specifications, instead of wheel positions) but these higher-level abstractions would be selected on a driver-by-driver basis, and as such, should not be included in the base interface.

The Future of ACE:

The choice of Java was made primarily for the language's ease of use. The time allotted for the software code was only 1 week, which did not allow very much time for tracking down “.dll” link errors and memory issues. That being said, I do feel that more advanced driver implementations would benefit from the speed of C/C++. I am currently debating re-implementing the architecture in C++. Now that the class/architectural details have been worked out, that task would be easier.

POSITIONING SYSTEM:



Overview of the Positioning System:

The positioning system is a combination of hardware and software that calculates the location of the car in a 2-Dimensional coordinate system. Again, the requirements of the system were as follows: greater than 10Hz update rate, less than 6" accuracy, and greater than 10,000ft² area. As for the environment, I assume line of sight at all times. I explored four options to satisfy the above constraints: 1) a vision/fudicial system, 2) an ultrasound beacon system, 3) a Carrier Phase Differential GPS-based system, and 4) a rotary optical system. Only the rotary optical system was able to meet the above constraints, given the additional constraints of time and money.

The first system explored was the vision/fudicial system. It uses an overhead camera and vision software to track known markings known as fudicials on the car. Given my time constraints, I opted to leverage a software package called Mezzanine [1] created by a group at the University of Southern California. Mezzanine runs on Linux and utilizes a standard video capture interface called Video4Linux (<http://www.video4linux.net>). The use of Video4linux allows one to use any type of video capture hardware, from a simple T.V. card, to a Web-cam, to a high-end video camera.

To use the Mezzanine software, one specifies a translation from camera coordinates to real world coordinates and a description of the fiducials to be tracked. Mezzanine then outputs real-world object coordinates to a memory-mapped file, from which the user can read. I intended to use a Logitech Quickcam Pro 3000, which is capable of providing the desired resolution and frame rate.

After a week of trying to get the software to work with my hardware, Linux version, etc., I finally threw in the towel. I do think that the software has real potential though, and I still intend to get it running once I have more time. The benefits of such a system are many: 1) it allows one to track numerous objects, limited only by the processing power of the computer and quality of the camera, and 2) it requires no external hardware. I would like to thank Gabe Sibley and Andrew Howard, the creators of Mezzanine from USC. They were extremely helpful with all questions I had; they were even willing to dive into the Mezzanine code with me and try and work out the problems. Envisioning weeks of code searching, I decided the safest bet was to explore another option.

The second positioning system I explored was an ultrasound beacon system. In this setup, the car is equipped with an RF transmitter and an ultrasound receiver. The beacons are equipped with an RF receiver and an ultrasound transmitter. The car sends out a message coded for a receiver, and waits for an ultrasound “blip” to come back. The car starts a timer after the transmission of the RF signal and stops the timer when it hears the ultrasound response. Based on the time recorded, the speed of sound, and the known locations of the beacons, the position of the car can be triangulated.

Most ultrasound beacon systems use an ultrasound pulse to trigger the beacons and not an RF signal. By using an RF trigger, I eliminated half of the propagation delay, since the propagation time of radio signals is effectively instantaneous relative to the speed of sound. More importantly, I eliminated error from ultra-sound echoes, a common source of errors in such a system, since the first ultrasound signal you hear must be the straight-line distance, assuming line of sight. Initial testing demonstrated update rates of faster than 10Hz and precision to within a couple inches. But after bread-boarding the circuits, I was disappointed to discover that I had a serious problem with range. I was only able to get a distance of 20 feet. I realized that I would have to increase the quality of my speakers to get better accuracy, but I lacked the funds for such an investment, so I had to abandon this option.

One interesting thing about this system is that almost all of the error in position information is introduced by the speed of the vehicle. Even though the RC car never reaches speeds above 20mph, the speed of sound is slow enough to create problems. That said, the maximum error introduced was still ~4 inches. I had planned to correct for this error, obtaining even more precise positions, by using a dead reckoning-like adjustment based on the believed speed of the car and the known location of the sensor. But then again, this error occurs at a time when precision is least important. That is, when the car is traveling at its top speed, which only happens on large long straight-aways, where a conservative path would introduce minimal delays.

Another problem with ultrasound systems is the sensitivity to temperature; a 10-degree shift can introduce a greater than 6inch error. Again, since the position of the sensors is known, and since planar operation is assumed, this error could be removed in

software. In the future, I may still pursue this option, since it has the nice ability to locate objects in 3-Dimensional space.

About the time I was discarding the Ultra-sound beacon option, I had a meeting with Professor Steve Rock from the Stanford Aero-Astro Department. He suggested looking further into Carrier Phase Differential GPS [2]. I had originally abandoned this option on the assumption that the hardware was too expensive. It turns out that units have come down to ~\$200.00. The Global Positioning System, or GPS, is a constellation of satellites with precisely synchronized atomic clocks that continually emit a signal. By shifting the clock in a receiver unit, you can match the signal of satellite to an internal signal, based on the amount of shift and the speed of light, you can calculate the distance to the satellite. Given the distance to 3 or more satellites and the position of those three satellites, you can triangulate back to your position. The problem with straight GPS is that errors are introduced as the signal passes through the atmosphere, and it is difficult to predict these errors. This limits straight GPS to ~20meter accuracy.

The next advancement is called Differential GPS, or DGPS. DGPS uses a second receiver, called a base station, which is positioned at a known location. Since the base station knows its actual position, it can calculate the error in the signals from each satellite. The receiver can then transmit these errors to the mobile receiver, which can use them to adjust their measurements. As long as the mobile receiver is within a few hundred miles of the base station, the chances that it is under the same portion of the atmosphere relative to a given satellite are high. This type of correction can produce meter level accuracy, but is still too imprecise for my application.

Carrier Phase DGPS addresses the following source of error. While the carrier signal for GPS has a frequency in the GHz range, the actual pseudorandom message used to synchronize the receivers has a much lower frequency, producing a much higher period of about 1 microsecond. Thus a receiver could sync itself with a satellite signal, but still be off by a microsecond. This produces the meter error plaguing DGPS. Carrier Phase DGPS takes advantage of the GHz carrier frequency. It first synchronizes up with the pseudorandom signal and then tries to sync phase with the carrier signal, which is running at a much higher frequency. Adding this adjustment to the aforementioned DGPS produces cm level accuracy.¹

While I am extremely excited about this technology for future projects, the expense and time of integration were prohibitive. In addition, I would have had to devise a system of transmitting the position information back to the laptop, once it had been calculated, and this system would need to have a data rate high enough to encode positions on a global scale.

The next and final design I tried uses a rotating laser in combination with three static co-linear sensors to triangulate the position of the vehicle. The rotating laser is mounted on the car and rotates at a known rate. From the distance between the light sensors, the rotation rate of the laser, and the time the sensors are triggered, the position of the car can be calculated. One advantage of this design is that the accuracy can be increased, simply by slowing down the rotation of the laser. Of course, there is a trade off: slowing down the laser can increase the stationary accuracy, but it also can increase errors due to motion of the car between same cycle sensor readings.

¹ A good summary of the various GPS technologies is located on the Trimble web site at <http://www.trimble.com/gps/how.html>.

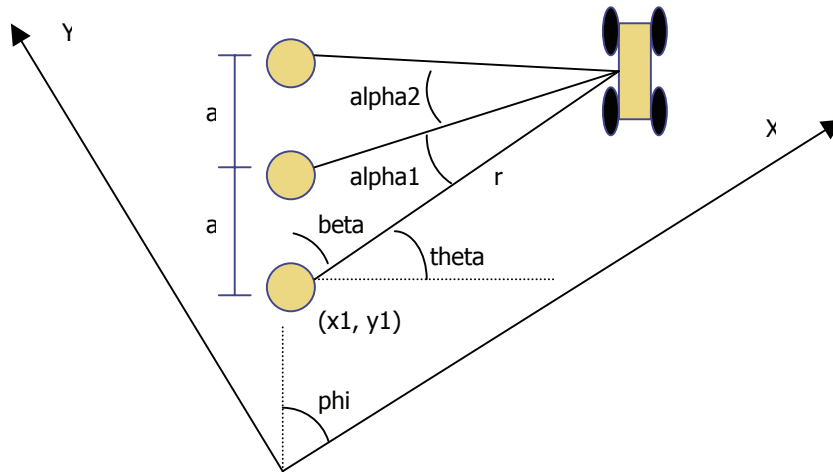


Figure 5: Components involved in the calculation of the cars position

The rotating laser component is made up of a small 4mW laser, fitted with a vertical line lens, and mounted on a modified hobby servo, equipped with a potentiometer to control the rate of spin. The hobby servo is used to gear down the motor, which has a minimum voltage higher than that which is necessary to directly produce the desired rate.

The three sensor components are made of a simple CDS cell, an op-amp, and a potentiometer to adjust the trigger intensity. When the light falling on the CDS cell is above the configured threshold, the op-amp sends a pulse to an RJ11 (phone jack) connector. A standard phone cord carries the signal to a micro-controller that alerts the computer of the event via a DB9 connector and RS232.

Positioning System Software:

The software for the rotating laser positioning system is broken up into two parts, the PIC micro-controller code and the computer code. The micro-controller code just runs a simple loop, constantly checking for a high voltage on one of the three input pins, which will be triggered by one of the three sensors. When an event is detected, the chip

sends a 3-byte message to the computer's serial port. The first byte is the ID of the sensor that triggered the event. The next two bytes contain the 2-byte value of the micro-controller's clock at the time of triggering. The computer software receiving the messages from the micro-controller was written using the Java Comm package, which gives Java applications access to the serial port. When the software has received times for all three sensors, it calculates the position of the car using the formula in figure 6 and triggers a position-changed event to all car listeners.

- $x = x1 + r\cos(\theta)$
- $y = y1 + r\sin(\theta)$
- $r = \sin(\alpha1 + \beta) / \sin(\alpha1)$
- $\beta = \arctan[(2\tan(\alpha1)\tan(\alpha2)) / (\tan(\alpha2) - \tan(\alpha1))] - 1$
- $\theta = \phi - \beta$.

Figure 6: formula for calculating the position of the car

The rotation rate of the laser could be statically specified by the user, but it might change during the trial, so it is better to calculate the rate dynamically. This is done by averaging the durations between the last 10 reception times of one of the sensors, and then using this average as the rotation rate in the position calculation. In addition, signals are discarded if an id has been skipped, signifying that a sensor has not fired. Signals are also discarded if the time between messages from consecutive ID's is larger than that which is possible given the current frequency, signifying that the sensor reading are not from the same cycle.

Almost all of the positioning systems explored required some form of micro-controller programming. I would like to thank CCS for providing a PIC C² compiler free

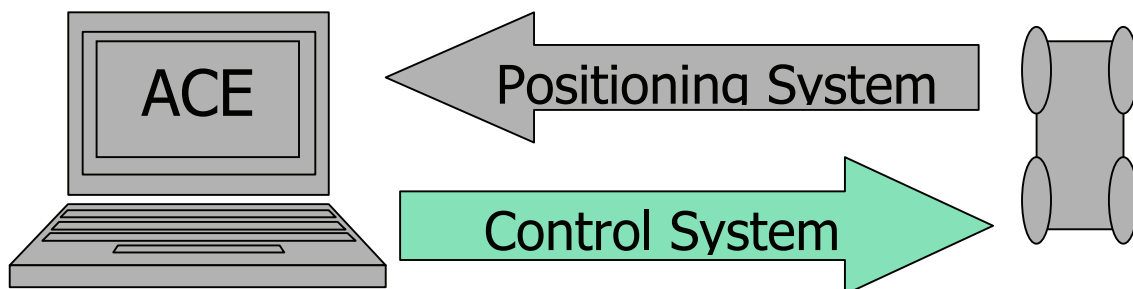
² Information on CCS products can be found at <http://www.ccsinfo.com>

of charge. Being able to write the code in C, as opposed to assembly, saved me countless hours of debugging.

Analysis of the Positioning System:

The next version of the positioning system will most likely be based on Carrier Phase Differential GPS. The CP-DGPS components seem to be reasonably priced and sufficiently accurate. The only problem with that system will be that the position information is initially determined on the mobile car and not at the computer, as it is with the current system. This will require me to either move control onboard, losing some ease of use, or message position data back to the computer, requiring substantial bandwidth. But the benefits of a CP-DGPS system definitely outweigh the disadvantages. For one thing, I won't be limited to certain weather conditions: the current system is very sensitive to light; the best performance is achieved at night, which is very limiting for demonstrations.

CONTROL SYSTEM:



Overview of the Control System:

The control system is made up of hardware and software components that allow software drivers running on the computer to issue wheel and velocity commands and then remotely control the R/C car based on these commands.

Control System Hardware:

Typical radio control cars have a radio transmitter, receiver, servo, electronic speed controller, electric motor, and batteries. The receiver receives a signal on one of its two channels, and then generates a Pulse Width Modulation (PWM) signal to either the servo or the speed controller, depending on which channel the signal was received. The servo adjusts its head position based on the PWM signal, which in turn controls the position of the wheels. The speed controller takes its PWM signal and adjusts the voltage across the motor leads accordingly.

One way to implement a control system, while still taking advantage of many of the standard RC components, would be to replace the receiver with a micro-controller, which could generate PWM pulses, but also execute code. This solution would work, but since I wanted to control the car remotely from a laptop, I needed a wireless solution.

Another option would be to use the aforementioned micro-controller, but create a wireless communication link between the computer and the micro-controller on the car.

This would require additional hardware and be fairly costly to provide any reasonable range.

In their simplicity, the standard R/C car components offer the perfect combination of wireless control with excellent range. All that is needed is a way for the computer to interface with the radio transmitter. Training radios are available with plugs to connect to another radio, but these radios start at around \$500.00. Since I already had a standard 2-channel radio, I decided to try and modify it. The style of the radio is called “pistol grip”, which means that one holds it like a pistol. The trigger controls the speed and a wheel exists to control the position of the wheels. Both the trigger and wheel are hooked up to potentiometers. As the driver changes the trigger or wheel position, a different voltage is applied across the respective potentiometer leads. By using a digital potentiometer and hooking into the mechanical potentiometer leads, I can digitally duplicate the behavior of the mechanical potentiometers. I chose the DS1803 digital potentiometer chip. It contains two onboard pots and is controlled via a 3 wire I2C bus (Clock, Data, GND). Using a voltage translator, I was able to directly bit bang the I2C protocol from the serial port, using the RS232 DTR and RTS signals.

Control System Software:

The control system software consists of a portion of the car class layered on top of a digital potentiometer abstraction class. The car class provides a wheel and pedal abstraction to the driver class, and the digital potentiometer class abstracts away the underlying bit-banging I2C details. The car class throws wheel and pedal changed events, so that visualization and logging components can keep up to "speed".

Analysis of the Control System:

I created and debugged the Radio Transmitter interface on my 1.7Ghz desktop computer. When running the program on my laptop, a 233Mhz machine, there appeared a noticeable delay between the issuance of a command and the reception at the receiver. I believe the delay occurs because the UART on the laptop is not designed for high speed flipping of the DTR and RTS pins. In the next revision, I plan to use a micro-controller between the digital potentiometer and the computer, which will allow me to command using standard Asynchronous RS232. Moreover, I will use a potentiometer controlled via stepper pulses; which will alleviate the overhead of I2C.

DRIVER:

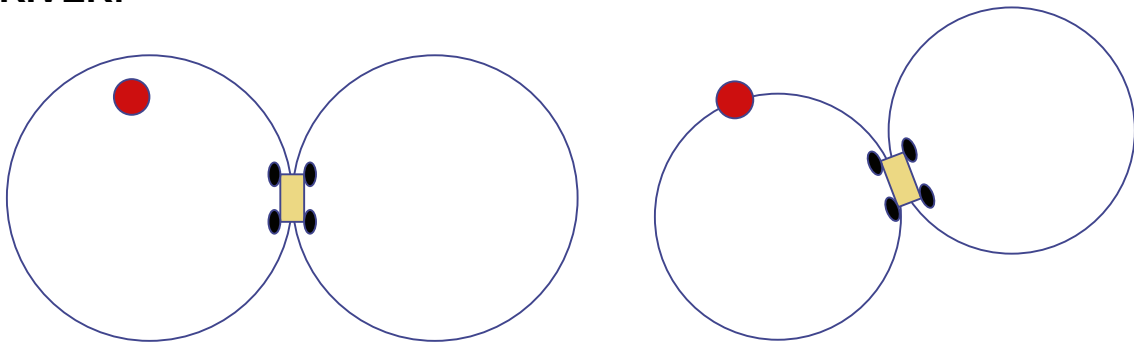


Figure 8: On the left, the driver finds itself going too fast. If it continues, it will miss the waypoint. On the right, the driver has slowed down, decreasing its minimum turn radius, and allowing it to hit the waypoint.

Overview of the Driver Algorithm:

The driver algorithm I implemented to test the system is a version of the standard waypoint algorithm used for computer opponents in race simulation video games, with the addition of accurate vehicle dynamics. It works as follows. A set of waypoints and wayregions are defined. The driver begins with the first waypoint as its next objective.

When the driver enters the wayregion for the current waypoint, the next waypoint becomes the objective.

The allowable turn radius is restricted by the current speed and the friction coefficient of the tires. At any given point in the configuration space (velocity, position, direction), the minimum turn angle projects two circles, both with the car on the circle, and with the direction of the car tangent to the circles. One of the circles defines the minimum angle for a left turn given that configuration and the other defines the minimum angle for a right turn given that configuration. If the next waypoint is within either of these circles, then the driver must slow down or it will miss the waypoint. If the waypoint is outside both circles, then the driver can continue to accelerate, all the while turning at the minimum allowable radius: turning to the right if the waypoint is to the right of the line defined by the (position, direction) vector and to the left if the waypoint is to the left. If the waypoint is on the line defined by the (position, direction) vector, then the driver continues to accelerate and maintains the current heading. If the waypoint is located on the circle, then the car continues to turn at the current angle and maintains the current speed. This driver implementation is a temporary solution, developed to test the system and to be used as a benchmark for more advanced drivers.

Future Drivers:

Immediate plans include incorporating maximum wheel change velocity into the path planning and moving to more A.I.'ish framing of the problem, with configuration spaces, etc. There exist limitless approaches to try out; for example, I could use some

form of machine learning to adjust the constraints of the car dynamically: tire friction coefficients, max acceleration, max velocity, etc.

I would also like to incorporate some form of opponent avoidance. One option would be to take the detected position of an opponent, assume a velocity, and then project the opponent as an obstacle into time space, whose path is defined by the path the current driver would take.

RELATED WORK:

While a deeper study of current research is warranted, there are a few related projects I came across in designing this system. The first project I looked into when exploring the positioning system was the Hummingbird project at Stanford [2]. This project involved controlling a model helicopter using only data from Carrier Phase Differential GPS and from no other telemetry data. The project was able to accurately control the Helicopter to within inches of desired positions. Since then, several helicopter projects have sprung up at various universities. For example, both U.C. Berkeley and C.M.U. have developed similar programs.

Also at Stanford, the Dynamic Design Laboratory³, led by Chris Gerdes, has developed a 1/4 scale R/C car platform. They have used the platform to develop a GPS-based sideslip estimation system, a GPS-based tire slip estimation system, and many other noteworthy endeavors.

The final Position System was based on work done by a group at a research corporation called MTI. Their work is explained in a paper on sensing systems [3].

Their system has the ability to locate an object to within 2 centimeters, at 40Hz, and over an 8 acre field.

SUMMARY:

The project as a whole has met the requirements set forth for it. In addition, the remotely controlled, 1/10th scale hardware brings the cost of development and experimentation from thousands of dollars down to just hundreds. Because of the componentized nature of the platform, incremental upgrades will be possible with minimal to no re-engineering. Moreover, the system as a whole shows promise as more than just a research tool. For example: the speed, size, and mobility of platform would be ideal for use in urban warfare settings or for surveillance and reconnaissance. I look forward to exploring future applications of the system and to using the system in my search for an optimal driver algorithm.

³ Information on the Stanford Dynamic Design Laboratory can be found at <http://www-cdr.stanford.edu/dynamic>

References:

1. Howard, Andrew. *Mezzanine User Manual*. <http://playerstage.sourceforge.net>
2. Conway, Andrew (1995, March). *Autonomous Control of an Unstable Model Helicopter Using Carrier Phase GPS Only*. Electrical Engineering Ph.D. Thesis, Stanford University
3. Borenstein, Everett, and Feng (1962). Where am I? Systems and Methods for Mobile Robot Positioning. A. K. Peters, Ltd., Wellesley, MA