Improvements of the Arbiter in the DARPA Grand Challenge project

Author:
Henrik Kjellander, Lund Institute of Technology, Sweden

Mentor:
Richard M. Murray, Department of Control and Dynamical Systems, Caltech, USA

Co-mentor:
Lars Clemean, Department of Control and Dynamical Systems, Caltech, USA

Abstract

In 2002 DARPA (Defense Advanced Research Projects Agency) announced a race for autonomous vehicles would be held in March 2004. The course was LA to Las Vegas and there was a $1 million cash prize to the winner. From Caltech a team driven by students applied for the race. One year of hard work resulted in a fully autonomous vehicle that ended up #5. No one completed the course and another race is announced for October 2005. This project is about software research in new techniques about how to decide which direction the vehicle shall go, depending on sensor data. A module called the Arbiter takes these decisions. The research also required a simulation system, which was developed in parallel. Experiments performed include having a human driver drive the car while recording all sensor data. Then algorithms were executed to find ways to have the computer imitate the human driver's decisions. Another idea developed was mode management that makes the vehicle react different depending on which kind of environment it is in.

Background

Autonomous vehicles has been a dream for engineers in many years. In many science fiction movies you can see driver less cabs and other autonomous vehicles. The DARPA Grand Challenge race will drive research towards such machines faster than ever before. The military has interest in putting less troops at risk for, as an example, transportation missions in risky areas. Such a mission is a good example where it would be very suitable with an autonomous vehicle you just feed with GPS way points, telling it where to go. Then just load it with the cargo and let it drive itself to the goal. To be able traveling safe in unknown environments it requires a lot of intelligence and sensing. Just compare a road (where successful autonomous vehicles has existed for a long time already [1]) with a completely unknown, off-road terrain. The variations of terrain are almost infinite off the road. Bushes, slopes, holes, rocks, trees and other vegetation creates a very complex environment to take decisions in.

DARPA still thought it was possible and used a proved way to get people put a lot of time into something – a competition with a cash prize. Knowing how hard it is to build such a vehicle, the plan was to re-run the race once a year until someone made it withing the ten allowed hours of racing. A race was announced with a set of rules which told the competitors what kind of terrain and obstacles they had to be able to tackle with their vehicles, although the specific route where the race would take place would not be announced until two hours before start! By then a CD containing a text file with GPS coordinates were given to the teams. Each of these way points has a speed and radius, Putting them all together gives the participant a corridor which is the area the vehicle must stay within to avoid disqualification. The route in the first race, that was ran in March 13, 2004, consisted of 2500 way points from Barstow to Primm (close to Las Vegas) and had a total distance of 210 miles.

Figure 1: Bob - a truly autonomous vehicle

Many universities and engineering companies were interested and in total 40 teams applied to participate in the race. A qualification event was held a few weeks before the race where every team had to prove it's vehicle's safety and performance. Only eight teams made it through the qualification event and were allowed to start.
At the race day in March 13 the eight qualified teams were lined up for start. One of them was Team Caltech with their vehicle called Bob. Bob started out pretty good and made it out of the starting area, following a dirt road. After a while Bob unexpectedly decides to leave the road and drive aside it. A bit later, trying to get back on the road again, Bob gets stuck on a barbed wire, 1.3 miles away from the start position. That was the end of thousands of man hours spent by students on Caltech, building one of the first successful autonomous vehicles ever seen.

Team Caltech started out as a project course for undergraduate students at Caltech, lead by graduate student David Van Gogh and Prof. Richard Murray. Students designed, implemented and tested the vehicle during one year until the race was held. The vehicle was a donated Chevy Tahoe 1996 which was heavily modified with

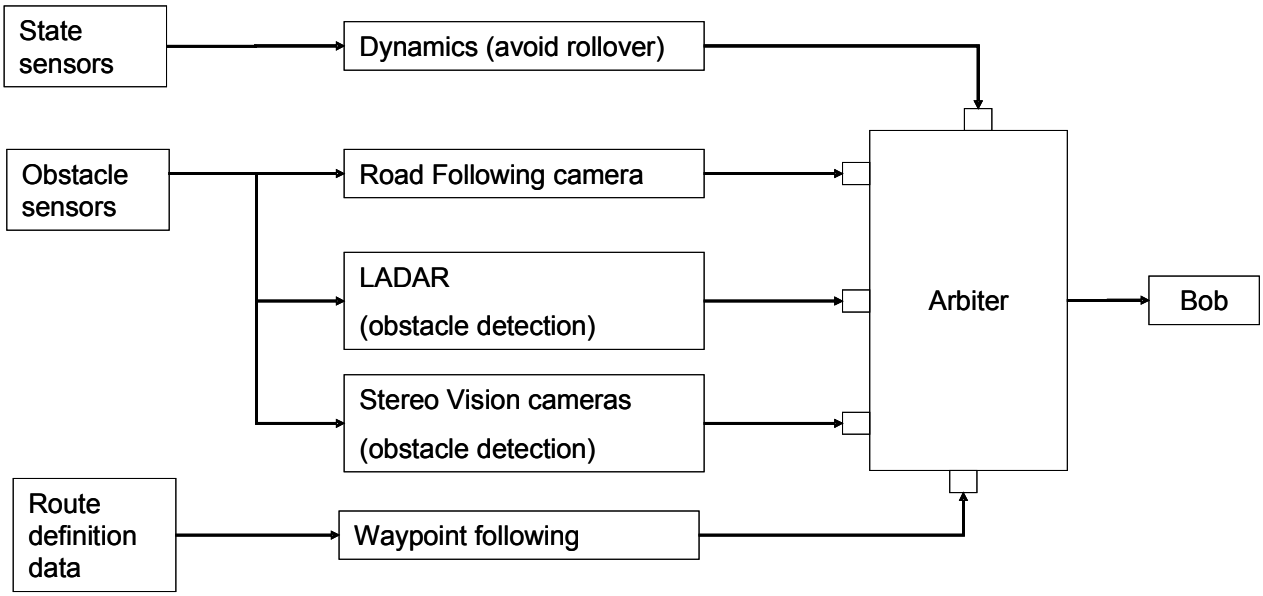

Figure 2. The planning and Arbiter architecture

actuators on throttle, steering and brakes. Other equipment includes eight desktop computers, two pairs of stereo vision cameras, two LADAR[1] units, a GPS receiver, an IMU[2] unit, a power generator and an external AC unit.
The stereo vision cameras provides depth information to the environment to detect obstacles [2]. So does the LADAR, which is a scanning laser beam that measures the distance to the objects that the beam hits.
Different software modules handles different tasks in a distributed application. The stereo vision cameras and the LADARs have their own modules, that analyzes the sensor data to decide how to turn to avoid them.
Another module looks at the current position, given the GPS data, and tries to keep the vehicle within the allowed corridor. To avoid making dangerous maneuvers there's a module that knows the vehicle dynamics and limits speed and steering commands to maintain safety.
Obviously, all these modules doesn't agree on where's the best direction to go all the time. That's where the Arbiter enters the system. The Arbiter acts like a judge which receives votes from each module, where each vote contains information of how that module rates the possible steering alternatives the vehicle has, and also a recommended speed for each steering. Looking at these votes, the Arbiter calculates a final decision which is sent to the actuators to command the vehicle that way.

The first version of the Arbiter module was written in 2003 with little modularization in mind and also lacked specifications and documentation. It was highly based upon two approached called DAMN[3] [3] and GESTALT[4] [2]. A new version was developed during the summer of 2004 which had the same functionality, but also bug fixes, additional functionality object orientated and documented design and well documented source code. During the development of the new Arbiter the need of another team objective became more and more obvious – the need of a simulation environment. Simulation was crucial for safe testing of the new Arbiter code without having to bring out the vehicle in a large flat area (for example a dry lake bed) to test all changes.

Simulation

The possibilities before the summer of 2004 were very limited for Team Caltech. A major objective was to investigate, select and then integrate a simulation environment for the existing software. After looking at available commercial products and open-source projects one decided to go for a simulation environment called Player/Gazebo [4]. Player is an interface server for robot simulation, which is designed to connect either to real hardware or a software simulation engine, called Gazebo. The "intelligent" software then talks only to player, receiving state of the robot/simulation and sends commands, without knowing if it's a real robot or a simulation engine that receives the command and delivers the state.
The simulation engine used is called Gazebo. It's a free, open-source, real-time, dynamics 3D-simulation engine. Gazebo was originally from a less detailed multi-robot simulation called Stage. Both started being developed by

[1] Laser Detection And Ranging
[2] Inertial Measurement Unit, to measure the vehicle position and angles.
[3] Distributed Architecture for Mobile Navigation
[4] Grid-based Estimation of Surface Traversability Applied to Local Terrain

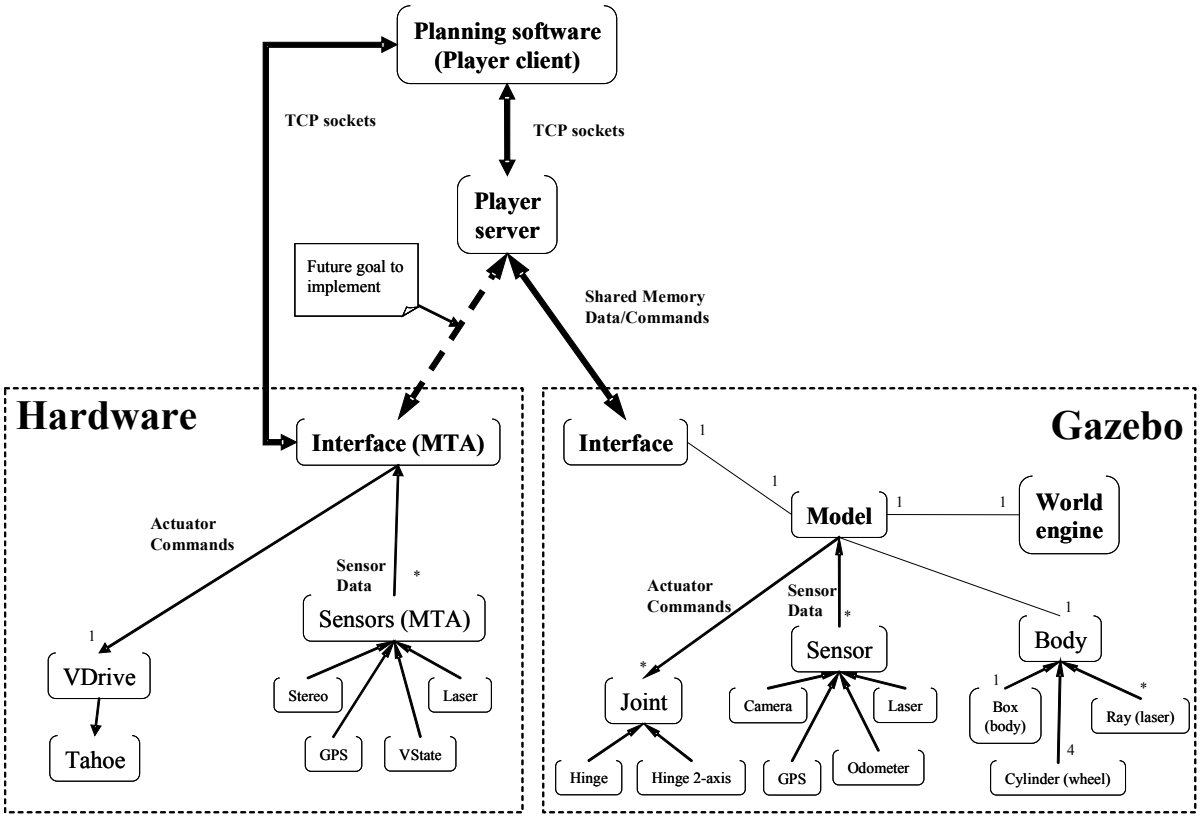

Figure 3. The simulation architecture. MTA is a message transport architecture used for sending messages between processes.

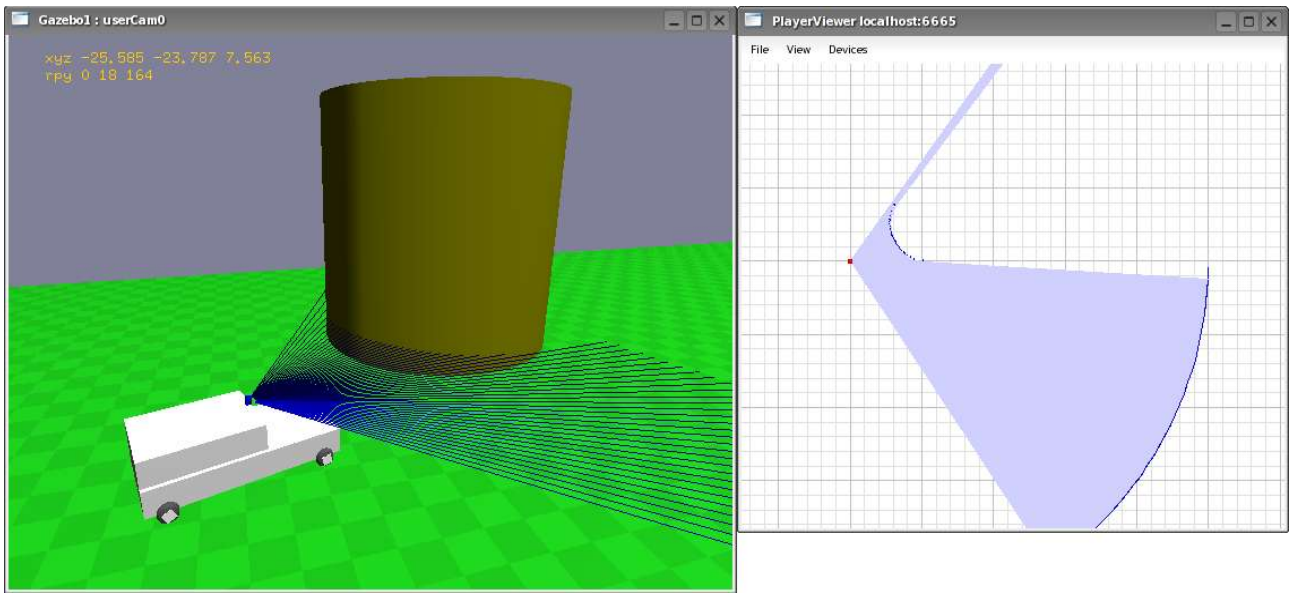

Figure 4: Gazebo screen shot. Left: the OpenGL display presented in real-time during running the simulation. The right image shows a commanding interface with the LADAR readings drawn.

people from the Robotics Research Lab at University of Southern California. Later on they were released under the GPL open-source license.
A large part of the summer 2004 was spent on getting used to the Player/Gazebo environment and integrating it to the existing planning software for Team Caltech's race vehicle. 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. A model of the Chevy Tahoe was developed which had the same length, width, height, mass etc as the real vehicle. Simulated sensor devices like LADAR, GPS and cameras were already included in Gazebo so connecting the existing software to those were 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.

Test reconstruction

To improve usability of logged test data, a program for playback of test logs was developed. It delivers recorded sensor and state data in the desired pace so the user can visualize what was happening at each time step and understand the decisions taken by the different modules and the Arbiter.

Voter weights

Each module that sends votes to the Arbiter is called a Voter. In the 2004 race all the voters had the same weights, which means no one was more important than the other. During the summer of 2004 research was performed to improve these weights and to find a way of intelligently set them. An approach for solving combinatorial problems called a "genetic algorithm" was developed [5]. The idea was to have a human driver drive a carefully set up course as good as he could while logging all possible sensor and state data. After collecting all the data, the data is used for calculating which set of weights would give a result, from the Arbiter, that is closest to the decision the human took. This algorithm was developed and sensor data was collected during a field test in El Mirage. Calculation of weight sets were performed but no general set of weights could be derived from the results. More effort into analyzing functions of speed and situation-specific dependencies of the best weight sets/functions is required.

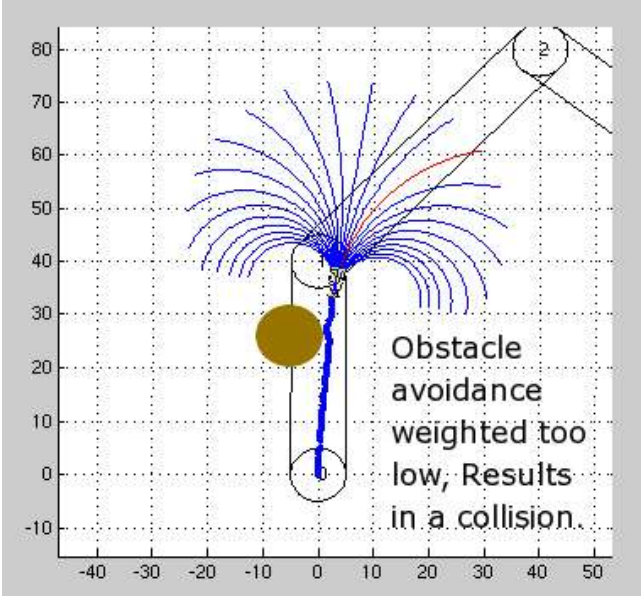

Figure 5: Test run with higher weighted path planning and corridor evaluation which means it priorities reaching the waypoints higher. Results in that obstacles are weighted too low, which results in a collision.

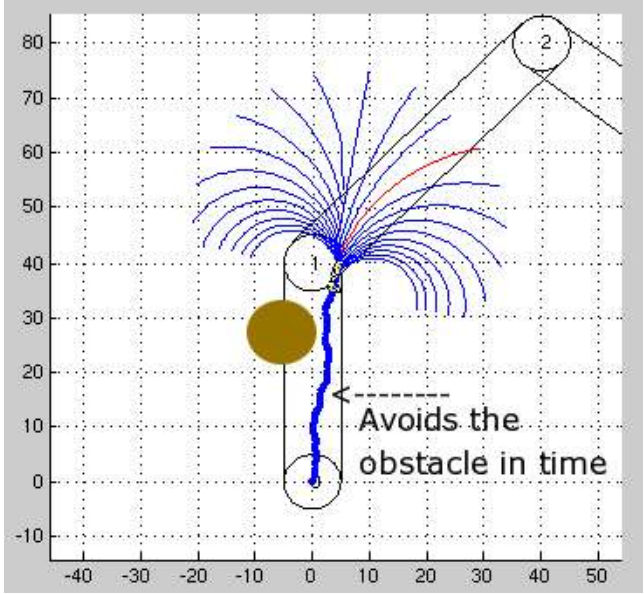

Figure 6: Test run with higher weighted obstacle avoidance and is therefore taking a longer path away from the obstacle (avoids it).

Methods

Re-design

Since most of the old Arbiter code was written in C-like function based code with lack of object orientation, the first thing that was performed was a clean design of classes that handles different responsibilities and hides unnecessary information away from the developer to avoid confusion. The object oriented design also made it possible to write class-specific unit tests to verify desired functionality in the different classes. The new code is also very well documented with a auto generated HTML-API available for other developers to look at when using or just trying to understand the structure of the Arbiter.

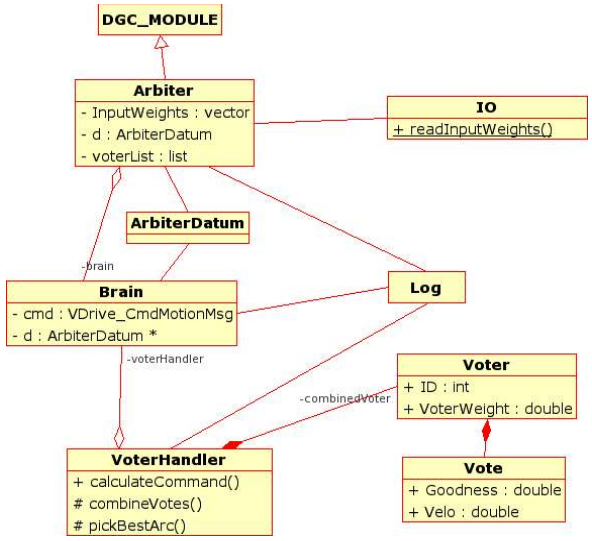

Figure 7. Simplified Arbiter class diagram

MTA modules

All the different programs in the software package are able to talk with each other between different computers using a message transport architecture called MTA. It was originally developed by Ike Grimmer at Caltech. Every module that shall be able to talk to another inherits the DGC_MODULE class. Then it's possible to also implement the different mailbox methods to send and receive messages. For example, the Arbiter is one module, LADAR and Stereo vision cameras have their own MTA modules for data processing and vote generation. VDrive is the name of the module that commands the actuators and VState is the name of the module that delivers vehicle state information.

Player Interface

To connect the existing planning software with the new simulation software, a new interface was written called PlayerInterface. It pretends to be VState and VDrive at the same time. When the Arbiter sends a steering command to VDrive it catches that command and updates the simulated Gazebo world. When an module asks for state information from VState, it replies with information taken from the simulation. This way the planning software can be tested transparently with the simulation environment without knowing what is executed in VDrive/VState

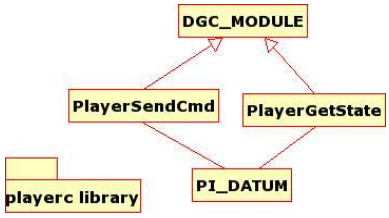

Figure 8. PlayerInterface class diagram

Genetic Algorithm

To solve the problem of selecting a good set of weights an application based on the concept called genetic algorithms was used. The process used is:

- recording all incoming sensor and state data when a human drives a detailed setup course.
- post-process this data with a genetic algorithm to find sets of weights in every time frame, that the Arbiter would need to have to take the same decision in that specific situation.
- use these sets of weights to improve the Arbiter decisions.

Results from this research within the genetic algorithm only gave clues about which voters to trust. More research has to be performed in speed- and situation-dependencies what has influence on the decisions. Probably one can derive a function of speed that relates to the weight-settings. Also distance to obstacles may have different influence on the decisions. Lack of available playback and test reconstruction tools in the project has made such research out of the scope of this SURF project, even though such applications were developed.

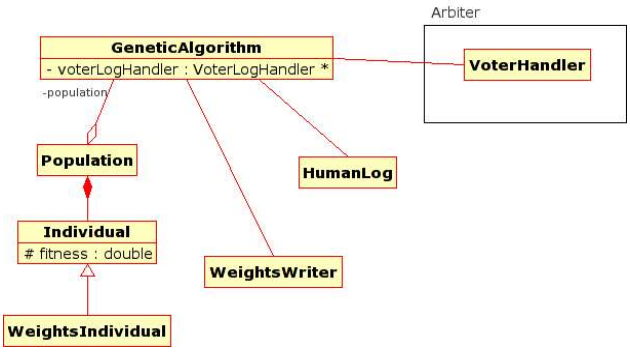

Figure 9. Genetic algorithm class diagram

Log Playback

To be able to understand what's really going on in a situation, when the vehicle drives and all the sensor modules are processing data (in real-time) and sending votes to the Arbiter, a playback library was developed called LogPlayer. It has the capability to read and parse all the log files and send messages just as all the modules were running (total transparency). It also allows the user do select how fast time shall progress.

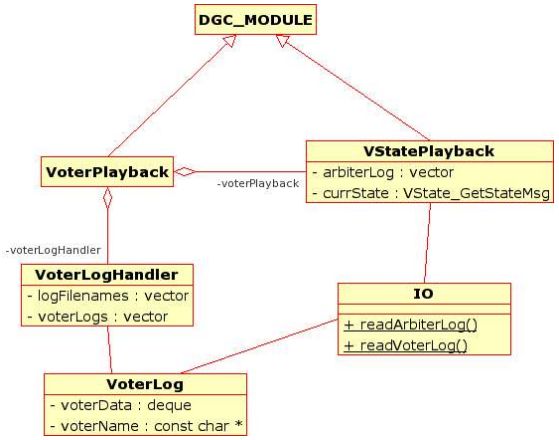

Figure 10. LogPlayer class diagram

Real-time execution

Since the project involves a lot of applications that executes concurrently on different computers, many issues about real-time execution appears. I improved the performance of the Arbiter by making it use a proper periodic sleep and I also researched profiling in general for multi-threaded Linux applications. The chosen application became gprof [6] with a multi-thread enabling fix [7] since gprof normally doesn't support multi-threaded programs. Profiling will probably have a large impact on the success of future programs in the DARPA Grand Challenge projects of Team Caltech. It is surprising that it hasn't been used until now.

References

[1] 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

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

[3] Julio K. Rosenblatt, "DAMN: A Distributed Architecture for Mobile Navigation", 1995
http://www.cs.cmu.edu/afs/cs.cmu.edu/user/jkr/www/papers/Dissertation

[4] Player/Stage homepage,
http://playerstage.sourceforge.net

[5] Stuart Russell, Peter Norvig, "Artificial Intelligence - A Modern Approach", 2003, Prentice Hall, page 119.

[6] GNU profiler manual
http://www.gnu.org/software/binutils/manual/gprof-2.9.1/html_mono/gprof.html

[7] Sam Hocevar, "HOWTO: using `gprof` with multithreaded applications",
http://sam.zoy.org/writings/programming/gprof.html

Acknowledgments

I thank my mentor, Richard Murray, and my Co-mentor Lars Clemean for all their help and advice. I would also like to say thank you to all the Team Caltech members for such a great team spirit and all the fun time we spent together.

Additional Suggestions

To really get more useful data out of the Genetic Algorithm, multiple field tests with larger maps are required to collect more reliable sensor data to process. Also further analysis in weight dependencies like functions of speed and distance to obstacles probably have a large impact on the resulting weights. A new system architecture is currently developed which has limited use for the Arbiter in it's current shape, but since the now existing code is very modularized and well documented, it can easily be re-used and adapted to new tasks in the future. The new profiling tools will also have a large impact on the total performance of the system.