ISSN 0280-5316
ISRN LUTFD2/TFRT--5730--SE

Arbiter and Simulation for for Team Caltech in DARPA Grand Challenge

Henrik Kjellander

Department of Automatic Control
Lund Institute of Technology
November 2004

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
November 2004

Document Number
ISRNLUTFD2/TFRT--5730--SE

Author(s)
Henrik Kjellander

Supervisor
Richard Murray Caltech, USA.
Anders Rantzer LTH in Lund, Sweden.

Sponsoring organization

Title and subtitle
Arbiter and Simulation for Team Caltech in DARPA Grand Challenge (Förbättringar och simulering för Team Caltech i DARPA Grand Challenge)

Abstract
I would like to thank my mentor Richard Murray at the Control and Dynamical Systems Department of California Institute of Technology (Caltech) for giving me the opportunity of conducting my master's thesis at Caltech and for his support during the project. I would also like to thank my other mentor Anders Rantzer at the Department of Automatic Control of Lund Institute of Technology under whose supervision this master's thesis was conducted.
Thanks also goes to Lars Cremean who has been very supportive during the whole project as my co-mentor at Caltech. I would also like to thank the members of Team Caltech for their help as well as making my visit at Caltech a very pleasant time: Elliott Andrews, Jeremy Gillula, Ben Brantley, Sue Ann Hong, Dimitry Kogan, Kristo Kriechbaum, Haomiao Huang, Henry Barnor, Adam Craig and Alan Somers.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
70

Recipient's notes

Security classification

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Purpose

The purpose of this master thesis was to work in the large project of Caltech's autonomous vehicle in the DARPA[1] Grand Challenge race[2]. The work consists of improving decision making, create a simulation environment and to assist in various project management tasks and system administration on the way to the next race that will be held in October 2005.

1.2 Goal

The overall goal for the project was to improve Team Caltech's vehicle platform that was used in the 2004 race so it will win the next race in 2005. In detail, the goals became to:

- Improve the Arbiter module of the planning software
- Build and demonstrate a complete simulation environment
- Participate in the process of decisions for the 2005 race vehicle
- Improve the project administration and development process in general

[1] Defense Advanced Research Projects Agency, see [2]
[2] A U.S. government sponsored race for autonomous vehicles

Chapter 2

Background and Motivation

For a long time *autonomous vehicles* have been a dream for engineers, but they have only appeared in science fiction so far. Some partially autonomous vehicles has seen the light, as for example highway following cars using camera sensors to track the edges of the road and radar for distance estimation to surrounding cars [3]. No one has yet been able to construct an autonomous vehicle that can handle arbitrary terrain and off-road environments. In 2003, the U.S. Department of Defense decided that in 2015, one third of all their ground vehicles shall be able to operate without a human driver. To speed up research in the autonomous vehicles areas, DARPA in 2002 announced a race for autonomous vehicles with a $1 million cash prize to the first vehicle that could travel on a specific route between Los Angeles and Las Vegas (see figure 2.2) in less than ten hours. The race, called DARPA Grand Challenge, caught a lot of interest at many companies and universities doing engineering and robotics. From Caltech a team was created, driven by mostly undergraduate students, which applied for the race. After one year of hard work a completely autonomous car was created and participated in the first race that was held in March 2004. Team Caltech[1] ended up #5 of 25[2] competitors after their vehicle, named Bob, had traveled autonomously for 1.3 miles until he got stuck on a barbed wire fence. Since no team was even close to complete the 142 mile course, a new race is announced for October 2005, now with a $2 million cash prize. This is what Team Caltech is currently working on.

[1] The official name for Caltech's team

[2] 25 teams were invited to the qualification event, only 15 were allowed to start in the race

Figure 2.1: Bob, Team Caltech's vehicle in the 2004 DARPA Grand Challenge race.

2.1 Autonomous vehicles

Fully autonomous vehicles are a very complex and difficult task to achieve. Many attempts have been made since computers has been available to the mass market. There are many different sizes and approaches that has been tested through the years. The by far hardest part is to perceive the environment and to take intelligent decisions out of the perceived data. A human brain is still far ahead of even the fastest computers because of it's ability to plan and maybe most important of all: to learn from it's mistakes. There's a lot of research going on within artificial intelligence about related techniques but autonomous vehicles that can handle a completely unknown environment is still quite far away because of the almost unlimited possibilities of different scenarios that can appear.

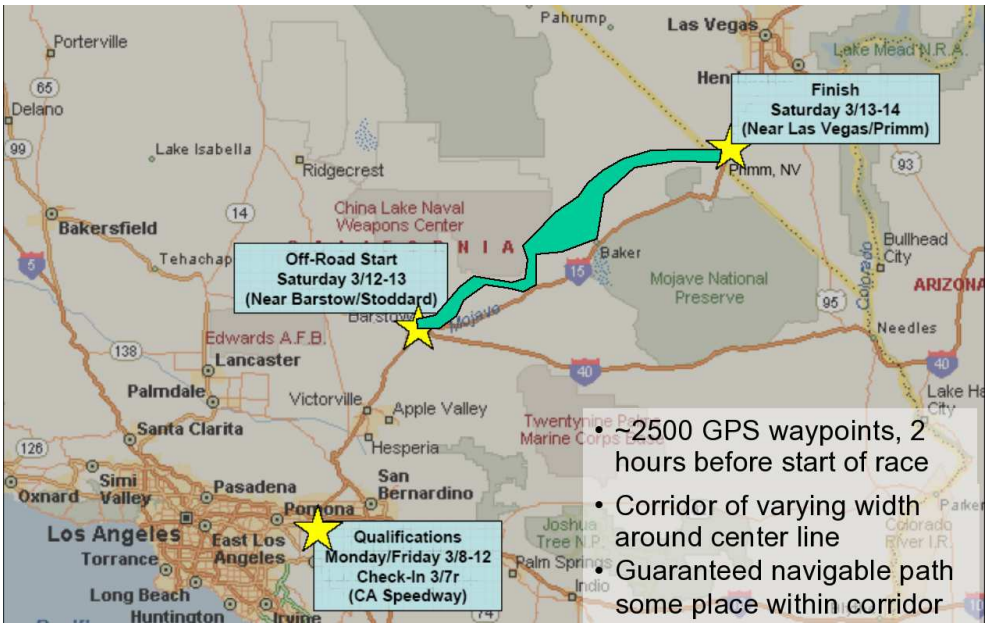

Figure 2.2: The course for the 2004 DARPA Grand Challenge race.

2.1.1 Arbiter

DAMN

DAMN[3] is a decision taking architecture created by Julio K. Rosenblatt at Carnegie Mellon University, Pittsburgh, USA [1]. It uses a collection of distributed task-achieving modules (or *behaviors*) that cooperatively determine a robot's path by expressing their preferences for each of various possible actions. An Arbiter then performs *command fusion* and selects a combined action that best satisfies the prioritized goals of the system which can be configured by weights on each input. This behavior-based architecture has the advantage of distributing different responsibilities to separate processes, that can be developed independent in separate teams. This flexibility was one of the key advantages that made Team Caltech implement this architecture since it would make it easy to have independent groups of students working on each module.
Each module analyzes its view of the situation considering a set of paths generated out

[3] Distributed Architecture for Mobile Navigation

of different steering angles (see figure 2.3). For each of these arcs it assigns a goodness value and a recommended speed. This data is called a *vote*. The Arbiter gets votes from all different modules and selects the arc that has the best goodness value, which is then sent to the actuators to steer the vehicle in that direction. For speed, the lowest voted speed for that specific arc is selected, for safety reasons. Lack of time for testing before the race made it impossible to fully test which weights on the different voter inputs that were suitable, so they all ended up being 1.0. This problem gave inspiration to the work of weight analysis with a genetic algorithm (see 4.2.2 at page 38).

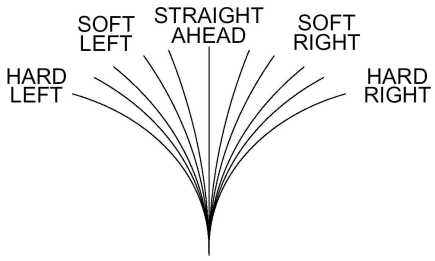

Figure 2.3: The different voteable arcs created out of different steering angle

2.2 DARPA Grand Challenge

2.2.1 Background

Grand Challenge is the name for the race that DARPA arranged in March 2004. It grew out of two Congressional mandates: one that allowed cash prizes to be awarded "in recognition of outstanding achievements that are designed to promote science, mathematics, engineering, or technology education in support of the missions of the U.S. Department of Defense."[4] and another that required "by 2015, one-third of the operational ground combat vehicles of the Armed Forces are unmanned."[5] Any team from any country[6] was allowed to apply for the race. To participate in the race, each team first had to file a technical report that described their approach and showed that their

[4]National Defense Authorization Act for Fiscal Year 2003 (H.R. 4546, Sec. 2374b)
[5]National Defense Authorization Act for Fiscal Year 2001 (S. 2549, Sec. 217)
[6]No government support or funding allowed and each team must have US citizen team lead

vehicle fulfilled all the safety requirements from DARPA. If the technical report was approved, the team had to participate in a qualification event, called QID[7], that was held three days before the race at the California Speedway[8]. The QID consisted of a small course with a variety of obstacles that represented typical situations the vehicles would be expected to face in the off-road terrain. Given a predefined file of GPS waypoints, a so called *RDDF-file*[9], the vehicle knew where it should go. If completing the QID the team was allowed to start in the Grand Challenge. Team Caltech did good in the QID and qualified as third team to the race.

Figure 2.4: Examples of different kind of terrain on the course of the race.

2.2.2 Rules

For safety and organizational reasons DARPA set up a large set of rules that the teams must follow. Here's a summary of the most important ones:

[7] Qualification, Inspection and Demonstration
[8] A large racing speedway in Fontana, California. http://www.californiaspeedway.com
[9] Route Data Definition File

- The vehicle must stay within the *RDDF corridor* at all times (this rule was given exceptions during the race). See figure 2.5 for a sample corridor plotted. The RDDF corridor is used for safety reasons to limit the area that needs to be secured by DARPA during the race.

- Each team must implement a radio signal system called E-stop used by DARPA. This means the vehicle must respond to a pause signal by stopping as soon as possible and enter a stand by mode. If a disable signal is given, the vehicle must also stop, but also shut down completely (race over). The pause signal was used in situations like if other competitors were close and needed to pass.

- GPS signals are the only allowed signals a team vehicle is allowed to use during the race.

- No communication of any kind may be performed with the vehicle (except the E-stop signal).

- It is not allowed to do any harm to the environment or destroy property during the race. Since Team Caltech ran through a barbed wire fence during the race, exceptions could obviously be made.

- Moving obstacles will appear on the course (handled by DARPA) and must be avoided.

During the 2004 race some teams relied almost entirely on *RDDF waypoint* following, lacking sensor processing about obstacles and intelligent behavior avoiding them. To prevent such approaches to be successful in the 2005 race, DARPA will put obstacles made to disable vehicles (tank traps, see figure 2.6), in the middle of roads between waypoints to efficiently end the race for such teams.

2.2.3 The Race

At 3:20 Saturday morning March 13, 2004, the teams were given the CD containing the RDDF, the file that included the waypoints, corridor widths, and speed limits that defined the course. By 6:00am, Team Caltech were at the starting gate ready to go.
At 6:40am, the green starting flag waved for Bob. At the mile mark, he veered off

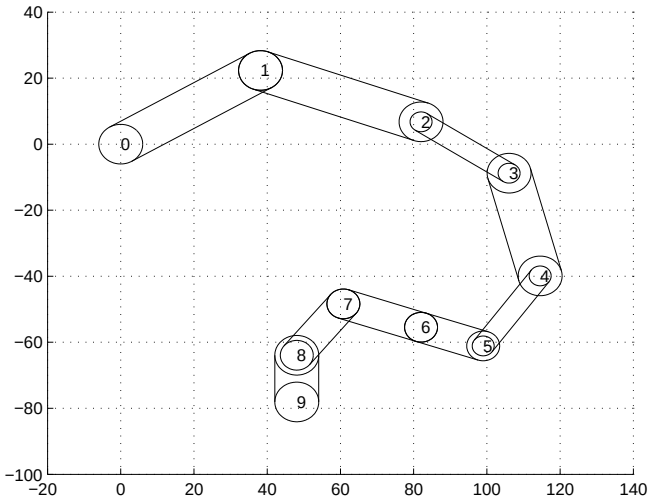

Figure 2.5: Example of a RDDF corridor plotted with Matlab.

course, probably confused by the long shadows on the road, and kept traveling off-road, avoiding bushes along the way. After getting back on the road, he decided to crash through a barbed wire fence at approximately the 1.0 mile point. From there, he drove off-road until 1.3 miles, at which point he again tried to break through the barbed wire fence. This time, the barbed wire won and Bob sat for over 40 minutes revving his engine until DARPA disabled the vehicle though their E-stop device.

Figure 2.6: Example of a tank trap obstacle. It is a simple construction of welded iron girders that efficiently makes any vehicle come to a sudden halt. ©Entertainment Earth, http://www.entertainmentearth.com

Chapter 3

Methods and Technology

3.1 Material provided

Team Caltech has a budget that involves a lot of money and sponsors. Several companies donated hardware and Caltech sponsors with available students that take courses available that relate to the Team Caltech race vehicle.

3.1.1 Computing Hardware

For software development standard PCs with various Linux distributions were used. These PCs were a part of the CDS[1] network. Team Caltech also have eight IBM laptops that are used for field testing.
Inside Bob, the race vehicle, eight IBM Pentium4 desktops with 3.0GHz CPUs, linked with gigabit network, were used to run the different modules.

3.1.2 Vehicle platform

Bob is a used Chevy Tahoe from 1996 that was purchased. It has four wheel drive and has custom suspension to couple with rough off-road terrain. Surrounding Bob a custom iron roll-cage was built that also acts as a mount platform for the sensors on the roof. To cool the computers inside, an external air condition supply had to be mounted

[1] Control and Dynamical Systems Department at California Institute of Technology

on the roof (the race takes place in hot desert environment). The large number of computers and the external air condition supply required so much power that an extra electricity generator had to be mounted in the vehicle. To ensure constant power to the equipment, two UPS power supplies are used. For emergency stopping the vehicle, red push buttons are placed all around the vehicle, that will kill the throttle if pushed.

3.1.3 Sensors

To create such an autonomous vehicle, a lot of different sensors need to be used to sense the environment. Examples of sensors can be radar, cameras, laser distance estimators, gyros and GPS receivers. The following sensors are the ones that Team Caltech used for the 2004 race:

- LADAR[2]: a laser-radar that uses a spinning mirror to scan a laser beam that reads the distance to the objects it hits. Used to detect obstacles. Team Caltech has two units but only one was used during the race.
- Stereovision cameras: two cameras that when combined give depth information about the images they capture. Two pairs of these cameras were created but only one pair was used during the race.
- Road following camera: A camera that was going to be used for a road-detecting algorithm to follow roads. The implementation wasn't ready enough to run at the race, so it was not used.
- Differential GPS[3] receiver: to keep track of the current position on the map.
- IMU[4]: a device that uses accelerometers and a gyro to measure the orientation.

3.1.4 Software

Many different applications and packages has been used in the project. As common in the university world, free and open-source software has been used as much as possible. Table 3.1 lists the most important ones and two project specific applications are described in detail below.

[2] Laser Detection And Ranging
[3] Global Positioning System
[4] Inertial Measurement Unit

Name	*Description*	*Open-source?*	*Availability*	*URL*
Boost	Free C++ libraries	yes	free	http://www.boost.org
CxxTest	Unit test library	yes	free	http://cxxtest.sf.net
Dia	Diagram editor	yes	free	http://gnome.org/projects/dia
Doxygen	Documentation generation	yes	free	http://www.doxygen.org
Eclipse	Java IDE	yes	free	http://www.eclipse.org
Gazebo	3D simulation engine	yes	free	http://playerstage.sf.net
GNU GCC/G++	C/C++ compiler	yes	free	http://gcc.gnu.org
GNU Profiler (gprof)	Code profiler	yes	free	http://gnu.org
ImageMagick	Image manipulation	yes	free	http://www.imagemagick.org
KDevelop	The KDE IDE tool	yes	free	http://www.kdevelop.org
Linux	Reliable operating system	yes	free	http://www.linux.org
Matlab	Numerical math tool	no	no	http://www.mathworks.com
MJPEG Tools	Toolkit for movie creation	yes	free	http://mjpegtools.sf.net
MTA	Real-time process communication	no	Caltech internal	See [9]
Player	Interface server to Gazebo	yes	free	http://playerstage.sf.net
Sparrow	Text display library	no	Caltech internal	See [12]
Sun Java SDK 1.4.2	Java SDK	no	free	http://java.sun.com
Umbrello	UML-drawing tool	yes	free	http://uml.sf.net

Table 3.1: Software that has been used in the project development.

MTA

MTA is a *messaging transport architecture* that was written by Isaac Gremmer at Caltech [9]. The purpose of it was to make it easy for the module developers to have a way to communicate with other modules over the network, without the need to manually open and close sockets etc. MTA also supports starting and shutting down modules through the messaging system.

Sparrow

Sparrow is a real-time text display library written by Richard Murray at Caltech. It's written in C and is very compact and fast, allowing user input and displaying variables with low CPU usage. For a screenshot of the Arbiter sparrow display, see figure 4.4 at page 46.

3.1.5 Development tools and methods

The development is performed in Linux operating system environment. The computers in Bob all run Linux and communicate via MTA over Gigabit Ethernet. For programming development mostly editors and the GNU G++ compiler with Make was used. The IDE[5] that ships with KDE, called KDevelop, was also used. For Java development the IDE called Eclipse was used.

3.1.6 Design- and coding standards

No specified standards existed in the DGC project. This was a bad thing since mostly undergraduate students without a lot of project experience developed the code during the year before the 2004 race. That meant a lot of code was unstructured and hardly documented at all. Design documents only existed at the top level of the architecture, and everything below were only source code. This became a real challenge.

[5] Integrated Development Environment, a tool that integrates multiple development tools

3.1.7 Versioning system

CVS[6] was used in the development until this project began. A conversion to use Subversion[7] took place the first weeks. Subversion is a much better versioning system but there were a lot of issues with the Subversion installation that slowed down the development initially.

3.2 Planning architecture

An overview of the planning architecture that was used during the 2004 race can be viewed in figure 3.1. Several additional modules existed, but were never reliable enough to be used. Examples of this are RoadFollower, a module that analyzes color images to detect roads (and then voted to follow them) and StaticMapper, a module that uses pre-recorded data about the environment to feed extra details to the Digital Elevation Map.

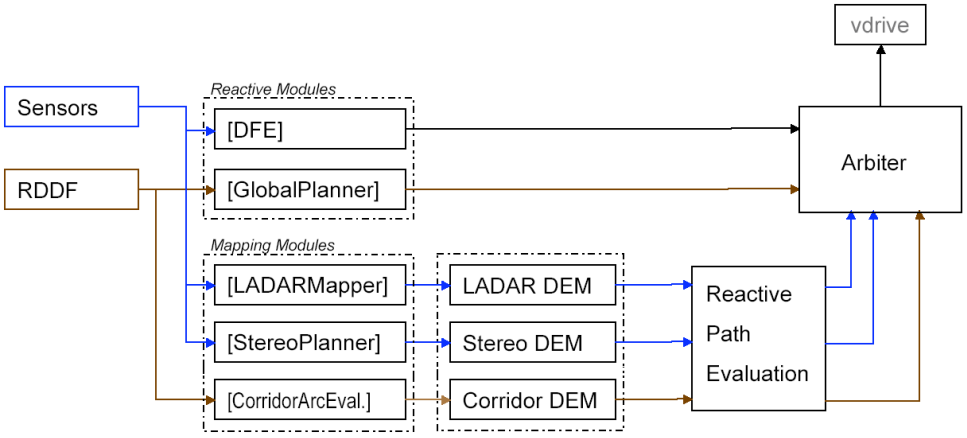

Figure 3.1: The Planning Architecture that was executed during the 2004 race.

[6] Concurrent Versioning System, a free and very common used versioning system

[7] Another versioning system. See http://subversion.tigris.org

The way of information flow in the planning architecture:

- Input:
  - Sensors, like LADAR distance points, Stereovision disparity maps, state and position, delivers data to the modules.
  - The RDDF symbolizes the pre-defined route of waypoints that creates the corridor. This data is also available for the modules.
- Processing:
  - The mapping modules creates DEM[8] maps from their perceived sensor data, which are processed by the Reactive Path Evaluation (which currently is just a conversion function) that creates the final votes of where the module recommends we travel next.
  - The reactive modules instantly creates votes out of their input data. No in-between steps are required, which makes it possible to have a low response time for these modules.
- Decision:
  - The Arbiter collects all the votes, weights the different voters and makes a decision out of the resulting combined vote.
- Execution:
  - The commanded steering and speed from the Arbiter is processed by the controllers in VDrive and the involved actuators are given new reference signals.

3.2.1 Voters

The Arbiter has different *voters* that give it input about the decision to take. In this section, a short description for each voter is given. The voters have different tasks to fulfill, some are meant to steer for goals (the GPS waypoints) and others have as their only purpose to avoid obstacles during the travel to reach the goal.

[8] Digital Elevation Map

GlobalPlanner

This global planning module uses the current position of the vehicle to calculate where to steer to reach the next waypoint in the RDDF file. It only cares about this and always votes maximum goodness for the shortest path to the next waypoint. The other arcs gets lower votes as the distance increases from the best arc. The GlobalPlanner module operates at 10 Hz.

Corridor Arc Evaluator

This module is created to make Bob staying within the RDDF corridor at all costs. It votes very low goodness for the arcs that takes Bob out of the corridor and high for all arcs that keeps Bob within the corridor. During the 2004 race, it was accepted to go outside the corridor to a certain extent, but this rule might be stricter in the next race, making this module important. The Corridor Arc Evaluator module operates at 10 Hz.

LADARPlanner

LADARPlanner uses distance data from a Sick[9] LMS221 [8] LADAR device to detect obstacles in the terrain. For each arc path the area that Bob would occupy if traveling at that path is evaluated and goodness values are lowered for the arcs that have obstacles. The laser readings are performed at 75Hz, where each scan consists of 201 distance values, each related to a different angle. These values combined with the current pitch, roll and yaw angle of the vehicle are used to calculate a DEM map. This DEM map is converted to a cost map from which the arcs are calculated. This calculation is very demanding and is currently running around only 4 Hz.

StereoPlanner

The second obstacle detecting module uses data from two cameras (Sony ICX 084AL) separated in space to create disparity images which can be used to analyze the distance to objects within the camera's vision [4]. This is performed by analyzing small windows of pixels for similarities. Simple trigonometry is then used to locate that point in space. The disparity map combined with the vehicle state at the time of the reading is

[9] Sick Inc. http://www.sick.com

used to calculate a DEM map. This map is converted to a cost map, similar to LADAR-Planner, from which the voted arcs are generated. The overall processing of the stereo vision data is very computationally intensive and is currently able to reach only 7Hz update frequency.

DFE - Dynamic Feasibility Evaluator

In order to avoid rolling over with Bob, a module was created to monitor the speed and vote against dangerous maneuvers. It has a physical dynamics model of the vehicle which it uses together with the current vehicle state to calculate which turns are safe at various speeds. It also takes the pitch, roll and yaw angles into account and prevents for example turning left if driving with a right lean and the opposite, which would result in a roll-over. Since the computer has no sense of what's appropiate maneuvers this module is important to avoid damaging the truck. The DFE currently operates at 10 Hz.

3.2.2 VDrive

The module that controls steering, throttle and brakes actuation is called VDrive (Vehicle Drive). It makes commands in the shape of a desired steering angle and a desired speed. The steering angle is given as a reference value to a controller that handles the steering wheel. The speed is given as reference value to a controller that is closely linked to the cruise controller and the brake actuator.

3.2.3 VState

The most critical thing for Bob is to know where he is. The VState (Vehicle State) module gets position estimation data from a NavCom SF-2050G differential GPS receiver [6] that updates at 10Hz. For orientation and acceleration measurements a high-accuracy Northrop Grumman LN-200 IMU unit [7] , operating at 100Hz, is used. This is a high-end IMU that is normally used in military aircraft and was loaned kindly by Northrop Grumman. The data from these two devices are used as input to a Kalman Filter, that operates at 40Hz, giving state output at a high race. When a module needs to know the state, it asks VState with a MTA message and the state data is delivered with minimum latency.

3.2.4 Matlab

Matlab is used for various tasks during development. It is also integrated into a user interface module called MatlabDisplay, which was built to give the user an overview of the vehicle state within the RDDF corridor and how the voters are voting (see figure 3.2).

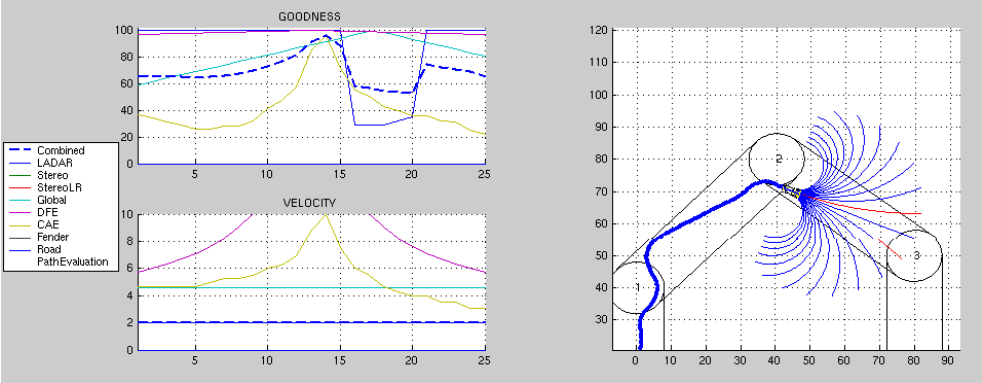

Figure 3.2: The MatlabDisplay module in use. The votes of the different modules are plotted in real-time together with the current vehicle position and the RDDF corridor.

3.2.5 Arbiter decisions

In figure 3.3 a schematic overview of the Arbiter decisions can be seen. The flow of information can be described as:

- Voters cast their votes by sending MTA messages to the Arbiter
- The Arbiter weights the received votes and calculates a resulting combined vote
- From the combined vote, the arbiter selects the best arc
- The desired steering angle and speed are sent as a MTA message to the vehicle controller (VDrive)
- VDrive controls the actuators to reach the commanded steering and speed.

The Arbiter has a few special features about how it handles the votes. For example if many votes next to each other are equal, the middle vote of that sequence is picked as the decided arc. The update rate for the Arbiter is currently 20Hz.

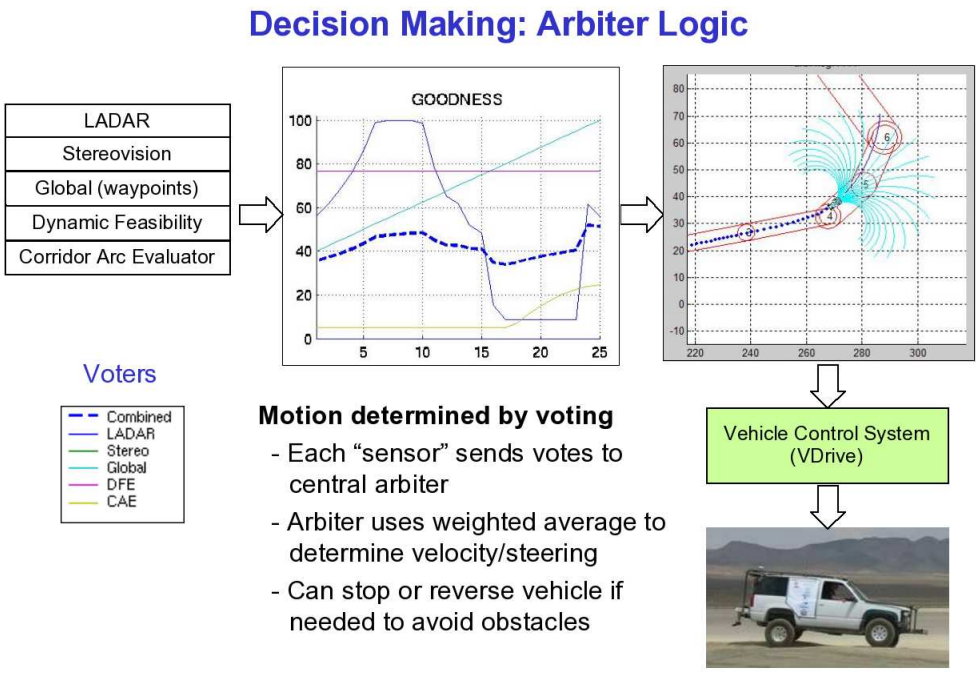

Figure 3.3: Schematic overview of the flow of information that leads to an Arbiter decision.

3.3 Objectives

The following objectives summarizes what was going to be accomplished with the project:

- A working simulation environment using Player/Gazebo
- Scenario reconstruction using logged state and sensor data.
- Redesign of the Arbiter code to improve modularity and object orientation.

- Research learning abilities of the weighting of the voters in the Arbiter.
- Implement situation awareness functionality for the Arbiter.
- Streamline the development process in general for the project.
- Improve the documentation for the project

3.4 Time schedule

This thesis project was divided into two parts. The first ten weeks were a SURF project[10] while the remaining ten weeks were individual work. The whole project was supervised by Richard Murray [10] (Caltech) and Anders Rantzer [11] (Lund Institute of Technology). The SURF project had three milestones:

1. Progress report after four weeks.
2. Progress report and an Abstract after eight weeks.
3. Create a vocal or poster presentation.
4. Final report after SURF had ended.

A poster presenting the project was created and presented (see [16] for download). After SURF ended the final SURF report was filed and project development continued as before but with the thesis report and the presentation as the only targets.

[10] Summer Undergraduate Research Fellowship, a 10 weeks project for students at Caltech, see [14]for more info

Chapter 4

Solution

4.1 Simulation

Before this project, there were no sufficient simulations available for Team Caltech[1]. All real testing relied on field tests, which is time demanding and requires exclusive use of the whole truck for each test. A goal for this thesis was to investigate, evaluate and integrate a simulation environment for use with the current Team Caltech planning software. Discussions had already taken place and various available simulation packages had been reviewed. The choice became Player/Gazebo, mostly because it was free, open-source and created for robot simulation. It also had built-in support for simulating some of the devices that were used, like GPS, cameras and LADAR.

4.1.1 Player/Gazebo

Player is an free, open-source, interface server for robot simulation created by people from the Robotics Research Lab at University of Southern California. It is designed to connect either to real hardware or a software simulation engine, called Gazebo. The planning software is intended to only talk to Player without knowing if it's talking to the robot or the simulation engine. Player provides robot state from the simulation engine and possibilities to send commands to the robot in the simulation.
The real-time simulation engine used is called Gazebo. It's also free, open-source,

[1] An planar kinematic simulator module called SimVStateVDrive existed

and created at USC. Gazebo was originally developed from a less detailed simulation engine called Stage, which was created for simulation of many robots simulataneously. Gazebo is a very capable simulation environment. A summary of important features include:

- Realistic simulation of rigid-body physics: robots can push things around, pick things up and generally interact with the world in a plausible manner. This means there's forces for gravity and friction, collision detection between objects etc.
- Models for commonly used robots.
- Simulation of standard robot sensors, including sonar, scanning laser range-finders (LADAR), GPS and IMU's.

Gazebo, Stage and Player all started out as tools developed for robot simulation and were later on released under the GPL source license, allowing people from many other places of the world to contribute. Now the project is growing faster and faster and is a very capable simulation environment.
A large part of the thesis was spent on getting used to the Player/Gazebo environment and integrating it to the existing planning software of Team Caltech.

4.1.2 Simulation Design

The simulation architecture can be seen in figure 4.1. Currently, Player is not used as it's intended to by the creators, but rather as an interface to the simulation engine. The link between the old planning software and the hardware is still established but will hopefully be migrating over to the true Player-interface in the future, completely isolating the planning software from the underlying layers.

4.1.3 PlayerInterface

A module called PlayerInterface was developed, which works as a wrapper between the commands used in the existing software and the Player protocol for world state and vehicle control. Simulated sensor devices like LADAR, GPS and cameras were already included in Gazebo so connecting the existing software to those was just an interface conversion task. Running the existing planning software in the simulated environment

gave realistic results compared to what had been seen in real world tests with the same software. Those results looks very promising since every student working in the team can now have access to a complete simulation environment to test algorithms and new modules in, without the need of exclusive access to the real vehicle.

Design

For transparency with the existing planning software, PlayerInterface was designed to simulate the two existing modules that interacts with the hardware: VState and VDrive. The new module names became PlayerGetState and PlayerSendCmd. When PlayerInterface is launched the planning modules detects it as VState and VDrive, which means no modifications are required to run with the simulation. PlayerGetState delivers vehicle state from the Gazebo simulation engine on request. PlayerSendCmd sends steering and speed commands to the simulation, when invoked. The class diagram for PlayerInterface can be viewed in figure B.3 at page 62. A sequence diagram showing typical operation can be seen in figure C.3 at page 68.

4.1.4 Gazebo models

Tahoe

A Gazebo plugin model of the Chevy Tahoe vehicle was developed; which had the same length, width, height, mass etc as the real vehicle. Further work needs to be performed to create a complete model with the real dynamic properties like suspension and turnrate of the real vehicle. This will also require a measuring operation on the real vehicle, which will be performed by STI[2]. STI is a company that is working with simulation of cars for major manufacturers and they have a complete measuring facility they've offered to let Team Caltech use. A screenshot of the Tahoe model in action can be seen in figure 4.2.

Sick LMS221

The LADAR device that ships with Gazebo is a model of the Sick LMS220 LADAR device which sends laser beams in a 180 degrees wide scan angle. Team Caltech uses the

[2] Systems Technology, Inc. http://www.systemstech.com

LMS221 model at a 110 degrees scan angle, so some modifications were required to fit the simulated LADAR device to the needs of this project. The simulated SickLMS221 can be seen as the blue rays in action in figure 4.2.

Corridor

When running a simulation, it is crucial from an user interface perspective to know where the RDDF corridor is compared to the environment. A Gazebo plugin was developed that parses the RDDF file and displays it in the simulated 3D world environment. A sample of this Corridor plugin can be seen in figure 4.3.

4.2 Arbiter

The Arbiter was already functional when this thesis project started. It had some advanced features and worked as intended with a few exceptions:

- All the weights were 1.0 for the voter modules
- It gave a stop signal with zero steering as soon as an obstacle sensor gave zero votes (which happened for example when the LADAR device got too much vibrations and had to reset itself)
- The code was very hard to maintain and not very well documented.

Like much of the other code for Bob, it had been written in a hurry by only one person that had full understanding of the code. However, this person was not present in the project anymore. An important decision made was to leave the external API to other modules untouched for backward compatibility with existing modules. A re-design into object oriented classes were made and a *branch* was created where the new development took place while the old, proved working version, was left intact.

4.2.1 Design

The first thing to do with the Arbiter design was to create classes. The old code was written mainly in two large (600 rows each) files with not much object orientation at all. To be able to reuse the code in the future, but also to be able to perform unit testing,

an object oriented design was created. The resulting class diagram can be viewed in figure B.1 at page 60. A sequence diagram showing a typical situation of the Arbiter operating can be seen in figure C.1 at page 66.

4.2.2 Improvements

In addition to the redesign of the Arbiter, new features were added.

Weight adjusting

One feature was support for changing the voter weights in real-time. The purpose of this was to be able to weight some voters higher than others to affect the decisions in an intelligent way for the Arbiter. This also included support for reading and writing configuration files for the weights.

Modes

To be able to implement situation awareness different modes were needed. This was achieved by adding flags for situations and conditions that would be able to be detected by other modules. Examples of these were:

- Terrain: rough or light terrain could affect how we prioritize decisions. For example we want to limit the maximum possible speed when we're driving in rough terrain.
- Lighting: bad lighting conditions is when we do not want to trust stereo vision cameras since they have a hard time to gather depth data in bad lightning or when the sun is shining into the camera lenses. In this case, the weight for the StereoPlanner module is lowered.
- Dustiness: if there's a dust cloud in front of the LADAR device, it gives invalid readings about obstacles that are not real. Then we don't want to trust the LADAR, meaning we lower the weights for the LADARPlanner module.

Currently, this is only supported in the Arbiter – other modules does not yet detect and signal these conditions, but as soon as that's available, the Arbiter will have this mode management.

Steering smoothing

Steering command smoothing was another feature implemented. It works as if the commanded steering is almost straight forward (currently the middle 3 arcs); it suppresses quick changes from left and right by averaging the output steering command. This avoids zigzagging when going forward which has been a problem since only 25 arcs are used for vote generation. One might think increasing the number of arcs would be a better solution but that would increase the computation time used for each module that calculates votes.

GeneticAlgorithm

An idea for managing the weights was to use logged data from a human driving Bob, with the sensor modules running. Analysis of the decisions taken by the human driver using a genetic algorithm is able to produce weight-sets that make it possible for the Arbiter to imitate the human's decisions in the situations the human faced. By that, good weight sets can be developed for different situations with this tool.
A class diagram of the Genetic Algorithm application can be seen in figure B.2 at page 61 and a sequence diagram showing a typical iteration can be seen in figure C.2 at page 67.

Unit Testing

The new object oriented design of the Arbiter made it possible to perform *unit testing*. A free unit test package called CxxTest[3] was used for these tests. With unit tests, the specifications of the important parts of the Arbiter were tested and verified working correctly. This test suite will also make a solid ground for future *regression testing* on the Arbiter when further modifications have been made.

4.3 LogPlayer

In addition to the lack of simulation capabilities, there was no way to reconstruct logged runs of Bob. To be able to analyze what decisions the Arbiter makes in each time segment it is crucial to have a playback utility where you can step and pause the playback,

[3] http://cxxtest.sf.net

giving time to analyze the situation for debugging purposes. To achieve this, a module called LogPlayer was developed. It allows playback from voter and state logs just as all the modules were actually running. Different playback modes includes timedriven playback and a user mode when the user selects when votes and state shall be sent. An example of the user interface can be seen in figure 4.5.

4.3.1 Design

The LogPlayer was designed to be transparent to the existing Arbiter. Two MTA modules were created: VStatePlayback and VoterPlayback. VStatePlayback parses the state log and responds to VDrive state requests just as it was the VState module. VoterPlayback parses all the voter logs and sends votes just as they came from running voting modules. A class diagram of the LogPlayer can be seen in figure B.4 at page 63 and a sequence diagram showing a typical playback step can be seen in figure C.4 at page 69.

4.4 Field tests

For trying out new software implementations on Bob, a large area of space is required if Bob is going to be able to travel safely around autonomously. This requirement combined with the size of Bob and the clumsy ways to transport it (requires a trailer since Bob cannot be driven on the streets manually) combined with the distance (a 2 hours drive) to suitable test areas, makes quick tests impossible to perform. Because of this, only two field tests were performed during the summer. They took place at a dry lake bed in El Mirage[4] and included staying there for two days of testing. It is quite a challenge to organize such a field test as it includes a lot of computers, having a site server out in the desert and all other logistic issues involved with having fifteen students camping in the desert. Many people were sharing Bob as a test platform and inefficiency in the handling of modules, logs and test setups made testing very time demanding. It was a great time though and field testing in large scale will probably be the key to success in the 2005 race.

[4] An off-road racing area about 60 miles north-east of Caltech in Pasadena, CA. For further info, see: http://www.ca.blm.gov/barstow/mirage.html

4.5 System Administration

A lot of system administration tasks were required to improve the efficiency of the project. The thesis author quickly took a role as one of the system administrators, handling various tasks.

4.5.1 Documentation

Doxygen

Even though the project library consisted of 50000 lines of source code, no overall documentation API existed. For this some research was performed to find a suitable documentation tool and a tool called Doxygen was chosen. Doxygen parses code source files for specific syntax that tells the parser which text means what in the documentation (like javadoc[5] does for Java source code). Doxygen is very powerful and also has features like creating class diagrams and *call graphs*. An example of a Doxygen generated HTML-page can be seen in figure 4.6.

Wiki

To gather all the documentation about tools, hardware and software projects, meetings and contact lists in one place, a web based documentation system called Wiki was used. Wiki is an easy way for users to create and edit web pages without any HTML knowledge required. It was a big success for the project, as for the first time all information was available at the Wiki webpage and constantly updated by users. Also all old documentation was entered into the Wiki. This thesis project administrated some of the Wiki pages and also wrote many installation guides about how to compile the Player/Gazebo, how to use different tools, code standards and many other things.

4.5.2 Coding Standards

As mentioned earlier (see 3.1.6), there existed no standards about how the source code should be written, formatted and documented. The coding standards were based on standard GNU Linux coding standards[6]. Guidelines about Doxygen documentation

[5] Sun's tool for creating the Java documentation

[6] For more info, see http://www.gnu.org/prep/standards/

formatting was also included in these standards to make it possible to view all the documentation for the different sources online in the Doxygen HTML-documentation.

4.5.3 Code Profiling

When building computationally intensive applications like the modules in the planning architecture, *code profiling* is an important tool for identifying performance bottlenecks in the code. For profiling the GNU profiler, called gprof, was used. One limitation with gprof is that it cannot normally profile multi-threaded programs, which all the MTA modules are. This limitation was solved with a patch created by Samuel Hocevar[7].

4.5.4 Versioning System

For version management, the common tool called CVS was used. In the beginning of the thesis a merge to Subversion was made, with a lot of issues on the administration side. The problems were solved and soon everyone could take advantage of the ease of use with Subversion and the many improvements it has over CVS. Some of the main advantages with Subversion compared to CVS are:

- It is backwards compatible with CVS and supports importing all the old versioning logs
- It uses a relative database to track changes in the files, which is very fast
- It only saves the changes between each revision instead of making a copy of a file every time it's edited. This saves a lot of disk space.
- Many clumsy operations from CVS like renaming and moving of files and directories are now easily performed
- Since all commands in CVS exists in Subversion, it's very easy to switch for users used to CVS.

[7] See http://sam.zoy.org/writings/programming/gprof.html for more info

4.5.5 Bug reporting

To keep track of bugs in the project, the widely known bug tracking system Bugzilla[8] was used. Bugzilla was originally created for tracking bugs in the open-source project of the Mozilla web browser but turned out to be so successful that it was released as an independent, free bugtracking system. It's a very capable tool for keeping track of current and past bugs and tasks that needs to be performed. It also has support for showing dependency trees between different bugs. Every time a new bug is submitted or modified, an e-mail is sent to the bug owner and all affected developers. The bug owner can then reassign the bug to other developers or accept it as his own responsibility to fix.

4.5.6 Hardware Maintenance

The computers used in Bob for the 2004 race were working fine but required a lot of space in the truck. For this reason, thin rack mounted servers has been purchased for the new platform that will be used in the 2005 race. The selection and maintenance of these and the old servers was one of the administrative tasks performed in this thesis project. Another was reinstallation of the field laptops used by the team to achieve a homogenous setup of workstations for the developers. The selected Linux distribution became Debian[9] since it is widely supported and has excellent package management. From the 2004 race, one important lesson learned was how important a good package management support was, since many of the tools that are used, depend on a lot of different Linux packages.

4.5.7 Scripting for automation

Many tasks in the project were inefficient and required the same work being performed on multiple computers. Most of them were automated by *bash*-scripts. Examples of such tasks are:

- Updating the subversion tree and compiling on multiple computers.
- Automated movie creation with time synchronization from logged images from the stereo vision cameras.

[8] See http://www.bugzilla.org
[9] See http://www.debian.org

- Conversion of RDDFs to Bob's native format and uploading to multiple computers before used in a test run.
- Creation and exchange of SSH-keys between multiple computers to enable automatic login which makes it possible for other scripts to perform operations on multiple computers without user interaction required.
- Flexible automatic setup of MTA settings between multiple computers since MTA requires config files with the IP adresses of all the computers that shall be able to talk to each other, on each of the involved computers.
- Collecting various log files from multiple computers to a single storage host. This was created when realized how much time that was wasted on this during the field tests.

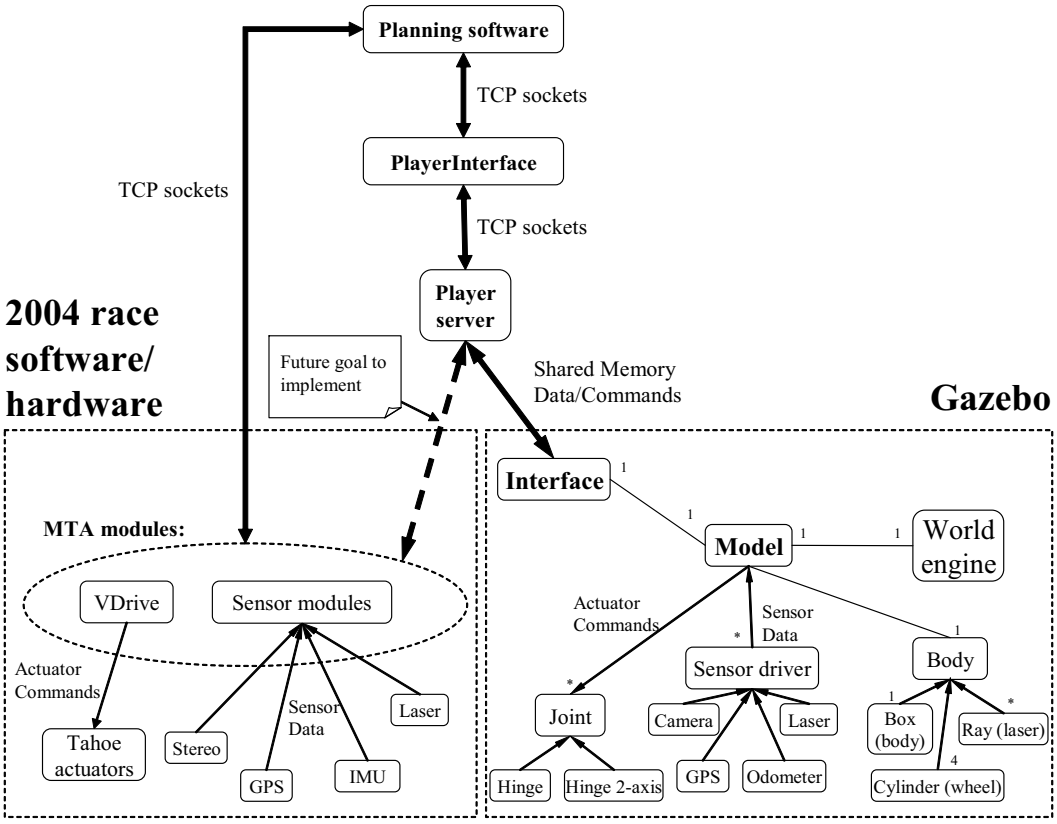

Figure 4.1: The Simulation Architecture. Note that the current implementation is not transparently connected to Player as the purpose of Player/Gazebo is. This will be a future implementation goal.

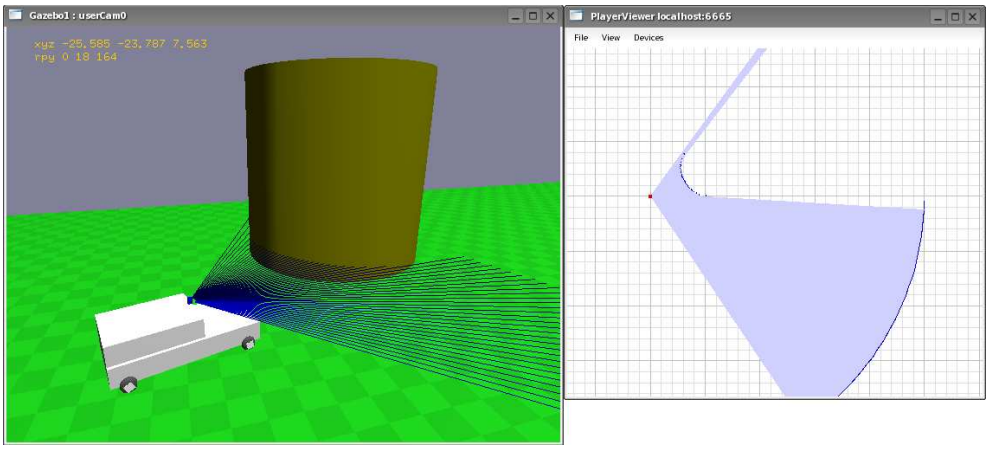

Figure 4.2: Left: The Gazebo simulator with a Sick LMS221 device mounted on the Tahoe model. Right: A client application that displays the current LADAR readings and allows interaction with the model.

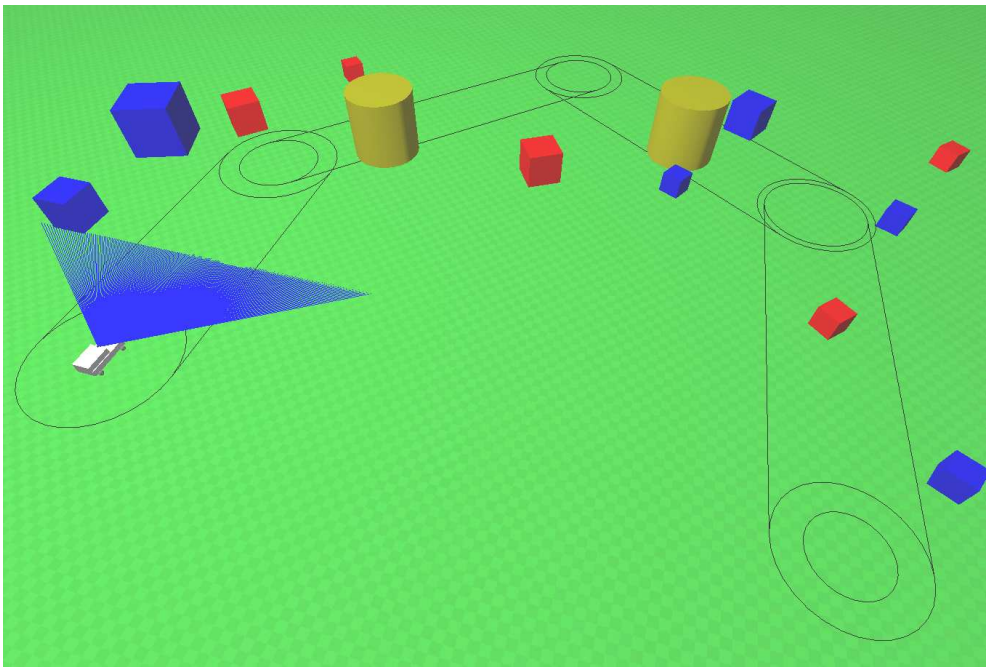

Figure 4.3: The Corridor plugin drawing the RDDF corridor in the 3D simulated environment.

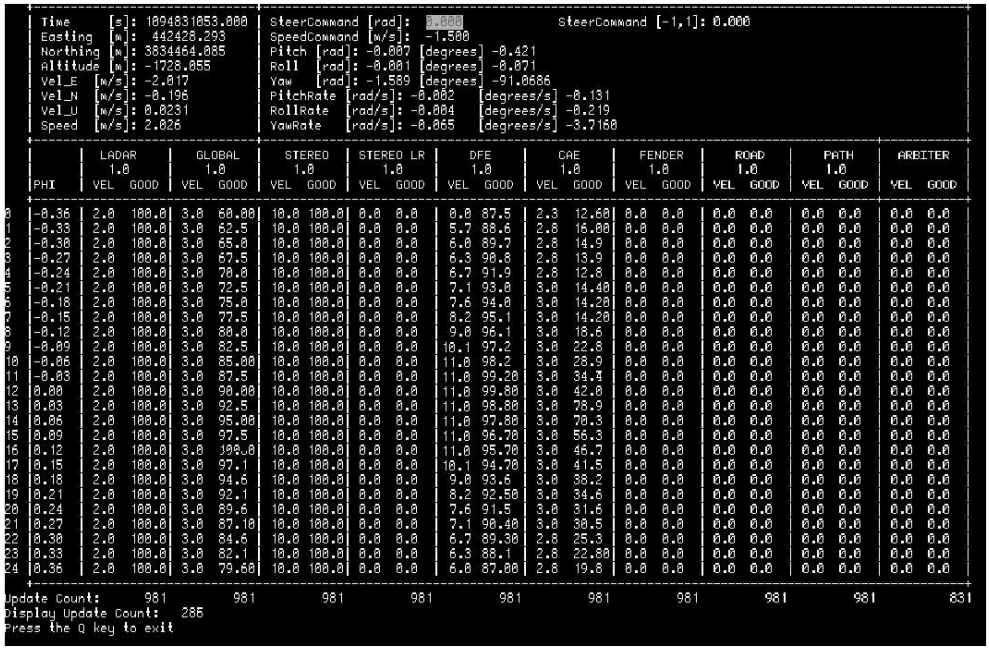

Figure 4.4: The Arbiter sparrow display. Votes with goodness and speed values can be seen for the 25 arcs for each module. Also all the vehicle state information is displayed.

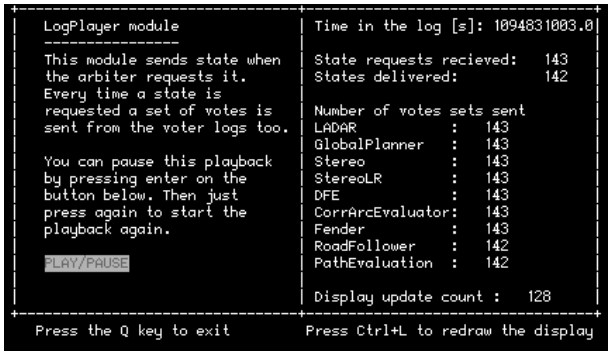

Figure 4.5: The LogPlayer interface. Playback of logs can be paused and played. Delivery speed can be adjusted and sending of votes can be stepped forward and backward.

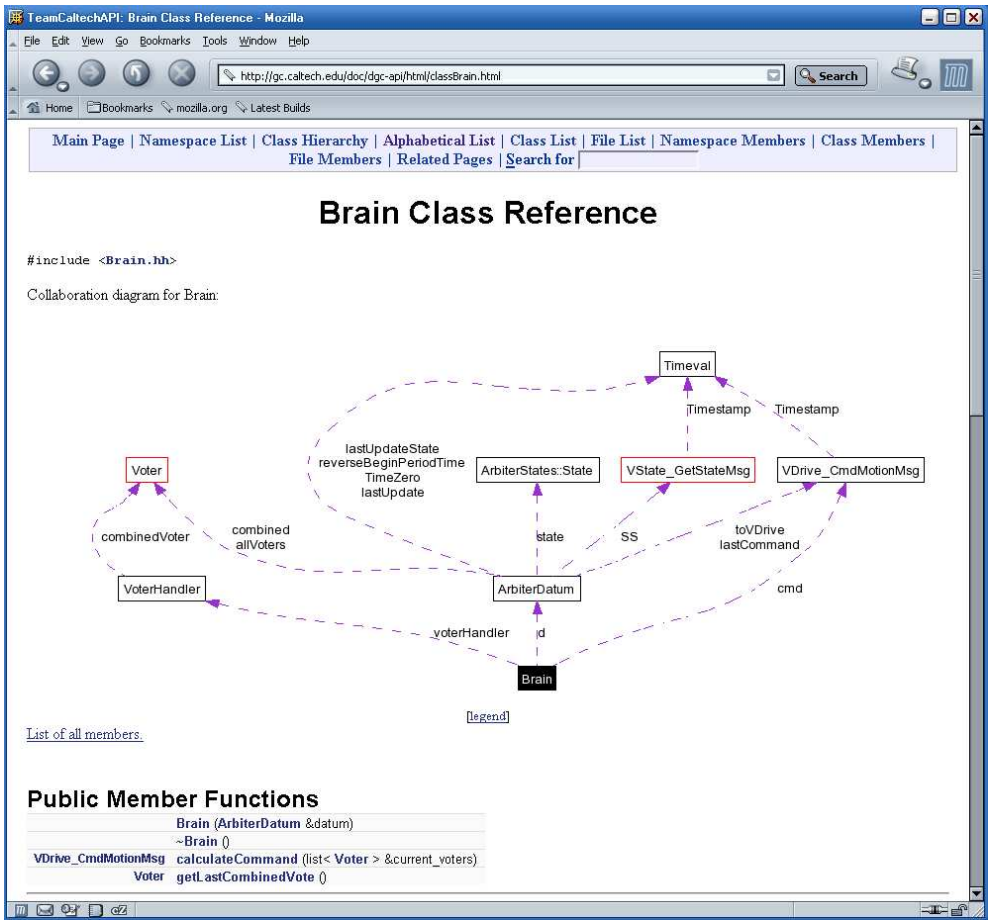

Figure 4.6: Example of a documentation web page generated by Doxygen. All class boxes and methods are links to their respective documentation pages. The class diagram is of a compact format but UML class diagrams can also be generated.

Chapter 5

Conclusion

5.1 Retrospective

From the beginning, this project was focused on remaking the Arbiter to something more intelligent and advanced. There are two main reasons for why most of the effort was spent on other objectives: the lack of good simulation possibilities and the lack of tools for reconstructing logged scenarios. It's almost impossible to test advanced algorithms and behaviors without those key requirements. As mentioned earlier (see 4.4 at page 39), only two field tests were performed during the thesis project. Unfortunate circumstances in sensor modules of the software rendered the collected obstacle sensor data useless for applying the Genetic Algorithm, which lead to that the theories about it couldn't be proved[1].

[1] although the base program it was built on has been proved working since it was used in the "Applied Artificial Intelligence" class at LTH spring 2003. So good results should be expected as soon proper test data is available.

5.2 Conclusion

What was accomplished with this master thesis? The Arbiter was improved, not as revolutionary as first proposed, but a number of improvements has been performed:

- Object oriented design
- Well documented classes
- Unit tested classes
- Support for mode management (still requires work on other modules)
- Adjustable weights in real-time
- Highly improved state logging with a new reusable global logger class
- Rewritten implementation of features handling decisions
- Smoothing of steering command when going straight.

Outside the Arbiter a collection of tools has been developed:

- Tool for generating Arbiter weight sets imitating human decisions (the genetic algorithm application)
- Simulation environment integrated with Player/Gazebo, capable of simulating all the involved decision making modules transparent[2] to simulation/hardware.
- A collection of scripts that simplifies monotonous tasks and increases the efficiency of development, administration, log handling and movie creation.

5.3 Future work

Team Caltech has the largest challenge still coming. The first race was a huge learning experience but also a project developed under high time pressure. A lot of experience was gained by the design mistakes in both software and hardware. Currently, a new

[2] not completely transparent until all modules communicate with the Player server instead of with VState/VDrive over MTA. Right now you must give a special flag when running the modules for them to adjust to the Player interface

vehicle is being built and a new planning architecture is being developed. The new architecture will use a deliberative planner that outputs paths instead of just steering angles. This will make it possible to do much more sophisticated planning.

5.3.1 In general

- Improve the documentation for the project.
- Redesign more of the DGC code for object orientation to isolate dependencies and problems between modules. Object orientation also has the advantages of an easier way to present the hierarchy graphically and the possibility to perform unit tests on objects.
- Decide on the new planning architecture, including a deliberative planner, path following and more.

5.3.2 Simulation

- To create a complete simulation environment that is transparent to the planning software (i.e. uses the Player server as it's meant to be used, with all modules talking to Player only).
- To make a detailed analysis of the race vehicle and create a dynamically correct model of it in Gazebo, with suspension dynamics etc. This will be possible after measuring the new vehicle at STI.
- Integrate stereo vision image processing with image data delivered from Player/Gazebo.

5.3.3 Arbiter

- Implement inputs to situation awareness so it can use the mode management.
- Run long and detailed tests with a human driving, logging data to be executed with the Genetic Algorithm.

5.3.4 LogPlayer

- Synchronize with a (not yet fully developed) user interface application so graphs, maps and camera images are displayed during playback.

5.3.5 New race vehicle

For the next race in 2005, Team Caltech will build a completely new vehicle. The major mistakes with Bob can be summarized to:

- Desktop computers took up too much space (but was chosen because they were donated), resulting in room for only one person, the safety driver
- An external generator had to be purchased to supply enough power. It makes a lot of noise and takes up space for one person.
- An external air conditioner had to be purchased to cool the computers. This adds complexity and weight on the roof.
- Basically all actuation of the vehicle was custom made for the vehicle, making it hard to drive manually to/from test sites, requiring a trailer to tow Bob with.

Currently, Team Caltech has closed a deal with a major sponsor called Sportsmobile[3]. Sportsmobile is a company that converts vans to four wheel drive and off-road terrain capabilities. A Ford van will be converted to have four wheel drive, custom suspension, a 6 litre diesel engine, 110V uninterruptable power builtin (no need for UPS), powerful retail air condition and much more. This will take away all the mistakes made during the building of Bob, and allow 4 persons to ride in the vehicle during testing. Rack mounted servers will be used instead of laptops, improving cabling issues and reducing space required. For actuation, a retail handicap specialized actuation system will be used, allowing easy switching to manual driving and a reliable and well documented interface to the lower levels of control of the actuators.
This new vehicle, combined with the new planning architecture, looks very promising for the next DARPA Grand Challenge race, which will have much tougher competition. Time will tell if Team Caltech succeeds with their quest!

[3]For more info, see http://www.sportsmobile.com

Bibliography

[1] Stentz, A., et al, "A Complete Navigation System for Goal Acquisition in Unknown Environments" 1995.
http://www.ri.cmu.edu/pub_files/pub1/
stentz_anthony__tony__1995_2/stentz_anthony__tony__1995_2.pdf

[2] DARPA, http://www.darpa.mil

[3] M. Juberts, K. Murphy, M. Nashman, H. Scheiderman, H. Scott, S. Szabo, "Development And Test Results for a Vision-Based Approach to AVCS", 1993.
http://www.isd.mel.nist.gov/documents/murphy/isata93.pdf

[4] Matthies, L., et al, "Stereo Vision and Rover Navigation Software for Planetary Exploration" 2002.
http://robotics.jpl.nasa.gov/people/mwm/visnavsw/aero.pdf

[5] DARPA Grand Challenge, http://www.darpa.mil/grandchallenge

[6] NavCom SF-2050G Differential GPS receiver.
http://www.navcomtech.com/products/sf2050g.cfm

[7] Northrop Grumman LN-200, an high-accurate airplane IMU
http://nsd.es.northropgrumman.com/Html/LN-200S/

[8] Sick LMS221 LADAR device.
http://team.caltech.edu/members/SICK/LMS%20220-221%20Tech%20Info.pdf

[9] MTA, developed by Isaac Gremmer at Caltech, 2003
http://gc.caltech.edu/project/2004/doc/EmbeddedSystems/MTA/MTA Users Manual.pdf

[10] Prof. Richard M Murray, Professor of Control and Dynamic Systems, Caltech, USA.
http://www.cds.caltech.edu/~murray

[11] Prof. Anders Rantzer, Professor of Automatic Control at Lund Institute of Technology, Sweden.
http://www.control.lth.se/~rantzer

[12] Sparrow, developed by Richard M. Murray at Caltech
http://www.cds.caltech.edu/~murray/software/1995a_sparrow.html

[13] Sue Ann Hong, "Arbiter readme and specification", 2003.
http://gc.caltech.edu/project/2004/doc/Planning/Arbiter/README.txt
http://gc.caltech.edu/project/2004/doc/Planning/Arbiter/ARBITER_SPEC.txt

[14] SURF, summer 2004. http://surf.caltech.edu

[15] Team Caltech, Caltech's team for the DARPA Grand Challenge.
http://team.caltech.edu

[16] The thesis' webpage. For further info and downloads.
http://exton.se/cv/thesis

Appendix A

Definition of Words

A.1 Terminology

Table A.1 and A.2 explains terminology used in this report.

A.2 Abbreviations

Table A.3 explains the abbreviations used in this report.

Term	*Explanation*
Actuator	An actuator is the mechanism by which an agent acts upon an environment. The agent can be either an artificial intelligence agent or any other autonomous being (human, other animal, etc).
Autonomous vehicle	A vehicle that can drive completely by itself, with no help of a human through remote control etc.
Bash	Bash stands for Bourne again shell and is the default command language interpretor (shell) for the Linux operating system. It is very suitable for scripting commands and operations.
Behaviour	In the subject of planning, an action-producing module is called a *behaviour*.
Branch	When an isolated state of the code repository is frozen and development goes on from that point, without being affected by changes in other parts of the tree.
Call graph	A graph that shows which methods a class calls during execution.
Code profiling	When you use a tool to probe a running application to measures the time spent in different parts of a program, to identify bottlenecks.
Command fusion	Combining different commands into one resulting command.
Deliberative planning	To plan for long-term goals, looking forward into what will happen later instead of just facing the current situation.
Differential GPS	A GPS system that, in addition to the satellite signals, uses signals from ground based GPS-towers to correct the errors from the satellites. This gives very high accuracy in position estimation, at best the error is as small as a couple of decimeters.
Messaging transport architecture	A way of hiding the underlying network layers for the programmer. Given an API to the architecture it's easier to write code that uses the network to send/receive messages.

Table A.1: Terminology used in this report

Term	*Explanation*
RDDF waypoint	A waypoint in the RDDF file. Consists of a Easing and a Northing coordinate, a maximum radius of distance and a maximum allowed speed. Multiple waypoints form a corridor (see figure 2.5 at page 19).
RDDF corridor	Multiple waypoints form a corridor. From each waypoint's outer limit a corridor is reaching to the same radius out of the next waypoint (see figure 2.5 at page 19).
Reactive planning	Planning for a short-term perspective, like avoiding obstacles that are close.
Regression testing	Tests that are executed after modifications has been made to code, to verify that the code is still running as proposed.
Unit Testing	A way of testing isolated parts (usually classes) of an application, to verify it's working according to the specifications.
Vote	A data set of 25 arcs where each arc has a goodness and a speed assigned to it.

Table A.2: Terminology used in this report

Abbreviation	*Full words*	*Description*
CAE	CorridorArcEvaluator	See 3.2.1 at page 27.
Caltech	California Institute of Technolgoy	See http://www.caltech.edu
CVS	Concurrent Versioning System	Version management system
DARPA	Defense Advanced Research Projects Agency	Grand Challenge organizers
DAMN	Distributed Architecture for Mobile Navigation	A steering arbiter architecture
DEM	Digital Elevation Map	Grid-based terrain representation
DFE	Dynamic Feasibility Evaluator	See 3.2.1 at page 28.
DGC	DARPA Grand Challenge	The name of the race for autonomous vehicles.
IDE	Integrated Development Environment	A program that integrates many different tools for software development
IMU	Inertial Measurement Unit	Device to measure accelerations and angular rates
MTA	Message Transport Architecture	Framework that provides messaging between modules.
NTG	Nonlinear Trajectory Generation	A library for real-time trajectory generation.
QID	Qualification, Inspection and Demonstration	Qualification event for vehicles that took place the week before the 2004 GC
RDDF	Route Data Definition File	File that contains the GPS waypoints used in the race.
SDK	Software Development Kit	All you need to develop in a language.
SSH	Secure Shell	The standard secure login to remote Linux hosts.
SURF	Summer Undergraduate Research Fellowship	A summer project for research at Caltech.

Table A.3: Abbreviations used in this report

Appendix B

Class diagrams

B.1 Arbiter

Figure B.1 shows the Arbiter's class diagram.

B.2 GeneticAlgorithm

Figure B.2 shows the Genetic Algorithm application's class diagram.

B.3 PlayerInterface

Figure B.3 shows the PlayerInterface module's class diagram.

B.4 LogPlayer

Figure B.4 shows the LogPlayer module's class diagram.

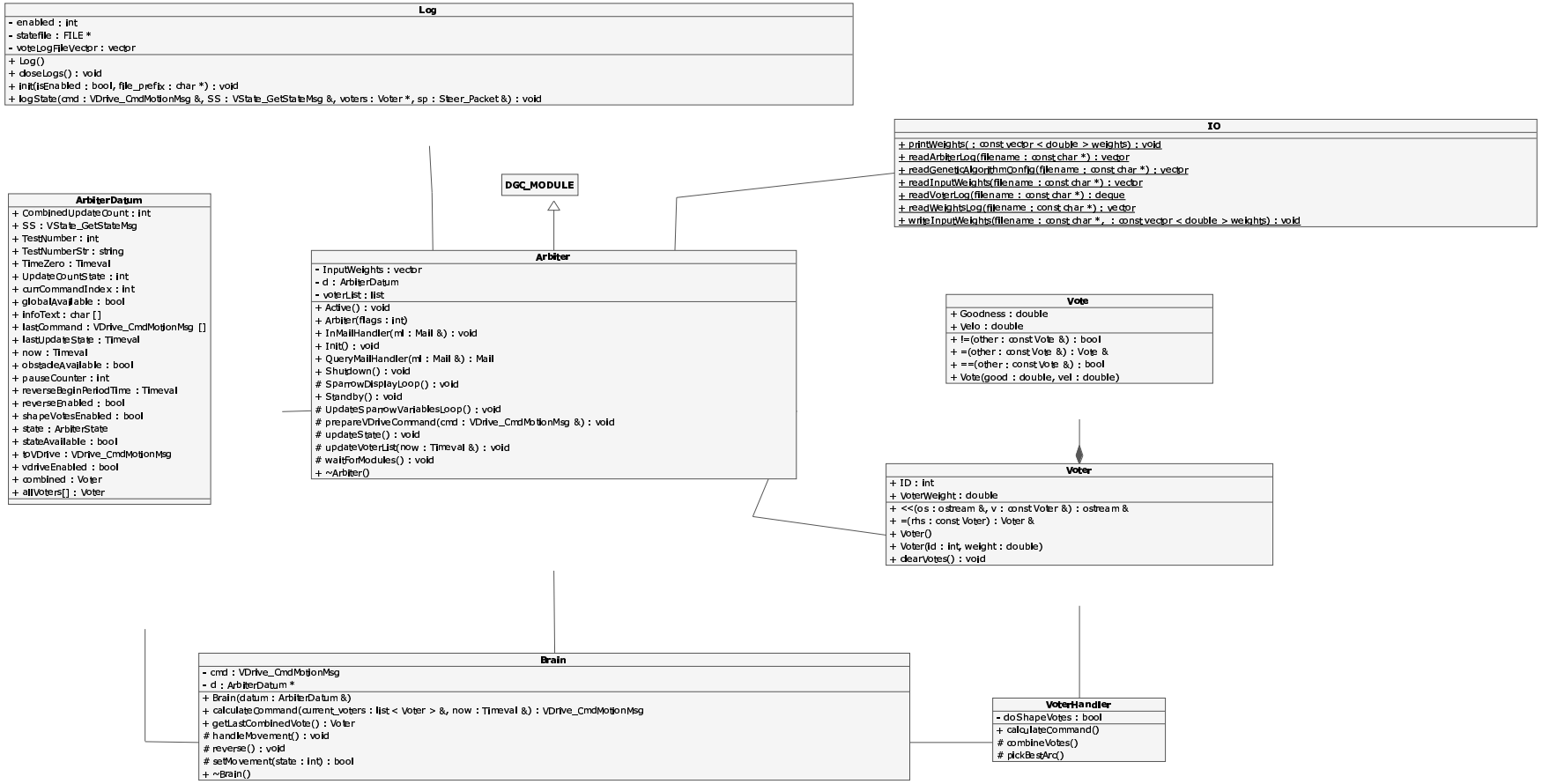

Figure B.1: Arbiter class diagram.

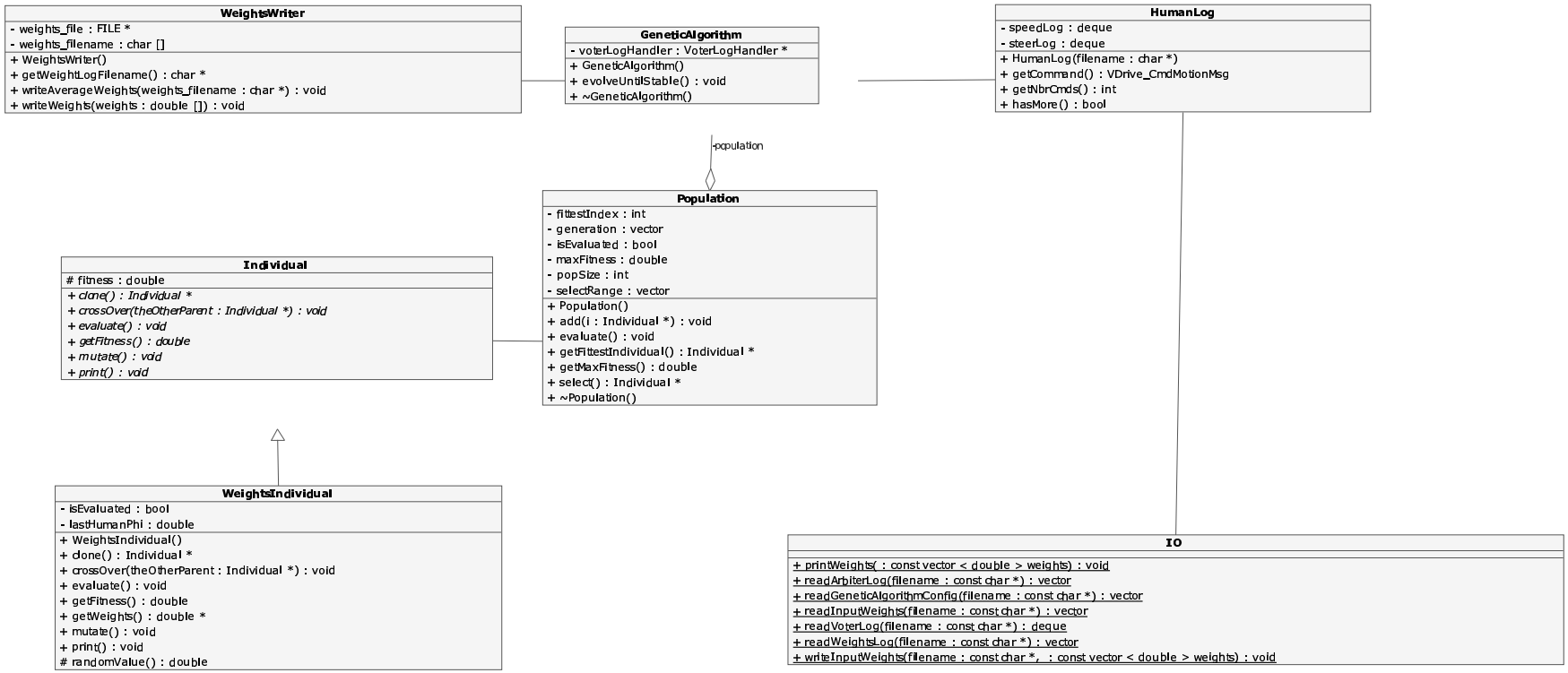

Figure B.2: The GeneticAlgorithm application class diagram.

Figure B.3: PlayerInterface module class diagram.

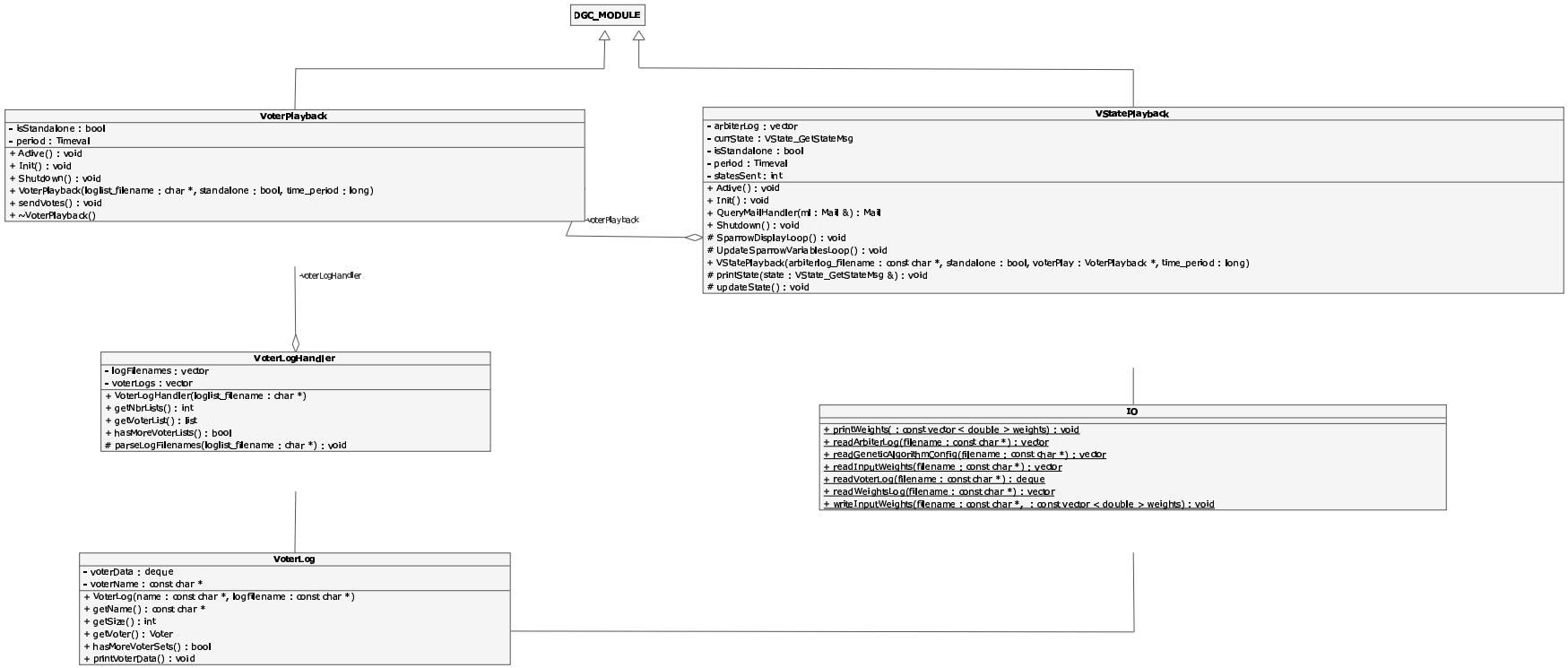

Figure B.4: LogPlayer class diagram.

Appendix C

Sequence Diagrams

C.1 Arbiter

Figure C.1 shows a sequence diagram for the Arbiter.

C.2 GeneticAlgorithm

Figure C.2 shows a sequence diagram for the Genetic Algorithm application.

C.3 PlayerInterface

Figure C.3 shows a sequence diagram for the PlayerInterface module.

C.4 LogPlayer

Figure C.4 shows a sequence diagram for the LogPlayer module.

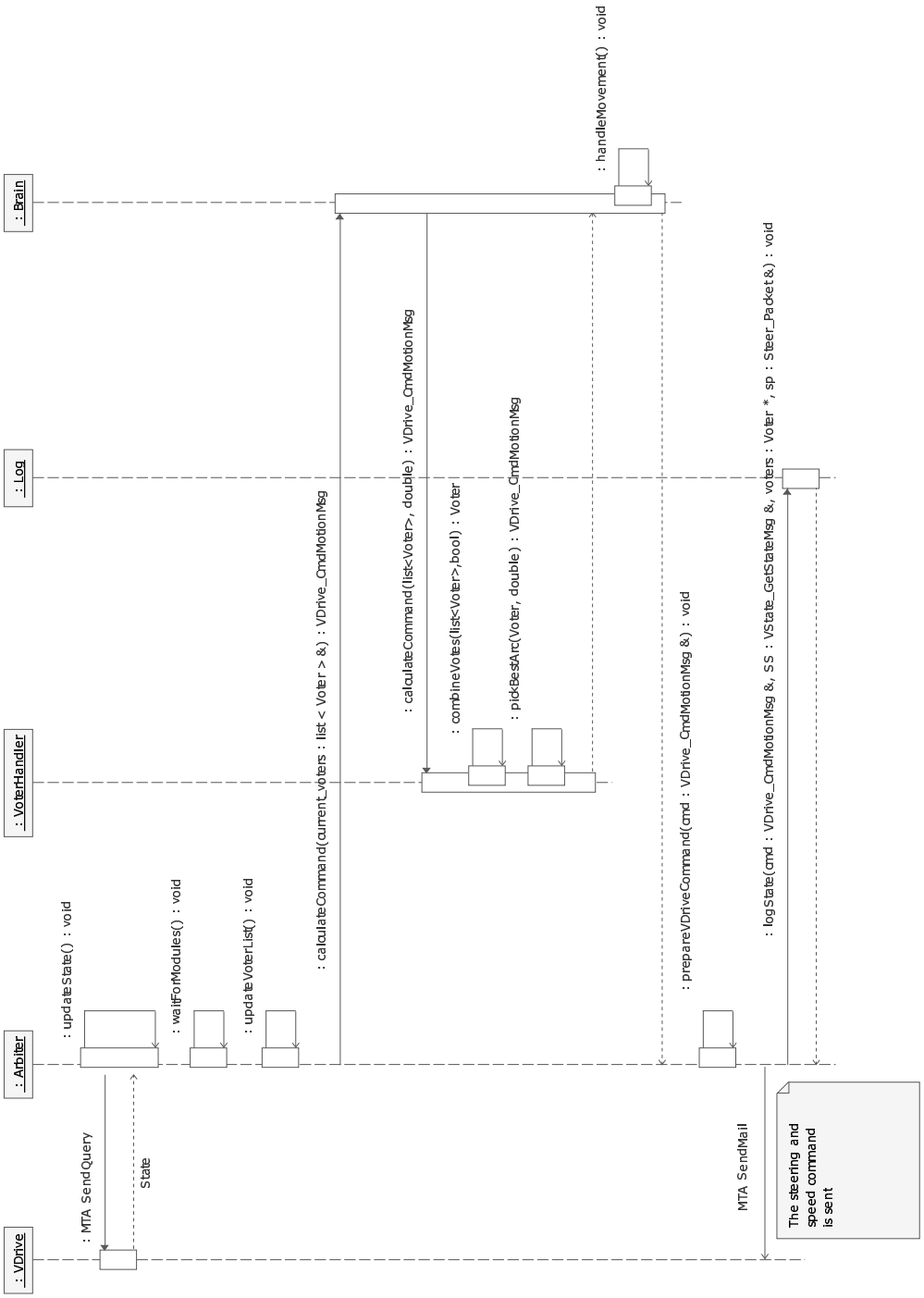

Figure C.1: Arbiter sequence diagram showing a typical iteration in the main loop of the arbiter.

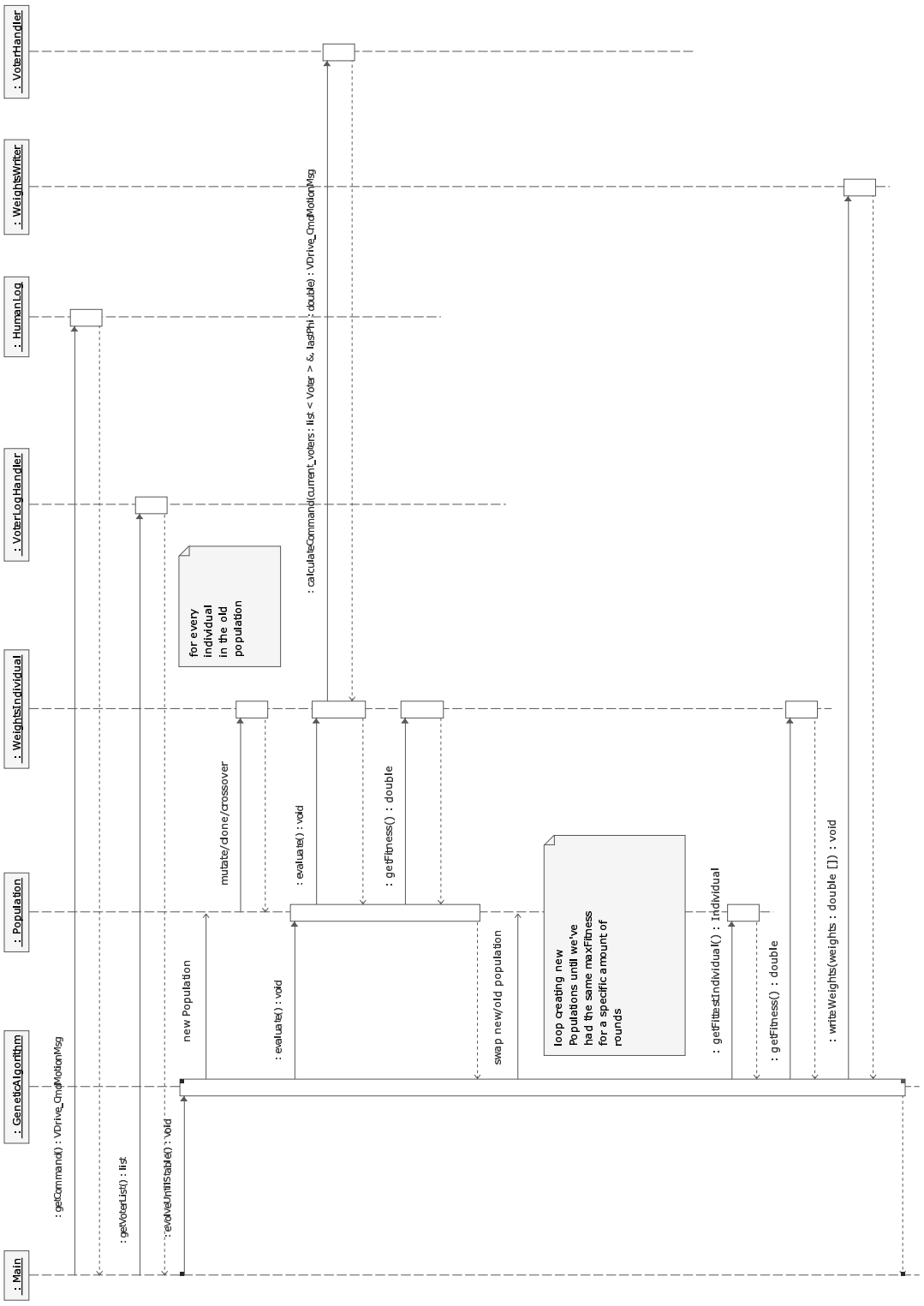

Figure C.2: The GeneticAlgorithm application sequence diagram, showing one iteration of the evolutionary algorithm.

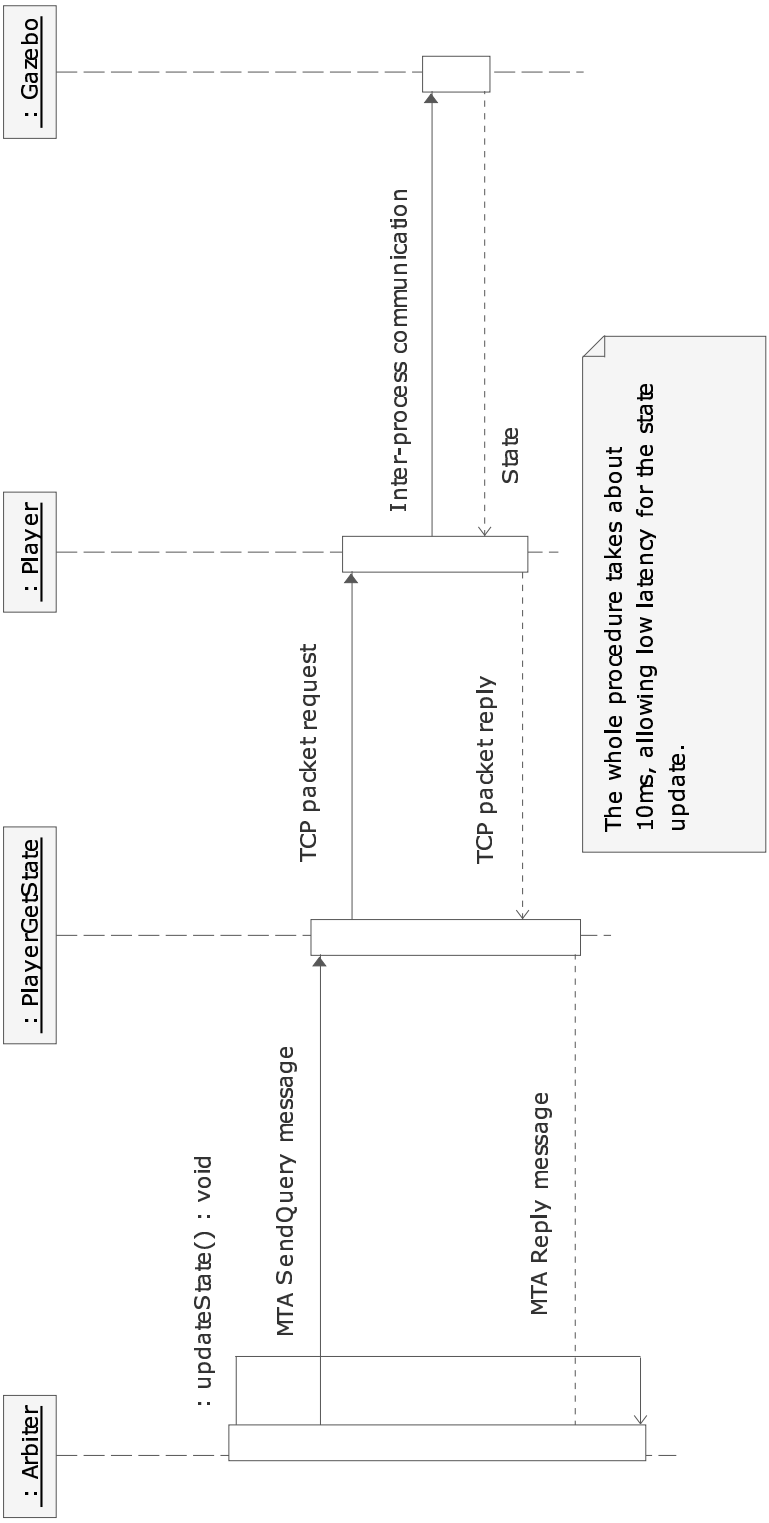

Figure C.3: PlayerInterface sequence diagram. The scenario showed is the Arbiter requesting the current vehicle state. The scenario of the Arbiter sending a steering command is very similar, but with the PlayerGetState module replaced by PlayerSend-Cmd.

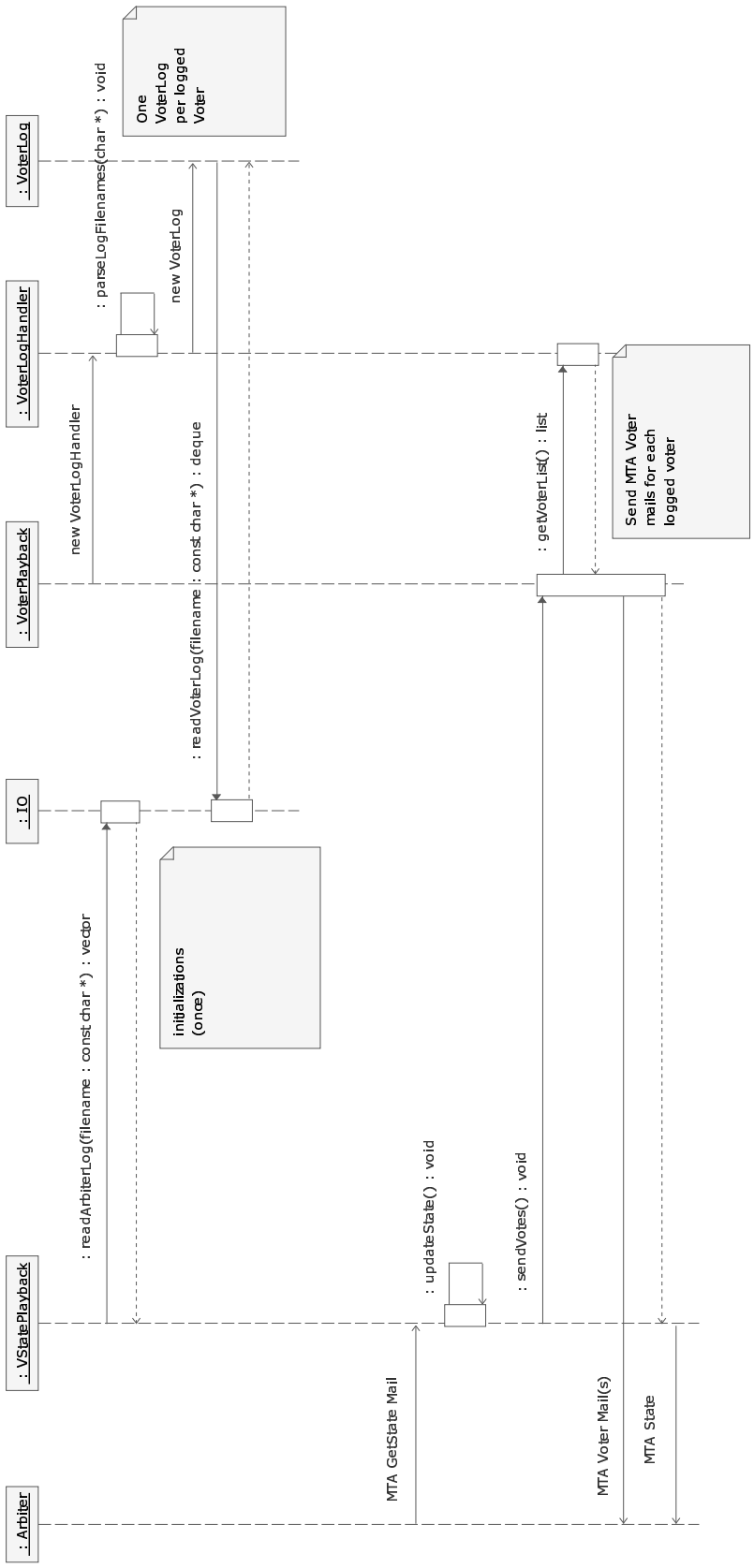

Figure C.4: LogPlayer sequence diagram. Shows one step of sending state and votes.