

an uncertain range of more or less acceptable values. This work describes a software architecture which accepts both fuzzy linguistic and hard numeric constraints on trajectory performance and, using a trajectory generator provided by the user, automatically constructs trajectories to meet these specifications as closely as possible. The system respects hard constraints imposed by system dynamics or by the user, and will not let the user's preferences interfere with the system and user needs.

The architecture's evaluation agent translates these requirements into cost functional weights expected to produce the desired motion characteristics. The quality of the resulting full-state trajectory is then evaluated based on a set of computed trajectory features compared to the specified constraints. If constraints are not met, the cost functional weights are adjusted according to precomputed heuristic equations. Heuristics are not generated in an *ad hoc* fashion, but are instead the result of a systematic testing of the simulated system under a range of simple conditions.

The system is tested in a 2DOF linear and a 6DOF nonlinear domain with a variety of constraints and in the presence of obstacles. Results show that the system consistently meets all hard numeric constraints placed on the trajectory. Desired characteristics are often attainable or else, in those cases where they are discounted in favor of the hard constraints, failed by small margins. Results are discussed as a function of obstacles and of constraints.

AN ARCHITECTURE FOR THE AUTONOMOUS GENERATION OF
PREFERENCE-BASED TRAJECTORIES

by

Jamie Lennon

Dissertation submitted to the faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Advisory Committee:

Assistant Professor Ella Atkins, Chair
Associate Professor David Akin
Dr. C. Glen Henshaw
Professor Dana Nau
Associate Professor Robert Sanner

DEDICATION

This dissertation is dedicated to my mother, who worried that NASA wouldn't be hiring, and to my father, whose *Star Trek* books we can blame for all of this.

ACKNOWLEDGEMENTS

I first have to acknowledge the help and support of my advisor, Dr. Ella Atkins. Ella, you made me think deeply when I wanted to think broadly, and to define processes and architectures when I thought a big grab bag of stuff was enough.

I think this is the last time that I will have to acknowledge that this work was performed at the Naval Research Laboratory under funding from the Office of Naval Research under work order N0001404WX30001. I want to especially thank Alan Schultz at the Naval Research Lab for hiring me on and supporting my work, and also thank my co-workers there for their help and support.

I want to thank Dave Akin, for some words of encouragement that I very much needed to hear.

I owe my sister Taryn many thanks for her loving support and enthusiastic help planning mayhem when I got frustrated. All that and my maid of honor, too!

My parents I can never thank enough, for everything that they've done. Their love, support and confidence gave me a rock-solid foundation to build myself upon. It's made all the difference.

Finally, I thank my dear, sweet, *patient* husband Moe, who has done absolutely everything in his power to make these last six months of "dissertating" easier on me, often at cost to himself.

TABLE OF CONTENTS

List of Figures	viii
List of Tables	xi
List of Symbols	xii
1 Introduction	1
1.1 Contributions	6
1.2 Organization	6
2 Related Work	7
2.1 Natural Language	7
2.2 Path Planning and Traversal	9
2.3 Kinodynamic Planning	12
2.4 Multi-objective Optimization	12
2.5 Evolutionary Approaches	13
2.6 Isoperformance and Adaptive Weighted Sums	16
2.7 Mixed Integer Linear Programming	17
2.8 Optimal Control	18
2.9 Fuzzy Set Theory	26
3 Architecture	28
3.1 Initialization	33
3.2 Trajectory Generation	39
3.3 Feature Extraction	41
3.4 Weight Adjustment	42
3.5 Implementation	48

4 Two Degree of Freedom Point Rover.....	51
4.1 System Dynamics	51
4.2 Terms of the Cost Functional.....	51
4.2.1 Energy Use.....	52
4.2.2 Time	52
4.2.3 Clearance to Obstacles.....	52
4.3 Development of Weight Adjustment Heuristics and Fuzzy Rules	54
4.3.1 Weight Adjustment Heuristics.....	54
4.3.2 Heuristic Equation Verification.....	57
4.3.3 Fuzzy Rule Database Z	59
4.4 Results.....	60
4.4.1 Detailed Examples	61
4.4.2 Overall 2DOF Results.....	74
4.5 Conclusions.....	84
5 Six Degree of Freedom Deep Space Satellite.....	87
5.1 System Dynamics	88
5.2 Terms of the Cost Functional.....	89
5.2.1 Thruster Fuel.....	89
5.2.2 Electrical Energy.....	90
5.2.3 Time	90
5.2.4 Clearance to and Speed Near Obstacles.....	90
5.3 Development of Weight Adjustment Heuristics and Fuzzy Rules	91
5.3.1 Weight Adjustment Heuristics.....	91

5.3.2 Fuzzy Rules.....	95
5.4 Results.....	95
5.4.1 Example Cases.....	96
5.4.2 Overall 6DOF Results.....	108
5.4.3 How <i>WADJ</i> Works.....	117
5.4.4 Torque.....	119
5.5 Conclusions.....	122
6 Conclusions and Future Work.....	123
6.1 Future Work.....	127
6.1.1 Improving <i>INIT</i>	127
6.1.2 Improving <i>WADJ</i>	127
6.1.3 Improving <i>EVAL</i>	128
6.1.4 Developing Fuzzy Rule Database.....	129
6.1.5 Application to Other Adjustable Parameters.....	129
6.2 Final Summary.....	130
Appendix A: Features and Limits.....	131
Features.....	131
Limits.....	132
Appendix B: Test Cases and Margin Data.....	136
2DOF Cases.....	136
Start Point, Goal Point, and Obstacle Sets.....	136
Constraint Sets.....	138
Margins of Success and Failure.....	139

2DOF Trajectory Data by Constraint Set.....	141
6DOF Cases	169
Start Point, Goal Point, and Obstacle Sets	169
Constraint Sets	172
Margins of Success and Failure	173
6DOF Trajectory Data by Constraint Set.....	175
Appendix C: <i>WADJ</i> Heuristic Graphs and Fuzzy Rule Definitions	210
2DOF <i>WADJ</i> Heuristics.....	210
2DOF Fuzzy Rule Definitions	214
6DOF <i>WADJ</i> Heuristics.....	217
6DOF Fuzzy Rule Definitions	221
References.....	224

List of Figures

Figure 1: “Schlock Mercenary,” © 2000-2006, The Tayler Corporation, All Rights Reserved. Used with permission.....	1
Figure 2: A potential function.....	24
Figure 3: A fuzzy membership function.....	26
Figure 4: System architecture.....	29
Figure 5: Paths through the architecture.....	31
Figure 6: Initialization procedure.....	34
Figure 7: Calculating Ω^l and extended S^0	36
Figure 8: Developing <i>WADJ</i> rules.....	43
Figure 9: Effect of changing obstacle penalty weight (solid lines) and obstacle penalty function influence limit (dashed lines) on separation from obstacle.....	46
Figure 10: Average speed as a function of W_1/W_2 for zero, one, and three obstacles in a 10m x 10m field.....	55
Figure 11: Average speed as a function of W_1/W_2 for four different empty field sizes....	56
Figure 12: Heuristic equation less predictive as obstacles are added to the environment	57
Figure 13: 2DOF paths for CS 3, OS 1.....	64
Figure 14: 2DOF path trajectories for CS 3, OS 1.....	64
Figure 15: 2DOF path for $\Omega^l = [0.214, 1.0, 1.0, 1.0]$	65
Figure 16: 2DOF trajectory for $\Omega^l = [0.214, 1.0, 1.0, 1.0]$	65
Figure 17: 2DOF path for $\Omega^2 = [0.214, 1.0, 1.0, 1.0]$	67
Figure 18: 2DOF trajectory for $\Omega^2 = [0.214, 1.0, 1.0, 1.0]$	67
Figure 19: 2DOF path for $\Omega^3 = [0.015, 1.0, 1.0, 1.0]$	69

Figure 20: 2DOF trajectory for $\Omega^3 = [0.015, 1.0, 1.0, 1.0]$	69
Figure 21: 2DOF path for $\Omega^4 = [0.050, 1.0, 1.0, 1.0]$	71
Figure 22: 2DOF trajectory for $\Omega^4 = [0.050, 1.0, 1.0, 1.0]$	71
Figure 23: 2DOF path for $\Omega^5 = [0.002, 1.0, 1.0, 1.0]$	73
Figure 24: 2DOF trajectory for $\Omega^5 = [0.002, 1.0, 1.0, 1.0]$	73
Figure 25: Failures for each 2DOF solution case out of 28 H^0 and 72 S^0	74
Figure 26: 2DOF iterations through architecture.....	75
Figure 27: 2DOF S^0 failure rates by obstacle set.....	76
Figure 28: 2DOF S^0 failure rates by constraint set	77
Figure 29: Margin of success for hard limits in 2DOF case	79
Figure 30: Margin of success for soft limits in 2DOF case	81
Figure 31: Margins of failure on soft limits for 2DOF case	82
Figure 32: Margins of failure for soft limits after one iteration in 2DOF case.....	83
Figure 33: Margins of success for soft limits after one iteration in 2DOF case	84
Figure 34: Preliminary work in multi-vehicle formation management	86
Figure 35: WADJ curve for <i>avg-speed</i> in 6DOF domain.....	92
Figure 36: First stage of <i>WADJ</i> heuristic for determining torque in 6DOF domain.....	94
Figure 37: Second stage in torque heuristic in 6DOF.....	95
Figure 38: 6DOF a) path, b) rates and c) inputs for $\Omega^1 = [1.00, 1.00, 1.00, 1.00]$	99
Figure 39: 6DOF a) path, b) rates, and c) input for $\Omega^2 = [1.00, 1.00, 4.57, 1.00]$	101
Figure 40: 6DOF a) path, b) rates and c) input for $\Omega^4 = [1.00, 1.00, 4.86, 1.00]$	103
Figure 41: 6DOF a) path, b) rates, and c) inputs for $\Omega^1 = [1.00, 1.00, 4.00, 1.00]$	106
Figure 42: 6DOF a) path, b) rates, and c) inputs for $\Omega^2 = [1.00, 1.00, 4.79, 1.00]$	107

Figure 43: Failures for each solution case out of 20 H^0 and 60 S^0	108
Figure 44: Margins of failure for hard limits after first trajectory generation for 6DOF cases	109
Figure 45: Number of iterations through architecture	110
Figure 46: 6DOF S^0 failures by obstacle set.....	111
Figure 47: 6DOF S^0 failures by constraint set	112
Figure 48: Margins of success for hard limits for 6DOF cases	115
Figure 49: Margins of success for soft limits for 6DOF cases	116
Figure 50: Margins of failure for soft limits for 6DOF cases.....	117
Figure 51: Weight values evolving through Constraint Set 5, Obstacle Set 1 starting from $INIT^l$	118
Figure 52: 6DOF S^0 failures for Constraint Sets 6, 7 and 8.....	121
Figure 53: Number of iterations required for Constraint Sets 6, 7 and 8.....	122

List of Tables

Table 1: Verifying <i>WADJ</i> heuristics with analytical predictions.....	58
Table 2: One-Norm of Inertial and Body Forces	98

List of Symbols

$a(\mathbf{x}(t), \mathbf{u}(t), t)$	System dynamic equations
$avg\text{-}speed$	Average forward speed over trajectory
α	Generic exponent
bc	Set of boundary conditions
$best_trajectory$	Best trajectory found by solver so far
c_i	Generic constant coefficient
D	Set of domain-dependent parameters
δJ	Variation in J
F	Set of trajectory features
$f(\mathbf{x}), g(\mathbf{x}), h(\mathbf{x})$	Generic functions of $\mathbf{x}(t)$
H	Matrix of moments of inertia
$H(\mathbf{x}(t), \mathbf{u}(t), \lambda(t), t)$	Hamiltonian of the system
H⁰	Set of hard numeric limits on trajectory
$half\text{-}range$	The value of one-half the range of a fuzzy interval
i, j, k	Generic indices
J	Cost functional
K	Constant parameter in the obstacle penalty function
L⁰	Set of all limits on trajectory
LIM	Adjustable parameter in the obstacle penalty function
$\lambda(t)$	Lagrange multipliers
MAX	Constant parameter in the obstacle penalty function

m	Mass
\mathbf{margin}_{error}	Vector of errors between \mathbf{F} and \mathbf{L}^0
\mathbf{margin}_{fail}	Vector of errors between \mathbf{F} and \mathbf{S}^0
\mathbf{margin}_H	Vector of margins of success computed between \mathbf{F} and \mathbf{H}^0
\mathbf{margin}_S	Vector of margins of success computed between \mathbf{F} and \mathbf{S}^0
$max\text{-}acc$	Maximum acceleration over trajectory
$max\text{-}speed$	Maximum forward speed over trajectory
$midpoint$	Value of center of fuzzy range
$min\text{-}sep$	Minimum distance from path to any obstacle
$\{\mathbf{O}\}$	Set of obstacles
$o_i(r_i)$	Obstacle penalty function
\mathbf{P}_0	Planning trajectory problem
\mathbf{P}	Set of solution vectors generated by multiobjective optimization methods
\mathbf{p}	3x1 position vector
r_i	Distance from vehicle to center of i th obstacle in $\{\mathbf{O}\}$
\mathbf{S}	Matrix representation of cross product
\mathbf{S}^0	Set of soft constraints on trajectory
σ	3x1 orientation vector
t	Time
t_0	Initial time
t_f	Final time
$time_limit$	Limit in cycles or seconds on soft constraint satisfaction loop
$\boldsymbol{\tau}(t)$	Torque inputs

$u(t)$	Force or energy input
V	Fuzzy language database
v	3x1 velocity vector
W_i	i th weight of weight vector Ω
Ω^i	Weight vector for i th iteration through architecture
ω	3x1 rotational velocity vector
$x(t)$	State vector
x_0	Initial state vector
x_f	Final state vector
Y	Generic set
Y^i	i th element of generic set
Y^i	The vector which is the i th element of Y
Y_j^i	j th element of vector Y^i
y_j^i	Value of j th element of vector Y^i
Z	Fuzzy rules set

1 Introduction



Figure 1: “Schlock Mercenary,” © 2000-2006, The Tayler Corporation, All Rights Reserved. Used with permission.

As shown above in Figure 1, mercenary captain Kaff Tagon has a problem. He needs to select an officer to command his troops in his absence. He chooses his human commander, Kevyn, and leaves verbal orders: “Don’t let the troops get into any trouble you can’t get back out of with reasonable bribes or zero friendly casualties.” Kevyn, more an engineer than a commander, asks for more specifics – a “range of values for ‘reasonable’ and ‘friendly.’” Unsure of what judgments his captain would have him make, he asks Tagon to quantify his order. In the last panel, Tagon reveals that he is unwilling or unable to make his orders any more specific, and that if he could assign numbers to the judgment call, he would have left the ship’s artificial intelligence agent in charge.

This comic strip neatly summarizes a dichotomy that points to a real problem in automated systems research. Both potential command candidates – the human and the A.I. agent – would like the captain to express his orders in a crisper, more quantifiable fashion. This is not an unreasonable request for either to make, especially since the captain will be very displeased indeed if company money is wasted on “unreasonable”

bribes or “friendly” casualties are not kept to zero. But the captain cannot give numbers to either – and only trusts the human decision-maker to be able to properly function without them. The A.I. agent is assumed to lack the ability to translate this vague order into a policy for overseeing the mercenary troops in the captain’s absence. But we might like to have a system that can do exactly that. We would like for it to understand our priorities and expectations without having to specify them as numbers that may be counter-intuitive or even inaccurate.

We can, for example, easily identify “aggressive driving” when we see it on the roads. An aggressive driver’s behavior is marked by high traveling speeds, frequent lane changes, sudden accelerations, and the maintenance of the very barest of safety margins between other vehicles. What is a “high traveling speed?” Even once the context is fixed (e.g., interstate vs. in-town), the linguistic term has some fuzziness to it. Certainly, it implies a speed higher than the legal, posted speed limit. It probably means a speed higher than the average speed of the other drivers. But is someone driving 5 m.p.h. faster than road speed an aggressive driver? And on the other extreme, is there a speed so high that we can say that we have gone past “aggressive driving” and are into a region of “reckless driving?” When, exactly, is that line crossed? The answers to these questions are easy for humans to intuit, but difficult to formalize.

These concerns follow us into the realm of trajectory optimization. Robot trajectories do not have to be optimal. In some domains, we may accept satisficing trajectories that simply get the job done without capsizing the robot or running it into walls. But in the space domain in particular, we will always be concerned with conserving precious fuel and power. Even if we want an “aggressive, fast” trajectory, we

will still want it to be the most fuel efficient aggressive trajectory. We are concerned with fuel even if the result is not the fuel-optimal solution.

Trajectory optimization, at its most general, will have multiple objectives and constraints. Multiple objectives in particular give rise to multiple possible solutions. Consider a two objective case: we desire to save both time and fuel. These objectives compete with one another. The most fuel-efficient trajectory is rarely the most time-efficient trajectory, and visa versa. There might be several solutions that take the same minimum time, and we would be interested in the most fuel-efficient one. Or, we might examine all of the minimum fuel solutions and pick the fastest of those. Or, we might want some solution in between – one that is neither the fastest nor the most fuel-efficient, but balances the two objectives in some fashion. Starting out, we may have some idea of the kind of solution we want or of the relative importance of saving time or fuel. How can we communicate those preferences to the numeric solver that will compute the optimal trajectory, so that it will find the “right” optimal?

In addition to these fuzzy preference ideas, we may also have constraints to place on the trajectory. The generated trajectory must obey the system’s dynamics, and it must avoid collisions. Beyond that, we might further impose limits on either the control inputs (to reflect the realities of travel limits or thruster saturation) and on the state (as safety measures). Some of these limits could be “soft,” like a posted speed limit. It’s a good idea to obey the speed limit, but circumstances might require one to exceed it (e.g., to avoid an erratic driver). Other limits are “hard.” Astronauts will start blacking out if their re-entry capsule exceeds acceleration limits.

There are many approaches to solving the constrained multi-objective optimization problem [1], some of which – evolutionary algorithms, mixed integer linear programming, and optimal control theory – will be reviewed in Chapter 2. To a greater or lesser extent, they all can handle soft and hard constraints. All, however, include an iterative refinement loop to find those solutions that correspond to user preference: that is, to find the “optimal optimal” solution. And in all of the papers reviewed, it was tacitly understood that a human user would be interacting directly with the optimization algorithm in that loop, injecting preference information to focus the optimization on the areas of interest to the user.

This work seeks to take the human user out of that loop as much as possible. Before optimization ever begins, classes of motion are typified with linguistic expressions: aggressive, curious, careful. Fuzzy logic [2] is an appropriate tool for approaching the problem of translating natural language utterances into numeric terms [3]. The words are correlated to fuzzy state values that the system estimates that the user expects to see in the resulting trajectory: the “numbers for it” that we need to leave an A.I. in charge. Some iteration may be required here to ensure that the user’s expectation matches the fuzzy definition of the linguistic expressions. However, once that process is complete, the user can interact with the optimal trajectory generator in a much more hands-off fashion.

This work considers a planetary rover and an Earth-observing satellite as motivating examples. The planetary rover example is a very simplified case with two degrees of freedom and linear dynamics that provided initial insight into the trajectory generation and modification problem. The satellite case has more sophisticated and

realistic dynamics and all six degrees of freedom. The satellite's hypothetical job is to provide imaging to support ground-based decision making. It can execute fuel burns to change its orbit in response to user demands on the ground. These user demands may have varying levels of urgency. Some may be matters of some curiosity, but no urgency at all, and the satellite is free to execute maneuvers whenever it is most fuel-efficient to do so. It may also need to maneuver around other space objects (perhaps other satellites that require observation).

This work proposes an architecture that can intelligently meet these demands. A cognitively-inspired expert system moderates the trajectory generation and optimization process. At initialization, a solution technique is selected based on problem characteristics. If an initial trajectory estimate is required for the solution technique, one is generated, again with consideration for the problem characteristics. Finally, expressed user preferences are transformed via fuzzy methods into an initial set of weights or other parameters, and the selected solution technique is run.

The expert system also considers the results of the solution. Often in these problems, one or more user-defined constraints or preferences will not be met after the first iteration. Making changes to the weight vector or to other parameters may solve the problem; so may a different initial trajectory estimate or the use of a different solution technique (e.g., if the problem will not solve using the first technique). Given its knowledge base and the current history of repair attempts for this problem, the expert system continues to search for an appropriate trajectory.

1.1 Contributions

This dissertation describes an architecture for organizing trajectory generation and optimization tools into an overall unit that allows optimization with respect to poorly defined, fuzzy user preferences as well as respecting hard limits or boundaries placed on the results (whether these are a result of system dynamics or are also matters of user preference). This problem is challenging because it seeks to impose an automated, formalized framework around a process that is usually accomplished by many man-hours of trial and error. Suggesting such a framework and showing that it is useful in both linear and nonlinear dynamical systems is one contribution of this work. Finding the rules and techniques that would allow such automation, and showing their generality, is another contribution.

1.2 Organization

Chapter 2 introduces work related to this research. As the proposed architecture draws from a wide variety of sources, including traditional optimization methods, cognitive modeling, and fuzzy set theory, Chapter 2 is wide-ranging. Chapter 3 presents the architecture in more detail. Chapter 4 presents the first implementation of the architecture and the results of the two degree of freedom rover model. The extension to a 6DOF space satellite, and the changes this required, are given in Chapter 5. Chapter 6 concludes with a summary and review of the contributions of this research and explores future avenues of research.

2 Related Work

This work incorporates results from many fields. This chapter starts with a look at some of the natural language literature dealing with position and motion. Then, it will consider the traditional robotics approaches to motion planning and identify the primary shortcomings there. The different approaches to solving multi-objective optimization problems are next addressed, and their fitness for solving trajectory optimization problems in particular is examined. Finally, fuzzy set theory is introduced, and the literature linking it both to natural language and to optimization is explored.

2.1 Natural Language

Verbal or written instructions are one possible mode of interaction between a human and a robot (e.g., as in [4]). Decoding the meaning of these utterances is the provenance of natural language processing. With respect to motion words: “some languages like English regularly encode manner of motion in verbs, such as ‘swagger,’ ‘slink,’ ‘slide,’ and ‘sway’... Choice of verb is open to construal.” [5]. That is, the speaker has a variety of choices to describe how a route from a location A to B is traversed. What verbs the speaker selects will indicate to some extent the “manner of motion” the speaker requires of the robot.

To what extent? There is no appreciable literature dealing with the transformation of verbs to numbers. There is, however, a literature on assigning numeric values to spatial expressions such as “near” or “in front of” [6, 7, 8]. Researchers use, among other techniques, a potential field (first developed for robotic path planning [9]) as a membership function in the fuzzy set theory sense; indeed, fuzzy set theory terms like

“crisp” and “scruffy” appear frequently. Essentially, one point or line is selected (by the researchers) to represent the ideal of “near,” “along,” or “in front of” some object in the space. This becomes the minimum for the potential field, which can be visualized as something like a bowl. The bottom of the bowl (the minimum of the potential field) is located at the location of the ideal representation of “near,” “along,” etc. As one moves farther away from that point, line, or region, the bowl slopes up. The potential field returns higher values the farther away from the minimum one moves. In this way, the field generates a value for how much “nearness” or “in frontness” any coordinate in the space possesses. The smaller the generated value, the fairer it is to apply the spatial expression to it. At sufficiently large values, the spatial expression can be judged entirely false (e.g., something behind a desk is in no way in front of it).

This work extends this idea to motion words. First, we define a “state feature space” composed of state features such as average forward velocity and maximum acceleration. A collection of points in state feature space is taken (by the researchers) as the ideal representation of a verb or adverb/verb pair, like “jog” or “move stealthily.” Fuzzy membership functions are then defined around these areas, so that similar but not identical kinds of motion can still be included in these classifications. This gives some flexibility when trying to satisfy the multiple constraints and objectives such terms imply while maintaining the user’s preference for motion type.

Motion type is unavoidably somewhat domain-dependent when it comes to translating words to actual numeric values. We may define “quickly,” for instance, as having “high average speed” but exactly how fast that is will depend not only on the dynamic agent but also possibly on its environment. “High speed” for a human runner is

different from “high speed” for a car; similarly, a car going “too fast” through a residential area is in absolute terms of velocity probably traveling at a speed that would be “too slow” on the highway. Our definitional databases for each domain will store these domain-based differences.

2.2 Path Planning and Traversal

The task of moving a robot from a start location to a goal location has been much-studied. *Path planning* focuses on finding a path through free space for the robot to follow [10]. The robot’s dynamics are not generally considered at all for holonomic robots. For nonholonomic robots, dynamic constraints that directly affect path, such as a turning radius, may be used to reject some paths. When following the path, the robot is typically pre-programmed with a simple trapezoidal velocity profile. It accelerates at a pre-programmed rate to achieve its traveling speed, and then it maintains that speed until an obstacle or the goal requires it to smoothly decelerate. The rates of acceleration and the traveling speed are set well within the robot’s operating parameters, so no impossible demands will be made of the motors, and so that emergency stops will have a very minimal stopping distance. For slow, wheeled robots, especially those in a laboratory or office environment, this model usually suffices to move the robot around.

A step beyond this simple model lies *behavior-based* motion control [11]. Here, environmental cues trigger one of a suite of pre-programmed responses. The resulting actions can seem quite sophisticated or even emotional [12]. A robot that is “frightened” of red lights can be programmed to respond to that stimulus by moving away from red lights very quickly. The same robot might also be programmed to approach green lights very slowly, giving it an air of cautious interest. However, the behaviors “flee from red

light” and “investigate green light” and the parameters that define them are typically fixed, selected either by human researchers or machine learning techniques [13].

Although some research has been done into making the parameters that define these behaviors adaptive to environmental stimuli, they are still reactive in nature, making any sort of overall optimization impossible [14, 15].

Whereas most behavior-based robotics can be modeled with some kind of Markov decision process (MDP), another branch of research looks at using a hybrid dynamical system approach [16, 17]. In a hybrid dynamical system, discrete events trigger shifts between different continuous dynamics, e.g., reaching a fill line (a discrete event) causes a shift in water flow rate into a tank (continuous system dynamics). In the cited work, a simulated mouse agent adjusts its trajectory in response to the environment. As in this work, this is accomplished via changing weights. The weights, however, are weights on the repelling and attracting potential functions used for local navigation – there is no possibility of global optimization. Further, all the weight changes are based on a single “comfort” parameter and are linked directly to speed. The longer the mouse has gone without encountering a nearby obstacle, the more “comfortable” it becomes and the faster it moves. Its level of aggression (e.g., how closely it will approach obstacles) is also linked directly to this comfort parameter. While a hybrid dynamical systems approach provides an excellent way to realistically animate computer agents reacting to objects in their world (which is its goal), it does not yet offer the ability to meet constraints or optimize cost objectives; this is an open research area.

“Programming by reward” is a technique that elicits different dynamic behaviors from a system [18]. Like our research, it uses preference information to create these

differences. Unlike our research, it injects the preference information into machine learning algorithms for the development of motion behaviors. These behaviors can even form an optimal policy, given preferences. However, the “interiors” of the behaviors are still black boxes. The number of lane changes in the authors’ driving example can be optimized for a safe driver and for a reckless driver, but the dynamics of that lane change are unexamined. Our research is especially interested in controlling the low-level inputs that result in the desired behaviors, rather than assembling pre-typed behaviors into a policy.

It is no coincidence that the fastest real-time motion planners – artificial potential function guidance – are the ones with the least capability for optimization [9], and that the most flexible optimization routines are the most computationally intensive. The high dimensionality, nonlinearity, constraints and multiple objectives all combine to make trajectory optimization a very difficult problem. In environments that are highly “dynamic” in the AI community’s sense of the word (that is, rapidly and unpredictably changing), optimization will rarely be feasible. Fast, reactive motion planning will undoubtedly continue to play a role in robot control. But in the space domain, optimization is a key tool in developing fuel- and time-efficient trajectories while ensuring safety. The environment is in most cases very well-known, and the current and future locations of obstacles can be predicted with a high degree of accuracy and certainty. This is very unlike the rapid and unpredictable changes encountered by robots operating in home, office, or street environments on Earth.

2.3 Kinodynamic Planning

Trajectory, or kinodynamic, planning is a newer field. Methods based on the idea of velocity obstacles [19, 20] are conceptually similar to path planning “roadmap” methods, with forbidden velocities (e.g., those that would cause collisions or exceed system capabilities) modeled as obstacles in a velocity space. Spline methods [21, 22] decouple the path and trajectory planning process; once a clear path through space is found, interpolating splines are used to find smooth trajectories along them. Randomized kinodynamic planning [23] explores the state space in a random fashion, working forward from the start state and backward from the goal state until the search trees meet. All of these methods search for dynamically feasible trajectories; none of them by themselves have any notion of optimality. They can be used as input to optimization routines, however. Velocity obstacles and spline methods have both been used to generate time optimal trajectories [24, 25] and randomized methods are often used as starting points for linear programming methods to generate fuel optimal trajectories [26].

2.4 Multi-objective Optimization

At its most general, trajectory optimization is a multi-objective problem with constraints. Both the multiple objectives and the constraints complicate the solution of the problem, which is already, except in certain very specialized cases like station-keeping, nontrivial. A variety of approaches have been developed to solve multi-objective optimization problems, and they will be surveyed below. While the different approaches have different strengths and weaknesses, they all have one thing in common: a parameter or set of parameters that can be adjusted to reflect the user’s priorities concerning objectives. This common problem motivates the work of this dissertation.

The general form of the constrained multi-objective optimization problem is:

$$\text{minimize } (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_k(\mathbf{x})) \quad (2.1)$$

subject to the m inequality constraints

$$g_i(\mathbf{x}) \leq 0 \quad i = 1, 2, \dots, m \quad (2.2)$$

and the p equality constraints

$$h_j(\mathbf{x}) = 0 \quad j = 1, 2, \dots, p \quad (2.3)$$

where f are the k objective functions $f_i : \mathfrak{R}^n \rightarrow \mathfrak{R}$ and $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$ is the vector of decision variables. \mathbf{P} is the set of vectors that satisfy Equations 2.2 and 2.3; the set of optimal vectors \mathbf{x}^* will be found in \mathbf{P} .

2.5 Evolutionary Approaches

Genetic and evolutionary algorithms (GAs and EAs) have become popular search and optimization tools since their introduction in the 1970s [27]. A population of potential problem solutions is generated and encoded (in binary form for GAs, and in other representations for EAs), then tested against some fitness function. A new population is then formed from the first by selecting the “fittest” individuals for survival and “breeding” them by combining features of their solutions into new solutions. The fittest individuals themselves may also be included in the new population. Mutations – small changes made at random to the solution – also increase novelty in the new population. There are variations on this pattern of EA iteration, but this is the basic technique.

Since there is rarely a single point where all of the multiple objectives are simultaneously maximized or minimized, evolutionary multi-objective optimization (EMO) frequently makes use of *Pareto optimality* [28, 29]. Given a set of k objective functions f , a vector of decision variables $\mathbf{x}^* \in \mathbf{P}$ is Pareto optimal if there does not exist another $\mathbf{x} \in \mathbf{P}$ such that $f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*)$ for all $i = 1, \dots, k$ and $f_j(\mathbf{x}) < f_j(\mathbf{x}^*)$ for at least one j . In other words, there is no feasible vector \mathbf{x} which would decrease some criterion without also increasing some other criterion. The set of vectors \mathbf{x}^* that are Pareto optimal are called *nondominated*. The corresponding set of Pareto optimal solutions to the objective functions f is called the *Pareto front*.

A variety of survey papers review the state-of-the-art in EMO research from the mid-1990s to the present [30, 31, 32], and a comprehensive website with over 1000 EMO references provides access to a wealth of literature [33]. Early attempts included the Vector Evaluated Genetic Algorithm (VEGA) [34], which has a tendency to find solutions that do very well in one dimension, but not “middling” or compromise candidates, and the application of EA heuristics to traditional approaches like aggregation (combining multiple objectives into a single objective) [35] and lexicographic ordering (optimizing the objectives in turn, beginning with the most important) [36]. The ideas of Pareto dominance were not fully utilized at this time.

The primary drawback of the aggregation approach, so common in traditional engineering applications, is that it can miss areas of the Pareto optimal front if the front is non-convex. A region is said to be convex if it always contains the line segment connecting two points when it contains the two points themselves. An EA that scores members of a population based on their values for each objective f individually, rather

than the aggregate of all of them, does not need to assume a convex Pareto front to explore all of it. More recent research has concentrated on ways to encourage population diversity, so that the EA will explore the entire Pareto front, and the addition of elitism, which usually uses an external file to store nondominated individuals so that they will not be lost. (The ways in which the members of the external population interact with the rest of the population varies from algorithm to algorithm.) The algorithms have been determined sufficiently robust to be applied to engineering, industrial and scientific domains [30]. The bulk of this work has been done for problems with only two objective functions; however, some research has been done with problems involving three. Beyond that, the field re-names the problem “many-objective optimization,” and there are many open questions there. In particular, it has been shown that, as the number of objectives increases, Pareto dominance becomes nearly useless in ranking individuals [37]. Furthermore, it is recognized that often the user does not really want to have to evaluate all members of the Pareto optimal front, and that incorporating user preference into the EAs to reduce the returned solution set is a high research priority [30].

There are two additional considerations to note. The first is that EAs do not explicitly calculate gradients along the solution set. A good fitness function and the judicious use of crossover and mutation ensures that the solutions will tend to follow the gradient down to the minimum, but this is accomplished by selecting ever more-fit individuals, not by taking advantage of trends. In some cases, this is a strength – EAs are robust to discontinuities in the solution space, including discontinuities occurring at constraint boundaries. Since they do not calculate a gradient, they are unaffected if it

should disappear or go to infinity. But when gradients are available, their use would greatly speed convergence.

Second, EAs are an unconstrained optimization technique. There is no explicit mechanism for enforcing constraints such as Equations 2.2 and 2.3. Researchers treat the problem in different ways. Sometimes, constraints can be recast as objectives. An assortment of penalty functions can be invoked, penalizing the fitness of solutions that do not meet the constraints – although this can run the risk of degenerating into a random walk if there are no good solutions at all in the initial population [32]. Another recent approach iteratively shrinks the search space to focus on zones where the constraints are met [38]. These approaches work, some faster than others, and most have a set of parameters (such as the “rate of shrink” in [38]) that must be set and then tuned by the user. If the parameters are poorly set, the methods do not work well at all.

2.6 Isoperformance and Adaptive Weighted Sums

de Weck et al. have investigated other, more deterministic methods of developing a Pareto front. Isoperformance [39, 40] sets a required performance level, indicated by a fixed value for a cost function. This can be the single output of a complex system, such as the displacement of space telescope subjected to disturbance forces. Through one of several algorithms, the design variables of the system (which for the hypothetical telescope could include parameters as different as its mass, the stiffness of its bending modes, the bandwidth of its attitude control system, and even the magnitude of the star being used by the guidance system to orient the system) are varied, and those combinations which give the desired performance level are recorded. From that set, a

nondominated front is further selected. A user would then select a single solution from among the nondominated solutions.

Adaptive weighted sum methods [41] can be used when the cost function is of the form of a weighted sum of terms. Traditional techniques for exploring the Pareto front of such a system would sample weights at constant fixed intervals, and so might miss many important features of the front. The adaptive weighted sum approach begins with a constant interval mesh of weights, then refines the mesh in areas where there are large gaps between returned cost. Additional inequality constraints are also added to restrict the calculations to areas where the Pareto front is thought to lie.

These techniques seem very promising for systems design, or in any application where a user would want to obtain an entire nondominated set of solutions for consideration. If we were interested in only planning a route between two fixed points, the time required to form the Pareto front of trajectories using one of these techniques might be worthwhile. However, our interest is in calculating many trajectories in similar but not identical environments. Since the trajectory generation process is itself computationally intensive, we would like to avoid computing a Pareto front of solutions for any one set of boundary conditions.

2.7 Mixed Integer Linear Programming

Mixed Integer Linear Programming (MILP) has become increasingly popular as a relatively fast way to generate and optimize trajectories [42]. The technique has been well-known in the operations research field for many years, but recent increases in computing speed have allowed the technique to be considered for real-time planning applications. Equality and inequality constraints can all be handled robustly. It can

approximate non-convex constraints and can handle logical constraints as well. Obstacle avoidance is done by placing inequality constraints directly on the path space, forcing the trajectory outside the region of the obstacles; penalty functions are not used. The cost functional could in principle include many terms, although applications typically focus on either fuel or time optimal trajectories. MILP has been applied to a spacecraft rendezvous problem [43] and to multi-satellite reconfiguration problems [44].

However, as its name suggests, it is suitable only for problems with linear constraints, including dynamic constraints. [43] uses the linear Hill's/Clohessy-Wiltshire equations (although this is not inappropriate for a docking maneuver) and [44] linearizes gravity-free dynamics. MILP does not handle nonlinear constraints at all, except by linearizing them. This is appropriate for some domains, but not for all. Control of a satellite formation over a highly-elliptical orbit, for example, is not amenable to linearization.

2.8 Optimal Control

Optimal control methods [45, 46] have been used to solve trajectory planning problems for many years. The calculus of variations is used to frame the problem as a system of differential equations subject to conditions imposed at the initial and final time. The equations can be – and often are – nonlinear. System dynamics as well as other constraints can be included. Generally, a cost functional is of the form:

$$J = \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{x}'(t), t) dt \quad (2.4)$$

where $\mathbf{x}(t)$ is the state vector and $\mathbf{x}'(t)$ is its derivative. A variation in the functional, δJ , can be defined for small changes of $g(\mathbf{x}(t), \mathbf{x}'(t), t)$. If a relative minimum for J exists, it is necessary that δJ be zero at that point. Applying the definition of δJ to Equation 2.4 yields the Euler Equation:

$$\frac{\partial g}{\partial \mathbf{x}}(\mathbf{x}^*(t), \mathbf{x}'^*(t), t) - \frac{d}{dt} \left[\frac{\partial g}{\partial \mathbf{x}'}(\mathbf{x}^*(t), \mathbf{x}'^*(t), t) \right] = 0 \quad (2.5)$$

where $\mathbf{x}^*(t)$ is an extremal state vector and $\mathbf{x}'^*(t)$ its derivative.

The problem is to find an admissible input (or control) vector $\mathbf{u}^*(t)$ that causes a system described by the differential equations in Equation 2.6 to follow an admissible trajectory $\mathbf{x}^*(t)$ that minimizes the cost functional Equation 2.7.

$$\mathbf{x}'(t) = a(\mathbf{x}(t), \mathbf{u}(t), t) \quad (2.6)$$

$$J(\mathbf{u}) = \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{u}(t), t) dt \quad (2.7)$$

At all points along an admissible trajectory, Equation 2.6 holds and can be rewritten:

$$a(\mathbf{x}(t), \mathbf{u}(t), t) - \mathbf{x}'(t) = 0 \quad (2.8)$$

and added to $g(\mathbf{x}(t), \mathbf{u}(t), t)$ with Lagrange multipliers λ to form an augmented cost functional:

$$J_a(\mathbf{u}) = \int_{t_0}^{t_f} g_a(\mathbf{x}(t), \mathbf{x}'(t), \mathbf{u}(t), \lambda(t), t) dt$$

or, rewriting,

$$J_a(\mathbf{u}) = \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{u}(t), t) + \boldsymbol{\lambda}^T [a(\mathbf{x}(t), \mathbf{u}(t), t) - \dot{\mathbf{x}}(t)] dt \quad (2.9)$$

The extremals of the functional are where δJ_a is zero. Finding δJ_a and setting it to zero results in three necessary equations. They are most commonly expressed in terms of the Hamiltonian, which is defined as:

$$H(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t), t) = g(\mathbf{x}(t), \mathbf{u}(t), t) + \boldsymbol{\lambda}^T [a(\mathbf{x}(t), \mathbf{u}(t), t)] \quad (2.10)$$

The necessary conditions are then:

$$\dot{\mathbf{x}}^*(t) = \frac{\partial H}{\partial \boldsymbol{\lambda}}(\mathbf{x}^*(t), \mathbf{u}^*(t), \boldsymbol{\lambda}^*(t), t) \quad (2.11a)$$

$$\dot{\boldsymbol{\lambda}}^*(t) = -\frac{\partial H}{\partial \mathbf{x}}(\mathbf{x}^*(t), \mathbf{u}^*(t), \boldsymbol{\lambda}^*(t), t) \quad (2.11b)$$

$$0 = \frac{\partial H}{\partial \mathbf{u}}(\mathbf{x}^*(t), \mathbf{u}^*(t), \boldsymbol{\lambda}^*(t), t) \quad (2.11c)$$

for all $t \in [t_0, t_f]$. For a fixed final time and a fixed final state, we have boundary conditions

$$\mathbf{x}(t_0) = \mathbf{x}_0 \quad (2.12a)$$

$$\mathbf{x}(t_f) = \mathbf{x}_f \quad (2.12b)$$

which gives the equations needed to determine the constants of integration. Solving the system of equations returns the function (trajectory) that will minimize the cost functional.

If the final time and final state are not fixed but are free, a new boundary condition called the transversality condition is produced:

$$g_a(\mathbf{x}(t_f), \mathbf{u}^*(t_f), \boldsymbol{\lambda}(t_f), t_f) \delta t_f - \boldsymbol{\lambda}^T(t_f) \delta \mathbf{x}_f = 0 \quad (2.13)$$

When \mathbf{x}_f is fixed, as in this work, $\delta \mathbf{x}_f = 0$, so it must be that

$$g_a(\mathbf{x}(t_f), \mathbf{u}^*(t_f), \boldsymbol{\lambda}(t_f), t_f) = 0.$$

Except for certain special cases, there is no way to analytically solve the optimal control problem. A variety of numeric methods have been employed, including the shooting method [47] and collocation. The shooting method is so-called because it uses initial value problems (IVPs) as a starting point to “shoot” towards the solution of the optimal controls boundary value problem (BVP). Unfortunately, a stable BVP (one insensitive to changes in boundary values) may require the integration of unstable IVPs (ones highly sensitive to changes in boundary values). This drawback led to the development of the collocation approach.

In collocation, the actual solution to differential equations 2.11 is approximated over a mesh, defined by “knot points.” The approximation is made to satisfy the boundary constraints at t_0 and t_f , and further to satisfy Equations 2.11 a-c at each knot point and at the midpoint of each interval between them. An initial guess for the solution must be provided; the solution technique will alter the current solution estimate to bring its residual (a measure of error) to within acceptable bounds.

There are many ways to solve a collocation problem. Solution methods fall in general into two classes: direct and indirect. Direct methods [48] model the approximate solution as composed of basis functions; the solution is improved by altering a vector containing the coefficients for the basis functions. This allows vector optimization algorithms such as sequential quadratic programming, Newton-Gauss, or Levenberg-Marquardt to be applied [49]. Direct methods are considered faster and more robust than the indirect methods. Indirect methods (as in, e.g., [50]) link the knot points with continuous approximating functions (e.g., cubic splines) over each subinterval. The coefficients of each of these functions must then be solved. This makes indirect methods generally more computationally intensive than direct methods. Their advantage is additional flexibility; the basis functions in the direct methods must be chosen such that every function could be a feasible trajectory. The indirect method has no such constraint.

The optimal control problem's solution is governed by a single cost functional. Multiple objectives can only be optimized via an aggregation method. Since some constraints are likely to be non-convex, this means that certain solutions along the Pareto optimal front may be missed. Typically, a sufficient number of other solutions that also satisfy the user's preferences also exist where they can be detected.

Optimal controls problems can incorporate constraints and discontinuities. Equality constraints on the state (such as satisfying the system dynamics) are adjoined to the cost functional via Lagrangian multipliers, as discussed above. Constraints on the control inputs can be handled via Pontryagin's Minimum Principle and the resulting switching curves. Inequality constraints can be handled by the introduction of a function of a dummy variable, x_{n+1} , whose derivative is defined as:

$$x'_{n+1}(t) \equiv [f_1(\mathbf{x}(t), t)]^2 \hat{\uparrow}(-f_1) + [f_2(\mathbf{x}(t), t)]^2 \hat{\uparrow}(-f_2) + \dots + [f_l(\mathbf{x}(t), t)]^2 \hat{\uparrow}(-f_l) \quad (2.14)$$

where $\hat{\uparrow}(-f_i)$ is a unit Heavyside function defined by:

$$\hat{\uparrow}(-f_i) = \begin{cases} 0, & f_i(\mathbf{x}(t), t) \geq 0 \\ 1, & f_i(\mathbf{x}(t), t) < 0 \end{cases} \quad (2.15)$$

for $i = 1, 2, \dots, l$ (where $l \leq m$, the size of the control vector). x_{n+l} can then be defined as:

$$x_{n+l}(t) = \int_{t_0}^{t_f} \dot{x}_{n+l}(t) dt + x_{n+l}(t_0) \quad (2.16)$$

We then require boundary conditions $x_{n+l}(t_0) = 0$ and $x_{n+l}(t_f) = 0$. Since the derivative is never less than zero, $x_{n+l}(t)$ must be zero for all t . This is a constraint of the form $f(\mathbf{x}(t), t) = 0$, and it can be treated by the method of Lagrange multipliers.

However, the switching curves and Heavyside function are clearly discontinuous, making them problematic for many numeric solvers. They can be approximated by a series of increasingly steep polynomials to avoid this. However, the unchanging nature of x_{n+l} presents a further problem. Collocation solvers require gradient information to reduce the error between the current approximate solution and the true solution. Since x_{n+l} is identically zero for the entire trajectory, it provides no gradient information at all. In our particular case, the collocation solver required a Jacobian matrix which, when the Heavyside approximation was added, contained a full row of zeroes and so would not solve. Adjusting the Heavyside approximation further to provide some kind of gradient information in the allowable solution region amounted to instituting a penalty function, which is another way to treat state inequality constraints.

Penalty functions are often used in path and trajectory planning for obstacle avoidance. Often cubic in form, these penalty functions are centered over an obstacle and monotonically decrease as they move away from its center. Typically, they go to zero at some influence limit away from the obstacle, but this is not required. Figure 2 shows the piecewise continuous penalty function used for obstacle avoidance in the 2DOF work in Chapter 4. It has a fixed value at the object's center, at the edge of the object, and at a fixed distance from the edge of the object. The coefficients of the cubic functions used can be varied to achieve these conditions for obstacles of different sizes.

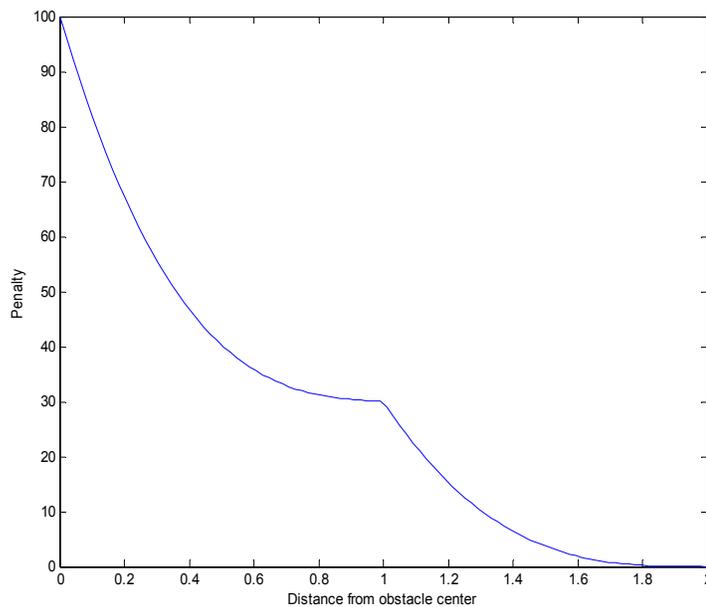


Figure 2: A potential function

The segment of the function that extends from the edge of the obstacle to the end of the “influence limit” some distance away provides uniform penalties for approaching obstacles of any size. No matter how large or small the obstacle, this segment of the function persists from its outer edge (where contact would occur) to the influence limit. The segment which lies over the obstacle itself aids in computation. If the penalty over the obstacle was flat, and a possible path was plotted through the obstacle, there would be

no information available to the solver to make a decision about moving the path. It would be clear that this was an expensive path, but perturbations around it would cost effectively the same. With this second segment, the solver can quickly determine a gradient that will “roll” it off of the obstacle.

The value of the penalty function is added to the cost functional. As cost is minimized, the trajectory will tend to be moved away from the obstacles. However, if other costs are sufficiently great, it may be numerically less expensive to accept the penalty – which means planning a path through the obstacle. Penalty functions do not offer guarantees on constraint satisfaction, which means that solutions generated via optimal control methods must be checked in a post-processing step.

Using penalty functions to disallow regions of the velocity space was attempted as part of this research. However, doing so caused an unacceptable slow-down of trajectory planner performance. The planners used in general have a more difficult time, computationally, when there are many obstacles near the generated trajectory, and the obstacles in velocity-space were apparently too near the solution velocity. This may not be a problem with another trajectory planner, and would be a valid technique for enforcing these limits. However, as with penalty functions used for physical obstacles, there is always a chance that there will be a trajectory whose minimum cost lies within the obstacle, if the penalty function is not sufficiently large. Trajectories would still have to be checked for these failures, and some parameters of the penalty function adjusted to shift the trajectory to the outside of the obstacle.

2.9 Fuzzy Set Theory

The main idea behind fuzzy set theory is that a member of a set may belong only partly to that set [2]. Classically, individuals either are or are not contained in a set; they are either hot or not hot, for example. In fuzzy set theory, an individual may be 50% hot and 50% not hot, or 30% hot and 50% warm. Complements, like “hot” and “not hot” must sum to 100% but non-complementary attributes may not. For example, the vertical line in Figure 3 indicates that generic feature value F is about 45% “low,” about 60% “medium,” and 0% “high.”

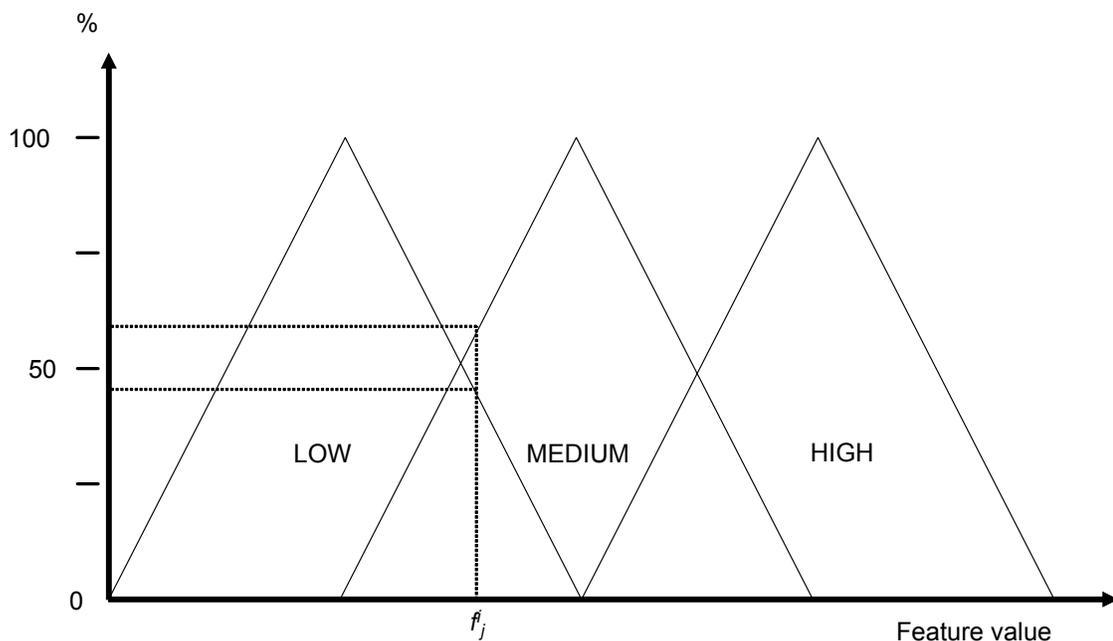


Figure 3: A fuzzy membership function

The triangles in Figure 3 are membership functions. They correlate “crisp” numeric values, as measured in the real world, to these fuzzy levels. A fuzzy rules set

then acts on these “fuzzified” inputs. For example, “If air temperature is LOW, turn heater fan to “HIGH” and “If air temperature is MEDIUM, turn heater fan to LOW.” The fans speeds will have similar fuzzy membership function correlating speeds like “HIGH” and “LOW” to revolutions per minute. These outputs are scaled by the membership function of the inputs.

Natural language, while a desirable input modality, is inherently ambiguous. From interpreting sounds into words to parsing the words into sentences to interpreting the possible shades of meaning of a sentence, there are ambiguities that must be dealt with. Classical mathematics does not manage ambiguities well. Fuzzy techniques, on the other hand, deal with them substantially better [3]. Ambiguities lend themselves to problems involving shades of meaning or even slight differences in pronunciation.

Fuzzy optimization, a fairly new field, applies fuzzy set theory to optimization problems. Fuzzy techniques are not themselves used to solve the problem, but are rather applied to candidate solutions to rank them. They are often used in conjunction with EAs, where the evolutionary algorithms generate the candidate solutions and the fuzzy methods are used to rank them before selection and breeding occurs. Recent work [6] has shown that an expanded and fuzzified notion of Pareto dominance seems to perform more in accord with common sense than strict Pareto dominance, and should not have the same problem as Pareto dominance (e.g., that all solutions become equally good) as the number of objective functions increases to infinity.

3 Architecture

The problem with which we are faced is this: to generate a dynamically feasible trajectory for some autonomous vehicle which also satisfies user-imposed constraints. Some of these constraints are “soft,” and indicate a user’s preference for a solution in some region of the state space. Other constraints are “hard” and indicate regions of the state space from which the vehicle is forbidden. Under these constraints, we still wish to optimize the trajectory with respect to fuel and/or time. We will use a vector of weights to direct the solution to the preferred regions of state space, automatically adjusting the vector if an acceptable solution is not found.

Figure 4 shows an outline of the agent’s processes. At the center sits the evaluation model, overseeing all activities. The human user interacts with this module, monitoring events rather than directly participating in trajectory generation processes. The evaluation module, *EVAL*, accepts a planning problem, \mathbf{P}_0 , which can be posed by the user or by any suitable high-level planner that builds task-level actions to achieve its goals, some of which may require vehicle motions.

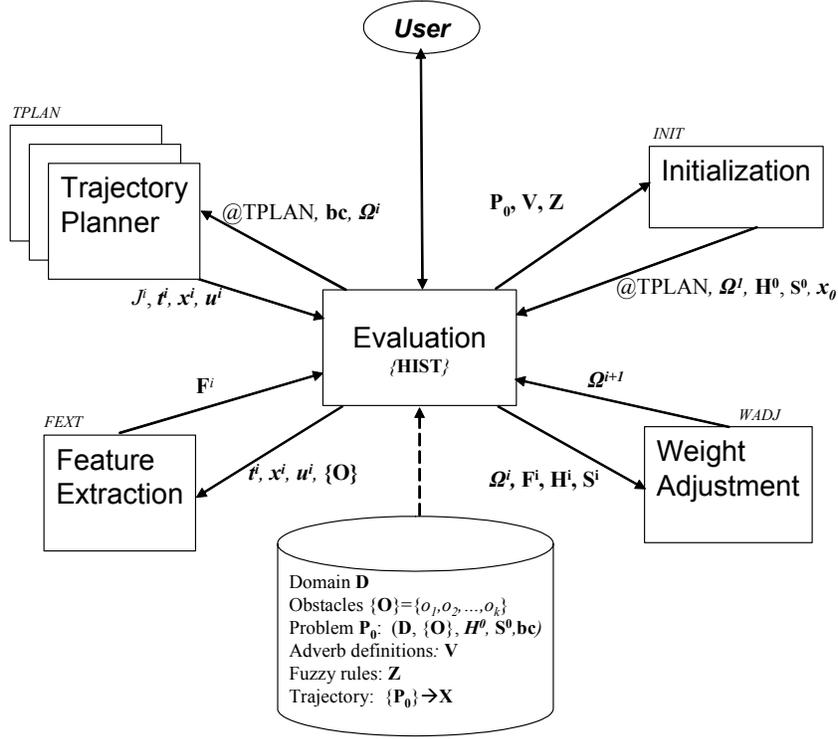


Figure 4: System architecture

A trajectory planning problem P_0 is defined as $\{D, \{O\}, H_0, S_0, bc\}$. Domain D describes system dynamics and the parameterized cost functional J to be minimized. $\{O\}$ represents the set of obstacles in the environment. H^0 describes the hard constraints (limits on state space values) to be met, whereas S_0 is a set of soft constraints that indicate user preference, but are ultimately flexible. H^0 are numeric; S^0 may be numeric or fuzzy linguistic terms. Fuzzy terms must eventually be converted into soft numeric limits; L , the set of all limits, includes H^0 and the extended S^0 . Members of L may be upper limits, lower limits, or range limits (when we want the state feature to be within an upper and a lower limit). The boundary conditions $bc = \{t_0, x_0, x_f\}$ are split and can include all of the usual optimal controls cases (e.g., fixed or free final time or state, final state constrained

to a fixed or moving surface). The goal is to return feasible and optimal solution $\mathbf{X} = \{J^n, L^n, t^n, x^n, u^n\}$, where J^n and L^n summarize solution cost and the feature limits/constraints, respectively, of the n th iteration and the set $\{t^n, x^n, u^n\}$ specifies the full-state trajectory (i.e., time sequence t_n , position/velocity vector sequence x_n , control inputs u_n) to be executed. This goal is achieved through intelligent selection of a trajectory planning function and selection and adjustment of a weight vector Ω^i that influences the relative importance of terms in the cost functional J . *EVAL* incrementally builds a history of activities $\{\mathbf{HIST}\} = \{\mathbf{HIST}^1, \mathbf{HIST}^2, \dots\}$ with \mathbf{HIST}^i including a record of the function used by *TPLAN*, the initial solution estimate, and the weight vector Ω^i used for the i th iteration. *EVAL* can then use $\{\mathbf{HIST}\}$ to identify which weight adjustment strategies it has already employed, to avoid infinite loops. Figure 5 shows the possible paths through the architecture.

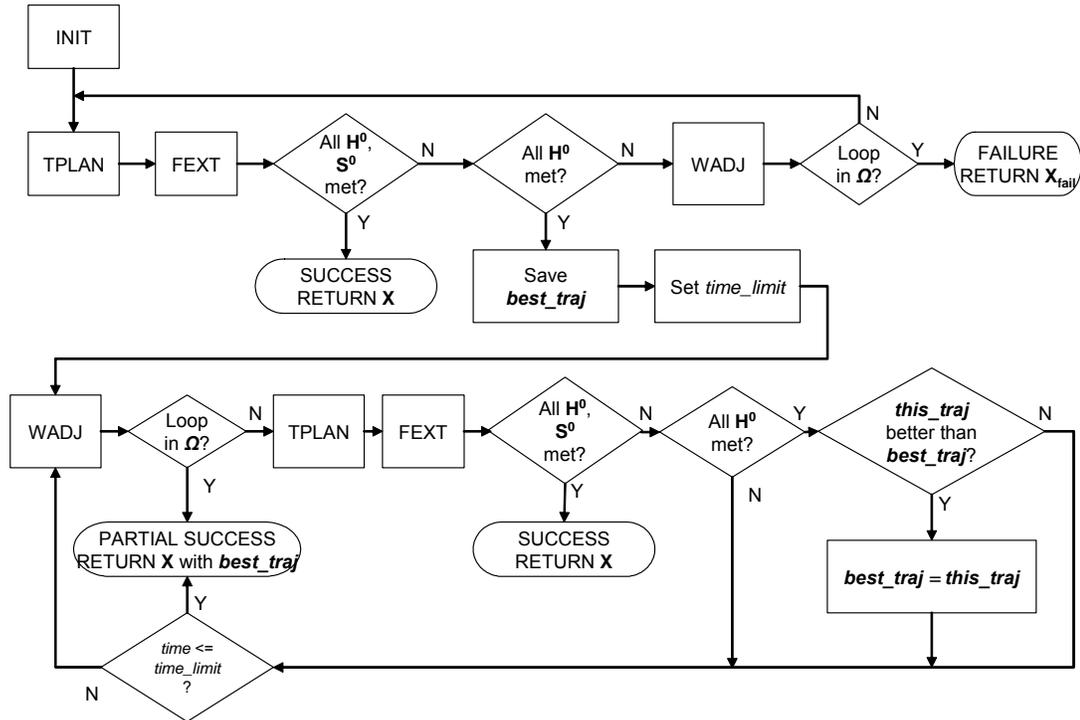


Figure 5: Paths through the architecture

INIT initializes the problem state, \mathbf{P}_0 , selecting a trajectory planning function and providing a vector of cost functional weights Ω^l and an initial guess for the trajectory x_0 , if needed. Next, *TPLAN* generates an optimal trajectory based on this initial \mathbf{P}_0 . *FEXT* extracts the relevant trajectory features, F^l , and returns them to *EVAl*. If all elements of F^l are within the limits L^0 , the trajectory is deemed a satisficing solution and the solution \mathbf{X} is returned by *RETURN* to the user and the higher-level strategic planner. Otherwise, the nature of the limit failures is analyzed. If the user-imposed hard limits (e.g., those which *must* be satisfied) are not met, *WADJ* is called to determine new weights Ω^{i+1} . *EVAl* then checks $\{\mathbf{HIST}\}$ for a cycle in the weights; if one has occurred, the process

fails. This loop continues until the hard limits are met. If the hard limits are met but the user-imposed soft limits (e.g., those which we *prefer* to be satisfied) fail, the process enters a second loop. The goal in this case is to improve the solution so that it meets all of the limits, hard and soft, but to provide some solution within a limited time. The user may adjust this *time_limit* to reflect his desire for constraint satisfaction versus his desire for timely results. (The loop which attempts to meet \mathbf{H}^0 does not have a similar *time_limit* because we assume that the user has no use for the illegal trajectory; that loop will run until \mathbf{H}^0 are met, the process fails, or the user interrupts it.) In any case, the trajectory which satisfied the hard limits but not the soft limits is stored, so that a legal trajectory is guaranteed to be returned. The process continues as for the hard limit case, with the exception that if a trajectory is found which again satisfies the hard limits but not the soft ones, this solution is compared to the prior *best_trajectory*. If it is a “better” solution than *best_trajectory*, it replaces *best_trajectory* in memory. Currently, the 2-norms of the respective error margin vectors, *margin_{error}*, are compared to determine which solution is “better,” and the solution with the smaller overall error margin norm is kept. The error margins are defined as:

$$margin_{error,j}^i = \begin{cases} \mathbf{F}_j^i - \mathbf{L}_j^0, & \mathbf{F}_j^i \neg \circ \mathbf{L}_j^0 \\ 0, & \mathbf{F}_j^i \circ \mathbf{L}_j^0 \end{cases} \quad (3.1)$$

where the operator \circ indicates that a features meet its corresponding limit, whether they are below an upper bound, above a lower bound, or within a range. For the i th iteration of the trajectory planner, the j th feature is compared to the j th element of the vector of limits. The elements of *margin_{error}* were not normalized for limit size in its first version and an upgrade was never made; certainly some sort of normalization would be more

appropriate. An element of \mathbf{margin}_{error} is negative when a lower limit is violated and positive when an upper limit is violated. (If \mathbf{L}_j^0 is a range limit, the end of the range that is violated is used.) When the 2-norm is taken, all the values are of course positive. However, the sign of $\mathbf{margin}_{error,j}$ is important because it is used by $WADJ$ to determine the direction of the weight change.

We now examine each of the architecture components in more detail.

3.1 Initialization

To guide the search toward an acceptable solution, the initialization routine $INIT$ (Figure 6) translates knowledge about the domain \mathbf{D} , constraints \mathbf{L}^0 and obstacles $\{\mathbf{O}\}$ into choices for the trajectory generation routine $TPLAN$, any seed information \mathbf{x}_0 required by $TPLAN$, and an initial weight vector $\mathbf{\Omega}^I$.

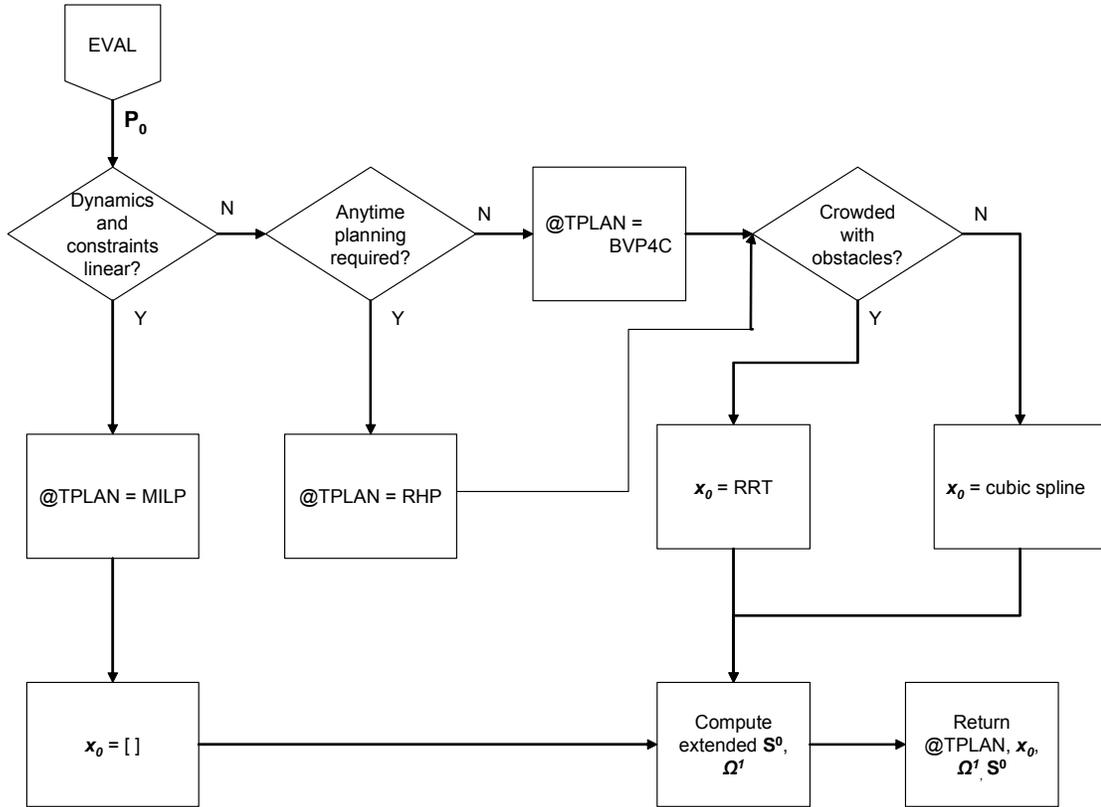


Figure 6: Initialization procedure

Most any trajectory generation tool set that optimizes over a weighted cost function can be incorporated into the architecture. With multiple tools in place, information for choosing between them must be made available. Figure 6 illustrates choices between a mixed integer linear programming module (MILP) [26], a receding horizon planner (RHP) [49], and MATLAB’s collocation-based boundary value solver BVP4C [50]. User-provided information as well as domain information guides the choices, although making a choice among multiple solvers is beyond the scope of this work which relies strictly on collocation, the strategy we consider more flexible than MILP given nonlinear dynamics and more mature than receding horizon algorithms.

Depending on the choice of trajectory planner *TPLAN*, some initial guess may need to be supplied to the trajectory generator (e.g., for collocation). We use a cubic spline which satisfies boundary conditions \mathbf{x}_θ and \mathbf{x}_f . In the future, it may be desirable to use a Rapidly-expanding Random Tree (RRT) [23] if the area in which the robot moves is very cluttered with obstacles. The RRT can find a dynamically plausible (but non-optimal) trajectory through the space. This solution can then be used as an initial guess for one of the optimization routines.

Once *TPLAN* and any inputs it requires are chosen, the limits \mathbf{L}^0 are used to compute the initial weight vector $\boldsymbol{\Omega}^I$ and an extended version of the limits themselves as described below. When no hard or soft constraints are specified, *INIT* defaults to equal weights ($\boldsymbol{\Omega}^I_{default}$) for all terms of the cost functional.

This computation of $\boldsymbol{\Omega}_I$, shown in Figure 7, accepts limits in either the form of qualitative adverbs (e.g., “quickly” or “safely”) with optional adverbial modifiers (e.g., “very”) or numeric constraints on a trajectory feature (e.g., “maximum speed ≤ 5 m/s”). The numeric constraints may be either hard or soft.

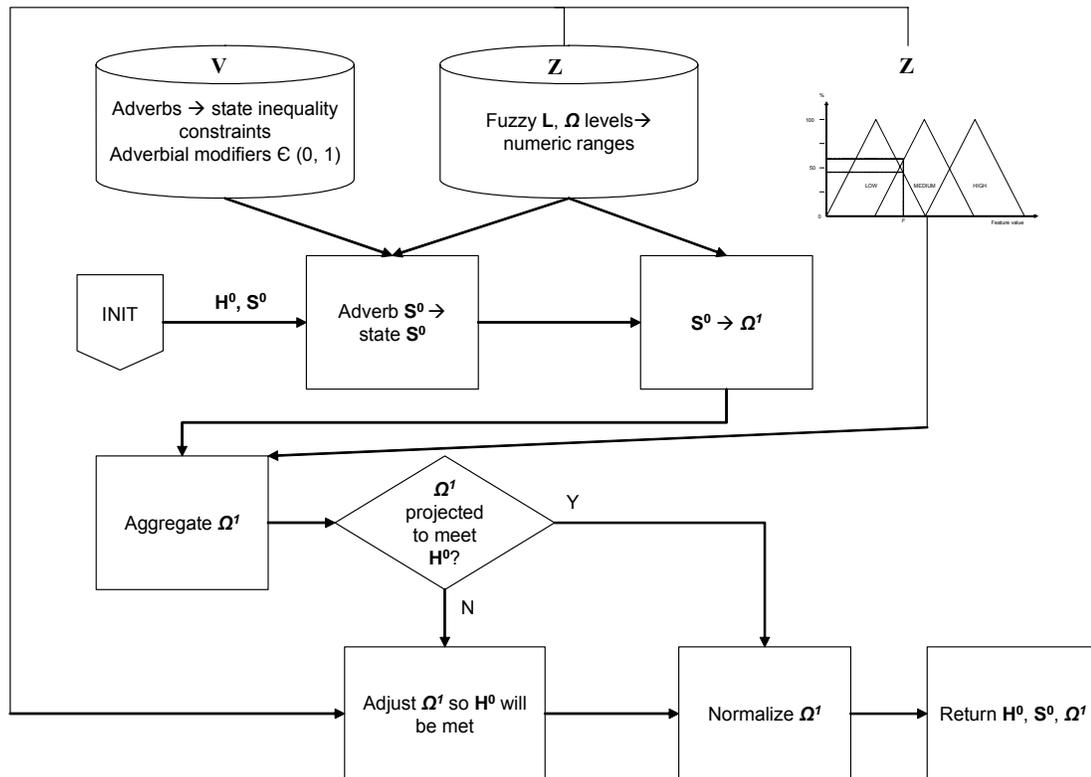


Figure 7: Calculating Ω^I and extended S^0

We deal firstly with the adverb constraints, all of which are considered soft constraints. We define each adverb that our system understands in terms of the trajectory features we can extract. For example, “quickly” involves maximum forward speed (*max-speed*) and average forward speed (*avg-speed*). We further define fuzzy levels for each feature. “Quickly” involves “high” *max-speed* and “high” *avg-speed*. These definitions are stored in the fuzzy language database V .

The ranges for these values are defined based on vehicle performance constraints and a typical definition of each adverb relative to these constraints. We used the same data that generated the weight adjustment curves to correlate feature levels to weight

levels (see “Weight Adjustment,” below), resulting in our fuzzy rule set \mathbf{Z} . We cannot use the weight adjustment equations themselves, as they require constants that cannot be calculated until the first set of trajectory data is available. But we can say that “low” values of a given weight produce “high” values of *avg-speed*. We define ranges for the weight levels as well. Since all of our fuzzy levels are defined geometrically as triangles with fixed endpoints, their centroids are fixed along the weight axis. Defuzzification for each weight correlated with each feature is the process of looking up this centroid value. This numeric information is also contained within the fuzzy rule set \mathbf{Z} and is represented by the generic fuzzy membership function depicted in the upper right of Figure 7.

Any numeric soft constraints in \mathbf{S}^0 are then also fuzzified (so that “ $4 \text{ m/s} \leq \text{max-speed} \leq 5 \text{ m/s}$ ” becomes “*max-speed* high”) and appropriate fuzzy weight levels found by referencing \mathbf{Z} . (In our current implementation, we only dealt with and so only treat soft numeric range constraints. Soft upper and lower constraints would be treated like hard constraints; see below.) We do this because, at this point in the algorithm, we lack quantitative equations which could directly correlate hard or soft numeric constraints directly to weight values. We have a general understanding that “*max-speed* high” requires “time weight high,” and fuzzy estimates of what values constitute “high” for both parameters. But we as yet lack a predictive equation which would accept as input a desired feature value and give as output an estimated weight. After our first trajectory has been computed, we can then solve for certain parameters that do allow such estimation, but here in *INIT* we do not have the required data.

Taken all together, these \mathbf{S}^0 represents the user’s *preference* for the vehicle’s behavior. It now remains to combine these preferences into a single weight vector, and

then to determine if the result is likely to satisfy any hard constraints \mathbf{H}^0 (the user's *requirements* on the vehicle's behavior) that have been given.

To combine all of these weights, we find a centroid that represents an average over each feature's "ideal" weight vector. The "mass" used for this centroid computation is an optional adverbial qualifier that can be stated in the adverb constraint. So "safely but *a little* quickly," where "a little" is the adverbial qualifier, will place more emphasis on weights resulting from features related to "safely" than weights computed from the "quickly" term.

Now we have our initial weight vector \mathbf{Q}^I . If there are additional hard numeric constraints on features in \mathbf{H}^0 , we fuzzify those constraints as we did for the numeric soft range constraints. Then we check if the current weight vector, when fuzzified via the rules in \mathbf{Z} , correlates to that fuzzy level. That is, if we require a hard limit "*max-speed* \leq high" and the relevant weight in \mathbf{Q}^I is "low," are we likely to generate an acceptable behavior? If we are not, the weight value in the required fuzzy range that is closest to the current weight value is selected. So, to continue the example, if we require a "high" weight to elicit "high" *max-speed*, but our current weight is "low," the algorithm will select the value in the "high" fuzzy weight triangle closest to the "low" triangle. Since this overrides the centroid calculation that was built on user preference, it is done only for hard limits \mathbf{H}^0 , which we assume the user *needs* rather than *wants*.

If this work is extended to include soft upper or lower numeric constraints, a similar checking procedure would be used to see if they would probably be satisfied by \mathbf{Q}^I before the hard constraint check had been performed.

3.2 Trajectory Generation

The *TPLAN* module takes the weight vector $\boldsymbol{\Omega}^I$ calculated by *INIT* as well as any initial estimates it requires for itself that *INIT* generates. It returns a full state trajectory, including position, velocity, and control inputs at each time step. The only requirement we place on how *TPLAN* does this is that it must accept some vector of adjustable weights which we can manipulate to change the features of the trajectory. We do not even require that *TPLAN* return optimized trajectories, although we have chosen to use such a *TPLAN*. The computational process of finding the optimal trajectory, given user preferences, was not the primary focus of this research. Users can incorporate their preferred trajectory generator (*TPLAN*) into this architecture, so long as it meets our requirements.

The current implementation assumes a cost functional in the form of a weighted sum of terms. If the trajectory generation process were to use substantially different adjustable parameters, new strategies for *WADJ* would have to be developed. For example, an evolutionary algorithm which used a weighted sum as its fitness function could be used as a *TPLAN*, since by changing the weights we can change what is a “fit” trajectory and hence what the EA will return. But a multi-objective EA that uses Pareto optimality as a fitness function would be different, since it by nature returns a whole family of nondominated trajectories. Injecting user preference into multi-objective optimization is an active area of research and techniques similar to the ones described here may be helpful, but some work would have to be done to adapt them to the field.

In our experimental domains, we used the collocation-based BVP4C solver and Henshaw’s extension BVP4C2 [51] in MATLAB for the split boundary value problem.

We found that we needed to make several assumptions to get the code to work, some of which may affect the probability that the solution returned is the true optimal for the given parameterized cost functional and not just a local minimum.

First, although BVP4C and BVP4C2 can theoretically solve for a free end time, to make a problem with free end time converge to a solution requires a very good initial guess, both as to the shape and the duration of the trajectory. Without such an educated guess, the solver returns a nonconvergent (invalid) solution. We instead provided the solver with an array of possible final times and made the assumption of smoothness between them. The solver iterated over each possible final time in the array and the costs for each resulting trajectory were compared. If the lowest cost trajectory was somewhere in the middle of the array, the times on either side of the lowest-cost trajectory were taken as new upper and lower time limits for a new array with smaller steps between final times. If the lowest-cost trajectory were at either end of the array, the current time step was preserved and the lowest-cost trajectory was used as either the new high or low end of the array. The lowest possible completion time was set for 1 sec; if forming the new array would require a final time less than one second, the lowest possible time was set to 1 sec and the time step recomputed to fit evenly between 1 sec and the high end of the time array. Second, although the collocation routines are fairly robust, they are still sensitive to certain numeric artifacts. We discovered, for instance, poor convergence for certain final times. The algorithm would converge well for some $t_f + dt$ and for $t_f - dt$, but for t_f itself, no good answer would be found. Other variables, including continuation schedules (where obstacle potential functions or vehicle step-response outputs are slowly brought from some smooth approximation to their sharper, final shape) and obstacle

placement (obstacles symmetric with respect to the initial path guess especially) could also cause problems. Unfortunately, our automated data generation scheme did not allow for the easy detection of these cases, nor did we have the time to hand-tune every parameter in every optimization run to get the best results. (Dozens to hundreds of optimal trajectories are generated for each set of \mathbf{L}^0 that we attempt to satisfy.) In many of these cases, the nonconvergence was not pathological; desired error bounds of 0.1 m might be exceeded by errors of 0.2 m, for instance. Further, these cases were rarely the lowest-cost trajectories, and we assumed that we could validly select the lowest-cost convergent trajectory in the presence of nonconvergent trajectories. Those cases where nonconvergent trajectories were selected as optimal will be discussed in the relevant Results sections in Chapters 4 and 5.

3.3 Feature Extraction

Feature extraction (*FEXT*) is a computational routine that takes as an input the generated trajectory and extracts from it certain gestalt properties useful in evaluating the trajectory. Total battery power expended, total time, maximum speed, average speed, and maximum acceleration during the traverse are typical features. A complete list of features, together with the qualitative adverbs they define, is found in Appendix A. Some are maximum or minimum values which are straightforward to express in \mathbf{L}^0 ; others are averages or percentile values that give an overall impression of the trajectory. The “percent plateau” values, for example, are the output of a routine which checks the velocity and acceleration profiles for significant periods of time (at least 10% of the total duration) during which the relevant value fluctuates no more than 1% of its total range.

This was intended to give a numerical approximation to the human technique of looking at a trajectory profile and estimating how “flat” it is.

The features in Appendix A are very loosely domain-dependent. More accurately, they can be defined independently of a domain, but simply may not apply to a particular domain. An average rotational rate is meaningless in a 2DOF simulation that has only linear motion in the x - y plane. Average vehicle separation is inapplicable to a single vehicle domain. Likewise, certain adverbs or verbs may not be relevant to all domains, even though they exist outside of them. We might prefer an Army field vehicle to move “stealthily,” but there is little call to require a space robot to behave in such a fashion.

3.4 Weight Adjustment

The development of good weight adjustment (*WADJ*) heuristics was a key part of this work. Our goal was to automate the process by which cost functional weights are tuned. This is typically done by hand, by a domain expert, until the desired results are achieved. We have attempted to encode these desired results into the limits \mathbf{L}^0 , as functions of the features defined above. What remains is to extract domain expert knowledge and techniques and automate the adjustment process.

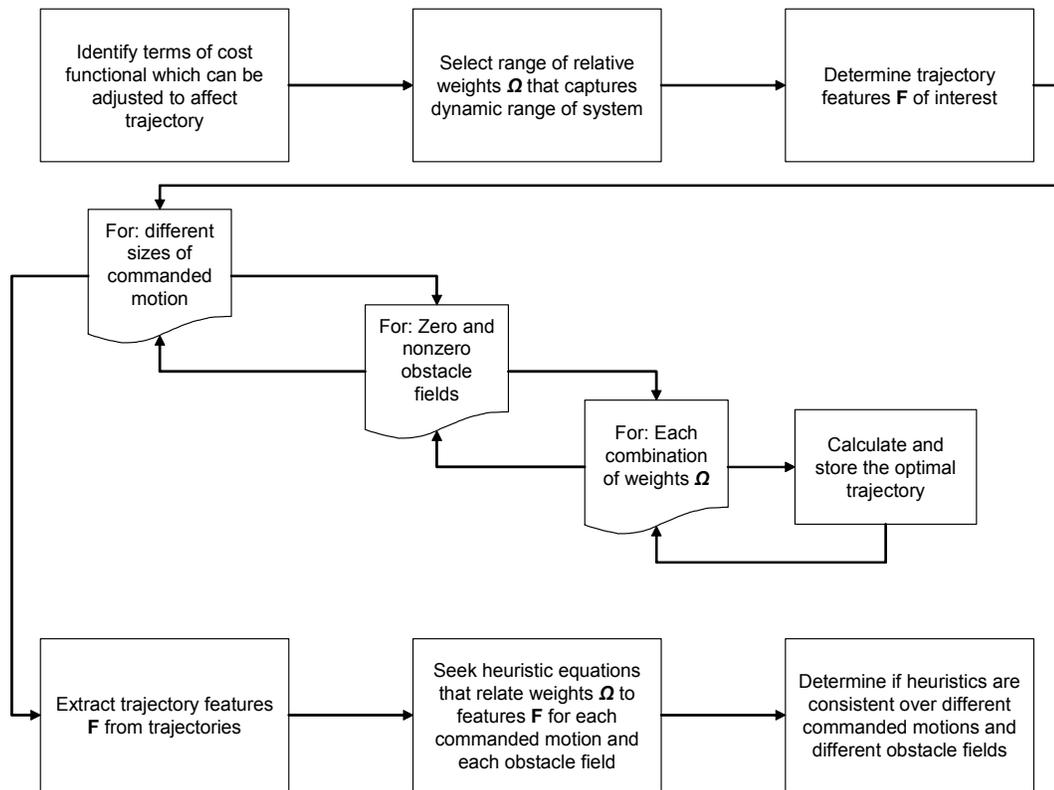


Figure 8: Developing *WADJ* rules

Early in this research we discovered that many of the features in our set could be expressed as functions of the weights used in the cost functional. The cost functional for the 2DOF linear domain contained three fairly typical terms: one penalized electrical energy use, another penalized time, and the last was the obstacle penalty function shown in Figure 2. The cost functional for the 6DOF nonlinear domain contained four terms, penalizing fuel used for thrust, electrical energy used for torque control, time, and an obstacle penalty function. Despite the different dimensionalities, cost functionals, and system dynamics, we were able to treat the generation of our *WADJ* heuristics in a similar fashion in both cases.

Each test matrix covered a combinatorial set of cost functional term weights, Ω . Since a cost functional can always be normalized, we knew that we would be looking at relative weights rather than be concerned with their absolute magnitudes. Early experimentation led us to conclude that a range of two orders of magnitude, from 0.1 to 10, would be sufficient to see a broad range of dynamic behavior in our systems. (If this were ever not the case, it would be a simple matter to extend the scope of the weight vector to see an even broader range of behavior.) We allowed each individual weight to vary through $\{1, 2, 4, 8\}$ which, in combination, gave us relative weights from 2^{-3} to 2^3 , which covered our two orders of magnitude.

We varied the magnitude of the commanded motions and also the number of obstacles in the field. This was to ensure that the answers we were getting were not too specific to a single domain subcase. At least one obstacle was needed to test features like minimum separation to obstacles (*min-sep*); adding more obstacles would show how performance changed as the field became more cluttered. The empty field and single obstacle test cases could be performed very quickly, because the absence of many obstacles greatly simplifies solving the problem.

Once we had collected the trajectories, we used *FEXT* to compute the overall trajectory features in which we were interested. For features relating to time (e.g., velocity, acceleration, power), we found strong power relationships between the feature values and the *ratio* of the energy or fuel term weight and the time term weight (W_1/W_2). That is:

$$time_feature = c_1 \left(\frac{W_1}{W_2} \right)^{-\alpha} \quad (3.2)$$

The exponent $-\alpha$ in these equations stayed fairly constant across field sizes, although the constant coefficient c_1 varied.

Similarly, for path-based features like minimum separation from obstacles (*min-sep*), there was a linear relationship between the feature and the influence limit (*LIM*) in the obstacle penalty function:

$$path_feature = c_2 LIM \quad (3.3)$$

Figure 2 shows what *LIM* is. There is an elbow in the obstacle penalty function that occurs at the edge of the obstacle; in Figure 2, this occurs at a value of 1 m on the x-axis. Some distance later, at 2 m, the obstacle penalty function goes to zero. The distance past the obstacle edge over which the penalty function goes to zero is *LIM*. Figure 9 shows an obstacle (circle with solid red line). The obstacle penalty term was also weighted, but the effect of the weight was nearly negligible in comparison to varying *LIM*, as shown in Figure 9.

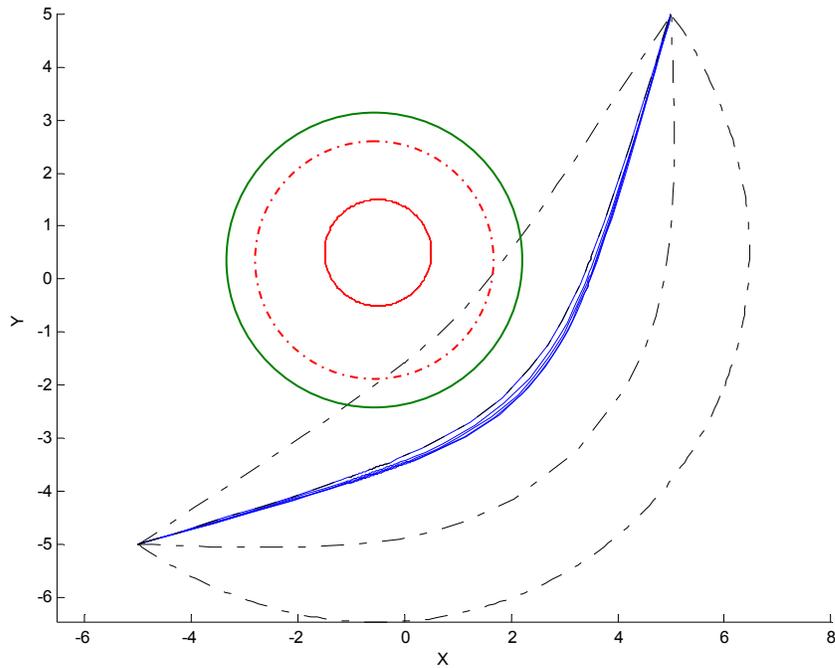


Figure 9: Effect of changing obstacle penalty weight (solid lines) and obstacle penalty function influence limit (dashed lines) on separation from obstacle

The dotted red line in Figure 9 corresponds to one *LIM* value. The black line closest the obstacle is a path which goes within this *LIM*. That isn't a problem by itself; we assume the trajectory generator found the cost incurred by getting closer to the obstacle was less than some other cost. However, we show a hypothetical constraint on path nearness to obstacles, *min-sep*, in green. (This *min-sep* was not used for these data runs; this is simply an illustration.) In this case, the path is less than *min-sep* away from the obstacle. This is a constraint violation, and the cost functional would be adjusted to correct it. As Figure 9 shows, the most efficient way to adjust path nearness to obstacles

is to change *LIM*. Increasing *LIM* increases the distance between the vehicle and the obstacle, meeting the required *min-sep* easily.

Rather than attempt to calculate tables for all possible constant coefficients c_1 and c_2 of these equations for all possible field sizes, they are computed online, using the current weight and feature values to back out the coefficient value. The coefficient, together with the *desired* feature value (e.g., the limit, if it was passed), are then used to recompute the weights.

As more obstacles are added to the field, the rules' accuracy is affected. (Domain-specific examples are given in the relevant chapters.) They remain, however, useful rules of thumb for guiding weight adjustment, as our results will show.

The effects of changing the fuel/time weight ratio or the energy/time weight ratio and *LIM* were largely independent. This allowed us to decouple them, an important and useful assumption. They are not, however, entirely independent. As *LIM* decreases, for example, more direct paths which save both fuel and time can be found. The effect is not dramatic, but can mean the difference between a successful and unsuccessful solution. If the standard *WADJ* rules have failed to find a solution that mediates between competing time and fuel goals, a secondary *WADJ* rule will change *LIM* in an attempt to take advantage of this secondary effect and find a successful solution.

We were concerned that the 6DOF spacecraft domain with nonlinear dynamics would not be amenable to this *WADJ* rule-generation process. Results for the 6DOF domain were in fact very similar to those for the 2DOF domain. A notable difference was the torque weight term, which is unsurprising given the coupled nature of the

rotational and translational mechanics. Our torque heuristic is discussed in detail in Chapter 5.

3.5 Implementation

Initially, the *TPLAN* and *FEXT* modules were implemented in MATLAB. While MATLAB runs more slowly than equivalent code implemented in C or C++, this gave us ready access to the BVP4C function and the BVP4C2 code and domains developed in [51] and minimized coding overhead. One could expect significant execution speed increases with a translation to C or C++, making this work quite practical for complex dynamic and constraint sets.

INIT and *EVAL* were first implemented in the cognitive architecture ACT-R [52], which is itself implemented in Lisp [53]. (*WADJ* was at this point a computational module written in Lisp, since it was used by *EVAL*.) Since we were attempting to model a decision making process currently handled by humans, our thinking was that a “cognitive model” would be best suited for the task. However, the code which resulted was far from a cognitive model, despite its implementation in a cognitive architecture [54]. Furthermore, as we moved into the nonlinear 6DOF domain, it became doubtful that human decision making would even be something that we would want to emulate. Humans are very adept at controlling linear systems, but our physical intuition breaks down for nonlinear ones. As the reasons for using a cognitive architecture became more uncertain, *INIT*, *EVAL*, and *WADJ* were all migrated to MATLAB since this offered the certain benefit of easier integration. Additionally, the implementation of iterative loops required for constraint checking was in many ways simpler and more obvious in

MATLAB. Mathematical computations, while certainly possible in Lisp, were also easier to represent in MATLAB.

Implementing these routines in MATLAB gave further insight into the strengths of the cognitive architecture. ACT-R excels in pattern-matching, which it uses to find the right “procedural knowledge” (an if-then rule) given the “declarative knowledge” (a piece of data) and the goals that it currently has. The main work of building the ACT-R model was finding the right representation of the declarative knowledge to enable general and useful procedures that were not all simply one special case after another.

When this was done correctly, it made extending the modules very easy. To add a new adverb, for instance, the user had only to write declarative chunks for the features which composed the adverb, and the levels at which those features were present. If any new features were defined, their relationship to the weights would have to be defined as well, but that would also only be a small set of “chunks” of declarative knowledge. The new adverb and new features could then be treated just like all of the pre-existing adverbs and features by the system.

In MATLAB, adding a new adverb requires adding new cases to switch commands in several subroutines. The actual amount of information that the user must add is not much more than in the ACT-R case, but it is not all contained in one localized area. The pattern-matching capabilities of ACT-R are simply more elegant than those in MATLAB and, when dealing with symbolic information such as words, this gives ACT-R a distinct advantage. Also, ACT-R and other cognitive architectures have learning functions built into them, which would make it easier for the system to learn individual users’ preferred fuzzy definitions. (That is, if the user always responds, “No, faster!” to

the returned trajectory for something “very fast,” the system can update the values for “very fast” to reflect this.) So, while the cognitive modeling aspect of the cognitive architecture has proven to not be very important to this work, the capabilities of the architecture may yet be useful in extending and generalizing the work.

4 Two Degree of Freedom Point Rover

A simplified 2-D domain model was developed as an intuitive baseline case for our architecture and as a method of developing initial modules to populate the Figure 4 architecture.

4.1 System Dynamics

We began our investigations with a 2DOF point-robot model, imagining a rover-like robot traveling in a plane, using electric motors for propulsion. We used this highly simplified domain to gain an intuition into the process of adjusting the cost functional weights and computing, then evaluating, the resulting trajectories. The model has simple linear dynamics:

$$\begin{bmatrix} \mathbf{x}'(t) \\ \mathbf{x}''(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -c_s/m \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{x}'(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{u}(t)/m \end{bmatrix} \quad (4.1)$$

where m is object mass and c_s is the coefficient of sliding friction. We assume an idealized system without motor saturation and perfect trajectory tracking.

4.2 Terms of the Cost Functional

In robotic applications, two concerns are usually paramount: conserving fuel or battery power and not running into obstacles. Additionally, there may be time constraints on a mission. Equation (4.2) gives the cost functional J and weight vector

$\boldsymbol{\Omega}^i = [W_1, W_2, W_3, LIM]$. Each term is described more fully below.

$$J = \int_{t_o}^{t_f} \left(W_1 \sum_{j=1}^{\text{length}(\mathbf{u})} (u_j^2(t)) + W_2 + W_3 \sum_{i \in \{O\}} (o_i(r_i)) \right) dt \quad (4.2)$$

4.2.1 Energy Use

Since we are considering a hypothetically battery-powered vehicle, we followed [46] in adding a minimum-energy term. We have simplified his representation somewhat; the author of includes a potentially different weight for each $u_j^2(t)$ in the control vector. We do, however, differentiate control vector elements in the 6-DOF example shown below, where translational actuators require fuel and rotational actuators require electrical power.

4.2.2 Time

Since J is an integral, the cost functional only needs a constant term, W_2 , to minimize time. Over the integral, the resulting $W_2 * t_f$ will be minimized.

4.2.3 Clearance to Obstacles

To keep the vehicle away from obstacles, we add the term $W_3 * \sum_{i \in \{O\}} o_i(r_i)$, presuming simple circular obstacle geometries. (A sum is used, rather than a maximum, to preserve the smoothness criterion required for convergence in the collocation solver.) We assume that our agent has an *a priori* map of the region it will traverse, possibly obtained from an orbiter or aerial overflight. $o_i(r_i)$ is a function which increasingly penalizes the agent as it approaches obstacle i . W_3 is the relative weight in overall cost from Equation 4.2, and r_i is the distance from the vehicle to obstacle i 's center:

$$r_i = \text{sqrt}((x - x_o)^2 + (y - y_o)^2 + (z - z_o)^2) \quad (4.3)$$

$o_i(r_i)$ is maximum value MAX over the center of the obstacle, attains fixed value K at the obstacle boundary a distance R_i from the obstacle center, and decreases to zero at a distance LIM away from the obstacle's edge (Figure 2). These constraints are described by Equation 4.4 and also include smoothness conditions. A third-order polynomial solution (Equation 4.5) that meets these constraints was selected as $o_i(r_i)$. This solution is positive within the region of influence ($r_i < LIM$) and effectively repels the path given sufficient MAX , K , LIM values.

$$\begin{cases} o_i(0) = MAX \\ o_i(R_i)_{left} = K \\ o_i(R_i)_{right} = K \\ o_i'(R_i)_{left} = o_i'(R_i)_{right} \\ o_i''(R_i)_{left} = o_i''(R_i)_{right} \\ o_i(LIM) = 0 \\ o_i'(LIM) = 0 \\ o_i''(LIM) = 0 \end{cases} \quad (4.4)$$

$$o_i(r_i) = \begin{cases} K(c_1 r_i^3 + c_2 r_i^2 + c_3 r_i + c_4), & r_i \leq R_i \\ K(c_5 r_i^3 + c_6 r_i^2 + c_7 r_i + c_8), & R_i < r_i \leq R_i + LIM \\ 0, & r_i > LIM \end{cases} \quad (4.5)$$

MAX and K were chosen in an *ad hoc* fashion, after some experimentation. They should be sufficiently large compared to the other costs to appear nearly infinite; Eq. 4.5 is equal to K when the vehicle is touching the obstacle, and we would like to model any further passage into the obstacle as being of infinite cost. (In practice, if the path does go within the obstacle, this will violate an implicit hard *min-sep* limit and cause a

recalculation of the trajectory.) The coefficients c are found by solving the two third-order equations and their first and second derivatives to satisfy all of the smoothness constraints.

In our analysis of the effects of changing the weighting parameters on the trajectory, we included changes not only in W_3 but also in LIM . They had distinctly different effects, as show in Figure 9 above.

In addition to these terms, the system dynamic equations are adjoined to J as shown in Eq. 2.9. This ensures that only dynamically feasible trajectories will be considered.

4.3 Development of Weight Adjustment Heuristics and Fuzzy Rules

4.3.1 Weight Adjustment Heuristics

The procedure outlined in Chapter 3 was followed, with $WADJ$ rules and fuzzy correlation rules established for all of the features addressed in the current implementation of the architecture. The general procedure was very similar for each, so one detailed example is presented here, while the rest of the $WADJ$ plots can be found in Appendix C.

Figure 10 shows the data collected for average speed as a function of the ratio of the energy weight, W_1 , to the time weight, W_2 . In the empty 10m x 10m field, this linear system shows very predictable behavior, described well by the Figure 10 equation plotted in black. As obstacles are added to the field, it becomes clear that, while the equation still has value as a heuristic, it is not as accurate in predicting the average speed for a given set of weights as it is in the empty field case.

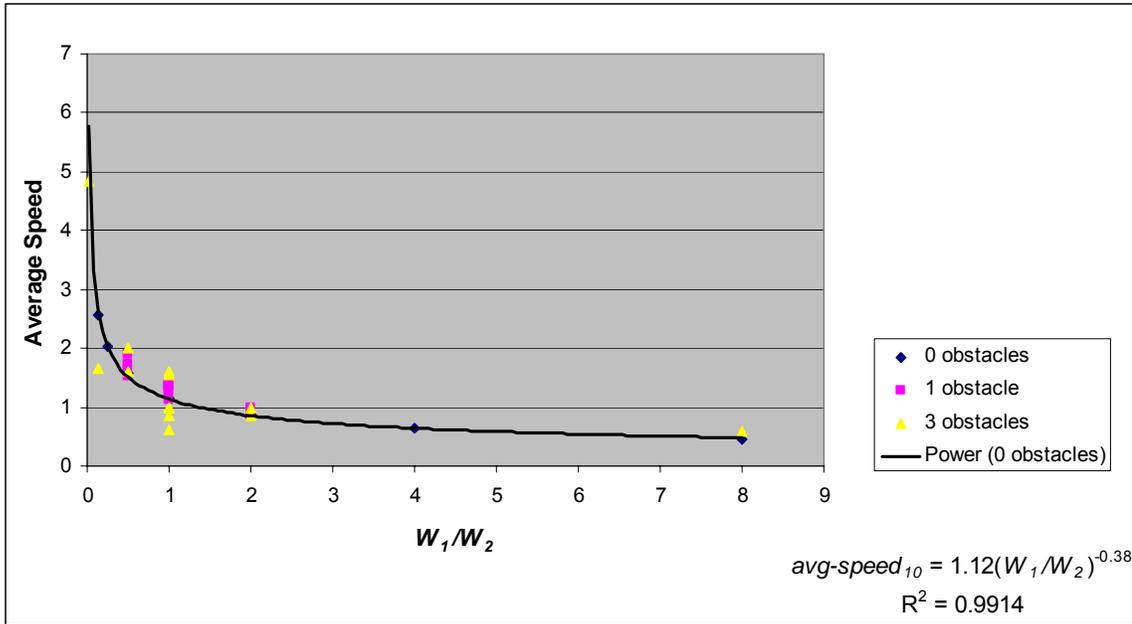


Figure 10: Average speed as a function of W_1/W_2 for zero, one, and three obstacles in a 10m x 10m field

Figure 11 shows the change in the equation linking average speed to the W_1/W_2 ratio as field size changes. 200m x 200m, 20m x 20m. and 2m x 2m fields were used, and the 10m x 10m data from the Figure 10 zero obstacle case is included as well. There is variation in the exponent, particularly between the mid-range and high field sizes and the small field. In practice, we would not expect to operate frequently in such a small field (unless the robot was exceptionally small, in which case we would probably not be operating in the larger regimes).

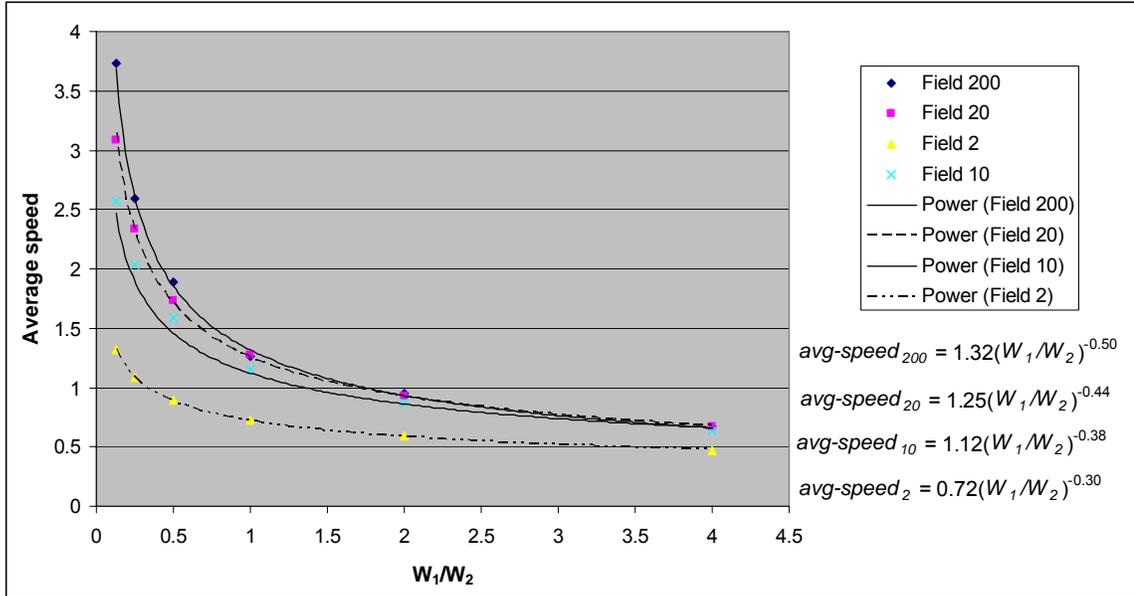


Figure 11: Average speed as a function of W_1/W_2 for four different empty field sizes

Since the equations serve only as a heuristic and become less accurate predictors in cluttered environments (Figure 12), we did not want to impart too much precision to the exponent. An average of all of the exponents yields a value of -0.41; the average of the three largest fields yields a value of -0.44. Both of these can be rounded to a single decimal place as -0.4, which is what was done. This resulted in the *WADJ* equation relating average speed with W_1/W_2 :

$$avg-speed = c_1(W_1/W_2)^{-0.4} \quad (4.5)$$

The constant c_1 is calculated from actual values of *avg-speed* and W_1/W_2 at runtime.

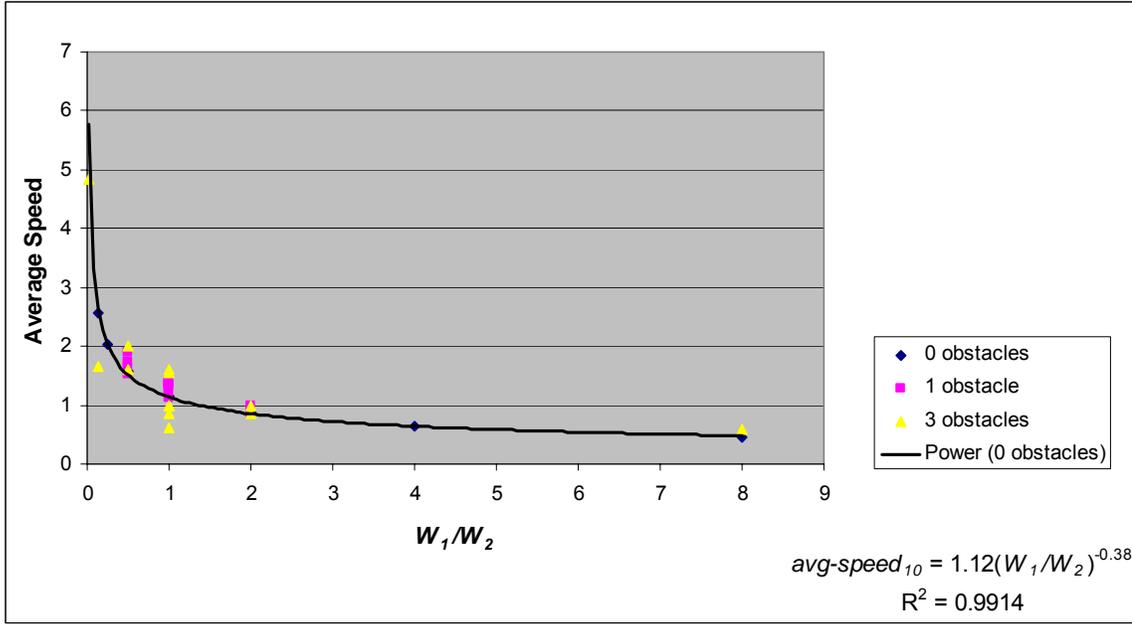


Figure 12: Heuristic equation less predictive as obstacles are added to the environment

4.3.2 Heuristic Equation Verification

To test our procedure, we compared it to analytical results. While there is no analytical solution for a cost functional which includes the obstacle penalty function, one can be derived for the zero obstacle case, where only energy and time are traded off. For greatest simplicity, we examined a system without friction and only along one axis (valid because the components are independent of each other). System dynamics are given in Equation 4.6.

$$\begin{bmatrix} \mathbf{x}'(t) \\ \mathbf{x}''(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{x}'(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{u}(t)/m \end{bmatrix} \quad (4.6)$$

Our simplified cost functional is given in Equation 4.7. As in our implementation, we normalized by the time weight.

$$J = \int_{t_o}^{t_f} \left(\frac{1}{2} W u^2 + 1 \right) dt \quad (4.7)$$

To this we appended the system dynamics. The interior of that integral gives us our Hamiltonian, as described in Equation 2.10. Computing the Euler-Lagrange Equations (Eq. 2.11a-c) yields four linear ordinary differential equations with four unknown coefficients. Final time is also unknown. We use known initial and final states to solve for four of the unknowns and the transversality condition to get the fifth. Solving the system for t_f in terms of final position, x_f , and weight W yields:

$$t_f = \sqrt{x_f \sqrt{18W}} \quad (4.8)$$

We ran our solver for a range of W for $x_f = 10$ m and compared them to the solution given in Equation 4.8. The results are precise to ± 0.005 seconds. The results are summarized in Table 1. The numbers agree to within our precision.

W	t_f computed	$t_f = \sqrt{x_f \sqrt{18W}}$
0.125	3.875	3.873
0.250	4.609	4.606
0.500	5.473	5.477
1.000	6.518	6.514
2.000	7.748	7.746
4.000	9.207	9.211
8.000	10.953	10.954

Table 1: Verifying *WADJ* heuristics with analytical predictions

We note that our BVP4C-based solver is not using BVP4C's ability to solve for a free final time. That proved too sensitive to initial estimates to be of use. Instead, we search with increasing granularity over a range of known final times, narrowing our search in the low-cost region. This method is vulnerable to local minima in cases where the cost functional results are not smooth, as they may be in the presence of many obstacles.

4.3.3 Fuzzy Rule Database Z

The fuzzy logic portion of the initialization routine required three different sets of rules or correlations to be made: weight values to fuzzy levels, feature values to fuzzy levels, and fuzzy feature levels to fuzzy weight levels.

The data in Figure 10 and Figure 11 was generated using W_1/W_2 ratios that were powers of two: 2^{-3} , 2^{-2} , ..., 2^2 , 2^3 . Before the fuzzy logic portion of this research was even fully expressed, these numbers were chosen because they appeared to span the range from “very low” to “very high” relative weights. It seemed reasonable, then, to formally assign these values to the “fuzzy triangles” that relate weight values to their fuzzy levels. The “medium” W_1/W_2 triangle (as in Figure 3), for example, is 100% medium at 2^0 , and 0% medium at 2^{-1} and 2^1 .

The assignment of feature values to fuzzy levels is a matter of judgment. In industry, teams of scientists expend significant effort to ensure that a washing machine's definition of “very clean” is as close to the definition used by the majority of the target consumers as they can make it [2]. Without a real robot or a real consumer (paying customer) to provide feedback for this problem, we have made our own judgments based on the Figure 10 data.

As a starting point, we correlated the feature values at the weight values to fuzzy levels. That is, if W_1/W_2 is “medium” between 2^{-1} and 2^1 , we looked at the feature values in Figure 10 that resulted from those weight values. If it seemed reasonable to call those feature values “medium,” we did so. If, because of rapid changes in the feature value curve, that interval clearly did *not* define a single interval, we looked for the nearest values that could reasonably define it.

The fuzzy relationships fell into two categories: directly and inversely related. In directly related cases, “very low” weight values resulted in “very low” feature values; “low” weight values resulted in “low” feature values, and so on. In inversely related cases, “very low” weight values resulted in “very high” feature values.

Although this implementation of fuzzy logic is not highly sophisticated, it was sufficient to generate improved estimates of initial weight sets for our optimization processes. The full set of fuzzy rules is found in Appendix C.

4.4 Results

To evaluate the performance of our system for the 2D robot domain, five different logical sets of constraints \mathbf{L}^0 of varied complexity were enforced on four different obstacle fields $\{\mathbf{O}\}$ for a total of twenty trials. There were, overall, 28 hard limits (\mathbf{H}^0) and 72 soft limits (\mathbf{S}^0). The simplest constraint set enforced one hard and two soft constraints; the most extensive had two hard constraints and six soft constraints. Appendix B fully details the constraint and obstacle sets used for the test cases.

Each of the twenty test cases was run from a default weight vector $\boldsymbol{\Omega}^j_{default} = [1, 1, 1, 1]$ and from a $\boldsymbol{\Omega}^j$ provided by *INIT*. Only the collocation *TPLAN* BVP4C was used for trajectory generation. The results after one iteration (labeled

$Default^l$ and $INIT^l$) and after program completion ($Default^n$ and $INIT^n$) were examined for both starting weight vectors.

Before looking at the overall results, we present a detailed walk-through of two particular solutions to give the reader a clearer picture of how the *EVAL* and *WADJ* processes work, and how they impact the generated trajectories. The first example shows the routine working smoothly. The second example shows a partial success even in the face of some unexpected behavior.

4.4.1 Detailed Examples

Constraint Set 3 was self-sabotaging. It asked for trajectories with $\mathbf{S}^0 = \{\text{a little quickly, exceedingly inquisitively}\}$. These expanded into high *avg-speed* and *max-speed* and medium-low *avg-speed* and low *min-sep*, respectively. It would not be possible to satisfy both high *avg-speed* and medium low *avg-speed*. The idea, of course, was to create a trajectory was that mostly inquisitive but on the fast end of that. The returned trajectories for Obstacle Sets 2 and 3 achieved exactly that; Obstacle Sets 1 and 4 converged to a solution that favored “quickly.” (User preference information *strength*, as described by the adverbial modifiers “a little” and “exceedingly,” is lost after initialization in the current version of the architecture.) We will look at the result for Obstacle Set 1.

INIT returned initial weights $\boldsymbol{\Omega}^l = [0.954, 1.00, 1.00, 0.50]$. (Recall that W_1 is the energy weight, W_2 is the time weight, W_3 is the weight on the obstacle penalty function term, and W_4 is *LIM*, the obstacle penalty function influence limit.) We normalized the weights by W_2 , the time weight, in the 2DOF case. The result was not too

different from the default weight vector [1, 1, 1, 1], although *LIM* was half its default, anticipating the desire for “medium-low *min-sep*”.

Figure 13 and Figure 14 show the results for all the trajectories; the paths in particular were so similar that it is necessary to graph them all together to detect differences. The *min-sep* requirements were met on all paths. Using the Ω^1 weights, *max-speed* was 1.44 m/s and *avg-speed* was 1.22 m/s. For “inquisitively,” $0.67 \text{ m/s} \leq \text{avg-speed} \leq 1.33 \text{ m/s}$, so this soft constraint was satisfied. It was of course too slow for “quickly, whose “high *avg-speed*” required $2.67 \text{ m/s} \leq \text{avg-speed} \leq 5.33 \text{ m/s}$, and “high *max-speed*” required $4 \text{ m/s} \leq \text{max-speed} \leq 8 \text{ m/s}$. Ω^2 was generated using the larger failure, on *max-speed*, giving $\Omega^2 = [0.07, 1.00, 1.00, 0.50]$.

Such a low W_1/W_2 ratio indicated that time should be excessively optimized, and that was exactly what happened. *max-speed* was raised to 4.17 m/s, just at the low end of satisfying “quickly.” *avg-speed* was raised to 3.21 m/s, also satisfying “quickly.” By necessity, *avg-speed* now violated the “medium low *avg-speed*” constraint within “inquisitively,” exceeding it by 1.87 m/s. The speeds were too fast, so the W_1/W_2 ratio must be raised. Ω^3 was calculated as $[0.67, 1.00, 1.00, 0.50]$.

This result was very similar to the Iteration 1 results. It was a little faster, so *max-speed* and *avg-speed* failed their “quickly” requirements by less this time, 2.44 m/s and 1.34 m/s undershoots respectively. These errors fed into the weight adjustment, giving $\Omega^4 = [0.06, 1.00, 1.00, 0.50]$ - almost the same as Ω^2 . Indeed, the Ω^4 results were close to the Ω^2 results. *avg-speed* was 3.42 m/s, 2.09 m/s too fast for “inquisitively.” *max-speed* was 4.52 m/s; both of these values were on the lower ends of the ranges for “quickly,” as we would hope that they would be as they moved toward some compromise with

“inquisitively.” But the compromise ends here; Ω^4 was too similar to Ω^2 . When the weights were readjusted, the results were too close to Ω^3 (the threshold was set at 0.01). The process terminated and returned the best identified trajectory.

Constraint Set 1 consisted of one hard constraint, $\mathbf{H}^0 = \{max-speed \leq 4.2 \text{ m/s}\}$, and one soft constraint, $\mathbf{S}^0 = \{\text{“somewhat quickly”}\}$. “Quickly” was defined in the fuzzy language database \mathbf{V} as correlating to the state features *max-speed* and *avg-speed*, both at level “high.” The fuzzy rules database \mathbf{Z} defined, for this particular domain with this particular simulated robot, “high max-speed” to be between 4 and 8 m/s; “high *avg-speed*” is similarly between 2.67 and 5.33 m/s. So to meet both \mathbf{H}^0 and \mathbf{S}^0 , the trajectory planner must find a solution with a *max-speed* greater than 4 m/s (the lower limit for “high *max-speed*”) but below 4.2 m/s (the hard constraint). Putting the *max-speed* in that 0.2 m/s window proved difficult; only one case out of eight managed it. We will look at one where it did not fully succeed, to see how the trade-offs were being made, and why the success was only partial.

INIT, given the constraints \mathbf{L}^0 above, returned an initial set of weights $\Omega^1 = [0.214, 1.00, 1.00, 1.00]$. So we should read this as saying minimizing time is roughly five times more important than minimizing energy. This generated the path and trajectory seen in Figure 15 and Figure 16, below. Although it met both \mathbf{S}^0 , the *max-speed* was 4.30 m/s, failing \mathbf{H}^0 by 0.10 m/s. *WADJ* used this error to compute a new W_1/W_2 ratio. The new W_1 was 0.273 relative to a W_2 of 1.0. (W_3 and W_4 were unchanged since we assume the time-dependent properties are independent of path properties.) By raising W_1/W_2 slightly, we hoped to elicit a slightly slower, more energy-efficient trajectory – one that has a *max-speed* below 4.2 m/s.

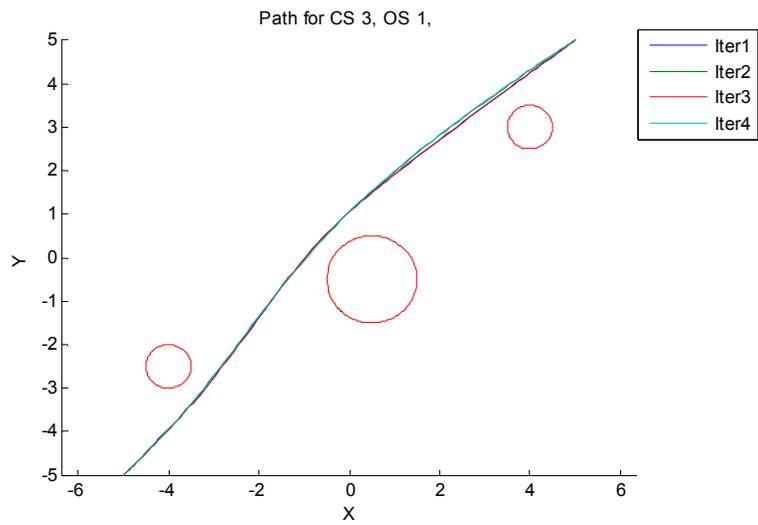


Figure 13: 2DOF paths for CS 3, OS 1

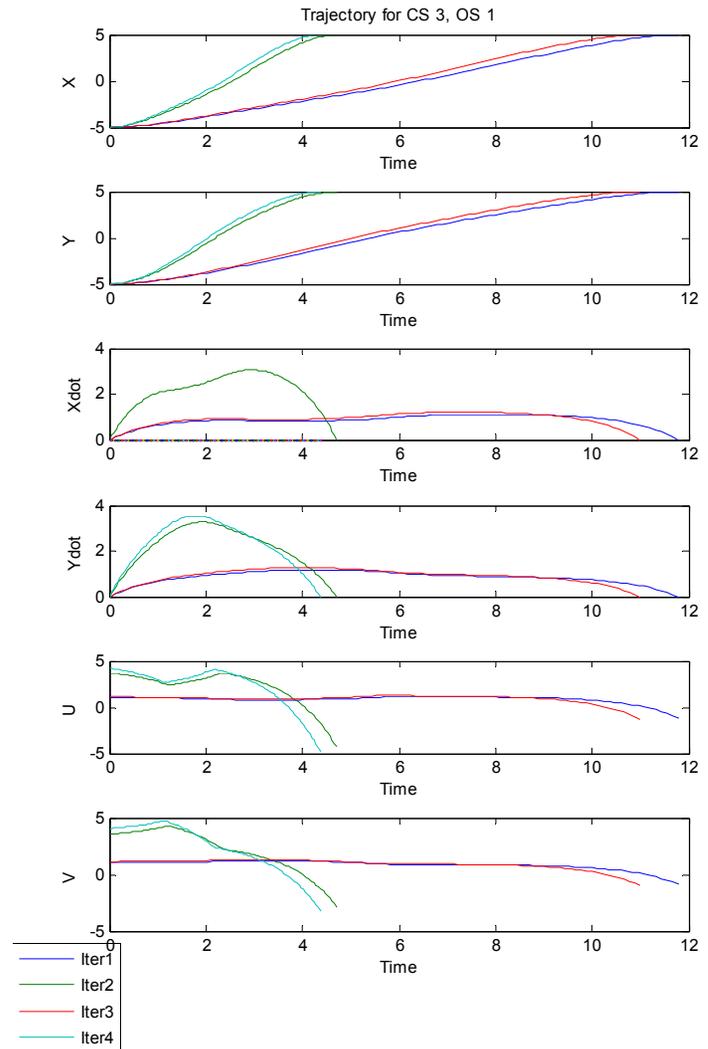


Figure 14: 2DOF path trajectories for CS 3, OS 1

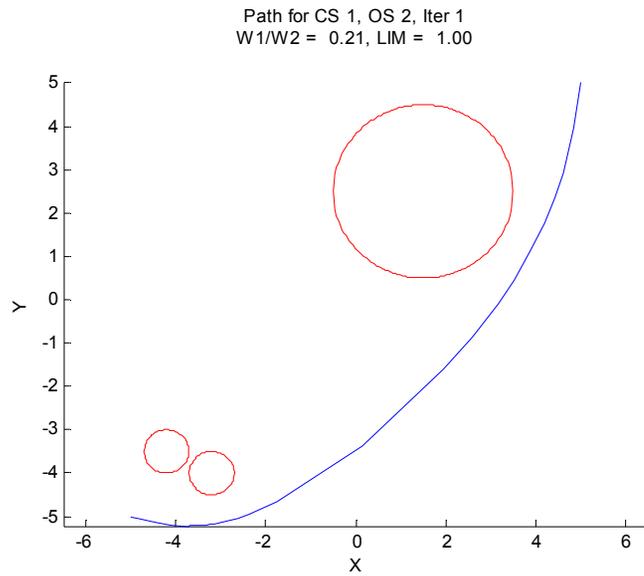


Figure 15: 2DOF path for $\Omega^I = [0.214, 1.0, 1.0, 1.0]$

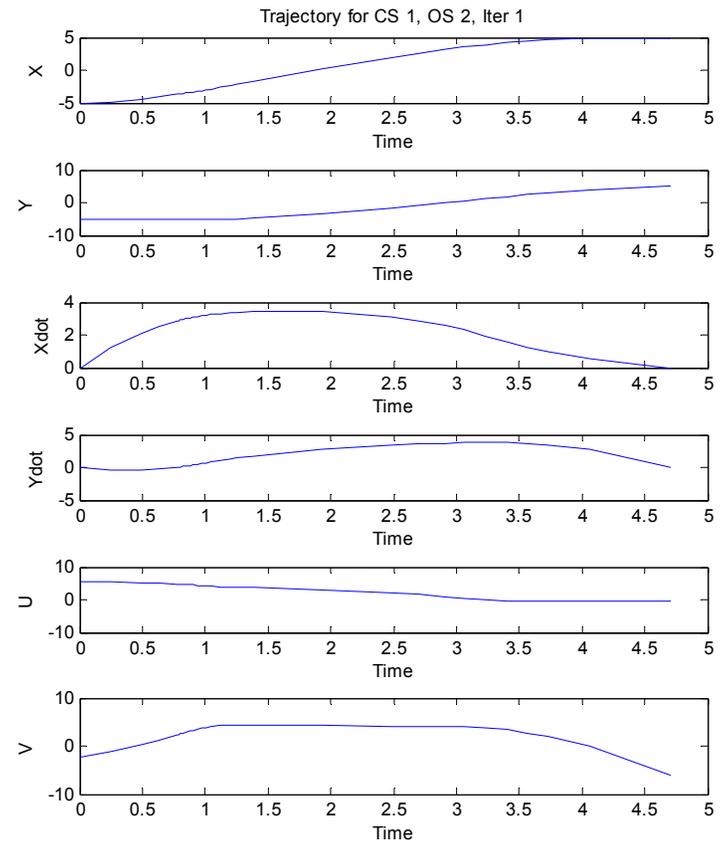


Figure 16: 2DOF trajectory for $\Omega^I = [0.214, 1.0, 1.0, 1.0]$

Figure 17 and Figure 18 show the results from the trajectory planner using Ω^2 . It had in this case converged to a path quite different from the prior iteration. Although this path was longer, the agent took more than three times as long to traverse it. This kept the energy consumption and the speeds down. The *max-speed* here was only 1.26 m/s, meeting \mathbf{H}^0 easily. Of course, this was too slow to satisfy \mathbf{S}^0 , failing the *max-speed* requirement by 2.74 m/s and the *avg-speed* requirement by 1.61 m/s. But we had a partial success, since \mathbf{H}^0 is met, so this trajectory was stored as the *best_trajectory* so far. Then we entered the \mathbf{S}^0 satisfaction loop shown in Figure 5; *time_limit* is set to five iterations. Attempts were then made to improve the solution to better meet soft constraints \mathbf{S}^0 while still meeting hard constraints. New Ω^3 were computed; the W_1/W_2 ratio was 0.015, smaller than it was in Ω^1 because of the rather large amount by which the soft *max-speed* constraint failed. *EVAL* does not yet have the sophistication to check for more than loops in the weights, so it did not notice that this was likely to be a bad choice of weights.

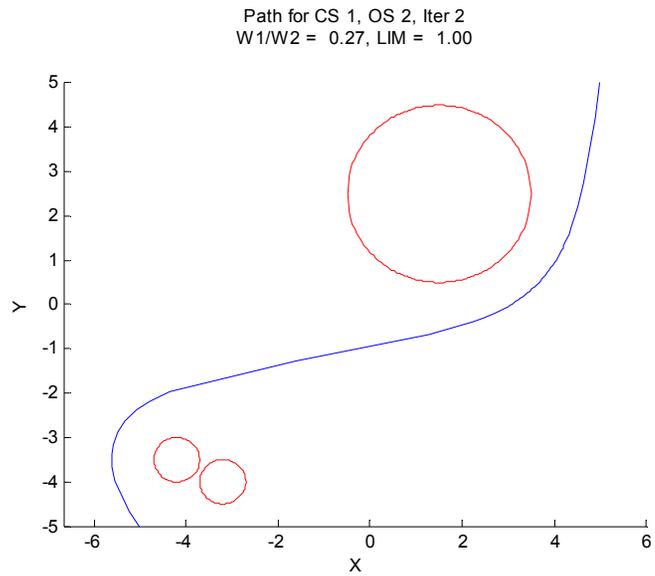


Figure 17: 2DOF path for $\Omega^2 = [0.214, 1.0, 1.0, 1.0]$

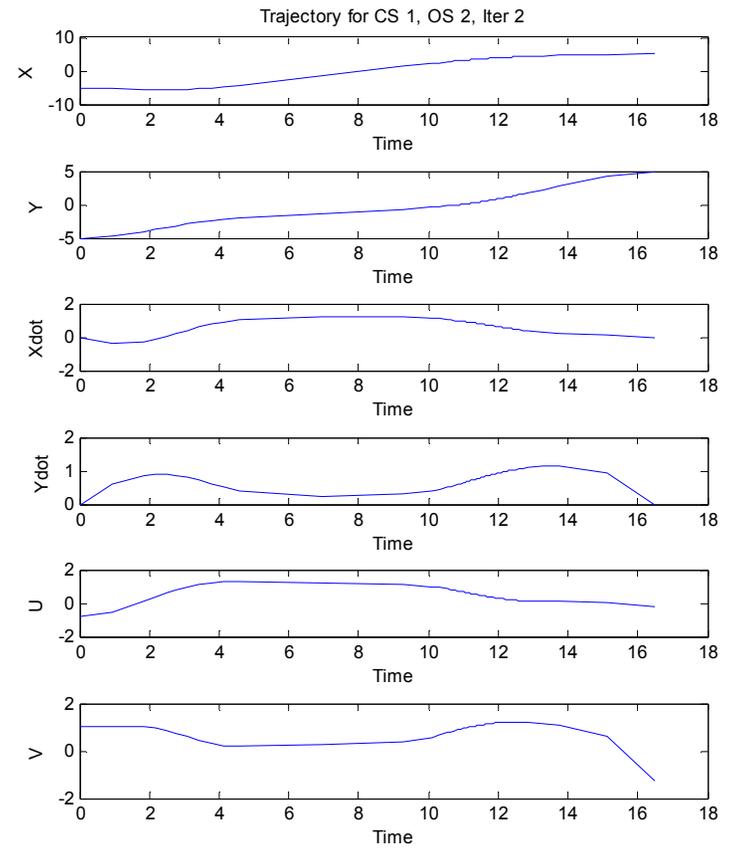


Figure 18: 2DOF trajectory for $\Omega^2 = [0.214, 1.0, 1.0, 1.0]$

Figure 19 and Figure 20 show the results of using \mathbf{Q}^3 to generate the trajectory. The path that had been found has the same shape as in Iteration 1, but the time to traverse it had decreased even more. Just as in Iteration 1, this was too fast for \mathbf{H}^0 , so *WADJ* backed off from the current value of W_1/W_2 . Recall Equation 4.5, which relates the desired feature value to W_1/W_2 . There is a constant coefficient, c_1 , in that equation, which can vary strongly according to field size. Rather than try to maintain a rigid table of c_1 values, we compute it online during every *WADJ* from the current W_1/W_2 value and the current feature value – *max-speed*, in this case. This means that, when we applied the same rule that we did after Iteration 1, we do not get the same new W_1/W_2 ratio that we obtained previously, because c_1 had changed. *max-speed* for this iteration was 6.79 m/s, since 2.50 m/s too fast. Using this to adjust the weight ratio to a value that better supported slow speeds, we obtained $\mathbf{Q}^4 = [0.050, 1.00, 1.00, 1.00]$ and ran the trajectory planner again.

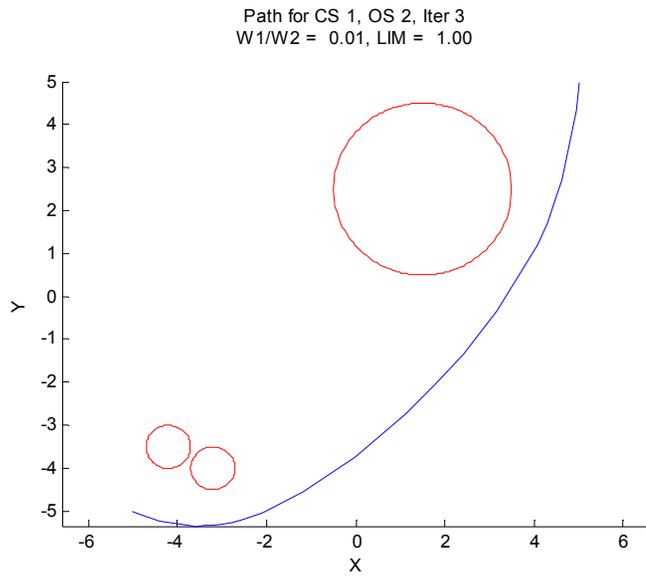


Figure 19: 2DOF path for $\Omega^3 = [0.015, 1.0, 1.0, 1.0]$

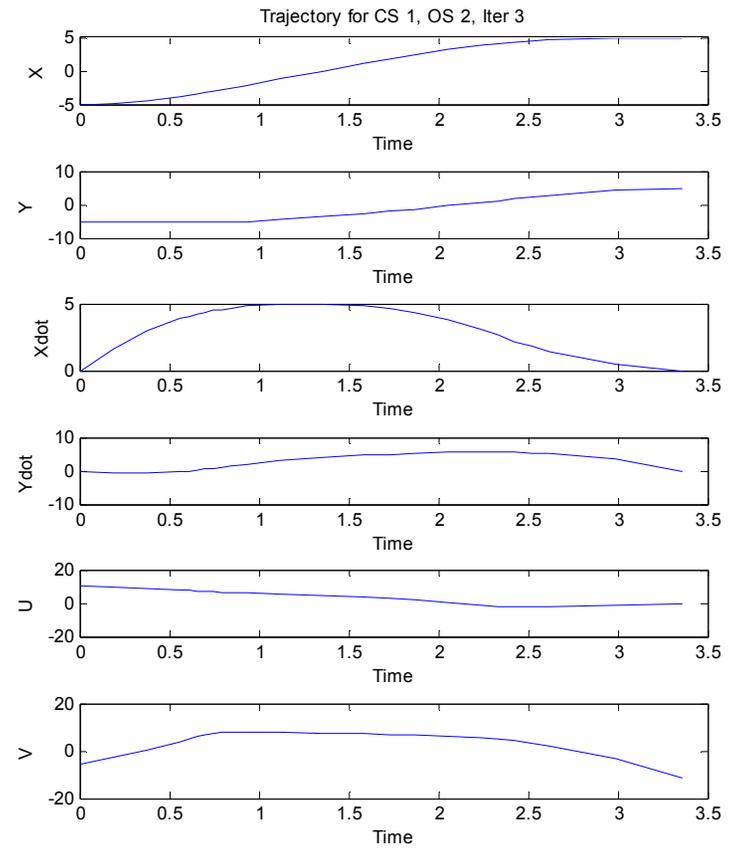


Figure 20: 2DOF trajectory for $\Omega^3 = [0.015, 1.0, 1.0, 1.0]$

Despite a very low W_1/W_2 ratio (time is 20 times more important to minimize than fuel) we obtained a trajectory much like that in Iteration 2 (where $W_1/W_2 = 0.274$). The path in this case (Figure 21) was shorter and more direct than in Iteration 2, but the time to traverse it was comparable (Figure 22). This was a curious result; we would expect from the *WADJ* heuristics a trajectory generated with such a low W_1/W_2 to have higher speeds. However, sometimes the weighted sum approach that we use for our cost functional can, for different weights, return the same result [41]. Another possibility is that this is a local minima, if the search over the time domain of trajectories missed the low-time valley containing the expected solution. Our lowest allowable time for a trajectory was 1 second. Since our previous fast trajectories had final times on the order of 3 – 5 seconds, it is possible that the granularity of the final time matrix was too coarse thus missed this solution.

The *max-speed* in this case was only 1.11 m/s, meeting \mathbf{H}^0 (as *WADJ* was attempting). As before, this failed the \mathbf{S}^0 by fair margins (2.89 m/s too slow to make the low end of “high *max-speed*”). The W_1/W_2 weight ratio was again reduced, this time to the very small value of 0.002. *EVAL* checks for loops when weights are “equivalent,” that is, within some preset threshold of each other. In our tests, this threshold was set to 0.01, so this new ratio was considered to cover the region between 0 – 0.01.

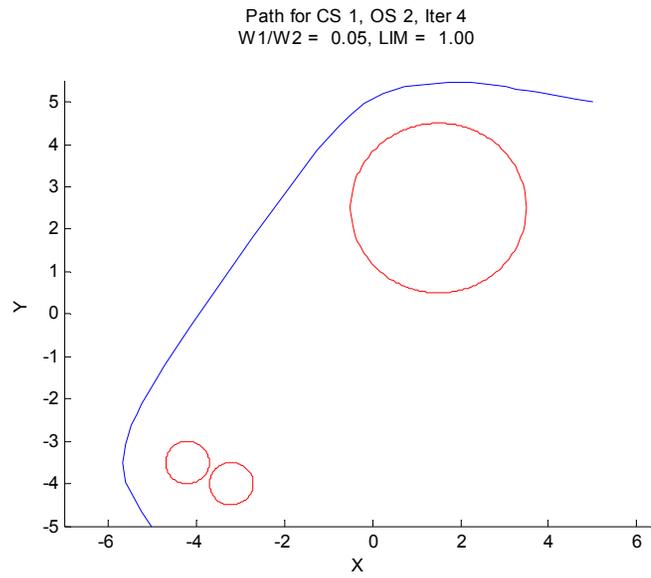


Figure 21: 2DOF path for $\Omega^d = [0.050, 1.0, 1.0, 1.0]$

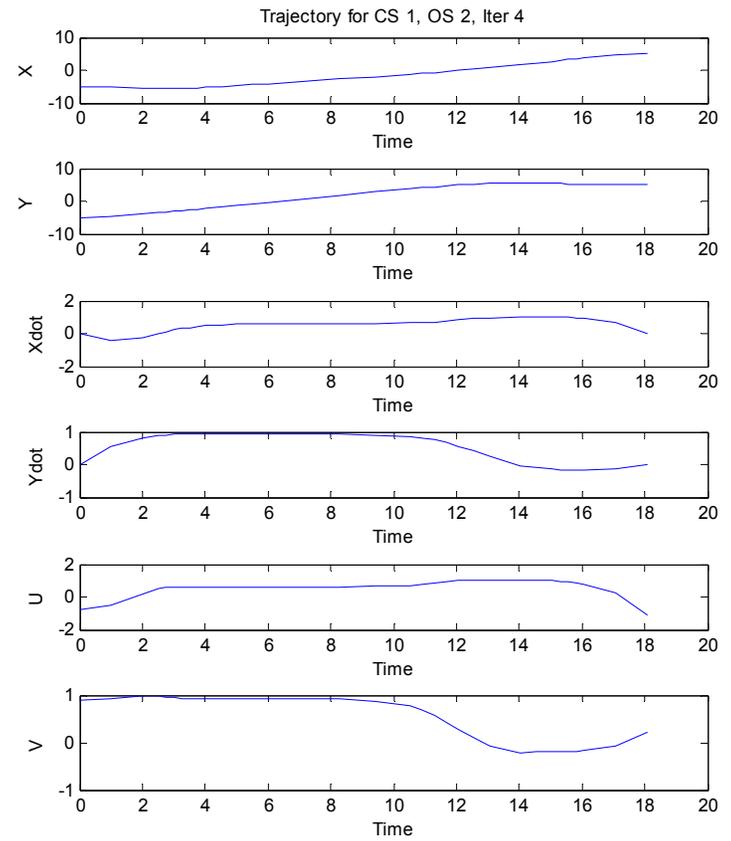


Figure 22: 2DOF trajectory for $\Omega^d = [0.050, 1.0, 1.0, 1.0]$

Iteration 5 returned unusual results. Again despite the very low W_1/W_2 value, we were near the same final time as in Iteration 4. The path shape (Figure 23) was back to something similar to that found for Iteration 1, but it traversed more widely around the obstacles. At this point, the energy term had been so heavily discounted that the obstacle penalty term is exerting a stronger influence on the solution. Figure 24 shows the trajectory information, which was not at all as “quickly” as we might have liked. \mathbf{H}^0 was still met ($max-speed = 1.70$ m/s) for this case, but that was too low for \mathbf{S}^0 . Perhaps we were in the same local minima? *WADJ* tried again for new weights and returned a still-smaller W_1/W_2 value, 0.0002 – but that is, to our level of precision, the same as 0.002. The weights had made a loop, and the *best_trajectory* was returned. Trajectory 1 and 3 failed \mathbf{H}^0 and so are clearly not acceptable. Trajectory 2 failed $max-speed$ by 2.74 m/s and $avg-speed$ by 1.60 m/s; Trajectory 4 failed them by 2.89 and 1.73 m/s, respectively; Trajectory 5 was ultimately returned as the *best_trajectory* even though it failed $max-speed$ by 2.31 m/s and $avg-speed$ by 1.26 m/s.

What was the problem (beyond competing hard and soft constraints)? The initial guess got us quite close to a good solution, and the adjustment to meet \mathbf{H}^0 seemed like a reasonably small change in the weights. But the convergence to the wildly different path shape and its very low speeds put us into a *WADJ* cycle around extreme weight values. The trajectory planner’s search over the free final time parameter was at a fixed starting granularity, which may have been unsuitable for this problem may have resulted in repeated convergence to a local minima. Our technique is, as all numerical optimizer are, vulnerable to these problems.

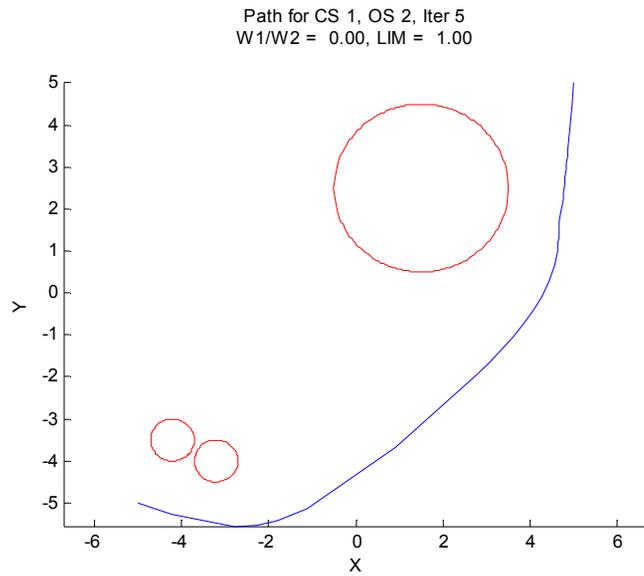


Figure 23: 2DOF path for $\Omega^5 = [0.002, 1.0, 1.0, 1.0]$

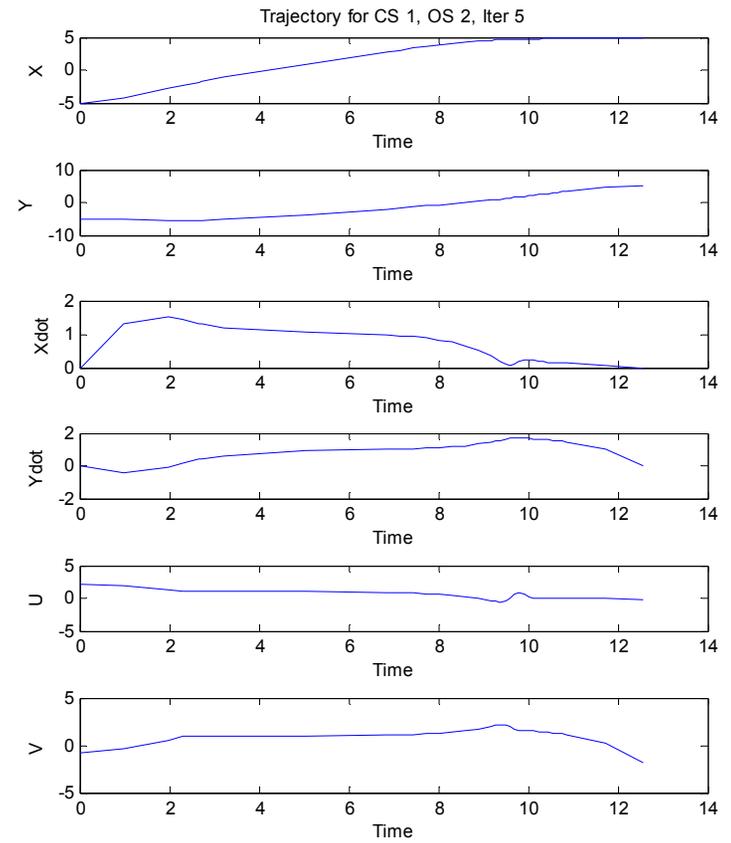


Figure 24: 2DOF trajectory for $\Omega^5 = [0.002, 1.0, 1.0, 1.0]$

4.4.2 Overall 2DOF Results

Figure 25 shows the total number of failures for each of the four cases $Default^l$, $INIT^l$, $Default^n$, and $INIT^n$. $INIT^l$ shows a clear advantage over $Default^l$, with both fewer failures to meet H^0 and S^0 . We are not in this work formally working within an anytime planning framework, but this significant improvement in solution quality for the first iteration would be of benefit should we extend the work in that direction.

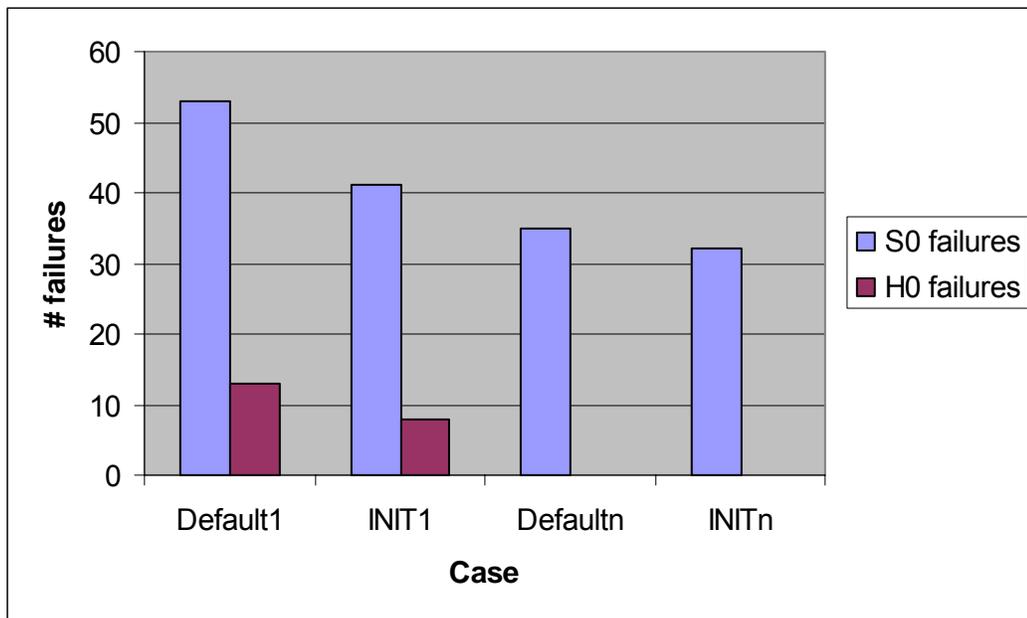


Figure 25: Failures for each 2DOF solution case out of 28 H^0 and 72 S^0

Final results are not nearly as dramatically different as initial results. Our different starting points in these cases did not, after repeated applications of $WADJ$, result in significant differences in final solution. However, $INIT^n$ converged to an acceptable solution in, on average, 5.25 iterations. Three times, only one iteration was required, and the maximum number of iterations was 13. $Default^n$ required on average 6.10 iterations; it never found an acceptable trajectory on the first try, but its maximum number of

iterations was only 9. Figure 26 shows a histogram of the number of iterations each solution required before returning. Overall, *INITⁿ* has more returns with fewer iterations than *Defaultⁿ*.

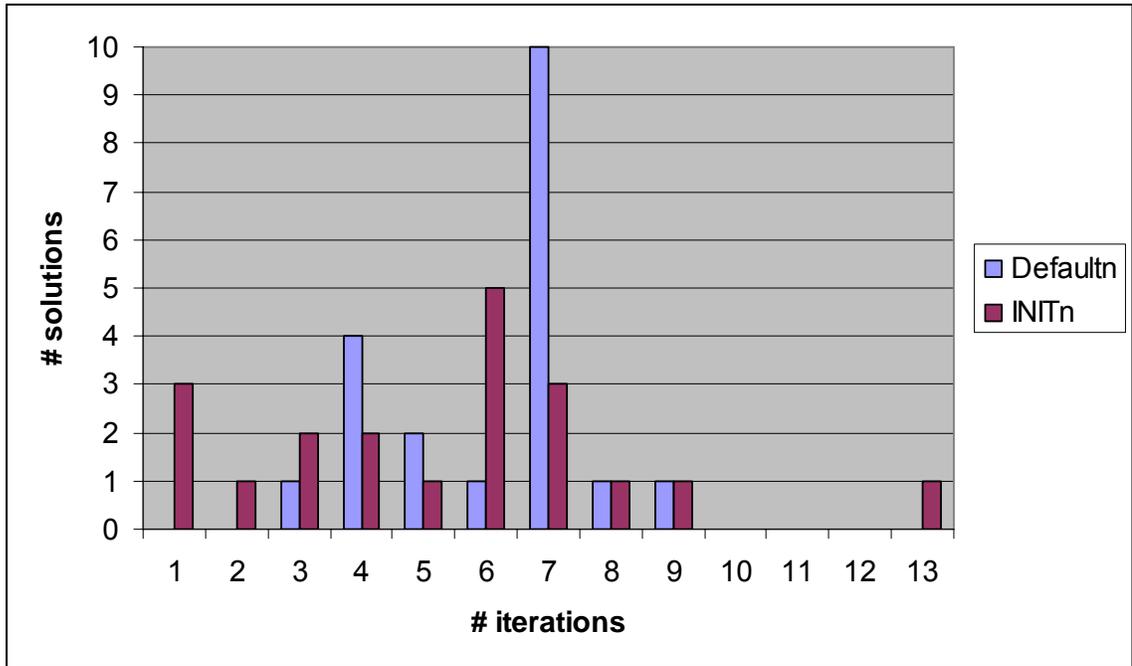


Figure 26: 2DOF iterations through architecture

The effects of obstacle arrangement did not have a strong affect on the number of S^0 failures, as show in Figure 27 below. Except for *INITⁿ* in Obstacle Set 4, all final solutions had between 44% and 56% S_0 failure rates when grouped by obstacle field. We are pleased to see that solution quality is not greatly affected by obstacles.

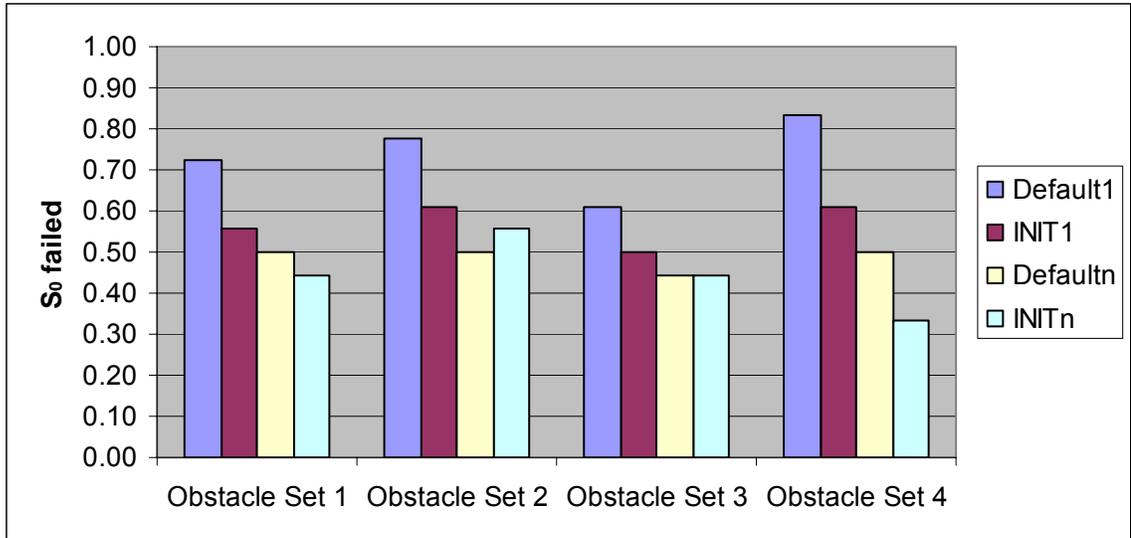


Figure 27: 2DOF S^0 failure rates by obstacle set

Failure rates in S^0 were impacted by the constraint set, as might be expected.

Figure 28 gives the failure rates. Constraint Set 1 and 4 had the overall highest rates. In Constraint Set 1, a hard limit required a *max-speed* of less than 4.2 m/s, while a fuzzy user preference for “very quickly” was also expressed. “Very quickly” was expanded to “high *max-speed*” and “high *avg-speed*,” which defuzzified into $4.0 \text{ m/s} \leq \text{max-speed} \leq 8.0 \text{ m/s}$ and $2.67 \text{ m/s} \leq \text{avg-speed} \leq 5.33 \text{ m/s}$. So we were forcing the system response into the very low end of “high *max-speed*” to meet the H^0 . In many cases, it undershot *max-speed* (and sometimes *avg-speed* as well) in satisfying H^0 and subsequent applications of the *WADJ* heuristics were, in the presence of obstacles, not precise enough to hit the 0.2 m/s window that would satisfy both *max-speed* constraints exactly.

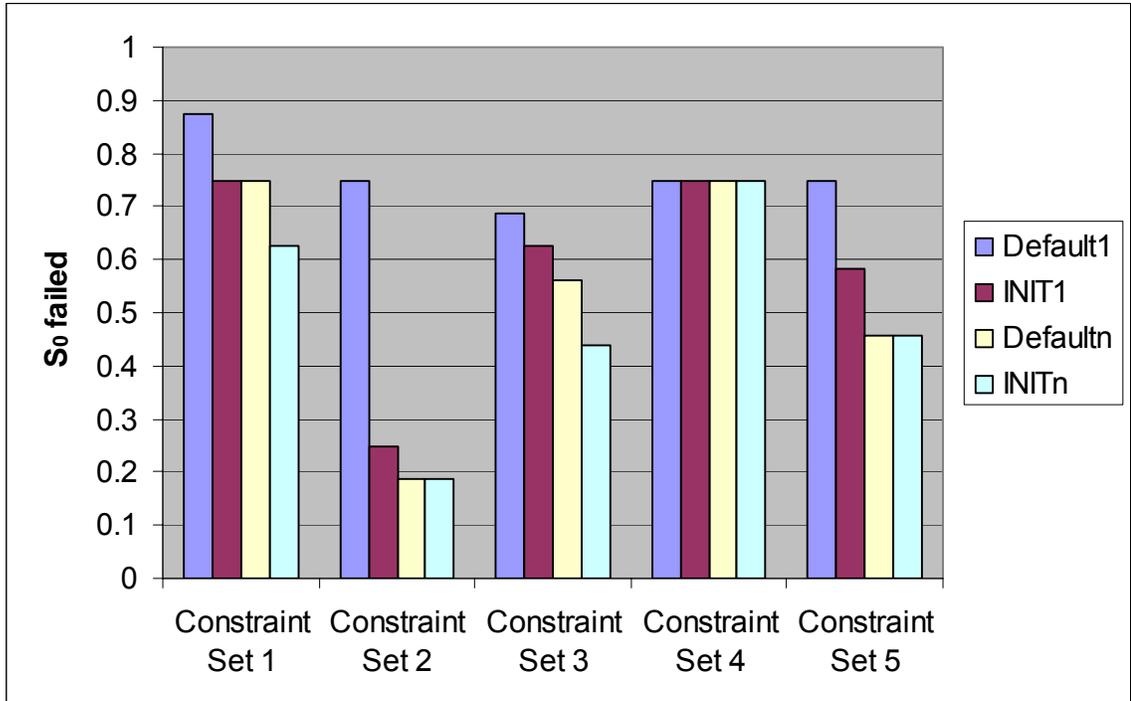


Figure 28: 2DOF S^0 failure rates by constraint set

The Constraint Set 4 failure rates may reflect a weakness in the *WADJ* rule generation method. Constraint Set 4 included two H_0 upper constraints on *max-acc* and *max-speed*, and then S^0 numeric range constraints on *energy* and *avg-speed*. Except for Obstacle Set 4 (which seems to be the easier obstacle set in Figure 27), the *energy* range was uniformly failed. Maneuvering around obstacles may take more energy than predicted by the fuzzy rules gotten from the *WADJ* curve data generated in empty space or in a field with only one obstacle.

Constraint Set 2 was the most successful. It combined a hard upper limit on *max-acc* and a hard lower limit on *min-sep* ($min-sep \geq 1.7$ m) with the fuzzy constraint “safely,” which entailed “high *min-sep*,” and “low” *max-speed*, *avg-speed*, and *max-acc*. Only the “high *min-sep*” was failed in *Defaultⁿ* and *INITⁿ*; it defuzzifies to $3.0 \text{ m} \leq min-$

$sep \leq 5.0$ m. (Note that this is the soft limit on $min-sep$; the hard limit of $min-sep \geq 1.7$ m was met in all final cases.) $min-sep$ that high may have forced paths that detoured very widely around the obstacles and the extra time required to traverse them may have made the trajectories too costly. While the time-based parameters were certainly discounted in these trajectories, they are not entirely ignored; “low” speed does include a fuzzy lower bound. There is a category of speed variables slower than “low,” “very low,” and the *EVAL* module works on the assumption that the user does not want something lower than low – slow, but not *that* slow.

Out of all 40 test cases run to completion, 5 were able to meet all \mathbf{H}^0 and \mathbf{S}^0 . Both the *Default*ⁿ and *INIT*ⁿ solutions for Constraint Set 2/Obstacle Set 3 and Constraint Set 4/Obstacle Set 4 were successful. *INIT*ⁿ also met all limits on Constraint Set 1/Obstacle Set 4. It is interesting that even though Constraint Sets 1 and 4 had the highest \mathbf{S}^0 failure rates overall, they also contained examples of fully successful trajectories.

In addition to the number of failures and successes, we were interested in the overall solution quality. Since \mathbf{H}^0 and \mathbf{S}^0 represent two different things, they were treated separately. Our assumption is that \mathbf{H}^0 represents boundaries on state regions into which the vehicle may not go. They do not describe any preference by the user to be at or near the limit; in fact, we assume the opposite, that being well away from the hard limits is preferable. To this end, we define the hard margin of success to be:

$$margin_{\mathbf{H},j}^i = \begin{cases} \text{abs}(\mathbf{F}_j^i - \mathbf{H}_j^0) / \mathbf{H}_j^0, & \mathbf{F}_j^i \circ \mathbf{H}_j^0 \\ \text{undefined}, & \mathbf{F}_j^i \neg \circ \mathbf{H}_j^0 \end{cases} \quad (4.9)$$

That is, if the j th element of the i th feature vector \mathbf{F}^i in the set of feature vectors \mathbf{F} is constrained by a j th element in \mathbf{H}^0 and, further, meets that constraint, $margin_{\mathbf{H},j}^i$ has a

value. The larger $margin_{H,j}^i$ (which, dropping the indices, we refer to now as $margin_H$) is, the farther from the hard limit \mathbf{H}^0 the relevant trajectory feature is. Large values of $margin_H$ are desirable. (We do not define a similar margin of failure because there were no \mathbf{H}^0 failures in the data sets run to completion.) Figure 29 shows the histogram for values of $margin_H$ over this data set.

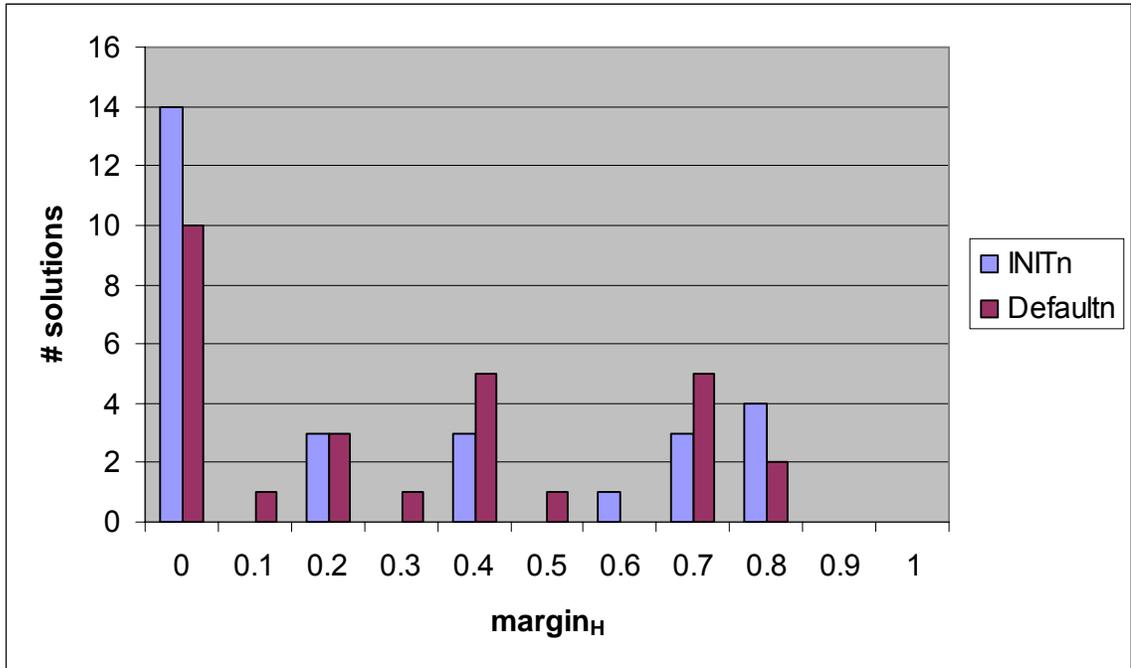


Figure 29: Margin of success for hard limits in 2DOF case

This data suggests the initial choice of weighting parameters has little effect on the final solution quality. Both $INIT^n$ and $Default^n$ achieved 8 instances of $margin_H \geq 0.5$, meaning that the solution feature was less than 50% of the hard limit. $INIT^n$ had more instances of meeting the \mathbf{H}^0 by less than 10% than $Default^n$.

Soft limits \mathbf{S}^0 , on the other hand, represent those areas of the state space where the user *prefers* that the vehicle operate. Those that arise from verbs or adverbs are broken down into upper and lower limits on various state features, defining a range in which we

prefer the vehicle to operate. For these, we assume that the user would prefer to be toward the center of the range; that, as in fuzzy logic, the “ideal” expression of the desired behavior is not near the limits of permissibility, but toward the center. We define the margin of success for \mathbf{S}^0 as:

$$margin_{s,j}^i = \begin{cases} \text{abs}(\mathbf{F}_j^i - \text{midpoint}(\mathbf{S}_j^0)) / \text{half-range}(\mathbf{S}_j^0), & \mathbf{F}_j^i \circ \mathbf{S}_j^0 \\ \text{undefined}, & \mathbf{F}_j^i \neg \circ \mathbf{S}_j^0 \end{cases} \quad (4.10)$$

where F_j^i is again the j th element of the i th feature vector computed by *FEXT* from the trajectory, $\text{midpoint}(\mathbf{S}_j^0)$ returns the center of the soft limit range that corresponds to the feature under consideration and $\text{half-range}(\mathbf{S}_j^0)$ returns the distance from the midpoint to either limit. When an individual $margin_s = 0$, the feature perfectly matches the desired feature value. When $margin_s = 1$, the feature is exactly on one of the range limits. So in these cases, smaller margin magnitudes are desired. Negative values indicate a feature that was less than $\text{midpoint}(\mathbf{S}_j^0)$, while positive values indicate the feature exceeded $\text{midpoint}(\mathbf{S}_j^0)$. (Note that soft upper or lower limits can also be treated by the system; however, we had none in our test suite. Their margins would be computed in the same fashion as $margin_H$). Figure 30 shows the histogram for soft limit success margins.

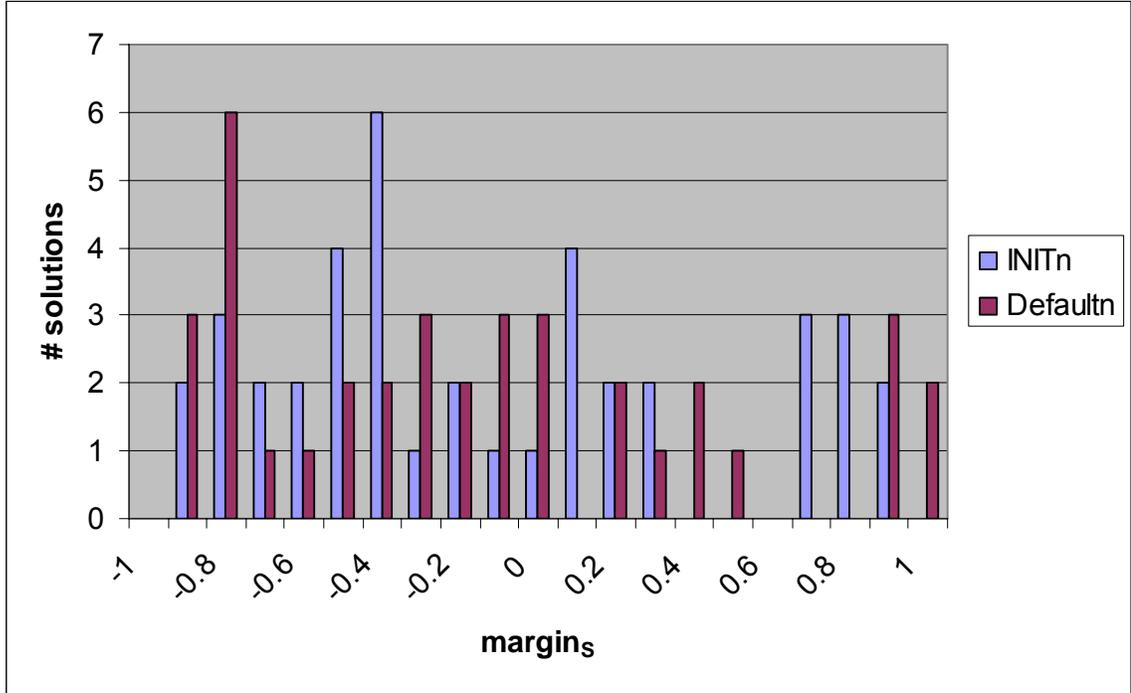


Figure 30: Margin of success for soft limits in 2DOF case

There are no strong trends in this data for either the $INIT^n$ or the $Default^n$ case. S^0 are met without necessarily being driven entirely to the center of the acceptable range. Given our satisficing approach, this is not surprising. Neither $INIT^n$ nor $Default^n$ noticeably outperformed the other in finding more solutions closer to midpoint. This shows the robustness of $WADJ$, which (eventually) achieves similar results regardless of starting point.

Unlike the hard limits case, there were enough S^0 which went unmet to warrant a comparison of margins of failure for soft constraints. We define the margin of failure to be:

$$margin_{fail,j}^i = \begin{cases} (F_j^i - limit(S_j^0)) / half-range(S_j^0), & F_j^i \neg \circ S_j^0 \\ \text{undefined}, & F_j^i \circ S_j^0 \end{cases} \quad (4.11)$$

where $limit(S^0_j)$ is the upper or lower limit of the j th element in S^0 . The sign of the margin tells us how it failed: negative $margin_{fail}$ indicate that a lower limit was failed, whereas positive $margin_{fail}$ means that an upper limit was exceeded. In either case, smaller magnitude values are better, since it means that the limits were failed by a smaller amount. $half-range(S^0)$ is used as the scaling factor rather than the numeric value of the limits or midpoint on the assumption that a narrower range is more sensitive to errors of a given size than a larger range. That is, if the range is 100 units wide, an error of one unit is of less import than if the range were only 2 units wide, regardless of the numeric values of the ranges' endpoints. Figure 31 shows the histogram for margins of failure on S^0 .

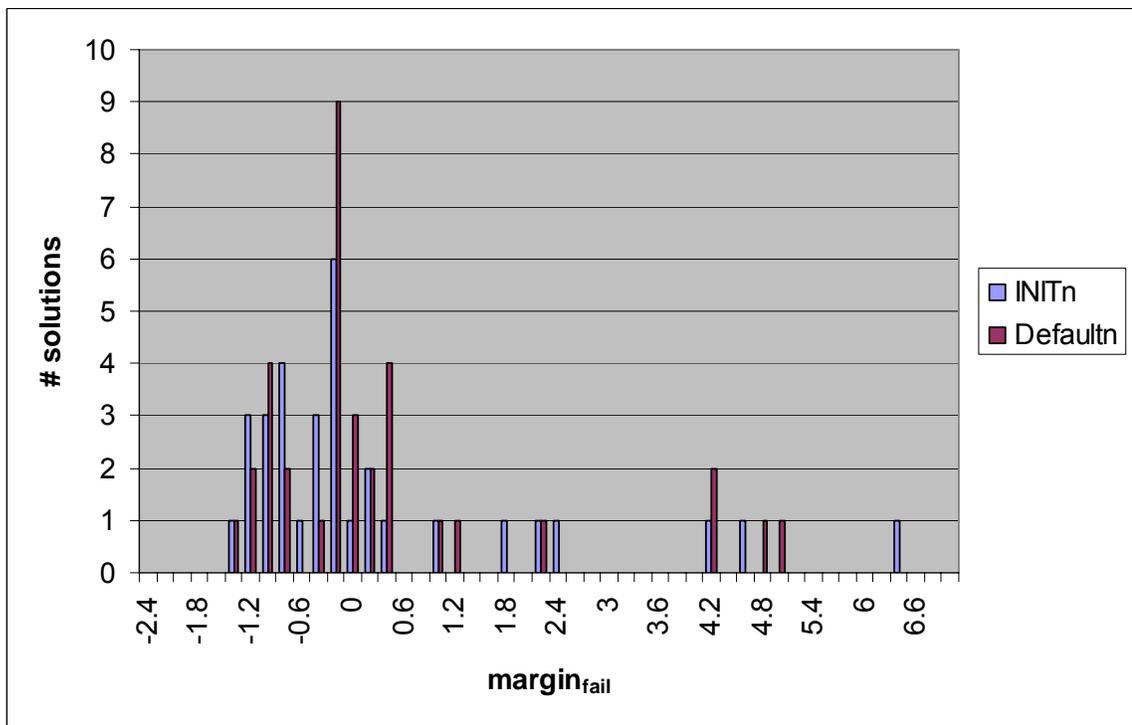


Figure 31: Margins of failure on soft limits for 2DOF case

The results are again very similar for $INIT^n$ and $Default^n$. $Default^n$ has four more solutions near the center, indicating smaller errors; however, it also has two more

solutions on the tails of the histogram. The standard deviation in the Figure 31 is 2.41 for *Defaultⁿ* and 2.15 for *INITⁿ*. There does not seem to be a clear advantage to either technique.

On the other hand, if *INIT* is used to support an anytime planning algorithm, there is a definite benefit to using it. In addition to meeting more of the \mathbf{L}^0 , *INIT* meets them with better margins. Figure 32 shows the histogram of $margin_{fail}$ for the returned trajectories after one iteration of *TPLAN*. Results are similar for failures to meet \mathbf{H}^0 .

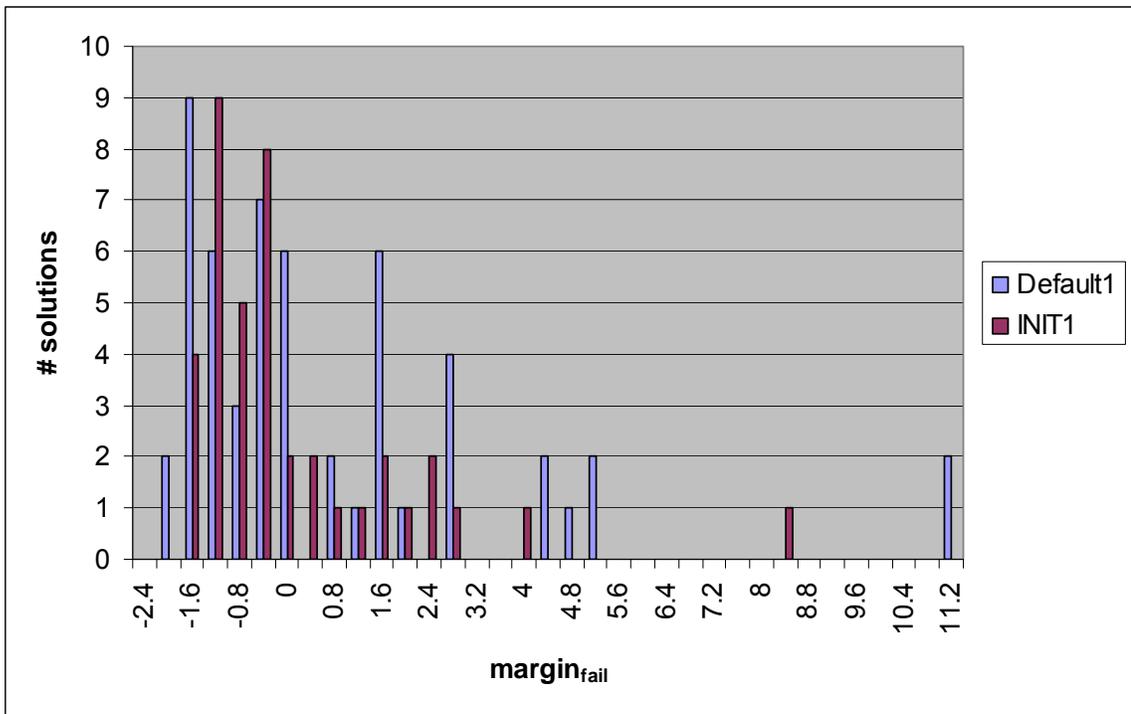


Figure 32: Margins of failure for soft limits after one iteration in 2DOF case

INIT^l shows four more failures with the very smallest $margin_{fail}$. While *Default^l* has more failures in the general central region, this may be because *Default^l* simply had more failures overall. *Default^l* also has four more failures in the high-error tail region. *INIT^l* certainly has superior values for $margin_s$ when compared to *Default^l*, as shown in Figure

33. *INIT*^d not only has more successes, it has many more successes in the low-valued region nearest the goal *midpoint*.

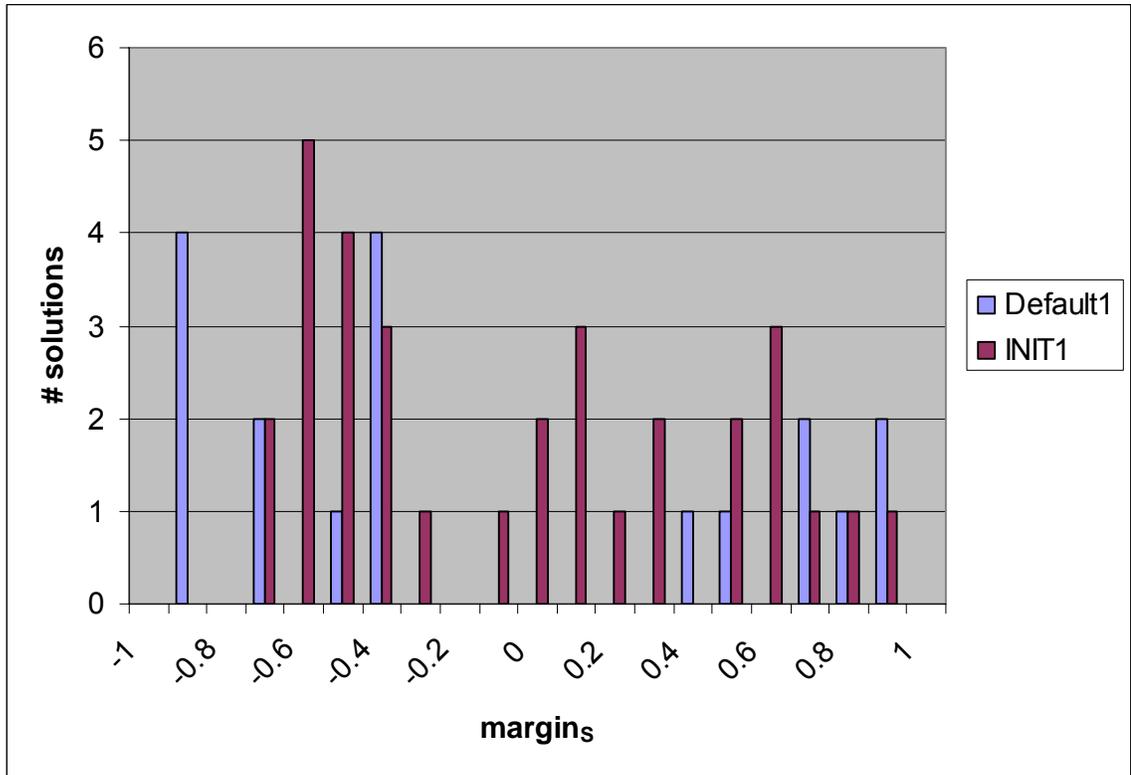


Figure 33: Margins of success for soft limits after one iteration in 2DOF case

4.5 Conclusions

We are overall satisfied with the performance of our architecture and the *INIT* and *WADJ* routines in this 2DOF case. *INIT*, while not greatly impacting final solution quality, did shorten the time needed to arrive at an acceptable solution and did improve the quality of early solutions – a boon if anytime planning is to be used. The *WADJ* heuristics we developed reliably moved the solution into areas that satisfied hard constraints \mathbf{H}^0 100% of the time, regardless of starting condition. The sometimes-competing \mathbf{S}^0 were not always satisfied; however, definite improvement over the initial

solutions achieved using either *INIT* or default weights was seen in the solution as *WADJ* refined the trajectory.

Some preliminary work has been done into extending this 2DOF work into the multi-agent formation maintenance field. A term can be added to the cost functional which penalizes the variation from the desired formation and weighted along with the rest. Proof-of-concept work was done in the linear 2DOF case, with two vehicles maneuvering in the presence of obstacles (Figure 34). High relative weights placed on formation maintenance resulted in both vehicles moving around the obstacle. High relative weights for time resulted in trajectories where the vehicles parted ways to move around the obstacle. Energy-efficient trajectories had a minimum disturbance from straight-line trajectories – the vehicles parted ways to circumvent the obstacle, but with smaller margins than they had in the “quickly” case. However, no rigorous collection of multi-agent *WADJ* data has been performed to-date, although such analysis could be applied to formations maneuvering in the presence of obstacles in future work.

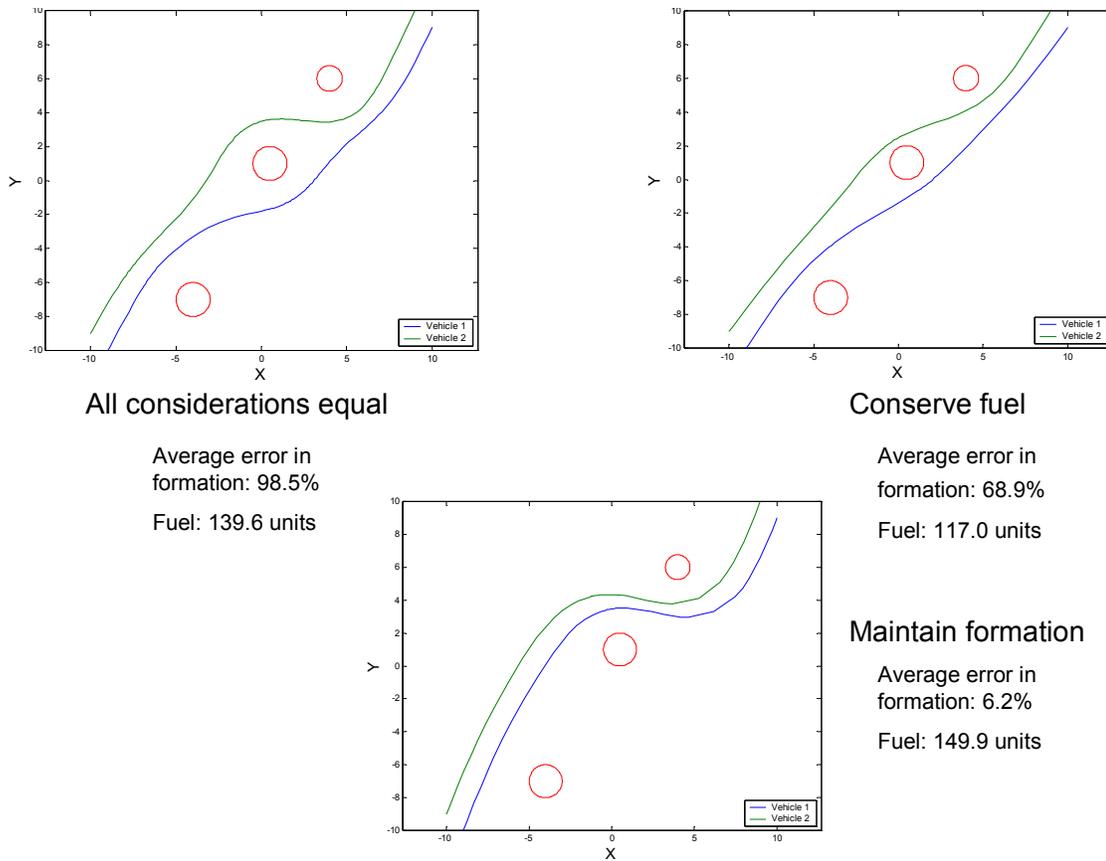


Figure 34: Preliminary work in multi-vehicle formation management

It now remains to show how well – or even if – these techniques translate to a more challenging six degree-of-freedom domain. Chapter 5 will consider the a simplified version of the space shuttle/ISS docking problem first solved with hand-tuned weights in [51]. We will attempt to find what different solutions can be found when other constraints are placed on the problem as well as further evaluating the performance of our system for a 6-DOF domain with nonlinear dynamics.

5 Six Degree of Freedom Deep Space Satellite

The experiments described in Chapter 4 showed our architecture could be useful. But the 2DOF point rover problem was highly simplified and linear. Would the techniques developed, the *WADJ* rules in particular, be useful in a nonlinear space domain as well?

We began research into reproducing the International Space Station/Space Shuttle docking simulation presented in [51]. Unfortunately, the implementation from [51] was sufficiently complicated that the optimal controls-based trajectory planner could only reliably identify solutions via substantial tuning of additional parameters beyond cost function weights for each iteration. Changing the cost functional weights changed the character of the problem sufficiently that the gain schedules used in [51] to incrementally refine thruster approximations, error tolerances, and obstacle penalty function values no longer guided the solver to convergence in many cases. Re-tuning those gain schedules for every new weighted cost functional was simply beyond the scope of the current research. Unable to generate trajectories for analysis and adjustment, we were forced to abandon this problem for a somewhat simpler three-dimensional, six degree of freedom (space) domain example with dynamic properties that were nonlinear but less complex than orbital motion.

We chose to examine a six degree of freedom satellite operating in deep space, away from gravitational fields but potentially in proximity to obstacles (e.g., asteroids). While this does result in very simple and linear translational dynamics, the rotational dynamics are nonlinear and make the problem more interesting than the simple 2DOF

rover. We assume impulse thrusters for translation and reaction wheels for torque generation, following the modeling described in [51].

5.1 System Dynamics

The general state space form of a rigid body in deep space is given by:

$$\mathbf{x}'(t) = \begin{bmatrix} \mathbf{v}' \\ \mathbf{p}' \\ \boldsymbol{\omega}' \\ \boldsymbol{\sigma}' \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{v} \\ -\mathbf{H}^{-1}\mathbf{S}(\mathbf{H}\boldsymbol{\omega})\boldsymbol{\omega} \\ G_{\sigma}(\boldsymbol{\sigma})\boldsymbol{\omega} \end{bmatrix} + \begin{bmatrix} \mathbf{u}(t)/m \\ 0 \\ \mathbf{H}^{-1}\boldsymbol{\tau}(t) \\ 0 \end{bmatrix} \quad (5.1)$$

where:

\mathbf{p} is the location (position) vector of the body in the inertial reference frame

\mathbf{v} is its velocity vector

$\boldsymbol{\omega}$ is the rotational velocity vector expressed in the body frame

$\boldsymbol{\sigma}$ is a representation of the body's attitude (a modified Rodrigues vector [55])

\mathbf{H} is a matrix of moments of inertia

\mathbf{S} is the matrix representation of the cross product

G_{σ} is an expression which, when multiplied by $\boldsymbol{\omega}$, gives the rate of change in $\boldsymbol{\sigma}$

$$\dot{\boldsymbol{\sigma}} = G_{\sigma}(\boldsymbol{\sigma})\boldsymbol{\omega} = \frac{1}{2} \left(\mathbf{I} - \mathbf{S}(\boldsymbol{\sigma}) + \boldsymbol{\sigma}\boldsymbol{\sigma}^T - \frac{1 + \boldsymbol{\sigma}^T\boldsymbol{\sigma}}{2} \mathbf{I} \right) \boldsymbol{\omega} \quad (5.2)$$

\mathbf{R} is a rotation matrix that converts body coordinates to inertial coordinates

\mathbf{u} is the vector of translational control input (force) expressed in the body frame

m is the mass of the rigid body

$\boldsymbol{\tau}$ is the vector of rotational control input (torque)

As in [51], the mass used was 1 kg and $I_{xx} = I_{yy} = I_{zz} = 1 \text{ N-m/s}^2$. Maximum thruster output was $\pm 30,000 \text{ N}$ in each axis. Maximum torque output about each axis was smooth until a saturation value of $\pm 1,000 \text{ N-m}$.

5.2 Terms of the Cost Functional

We have the same concerns in the 6DOF case as we do in the 2DOF: we wish to conserve our fuel and power inputs, accomplish our goals in a timely fashion, and not impact obstacles. Our inputs are different in this case; rather than a continuous electrically-powered motor, we have saturating thrusters for 3DOF translation and an electrically-powered reaction wheel for 3DOF attitude control. As a result, the cost functional has the form:

$$J = \int_{t_o}^{t_f} \left(W_1 \|\mathbf{u}(t)\|_1 + \boldsymbol{\tau}(t)^T W_2 \boldsymbol{\tau}(t) + W_3 + W_4 \max_{i \in \{O\}}(o_i(r_i)) \right) dt \quad (5.3)$$

Each of these terms, and their rationale, is explained in detail in [51]. Brief descriptions are given below.

5.2.1 Thruster Fuel

The one-norm of the thruster force results in a minimum-fuel control law. This control law is, however, discontinuous and so violates the assumptions that underlie the numeric solution of the Euler-Lagrange equations. The solution in [51] was to approximate the discontinuous control law, beginning with an approximation with moderate slopes and gradually increasing the slope to nearer a step function.

5.2.2 Electrical Energy

Still following [51], the rotational actuators are assumed to be powered by electricity. This is the case on long-duration missions, and has the benefit of separating translational and rotational control parameters. This form of the cost functional is an energy-minimizing term, a standard cost representation for electrically-powered systems.

5.2.3 Time

Since J is an integral, the cost functional only needs a constant term, W_3 , to minimize time. Over the integral, $W_3 * t_f$ will be minimized.

5.2.4 Clearance to and Speed Near Obstacles

As in Chapter 4, the obstacle penalty function contains a cubic spline term that penalizes nearness to the obstacles. This function also includes a velocity-based component that penalizes speed near the obstacle. Since the cost functional is an integral over time, a penalty based purely on clearance to the obstacle can be minimized by being very close to the obstacle but going very fast, so that the sum over time is less. The clearance penalty is now multiplied by a smoothed one-norm of the velocity (at some epsilon near zero, the one-norm is approximated by a cubic to maintain smoothness properties necessary for convergence). As in the 2DOF case, the obstacle clearance penalty goes to zero at some LIM from the obstacle; also as in the 2DOF case, the penalties for each obstacle in the domain are summed.

5.3 Development of Weight Adjustment Heuristics and Fuzzy Rules

5.3.1 Weight Adjustment Heuristics

For the 6DOF weights, the *TPLAN* code BVP4C2 assumed that the force weight W_1 was normalized to 1, and all other weights were relative to this. As a result, it was more intuitive to work with the W_1 as the denominator in the weight ratios for our 6DOF spacecraft domain. All of the code written to implement these 6DOF heuristics uses torque/force and time/force weights, rather than the inverse as in the 2DOF case. However, for some select charts and examples presented in this chapter, the weight ratios were inverted for easier comparison to the Chapter 4 results. Not all of the *WADJ* heuristic graphs nor the fuzzy rules sets contained in here and in Appendix C reflect this inversion and are labeled W_2/W_1 and W_3/W_1 as they were implemented. Our weight vector Ω included W_1 the force weight, W_2 the torque weight, W_3 the time weight, and *LIM*. (Since the obstacle penalty function weight W_4 is never adjusted, we do not include it in our weight vector representation.)

To develop *WADJ* rules, we followed the general procedure outlined in Chapter 3 again in the 6DOF domain. We did not test different field sizes this time, as we had confidence from the 2DOF results that they would scale well. (This confidence was well-placed; our *WADJ* curves, below, were generated at a scale of 50 m while our test cases were on the order of 10 – 20 m.) We tested pure translation (along one axis and along three), translation plus rotation, and a translation in the presence of obstacles. For the translation in the presence of obstacles, the final state orientation was identical to the initial state orientation; rotation was not required, but it was not forbidden, either. Following the insights gained in the 2DOF domain, we plotted “per second” features

versus the ratio of the weight of the translational inputs (here, thruster force) W_1 to the time weight W_3 . The results for the feature *avg-speed* are shown in Figure 35, below. Once again, there is the power relationship between speed and the force/time weight ratio. We found this to be the case for the other force and time-based quantities as well.

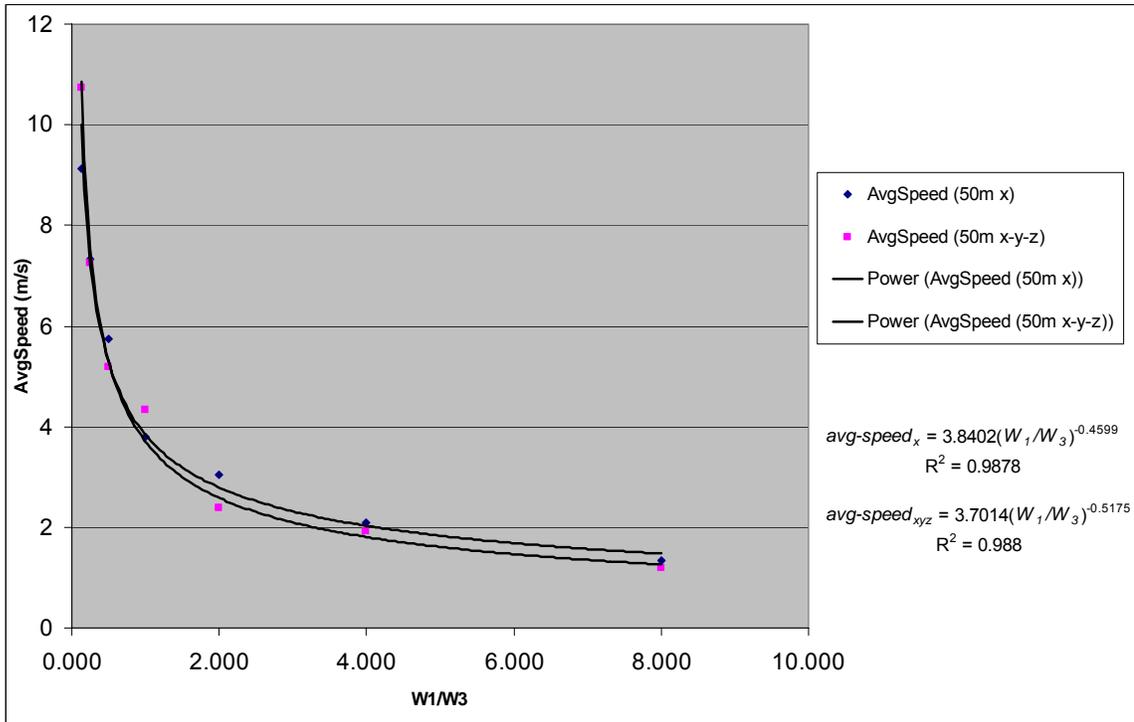


Figure 35: WADJ curve for *avg-speed* in 6DOF domain

The path-based features (e.g., *min-sep*) were once again linear with the influence limit of the obstacle penalty function. Unlike the 2DOF case, the trajectories were much more likely to be plotted through obstacles. To handle this, we added an implicit hard constraint to every trajectory, $min-sep > 0$ m. If the path went inside an obstacle, the trajectory failed and *LIM* was adjusted to move the path out of the obstacle. This solved that problem.

Torque presented us with a challenge. Our *WADJ* test for torque included a translation and a rotation, so that we would see the effects of dynamic coupling.

Following the intuition we had from the translational features, we tried plotting the ratio of the torque weight, W_2 , versus the time weight W_3 . However, since torque and the resulting rotational motions are the source of nonlinearity in the system, initial results indicated no power law for $WADJ$ thus we at first were concerned this heuristic may not be applicable.

Upon further examination, however, we identified a more promising heuristic. Figure 36 shows the torque data grouped by time and force. First, the data were grouped by their torque/force weight ratio (W_2/W_1) but plotted versus the time/force ratio (W_3/W_1) as shown in Figure 36. Each line in Figure 36 represents a fixed W_2/W_1 ratio. Even though that ratio is fixed, the amount of torque applied can be increased or decreased by adjusting the W_3/W_1 ratio. Conversely, if the W_3/W_1 were known and fixed, changing the W_2/W_1 ratio could jump the torque up or down that family of linear curves. Was there a predictable relationship between the slopes of the lines in Figure 36 and W_2/W_1 ?

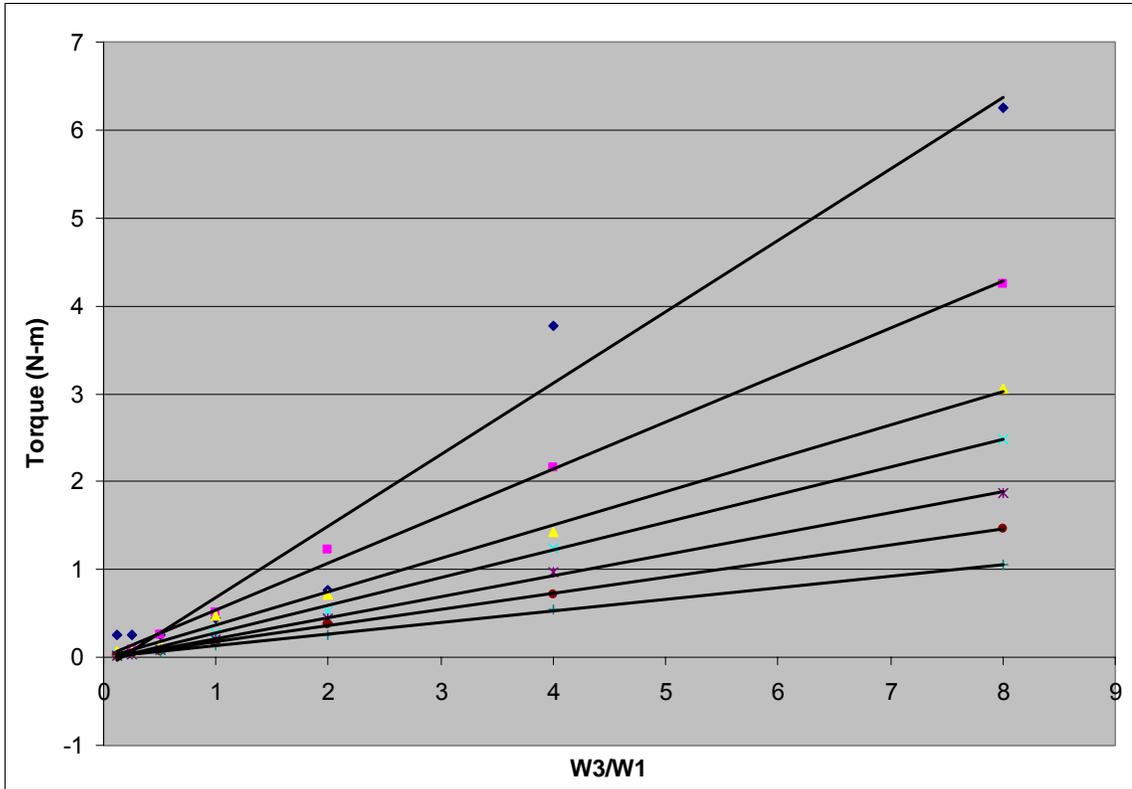


Figure 36: First stage of *WADJ* heuristic for determining torque in 6DOF domain

Figure 37 shows that there was. Our torque heuristic was implemented as follows: First, all non-torque features were checked for limit failures and, if there were failures, the weights were adjusted. Then the torque feature was checked. If it failed, the desired torque value was divided by the current W_3/W_1 value to get the slope of the line we would like to be traverse in Figure 36. Then the power relationship shown in Figure 37 was used to calculate the necessary W_2/W_1 ratio from the desired slope.

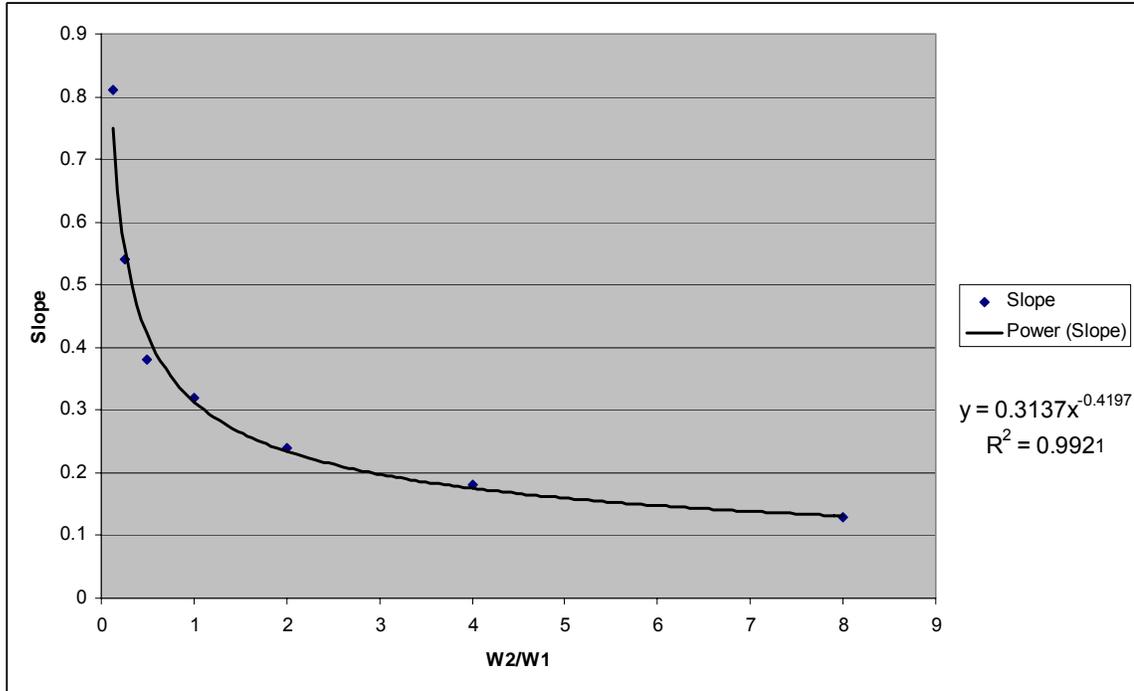


Figure 37: Second stage in torque heuristic in 6DOF

5.3.2 Fuzzy Rules

The fuzzy rules were generated as they had been for the 2DOF case. The WADJ data was reverse engineered so that “very high,” “high,” etc., weight values were correlated back to the trajectory features they elicited. We again found that the range from 2^{-3} , 2^{-2} , ..., 2^2 , 2^3 was sufficient to describe the WADJ relationship. The full set of fuzzy rules for the 6DOF spacecraft domain is again found in Appendix C.

5.4 Results

Six different sets of constraints \mathbf{L}^0 were enforced on four different obstacle fields $\{\mathbf{O}\}$ for a total of twenty-four trials. However, one constraint/obstacle pairing proved to be intractable and BVP4C2 could not converge on a solution. This trial (Constraint Set 4, Obstacle Set 4) is omitted from the following results. There were, overall, 20 hard limits (\mathbf{H}^0) and 60 soft limits (\mathbf{S}^0). The simplest constraint set enforced one soft constraint; the

most extensive had two hard constraints and six soft constraints. Appendix B fully details the test cases.

Each of the twenty-three successful test cases was run from a default weight vector $\mathbf{\Omega}^l_{default} = [1, 1, 1, 1]$ and from a $\mathbf{\Omega}^l$ provided by *INIT*. Again, only the collocation *TPLAN BVP4C2* was used for trajectory generation. The results after one iteration (labeled *Default^l* and *INIT^l*) and after program completion (*Defaultⁿ* and *INITⁿ*) were examined for both starting weight vectors.

Before considering the overall results, we again present particular solutions to two example cases for illustrative purposes. Both are for Constraint Set 1 and Obstacle Set 1. We first show the results obtained when starting from the default weight vector, then show results when the weight vector is initialized via our fuzzy logic rules.

5.4.1 Example Cases

Constraint Set 1 in 6DOF is analogous to Constraint Set 1 in 2DOF. The soft constraint “somewhat quickly” is requested, while a hard constraint $\mathbf{H}^0 = \{max-speed \leq 5.5 \text{ m/s}\}$ is also included to force the solution toward the low end of the range of *max-speed* encompassed by “quickly.” “Quickly” is still defined as high *max-speed* and high *avg-speed*, but the values that define the fuzzy ranges of “high” have of course changed for the new domain. We required that *max-speed* be between 5.0 and 10.6 m/s, and *avg-speed* be between 4.0 and 8.6 m/s.

We look first at the default case, where we started with all weights in $\mathbf{\Omega}^l$ equal and normalized to 1 and the obstacle influence limit $LIM = 1$ as well. The results are shown in Figure 38a-c, below. The force response (shown in body coordinates) was typical for a time-efficient trajectory, accelerating to the midpoint and then decelerating.

max-speed was 2.33 m/s, well under the limit imposed by \mathbf{H}^0 , but the \mathbf{S}^0 limits were failed. *max-speed* was 2.66 m/s too slow for “quickly,” and *avg-speed* was -2.10 m/s too slow. The weight on time was increased to 4.57, up from its prior value of 1.0, so that $\mathbf{\Omega}^2 = [1.00, 1.00, 4.57, 1.00]$, where W_1 is the fuel weight, W_2 is the torque weight, W_3 is the time weight, and W_4 is *LIM*. In this domain, the weights were normalized by W_1 . Increasing W_3 indicated that time was of more value to us, so the next trajectory should take less time and hence be faster.

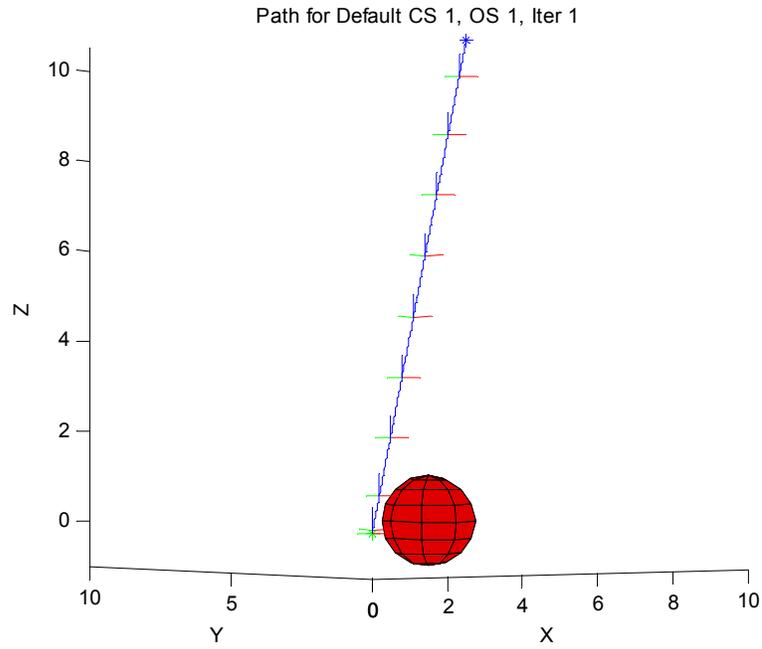
The torques seen in the $\mathbf{\Omega}^j$ are a result of the use of the (approximated) one-norm in the cost functional fuel term. Consider a planar robot able to move in x , y , and θ . To translate at a 45° angle to the x -axis with 1 N of force, it could fire the x and y thrusters each at 1.4142 N, for a total expenditure of 2.8284 N as computed by the one-norm. Or, it could rotate 45° so that its x thruster was aligned with the direction of motion, then thrust with 1 N. As long as the cost of the torque maneuver is less than the force saved, this is more efficient.

We see such fuel-minimizing torques in Constraint Sets 1-3; the numeric solver did not identify cost savings via torque maneuvers in Constraint Sets 4 and 5. To validate the use of torque commands to minimize fuel usage, Table 2 summarizes the one-norms of the forces for Constraint Sets 1-3 represented in the inertial frame, representing initial/default spacecraft thruster alignment, and in the body frame, representing actual thruster alignment when the designated torques are applied. The forces in the body frame are always less than or equal to the inertial forces, demonstrating that by rotating the vehicle, less force needed to be applied. The effects are very small, however, thus it is

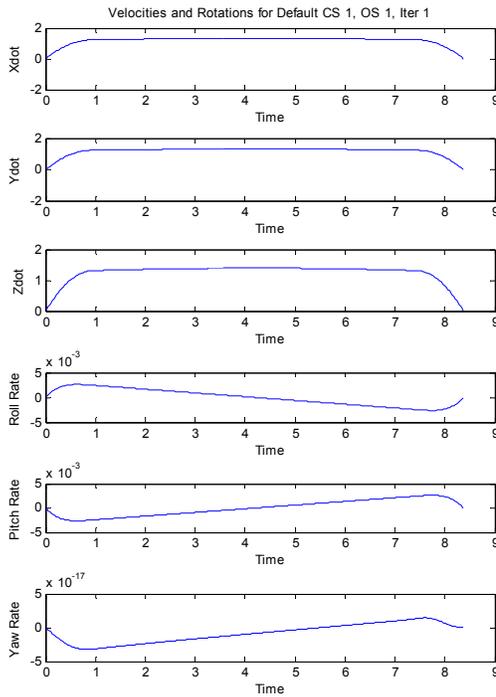
not surprising that some cases did not alter spacecraft attitude (Constraint Sets 4 and 5), especially given a small but non-zero penalty for the energy (torque) term.

Constraint Set	Obstacle Set	One-Norm of Inertial Forces (N)	One-Norm of Body Forces (N)
1	1	17.7316	17.7311
1	2	18.9900	18.9886
1	3	18.5617	18.4592
1	4	13.6732	13.4170
2	1	4.5407	4.5407
2	2	5.3882	5.3877
2	3	5.4126	5.0528
2	4	3.8322	3.8146
3	1	17.7316	17.7311
3	2	20.1985	20.1971
3	3	24.2088	24.0482
3	4	14.3041	14.0320

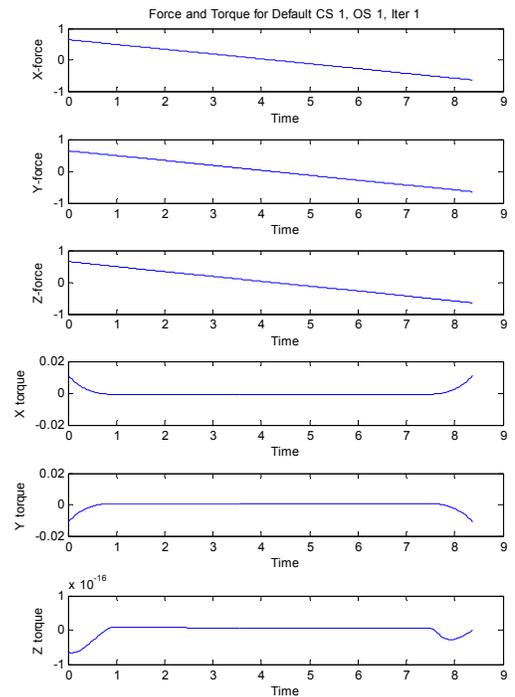
Table 2: One-Norm of Inertial and Body Forces



(a)



(b)

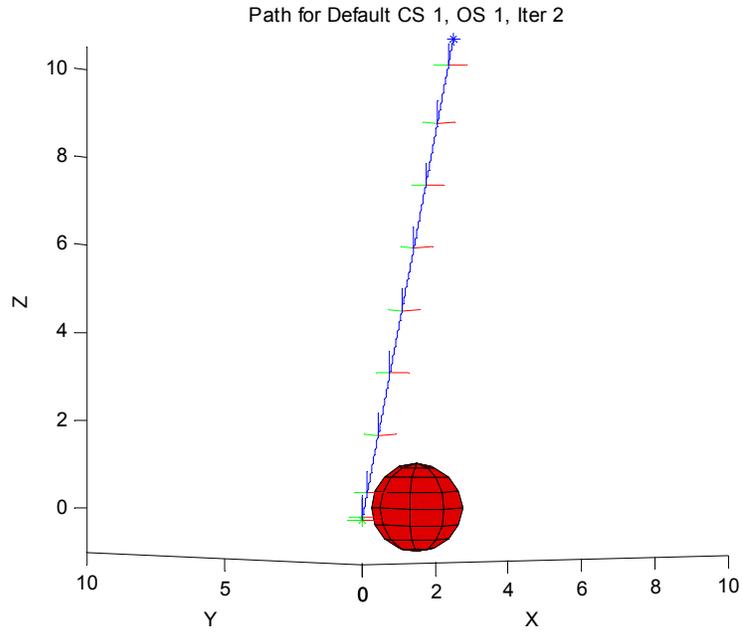


(c)

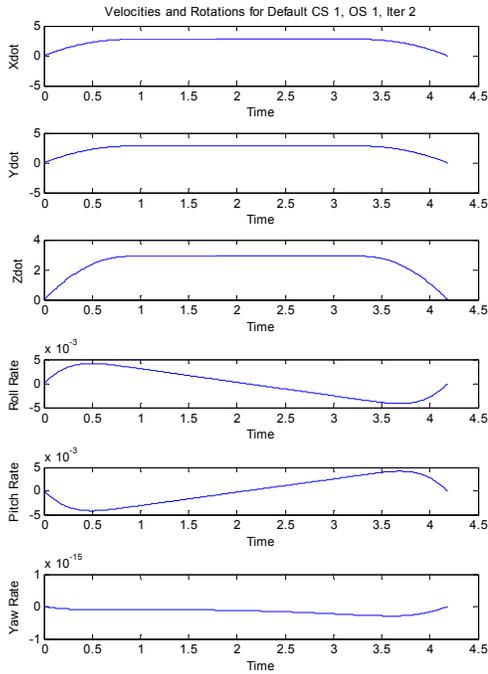
Figure 38: 6DOF a) path, b) rates and c) inputs for $\Omega^I = [1.00, 1.00, 1.00, 1.00]$

Figure 39 shows the results for Ω^2 . The overall completion time was almost halved, with the speeds increasing accordingly. Indeed, the shape of the path, the rates, and the input curves are almost unchanged except in magnitude. *max-speed* was 4.92 m/s for this run, still meeting \mathbf{H}^0 , and still failing the *max-speed* component of \mathbf{S}^0 – but only by 0.08 m/s this time. *avg-speed* was 4.19 m/s, within its \mathbf{S}^0 requirements. We were converging toward a full solution, but not quite there. The time weight was increased a small amount, to 4.72, and another trajectory found.

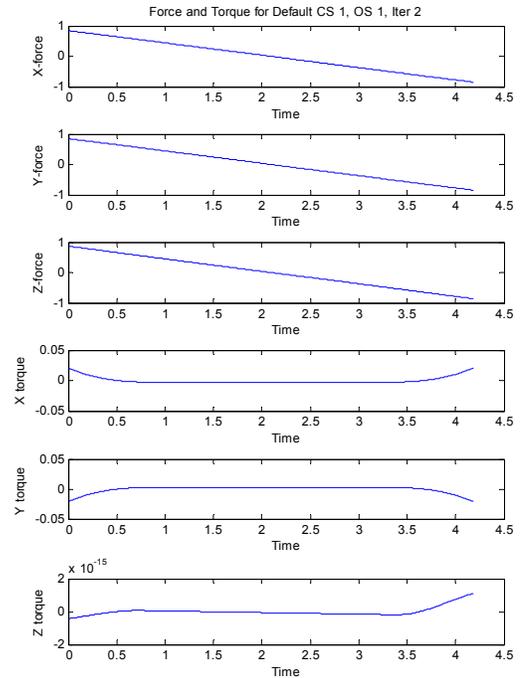
Ω^3 produced the same trajectory as Ω^2 . The returned features were the same; the *max-speed* failed its \mathbf{S}^0 by the same amount. The plots of path, rates and inputs were the same. The change in weight was not sufficient to elicit a different response in the system at our levels of precision. *WADJ* increased the time weight again, applying the 0.08 m/s failure margin to the base time weight of 4.72 from Ω^3 . The result was $\Omega^4 = [1.00, 1.00, 4.86, 1.00]$.



(a)



(b)

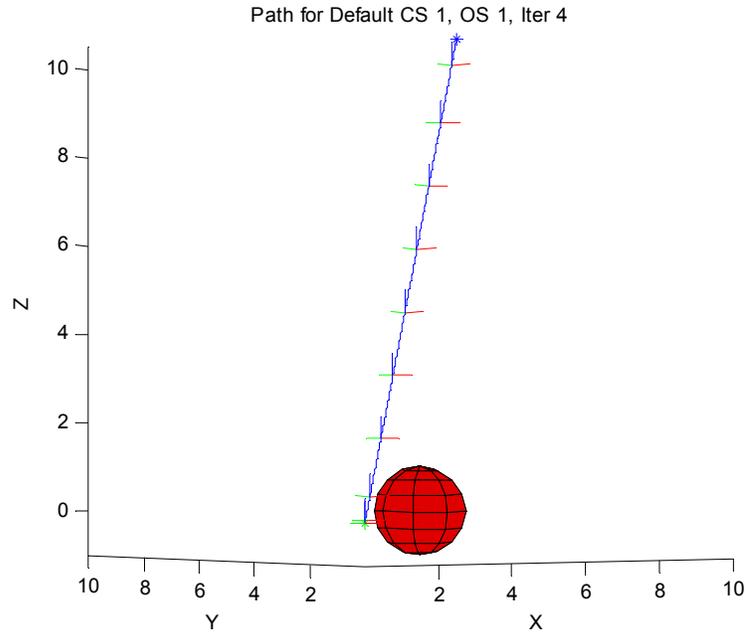


(c)

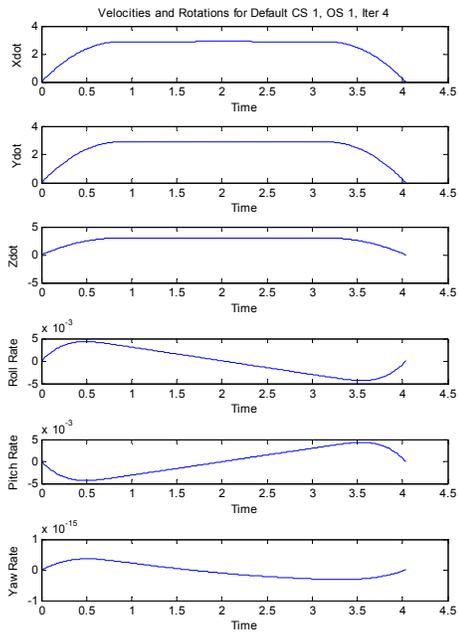
Figure 39: 6DOF a) path, b) rates, and c) input for $\Omega^2 = [1.00, 1.00, 4.57, 1.00]$

This change was successful. Figure 40 shows the results for Ω^4 . Again, the path and the general response of the system was similar to all other solutions. But this time, the feature values met both the \mathbf{H}^0 and \mathbf{S}^0 requirements. *max-speed* was 5.12 m/s, just over the 5.0 m/s low end of “quickly,” and still below the \mathbf{H}^0 limit of 5.5 m/s. *avg-speed* was 4.26 m/s, also still within “quickly.” Here we see a case of convergence without overshoot or cycling. Most of the benefit was gained in the first iteration of *WADJ*, when the time weight was changed from 1.00 to 4.57. Had we stopped there, we would have had only a partial success, but one with a very small soft constraint failure margin. Continuing, we refined the answer further until all constraints were met and a fully acceptable solution was found.

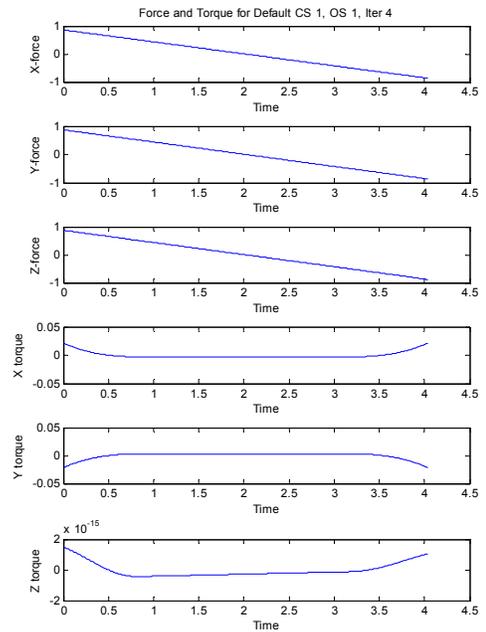
In each of these figures, we noted an apparent discrepancy between the applied forces and the resulting velocities. The forces appeared to be linear, but the velocities did not look quadratic. We plot the velocity gradient in the inertial x -direction along with the applied forces in both the body x -direction and inertial x -direction in Figure 41. Note first that the difference between the body and inertial force is negligible. Although the numeric solver found some small force savings in this case by torquing the vehicle, the savings occurred well below our levels of precision in most cases. Figure 41 also shows exact agreement between the slope of the x -velocity curve and the x -input, as expected.



(a)



(b)



(c)

Figure 40: 6DOF a) path, b) rates and c) input for $\Omega^d = [1.00, 1.00, 4.86, 1.00]$

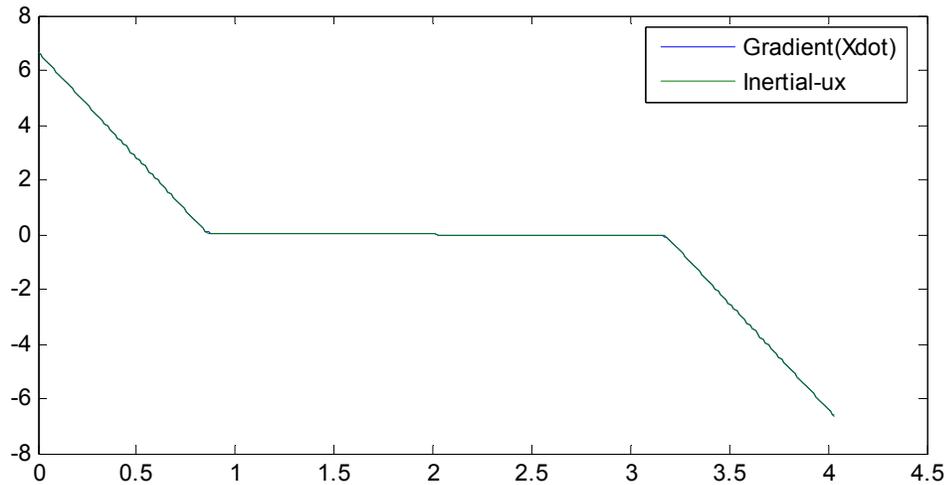
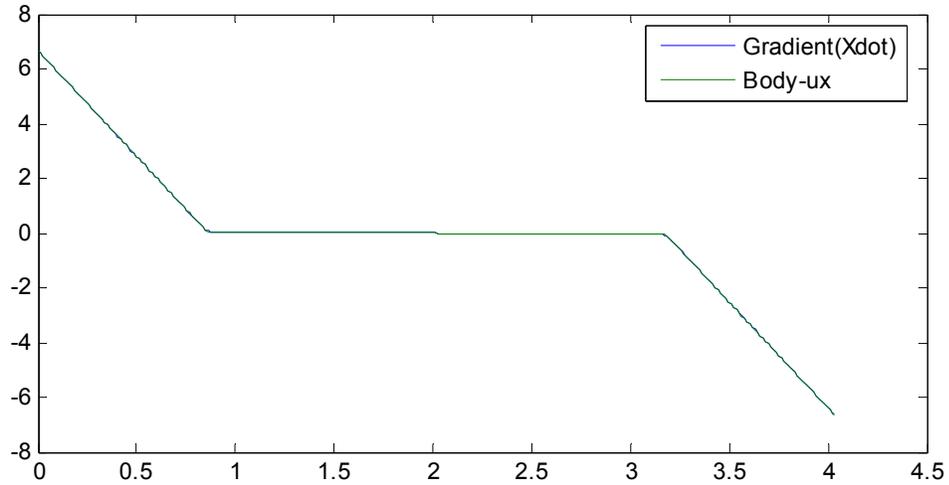


Figure 41: Velocity gradient compared to body and inertial forces

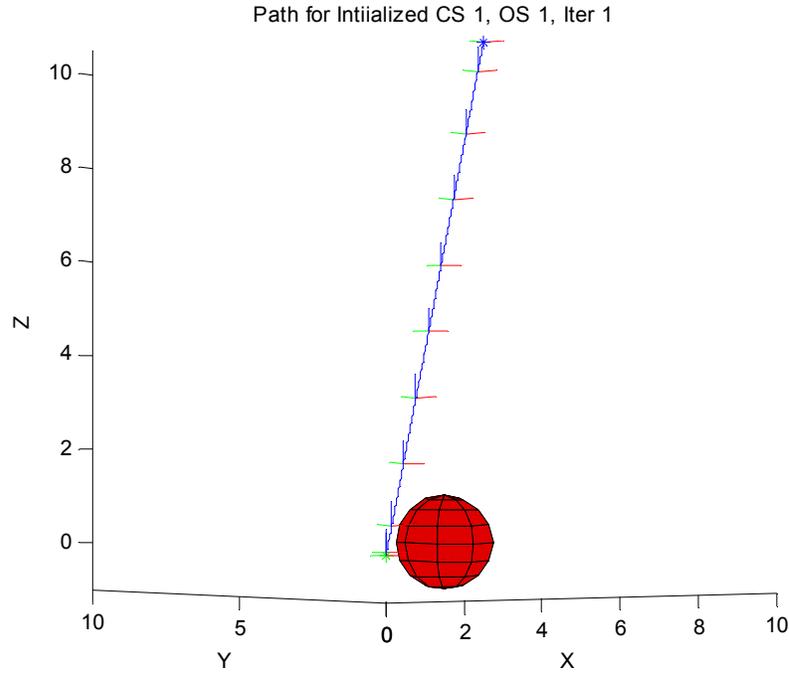
We continue with a shorter example: the same constraint and obstacle set, but starting with an initialized weight vector. *INIT* returns for this $\Omega^1 = [1.00, 1.00, 4.00, 1.00]$. Already, we see that this is much closer to the Ω^2 of the default case. We correspondingly expect that the first returned trajectory will be closer to our expressed requirements and preferences.

Figure 42 shows that our expectations were met. Still similar in form to all the other solutions, *max-speed* was 4.57 m/s and *avg-speed* was 3.90 m/s. \mathbf{H}^0 was met, but

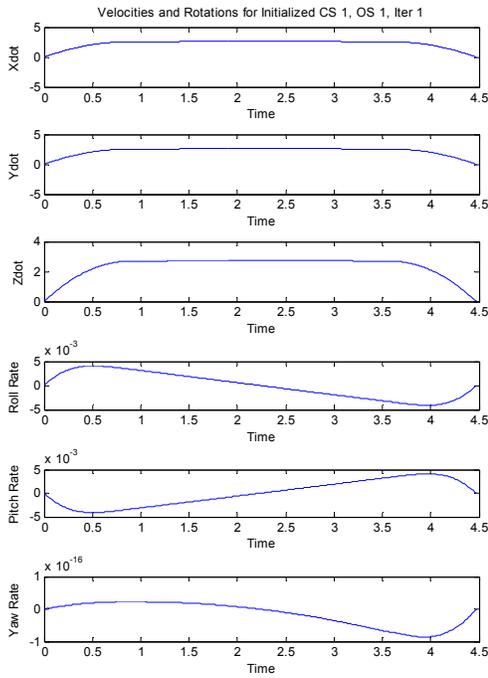
both \mathbf{S}^0 constraints failed. The magnitudes of the failures were much smaller than after the first iteration of the default case, however: only 0.43 m/s for *max-speed* and 0.10 m/s for *avg-speed*. The time weight was, as before, increased by *WADJ* to elicit a faster response; $\mathbf{\Omega}^2 = [1.00, 1.00, 4.79, 1.00]$. This was between the unsuccessful $\mathbf{\Omega}^3$ and successful $\mathbf{\Omega}^4$ of the default case, but is definitely in the area of a known successful weight vector. $\mathbf{\Omega}^2$ succeeded in meeting all constraints. The resulting trajectory, shown in Figure 43, appears to be the same as was found for the default weight vector after four iterations. The *max-speed* and *avg-speed* were the same, 5.12 m/s and 4.26 m/s.

Here, then, was shown the main advantage of *INIT* and the strength of *WADJ*. *INIT* saved two iterations in this example because we were placed in the general area of a successful weight vector on the first try. When trajectory calculations can take from twenty minutes to several hours, this is no small savings. However, regardless of starting point, the *WADJ* heuristics got us to near-identical solutions. (Actually, as the *z*-axis torque results are so very far below our levels of precision that they could be counted as identically zero, we could say that they are identical). Further, we saw that the first use of *WADJ* in the default case resulted in large improvements in solution quality in a single step.

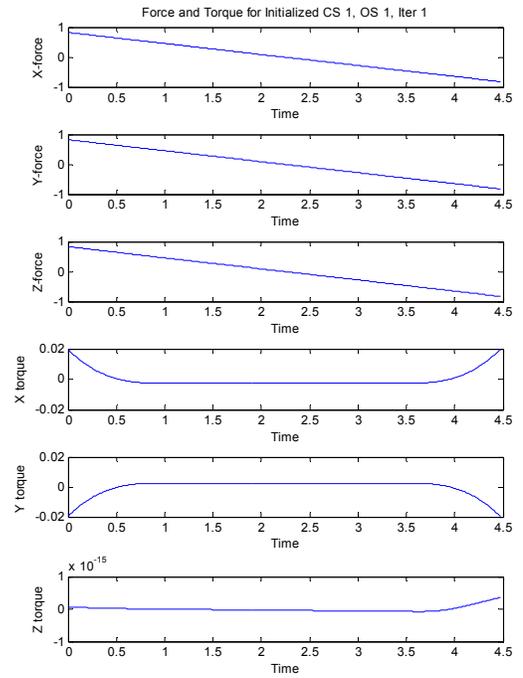
Such success with *INIT* will not always be observed; in cases of competing constraints, as seen in Chapter 4, the weights can oscillate between those satisfying one constraint and those satisfying the other. Either the oscillation magnitude will decrease until eventually settling, or else a cycle will be detected by *EVAL*, upon which case the process will terminate. In either of those cases, solution quality may still improve, although not as consistently as in these examples.



(a)

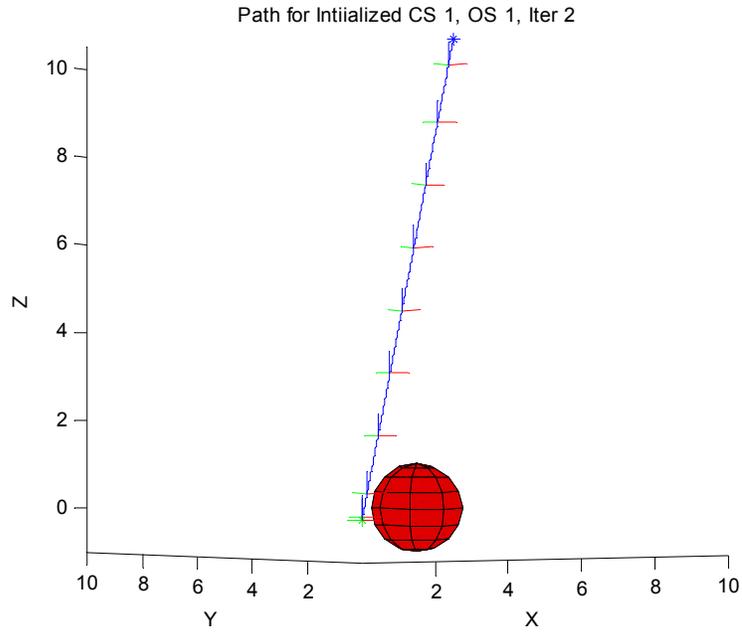


(b)

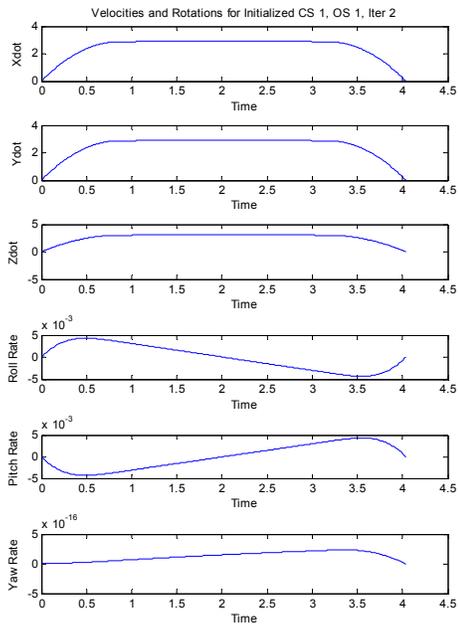


(c)

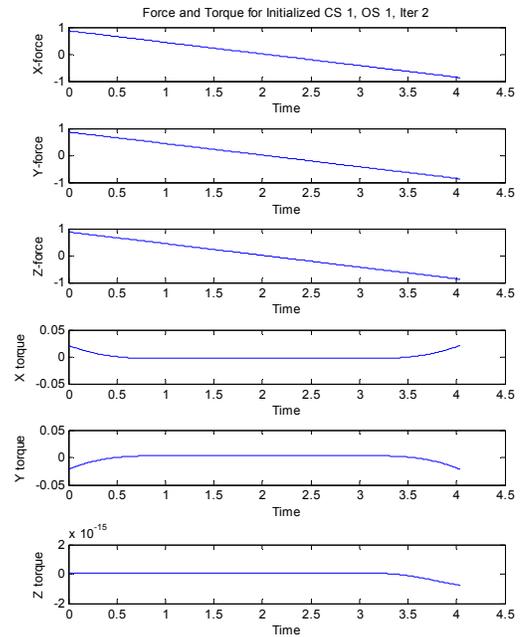
Figure 42: 6DOF a) path, b) rates, and c) inputs for $\Omega^I = [1.00, 1.00, 4.00, 1.0]$



(a)



(b)



(c)

Figure 43: 6DOF a) path, b) rates, and c) inputs for $\Omega^2 = [1.00, 1.00, 4.79, 1.00]$

5.4.2 Overall 6DOF Results

Figure 44 shows the total number of failures for each of our solution cases. As in Chapter 4, these are $INIT^l$, the solution generated using the weights suggested by $INIT$, $INIT^n$, the solution generated by running the $INIT^l$ solution to conclusion through the architecture, $Default^l$, the solution generated using a default weight vector with all weights equal and $LIM = 1$, and $Default^n$, the $Default^l$ solution run to completion.

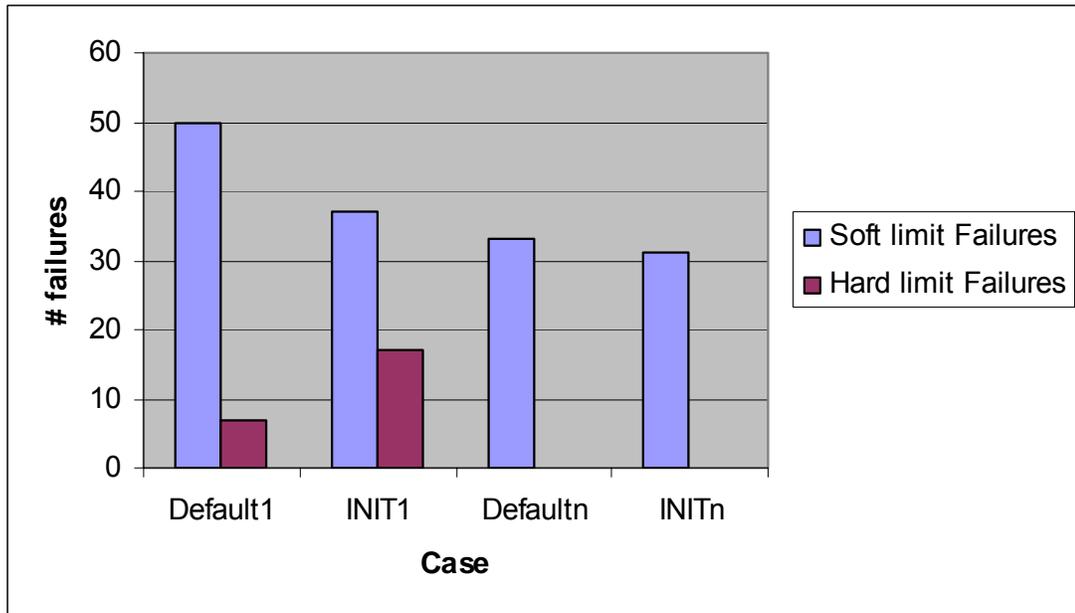


Figure 44: Failures for each solution case out of 20 H^0 and 60 S^0

The results for soft limit failures S^0 are similar to the 2DOF case. The $INIT$ procedure results in noticeably fewer soft constraint failures. There are more hard limit H^0 failures with $INIT$ due to our examples with competing constraints, but again no hard limit failures were present in the final solutions. A closer look at the data in Figure 45 shows that the differences from 2D results are not quite as compelling as might appear at first glance.

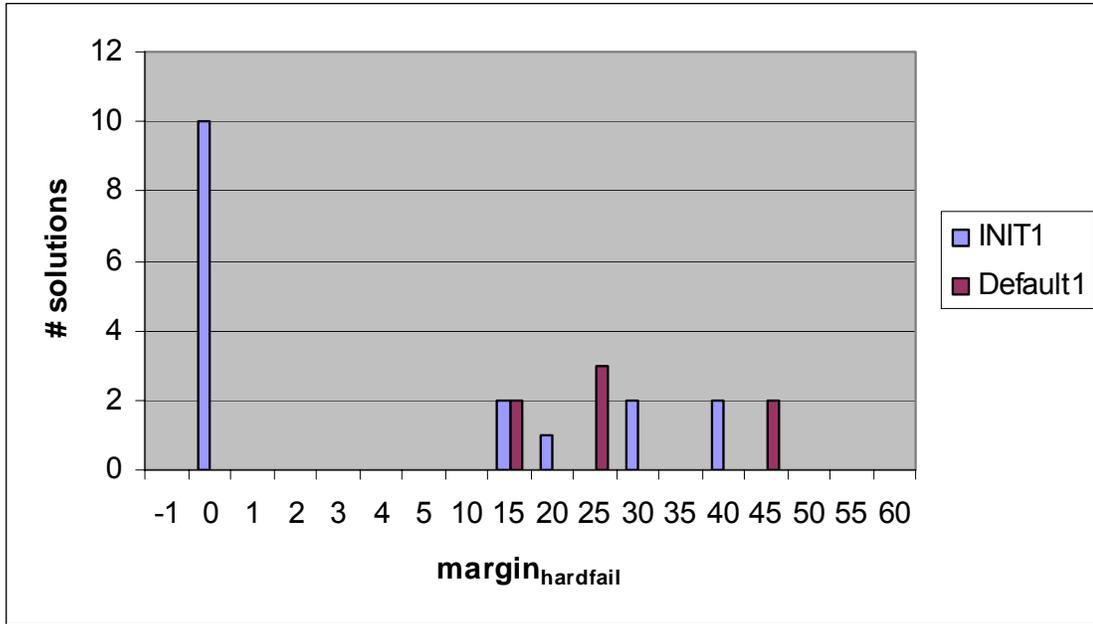


Figure 45: Margins of failure for hard limits after first trajectory generation for 6DOF cases

Although Eq. 4.6 is presented as a margin of success for a hard limit, we use the same equation to calculate the margins of failure here. Most of the $INIT^l$ \mathbf{H}^0 failures, after normalization, fall between -1 and 1. That is, they are relatively small, and the solution was actually fairly close to meeting the hard limits. But it did not, so the question is raised: was the $INIT$ routine helpful in this case? Total limit failures after one trajectory was computed were 54 with $INIT^l$ and 57 with $Default^l$ – not a significant difference. After completion, both $INIT^n$ and $Default^n$ were again without \mathbf{H}^0 failures, and the difference in \mathbf{S}^0 failures was also small (31 for $INIT^n$ and 33 for $Default^n$). The average number of iterations required for completion was 5.1 for $INIT^n$ and 5.3 for $Default^n$, so there was not even the time savings that was seen for the 2DOF case. Although, as in the 2DOF case, $INIT$ solved five problems on the first try, which the default never did, and it had a single case with a large number of iterations (24) that was greater than the longest

default run (11 iterations). But an examination of that 24-iteration run (Constraint Set 1, Obstacle Set 4) shows that, after 2 iterations, all the limits were met within our level of precision. However, the hard limit on max-speed was still exceeded by an amount smaller than that. Since *EVAL* did not round the failure margins to our level of precision, this was viewed as a failure and the architecture kept trying to find a solution until the weights looped. *INIT* had a much-reduced impact on the 6DOF cases, neither significantly helping nor hurting the results. Figure 46 shows a frequency histogram for the number of iterations required (Constraint Set 1, Obstacle Set 4 is omitted for *INIT*ⁿ). The advantage of *INIT* is clearer here; a majority of the runs started with *INIT* finish in one or two runs, while those starting from default weights need three runs at the least.

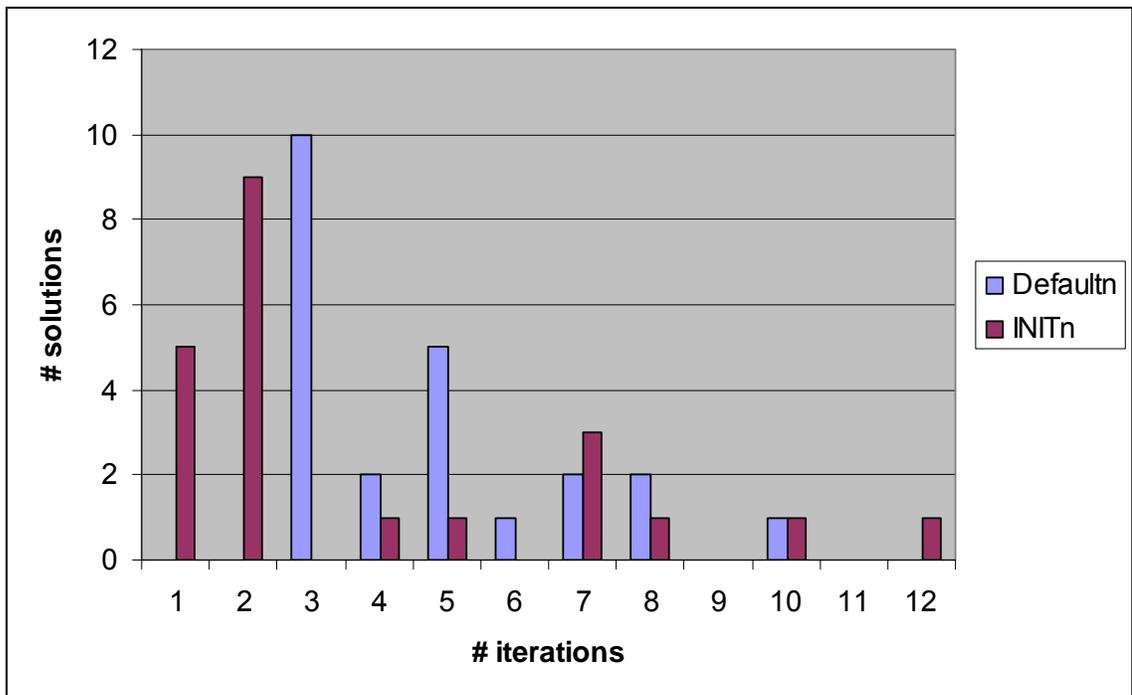


Figure 46: Number of iterations through architecture

Displaying the S^0 failures by obstacle set shows trends similar to the 2DOF case, although the median failure rate is higher. Excluding the $INIT^n$ case in Obstacle Set 4 in Figure 27, the median S^0 failure rate by obstacle set was 50% for the completed cases ($Default^n$ and $INIT^n$) with a spread of about 6%. Here, neglecting the one very low failure rate case in Obstacle Set 1 this time, we have a range from 54% to 77%, with the median at 65.5% S^0 failures in the completed cases. The range of failure rates is broader, but there still does not seem to be an obstacle-based trend in solution fitness.

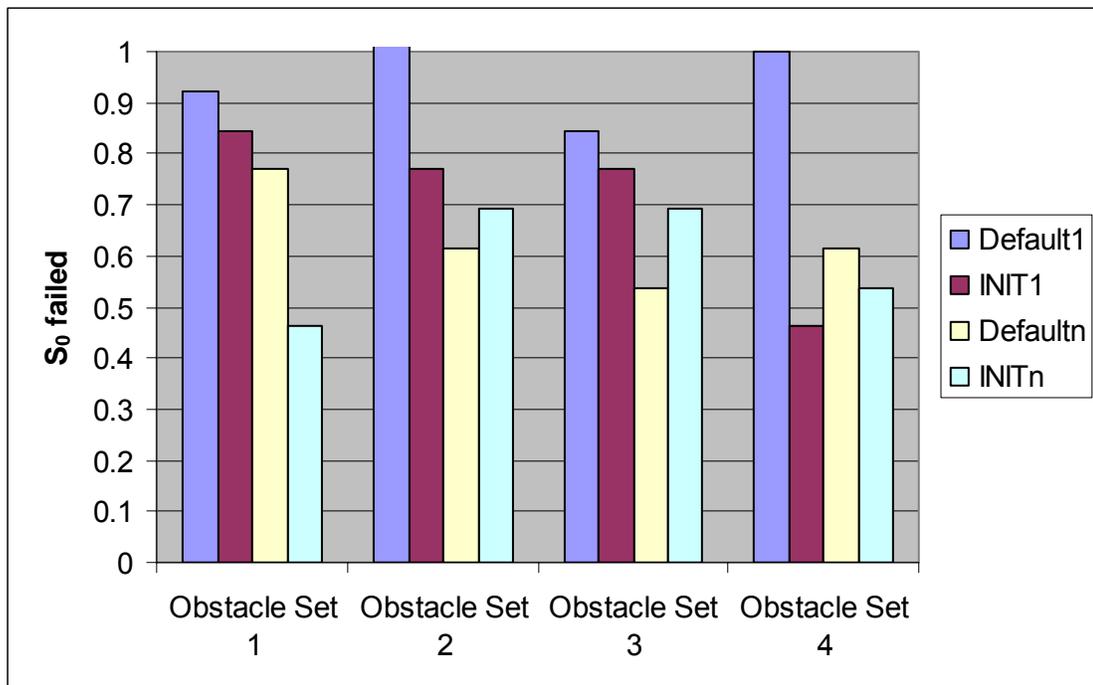


Figure 47: 6DOF S^0 failures by obstacle set

Figure 48 shows the percentage of S^0 failures by constraint set. Here we see definite trends, with some constraint sets being apparently simple to entirely satisfy, while others had 100% S^0 failure rates.

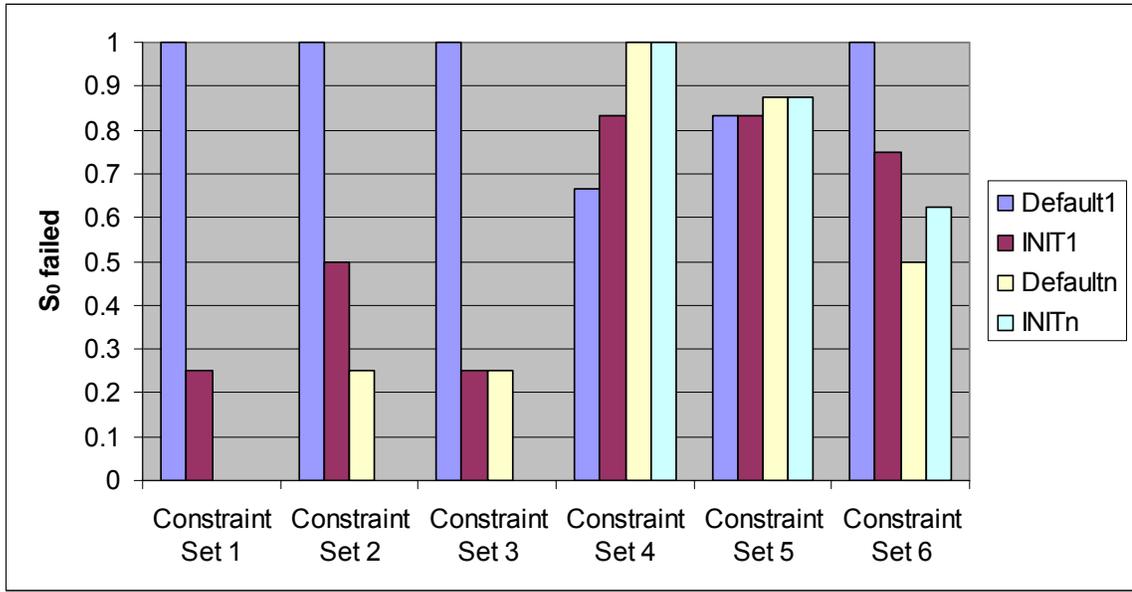


Figure 48: 6DOF S^0 failures by constraint set

Constraint Sets 2 and 3 had small numbers of noncompeting soft constraints and no hard constraints. Constraint Set 2 had a soft numeric range limit on thrust; Constraint Set 3 was “a little quickly,” which defuzzified into soft constraints on *max-speed* and *avg-speed*. With no other requirements, these constraints were solved much more successfully. *INITⁿ* solved them entirely for all obstacle cases; *Defaultⁿ* had small errors with Obstacle Set 1 (in combination with Constraint Set 2) and Obstacle Set 4 (in combination with Constraint Set 3).

Constraint Set 1 was very similar to Constraint Set 1 in the 2DOF case; we set a hard limit on *max-speed* and also the soft preference for “somewhat quickly.” The hard limit was toward the low end of the fuzzy ranges that define “quickly,” forcing the system to hit a small window of feature values that would satisfy both. While this constraint set gave the 2DOF case some problems, here it was entirely successfully solved in all cases that went to completion.

We also note that for these first three constraint sets, *INIT*^d returns remarkably better initial solutions than *Default*^d. By happenstance, the default weights produce results that do not meet any of the \mathbf{S}^0 , while *INIT*^d achieves at least partial success in that. So we see here one practical application for *INIT*: in those cases where there are few or noncompeting constraints, it provides an excellent initial guess compared to using default weights.

Constraint Sets 4 and 5 were clearly less successful. Constraint Set 4 included two hard upper limits on *max-acc* and *max-speed* and two soft range limits on *force* and *avg-speed*. It appears that when the suggested weights for the *force* and *avg-speed* ranges were combined via the centroid calculation, the force terms were much more sensitive to the change away from their own desired values. Further, the hard limit on *max-acc* was greatly exceeded in all initial cases. By the end of the iterations, the hard limits were all met, but *force* was failed in all cases: failed under the lower limit of the range. By requiring such a low *max-acc*, we were required to use less thrust than specified by the soft range. Similarly, all final *avg-speeds* failed low, as the trajectory had to go slowly enough to meet the upper limit on *max-speed* (a hard limit). Essentially, the stated soft constraints in Set 4 *had* to fail for the hard constraints to be met. (This constraint set, when combined with Obstacle Set 4, failed to converge for either the default or initialized case. So only results from Obstacle Sets 1-3 have been considered here.)

Constraint Set 5 added the soft fuzzy preference “moderately safely” to Constraint Set 4. This defuzzified into four more soft range constraints. We were practically guaranteed a certain failure rate, since the upper limit of “safely’s” *avg-speed* constraint equaled the lower limit of the soft range constraint on *avg-speed* from Constraint Set 4.

Of course, if we failed low on *avg-speed*, as we did for all of the Constraint Set 4 cases, we would be making the “safely” constraint, decreasing our overall failure rate. Soft constraints on *max-acc* and *max-speed* arising from “safely” were sometimes met when the hard constraints were met, again decreasing the failure rate. (And when they were failed, they failed low as in Constraint Set 4.) We saw very large failure margins which were greatly reduced by the end of the iterations.

Constraint Set 6 was “very energy-saving,” which decomposed into soft range constraints on *force* and *torque*. But since the only torque needed in the trajectories was that required to avoid obstacles, the trajectories all failed low; they could not use enough torque to satisfy the “low *torque*” constraint. “Low *force*” was more typically made, or failed high with very small margins (0.05, 0.02 N) for the completed cases (*Default_n*, *INIT_n*). Since the nonlinearity of the system is in its rotational dynamics, and since we wish to show that our architecture will work with nonlinear systems, we decided to rewrite the \mathbf{x}_f to include explicit rotational changes rerun it over the set of obstacles under Constraint Set 6. The results are presented separately in Section 5.4.2, Torque, below.

Margins of success for \mathbf{H}^0 were overall much better than in the 2DOF case. Using Eq. 4.6 to define *margin_H*, we had many more cases where the margins were very near 1, as shown in Figure 49.

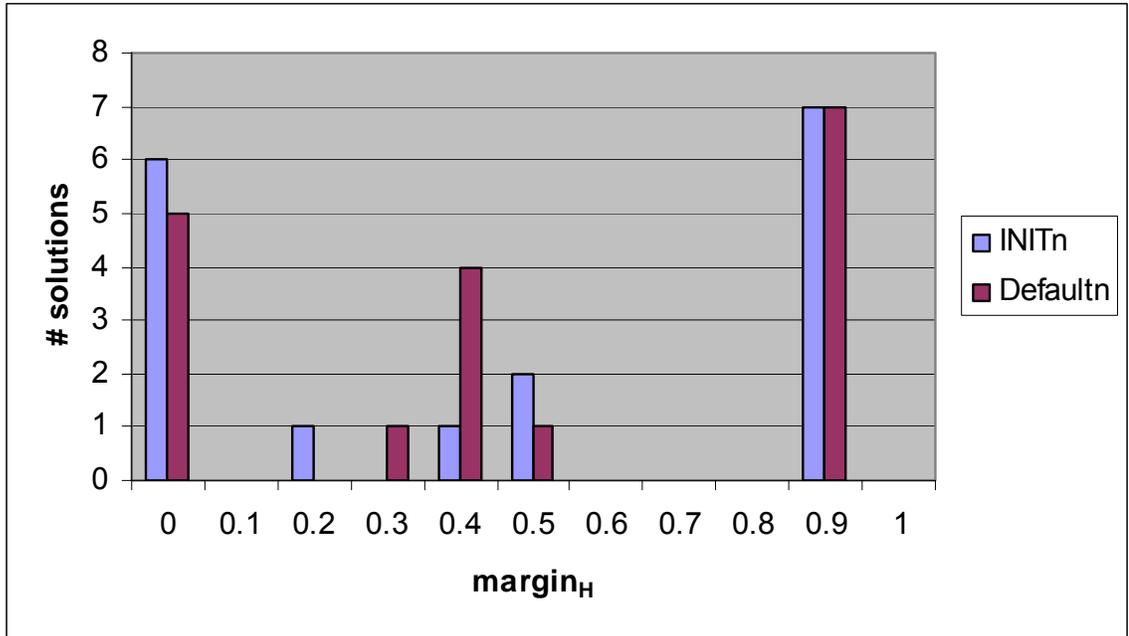


Figure 49: Margins of success for hard limits for 6DOF cases

Recall that we assume that we would like to stay as well under \mathbf{H}^0 as possible. Here we have a full 2/5 of all solutions having features less than 90% of the \mathbf{H}^0 values. We again see that the final solution quality (with respect to \mathbf{H}^0) does not depend significantly on the starting point of the solution. We continue to interpret this as a sign of the robustness of the overall architecture.

Figure 50 shows the margins of success for \mathbf{S}^0 in these cases. $margin_S$ is defined in Eq. 4.10. More of the data lies at the edges of the histogram than in the 2DOF case. Thirteen of the cases binned at -0.9 and -0.8 (for both $INIT^n$ and $Default^n$) resulted from Constraint Set 1, which required “high *max-speed*” as a soft limit but “*max-speed* ≤ 5.5 m/s” as a hard limit. The range for “high *max-speed*” was defined for the 6DOF domain as $5.0 \leq max-speed \leq 10.6$. The result is that the architecture takes full advantage of the fuzziness of the soft constraint and drives it well into the low end of that range to meet the more restrictive hard constraint.

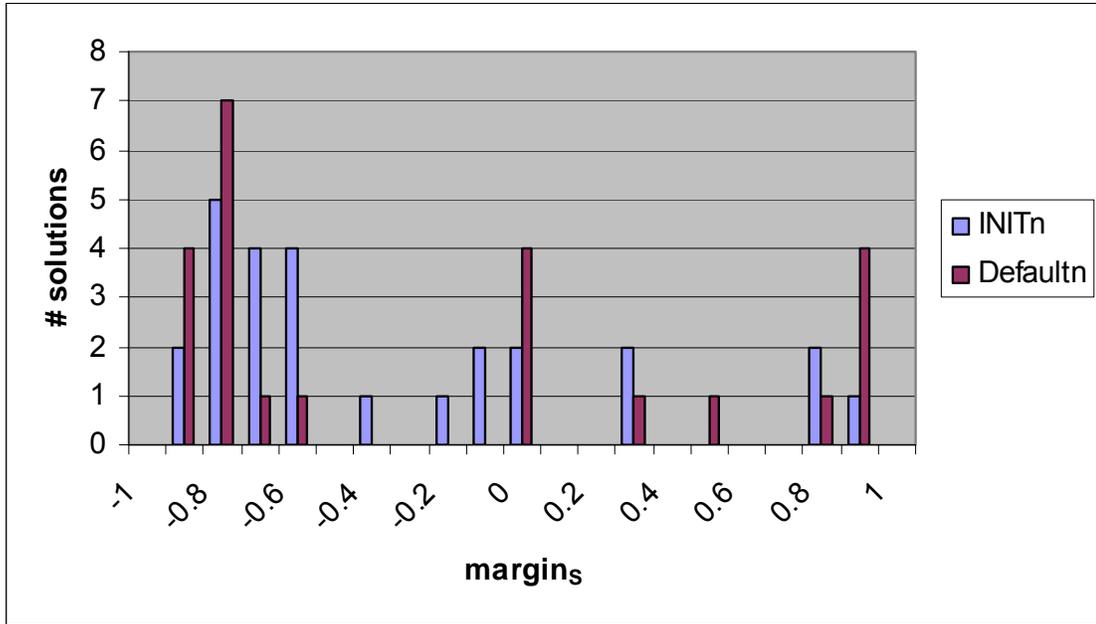


Figure 50: Margins of success for soft limits for 6DOF cases

Figure 51 shows the margins for S^0 failures, calculated according to Eq. 4.11.

Unlike Figure 31, there is no tail of high margins. In this respect, we had better performance in the complex 6DOF domain than in the 2DOF domain. Again, final results are not much affected by initialization.

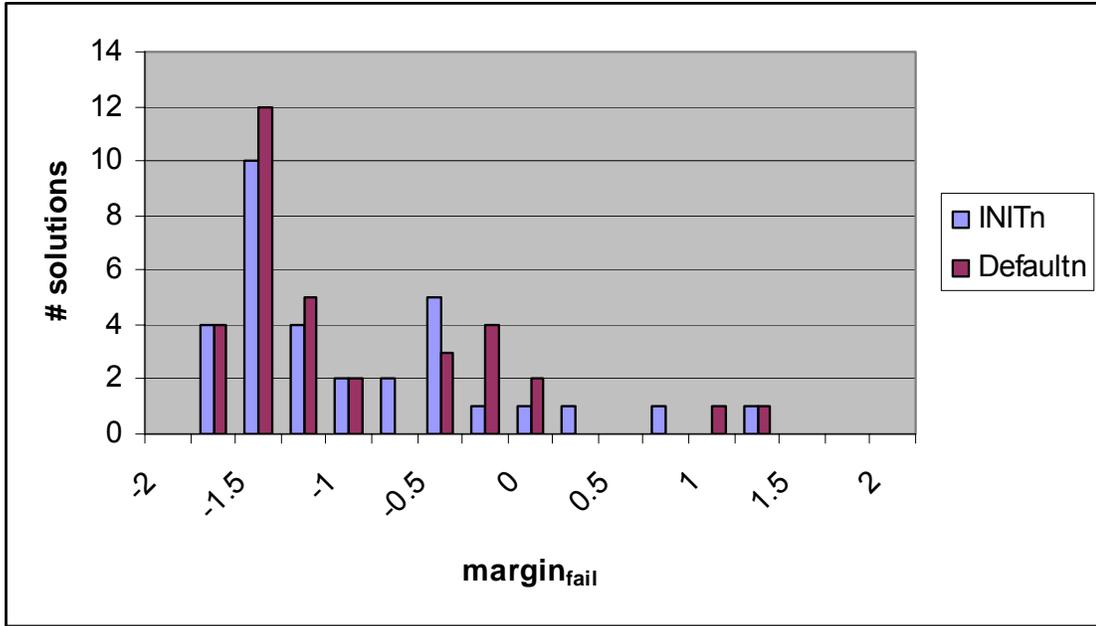


Figure 51: Margins of failure for soft limits for 6DOF cases

5.4.3 WADJ Performance

For many of the runs, a solution was returned after between one and three iterations (Figure 46). For one or two iterations, the weights are converged upon with no overshoot. For those default weight cases that took three iterations, some converged monotonically to the correct weights, while others overshoot on the second iteration and corrected with the third. How did the system behave for more complex constraint sets? Figure 52 shows a trace for the W_3/W_1 and LIM weights for the initialized run through Constraint Set 5, Obstacle Set 1.

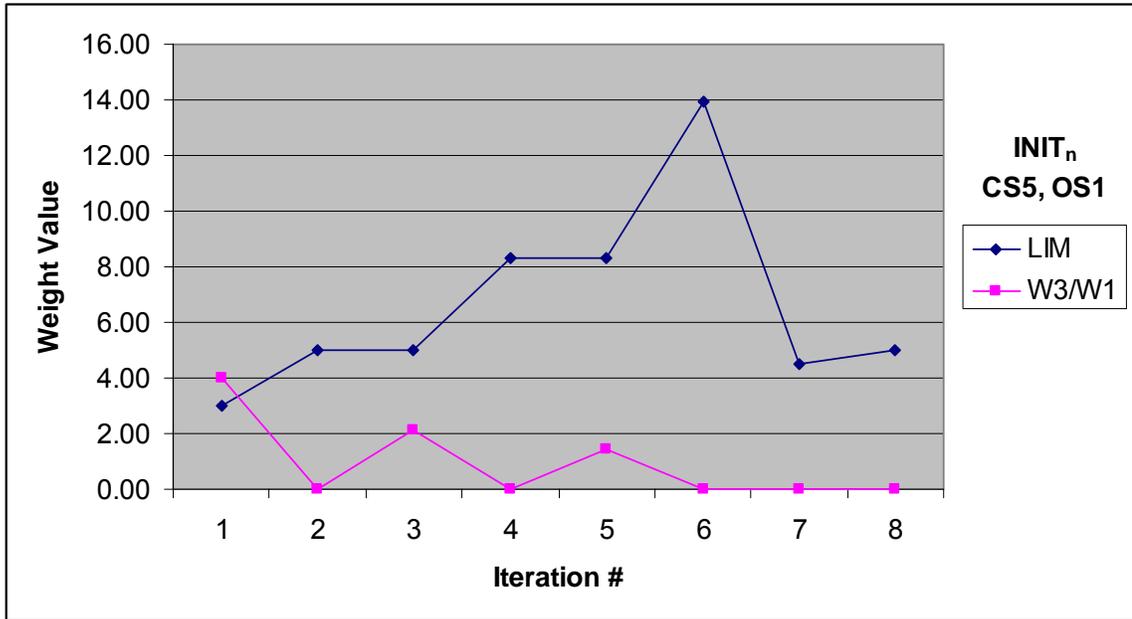


Figure 52: Weight values evolving through Constraint Set 5, Obstacle Set 1 starting from $INIT^d$

Constraint Set 5 had hard upper limits on *max-speed* and *max-acc*, soft limits on *force* and *avg-speed*, and the fuzzy constraint “moderately safely.” It was nearly impossible to satisfy the soft requirements on *avg-speed*, which would only be met by achieving an *avg-speed* of 1.8 m/s exactly. The initial guess for W_3/W_1 greatly underpredicted the *max-acc* for the trajectory, and $WADJ$ adjusted that weight down from 4.0 to 0.03 to compensate. At the same time, the minimum *min-sep* entailed by “safely” was not met, so LIM was increased.

In the second iteration, the hard limits and the min-sep soft limit were all met, but the remaining soft range limits were all failed low. The system was too slow and used too little force. Since the current weight vector Ω^2 satisfied \mathbf{H}^0 , it was stored as a possible return value. Then the routine continued to see if it could also meet the \mathbf{S}^0 . W_3/W_1 was increased again to try and meet these soft limits.

That failed the *max-acc* hard upper limit again and the *min-sep* soft limit. W_3/W_1 was decreased to 0.01 and *LIM* increased even further to get the weights for the fourth iteration Ω^4 .

This pattern: decreasing W_3/W_1 and increasing *LIM*, meeting \mathbf{H}^0 but not all \mathbf{S}^0 , and then readjusting W_3/W_1 to a higher value was repeated in iterations 4 through 6. After iteration 6, the first attempt at *WADJ* caused a loop in the weights. So *EVAL* reverted to the best weights found thus far, Ω^2 , and called the secondary *WADJ* heuristic rule. This decreased *LIM* by 10% while leaving W_3/W_1 unchanged.

Although the results of the second and seventh iteration weight vectors are identical to our levels of precision, the *EVAL* routine found some small differences and judged Ω^2 and its results superior. When the seventh iteration returned with \mathbf{S}^0 failures, our *time_limit* was reached and Ω^2 and its resulting trajectory were returned.

5.4.4 Torque

In our original data set, only one of eight completed trajectories (4 *Default*ⁿ, 4 *INIT*ⁿ) met the “low *torque*” requirement. The rest were too low to be considered “low” by the standards of our fuzzy rule set. Since none of our goal states required a rotation, the only rotations required were those needed to avoid obstacles. These did not use sufficient torque to be considered low. So, to test the *torque WADJ* rules, we included a rotation change in each axis at the goal state and re-ran the tests.

We also had some concerns about the possible interaction of *force* and *torque*. In *WADJ*, all features except *torque* are first checked for adjustment. Then the selected W_3/W_1 ratio is used together with the desired *torque* value to calculate a slope from Figure 36; that slope is then used to pick a W_2/W_1 ratio via the equation in Figure 37. The

“low *force*” requirement was keeping W_3/W_1 small, and the heuristic is less well-conditioned for W_3/W_1 less than 1. Although meeting mixed constraints is an important goal, we also wanted to isolate the torque response to the *WADJ* process, since it is so different from the other *WADJ* heuristics. So we created additional test cases: Constraint Set 7, “low *torque*,” and Constraint Set 8, “medium *torque*.”

Figure 53 shows our overall failure rates for these three constraint sets (CS 7, 8, and the revised CS 6); each case had 16 runs (four soft constraints run over four obstacle sets). The *Default*^l and *INIT*^l cases are high again, not unexpectedly, and the failure rates for the completed runs are much lower. All seven failures at run completion were torque failing low; of those seven, four were from the “medium torque” Constraint Set 8. W_2/W_1 was continually adjusted down to discount it, to allow for greater torque in these cases, but what was required was that W_3/W_1 be increased. Since there was a tacit assumption that some other state feature would be relying on W_3/W_1 and that it may have been adjusted to affect that other feature, the torque *WADJ* never altered W_3/W_1 , and W_2/W_1 could not be adjusted sufficiently before *time_limit* was reached. Our current *TPLAN* cannot handle direct maximization of trajectory qualities; it can only minimize. We have found that we can minimize features which are inversely related to our feature of interest for a maximizing effect; thus by penalizing time, we can usually force in increase in speed. Another *TPLAN* might allow for direct maximization of features.

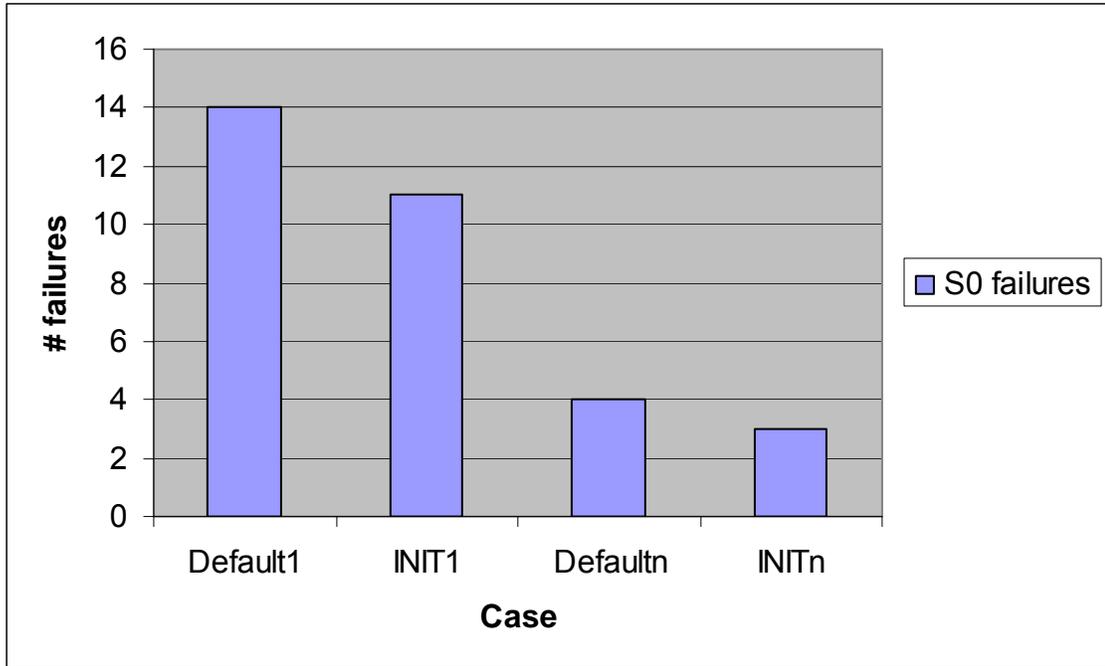


Figure 53: 6DOF S^0 failures for Constraint Sets 6, 7 and 8

Figure 54 shows the number of iterations required for these runs. All of those runs which took four or fewer iterations to return a solution returned a complete success. The utility of *INIT* is again shown in the large number of runs that returned successful trajectories after only one or two iterations; eight ($2/3$ of the total) of the trajectories created using *INIT* were solved in two or fewer iterations, while only 3 trajectories created using default weights met this standard.

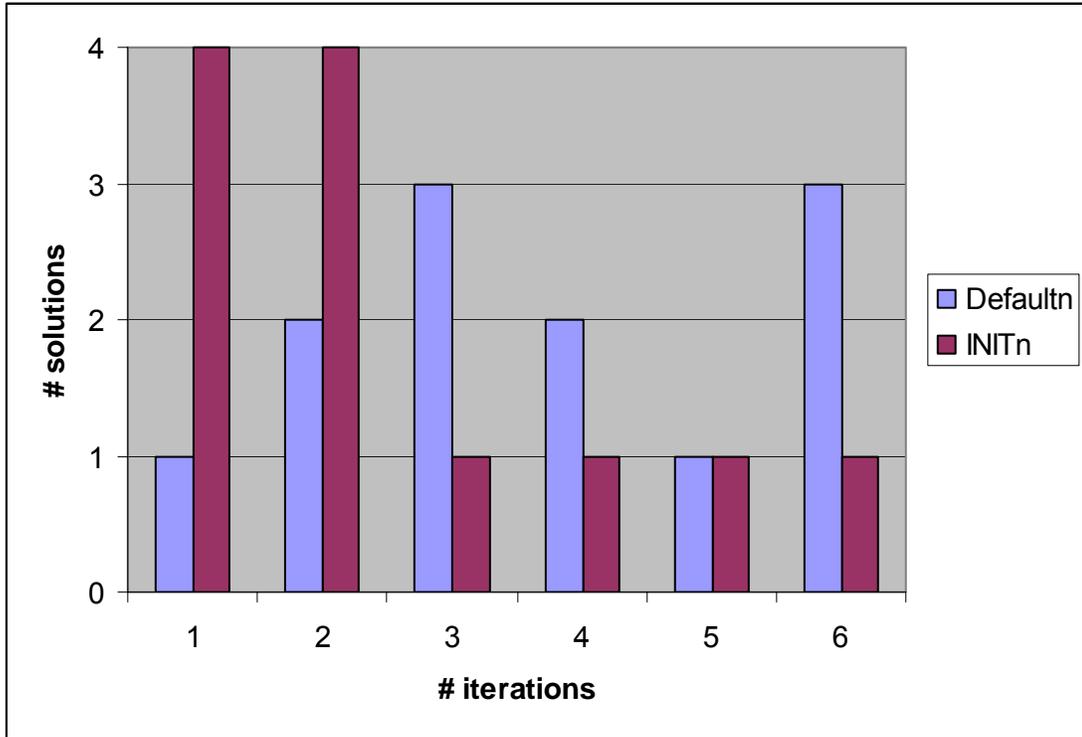


Figure 54: Number of iterations required for Constraint Sets 6, 7 and 8

The margins of success on these runs showed no particular trends. The margins of failure on the cases run to completion were all small, with the normalized magnitude of the largest being 0.51; the smallest on Constraint Set 6, Obstacle Set 3, which failed low on *torque* by a normalized margin of only -0.07.

5.5 Conclusions

In this chapter we have demonstrated that *WADJ* heuristics can be developed for a deep space 6DOF domain with nonlinear dynamics. Our results were, if anything, better in the 6DOF domain than in our 2DOF domain, with smaller \mathbf{S}^0 failure margins and larger success margins for \mathbf{H}^0 . The average number of iterations required to find a

solution was commensurate with the 2DOF case, also arguing that the technique implemented in the architecture will scale well with the dynamic complexity of the domain. The surprising similarity of the 2DOF and 6DOF *WADJ* curves, even to the values of the coefficients, is noteworthy. That also argues for the potential for a general application to the optimization of dynamic systems.

The performance of our *TPLAN*, BVP4C2, was less robust and far more slow than we had hoped. Prior work on this algorithm required extensive hand-tuning of several sets of gains just to solve a single trajectory problem. We were running it, on average, 5.2 times per problem for 24 fairly different problems. So these difficulties are not entirely unexpected. In the future, however, a different *TPLAN* should be selected for work with nonlinear systems.

6 Conclusions and Future Work

The examples in Chapter 4 and 5 have shown that our architecture from Chapter 3 to optimize trajectories over hard constraints and natural language preferences can indeed achieve our requirements:

- The *WADJ* heuristics consistently direct the weights toward values that meet hard and soft constraints and are robust to differences in initial weight sets
- The fuzzy logic enables a more natural human interface, opening a route to easy tasking of autonomous agents by non-expert users (e.g., hospital staff commanding a robotic assistant, warfighters with a Future Combat System robot, the elderly using a companion robot).
- The ability to meet hard numeric constraints is not lost in adding the fuzziness. This allows the system to be used as an “automated graduate student,” overseeing trajectory generation, rejecting those which do not meet required hard constraints, and making intelligent adjustments to the weights to move the solution in the required direction.
- Proper weight initialization saves computation time.
 - One iteration saved on average in 2DOF
 - 60% of 6DOF cases solved in under 3 iterations
- Substantial knowledge engineering and preprocessing was required to develop the fuzzy rules, the *WADJ* rules, and the *TPLAN* implementation.
- But once this offline process was completed, the system was applicable to a wide range of obstacle and constraint conditions with no further adjustments.

- This makes the architecture useful for robots operating long-term in a consistent environment, but not so useful for “one off” operations such as technology demonstrations.

In all of our 86 cases run to completion, every hard constraint placed on the trajectory was met. No maximum accelerations or velocities were ever exceeded, and no paths through obstacles were ever returned.

Our test runs were specifically constructed to include competing soft constraints so that it was not possible to satisfy them all. Our architecture did manage to balance these competing constraints well overall, returning trajectories that had typically small margins of failure. A few soft constraint failures were notably larger; future work should discover if this is the result of fuzzy rules degradation in the presence of obstacles (as seems likely).

The techniques developed for construction of *WADJ* curves that the architecture uses to adjust cost functional weights and affect trajectory features were generalizable from the linear 2DOF case to the nonlinear 6DOF case, although more manipulation and insight was required of the 6DOF case.

The *INIT* routine was somewhat useful in the 2DOF linear case, but on average made little difference in the 6DOF nonlinear case, which was surprising. It did allow, in certain constraint/obstacle combinations where there were no competing constraints, the one-iteration solution of the problem, which a default weight vector never achieved. It could also set the architecture up for a cycle between two opposing weights that would take longer to resolve than the default weights typically did. Those problems with fewer

and non-competing constraints more frequently achieved the one-step solution; certainly *INIT* should continue to be used for problems with similar characteristics.

A substantial knowledge engineering effort was required to develop the *WADJ* curves and fuzzy rules sets. Once this was done, the resulting heuristics were useful over a range of constraint and obstacle sets. While the work required to set up *WADJ* and an appropriate *INIT* is not trivial, it is far less than the time required to compute the Pareto front for the trajectory planning problem. Once the framework for developing *WADJ* rules and the databases used by *INIT* was developed, the domain-specific development could be accomplished within a few days. This is because the simulations were run in empty or mostly-empty fields, for which the trajectory planning problem solves quickly. The architecture was then ready to run trajectory planning problems, returning good solutions after an average of five or six iterations of trajectory generation. A run of twenty-four iterations was considered anomalously high. For a Pareto front to be developed for a single point-to-point traversal, the EMO methods would have to generate and evaluate hundreds or thousands of candidate trajectories. In a cluttered field, for a nonlinear system, using a computationally-intensive solver like BVP4C, each trajectory generation can take hours.

For one-off technology demonstrations or single-mission robots, this approach may not be the right one. In those cases, effort spent creating a generic profile of the system might be better invested in carefully engineering the one particular trajectory of interest. This architecture is useful when a robot is going to be active in the same domain for a long time, performing a variety of missions under different dynamic constraints.

6.1 Future Work

6.1.1 Improving *INIT*

A more sophisticated version of *INIT* could look at the constraint set as a whole and recognize potential conflicts. Currently, the system will go through many iterations trying to satisfy conflicting constraints. If the conflicting constraints are both hard constraints, it could be many iterations before a weight cycle is detected. (In this research, the only such cycle took 24 iterations to be detected). An early detection of this kind of possible conflict, or else a software monitor that detects a pattern of cycling back and forth for “too many” iterations (where “too many” may be set at the user’s discretion) would both be useful to have.

INIT should also be invariant to the order in which constraints are processed. In this implementation, the order in which the hard constraints are considered impacts the returned weight vector. After the soft constraints have been aggregated via a centroid computation, *INIT* cycles through the hard constraints and checks to see if the currently suggested weights are liable to meet them. If they are not, *INIT* adjusts the weights up or down as needed. If competing constraints are being considered, the last one addressed by *INIT* will be favored, rather than a median weight which might satisfy both. Given that we do not know the number nor type of our hard constraints *a priori*, this problem might be better addressed within a cognitive architecture, where its pattern-matching abilities would be very useful.

6.1.2 Improving *WADJ*

After the *INIT* cycle, the “adverbial modifiers” like “very” or “somewhat” are lost in the weight adjustment process. The endpoints of the fuzzy regions for the soft

constraints are fixed, without regard to the strength of the user's preference. Nor are they currently considered when deciding which of several competing soft constraints must fail. The assumption has been that the *INIT* process would put the solution in approximately the correct region in weight-space, and further iterations would reflect that. That assumption does not necessarily hold, as the *WADJ* rules can cause oscillations of initially very large, then decreasing, magnitude in weight space. Something that preserves the knowledge of soft preference strength past the *INIT* phase would help this adhere more closely to true user preference, and perhaps reduce total iterations needed.

A more sophisticated notion of error margins in *FEXT* might also be of use here. A *WADJ* algorithm that seeks to minimize the entire vector of errors, rather than each error individually, would be computationally more expensive (an optimization within an optimization) but could yield superior results with fewer iterations.

Finally, other forms of *WADJ* specific to other cost functionals could be explored. A cost functional based on a linear quadratic regulator (LQR), in which components of the state vector like the velocities are directly penalized, could replace the time component of the cost functionals used here. Of course, these new terms would still have weighting terms and the relationships between them would have to be investigated, following the procedures outlined here.

6.1.3 Improving *EVAL*

We would like to augment *EVAL* with an understanding of the adverbial modifiers, as mentioned above, so that preferences the user described as weaker would be violated in favor of meeting more strongly-held preferences. Additionally, some mechanism whereby the original set of limits \mathbf{L}^0 can be revisited and perhaps altered by

the architecture is an avenue of further research. There could be cases where the slight easement of a limit could lead to an overall acceptable solution; we would like to be able to identify these cases and flag them for the user. In this vein, the addition of “firm” versus “hard” or “soft” constraints might be considered: those constraints which the user very greatly prefers to be met, but which do not indicate total failure if failed.

6.1.4 Developing the Fuzzy Rule Database

The fuzzy rules used in this research were created in an *ad hoc* fashion. In practice, they would possibly be generated in a more principled way. For a robot that was to interact with the general public, a human user survey could be conducted to learn what the user would consider a typical robot response to commands such as “come quickly” or “follow carefully.” These expectations would then be incorporated into the fuzzy rules database.

If the robot was to be a personal assistant, then either some programmable interface or else an automated machine learning technique could be implemented to allow the robot to more closely conform to a single user’s expectations.

6.1.5 Application to Other Adjustable Parameters

Many numerical solution techniques used today involve the adjustment, usually by hand and by good judgment, of certain parameters. They include parameters such as the continuance schedule gains used for BVP4C and BVP4C2 solutions or the rates controlling mutation and crossover in an evolutionary algorithm. Can the human judgment gained by trial and error for these parameters be distilled into some sort of adjustment curves, as the cost functional weights were distilled in the *WADJ*

calculations? Can we make plots of “algorithm convergence vs. parameter?” This would serve to generalize this work beyond the optimal control community, if it could be done.

Some optimization routines use negative weights in the cost functional, to allow certain terms (e.g., a quality measure) to be maximized. Users must be very careful when doing this, because it becomes possible for the term to grow without bound as time goes to infinity. The cost goes to negative infinity, dominated by this term times its negative constant. If the user has determined that, due to the properties of his particular problem, this will not happen, then such a term can be used. This work does not investigate the possibility of adding such terms, and we could look to that in the future as well.

6.2 Final Summary

In this dissertation, we have developed an architecture which brings together several tools for the autonomous generation of preference-optimized trajectories. Both hard and soft constraints are handled, with tradeoffs between soft constraints being made in an intelligent fashion. This intelligence comes from cost functional weight adjustment guidelines developed from domain-specific data, and the methods for collecting and interpreting this data that have been developed seem to be generalizable from linear to nonlinear domains. We have been able to find no prior work that addresses this problem of weight selection and adjustment in anything other than an *ad hoc* fashion, much less provide an overarching framework for its application to a variety of domains. As we require humans to interact with and task robots to perform autonomous missions, whether on land, sea, air, or in space, we will need ways for the robot to understand the human user’s requirements on it. This work is a step in that direction.

Appendix A: Features and Limits

Features

The following trajectory features were used or considered for use in this research:

max-speed, the maximum forward speed of the vehicle over the trajectory

avg-speed, the average forward vehicle speed over the trajectory

max-acc, the maximum absolute value of the vehicle acceleration over the trajectory

avg-acc, the average absolute value of the vehicle acceleration over the trajectory

speed-plat, a measure of how long the vehicle maintained a constant speed (plateau);

taken to be a region where speed does not vary by more than $\pm 0.5\%$

(or 1% total) of its overall range. (Condition 1) and which extends for at

least 10% of the time domain. (Condition 2)

acc-plat, defined as speed-plat but for acceleration

min-sep, the minimum separation of the path from each obstacle in the environment

max-rot-vel, *avg-rot-vel*, and *rot-plat-vel* were not considered, although they could have been defined as the terms above.

Additionally, the cost functional terms were considered features. In the 2DOF domain, these were:

energy, the electrical energy used to move the vehicle forward

time, the time taken for the trajectory to complete

obstacle_penalty, as described in Eq. 4.5

In the 6DOF domain, the cost functional feature terms were:

force, the force exerted by the thrusters

torque, expressed as an electrical minimum-energy term

time, time taken for the trajectory to complete

obstacle penalty

Limits

Hard or soft numeric limits could be placed on any of the features. In practice, we used only upper for lower hard limits and soft range limits. That is, for hard limits, we required that the feature value be above or below some value. For soft numeric limits, we required that the feature be between two values. However, the architecture handles soft upper or lower limits as well (although the centroid calculation in *INIT* would have to be modified for best results).

The other type of limit used was the soft word constraints. We used words like “quickly” or “safely” as optimization parameters. To translate these words into something the architecture could use, we broke them down into sets of fuzzy feature values on different state features. These definitions are applied across domains: “quickly” means “high *max-speed*” and “high *avg-speed*” everywhere it is used. However, the actual value in meters per second that constitute high speed in a given domain may change. They are loaded from a domain-specific file in our implementation.

We defined the following words for this research, although not all were used in the results presented in this dissertation:

- Boldly = {low *min-sep*, high *speed-plat*, high *acc-plat*, medium high *avg-speed*}
- Cautiously = {low *min-sep*, low *max-speed*, low *avg-speed*, low *max-acc*, low *speed-plat*, low *acc-plat*}
- Efficiently = {low *force*} (for 6DOF) or {low *energy*} (for 2DOF)

- Energy-saving = {low *force*, low *torque*} (for 6DOF only)
- Inquisitively = {low *min-sep*, medium-low *avg-speed*}
- Quickly = {high *max-speed*, high *avg-speed*}
- Safely = {medium high *min-sep*, low *max-speed*, low *avg-speed*, low *max-acc*}
- Stealthily = {low *min-sep*, medium low *avg-speed*}

Although *WADJ* heuristic development results seemed to show that *speed-plat* and *acc-plat* could be predictably controlled, that proved to not be the case in the presence of obstacles. In very open or uncluttered terrain, one might expect to be able to command them, but not otherwise.

We additionally defined a large number of verbs and adverbs, correlating them to upper, lower, or range limits on a host of feature values, including:

speed smoothness, a hypothetical measure of the rate of change in speed over the trajectory; perhaps the linearity of the acceleration

acceleration smoothness, a hypothetical measure of the rate of change of the acceleration over the trajectory

sign changes of acceleration: the number of times the vehicle changes from accelerating to decelerating or visa versa

length, path length; perhaps normalized with respect to the straight-line distance between start and goal

speed near obstacle

sign changes in rotational velocity

rotational velocity smoothness

max-rot-acc, *avg-rot-acc*, *rot-acc-plat*, as other *max*, *avg*, and *plat* values for
rotational acceleration

sign changes in rotational acceleration

rotational acceleration smoothness

target_acqu, the precision with which a target is acquired

under_cover, the percent of the trajectory the vehicle remains in terrain known to be
cover

The definitions are given below. The state features involved with each word limit are marked with an x.

Appendix B: Test Cases and Margin Data

2DOF Cases

Start Point, Goal Point, and Obstacle Sets

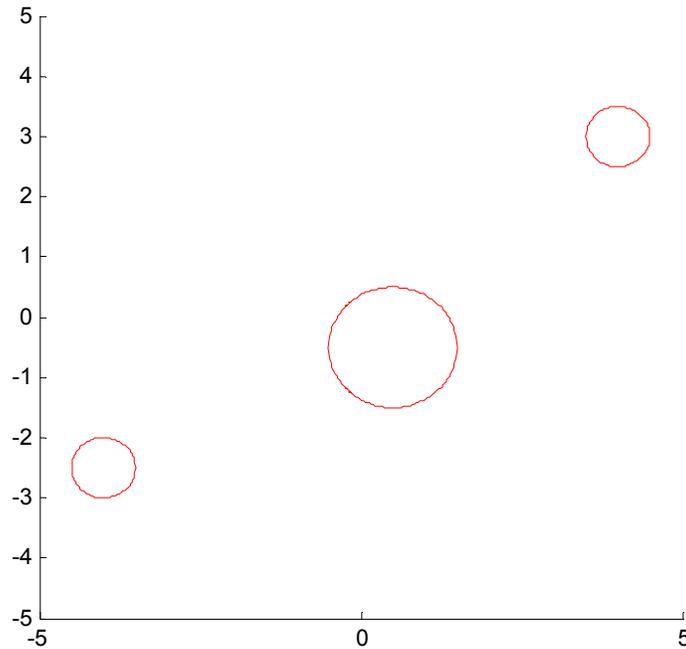
The state x is a 4-vector [position, velocity] in the x - y plane. In all cases:

$$x_\theta = [-5 \ -5 \ 0 \ 0]$$

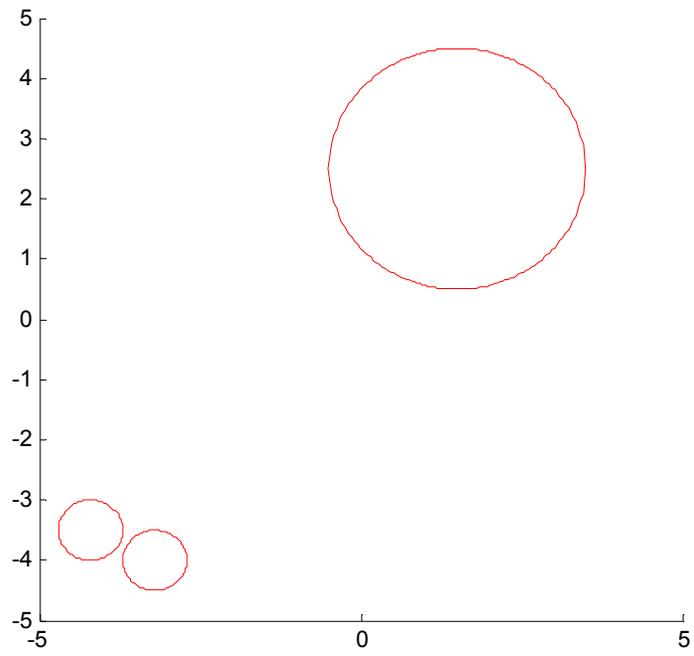
$$x_f = [5 \ 5 \ 0 \ 0]$$

Circular obstacles are denoted as a set containing their center in the plane (x, y) and their radius r , all in meters. The set of obstacles is denoted $\{\mathbf{O}\}$.

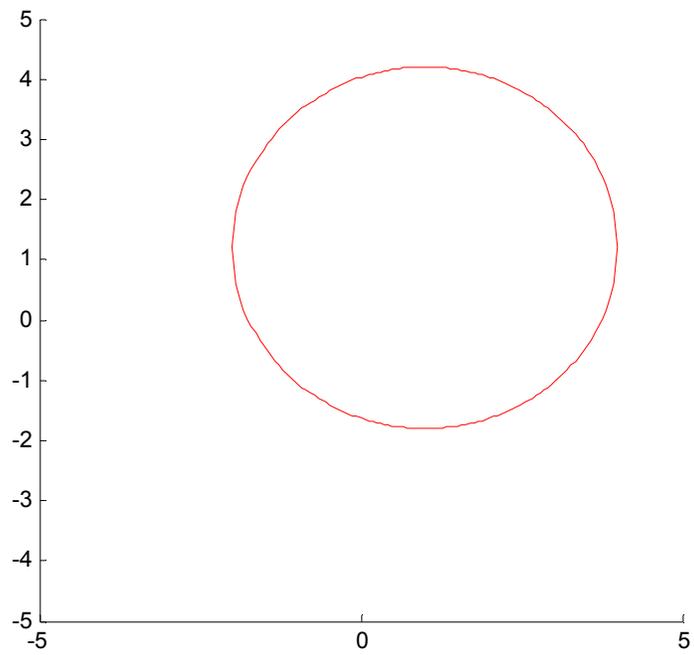
Obstacle Set 1: $\{\mathbf{O}\} = \{ \{(0.5, -0.5), 1\}, \{-4.0, -2.5\}, 0.5\}, \{(4.0, 3.0), 0.5\} \}$



Obstacle Set 2: $\{\mathbf{O}\} = \{ \{(1.5, 2.5), 2.0\}, \{-4.2, -3.5\}, 0.5\}, \{-3.2, -4.0\}, 0.5\} \}$

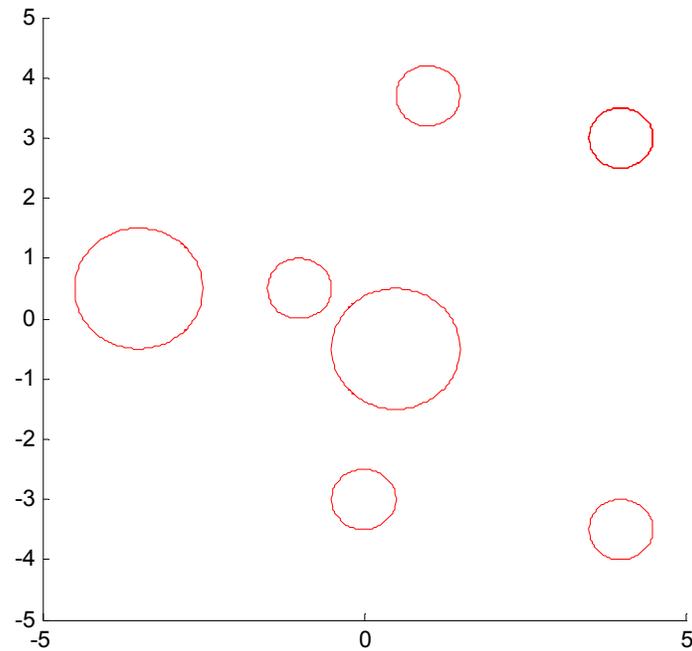


Obstacle Set 3: $\{\mathbf{O}\} = \{ \{(1, 1.2), 3\} \}$



Obstacle Set 4: $\{\mathbf{O}\} = \{ \{(0.0, -3), 0.5\}, \{(4.0, -3.5), 0.5\}, \{(-3.5, 0.5), 1.0\},$

$\{(-1.0, 0.5), 0.5\}, \{(1.0, 3.7), 0.5\}, \{(4.0, 3.0), 0.5\},$
 $\{(0.5, -0.5), 1.0\}$



Constraint Sets

Constraint Set 1

$$\mathbf{H}^0 = \{max-speed \leq 4.2\}$$

$$\mathbf{S}^0 = \{\text{somewhat quickly}\}$$

Constraint Set 2

$$\mathbf{H}^0 = \{max-acc \leq 1.0, min-sep \geq 1.7\}$$

$$\mathbf{S}^0 = \{\text{safely}\}$$

Constraint Set 3

$$\mathbf{S}^0 = \{\text{a little quickly, exceedingly inquisitively}\}$$

Constraint Set 4

$$\mathbf{H}^0 = \{max-acc \leq 1.0, max-speed \geq 4.0\}$$

$$S^0 = \{10 \leq \text{energy} \leq 15, 1.0 \leq \text{avg-speed} \leq 2.0\}$$

Constraint Set 5

$$H^0 = \{\max \text{acc} \leq 1.0, \max\text{-speed} \leq 4.0\}$$

$$S^0 = \{10 \leq \text{energy} \leq 15, 1.0 \leq \text{avg-speed} \leq 2.0, \text{moderately safely}\}$$

Margins of Success and Failure

Green cells represent margins of success. Margins are normalized as described in Chapter 4. For hard constraints (all uppers in these runs), a positive sign indicates how far under the limit the value is. For soft constraints, positive sign indicates that the margin normalized to the high side of the center of the fuzzy range and negative sign indicates that it normalized to the low side.

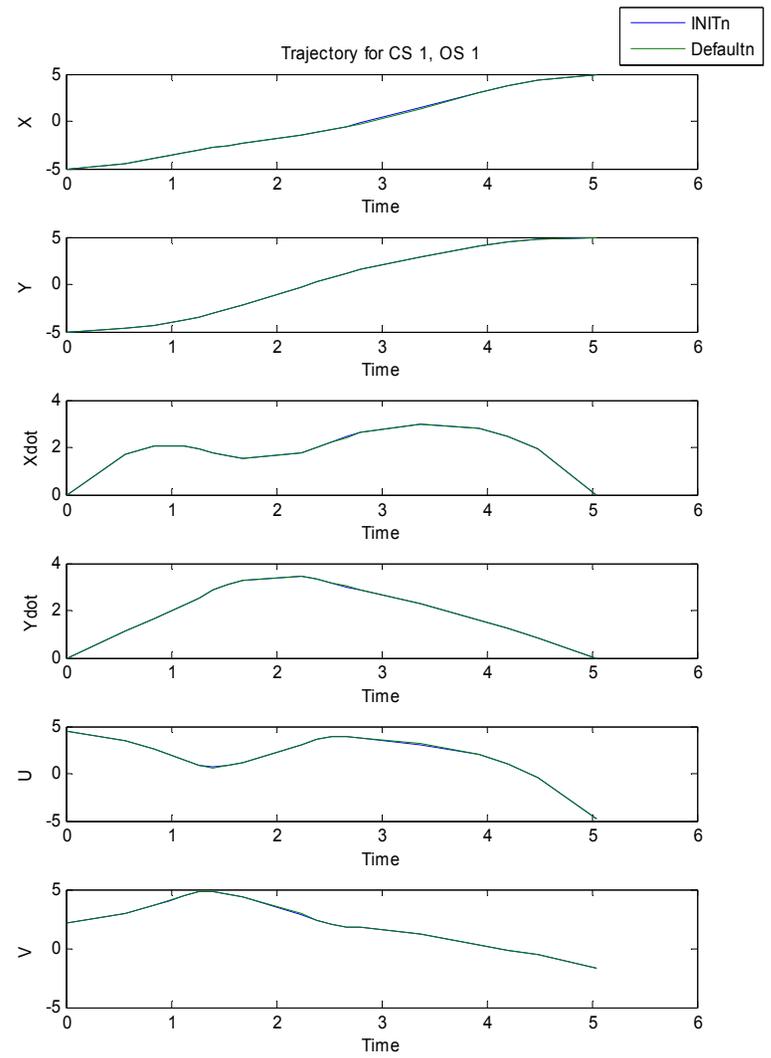
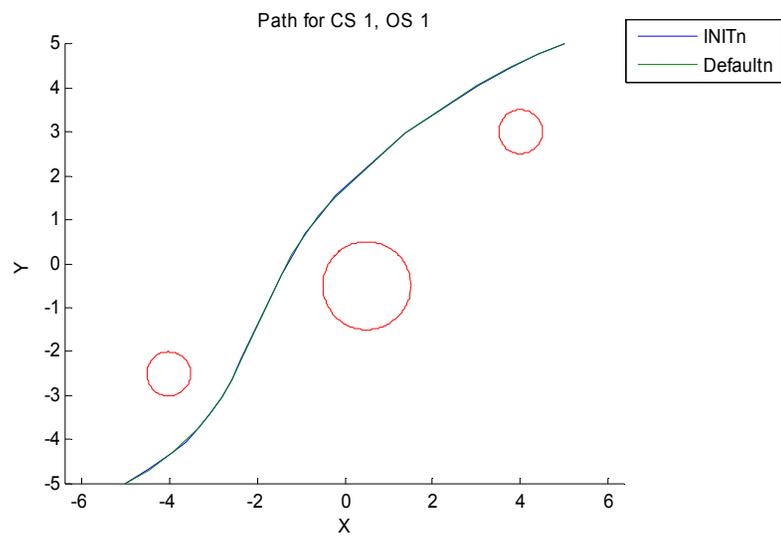
Pink cells represent margins of failure. For hard failures, signs are positive and indicate how far over the limit the failure went. For soft range failures, a positive sign indicates that the upper limit was exceeded (“failed high”) while a negative sign indicates that the lower limit was exceeded (“failed low”)

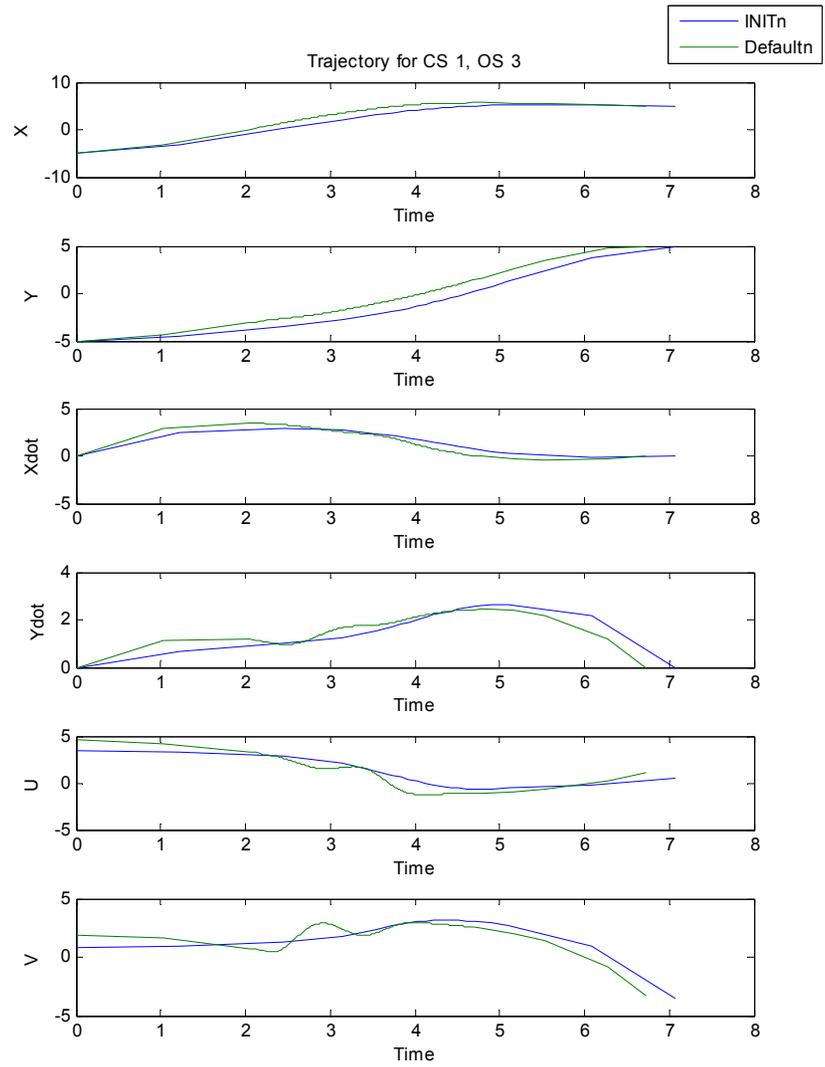
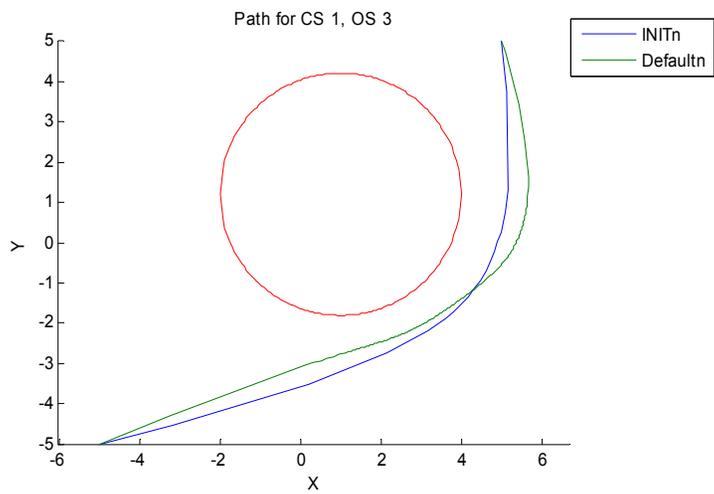
	Margins															
	Obstacle Set 1				Obstacle Set 2				Obstacle Set 3				Obstacle Set 4			
	INIT0	INITn	Default0	Defaultn												
Constraint Set 1																
max speed <= 4.2	0.37	0.08	0.67	0.08	0.10	0.60	0.08	0.75	0.76	0.27	0.87	0.13	0.57	0.04	0.29	0.23
somewhat quickly																
hi 4 <= max speed <= 8	-0.68	-0.07	-1.32	-0.06	-0.69	-1.15	-0.06	-1.49	-1.50	-0.47	-1.74	-0.18	-1.41	-0.98	-0.50	-0.39
hi 2.67 <= avg speed <= 5.33	-0.44	-0.81	-1.14	-0.81	-0.62	-0.95	-0.93	-1.39	-1.36	-0.20	-1.62	-0.77	-0.80	-0.50	-0.53	-0.04
Constraint Set 2																
max acc <= 1	0.25	0.25	0.41	0.25	0.78	0.04	3.78	0.29	0.28	0.28	0.47	0.39	0.08	0.08	0.50	0.44
min sep >= 1.7	0.02	0.02	0.51	0.02	0.00	0.00	0.43	0.00	0.48	0.48	0.44	0.48	0.02	0.02	0.95	0.02
safely																
hi 3 <= min sep <= 5	-0.26	-0.26	-1.16	0.26	-0.80	-0.80	-1.53	-0.80	-0.48	-0.48	-1.04	-0.48	-0.26	-0.26	-1.91	
lo 0.5 <= max speed <= 1.5	-0.72	-0.72	0.74	-0.50	-0.56	-0.92	4.76	-0.88	-0.56	-0.56	-0.94	-0.86	-0.42	-0.42	3.00	-0.69
lo 0.33 <= avg speed <= 1	-0.36	-0.36	0.45	-0.18	-0.18	-0.57	5.25	-0.48	0.00	0.00	-0.45	-0.96	0.36	0.36	2.84	-0.33
lo 0.33 <= max acc <= 1	0.27	0.27	-1.22	0.27	2.33	0.87	11.25	-0.12	0.15	0.15	-0.42	-0.15	0.75	0.75	1.49	-0.32
Constraint Set 3																
a little quickly																
hi 4 <= max speed <= 8	-1.29	-0.74	-1.32	-0.06	-1.63	-1.48	-0.06	-0.89	-1.34	-1.28	-1.74	-0.31	-1.02	0.14	-0.50	-1.05
hi 2.67 <= avg speed <= 5.33	-1.09	-0.44	-1.14	-0.81	-1.58	-1.30	-0.93	-0.83	-1.09	-1.20	-1.62	-0.97	-1.18	0.14	-0.53	-0.90
exceedingly inquisitively																
medlo 0.67 <= avg speed <= 1.33	0.67	6.36	0.45	4.82	-0.30	-0.61	4.33	4.70	0.67	-0.64	-0.45	4.18	0.30	4.58	1.88	0.39
lo 0.3 <= min sep <= 1	-0.60	0.77	0.54	0.49	-0.66	-0.20	-0.51	-0.20	-0.54	0.20	0.89	0.29	-0.46	-0.60	-0.60	2.11
Constraint Set 4																
max acc <= 1	0.41	0.04	0.41	0.04	3.78	0.46	3.78	0.46	0.47	0.04	0.47	0.04	0.50	0.05	0.50	0.05
max speed <= 4	0.66	0.76	0.66	0.76	0.03	0.87	0.03	0.87	0.87	0.76	0.87	0.76	0.25	0.46	0.25	0.46
soft 10 <= energy <= 15	0.23	0.10	0.23	0.10	1.62	0.37	1.62	0.37	0.84	0.84	0.84	-0.05	0.14	0.98	0.14	0.98
soft 1 <= avg speed <= 2	-0.70	-0.38	-0.70	-0.38	1.52	-1.06	1.52	-1.06	-0.96	-0.22	-0.96	4.18	0.90	-0.28	0.90	-0.28
Constraint Set 5																
max acc <= 1	0.38	0.06	0.41	0.04	2.76	0.01	3.78	0.46	0.06	0.07	0.47	0.03	0.10	0.05	0.50	0.02
max speed <= 4	0.74	0.82	0.66	0.76	0.74	0.85	0.03	0.87	0.74	0.77	0.87	0.76	0.82	0.81	0.25	0.76
soft 10 <= energy <= 15	2.46	2.23	0.23	1.02	2.74	1.62	1.62	0.37	2.08	2.20	0.84	0.84	4.56	4.13	0.14	0.89
soft 1 <= avg speed <= 2	0.32	-0.58	-0.70	-0.38	-0.38	-1.04	1.52	-1.06	-0.12	-0.44	-0.96	-0.44	-1.10	-0.98	0.90	-0.38
moderately safely																
hi 3 <= min sep <= 5	-0.26	-0.26	-1.16	-0.35	-0.80	-0.80	-1.53	-1.28	-0.48	-0.48	-1.03	-1.04	-0.26	-0.26	-1.91	-2.28
lo 0.5 <= max speed <= 1.5	0.12	-0.54	0.74	-0.08	0.08	-0.80	4.76	-0.92	0.12	-0.16	-0.94	-0.08	-0.58	-0.46	3.00	-0.06
lo 0.33 <= avg speed <= 1	0.51	0.12	0.45	0.42	0.51	-0.57	5.25	-0.57	0.18	0.36	-0.45	0.36	-0.66	-0.45	2.84	0.45
lo 0.33 <= max acc <= 1	1.13	0.84	1.22	0.90	8.24	0.99	11.25	-0.36	0.81	0.78	-0.42	0.93	0.69	0.87	1.49	0.93

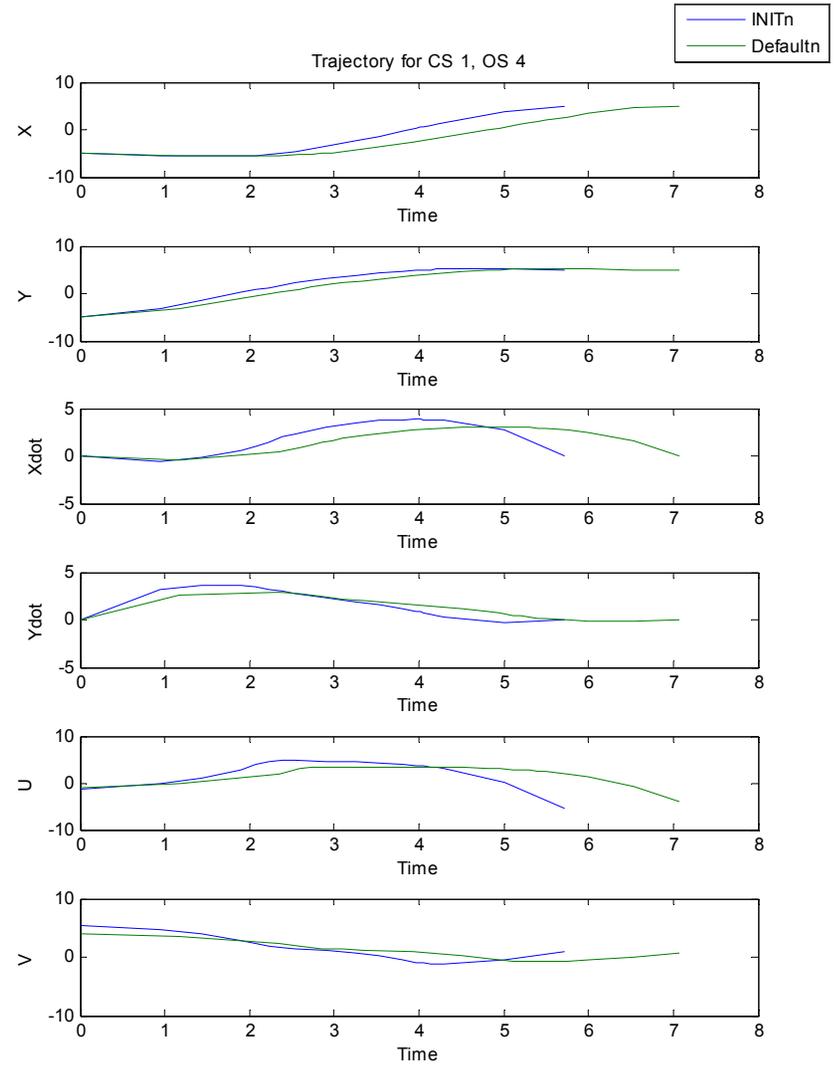
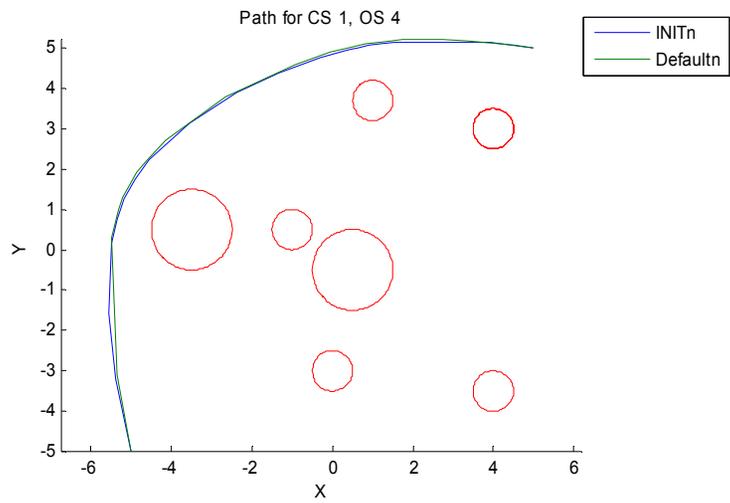
2DOF Trajectory Data by Constraint Set

We append here charts showing the paths and trajectories of all 40 2DOF test cases. Each chart displays the *Default*ⁿ case in green and the *INIT*ⁿ case in blue. Each constraint set and obstacle set solution is represented by two charts. The first shows the path that the agent took through the obstacle field. The other shows trajectory information over time. The trajectory information charts show the x -coordinate (X), y -coordinate (Y), x -velocity ($Xdot$), y -velocity ($Ydot$), x -input (U) and y -input (V).

Each chart is labeled using “CS” to denote “Constraint Set” and “OS” to denote “Obstacle Set.” So “CS1, OS1” is the information for Constraint Set 1, Obstacle Set 1.







Constraint Set 1 included $\mathbf{H}^0 = \{max-speed \leq 4.2\}$, $\mathbf{S}^0 = \{\text{somewhat quickly}\}$. The soft constraints were extended to $\mathbf{S}^0 = \{4 \text{ m/s} \leq max-speed \leq 8 \text{ m/s}, 2.67 \text{ m/s} \leq avg-speed \leq 5.33 \text{ m/s}\}$. This constraint set was difficult because of the small window on *max-speed* that would satisfy both \mathbf{H}^0 and \mathbf{S}^0 . The \mathbf{H}^0 was mostly made by very small margins; for instance, Constraint Set 1, Obstacle Set 1 had a \mathbf{H}^0 margin of 0.08, which means that the actual *max-speed* was 3.864 m/s. This was still lower than the lower limit on the *max-speed* soft constraint, so that returned a soft failure. An examination of the \mathbf{H}^0 margins in the solution histories shows a see-saw pattern with the \mathbf{H}^0 margins shrinking overall. At first, the \mathbf{H}^0 was met but the \mathbf{S}^0 was not; they fail low. The time weight was raised to try to increase the *max-speed* and *avg-speed* to meet the \mathbf{S}^0 . This typically failed the \mathbf{H}^0 . The time weight was lowered again, but not as low as it had been initially. This traded off some of the margin on \mathbf{H}^0 to get closer to \mathbf{S}^0 . Let \mathbf{S}^0_1 be the soft constraint on *max-speed* and \mathbf{S}^0_2 be the soft constraint on *avg-speed*. Then the following results are for the *INITⁿ* runs, showing the \mathbf{H}^0 margins for each iteration and the final success/failure conditions for \mathbf{S}^0 :

CS1, OS1: {0.369195, 0.077382} (failed \mathbf{S}^0_1 , succeeded \mathbf{S}^0_2)

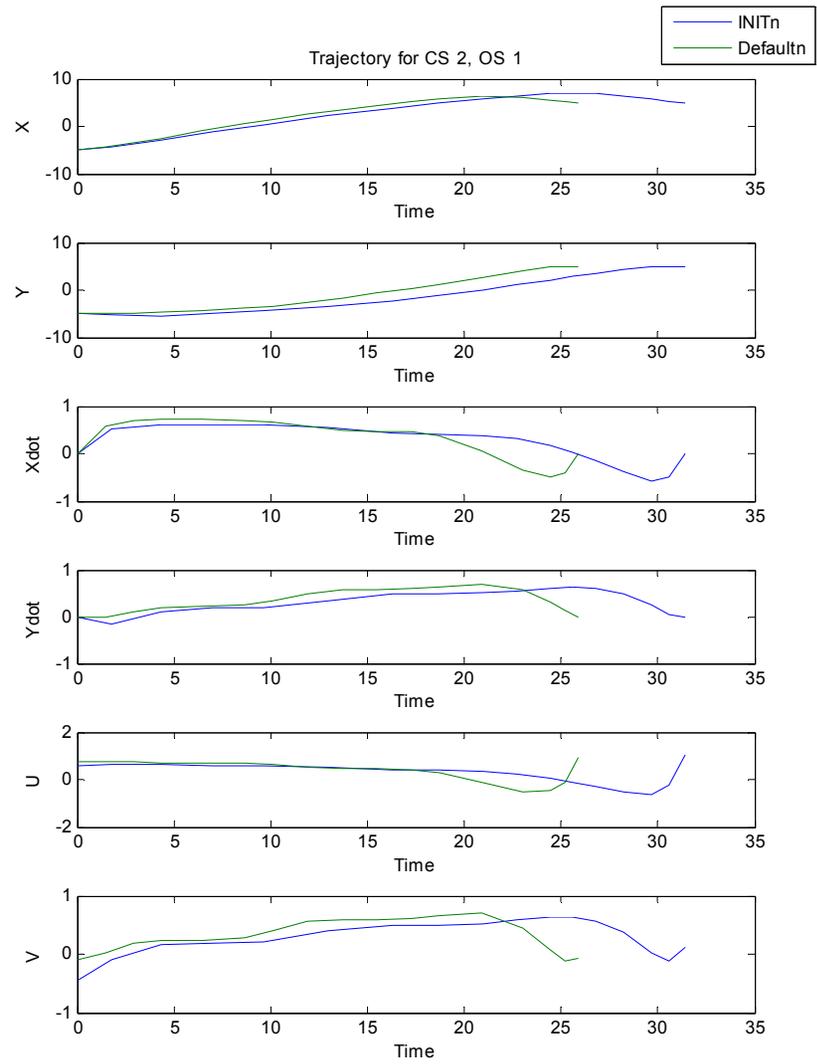
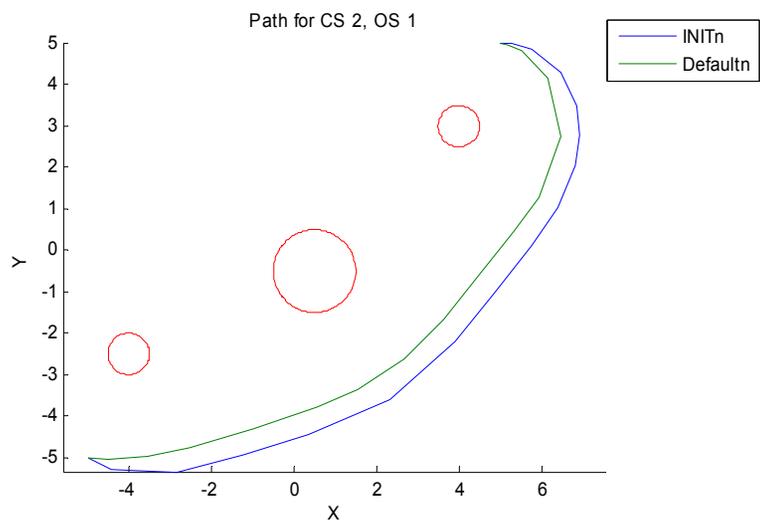
CS1, OS2: {0.102730 (failed), 0.702101, 0.615724 (failed), 0.734784, 0.596354}
(failed \mathbf{S}^0_1 , \mathbf{S}^0_2)

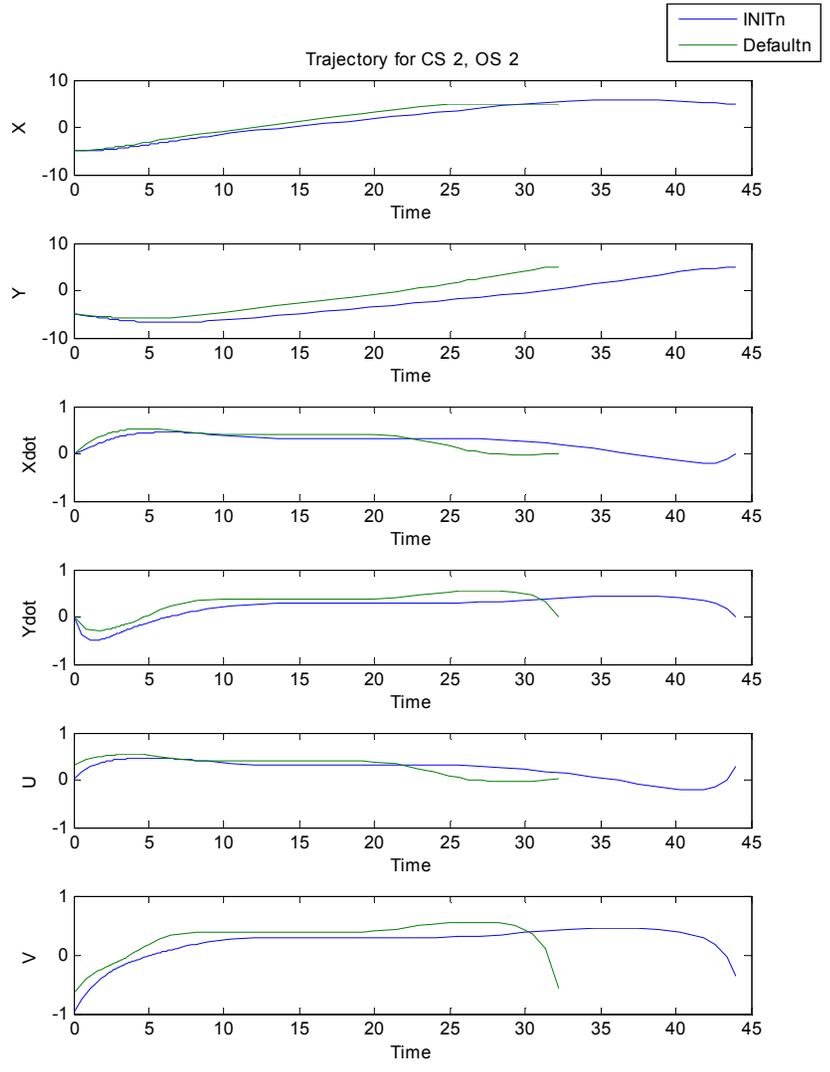
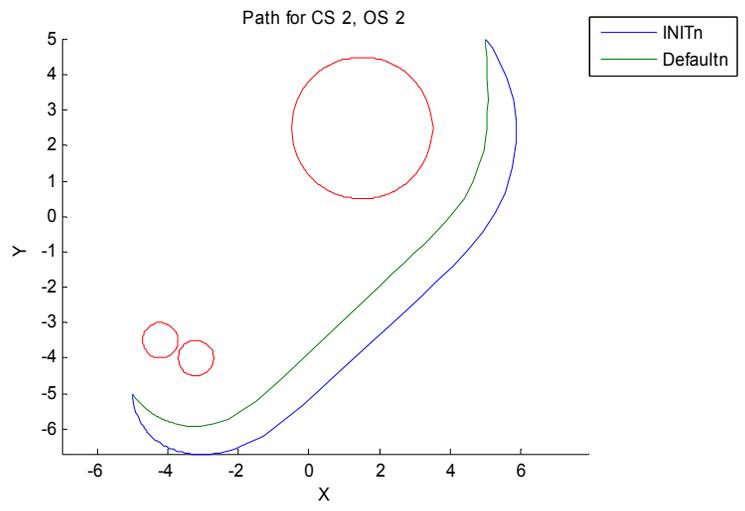
CS1, OS3: {0.762050, 0.460474 (failed), 0.269521} (failed \mathbf{S}^0_1 , \mathbf{S}^0_2)

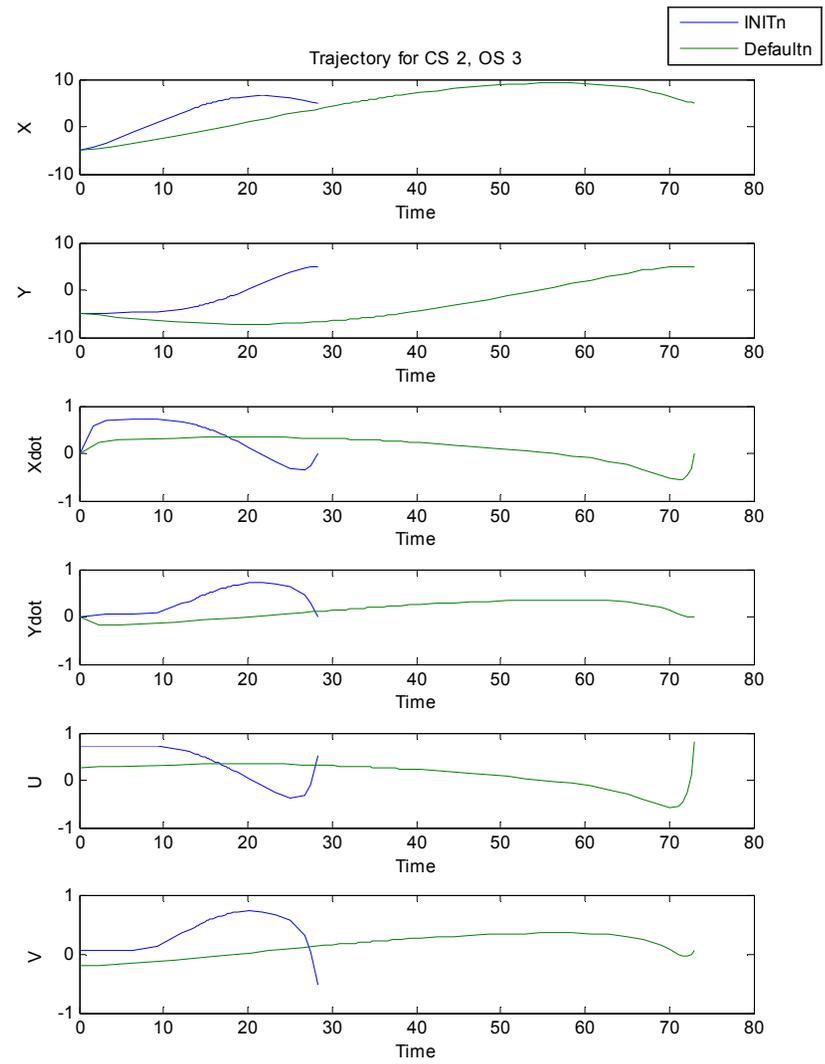
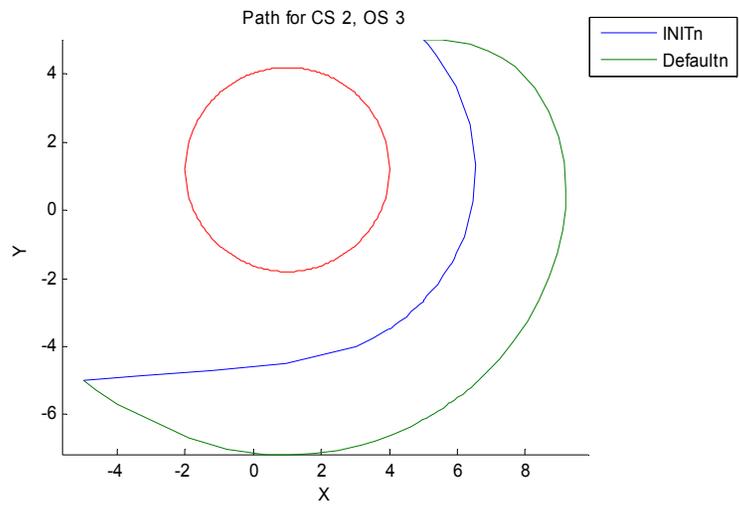
CS1, OS4: {0.566147, 0.284714 (failed), 0.034656} (succeeded \mathbf{S}^0_1 , \mathbf{S}^0_2)

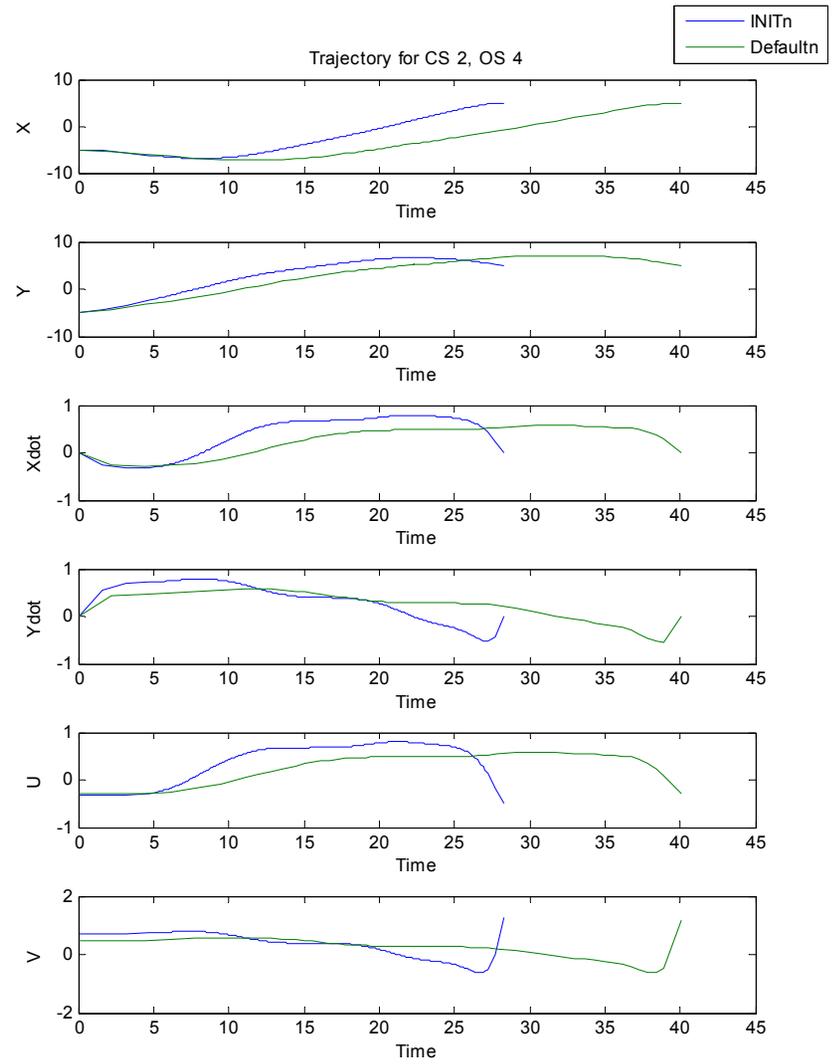
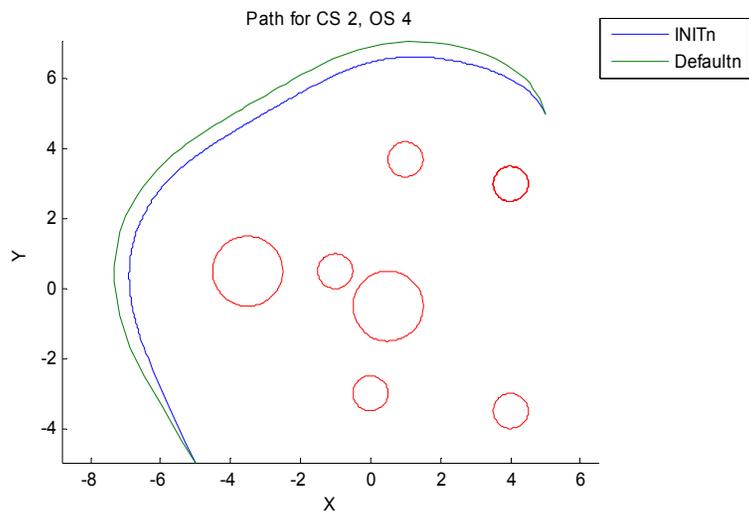
Obstacle Set 2 apparently presented some difficulty; the solution required more iterations from both *Defaultⁿ* and *INITⁿ*, and it was longer (in traversal time) than the other trajectories. For the other obstacle sets, the trajectory was traversed in ~5 – 7 seconds.

For Obstacle Set 2, it took *INIT*'s solution ~12 seconds and *Default*'s just over 20 seconds. These runs did not show more computational problems than the others. This seems to indicate that the trajectory planner found a local minimum that represented a different solution mode than the other three.







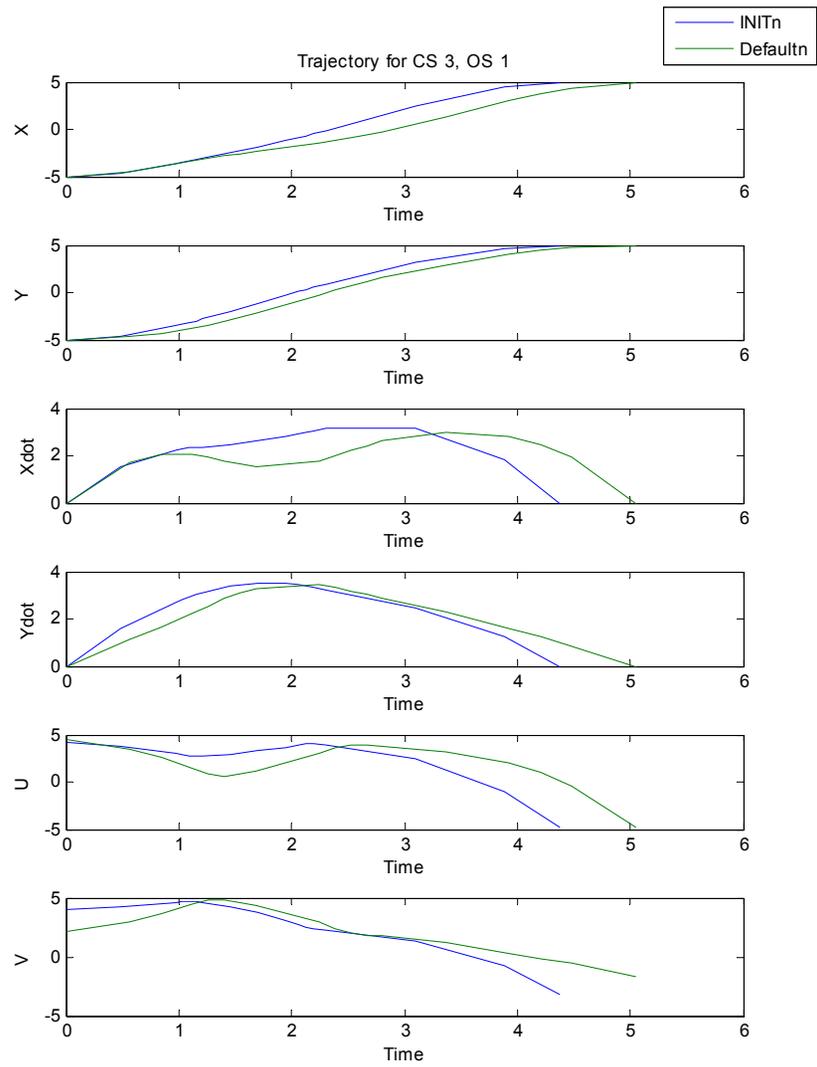
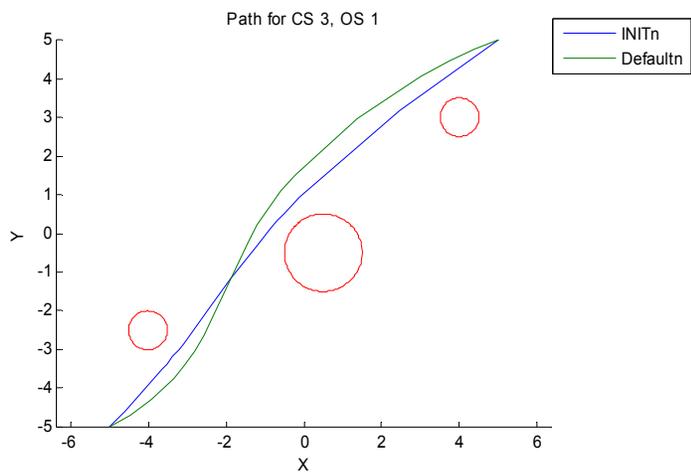


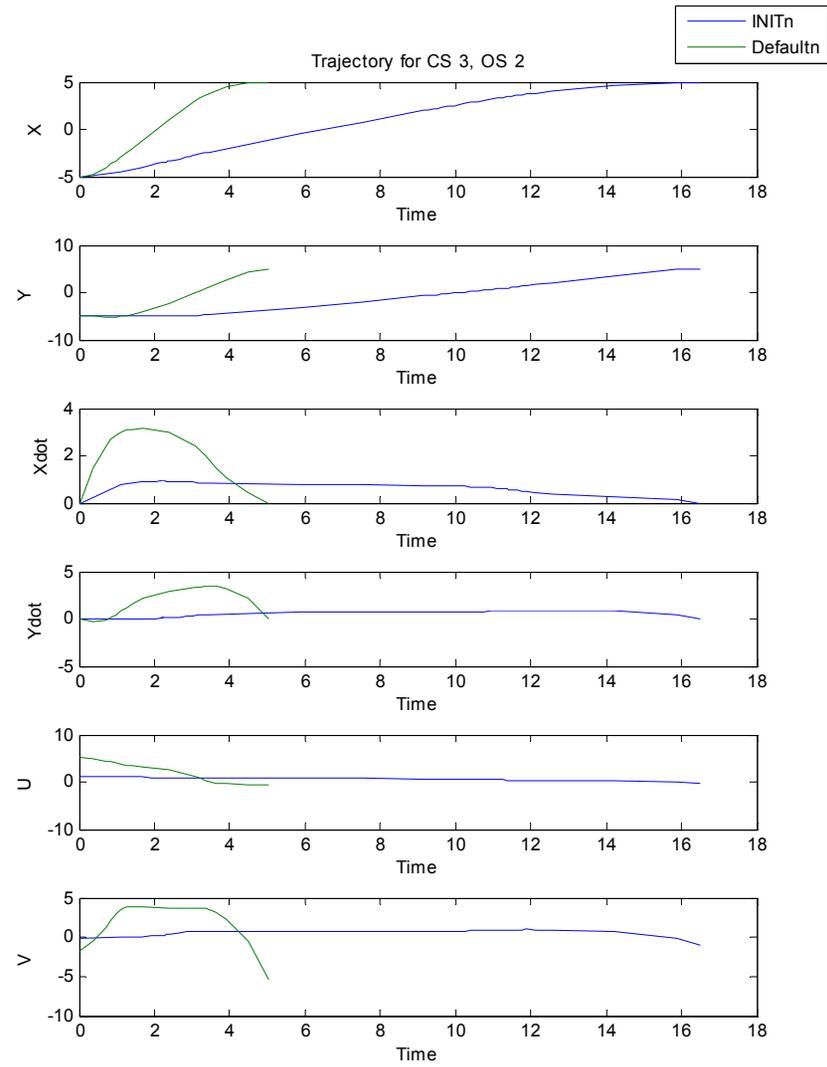
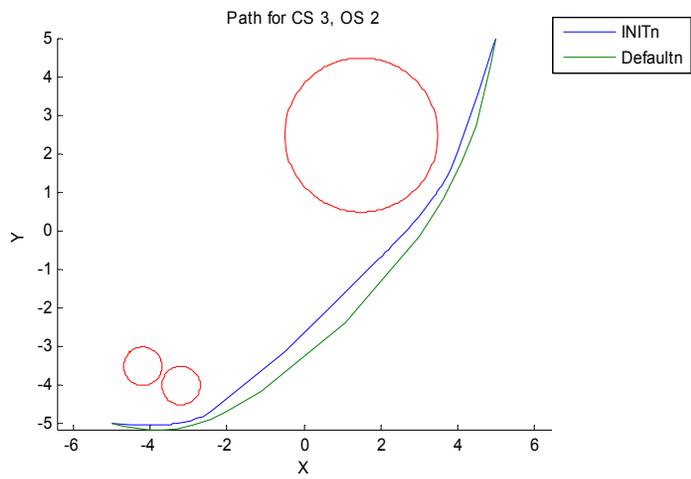
Constraint Set 2 had $\mathbf{H}^0 = \{max-acc \leq 1.0, \ min-sep \geq 1.7\}$, $\mathbf{S}^0 = \{\text{safely}\}$.

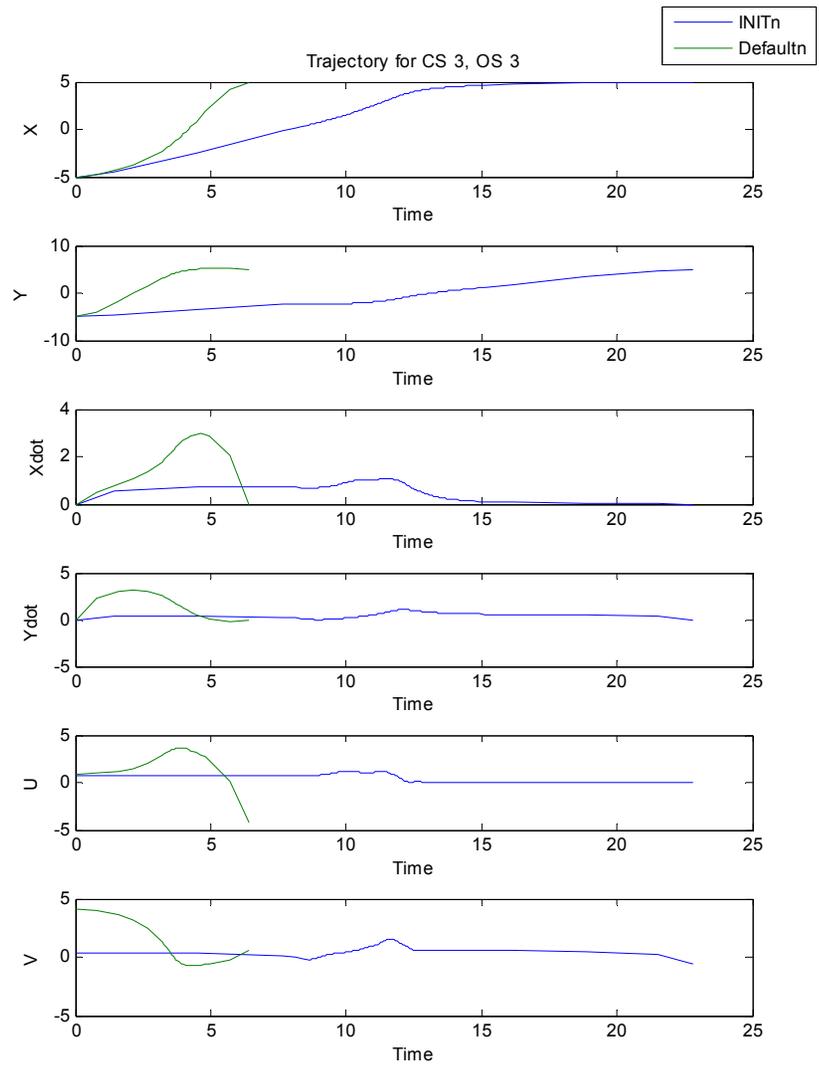
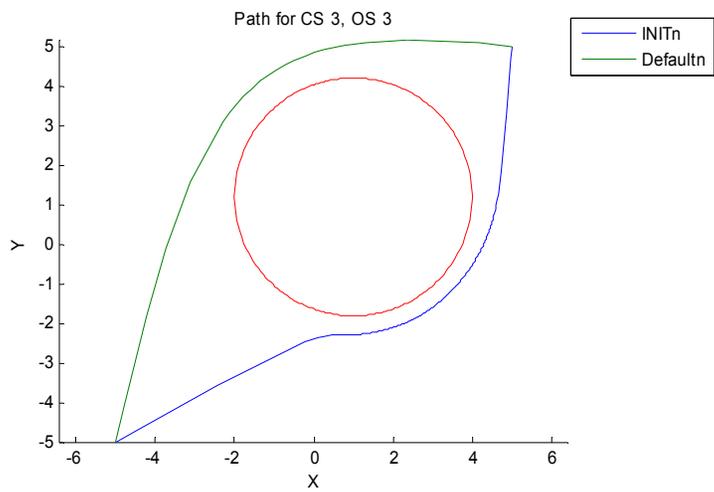
“Safely” expanded into soft range constraints on *min-sep*, *max-speed*, *avg-speed* and *max-acc*. Due to a bug, *EVAL* did not record failed soft *min-sep* limits as failures. (Hard *min-sep* failures and other soft failures were all detected; the line of code to detect this particular kind of error was just missing and not discovered until data post-processing.) While these failures were counted in our analysis, it does mean that the routine exited early in some cases, where if it had continued, it might have found a solution. That is, however, doubtful; the soft *min-sep* range was on the order of a quarter to half of the field. In all but the single-obstacle case, it would be very impractical to get sufficiently far away from the obstacles to not trigger that soft failure. This is a case where perhaps our *ad hoc* definitions were a bit off; a “high *min-sep*” required by “safely” might have been set to smaller actual values.

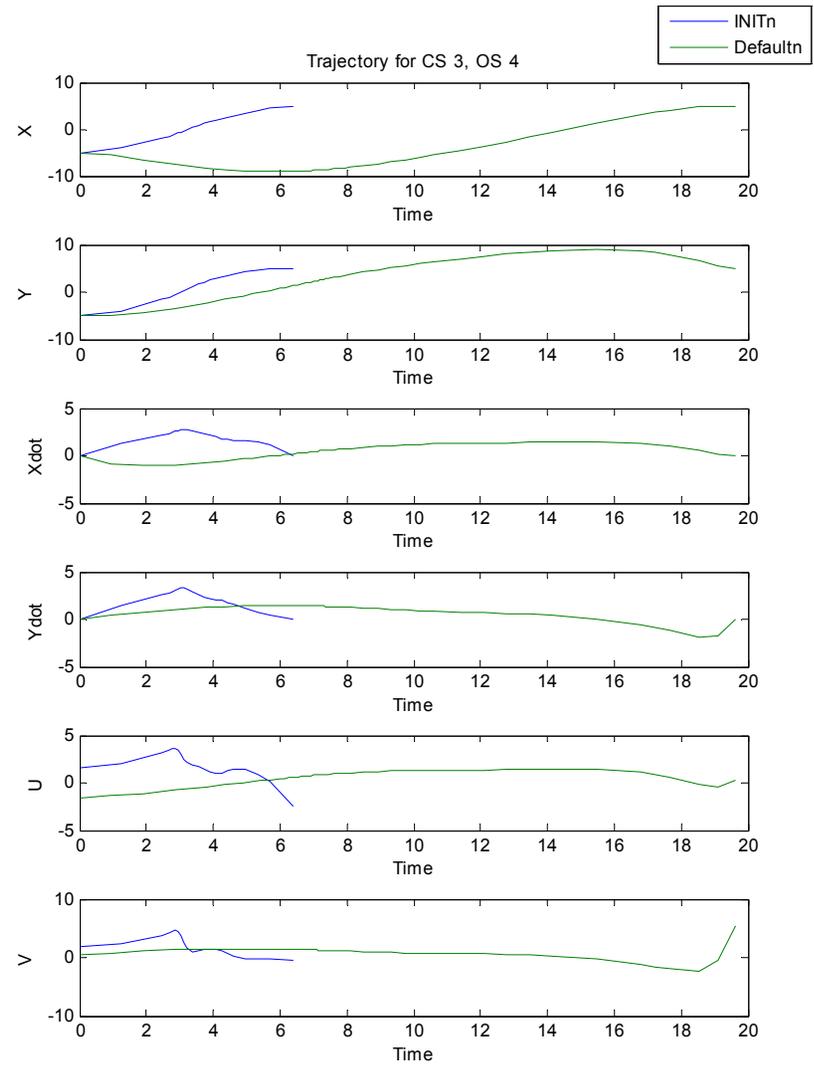
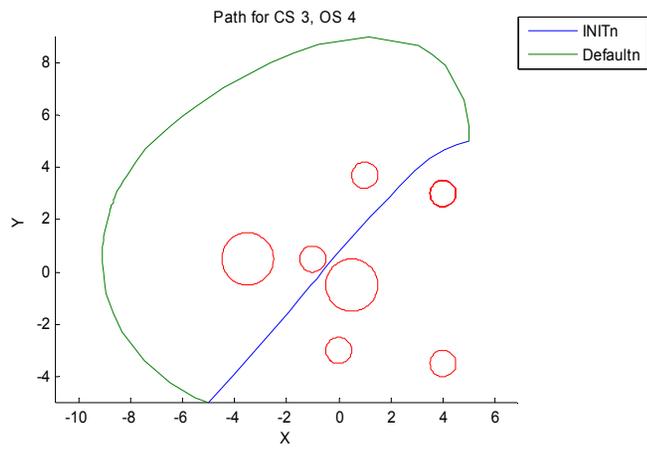
*INIT*ⁿ was particularly successful with this constraint set; three out of four runs ended after one iteration (all with the soft *min-sep* failure which went undetected by *EVAL*). They all took about 30 seconds (\pm about 5 seconds) to traverse their respective obstacle fields. Obstacle Set 2, which had a W_1/W_2 weight ratio almost 4 times that of the others (17.84 versus 4.67) took longer. Obstacle Set 2 was having a very hard time meeting the *max-acc* hard limit, and kept raising W_1/W_2 until it was met. *Default*ⁿ did not have this problem, ironically, because it started with a lower *LIM* value. The *INIT* procedure set *LIM* to 3.0 for the initialized runs because of the “safely” constraint. This met the hard *min-sep* limit of 1.7 m with room to spare. The default value for *LIM* was 1.0, which had to be raised in all cases; but in the Obstacle 2 case, it only had to be raised to 1.7 to meet the hard *min-sep* limit exactly. This lower *LIM* meant a shorter, straighter

path that could be traversed with lower accelerations for a much higher W_1/W_2 value (5.1 for *Default*ⁿ in the OS2 case). *Default*ⁿ had a similar problem with OS3. It started off failing both *min-sep* and *max-acc* hard limits, and raised both W_1/W_2 and *LIM* until W_1/W_2 was met. Then it raised *LIM* alone until *min-sep* was met. But W_1/W_2 was so high that the lower limits on the “low *avg-speed*” and “low *max-speed*” elements of “safely” were missed; W_1/W_2 was lowered (which is to say, W_2 the time weight was increased) so that the hard and soft limits were all met.

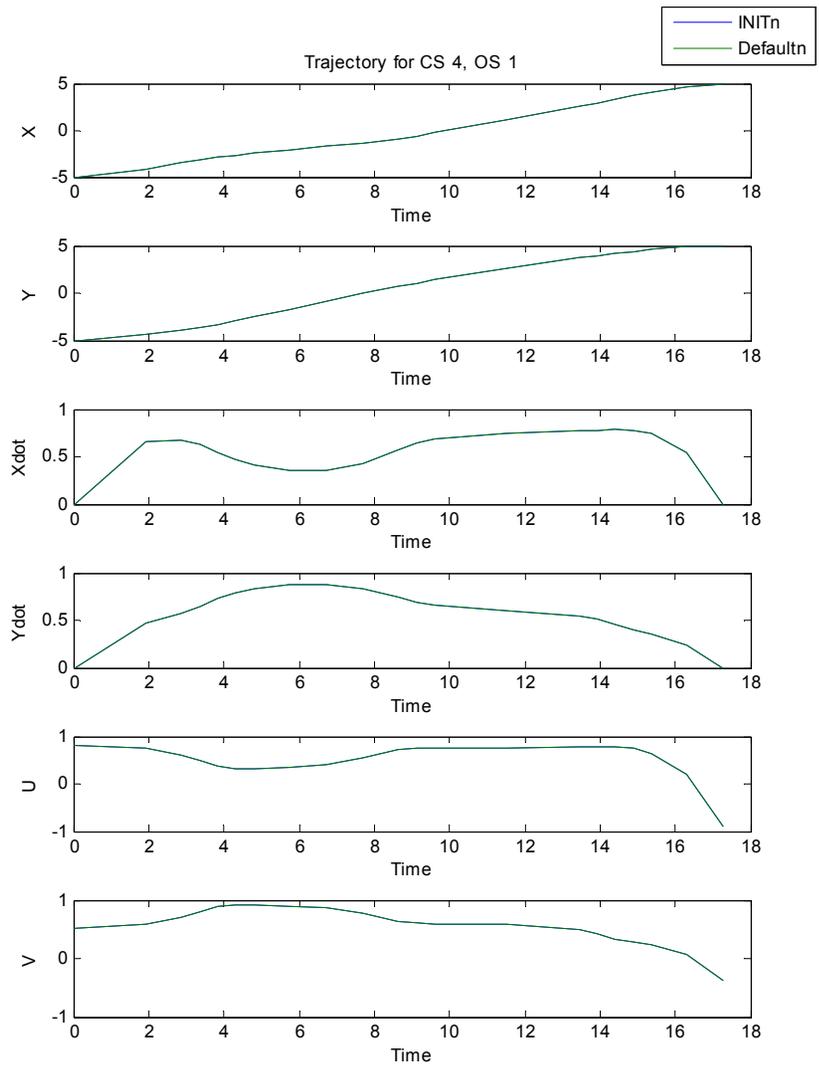
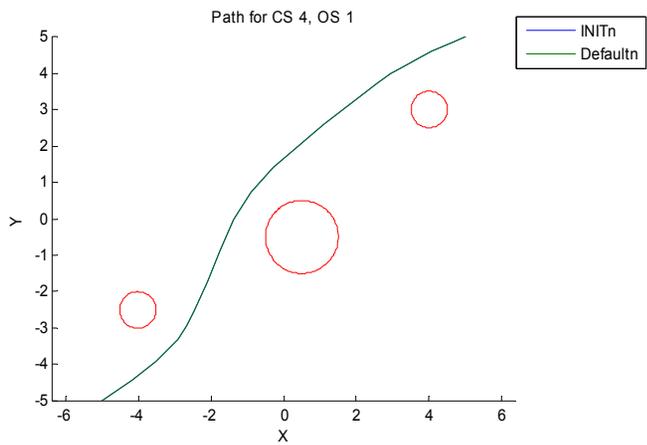


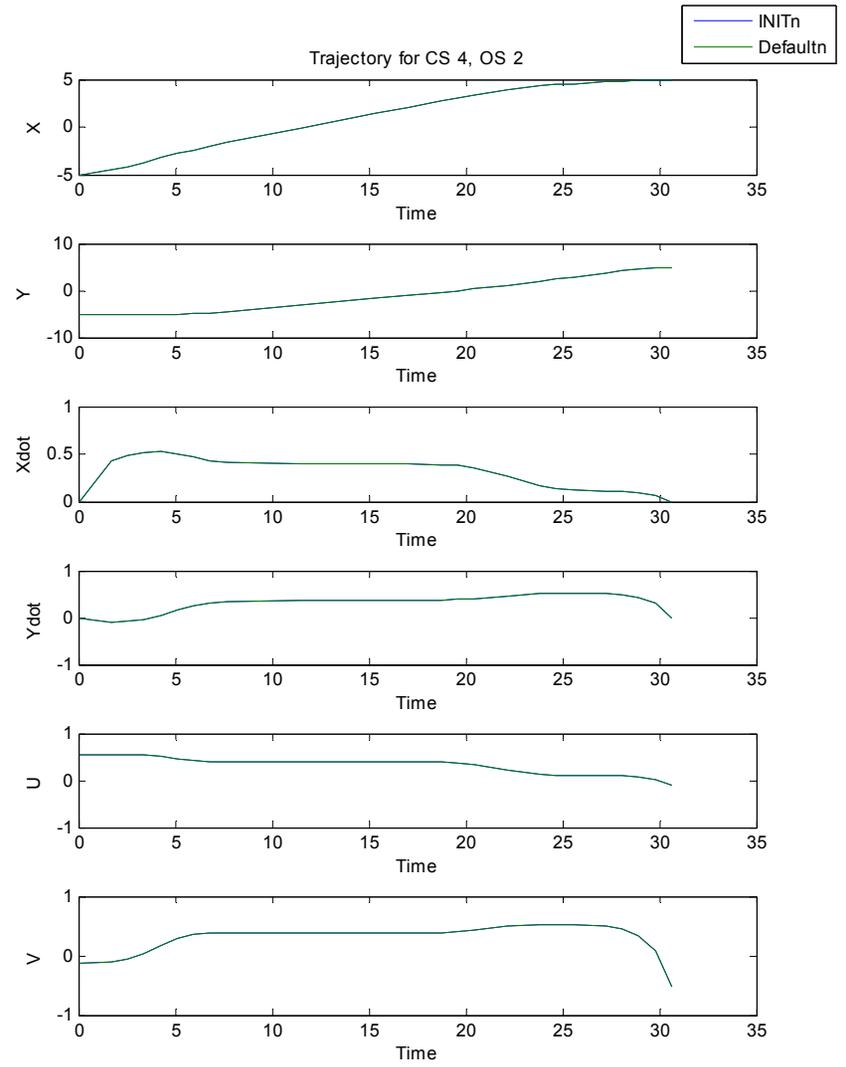
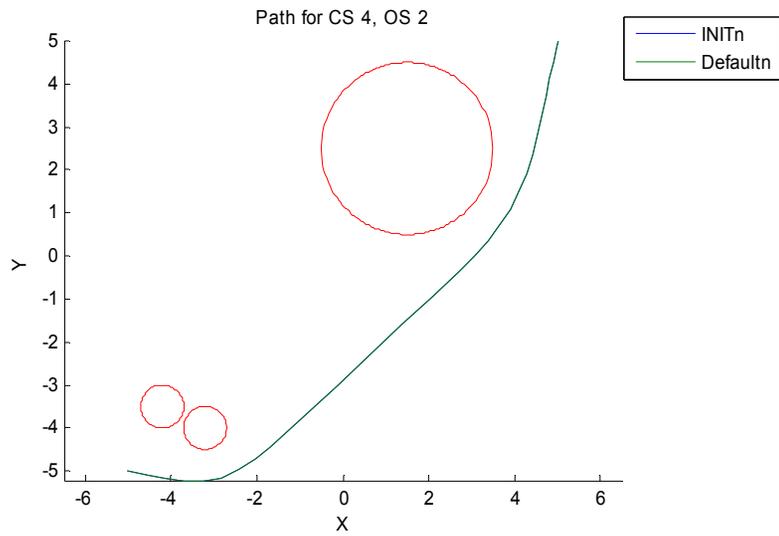


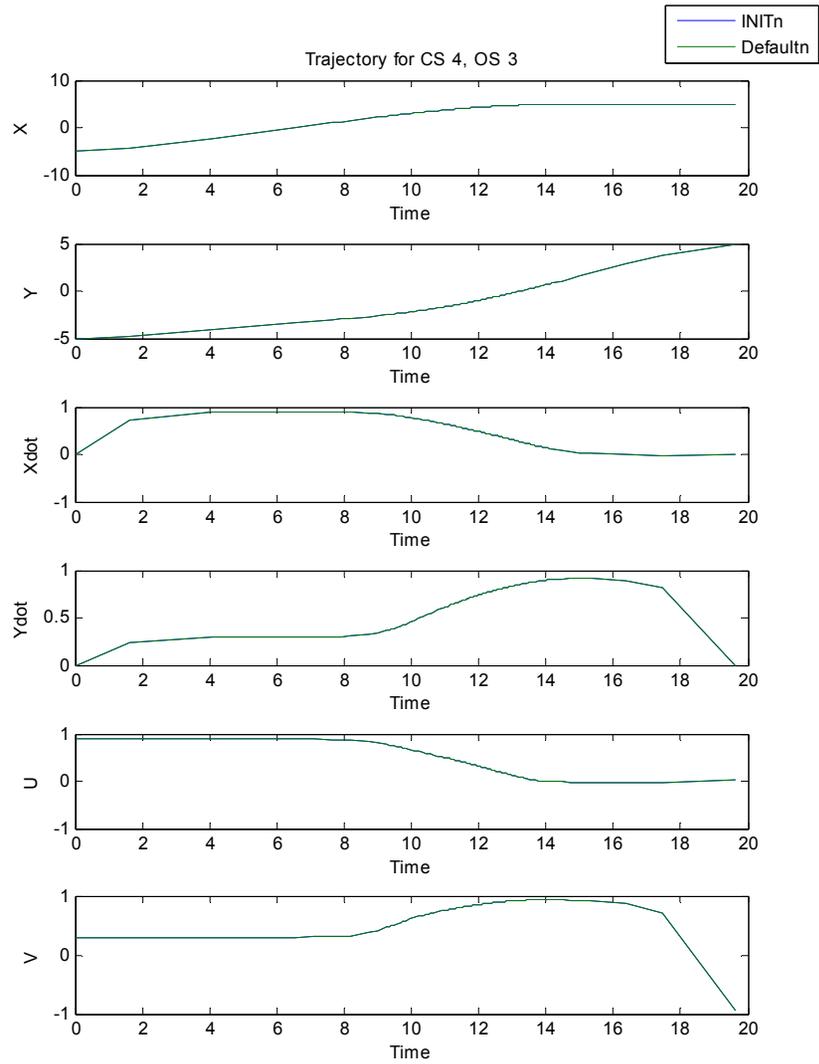
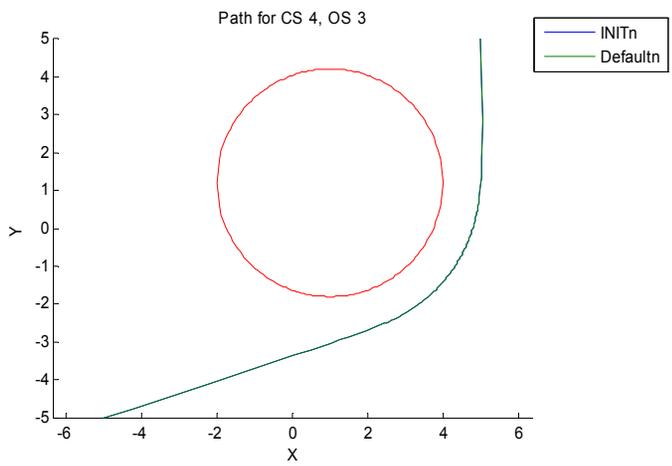


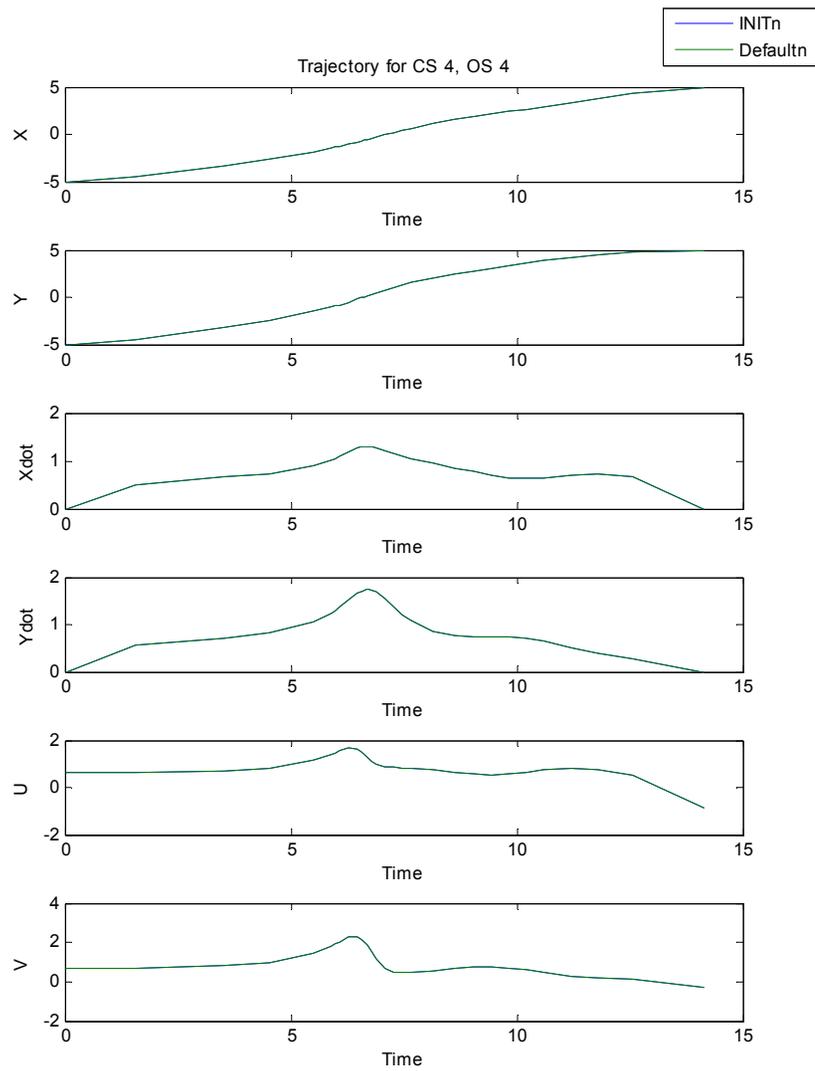
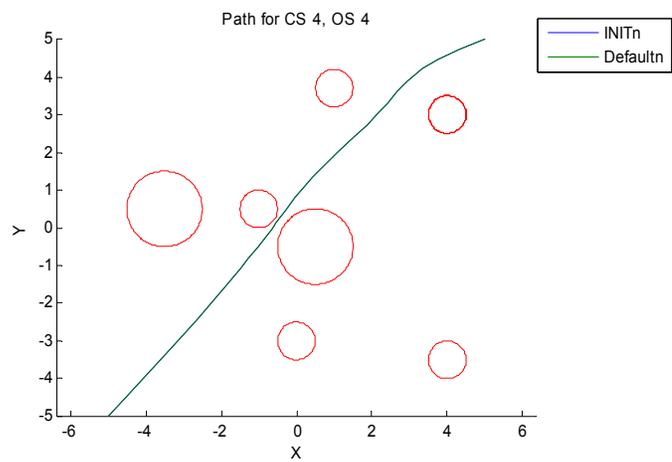


Constraint Set 3 was self-sabotaging. It asked for trajectories with $S^0 = \{\text{a little quickly, exceedingly inquisitively}\}$. These expanded into high *avg-speed* and *max-speed* and medium-low *avg-speed* and low *min-sep*, respectively. It would not be possible to satisfy both high *avg-speed* and medium low *avg-speed*. The idea, of course, was to create a trajectory was that mostly inquisitive but on the fast end of that. For OS 2 and 3, *INIT* seemed to do just that, finding *avg-speeds* that were within the “inquisitively” requirements (although not always on the higher side). OS 1 and 4 seemed to converge to the “quickly” requirements instead. The *Default* trajectories more often made the “quickly” constraints over the “inquisitively” constraints, possibly a result of constraint ordering. This is why they are so much faster than the ones arising from *INIT* for OS 1-3. OS 4 gave *Default* problems as well, as it did not make any of the constraints before timing out of the search procedure. (All of these runs, for *INIT* and for *Default*, exited either with a loop in the weights or with the time limit expired.)



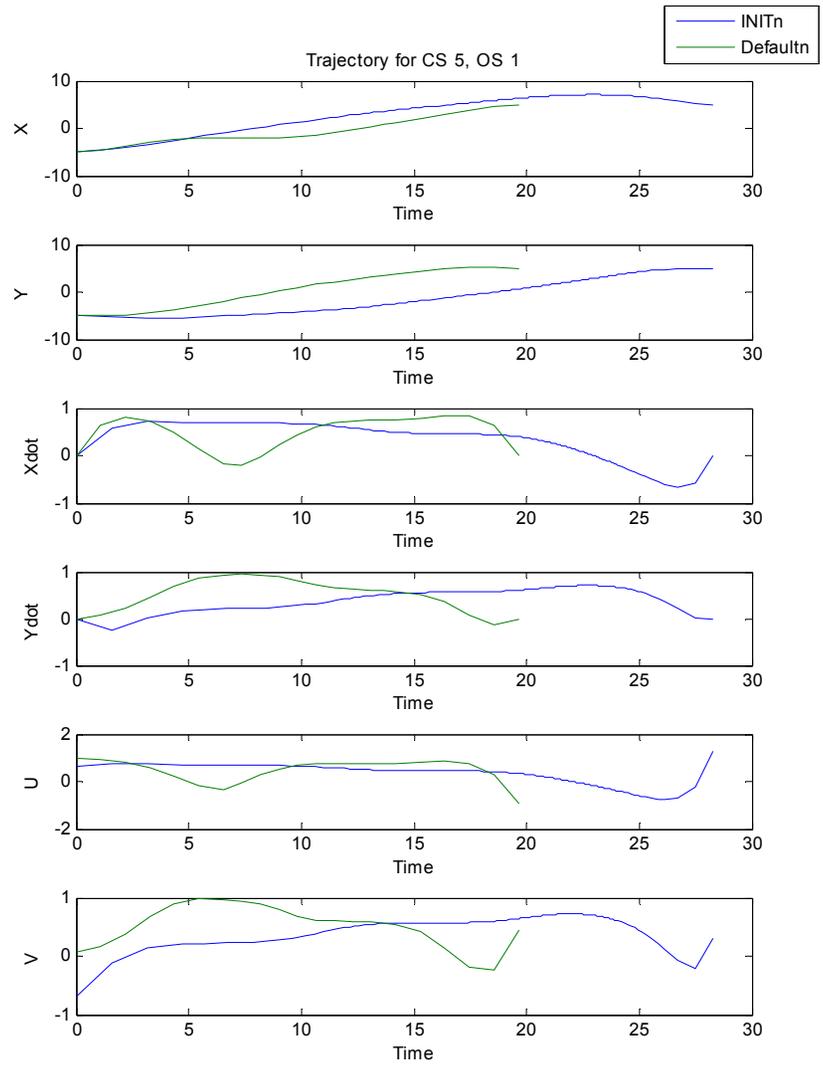
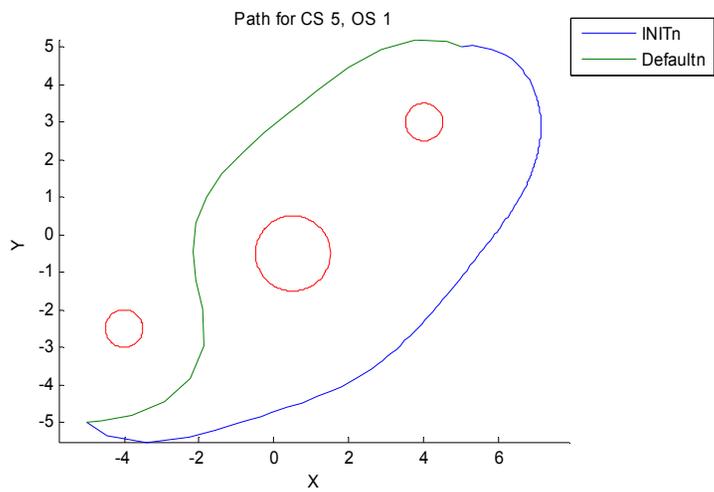


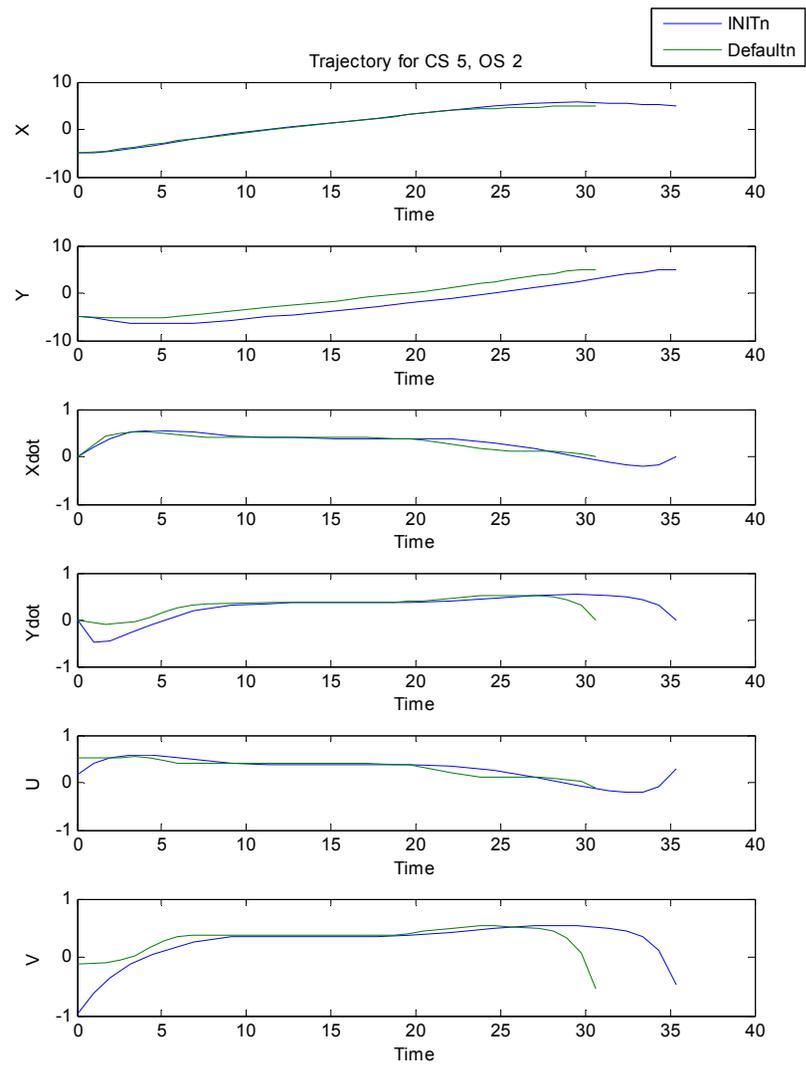
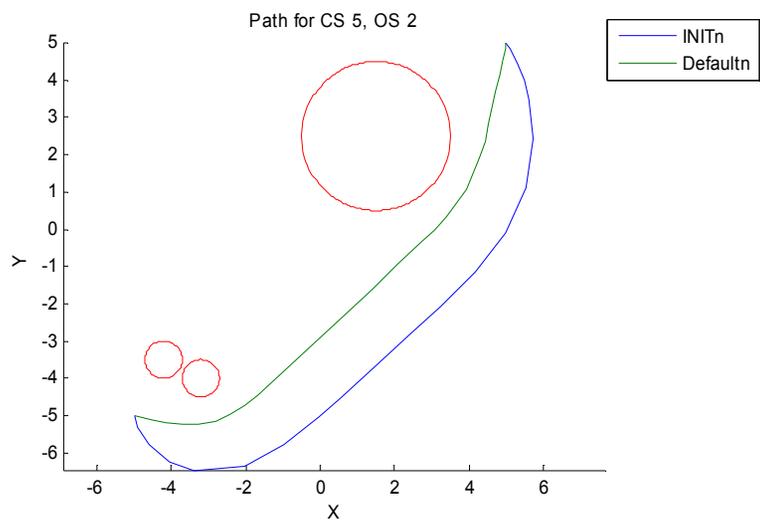


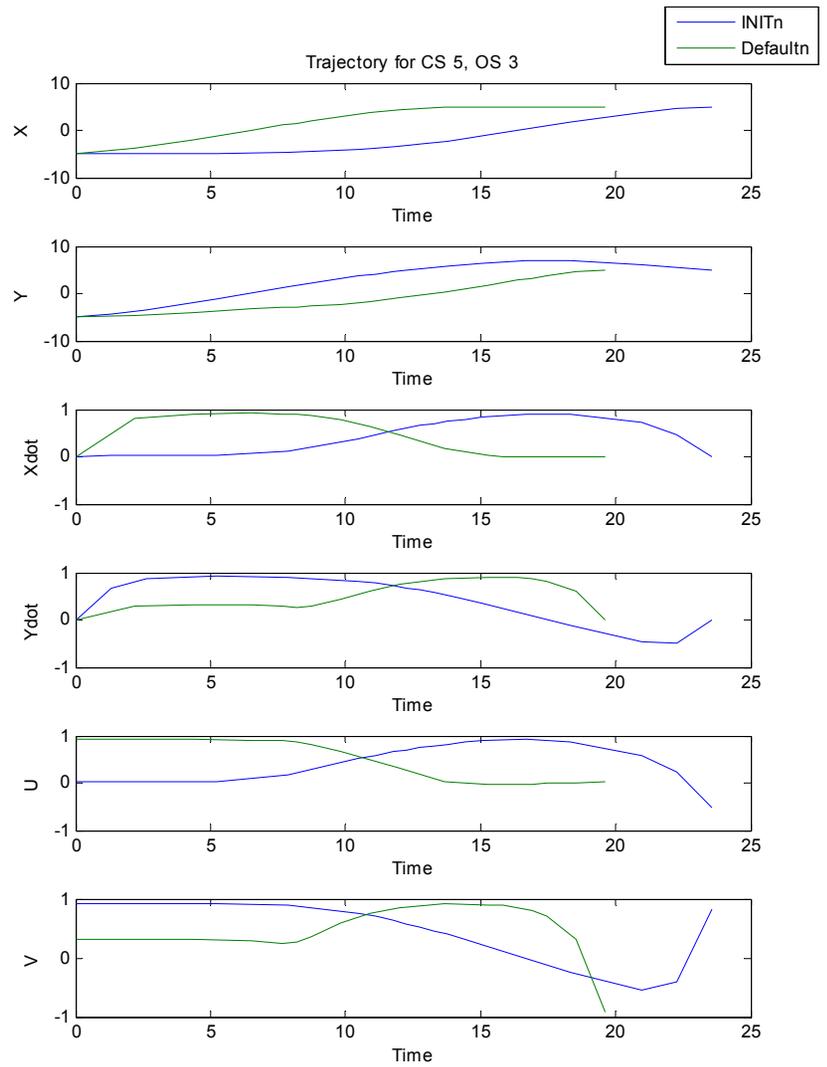
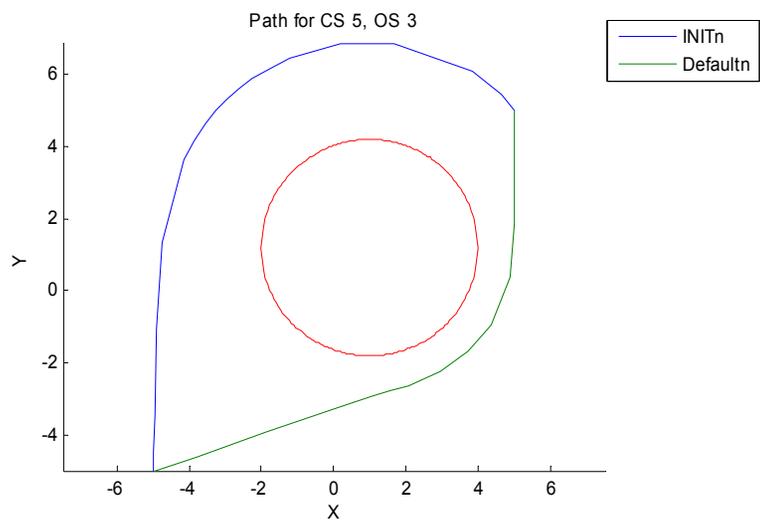


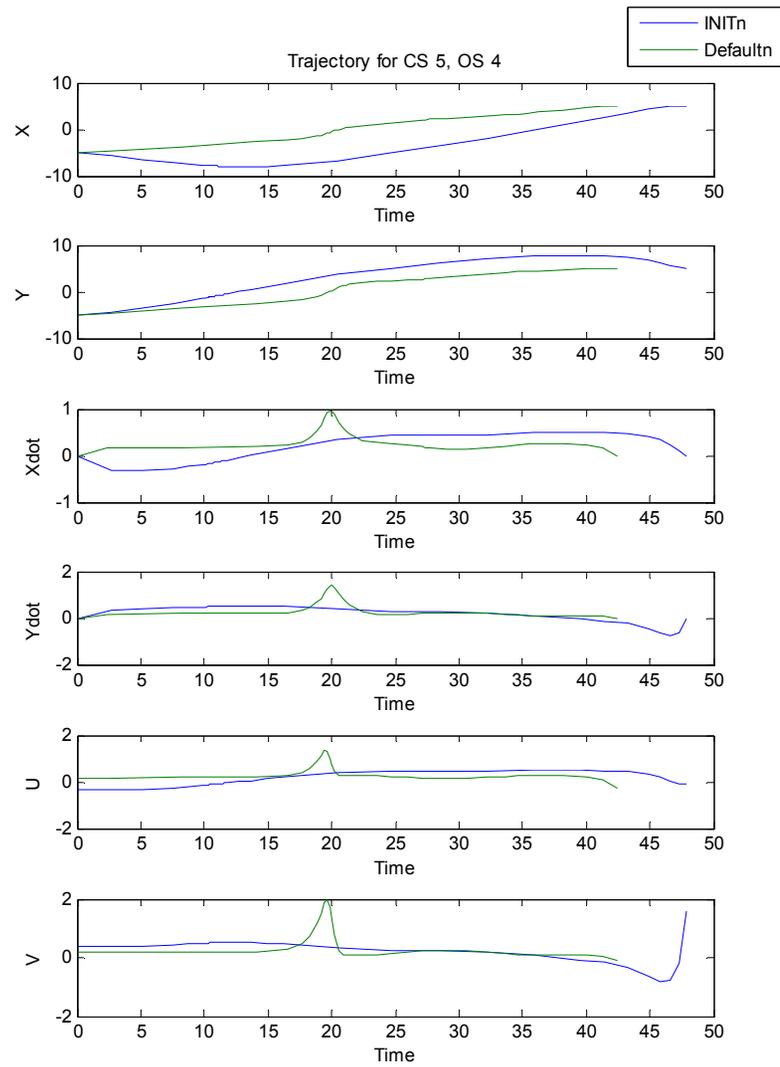
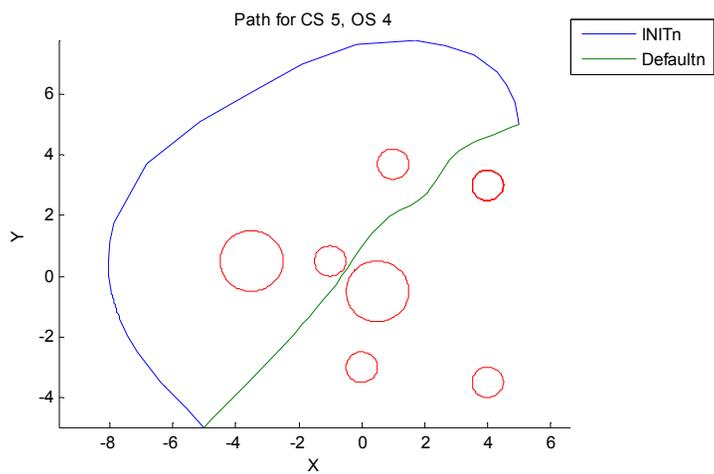
Constraint Set 4 had $\mathbf{H}^0 = \{max-speed \leq 4.0 \text{ m/s}, max-acc \leq 1 \text{ m/s}^2\}$, $\mathbf{S}^0 = \{10 \text{ J} \leq energy \leq 15 \text{ J}, 1 \text{ m/s} \leq avg-speed \leq 2 \text{ m/s}\}$. That low acceleration required a high W_1/W_2 weight, which meant that it was difficult to satisfy the *avg-speed* soft constraint. It was also difficult to meet the *energy* constraint, even with very high W_1/W_2 values (e.g., 22; that is, energy is weighted 22 times more heavily than time). This implied that something may have been off with the weight adjustment heuristic for *energy*. Interestingly, in the OS 3 cases, the W_1/W_2 ratio was adjusted *down*, not up, because the *avg-speed* was failing low. Because of constraint ordering, it was processed after the energy constraint failure and the net adjustment favored it.

The resulting trajectories are identical in these cases because the constraints, when processed by *INIT*, coincidentally happen to return the same value as the *Default* weight vector.









Constraint Set 5 was another difficult set with competing soft constraints. $\mathbf{H}^0 = \{max-acc \leq 1.0, max-speed \leq 4.0\}$, $\mathbf{S}^0 = \{10 \leq energy \leq 15, 1.0 \leq avg-speed \leq 2.0, moderately\ safely\}$. So it was the same as Constraint Set 4 and had the same difficulties, plus the soft constraint of “safely” added. (That “safely” caused *INIT* to increase the initial *LIM* value from 1.0 to 3.0, so the trajectory planner did not start from identical weight vectors as it did for Constraint Set 4.) “Safely” requires low speeds and low acceleration, all of which were under the hard constraint values, so there was no competition there: if the trajectory could make those \mathbf{S}^0 , it would also make the \mathbf{H}^0 for free. However, the lower limit of the soft *avg-speed* constraint that was stated explicitly was equal to the upper limit of the *avg-speed* range arising from “safely;” both equal 1 m/s. Unless the trajectory’s average speed could be made exactly 1.0 m/s (highly unlikely), one of those constraints would have to fail. This is another constraint set where every run exhausted the time limit on iterations trying to find acceptable solutions to the soft constraints.

The primary difference in the trajectories arose from the *min-sep* requirement. When *INIT* was run, *LIM* was set to 3.0 to meet it, resulting in the longer trajectories seen for the *INIT* cases. *Default*, starting from *LIM* = 1.0, found more trajectories that stayed closer to obstacles.

Notable is the velocity spike in the middle of the CS 5, OS 4 *Default* trajectory. To minimize the penalty over time for being so close to the obstacles it was passing between at that point, the vehicle speeded up. This is not necessarily desirable behavior! For this reason, the cost functional used in Chapter 5 penalized velocity near obstacles as well as distance from them.

6DOF Cases

Start Point, Goal Point, and Obstacle Sets

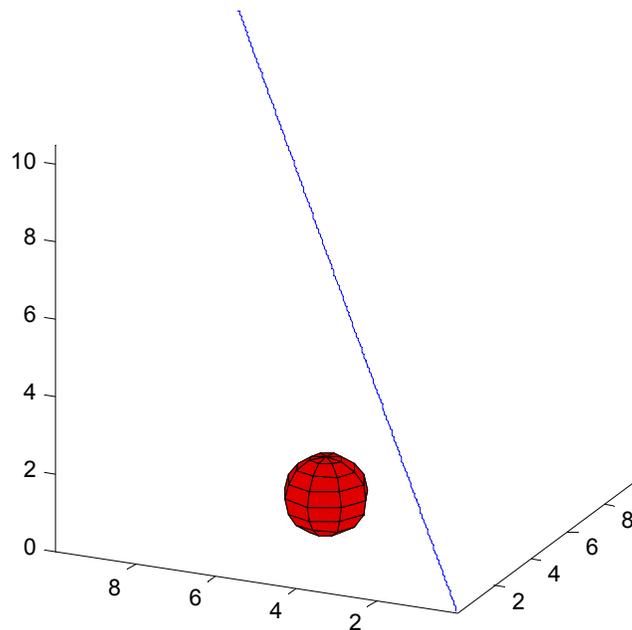
The state \mathbf{x} is a 12-vector [position, velocity, rotation, rotational velocity]. The rotation vector is the modified Rodrigues vector. Spherical obstacles are denoted as a set containing their center in space (x, y, z) and their radius r , all in meters. The set of obstacles is denoted $\{\mathbf{O}\}$

Obstacle Set 1: One large obstacle in the way.

$$\mathbf{x}_0 = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

$$\mathbf{x}_f = [10.0 \ 10.0 \ 10.5 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

$$\{\mathbf{O}\} = \{ \{(6, 6, 0), 1\} \}$$

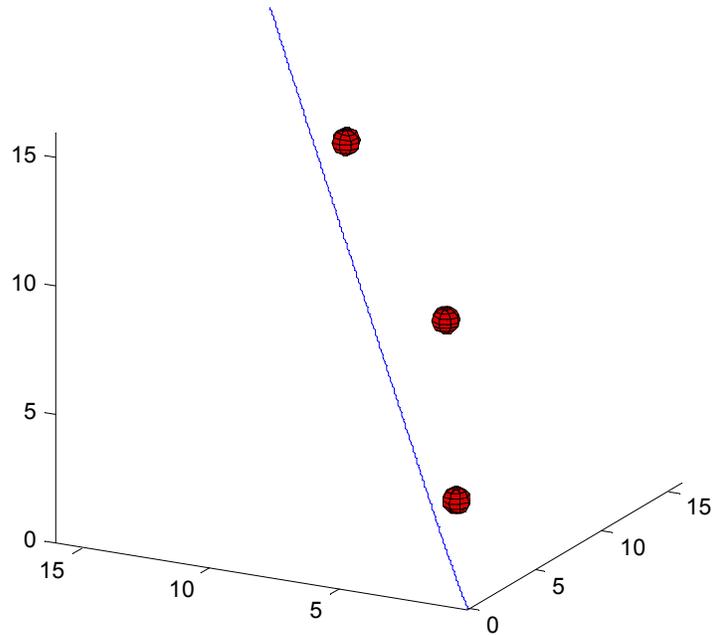


Obstacle Set 2: Scattered obstacles.

$$\mathbf{x}_0 = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$\mathbf{x}_f = [16\ 16\ 16\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$\{\mathbf{O}\} = \{ \{(3, 2, 3), 0.5\}, \{(8, 5, 8), 0.5\}, \{(14, 12, 12), 0.5\} \}$$

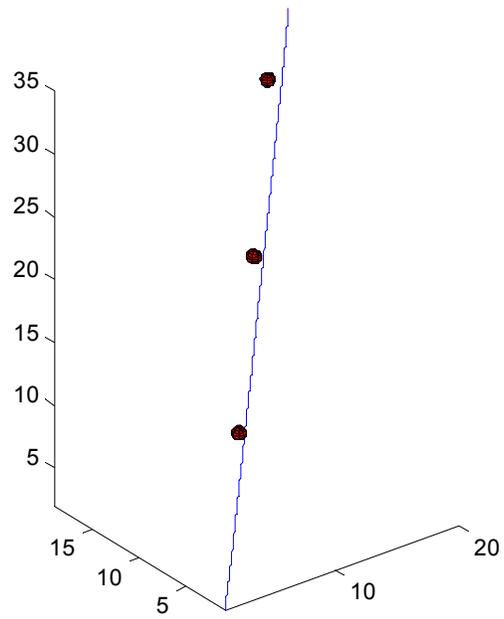


Obstacle Set 3: Moving from one end of a line of satellites to the other.

$$\mathbf{x}_0 = [20\ 19\ 35\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$\mathbf{x}_f = [1\ 1\ 2\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$\{\mathbf{O}\} = \{ \{(6, 6, 12), 0.5\}, \{(11, 11, 22), 0.5\}, \{(16, 16, 32), 0.5\} \}$$

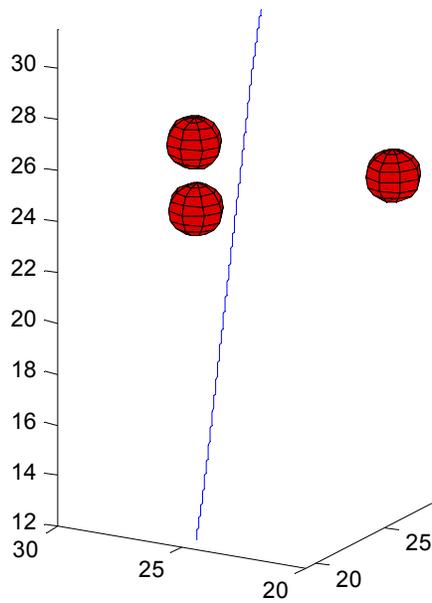


Obstacle Set 4: Assuming a position in a tetrahedron.

$$\mathbf{x}_0 = [20.4 \ 25 \ 12 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

$$\mathbf{x}_f = [25 \ 25 \ 31.53 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

$$\{\mathbf{O}\} = \{ \{(20.38, 25, 25), 1\}, \{(27.31, 21, 25), 1\}, \{(27.31, 29, 25), 1\} \}$$



Constraint Sets

Constraint Set 1

$$\mathbf{H}^0 = \{max-speed \leq 5.5 \text{ m/s}\}$$

$$\mathbf{S}^0 = \{\text{somewhat quickly}\}$$

Constraint Set 2

$$\mathbf{S}^0 = \{\text{exceedingly efficiently}\}$$

Constraint Set 3

$$\mathbf{S}^0 = \{\text{a little quickly}\}$$

Constraint Set 4

$$\mathbf{H}^0 = \{max-acc \leq 1.3, max-speed \leq 4\}$$

$$\mathbf{S}^0 = \{2.7 \leq force \leq 5.4, 1.8 \leq avg-speed \leq 4\}$$

Constraint Set 5

$$\mathbf{H}^0 = \{max-acc \leq 1.3, max-speed \leq 4\}$$

$$\mathbf{S}^0 = \{2.7 \leq \text{force} \leq 5.4, 1.8 \leq \text{avg-speed} \leq 4, \text{moderately safely}\}$$

Constraint Set 6

$$\mathbf{S}^0 = \{\text{very energy-saving}\}$$

Constraint Set 7

$$\mathbf{S}^0 = \{\text{low torque}\}$$

Constraint Set 8

$$\mathbf{S}^0 = \{\text{medium torque}\}$$

Margins of Success and Failure

This table is formatted the same way as the one in the 2DOF section.

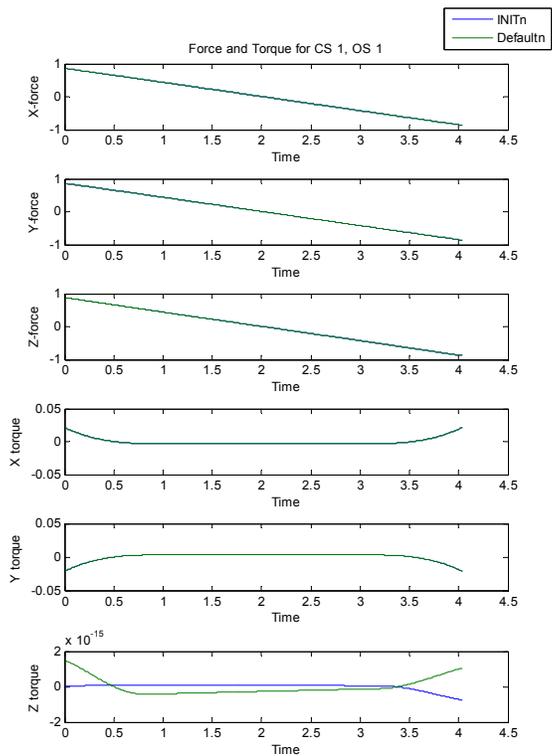
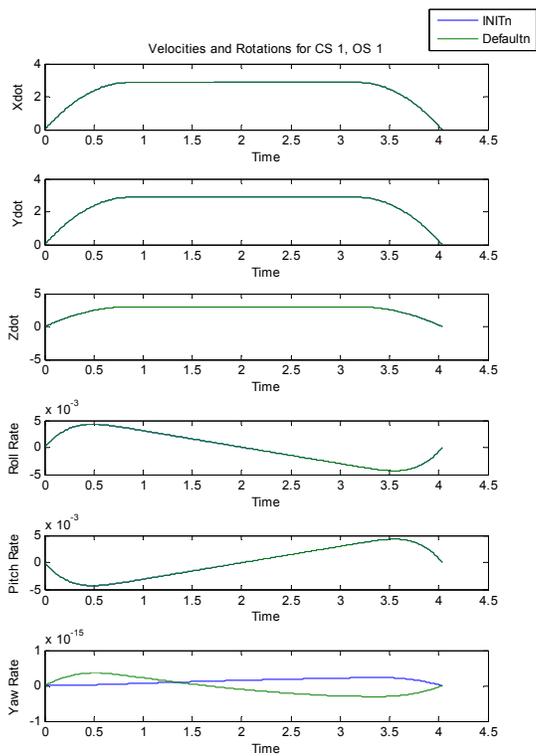
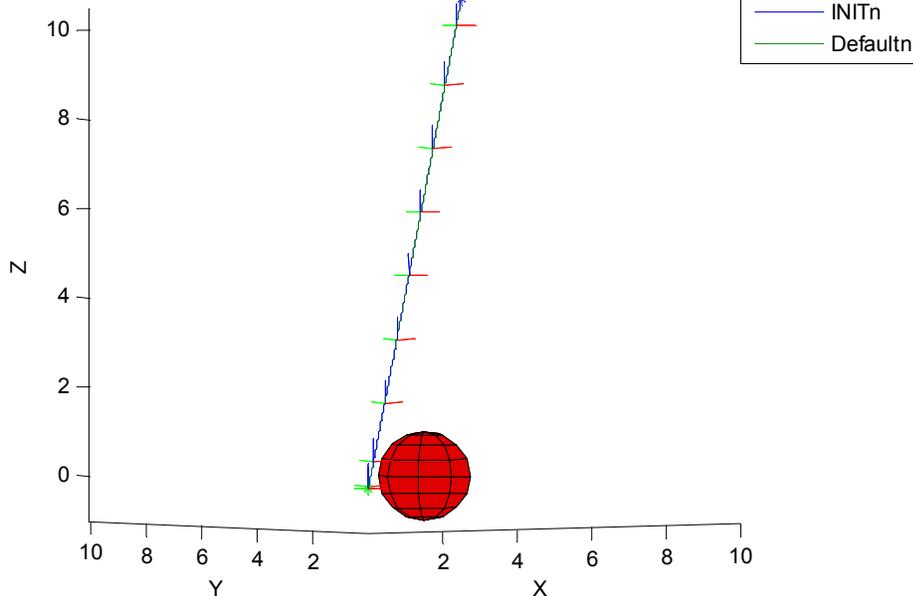
	Margins															
	Obs1				Obs2				Obs3				Obs4			
	INIT0	INITn	Default0	Defaultn	INIT0	INITn	Default0	Defaultn	INIT0	INITn	Default0	Defaultn	INIT0	INITn	Default0	Defaultn
Constraint Set 1																
max speed <= 5.5	0.17	0.07	0.57	0.07	0.06	0.00	0.46	0.09	0.30	0.00	0.28	0.02	0.05	0.00	0.44	0.04
somewhat quickly																
hi 5 <= max speed <= 10.6	-0.15	-0.96	-0.95	-0.86	-0.70	-0.83	-0.72	-0.99	-0.23	-0.82	-0.37	-0.85	-0.73	-0.82	-0.68	-0.90
hi 4 <= avg speed <= 8.6	-0.04	-0.89	-0.91	-0.89	-0.48	-0.64	-0.62	-0.88	0.06	-0.61	-1.23	-0.60	-0.65	-0.76	-1.68	-0.87
Constraint Set 2																
exceedingly efficiently																
lo 2.7 <= force <= 5.4	0.36	0.36	2.00	0.05	0.16	0.99	3.63	0.99	0.98	0.00	4.40	0.00	-0.17	-0.17	0.64	0.02
Constraint Set 3																
a little quickly																
hi 5 <= max speed <= 10.6	-0.15	-0.96	-0.95	-0.96	-0.70	-0.70	-0.72	-0.99	-0.23	-0.23	-0.37	0.58	-0.73	-0.73	-0.68	-0.68
hi 4 <= avg speed <= 8.6	-0.04	0.89	-0.91	-0.89	-0.48	-0.48	-0.62	-0.88	0.06	0.06	-1.23	0.90	-0.65	-0.65	-1.68	-1.68
Constraint Set 4																
max acc <= 1.3	19.31	0.08	17.82	0.51	30.56	0.47	26.98	0.45	42.64	0.23	49.79	0.35	22.74	60.72	26.55	60.72
max speed <= 4	0.14	0.90	0.42	0.95	0.46	0.94	0.26	0.95	0.79	0.92	0.01	0.93	0.44	6.24	0.23	6.24
soft 2.7 <= force <= 5.4	7.73	-0.85	2.00	-1.39	10.96	-1.34	3.63	-1.45	13.81	-1.08	4.40	-1.22	6.39	47.53	0.64	47.53
soft 1.8 <= avg speed <= 4	0.91	-1.45	-0.91	-1.54	1.08	-1.53	-0.30	-1.55	2.22	-1.48	-0.56	-1.51	0.74	18.16	-1.50	18.16
Constraint Set 5																
max acc <= 1.3	19.31	0.08	17.82	0.41	34.02	0.05	26.98	0.09	43.27	0.55	49.79	0.46	22.74	0.50	26.55	0.45
max speed <= 4	0.14	0.90	0.42	0.94	0.50	0.92	0.26	0.92	0.83	0.95	0.01	0.94	0.44	0.95	0.23	0.94
soft 2.7 <= force <= 5.4	7.73	-0.85	2.00	-1.26	11.75	-1.13	3.63	-1.12	14.69	-1.51	4.40	-1.40	6.39	-1.56	0.64	-1.52
soft 1.8 <= avg speed <= 4	0.91	-1.45	-0.91	-1.52	0.99	-1.47	-0.30	-1.49	2.43	-1.56	-0.56	-1.54	0.74	-1.54	-1.50	-1.53
moderately safely																
hi 1.5 <= min sep <= 2.5	-0.80	0.34	-0.80	0.34	0.43	0.77	-1.20	1.02	0.66	0.39	-1.18	0.87	0.95	1.45	0.41	1.45
lo 1.4 <= max speed <= 2.7	2.88	-1.54	0.44	-1.76	5.11	-1.65	0.42	-1.68	7.09	-1.87	1.93	-1.80	4.70	-1.82	0.61	-1.79
lo 1 <= max acc <= 2.5	31.87	-0.73	29.30	-0.30	57.37	-0.68	45.17	-0.76	73.40	-0.55	84.70	-0.40	37.82	-0.46	44.42	-0.38
lo 0.9 <= avg speed <= 1.8	4.67	-1.56	0.22	-1.71	7.31	-1.60	1.71	-1.64	10.82	-1.80	-0.37	-1.77	6.69	-1.76	-1.67	-1.74
Constraint Set 6																
very energy-saving																
lo torque 0.05 <= torque <= .2	-0.67	-0.67	-0.67	-0.67	-0.67	-0.67	-0.66	-0.67	-0.62	-0.64	-0.20	0.90	-0.64	-0.64	-0.63	-0.43
force	0.32	0.32	2.00	0.05	0.16	0.02	3.63	0.99	0.96	0.87	4.40	0.00	-0.19	-0.19	0.64	0.02

6DOF Trajectory Data by Constraint Set

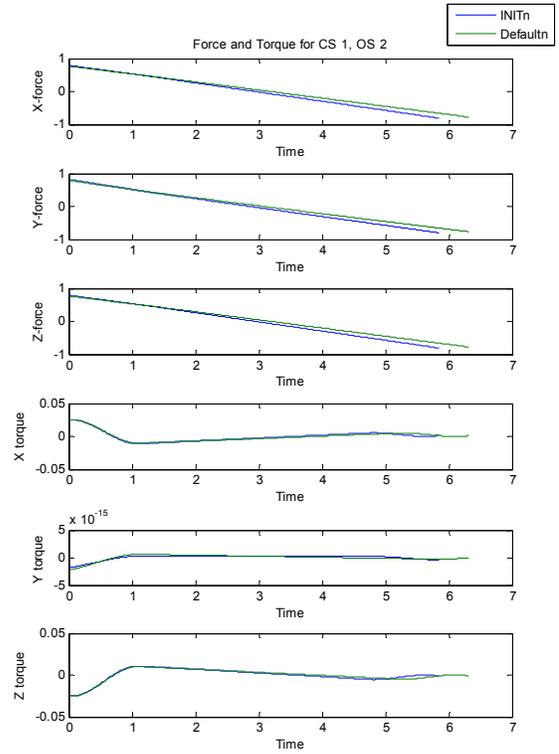
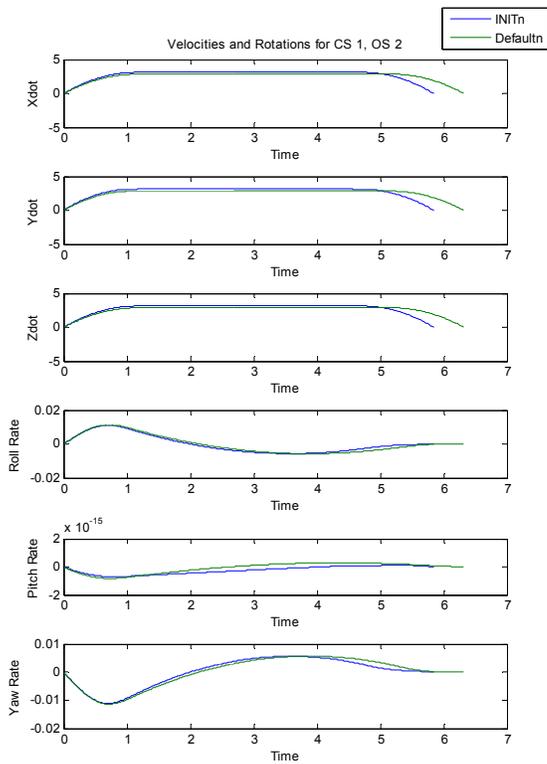
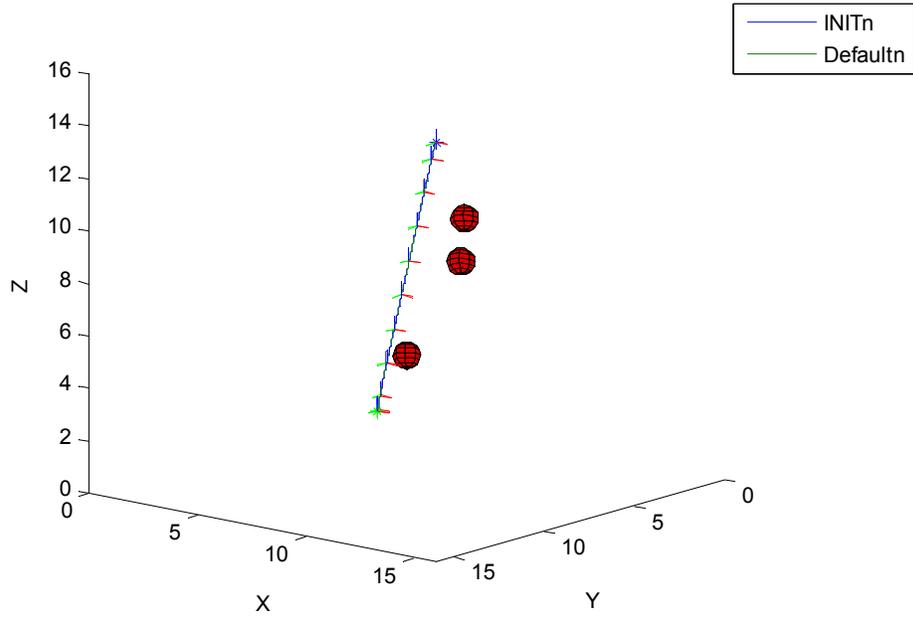
We append here charts showing the paths and trajectories of all 40 2DOF test cases. Each chart displays the *Default*ⁿ case in green and the *INIT*ⁿ case in blue. Each constraint set and obstacle set solution is represented by two charts. The first shows the path that the agent took through the obstacle field. The second shows translational and rotational rates. The third shows force and torque inputs.

Each chart is labeled using “CS” to denote “Constraint Set” and “OS” to denote “Obstacle Set.” So “CS1, OS1” is the information for Constraint Set 1, Obstacle Set 1.

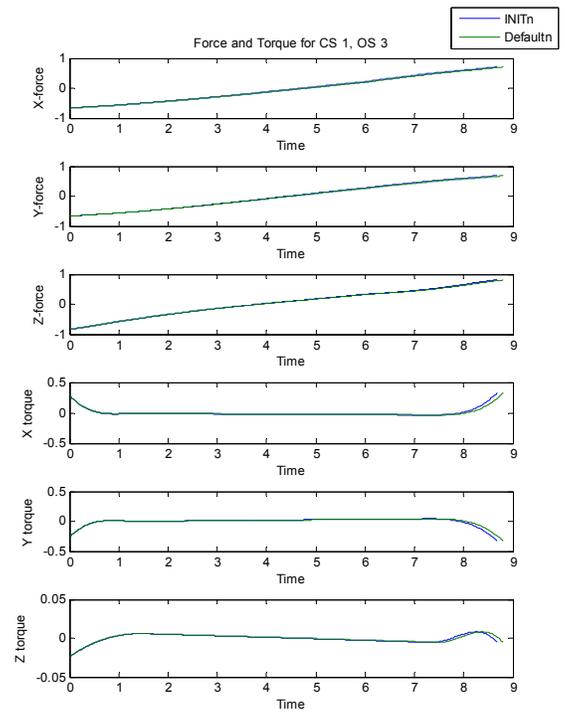
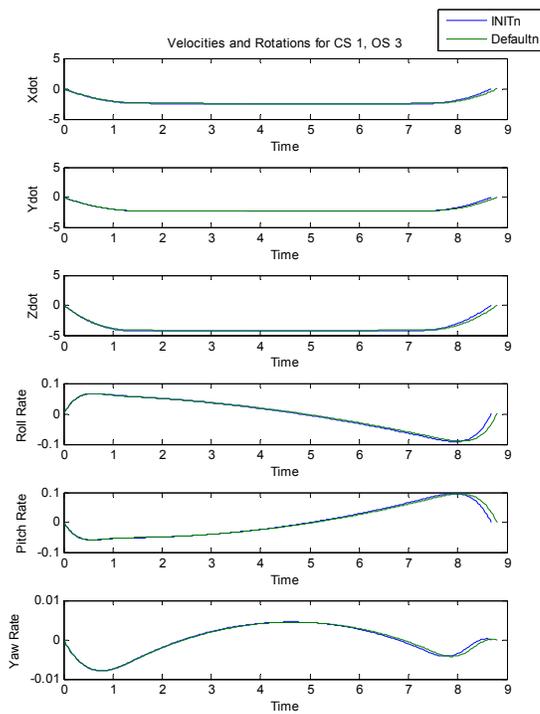
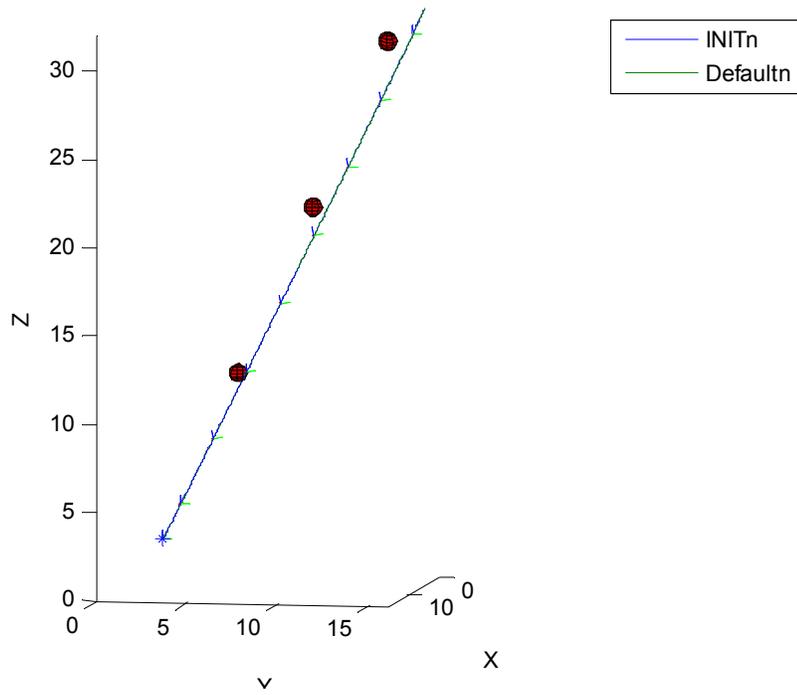
Path for CS 1, OS 1



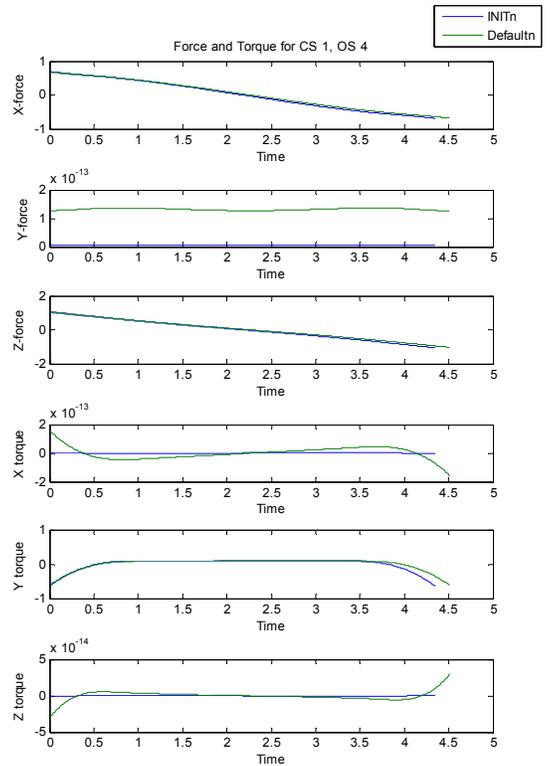
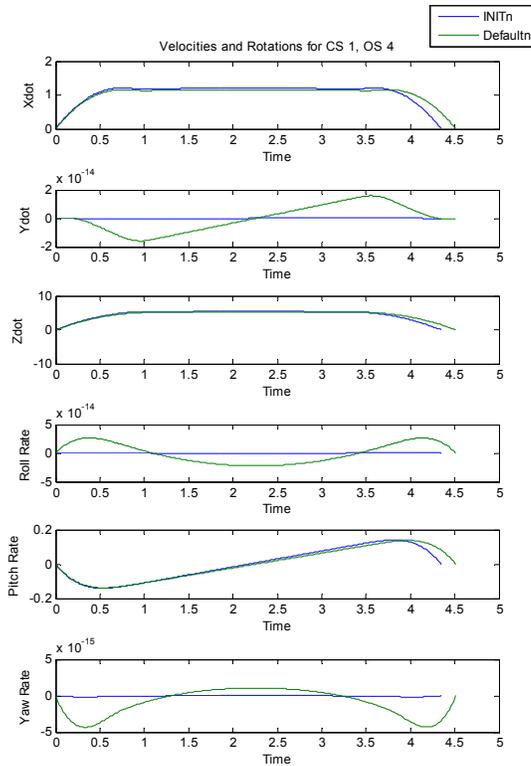
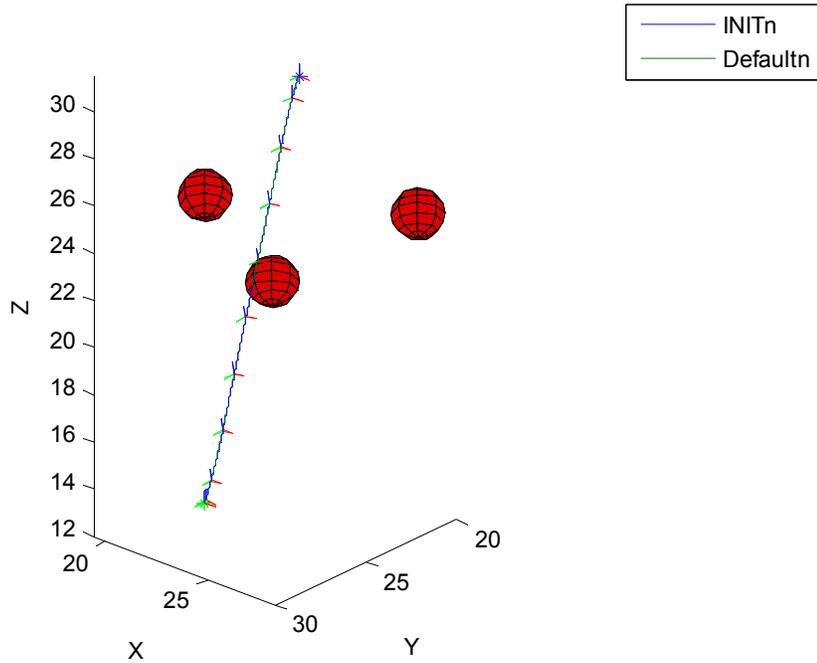
Path for CS 1, OS 2



Path for CS 1, OS 3



Path for CS 1, OS 4

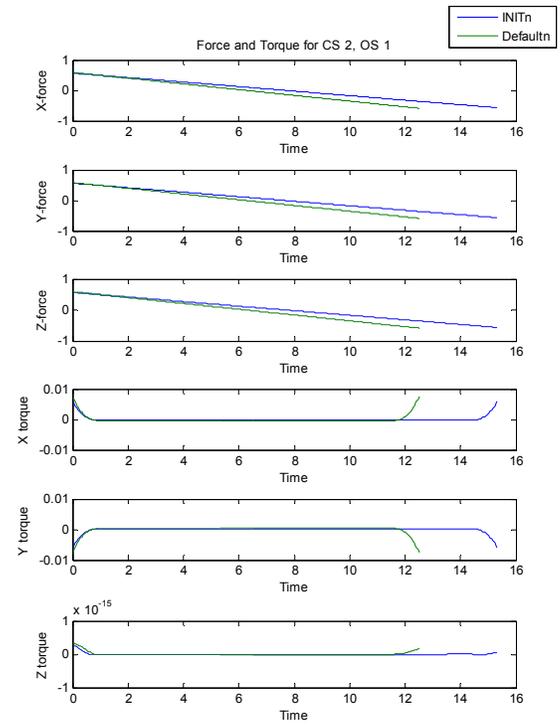
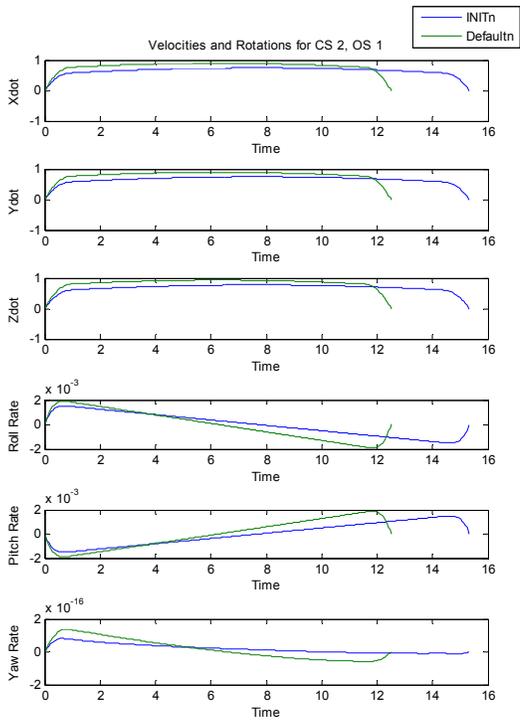
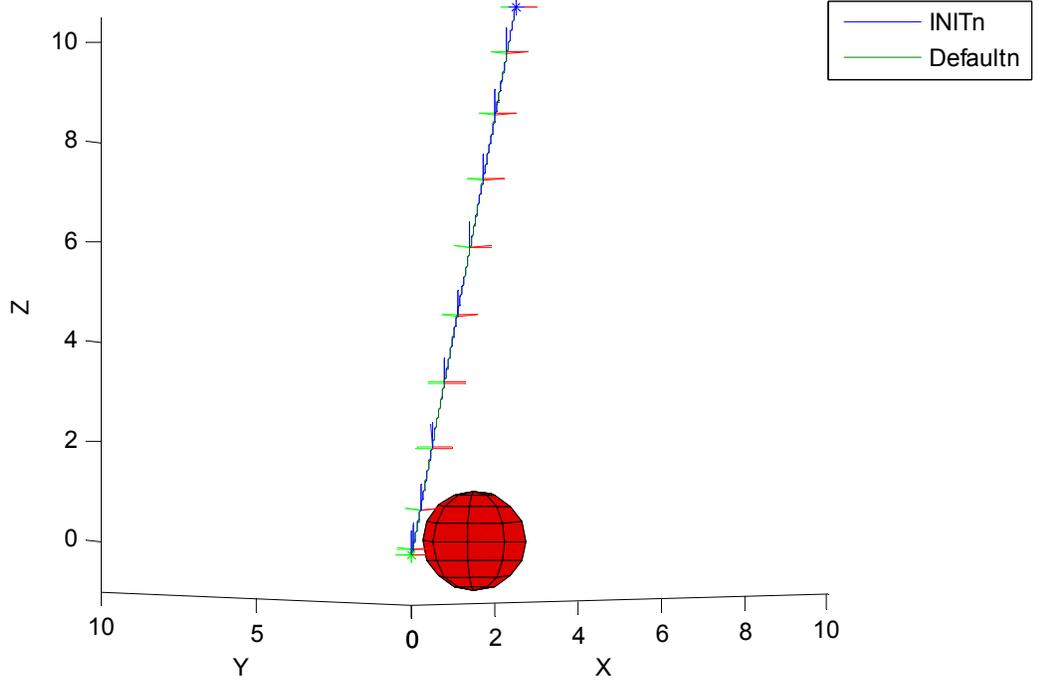


Constraint Set 1 included $\mathbf{H}^0 = \{max-speed \leq 5.5\}$, $\mathbf{S}^0 = \{\text{somewhat quickly}\}$. The soft constraints were extended to $\mathbf{S}^0 = \{5.0 \text{ m/s} \leq max-speed \leq 10.6 \text{ m/s}, 4.0 \text{ m/s} \leq avg-speed \leq 8.6 \text{ m/s}\}$. These in general were very successful. For the *INIT* cases, only two iterations were needed for total success. (We did have a problem with Obstacle Set 4, where a hard limit failure below our level of precision forced 24 iterations when, in effect, we were meeting our constraints exactly.) In the default cases, Obstacle Sets 1 and 2 reacted similarly, increasing the time weight until success was found. For Obstacle Sets 3 and 4 in the default cases, the results were more similar to the 2DOF Constraint Set 1 behavior, with the time weight initially over-adjusted up, then adjusted back down in one or more steps.

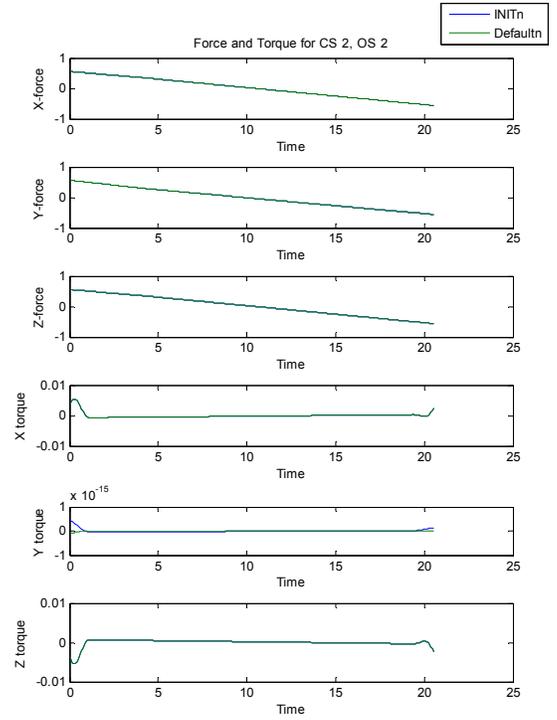
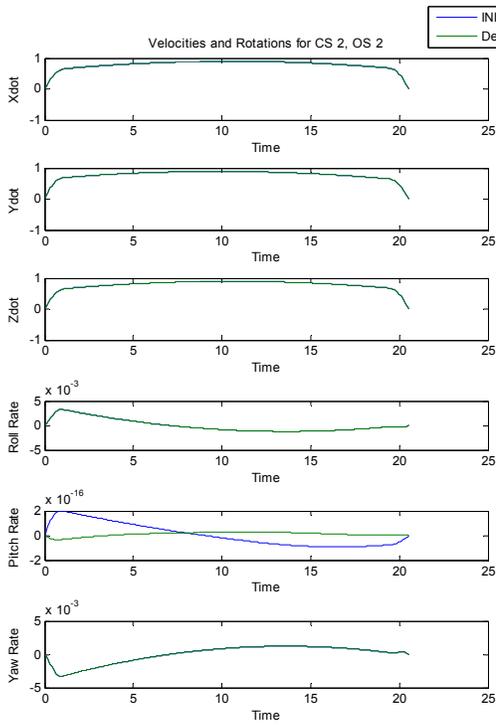
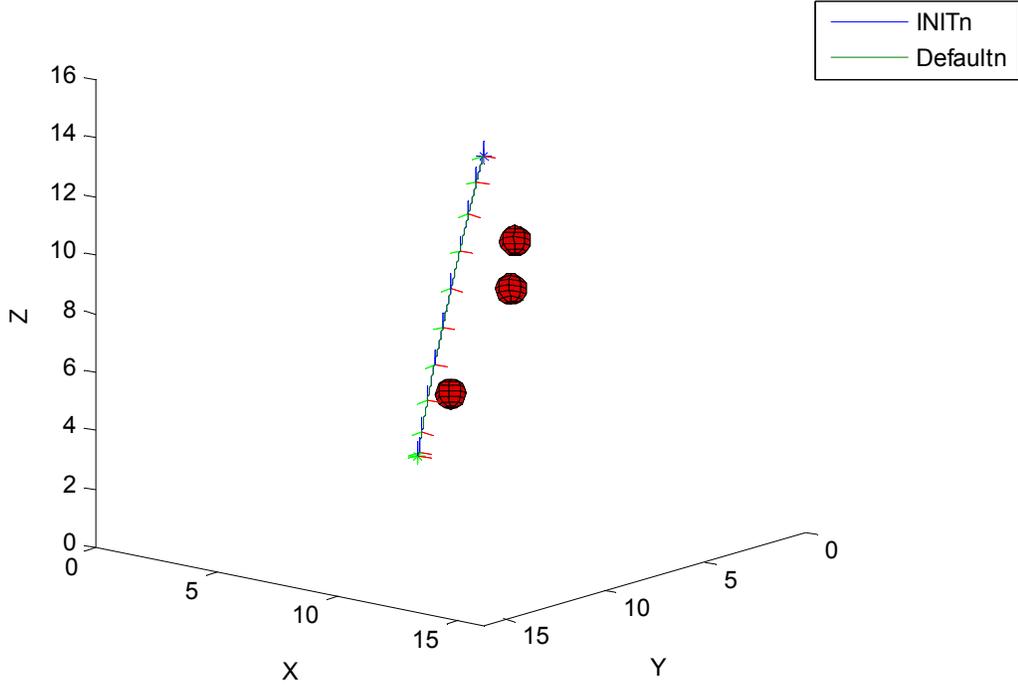
Since path lengths varied from obstacle set to set, final times were not necessarily similar across the board. Similar speed results were achieved for all cases, however.

Obstacle Set 4 requires no translation in the y -direction, so there is no force there in any of the Constraint Sets.

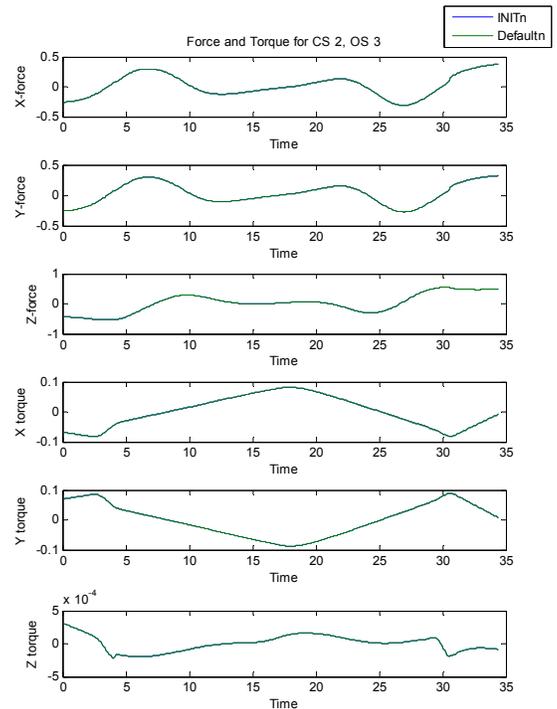
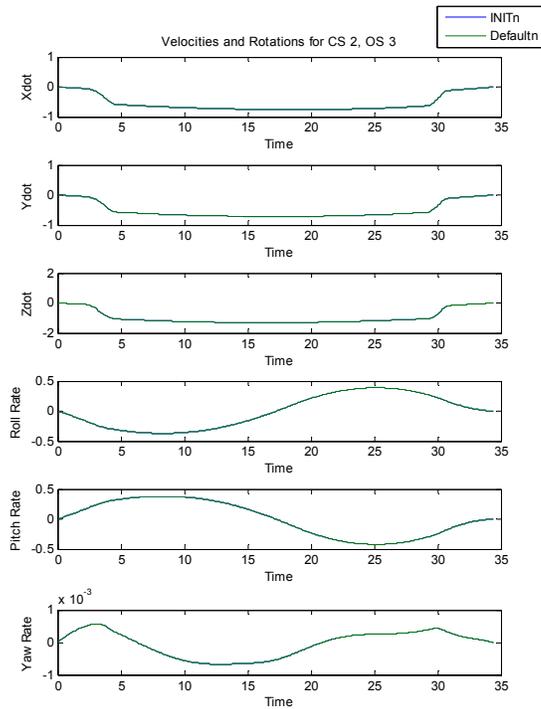
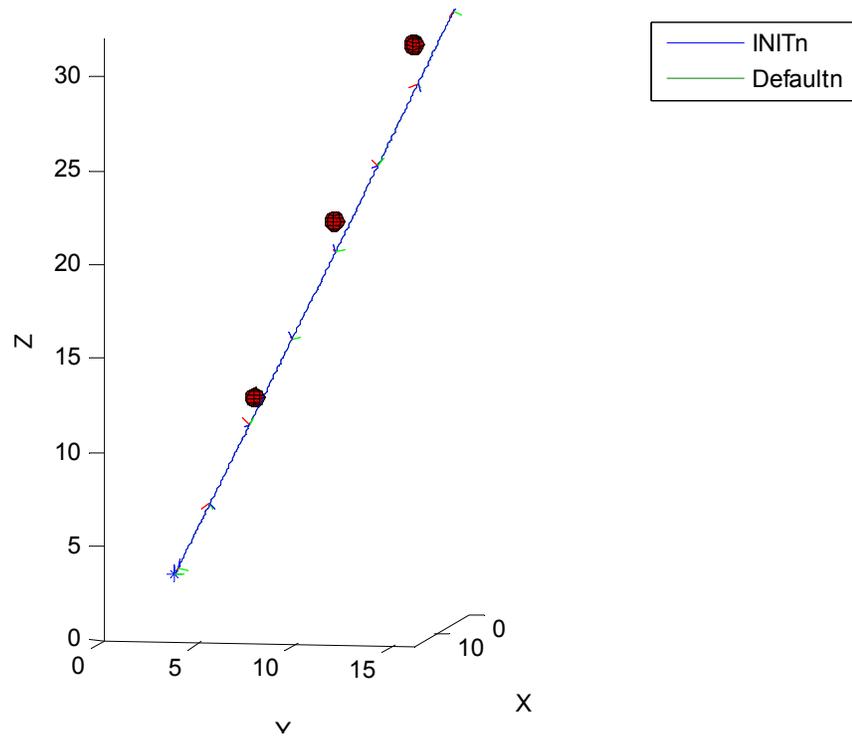
Path for CS 2, OS 1



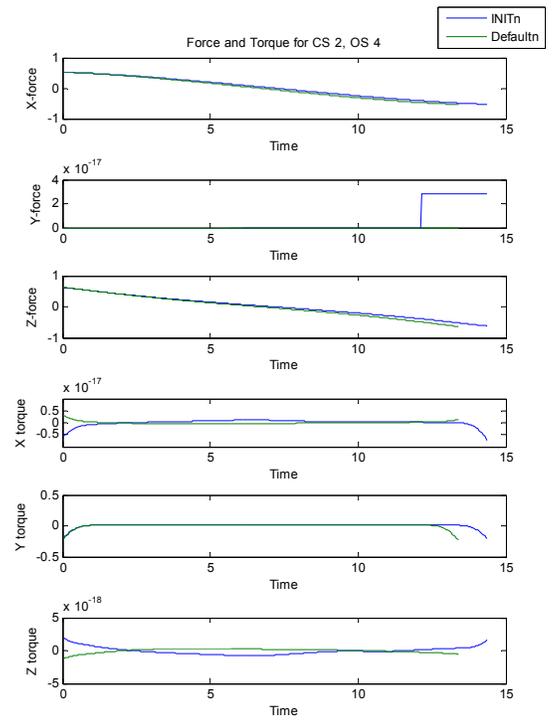
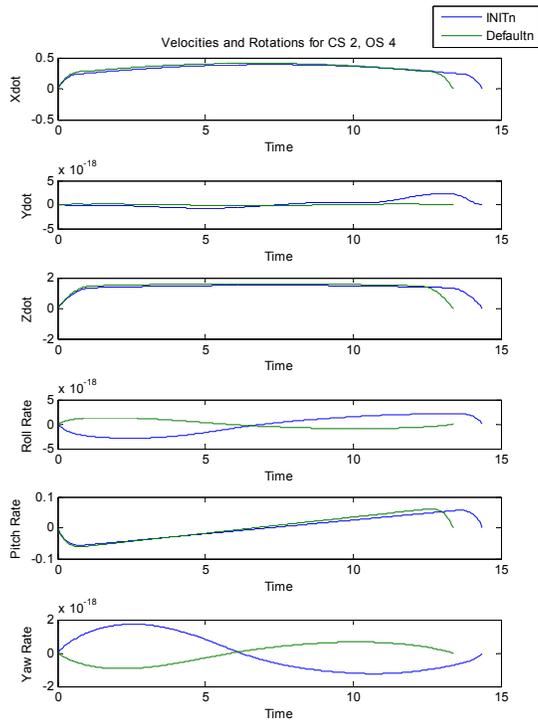
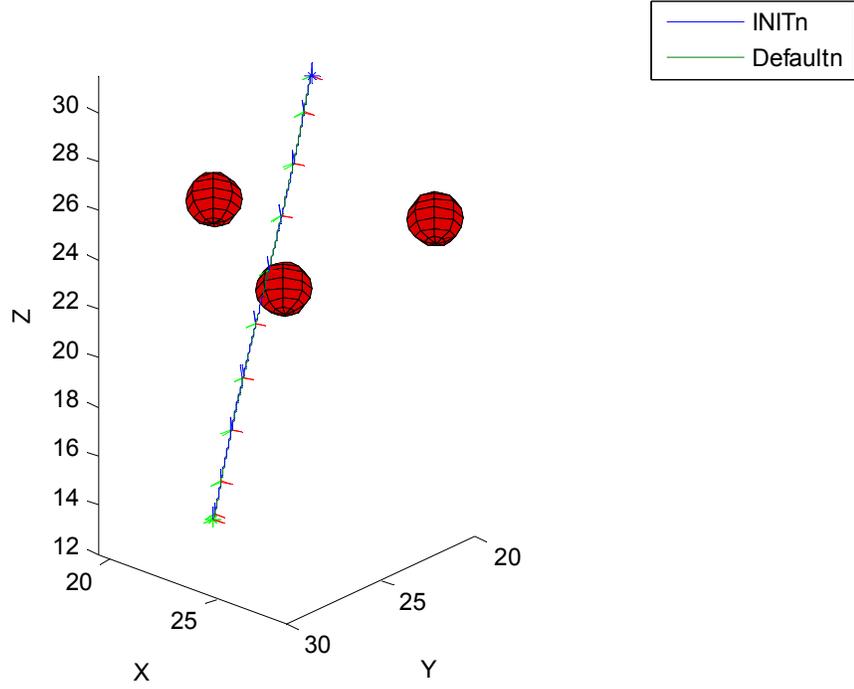
Path for CS 2, OS 2



Path for CS 2, OS 3



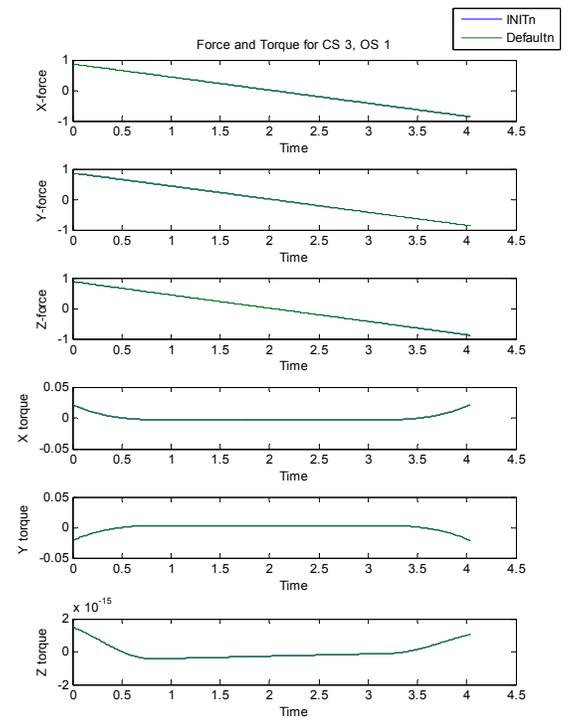
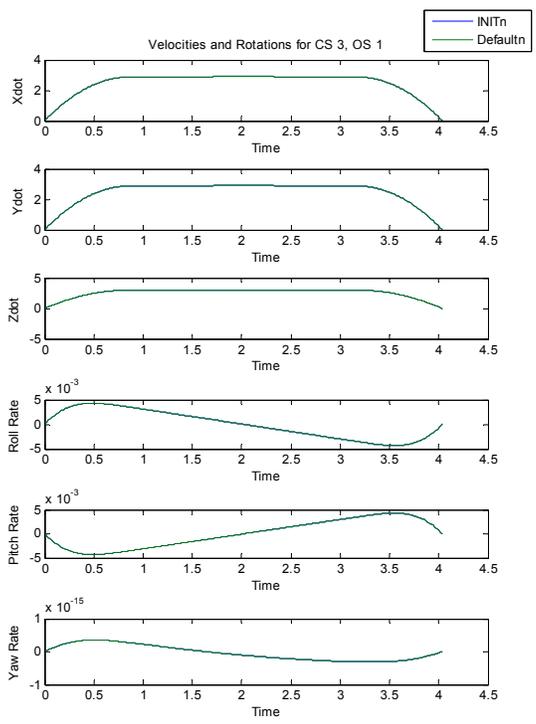
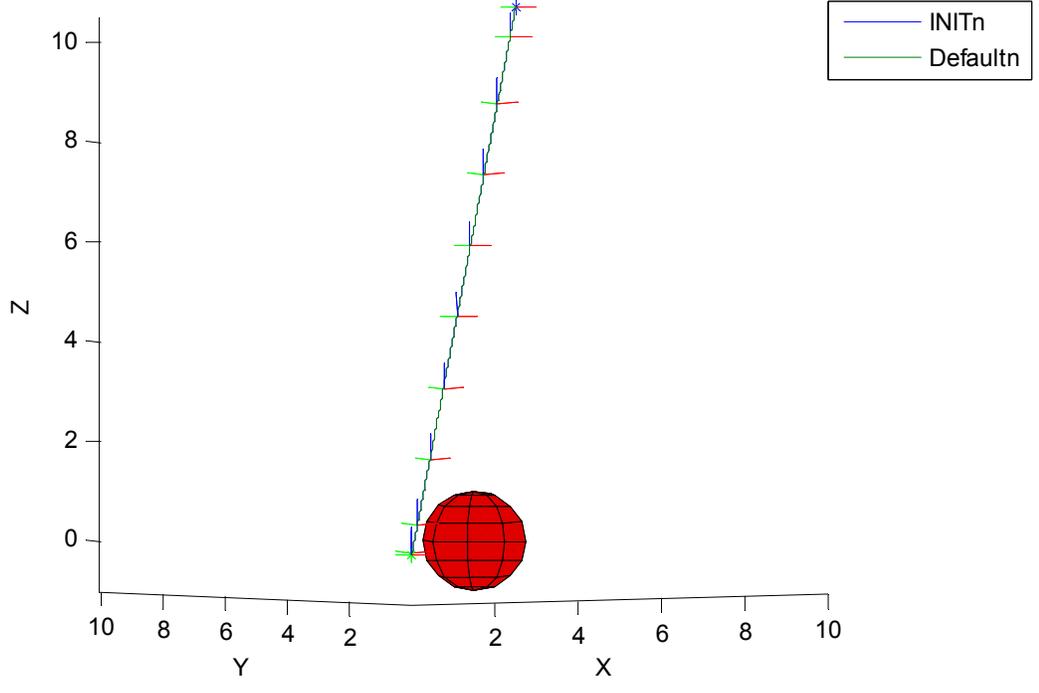
Path for CS 2, OS 4



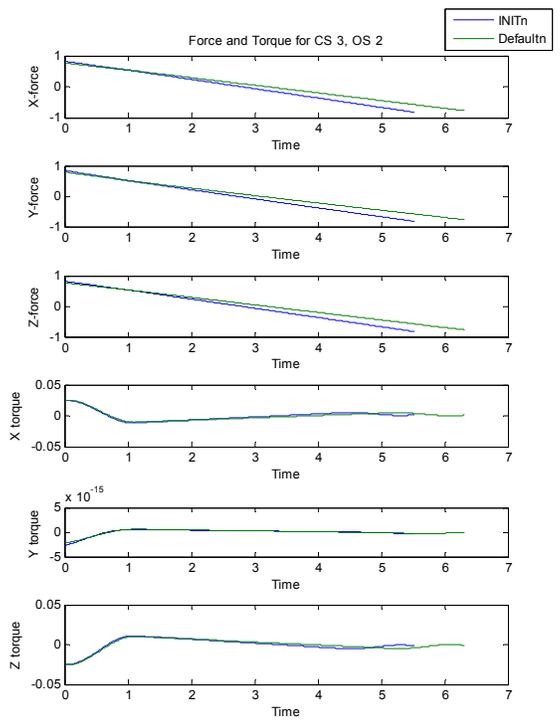
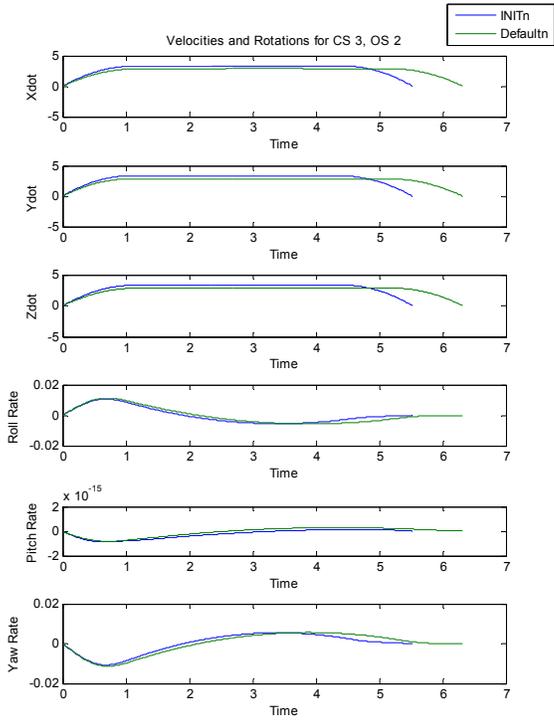
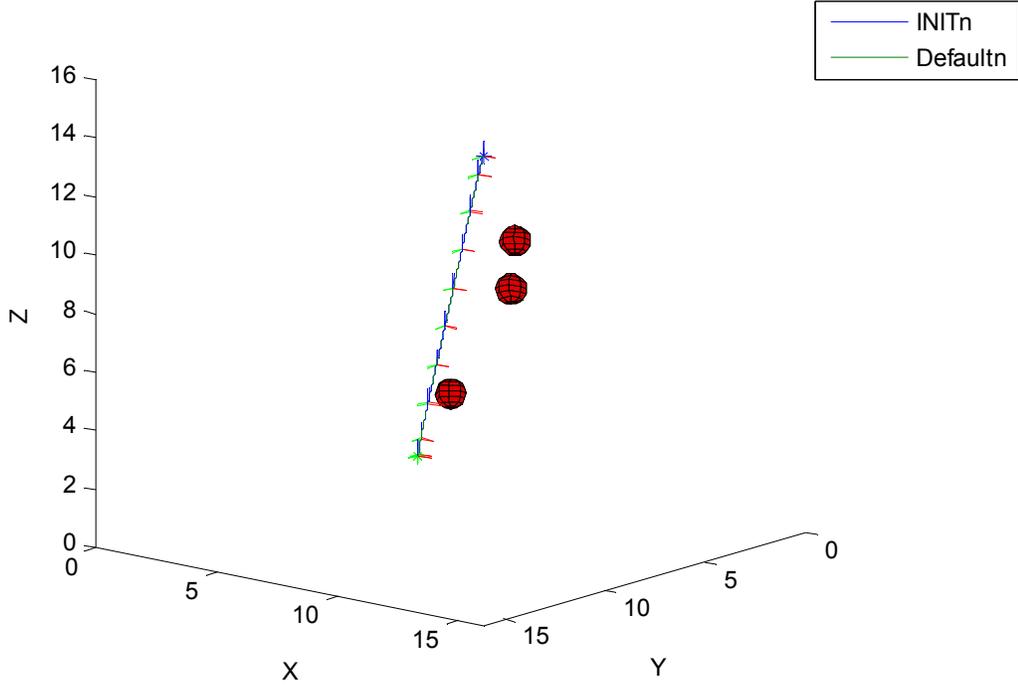
Constraint Set 2 was only a soft constraint, $\mathbf{S}^0 = \{\text{“exceedingly efficiently”}\}$. This was extended to “low *force*,” and that was in turn defined for this system as $2.7 \text{ N} \leq \textit{force} \leq 5.4 \text{ N}$. *INIT* was again very successful with this set; for Obstacle Sets 1 and 4, it solved it with the first weight vector. For OS 2 and 3, another iteration was needed; the time weight (which was set to 0.25 by *INIT*) was a little too high and had to be lowered further. The default cases all needed to adjust the time weight down, which they did in between three and five steps. *Default* results were not identical to *INIT* results, but all were very similar.

For OS 1, 2 and 4, we had the typical smooth acceleration followed by a smooth deceleration, which is more what we would expect for a time-efficient trajectory. Perhaps the values of “low” *force* were overestimated for smaller fields; Obstacle Set 3 gives something closer to a bang-coast-bang approximation, which is what we would expect to see for an efficient trajectory. Our *WADJ* curves were generated in 50m cube volumes, which is closer in size to the field of Obstacle 3 than the others. We reiterate that the returned trajectories met the numeric values that defined the fuzzy preference “efficiently” – *WADJ* did not fail, based on what it knew. However, this result suggests that its knowledge was faulty and that some of our *ad hoc* definitions require more refinement.

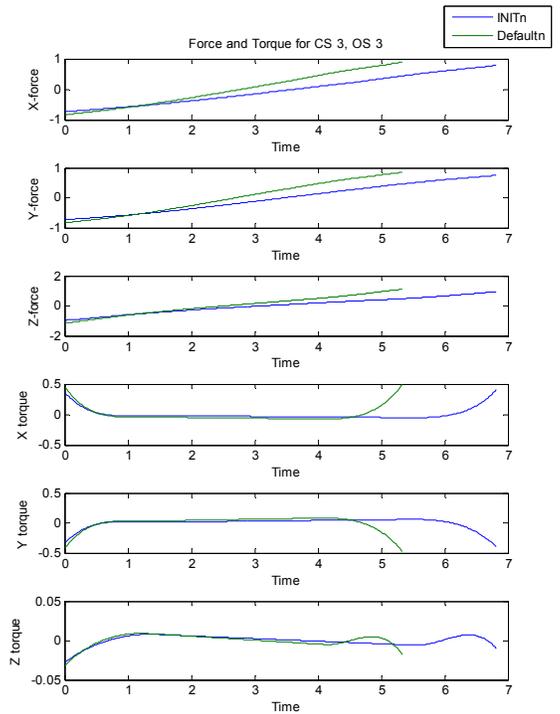
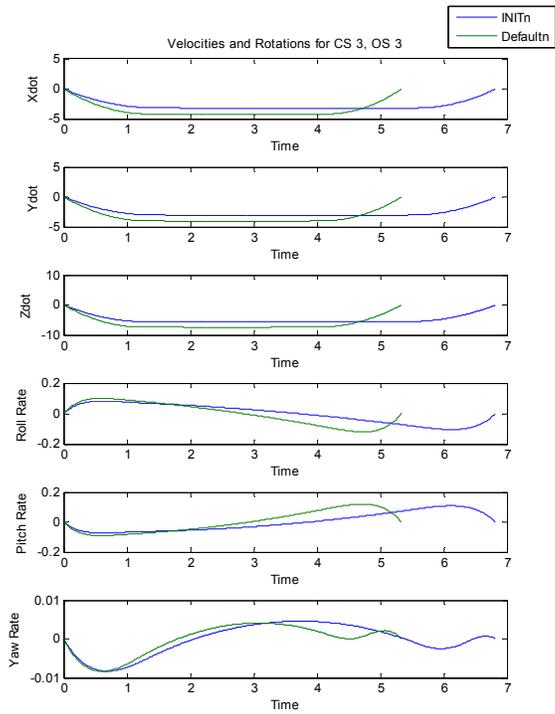
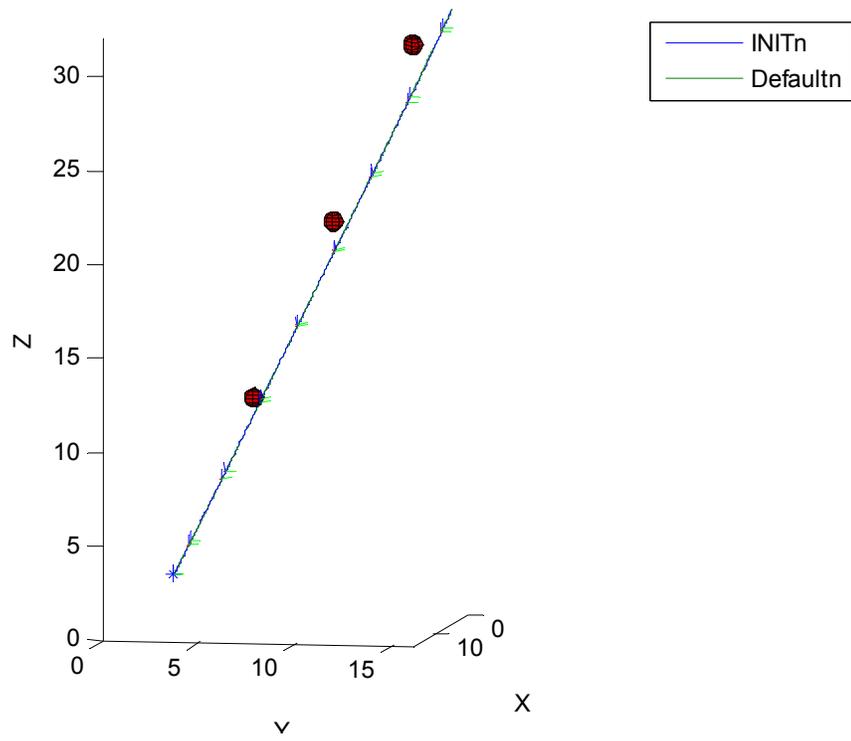
Path for CS 3, OS 1



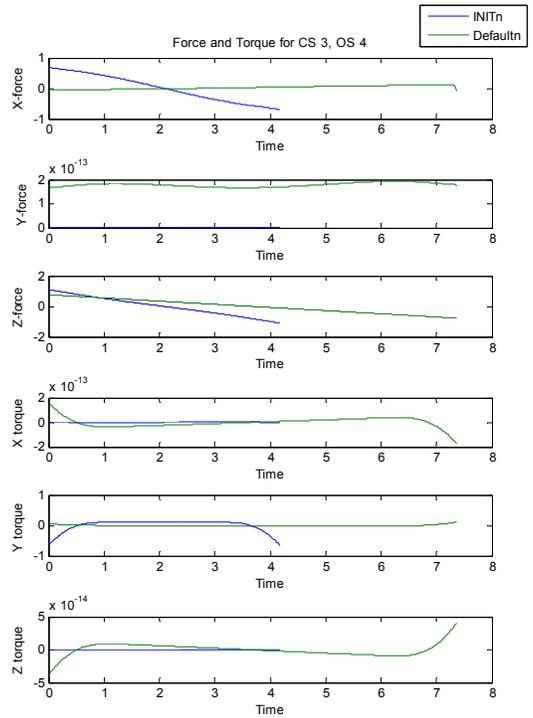
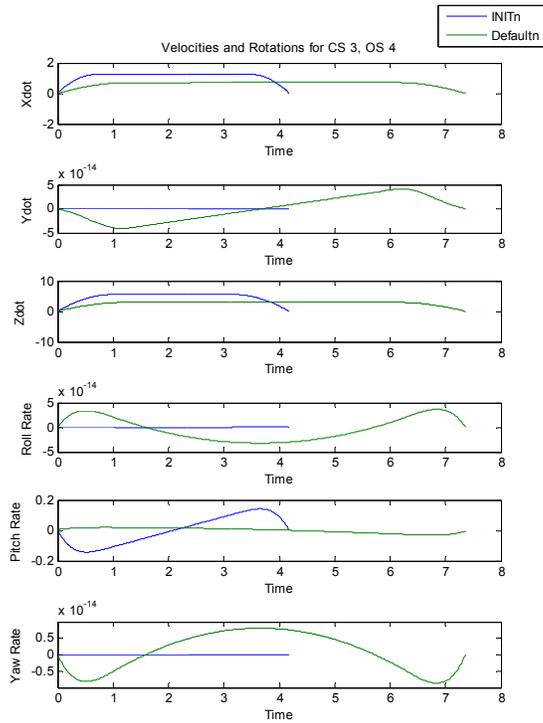
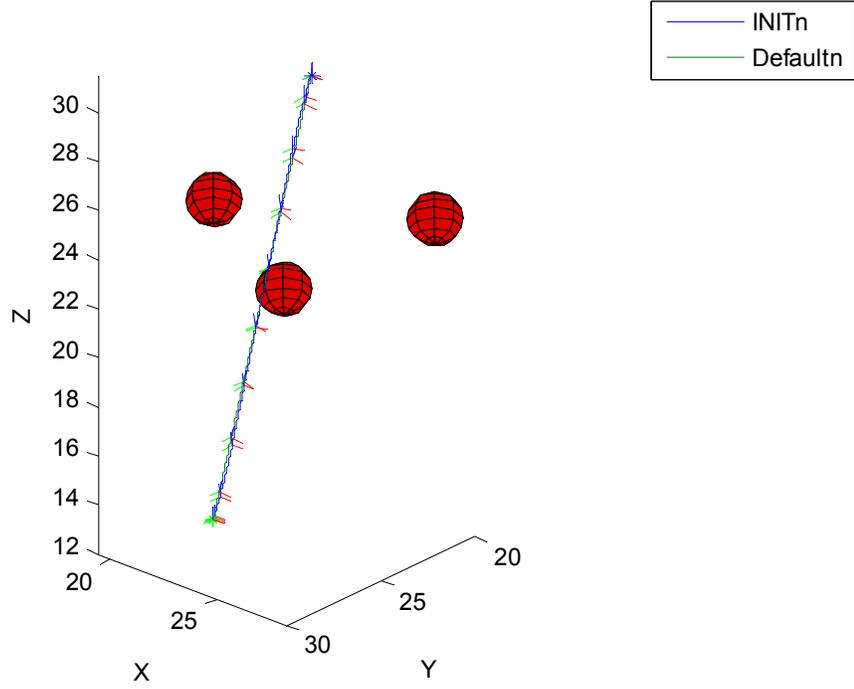
Path for CS 3, OS 2



Path for CS 3, OS 3



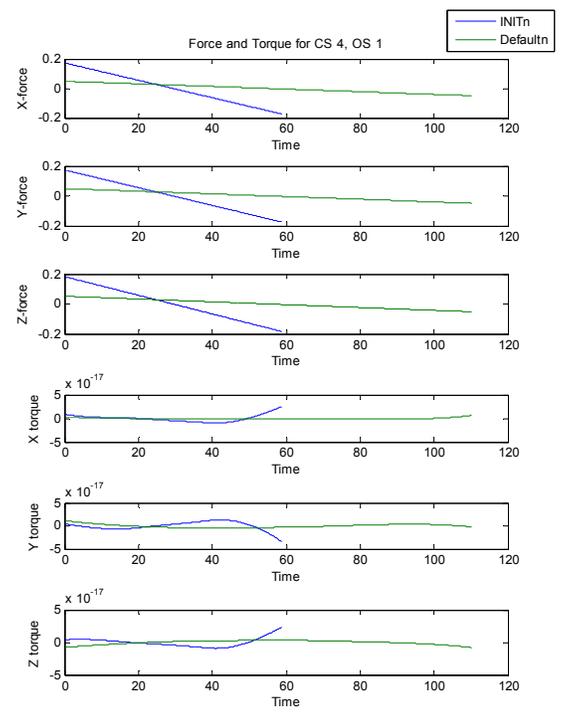
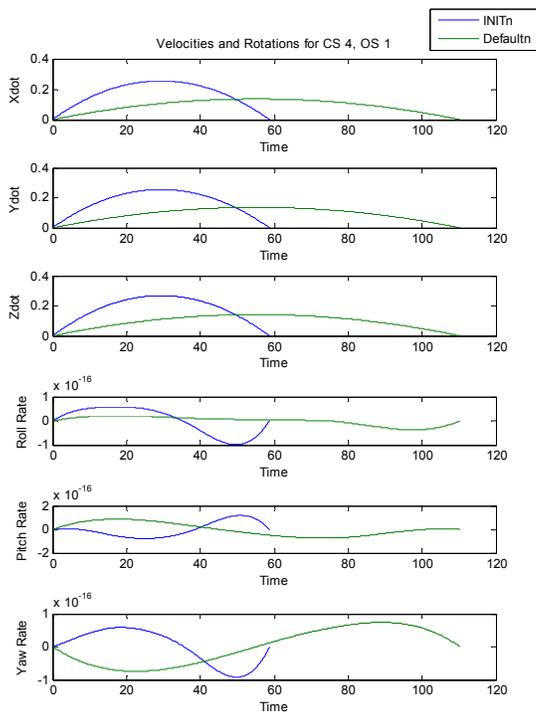
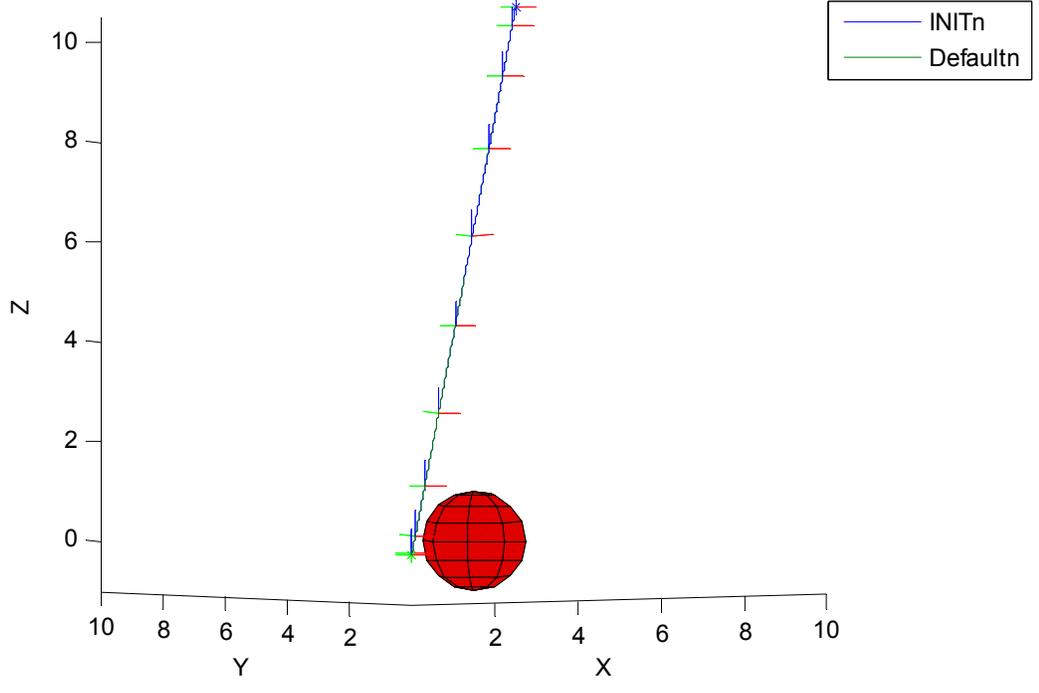
Path for CS 3, OS 4



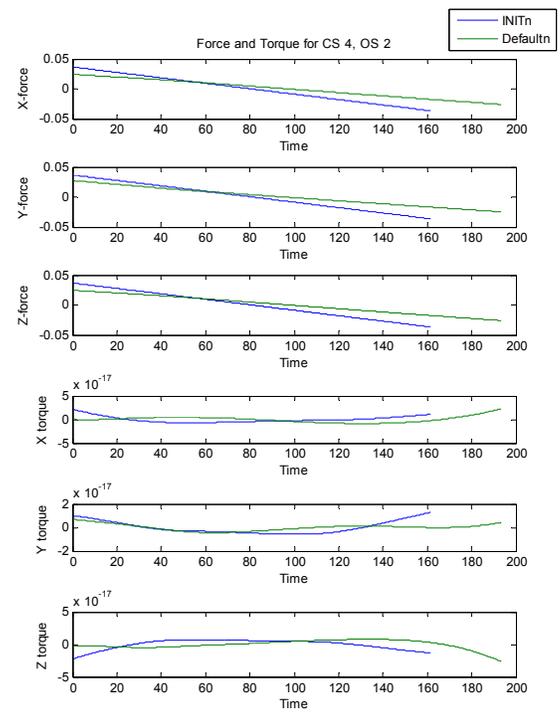
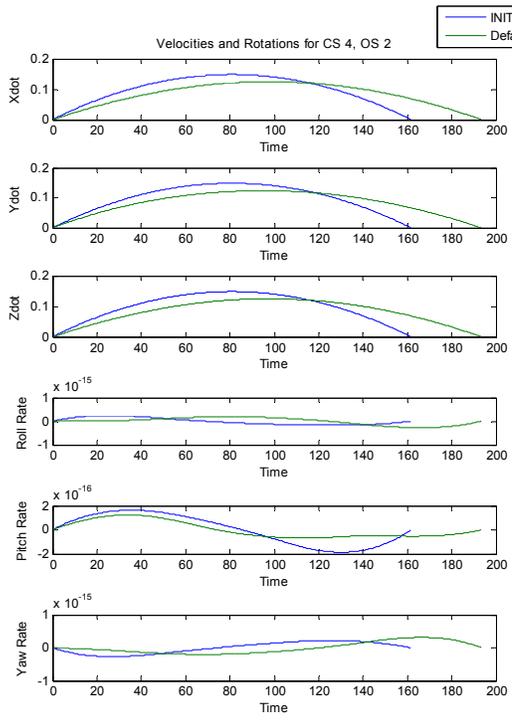
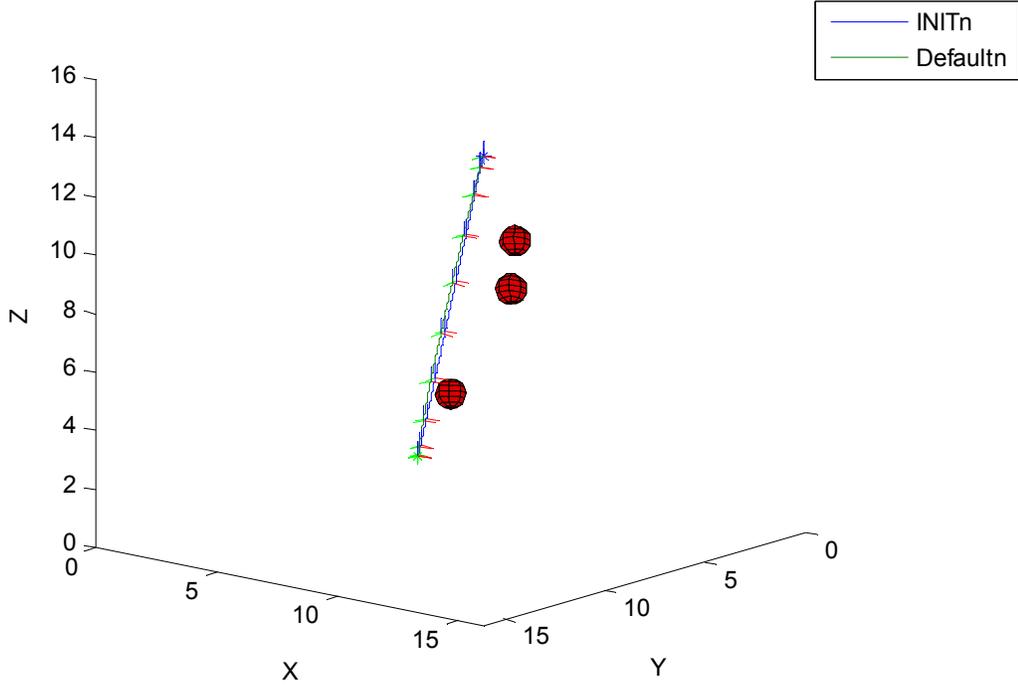
Constraint Set 3 was another soft-only, $S^0 = \{\text{"a little quickly"}\}$. *INIT* was generally more successful than the default case here, as it solved for all obstacle sets in a single iteration in three cases, and required only two for the fourth. The default approach typically required three or four for OS 1-3. For OS 4, it seriously over-estimated the time weight needed in its second iteration. By the fourth iteration, it was back to a more reasonable answer, but the margin of failure was so small that the weight adjustments to the time weight were also very small, and the weight was not lowered enough before the time limit expired. The first iteration, with default weights, was found to have the smallest errors and returned; this is the only time in this CS that the default case found a solution significantly different from the *INIT* case.

We see again in all the OS the typical time-efficient force curves that we would hope to see in a “quickly” preference.

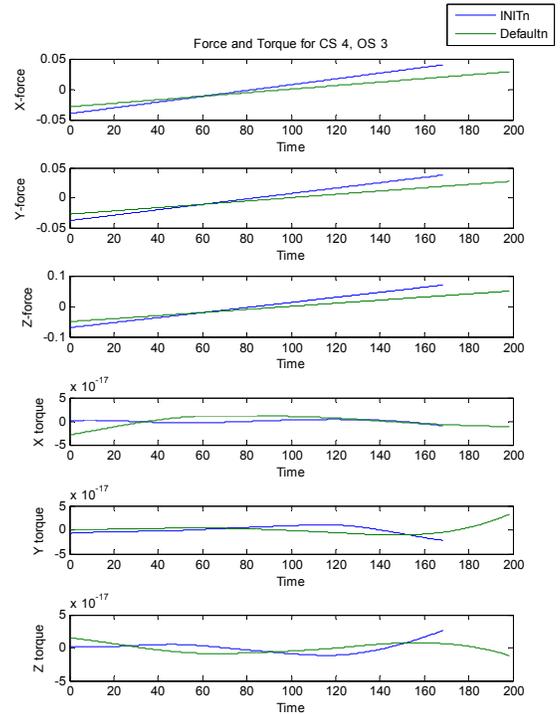
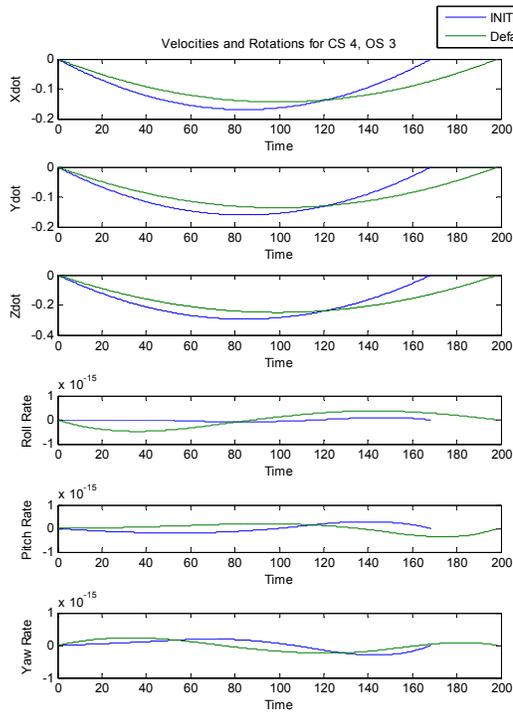
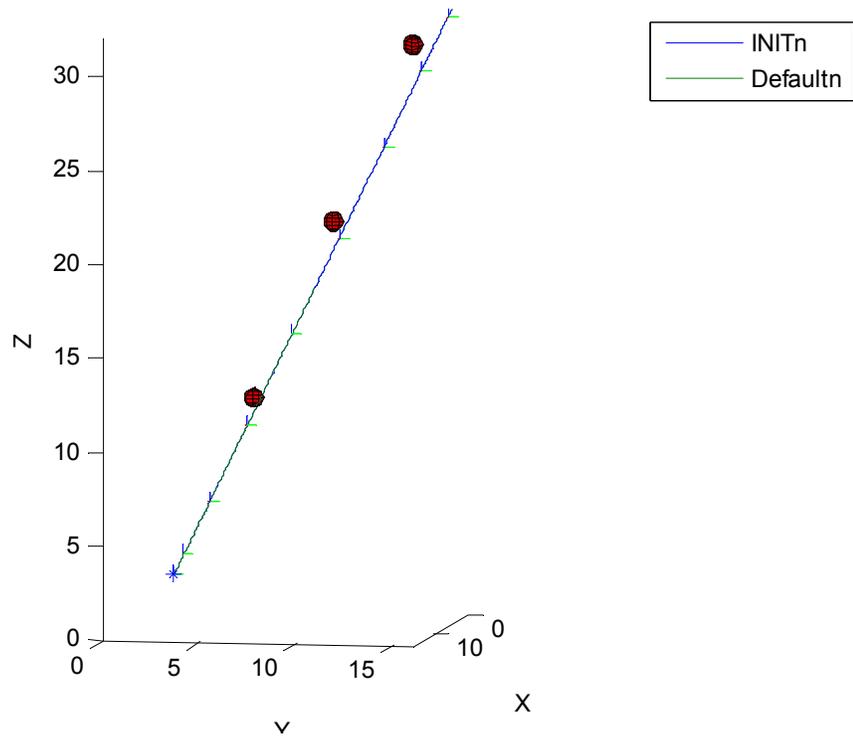
Path for CS 4, OS 1



Path for CS 4, OS 2



Path for CS 4, OS 3

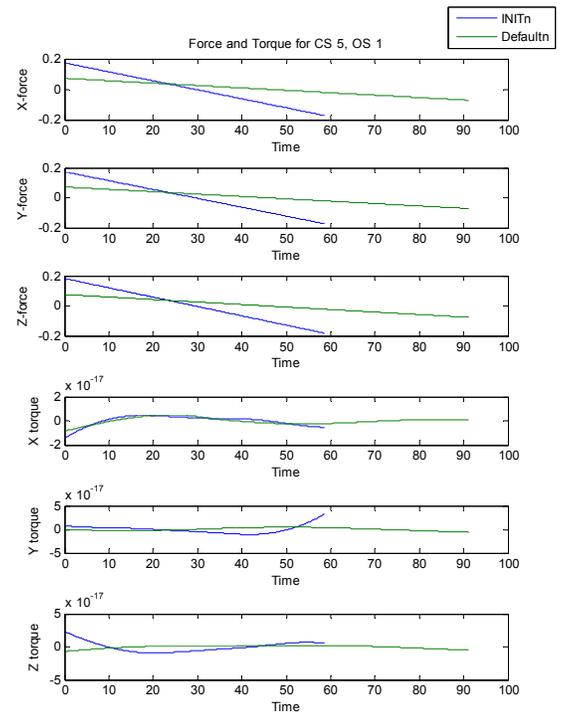
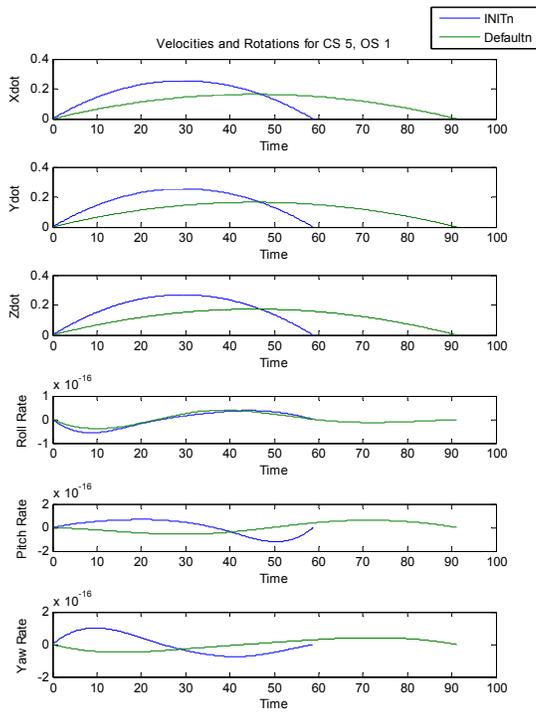
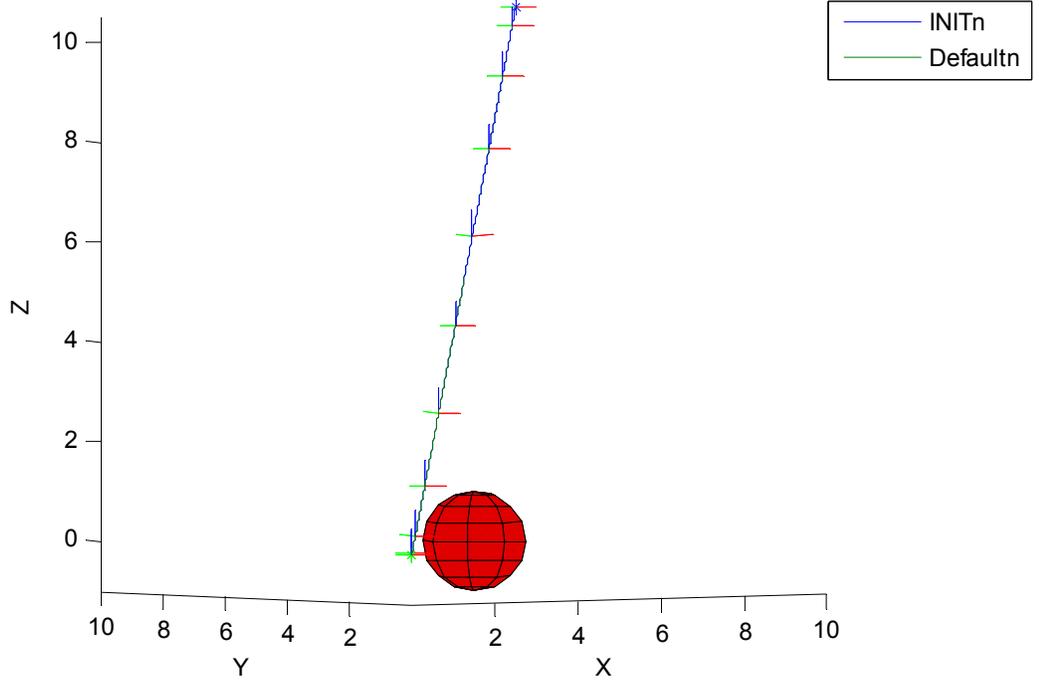


Constraint Set 4 consisted of two hard numeric constraints, $\mathbf{H}^0 = \{max-acc \leq 1.3 \text{ m/s}^2, \quad max-speed \leq 4.0 \text{ m/s}\}$, and two soft numeric constraints $\mathbf{S}^0 = \{2.7 \text{ N} \leq force \leq 5.4 \text{ N}, 1.8 \text{ m/s} \leq avg-speed \leq 4.0\}$. The acceleration and force are low for this system; the *avg-speed* is medium-low or low; and the *max-speed* must be less than medium-high. The constraints do not obviously conflict, but they will interact. We note that this CS failed to converge for OS 4.

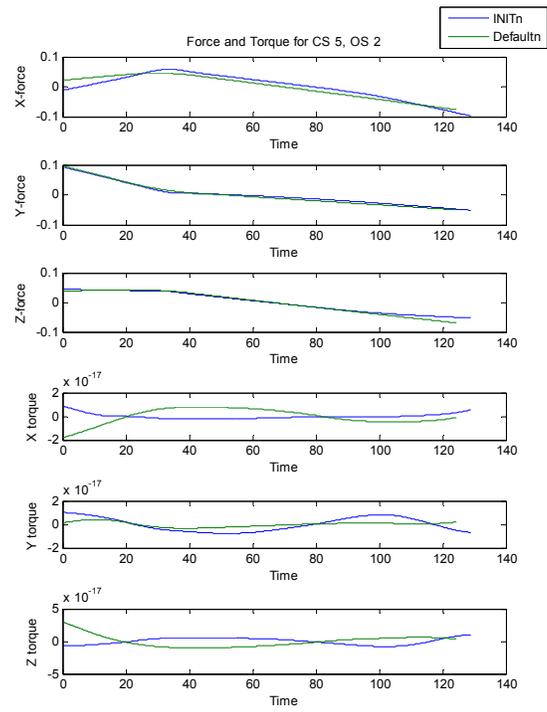
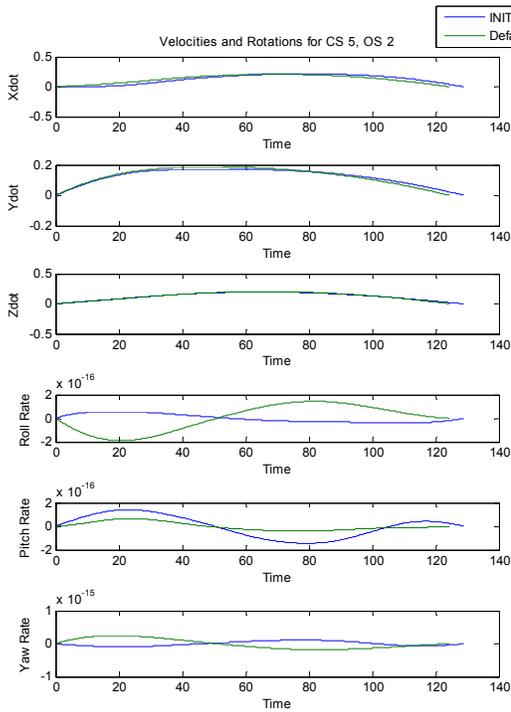
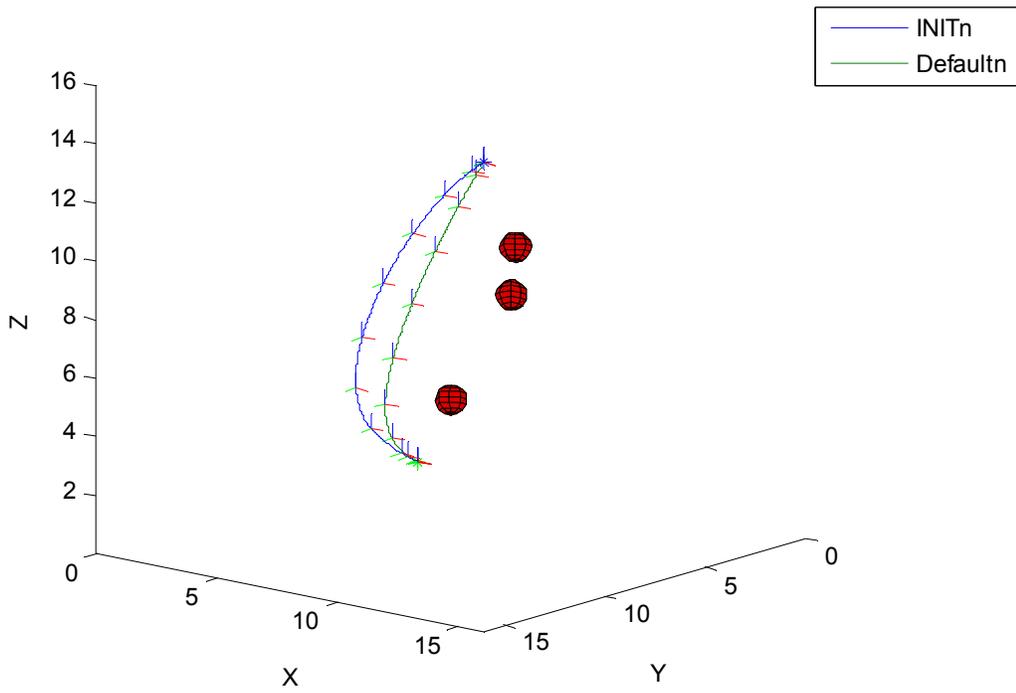
INIT selected weights that typically made the *max-speed* \mathbf{H}^0 but not the *max-acc*. The default weights failed similarly. In all cases, this set up a cyclic solution cycle. Weights were lowered to meet *max-acc*; typically on Iteration 2 both \mathbf{H}^0 were met. (For *INIT*, OS 2 required five iterations to meet both; for default, OS 3 required two.) In all cases where both \mathbf{H}^0 were met, both \mathbf{S}^0 failed low.

Despite the fact that most of the time weights were on the order of 0.01, the force profiles still have the constant thrust characteristic of time-efficient trajectories. The levels of thrust are greatly decreased from the prior constraint sets, and the time for completion greatly extended. The velocity profiles match well, and the torques are zero to our level of precision.

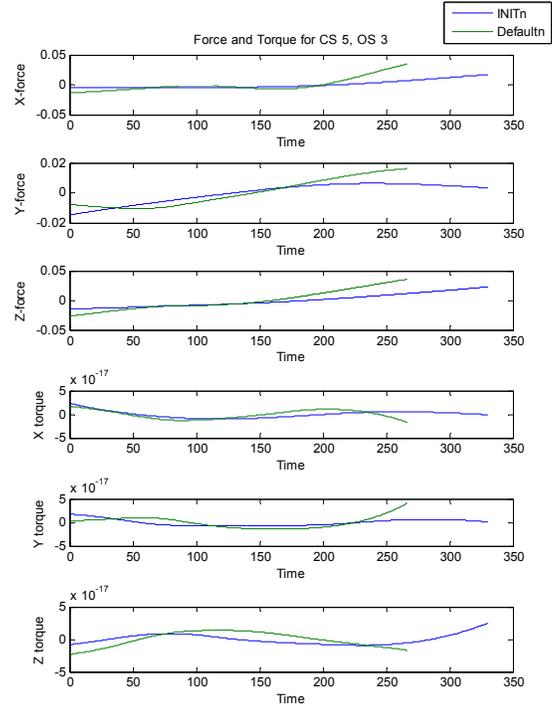
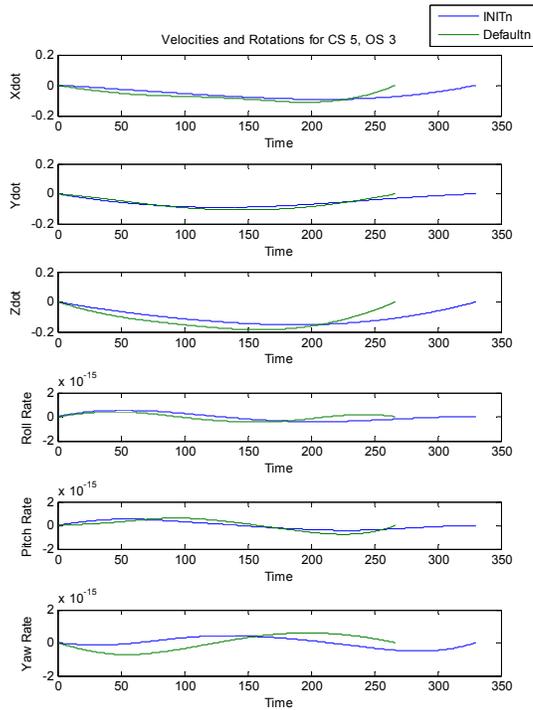
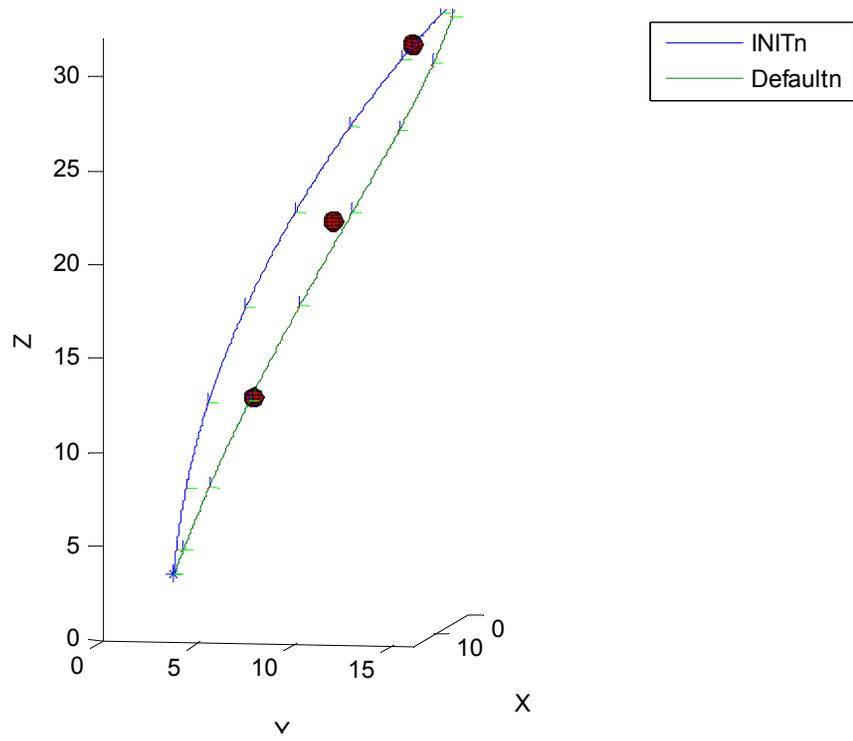
Path for CS 5, OS 1



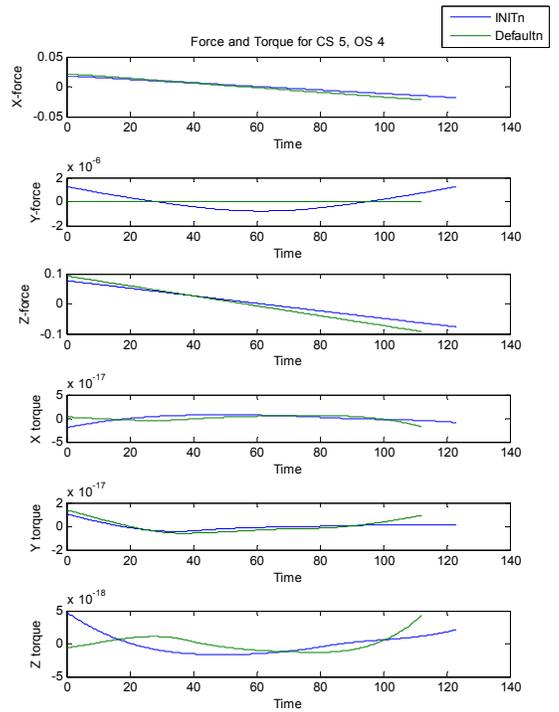
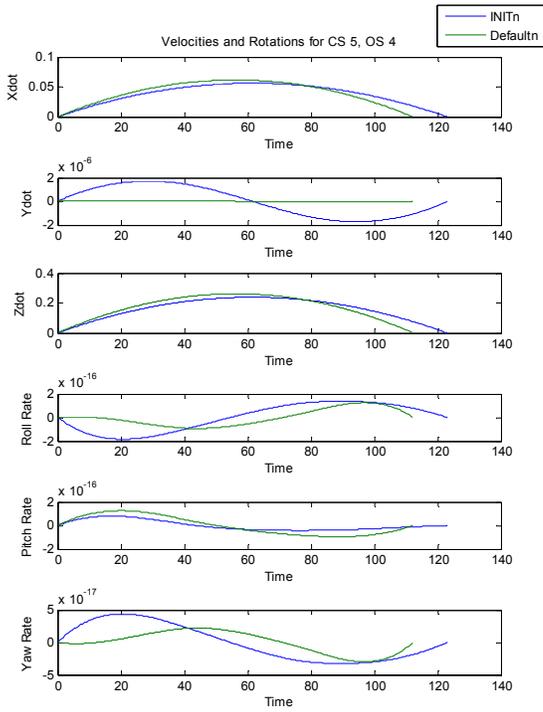
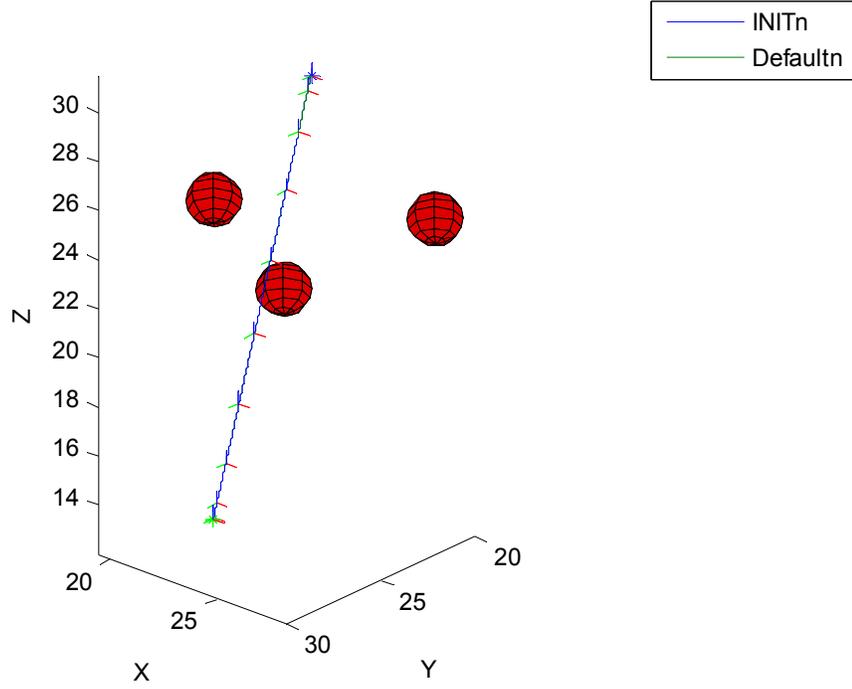
Path for CS 5, OS 2



Path for CS 5, OS 3



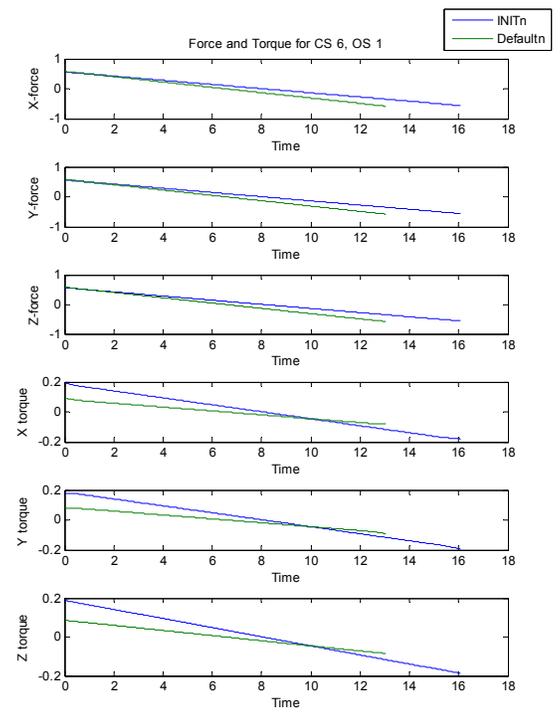
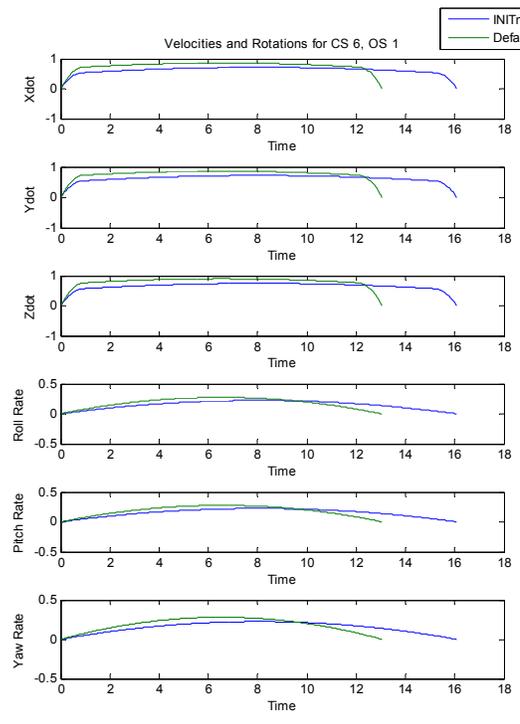
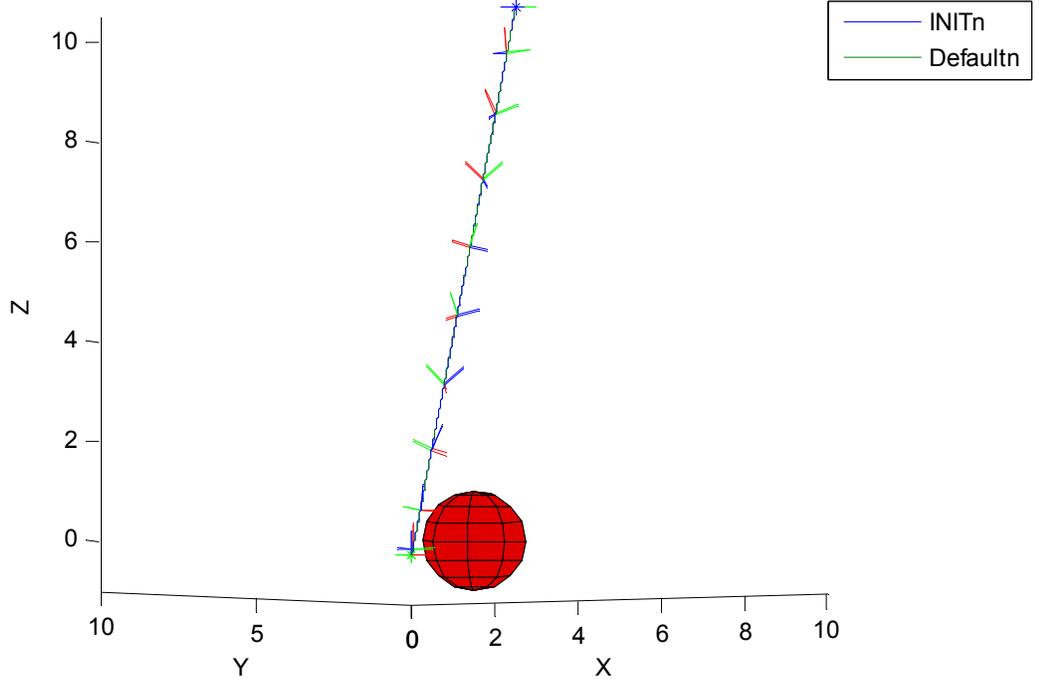
Path for CS 5, OS 4



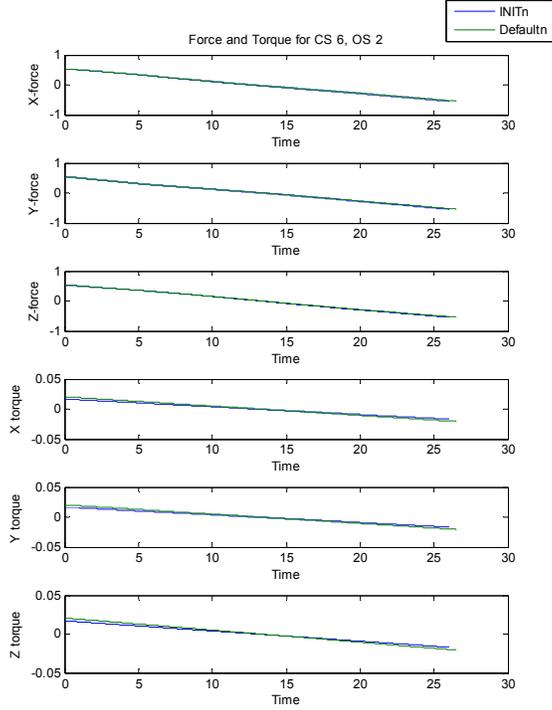
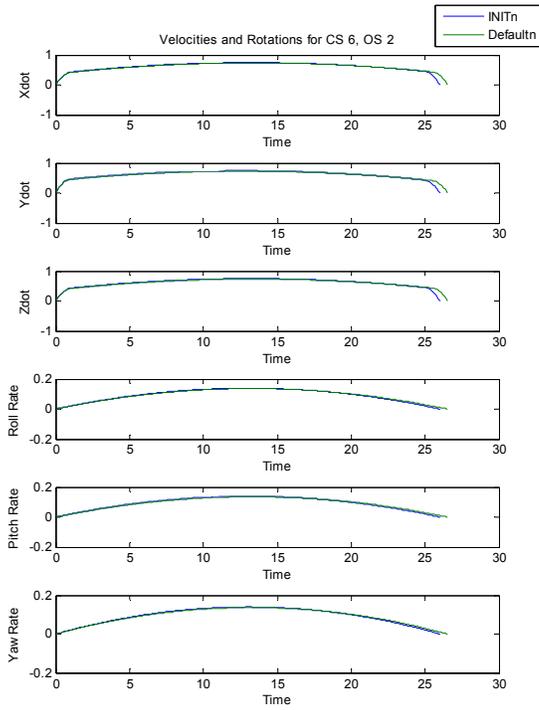
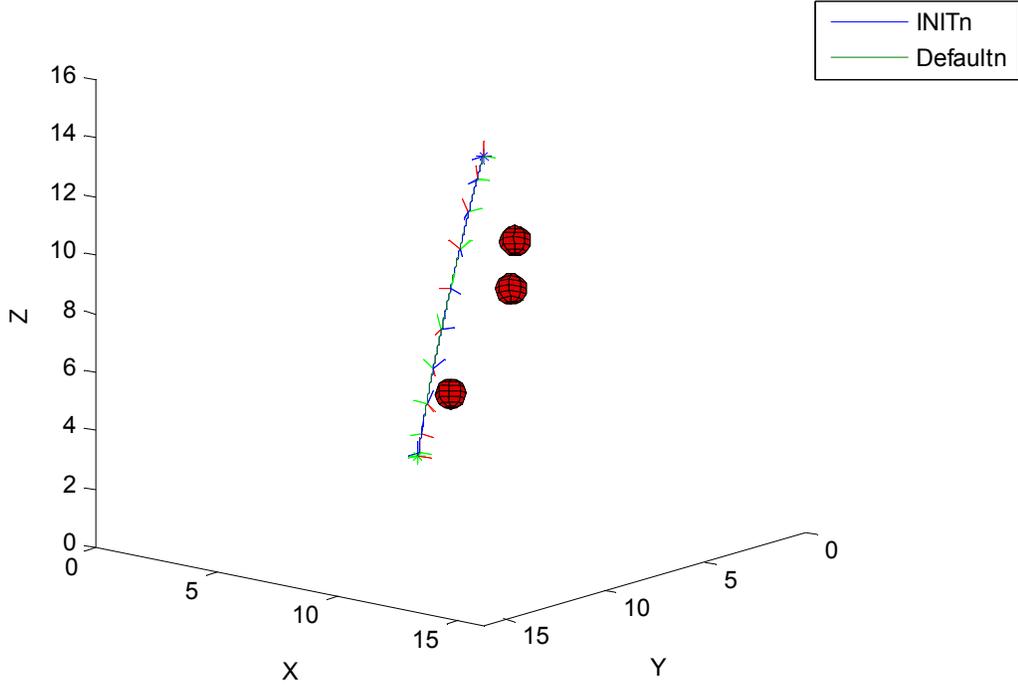
Constraint Set 5 was identical to CS 4, with “moderately safely” added to \mathbf{S}^0 . “Safely” includes a “medium high *min-sep*” in its definition, which is defuzzified to a *min-sep* between 1.0 and 2.0 m. Like CS 4, the solution cycled back and forth between high and low values for the time weight; in this case, *LIM* was overall higher, leading to the arcs seen in OS 2 and OS 3. This does not affect the path in OS 1 at all, as it is already sufficiently far away from the obstacle; the path and trajectory are almost identical to those found for CS 4.

It seems strange at first that CS 5, with more constraints, would converge for OS 4 while CS 4 would not. We wondered if the greater *min-sep* could be the cause – if initialized to a higher value, might the path have stayed farther away from solutions near constraint boundaries, which are computationally difficult for BVP4C2? But the path in CS 5 shows no significant deflection as it passes through the formation of obstacles. And if this were the case, then the default case would still have failed to converge, as it began with the same *LIM* in all runs. We can only assume some numeric fluke akin to a symmetry in the problem space, where the solver was faced with two identical-cost variations and cannot select between them.

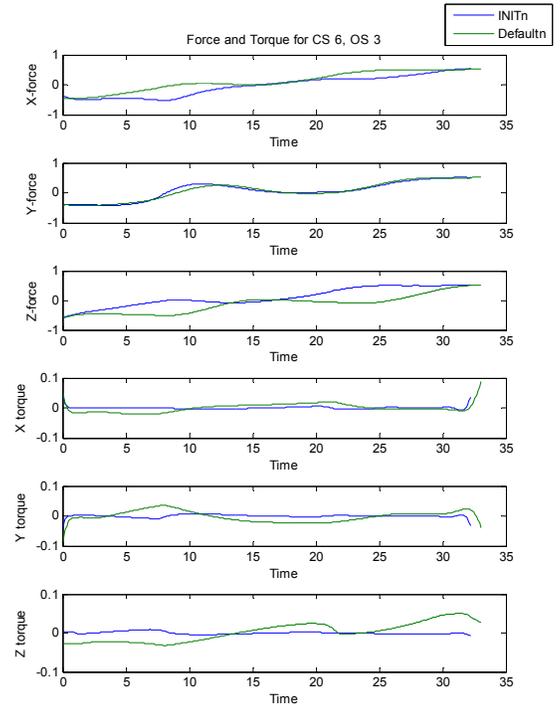
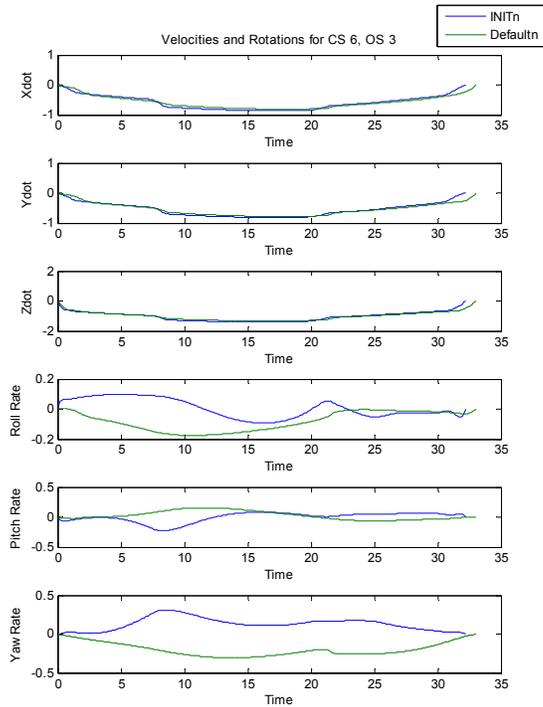
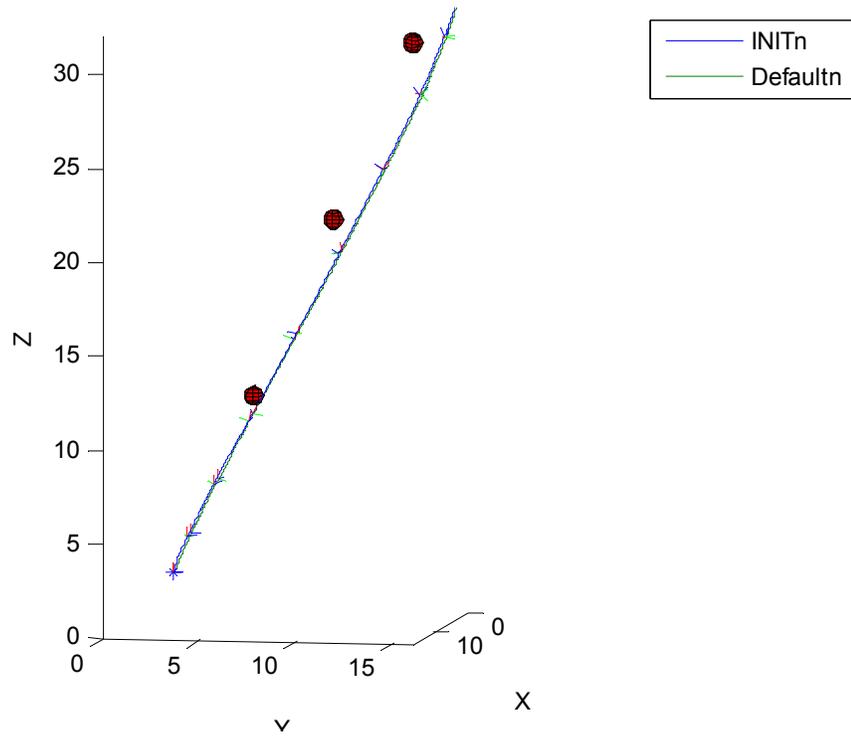
Path for CS 6, OS 1



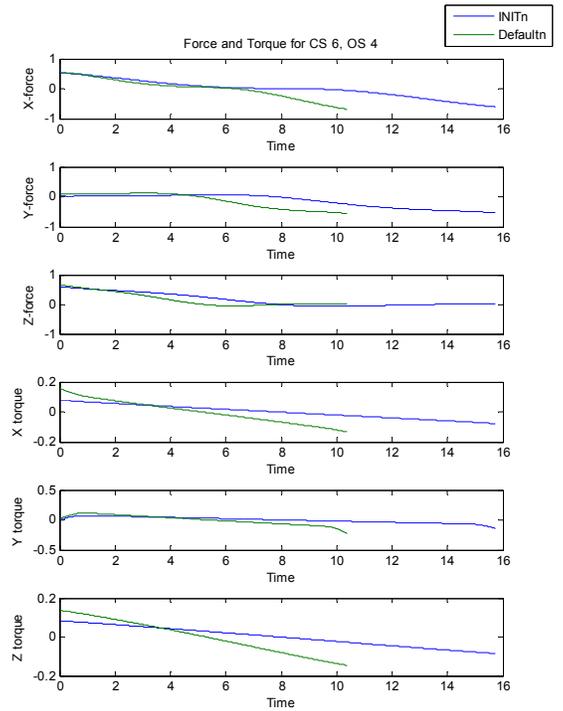
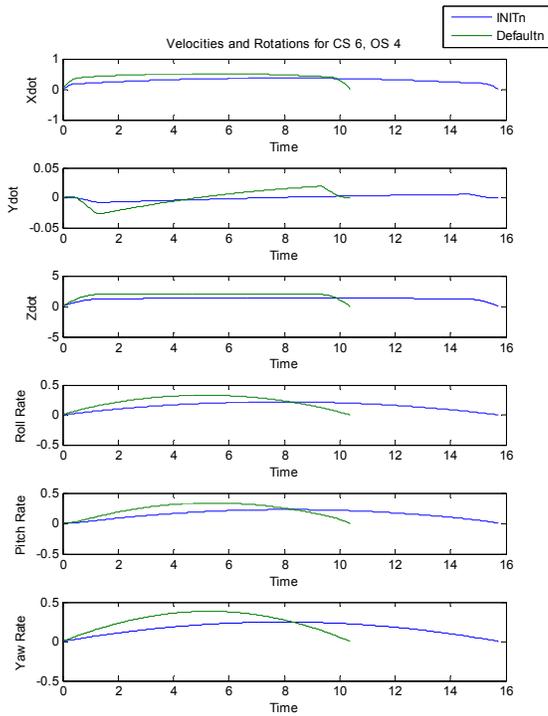
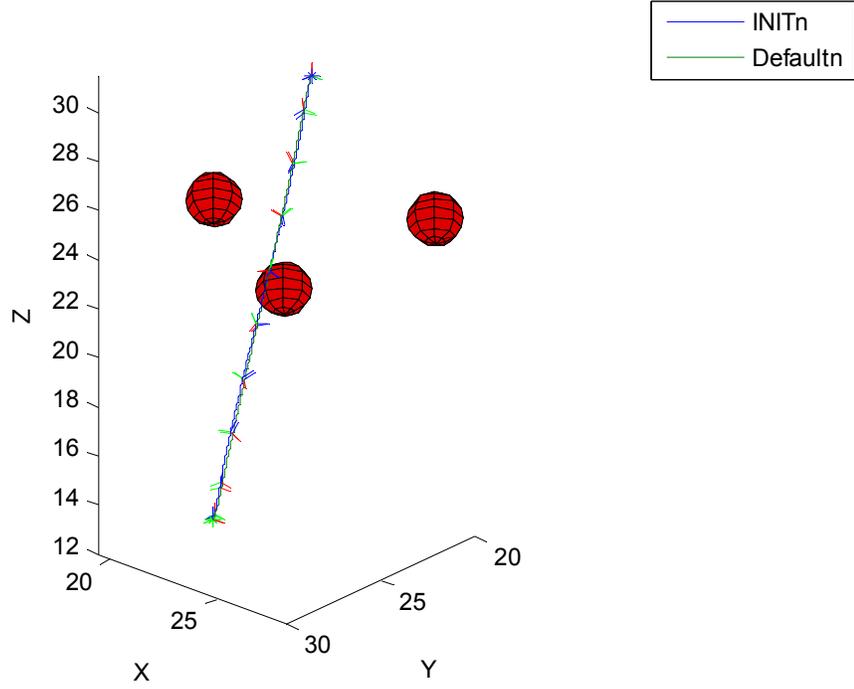
Path for CS 6, OS 2



Path for CS 6, OS 3



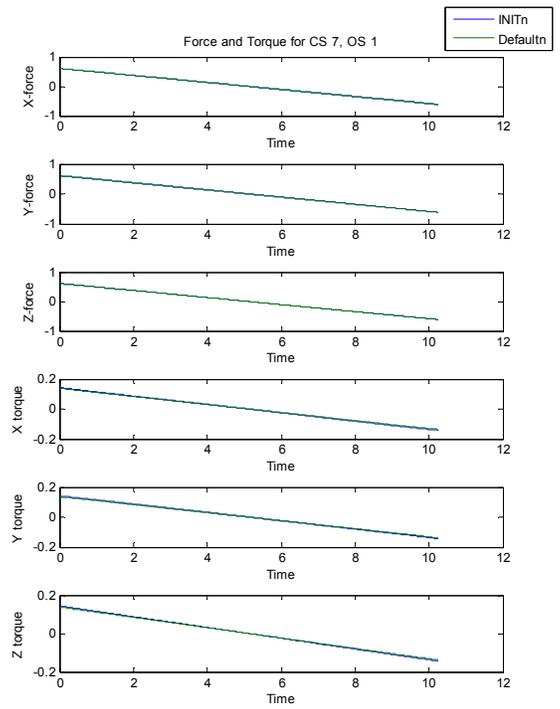
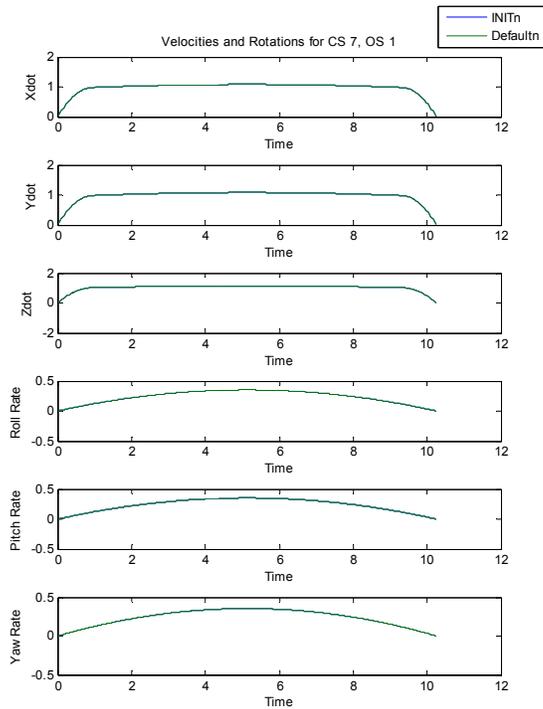
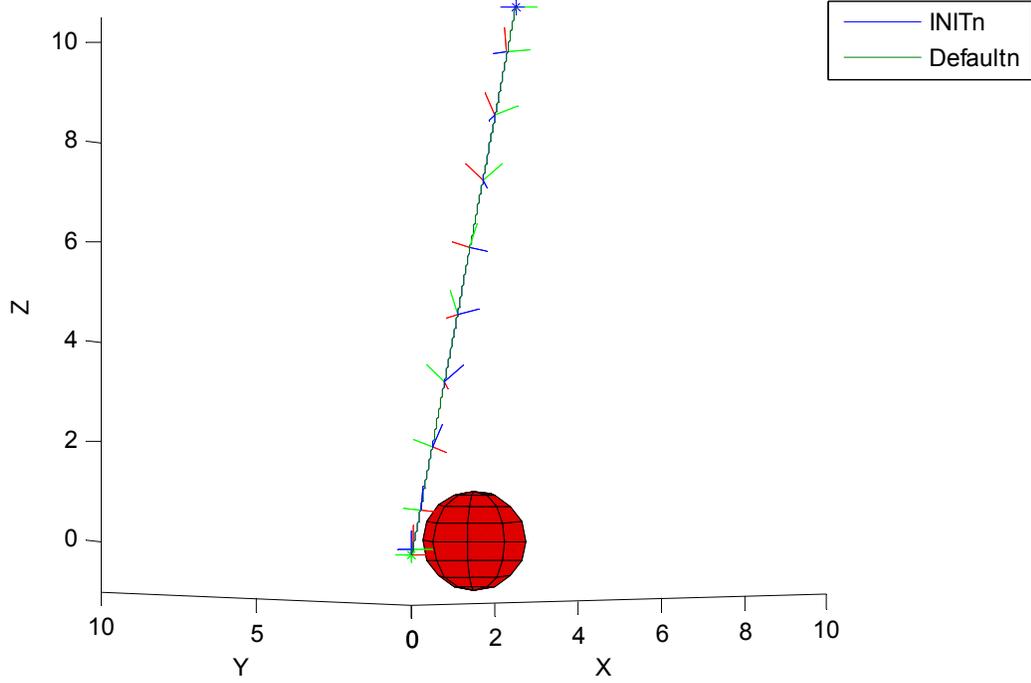
Path for CS 6, OS 4



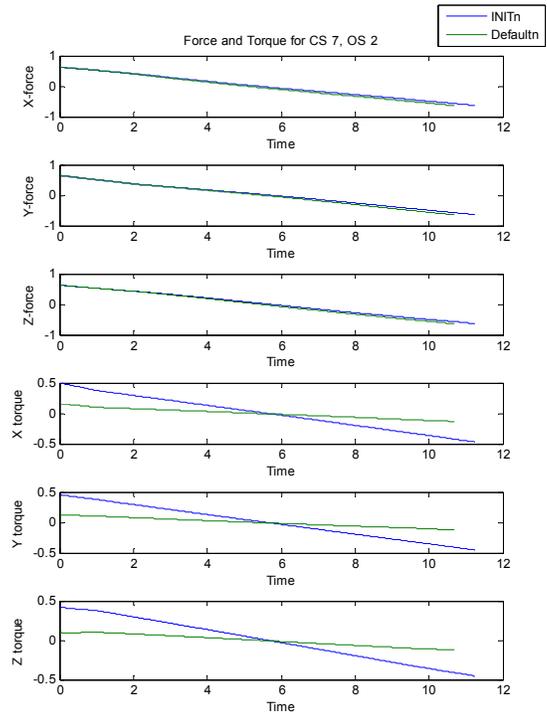
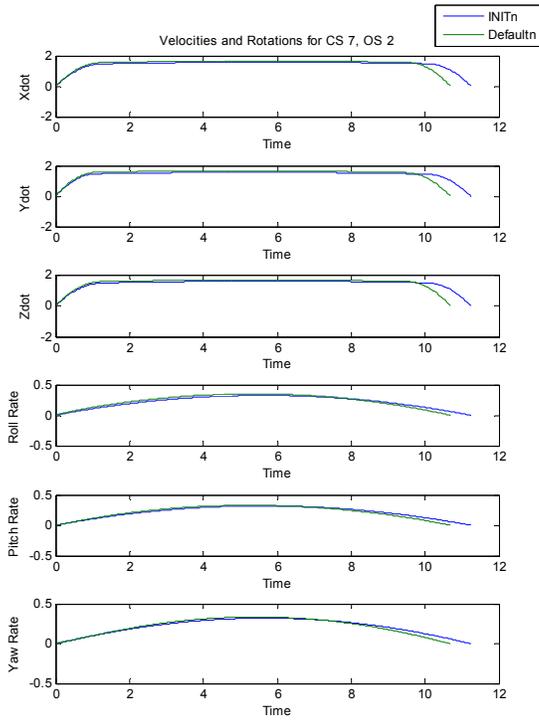
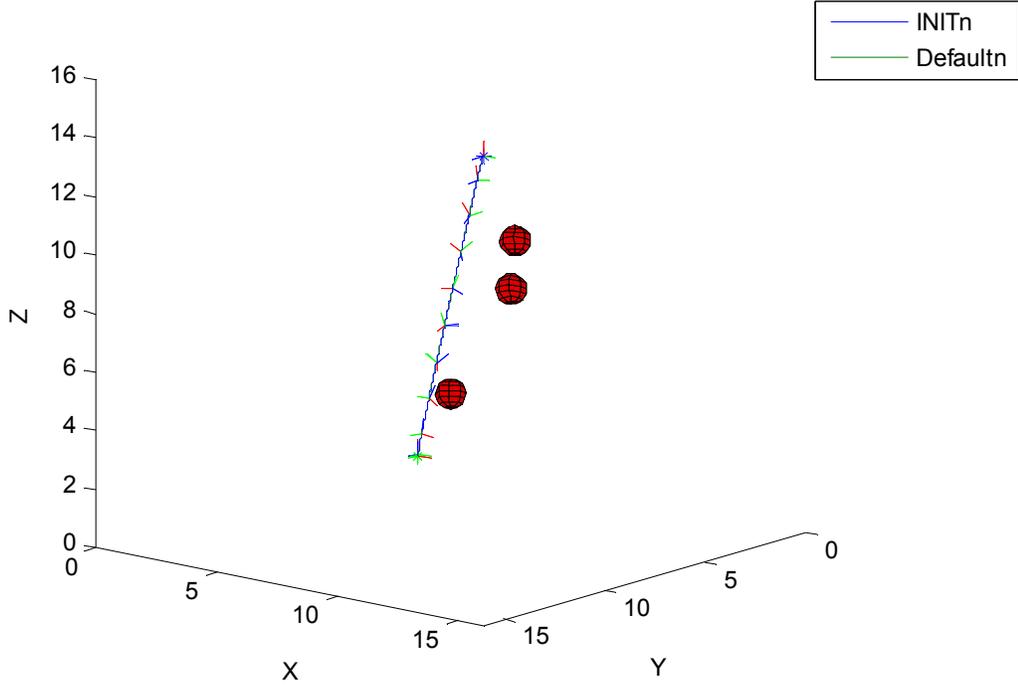
Constraint Set 6 was to be “very energy-saving,” a term we defined to mean having low *force* and low *torque*. At first, we ran these without commanding orientation changes; predictably, little to no torque was required. We re-ran the tests again, with a commanded orientation change, and with a bug in the torque *WADJ* rule fixed to get the results shown above.

The forces are not so low as in CS 4 and 5 because the constraint is directly on *force* rather than on *max-acc*; a low *max-acc* apparently corresponds to a very low *force*, these trajectories aren't as fuel-efficient as those. They are considerably more so than CS 1 and 3, which were “quick,” but are comparable to CS 2, “exceedingly efficiently,” which required low *force* as well. For the initialized cases, OS 4 needed only one iteration; OS 1 and 3 required only two. OS 2 had problems here and in the default case using enough torque to meet the low end of the “low *torque*” requirement, and so required six iterations, quitting when the time limit was reached (five iterations after the first). In the default cases, OS 1 and 4 required three and six iterations, respectively, achieving total success. Their weight adjustments were monotonic, steadily increasing torque weight and decreasing time weight. As seen in the Chapter 5 examples, most of the benefit was derived in the first correction. OS 2 caused the same problem as in the initialized case; the default starting weight for OS 3 put it in an unfortunate position, and it got caught in a iterative cycle that timed out after five runs.

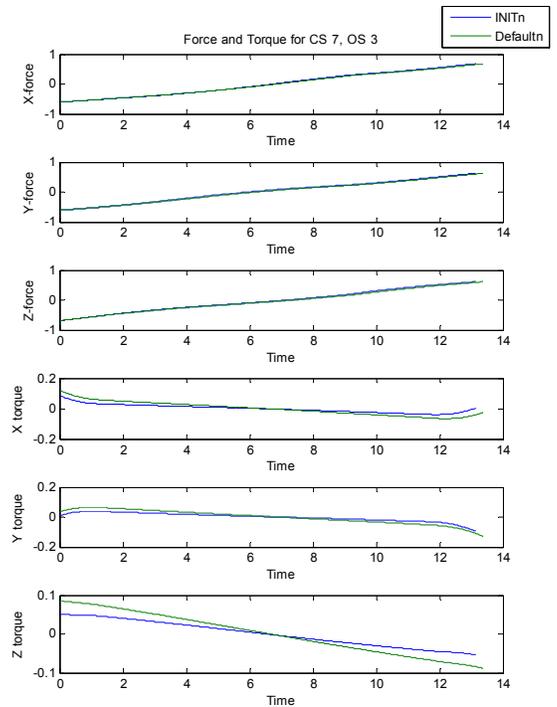
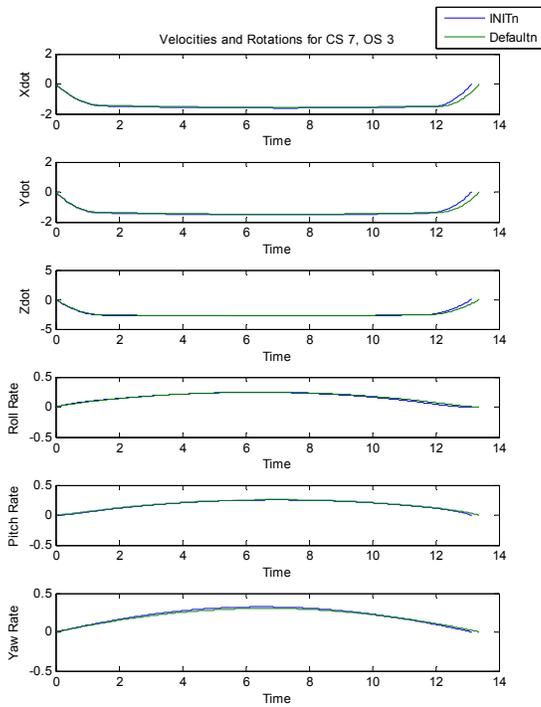
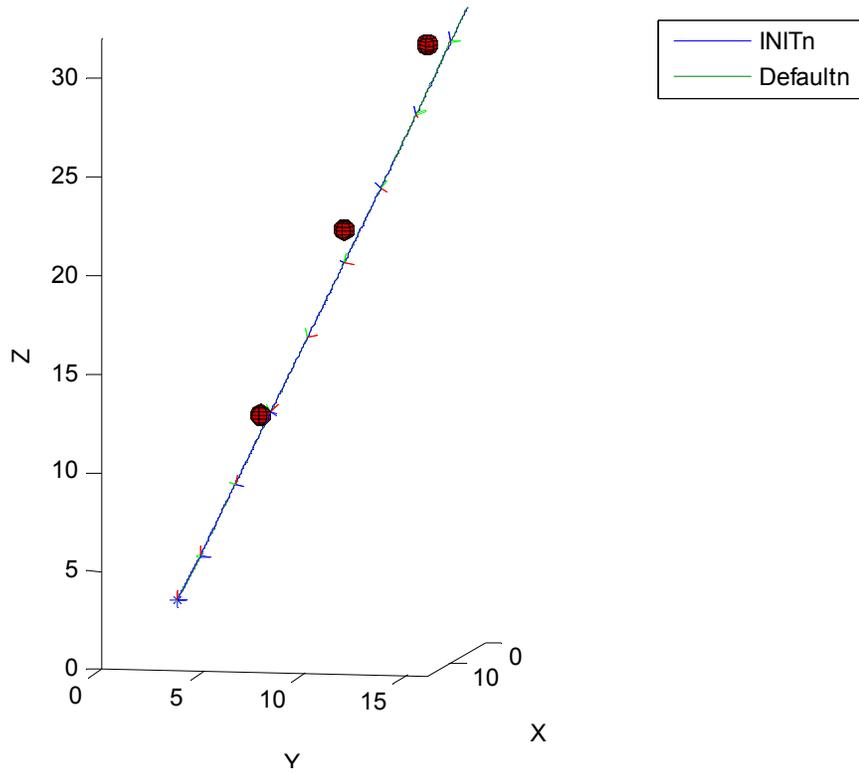
Path for CS 7, OS 1



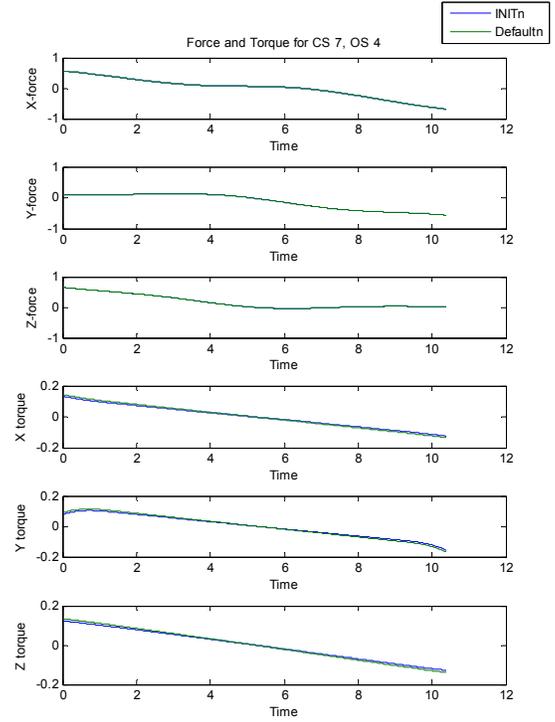
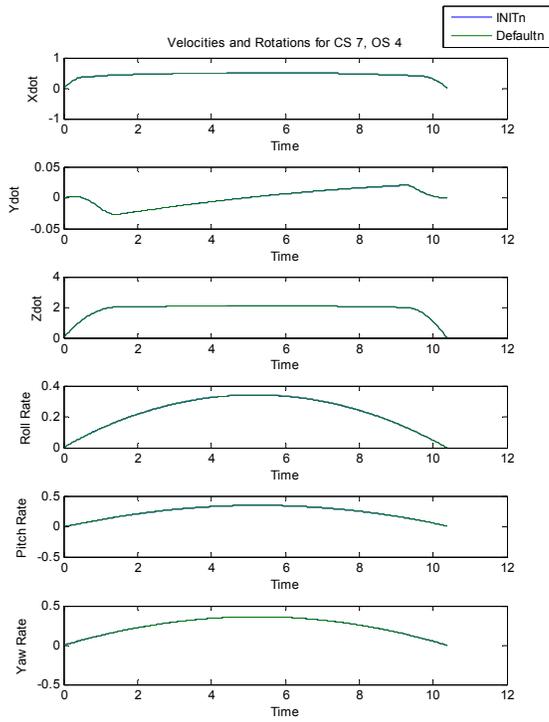
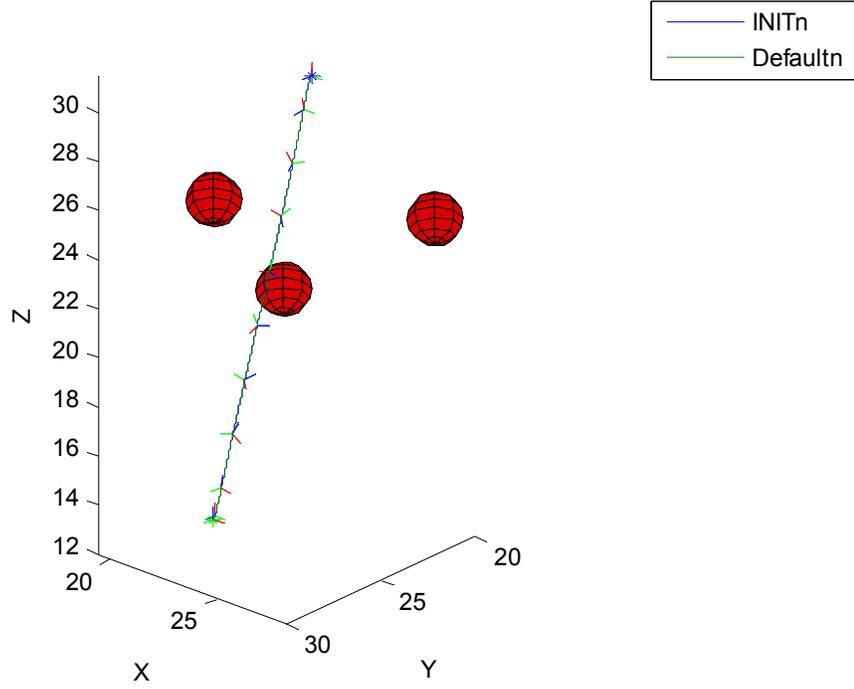
Path for CS 7, OS 2



Path for CS 7, OS 3



Path for CS 7, OS 4



Constraint Sets 7 and 8 were soft constraints on *torque* only, at “low” and “medium,” respectively. The maneuvers required little enough torque that CS 8 had difficulties meeting those requirements at all; the returned trajectories were so similar to the CS 7 solutions that they are not included separately here.

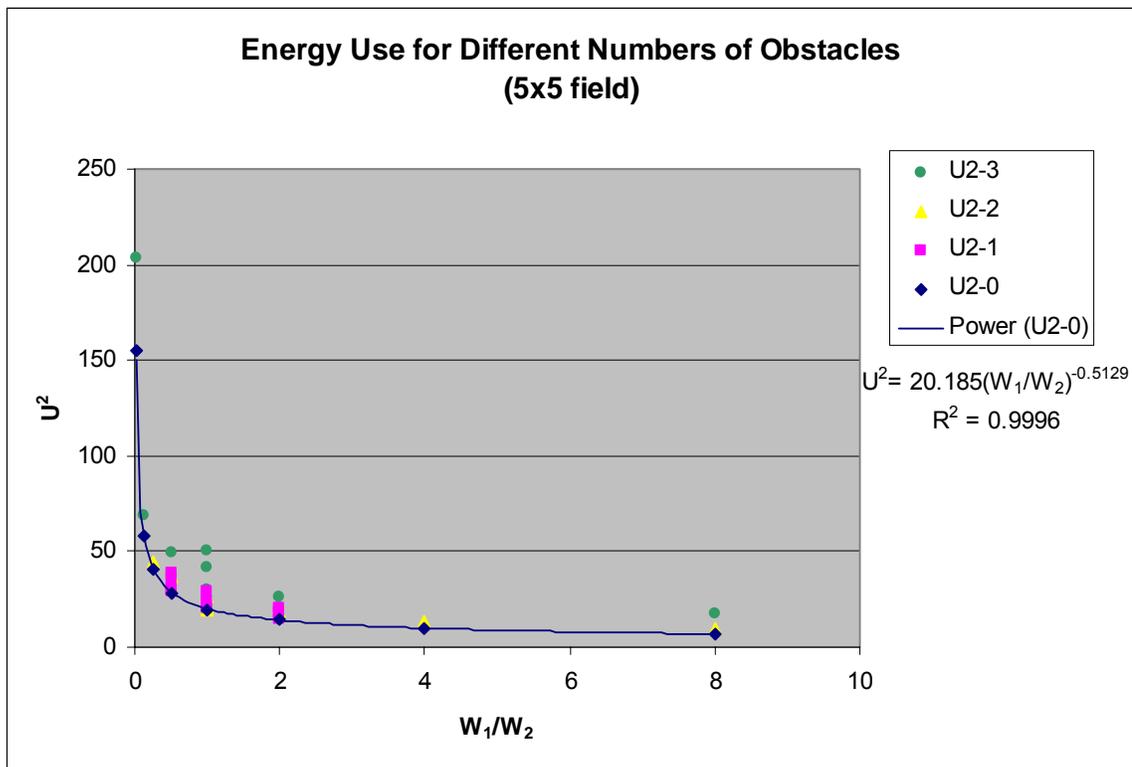
CS 7 was very successful; all the cases made the constraint. Either the first solution worked, or it used too much torque; the torque weight was smoothly adjusted up until the feature value dropped sufficiently to make the limit. In CS 8, the first solutions were all too low, and the weight was adjusted down, typically to a point where it was below our granularity for detecting weight loops; when it was decreased again, both attempts counted as zero and were detected as a loop. OS 1 and 2 failed for CS 8, while OS 3 and 4 eventually succeeded.

Appendix C: *WADJ* Heuristic Graphs and Fuzzy Rule

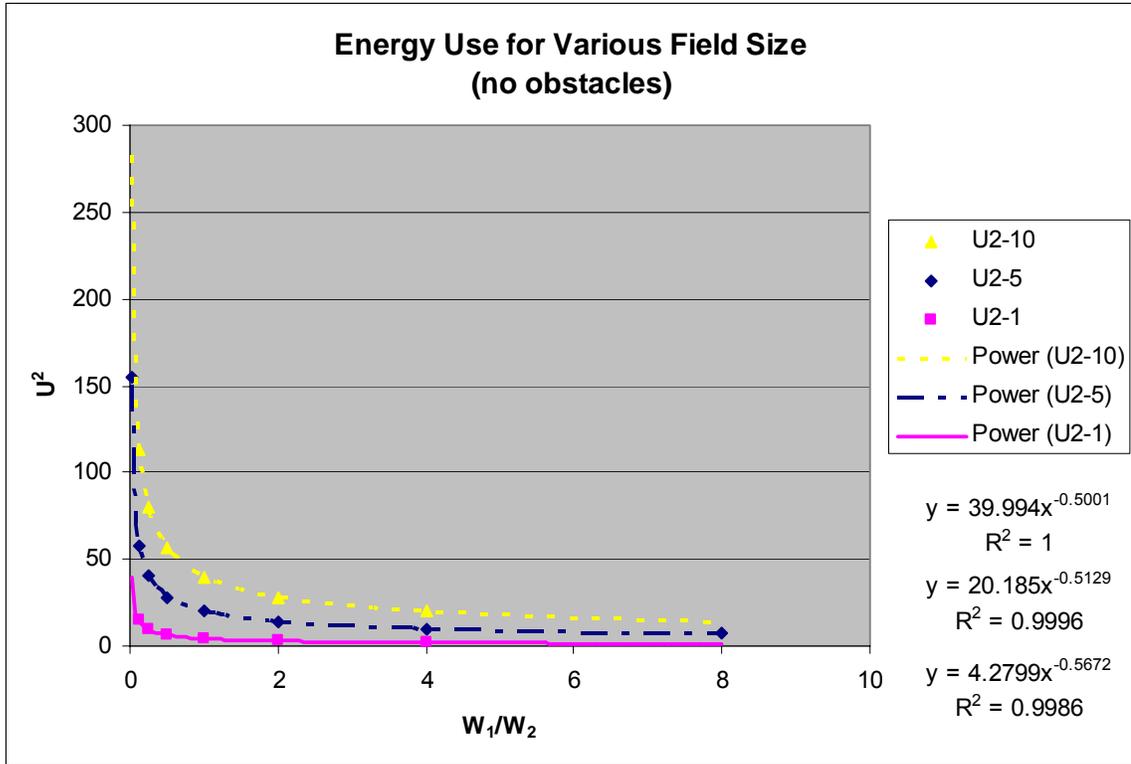
Definitions

2DOF *WADJ* Heuristics

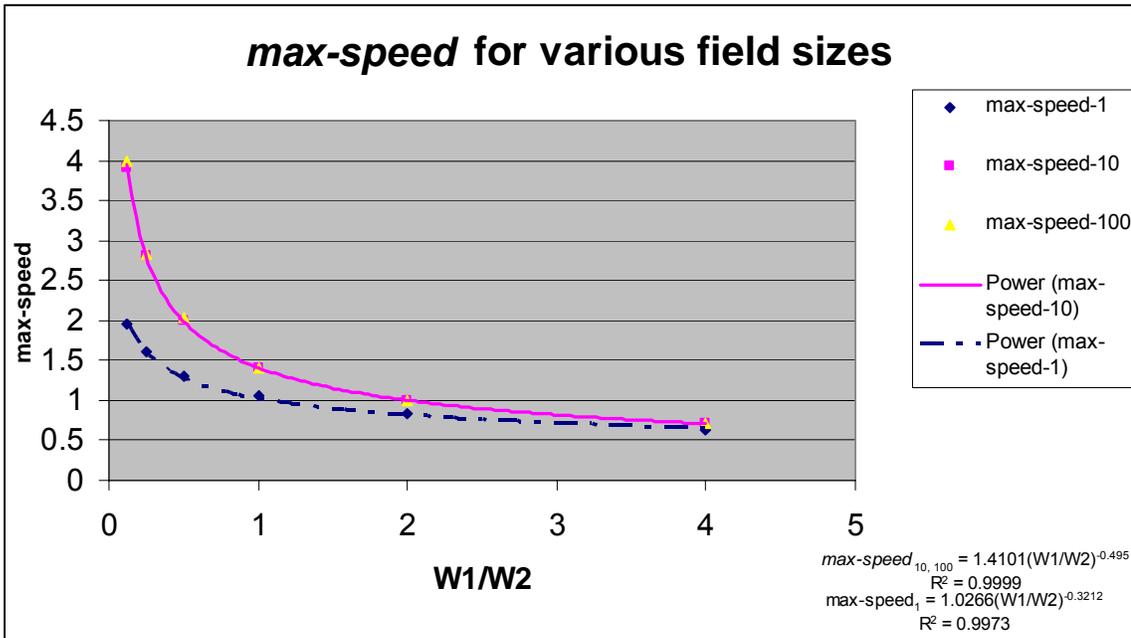
In this early *WADJ* graph; the *energy* feature is called U2, after its representation as a 2-norm in the cost functional. Data is for an empty field (U2-0) and for 1, 2, and 3 obstacles (U2-1, U2-2, U2-3). $W_1/W_2 = \{2^{-3}, 2^{-2}, \dots, 2^2, 2^3\}$.



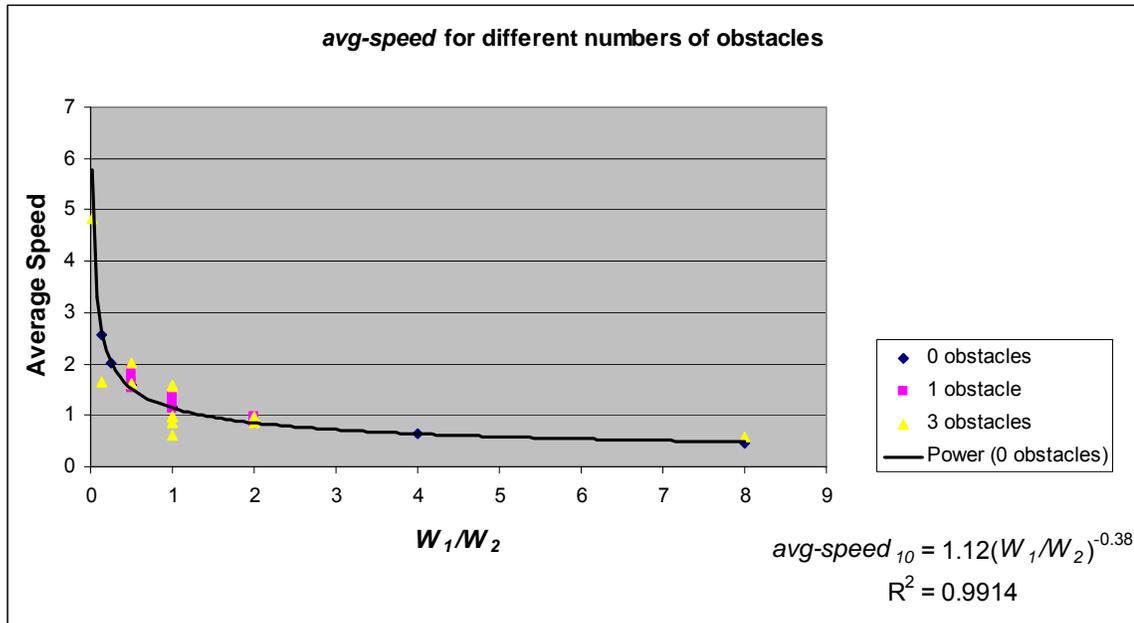
This shows how the *energy* heuristic exponent changes with field size. 2m, 10m, and 20m square fields were used (noted in the legend as U2-1, U2-5, and U2-10, because the coordinates in each went from (e.g.) (-5, -5) to (5, 5)).



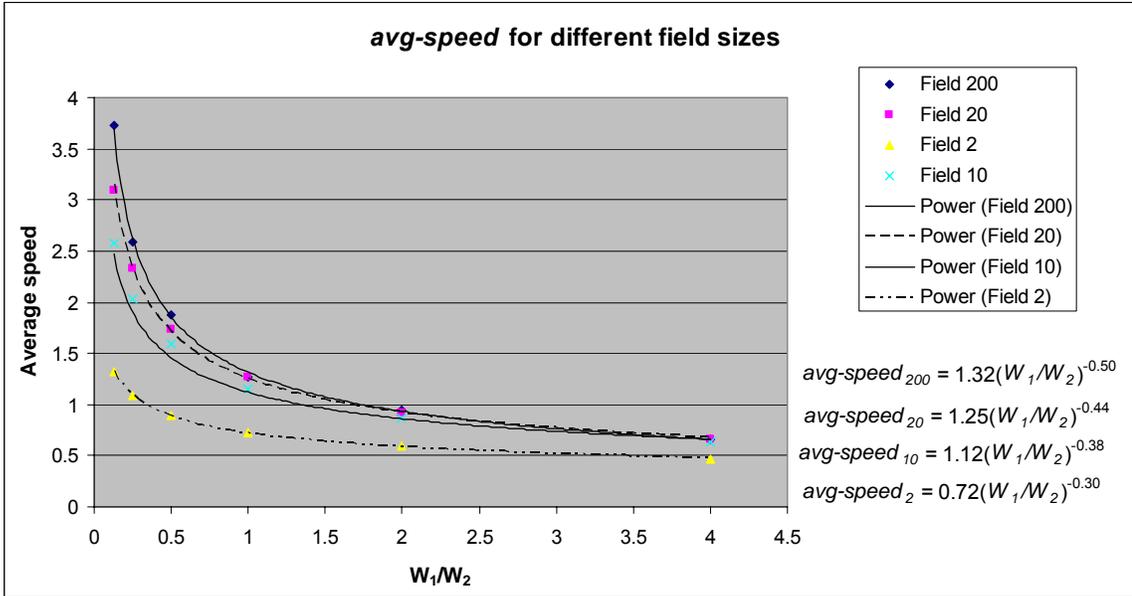
This shows the *max-speed* heuristic for 2m, 20m, and 200m square fields. Again the indices (max-speed-1, etc.) refer to the corner coordinates used in the runs.



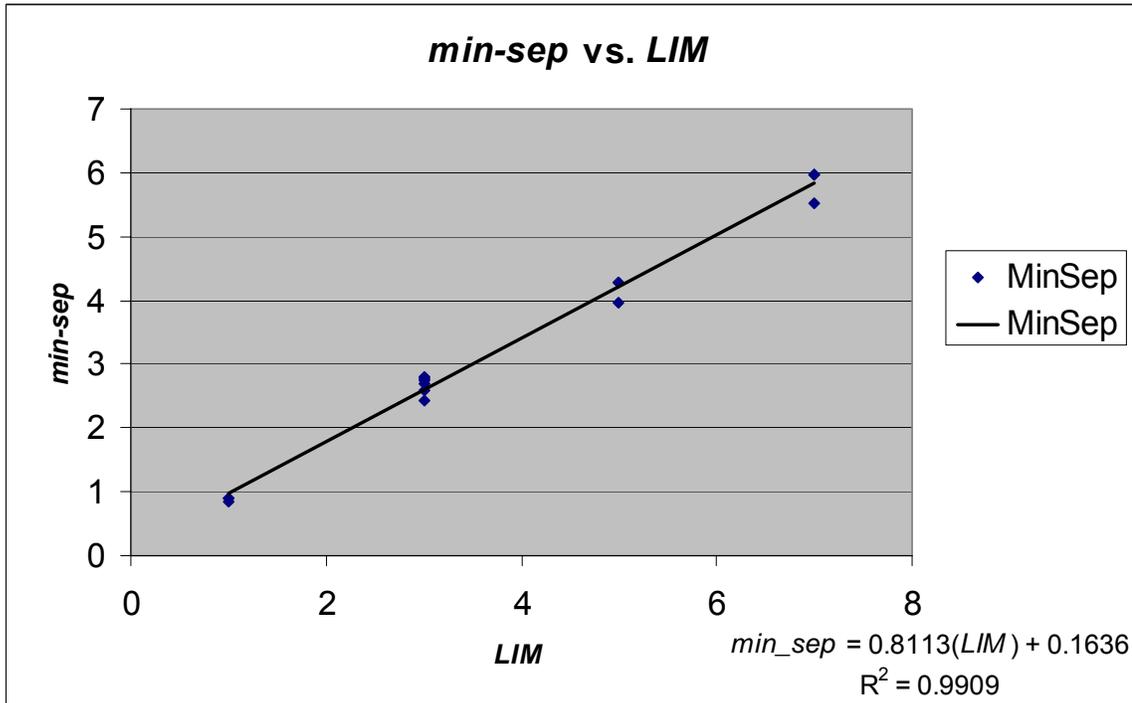
This graph shows the *avg-speed* heuristic degrading in the presence of obstacles. This is Figure 10 in the text.



This graph shows the *avg-speed* heuristic for different field sizes. This is Figure 11 in the text, and the field size indicators were changed to reflect the actual field size rather than the local coordinates. So this shows corner-to-corner motion through 2m, 10m, 20m, and 200m square empty fields.



Min-sep varying with *LIM* for *min-sep* = {1, 3, 5, 7} and a single obstacle in a 10m square field.



2DOF Fuzzy Rule Definitions

Fuzzy rules were generated from *WADJ* data in the following fashion:

For time-based features (e.g., everything besides *min-sep*): We took the W_1/W_2 weight vector $\{2^{-3}, 2^{-2}, \dots, 2^2, 2^3\}$ to correlate to the midpoints of the fuzzy levels {very low, low, medium low, medium, medium high, high, very high}. The fuzzy triangles would be bounded in most cases by the adjacent weight. Thus $W_1/W_2 = 1$ was the “ideal” medium, but “medium” could range from $\frac{1}{2}$ to 2. Then the feature values in the *WADJ* data that resulted from these weights were taken as the values that defined the feature fuzzy triangles.

For *min-sep*: We had defaulted to a *LIM* of 3 in our 2DOF work and so that became “medium.” The rest of the *LIM* levels were defined fairly linearly from there. Since *min-sep* and *LIM* are related linearly, but the slope of the relationship is unknown at initialization, we correlated them directly so that “high *LIM*” will result in “high *min-sep*.”

W1/W2 Ratio			
Fuzzy Level	Low End	Center	High End
very low	0	0.15	0.25
low	0.125	0.21	0.5
medium low	0.25	0.43	1
medium	0.5	1	2
medium high	1	2.33	4
high	2	4.67	8
very high	4	7	100

The center values for W_1/W_2 should have been $\{2^{-3}, 2^{-2}, \dots, 2^2, 2^3\}$. The centroid computation required that they be expressed separately, as whole numbers, and divided

only at the end of the computation. Rather than express them exactly as $\{1/8, 1/4, \dots, 4/1, 8/1\}$, I approximated them with ratios $\{3/21, 3/14, 3/7, 3/3, 7/3, 14/3, 21/3\}$. While I believe at the time (late 2003 – early 2004) I had good reasons for choosing these numbers, I today have no idea why I did so. Given the overall fuzziness of the fuzzy rules, the lack of precision did not hurt the algorithms, but this should be fixed in any future version of the software.

LIM			
Fuzzy Level	Low End	Center	High End
very low	0	0.3	0.5
low	0.3	0.5	1
medium low	0.5	1	2
medium	1	2	3
medium high	2	3	4
high	3	4	5
very high	4	5	10

energy (J)			
Fuzzy Level	Low End	Center	High End
very low	0	7	10
low	7	10	15
medium low	10	15	20
medium	15	20	40
medium high	20	40	60
high	40	60	100
very high	60	100	200

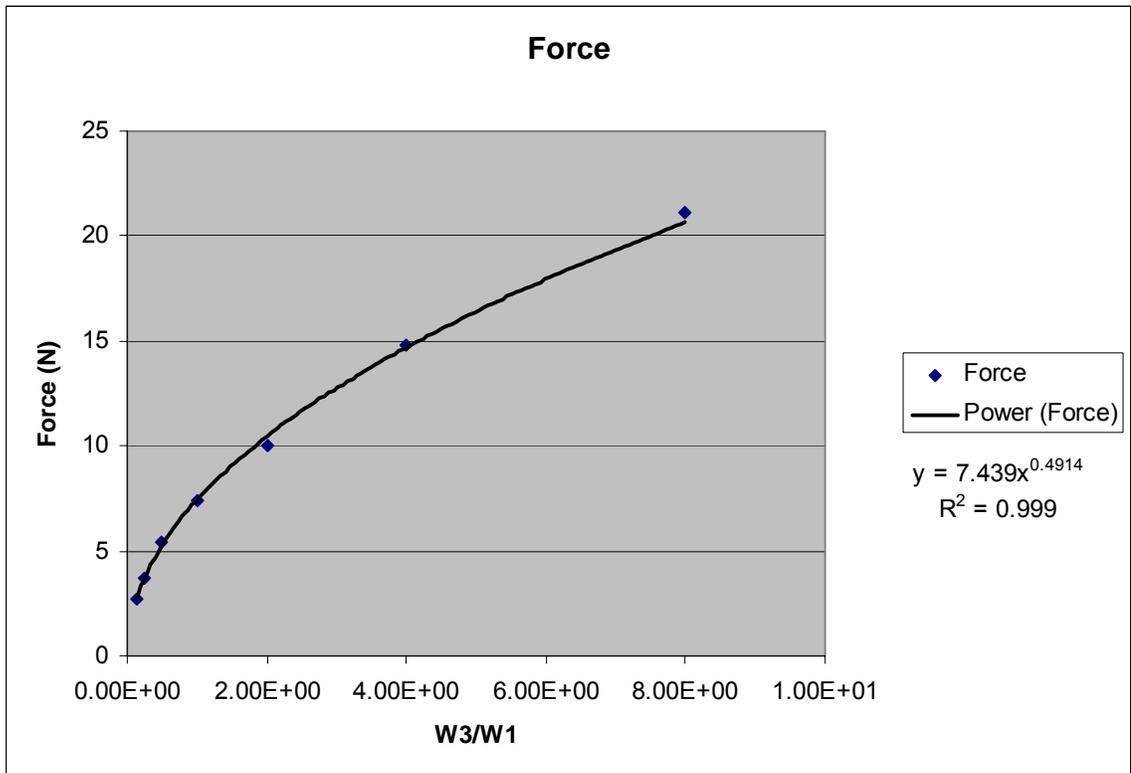
max-speed (m/s)			
Fuzzy Level	Low End	Center	High End
very low	0	0.5	1
low	0.5	1	1.5
medium low	1	1.5	2
medium	1.5	2	3
medium high	2	4	6
high	4	6	8
very high	7	9	20

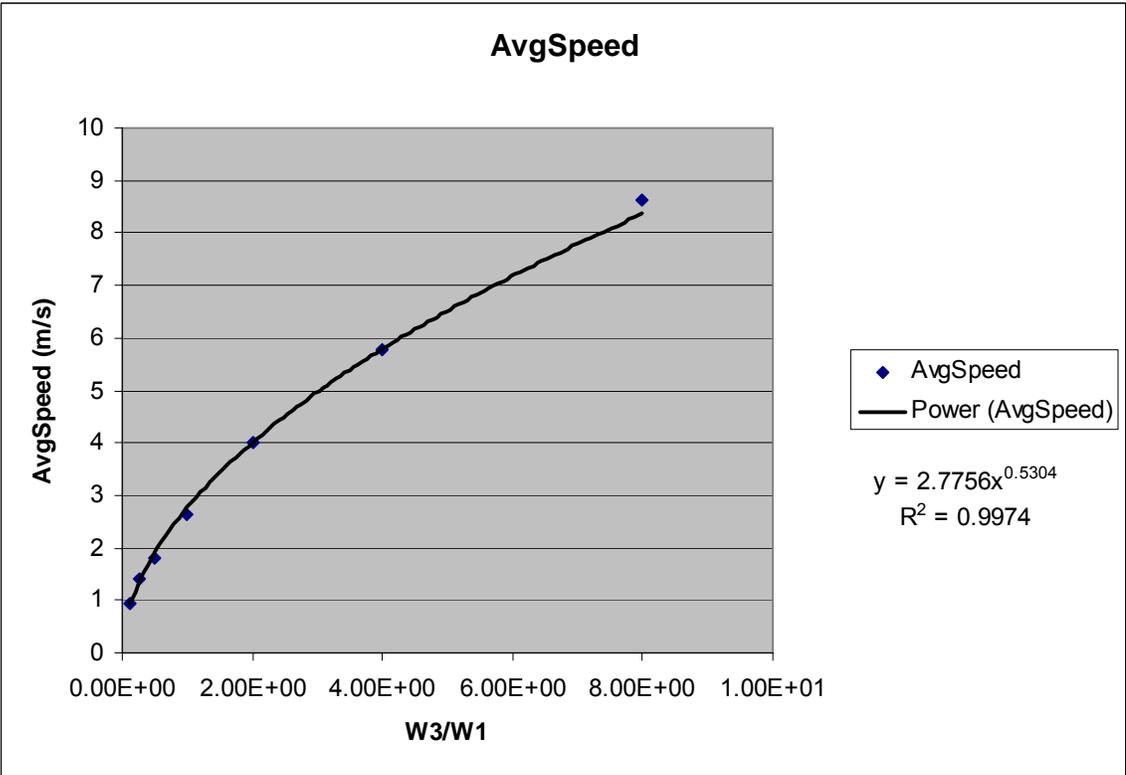
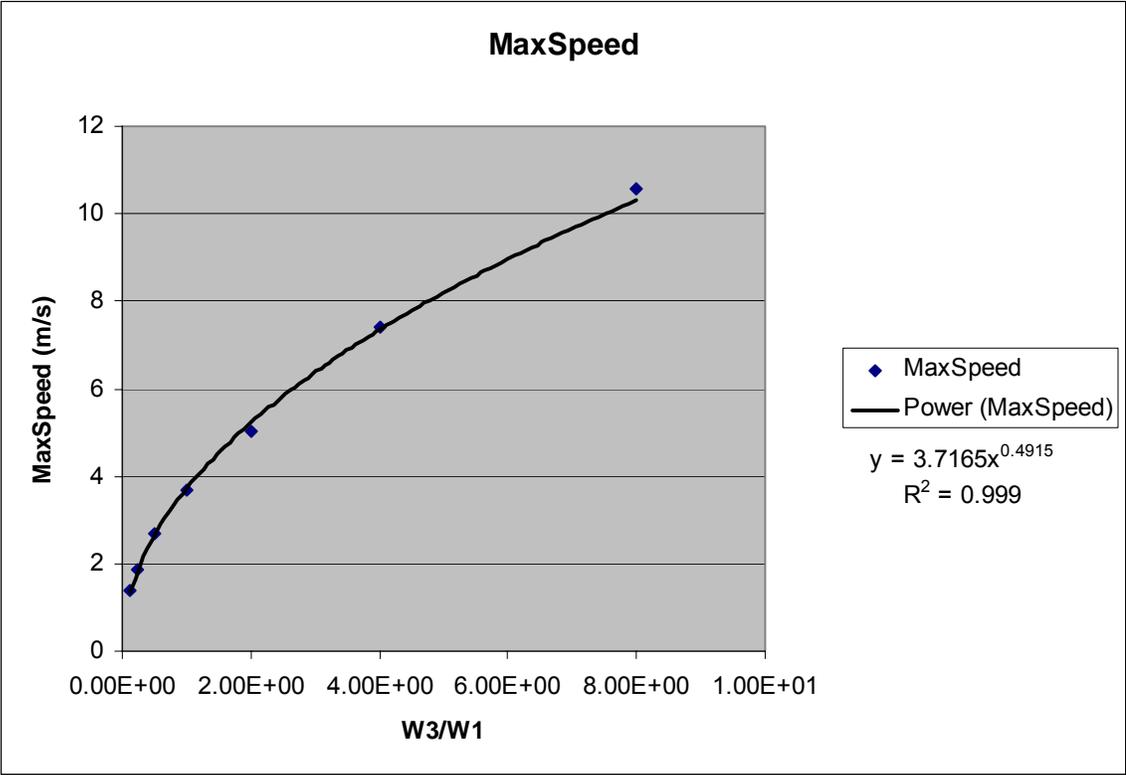
avg-speed (m/s)			
Fuzzy Level	Low End	Center	High End
very low	0	0.333	0.667
low	0.333	0.667	1
medium low	0.667	1	1.333
medium	1	1.333	2
medium high	1.333	2	4
high	2.667	4.667	5.333
very high	4.667	6	20

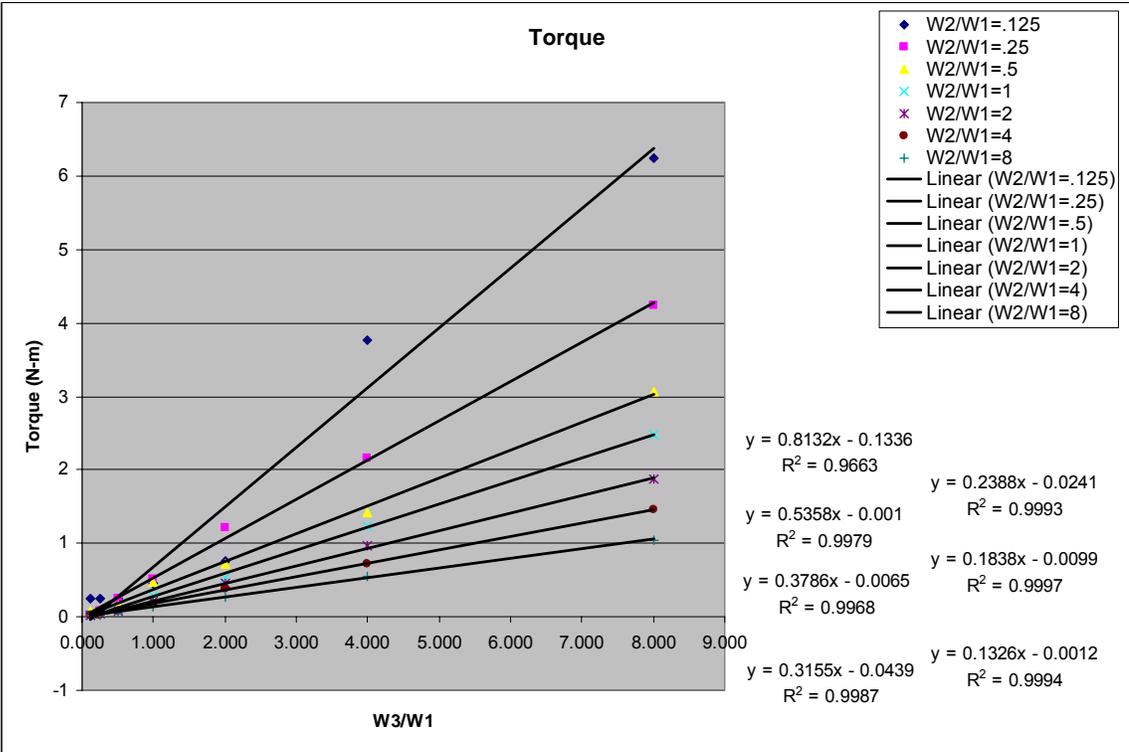
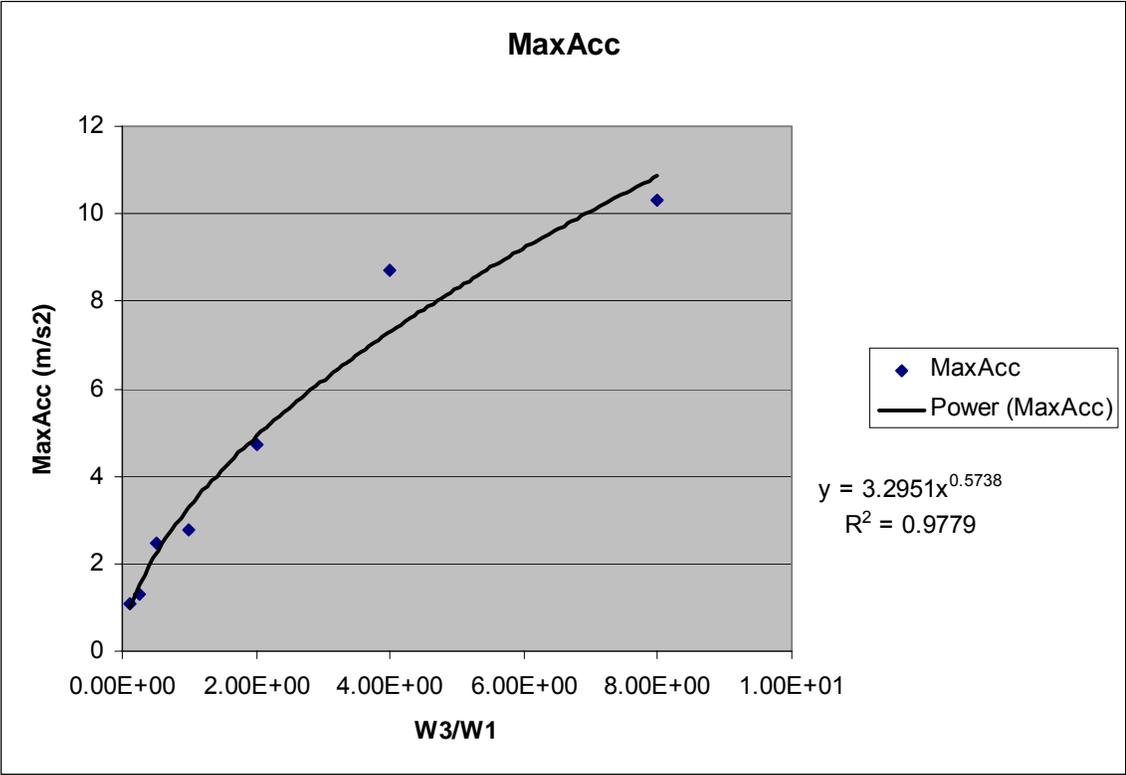
max-acc (m/s ²)			
Fuzzy Level	Low End	Center	High End
very low	0	0.333	0.667
low	0.333	0.667	1
medium low	0.667	1	1.333
medium	1	1.333	2
medium high	1.333	2	4
high	2.667	4.667	5.333
very high	4.667	6	20

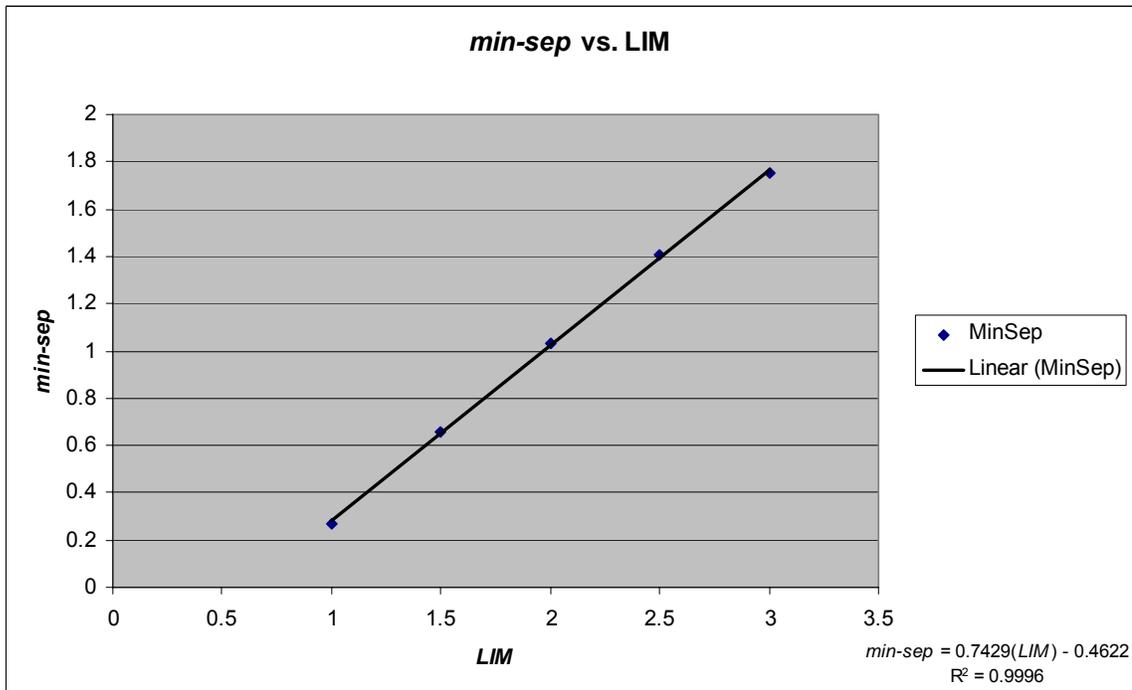
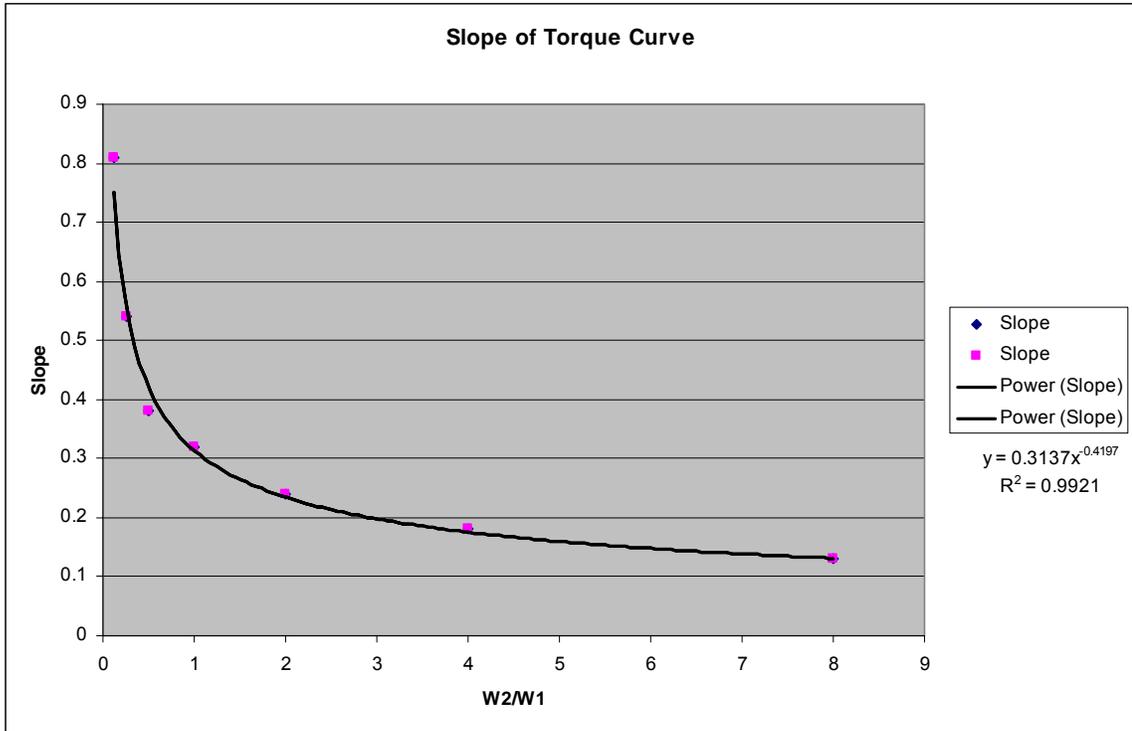
min-sep (m)			
Fuzzy Level	Low End	Center	High End
very low	0	0.3	0.5
low	0.3	0.5	1
medium low	0.5	1	2
medium	1	2	3
medium high	2	3	4
high	3	4	5
very high	4	5	10

6DOF *WADJ* Heuristics









6DOF Fuzzy Rule Definitions

W2/W1 Ratio			
Fuzzy Level	Low End	Center	High End
very low	0	0.125	0.25
low	0.125	0.25	0.5
medium low	0.25	0.5	1
medium	0.5	1	2
medium high	1	2	4
high	2	4	8
very high	4	8	100

W3/W1 Ratio			
Fuzzy Level	Low End	Center	High End
very low	0	0.125	0.25
low	0.125	0.25	0.5
medium low	0.25	0.5	1
medium	0.5	1	2
medium high	1	2	4
high	2	4	8
very high	4	8	100

LIM			
Fuzzy Level	Low End	Center	High End
very low	0	0.3	0.5
low	0.3	0.5	1
medium low	0.5	1	2
medium	1	2	3
medium high	2	3	4
high	3	4	5
very high	4	5	10

force (N)			
Fuzzy Level	Low End	Center	High End
very low	0	2.7	3.7
low	2.7	3.7	5.4
medium low	3.7	5.4	7.4
medium	5.4	7.4	10.1
medium high	7.4	10.1	14.8
high	10.1	14.8	21.1
very high	14.8	21.1	60

torque (N-m)			
Fuzzy Level	Low End	Center	High End
very low	0	0.05	0.1
low	0.05	0.1	0.2
medium low	0.1	0.2	0.6
medium	0.2	0.6	1.5
medium high	0.6	1.5	4
high	1.5	4	6
very high	4	6	20

max-speed (m/s)			
Fuzzy Level	Low End	Center	High End
very low	0	1.4	1.9
low	1.4	1.9	2.7
medium low	1.9	2.7	3.7
medium	2.7	3.7	5
medium high	3.7	5	7.4
high	5	7.4	10.6
very high	7.4	10.6	30

avg-speed (m/s)			
Fuzzy Level	Low End	Center	High End
very low	0	0.9	1.4
low	0.9	1.4	1.8
medium low	1.4	1.8	2.6
medium	1.8	2.6	4
medium high	2.6	4	5.8
high	4	5.8	8.6
very high	5.8	8.6	20

max-acc (m/s ²)			
Fuzzy Level	Low End	Center	High End
very low	0	1	1.3
low	1	1.3	2.5
medium low	1.3	2.5	2.8
medium	2.5	2.8	4.7
medium high	2.8	4.7	8.7
high	4.7	8.7	10.3
very high	8.7	10.3	30

min-sep (m)			
Fuzzy Level	Low End	Center	High End
very low	0	0.1	0.25
low	0.1	0.25	0.5
medium low	0.25	0.5	1
medium	0.5	1	1.5
medium high	1	1.5	2
high	1.5	2	2.5
very high	2	2.5	5

References

- [1] Johan Andersson. “A survey of multiobjective optimization in engineering design.” Technical report LiTH-IKP-R-1097, Department of Mechanical Engineering, Linköping University, Linköping, Sweden, 2000.
- [2] Bart Kosko and Satoru Isaka. “Fuzzy logic.” *Scientific American* 269, pages 76-81, July 1993.
- [3] Vilém Novák. “Fuzzy logic: applications to natural language.” In Stuart C. Shapiro (editor), *Encyclopedia of Artificial Intelligence, Second Edition*,, pages 515-521, John Wiley & Sons, New York, 1992.
- [4] Dennis Perzanowski, Alan C. Schultz and William Adams. “Integrating natural language and gestures in a robotics domain.” In *Proceedings of the IEEE International Symposium on Intelligent Control*, pages 247-252. National Institute of Standards and Technology, Gaithersburg, MD, 1998.
- [5] Barbara Tversky and Paul U. Lee. “How space structures language.” In C. Freksa, C. Habel, and K. F. Wender (editors). *Spatial cognition. An Interdisciplinary Approach to Representing and Processing Spatial Knowledge*. pages 157-175, Springer-Verlag, Berlin, 1998.
- [6] Alicia Abella and John R. Kender. “Qualitatively describing objects using spatial prepositions.” In *Proceedings of IEEE Workshop on Qualitative Vision*, pages 33-38, New York, 1993.
- [7] Patrick Oliver, Toshiyuki Maeda and Jun-ichi Tsujii. “Automatic depiction of spatial descriptions.” *Spatial Reasoning* 2, pages 1405-1410, 1994.

- [8] Amitabha Mukerjee. “Neat vs. scruffy: a survey of computational models for spatial expressions.” In Patrick Olivier and Klaus-Peter Gapp (editors), *Computational Representation and Processing of Spatial Expressions*, Kluwer Academic Press, Boston, MA, 1998.
- [9] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots.” *The International Journal of Robotics Research* 5(1), pages 90-98, 1986.
- [10] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.
- [11] Ronald C. Arkin. *Behavior-Based Robotics*. The MIT Press, Cambridge, MA, 1998.
- [12] Juan C. Velásquez. “An emotion-based approach to robotics.” In *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robotics and Systems*, pages 235 – 240, Kyongju, Korea, 1999.
- [13] Ashwin Ram, Ronald Arkin, Gary Boone and Michael Pearce. “Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation.” *Adaptive Behavior* 2(3), pages 277 – 304, 1994.
- [14] Maxim Likhachev, Michael Kaess, and Ronald C. Arkin. “Learning behavioral parameterization using spatio-temporal case-based reasoning.” in *Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA 2002)*, pages 1282 – 1289, Washington, DC, 2002.

- [15] Juan C. Santamaría and Ashwin Ram. “Learning of parameter-adaptive reactive controllers for robotic navigation.” In *Proceedings of the World Multiconference on Systems, Cybernetics, and Informatics (CSI '97)*, Caracas, Venezuela, 1997.
- [16] Eric Aaron, Harold Sun, Franjo Ivančić, and Dimitris Metaxas. “A hybrid dynamical systems approach to intelligent low-level navigation.” In *Proceedings of Computer Animation 2002*, 154-163, San Antonio, TX, 2002.
- [17] Siome Goldstein, Menelaos Karavelas, Dimitris Metaxas, Leonidas Guibas, Eric Aaron, and Ambarish Goswami. “Scalable nonlinear dynamical systems for agent steering and crowd simulation.” *Computers and Graphics* 25(6), 983-998, 2001.
- [18] Daniel Shapiro and Pat Langley. “Separating skills from preference: using learning to program by reward.” In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 570-577, Sydney, Australia, 2002.
- [19] Paolo Fiorini and Zvi Shiller. “Motion planning in dynamic environments using velocity obstacles.” *International Journal of Robotics Research*, 17(7), pages 760-772, 1998.
- [20] Zvi Shiller, F. Large and S. Sekhavat. “Motion planning in dynamic environments: obstacles moving along arbitrary trajectories.” In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2001)*, Seoul, Korea, 2001.
- [21] Jianwei Zang, Jörg Raczkowski and Andreas Herp. “Emulation of spline curves and its applications in robot motion control.” In *Proceedings of the IEEE Conference on Fuzzy Systems*, pages 831-836, 1994.

- [22] Jung-Hoon Hwang, Ronald C. Arkin and Dong-Soo Kwon. “Mobile robots at your fingertips: Bezier curve on-line trajectory generation for supervisory control.” In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, pages 1444-1449, Las Vegas, Nevada, 2003.
- [23] Steven M. LaValle and James J. Kuffner, Jr. “Randomized kinodynamic planning.” *International Journal of Robotics Research*, 20(5), pages 378-400, 2001.
- [24] Paolo Fiorini and Zvi Shiller. “Time optimal trajectory planning in dynamic environments.” *Journal of Applied Mathematics and Computer Science*, 7(2), pages 101-126, 1997.
- [25] Zvi Shiller and Yu-Rwei Gwo. “Dynamic motion planning of autonomous vehicles.” *IEEE Transactions on Robotics and Automation*, 7(2), pages 241-249, 1991.
- [26] Ian Garcia and Jonathan P. How. “Trajectory optimization for satellite reconfiguration maneuvers with position and attitude constraints.”
- [27] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.
- [28] Francis Ysidro Edgewood. *Mathematical Physics*. P. Keagan, London, 1881.
- [29] Vilfredo Pareto. *Cours D’Economie Politique*, volume I and II. F. Rouge, Lausanne, 1896.
- [30] Carlos A. Coello Coello. “Evolutionary multiobjective optimization: current and future challenges.” In Jose Benitez, Oscar Cordon, Frank Hoffmann and Rajkumar

- Roy (editors), *Advances in Soft Computing---Engineering, Design and Manufacturing*, pages 243--256, Springer-Verlag, New York, 2003.
- [31] Eckart Zitzler, Marco Laumanns and Stefan Bleuler. "A tutorial on evolutionary multiobjective optimization." In Xavier Gandibleux, Marc Sevaux, Kenneth Sörensen and Vincent T'kindt (editors), *Metaheuristics for Multiobjective Optimisation*, pages 3-37, Springer, Berlin, 2004.
- [32] Carlos M. Fonseca and Peter J. Fleming. "Multiobjective optimization and multiple constraint handling with evolutionary algorithms I: a unified formulation." Technical Report 564, University of Sheffield, Sheffield, UK, 1995.
- [33] Carlos A. Coello Coello. "EMOO Home Page."
<http://www.lania.mx/~ccoello/EMOO/> accessed 2005.
- [34] J. David Schaffer. "Multiple objective optimization with vector evaluated genetic algorithms." In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93-100, 1985.
- [35] W. Jacob, M. Gorges-Schleuter, and C. Blume. "Application of genetic algorithms to task planning and learning." In R. Männer and B. Manderick, (editors), *Parallel Problem Solving from Nature, 2nd Workshop*, pages 291-300, North Holland Publishing Company, Amsterdam, 1992.
- [36] Michael P. Fourman. "Compaction of symbolic layout using genetic algorithms." In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 141-153, 1985.

- [37] M. Farina and P. Amato. “On the optimal solution definition for many-criteria optimization problems.” In *Proceedings of the NAFIPS-FLINT International Conference 2002*, pages 233--238, New Orleans, LA, 2002.
- [38] Arturo H. Aguirre, Salvador B. Rionda, Carlos A. Coello Coello, Giovanni L. Lizárraga, and Efrén M. Montes. “Handling constraints using multiobjective optimization concepts.” *International Journal for Numerical Methods in Engineering*, 59(15), pages 1989-2017, 2004.
- [39] Oliver de Weck and Marshall B. Jones. “Isoperformance: Analysis and Design of Complex Systems with Known or Desired Outcomes.” In Proceedings of the 14th Annual International Council on Systems Engineering, Toulouse, France, June 2004.
- [40] Oliver de Weck, David W. Miller and Gary E. Moiser. “Multivariable Isoperformance Methodology for Precision Opto-Mechanical Systems.” 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Denver, CO, April 2002.
- [41] Il Yong Kim and Oliver de Weck. “Adaptive Weighted-Sum Method for Bi-Objective Optimization: Pareto Front Generation.” *Structural and Multidisciplinary Optimization*, 29 (2), pages 149-158, February 2005.
- [42] Tom Schouwenaars, Bart De Moor, Eric Feron and Jonathan How. “Mixed integer programming for multi-vehicle path planning.” In *Proceedings of the European Control Conference*, pages 2603-2608, 2001.

- [43] Arthur Richards and Jonathan How. “Performance evaluation of rendezvous using model predictive control.” In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Austin, Texas, 2003.
- [44] Ian Garcia and Jonathan P. How. “Trajectory optimization for satellite reconfiguration maneuvers with position and attitude constraints.” In *Proceedings of the IEEE American Control Conference*, pages 889-895, 2005.
- [45] A. E. Bryson Jr. and Y. C. Ho. *Applied Optimal Control*. Blaisdell, Waltham, MA, 1969.
- [46] Donald E. Kirk. *Optimal Control Theory: An Introduction*. Dover Publications, Inc., Mineola, New York, 2004 (reprint of 1970 edition by Prentice-Hall, Inc., Englewood Cliffs, NJ).
- [47] S. Roberts and J. Shipman. *Two-Point Boundary Value Problems: Shooting Methods*. Elsevier, New York, 1972.
- [48] Oscar von Stryk. “Numerical solution of optimal control problems by direct collocation.” In R. Bulirsch, A. Miele, J. Stoer, and K.-H. Well (editors), *Optimal Control – Calculus of Variations, Optimal Control Theory and Numerical Methods, number 111 in International Series of Numerical Mathematics*, Birkhauser, Basel, 1993.
- [49] Carl Glen Henshaw. “A unification of artificial potential function guidance and optimal trajectory planning.” In *Proceedings of the 28th AAS Annual Rocky Mountain Guidance and Control Conference*, pages 219-234, Breckenridge, Colorado, 2005.

- [50] Jacek Kierzenka and Lawrence Shampine. "A BVP solver based on residual control and the Matlab PSE." *ACM Transactions on Mathematical Software* 27(3), pages 299-316, 2001.
- [51] Carl Glen Henshaw. *A Variational Technique for Spacecraft Trajectory Planning*. PhD dissertation, University of Maryland Department of Aerospace Engineering, 2003.
- [52] John R. Anderson and Christian Libeire. *Atomic Components of Thought*. Lawrence Erlbaum Associates., Mahwah, New Jersey, 1998
- [53] Jamie Lennon and Ella Atkins. "Multi-objective spacecraft trajectory optimization with synthetic agent oversight." *Journal of Aerospace Computing, Information and Communication* 1(1): pages1-20, 2004.
- [54] Robert Wray and Ron Chong. "Comparing cognitive models and human behavior representations: Computational tools for expressing human behavior." *Proceedings of the Infotech@Aerospace 2005 Conference*, Arlington, VA. American Institute of Aeronautics and Astronautics, September 2005.
- [55] Panagiotis Tsiotras. "Stabilization and optimality results for the attitude control problem." *AIAA Journal of Guidance, Control, and Dynamics*, 19(4): pages 772-779, July-August 1996.