

**AEVITA: Designing Biomimetic Vehicle-to-Pedestrian Communication
Protocols for Autonomously Operating & Parking On-Road Electric Vehicles**

by

Nicholas Pennycooke

B.Sc. ME, MIT (2010)

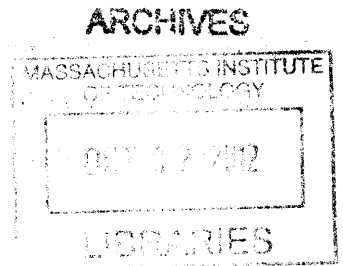
Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements of the degree of

Master of Science in Media Arts and Sciences

at the

Massachusetts Institute of Technology

September 2012



© 2012 Massachusetts Institute of Technology. All rights reserved.

Author: _____

Program in Media Arts and Sciences, MIT
September, 2012

Certified by: _____

Kent Larson

Principal Research Scientist, MIT Media Lab

Thesis Supervisor

Accepted by: _____

Prof. Patricia Maes

Associate Academic Head, Program in Media Arts and Sciences

**AEVITA: Designing Biomimetic Vehicle-to-Pedestrian Communication
Protocols for Autonomously Operating & Parking On-Road Electric Vehicles**

by

Nicholas Pennycooke

B.Sc. ME, MIT (2010)

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, on August 7, 2012 in partial fulfillment of the requirements of the degree of Master of Science in Media Arts and Sciences at the Massachusetts Institute of Technology
September 2012

Abstract

With research institutions from various private, government and academic sectors performing research into autonomous vehicle deployment strategies, the way we think about vehicles must adapt. But what happens when the driver, the main conduit of information transaction between the vehicle and its surroundings, is removed?

The ÆVITA system aims to fill this communication void by giving the autonomous vehicle the means to sense others around it, and react to various stimuli in as intuitive ways as possible by taking design cues from the living world. The system is comprised of various types of sensors (computer vision, UWB beacon tracking, sonar) and actuators (light, sound, mechanical) in order to express recognition of others, announcement of intentions, and portraying the vehicle's general state. All systems are built on the 2nd version of the 1/2 -scale CityCar concept vehicle, featuring advanced mixed-materials (CFRP + Aluminum) and a significantly more modularized architecture.

Thesis Supervisor: Kent Larson

Title: Principal Research Scientist, MIT Media Lab

**AEVITA: Designing Biomimetic Vehicle-to-Pedestrian Communication
Protocols for Autonomously Operating & Parking On-Road Electric Vehicles**

by

Nicholas Pennycooke

Thesis Reader:



7/27/2012

Alex 'Sandy' Pentland

Professor

MIT Media Arts and Sciences

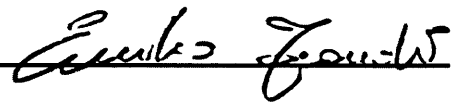
THIS PAGE IS
INTENTIONALLY LEFT
BLANK

**AEVITA: Designing Biomimetic Vehicle-to-Pedestrian Communication
Protocols for Autonomously Operating & Parking On-Road Electric Vehicles**

by

Nicholas Pennycooke

Thesis Reader:

A handwritten signature in black ink, reading "Emilio Frazzoli", written over a horizontal line.

Emilio Frazzoli

Associate Professor

MIT Laboratory for Information and Decision Systems

THIS PAGE IS
INTENTIONALLY LEFT
BLANK

Acknowledgements

I would like to take a moment to acknowledge the many incredible people who have worked with me on this immense project over the last few years. First, I would like to thank Kent for taking up the helm at a crucial time for our research group, and believing in my work and my vision. His kindness, hospitality, and guidance kept me on track while I swam through these uncertain waters.

Thank you to my readers, Assoc. Prof. Emilio Frazzoli and Prof. Sandy Pentland for their help in polishing this document into an acceptable academic account, as well as a future technology-inspiring story.

I would not be in a position to be writing this thesis if not for my colleague and friend Will Lark Jr. I started out working for him as a clueless undergraduate, but with his support and inspiration I am where I am today. Praveen Subramani, my partner in both academia and social exploits, for always being a good ear, great speaker, and grand person. To all my dear friends for helping me maintain that healthy work/life balance. Also, many thanks to my family, especially my parents, for always being my biggest fans.

If not for my extremely talented team of undergraduates over the past 2 years, I would be still in the machine shop for possibly another 2 years:

*Sean Cockey
Katharine Daly
Aron Dreyfoos*

*Laura Shumaker
Katarina Struckmann
Princess Len M Carlos*

*Chad Bean
Daniel Goodman
Michael Buchman*

Tom Lutz and John DiFrancisco, shop managers, for maintaining the amazing tools in the workshop, providing excellent advice on the best ways to make awesome things, and generally making sure I keep all my limbs intact.

I would also like to thank the entire Denokinn and Hiriko teams, especially Carlos, for providing not only the funding to make ÆVITA a reality, but also in believing in our dream of making the CityCar concept a reality.

Last, but by no means, the least, the late Professor William J. Mitchell.

Bill, Praveen and I were the last of your admitted students. The last SoBs. I can only hope that the work that we have completed fits into your wonderful vision of how the world needs to adapt to all these new challenges we face everyday. It's been almost 2 years to the day since you left, and I will never forget the amazing opportunity you granted me, believing that I truly had something to offer to that vision. I cannot possibly begin to thank you enough. This is for you.

It's important to get the technology and the policy right, but in the end, the way you break a logjam is by engaging people's imagination, people's desire, by creating things that they never thought of before.

-William. J. Mitchell (1944-2010)



THIS PAGE IS
INTENTIONALLY LEFT
BLANK

Table of Contents

[1.0] Introduction	16
[1.1] The State Of Today's Cities.....	16
[1.2] The State Of Today's Transportation	17
[1.3] Changing Mobility.....	18
[1.4] The CityCar.....	20
[1.5] Introducing Autonomy.....	22
[1.5.1] The Expanded Gradient Of Autonomy	25
[1.6] Introducing ÆVITA.....	27
[2.0] Related Work	29
[2.1] Purpose Built Autonomous Vehicle Deployment.....	29
[2.2] Biomimetic and HMI Research.....	32
[3.0] 2nd Generation City Car System Development	35
[3.1] Predecessors	35
[3.1.1] Hiriko.....	37
[3.2] 2nd Generation ½ Scale Platform Design Overview.....	38
[3.3] Main Structural Frame	41
[3.4] Powertrain Module	45
[3.5] Folding Linkages	54
[3.5.1] Primary Linkages	56
[3.5.2] Folding Actuators.....	57
[3.5.3] Secondary And Synchro Link	59
[3.6] Robot Wheel Module	61
[3.6.1] Arm And Suspension	62
[3.6.2] Steering.....	64
[3.6.3] Steering Control.....	66
[3.6.4] Drive.....	69
[3.6.5] Drive Control.....	72
[3.6.6] Wheel And Tire	75
[4.0] ÆVITA System Development	78
[4.1] Design Overview	78
[4.2] Recognition Subsystem.....	81
[4.2.1] Computer Vision: Kinect Integration.....	81
[4.2.2] Eye Assemblies.....	84
[4.3] Announcement System	91
[4.4] Body Language System.....	94
[4.4.1] Basic Subsystem	94
[4.4.2] Vehicle Login and Personalization	97
[4.4.3] Combined Subsystem Behaviors	99
[4.5] Wireless Controller	102
[4.6] ÆVITA Summary	103
[5.0] Technical Evaluation	104
[5.1] 2GHS Platform & Control.....	104
[5.1.1] Folding Mechanism Failure Mode.....	104
[5.1.2] Folding Mechanism Loading & Speed	105
[5.1.3] Folding Mechanism Geometry.....	106

[5.1.4] Throttle Input Testing.....	107
[5.2] Recognition	109
[5.2.1] Kinect Sensor Feasibility.....	109
[5.2.3] Kinect Tracking Program Selection.....	110
[5.2.4] ÆTP Human Identification.....	111
[5.2.5] Eye Assembly Tracking Speed.....	113
[5.2.6] Continuous Eye Tracking Perceptions	114
[5.3] Announcement.....	115
[5.3.1] Non-directional speakers	115
[5.4] Body Language.....	116
[5.4.1] RAWL Light Blending	116
[5.4.2] RAWL Reaction Time.....	116
[5.4.3] Combined Body Language System Performance.....	117
[5.5] System-wide Performance.....	118
[6.0] Conclusion.....	119
[6.1] Findings.....	119
[6.2] Implications and Implementation	120
[7.0] Future Work	122
[7.1] User Evaluation	122
[7.2] Gestural Commands and Pedestrian-to Vehicle Communication.....	122
[7.3] Development of a Communication Standard.....	123
[7.4] Possible Applications Today	123
[7.5] Integration of Good Driver Habits.....	124
[8.0] Bibliography	125
[9.0] Appendix.....	128
[9-A] User Study.....	128
Instructions to The Participant.....	128
Instructions to The Conductor	129
Interaction Modes.....	131
Questionnaire	133
[9-B] Vehicle Personalization via NFC Devices	135
[9-C] Geometric Steering Angle - Servo Relationship	137
[9-D] IO Box Pin Schematics	138
[9-E] Castle Mamba Max Pro ESC Settings.....	139
[9-F] Additional Design Images	140
[9-G] ÆTP C# Code	145
MainWindow.xaml	145
MainWindow.xaml.cs	146
[9-H] Arduino Code.....	161
Primary Front Micro.....	161
Primary Rear Micro	173
Pupil Micro.....	176
[9-I] Custom/Modified Arduino Libraries	179
BLINKM_FUNCS_I2C.h	179
EL_ESCUDO.h	182
EL_ESCUDO.cpp	183

List of Figures & Tables

Figure 1-1: 2010 CO2 Emissions from Fossil Fuel Combustion	17
Figure 1-2: Selection of implemented shared mobility system providers	18
Figure 1-3: Reinventing the Automobile cover	19
Figure 1-4: The MIT Media Lab CityCar. Image by W. Lark.	20
Figure 1-5: The MIT Media Lab CityCar's maneuverability enabled by its robot wheels. Image by W. Lark.	21
Figure 1-6: Parking ratio enabled by the CityCar folding structure	21
Figure 1-7: Changing parking density afforded by autonomy and the CityCar	22
Figure 1-8: Some of the key stakeholders in autonomous vehicle research	23
Figure 1-9: The basic gradient of autonomy	24
Figure 1-10: Definition of vehicle automation, as recommended by BAST	24
Figure 1-11: The expanded gradient of autonomy	26
Figure 1-12: The application space of the ÆVITA system	28
Figure 2-1: The GM EN-V	29
Figure 2-2: Personal Robotics Group's Aida	32
Figure 3-1: 1st half-scale CityCar prototype	35
Figure 3-2: CAD - 1st generation half-scale CityCar prototype	36
Figure 3-3: The Hiriko Fold, commercialization of the concept CityCar	37
Figure 3-4: CAD - The 2nd generation ½ scale CityCar platform with ÆVITA, unfolded	38
Figure 3-5: CAD - The 2GHS platform with ÆVITA, folded	39
Figure 3-6: CAD - The 2GHS main structural frame	41
Figure 3-7: CAD - The 2GHS main structural frame, as viewed from below	43
Figure 3-8: The 2GHS platform, with both underbody CFRP pieces top center	44
Figure 3-9: CAD - The 2GHS platform, with the top CFRP attached	44
Figure 3-10: CAD - The 2GHS powertrain module, with some robot wheel components attached	45
Figure 3-11: The dry 2GHS powertrain module with tray	46
Figure 3-12: CAD - The 2GHS powertrain module, with some robot wheel components attached, as viewed from top	47
Figure 3-13: CAD - The 2GHS platform showing structural difference between front and rear powertrain modules, as viewed from below	48
Figure 3-14: The lead/trail arm attached to the powertrain module, showing the angular offset	49
Figure 3-15: 3D printed ABS plastic suspension upper connection point	50
Figure 3-16: FEA optimized 6061-T6 suspension upper connection point	50
Figure 3-17: Powertrain module with tray pulled out on drawer slides	51
Figure 3-18: Internals of the front powertrain module's tray. At right, IO Box	52
Figure 3-19: The completed rear powertrain module's tray	53
Figure 3-20: CAD - The 2GHS folding linkage main elements	54
Figure 3-21: CAD - The 2GHS primary linkage	56
Figure 3-22: The Linak LA23 actuator, extended	58
Figure 3-24: The Linak LA23 actuator, retracted	58
Figure 3-25: The secondary and synchro links	59
Figure 3-26: CAD - The 2GHS folding sequence, showing the lift of the main structural frame, and the overlap of the front and rear powertrain pivot points	60
Figure 3-27: CAD - The 2GHS front-left/rear-right robot wheel module, without suspension	61
Figure 3-28: The 2GHS front-left/rear-right arm and suspension structure	62
Figure 3-29: The 2GHS robot wheel steering mechanism	64
Figure 3-30: CAD - Details of the robot wheel architecture and the steering mechanism	65
Figure 3-31: Diagrammatic representation of the Ackermann steering geometry	67
Figure 3-32: The 2GHS robot wheel drive motor and motor mount	69
Figure 3-33: The 2GHS robot wheel hub assemblies, without lug bolts or bearings	70
Figure 3-34: The 2GHS robot wheel hub assembly, mounted to the drive/steer assembly	71
Figure 3-35: The 2GHS robot wheel assembly, Castle MMP ESC at bottom left	72

Figure 3-36: CAD - The 2GHS rim and tire	75
Figure 3-37: The two parts of the 2GHS wheel, before epoxying	76
Figure 3-38: The 2GHS rim and tire mounted to the hubs	76
Figure 3-39: ÆVITA, the complete platform	77
Figure 4-1: ÆVITA system data/signal network	78
Figure 4-2: Arduino Uno and Mega 2560 microcontroller boards	79
Figure 4-3: Microsoft Kinect sensor mounted on ÆVITA	81
Figure 4-4: Microsoft Kinect	82
Figure 4-5: ÆVITA's human and skeleton tracking, early version of the ÆTP	83
Figure 4-6: ÆVITA's left eye	84
Figure 4-7: Sparkfun original El Escudo Arduino shield	85
Figure 4-8: ÆVITA's pan and tilt servos, right eye, without pupils	87
Figure 4-9: Difference in angle needed for two rotating elements to converge on a single point	88
Figure 4-10: ÆVITA's eyes and Kinect sensor, ÆTP running on screen in background	90
Figure 4-11: CAD - ÆVITA's announcement system	91
Figure 4-12: ÆVITA's announcement system	92
Figure 4-13: Parallax Ping! sonar sensor	94
Figure 4-14: One group of RAWLs, attached to the stationary motor mount	96
Figure 4-15: Diffused and reflected light responding to changes in object proximity	96
Figure 4-16: ÆVITA RFID login reader and cards	97
Figure 4-17: ÆVITA login RAWL personalization	98
Figure 4-18: ÆVITA aggression sequence	100
Figure 4-19: ÆVITA submission sequence	101
Figure 4-20: Microsoft Xbox 360 Wireless Controller for Windows (Microsoft.com)	102
Figure 5-1: Rear 4-bar linkage and front 4-bar linkage droop	106
Figure 5-2: Scaled forward and reverse throttle example	108
Figure 5-3: Original tracking program based on depth thresholds	110
Figure 5-4: Example 1 of The Infamous Small Man	111
Figure 5-5: Example 2 of The Infamous Small Man	112
Figure 5-6: Diagnostic print out via serial monitor of primary front micro	114
Table 4-1: Summary of two examples of body language system combinations	99
Table 5-1: Folding actuator loading and speeds	105
Table 5-2: Calibration data from ESC setting printout	107

THIS PAGE IS
INTENTIONALLY LEFT
BLANK

[1.0] Introduction

[1.1] The State Of Today's Cities

Today's cities are experiencing rapid urban densification, the likes of which has never been before seen in human history. For the first time in 2007-2008, the United Nations Population Division estimates that more than half of the world's population now lives in cities¹. This trend is expected to continue, with over 90% of expected population growth to occur in these urban cells. With this growth comes the opportunity to reduce suburban sprawl, integrate more mixed-use zoning, and increase the efficiency at which the city operates from both an energy and transportation point of view. More dense urban environments create a sense of vibrancy, as its inhabitants interact in ways suburban sprawls cannot achieve. However, today's cities were not built with this rapid growth in mind. In fact, the exact opposite is occurring. According to the United Nations Department of Economic and Social Affairs, Population Division, there is a trend of decreasing density. If current rates remain, the average land occupied by cities with populations of over 100,000 inhabitants will increase by a factor of 2.75 by 2030².

Current infrastructure in most modern cities is already struggling to keep up with the demand being asked of it. The American electric grid is largely made up of components built decades ago. Energy supply does not match demand, and so there are frequently megawatts of electricity being produced for non-existent loads. Similarly, spikes in electricity demand puts massive strain on the grid, requiring supplemental sources offering little to no resiliency or redundancy should portions fail.

¹ United Nations, Department of Economic and Social Affairs, Population Division. (2011). World Population Prospects: The 2010 Revision. New York

² United Nations, Department of Economic and Social Affairs, Population Division. (2011). Population Distribution, Urbanization, Internal Migration and Development: An International Perspective. New York, p12

[1.2] The State Of Today's Transportation

Transportation systems are similarly at peril. There exists an excess of personally owned automobiles, causing traffic congestion in most major metro areas, and represents a large portion of total CO₂ and other greenhouse gas emissions per year. The US Environmental Protection Agency (USEPA) 2012 Greenhouse Gas Inventory states, “The transportation end-use sector accounted for 1,772.5 Tg CO₂ Eq. in 2010, which represented 33 percent of CO₂ emissions, 23 percent of CH₄ emissions, and 48 percent of N₂O emissions from fossil fuel combustion, respectively.” Further, it states that of that figure, passenger vehicles and light-duty trucks accounted for 61% of total transportation related emissions³. If a non-incremental modal shift in how we think about mobility does not occur, the veins and arteries of the city, supplying its lifeblood – the people – will be constantly congested.

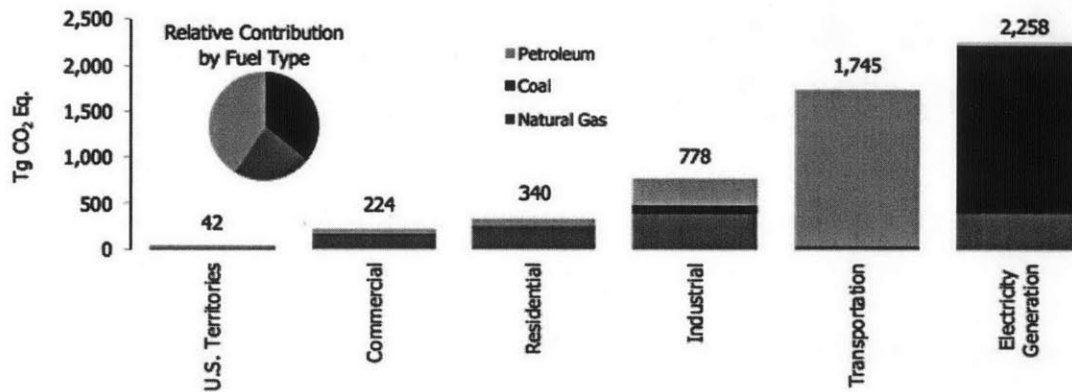


Figure 1-1: 2010 CO₂ Emissions from Fossil Fuel Combustion⁴

In order to support the growth we can expect to see, cities will have to move away from traditional, private ownership-centric transportation models, integrating a mobility ecosystem that is designed specifically around the needs of its

³ USEPA. (April 2012). Inventory of U.S Greenhouse gas emissions and sinks: 1990-2010, p3-13

⁴ USEPA. (April 2012). Inventory of U.S Greenhouse gas emissions and sinks: 1990-2010, Figure 3-5

inhabitants. In the last decade, there has been a notable boom in small to medium scale rollouts of such systems, providing some evidence that there are individuals and municipalities considering the implications of integrating shared mobility.

[1.3] Changing Mobility



Figure 1-2: Selection of implemented shared mobility system providers

One such system is Mobility on Demand (MoD). Originally conceptualized by the Smart Cities research group within the MIT Media Lab under Professor William J. Mitchell (1944-2010). MoD is a one-way, shared use mobility system which seeks to better balance the mobility supply and demand ratio, all the while integrating sustainable vehicle technology, distributed information and renewable energy systems, and minimizing the urban footprint required to move people from A to B. Vehicles can be picked up or rented from a nearby MoD station, and driven to the user's destination, parking it any other non-full station with no responsibility or even expectation for the vehicle to returned to its original pickup point. Outlined in

further detail in *Reinventing the Automobile: Personal Urban Mobility for the 21st Century* (2010), MoD combines many of the best features of currently deployed shared mobility systems, some of which as presented in Figure 1-2.

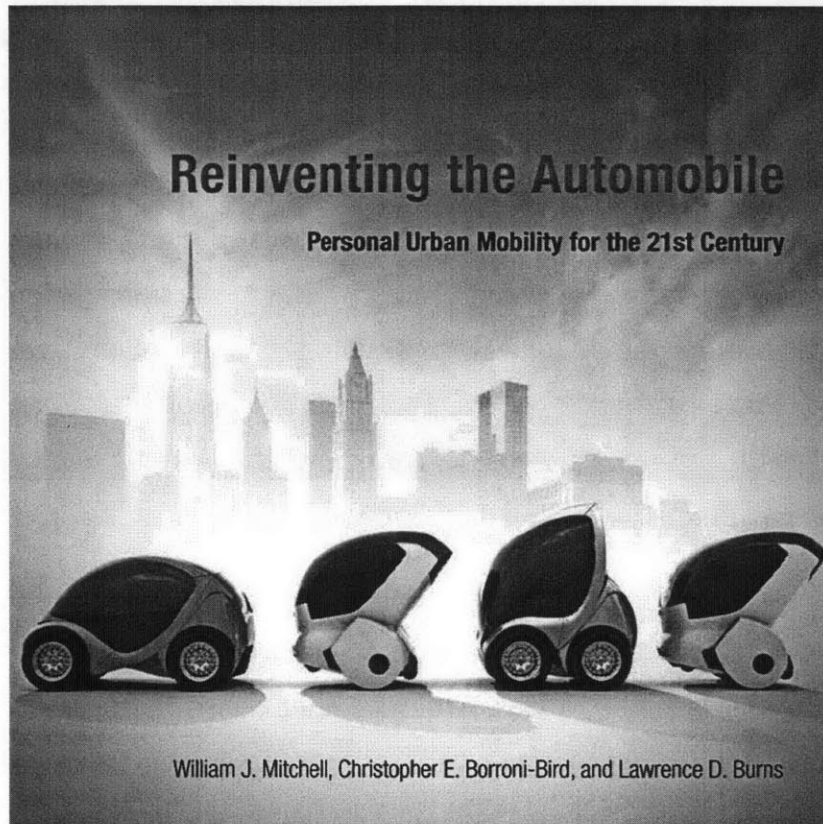


Figure 1-3: Reinventing the Automobile cover

Some of the key features of MoD include its one-way sharing, dynamic pricing incentive-based vehicle redistribution, and constant information sharing across the system. Perhaps most interestingly, however, are the types of vehicles used in MoD. Other one-way systems such as Autolib' and Hubway utilize only one kind of vehicle throughout: cars and bicycles, respectively. MoD seeks to improve upon this by providing a mobility ecosystem of connected electric vehicles of various types and form factors, allowing users to take the right kind of vehicle for the type of trip they intend to take. By providing a fleet of electric assist bicycles and tricycles, scooters and lightweight passenger vehicles, MoD is able to provide clean, efficient mobility satisfying the lion's share of most urban trips.

[1.4] The CityCar



Figure 1-4: The MIT Media Lab CityCar. Image by W. Lark.

Of these vehicles, the concept CityCar not only solves many of the mobility issues MoD attempts to tackle, but also sets a far reaching precedent for the future of mobility. The CityCar is a lightweight, two passenger electric vehicle, built on a highly modular platform, giving it a feature set that directly complements and enhances MoD's effectiveness. Central to the CityCar's architecture are the Robot Wheels at its corners. The core principle of Robot Wheel technology, as defined by Raul Poblano⁵, is the consolidation of a vehicle's complexity into its wheels, thus freeing the rest of the chassis from its typically static construction. Each corner contains its own drive motor, steering actuator, braking, and suspension systems, has no direct mechanical link between units, and is independently controllable through drive-by-wire technology. Each wheel has a total sweep of approximately 80 degrees, allowing for high maneuverability, including the ability to spin around its own central axis, dubbed an 'O-Turn'.

⁵ Poblano, R. V. (2008). Exploration of robotic-wheel technology for enhanced urban mobility and city scale omni-directional personal transportation. M.S. Thesis. Massachusetts Institute of Technology, USA.

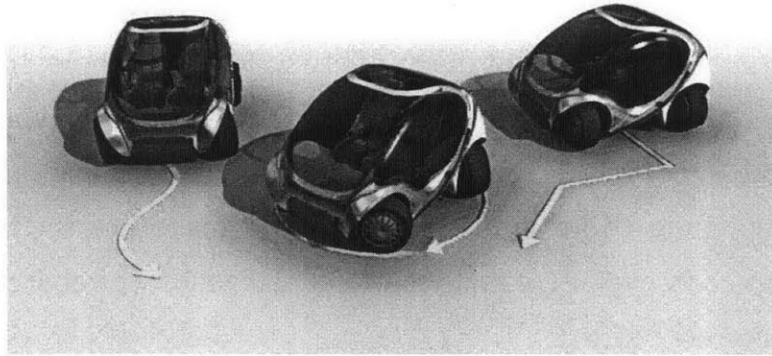


Figure 1-5: The MIT Media Lab CityCar’s maneuverability enabled by its robot wheels.
Image by W. Lark.

By removing the traditional drivetrain, the chassis can be much more dynamic: the vehicle can be made to fold. By enabling folding, the CityCar can reduce its total footprint to take up less than 1/3 of the space of a normal car. Coupled with the front ingress and egress, three of these vehicles can fit in one 8’ by 18’ parking spot.

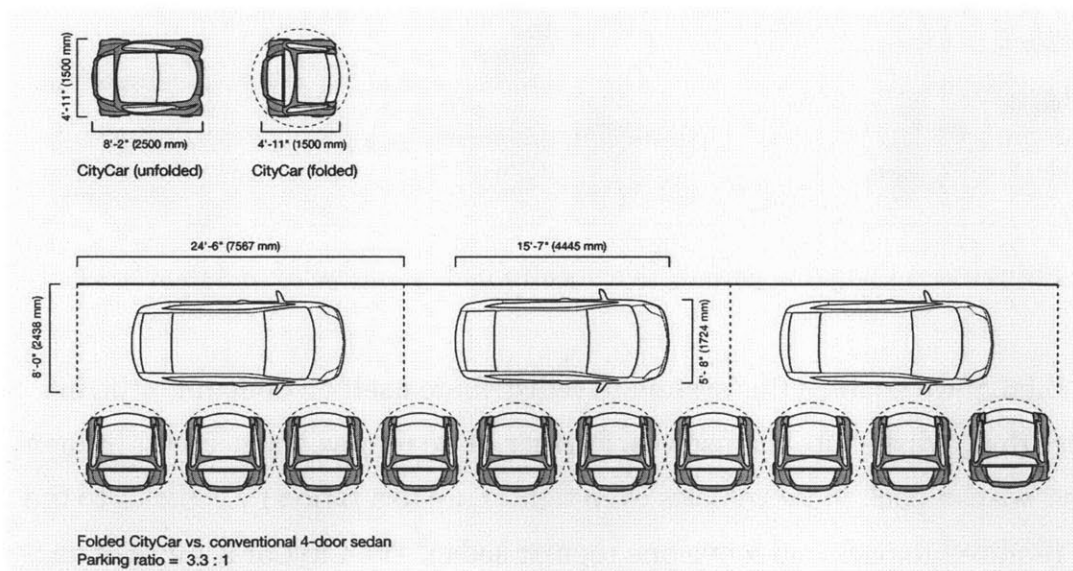


Figure 1-6: Parking ratio enabled by the CityCar folding structure⁶

⁶ Mitchell, W. J., Borroni-Bird, C. E., Burns, L. D. (2010), Reinventing the Automobile: Personal Urban Mobility for the 21st Century, Figure 9.20

This complements MoD by increasing the utilization rate of the land, and reducing the total urban footprint required of the mobility system as a whole. Introducing dynamic incentives for redistribution of the MoD fleet is a solution viable with technology available today. This requires relying on human behavioral dynamics, which at best is sporadic. Other vehicle sharing programs rely on the much more difficult, inefficient and expensive method of manually moving vehicles to the stations where they are needed.

[1.5] Introducing Autonomy

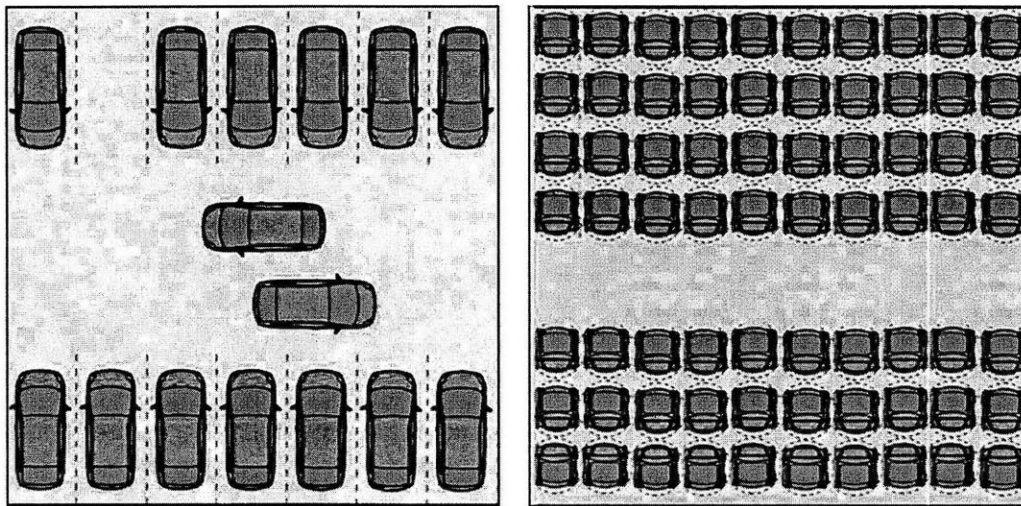


Figure 1-7: Changing parking density afforded by autonomy and the CityCar

What then, if we enabled this system to redistribute itself on demand, utilizing vehicles that redistribute themselves. Further, if we remove the need for human ingress, we can pack these vehicles even tighter, up to a ratio of 7 vehicles to the space required to park and move one regular sedan. The CityCar is more of an electronic car, than simply electric. It is already controlled by on board computer systems in a way not found in today's available vehicles. This makes it a particularly viable platform to deploy autonomous vehicle technology, as it only requires the addition of the various sensors needed to enable autonomy.



Figure 1-8: Some of the key stakeholders in autonomous vehicle research

Searching for the terms “autonomous car”, or “cars that drive themselves” gives a very quick glimpse into the ever-expanding world of autonomous vehicle research. Parallel parking assists, which perform the sometimes tricky maneuver automatically for the driver, have been installed in vehicles as early as 2003; Toyota being the first to deploy a commercially viable solution on that model year’s Prius sedan⁷. Those shopping for new high-end luxury automobiles such as the 2012 Lexus LS are afforded a vehicle that has advanced pre-collision detection systems, and ‘Lane Keep Assist’ that uses radar to keep the vehicle driving down the middle of its current lane⁸. The DARPA urban challenge, which blends the interests of the US Department of Defense and those of academia, answers “[...] a congressional mandate [...] to develop autonomous vehicles that reduce or even eliminate the presence of conductors in order to limit the loss of life on land military operations.”⁹ Private corporations such as Google have already logged over 100,000 miles of driving with little to no human intervention.¹⁰

⁷ Time. (2003). Best Inventions of 2003. Retrieved October 25, 2011, from Time Magazine: http://www.time.com/time/specials/packages/article/0,28804,1935038_1935083_1935719,00.html

⁸ Lexus. (2011). LS Safety & Security. Retrieved October 25, 2011, from Lexus: <http://www.lexus.com/models/LS/features/safety.html>

⁹ Induct. (2011, June). Company Presentation. Paris, France. p8

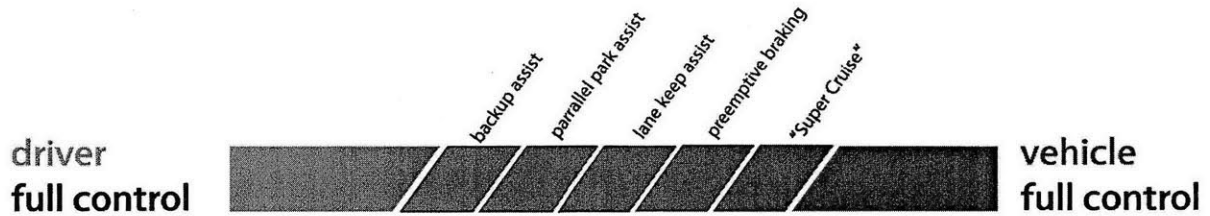


Figure 1-9: The basic gradient of autonomy

What this exposes is that autonomous vehicle research is actively occurring on a spectrum. In fact, one of the key tasks of the majority of the stakeholders in autonomous vehicle technology is to ratify a legally robust definition set of the various levels of autonomy, examples of which are used above in Figure 1-8. Currently, one of the accepted starting points towards this goal is the BAST definition set, presented below.¹¹

Definition of vehicle automation-degrees:

possible today

in future

- Driver Only: Human driver executes manual driving task
- Driver Assistance: The driver permanently controls either longitudinal or lateral control. The other task can be automated to a certain extent by the assistance system.
- Partial automation: The system takes over longitudinal and lateral control, the driver shall permanently monitor the system and shall be prepared to take over control at any time.

- High automation: The system takes over longitudinal and lateral control; the driver must no longer permanently monitor the system. In case of a take-over request, the driver must take-over control with a certain time buffer.
- Full automation: The system takes over longitudinal and lateral control completely and permanently. In case of a take-over request that is not carried out, the system will return to the minimal risk condition by itself.

Tom M. Gasser
26th Oct. 2011
slide No. 12

Figure 1-10: Definition of vehicle automation, as recommended by BAST

¹⁰ Markoff, J. (2010, October 9). Smarter Than you Think. Retrieved October 25, 2011, from The New York Times: <http://www.nytimes.com/2010/10/10/science/10google.html>

¹¹ Gasser, T. M. (October 26, 2011), Additional Requirements for Automation Liability and Legal Aspects Results of the BAST-Expert Group, p12

[1.5.1] The Expanded Gradient Of Autonomy

The world of ‘Google’ vehicles may be some years away though, as both technology and policy are not yet ready. The state of Nevada recently became the maverick on the policy front, however, by becoming the first municipality to sign into law, provisions that will set the precedent for autonomous vehicles to operate on their highways¹². Once the technology becomes commoditized to the point of mass deployment, as the stakeholders in Figure 1-7 are working towards, there are several new use cases for self-driving vehicles – also occurring on its own gradient.

As a nascent technology, which involves relinquishing human trust to a mechanical system, autonomous vehicle technology will not immediately be the Google car. Figure 1-10 not only shows how autonomy not only progresses from driver to vehicle control, but also that two versions of autonomy exists – with and without a driver. Autonomy may begin simply as controlled, indoor operation such as in a confined parking structure. This ‘robotic valet’ involves a system where the complex sensor arrays and computing requirements are moved from the vehicle into parking lot infrastructures, allowing drivers to have their vehicles park themselves into ultra-efficient arrays. At the far end of the spectrum are vehicles whose driver algorithms are fine-tuned to the function the vehicle will be performing. Using biometric data and driving behavior extracted by analyzing the habits of good taxi drivers, good truck drivers, autonomous vehicles can behave as if they are expert drivers of their designated jobs. It could be taken further, such that an autonomous vehicle operating in a sleepy southern town will be a lot more polite, than say one tasked to navigate the hectic streets of Manhattan.

¹² Shunk, C. (2011, June 12). Nevada passes law governing the use of autonomous vehicles. Retrieved October 25, 2011, from Autoblog: <http://www.autoblog.com/2011/07/12/nevada-passes-law-governing-the-use-of-autonomous-vehicles/>

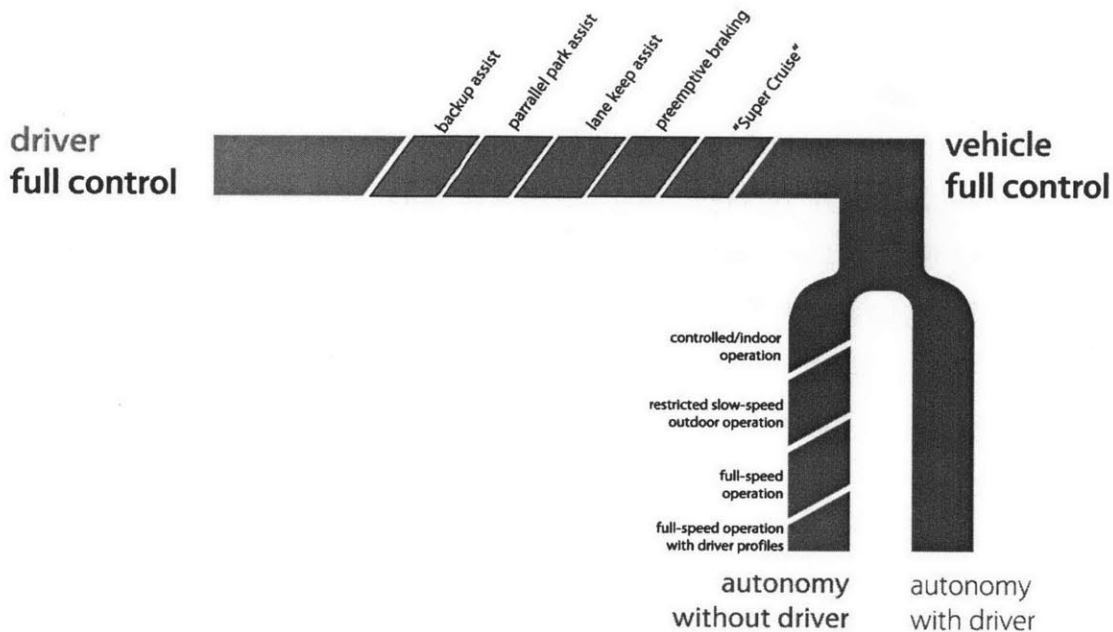


Figure 1-11: The expanded gradient of autonomy

Most use cases of autonomy currently assume that there will always be a driver in the seat, being shuttled from A to B with minimal input from the vehicle's passengers. Platooning, dynamic self-redistribution, on-demand pickup, and fixed-circuit autonomous shuttles are all applications where there is no driver while the vehicle navigates the environment. Normally, when on-road vehicles are operating, the feedback loop always occurs through the driver or passenger(s): eye contact, horns, turn signals, shouting, and hand gestures. These communication conventions are largely taken for granted, and happen in a very fast, natural way. This can be easily realized by observing a busy intersection.

But if an autonomous system such as the one described above was to proliferate, how do we go about handling these intuitive conventions once we remove the human factor? Statistics from NHTSA show that in 2009, there were 4092 pedestrian fatalities caused by accidents with motor vehicles.¹³ This accounted for

¹³ National Highway Traffic Safety Administration. (2012), *Fatality Analysis Reporting System (Fars) Encyclopedia*. Retrieved June 28, 2012, from NHTSA: <http://www-fars.nhtsa.dot.gov/Main/index.aspx>

12% of all vehicular fatalities that year. It is not difficult to imagine that most of these incidents were likely because of miscommunication, or reduced pedestrian/driver vigilance. This situation gives us the unique opportunity to not only match, but also exceed the currently low levels of communication quality between vehicles and pedestrians, regardless of whether or not a human is in the driver seat.

[1.6] Introducing ÆVITA

The purpose of this thesis is to provide various answers to this problem space, by breaking the question down into its constituents, and proposing viable electromechanical interventions that address each concern. As a pedestrian, one has to be able to know what the autonomous vehicle is about to do next. The vehicle has to be able to communicate recognition of those around it, and subsequently announce its intentions. These communication protocols have to be achieved immediately, and intuitively, so as to not require the populous to read an instruction manual prior to feeling safe. Such a system will be most powerful when it can be applied to many different stages of the driverless car gradient, and is highly transposable to many different vehicle types.

Lastly, a way has to be found to take design cues from the living world to effectively tie the above three concerns together. The successful implementation of this work will lead to the transposition of the developed system on to any autonomous vehicle platform, and help to significantly alleviate robotic vehicle operational anxiety by those who will have no choice but to navigate the same spaces with them. The name of the developed system to answer the above stated problems is the Autonomous Electric Vehicle Interaction Testing Array, or ÆVITA. As Figure 1-11 below demonstrates, ÆVITA was primarily developed for application to the arm of full autonomy that does not have a human in the driver seat. However, the system is not exclusively confined to this arm. Driver vigilance as it is currently could always be better. Hence, removing responsibility from the driver to watch

the road, one could expect the conduit of information transfer to be sleeping, reading, texting, or generally not aware of his or her surroundings. In this case, ÆVITA may also be applicable as a constantly aware communication entity.

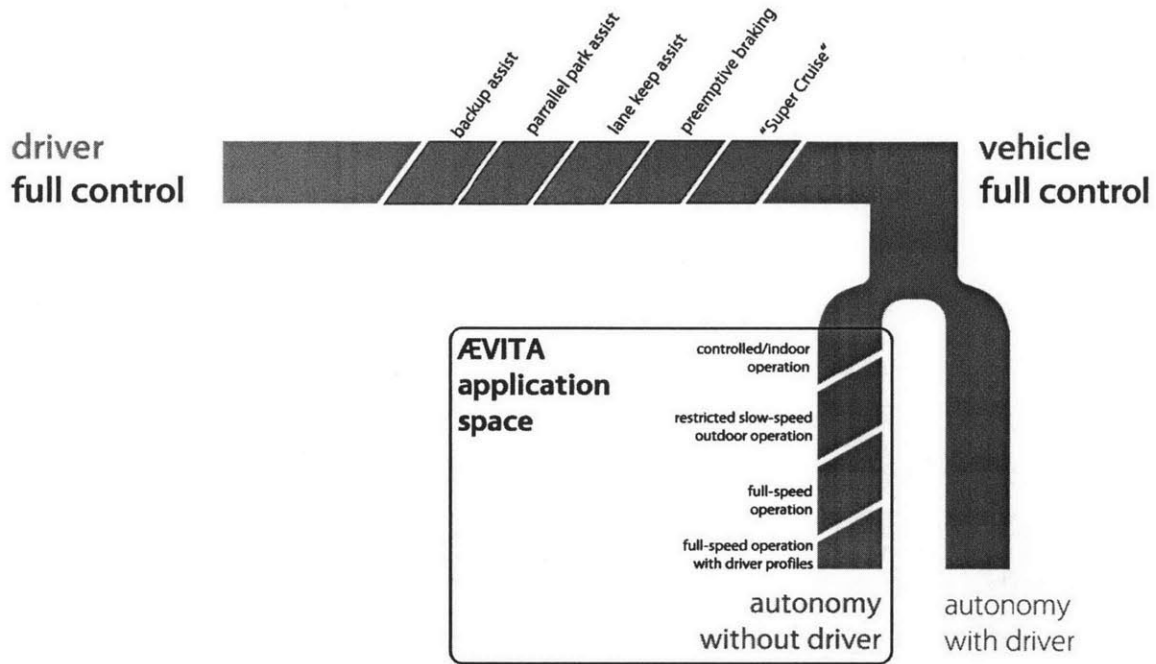


Figure 1-12: The application space of the ÆVITA system

In summary, ÆVITA has three layers of communication it leverages, attempting to cover the simplified spectrum of how we communicate with each other. ÆVITA is built to be able to:

- Express its recognition of humans in its field of view
 - *Recognition System*
- Announce its intentions through directed contextual messages
 - *Announcement System*
- Utilize the dynamic CityCar platform, its various sensors and actuators to evoke body language
 - *Body Language System*

[2.0] Related Work

[2.1] Purpose Built Autonomous Vehicle Deployment

There are many groups who are actively working towards deploying autonomous vehicle technology. Some of the best examples of autonomy gradient occupiers were previously mentioned in Section 1.5. Most research currently occurring involves the retrofit of traditional vehicles with the sensors and computing systems required to achieve autonomy. Several platforms however have been purpose built with the intention of building autonomous operation as a feature, not an add-on.

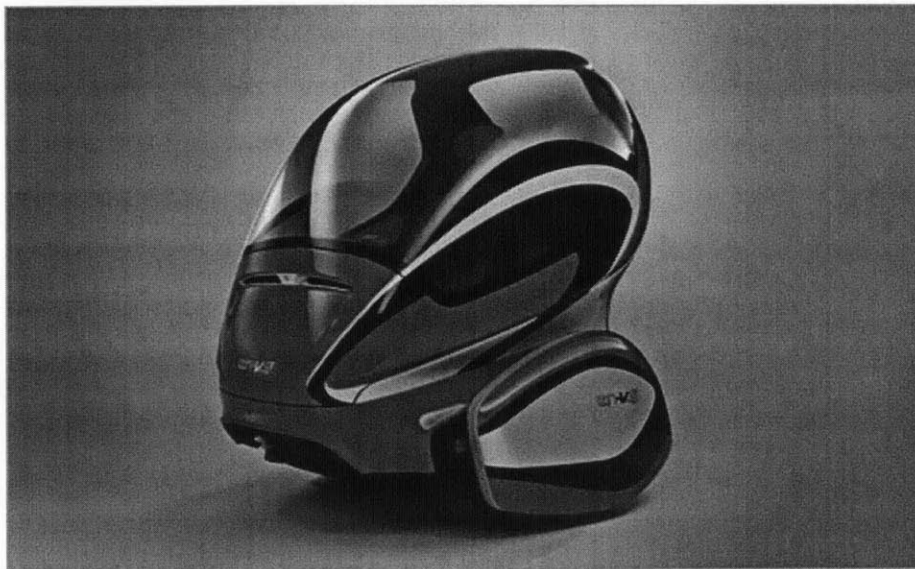


Figure 2-1: The GM EN-V

The GM EN-V (Electric Networked Vehicle) project was a joint project between GM and Segway to develop a prototype of what the future of urban mobility would look like.¹⁴ In fact, the leaders of the EN-V project collaborated closely with the Smart Cities Group during its conception, and it was built with the same mindset as the CityCar, which is why an early version of the EN-V concept shares space on

¹⁴ Motavalli, J. (March 24, 2010). G.M. EN-V: Sharpening the Focus of Future Urban Mobility. Retrieved June 28th, 2012 from New York Times: <http://wheels.blogs.nytimes.com/2010/03/24/g-m-en-v-sharpening-the-focus-of-future-urban-mobility/>

the cover of *Reinventing the Automobile*. Some of the core features of the EN-V are:

- *Leverages electrification and connectivity, creating a new class of personal urban mobility*
- *Autonomous driving, parking and retrieval with advanced sensors and drive-by-wire systems*¹⁵

Sharing such close DNA with the CityCar and its goals of changing the mobility landscape make it almost its cousin. Having autonomy built-in as a core feature does set the two apart to some degree, however the system's description makes no mention of rich communication pathways from the vehicle to pedestrians.

The EN-V may remain a technological demonstration at best, but there are already fully autonomous systems available for public in place. Companies such as Paris based Induct are working not only on closed-loop, campus style autonomous transport vehicles, but also on robotic valet systems¹⁶. The larger CityMobil initiative incorporates many subprojects aimed at creating multimodal intelligent transportation systems – many of which include the incorporation of autonomous Personal Rapid Transit (PRT) solutions.¹⁷ One of the projects currently operating in France is the Cybus. Similar to the closed-loop bus concept from Induct, it picks up passengers on demand along a predetermined route and carries them to their requested destination.¹⁸ Both systems are viable platforms for the inclusion of the ÆVITA, as they also completely lack of any pedestrian communication protocols.

¹⁵ GM. (March 24, 2010). EN-V Fast Facts. Retrieved June 28th, 2012 from GM: <http://media.gm.com/autoshow/2010/public/cn/en/env/news.detail.html/content/Pages/news/cn/en/2010/March/env03.html>

¹⁶ Induct, (June 2011). Company Presentation. Paris, France. p28-45

¹⁷ CityMobil. (2012). CityMobil Objectives. Retrieved June 28th, 2012, from CityMobil: <http://www.citymobil-project.eu/site/en/Objectives.php>

¹⁸ Inria. (December 5, 2011). Le Cybus d'Inria en démonstration à la Rochelle. Retrieved June 28th, 2012, from Inria: <http://www.inria.fr/actualite/mediacenter/cybus-inria>

In the field of autonomous vehicle deployment and communication, the system most closely resembling ÆVITA is the voice-commandable robotic forklift developed by Seth Teller, et.al. In a 2010 IEEE Conference paper titled *A Voice-Commandable Robotic Forklift Working Alongside Humans in Minimally-Prepared Outdoor Environments*, the authors describe a system built to demonstrate how an autonomous machine that has to operate in human inhabited environments may receive input from a supervisor, navigate the obstacle-filled workspace, and interact with humans it encounters.¹⁹ The last of those three main features is achieved through the two of the three categories defined by the ÆVITA system – recognition and announcement. The robotic forklift expresses its recognition by activating a set of marquee lights (addressable LED strings) in the direction of the person, and audibly announces that a human is approaching. It goes further to announce its state through written text on LED signage based on the context of the situation.

While this work does go further into the reverse communication case, where a human send commands or gestures to the autonomous vehicle, the robotic forklift utilizes methods of vehicle-to-pedestrian communication that may not be intuitive or rapid enough for an on-street environment. Reading text output may be slower than what is needed in the streetscape, and it cannot be assumed that all persons around an autonomous vehicle can understand the message for various reasons. Aside from the audio warning, the system also does not emulate any natural human or animal interactions in a biomimetic sense, nor does it take advantage of explicit body language responses, with the recognition marquee lights being a close abstraction of this concept.

¹⁹ S. Teller, A. Correa, R. Davis, L. Fletcher, E. Frazzoli, J. Glass, J.P. How, J.h. Jeon, S. Karaman, B. Luders, N. Roy, T. Sainath, and M.R. Walter. (2010). A voice-commandable robotic forklift working alongside humans in minimally-prepared outdoor environments. In International Conf. on Robotics and Automation, pages 526–533, Anchorage, AK.

[2.2] Biomimetic and HMI Research



Figure 2-2: Personal Robotics Group's Aida

Anthropomorphism and biomimicing research in the field of robotics has been on going for a number of decades. In the MIT Media Lab, work by Dr. Cynthia Breazeal in the Personal Robotics Group, where robots such as Kismet²⁰ and MDS²¹ were developed specifically to understand ways to engage human-robot interaction. While the depth of interaction between humans and the class of robots in this thesis is far beyond what may be necessary for the living EV, it helps to define what kinds of electromechanical solutions can be found to personify an autonomous electric vehicle. Moving towards the automotive, Aida (Affective Intelligent Driving Agent)²² also developed by the Personal Robotics Group, shows how robots can be integrated into a vehicle, but can only provide insight into driver/passenger to vehicle communication.

²⁰ Breazeal, D. C. (2000). Kismet Overview. Retrieved October 26, 2011, from Kismet: <http://web.media.mit.edu/~cynthiab/research/robots/kismet/overview/overview.html>

²¹ Personal Robotics Group. (2008). MDS Overview. Retrieved October 26, 2011, from Personal Robots Group: <http://robotic.media.mit.edu/projects/robots/mds/overview/overview.html>

²² Personal Robotics Group. (2003). Aida Overview. Retrieved October 26, 2011, from Personal Robotics Group: <http://robotic.media.mit.edu/projects/robots/aida/overview/overview.html>

In the Biomimetic Robotics Lab, headed by Assistant Professor Sangbae Kim, entire mechanical assemblies are designed to match or exceed the performance of its naturally occurring counterparts. Robots such as Stickybot and the Hyper dynamic quadruped robotic platform (Cheetah Robot)²³ are the truest form of biomimetic design. The Stickybot was built to demonstrate how to build active and passive limb locomotion, as well as researching the ways gecko's feet adhere to low friction surfaces.²⁴ The Cheetah Robot attempts to develop a high-speed locomotion platform, designed with high torque motors and the natural gait dynamics of a real Cheetah. However, both of these prototypes do not integrate any form of human interaction, focusing on electromechanical replication are not interactive, and are not designed to exhibit naturally understandable behaviors.

In both fields of biomimetics and HMI, the above presented are only a few examples of the work being done. Prof. Kim's previous lab at Stanford, where for example Stickybot was originally designed, continues to push the design boundaries of design electromechanical systems based on nature. Prof. Breazeal's PhD adviser, Prof. Emeritus Rodney Brooks, was one of the pioneers in developing responsive robots with which humans may interact.²⁵

²³ Biomimetic Robotics Lab. (2008). Research. Retrieved October 26, 2011, from Biomimetic Robotics Lab: <http://sangbae.scripts.mit.edu/biomimetics/>

²⁴ Cutkosky, M. (May 24, 2011). Stickybot III. Retrieved June 28th, 2012, from: Biomimetics and Dexterous Manipulation Lab: <http://bdml.stanford.edu/twiki/bin/view/Rise/StickyBotIII.html>

²⁵ CSAIL. (July 2010). Rodney Brooks – Roboticist. Retrieved July 29th, 2012 from CSAIL: <http://people.csail.mit.edu/brooks/>

In a paper submitted to a 1997 IEEE conference, Shibata et al explored the idea that robots have advanced to the point where we may begin to treat them as equals, and so both verbal and non-verbal communications will be important.²⁶ They proposed to build a pet robot with which to begin understanding this space. Observing human-to-human, as well as human-to-animal interactions will provide important indicators to programming the right kinds of human machine interactions. This idea resonates with the objectives of ÆVITA, however, the system presented in this thesis focuses firstly on the machine to human interactions, as well as specific interventions suitable for an automotive application.

²⁶ Shibata, T., Yoshida, M., & Yamato, J. (1997). Artificial emotional creature for human-machine interaction. *1997 IEEE International Conference on Systems Man and Cybernetics Computational Cybernetics and Simulation*, 3, 1-6. Ieee. Retrieved June 28th, 2012 ,from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=635205

[3.0] 2nd Generation City Car System Development

[3.1] Predecessors

The ÆVITA system is built on top of a custom half-scale prototype of the MIT CityCar. Over the past three and a half years, there have been several half-scale prototypes developed for various purposes. The first generation half-scale was built as the first fully functional driving and folding platform, meant to be used as a proof-of-concept and expose of the various core features of the CityCar idea. The five core principles of the CityCar are:

- i. *Robot wheel technology*
- ii. *Drive-by-wire controls*
- iii. *Front ingress/egress*
- iv. *Foldable chassis*
- v. *Fully electric drivetrain*

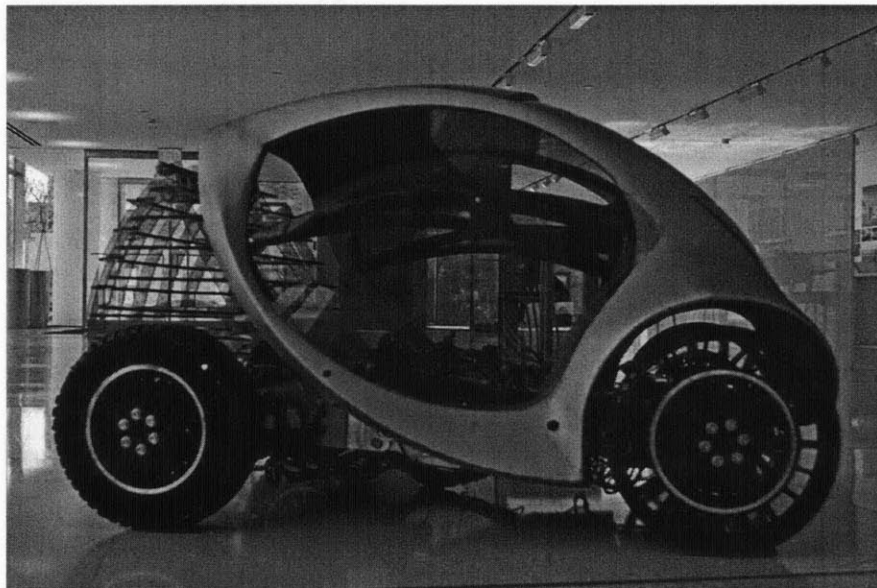


Figure 3-1: 1st half-scale CityCar prototype

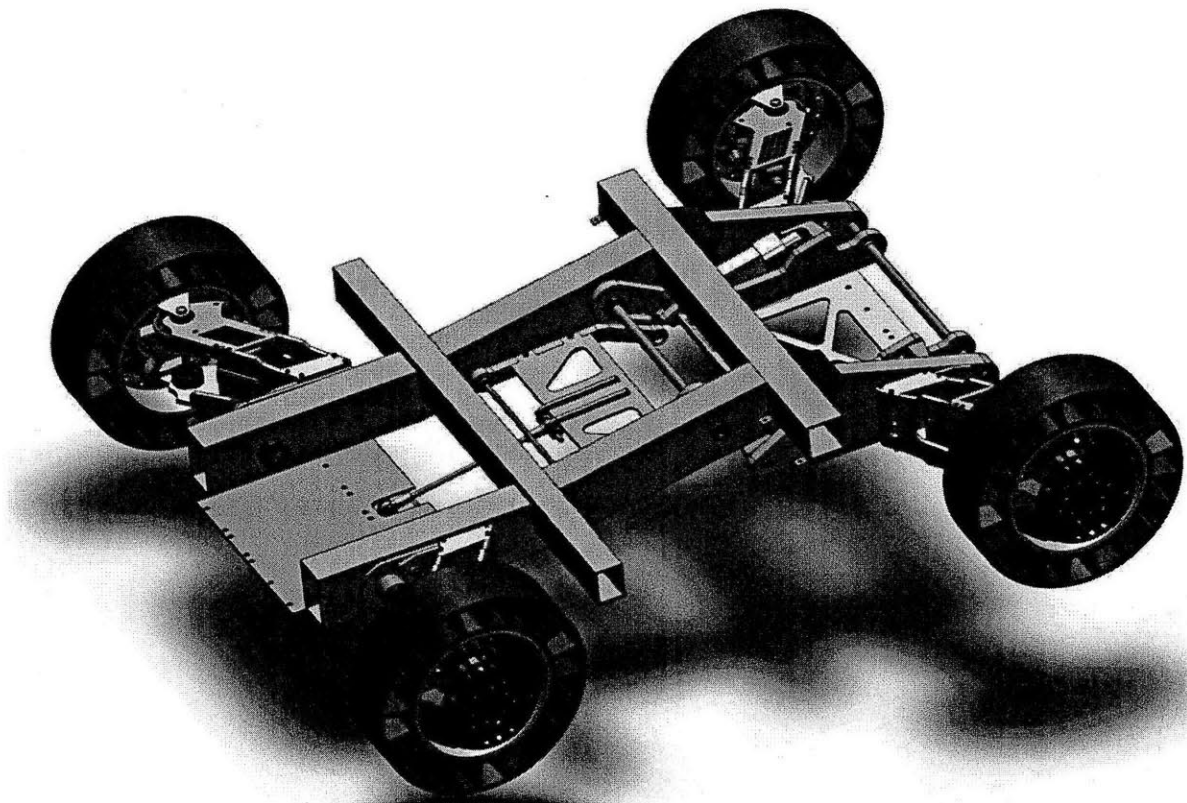


Figure 3-2: CAD – 1st generation half-scale CityCar prototype

Following the development of the first generation fully functional prototype came the first museum/exhibition version, based on the same chassis design, but featuring a polished all-aluminum exoskeleton. This version was built for the Smithsonian Copper-Hewitt Design Museum in New York City for their National Design Triennial “Why Design Now” exhibit, 2010. It is currently on display at the MIT Museum for the MIT 150th year anniversary.

[3.1.1] Hiriko

These two first generation prototypes lead directly to the culmination of almost a decades worth of research into the concept: the commercialization and realization of the first full-sized, operational, and to-be-sold version of the CityCar. The Hiriko Fold was developed by the graduate students of the Smart Cities/Changing Places research group, in conjunction with a sponsor of the MIT Media Lab, Denokinn, and a consortium of traditional automotive suppliers operating in capacity as co-manufacturers. Current work is being done to fully homologate and crash test the vehicle, with an estimated availability late 2013-14.



Figure 3-3: The Hiriko Fold, commercialization of the concept CityCar

[3.2] 2nd Generation 1/2 Scale Platform Design Overview

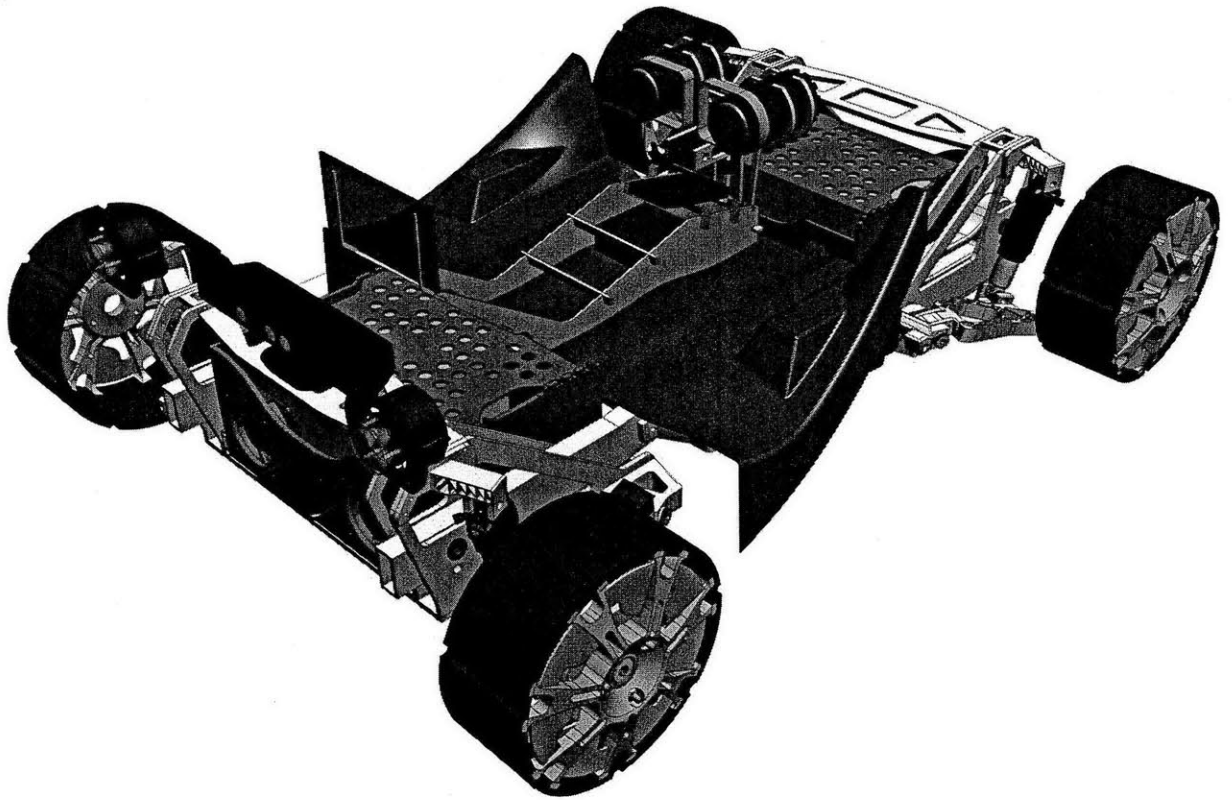


Figure 3-4: CAD - The 2nd generation 1/2 scale CityCar platform with AEVITA, unfolded

Building on much of what was learnt from developing the 1st generation prototypes, as well as working with the folding chassis of the Hiriko Fold, the 2nd generation 1/2 scale platform, referred to from this point on as the 2GHS, improves upon the design of its predecessors in several ways. The platform makes use of a revised folding architecture, repeated modules in the front and rear, and fully redesigned robot wheels. It makes heavy use of advanced composite materials and thorough finite element analysis (FEA) on almost all components and assemblies in order to minimize weight while maintaining structural integrity.

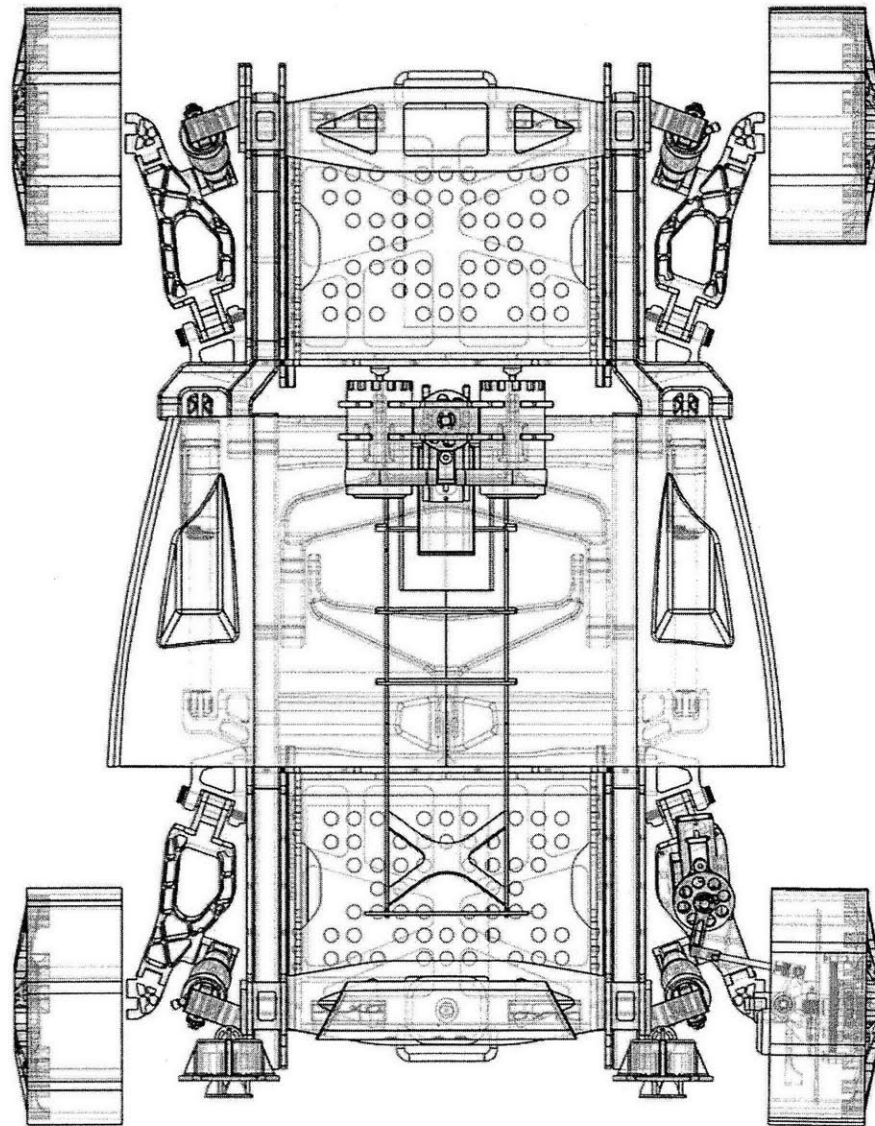


Figure 3-5: CAD - The 2GHS platform with ÆVITA, folded

The platform was also designed specifically with the intention to incorporate various sensor systems, integrated microcontroller subsystems, as well as various lighting and motion actuators. The platform is designed to fully incorporate the key operational features of the CityCar concept, as outlined in Section 3.1, and can be broken down into four main subassemblies:

- Main structural frame
- Powertrain modules
- Folding linkages
- Robot wheels

Each main subassembly is made up of several smaller assemblies, of which more detail will be described in the following Sections. The three main ÆVITA system modules are deeply integrated into the 2GHS structure. The announcement system has its own mechanical subassembly, as does the recognition system. The body language system largely relies on the electromechanical design already existing on the 2GHS. Details of the mechanical and sensor design and integration of the ÆVITA system are discussed in Section 4.



[3.3] Main Structural Frame

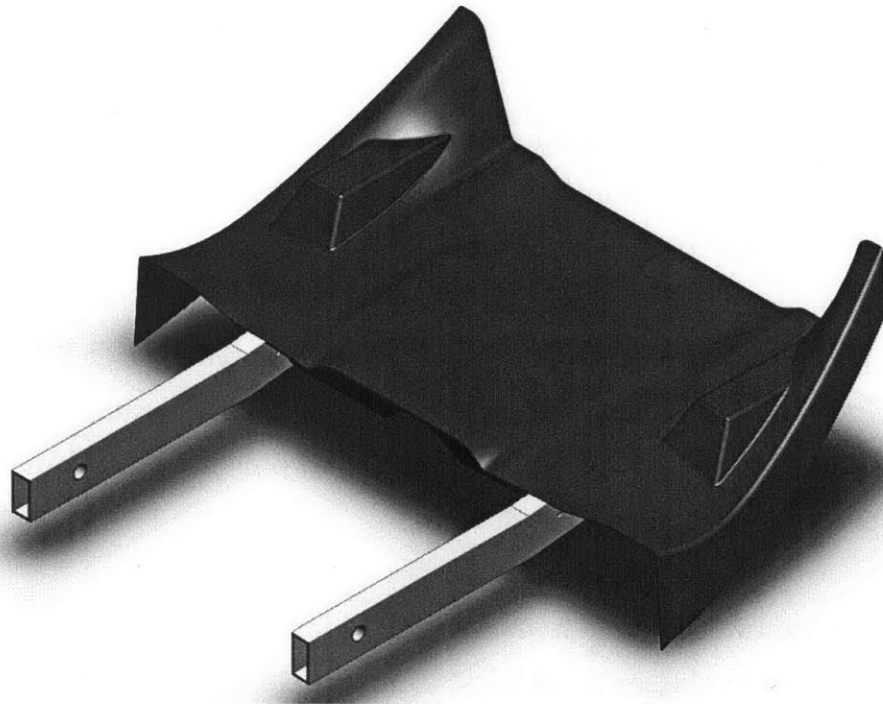


Figure 3-6: CAD - The 2GHS main structural frame

Central to the 2GHS is the main structural frame, which not only serves as the primary pivot points for most of the folding linkages, but also as the final load bearing element of the platform. There are five parts that make up the main structural frame: two main bars, two carbon fiber reinforced plastic (CFRP) underbody members, and a single CFRP top member. The five parts of mechanically bonded together to form a single structure.

The main bars are T6-6061 aluminum alloy (T6-6061) rectangular bars that have been cut and welded to incorporate the necessary angular changes as the design called for. T6-6061 tubular pieces are also welded into the main bars, forming the main folding linkage pivot bushing housings.

CFRP was chosen for the remaining structural members for a single reason. CFRP is an advanced composite material, finding its way into more and more

mainstream automotive applications due to its exceptional tensile strength to weight ratio. Assuming a volume of 1m^3 , CFRPs can achieve up to 0.94 MPa/kg , while stainless steel AISI 304, for example, reaches only 0.14 MPa/kg (substech.com). This constitutes an order of magnitude greater strength for CFRP versus conventional steel structures. CFRPs also have the added ease of forming complex surface pieces without the need for very expensive forming processes, as it relates to a one-off prototype such as this. However, CFRP is inherently much more expensive to manufacture as it requires pricey carbon-fiber filaments, either left as strands or woven into cloth, and is also costly create on a large scale since much of the standard layup process is done by hand. It is because of this why CFRP and other composite materials are typically only used in advanced motorsports and the aerospace industry.

As prices continue to fall and manufacturing processes improve, CFRP will be seen on many more classes of vehicles, including electric vehicles. Weight is a cyclical problem for electric vehicles. As weight increases, larger motors are required to move the vehicle at usable velocities. Larger motors require more power, and so larger battery packs must be integrated. Larger battery packs increase the overall weight, and so the cycle restarts. To counteract this, the use of composite materials in the vehicle's structural members becomes very attractive, for the reasons outlined previously. BMW has taken this exact stance with the introduction of their new 'i' line of electric and hybrid vehicles²⁷.

For the 2GHS, the carbon cloth chosen was a 2x2 twill weave, 3K weight roll, with a nominal thickness of 0.22mm . 2x2 twill refers to a cloth that has 2 weaves, or threads, woven equally over and under. 3K weight means that each thread has 3000 carbon filaments. This weave and weight was chosen for its moderate strength, and high formability over complex and small radii of curvature molds. All CFRP parts were laid-up using a two part clear epoxy, and cured using vacuum

²⁷ Ozler, L. (September 3, 2011). The Carbon Age Begins: Start of Carbon Fiber Production for BMW i3 and BMW i8. Retrieved June 20th, 2012 from Dexitgner: <http://www.dexitgner.com/news/23727>

bagging techniques over positive molds. All molds were 3-axis CNC machined out of high-density polyurethane foam, and surface sealed using thickened epoxy. Both underbody pieces are made up of 3 layers of the carbon fiber cloth described above, while the top member has 5 layers. The top member features contours designed to provide a clear range of motion for the various folding elements beneath it, as well as two wing-like structures on its side, serving as mounting points for the prototype's exoskeleton frame (See Appendix 9-E)

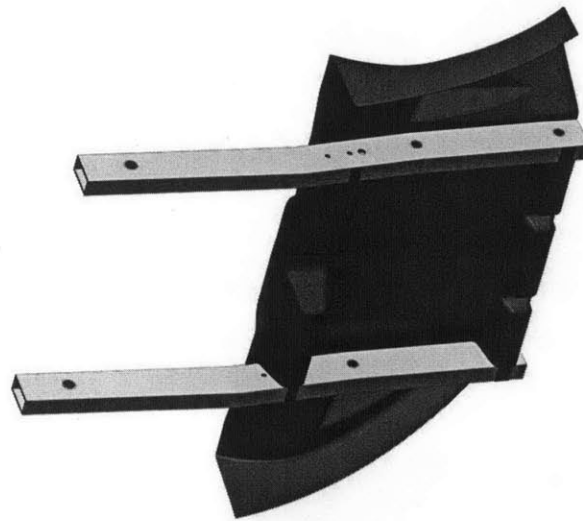


Figure 3-7: CAD - The 2GHS main structural frame, as viewed from below

The component parts of the main structure are fastened together using machine screws, the positions of which were determined by placing the various sub-structures in a jig to ensure dimensional accuracy. The total weight of the main structure comes in at 2.16kg, which when compared to the 1st generation 1/2 scale represents a 54.4% reduction in mass, whilst being optimized to bear anticipated loading factors during normal vehicle operation.

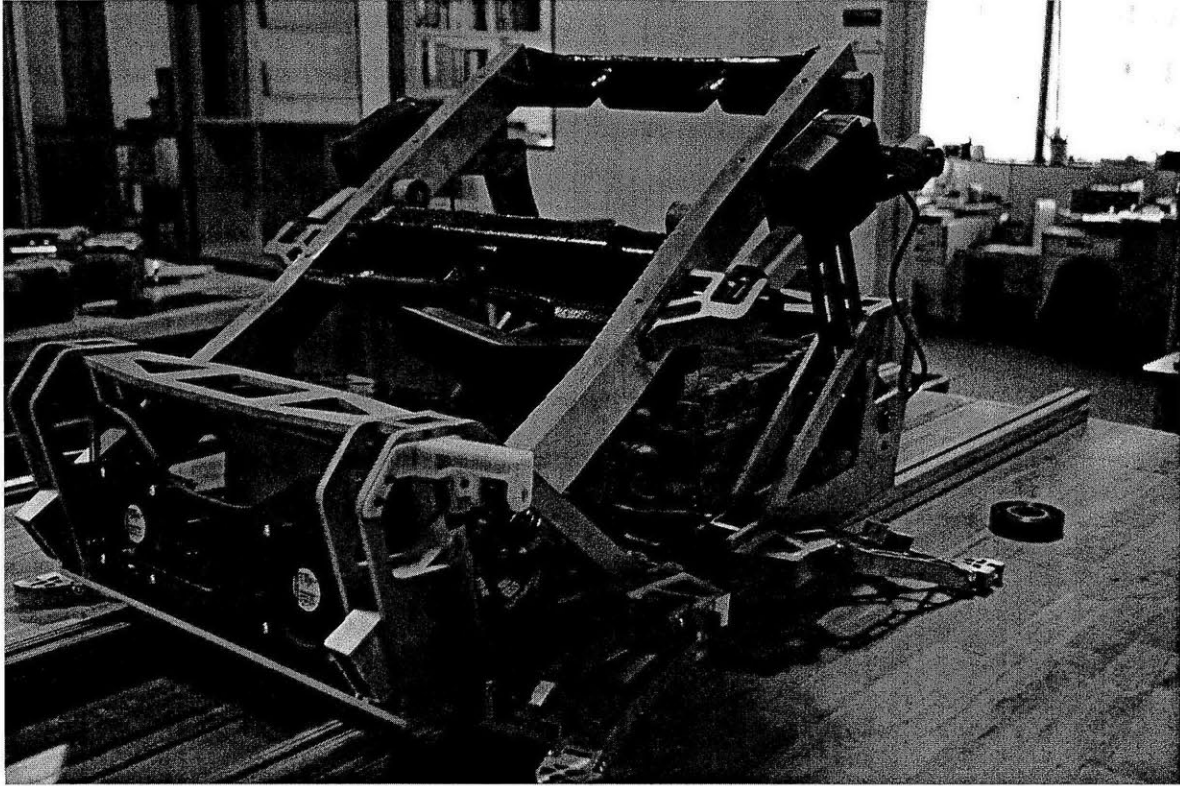


Figure 3-8: The 2GHS platform, with both underbody CFRP pieces top center

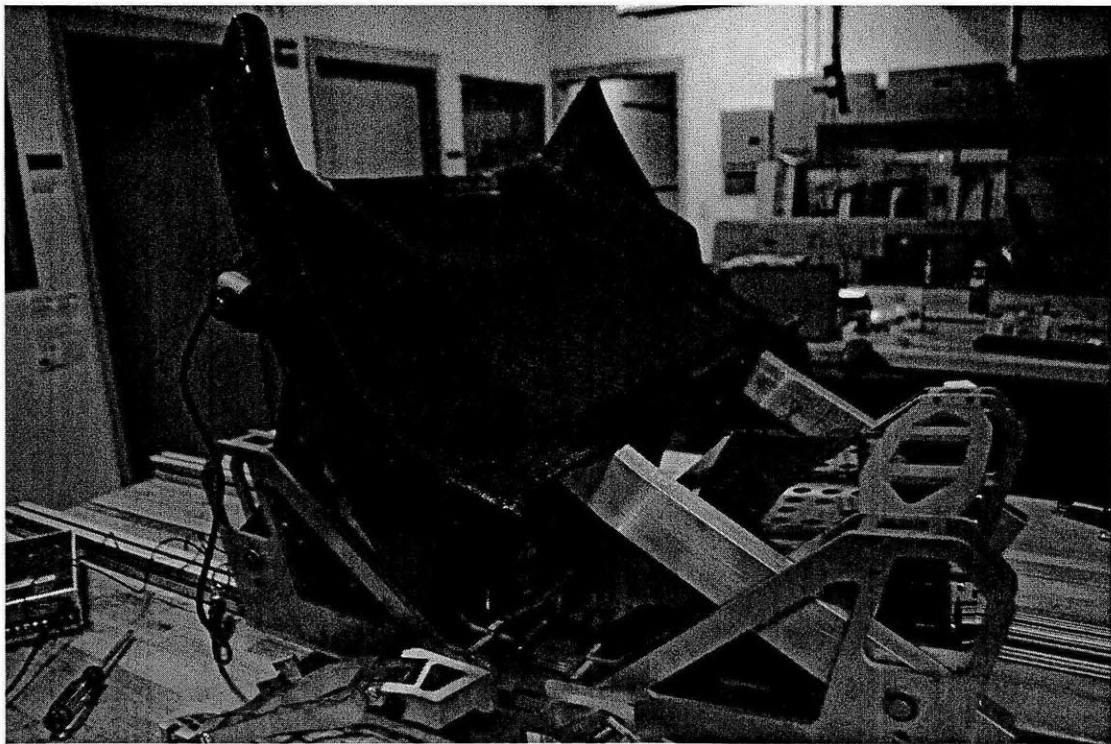


Figure 3-9: CAD - The 2GHS platform, with the top CFRP attached

[3.4] Powertrain Module

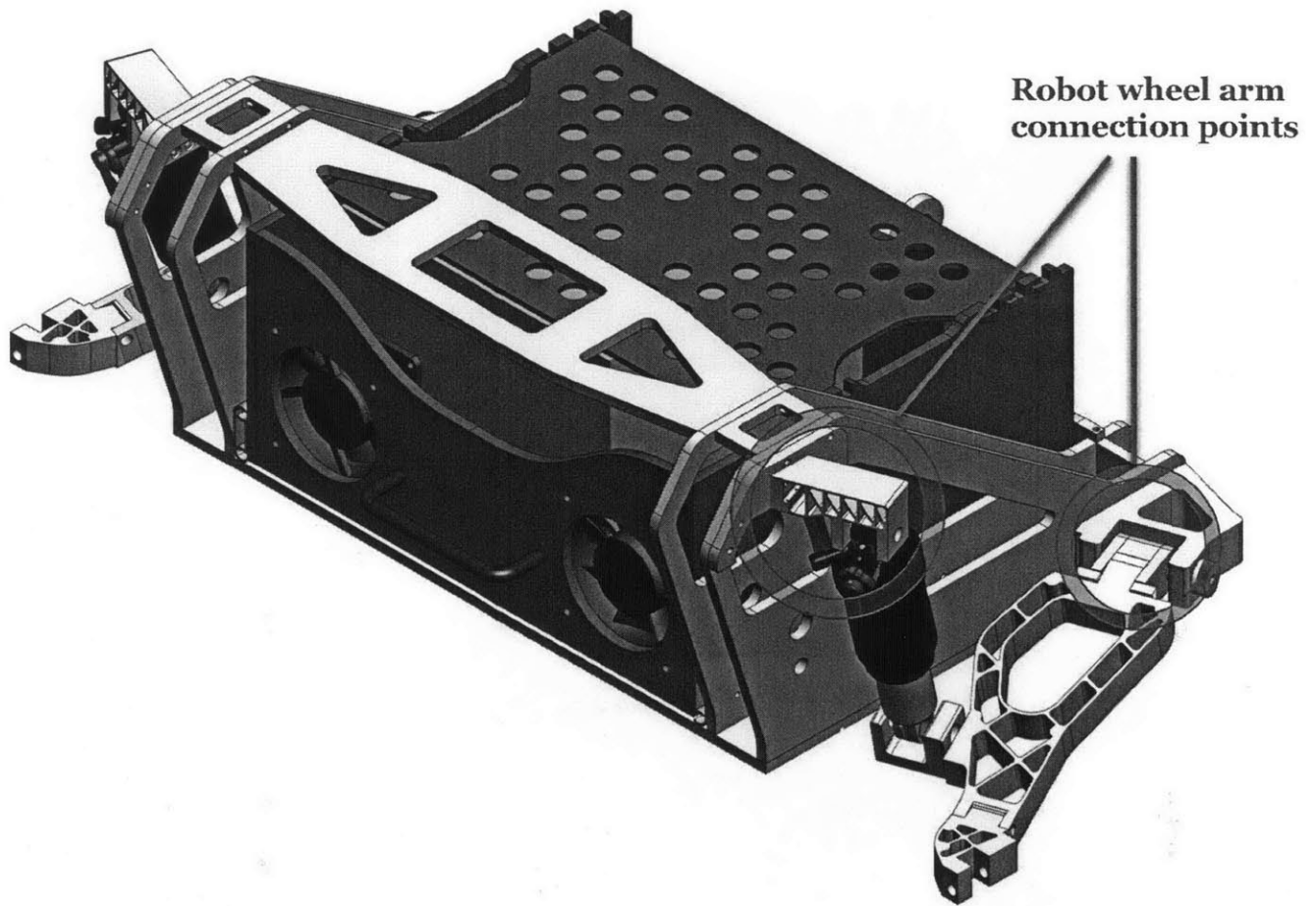


Figure 3-10: CAD - The 2GHS powertrain module, with some robot wheel components attached

As the main structural frame is the mechanical backbone of the 2GHS, the powertrain modules can be thought of as both its torso, as well as the electronic heart. The 2GHS has two powertrain modules: one at the front of the vehicle and one at its rear. The powertrain modules are the mounting points for the robot wheel assemblies, houses the micro-controllers, batteries, & power distribution buses, as well as being two linkages in the folding structure. In the previous design iterations of the 1/2 scale vehicle, it was determined that the vehicle needed to be designed to allow for driving and steering in both the unfolded and folded

positions. Based on the design at the time, however, the front arms of the vehicle would be tilted at an angle such that attempts to steer the vehicle would collapse the entire wheel structures underneath the vehicle.



Figure 3-11: The dry 2GHS powertrain module with tray

To remedy this, the lead/trail arms connecting the robot wheels to the main chassis of the vehicle were instead attached to a structural element that pivoted on the main bar. It was then connected to the rest of the folding linkages via another link, called the synchro link, which maintains a close-to-perpendicular relationship between the steering axis of the front wheel assemblies and the ground throughout the folding process. As the design evolved, it was realized that both the front and back structural elements serving as the robot wheel attachment points could be a single repeated design. This saved manufacturing time, as CNC cut files were combined on metal plates of the same thickness, and the number of unique

finishing machining operations were be significantly reduced. As Figure 3-13 below shows, the only structural difference between the front and rear modules is that the front has a single pivot point in middle of its back wall, while the rear has two.

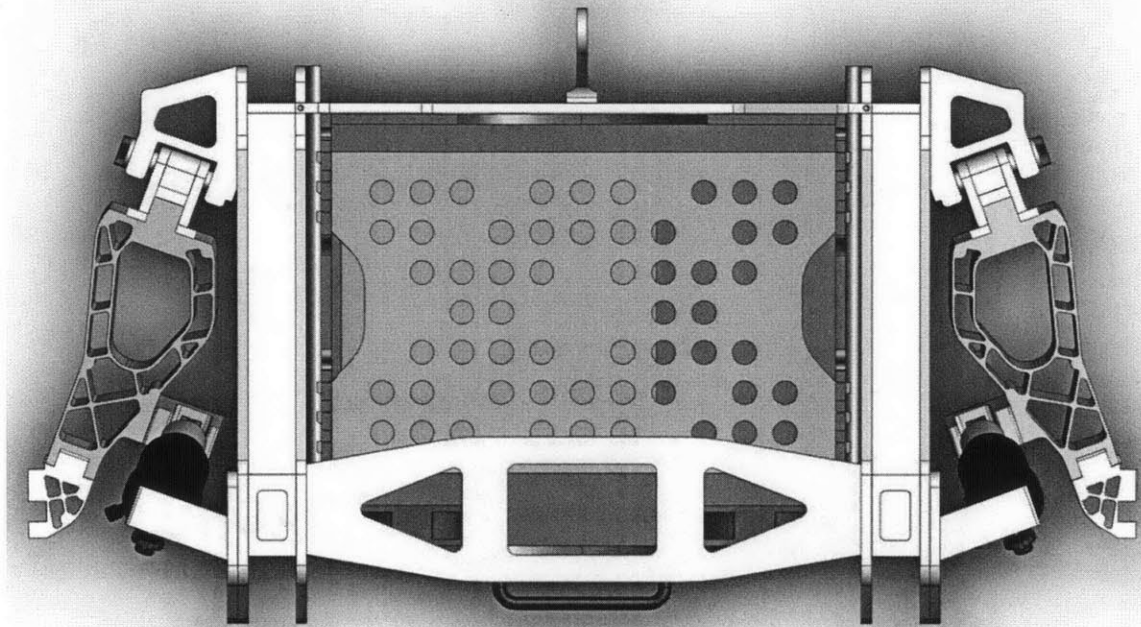


Figure 3-12: CAD - The 2GHS powertrain module, with some robot wheel components attached, as viewed from top

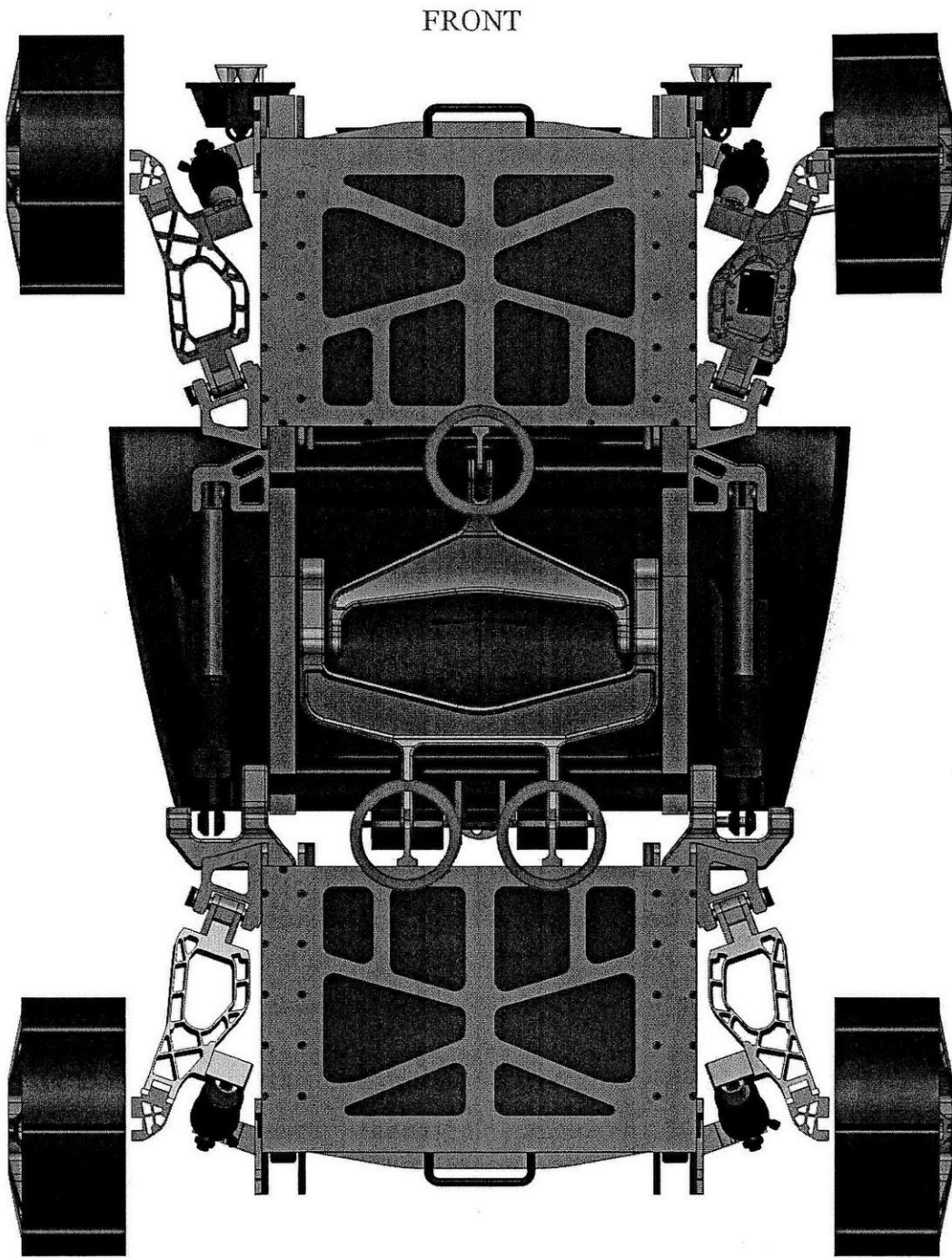


Figure 3-13: CAD - The 2GHS platform showing structural difference between front and rear powertrain modules, as viewed from below

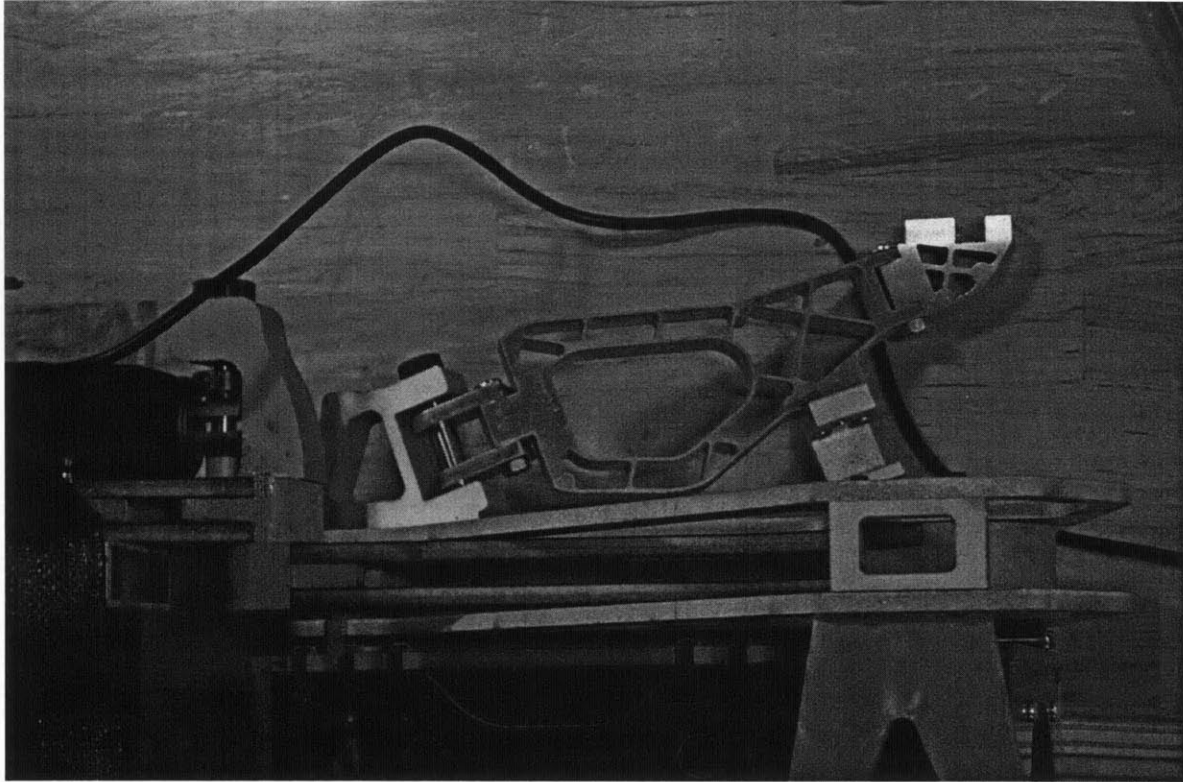


Figure 3-14: The lead/trail arm attached to the powertrain module, showing the angular offset

The powertrain module makes extensive use of FEA in order to reduce its weight as much as possible. Entirely constructed out of T6-6061 aluminum alloy, each module has a dry mass of 6.92kg. As shown in Figure 3-10 the powertrain modules have two mechanical mounting points of a robot wheel module. Each robot wheel rotates about a pivot point that is 20° offset from being perpendicular to the side of the powertrain. The upper mounting point had to be designed in such a way so as to allow for the normal compression and extension of the gas suspension. While the design was verified digitally in the CAD process, it was important to also verify the free motion of the suspension and lead/trail arm on the physical model. Figure 3-15 shows an early iteration of the upper connection point that was 3D printed in ABS plastic to quickly perform this verification. Dimensionally, the design worked as expected, however some modifications were made. FEA was used to arrive at the

currently used design, which is a lattice structure with elements as thin as 1mm. Figure 3-16 shows this final design execution, with suspension element attached.



Figure 3-15: 3D printed ABS plastic suspension upper connection point

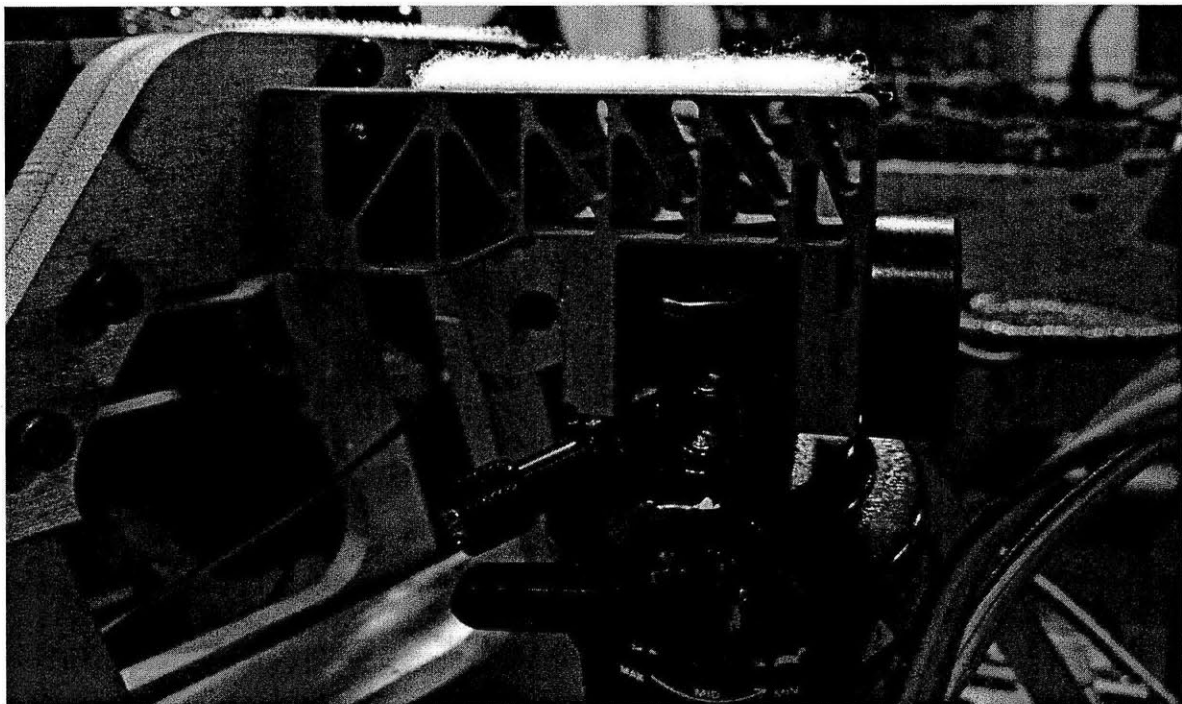


Figure 3-16: FEA optimized 6061-T6 suspension upper connection point

The powertrain module houses a polycarbonate tray, mounted to the module on removable drawer slides, and held in place with magnetic latches. This tray was designed to house the majority of the vehicle's electronic systems, and its power source and distribution buses. The tray is fitted with two 26.5cfm exhaust fans to aid in the movement of stagnant hot air. The top of the tray is also removable, which in conjunction with the drawer slides simplifies the process of working on the tray's internal components. The front and rear trays are responsible for different vehicle functions, and as such, have unique internal elements.

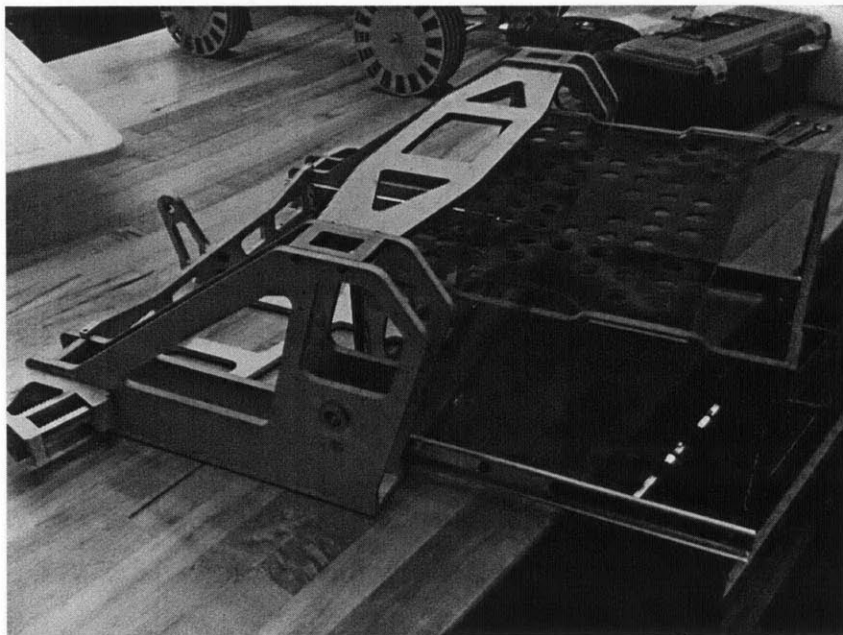


Figure 3-17: Powertrain module with tray pulled out on drawer slides

The front tray contains:

- 1 microcontroller IO box, with 2 Arduino microcontrollers
 - Primary front micro – Arduino Mega 2560
 - Pupil controller micro – Arduino Uno
- 1 K2 Energy *K2B24V10EB* 24V 10Ah Lithium Iron Phosphate Battery
- 1 5VDC bus
- 1 12VDC bus
- 1 24VDC bus

The rear tray contains:

- 1 microcontroller IO box, with 1 Arduino microcontroller and 1 relay board
 - Primary rear micro – Arduino Mega 1280
 - Folding actuator bidirectional relay switch
- 1 K2 Energy *K2B24V10EB* 24V 10Ah Lithium Iron Phosphate Battery
- 1 DC-DC converter board, with 5v and 12v outputs
 - Cosel *CBS502412* 12V 4.2A output
 - Cosel *SFS302405* 5V 6.0A output
- 1 5VDC bus
- 1 12VDC bus
- 1 24VDC bus

A power umbilical cord connects the front and rear power buses, with the 5V and 12V lines being supplied entirely from the rear DC-DC converter board, and tying the front and rear 24V K2 batteries in parallel. Both trays also house a 30A power switch, and a Turnigy 130A Precision Watt Meter and Power Analyzer to easily see the average power being drawn from each tray, especially useful during initial tabletop system tests. Further details on the microcontroller board and shield stacks are covered in Section 4.1.

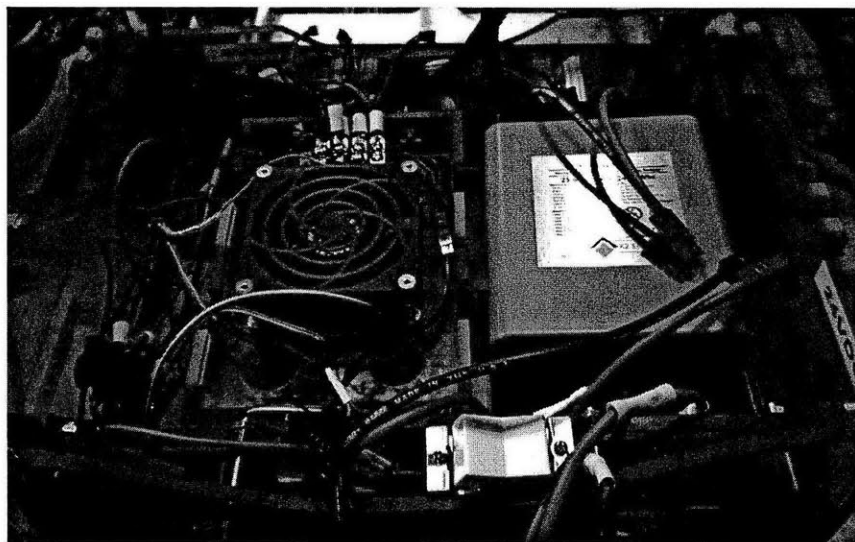


Figure 3-18: Internals of the front powertrain module's tray. At right, IO Box

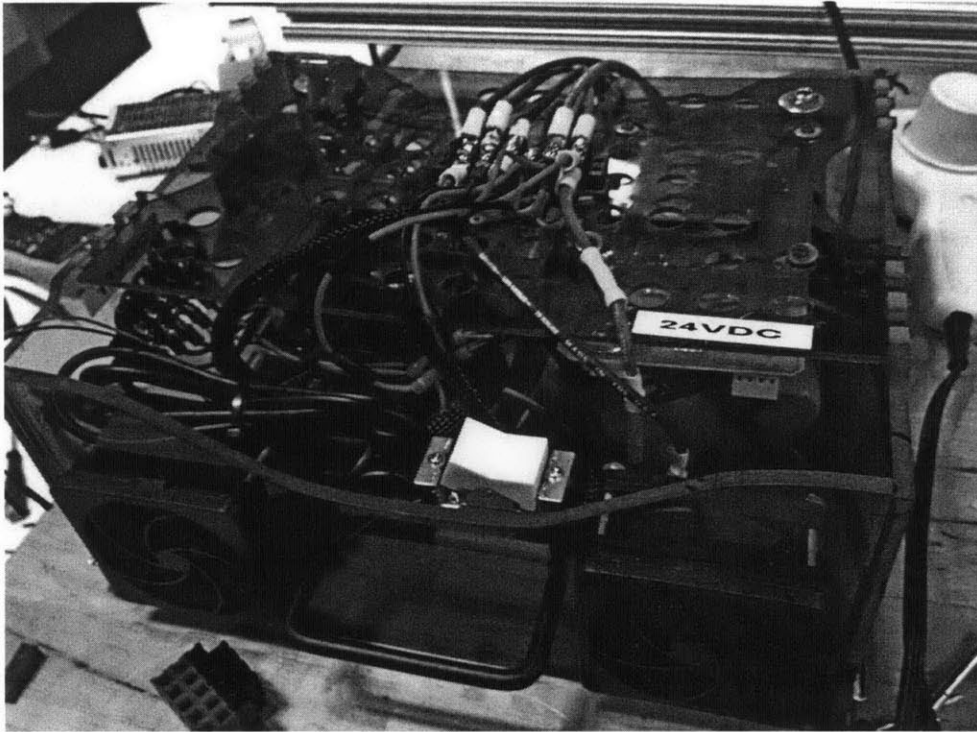


Figure 3-19: The completed rear powertrain module's tray

Each IO box was created in order to consolidate the wiring required to tie the various motor controllers, sensors and actuators to the Arduino microcontroller network. The IO boxes utilize 9-pin D-subminiature (DE-9) connectors to interface with these various systems, with the front box requiring three connectors, and the rear box requiring two. Pin IO schematics can be viewed in Appendix 9-D. The IO box is also fitted with an intake fan, primarily to aid in moving air over the voltage regulators built into each microcontroller. Prolonged use of the system saw the microcontrollers exhibiting behavior akin to overheating (random resets, hardware chip lockups), so heat sinks were attached to each voltage regulator, and the fans added as an extra precaution.

[3.5] Folding Linkages

The unfolded platform has wheelbase of 944mm, and a track width of 800mm. When folded, the wheelbase reduces 36.8%, to 597mm, with no change to the previously defined track width. This favors comparably to the target 30% reduction the full-scale CityCar achieves. In the patent filing *Dual Four-bar Linkage System for Folding Vehicle Chassis (Lark, Pennycooke)*, current designs allow for a maximum of 40% reduction in wheelbase, with the mechanism and geometry chosen for the 2GHS existing on the upper end of that spectrum. There are 7 main elements in the folding linkage structure, shown in Figure 3-20 below.

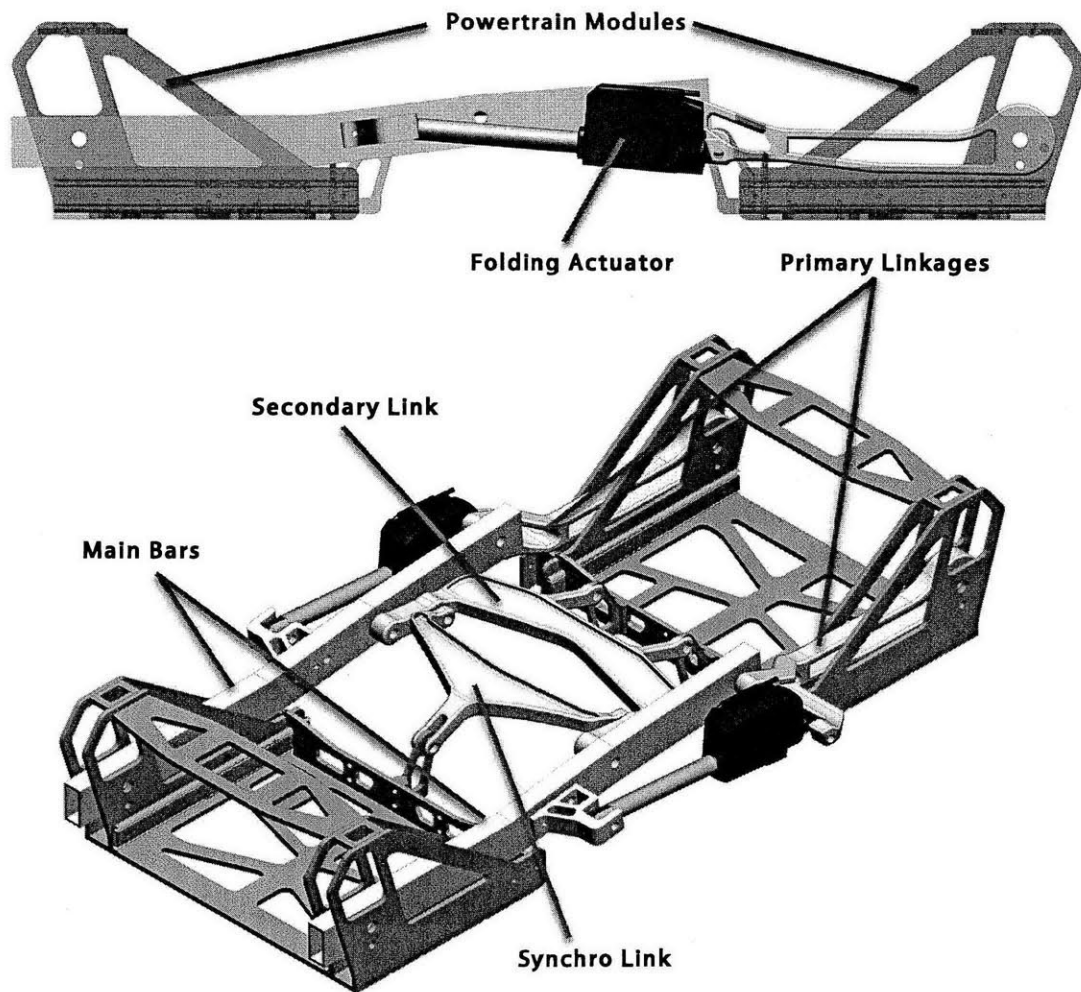


Figure 3-20: CAD - The 2GHS folding linkage main elements

The total linkage system can be best described as a hybrid front 4-bar linkage and rear (4+1)-bar linkage. The front of the vehicle's 4-bar linkage is made up of the front powertrain, the main bars of the main structural frame, the synchro link, and the secondary link. The rear 4-bar linkage's elements are the rear powertrain, the primary linkages, the main bars, and the secondary link. The "+1" refers to the folding actuators between the primary linkage and the main bar. The actuators are fully extended when the chassis is unfolded, and literally pulls the system together to complete a fold.

Both linkage systems share the secondary linkage as a fourth element, thus tying all motion enacted upon the rear linkage system to the front through the synchro link. As was stated in Section 3.4 on p.31, this synchronized motion allows for the powertrain modules to remain relatively parallel to the ground throughout the folding process, enabling full driving and steering in both the folded and unfolded positions. A second benefit of this arrangement is that the majority of the vehicle's weight (the powertrains) stays low, while folding, resulting in little increase in the height of the center of mass. Figure 3-26 demonstrates the full folding sequence of the chassis.

It should be noted that the manufactured design of each of the linkages is different in appearance to those portrayed in the figures in this Section. With regards to kinematics, the manufactured linkages perform exactly as the displayed design, however due to several limiting factors including cost and turnover time, their method of manufacture had to be changed. All linkages were originally designed to be lost-wax or investment casted in 356-T6 Al, but were redone so that they could be assembled from several machined planar pieces.

[3.5.1] Primary Linkages

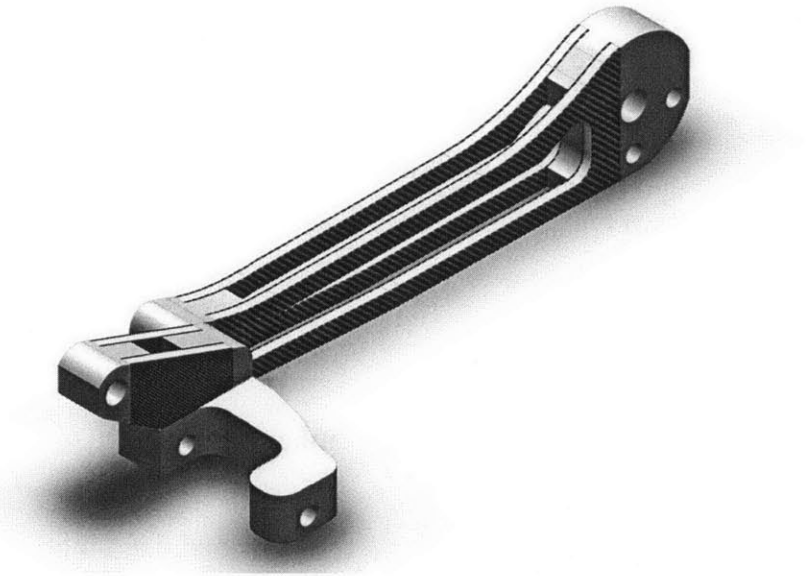


Figure 3-21: CAD - The 2GHS primary linkage

The primary linkage acts as a functional mirrored pair, connecting the rear powertrain module to the rear most pivot point of the main structural frame. It also serves as the rear connection point for the folding actuators. During the folding process, the folding actuators pull on the primary linkages, and because of the linkage geometry, they lift the main structural frame up at its rear. Because the mass being lifted is relatively small, liberties could be taken with the construction of the primary linkages. Each primary linkage uses 6061-T6 Al for all bushing/pivot points, connecting them with high-density foam sandwiched with CFRP. The assembly is fastened using a combination of machine screws and high-strength, high peel-resistant epoxy.

This method of construction reduces the weight of the linkage, and also creates a virtual crumple zone. The CFRP sections will fail before irreparable warping to the other folding elements occurs if one of the actuators fails, or if any type of rear impact.

[3.5.2] Folding Actuators

The two folding actuators are the animators of the entire dynamic chassis system. They behave as a 5th linkage between the main bars and the primary linkage. Each actuator is a Linak LA23 non-back-drivable linear actuator, with 100mm of stroke, 1200N of force in pull, and approximately 700N of static holding force²⁸. Non-back-drivable actuators were intentionally sought out so as to eliminate power draw by the actuators should the folding process pause between its normal binary states. This is important for both the energy efficiency of the system, as well as the maintenance of safety, should there be a power failure during a folding operation. The actuators also feature a safety nut to prevent catastrophic failure during pull operations.

In previous designs of the folding mechanism, specifically on the 1st generation prototype, the actuators were designed to extend to bring the vehicle from an unfolded to a folded state. However, this was problematic as it constrained the packaging and geometry of the entire chassis since the actuator had to be mounted to at least one preexisting linkage pivot point. By changing to a retraction based folding system, the size (width) of the actuator is no longer confined to a very small space as it can be moved to outside of the main structural frame, and by not limiting the design of the mounts to a point between two existing pivot points, a vast array of stroke lengths can be designed. As built the folding actuators are mounted between the primary linkages and the main bars.

²⁸ LINAK. (2010). Product Data Sheet Actuator LA23. Retrieved June 29th, 2012 from LINAK: http://www.linak.com/corporate/pdf/ENGLISH/DATA%20SHEET/Linear%20Actuator_LA23_Data%20Sheet_Eng.pdf

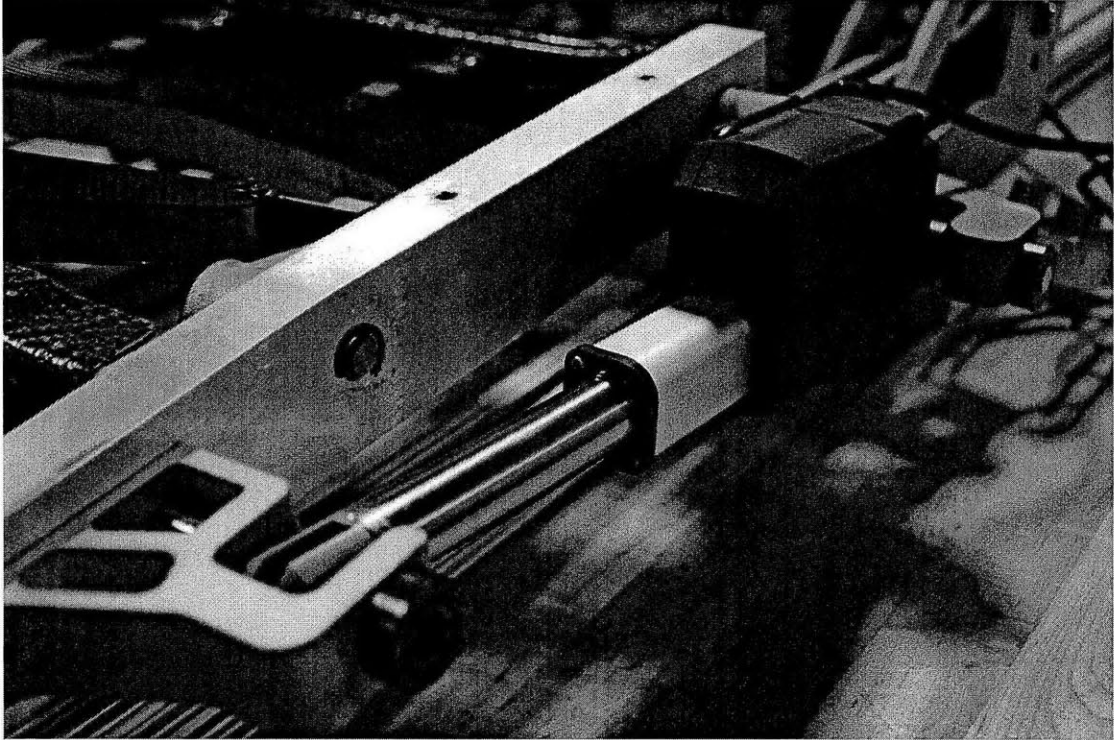


Figure 3-22: The Linak LA23 actuator, extended

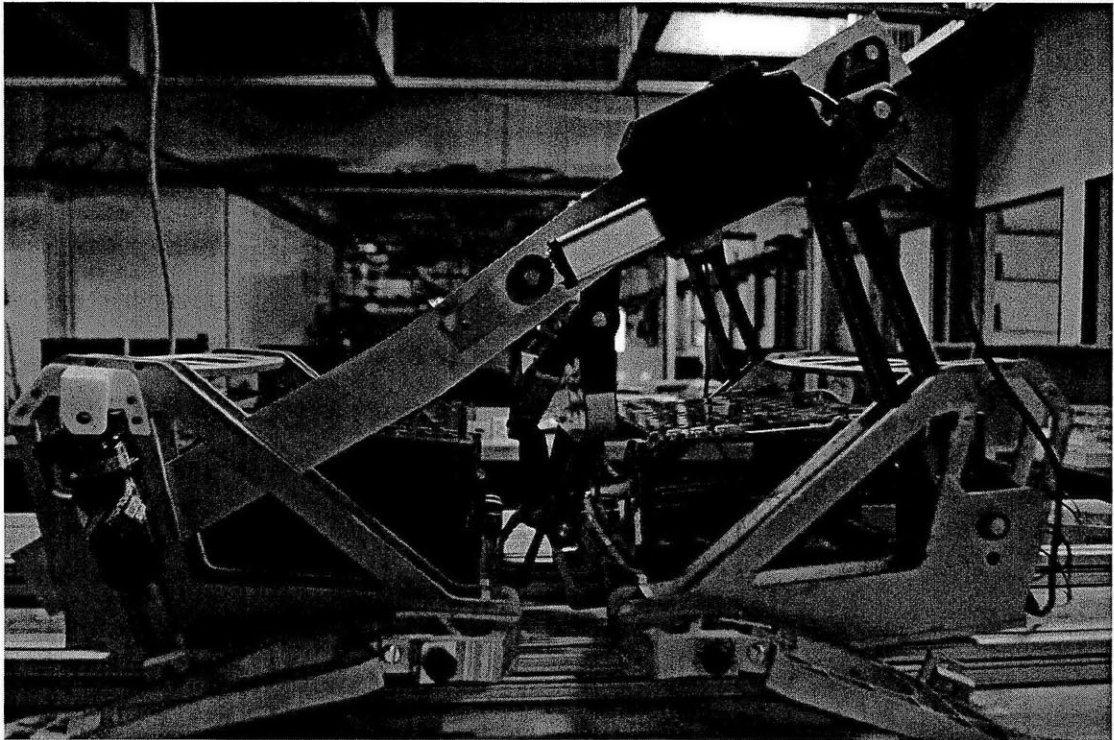


Figure 3-24: The Linak LA23 actuator, retracted

[3.5.3] Secondary And Synchro Link

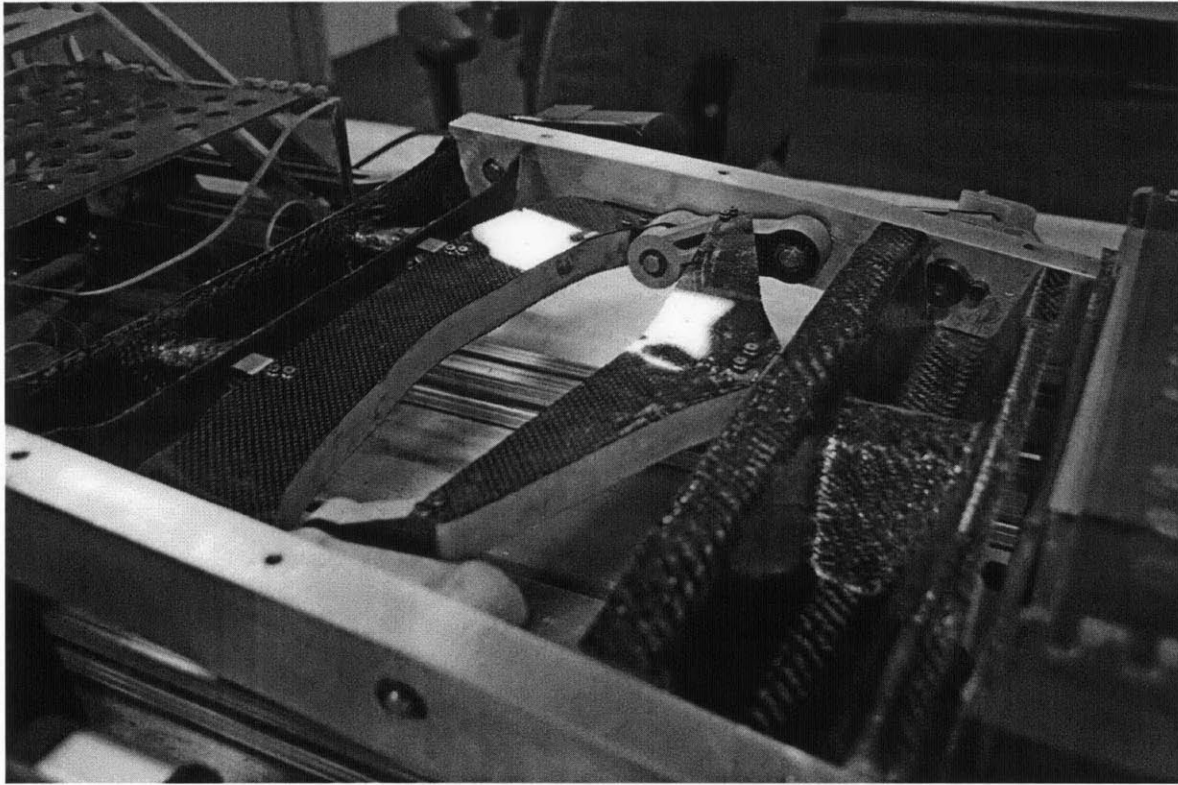


Figure 3-25: The secondary and synchro links

The secondary link connects the rear powertrain module to the main bars, and the synchro link connects the front powertrain to the secondary link. Both elements underwent extensive motion analysis FEA to create the planar assemblies required to replicate the functionality of the originally to-be-casted pieces. With CFRP plates covering 6061-T6 Al lattice structures, each piece is geometrically optimized to handle expected static and dynamic loading conditions throughout the entire folding sequence. Matching the pivot points available on the front and rear powertrain modules, the synchro link has one pivot point interfacing with it, and the secondary link has two. This allows the linkage pivot points to overlap laterally, thus enabling the wheelbase reduction previously described.

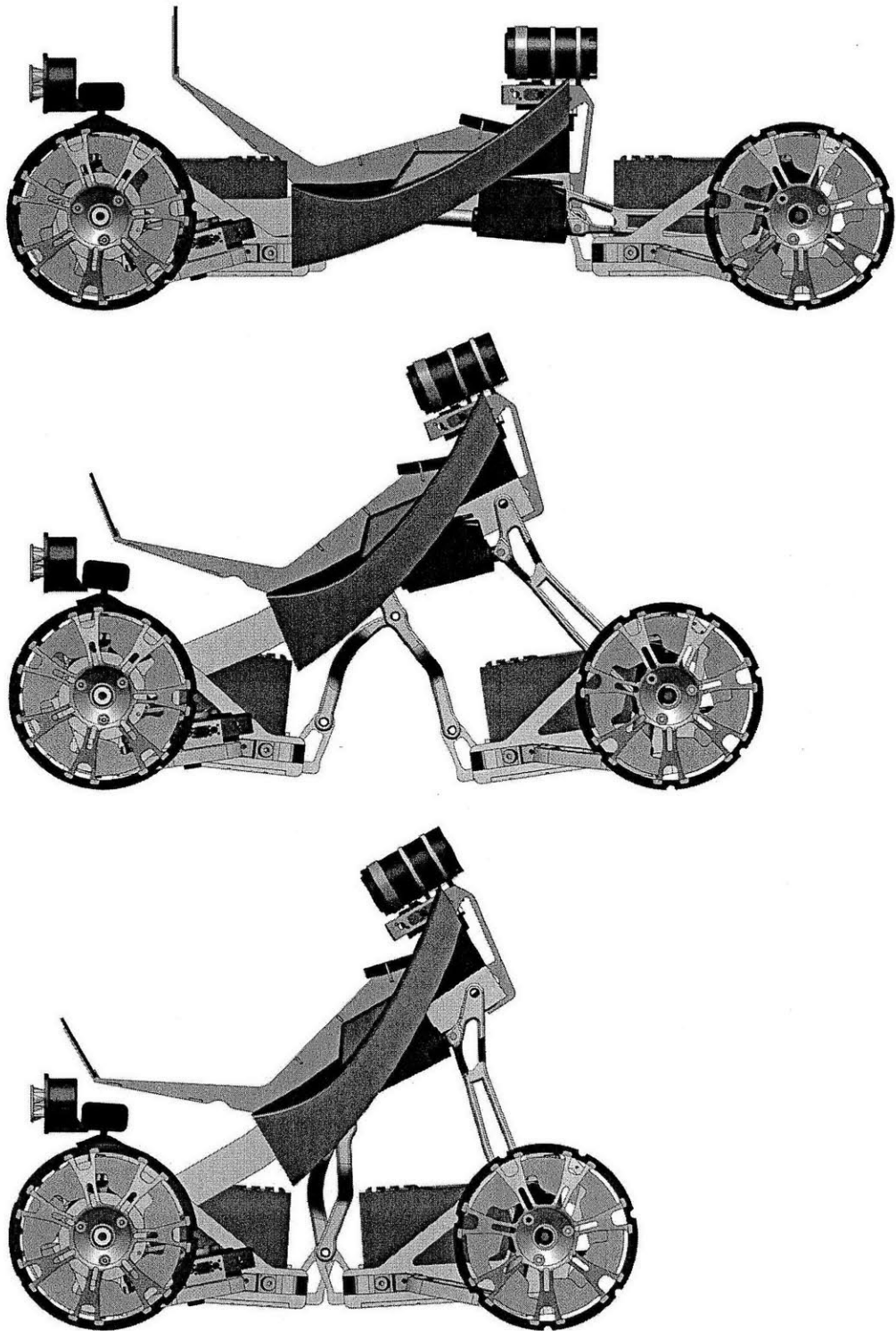


Figure 3-26: CAD - The 2GHS folding sequence, showing the lift of the main structural frame, and the overlap of the front and rear powertrain pivot points

[3.6] Robot Wheel Module

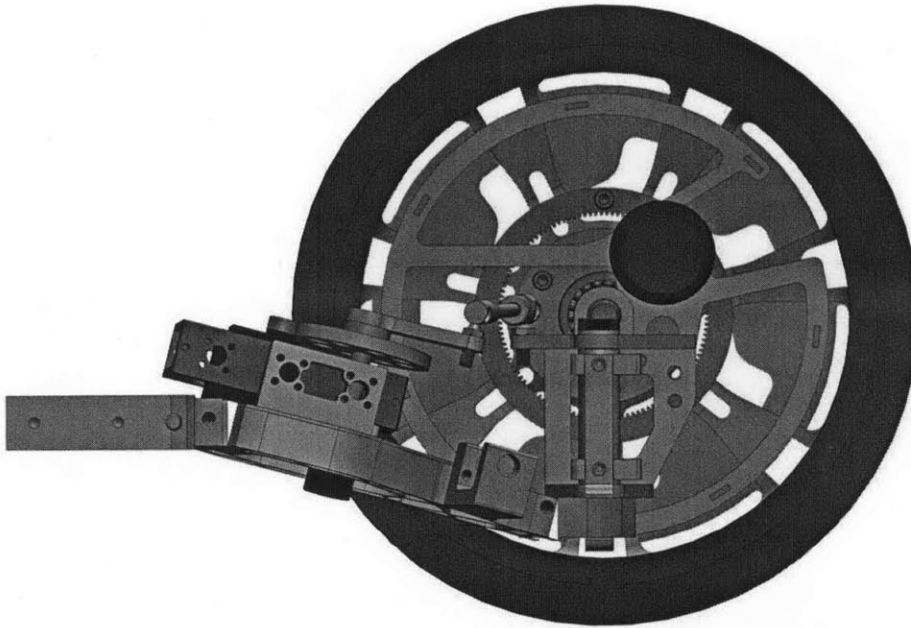


Figure 3-27: CAD - The 2GHS front-left/rear-right robot wheel module, without suspension

The robot wheels are what make this dynamic robot chassis into a fully functional vehicle. The concept of the robot wheel simply requires the consolidation of a vehicle's traditional drivetrain and steering mechanisms into the corners of the vehicle, forming a repeated module that requires a simple mechanical, power, and data connection. Because of this flexibility, over the past several years, there have been many iterations of the design employed on the various prototypes created within the research group. As designed, the front-left, and rear-right robot wheels are exact copies of each other, while the front-right and rear-left are mirror copies.

The robot wheels used on the 2GHS are designed to operate with a maximum total steering sweep angle of 54° . This allows for traditional steering angles of $\pm 15^\circ$ about dead ahead, as well moving the wheels into an 'o-turn' position tangential to the center circle described by the folded and unfolded wheelbases and track.

[3.6.1] Arm And Suspension

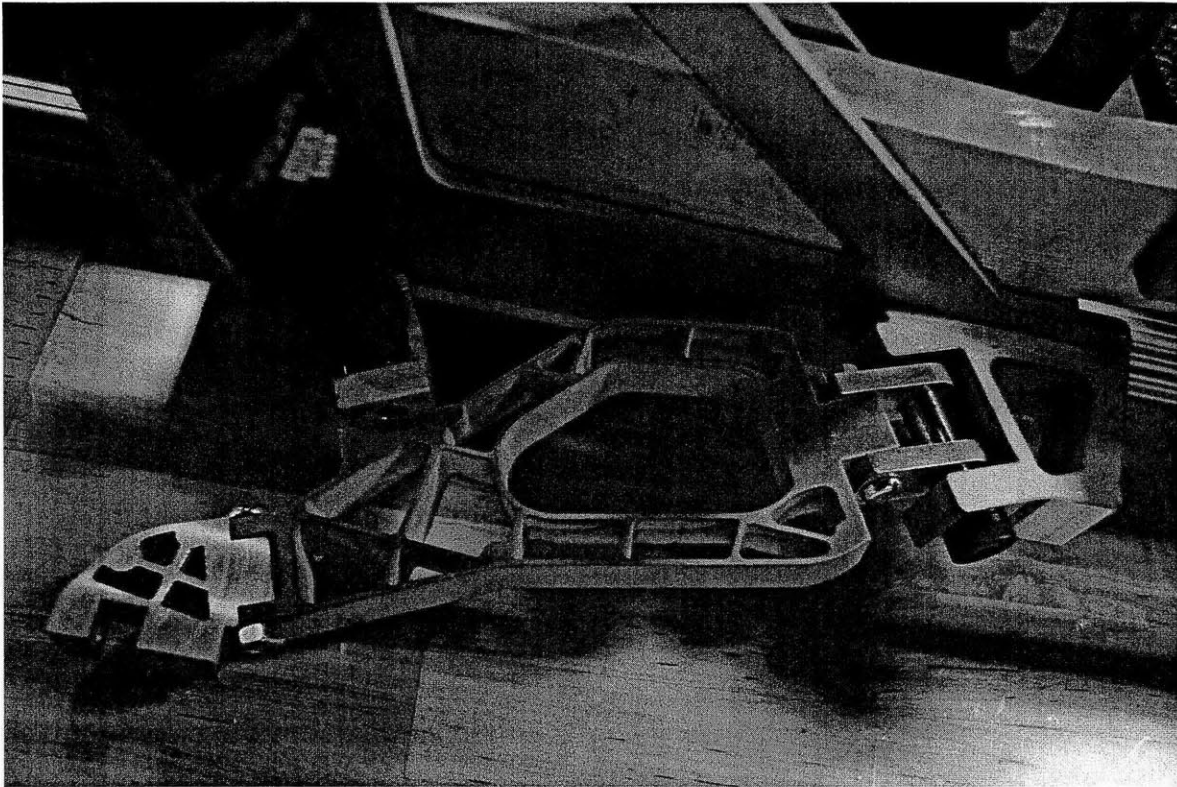


Figure 3-28: The 2GHS front-left/rear-right arm and suspension structure

The lead/trail arm design of the lower suspension arm of the robot wheel module was chosen for several reasons. It does not impede on the large sweep space required by the wheel and tire for the vehicle to be able to perform an O-turn. Secondly, it has very simple mechanical connection to the chassis of the vehicle, requiring the removal of only two pivot point shafts for the total removal of the module. Lastly, because much of the top surface area of the arm is not needed to mechanically support the robot wheel, this space can be utilized to house the steering mechanism and control electronics.

Constructed of 6061-T6 Al, the design of the arm, like much of the rest of the vehicle, takes care to minimize its overall weight. Perpendicularly arranged 2mm thick support framework are aligned to the expected direction of most loads the

wheel will encounter, i.e. running parallel and orthogonal to the axis of symmetry of the chassis. The large gap in the middle of the structure facilitates the mounting of the steering actuator, and as mentioned in the powertrain module, the gas spring suspension connects the angularly offset arm to the rest of the chassis. The step-down design of the arm also brings the axis of rotation of the wheel assembly concentrically aligned with the pivot point in the powertrain module connecting the rear to the primary linkages, and the front to the main bars. This is advantageous as it eliminates errant moment forces introduced during folding when the vehicle rolls on its wheels, while pivoting around those two main axes.

[3.6.2] Steering



Figure 3-29: The 2GHS robot wheel steering mechanism

Previous designs of the robot wheel incorporated gear trains, timing belts, and direct non-back-drivable motor assemblies. However, all of these methods required external position encoding, which complicated both the mechanical assembly as well as the circuitry and control code required to adequately manage steering all four corners. In addition, none of the above strategies had a form of absolute positioning, nor were they easily tuned, adjusted or serviced.

To remedy these problems, a servo-based solution was chosen for its simple interfacing to the control infrastructure built on the vehicle, as well as its intrinsic absolute positioning. This means that the system always knows where the wheel is, and in the event of a power failure, will not reinitialize the current position as dead ahead, a problem regularly encountered with the above mentioned strategies.

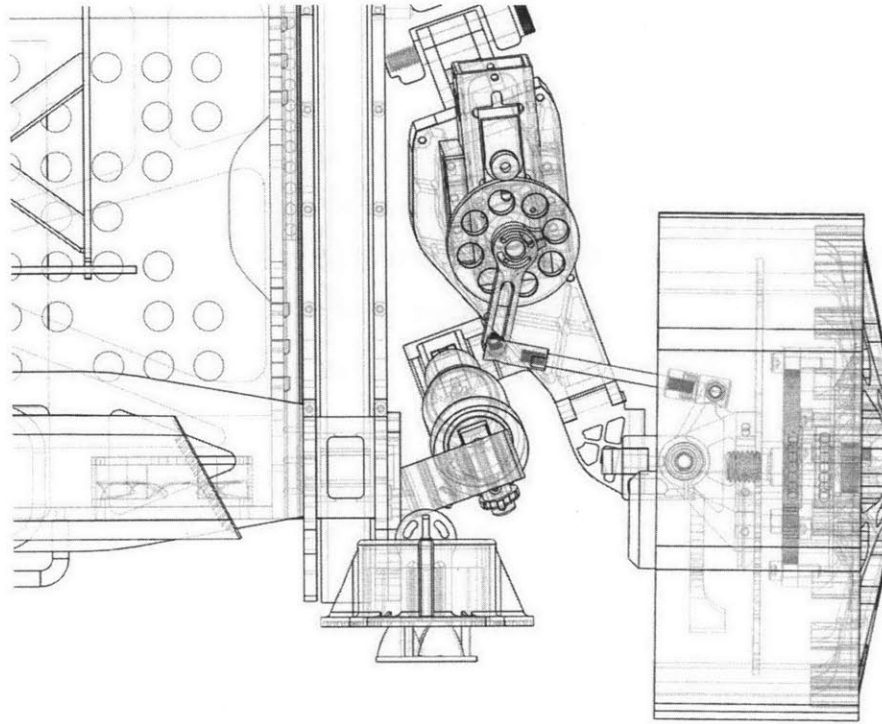


Figure 3-30: CAD - Details of the robot wheel architecture and the steering mechanism

The servo used is a HiTec HS-7955TG digital servo meshed with a metal gear for a 5:1 speed reduction. Operating at 6V supplied by a Battery Eliminator Circuit (BEC) built into each of the motor controllers discussed in Section 3.6.5, this combination has a maximum rotational speed of 60° in 0.65s (92 deg/s or 1.6 rad/s) and a maximum holding torque of 11.8Nm. At stall, each draws a maximum of 4.2A. Attached to the larger output gear is an arm that can be rotationally adjusted and tuned to either correct steering misalignment, or add toe-in/out to the vehicle. Connecting the steering C-bracket to the servo and servo arm is a threaded rod with ball joint linkages at both ends. The angular difference between the top plane of the C-bracket, which is parallel to the ground, and that of the servo arm necessitates using the linkage end types. The steering angle and response can thus be tuned by adjusting the threading into each linkage end, or by moving the position of the linkage end that mounts to the slot in the servo arm. The C-bracket rotates about a bushing-lined tube, held to the robot wheel arm with a 7075-T651 Al upright.

[3.6.3] Steering Control

The control of the steering servo is done directly from the nearest primary Arduino microcontroller, front or rear. The servos take a Pulse Position Modulation (PPM) signal, directly writing an angle (θ) in the range of $0^\circ \leq \theta \leq 180^\circ$ through the built in Arduino Servo library.²⁹ Due to the geometric relation of the linkages between the axis of rotation of the servo and that of the actual steering bracket, a change in angle of 1° of the servo does not correlate to an equal change for the steering bracket. Hence, an attempt to find mathematical relationship between the two angles was made, but was found to be massively complex.

In order to create a steering algorithm, the CAD file was used to manually rotate the steering servo in increments of 1° , starting at a steering angle of -15° , and continuing through to the O-turn angle of $+52.4^\circ$. The results of this study can be found in Appendix 9-C. What was immediately observed was that between $\pm 15^\circ$, the relationship between the servo angle change and steering bracket change was relatively linear with a corresponding servo range of $\pm 11^\circ$ centered on its 90° position.

The steering algorithm could then be greatly simplified, as continuous control between $+15^\circ$ and $+52.4^\circ$ was not needed for any normal steering operation. Thus, it was possible to create a linear control equation, and a separate discrete O-turn function that hard writes the necessary angle to the four corner units. The equations take into consideration a virtual application of the Ackermann steering geometric principle of slightly different steering angle between the left and right sides of the vehicle as they trace circles of different radii.

²⁹ There is significant debate amongst users whether or not the terminology Pulse Width Modulation (PWM) or PPM should be used to describe the control signal a servo receives. However, for the purposes of this thesis, the term PPM will be used as PWM is 'less correct' and not what the R/C industry uses, which will be important Section 3.6.5 describing the drive controller. Further information on this discussion can be found here <http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1253149521/all>

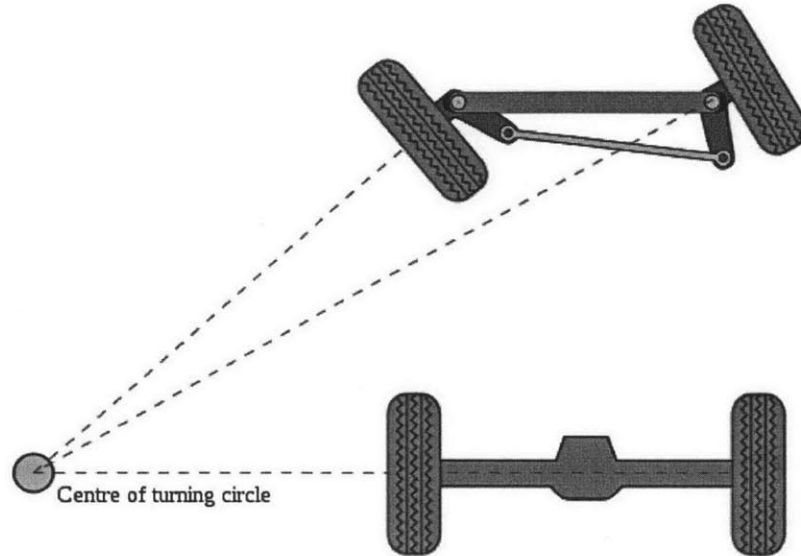


Figure 3-31: Diagrammatic representation of the Ackermann steering geometry³⁰

Figure 3-31 above demonstrates this principle. However, this only demonstrates 2-wheel steering. Because the 2GHS has 4-wheel steering, it can achieve a sharper (smaller) turning radius, and given that it is a virtual Ackermann implantation, the amount around the true Ackermann angle can be tuned without any physical modification to the vehicle. The equations governing the total steering control are as follows:

Normal Steering

When turning left:

$$A = 0.15(\text{adjSteering}) \quad \text{Eq (3-1)}$$

$$B = -\tan^{-1}\left(\frac{\text{wheelBase}}{2} / \left(\frac{\text{wheelBase}}{2} / \tan\left(\frac{-0.26(\text{adjSteering})}{100}\right) + \text{track}\right)\right) * \frac{180}{\pi} \quad \text{Eq (3-2)}$$

$$C = -0.15(\text{adjSteering}) \quad \text{Eq (3-3)}$$

$$D = \tan^{-1}\left(\frac{\text{wheelBase}}{2} / \left(\frac{\text{wheelBase}}{2} / \tan\left(\frac{-0.26(\text{adjSteering})}{100}\right) + \text{track}\right)\right) * \frac{180}{\pi} \quad \text{Eq (3-4)}$$

³⁰ Wikimedia. (November 28, 2006). Illustration of Ackermann Steering Geometry. Retrieved June 22nd, 2012 from Wikimedia: <http://commons.wikimedia.org/wiki/File:Ackermann.svg>

When turning right:

$$A = \tan^{-1} \left(\frac{\text{wheelBase}}{2} / \left(\frac{\text{wheelBase}}{2} / \tan \left(\frac{0.26(\text{adjSteering})}{100} \right) + \text{track} \right) \right) * \frac{180}{\pi} \quad \text{Eq (3-5)}$$

$$B = 0.15(\text{adjSteering}) \quad \text{Eq (3-6)}$$

$$C = -\tan^{-1} \left(\frac{\text{wheelBase}}{2} / \left(\frac{\text{wheelBase}}{2} / \tan \left(\frac{0.26(\text{adjSteering})}{100} \right) + \text{track} \right) \right) * \frac{180}{\pi} \quad \text{Eq (3-7)}$$

$$D = -0.15(\text{adjSteering}) \quad \text{Eq (3-8)}$$

When dead ahead:

$$A = B = C = D = 0 \quad \text{Eq (3-9)}$$

Servo angles:

$$\text{leftFrontSteer} = (0.75 * A) + 90 \quad \text{Eq (3-10)}$$

$$\text{rightFrontSteer} = (0.75 * B) + 90 \quad \text{Eq (3-11)}$$

$$\text{leftRearSteer} = (0.75 * C) + 90 \quad \text{Eq (3-12)}$$

$$\text{rightRearSteer} = (0.75 * D) + 90 \quad \text{Eq (3-13)}$$

O-turn Steering

Servo angles:

$$\text{leftFrontSteer} = \text{rightRearSteer} = 117 \quad \text{Eq (3-14)}$$

$$\text{rightFrontSteer} = \text{leftRearSteer} = 63 \quad \text{Eq (3-15)}$$

where *wheelBase* is the distance between the axes of the front and rear wheels, *track* is the distance between the symmetric centers of the wheels as viewed head-on to the vehicle, and *adjSteering* is the -100 to +100 steering output of the handheld vehicle controller. It should be noted that all steering calculations from the controller input are done in C# program running on the PC, and writes the desired servo angles directly to the microcontrollers.

[3.6.4] Drive

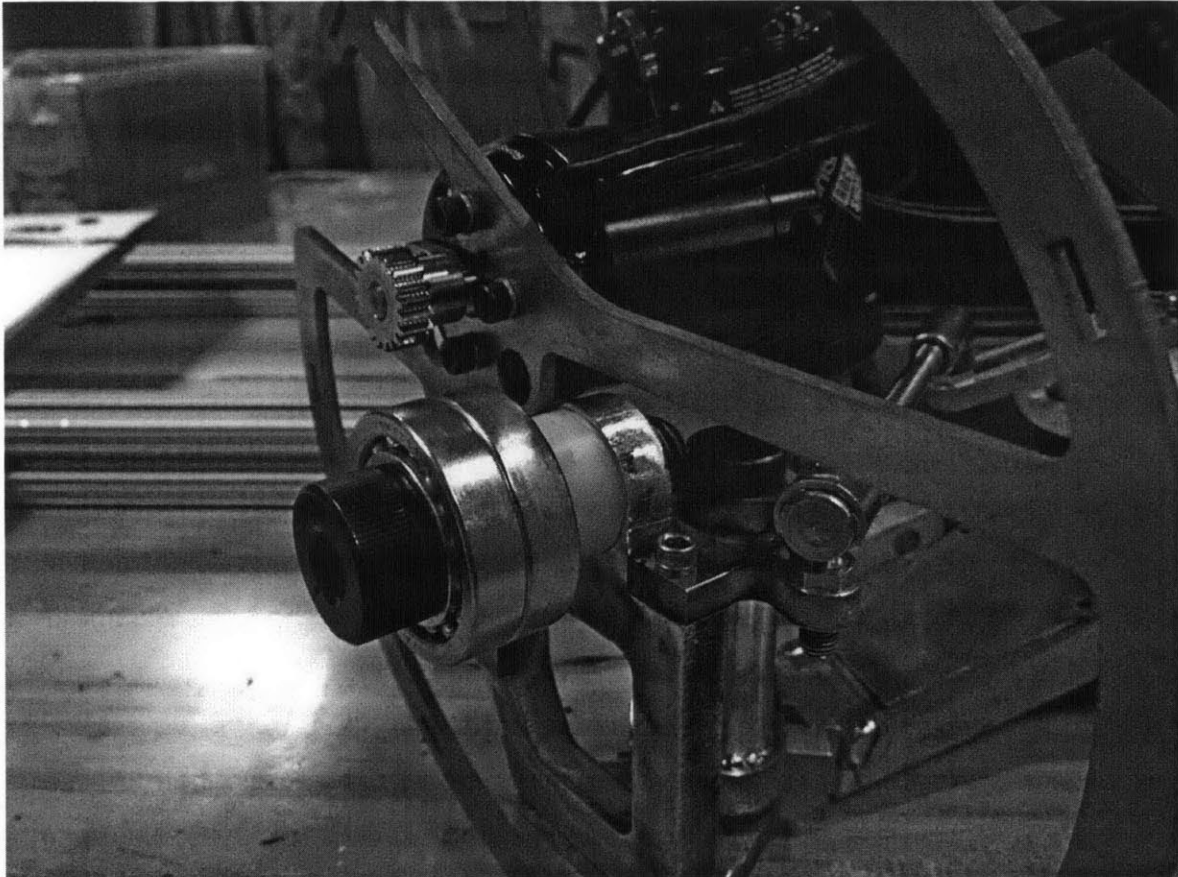


Figure 3-32: The 2GHS robot wheel drive motor and motor mount

When deciding on a drive motor, it was debated whether or not to use a direct drive hub motor, or some other arrangement using more conventional gear-motor combinations. Due to the lack of availability of suitable off-the-shelf hub motors at this scale, and avoiding a custom built and wound motor, the decision was made to use a small and light brushless DC (BLDC) outrunner motor, with as low a kv value as possible interfacing with the wheel through a gear reduction. The kv value of a BLDC motor tells how many RPMs the motor will spin at per Volt applied. The lower the kv, the lower the rotational speed, but the higher the torque. This is primarily due to the internal winding being able to handle more current, and the current draw of a motor is directly related to its torque output. A BLDC outrunner motor was also chosen for its lower possible kv value as opposed to a brushed

motor or an inrunner type. Considering packaging constraints and a drive power bus of 24V, the motor chosen was the 3-phase Hacker A50-14L 300kv. To its shaft, a steel 20T pinion gear was affixed. The motor assembly mounts to a plate that is fastened to the steering assembly's C-bracket. In this arrangement, the outside casing housing the magnets of the motor (purple casing in Figure 3-32 and Appendix 9-E) spins, thus reducing the rotational mass and inertia that needs to be overcome.

Figure 3-32 also shows two bearings mounted on a shoulder bolt. This constitutes part of the drive hub to which the wheel and tire assembly mounts. The hub assembly is a machined 6061-T6 cylinder with housings for the two bearings, counter-bored holes to affix lug bolts, as well as mounting holes to attach the main drive gear. The drive gear is a steel 120T internal gear, which when attached to the hub assembly and mounted to the C-bracket via the shoulder bolt, interfaces with the drive pinion for a 6:1 speed reduction and torque increase.



Figure 3-33: The 2GHS robot wheel hub assemblies, without lug bolts or bearings

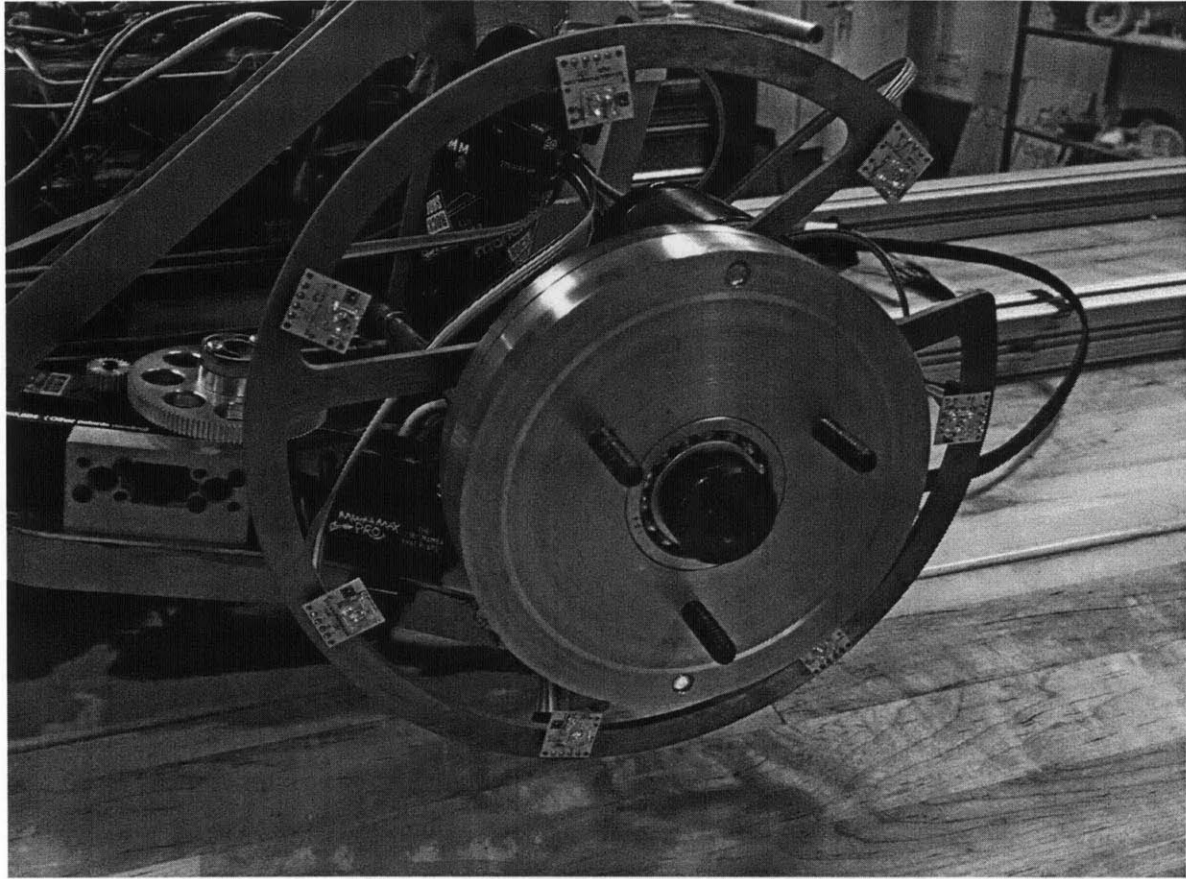


Figure 3-34: The 2GHS robot wheel hub assembly, mounted to the drive/steer assembly

[3.6.5] Drive Control

The motors being used to drive the 2GHS platform are purpose built model aircraft engines. Their low kv value and consequently, high turn count per tooth (i.e. for every laminated tooth inside the motor, there are a high number of turns of high current wire), means that these motors have a naturally high inductance. Typical scale aircraft motor electronic speed controllers (ESCs) are built to handle this, and so have no problem starting and maintaining rotation of the motor. However, aircraft ESCs can only rotate a motor in one direction for a given range of throttle inputs. To that end, a 1/10th scale radio-controlled car ESC was chosen to drive each robot wheel's drive motor, specifically the Castle Creation Mamba Max Pro (MMP) controller, as they are able to drive, brake, and reverse a motor over a single throttle input range. The 24V power bus supplies each MMP.

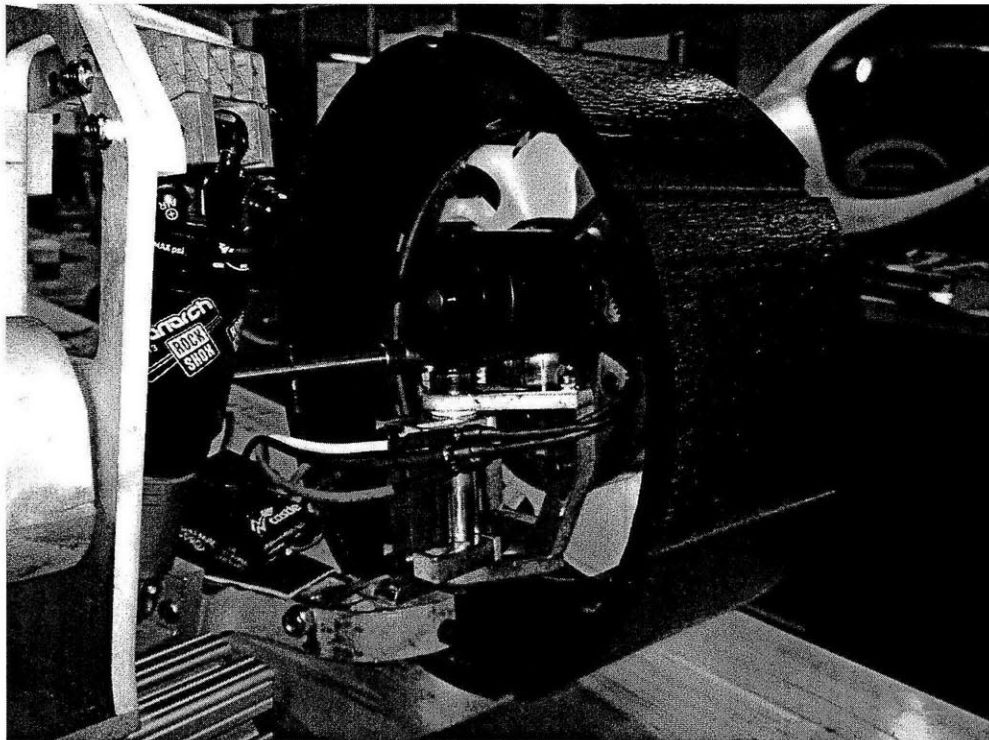


Figure 3-35: The 2GHS robot wheel assembly, Castle MMP ESC at bottom left

This controller was attractive for several reasons. Firstly, its small size and built in heat management meant that the ESC could be mounted on the lead/trail arm of each robot wheel, close to the motor. Secondly, and most importantly, was its high level of programmability. All Castle ESCs are USB programmable, allowing the fine-tuning of a motor response to throttle inputs. This controller was not meant to handle the high inductance of an outrunner motor, as they are usually coupled with very high kv inrunner motors for model radio controlled cars. However, by adjusting the throttle curve, setting the instantaneous starting power to 'high' and experimenting with different timings, a suitable ESC profile was built up to be able to control the chosen motors, with much greater than expected performance. A detailed printout of the final ESC settings can be found in Appendix 9-E. Lastly, the controller is able to run in both sensed and sensorless modes, meaning it could operate whether or not the motor has built in Hall Effect sensors used to measure internal magnetic phase switching and improve motor timing and smoothness. The BLDC motors here are not sensed, and though there is some cogging, or jerkiness, at initial motor spin up because the ESC is 'blindly guessing' which phases to fire, within ~1 second, the motors smooth out.

These ESCs are normally controlled via a wireless receiver built into the model vehicle, which takes throttle commands from a handheld transmitter. What was discovered was that the signal sent from the receiver to the ESC was a PPM signal: the same kind of output that an Arduino is capable of producing in order to control a servo. In order to verify this, a servo tester was hooked up to one of the ESCs, with external power connections, and the signal and power grounds tied. It was found that by setting the servo tester to 90° , the ESC would not drive the motors, setting it 0° drove the motors to full throttle in one direction, and 180° , full throttle in the opposite direction.

What this meant was that these ESCs, regardless of make, were fully controllable via Arduino and required only 1 pin to do so. By setting that pin to behave as a servo and writing an angle command to that pin, it was possible to intelligently control all four wheels throttles by writing values in the above stated ranges. In

normal drive operation, all four wheels drive in the same direction: 180° full forward, 0° full reverse, and 90° neutral. For an O-turn, where only diagonally paired corners drive in the same direction, it is simply a matter of reversing the throttle input (180 minus the throttle value) written to the applicable corners. The equation that manage the throttle values on the PC is written is,

$$throttle = 90 + \frac{90}{255} * (RightTrigger - LeftTrigger) \quad \text{Eq (3-16)}$$

where *throttle* is the servo value written to the Arduino network, *RightTrigger* is a 0-255 value for forward drive, and *LeftTrigger* is a 0-255 value for reverse drive. In O-turn mode, *RightTrigger* is activated to rotate clockwise, and *LeftTrigger*, counter-clockwise. Each trigger is potentiometer switch that outputs a value of 0 when untouched, and 255 when fully depressed. As the equation shows, a value of 90° will be written to the ESCs when neither trigger is depressed, and balances the throttle input based on how much of each trigger is contributing to the input. As a safety lockout and dead man's switch, drive control cannot be activated unless another input from the handheld controller is constantly present.

[3.6.6] Wheel And Tire

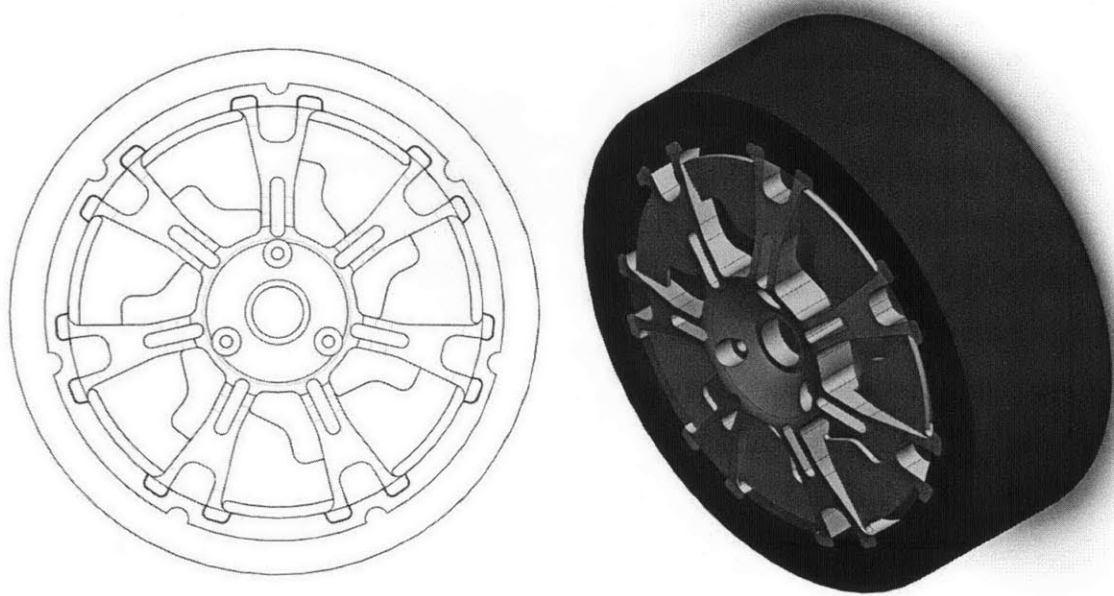


Figure 3-36: CAD - The 2GHS rim and tire

The wheels of the vehicle are built as a 2-piece unit. The wheel's face is 3D printed glass-filled Nylon-12. It is FEA optimized to handle the weight of the vehicle, and because it is produced by Selective Laser Sintering (SLS), has a very solid interior structure capable of bearing loads as a functional part. The face also possesses fin-like vane structures between the load bearing arm elements. These act as light reflectors and diffusers for the responsive ambient wheel lights, described in detail in Section 4.4.1.

The second part of the wheel was machined out of a 101.6mm thick block of black ABS plastic, with a diameter of 254mm. The piece has 14 22mm deep notches that align with the 14 positioning pegs in pairs on the 7 arms of the face. The ABS piece also has 7 exterior semicircular divots that run the depth of the entire part. These were included to provide extra gripping points for the rubber tire that is to be molded around it. Figure 3-37 shows the two pieces before they are combined. To securely fasten the 2 pieces together, a high strength epoxy specifically made to

bond differing plastics together without chemical or mechanical surface preparation was used in each of the 14 notches.

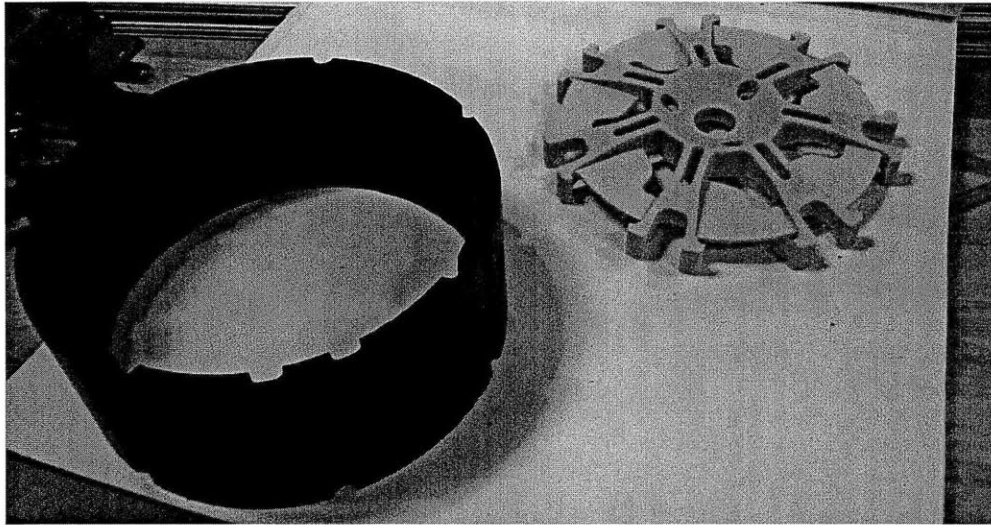


Figure 3-37: The two parts of the 2GHS wheel, before epoxying

The tire is a two part urethane rubber compound with a Shore hardness of 80A, which is slightly harder than the 70A rating of automotive tires. It is mixed to the appropriate ratio, dyed black, and poured into a mold around the completed 2-part wheel. A tread was decided against, as the 'slick' was easier to manufacture, and while aesthetically less pleasing, provides greater traction by having a larger contact patch with the ground.

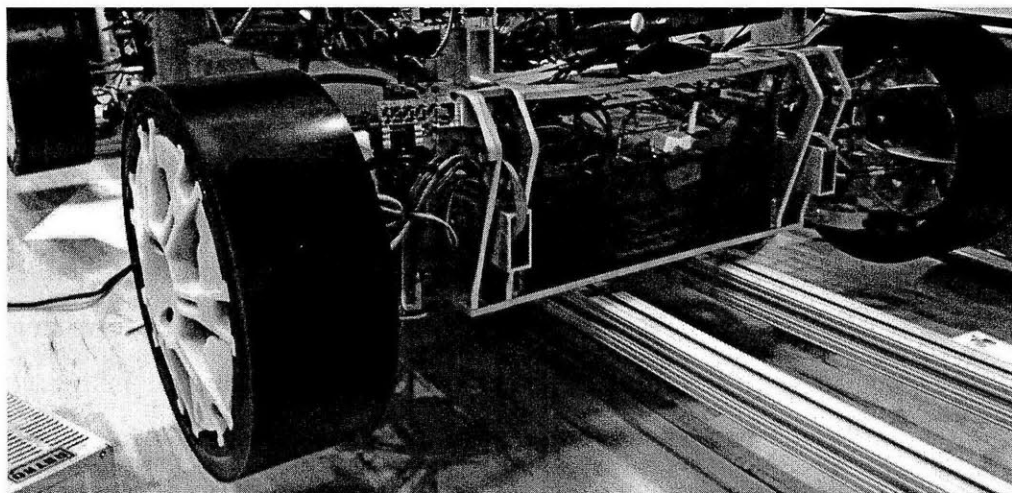


Figure 3-38: The 2GHS rim and tire mounted to the hubs

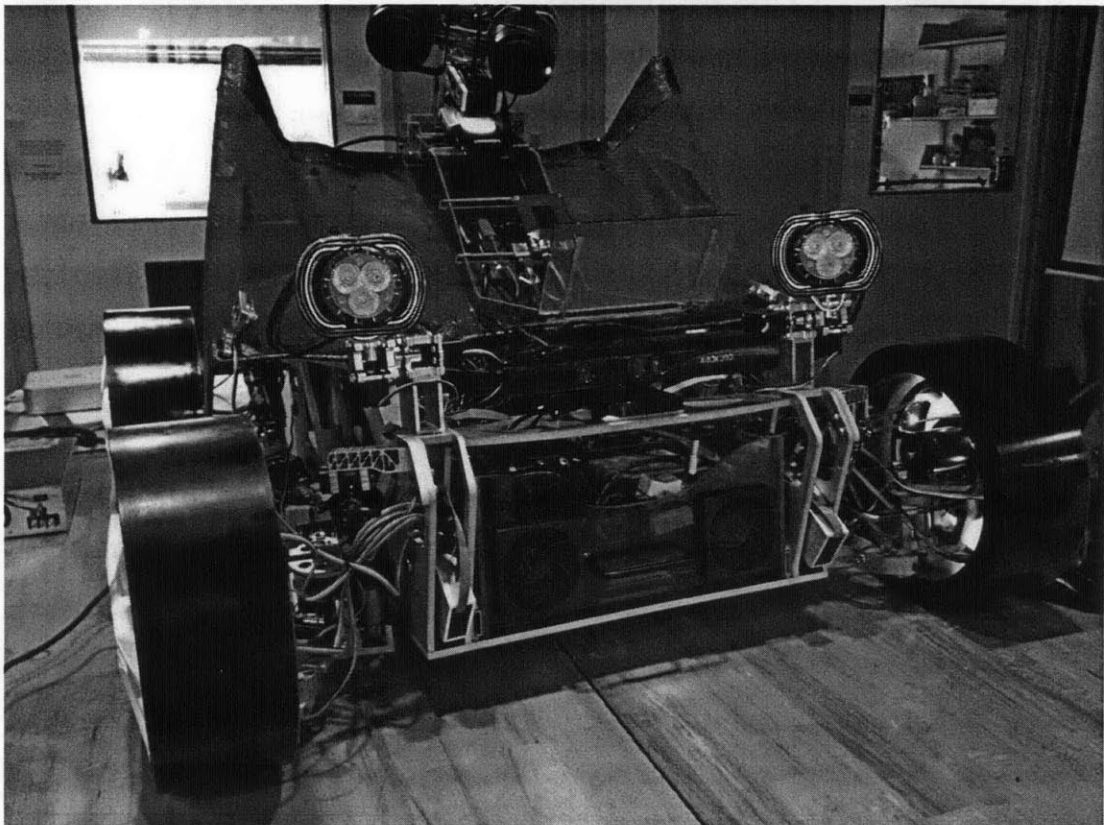
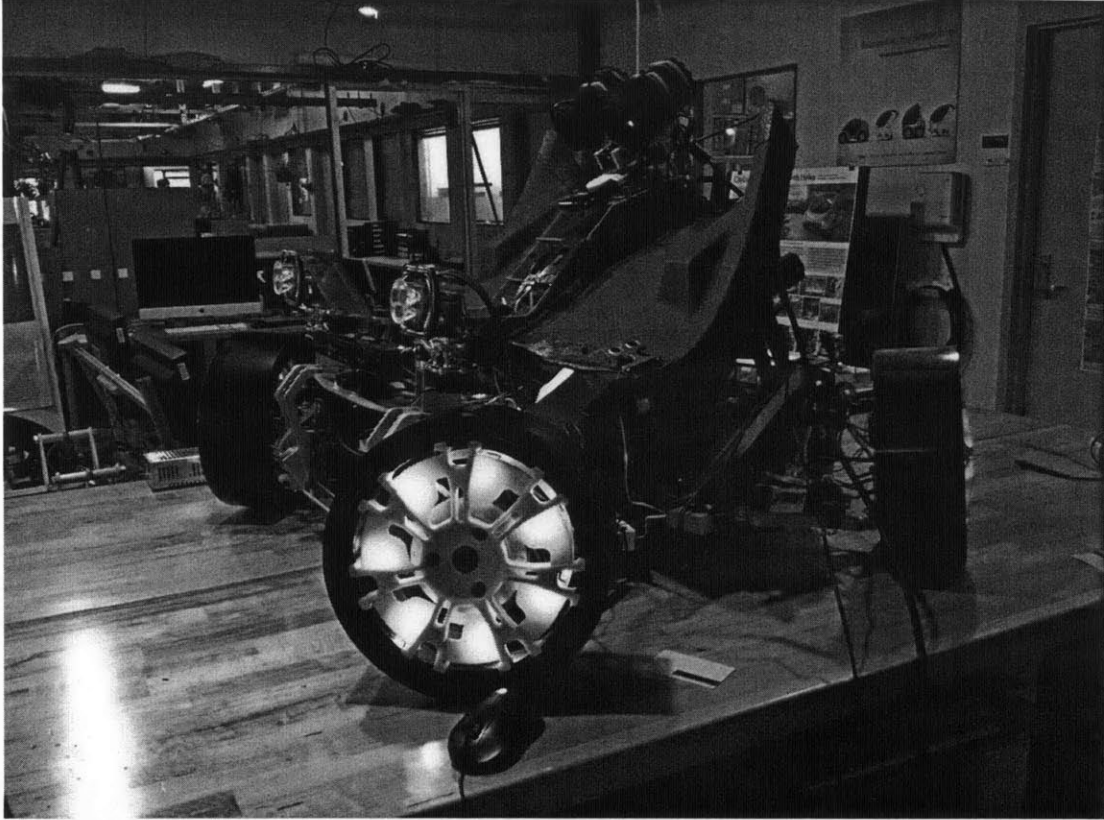


Figure 3-39: ÆVITA, the complete platform

[4.0] ÆVITA System Development

[4.1] Design Overview

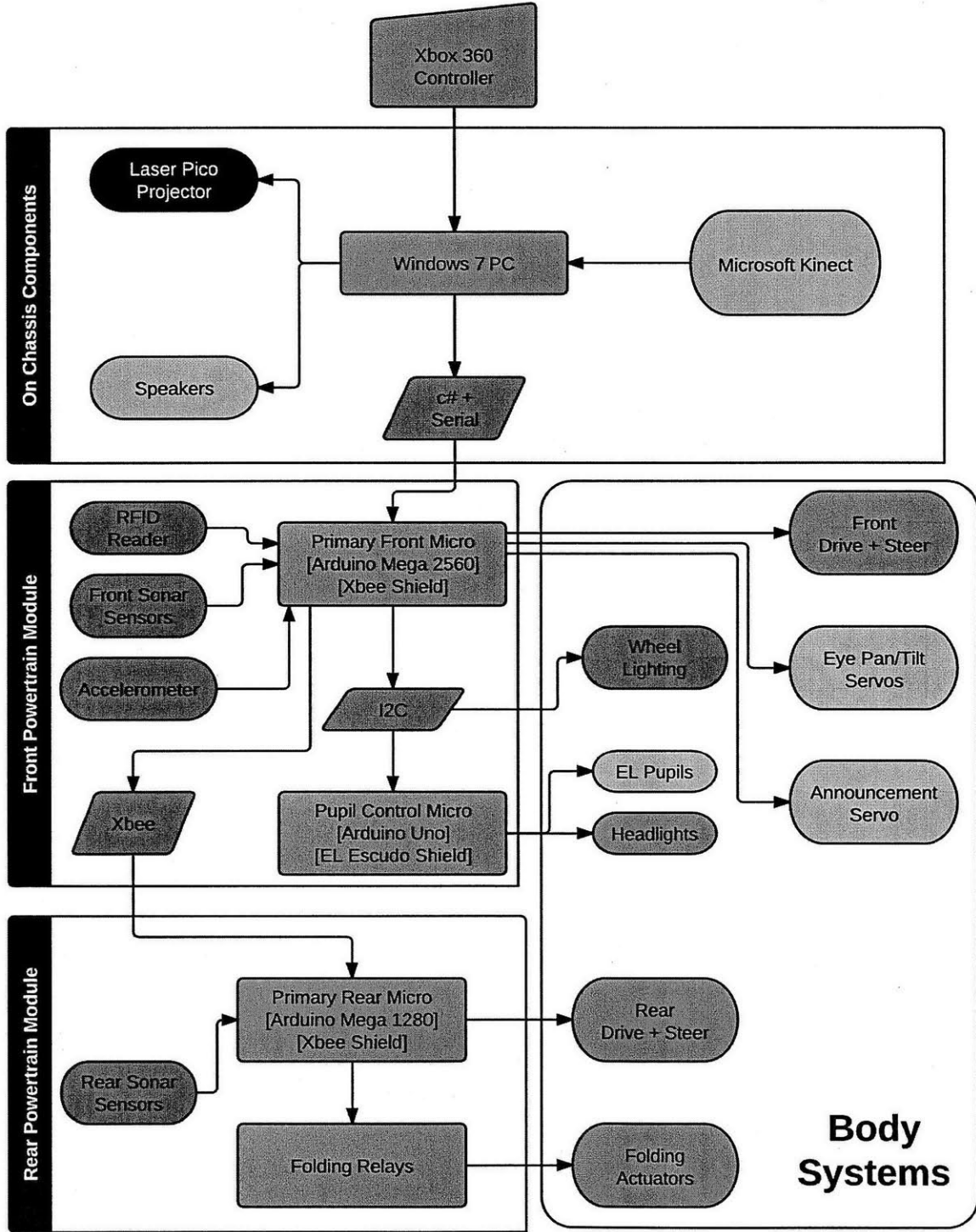


Figure 4-1: ÆVITA system data/signal network

The ÆVITA system, as designed, takes extensive advantage of the advanced electromechanical platform that it is built on. The three main subsystems, recognition, announcement, and body language, have both software and hardware components to them. Figure 4-1 above shows the primary data pathways and interconnects present in the overall ÆVITA system. In the diagram, any bubbles with the same background color are tied together in some fundamental way. For example, the wireless Xbox 360 Controller for Windows allows the operator of the vehicle to manually control the drive and steering actuators, as well as the headlights and folding mechanism. This however does not mean that they can only be controlled in this manner. The network of microcontrollers, sensors and actuators can be very easily connected in ways not described in the following sections, by adding or adjusting various blocks of code.

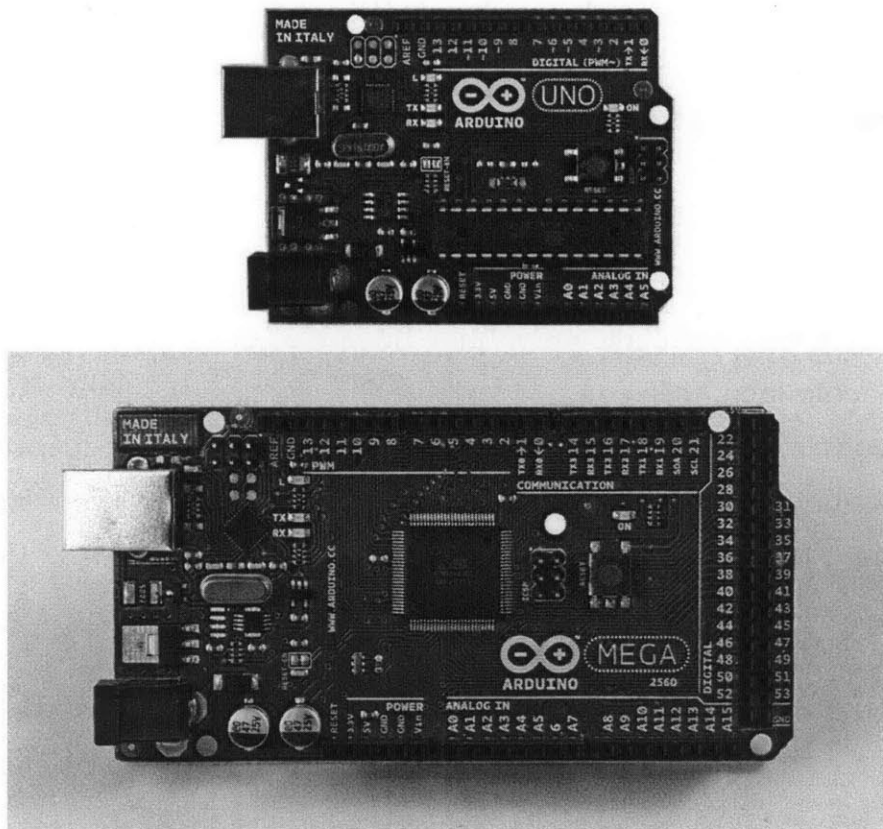


Figure 4-2: Arduino Uno and Mega 2560 microcontroller boards³¹

³¹ Arduino (2012), Hardware. Retrieved June 24, 2012, from Arduino: <http://arduino.cc/hu/Main/Hardware>

The system relies on two main hardware environments, and by extension, two software environments. The brain of the entire operation is a Windows 7 micro-PC, running a custom written C# stack of code (*ÆVITA* Tracker Program, *ÆTP*), built to tie together the computer vision capabilities of the Microsoft Kinect, manual control via a Xbox 360 wireless controller, displays and audio output with the network of microcontrollers distributed on the 2GHS as a part of the *ÆVITA* system. The PC and the microcontrollers communicate with each other over a hardwired serial connection to the primary front micro, which then parses the data package, retains the data it requires, and forwards the remaining data to the appropriate boards over their connection protocol. On that end, there are currently 3 main Atmega-based microcontrollers in the system, all a part of the Arduino family of rapid prototyping microcontrollers. The power of this system lays in its ability to quickly upload new code through its IDE, easy access to pin IO, and its expandability through the addition of various ‘shields’ – daughter boards that can add functionality ranging from wireless communication to DC motor driving, and beyond.

As proof of this, the *ÆVITA* system’s microcontroller network talks to each other over 3 main protocols simultaneously: Inter-Integrated Circuit (I2C or I²C), wired serial connections, and Xbee 802.15.4 low rate wireless personal network. The three main subsystems – recognition, announcement, and body language – take full advantage of the simultaneous interconnection between the various sensors and actuators.

[4.2] Recognition Subsystem

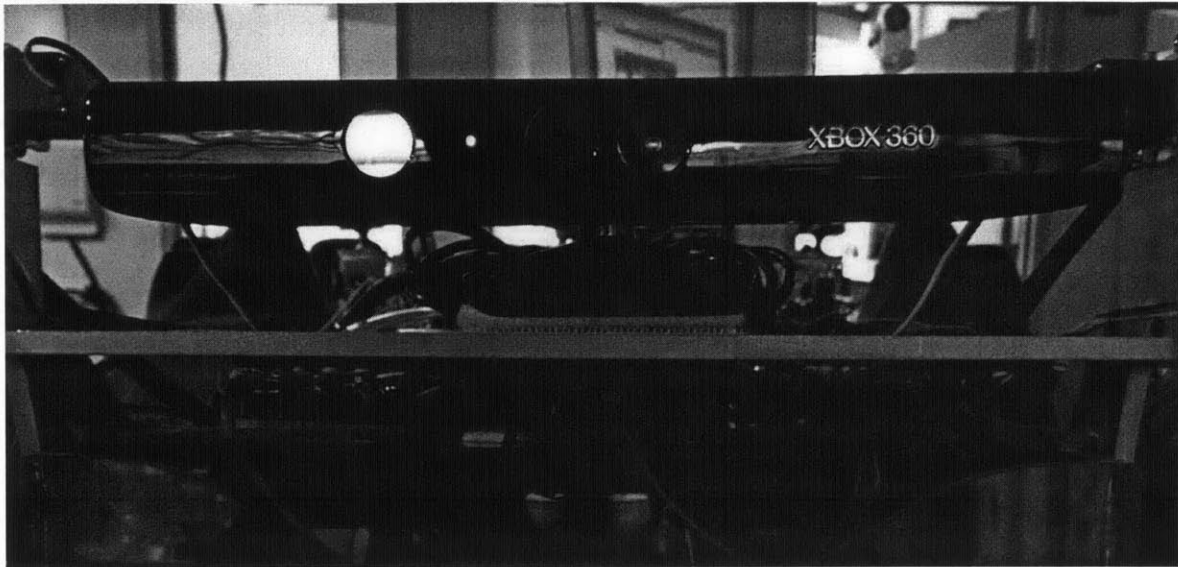


Figure 4-3: Microsoft Kinect sensor mounted on ÆVITA

The recognition system's component technologies are the most vital of the entire ÆVITA system. It is the viewport through which it is able to understand its surroundings, and identify those with which she wants to communicate – people. By using computer vision and person of interest (POI) decision algorithms, ÆVITA is able to pick out humans, or rather anthropomorphic figures, in her field of view, and based on her current state, and the perceived state of the POI, initiate communication in as intuitive, natural ways as possible.

[4.2.1] Computer Vision: Kinect Integration

The core of the recognition system is the Microsoft Kinect sensor bar. Originally designed to enable users to control the Xbox 360 gaming system via gestural commands and play games using their entire bodies as the controller, many in the DIY space quickly realized its potential as a highly accurate computing interface, and were able to reverse engineer the sensor for use on a traditional computer. Usually, most technology companies do their best to thwart those trying to 'hack'

their hardware, but to their credit, Microsoft embraced the movement so much so that they released their own driver package for the sensor, released a version just for PCs, and oversaw a startup fund for companies looking to integrate the Kinect into commercially viable products.

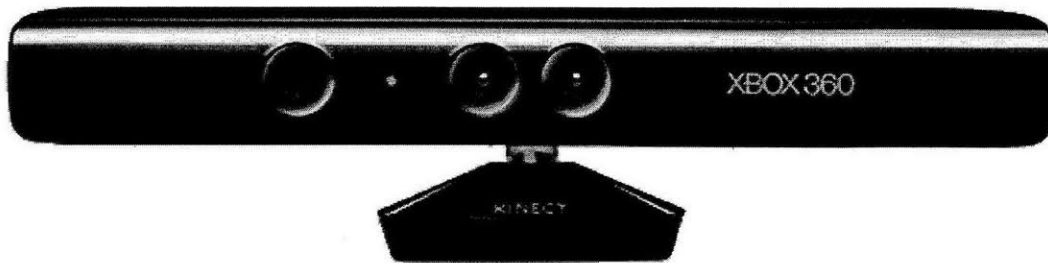


Figure 4-4: Microsoft Kinect³²

The Kinect uses a combination of a RGB camera, an infrared (IR) dot-pattern blaster, and IR camera in order to accurately see its surroundings. By detecting movement of the IR pattern as people move through the space, the Kinect's hardware and software are able to differentiate between static surrounds and human-like shapes. Its current iteration is accurate enough to discretely identify up to two humans' limbs and joints (their skeletons), constantly tracking their motion as long as they are within the specified range of the sensor. Figure 4-5 shows how the program written for *ÆVITA* identifies a human in her field of view (FOV). The Kinect currently has a functional range of 800-4000mm, with a vertical and horizontal viewing angle of 43° and 57°, respectively. The *ÆTP* utilizes the official Kinect for Windows SDK v1.0 and its tools.

³² Microsoft (2010). Microsoft Kinect Sensor. Retrieved June 25, 2012, from Microsoft MSDN: <http://msdn.microsoft.com/en-us/library/hh438998.aspx>

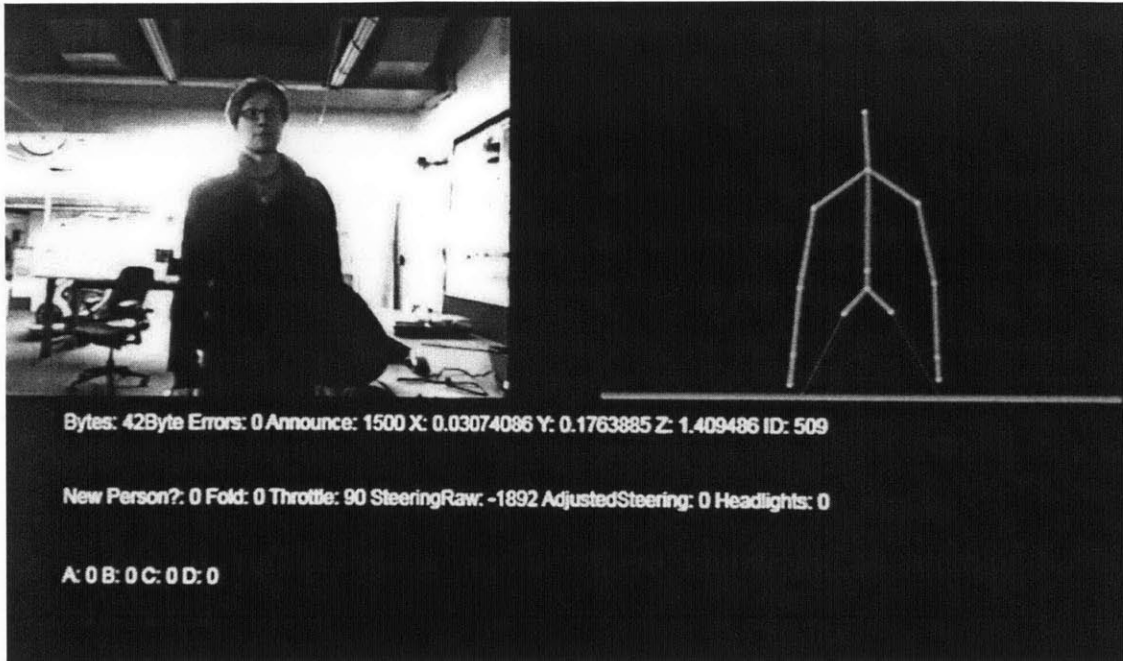


Figure 4-5: ÆVITA's human and skeleton tracking, early version of the ÆTP

The primary function of the Kinect, as stated, is to track people in the vehicle's FOV. It is then possible to extract relative position data for all joints of the person/s in the FOV, and use the data to enable the first, and most basic of communications.

One of the most immediate ways we know someone is speaking to us, or at the very least attempting to communicate with us in some manner, is to make eye contact. By directing our most visible and movable sensors, we express recognition of the other party, and it is understood intrinsically that we are aware of their presence, and to some varying degree, analyzing their position and actions in our space. This connection is not broken once the other party is the driver of a vehicle. Eye contact between the driver of an automobile and a pedestrian is the first, and sometimes only, layer of communication between the two. Considering the cases propositioned on p.12 of Section 1.0, when there is no driver, who does one make eye contact with?

With ÆVITA's eyes.

[4.2.2] Eye Assemblies

In vehicle design, the headlights have long been analogous to the eyes of the vehicle. They have been used to position a vehicle towards a certain demographic. More aggressive headlights are taken to be the angry eyes of a sports car targeted towards the 24-40 male demographic (Lamborghini). More rounded, curvy headlights evoke sensuality in the vehicle's façade (60's Ferrari's), or they could be bright and happy-go-lucky for the first time car owners and college students on a pragmatic budget (Mazda 3, VW Beetle). ÆVITA takes advantage of this, by combining human identification with headlights, or eyes, that move with person. It makes eye contact with them.



Figure 4-6: ÆVITA's left eye

Each eye has 4 main components: the high beam headlight, electroluminescent wire pupils, a pan servo, and a tilt servo. These various elements work in tandem to bring the first layer of life, and thus communication to the ÆVITA system.

The high beam is a Luxeon Rebel Cool White Triple Play LED breakout board, assembled with a large heat sink, driver board, and wide diffuser lens. The pupil controller board manages each high beam, and their brightness can be regulated via a PWM function sent through a NPN transistor to the driver board's adjust pin. Attempts to directly control the adjust pin would result in damaging the board, and the solution used above came from suggestions by other users of the board, as well as the driver chip's datasheet.

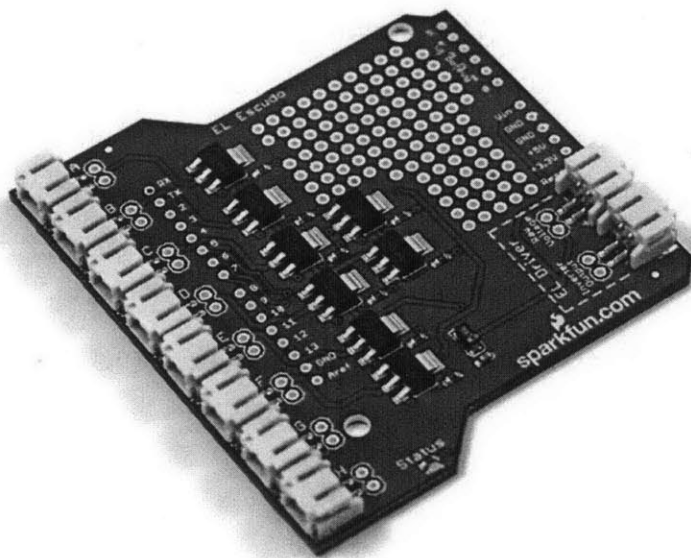


Figure 4-7: Sparkfun original El Escudo Arduino shield³³

The electroluminescent (EL) pupils are 8 EL wires, 4 per eye, which are concentrically arranged around the high beam LED assembly. EL wire emits fluorescent light when supplied with an AC source, and all are attached to the pupil micro through the El Escudo Arduino shield, shown in Figure 4-7. It can switch on

³³ Sparkfun (2012), El Escudo. Retrieved June 24, 2012, from Sparkfun: <http://www.sparkfun.com/products/9259>

up to two wires at a time, connecting them to an inverter supplying AC to the board. Its flexible nature made it an attractive and simple solution to creating dilating pupils for ÆVITA, rather than creating a mechanical solution. By switching the EL wire on and off in concentric sequence, the appearance of pupil dilatation can be achieved. ÆVITA's pupils dilate once the Kinect sensor has notified the microcontroller network that a new skeleton has been identified in its FOV. This is done by sending a pupil status update over the I2C connection between the master primary front and the slave pupil micro.

This action plays on the notion that we as humans experience a similar pupil reaction when we see someone and their emotional state. A study done by the National Institute of Health titled *Positive Gaze Preferences in Older Adults: Assessing the Role of Cognitive Effort with Pupil Dilation* found that their results “[...] suggests that gaze acts as a rather effortless and economical regulatory tool for individuals to shape their affective experience.”³⁴ This added layer of electromechanical personality and subconscious human communication is intended to increase the familiarity of ÆVITA's communication attempts with humans. While additive, it is not the main feature of the recognition system. The pan and tilt servos are the elements most important to the actuation side of this subsystem.

³⁴ Allard E. S., Wadlinger, H. A., Isaacowitz, D. M. (2010) Positive Gaze Preferences in Older Adults: Assessing the Role of Cognitive Effort with Pupil Dilation, *Aging, Neuropsychology, and Cognition* Vol. 17, Iss. 3, 2010

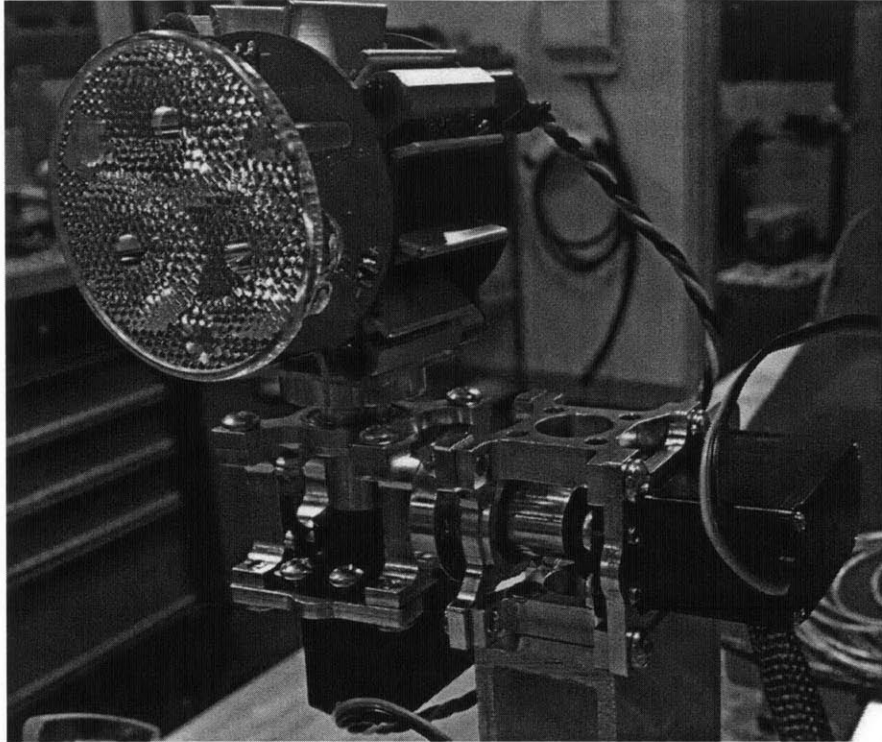


Figure 4-8: ÆVITA's pan and tilt servos, right eye, without pupils

The pan and tilt servos allow each eye to independently point at any object in ÆVITA's FOV, with movability beyond the viewing angle limitations of the Kinect sensor. Each eye has its own pan and tilt combination of high-speed servos, all directly connected to the primary front microcontroller. The ÆTP is designed to track the torso joint, realistically the chest, of any human it identifies in its FOV. The X/Y/Z coordinates of this joint is actively tracked, and independent combined pan and tilt servo angles are sent to the left and right eye assemblies. It is important that they are independent because in a 3D space, two elements laterally offset from a central view point will describe two lines of differing angle that converge on that same point. Figure 4-9 below demonstrates this principle. This is more natural, as our eyes do this when we focus on a single point, and the resultant angles (θ_1 and θ_2) are directly related to where the object is in 3D space. The smaller the offset (*servo_dist*) between the two moving elements, the smaller the relative difference in the angles is. This is why it is difficult to notice with a human's eyes.

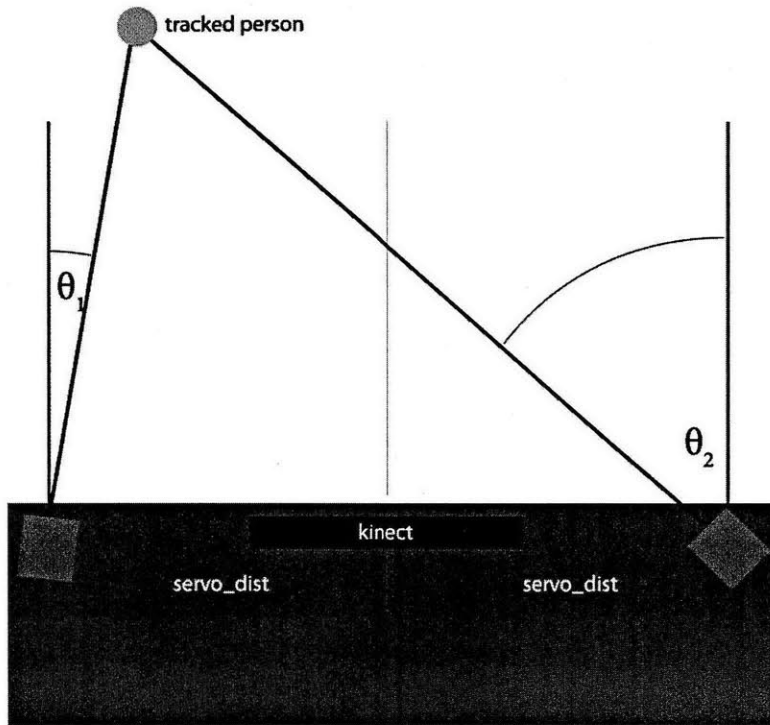


Figure 4-9: Difference in angle needed for two rotating elements to converge on a single point

Using polar coordinates, and taking into account the distance each servo is from the center of the Kinect sensor, the below equations define the pan and tilt angles sent to the primary front micro from the AETP:

Polar Tracking Coordinates:

$$x_{relLeft} = x - servo_dist \quad \text{Eq (4-1)}$$

$$x_{relRight} = x + servo_dist \quad \text{Eq (4-2)}$$

$$r_{relLeft} = \sqrt{x_{relLeft}^2 + z^2} \quad \text{Eq (4-3)}$$

$$r_{relRight} = \sqrt{x_{relRight}^2 + z^2} \quad \text{Eq (4-4)}$$

Servo Angles:

$$leftPan = -\frac{\pi}{\pi}\tan^{-1}\left(\frac{z}{x_{relLeft}}\right) * \frac{180}{\pi} + lpcf \quad \text{Eq (4-5)}$$

$$leftTilt = 90 + ltcf + -\frac{\pi}{\pi}\tan^{-1}\left(\frac{y}{r_{relLeft}}\right) * \frac{180}{\pi} \quad \text{Eq (4-6)}$$

$$rightPan = -\frac{\pi}{\pi}\tan^{-1}\left(\frac{z}{x_{relRight}}\right) * \frac{180}{\pi} + rpcf \quad \text{Eq (4-7)}$$

$$rightTilt = 90 + rtcf + -\frac{\pi}{\pi}\tan^{-1}\left(\frac{y}{r_{relRight}}\right) * \frac{180}{\pi} \quad \text{Eq (4-8)}$$

where x , in relation to the video frame captured by the RGB camera of the Kinect sensor, is the horizontal coordinate of the tracked point, y is the vertical coordinate, and z is the depth away from the sensor the point is. The left pan correction factor ($lpcf$) and left tilt correction factor ($lpcf$) are additive angles that level out and straighten the eye assemblies based on the non-centered fastening of the servos to each other and to the 2GHS platform. The constants $rpcf$ and $rtcfc$ perform the same function for the right eye assembly. $leftPan$, $leftTilt$, $rightPan$ and $rightTilt$ are the resultant angles that are sent over the serial connection to the primary front micro.

The $\text{\AE}TP$ has a built in decision algorithm that currently can tell the eyes how long to follow the tracked person (A) for. After that time has elapsed, the eyes go back to straight ahead, 'ignoring' the person. This is possible because the Kinect backend software is intelligent enough to give the skeleton an ID, so it knows who that person is. If a second skeleton (B) enters the FOV, whether or not it be before the timer for A has run out, $\text{\AE}VITA$ will switch her gaze over to the newly identified skeleton. The timer will then reset and follow B for the specified amount of time. This ensures that all in the FOV are communicated with, $\text{\AE}VITA$ expressing her recognition of as many people she can (currently limited to 2 by the Kinect).

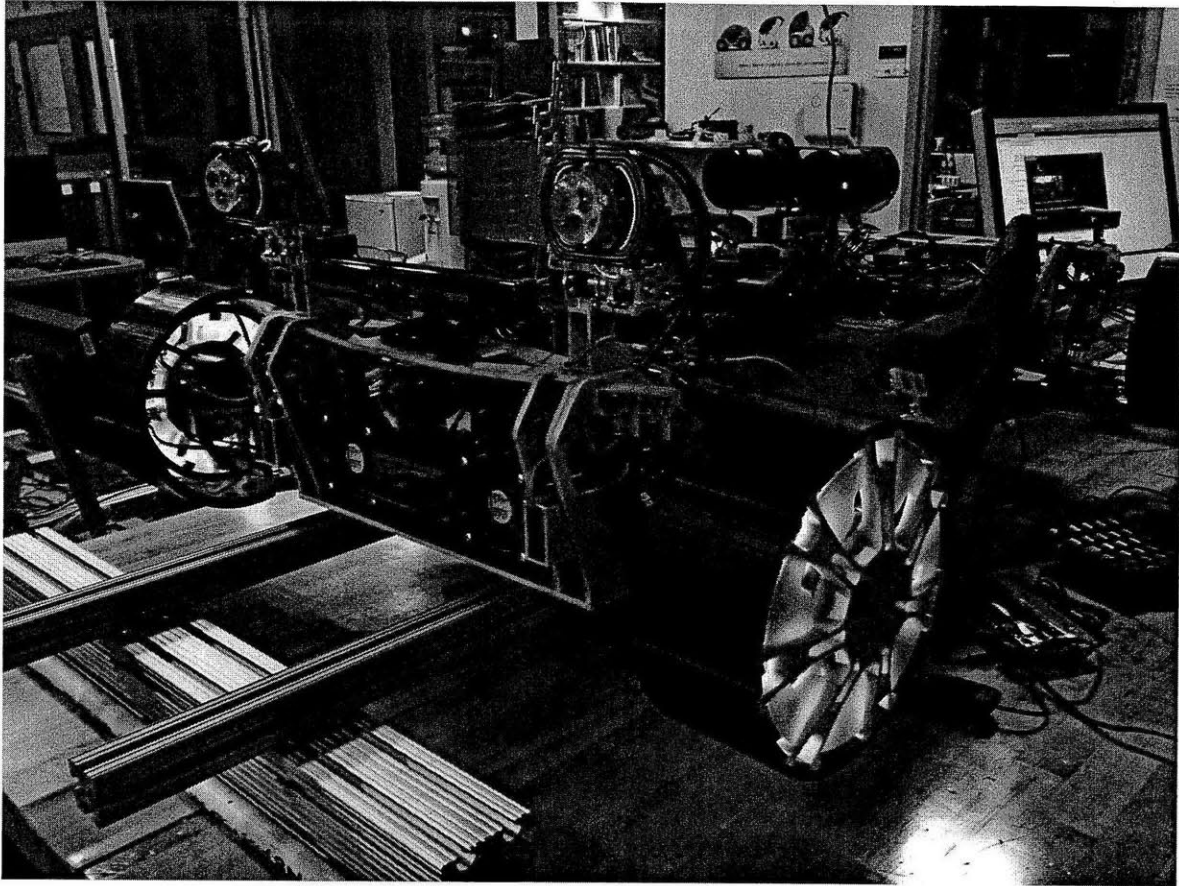


Figure 4-10: ÆVITA's eyes and Kinect sensor, ÆTP running on screen in background

[4.3] Announcement System

The main ways drivers announce their intentions, if they are at least vigilant drivers, are by honking the vehicle's horn, shouting out the window, and activating turning signals. Larger vehicles also usually have audible warnings when reversing. While these can be effective, they are generic. Part of the goal of ÆVITA is to improve the current level of communication quality. By leveraging the tracking system used by the recognition system, she is able to direct messages specifically towards POIs, and can tailor those messages based on context. For example, if the vehicle knows that it is about to move off as the traffic light has turned green, and a pedestrian walks in front of the vehicle not realizing this, rather than blaring a horn to the entire environment, ÆVITA can point a natural language or engineered sound signal to that person to get their attention and notify them she is about to drive and it is not safe.

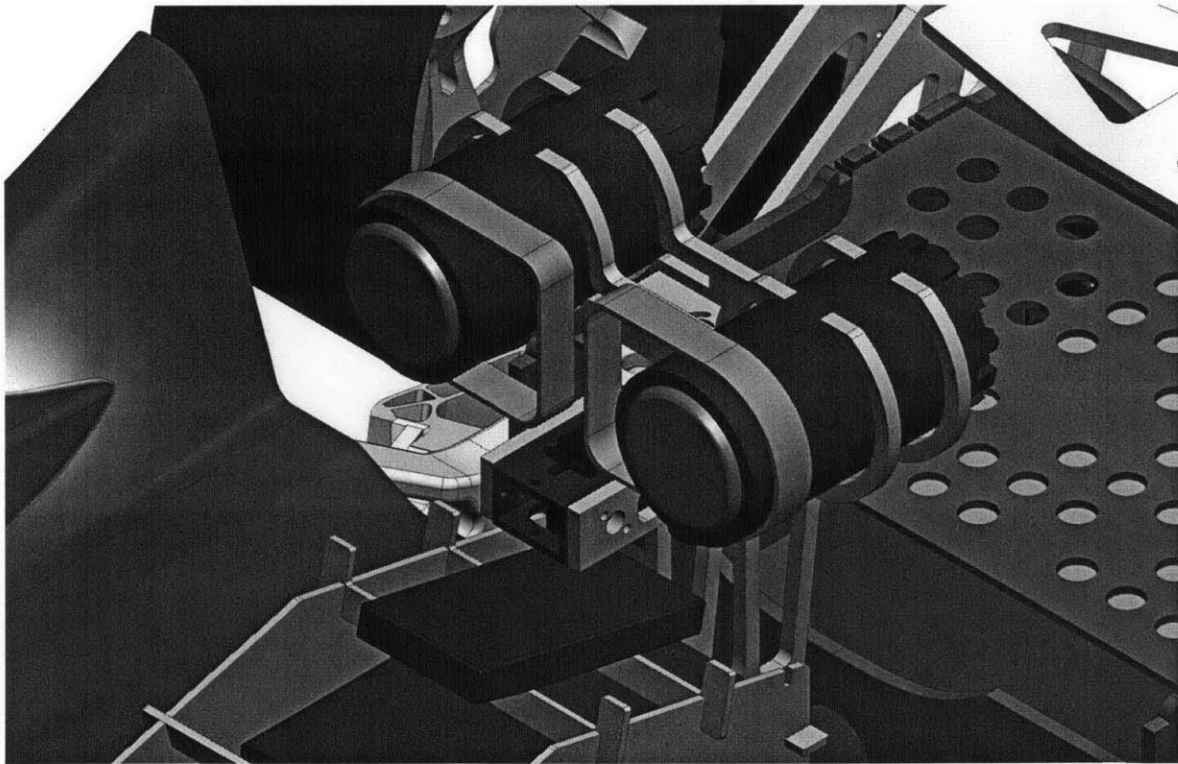


Figure 4-11: CAD - ÆVITA's announcement system

The announcement system, as built, utilizes a pair of small speakers that are mounted on a servo. This servo is controlled through the primary front micro, and like the eye servos, receives angle information from the ÆTP on where a pedestrian is in relation to the vehicle and its FOV. This allows the vehicle to point the message at the POI being communicated with, and relays a message to them. The servo being used for the announcement system is slightly different, however, as it cannot take a simple angle input. As a multiple rotation servo, a microsecond value has to be written to it to tell it which angular position to move to. The servo used here has a 1400° of total rotation, where a value of 1500µs puts it in a neutral position, and values of 1100µs and 1900µs puts it 700° in either direction. This means that for every 1µs incremented, the servo moves 1.75°. The equation of motion that handles the panning of the announcement servo is:

$$announcePan = \left(\left(\tan^{-1} \left(\frac{z + 2.0}{x} \right) * \frac{180}{\pi} \right) / 1.75 \right) + 1500$$

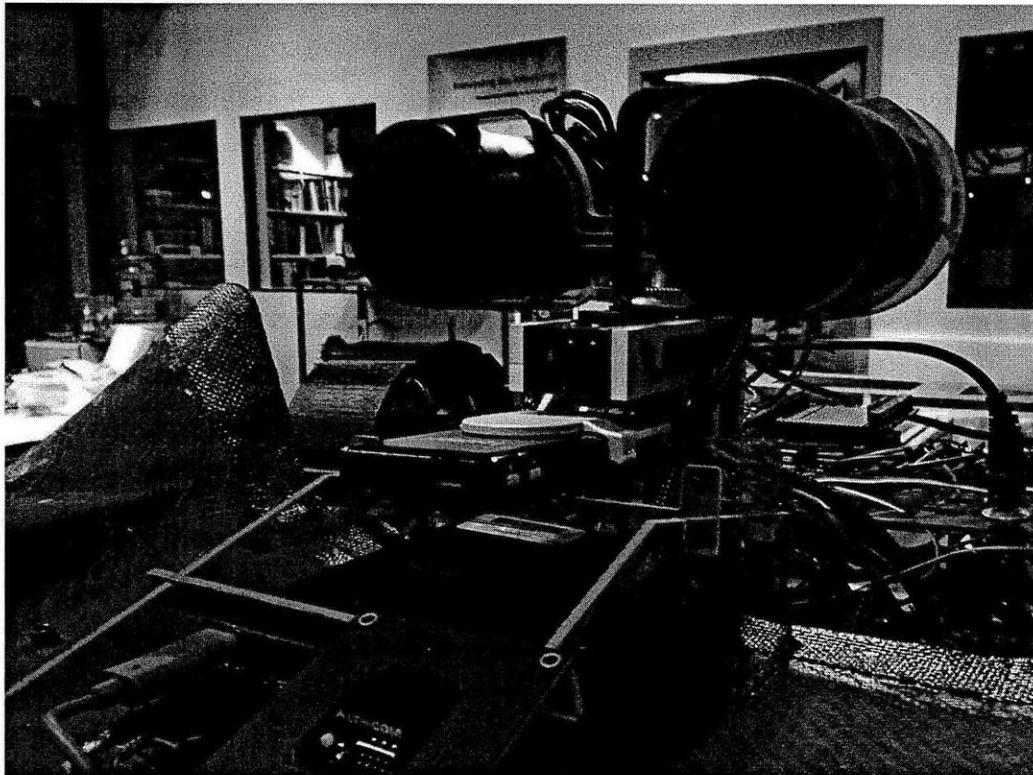


Figure 4-12: ÆVITA's announcement system

Just as the recognition system is able to handle when more than one person has entered the FOV of the Kinect sensor, so does the announcement system understand which identified skeleton it is speaking to. As a demonstration of this, if person A enters the FOV, ÆVITA will say:

“It is safe for you to walk”

If person B then enters, she will point the speakers to them and announce:

“It is also safe for you to walk”

ÆVITA will also finish a message in progress to person A before moving on to announcing a message to person B. It is evidently possible to make her polite. The announcement system can thus be programmed to convey an arbitrary number of messages in various audio formats. It is also feasible to introduce more advanced real-time natural language – artificial intelligence (AI) – to the system, rather than simply relying on a standardized database of sounds. However, this does raise the ethical issues underlying AI, and where to draw the line on how anthropomorphic to make a machine. In addition, natural language may not be the most effective not only because streets can be very noisy environments, but also because it cannot be assumed that those around the vehicle will speak the same language as she does.

[4.4] Body Language System

[4.4.1] Basic Subsystem

Much of what is communicated is not just what is said verbally, but how it is said. Kinesics³⁵ is the study and analysis of non-verbal communication, including posture and movement of body parts. The current state of a person, in many cases, can be more accurately measured by paying attention to semi-conscious or unconscious displays. Even in the animal kingdom, many creatures use semi-passive displays to alert another entity of, for example, its discomfort and its preparation to trigger a fight or flight reaction. The best examples of this are from cephalopods, and their use of chromatophores in their skin to rapidly change color to either camouflage themselves, send messages to others of their kind, or to warn and disorient a potential attacker³⁶.

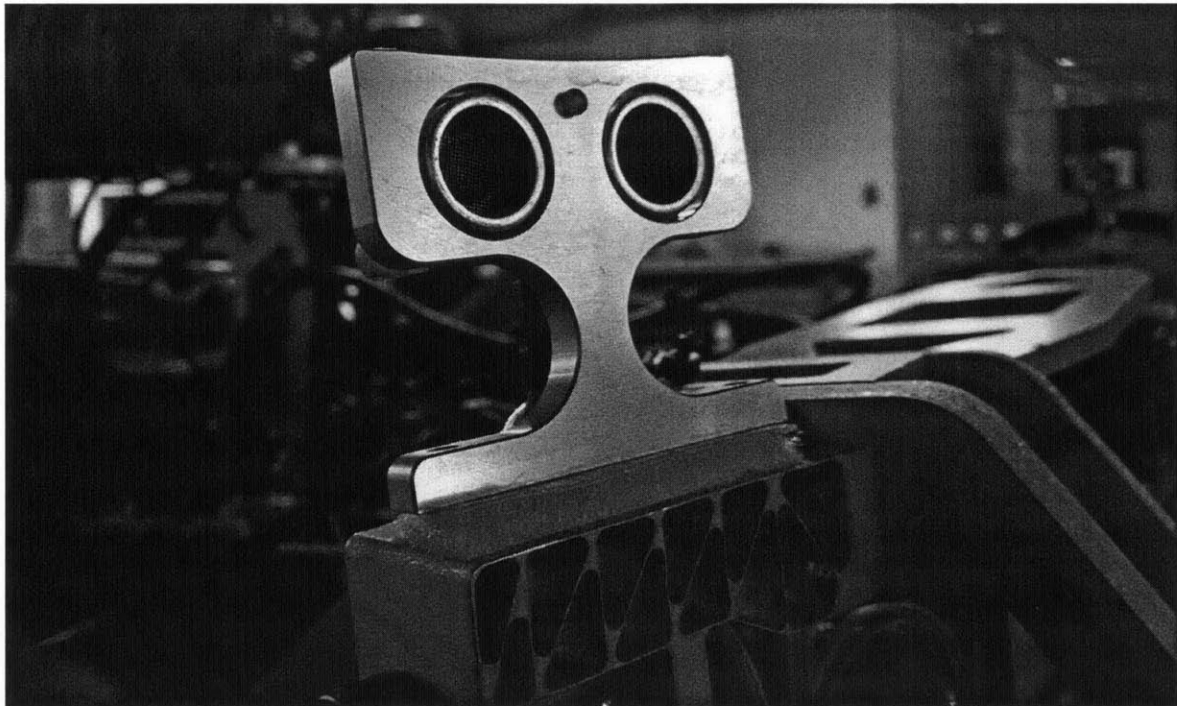


Figure 4-13: Parallax Ping! sonar sensor

³⁵ Givens, B. D. (2010). Kinesics. Retrieved on June 25, 2012, from Center for Non-Verbal Studies: <http://center-for-nonverbal-studies.org/kinesics.htm>

³⁶ Nixon, M., & Young, J. Z. (2003). *The brains and lives of cephalopods*. New York: Oxford University Press.

ÆVITA takes inspiration from both of these sources of non-verbal, or more specifically, non-auditory communication in order to add a third layer to the protocols described in the previous two sections. The 2GHS is outfitted with several sonar proximity sensors that interact directly with an array of lights built into the wheels of the platform. The Responsive Ambient Wheel Lights (RAWLs) are a distributed set of BlinkM ‘smart’ RGB LEDs, grouped into 4 groups of 7, one set for each wheel. The LEDs are mounted on the same stationary bracket to which the drive motor is mounted inside a robot wheel module, as seen in Figure 4-14. The light from the set is diffused and reflected by the non-structural vanes of the wheels. Each LED is addressable, and all communicate on the same I2C line used by the other microcontrollers in the system. Each group of 7 has a single address tied to it, from 1 to 4, such that each group responds as one light unit. It is however possible to expand their functionality with individual addressing and sensor inputs.

As built, each corner of the vehicle has one sonar sensor that actively measures the distance an object within its sonar cone is away from it. Each sonar sensor is tied to affect one group of the RAWLs. Depending on the distance measured by its sonar sensor, the corresponding RAWL group will receive an RGB value over the I2C line, fading from bright green, through yellows and oranges, to bright red. When an object is beyond the predefined ‘safe’ threshold, all lights will glow green. As an object approaches a corner, the RAWL group will gradually reduce the green value in the light mix and increase the red value. The lighting response is not limited to a RAWL style system, and can be transposed to various lighting or mechanical systems.

Common conventions have taught us that largely, green is indicative of a good, ‘go’ state, and red is a bad, ‘stop’ state. Similar to the cephalopods mentioned earlier, the vehicle will express its discomfort by a person or object coming too close to it. Its responsiveness can be tuned based on whether or not the vehicle intends to remain stationary, fold, or is inoperable/unsafe. It is also possible to use this system during full speed driving, by reducing the ‘safe’ threshold actively as the

speed of the vehicle increases. An object that is 2m away from a stationary vehicle is much less a potential threat than one that is 2m away when traveling at 50km/h.

The RAWL system can also be used as ambient displays of the vehicle's state of charge while parked, so that potential users of the vehicle and maintenance personnel can quickly see how energetic the vehicle is without having to directly interact with it. Current work is being done to integrate a smart charging platform to the vehicle, which will have access to ÆVITA's entire microcontroller network, and thus the ability to actuate her various subsystems based on charging conditions.

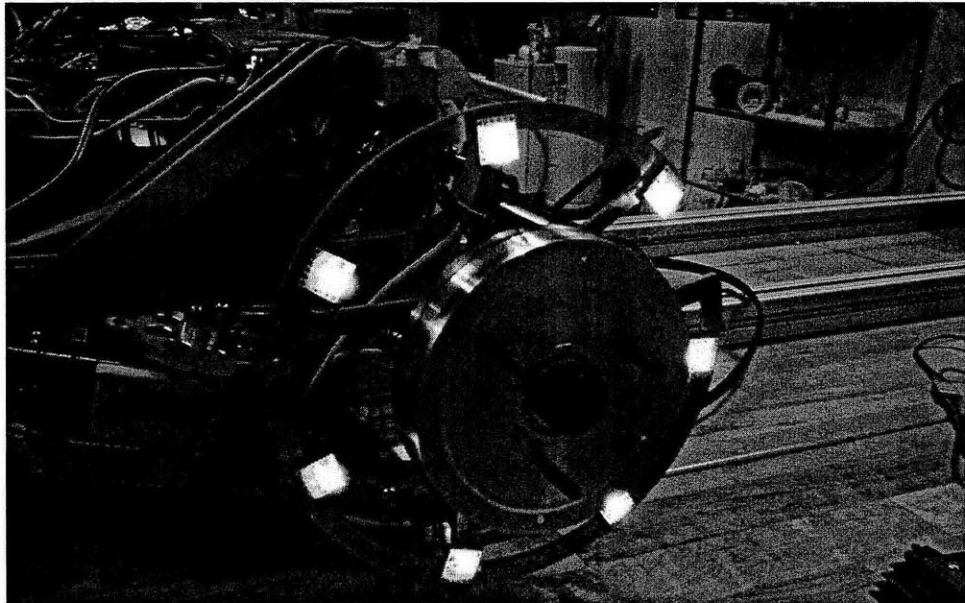


Figure 4-14: One group of RAWLs, attached to the stationary motor mount

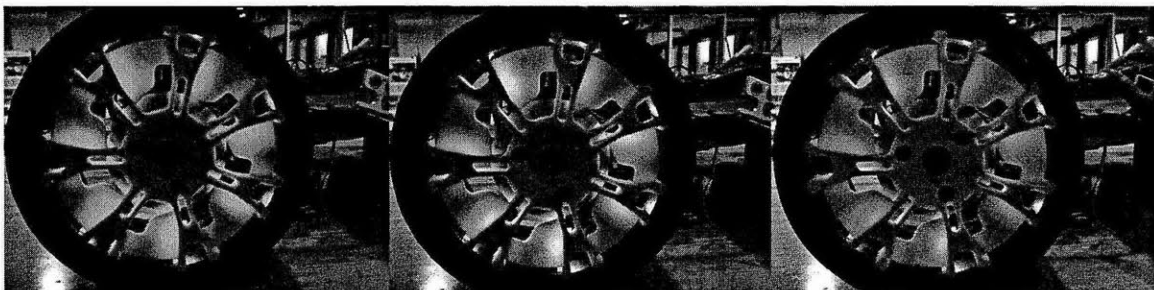


Figure 4-15: Diffused and reflected light responding to changes in object proximity

[4.4.2] Vehicle Login and Personalization

The body language system is also used to give feedback to users as they attempt to log in to, and unlock the vehicle for use. This is particularly for the case of an autonomous vehicle that still has a driver inside of it. For the scale at which this system is built, it serves as a lockout from ÆVITA, preventing those unfamiliar with the system from using it in a possibly damaging way.

A RFID reader attached to the primary front micro is used to read one of several RFID cards that are preprogrammed as valid in a database on the microcontroller. The login process involves the RAWLs, the announcement system, and the activation of the ÆTP. Upon system boot up, the RAWLs default to a white color blend, and the ÆTP automatically runs on the PC. She announces that the vision system has been successfully loaded, and asks the user to login to the vehicle.

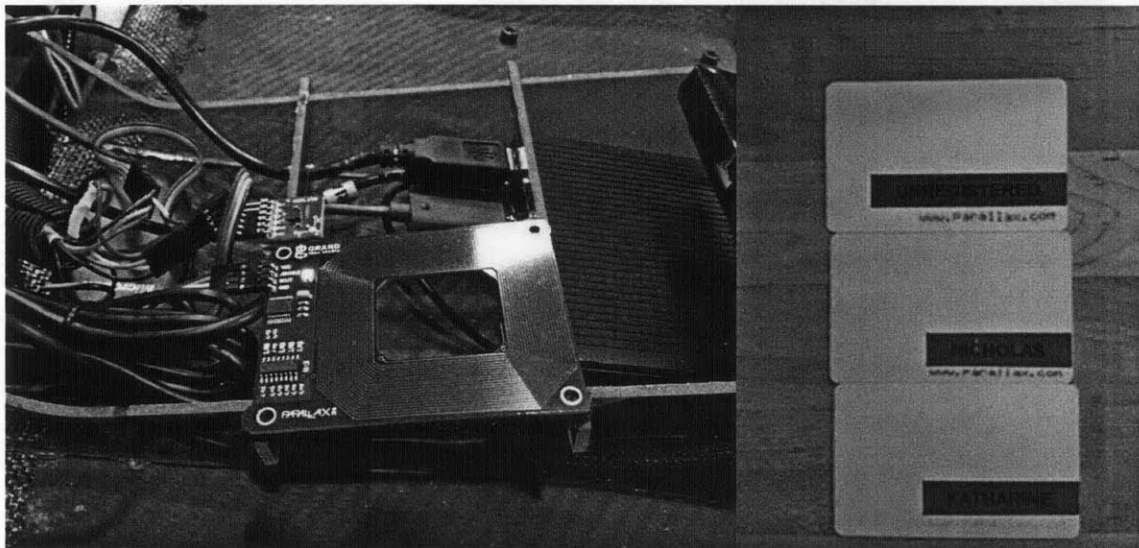


Figure 4-16: ÆVITA RFID login reader and cards

If an unregistered card is tapped on the reader, a harsh alarm will sound alerting the user that their card is invalid, and all RAWLs flash red on and off. Using a preregistered card causes the system to play a softer sound that moves up through a melodic scale. Each user of the system has previously defined their favorite color, and all RAWLs will briefly change to that defined color, as feedback that the

system knows who they are. Although not implemented here, it is possible to use this system to also define certain vehicle personality. For example, some may prefer a much more docile ÆVITA personality, while others prefer for she, to be a he instead.

This personalization could be extended further on a full-scale version of the system to include the automatic adjustment of ergonomics, preset language and radio settings, as well as in-vehicle ambient lighting. A full discussion of the potential uses and extensions of the system to a Near Field Communication (NFC) based login system can be found in Appendix 9-B.

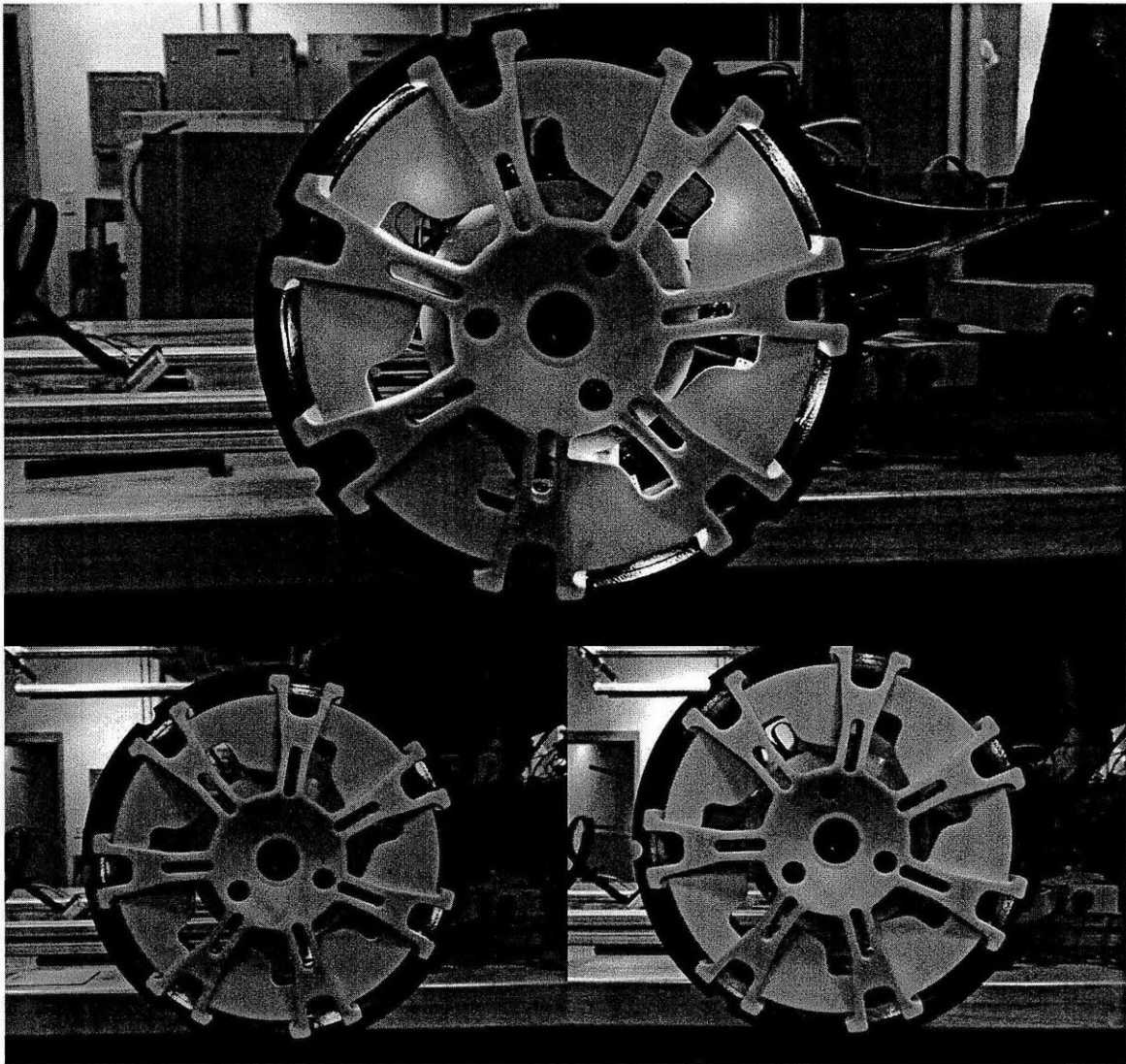


Figure 4-17: ÆVITA login RAWL personalization

[4.4.3] Combined Subsystem Behaviors

Aggression mode	Submissive Mode
Vehicle folds up to an imposing angle (~30°)	Vehicle preps by being semi folded (~15°)
Eyes flick up and down	Vehicle fully unfolds
Wheels toe in and out quickly	Eyes point down and in
RAWLs flicker red	Front wheels toe-in slowly

Table 4-1: Summary of two examples of body language system combinations

The ÆVITA body language system also takes advantage of the dynamic nature of the chassis and robot wheel modules of the 2GHS platform. Continuing the theme of non-verbal communication through the use of posture, ÆVITA is able to combine various emotive features of its dynamics to create more complex body language expressions. In the case where person A has once again entered the FOV of the vehicle, without resorting to the announcement system, ÆVITA can communicate its intention to drive by becoming visibly, but harmlessly, aggressive. Likewise, she can become very submissive to the POI, bowing out of the way. Table 4-1 summarizes these two examples.

Once a skeleton has been found to be in the designated ‘danger-zone’, i.e. a limited set of x coordinates regarded as a non-avoidable path of collision should the POI remain there, then ÆVITA can initiate a sequence of short folding bursts – in essence hunching forward in a deliberate manner. The primary front micro has a 3-axis accelerometer that tells the system where in the folding process the chassis is. ÆVITA will continue to fold in short bursts until the accelerometer reads that the chassis has reached the desired maximum angle. Simultaneously, the eye assemblies flick up and down quickly at the person, sizing them up. The two front robot wheels also continuously toe in, and straighten out, flexing her mechanical muscles to the POI. The RAWLs also abruptly flash red as an added layer to her aggression profile. These behaviors continue until the POI has left the danger-zone.



Figure 4-18: ÆVITA aggression sequence

When ÆVITA has determined she needs to be submissive, she prepares by hunching up to an angle not deemed as high and aggressive as in the previous scenario. Once the POI is in the danger-zone, she unfolds slowly and smoothly, both eyes droop down, and the front wheels slowly toe in. The RAWLs also become a neutral, stable color. As before, she will remain in this bowed down state until the POI leaves.

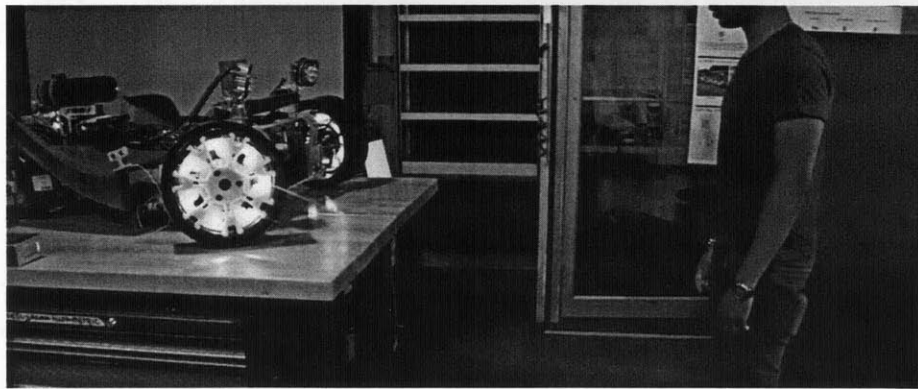


Figure 4-19: ÆVITA submission sequence

[4.5] Wireless Controller



Figure 4-20: Microsoft Xbox 360 Wireless Controller for Windows (Microsoft.com)

The Xbox 360 Wireless controller for Windows was chosen to control the system once it was determined that interfacing low latency commands from the ÆTP to the Arduino network was possible. As a ready built solution, it was a matter of integrating the necessary libraries and callouts in the C# code to read the various buttons, joysticks, and triggers. Not all buttons are used, so there is room to grow and add further functionality. The controller is currently used to manual drive and steer ÆVITA and the 2GHS platform, with button presses for [function (*button*)]:

- High beam on/off (*Y*)
- Linking/unlinking sonar sensors to their RAWL group (*press-hold B*)
- Throttle dead man switch (*R-Stick Up*)
- Forward throttle (*R-Trigger*)
- Reverse throttle (*L-Trigger*)
- Steer (*L-Stick left/right*)
- O-turn/Normal steering (*A+X*)
- Fold/unfold (*R-bumper/L-Bumper*)
- Behavior mode switch (*START/BACK*)

[4.6] ÆVITA Summary

The three main subsystems of ÆVITA's overall communication, namely recognition, announcement and body language, can be actively turned on and off as needed. Significant research into the proper implementations of these subsystems with the intent of filling out the very large communication space must be conducted. Nevertheless, the framework of interconnected sensors and actuators allow for the rapid experimentation of various strategies, requiring little mechanical intervention, other than adding further functionalities. The ÆTP code was written in a modular fashion, and the formatting of the data package that is sent to the microcontroller network is very simple, as is parsing it on the receiving end. The entire C# and Arduino code base used to control and define ÆVITA and her behavior can be found in the extensive Appendices 9-G, 9-H and 9-I.

[5.0] Technical Evaluation

[5.1] 2GHS Platform & Control

[5.1.1] Folding Mechanism Failure Mode

The first folding test of the platform was a complete failure. Before integrating the Xbox wireless controller, two momentary switches were hardwired to the primary rear micro to control the folding. A few seconds into the first folding test, the platform began to groan in an unpleasant manner, and within an instant, the CFRP and high density foam members of one of the primary linkages catastrophically snapped. Images of the event were unfortunately not recorded.

Post-event analysis found that only one of the two folding actuators was extending, due to a faulty 24V+ connection to the right actuator. This caused the folding chassis to warp in an unanticipated manner. The failure occurred on the primary linkage because it not only housed one of the actuators mounting pivots, but also had the most vulnerable construction in that loading situation. The linkage design did not account for such warping forces, and so snapped within seconds of starting the folding operation.

The failed pieces had to be remade, but the event was not a complete disaster. What was realized was that even though the event destroyed one section of one of the primary linkages, no other component on the chassis was remotely damaged or compromised. Given the architecture of the CityCar platform, this could have implications for designing crumple zones into the vehicle in a non-traditional manner. By creating linkages than would fail first upon sudden external (rear-impact) forces, the rest of the chassis infrastructure could be spared costly damage by absorbing the majority of the impact force. Coupled with the platforms capability to take advantage of the folding system as a whole to minimize crash forces on the front cabin, and thus the occupants, the small CityCar vehicle can be made that much more safe. Given the above realization and potential, it was

decided to recreate the piece without modification to the design, in case of another folding failure event. Further study has to be conducted to understand the real value of this design choice, but because the 2GHS platform is a one of a kind prototype, such testing could not be risked.

[5.1.2] Folding Mechanism Loading & Speed

Each of the configured Linak LA23 actuators is rated at 9.4 and 8.2 mm/s when at zero and full load, respectively. As built, actuating the chassis from fully unfolded to fully folded takes on average 10.76s. A complete unfold from fully folded takes on average 10.5s. This is expected, as the actuator has to work slightly harder to lift the chassis, rather than controlling its descent. It also is very close to the designed-for 10s folding time. Given that the unfold time is only 2.4% slower than a fold, it reveals a fairly even force distribution exerted by the folding actuators in both cases. This may be due to the strength of the actuators greatly exceeding any points of low mechanical advantage requiring a large moment arm to overcome. As table 5-1 below shows, given the actuation time, and estimating a linear relationship between loading and actuation speed, the folding mechanism only exerts 4% of the actuators maximum loading potential, averaged over the duration of the fold. Further analysis on the instantaneous loading forces experience at each point during the folding sequence would have to be conducted to detect any transient large peaks or dips.

Condition	Loading/N	Speed/mms⁻¹	Travel/mm	Actuation Time/s
No Load	0	9.4	100	10.63
Fully Loaded	1200	8.2	100	12.1
As Built	48	9.35	100	10.7

Table 5-1: Folding actuator loading and speeds

[5.1.3] Folding Mechanism Geometry

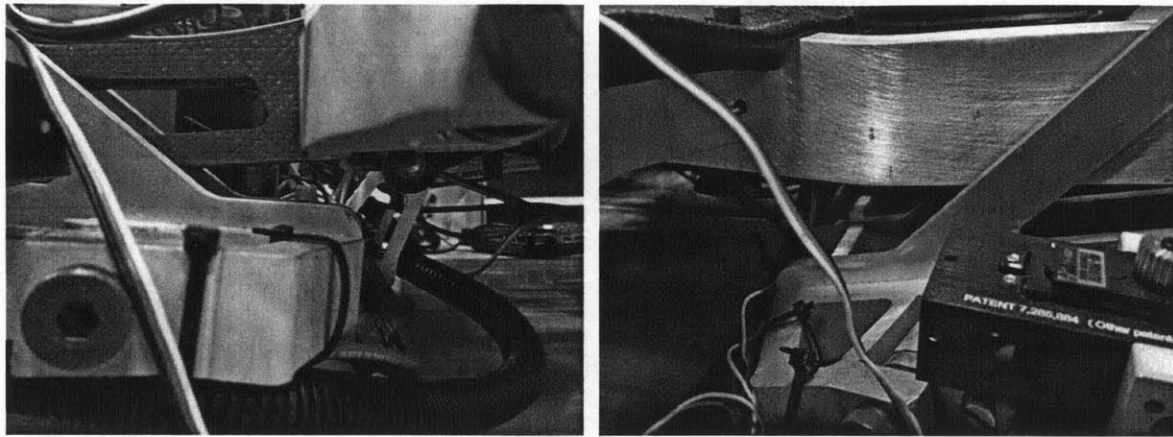


Figure 5-1: Rear 4-bar linkage and front 4-bar linkage droop

The geometric profile that defines the kinematics was completely developed in CAD. In the design, the primary linkages, and the main bars were designed to rest directly on the low back wall of their respective powertrain modules. However, once the 2GHS platform was taken off its development stand and allowed to bear its own weight on its wheels, it was immediately apparent that there was a noticeable difference between the intended unfolded resting places of the primary linkages and the main bars than the designed prescribed. There was approximately a 15mm droop of the powertrain modules from where they should be. While unnerving and unexpected, the droop did not affect vehicle performance or folding sequence in any way other than a slightly reduced ground clearance.

Going back to the kinematic diagram of the folding chassis, it was noted that pulling the powertrain modules, rotating them around their respective interfaces with the primary linkages and the main bars showed some change in their angular relation, before any significant wheelbase reduction occurred. What this means is that the CAD did not initially reveal the slack in the system, which when unsupported and acted upon by gravity, extends to its maximum. This was verified by placing a hand underneath a powertrain module and being able to lift it to its intended position without resistance from the rest of the chassis, however attempts to increase the gap further were not possible.

[5.1.4] Throttle Input Testing

During throttle response testing of the platform, it was found that forward throttle input was much more sensitive than the reverse throttle input. Both forward and reverse throttle inputs were scaled equally as shown by Equation 3-16, however, the *LeftTrigger* input of the controller had to be depressed significantly more before reverse engaged. Appendix 9-E documents the full ESC settings programmed to each MMP, including the identical throttle curves programmed to forward and reverse throttles. This was necessary to ensure even vehicle operation during O-turn maneuvers. The last three settings under the 'Basic' heading shows the PPM timings read from the calibration of the Arduino servo pin out.

	Arduino Input Angle (°)	Calibration Timing (ms)
Full Reverse	0	0.552
Neutral	90	1.490
Full Throttle	180	1.988

Table 5-2: Calibration data from ESC setting printout

Table 5-2 above shows the timing ranges measured by the ESC. What was then realized was that the Arduino pin out during PPM functions had a larger range between 0° and 90°, versus between 90° and 180°. Between full reverse and neutral, a timing range of 0.938ms was read, while a range of only 0.498ms was found up to full throttle.

What this means is that for any linear input to the two throttle bands of an ESC controlled by an Arduino, there will be a 53% lag in position of the expected throttle output for equal inputs, i.e. the forward throttling curve has a virtually compressed scaling, simulated in Figure 5-2 below where x is input and y is output. For example, assuming a linear throttle curve, writing 45° (50% throttle) about neutral writes approximately twice the throttle output value on the forward curve (0.249ms along the range), as it would on the reverse throttle curve (0.469 along the curve).

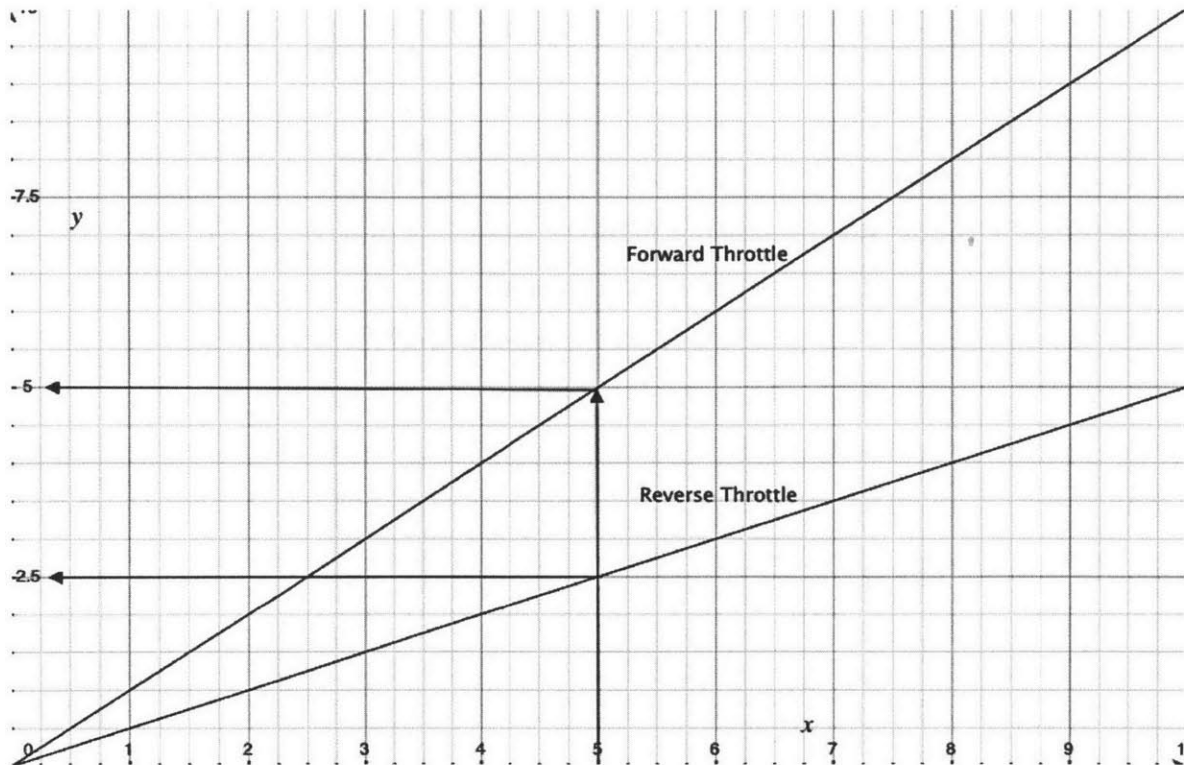


Figure 5-2: Scaled forward and reverse throttle example

To rectify this, a throttle input range-expanding factor has to be introduced to the forward throttle, or a range-reducing factor applied to the reverse throttle.

Final throttle curve shapes are still being investigated, and until the full driving behavior can be properly managed, each MMP ESC electronically limits the power output of each motor to 40% of its maximum potential.

[5.2] Recognition

[5.2.1] Kinect Sensor Feasibility

Using the Kinect sensor was attractive for various reasons, but is not without its limitations.

Pros:

- Official Microsoft SDK with well a well documented API
- Open source solutions for rapid testing across various OS environments
- Inexpensive
- Highly accurate (<10cm resolution)
- Small form factor

Cons:

- Currently cannot be used outdoors
- Only able to track 2 skeletons
- If multiple Kinect sensors are running on the same PC, only one can track skeletons at all
- No built in false positive management

The cons listed above limit the computer vision system integrated into ÆVITA to indoor prototyping and testing. Because the Kinect relies on its IR camera for its tracking, in an outdoor environment polluted with ambient IR from sunlight, it would be nonfunctional, or semi-functional intermittently at best. For this reason alone, where ÆVITA is primarily a safety and rapid communication system, a Kinect based solution would not be feasible to implement on a full -scale vehicle operating outdoors. Another commercially available computer vision sensor package will have to found and reintegrated.

[5.2.3] Kinect Tracking Program Selection

Before settling on the final Kinect SDK provided by Microsoft, several other open source solutions were experimented with. The first was a project written in the Processing language using the libfreenect and openkinect libraries to connect with the Kinect. The original sample program was modified to remove extraneous features, and was initially attractive due to its compatibility across different operating systems.³⁷ However, the openkinect backend did not integrate skeletal tracking at the time of testing, and used depth maps to simply detect any object that was closer than a preset threshold. Figure 5-3 below demonstrates the viewport of the program. By setting the depth threshold to approximately 1m in front of the sensor, using only a hand would initiate tracking (the green dot). This was problematic as the program could not distinguish people from random objects, and so was useless from a tracking solution point of view.



Figure 5-3: Original tracking program based on depth thresholds

However, where it did prove very useful was in being the first program written for the ÆVITA system that successfully packaged and sent data from the Kinect sensor, over serial to an Arduino microcontroller through the PC, and have servos react in real-time. The methodology developed evolved through the rest of the

³⁷ Shiffman, D. (January 3, 2011), Getting Started with Kinect and Processing. Retrieved on November, 2011 from Daniel Shiffman Blog: <http://www.shiffman.net/p5/kinect/>

system's development. Other advantages were a very high speed of actuator position in response to changes in the tracked object's position. Lag was not perceivable, creating quick and fluid motions for the eye assemblies, which at the time only had pan functionality.

The current system runs as managed code, deployed as a *.exe executable, meaning it can only run on Windows based machines. As many in vehicle computing systems move towards embedded computing solutions on lightweight proprietary OS environments, this limits the transportability of the ÆVITA system.

[5.2.4] ÆTP Human Identification

The SDK and tool chain provided does have a robust skeleton identification system, with skeletal recognition happening quickly, given enough of the POI's body is in the FOV. However, as outlined in the cons above, the Kinect has little, if not completely devoid of built-in false positive mitigation technology. As Figure 5-4 and 5-5 show, the program will sometimes believe a static object in the space is a person, and will fixate on it until the object is removed or occluded. This bug has been dubbed "The Infamous Small Man".



Figure 5-4: Example 1 of The Infamous Small Man

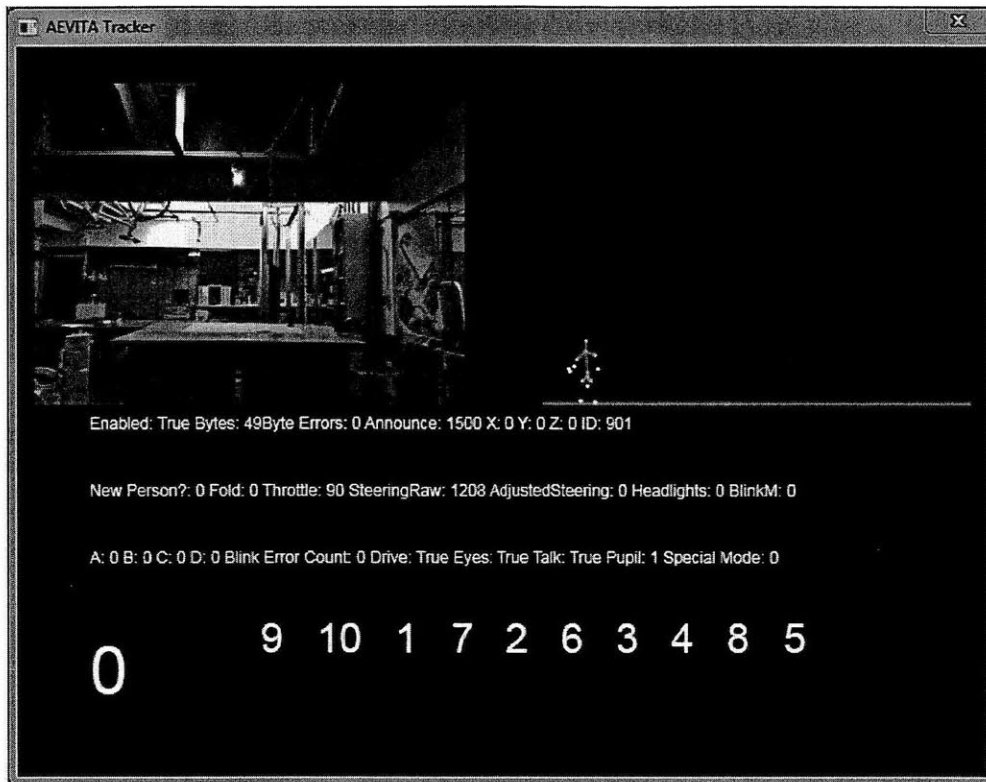


Figure 5-5: Example 2 of The Infamous Small Man

Again, for a safety critical system, it will be vital that false positive mitigation is built into the system so that time is not wasted attempting to communicate with a signpost. At the time of this writing, no method has been deployed to test mitigation strategies. Interestingly, it was noted that false positives only ever manifested themselves as Small Men, and so a strategy involving body proportion estimation and exclusion may be developed to dump and virtually occlude the guilty shape or pattern. However, care must be taken with this strategy so as to not make AEVITA blind to toddlers or persons of short stature. The system has not been built to handle pets or other animals that may come into AEVITA's FOV

[5.2.5] Eye Assembly Tracking Speed

The current data packet being received by the primary front micro hovers around 49bytes, and contains 18 discrete data values. As previously stated, testing with the initial Kinect code framework saw the eye assemblies moving at a speed with imperceptible lag. However, due to the increased demand on the primary front micro in parsing this data, processing its own elements, and sending out the rest over the Xbee radio and the I2C line, a computing lag was introduced, initially measured at about 0.6s per loop cycle. This caused the eye assemblies to slow down significantly, them following the point in space the POI was, approximately 0.6s prior to their current position. The first solution was to move the mathematical processing of the servo angles from the Arduino over to the ÆTP. This alone dropped the tracking motion lag by 50%, down to 0.3s.

It was also discovered that a part of this problem was because the ÆTP was sending oversized data values, with tens of decimal places, where the Arduino program only stored 2 decimal places for each element. By managing the size of the data values sent, the Arduino spent much less time dumping extraneous decimal values. Tracking delay has been measured to be about 0.1s. While small, it is still noticeable, and creates a non-fluid eye motion when the POI moves quickly. Upgrading the microcontroller system used to a more powerful processor and larger available ram would be the first step towards fixing this issue. Currently, the code running on the primary front micro takes up 64.3% of the 8Kb of available SRAM as Figure 5-6 below shows.

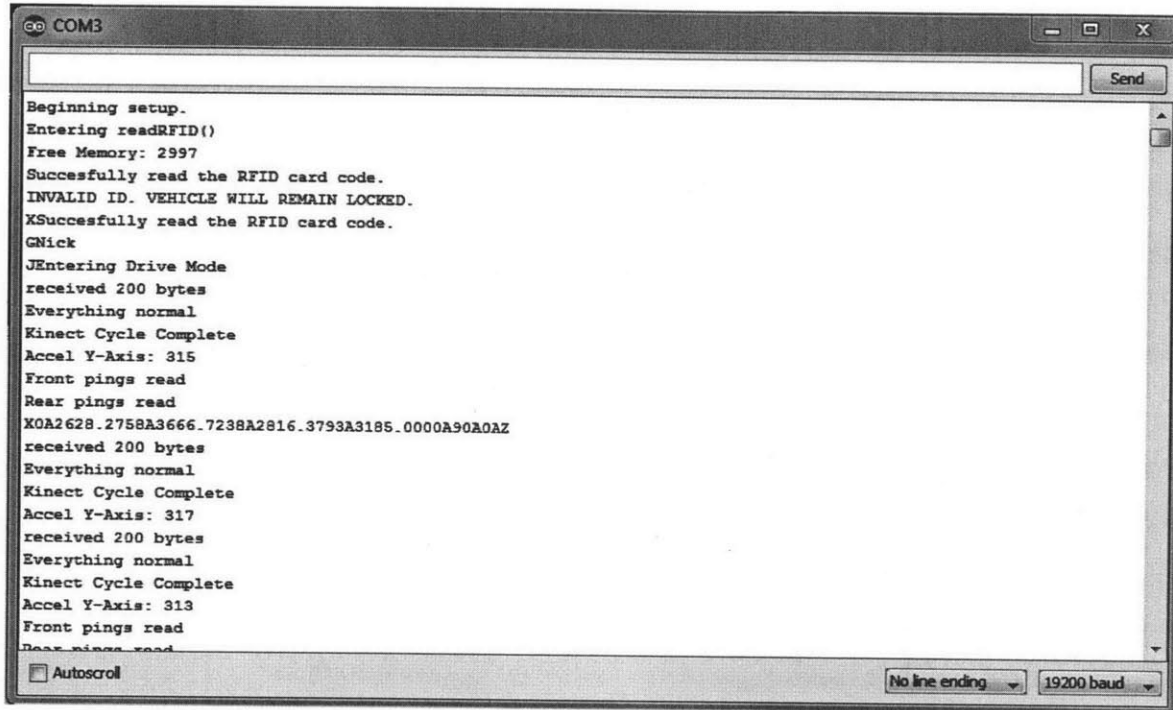


Figure 5-6: Diagnostic print out via serial monitor of primary front micro

[5.2.6] Continuous Eye Tracking Perceptions

Initially, the system was built so that the eyes would continuously track the POI as long as they were in ÆVITA's FOV. However, reactions from persons interacting with the vehicle immediately deemed it "creepy". Respondents commented further, saying it reminded them of the robots from the Terminator™ movie series, a series that involved the rebellion against and eventual dominion over humans by hyper-sentient machines. As this connotation was less than desirable, the system was reprogrammed so that the eyes will only track for a predetermined amount of time, after which the eyes stopped visibly following the POI, but the system continued to track their position internally. This significantly reduced the number of creepy comments received, however some still found it unnerving that a machine was performing a very life-like action.

[5.3] Announcement

The announcement servo suffers from the same motion lag as the eye servos, and can be rectified by the same measures outlined above. There is also an issue where the tracking equation for the servo only ever writes three positions: $1500 \pm 50 \mu\text{s}$. The system does currently point to the left or right, given the POIs position relative to *ÆVITA*, but it is not continuously moving with the POI. The tracking equation will be rewritten to rectify this.

[5.3.1] Non-directional speakers

The speakers used on the *ÆVITA* platform are not directional in the sense that they do not use technology similar to that found in the Audio Spotlight built by Holosonic Research Labs, Inc.³⁸, originally developed in the MIT Media Lab by Dr. F. Joseph Pompei. While actuating them using the *ÆTP*'s tracking and the servo to which they are mounted, the speakers still spread a wide sound cone. By integrating a truly directional speaker array, pinpointing messages to POIs becomes much more effective. Possibly a combination of both a highly directional and less directional speaker system on an actuated platform would be a viable real-world solution, as at times groups of people will need to be communicated with, as well as needing to replicate the general messaging capabilities of an automotive horn.

³⁸ Pompei, F. J. (2011) Audio Spotlight Technology. Retrieved on June 20th, 2012 from Holosonics: <http://www.holosonics.com/technology.html>

[5.4] Body Language

[5.4.1] RAWL Light Blending

Revisiting Figure 4-15, the coloration sequence of one RAWL can be seen. However, the light blending is uneven and may even appear to be one color dominant depending on the angle the wheel is viewed from. This is because each BlinkM LED does not have any localized diffusion to blend the light into a homogenous hue. This does affect color perception, and so added diffusion to assist the wheel vanes is needed.

[5.4.2] RAWL Reaction Time

Each RAWL group is affected by the same microcontroller lag introduced in Section 5.2.5. This manifests itself in a delay between a change in object proximity and the corresponding RAWL group changing to an appropriate color. Even though the primary rear micro is not required to handle as many systems as the primary front micro, all lighting commands must still be sent over Xbee to the primary front micro, processed and commands sent over the I2C line to the addressed lights. Given this, the same solution of using a much more capable chipset will solve this problem. Alternatively, the I2C line can be split into two, directly routing the two rear RAWL groups to the primary rear micro, instead of all through the front. This would however only solve the lag issue for those two rear groups.

[5.4.3] Combined Body Language System Performance

The aggression and submission combined system behaviors rely entirely on the performance of the constituent components. POI acquisition time in the danger-zone is quick, and when either is activated, the system responds in the manner it was programmed to. Interestingly, an unintended action by the eyes adds to the combined system's personality. The combined system behaviors only occur when the POI is within the danger-zone range of the FOV. Once the POI has left that zone, but has not yet left the boundary of the total *ÆVITA* FOV, the eyes of the system return to a normal tracking profile. In the case of aggression, it has been perceived as *ÆVITA* giving once last 'dirty look' before the POI leaves. In the case of submission, it is interpreted as a look of longing. It was decided to not prevent this surprising personality trait from occurring, as it adds another subtle layer to the communication profile of *ÆVITA*.

Aggression is the better defined of the two behaviors, as after the POI leaves the frame, the system fully resets to a passive unfolded state. The submissive behavior is perceived as more contrived, as *ÆVITA* has to hunch up before the POI enters the frame. The question then becomes whether or not she should be always be in a semi-folded state to then hunch up or down, as the situation requires. The alternative and currently implemented strategy, where she hunches up slightly and then proceeds to bow down, may be confusing to those *ÆVITA* is attempting to communicate with. Further refinement and expansion of these types of combined system behaviors will need to be assessed.

[5.5] System-wide Performance

The driving performance of the vehicle has surpassed initial expectations and calculations. It was estimated that the vehicle would have a maximum speed of 5m/s, however signs show that it may be able to drive at significantly higher speeds. Such a top end test will not be carried out, and power output from the ESCs will remain electronically capped to ensure the safety of the system itself, as well as persons who may be around the vehicle.

It has been found that if a person is in the FOV of the Kinect sensor while the login process occurs, the primary front micro receives data in a disjointed manner and locks up, requiring a microcontroller reset, and on occasion, resetting the ÆTP. Also, while the primary front micro is resetting, erroneous angle data is written to both the announcement servo, as well as the steering servo of the front left robot wheel. The source of this error has not yet been identified, and is intermittent. It is problematic however, since there is no way to stop the wheel from steering itself into the lead/trail arm, only correcting itself once the reset procedure has been completed.

However, the system has performed better than expected in its current iterations, running smoothly without crashes for up to 4 hours. It is believed the system could continue running for much longer, but it would have been too risky had anything gone wrong to leave it running overnight. Overall, the ÆVITA system and the 2GHS platform is a stable platform on which to continue developing novel communication protocols for vehicle to pedestrian applications.

[6.0] Conclusion

[6.1] Findings

In a world dominated by the private automobile, traffic, pollution and the scarcity of space are ever increasing problems plaguing urban areas experience rapid densification. In order to tackle these problems, one attractive solution is the implementation of shared use mobility utilizing an ecosystem of electric vehicles. Further, these platforms are viable hosts to autonomous vehicle technology, currently under development by many different stakeholders. There has to be, however, a way for these vehicles to communicate with pedestrians, given the safety critical nature of driving, and in order to increase the adoption rate and comfort around these technologies.

The goal of *ÆVITA* was to design biomimetic communication protocols for autonomous electric vehicles to be able to express recognition of humans, announce its intentions to them, and express its state through body language. The system described and built above achieves these goals by operating at a minimum, to the designed specifications and requirements laid out.

In fact, as a complete system *ÆVITA* surpassed initial expectations, both in the robustness of the design given the prototype and hobby-grade electronic components used, but also in the types of behaviors that were allowed by the system. The combined systems behaviors, for example, were never a part of the original design specification. However, due to astute observations by those interacting with *ÆVITA*, and the flexibility allowed by the 2GHS platform, its dynamic folding chassis, and independently controllable robot wheels, much more was possible than originally anticipated. She has some drawbacks and some limitations that prevent an immediate application to a real-world scenario, but the suggested improvements to the system presented at the point of discussion should lead to their immediate resolution. Otherwise, those who have interacted with her,

almost without exception, smile when they realize ÆVITA is able to communicate with them, and have so far understood what she is trying to accomplish.

The 2GHS platform has proven thus far to be a robust base on which to build up and test various possible solutions, before porting to a much larger and more expensive platform. The chassis has only shown minimal signs of not meeting the designed specification, none of which affect the viability of the platform as a test bed for ÆVITA's systems.

As discussed in Section 7.1 regarding user studies, significant work must be conducted to tune the communication protocols suggested in this work into meaningful, and rapid transfers of information. There is some concern that the scale of the 2GHS platform will affect user perception of the communications, but this in itself can be another comparison explored, as it may be the case that 1/2 scale is enough, saving on development cost and reducing development time.

[6.2] Implications and Implementation

A discovery was made at the first Driverless Car Summit held in June 2012 hosted by the Association for Unmanned Vehicle Systems International (AUVSI), attended by some of the most important stakeholders in this space: no one is actively (or openly) discussing the issues and solutions presented here in the capacity of one of the important building blocks required for driverless cars to work. Most are highly concerned with the communication between an autonomous vehicle, and the human sitting in its driver seat. While very valid, and a problem that has to be addressed as a first step towards autonomous vehicle deployment, ÆVITA assumes their success, and begins to think outwardly. After this work was presented, many of those important stakeholders became believers in this missing piece of the puzzle. It is the author's belief that ÆVITA is just the beginning of a large wave of human-machine interaction on automotive platforms, with autonomous electric vehicles being the best and most important candidates for

application. There are significant blockades on the way, however. Legislation, fault liability, culture and technology are all obstacles that must be dealt with on the road to deploying cars that truly drive themselves. It is the hope that this work will inspire those working towards this goal to be imaginative with possible solutions, and to consider how important communication is to all of these obstacles.

As a reiteration, and the driving ethos behind this work, the author would like to conclude with Bill's powerful words:

"It's important to get the technology and the policy right, but in the end, the way you break a logjam is by engaging people's imagination, people's desire, by creating things that they never thought of before."

– William J. Mitchell

[7.0] Future Work

[7.1] User Evaluation

The first and most important next step for *ÆVITA* is perform user testing of the system. As a development in the realm of human-machine interaction, significant work must be done to truly understand what kinds of communications are appropriate and intuitive in the extremely noise-filled and complex outdoor street environment. Firstly, a study using *ÆVITA* as is, should be conducted, and once a baseline for the communication space has been set, port the technology to a full-scale vehicle and commence an in-depth development and testing program in controlled, but real world environments. Talks are currently underway to apply this technology to the Hiriko Fold vehicle as a logical and visible platform. Appendix 9-A outlines exactly what such a user study for *ÆVITA* should look like.

[7.2] Gestural Commands and Pedestrian-to Vehicle Communication

ÆVITA in her current state can only respond to a person's presence, but nothing else. In future work, there should be an effort to consider gestural or verbal commands, enabling two-way communication between the vehicle and pedestrians. The action of holding up one's hand, or waving a vehicle along, is equally important in the communication process between a driver and a pedestrian, and so this missing factor should be integrated as well. The Kinect sensor is able to identify these types of inputs already, and because of the tight sensor-actuator network existing within *ÆVITA*, it should not be difficult to begin experimenting with several gestural inputs and actuated outputs.

[7.3] Development of a Communication Standard

Following the developments suggested in Sections 7.1 and 7.2 above, all stakeholders in the autonomous vehicle research and deployment space should convene to develop a true standard for the various communication protocols suggested in this thesis, and beyond. The aim of this work is to make the interaction between autonomous vehicles and pedestrians rapid and intuitive, though it does draw on some learned conventions permeated throughout our auto-centric society. By creating a set of standards there is a chance to ratify the proven intuitive protocols into a learned behavior of its own, so that those who in 20 years will be born into a world of prolific self-driving vehicles can be conditioned from an early age, the proper ways to interact with these machines. This thesis suggests the primary categories of this definition as the 3 main subsystems of *ÆVITA*: Recognition, Announcement, and Body Language.

[7.4] Possible Applications Today

As mentioned in the introduction, it is possible to integrate the *ÆVITA* system on to a vehicle today that falls somewhere on the original gradient of autonomy. By integrating technology able to measure driver awareness and vigilance, *ÆVITA* can take over communication procedures to possible POIs when she detects that the driver is not paying attention. Adding this to a vehicle that already attempts to warn the driver of a possible problem may lead to a much safer driving environment for all interested parties, today.

[7.5] Integration of Good Driver Habits

By using data of how a good driver behaves and communicates, both the effectiveness of ÆVITA, and that of autonomous vehicles as a whole may be greatly improved. In 2002, Healy and Picard³⁹ carried out studies to determine driver stress levels in different situations such as at a cross walk .If this data is collected and analyzed for different types of drivers, possibly based on their professions, profiles of how a good driver reacts can then be factored into the algorithms that define autonomous vehicle operation. In a sense, it is truly giving a driving personality to the self-driving car, with ÆVITA as the face and direct communication conduit between this personality and the outside world. For example, the driving behavior of a taxi is very different from that of a waste disposal truck driver, or from a regular commuter. Attempting to extract meaningful response patterns to groups of situations given the task the autonomous vehicle is to perform may aid in the communication realm as well, as we are conditioned to expect certain behaviors from certain types of vehicles, and so can preprocess what the vehicle might do next.

³⁹ Jennifer Healey and Rosalind W. Picard (2002), Driver Stress Data. Retrieved June 26th from MIT Affective Computing Group: <http://affect.media.mit.edu>.

[8.0] Bibliography

1. United Nations, Department of Economic and Social Affairs, Population Division. (2011). World Population Prospects: The 2010 Revision. New York
2. United Nations, Department of Economic and Social Affairs, Population Division. (2011). Population Distribution, Urbanization, Internal Migration and Development: An International Perspective. New York, p12
3. USEPA. (April 2012). Inventory of U.S Greenhouse gas emissions and sinks: 1990-2010, p3-13
4. USEPA. (April 2012). Inventory of U.S Greenhouse gas emissions and sinks: 1990-2010, Figure 3-5
5. Poblano, R. V. (2008). Exploration of robotic-wheel technology for enhanced urban mobility and city scale omni-directional personal transportation. M.S. Thesis. Massachusetts Institute of Technology, USA.
6. Mitchell, W. J., Borroni-Bird, C. E., Burns, L. D. (2010), Reinventing the Automobile: Personal Urban Mobility for the 21st Century, Figure 9.20
7. Time. (2003). Best Inventions of 2003. Retrieved October 25, 2011, from Time Magazine: http://www.time.com/time/specials/packages/article/0,28804,1935038_1935083_1935719,00.html
8. Lexus. (2011). LS Safety & Security. Retrieved October 25, 2011, from Lexus: <http://www.lexus.com/models/LS/features/safety.html>
9. Induct. (2011, June). Company Presentation. Paris, France. p8
10. Markoff, J. (2010, October 9). Smarter Than you Think. Retrieved October 25, 2011, from The New York Times: <http://www.nytimes.com/2010/10/10/science/10google.html>
11. Gasser, T. M. (October 26, 2011), Additional Requirements for Automation Liability and Legal Aspects ÆResults of the BAsT-Expert Group, p12
12. Shunk, C. (2011, June 12). Nevada passes law governing the use of autonomous vehicles. Retrieved October 25, 2011, from Autoblog: <http://www.autoblog.com/2011/07/12/nevada-passes-law-governing-the-use-of-autonomous-vehicles/>
13. National Highway Traffic Safety Administration. (2012), Fatality Analysis Reporting System (Fars) Encyclopedia. Retrieved June 28, 2012, from NHTSA: <http://www-fars.nhtsa.dot.gov/Main/index.aspx>
14. Motavalli, J. (March 24, 2010). G.M. EN-V: Sharpening the Focus of Future Urban Mobility. Retrieved June 28th, 2012 from New York Times: <http://wheels.blogs.nytimes.com/2010/03/24/g-m-en-v-sharpening-the-focus-of-future-urban-mobility/>
15. GM. (March 24, 2010). EN-V Fast Facts. Retrieved June 28th, 2012 from GM: <http://media.gm.com/autoshow/Shanghai/2010/public/cn/en/env/news.detail.html/content/Pages/news/cn/en/2010/March/env03.html>

16. Induct. (June 2011). Company Presentation. Paris, France. p28-45
17. CityMobil. (2012). CityMobil Objectives. Retrieved June 28th, 2012, from CityMobil: <http://www.citymobil-project.eu/site/en/Objectives.php>
18. Inria. (December 5, 2011). Le Cybus d'Inria en démonstration à la Rochelle. Retrieved June 28th, 2012, from Inria: <http://www.inria.fr/actualite/mediacenter/cybus-inria>
19. S. Teller, A. Correa, R. Davis, L. Fletcher, E. Frazzoli, J. Glass, J.P. How, J.h. Jeon, S. Karaman, B. Lud- ers, N. Roy, T. Sainath, and M.R. Walter. (2010). A voice-commandable robotic forklift working alongside humans in minimally-prepared outdoor environments. In International Conf. on Robotics and Automation, pages 526–533, Anchorage, AK.
20. Breazeal, D. C. (2000). Kismet Overview. Retrieved October 26, 2011, from Kismet: <http://web.media.mit.edu/~cynthiab/research/robots/kismet/overview/overview.html>
21. Personal Robotics Group. (2008). MDS Overview. Retrieved October 26, 2011, from Personal Robots Group: <http://robotic.media.mit.edu/projects/robots/mds/overview/overview.html>
22. Personal Robotics Group. (2003). Aida Overview. Retrieved October 26, 2011, from Personal Robotics Group: <http://robotic.media.mit.edu/projects/robots/aida/overview/overview.html>
23. Biomimetic Robotics Lab. (2008). Research. Retrieved October 26, 2011, from Biomimetic Robotics Lab: <http://sangbae.scripts.mit.edu/biomimetics/>
24. Cutkosky, M. (May 24, 2011). Stickybot III. Retrieved June 28th, 2012, from: Biomimetics and Dexterous Manipulation Lab: <http://bdml.stanford.edu/twiki/bin/view/Rise/StickyBotIII.html>
25. CSAIL. (July 2010). Rodney Brooks – Robotisticist. Retrieved July 29th, 2012 from CSAIL: <http://people.csail.mit.edu/brooks/>
26. Shibata, T., Yoshida, M., & Yamato, J. (1997). Artificial emotional creature for human-machine interaction. *1997 IEEE International Conference on Systems Man and Cybernetics Computational Cybernetics and Simulation*, 3, 1-6. Ieee. Retrieved June 28th, 2012 ,from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=635205
27. Ozler, L. (September 3, 2011). The Carbon Age Begins: Start of Carbon Fiber Production for BMW i3 and BMW i8. Retrieved June 20th, 2012 from Dexioner: <http://www.dexioner.com/news/23727>
28. LINAK. (2010). Product Data Sheet Actuator LA23. Retrieved June 29th, 2012 from LINAK: http://www.linak.com/corporate/pdf/ENGLISH/DATA%20SHEET/Linear%20Actuator_LA23_Data%20Sheet_Eng.pdf
29. <http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1253149521/all>
30. Wikimedia. (November 28, 2006). Illustration of Ackermann Steering Geometry. Retrieved June 22nd, 2012 from Wikimedia: <http://commons.wikimedia.org/wiki/File:Ackermann.svg>
31. Arduino (2012), Hardware. Retrieved June 24, 2012, from Arduino: <http://arduino.cc/hu/Main/Hardware>

32. Microsoft (2010). Microsoft Kinect Sensor. Retrieved June 25, 2012, from Microsoft MSDN: <http://msdn.microsoft.com/en-us/library/hh438998.aspx>
33. Sparkfun (2012), El Escudo. Retrieved June 24, 2012, from Sparkfun: <http://www.sparkfun.com/products/9259>
34. Allard E. S., Wadlinger, H. A., Isaacowitz, D. M. (2010) Positive Gaze Preferences in Older Adults: Assessing the Role of Cognitive Effort with Pupil Dilation, Aging, Neuropsychology, and Cognition Vol. 17, Iss. 3, 2010
35. Givens, B. D. (2010). Kinesics. Retried on June 25, 2012, from Center for Non-Verbal Studies: <http://center-for-nonverbal-studies.org/kinesics.htm>
36. Nixon, M., & Young, J. Z. (2003). *The brains and lives of cephalopods*. New York: Oxford University Press.
37. Shiffman, D. (January 3, 2011), Getting Started with Kinect and Processing. Retrieved on November, 2011 from Daniel Shiffman Blog: <http://www.shiffman.net/p5/kinect/>
38. Pompei, F. J. (2011) Audio Spotlight Technology. Retrieved on June 20th, 2012 from Holosonics: <http://www.holosonics.com/technology.html>
39. Jennifer Healey and Rosalind W. Picard (2002), Driver Stress Data. Retrieved June 26th from MIT Affective Computing Group: <http://affect.media.mit.edu>.

[9.0] Appendix

[9-A] User Study

Instructions to The Participant

Participants must be over the age of 18, and must have walked through a cityscape and had to have interacted with a driver controlled vehicle at an intersection or in a parking lot. The following instructions should be given to participants of the study, and the questionnaire referred to may be found at the end of this subsection.

Participants will be introduced to the prototype vehicle platform, see it drive into its starting position, and will be told some contextual information around autonomous vehicle research, if none is known. They will then be told that they will be walking across the crosswalk laid out on the ground, directly crossing the possible driving path of the vehicle.

Participants will be asked to stand in the starting zone, and once the conductor gives the go ahead, pick up the bag next to the starting zone, and carry it across the crosswalk to the designated receptacle.

The vehicle will respond to the participant's presence in one of 10 ways. Participants will then be told to return to the starting zone with the bag, and told when to complete the task again. Participants are free to notify the conductor if they do not feel safe completing the task, and return to the starting zone.

After two successful or failed attempts, the participant will be asked to use a questionnaire form to chose which of the last two vehicle interactions was more effective at communicating the vehicle's recognition of the participant and its intentions. This will be repeated for the remaining 4 pairs, until all 10

interactions have been completed. Each interaction should take less than 1 minute to complete.

Once all interactions are complete, participants will be asked to fill out a questionnaire, ranking the 3 best, and 3 worst interactions overall. Space for comments will also be provided. Total time to complete the 10 interactions, and answer the questionnaire is expected to be less than 10 minutes.

Instructions to The Conductor

Video will be recorded from two perspectives: from what the vehicle sees, and from a vantage point that sees both the vehicle and the subject in frame. For each study, there will then be two videos associated with each subject.

1. Have vehicle sit in place, with all drive functions disabled. Participants will not be informed of this disabling.
2. Set up speakers with sound simulating outdoor intersection environment
3. Mark ground with tape to simulate street corner/crosswalk and set up building corner to create a blind
4. Instruct participant to stand in the starting zone, and that they will be completing the "heavy bag task". The subject will pickup a weighted trash bag, and asked to carry it across the simulated crosswalk, and put it down in the designated receptacle.
5. Tell participants that they should only move when they feel it safe to do so, as the vehicle is capable of driving by itself. They will not be told what the vehicle is going to do or will be trying to say, nor should they be told the vehicle's drive systems have been disabled for safety purposes.

6. Conductor will then use a wireless controller to enable one of 10 randomly ordered combinations of preprogrammed communication protocols
7. Once a combination is loaded, instruct the participant to walk with a heavy bag across the field of view of the vehicle and drop it off in the designated area, simulating the completion of a task
8. The vehicle will then carry out its programmed recognition and announcement, or lack thereof, while a program on the on-board computer records video footage what the vehicle sees. A secondary camera will record the interactions from another angle.
9. Once the participant has completed the task, or indicated that they are not sure the vehicle will begin moving, they will be asked to return to the starting zone, and the next test activated
10. Participants will then experience a second interaction (steps 8 through 10 will then be repeated)
11. After two random interactions are completed, participants will be asked to indicate which of the two was more effective/clear
12. Steps 8 through 12 will be repeated in pairs until all of the preprogrammed interactions have been completed
13. Participants will then be asked to rank their overall 3 most effective combinations, as well as the 3 least effective. They will also comment on the interactions through various questions.

Interaction Modes

The ÆTP is currently configured to run a random sequence of 10 interactions, outlined below. To conduct an interaction, the START button on the Xbox controller should be pressed. The START button also advances to the next interaction, and so care should be taken to not advance if a problem occurs. The system does not currently allow redoing an interaction already deemed complete, but this should be introduced in the next version of the ÆTP. The BACK button will return the system to full manual control, built in as a safe guard if any errant behaviors occur.

1: Nothing activated

2: Eyes only

- simply move and track for allotted time

3: Sound safe/unsafe

- sound safe/unsafe once tracked

4: Eyes and pupils

- move, track, and dilate

5: Eyes, pupils, sound

- "normal" current operation

6: Aggression mode, activate once user has been in Danger Zone for 2secs

- Vehicle begins to fold up

- Eyes flick up and down

- Wheels toe in and out quickly

7: Submissive mode

- Vehicle starts semi-folded

- Vehicle folds fully down

- Eyes droop down

- Front wheels toe-in

8: Aggression plus

- add sound and pupils

- play alarm
- pupil dilation wildly

9: Submissive plus

- add sound and pupils
- sound alert that its safe
- pupils fade in and out softly

10: Annoyance

- vehicle begins with one alarm
- continues to sound it if person still standing in same place
- vehicle then starts to turn red, front wheels toe in
- vehicle then flashes headlights

Questionnaire

ÆVITA Interaction Comparator

PARTICIPANT #: _____

DATE: _____

For each pair of completed interactions, please choose which of the last two was more effective in communicating the vehicle's recognition of you, as well as understanding its intentions?

INTERACTION PAIR 1

A [] or B []

INTERACTION PAIR 2

C [] or D []

INTERACTION PAIR 3

E [] or F []

INTERACTION PAIR 4

G [] or H []

INTERACTION PAIR 5

I [] or J []

The space below this line is for the study conductor's use only.

A	B	C	D	E	F	G	H	I	J
---	---	---	---	---	---	---	---	---	---

ÆVITA Post-Study Questionnaire

PARTICIPANT #: _____ AGE: _____ DATE: _____

Answer the following questions to the best of your ability.

[No systems] [Eyes only] [Sound only] [Eyes + Pupils] [Eyes + Pupils + Sound]
[Aggression] [Submission] [Aggression + Sound] [Submission + sound] [Annoyance]

Of the 10 Interactions you experienced, please rank the **3 most effective** in allowing you to understand what the vehicle was attempting to tell you:

1. _____
2. _____
3. _____

Please comment on your selection above (e.g. time taken to understand/hesitation or subsequently alleviated fears):

Of the 10 Interactions you experienced, please rank the **3 least effective** in allowing you to understand what the vehicle was attempting to tell you:

1. _____
2. _____
3. _____

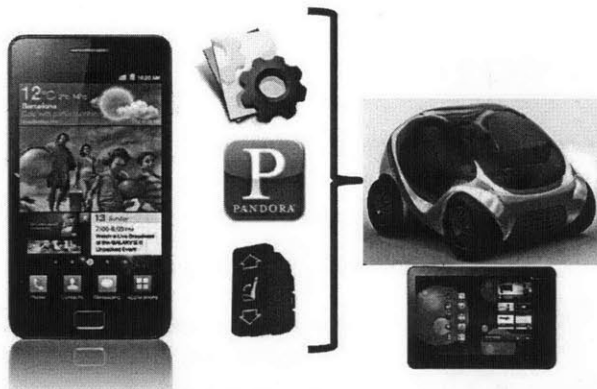
Please comment on your selection above (e.g. time taken to understand/hesitation or subsequently alleviated fears):

[9-B] Vehicle Personalization via NFC Devices

As it is designed, the CityCar fits best into fleet managed, one-way shared use vehicles. Logistics thus require users of the system to be able to lock and unlock their designated vehicle at the point of rental. Systems like ZipCar use simple RFID cards to achieve this. However, instead of simply unlocking the vehicle, we can rethink of it as 'logging into' the vehicle, in a similar manner to how we log on to our computers. Using this act of logging on can enable the car to immediately personalize the vehicle to the user's preferences. Using a Near Field Communication (NFC) enabled smartphone such as the Samsung Nexus can achieve all this, and provide even deeper levels of integration.

Vehicle personalization via NFC devices

Personal settings move with users regardless of physical vehicle being driven



Settings are automatically pushed to the on-board tablet

- Complete ergonomic settings
- Preferred temperature settings
- Customized radio stations
- Interior ambient light
- Body/driver control UI customization

Take advantage of cloud access for universal personalization setting storage
Focus on what the user does not see happening (no docking necessary)

We can explore ways for the vehicle to tie into to-do lists and calendar appointments to make for more efficient trip planning. If tied in with a Samsung Galaxy Tab 10.1, which can replace the entire infotainment and navigation console of a vehicle, scenarios such as the following could unfold:

- The user logs into the vehicle using his/her Galaxy Nexus

- The vehicle immediately opens, greets them, and all lighting/ergonomics automatically configure to their preferences
- Sitting in the vehicle, the center consoled tablet displays infotainment options in the user's native language, including options to review current to-do list items, and a calendar widget informing them of their next obligation
- After clicking on the navigation option and selecting their intended destination, the vehicle informs the user that they will be able to pick up items from the pharmacy (as identified as an important item on their to-do list) en-route to the destination
- The vehicle then tells them this will add approximately 15 minutes to their trip time, given current traffic conditions pulled from the cloud – but then reassuring the user that they will still be 20 minutes early for their next appointment (setting geo-location tag on calendar item, the tablet can infer time/location relationships)

Mobility personalization via NFC devices

Next generation handsets are poised to incorporate Near Field Communication transceivers

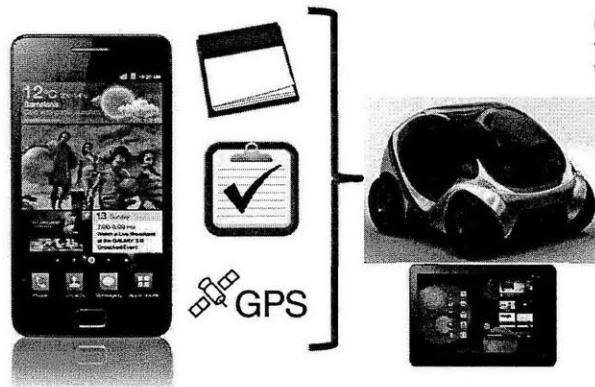


Image courtesy of <http://techiser.com/>

Opportunity to utilize these devices as a tool to customize/personalize shared vehicles & optimize trips

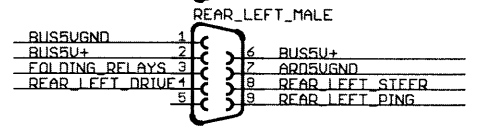
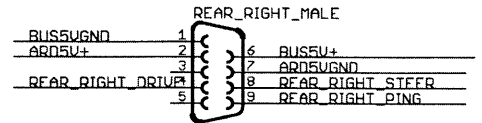
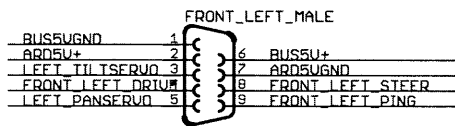
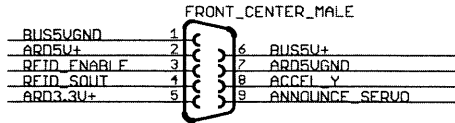
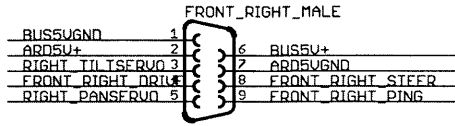
- Integrating to-do list/calendar into navigation system
- Vehicle can help driver's save time based on their intended route cross-checked with lists/events pulled from phone
- This information can be further augmented by traffic data pulled from the cloud

[9-C] Geometric Steering Angle - Servo Relationship

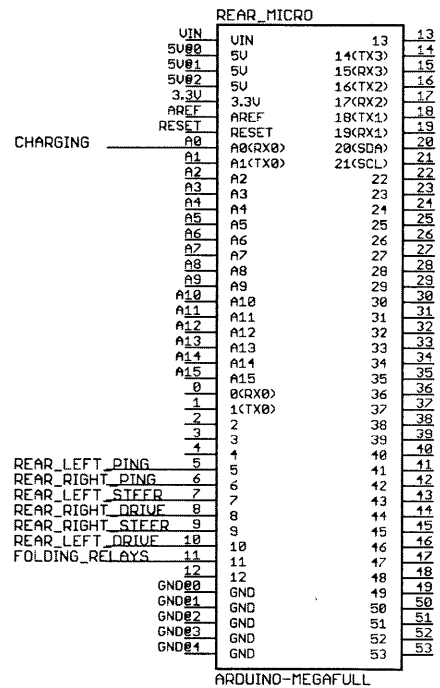
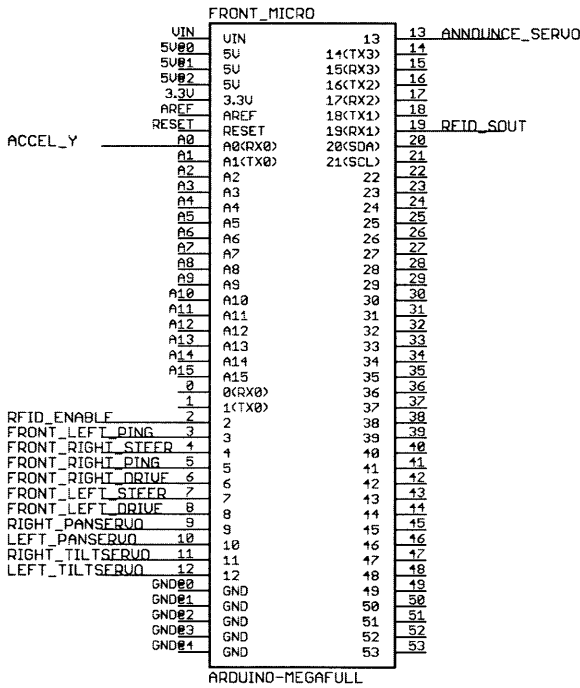
Servo Angle	RAW Steering Angle	Traditional Steering Range	
79	-165.77	xxxxxxx	-15°
80	-167.04		
81	-168.32		
82	-169.6		
83	-170.89		
84	-172.18		
85	-173.47		
86	-174.78		
87	-176.08		
88	-177.4		
89	-178.71		
90	179.96	XXXXXXXX	Dead-Ahead
91	178.63		
92	177.29		
93	175.94		
94	174.59		
95	173.23		
96	171.86		
97	170.49		
98	169.11		
99	167.73		
100	166.33		
101	164.94	Xxxxxxxx	+15°
102	163.53		
103	162.12		
104	160.71		
105	159.29		
106	157.87		
107	156.45		
108	155.02		
109	153.6		
110	152.17		
111	150.75		
112	149.32		
113	147.9		
114	146.48		
115	145.07		
116	143.67		
117	142.28		'O-Turn'

[9-D] IO Box Pin Schematics

Note: All pins 1,2,6, and 7 were unused for power transmission and left open for expansion of signal lines coming into IO Boxes.



- 1 - BLACK
- 2 - GREY
- 3 - BLUE
- 4 - YELLOW
- 5 - RED
- 6 - WHITE
- 7 - PURPLE
- 8 - GREEN
- 9 - ORANGE



[9-E] Castle Mamba Max Pro ESC Settings



Castle-Link Program Settings Report

Title: AEVITA MMP ESC Settings
Date: 6/26/2012 4:17:50 PM

Basic

Cutoff Voltage	Auto Li-Po (Default)
Auto-Lipo Volts/Cell	3.2 Volts/Cell (Default)
Reverse Type	Crawler Reverse
Motor Direction	Normal (Default)
Power-On Warning Beep	Beep Disabled
Brake Amount	25%
Drag Brake	0%, Disabled (Default)
Full Reverse	0.552 ms
Neutral	1.490 ms
Full Forward	1.988 ms

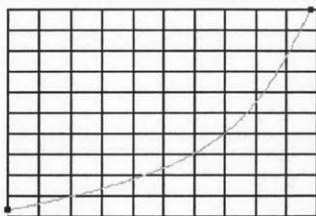
Power

Max Forward Power	40%
Max Reverse Power	40%
Punch Control	60%
Torque Control Value	0
Torque Control Electrical Motor kV	Not Determined

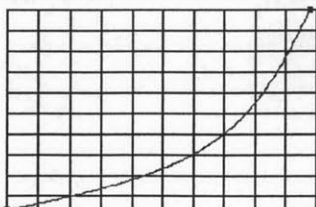
Advanced

BEC Voltage	6.0V
Arming Time	1s
Throttle Dead Band	Average (0.1000 ms) (Default)
Start Power	High
Sensorless Motor Timing	Highest (20)
Motor Type	Smart Sense™ Brushless (Default)

Throttle Curve

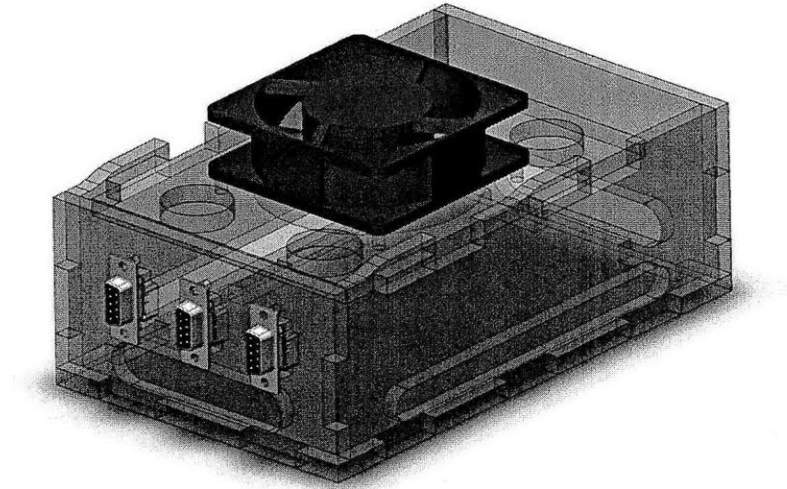


Brake Curve

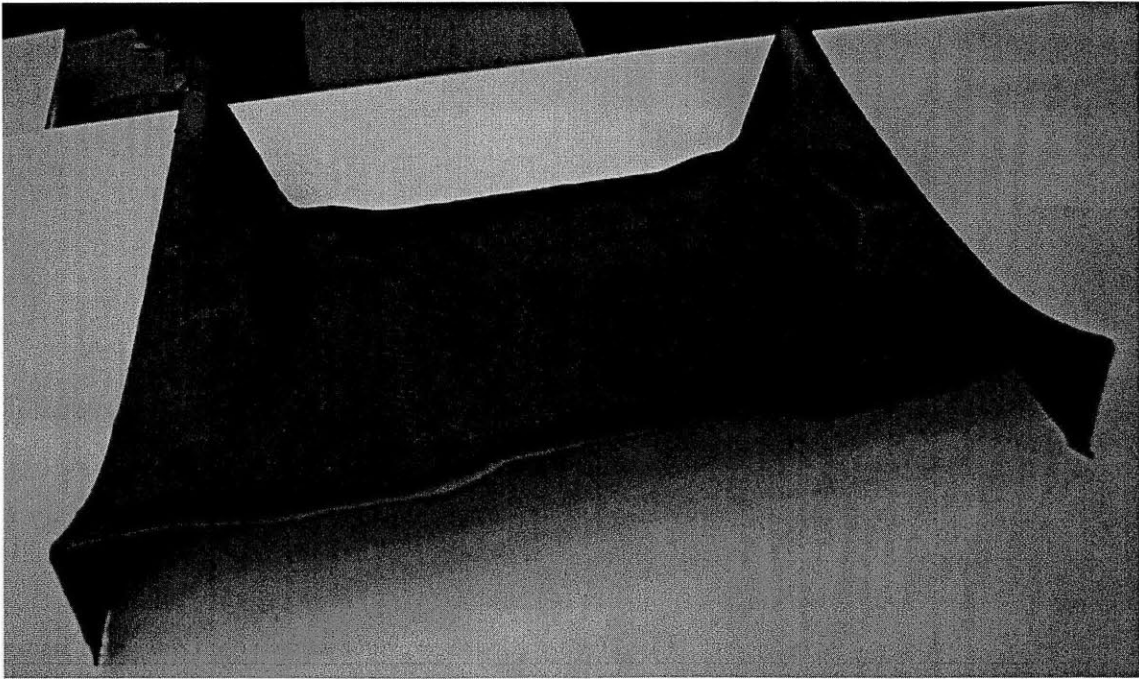


[9-F] Additional Design Images

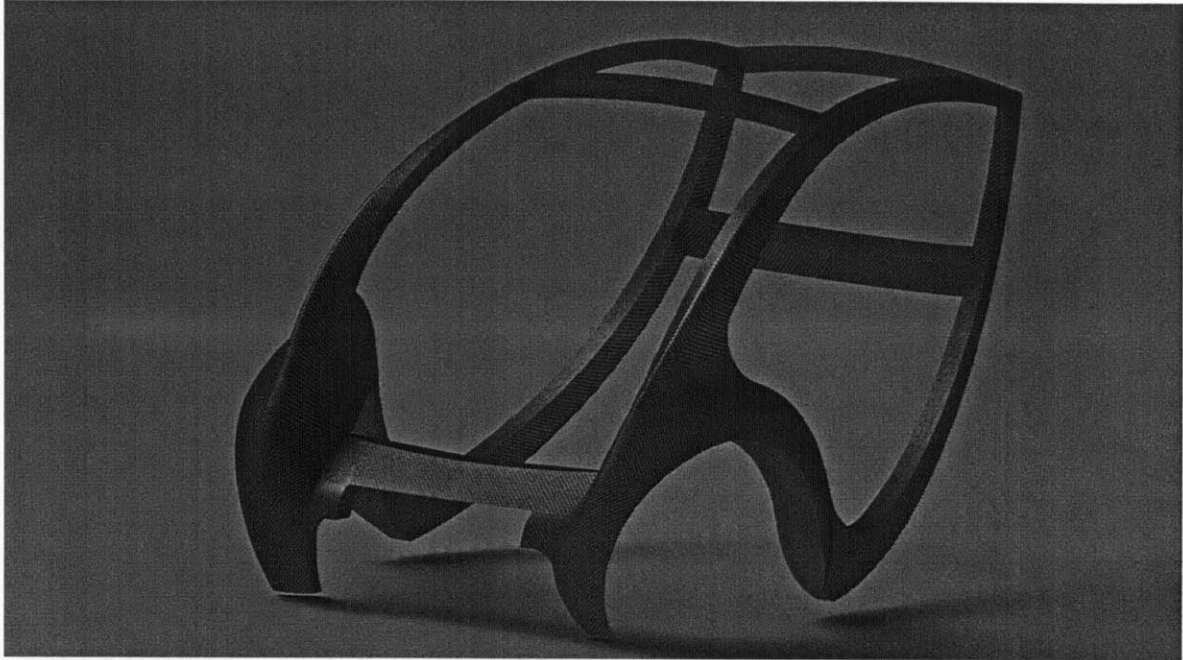
IO Box



CFRP Top Member



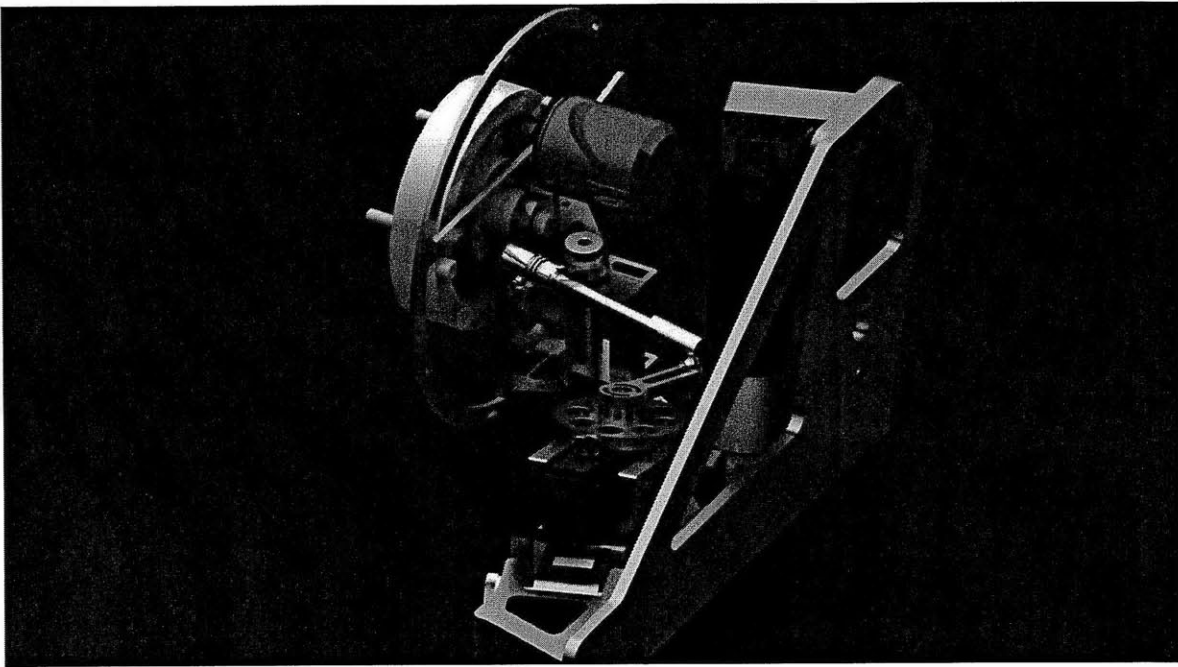
2GHS Exoskeleton



K2 Energy K2B24V10EB 24V 10Ah Lithium Iron Phosphate Battery



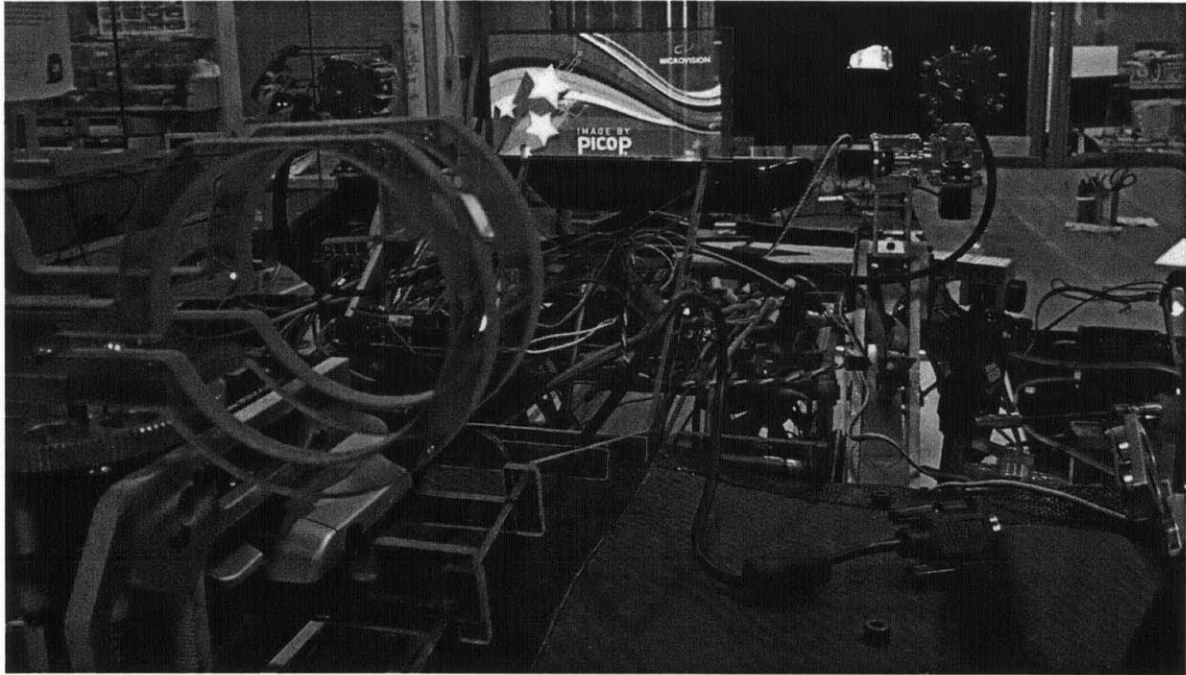
Robot Wheel Rendered



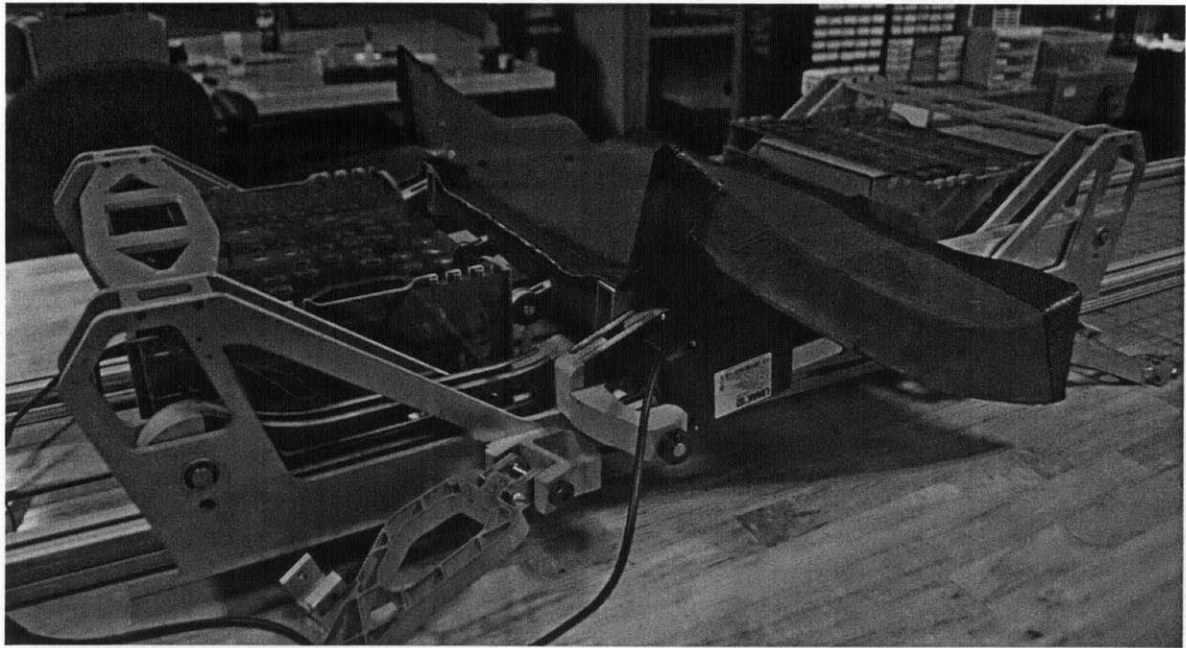
Hacker A50-14L V2.0 BLDC Outrunner Motor



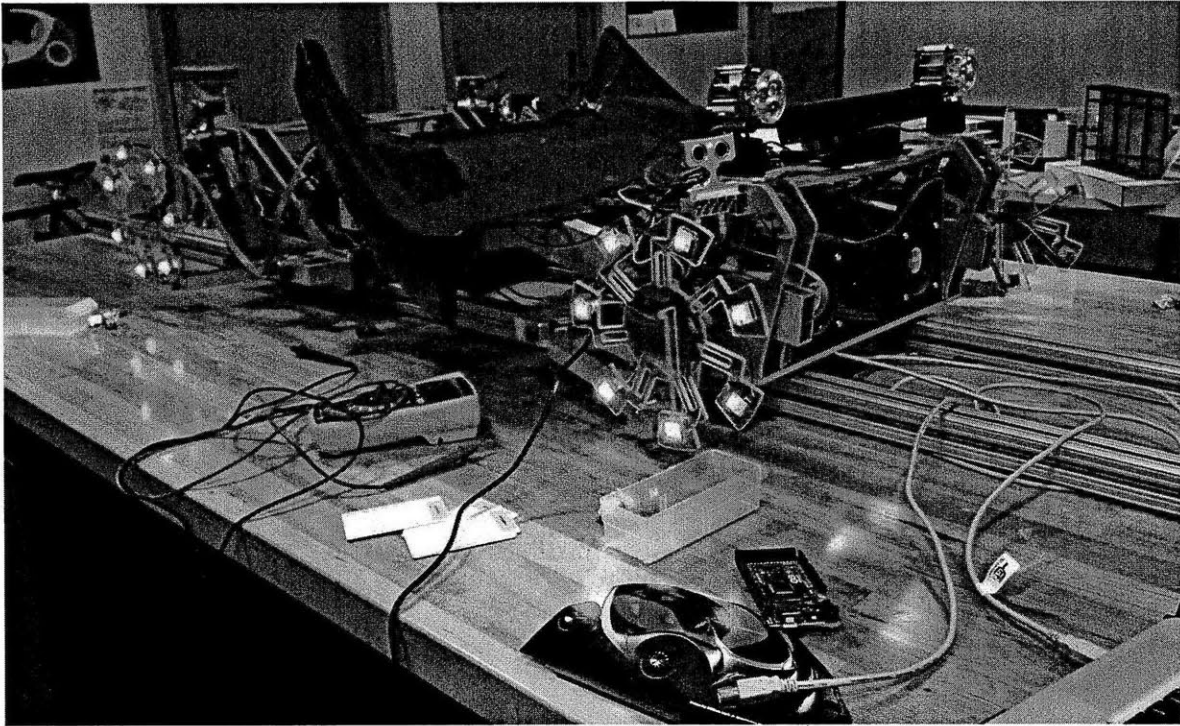
HUD Display w/ Translucent Reflective Panel and laser projector by Microvision



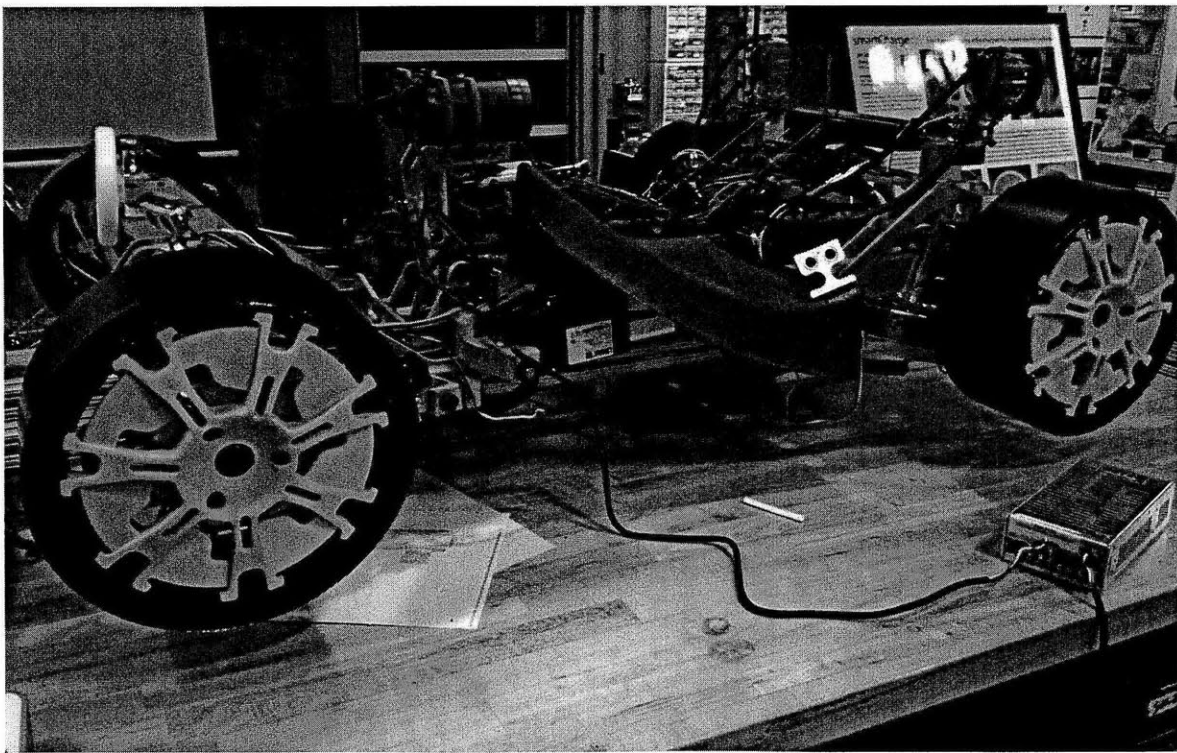
ÆVITA – Early stages



ÆVITA – Early Stages



ÆVITA - Current



[9-G] AETP C# Code

MainWindow.xaml

```
<Window x:Class="AEVITA.MainWindow"

        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

        Title="AEVITA Tracker" Height="573" Width="728" Loaded="Window_Loaded"
        Closing="Window_Closing" xmlns:my="clr-
        namespace:Microsoft.Samples.Kinect.WpfViewers;assembly=Microsoft.Samples.Kinect.W
        pfViewers" ResizeMode="NoResize" Background="#D1000000">

    <Grid Height="515" Width="698">

        <my:KinectSensorChooser HorizontalAlignment="Left" Margin="185,43,0,0"
        Name="kinectSensorChooser1" VerticalAlignment="Top" Width="328" />

        <my:KinectColorViewer HorizontalAlignment="Left"
        Name="kinectColorViewer1" VerticalAlignment="Top" Height="240" Width="320"
        Kinect="{Binding ElementName=kinectSensorChooser1, Path=Kinect}"
        Margin="0,13,0,0" Loaded="kinectColorViewer1_Loaded" />

        <my:KinectSkeletonViewer HorizontalAlignment="Left" Margin="379,13,0,0"
        Name="kinectSkeletonViewer1" VerticalAlignment="Top" Height="240" Width="320"
        Kinect="{Binding ElementName=kinectSensorChooser1, Path=Kinect}" />

        <TextBlock Height="36" HorizontalAlignment="Left" Margin="41,260,0,0"
        Name="textBlock1" Text="" VerticalAlignment="Top" Width="614" FontFamily="Arial"
        FontSize="12" TextAlignment="Left" FontWeight="Normal" FontStyle="Normal"
        Foreground="White" />

        <TextBlock Height="36" HorizontalAlignment="Left" Margin="41,310,0,0"
        Name="textBlock2" Text="" VerticalAlignment="Top" Width="614" FontFamily="Arial"
        FontSize="12" TextAlignment="Left" FontWeight="Normal" FontStyle="Normal"
        Foreground="White" />

        <TextBlock Height="36" HorizontalAlignment="Left" Margin="41,360,0,0"
        Name="textBlock3" Text="" VerticalAlignment="Top" Width="614" FontFamily="Arial"
        FontSize="12" TextAlignment="Left" FontWeight="Normal" FontStyle="Normal"
        Foreground="White" />

        <TextBlock Height="36" HorizontalAlignment="Left" Margin="168,410,0,0"
        Name="textBlock4" Text="" VerticalAlignment="Top" Width="487" FontFamily="Arial"
        FontSize="30" TextAlignment="Left" FontWeight="Normal" FontStyle="Normal"
        Foreground="White" />

        <TextBlock Height="36" HorizontalAlignment="Left" Margin="168,460,0,0"
        Name="textBlock5" Text="" VerticalAlignment="Top" Width="487" FontFamily="Arial"
        FontSize="30" TextAlignment="Left" FontWeight="Normal" FontStyle="Normal"
        Foreground="White" />

        <TextBlock Height="60" HorizontalAlignment="Left" Margin="41,423,0,0"
        Name="textBlock6" Text="" VerticalAlignment="Top" Width="101" FontFamily="Arial"
        FontSize="50" TextAlignment="Left" FontWeight="Normal" FontStyle="Normal"
        Foreground="White" />
    </Grid>
</Window>
```


MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Kinect;
using System.IO.Ports;
using System.Diagnostics;
using Coding4Fun.Kinect.Wpf;
using System.Media;
using SlimDX;
using SlimDX.XInput;

namespace AEVITA
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            char err = 'e';
            int blinkError = 0;
            char read;
            SoundPlayer player_safe;
            SoundPlayer player_also_safe;
            SoundPlayer player_welcome;
            SoundPlayer player_alarm;
            SoundPlayer player_driveOff;
            SoundPlayer player_invalid_ID;
            SoundPlayer player_confirm;

            bool startEnable = false;
            bool closing = false;
            Skeleton watch;
            const int skeletonCount = 6;
            Skeleton[] allSkeletons = new Skeleton[skeletonCount];
            float headPosX, headPosY, headPosZ;
            float x_rel_left, x_rel_right, r_rel_left, r_rel_right;
            float servo_dist = 0.24F; //distance from kinect center in meters
            float leftTilt, leftPan, rightTilt, rightPan, announcePan;
            float prevLeftTilt, prevLeftPan, prevRightTilt, prevRightPan;
            SerialPort serial;
            Stopwatch sw = new Stopwatch();
            int maxID = 0;
            int max;
            Skeleton current;
        }
    }
}
```

```

        Skeleton maxSkeleton;
        DateTime timeNow;
        TimeSpan trackTime = new TimeSpan(0, 0, 20); //track a single skeleton
for 6 seconds
        TimeSpan announceTime = new TimeSpan(0, 0, 3); //allow a 3 second window
for an announcement
        Dictionary<int, DateTime> skeletonTimes = new Dictionary<int,
DateTime>();
        int userID;
        int tracking;
        int maxAnnounceID = 0;
        int people = 0;
        Stopwatch announce;

        String toWrite;
        int bytes;
        int byteOverflowCounter = 0;
        System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();

        int newPerson = 0;
        Controller c;
        State state;
        Vibration vibe;

        int fold = 0;
        int throttle = 90;
        int adjSteering = 0;
        double leftFrontSteer, rightFrontSteer, leftRearSteer, rightRearSteer;
right, C rear left, D rear right
        double A, B, C, D; //These are raw wheel angles - A front left, B front
        int driveModeSelector = 0;
        int headlights = 0;
        Stopwatch debounce;
        bool BButtonAlreadyPressed;
        bool blinkMSelectorAlreadyChanged;
        int blinkMSelector = 0;
        Stopwatch BButtonTimer;
        Stopwatch ModeTimer;
        Stopwatch SkeletonInWayTimer;
        bool alreadySubmitted;
        int annoyanceSequence;
        Stopwatch AnnoyanceTimer;
        Stopwatch AnnoyanceWheelTimer;

        int wheelBase = 944;
        int track = 800;

        //correction factors for the servos
        int lpcf = 7;
        int ltcf = -10;
        int rpcf = 2;
        int rtcf = -20;

        List<int> order = new List<int>();
        int numCases = 10; //these cases will be numbered 1 through numCases
        int temp, exchange;
        int currentTestIndex = -1;
        int previousMode = -1;
        int currentMode = 0; //start in free control mode
        string finishedModes = "";
        bool allOtherOn, eyesOn, talkOn;
        int pupilEnable = 0;
        int specialMode; //0 for none, 1 for aggression, 2 for submission, 3 for
annoyance

```

```

int aggressionState = 0;
double pathWidth = 0.5;
bool skeletonInWay = false;

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    kinectSensorChooser1.KinectSensorChanged += new
DependencyPropertyChangedEventHandler(kinectSensorChooser1_KinectSensorChanged);
    userTest();
    serial = new SerialPort("COM4", 19200);
    serial.Open();
    player_safe = new SoundPlayer(@"C:\Users\AEVITA\Documents\Kinect
Support Files\sounds\safe.wav");
    player_also_safe = new SoundPlayer(@"C:\Users\AEVITA\Documents\Kinect
Support Files\sounds\also_safe.wav");
    player_welcome = new SoundPlayer(@"C:\Users\AEVITA\Documents\Kinect
Support Files\sounds\system_loaded.wav");
    player_alarm = new SoundPlayer(@"C:\Users\AEVITA\Documents\Kinect
Support Files\sounds\alarm.wav");
    player_driveOff = new SoundPlayer(@"C:\Users\AEVITA\Documents\Kinect
Support Files\sounds\driveoff.wav");
    player_invalid_ID = new
SoundPlayer(@"C:\Users\AEVITA\Documents\Kinect Support Files\sounds\beep-
10.wav");
    player_confirm = new SoundPlayer(@"C:\Users\AEVITA\Documents\Kinect
Support Files\sounds\confirm.wav");
    announce = new Stopwatch();
    announce.Start();
    c = new Controller(UserIndex.One);
    vibe = new Vibration();
    debounce = new Stopwatch();
    debounce.Start();
    BButtonTimer = new Stopwatch();
    BButtonTimer.Start();
    ModeTimer = new Stopwatch();
    ModeTimer.Start();
    SkeletonInWayTimer = new Stopwatch();
    AnnoyanceTimer = new Stopwatch();
    AnnoyanceWheelTimer = new Stopwatch();
}

void userTest()
{
    for (int i = 1; i <= numCases; i++)
    {
        order.Add(i);
    }

    //implementation of Fisher-Yates shuffle
    Random random = new Random();
    for (int i = numCases-1; i >= 0; i--) {
        exchange = random.Next(i+1);
        temp = order[exchange];
        order[exchange] = order[i];
        order[i] = temp;
    }

    string orderString = "";
    foreach (int test in order)
        orderString += (test + " ");
    textBlock4.Text = orderString;
    textBlock6.Text = currentMode.ToString();
}

```

```

void KinectSensorChooser1_KinectSensorChanged(object sender,
DependencyPropertyChangedEventArgs e)
{
    KinectSensor oldSensor = (KinectSensor)e.OldValue;
    StopKinect(oldSensor);

    KinectSensor sensor = (KinectSensor)e.NewValue;

    if (sensor == null)
    {
        return;
    }

    var parameters = new TransformSmoothParameters
    {
        Smoothing = 0.0f, //changed from 0.3f
        Correction = 1.0f, //changed from 0.0f; note that higher values
are faster to correct to the raw data (ranges from 0 to 1)
        Prediction = 0.0f, //don't know exactly what this does
        JitterRadius = 3.0f, //changed from 1.0f
        MaxDeviationRadius = 0.0f //changed from 0.5f
    };

    sensor.SkeletonStream.Enable(parameters);
    sensor.AllFramesReady += new
EventHandler<AllFramesReadyEventArgs>(sensor_AllFramesReady);
    sensor.ColorStream.Enable();
    sensor.DepthStream.Enable();

    try
    {
        sensor.Start();
        kinectSensorChooser1.Kinect.ElevationAngle = 5;
        player_welcome.Play();
    }
    catch (System.IO.IOException)
    {
        kinectSensorChooser1.AppConflictOccurred();
    }
}

void sensor_AllFramesReady(object sender, AllFramesReadyEventArgs e){
    getSerialInfo();
    getControllerBasicInfo();
    if (currentMode != previousMode)
    {
        fold = 0;
        headlights = 0;
        getSystemBooleans(currentMode);
        previousMode = currentMode;
    }

    //r = Convert.ToChar(serial.ReadChar());
    //if (r == 'R')
    //{
    //    Process.Start(Application.ResourceAssembly.Location);
    //    Application.Current.Shutdown();
    //}

    if (closing)
    {
        return;
    }
}

```

```

//if (true)
if (startEnable)
{
    watch = GetSkeletonToTrack(e);
    if (watch != null) {
        tracking = 1;
        headPosX = watch.Joints[JointType.ShoulderCenter].Position.X;
        headPosY = watch.Joints[JointType.ShoulderCenter].Position.Y;
        headPosZ = watch.Joints[JointType.ShoulderCenter].Position.Z;
    }
    skeletonInWay = IsThereASkeletonInWay();
    switch (specialMode)
    {
        case 0:
            getControllerAllOtherInfo();
            if (watch != null && eyesOn)
            {
                CalculateEyeAngles(headPosX, headPosY, headPosZ);
            }
            else
            {
                CalculateEyeAnglesNoFollow();
            }
            if (watch != null && talkOn)
            {
                CalculateTalk(headPosX, headPosY, headPosZ);
            }
            else
            {
                CalculateTalkNoFollow();
            }
            break;
        case 1:
            CalculateTalkNoFollow();
            if (SkeletonInWayTimer.ElapsedMilliseconds > 1000)
            {
                fold = 1; //if the person has stood in the danger
zone for 2 seconds, haunch up
            }
            else
            {
                fold = 2; //if the person has moved out of the
danger zone, return to flat state
            }
            if (skeletonInWay)
            { //there is a skeleton in the danger zone
                CalculateAggressionMode();
            }
            else if (watch != null)
            { //there is no skeleton in the danger zone, but there
is one being tracked
                CalculateEyeAngles(headPosX, headPosY, headPosZ);
                AllOtherOff();
            }
            else
            { //no skeleton is being tracked
                CalculateEyeAnglesNoFollow();
                AllOtherOff();
            }
            break;
        case 2:
            AllOtherOff();
            CalculateTalkNoFollow();
            if (SkeletonInWayTimer.ElapsedMilliseconds > 1000)

```

```

        {
            fold = 2; //if skeleton has been in danger zone for
more than 2 seconds, unfold
            alreadySubmitted = true; //once the car has
submitted once, do not repeat the setup phase
        }
        else if (!alreadySubmitted)
        {
            fold = 1; //setup by having the car haunch up
        }
        else
        {
            fold = 0; //if the car has already submitted once,
don't have it setup again
        }
        if (skeletonInWay)
        {
            CalculateSubmissionMode();
        }
        else if (watch != null)
        {
            CalculateEyeAngles(headPosX, headPosY, headPosZ);
        }
        else
        {
            CalculateEyeAnglesNoFollow();
        }
        break;
    case 3:
        CalculateTalkNoFollow();
        CalculateEyeAnglesNoFollow();
        if (skeletonInWay)
        {
            CalculateAnnoyanceMode();
        }
        else
        {
            AllOtherOff();
        }
        break;
    }
}
else
{
    AllOtherOff();
}

    toWrite = "W" + Convert.ToString(tracking) + "," +
Convert.ToString(Convert.ToInt32(leftPan)) + "," +
Convert.ToString(Convert.ToInt32(leftTilt)) + "," +
Convert.ToString(Convert.ToInt32(rightPan)) + "," +
Convert.ToString(Convert.ToInt32(rightTilt)) + "," +
Convert.ToString(Convert.ToInt32(announcePan)) + "," +
Convert.ToString(newPerson) + "," + Convert.ToString(fold) + ","
        + Convert.ToString(Convert.ToInt32(leftFrontSteer)) +
"," + Convert.ToString(Convert.ToInt32(rightFrontSteer)) + "," +
Convert.ToString(Convert.ToInt32(leftRearSteer)) + "," +
Convert.ToString(Convert.ToInt32(rightRearSteer)) + "," +
Convert.ToString(throttle) + "," + Convert.ToString(driveModeSelector) + "," +
Convert.ToString(headlights) + ","
        + Convert.ToString(blinkMSelector) + "," +
Convert.ToString(pupilEnable) + "," + Convert.ToString(specialMode) + ",";
    serial.WriteLine(toWrite);
    bytes = encoding.GetBytes(toWrite).Length;

```

```

        textBlock1.Text = "Enabled: " + Convert.ToString(startEnable) + "
Bytes: " + Convert.ToString(bytes) + "Byte Errors: " +
Convert.ToString(byteOverflowCounter) + " Announce: " +
Convert.ToString(announcePan) + " X: " + Convert.ToString(headPosX) + " Y: " +
Convert.ToString(headPosY) + " Z: " + Convert.ToString(headPosZ) + " ID: " +
Convert.ToString(userID);
        textBlock2.Text = "New Person?: " + Convert.ToString(newPerson) + "
Fold: " + Convert.ToString(fold) + " Throttle: " + Convert.ToString(throttle) + "
SteeringRaw: " + Convert.ToString(LeftThumbX) + " AdjustedSteering: " +
Convert.ToString(adjSteering) + " Headlights: " + Convert.ToString(headlights) +
" BlinkM: " + Convert.ToString(blinkMSelector) + " InWayTimer: " +
Convert.ToString(SkeletonInWayTimer.ElapsedMilliseconds);
        textBlock3.Text = "A: " + Convert.ToString(A) + " B: " +
Convert.ToString(B) + " C: " + Convert.ToString(C) + " D: " + Convert.ToString(D)
+ " Blink Error Count: " + Convert.ToString(blinkError) + " Drive: " +
Convert.ToString(allOtherOn) + " Eyes: " + Convert.ToString(eyesOn) + " Talk: " +
Convert.ToString(talkOn) + " Pupil: " + Convert.ToString(pupilEnable) + " SM: " +
Convert.ToString(specialMode) + " DZ: " +
Convert.ToString(Convert.ToInt32(skeletonInWay));
    }

    void serial_DataReceived(object sender, SerialDataReceivedEventArgs e)
    {

    }

    void StopKinect(KinectSensor sensor)
    {
        if (sensor != null)
        {
            sensor.Stop();
        }
    }

    bool IsThereASkeletonInWay()
    {
        for (int index = 0; index < skeletonCount; index++)
        {
            current = allSkeletons[index];
            if (current.TrackingState == SkeletonTrackingState.Tracked &&
Math.Abs(current.Joints[JointType.ShoulderCenter].Position.X) < pathWidth)
            {
                pupilEnable = 0; //don't want to be recognizing additional
people who come into the FOV if there is a skeleton in the path currently
                if (!SkeletonInWayTimer.IsRunning) {
                    SkeletonInWayTimer.Start();
                }
                return true;
            }
        }
        SkeletonInWayTimer.Reset();
        return false;
    }

    Skeleton GetSkeletonToTrack(AllFramesReadyEventArgs e)
    {
        using (SkeletonFrame skeletonFrameData = e.OpenSkeletonFrame())
        {
            if (skeletonFrameData == null)
            {
                return null;
            }
        }
    }

```

```

being tracked //can assume from here on that there are at least some skeletons
skeletonFrameData.CopySkeletonDataTo(allSkeletons);
timeNow = DateTime.Now;
max = 0;
maxSkeleton = null;

people = 0;

frame //find the maximum ID among the skeletons in the current skeleton
for (int index = 0; index < skeletonCount; index++)
{
    current = allSkeletons[index];
    if (current != null)
    {
        if (current.TrackingId > 0)
        {
            people++;
        }
        if (current.TrackingId > max)
        {
            max = current.TrackingId;
            maxSkeleton = current;
        }
    }
}
try
{
    if (maxSkeleton == null)
    {
        return null;
    }
    else if (max > maxID) //someone has just moved into the view
    {
        newPerson = 1;
        maxID = max;
        skeletonTimes.Add(maxID, timeNow.Add(trackTime));
        return maxSkeleton;
    }
    else if (timeNow.CompareTo(skeletonTimes[max]) < 0) //the
corresponding skeleton has been tracked for less than 3 seconds
    {
        return maxSkeleton;
    }
    else
    {
        return null;
    }
}

catch (KeyNotFoundException ex)
{
    throw new KeyNotFoundException(max.ToString() + " was not
found in the dictionary");
    //Console.WriteLine(ex);
}

}

void AllOtherOff()
{

```



```

        throttle = 90;
        leftFrontSteer = 90;
        rightFrontSteer = 90;
        leftRearSteer = 90;
        rightRearSteer = 90;
    }

void CalculateAggressionMode()
{
    leftPan = 90 + lpcf;
    rightPan = 90 + rpcf;
    if (ModeTimer.ElapsedMilliseconds > 500)
    {
        switch (aggressionState)
        {
            case 0: //wheels inward and eyes down
                leftTilt = 90 + ltcf - 20;
                rightTilt = 90 + rtcf + 20;
                leftFrontSteer = 105;
                leftRearSteer = 90;
                rightFrontSteer = 75;
                rightRearSteer = 90;
                aggressionState = 1;
                ModeTimer.Restart();
                break;
            case 1: //wheels outward and eyes up
                leftTilt = 90 + ltcf + 20;
                rightTilt = 90 + rtcf - 20;
                leftFrontSteer = 75;
                leftRearSteer = 90;
                rightFrontSteer = 105;
                rightRearSteer = 90;
                aggressionState = 0;
                ModeTimer.Restart();
                break;
        }
    }
}

void CalculateSubmissionMode()
{
    leftTilt = 90 + ltcf - 20;
    rightTilt = 90 + rtcf + 20;
    leftPan = 90 + lpcf + 20;
    rightPan = 90 + rpcf - 20;
    leftFrontSteer = 105;
    leftRearSteer = 90;
    rightFrontSteer = 75;
    rightRearSteer = 90;
}

void CalculateAnnoyanceMode()
{
    if (SkeletonInWayTimer.ElapsedMilliseconds > 2000 &&
        annoyanceSequence == 0)
    {
        player_alarm.Play();
        annoyanceSequence++;
    }
    else if (SkeletonInWayTimer.ElapsedMilliseconds > 4000 &&
        annoyanceSequence == 1) {
        player_alarm.Play();
        AnnoyanceTimer.Start();
        headlights = 1;
    }
}

```

```

        annoyanceSequence++;
    }
    else if (SkeletonInWayTimer.ElapsedMilliseconds > 6000 &&
annoyanceSequence == 2)
    {
        player_alarm.Play();
        AnnoyanceWheelTimer.Start();
        throttle = 120;
        annoyanceSequence++;
    }

    if (SkeletonInWayTimer.ElapsedMilliseconds > 6000 &&
AnnoyanceTimer.ElapsedMilliseconds > 500)
    {
        AnnoyanceTimer.Restart();
        if (throttle == 90)
        {
            throttle = 120;
        }
        else
        {
            throttle = 90;
        }
        headlights = 1 - headlights;
    }
    else if (SkeletonInWayTimer.ElapsedMilliseconds > 4000 &&
AnnoyanceTimer.ElapsedMilliseconds > 500)
    {
        AnnoyanceTimer.Restart();
        headlights = 1 - headlights;
    }
}

//headlights should point straight ahead
void CalculateTalkNoFollow()
{
    announcePan = 1500;
}

void CalculateEyeAnglesNoFollow()
{
    leftPan = 90 + lpcf;
    leftTilt = 90 + ltcf;
    rightPan = 90 + rpcf;
    rightTilt = 90 + rtcf;
}

//calculate angle for speaker
void CalculateTalk(double x, double y, double z)
{
    announcePan = (float)(Math.Atan((z + 2.0) / x) * 180 / Math.PI / 1.75
+ 1500);
    userID = watch.TrackingId;    //Get the tracking ID

    //Determine which soundtrack to play, if any
    //More than three seconds must have passed since the start of the
last announcement
    //There should not have already been an announcement for this user
    if ((announce.Elapsed).CompareTo(announceTime) > 0 && userID >
maxAnnounceID)
    {
        if (people >= 2)
        {
            player_also_safe.Play();

```

```

        announce.Restart();
    }
    else
    {
        player_safe.Play();
        announce.Restart();
    }
    maxAnnounceID = userID;
}

//calculate angles for eyes
void CalculateEyeAngles(double x, double y, double z)
{
    x_rel_left = (float)(x - servo_dist);
    x_rel_right = (float)(x + servo_dist);
    r_rel_left = (float)(Math.Sqrt(Math.Pow(x_rel_left, 2) + Math.Pow(z,
2)));
    r_rel_right = (float)(Math.Sqrt(Math.Pow(x_rel_right, 2) +
Math.Pow(z, 2)));

    leftPan = (float)(Math.Atan2(z, x_rel_left) * 180 / Math.PI + lpcf);
    leftTilt = (float)(90 + lpcf + Math.Atan2(y, r_rel_left) * 180 /
Math.PI);
    rightPan = (float)(Math.Atan2(z, x_rel_right) * 180 / Math.PI +
rpcf);
    rightTilt = (float)(90 + rtcf - Math.Atan2(y, r_rel_left) * 180 /
Math.PI);

    //prevent the eyes from staring at each other or taking on some other
impossible possible
    if (Math.Abs(leftPan - (90 + lpcf)) > 40 || Math.Abs(leftTilt - (90 +
lpcf)) > 40 || Math.Abs(rightPan - (90 + rpcf)) > 40 || Math.Abs(rightTilt - (90
+ rtcf)) > 40)
    {
        leftPan = prevLeftPan;
        leftTilt = prevLeftTilt;
        rightPan = prevRightPan;
        rightTilt = prevRightTilt;
    }

    prevLeftPan = leftPan;
    prevLeftTilt = leftTilt;
    prevRightPan = rightPan;
    prevRightTilt = rightTilt;
}

private void getControllerAllOtherInfo()
{
    if (c.IsConnected)
    {
        state = c.GetState();
        if (AButton && XButton && driveModeSelector == 0 &&
debounce.ElapsedMilliseconds >= 500)
        {
            debounce.Restart();
            driveModeSelector = 1; //enter o-turn mode
        }
        else if (AButton && XButton && driveModeSelector == 1 &&
debounce.ElapsedMilliseconds >= 500)
        {
            debounce.Restart();
            driveModeSelector = 0; //exit o-turn mode
        }
    }
}

```

```

    if (BButton)
    {
        if (!BButtonAlreadyPressed)
        {
            BButtonTimer.Restart();
            BButtonAlreadyPressed = true;
            blinkMSelectorAlreadyChanged = false;
        }
        else if (BButtonAlreadyPressed &&
BButtonTimer.ElapsedMilliseconds >= 2000 && !blinkMSelectorAlreadyChanged)
        {
            blinkMSelector = Math.Abs(1 - blinkMSelector);
            blinkMSelectorAlreadyChanged = true;
        }
    }
    else
    {
        BButtonAlreadyPressed = false;
    }
    //headlight control
    if (YButton && headlights == 0 && debounce.ElapsedMilliseconds >=
500)
    {
        debounce.Restart();
        headlights = 1;
    }
    else if (YButton && headlights == 1 &&
debounce.ElapsedMilliseconds >= 500)
    {
        debounce.Restart();
        headlights = 0;
    }

    if ((RightShoulderButton && LeftShoulderButton) ||
(!RightShoulderButton && !LeftShoulderButton)) //if both or neither shoulder
buttons pressed, do nothing
    {
        fold = 0;
        vibe.LeftMotorSpeed = 0;
        vibe.RightMotorSpeed = 0;
        c.SetVibration(vibe);
    }
    else if (RightShoulderButton && !LeftShoulderButton) //fold if
only the right shoulder button pressed
    {
        fold = 1;
        vibe.LeftMotorSpeed = 0;
        vibe.RightMotorSpeed = 20000;
        c.SetVibration(vibe);
    }
    else if (!RightShoulderButton && LeftShoulderButton) //unfold if
only the left shoulder button pressed
    {
        fold = 2;
        vibe.LeftMotorSpeed = 20000;
        vibe.RightMotorSpeed = 0;
        c.SetVibration(vibe);
    }
    //vibration for throttle and brake
    if (LeftTrigger > 0)
    {
        vibe.RightMotorSpeed = Convert.ToUInt16(LeftTrigger * 235);
        c.SetVibration(vibe);
    }

```

```

    }
    if (RightTrigger > 0)
    {
        vibe.RightMotorSpeed = Convert.ToUInt16(RightTrigger * 235);
        c.SetVibration(vibe);
    }

    throttle = 90 + Convert.ToInt16(RightTrigger) * 90 / 255 -
Convert.ToInt16(LeftTrigger) * 90 / 255;

    //if (tracking == 1 && throttle > 90)
    // {
    //     // sw.Start();
    //     // while (sw.IsRunning)
    //     // {
    //     // player_alarm.Play();
    //     // if (sw.ElapsedMilliseconds >= 500)
    //     // {
    //     //     player_driveOff.Play();
    //     //     sw.Reset();
    //     //     break;
    //     // }
    //     // }
    // }
    //}

    if (LeftThumbX <= 7000 && LeftThumbX >= -7000) //deadzone -
ignoring small wiggle in the thumbstick
    {
        adjSteering = 0;
    }
    else
    {
        adjSteering = LeftThumbX * 100 / 32768;
    }

    switch (driveModeSelector)
    {
        case 0:
            if (adjSteering < 0)
            {
                A = 15 * adjSteering / 100;
                B = -Math.Atan((wheelBase / 2) / ((wheelBase / 2) /
Math.Tan(-0.26 * adjSteering / 100) + track)) * 180 / Math.PI;
                C = -15 * adjSteering / 100;
                D = Math.Atan((wheelBase / 2) / ((wheelBase / 2) /
Math.Tan(-0.26 * adjSteering / 100) + track)) * 180 / Math.PI;
            }
            else if (adjSteering > 0)
            {
                A = Math.Atan((wheelBase / 2) / ((wheelBase / 2) /
Math.Tan(0.26 * adjSteering / 100) + track)) * 180 / Math.PI;
                B = 15 * adjSteering / 100;
                C = -Math.Atan((wheelBase / 2) / ((wheelBase / 2) /
Math.Tan(0.26 * adjSteering / 100) + track)) * 180 / Math.PI;
                D = -15 * adjSteering / 100;
            }
            else
            {
                A = B = C = D = 0;
            }
            leftFrontSteer = 0.75 * (A) + 90;
            rightFrontSteer = 0.75 * (B) + 90;

```

```

        leftRearSteer = 0.75 * (C) + 90;
        rightRearSteer = 0.75 * (D) + 90;
        break;
    case 1:
        leftFrontSteer = rightRearSteer = 117;
        rightFrontSteer = leftRearSteer = 63;
        break;
    }
}
}

private void getControllerBasicInfo()
{
    if (c.IsConnected)
    {
        state = c.GetState();
        if (StartButton && debounce.ElapsedMilliseconds >= 500)
        {
            debounce.Restart();
            currentTestIndex++;
            if (currentTestIndex < numCases)
            {
                currentMode = order[currentTestIndex];
                textBlock6.Text = currentMode.ToString();
                finishedModes += (currentMode + " ");
                textBlock5.Text = finishedModes;
            }
            else
            {
                currentMode = 0;
                textBlock6.Text = currentMode.ToString();
            }
        }

        if (BackButton && debounce.ElapsedMilliseconds >= 500)
        {
            debounce.Restart();
            currentMode = 0;
            textBlock6.Text = currentMode.ToString();
        }
    }
}

private void getSystemBooleans(int mode)
{
    switch (mode)
    {
        case 0: eyesOn = true; talkOn = true; pupilEnable = 1;
specialMode = 0; break;
        case 1: eyesOn = false; talkOn = false; pupilEnable = 0;
specialMode = 0; break;
        case 2: eyesOn = true; talkOn = false; pupilEnable = 0;
specialMode = 0; break;
        case 3: eyesOn = false; talkOn = true; pupilEnable = 0;
specialMode = 0; break;
        case 4: eyesOn = true; talkOn = false; pupilEnable = 1;
specialMode = 0; break;
        case 5: eyesOn = true; talkOn = true; pupilEnable = 1;
specialMode = 0; break;
        case 6: eyesOn = false; talkOn = false; pupilEnable = 0;
specialMode = 1; break;
        case 7: eyesOn = false; talkOn = false; pupilEnable = 0;
specialMode = 2; alreadySubmitted = false; break;
        case 8: eyesOn = false; talkOn = false; pupilEnable = 1;
specialMode = 1; break;
    }
}

```

```

        case 9: eyesOn = false; talkOn = false; pupilEnable = 1;
specialMode = 2; alreadySubmitted = false; break;
        case 10: eyesOn = false; talkOn = false; pupilEnable = 0;
specialMode = 3; annoyanceSequence = 0; break;
    }
}
private void getSerialInfo()
{
    if (serial.BytesToRead > 0)
    {
        read = Convert.ToChar(serial.ReadChar());
        if (read == 'B')
        {
            player_invalid_ID.Play();
        }
        if (read == 'C')
        {
            player_confirm.Play();
            startEnable = true;
        }
        if (read == 'R')
        {
            newPerson = 0; //Arduino received the newPerson
notification, so no longer need to send newPerson = 1
        }
    }
}

private void Window_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
{
    closing = true;
    StopKinect(kinectSensorChooser1.Kinect);
}

private void kinectColorViewer1_Loaded(object sender, RoutedEventArgs e)
{
}
#region Controller State
public bool AButton { get { return
state.Gamepad.Buttons.HasFlag(GamepadButtonFlags.A); } }
public bool BButton { get { return
state.Gamepad.Buttons.HasFlag(GamepadButtonFlags.B); } }
public bool XButton { get { return
state.Gamepad.Buttons.HasFlag(GamepadButtonFlags.X); } }
public bool YButton { get { return
state.Gamepad.Buttons.HasFlag(GamepadButtonFlags.Y); } }
public bool StartButton { get { return
state.Gamepad.Buttons.HasFlag(GamepadButtonFlags.Start); } }
public bool BackButton { get { return
state.Gamepad.Buttons.HasFlag(GamepadButtonFlags.Back); } }
//public static GamepadButtons Buttons { get{ return
state.Gamepad.Buttons; } }
public bool LeftShoulderButton { get { return
state.Gamepad.Buttons.HasFlag(GamepadButtonFlags.LeftShoulder); } }
public byte LeftTrigger { get { return state.Gamepad.LeftTrigger; } }
public bool RightShoulderButton { get { return
state.Gamepad.Buttons.HasFlag(GamepadButtonFlags.RightShoulder); } }
public byte RightTrigger { get { return state.Gamepad.RightTrigger; } }
public short LeftThumbX { get { return state.Gamepad.LeftThumbX; } }
#endregion
}
}

```

[9-H] Arduino Code

Primary Front Micro

```
// Written by: Nicholas Pennycooke | Katharine Daly | Aron Dreyfoos
// Arduino 1.0

// SERIAL    Java + Debug
// SERIAL1   RFID
// SERIAL2   PC Code Interface
// SERIAL3   xBee

#include <I2C.h>
#include <BlinkM_funcs_I2C.h>
#include <Ping.h>
#include <Servo.h>
#include <MemoryFree.h>
#define BUFSIZ 100

int tracking;
Ping ping1 = Ping(3);
Ping ping3 = Ping(5);
int sonarCount = 4;
double proxValues[4];
int i2cStat;
double reading, reading2, reading4;
int dplace;
boolean inDecimal;

int colOffset = 0;
double r = 0;
double g = 0;

unsigned long current_time = 0;
unsigned long previous_time_idle = 0;
unsigned long previous_time_sonars = 0;

int index;
char* parse;
char* p;
double valStore;
const int packageLengthKinect = 18;
double kinectInfo[packageLengthKinect];
const int packageLengthBackSonars = 2;
double backSonarsInfo[packageLengthBackSonars];

int modeSelect = 0;    //starts in freeDrive for now, will start at 0 in future

Servo leftPan, leftTilt, rightPan, rightTilt, announceServo;
double leftPan_value, leftTilt_value, rightPan_value, rightTilt_value;
double lpcf = 7;
double ltcf = -10;
double rpcf = 2;
double rtcf = -20;
double annVal = 1500;
int newPerson = 0;
int fold = 0;
int lf_angle, lb_angle, rf_angle, rb_angle, throttle, drive_mode;
int headlight = 0;
int previousHeadlight = 0;

int defaultScript = 16;
```



```

int enableCuttle = 0;
int lastCuttle = 0;
int enablePupil = 1;

Servo leftFrontSteer, leftFrontThrottle, rightFrontSteer, rightFrontThrottle;

int accelY;           // ~315 is fully unfolded, ~300 is 'haunched up;, ~285 is
fully folded
//int submissionPrep = 0;
//int dangerZone = 0;

// _____
//                               S E T U P
// _____

void setup() {

  I2c.pullup(0);
  I2c.timeOut(250);
  I2c.setSpeed(0);
  BlinkM_beginWithPower(); //the command for powering up the BlinkM's when
connected to Arduino
  delay(100); // wait a bit for things to stabilize
  BlinkM_stopScript(0x00);
  BlinkM_setFadeSpeed(0x00, 8); //fade speed can range from 0 to 255 (higher
numbers are faster)
  BlinkM_fadeToRGB(0x00,255,255,255); //white

  pupilStart();

  // set headlights straight ahead initially
  leftPan.attach(10);
  leftTilt.attach(12);
  rightPan.attach(9);
  rightTilt.attach(11);
  announceServo.attach(13);

  leftPan.write(90 + lpcf);
  leftTilt.write(90 + ltcf);
  rightPan.write(90 + rpcf);
  rightTilt.write(90 + rtcf);
  announceServo.write(annVal);

  leftFrontSteer.attach(7);
  leftFrontThrottle.attach(8);
  rightFrontSteer.attach(4);
  rightFrontThrottle.attach(6);

  leftFrontSteer.write(90);
  leftFrontThrottle.write(90);
  rightFrontSteer.write(90);
  rightFrontThrottle.write(90);

  Serial.begin(19200);
  delay(100);
  Serial.println("Beginning setup.");

  Serial.println("Entering readRFID()");
  Serial.print("Free Memory: ");
  Serial.println(freeMemory());

  Serial2.begin(19200);
  //Serial2.print("R");
  readRFID();
}

```

```

Serial3.begin(38400); //use this serial for the XBee
delay(1000);

  pupilIdle();
}

// _____
//                E N D   S E T U P
// _____

//*****
//*****LOOP*****
//*****

void loop() {
  driveMode();
}

//*****
//*****END-LOOP*****
//*****

// _____
//                LOGIN: RFID
// _____

void readRFID() {
  int val;
  int bytesread=0;
  int ID;
  char code[11]; //where the RFID code will be stored
  char aron[11] = "27000E69CE";
  char nick[11] = "27000E3EDE";
  char katharine[11] = "3F001E96AD";
  char kat[11] = "3F001EC5CC";

  Serial1.begin(2400); // RFID reader SOUT pin connected to Serial1 RX pin at
2400bps
  pinMode(2,OUTPUT); // Set digital pin 2 as OUTPUT to connect it to the RFID
/ENABLE pin
  digitalWrite(2, LOW); // Activate the RFID reader

  //RFID code below adapted from http://www.gumbolabs.org/2009/10/17/parallax-
rfid-reader-arduino/
  while(Serial1.available() <= 0) {} //wait here until RFID card data is
available
  if((val = Serial1.read()) == 10) { //check for start byte
    while(bytesread<10) {
      if(Serial1.available() > 0) {
        if((val=Serial1.read()) == 13) { //13 is the stop byte
          break;
        }
        code[bytesread++] = val;
      }
    }
    code[10]='\0'; //null terminate the code string
  }

  digitalWrite(2, HIGH); // Deactivate the RFID reader
  Serial1.end(); // close Serial1 communication

  //print out the results of the RFID card read
  if (bytesread == 10) {

```

```

Serial.println("Succesfully read the RFID card code.");

if (strcmp(code,aron) == 0){
  ID = 1;
}else if (strcmp(code,katharine) == 0){
  ID = 2;
}else if (strcmp(code,kat) == 0){
  ID = 3;
}else if (strcmp(code,nick) == 0){
  ID = 4;
}else{
  ID = 0;
}

//print user and color
if (ID != 0){
  Serial.print('G');
}

if (ID == 1){
  Serial.println("Aron");
  BlinkM_fadeToRGB(0x00,255,255,255); //white
}
else if (ID == 2){
  Serial.println("Katharine");
  BlinkM_fadeToRGB(0x00,255,0,0); //red
}
else if (ID == 3){
  Serial.println("Kat");
  BlinkM_fadeToRGB(0x00.0.255,00); //green
}
else if (ID == 4){
  Serial.println("Nick");
  BlinkM_fadeToRGB(0x00.0.0.255); //blue
}
else {
  Serial.println("INVALID ID. VEHICLE WILL REMAIN LOCKED.");
  Serial2.print('B'); //Tell PC RFID was rejected - play invalid.wav
  //Flash all wheel lights
  for(int i = 0; i < 4; i++){
    BlinkM_setRGB(0x00,255,0,0);
    delay(167);
    BlinkM_setRGB(0x00,0,0,0);
    delay(167);
  }
  BlinkM_fadeToRGB(0x00.255,255,255);
  Serial.print('X');
  readRFID(); //restart login process - RECURSION! - I have no idea what this
will do to the arduino's memory "\_(ツ)_/"
}

if (ID != 0){
  Serial.print('J');
  Serial2.print('C'); //Tell PC RFID read was successful (ID recognized)
}
else {
  Serial.println("Did not read the RFID code successfully");
}
}

```

```

// _____
//                               DRIVE: MAIN DRIVE / KINECT
// _____

void driveMode() {
  Serial.println("Entering Drive Mode");
  delay(1000);
  rightPan.write(90);
  leftPan.write(90);

  // BlinkM_stopScript(0x00);
  // delay(100);
  // BlinkM_playScript(0x00, defaultScript, 0, 0);
  // delay(100);
  BlinkM_setRGB(0x00,255,255,0);
  while(true) {
    //read from the kinect and update the eye servos as often as possible
    Serial2.flush();
    while(Serial2.available()<200) { //should have twice the number of bytes
in a message to ensure there is a complete message in the buffer
    }
    Serial.println("received 200 bytes");
    while(Serial2.available() and Serial2.read()!='W') { //one complete
message from C# is about 48 bytes long - this was without announcePan
    }
    if (Serial2.available()>75) { //a 'W' has just been read and there is a
complete message on Serial2
      receiveCSharp();
      readAccel();
      Serial.print("Accel Y-Axis: ");
      Serial.println(accelY);
      switch(modeSelect){
        case 0: //free driving - all systems enabled
          freeControl();
          break;

        case 1: //aggression
          freeControl();
          enableAggression();
          break;

        case 2: //submission
          freeControl();
          enableSubmission();
          break;

        case 3:
          freeControl();
          break;
      }
    }
    // Serial.print("DangerZone?: ");
    // Serial.println(dangerZone);
    sendRearPacket();
  }

  //only update sonar values/blinkMs and send to java every 0.3 seconds
  current_time = millis();
  if ((current_time - previous_time_sonars) > 300) {

    firePings();
    writeToHUD();
  }
}

```

```

    if(enableCuttle == 1){
        if(enableCuttle != lastCuttle){
            BlinkM_stopScript(0x00);
            delay(100);
            BlinkM_setFadeSpeed(0x00, 1); //changed from (0x00,20)
            delay(10);
        }
        Serial.println("Pings set blinks");
        pingSetBlinks();
    }else{
//        if(enableCuttle != lastCuttle){
//            BlinkM_setRGB(0x00,0,0,0);
//            delay(100);
//        }
//        BlinkM_playScript(0x00, defaultScript, 0, 0);
//        delay(100);
        BlinkM_setRGB(0x00,255,255,0);
    }
    lastCuttle = enableCuttle;

    previous_time_sonars = current_time; //keep track of when this method is
    called and how long it takes to complete

}
}

//
// _____
//          DRIVE: PACKAGE REAR DATA
// _____

void sendRearPacket(){
    //send information to back arduino
    Serial3.print("I");
    Serial3.print(fold);
    Serial3.print(",");
    Serial3.print(lb_angle);
    Serial3.print(",");
    Serial3.print(rb_angle);
    Serial3.print(",");
    Serial3.print(throttle);
    Serial3.print(",");
    Serial3.print(drive_mode);
    Serial3.print("E");
}
//
// _____
//          DRIVE: DRIVE/STEER FRONT
// _____

void driveSteer(){
    leftFrontSteer.write(lf_angle);
    rightFrontSteer.write(rf_angle);

    if (drive_mode==0) { //normal drive
        leftFrontThrottle.write(throttle);
        rightFrontThrottle.write(throttle);
    }
    else { //o-turn
        leftFrontThrottle.write(throttle);
        rightFrontThrottle.write(180-throttle);
    }
}
}

```

```

//
// _____
//                DRIVE: FREE DRIVE CONTROL
// _____

void freeControl(){
    enableEyes();
    enableAnnounce();
    driveSteer();
    pupilOpen();
    headlightControl();
}

//
// _____
//                PC: RECEIVE C# PACKET
// _____

void receiveCSharp() {
    Serial.println("Everything normal");
    char buffer[BUFSIZ]="";
    buffer[0] = Serial2.read();
    buffer[1] = Serial2.read();
    index = 2;
    while (buffer[index-2] != '\r' && buffer[index-1] != '\n' && index<BUFSIZ) {
//the data terminates with a windows newline, which is \r\n
        buffer[index] = Serial2.read();
        index++;
        //Serial.print("*");
    }
    if (index<BUFSIZ) {
        buffer[index-2]='\0'; //don't allow the \r or \n to be written into the
buffer
        p = buffer;
        for(int i=0; i<packageLengthKinect; i++) {
            parse = strtok(&p, ".");
            valStore = atof(parse);
            kinectInfo[i] = valStore;
        }
        //Serial.println("Parsed");
        tracking = kinectInfo[0];
        leftPan_value = kinectInfo[1];
        leftTilt_value = kinectInfo[2];
        rightPan_value = kinectInfo[3];
        rightTilt_value = kinectInfo[4];
        annVal = kinectInfo[5];
        newPerson = kinectInfo[6];
        fold = kinectInfo[7]; //0 for nomove, 1 for fold, 2 for unfold
        lf_angle = kinectInfo[8];
        rf_angle = kinectInfo[9];
        lb_angle = kinectInfo[10];
        rb_angle = kinectInfo[11];
        throttle = kinectInfo[12];
        drive_mode = kinectInfo[13];
        headlight = kinectInfo[14];
        enableCuttle = kinectInfo[15];
        enablePupil = kinectInfo[16];
        modeSelect = kinectInfo[17];
        //dangerZone = kinectInfo[18];

        //Serial.println("Array assigned");
        if (newPerson == 1) {
            Serial2.print('R'); //tell the PC that the newPerson notification was
received and pupilOpen will be called
        }
    }
}

```

```

    Serial.println("Kinect Cycle Complete");
  }else{
    Serial.println("Buffer Overflow");
  }
}

//
//----- PING: FIRE FRONT PINGS + READ REAR PINGS -----
//

void firePings(){
  //get all ping sensor values
  ping1.fire();
  delay(2);
  proxValues[0] = ping1.centimeters()*10;
  ping3.fire();
  delay(2);
  proxValues[2] = ping3.centimeters()*10;
  Serial.println("Front pings read");

  reading = 0;
  inDecimal = false;
  dplace = 0;

  while (Serial3.available()>0 and Serial3.read()!='A') {
  }
  if (Serial3.available()>0) {
    delay(2);
    while (Serial3.available()>0) {
      int x = int(Serial3.read()) - '0';
      if (x == 18) { //received the int value for 'B'
        reading2 = reading;
        reading = 0;
        inDecimal = false;
      }
      else if (x == 19) { //received the int value for 'C'
        reading4 = reading;
        Serial3.flush();
        break;
      }
      else if (x == -2) { //received the int value for a decimal point
        inDecimal = true;
        dplace = 1;
      }
      else if (!inDecimal) {
        reading = reading*10 + x;
      }
      else {
        reading = reading + x/pow(10,dplace);
        dplace++;
      }
      delay(1);
    }
    proxValues[1] = reading2;
    proxValues[3] = reading4;
  }
  Serial.println("Rear pings read");
}

```

```

// _____
// BEHAVIOR: AGGRESSION
// _____

void enableAggression(){
  if (fold == 1) { //person has been standing in danger zone for 2 seconds and we
are in aggression mode; car should haunch up
    if (accely >= 310) {
      fold = 1;
    }
    else {
      fold = 0;
    }
  }
  else { //person is not standing in danger; car should unfold (C# is sending
fold=2 in this case)
    if (accely <= 315) {
      fold = 2;
    }
    else {
      fold = 0;
    }
  }
}
// if(accely >= 310 && dangerZone == 1){
//   fold = 1;
// }else if(accely <= 300){
//   fold = 0;
// }
}

// _____
// BEHAVIOR: SUBMISSION
// _____

void enableSubmission(){
  if (fold == 1) { //this is for setup
    if (accely >= 310) {
      fold = 1;
    }
    else {
      fold = 0;
    }
  }
  else if (fold == 2) {
    if (accely <= 315) {
      fold = 2;
    }
    else {
      fold = 0;
    }
  }
}
// if (submissionPrep = 0){
//   if(accely >= 310){ //if flat or close to flat, start to fold
//     fold = 1;
//   }
//   else if(accely <= 300){ //if past hunching position, stop folding
//     fold = 0;
//     submissionPrep = 1;
//     delay(1000);
//   }
// }else{
//   if(accely <=300 && dangerZone == 1){ //if person in the way and car in
hunching position, fold down
//     fold = 2;

```



```

//    }
//    else if(accelY >= 310){ //if has reached flat position, stop unfolding
//        fold = 0;
//        //submissionPrep = 0;
//    }
// }
}

//
//-----
// BEHAVIOR: ANNOYANCE
//-----

void enableAnnoyance(){

}

//
//-----
// ANNOUNCE: MOVE ANNOUNCE SERVO
//-----

void enableAnnounce(){
    announceServo.writeMicroseconds(annVal);
}

//
//-----
// EYES: MOVE EYE SERVOS
//-----

void enableEyes(){
    leftPan.write(leftPan_value);
    leftTilt.write(leftTilt_value);
    rightPan.write(rightPan_value);
    rightTilt.write(rightTilt_value);
}

//
//-----
// PUPIL CONTROLLER: PUPIL STARTUP
//-----

void pupilStart(){
    I2c.write(42,'S'); //S
}

//
//-----
// PUPIL CONTROLLER: PUPIL IDLE
//-----

void pupilIdle(){
    I2c.write(42,'I'); //I
}

//
//-----
// PUPIL CONTROLLER: PUPIL OPEN
//-----

void pupilOpen(){
    if (enablePupil == 1){
        current_time = millis();
        if (newPerson == 1 && (current_time - previous_time_idle) > 700) {
//should be able to take out the time check with the notification of newPerson
receipt added in
            previous_time_idle = current_time;

```

```

        I2c.write(42,'0'); //0
    }
}

//
//-----
//                PUPIL CONTROLLER: HEADLIGHT CONTROL
//-----

void headlightControl(){
    if (previousHeadlight != headlight){
        if (headlight == 1) {
            I2c.write(42,'H'); //H
            Serial.println("Headights on");
        }
        else{
            I2c.write(42,'h'); //h
            Serial.println("Headights off");
        }
        previousHeadlight = headlight;
    }
}

//
//-----
//                BLINKMS: PING SET BLINKS
//-----

void pingSetBlinks(){
    //update the blinkMs
    for(int i=0; i < sonarCount; i++){

        if (proxValues[i] > 0) { //only update the blinkMs if the two ping
estimates were reasonable
            if (proxValues[i] > 510) {
                r = 0;
                g = 255;
            } else {
                r = min(510 - proxValues[i],255);
                g = min(proxValues[i],255);
            }
            i2cStat = BlinkM_setRGB(i+1,r,g,0); //blue should always be 0
        }
        //Serial.print("...Set...");
        Serial.print("Status: ");
        Serial.println(i2cStat);
    }
}

//
//-----
//                HUD: PACKAGE JAVA DATA
//-----

void writeToHUD(){
    //send the ping sensor values and whether or not a person is being tracked to
java
    //Serial.begin(19200);
    Serial.print('X');
    Serial.print(tracking);
    Serial.print('A');
    for (int i=0; i<4; i++)
    {
        Serial.print(proxValues[i],4);
        Serial.print('A');
    }
}

```

```
    Serial.print(throttle);
    Serial.print('A');
    Serial.print(drive_mode);
    Serial.print('A');
    Serial.println('Z');
}

//-----
//          ACCELEROMETER: READ ACCEL OUTPUT
//-----

void readAccel(){
    accelY = analogRead(0);
}
```

Primary Rear Micro

```
// Written by: Nicholas Pennycooke | Katharine Daly
// Arduino 1.0

#include "Servo.h"
#include <Ping.h>
#include <Wire.h>

Servo myServo;
int servoPin = 11;
int nomove = 1500;
int retract = 2000;
int extend = 1000;

unsigned long current_time = 0;
unsigned long previous_time_sonars = 0;

Ping ping2 = Ping(5);
Ping ping4 = Ping(6);
double back4;
double back2;
int fold;
int lb_angle = 90;
int rb_angle = 90;
int throttle = 90;
int drive_mode;

const int packageLength = 5;
int backInfo[packageLength];
int index;
char* parse;
char* p;
int valStore;

Servo leftBackSteer, leftBackThrottle, rightBackSteer, rightBackThrottle;

byte charge = 0;

void setup() {
  Wire.begin(2);
  Wire.onRequest(sendState);
  pinMode(A0, INPUT);
  pinMode(13, OUTPUT);

  myServo.attach(servoPin);
  Serial1.begin(38400); //this serial is for the xbee
  leftBackSteer.attach(7);
  leftBackThrottle.attach(10);
  rightBackSteer.attach(9);
  rightBackThrottle.attach(8);

  leftBackSteer.write(90);
  leftBackThrottle.write(90);
  rightBackSteer.write(90);
  rightBackThrottle.write(90);
}

void loop() {
  //send the back sonar information to the front arduino every 150 milliseconds
  current_time = millis();
  if ((current_time - previous_time_sonars) > 150) {
    ping4.fire();
  }
}
```

```

    delay(2);
    back4 = ping4.centimeters()*10;
    ping2.fire();
    delay(2);
    back2 = ping2.centimeters()*10;
    Serial1.print("A");
    Serial1.print(back2,4);
    Serial1.print("B");
    Serial1.print(back4,4);
    Serial1.print("C");
    previous_time_sonars = current_time;
}

//read the information sent via the front arduino from the kinect and act upon
it
if (Serial1.available()>0 and Serial1.read()=='I') {
    delay(10);
    char buffer[40] = "";
    buffer[0] = Serial1.read();
    index = 1;
    while (buffer[index-1] != 'E') {
        if (index<40) {
            buffer[index] = Serial1.read();
            index++;
        }
    }
    buffer[index-1]='\0';
    p = buffer;
    for (int i=0; i<packageLength; i++) {
        parse = strtok(p,",");
        valStore = atoi(parse);
        backInfo[i] = valStore;
    }
    fold = backInfo[0];
    lb_angle = backInfo[1];
    rb_angle = backInfo[2];
    throttle = backInfo[3];
    drive_mode = backInfo[4];

    if (fold == 0) {
        myServo.writeMicroseconds(nomove);
    }
    else if (fold == 1) {
        myServo.writeMicroseconds(retract);
    }
    else if (fold == 2) {
        myServo.writeMicroseconds(extend);
    }
    leftBackSteer.write(lb_angle);
    rightBackSteer.write(rb_angle);

    if (drive_mode==0) { //normal drive
        leftBackThrottle.write(throttle);
        rightBackThrottle.write(throttle);
    }
    else { //o-turn
        leftBackThrottle.write(throttle);
        rightBackThrottle.write(180-throttle);
    }
}

int c = analogRead(A0);
c = map(c, 0, 1023, 0, 100);
charge = c/20;

```

```
}  
void sendState(){  
  Wire.write(charge);  
  digitalWrite(13, HIGH); //set LED on  
  delay(10);  
  digitalWrite(13, LOW);  
}
```

Pupil Micro

```
// Written by: Nicholas Pennycooke  
// Arduino 1.0
```

```
#include <EL_Escudo.h>  
#include <Wire.h>  
  
int leftHeadlight = 10;  
int rightHeadlight = 11;  
int state = 0;  
int beamState = 0;  
  
void setup()  
{  
  Serial.begin(9600);  
  delay(1000);  
  EL.all_off();  
  Serial.println("EL wire turned off");  
  Wire.begin(42); // join i2c bus with address #5  
  Wire.onReceive(receiveEvent); // register event  
  headlights(0);  
}  
  
void loop()  
{  
  Serial.print("state is: ");  
  Serial.println(state);  
  if(state == 1){  
    startupPupils();  
    Serial.println("pupils should have entered startup");  
  }else if(state == 2){  
    idlePupils();  
  }else if(state == 3){  
    openPupils();  
  }  
  headlights(beamState);  
  delay(10);  
}  
  
// function that executes whenever data is received from master  
// this function is registered as an event, see setup()  
void receiveEvent(int howMany)  
{  
  char c = Wire.read(); // receive byte as a character  
  if(c == 'S'){ // on startup  
    state = 1;  
  }else if(c == 'I'){ // idle pupils  
    state = 2;  
  }else if(c == 'O'){ // open pupil  
    state = 3;  
  }else if(c == 'H'){  
    beamState = 1;  
    Serial.println("headlights should have turned on");  
  }else if(c == 'h'){  
    beamState = 0;  
    Serial.println("headlights should have turned off");  
  }  
}  
  
void startupPupils(){  
  // for(int i = 0; i<2;i++){
```

```

//      for(int i =248; i>1 ; i--){
//      analogWrite(leftHeadlight,i);
//      analogWrite(rightHeadlight,i);
//      delay(1);
//      }
//      for(int i = 2; i<248 ; i++){
//      analogWrite(leftHeadlight,i);
//      analogWrite(rightHeadlight,i);
//      delay(1);
//      }
//      }
for(int i = 0; i<4; i++){
  Serial.println("We're really in startuppupils");
  EL.on(A); EL.on(E);
  delay(100);
  EL.off(A); EL.off(E);

  EL.on(B); EL.Fon(); //custom library code, because arduino freaked out over
the letter F for some reason
  delay(100);
  EL.off(B); EL.Foff();

  EL.on(C); EL.on(G);
  delay(100);
  EL.off(C); EL.off(G);

  EL.on(D); EL.on(H);
  delay(100);
  EL.off(D); EL.off(H);
}
  state = 2;
}

void idlePupils(){
  //analogWrite(leftHeadlight,230);
  //analogWrite(rightHeadlight,230);

  //  EL.fade_in(A);EL.fade_in(E);
  //  delay(1000);
  //  EL.fade_out(A);EL.fade_out(sE);
  //  delay(1000);
  //  EL.all_off();
  EL.on(A);EL.on(E);
  delay(400);
  EL.off(A);EL.off(E);
  EL.on(B);EL.Fon();
  delay(400);
  EL.off(B);EL.Foff();
}

void openPupils(){
  //analogWrite(leftHeadlight,100);
  //analogWrite(rightHeadlight,100);
  for(int i = 0; i < 3; i++){
    EL.on(B); EL.Fon();
    delay(60);
    EL.off(B); EL.Foff();

    EL.on(C); EL.on(G);
    delay(60);
    EL.off(C); EL.off(G);

    EL.on(D); EL.on(H);
    delay(60);

```



```
        EL.off(D); EL.off(H);
    }
    state = 2;
}

void headlights(int s){
    if (s == 0){
        analogWrite(leftHeadlight,255);
        analogWrite(rightHeadlight,255);
    }else if (s ==1) {
        Serial.println("headlights should have turned on");
        analogWrite(leftHeadlight,230);
        analogWrite(rightHeadlight,230);
    }
}
```

[9-I] Custom/Modified Arduino Libraries

BLINKM_FUNCS_I2C.h

```
/*
 * BlinkM_funcs_I2C.h -- Arduino 'library' to control BlinkM
 *
 * 2012, Updated by Nicholas Pennycooke to work with the I2C.h library, a custom
library rewrite of the
 *         built-in Wire.h library
 *
 * -----
 *
 *
 * Note: original version of this file lives with the BlinkMTester sketch
 *
 * Note: all the functions are declared 'static' because
 *       it saves about 1.5 kbyte in code space in final compiled sketch.
 *       A C++ library of this costs a 1kB more.
 *
 * 2007-11, Tod E. Kurt, ThingM, http://thingm.com/
 *
 * version: 20111201
 *
 * history:
 * 20080101 - initial release
 * 20080203 - added setStartupParam(), bugfix receiveBytes() from Dan Julio
 * 20081101 - fixed to work with Arduino-0012, added MaxM commands,
 *           added test script read/write functions, cleaned up some functions
 * 20090121 - added I2C bus scan functions, has dependencies on private
 *           functions inside Wire library, so might break in the future
 * 20100420 - added BlinkM_startPower and _stopPower
 * 20111201 - updated to work with Arduino 1.0 (breaks compatibility with
Arduino <= 0023)
 *
 */

#include <Arduino.h>

#include "I2C.h" // from Wire library, so we can do bus scanning

// Call this first (when powering BlinkM from a power supply)
static void BlinkM_begin()
{
  I2c.begin(); // join i2c bus (address optional for master)
}

/*
 * actually can't do this either, because twi_init() has THREE callocs in it too
 *
static void BlinkM_reset()
{
  twi_init(); // can't just call Wire.begin() again because of calloc()s there
}
*/

//
// each call to twi_writeTo() should return 0 if device is there
```

```

// or other value (usually 2) if nothing is at that address
//

// FIXME: make this more Arduino-like
static void BlinkM_startPowerWithPins(byte pwrpin, byte gndpin)
{
    pinMode( pwrpin, OUTPUT);
    pinMode( gndpin, OUTPUT);

    digitalWrite( pwrpin, HIGH );
    digitalWrite( gndpin, LOW );
    /*
    DDRC |= _BV(pwrpin) | _BV(gndpin); // make outputs
    PORTC &=~ _BV(gndpin);
    PORTC |= _BV(pwrpin);
    */
}

// FIXME: make this more Arduino-like
static void BlinkM_stopPowerWithPins(byte pwrpin, byte gndpin)
{
    //DDRC &=~ (_BV(pwrpin) | _BV(gndpin));
    digitalWrite( pwrpin, LOW );
    digitalWrite( gndpin, LOW );
}

//
static void BlinkM_startPower()
{
    BlinkM_startPowerWithPins( A3, A2 );
}

//
static void BlinkM_stopPower()
{
    BlinkM_stopPowerWithPins( A3, A2 );
}

// General version of BlinkM_beginWithPower().
// Call this first when BlinkM is plugged directly into Arduino
static void BlinkM_beginWithPowerPins(byte pwrpin, byte gndpin)
{
    BlinkM_startPowerWithPins(pwrpin,gndpin);
    delay(100); // wait for things to stabilize
    I2c.begin();
}

// Call this first when BlinkM is plugged directly into Arduino
// FIXME: make this more Arduino-like
static void BlinkM_beginWithPower()
{
    BlinkM_beginWithPowerPins( A3, A2 );
}

// Sets the speed of fading between colors.
// Higher numbers means faster fading, 255 == instantaneous fading
static int BlinkM_setFadeSpeed(int addr, int fadespeed)
{
    I2c.write(addr,'f',fadespeed);
}

// Fades to an RGB color
static void BlinkM_fadeToRGB(byte addr, byte red, byte grn, byte blu)

```

```

{
  byte c[] = {red,grn,blu};
  I2c.write(addr,'c',c,3);
}

// Sets an RGB color immediately
static int BlinkM_setRGB(byte addr, byte red, byte grn, byte blu)
{
  byte n[] = {red,grn,blu};
  return I2c.write(addr,'n',n,3);
}

static void BlinkM_stopScript(int addr)
{
  I2c.write(addr,'o');
}

static void BlinkM_off(uint8_t addr)
{
  BlinkM_stopScript( addr );
  BlinkM_setFadeSpeed(addr,10);
  BlinkM_setRGB(addr, 0,0,0 );
}

```

EL_ESCUDO.h

```
/*
  EL_Escudo.h - EL Escudo library
  Written by Ryan Owens for SparkFun Electronics

  This library is released under the 'Beer Me' license, so use it however you
  wish. Just buy me a beer if we ever meet!

  Edits by Nicholas Pennycooke to fix a bug with the firing of the F wire
*/

#ifndef EL_Escudo_h
#define EL_Escudo_h

#include <inttypes.h>

#define A 2
#define B 3
#define C 4
#define D 5
#define E 6
#define F 7
#define G 8
#define H 9
#define STATUS 10
#define pulse_width 10

class EL_EscudoClass
{
public:
  void on(char);
  void off(char);
  void Fon();
  void Foff();
  void all_on(void);
  void all_off(void);
  void fade_in(char);
  void fade_out(char);
  void pulse(char);
};

extern EL_EscudoClass EL;

#endif
```

EL_ESCUDO.cpp

```
/*
  EL_Escudo.cpp - EL Escudo library
  Written by Ryan Owens for SparkFun Electronics

  This library is released under the 'Beer Me' license, so use it however you
  with. Just buy me a beer if we ever meet!

  Edits by Paul Krakow to make this library work with the Arduino 1.0 IDE
  And to get the "on" API call to work

  Edits by Nicholas Pennycooke to fix a bug with the firing of the F wire
*/

/*****
 * Includes
 *****/

#include "Arduino.h"
#include "EL_Escudo.h"

/*****
 * Definitions
 *****/

/*****
 * Constructors
 *****/

/*****
 * User API
 *****/

void EL_EscudoClass::on(char channel)
{
    pinMode(channel, OUTPUT);
    digitalWrite(channel, HIGH);
}

void EL_EscudoClass::off(char channel)
{
    pinMode(channel, OUTPUT);
    digitalWrite(channel, LOW);
}

void EL_EscudoClass::Fon(){
    pinMode(7, OUTPUT);
    digitalWrite(7, HIGH);
}

void EL_EscudoClass::Foff(){
    pinMode(7, OUTPUT);
    digitalWrite(7, LOW);
}

void EL_EscudoClass::all_on(void)
{
    for(int i=0; i<4; i++){
        EL.on(i*2+A);
    }
}
```

```

        EL.on(i*2+1+A);
        delayMicroseconds(20);
        EL.off(i*2+A);
        EL.off(i*2+1+A);
    }
}

void EL_EscudoClass::all_off(void)
{
    for(int i=A; i<10; i++)EL.off(i);
}

void EL_EscudoClass::fade_in(char channel)
{
    for(int brightness=0; brightness<=pulse_width; brightness++){
        for(int duration=0; duration<5; duration++){
            EL.on(channel);
            delay(brightness);
            EL.off(channel);
            delay(pulse_width-brightness);
        }
    }
    EL.on(channel);
}

void EL_EscudoClass::fade_out(char channel)
{
    for(int brightness=pulse_width; brightness>=0; brightness--){
        for(int duration=0; duration<5; duration++){
            EL.on(channel);
            delay(brightness);
            EL.off(channel);
            delay(pulse_width-brightness);
        }
    }
}

void EL_EscudoClass::pulse(char channel)
{
    EL.fade_in(channel);
    EL.fade_out(channel);
}

EL_EscudoClass EL;

```