Experiments in Real Time Path Planning for a Small Unmanned Helicopter using Mixed Integer Linear Programming

by

Ioannis Martinos

BSME Tufts University 1999
MSME Tufts University 2001

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2003

Author_____
Department of Aeronautics and Astronautics
August 8, 2003

Certified by_____
Eric Feron
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by_____
Edward M. Greitzer
H.N. Slater Professor of Aeronautics and Astronautics Chair, Committee on Graduate Students

Experiments in Real Time Path Planning for a Small Unmanned Helicopter using Mixed Integer Linear Programming

by

Ioannis Martinos

Submitted to the Department of Aeronautics and Astronautics
on August 8, 2003, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

We use mathematical programming to perform simulated and actual flight experiments with the MIT autonomous helicopter platform. The experimental platform mechanical hardware, avionics and software architecture are described. Mixed Integer Linear Programming formulations for guidance experiments in obstacle avoidance and threat evasion are presented. Results from an experimental flight and flight simulations are presented and discussed. The performance of the GNU Linear Programming Kit (GLPK) software library, used in the experiments, is examined. The solvers internal branching heuristic, as it pertains to our formulations, is illustrated graphically. Finally, we discuss ways to improve the solvers performance by adding stronger objective bounds to the subproblems it creates while applying the branch and bound algorithm.

Thesis Supervisor: Eric Feron
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

I would like to thank Prof. E. Feron, the best advisor I could have asked for! I am thankful to Prof. J. How who gave me the opportunity to jump into the topic presented in this thesis.

Vlad Gavrilets has been a grate teacher to me. Tom Schouwenaars developed the bulk of the theory used in this thesis. Big thank you to Jan De Mot who helped me enormously and whom I admire like I do few people. It was grate pleasure working with Kara Sprague on the helicopter. Among other contributions, Rodin Lyasoff designed the cool hammerhead maneuver.

Ji Hyun Yang defined my MIT experience, I am grateful. Our lab is the coolest of its kind and I was glad to have been there with two highly respected members, David Dugail and Emilio Frazzoli.

Thank you Prof. D. Vogan.
Thank you Prof. J. N. Tsitsiklis.

Alex Lob and Raja Bortcosh have contributed substantially to the success of our projects.

Finally I wish to thank the patient Dinos, Lia, Ioanna and Nicholas Martinos.
Alex thanks!
And obviously ... my saper friends.

This work was funded by NASA Grant NAG 2-1552 Motion Planning For Agile Maneuvering Vehicles and Office of Naval Research (ONR) Grant N00014-03-0171.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Unmanned Aerial Vehicles (UAVs) are becoming increasingly popular for scientific, military and civilian applications. Projects like the Helios UAV, which is to fly for months at a time acting as a telecommunications relay, are continuously expanding the UAV application set. As UAVs are becoming more reliable and more capable their popularity is bound to increase. Many of the operational UAVs are used for surveillance, imaging and environmental monitoring. These applications typically require long range and endurance, characteristics which pertain mostly to fixed wing aircraft. In part due to these reasons, at the time of this writing most UAVs are based on fixed wing aircraft.

Helicopters are agile machines with vertical takeoff and landing capabilities. Additionally they can hover efficiently for long periods of time. This characteristics make them ideal for urban environments which are becoming increasingly important in military operations. Rotorcraft UAV could also be used by the entertainment industry for precision areal shots. Other applications include structural inspections, crop management and aerial photography.

Recent advances in automatic control of small unmanned helicopters [6] are making better use of helicopter agility. This results were demonstrated through experimental work, validating the control architecture. A helicopter that operates under these controllers can take high level velocity and heading rate commands which it can track at forward speeds up to $15m/s$. Furthermore preprogrammed maneuvers can be triggered under some initial condition restrictions.

A practical and general guidance framework that allows autonomous planning is particularly attractive. Such a framework can be used to delegate the planing details of a mission to the UAV. An example of this is when a UAV has to fly through an area with obstacles. The best way to avoid the obstacles is decided by the UAV as opposed to a human operator. In other cases a human operator in the loop may delay decisions significantly. For example when a UAV detects a threat

ahead of it, making a prompt decision is vital to evading the threat. UAVs should be able to handle realistic examples like these ones in order to further their use.

1.2 Work Description

The work presented in this thesis is part of a greater collection of experimental work done at MIT, surrounding small unmanned helicopters. In these experiments we are concerned with the practical aspects of implementing a MILP based guidance framework onboard a small unmanned helicopter.

This thesis presents the experimental platform, MILP formulations and software used to fly an obstacle avoidance mission and simulate a thread evasion scenario. A mathematical programming guidance framework is used to fly guidance missions with the MIT helicopter. Furthermore we look at the mathematical structure of the MILP formulations that are involved in our experiments. In particular we are interested in characteristics that they poses, which will allow us to solve them more efficiently.

1.3 Previous Work

Experiments like the ones presented in this thesis require the integration of numerous mechanical, electronic and software components. Furthermore extensive modeling and control system design is required. More details on these topics can be found in the following references; hardware and software architecture [9, 26], dynamic modeling [8, 6], automatic control [7, 6].

The guidance framework used in this thesis was developed mainly in [25, 24]. This mathematical programming framework is designed to compute optimal trajectories which make full use of vehicle dynamics. Since the MILP framework is computationally intensive, a receding horizon algorithm that ensures safety a priori is proposed in [23]. In this receding horizon approach a safe path to a predefined safe state is always maintained. Guidance experiments using multiple wheeled ground vehicles are presented in [21]. The experiments revolve around obstacle avoidance and vehicle low level control while performing a rendezvous.

Multi agent navigation problems, where the environment is only partially known, are also of interest. The MILP framework can be used as the lower level of a two level hierarchical guidance framework. In such an environment the higher level guidance can be based on dynamic programming, which is used to navigate efficiently a cluster of agents to a target. More information on multi-agent navigation and their spacial distribution can be found in [18, 17].

An alternative approach to maneuver automation, based on a hybrid automaton, was developed in [5]. In this framework computed paths consist of trim trajectories and maneuvers used to transition between trims. In this framework the vehicle decides on staying on the current trim trajectory or

performing a maneuver, base on a value function which was computed off-line. This approach is limited by the discretization of the vehicles dynamics into a discrete set of trajectories. The less trajectories the less precise the motion of the vehicle. On the other hand the finer the discretization the larger the size of the corresponding dynamic program.

Chapter 2

Experimental Platform and Theoretical Background

2.1 Helicopter Description

The experiments described in this document are based on the MIT instrumented helicopter platform that we developed specifically for controls and guidance research. The helicopter is based on an off the shelf airframe augmented with a custom avionics package housed in a vibration isolated box [26]. The box is mounted on the underbelly of the airframe. Figure 2-1 shows the complete assembly in flight.

2.1.1 Airframe

The helicopter airframe is a a Miniature Aircraft X-Cell .60 SE which has been outfitted with a 90 size engine in order to increase its payload capacity. The design makes extensive use of carbon fiber resulting in a stronger and lighter airframe. The main rotor blades are symmetrical and negative collective is available, allowing for inverted flight. The main hub has no flap hinge, another critical attribute for aerobatic flight. Without any instrumentation the airframe weighs 8 lbs. The helicopter can perform aerobatic maneuvers while hulling 7 lbs of payload.

Mounting the avionics payload results in little modification to the original airframe. The changes are limited to removing the canopy and extending the landing gear. The avionics is housed in a 12x7x4 inch aluminum box. The box is mounted to the airframe using a custom vibration isolation system designed to attenuate the main rotor generated vibrations. Mechanical isolation is key to extending electronics life by protecting solder joints from cracking. In addition, the isolation results in significant reduction of gyro drift.

Although the airframes weight almost doubles, the assembly is still capable of aerobatic flight. In

Figure 2-1: MIT Instrumented Helicopter (Photo by David Dugail)

particular we have performed barrel roll, split-s and hammerhead maneuvers. Other than modifying the maneuver profiles the avionics prohibits prolonged inverted flight. The neoprene isolators used in the isolation are not as effective when the helicopter is inverted.

2.1.2 Avionics

The avionics package is based mostly on off-the-shelf components. We use smart sensors, that is sensors which provide filtered data through a digital interface. In our case all sensors are interfaced to the flight computer via standard serial ports. This approach facilitates sensor integration and delegates the difficult task of sensor calibration and packaging to the sensors manufacturers. Figure 2-2 shows the avionics systems architecture.

The X-Cell's avionics software runs on a DSP Design TP400B PC-104 single board computer. The TP400B is based on the National Semiconductor Geode GX1 chip, which operates at up to 300MHz. The computer runs the real time operating system QNX 4.25. 128MB of RAM and 128MB of flash memory complete the x86 computer.

The inertial motion unit (IMU) is used to measure the three vehicle angular rates and translational accelerations with respect to its body axes. The IMU uses three gyroscopes for angular measurements. The gyroscopes have a 300 deg/sec range and their output biases drift at a rate of 0.02 deg/sec, after a 10 minute warm-up. The IMU uses three accelerometers to measure translational accelerations. The three accelerometers have a $\pm$10g range and their outputs drift in the order of 5mg during a 10 minute flight. The IMU is sampled at 100Hz through a 115200 bit/sec serial interface. The IMU weighs 250g and it costs $10,000.

A precision barometer, manufactured by Honeywell (HPB200A), provides relative altitude information. The sensor measures absolute pressure and has a 0-17 psi range. It can measure 1 mpsi pressure variations, which corresponds to approximately 2 feet altitude change near sea level. The

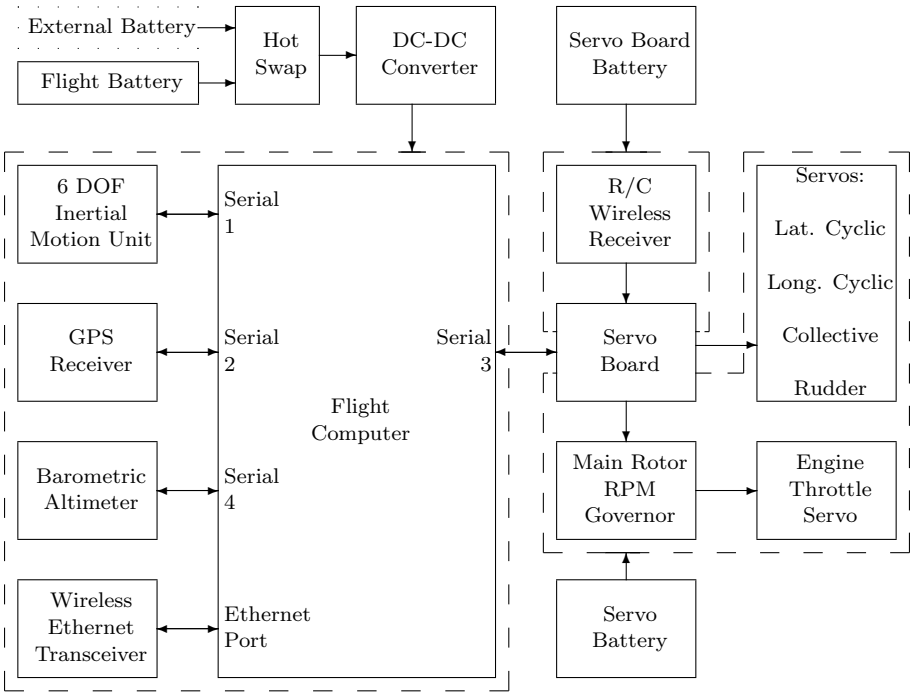

Figure 2-2: Avionics System Architecture.

barometric altimeter has a 2400 bit/sec TTL serial interface and is sampled at 5Hz. An important feature of this sensor is that it is insensitive to acceleration unlike the majority of pressure sensors on the market. The HPB200A costs $700.

The industry standard G12 GPS receiver from Ashtech provides position and velocity data at 10 Hz. Although the data is available for all three inertial axis only North and East axis data is used. Down axis data is less accurate and is discarded. The receivers 10Hz rate is instrumental for the implementation of the extended Kalman filter running onboard the helicopter. More information on the filter can be found at [6].

For low frequency heading component a magnetic compass was flown onboard the helicopter. Due to several sources of noise, including the spinning carbon fiber blades, the accuracy of the compass was inadequate for out purposes. So, instead the EKF estimates heading by blending the remaining sensor information. For better EKF heading estimates, prolonged hovering is avoided as heading becomes unobservable. The compass is no longer flown onboard the system. Details on the compass design can be found in appendix C.

The tree way interface between the pilot the flight computer and the actuators is provided by a custom circuit board. The servo board, as we call it, is based on a PIC 16F877 and 2 SX28 chips. It allows switching between computer and pilot control via the pilots radio. It also measures pilot commands and reports them to the onboard computer for processing.

Telemetry data is transmitted from the helicopter to a laptop computer running QNX. The

data is transmitted via a Proxim wireless ethernet adapter. This link has proven to be rather unreliable. At the time of this writing we are near completion of integrating a new set of wireless serial transceivers. When integration is completed these transceivers will permanently replace the Proxim. The new transceivers are manufactured by Maxstream, in addition to being more reliable they are lighter than the Proxims. The freed payload weight will be used to augment a camera to the system.

Minimizing actuator lag is of paramount importance to the control system. For this reason we use fast Futaba S9402 servos for cyclic and collective controls. For the tail rotor we use the even faster Futaba 9450. The engine throttle servo does not have to be as fast since the engine response is slower than most servos.

To power the avionics we use 2 9.6V 1400 mAh NiMh batteries which can go for 90 minutes on a single charge. Since we only fly for less than 30 minutes on each flight operation we have decided to remove one of them. As with the wireless link this is still work in progress. In addition we carry two six-cell NiCad batteries to power the the servos and the logic side of the servo board. The devices powered by each battery are grouped using a dashed line in figure 2-2.

2.1.3 Hardware in the Loop Simulator

Essential to the experimental platforms success is the Hardware In the Loop Simulator (HILSim). The HILSim allows us to test on the ground the exact flight software as it runs onboard the helicopter. The simulator consists of the same hardware that is actually flown, other than the sensors and the airframe. The airframe dynamics are replaced by simulation software running on a 700MHz PIII computer running QNX. The nonlinear dynamic model can be found in [6]. Dynamics are propagated in software and sensor measurements are emulated. The sensor data streams are passed into the clone flight computer using the same protocols the actual sensors used.

Input to the simulator is provided through the same radio we use in-flight. The servoboard reads the receiver signals and the computer commands and drives a set of servos like the ones found on the helicopter. Using push rods the servos turn potentiometers whose position is read through an analog to digital card in the simulator computer. Since the closed loop response of servos varies little when they are moderately loaded, this input method provides good simulation of the control input lag to the system. For example when we used an S9402 servo for the tail rotor in the simulator we got a lightly damped mode on the yaw axis due to lag.

2.1.4 Platform Capabilities

The MIT X-Cell helicopter is the first helicopter to demonstrate aggressive maneuvers under computer control. Information on its control system design, the linearized models used for control system design and the non-linear model used for simulation can be found in [6, 7, 8].

At the time of this writing the helicopter has performed the following maneuvers under computer control: barrel roll, split-s and hammerhead. In velocity/heading rate mode the control system accepts high level forward velocity and heading rate commands. Depending on forward speed the heading rate command can be interpreted as a tuning rate command. Furthermore basic waypoint navigation logic has been implemented. These characteristics of the platform make it ideal for guidance research.

Simplified Guidance Model

The helicopter closed loop response to forward velocity, clime rate and turn rate commands, can be approximated by decoupled first order equations [6]. Since all planning in this document is restricted to North-East plane, we are only concerned with forward velocity and turn rate commands.

The following simple model can be used for two dimension planning:

$$\dot{u} = -\frac{1}{\tau_u}u + \frac{1}{\tau_u}u_c \tag{2.1}$$

$$\dot{r} = -\frac{1}{\tau_r}r + \frac{1}{\tau_r}r_c \tag{2.2}$$

$$\dot{\psi} = r \tag{2.3}$$

$$\dot{p_N} = u\cos\psi \tag{2.4}$$

$$\dot{p_E} = u\sin\psi, \tag{2.5}$$

where $\tau_u = 1.3$ and $\tau_r = 0.7$.

Although simple, this model can not be directly used with our MILP framework. It can only apply directly if we assume that the heading is fixed. If that is the case the model is linear and can be used in the MILP formulation of a guidance problem. Dealing with non-linearities resulting from coordinate transformations is still work in progress.

2.2 MILP Framework

The ideas behind the MILP framework discussed here were developed in [25, 24]

We discuss here the mathematical programming framework behind the guidance experiments in the next chapter. The framework applies to linear dynamic systems moving through a known environment. Although all constraints used are linear, or can be expressed using linear approximations, the feasible solution set is not always convex. The latter introduces binary variables to the formulation, which makes these problems fall in the Mixed Integer Linear Programming (MILP) category.

In the following sections we discuss how different equation blocks can be used to model different scenarios. For example we examine how dynamical constraints can be imposed while obstacle regions can be excluded from the solution space. Although this MILP framework is very powerful it is also computationally intensive. This becomes less of a problem as modern computers become faster and MILP solvers become more efficient.

Independently of which constraints are imposed in the different formulations, all of them have some common characteristics. Each formulation makes use of a reduced state vector. For example a vector that consists of the position and velocity of the vehicle. Constrains are associated with the initial and final conditions of the vehicle. For example they specify that the vehicle starts at a certain point in space with a certain speed. Then the vehicle has to reach a certain state condition at the end of the planned trajectory.

2.2.1 Linear Dynamics

Since our purpose is to use MILP to guide a vehicle, we need to incorporate its dynamics into the framework. Obviously the dynamics must be expressed in linear form. On the other hand dynamics do not necessarily need to be time invariant. We show here how to model a velocity dependent dynamic model of the vehicle. Of course several variations of the following scheme can be devised for different vehicles. Since in our case we use the MIT X-Cell helicopter for our experiments we schedule the modes with respect to speed. We use a double integrator model here with acceleration commands as inputs.

For modeling several LTI modes are used, at any given time one of them determines the vehicles dynamics. A dynamic matrix $\mathbf{A}$, and an input matrix $\mathbf{B}$ are associated with each LTI mode. Which LTI mode is in effect at a given time step is determined by the vehicles forward velocity. The LTI models are discretized using a time step size specific to each formulation. The more time steps there are the more accurate the planned trajectory but the more time it takes to compute the optimal trajectory. Furthermore when maneuvers are also included in the formulation, instead of time steps we use decision steps. At each decision step the vehicle can be either executing a maneuver or it can be in an LTI mode. Equation 2.6

$$\begin{aligned} \forall l \in [1 \dots L]: \quad \mathbf{x}_{i+1} - \mathbf{A}_l \mathbf{x}_i - \mathbf{B}_l \mathbf{u}_i &\leq M(1 - b_{il}) \\ -\mathbf{x}_{i+1} + \mathbf{A}_l \mathbf{x}_i + \mathbf{B}_l \mathbf{u}_i &\leq M(1 - b_{il}) \\ \textstyle\sum_{i=1}^{L} b_{il} &= 1 \,, \end{aligned} \tag{2.6}$$

specifies which of the LTI modes is active. This is determined by the binary variables b_{il}. The same binary variables are used to control the velocity bounds for each LTI mode. The first LTI mode, e.g. the hovering dynamics, do not need velocity lower bound but impose an acceleration command

bound. For the remaining modes we we bound velocity from both sides. These bounds are are expressed as follows:

$$
\begin{aligned}
l = 1: \quad & \forall k \in [1 \dots K]: \\
& v_{xi} \sin\left(\tfrac{2\pi k}{K}\right) + v_{yi} \cos\left(\tfrac{2\pi k}{K}\right) \leq v_{max,h} + M(1 - b_{i1}) \\
& a_{xi} \sin\left(\tfrac{2\pi k}{K}\right) + a_{yi} \cos\left(\tfrac{2\pi k}{K}\right) \leq a_{max} + M(1 - b_{i1}) \\
\forall l \in [2 \dots L]: \quad & \forall k \in [1 \dots K]: \\
& v_{xi} \sin\left(\tfrac{2\pi k}{K}\right) + v_{yi} \cos\left(\tfrac{2\pi k}{K}\right) \leq v_{max,l} + M(1 - b_{il}) \\
& -v_{xi} \sin\left(\tfrac{2\pi k}{K}\right) - v_{yi} \cos\left(\tfrac{2\pi k}{K}\right) \leq -v_{min,l} + M(1 - b_{il}) + M c_{ilk} \\
& \textstyle\sum_{k=1}^{K} c_{ilk} \leq K - 1 \,,
\end{aligned}
\tag{2.7}
$$

where K is the number of segments in the circular bound discretization. The binary variables c_{ilk} are used to accommodate the non-convex velocity constraints.

When the helicopter is flying fast, lateral commands have to be kept small. Equation 2.8:

$$
\begin{aligned}
(a_x + \alpha_l v_y) \sin\left(\tfrac{2\pi k}{K}\right) + (a_y - \alpha_l v_x) \cos\left(\tfrac{2\pi k}{K}\right) &\leq \beta_l a_{max} + M(1 - b_{il}) \\
(a_x - \alpha_l v_y) \sin\left(\tfrac{2\pi k}{K}\right) + (a_y + \alpha_l v_x) \cos\left(\tfrac{2\pi k}{K}\right) &\leq \beta_l a_{max} + M(1 - b_{il}) \,,
\end{aligned}
\tag{2.8}
$$

is used to achieve this.

2.2.2 Obstacles

The LTI modes determine how the vehicles state can move about the state space. Obstacles can be modeled by removing regions of the feasible set. By doing this we guarantee that any feasible trajectory is a collision free trajectory. Unfortunately this makes the solution space non-convex and we have to introduce binary variables to handle this issue. Here we are modeling the obstacles using line approximations. A collision can be avoided by requiring that the vehicle does not violate, at any given time, at least one of the constraints, corresponding to an obstacle side. This can be achieved be relaxing a maximum of three constraints when the obstacle has four sides. The above statement is represented by the following constraints.

$$
\begin{aligned}
x_i &\leq x_{c,min} + M t_{c,i,1} \\
-x_i &\leq -x_{c,max} + M t_{c,i,2} \\
y_i &\leq x_{c,min} + M t_{c,i,3} \\
-y_i &\leq -y_{c,max} + M t_{c,i,4} \\
\textstyle\sum_{k=1}^{4} t_{c,i,k} &\leq 3
\end{aligned}
\tag{2.9}
$$

where the binary variables $t_{c,i,k}$ are used to relax the constraint corresponding to side k.

2.2.3 Using Aggressive Maneuvers

Maneuvers are represented as a jump of the vehicle state. Each maneuver is preprogrammed into the control system and can be characterized by the time it takes to execute, the exit speed and the body axis displacement. Using this information a linear transformation can be calculated which maps the initial state to the vehicle state after the maneuver in the inertial reference frame. This transformation matrix is denoted by $\mathbf{C}_m$ where m is the maneuver index. See [24] for a full description of $\mathbf{C}_m$. The dynamic constrains of a maneuver are imposed to the vehicle using the same binary variable mechanism as the one used for LTI modes. Since the vehicle has to be within a certain velocity range, additional velocity constraints are imposed. The equations follow:

$$\begin{aligned}
\forall m \in [1 \dots P]: \quad \mathbf{x}_{i+1} - \mathbf{C}_m \mathbf{x}_i &\leq M(1 - d_{im}) \\
-\mathbf{x}_{i+1} + \mathbf{C}_m \mathbf{x}_i &\leq M(1 - d_{im}) \\
\textstyle\sum_{m=1}^{P} d_{im} &\leq 1 \\
\\
\forall k \in [1 \dots K]: & \\
v_{xi} \sin\left(\tfrac{2\pi k}{K}\right) + v_{yi} \cos\left(\tfrac{2\pi k}{K}\right) &\leq v_{init,m} + M(1 - d_{im}) \\
-v_{xi} \sin\left(\tfrac{2\pi k}{K}\right) - v_{yi} \cos\left(\tfrac{2\pi k}{K}\right) &\leq -v_{init,m} + M(1 - d_{im}) + M e_{imk} \\
\textstyle\sum_{k=1}^{K} e_{imk} &\leq K - 1 \ .
\end{aligned} \tag{2.10}$$

2.2.4 Minimizing Energy

Depending on the vehicle model used and the application, minimizing energy can be of interest. A good example of this is in space satellite applications where the actuators are thrusters. In these cases one can use a double integrator model with acceleration inputs. Then minimizing the inputs is equivalent to computing an energy optimal trajectory. Equation 2.11 depicts an objective function that will minimize the system input,

$$J_N = \sum_{i=0}^{N-1} \sum_{j=1}^{n_u} |u_{ij}|. \tag{2.11}$$

For more information on MILP guidance for space applications see [22].

2.2.5 Minimizing Time

To minimize time we introduce the binary variable t_i which is equal to 1 at the arrival decision step. That is, when $t_i = 1$ the final state constraints are enforced. The objective function in this case consists of the amount of time we spend in LTI modes plus the amount of time we spend executing maneuvers. There is one more component minimizing the input. This component is scaled such that

it does not influence the arrival time. Minimizing the input, in addition to time, results in smoother trajectories. Equation 2.12:

$$J_N = \sum_{i=0}^{N} t_i i\,\Delta t + \sum_{i=0}^{N-1}\sum_{m=1}^{P} d_{im}(\Delta T_m - \Delta t) + \epsilon \sum_{i=0}^{N-1}\sum_{n=1}^{n_u} |u_{in}| \ , \tag{2.12}$$

is used for time minimization problems that include maneuvers. Since time minimization is our primary concern in this case, the input minimization part of the objective function is scale by a small number ϵ.

2.2.6 Minimizing Distance

To minimize distance we minimize the sum of the euclidean norm of velocities at each time step. This results in the shortest dynamically feasible path for the given amount of time. We use the following expression for the cost function J_N:

$$J_N = \sum_{i=0}^{N-1} \sqrt{u_i^2 + v_i^2}. \tag{2.13}$$

J_N is a convex nonlinear function, so a piecewise linear convex function is used to approximate it. Consequently linear programming can still be used to solve this problem. The piecewise linear function used is of the form:

$$\sum_{i=0}^{N-1} \max_{j=1,\ldots,m} (a_j u_i + b_j v_i + c_j), \tag{2.14}$$

where m is the number of linear pieces used, $a_j = \frac{\partial\sqrt{u_i^2+v_i^2}}{\partial u_i}$, $b_j = \frac{\partial\sqrt{u_i^2+v_i^2}}{\partial v_i}$ and $c_j = \sqrt{u_i^2 + v_i^2} - (a_j u_i + b_j v_i)$. Actually in the particular case $c_j = 0$ for all j.

The approximation comes at the expense of additional constraints which we have to add to the MILP formulation. There are an additional m constraints for each time step. The constraints have the following form:

$$z_i \geq a_j u_i + b_j v_i, \tag{2.15}$$

and the objective function becomes:

$$\sum_{i=0}^{N-1} z_i. \tag{2.16}$$

It should be noted that the objective function does not explicitly minimize time. Since we are minimizing distance and the MILP solution is passed to the waypoint navigation logic, the arrival time is not as important since it is mostly determined by the control system. If need be, we could

minimize time and minimize distance for the given arrival time by scaling the above objective function by an ϵ such that the time minimization binaries would dominate the objective function.

Chapter 3

Flight Experiments

In this chapter we present three MILP missions specific to the MIT X-Cell helicopter [26, 6] . During these missions MILP problems are solved online and their output is used to guide the helicopter. The MILP problems solved are small instances of the formulations in the previous chapter. In particular we discussed three MILP missions. The first mission demonstrates obstacle avoidance using the MILP framework. The second and third missions, make use of the agile maneuvering formulation to handle to the same scenario with parametric differences.

3.1 Software Architecture

The X-Cell's guidance and control software runs on a DSP Design TP400B single board computer. The TP400B is based on the National Semiconductor Geode GX1 chip, which operates at up to 300MHz. The computer runs the real time operating system QNX 4.25.

Flight software is implemented in a modular process by assigning independent processes to the many required tasks. For example the sampling of each sensor on the helicopter is performed by a separate process. The main process, logflight, orchestrates all these processes and runs the control logic which eventually produces the flight surface angles after each control loop. The control software has several modes of operations. These modes are manual, velocity/heading rate and mission. In manual mode logflight just provides yaw rate augmentation. Velocity/heading rate mode interprets the pilot stick positions as high level velocity and heading rate commands which the control system tracks. Additionally in this mode, given proper initial conditions several aggressive maneuvers can be triggered. Building upon the velocity control mode the mission mode provides mission specific high level commands removing the human pilot completely out of the loop. Adding MILP path planning capabilities requires a MILP solver to run onboard the helicopter and to produce on the fly the high level commands used in the mission mode.

Several MILP solvers are available. Based on performance and portability the open source linear

programming library GLPK [1] is used in our online experiments. Additional details about the solver and the selection process can be found in Appendix A. Based on the GLPK library a solver has been integrated into our flight code. The solver is implemented as an independent low priority process which interacts with the remaining flight code through shared memory. This architectures assures that the primary flight control system is unaffected by the computationally demanding solver.

In theory the MILP formulations could be generated inflight as obstacles or other scenarios appear. There could be a lookup table with the appropriate constraints to be augmented to the problem for a range of sensed conditions. Once all the sensor data is processed the MILP is augmented with the initial conditions and solved to optimality. Ideally the computation should take no time. Instead we consider virtual scenarios whose formulations are predetermined and are sized so that the problems can be solved online with the given hardware and software.

The formulations for each considered mission are encoded in the standard Mathematical Programming System (MPS) file format. During a mission its corresponding MPS file is loaded. Initial conditions or other parameters can be modified when the MILP has been loaded into memory. The scenario conditions are virtually sensed and the solver starts solving the guidance problem. When the problem is solved to optimality the solution is processed appropriately.

The solver is controlled and communicates with logflight through a structure in share memory. All control flags are operated upon with atomic operations preventing any racing conditions from happening. The solver is triggered by logflight using a proxy. The flight code then poles the solver for the solution status. Once the problem is solved to optimality the solver updates the shared memory location with the computed decision variables and sets a flag indicating that a valid solution is available. The MILP solution is then processed by the control software.

The MILP problems described here are sufficiently small to be solved onboard the helicopter.

3.2 Obstacle Avoidance FLight Experiment

3.2.1 Flight Description

This experiment demonstrates a simple obstacle avoidance scenario. For this experiment the pilot takes of in manual mode, the pilot then switches to automatic control at a safe altitude and in moderate forward flight. While in velocity/heading rate mode the helicopter is brought close to 40 meters at which point its speed command is reduced to zero. The fully autonomous sequence is engaged at this point. The helicopter flies at the preprogrammed altitude of 50 meters. Next the sequence consists of three legs. Legs one and three consist of waypoint navigation at 5 m/s through a series of predetermined waypoints. At the end of leg one the helicopter encounters a virtual obstacle. The helicopter hovers for a few seconds and computes online the shortest dynamically feasable path around the obstacle. Once it has found a solution it generates a sequence of waypoints which lead

to the beginning of the third leg. These waypoints are then tracked with a 3 m/s forward speed. The helicopter then goes on to the third leg after which the sequence is completed. The helicopter then returns to the location where the sequence was engaged and maintains that position. The pilot takes over from there and brings in the helicopter for landing.

3.2.2 Formulation

Dynamic Model

The leg of the mission generated by solving a MILP is flown at 3 m/s, which is well under the 12m/s top speed of the velocity/heading rate controllers. The low speed makes the flight safer from an operational point of view, it also make the formulation lighter from a computational point of view. Since the flight is limited to 3 m/s we can use a symmetrical dynamic model describing the longitudinal and lateral closed loop response of the helicopter at low speeds. A symmetrical model can be used because at such low speed the aerodynamic forces are less dependent on flight direction and are masked well by the control logic.

In [25] a double integrator model is used for the dynamic constraints. In this case the inputs to the system are accelerations, the sum of which is minimized by the MILP. In the case of the helicopter the underlying control system accepts velocity commands with respect to the vehicles body axis. The MILP framework assumes an inertial reference frame. Unfortunately transformations from inertial to body coordinates are nonlinear. To overcome this problem while modeling we assume that the helicopters heading is fixed and aligned to the north. This way inertial velocity commands can be directly translated to inputs for the controllers.

When flying at a lower speed than $3m/s$, longitudinal and lateral dynamics can be consider to be decoupled. This results in a simple model that adequately describes the closed loop dynamics, of the low speed velocity controllers, for path planning purposes. The model consist of two first order systems, one for longitudinal and one for lateral dynamics. The time constant τ is equal to 2 seconds for both the longitudinal and lateral axis. The state space representation of the model is:

$$\begin{bmatrix} \dot{x} \\ \dot{u} \\ \dot{y} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{1}{\tau} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\frac{1}{\tau} \end{bmatrix} \begin{bmatrix} x \\ u \\ y \\ v \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{1}{\tau} & 0 \\ 0 & 0 \\ 0 & \frac{1}{\tau} \end{bmatrix} \begin{bmatrix} u_c \\ v_c \end{bmatrix}, \tag{3.1}$$

where, x, y and u, v are the positions and velocities along the x and y axis respectivly. Thew velocity commands, u_c and v_c correspond to the x and y axis respectivly.

3.3 Objective Function

To minimize distance we minimize the sum of the euclidean norm of velocities at each time step. This results in the shortest dynamically feasible path for the given amount of time. We use the following expression for the cost function J_N :

$$J_N = \sum_{i=0}^{N-1} \sqrt{u_i^2 + v_i^2} \tag{3.2}$$

J_N is a convex nonlinear function. A piecewise linear convex function is used to approximate J_N, so that it fits as a cost function in the MILP framework. See section 2.2.6 for additional information on how this is done.

MILP Formulation

The MILP formulation corresponding to the experiment follows.

$$\begin{array}{rrrrrrr}
\text{minimize} & \sum_{i=0}^{N-1} z_i & & & & & \\
\text{subject to} & z_i & \geq & a_j u_i & + & b_j v_i \\
& \mathbf{s}_{i+1} & = & \mathbf{A}\mathbf{s}_i & + & \mathbf{B}\mathbf{u}_i \\
& \mathbf{s}_0 & = & \mathbf{s}_{initial} & & \\
& \mathbf{s}_{N-1} & = & \mathbf{s}_{final} & & \\
& x_i & \leq & x_{c,min} & + & M t_{c,i,1} \\
& -x_i & \leq & -x_{c,max} & + & M t_{c,i,2} \\
& y_i & \leq & x_{c,min} & + & M t_{c,i,3} \\
& -y_i & \leq & -y_{c,max} & + & M t_{c,i,4} \\
& \sum_{k=1}^{4} t_{c,i,k} & \leq & 3 & & \\
& & & & & \\
& t_{c,i,k} & \in & \{0,1\} & & \\
& i & \in & \{0 \ldots N-1\} & & \\
& j & \in & \{0 \ldots m\} & & \\
& k & \in & \{1 \ldots 4\} & & \\
& c & \in & \{1 \ldots L\} & &
\end{array}$$

The problem that was actually solved online has twelve time steps ($N = 12$), one obstacle ($L = 1$) and the euclidean approximation function consists of 16 pieces ($m = 16$). An AMPL [4] model is used to translate this problem into an MPS file that can be read by the helicopters MILP solver. This instance take about eight seconds to solve onboard the helicopter.

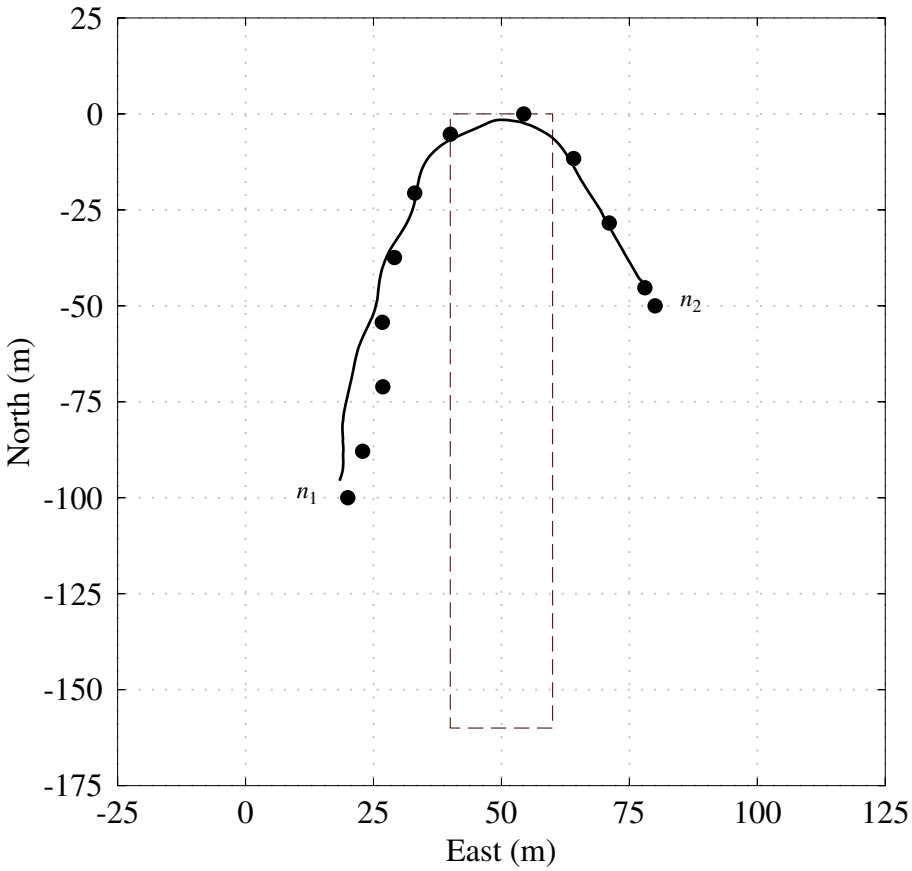

Figure 3-1: Obstacle avoidance flight experiment. The helicopter starts from n_1, computes the shortest path to n_2 and tracks generated waypoints. The dashed line represents the obstacle.

3.4 Agile Maneuvering Experiments

3.4.1 Description

These experiments where designed to show how a MILP can be used to make a path planning choice which would make use of set of available maneuvers. Suppose that the helicopter is flying in an urban environment and that after it makes a turn onto a street it senses a threat ahead of it. In this scenario the only way to avoid the threat is to turn around and head in the opposite direction as fast as possible. Figure 3-2 depicts this scenario. The problem is formulated as a MILP and two choices are available to turn around. Either the helicopter slows down turns around and accelerates in the opposite direction, or the helicopter performs an aggressive maneuver which reverses its heading and carries on in the opposite direction.

The best decision will depend on the parameters of the problem. In this case, the distance between the buildings is the determining factor. That is, if there is enough space an aggressive maneuver can be executed on the premise that it reverses the helicopters direction faster than the slowing down, pivoting and accelerating again. In some cases even if there is space for the maneuver

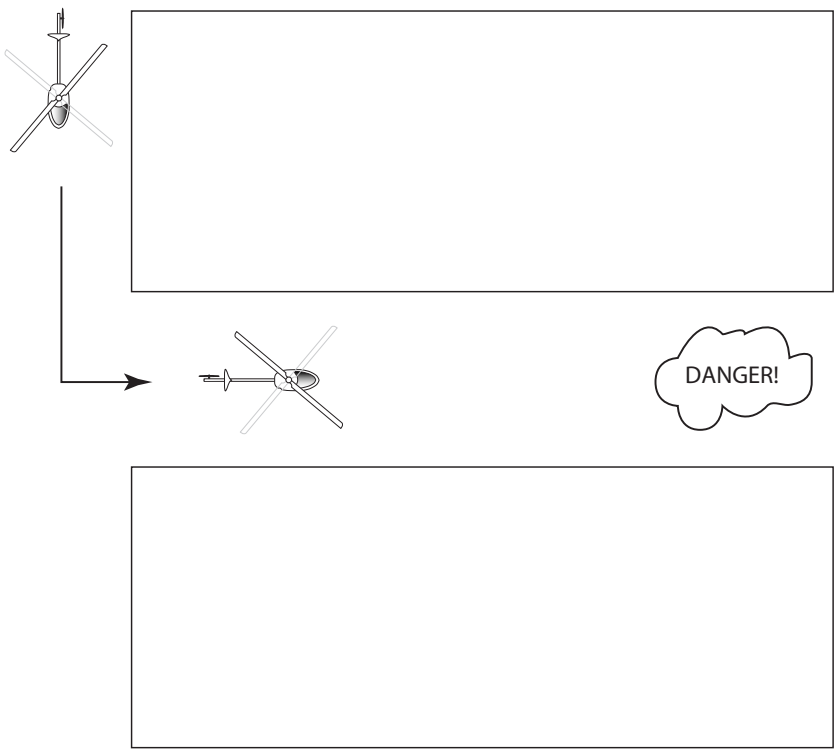

Figure 3-2: Considered scenario

it might not be the best choice. For example if the helicopter is flying at 5 m/sec it will takes too long to speed up to the 14 m/s speed required to initiate the maneuver. The guidance routines in both experiments are identical in every sense except that the MILP representations of the scenario differ. The difference being that 30 meters of later displacement are allowed in the first experoment whereas only 10 meters are allowed in the second experiment. With the given initial conditions and these constraints after solving the corresponding MILP problems online the guidance function take two different approaches to avoiding the approaching danger. In the the first experiment the helicopter speeds up to 14 m/sec and performs a hammerhead maneuver. In the second experiment the helicopter chooses to slow down, perform a zero velocity pivot and then accelerate in the opposite direction.

The two missions were simulated using our hardware in the loop simulator [ref]. The resulting trajectories for the two missions, from the time the threat was detected to the time the helicopter crosses the same point as it flies in the opposite direction, are shown in figure 3-3.

Although the underlying control system is capable of performing several aggressive maneuvers [6] only one was considered in the formulations in these experiments. This way the number of binary variables are dramatically reduced making the problem tractable for the onboard computer. The

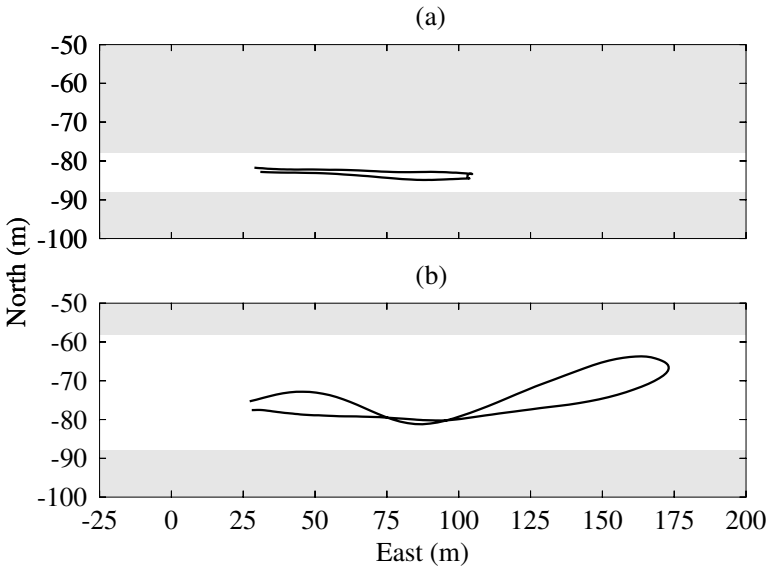

Figure 3-3: Agile maneuvering experiments. The top figure (a) shows helicopter reverse direction by slowing down and pivoting. In bottom figure (b) the helicopter performs a hammerhead instead. Gray arrears represent buildings.

problems solved here, which have one LTI mode, one maneuver and nine time steps; take about 3 three seconds to solve onboard the helicopter.

3.4.2 MILP Formulation

The MILP formulation corresponding to the experiments.

$$\tilde{J} = \sum_{i=0}^{T} t_i i\,\Delta t + \sum_{i=0}^{T-1}\sum_{m=1}^{P} d_{im}(\Delta T_m - \Delta t) + \epsilon \sum_{i=0}^{T-1}\sum_{n=1}^{n_u} |u_{in}| \tag{3.3}$$

$$\mathbf{s}_0 = \mathbf{s}_{initial} \tag{3.4}$$

$$u \leq u_{final} + K(1 - t_i) \tag{3.5}$$

$$\sum_{i=0}^{T-1} t_i = 1 \tag{3.6}$$

$$y \leq y_{max} \tag{3.7}$$

$$y \geq y_{min} \tag{3.8}$$

$$\begin{aligned} \mathbf{x}_{i+1} - \mathbf{A}\mathbf{x}_i - \mathbf{B}\mathbf{u}_i &\leq b_i\mathbf{M} \\ -\mathbf{x}_{i+1} + \mathbf{A}\mathbf{x}_i + \mathbf{B}\mathbf{u}_i &\leq b_i\mathbf{M} \end{aligned} \tag{3.9}$$

$$\begin{aligned} u_i &\leq u_{i-1} + u_c^{max}T_s + Kb_{i-1} \\ v_i &\leq v_{i-1} + v_c^{max}T_s + Kb_{i-1} \\ u_i &\geq u_{i-1} - u_c^{max}T_s - Kb_{i-1} \\ v_i &\geq v_{i-1} - v_c^{max}T_s - Kb_{i-1} \end{aligned} \tag{3.10}$$

$$\begin{aligned} u_i \sin\left(\tfrac{2\pi k}{K}\right) + v_i \cos\left(\tfrac{2\pi k}{K}\right) &\leq v_{init}^{max} + M(1-b_i) \\ -u_i \sin\left(\tfrac{2\pi k}{K}\right) - v_i \cos\left(\tfrac{2\pi k}{K}\right) &\leq -v_{init}^{min} + M(1-b_i) + Me_{ik} \\ \textstyle\sum_{k=1}^{K} e_{ik} &\leq K-1 \end{aligned} \tag{3.11}$$

$$\begin{aligned} \mathbf{x}_{i+1} - \mathbf{C}\mathbf{x}_i &\leq M(1-b_i) \\ -\mathbf{x}_{i+1} + \mathbf{C}\mathbf{x}_i &\leq M(1-b_i) \end{aligned} \tag{3.12}$$

3.4.3 Processing the MILP Output

Since the MILP input vector refers to the inertial reference frame we need to post process the input sequence to be used as input to the velocity controllers. A simple algorithm to do this is discussed here. The algorithm uses the fact that the helicopter flies along the line spanning from east to west. This means that the heading can either be -90 or 90 degrees. Initially the heading command is known to be 90 degrees.

Heading changes occur when a velocity sign change occurs or when a maneuver that reverses the helicopters direction is performed. The east axis velocity commands generated by the MILP are examined for sign alternations. If an alternation without a maneuver has occurred, a 180 degree turn is inserted in the helicopters command sequence path. If a maneuver, cased the velocity reversal no action is taken. In both cases the absolute value of the east axis velocity command is interpreted as a body axis forward velocity command.

Chapter 4

Solver Speed

In this chapter we discus solver specific attributes as they apply to our fomulations. In particular we examine closly the branching heuristic used by the GLPK library. The heuristic is based on assigning penalties to each binary variable, the penalties calculated as in [27]. In this chapter we examine how well these work with our formulations. Furthermore we make sugestions on how GLPK can be improved, by making better use of these penalties. Finally we describe a methode, base on visibility graphs, that can be used to drastically reduce the number of binary variables.

4.1 Tomlin Branching Heuristic

GLPK like many mixed integer solvers is based on the branch and bound algorithm. That is given a MILP its LP relaxation is solved to optimality, progressively stronger bounds are then applied to any integer variables which do not satisfy integrality. The LP relaxation of any subroblem created, is used to set an objective function bound to the best integer solution that could be obtained, by fully solving the particular problem. If the LP relaxation is not better than the current best known integer solution the particular problem is deleted. This methology results in an implicit enumeration tree which dramaticaly reduces the nuber of broblems that would have to be solved if we where to examine all possible integer variable assignments. The defigning characteristic of any such procedure is the heuristics used for branching and backtracking.

The branching heuristic used by GLPK is a partial implementation of the Tomlin heuristic as described in [27]. Given any unsatisfied integer variable $\bar{a}_{p0}$ we can write $\bar{a}_{p0} = n_{p0} + f_{p0}$ where n_{p0} is integer and $0 < f_{p0} < 1$. If the particular variable is chosen to branch on, GLPK crates two subproblems. Each subroblem is augmented with an additional constraint. The up and down branch are augmented with the constraints $X_p \geq n_{p0} + 1$ and $X_p \leq n_{p0}$. GLPK creates two child nodes on the implicit enumeration tree, each node corresponding to one of the two subporblems. Since constraints are added to the subproblems, their optimal solution can only be degraded with

respect to the parent problem.

Although the additional constraints make the the subproblems primal infeasable, they maintain dual feasability. This is due to the fact that the reduced cost of the slack variable, corresponding to the new constraint, is zero [3]. We can thus restore primal feasability by applying the dual simplex methode. In the process of doing so the objective functions will be degrated by at least P_U and P_D, which correspond to the up and down branches respectively. The calculation of P_U and P_D is discussed later. GLPK calculates P_U and P_D for all unsatisfied integer variables, it then choose to branch on the variable X_p having the gratest penalty, P_U or P_D, associated with it. Having chosen the branching variable GLPK choses to solve the brach with the smallest of the two penalties. By doing so, the hope is that the expensive branch will be pruned before we get to solving it.

For each subroblem primal feasability is restored by removing the basic column corresponding to the non-integer variable. At the same time a non-basic variable will enter the basis. Since this one will have a positive reduced cost, it will increase the objective function. The variable chosen to enter the basis is determined using the dual ratio test, which determine to what value the variable that enters the basis should be set, in order to move to the next basic dual feasable solution. Thus this change of basis has an associated cost with it that can be calculated. Additionally if the the variable entering the basis is marked as an integer variable, we can assume that it will have to be changed by at least one. P_U and P_D are defined to be the costs associated with the change of basis. Their precice definition follows:

$$
\begin{aligned}
P_U &= \min_{j,\bar{a}_{pj}<0} \begin{cases} \bar{a}_{0j}(1-f_{p0})/(-\bar{a}_{pj}) & j \notin J \\ \max\{\bar{a}_{0j}, \bar{a}_{0j}(1-f_{p0})/(-\bar{a}_{pj})\} & j \in J \end{cases} \\
P_D &= \min_{j,\bar{a}_{pj}>0} \begin{cases} \bar{a}_{0j} f_{p0}/(\bar{a}_{pj}) & j \notin J \\ \max\{\bar{a}_{0j}, \bar{a}_{0j} f_{p0}/(\bar{a}_{pj})\} & j \in J \end{cases}
\end{aligned}
\tag{4.1}
$$

Where J is the set of all non-basic integer variables and p is the index of the unsatisfied integer basic variable for which we are calculating the penalties. $\bar{a}_{pj}$ is the simplex tableu coefficient of the non-basic variable j and $\bar{a}_{0j}$ is its reduced cost.

GLPK implents the Tomlin heuristic as described above. For more an example on how the heuristic applies to an obastcle avoidnace problem see B Next we discuss some improvements that can be made to this implementation.

4.2 Improving GLPK

4.2.1 Child Node Bounds

Although GLPK uses the penalties P_U and P_D to chose the branching integer variable, it does not make use of the costs to tighten the LP bound of the created subproblems. That is, in a minimization problem, the up node optimal solution is at best $\bar{a}_{00} + P_U$, where $\bar{a}_{00}$ is the parent nodes optimal solution. This means that up node can be discarded if $P_U \geq x_{0c} - \bar{a}_{00}$, where x_{0c} is the best known integer solution so far. Further more P_U and P_D are not the strongest bounds available to apply to the child node objective functions. We can also calculate P_G, the penalty associated with satisfying the Gomory cut, see [27] for the definition of P_G.

4.2.2 Special Ordered Set

A collection of binary varaibles is called a special ordered set of type 1 (SOS) if all the variables except one are zero. What is interesting about SOS, is that they can be treated by the solver as a single entity. That is the set can be internally replaced by the solver with a single integer variable which marks the position of the nonzero binary variable.

In the formulations discussed in the previus chapter, by substitung constraints of the form $\sum_{k=1}^{K} b_k \leq K - 1$ with $\sum_{k=1}^{K}(1 - b_k) = 1$ we get convexity constraints. The binary variables associated with these constraint can now be treated as a special ordered set. Such a manipulation dramatically reduces the number of binary variables and thus the computation time.

4.3 Using Visibility Graph to Reduce Feasable Set

An algorithm to generate a line segment path which approximates the optimal path, is described in [2]. Given a line segment path a good guess of the binary variable sequences can be created. One way to do this is to assume that the vehicle travels at a constant speed, select N equidistant points along the path and check the status of each obstacle constraint at every point. We can use this binary pattern guess to try and reduce the computational time required to solve the problem.

4.3.1 Test Problem Formulation

The problem considered as a test case is the single vehicle fixed arrival time problem described in section 6.2 of [25]. This problems formulation minimizes the energy spent while a vehicle tries to reach a target state and at the same time avoids obstacles on its way. We consider this to be a good test problem since it is at the hart of most problem formulations in [25]. The test problem formulation follows:

$$\min_{u_{i,j}} \sum_{i=0}^{N-1} \sum_{j=1}^{n_u} r_j \left| u_{i,j} \right| \tag{4.2}$$

$$subject\ to: \mathbf{s}_{t+1} = \mathbf{A}\mathbf{s}_t + \mathbf{B}\mathbf{u}_i \tag{4.3}$$

$$\mathbf{s}_0 = [x_0\ y_0\ \dot{x}_0\ \dot{y}_0] \tag{4.4}$$

$$\mathbf{s}_{N-1} = [x_f\ y_f\ \dot{x}_f\ \dot{y}_f] \tag{4.5}$$

$$x_i \leq x_{c,min} + Mt_{c,i,1} \tag{4.6}$$

$$-x_i \leq -x_{c,max} + Mt_{c,i,2} \tag{4.7}$$

$$y_i \leq x_{c,min} + Mt_{c,i,3} \tag{4.8}$$

$$-y_i \leq -y_{c,max} + Mt_{c,i,4} \tag{4.9}$$

$$\sum_{k=1}^{4} t_{c,i,k} \leq 3 \tag{4.10}$$

$$t_{c,i,k} \in \{0, 1\} \tag{4.11}$$

$$i \in \{0 \ldots N-1\} \tag{4.12}$$

$$k \in \{1 \ldots 4\} \tag{4.13}$$

$$c \in \{1 \ldots L\} \tag{4.14}$$

The particular instance of the problem used to test the software has the following characteristics:

$N = 20$: Number of time steps

$L = 10$: Number of obstacles

$n_u = 2$: Number of control inputs

An MPS file of this problem has been created using AMPL. It should be noted that the AMPL preprocessor was disabled in order to retain all variables and constraints in the MPS output. This allows tracking of variables and constraints in the AMPL model down to the MPS file and subsequently the GLPK internal problem representation.

The test problem MPS file contains 1129 constraints and 1004 variables, of which 800 are binary. The problems matrix consists of 2848 non-zero elements. Without any modification to the source code GLPK solves the test problem in 79 seconds. It takes GLPK 11505 simplex iterations and 6471 branch-and-bound nodes to solve the problem. The same problem is solved by CPLEX in 1.4 seconds. It takes CPLEX 1366 simplex iterations and 361 branch-and-bound nodes to solve the problem.

Time Step	Binary Variables									
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1	1	1
6	1	1	1	0	0	1	1	1	1	1
7	1	1	1	0	0	0	1	1	1	1
8	1	1	1	0	0	0	1	1	1	1
9	1	1	1	0	0	0	1	1	1	1
10	1	1	1	0	0	0	1	1	1	1
11	1	1	0	0	0	0	1	1	1	1
12	0	1	0	0	0	0	1	1	1	1
13	0	1	0	0	0	0	1	1	1	1
14	0	1	1	0	0	0	1	1	1	1
15	0	1	0	0	0	0	1	1	1	1
16	0	1	0	0	0	0	1	1	1	1
17	0	0	0	0	0	0	0	1	1	1
18	0	0	0	0	0	0	0	1	1	0
19	0	0	0	0	0	0	0	1	1	0

Table 4.1: Optimal solution binary variables corresponding to left side of obstacles. Not all binary variables are shown to save space. Still one can see that the vectors are highly structured.

4.3.2 Problem Structure

CPLEX is arguably the best solver on the market right now. The fact that CPLEX does better than GLPK is not a surprise, it is an indication though that GLPK is not fully taking advantage of the structure of the problem. In table 4.1 we show how the binary variables, corresponding to the constraints, evolve through time in the optimal solution.

It is easily seen that the binary vectors are highly structured. Taking advantage of the structure is not as easy as recognizing it exists. In the next subsection we propose a way of strengthening the formulation based on our observations. There are several ways one can customize GLPK to improve its performance when solving this problem.

4.3.3 Improving Performance by Fixing Binary Variables

It many cases is safe to assume that the optimal path will go through the same openings, in between the obstacles, as in the case where a imaginary vehicle could make arbitrarily sharp turns. Using further this assumption it is reasonable think that the binary patterns will be the same for the imaginary vehicle and the actual vehicles optimal binary sequences. For example, suppose that for the variable $t_{1,i,1}$ the optimal sequence is $t_{1,i,1} = [11111000000000111111]$ we would expect the guess to also have the following pattern: all ones then all zeros and all ones again. On the other hand because of the constant speed assumption the time at which an obstacle constraint gets violated (or satisfied) might not be the same. So the guess could look something like this $t_{1,i,1} = [11111110000000011111]$. We call the position in the binary sequence were we get a change in the variables value, a *seam* . For example in the latest vector we have two seams in positions 8

and 16.

Due to the constant velocity assumption the position of the seams in the optimal binary vector may be shifted with respect to those in the guess vector. If this was not the case we could actually fix all the binary variables, which would tighten the formulation all the way down to the convex hull of the discrete set. Having the convex hull would transform the MILP problem to an LP problem making it much easier to solve. The next best thing we can do is try to account for the uncertainty in the positions of the seams.

One way of dealing with potentially shifted seams is to declare a radius of uncertainty around each one of them. The radius is an integer number of time steps. Intuitively one can think of the following situation. Suppose that in the optimal solution the vehicle is to make a turn around the corner of an obstacle, in our guess the vehicle makes the same turn at approximately the same time. It is the time uncertainty that we are trying to compensate for. What the magnitude of this radius should be is not obvious.

Suppose that we have decided what that radius should be. We describe here an algorithm that takes the binary guess vectors and the radius as inputs and strengthens the formulation accordingly. For each binary guess vector, look at each element. If its value is different from any of the other elements within the radius, mark this element as uncertain. Now, fix all the elements that were not marked to the value specified by the guess. Doing this can dramaticly reduce the size of the solution space.

The mentioned algorithm has been implemented using GLPK. This implementation loads the problem file, the radius and the guess binary array, it then tightens the formulation by adding constraints to the internal problem representation. Branch and bound with the Tomlin heuristic is then applied to the problem. The problem was solved for all radiuses from 0 to 20. For each run we recorded the time it took to arrive to an optimal solution and the time it took to fully solve the problem. The results of all 21 runs are shown in figure 4-1.

As input guess vector we used an actual optimal vector. This allowed us to do a zero radius run. When the radius is zero we are solving a linear program, as oposed to a MILP. Using a vector with shifted seams with respect to the optimal ones should result in similar performance given that the radius is big enough to accommodate the shifts. Using a radius equal to the total number of time steps means that no assumptions whatsoever are made about any binary variables. In this case, results obtained with a radius of twenty are similar in computational time to results that do not use a guess at all.

It is interesting to see how the computation of the total time grows steeply as the radius increases. On the other hand the time it takes to get to an optimal solution does not increace as dramatically as the radius is varied. Also note that even when solving just the LP (radius=0) it still takes one second to solve the problem.

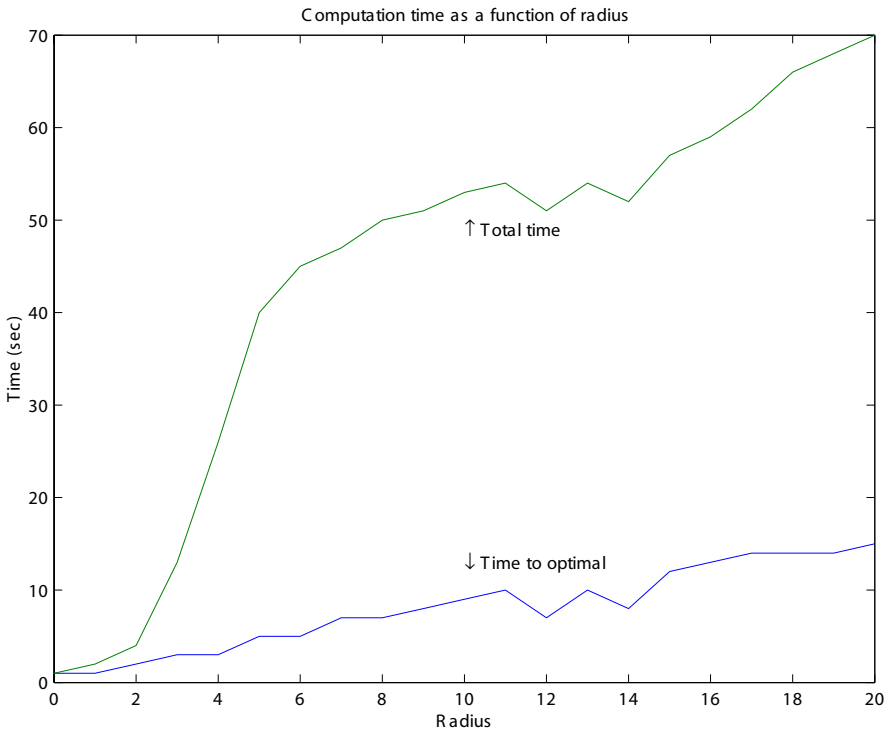

Figure 4-1: Time to reach optimal solution vs radius.

Appendix A

GLPK

A.1 Introduction

We discuss here the MILP solver selection process we went through and give additional information for the use of the chosen solver. The software selection is based on its performance and embedability. Performance, is simply the time it takes for the program to solve a problem instance. We define embedability as the portability of the code to platforms likely to be found in embedded systems. In our case we are interested in QNX and Linux. It is also important for the code to run under Windows because that makes development faster. Code readability is also considered important.

A.2 Free Codes

Several free codes for solving MILP problems are available in the public domain. In order to choose one of them we compared the codes in two major areas, speed and embedability. Fortunately at the time of this writing an up to date speed benchmark of the free codes was available online [16].

A.2.1 The Codes

The most popular free codes are GLPK, BonsaiG, MOMIP and LP_Solve. GLPK [1] is the most suitable code for our needs. This decision is based on the benchmarks, described bellow, and on the portability of the code. GLPK is intended for large scale optimization. It is written in ANSI C and is completely self contained. GLPK supports the MPS file format as one of the options for entering problem formulations to the solver.

BonsaiG [11] is one of the faster codes. BonsaiG also supports the MPS file format. Unfortunately it has Fortran component to it, a language which is hard to support on some embedded platforms like QNX 4. BonsaiG was also tested with the path planing problem used throughout this report;

Problem	Rows	Col	Integer	Nonzero
air04	824	8904	8904	81869
air05	427	7195	7195	59316
bell5	92	104	58	266
cap6000	2177	6000	6000	54238
l152lav	98	1989	1989	11911
misc07	213	260	259	8620
nw04	37	87482	87482	636666
pk1	46	86	55	915
stein45	332	45	45	1079
eilD76	76	1898	1898	21009
irp	40	20315	20315	118569
mas284	69	151	150	9782
bienst1	577	505	28	2185
dano3_3	3203	13873	423	79656
dano3_4	3203	13873	569	79656
dano3_5	3203	13873	708	79656
mksh1_1	7	62	45	324
mksh2_1	8	74	54	448
neos2	1104	2101	1040	7330
neos4	38578	22884	17136	116040
neos5	36703	21124	17136	109068
neos8	46325	23228	23228	313212
neos10	46794	23489	23489	251230
nug08	913	1632	1632	8304
qap10	1821	4150	4150	20810
swath2	885	6805	2213	34966
acc0	1738	1620	1620	7291
acc1	2287	1620	1620	12979
acc2	2521	1620	1620	15328

Table A.1: Benchmark problem characteristics.

the test resulted in an unsuccessful run. The test was performed remotely by submitting the MPS file to the NEOS server [19].

MOMIP [20] neither did well on the benchmarks nor is its source code available. It was briefly tested without much success using a binary distribution. When it was run using the path planning problem the code did not return an answer. LP_Solve [15] did poorly on the MILP benchmarks, it also failed to solve the path planning problem.

A.2.2 Benchmarks

The free codes have been tested using a collection of MILP problems available on the web. Table A.1 lists each problems major characteristics. That is, the problem name, the number of constraints, variables, integer variables and nonzero entries. The problem collection covers a broad range of MILP structures.

Table A.2 lists the times recorded in free code bench marks conducted in [16]. The problems used for the tests are all MILP problems. From the results GLPK seems to be the most reliable and fastest of the free codes. CPLEX as expected did better than all the free solvers.

Problem	GLPK	BonsaiG	MOMIP	LP_SOLVE	CPLEX
air04	—	534	—	—	58
air05	—	1868	—	—	47
bell5	37	93	—	461	1
cap6000	—	1944	1107	—	3
l152lav	44	22	221	62	2
misc07	92	116	385	104	121
nw04	1344	657	664	—	18
pk1	—	1814	629	2944	194
stein45	177	64	84	218	20
eilD76	1161	692	—	—	286
irp	165	1257	—	—	5
mas284	112	—	165	—	24
bienst1	357	3144	—	—	865
dano3_3	103	—	—	—	487
dano3_4	121	—	—	—	378
dano3_5	1665	—	—	—	1301
mksh1_1	—	—	9	—	290
mksh2_1	—	—	11	—	248
neos2	2277	—	—	—	210
neos4	2985	—	—	—	23
neos5	2629	—	—	—	19
neos8	170	6	—	—	196
neos10	—	2488	—	—	638
nug08	14	—	—	880	51
qap10	267	—	—	—	429
swath2	1558	—	—	—	771
acc0	—	10	—	—	1
acc1	43	27	—	—	5
acc2	347	208	—	—	26
acc3	747	669	—	—	100

Table A.2: Free code benchmarks vs. CPLEX. All times are in seconds. Dashes indicate that computation time exceeded the one hour time limit.

A.3 GLPK Source Code

GLPK was compiled under three platforms; namely, Linux, Windows and QNX. The code appears to be highly portable. We describe here any steps required for compiling GLPK under different platforms. The GLPK source code is available from [1] as a tar.gz file. The instructions here refer to GLPK-3.2.3 which was released in November of 2002.

A.3.1 Linux Compilation

Linux is GLPK's native platform. So it compiles as is. To compile the code four steps are needed.

1. Unpack the distribution % tar zxvf glpk-3.2.3.tar.gz
2. Switch to the created directory % cd glpk-3.2.3
3. Run the configuration script % ./configure
4. compile the code % make

A.3.2 Windows Compilation

Command Prompt Compilation

The batch file listed in [10] can be used to create a dll of the GLPK routines. The batch file makes use of the Visual C++ V6 compiler and contains detailed instructions on how to perform the compilation. This batch file was released with GLPK-3.2 and needs two additional lines to work with GLPK-3.2.3. In particular, the following two lines need to be added to the list of similar statements in the batch file.

```
echo glp_lpx_write_lpt >> glpk.def

echo glp_lpx_read_lpt >> glpk.def
```

If all goes well the batch file ends by calling the freshly compiled standalone solver with a test case MPS file.

Integrated Development Environment Compilation

For code development purposes it is preferable to work on a Windows machine. The use of a modern IDE like VC++ 6.0 make code development faster. When working with QNX for example we have always developed any math related code under windows and then just recompiled it on QNX. A VC++ project and workspace can easily be made out of the GLPK source distribution by following a few steps.

1. Create an empty console project and workspace.
2. Copy the "include", "source", and "sample" folders from the GLPK distribution directory to the new projects directory.
3. Add the source files and include files to the project trees "Source Files" and "Include Files" folders.
4. From the "sample" folder add glpsol.c to the "Source Files" folder.
5. Finally, go to Project->Settings C/C++ tab and under the Preprocessor category specify the additional include directory "./include".

At this point one should be able to compile the code. This provides a nice environment in which to explore and develop the source code.

A.3.3 QNX Compilation

1. Unpack the distribution % gzip -cd glpk-3.2.3.tar.gz — pax -vr

2. Switch to the created directory % cd glpk-3.2.3

3. Run the configuration script % sh configure

4. Edit the Makefile line "(AR) cru libglpk.a (OBJS)" to "(AR) -c libglpk.a (OBJS)"

5. Change the Makefile CFLAGS variable to "CFLAGS = -w9 -g -Or -5 -Ot -M -fpi87"

6. compile the code % make

Appendix B

Tomlin Branching Heuristic

Here we present both a graphical and tabular illustrations of the Tomlin branching heuristic used by GLPK. To do so we have selected a single obstacle problem with twelve time steps. The fixed arrival time in this problem is specified in such a way that a feasible trajectory is only possible when the vehicle passes to the north of the obstacle. This results to less branching making the following illustrations shorter.

First we show the graphical representation of the branching algorithm. Later we tabulate the recorded binary values, which gives good insight in how the heuristic works.

B.1 Graphical Representation of Branching Procedure

In figure B-1 we plot all the LP relaxations solved while GLPK fixes one binary variable at a time. Since no feasible path going south of the obstacle exists, the paths are clustered in two major groups; the paths relaxing side number four and the ones that do not. Two interesting facts can be noted in B-1. First that a grate number of nodes have identical LP relaxations, that is they produce the same path. Only when an when a restriction is imposed by fixing a binary variable to zero does the path change. Second, variables corresponding to time steps that fall within an obstacle, are chosen more often. This is because they have higher penalties associated with them.

B.2 Tabular Representation of Branching Procedure

The following table shows the binary variable branching selection as the solver computes a solution. Each line corresponds to one branching decision, which uses P_u and P_d to decide which binary to bound next. Dashes represent variables which do nor satisfy integrality. A star marks the selected branching variable. Place holders marked with a one or zero represent the value of variables satisfying integrality. The first four columns correspond to the binary varibles associated with each

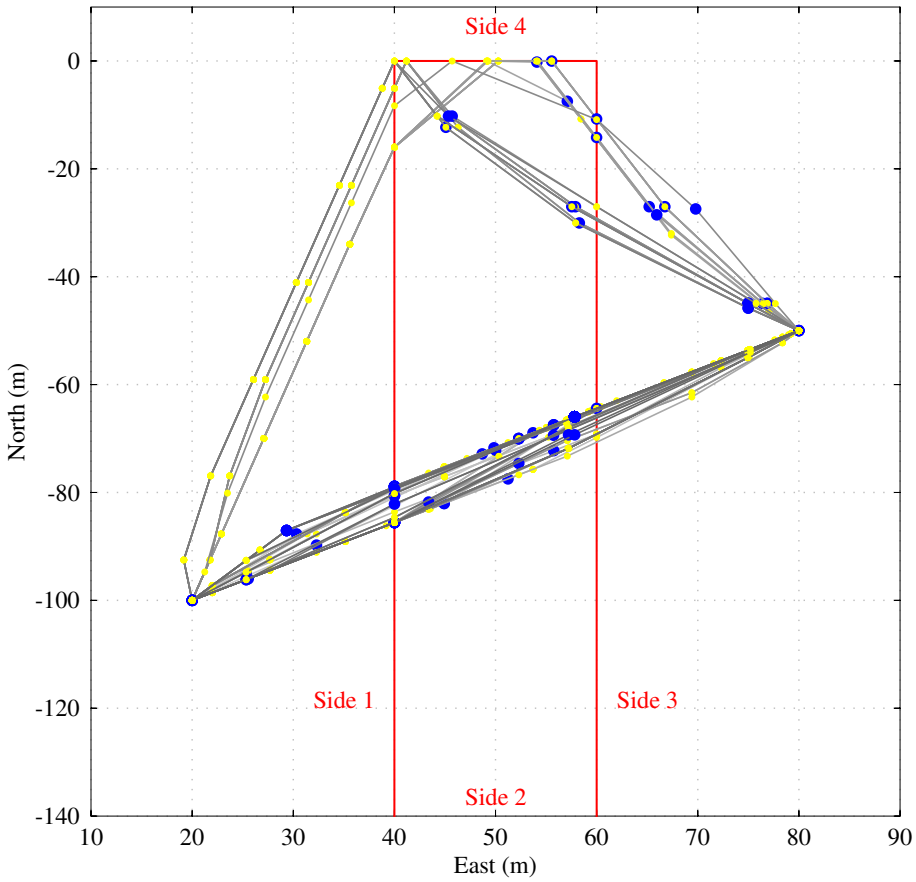

Figure B-1: This plot illustrates all the LP relaxations solved while fixing one binary variable at a time. The yellow dots along the paths, mark the the position of a vehicle at each time step. The blue dots indicate that the branching variable corresponds to that time step. Note that many branches are created while the path stays unchanged.

of the obstacles side. There is one of them for each side and time step. The first time step is excleded as no variables are required. The fifth column is the branching decision number. The next column corresponds to the side associated with the selected variable. The letter j marks the selecled binary variable number. The up and down penalties for the selected binary variable follow. The table follows.

```
0----------   ----------   *-------000   ----------     1 i= 2, side=3, j= 3, pu=0.00, pd=INF
00--------*   ----------   1--------00   ----------     2 i=12, side=1, j=41, pu=0.00, pd=INF
00--------1   ----------*   1--------00   ----------     3 i=12, side=2, j=42, pu=0.00, pd=INF
00--------1   ----------1   1--------00   ----------*    4 i=12, side=4, j=44, pu=0.00, pd=INF
00--------1   *---------1   1--------00   ----------1    5 i= 2, side=2, j= 2, pu=0.00, pd=0.00
0---------1   1---------1   1--------00   *---------1    6 i= 2, side=4, j= 4, pu=0.00, pd=INF
00--------1   1*--------1   1--------00   1---------1    7 i= 3, side=2, j= 6, pu=0.00, pd=0.00
0*--------1   11--------1   1-------000   1---------1    8 i= 3, side=1, j= 5, pu=0.00, pd=0.00
01--------1   11--------1   1*------000   1---------1    9 i= 3, side=3, j= 7, pu=0.00, pd=INF
Backtracking 7 times
```

```
00--------1   11--------1   1*-------00   1---------1    10 i= 3, side=3, j= 7, pu=0.00, pd=INF
00--------1   11--------1   11-------00   1*--------1    11 i= 3, side=4, j= 8, pu=0.00, pd=0.00
00*-------1   11--------1   11-------00   11--------1    12 i= 4, side=1, j= 9, pu=0.00, pd=0.00
001-------1   11*-------1   11------000   11--------1    13 i= 4, side=2, j=10, pu=0.00, pd=0.00
001-------1   111-------1   11*00000000   11--------1    14 i= 4, side=3, j=11, pu=0.00, pd=0.00
Backtracking
000-------1   11--------1   11*------00   11--------1    15 i= 4, side=3, j=11, pu=0.00, pd=INF
0000------1   11*-------1   111------00   11--------1    16 i= 4, side=2, j=10, pu=0.00, pd=0.00
000----*--1   111-------1   111-----000   11--------1    17 i= 9, side=1, j=29, pu=0.00, pd=0.00
000----1--1   111-------1   111----0000   11*-------1    18 i= 4, side=4, j=12, pu=0.00, pd=0.00
000*---1--1   111-------1   111-----000   111-------1    19 i= 5, side=1, j=13, pu=0.00, pd=0.00
0001---1--1   111*------1   111-0000000   111-------1    20 i= 5, side=2, j=14, pu=0.00, pd=0.00
0001---1--1   1111------1   111*0000000   111-------1    21 i= 5, side=3, j=15, pu=0.00, pd=0.00
Backtracking  4 times
000000-1--1   111-------1   111*----000   111-------1    22 i= 5, side=3, j=15, pu=0.00, pd=INF
000000-1--1   111*------1   1111----000   111-------1    23 i= 5, side=2, j=14, pu=0.00, pd=0.00
000000-1--1   1111------1   1111----000   111*------1    24 i= 5, side=4, j=16, pu=0.00, pd=0.00
000000-1--1   1111*-----1   1111----000   1111------1    25 i= 6, side=2, j=18, pu=0.00, pd=0.00
0000*--1--1   11111-----1   1111---0000   1111------1    26 i= 6, side=1, j=17, pu=0.00, pd=0.00
00001--1--1   11111-----1   1111*--0000   1111------1    27 i= 6, side=3, j=19, pu=0.00, pd=0.00
Backtracking
000000-1--1   11111-----1   1111*---000   1111------1    28 i= 6, side=3, j=19, pu=0.00, pd=INF
000000-1--1   11111-----1   11111---000   1111*-----1    29 i= 6, side=4, j=20, pu=0.00, pd=0.00
000000-1--1   11111*----1   11111---000   11111-----1    30 i= 7, side=2, j=22, pu=0.00, pd=0.00
00000*-1--1   111111----1   11111--0000   11111-----1    31 i= 7, side=1, j=21, pu=0.00, pd=0.00
000001-1--1   111111----1   11111*-0000   11111-----1    32 i= 7, side=3, j=23, pu=0.00, pd=0.00
000001-1--1   111111*---1   111111--000   111110----1    33 i= 8, side=2, j=26, pu=0.00, pd=INF
000001-1-*1   1111111---1   111111--000   1111100---1    34 i=11, side=1, j=37, pu=0.00, pd=0.00
000001-1-11   1111111---1   111111-0000   1111100--*1    35 i=11, side=4, j=40, pu=0.00, pd=0.00
000001*1-11   1111111---1   111111-0000   111110---11    36 i= 8, side=1, j=25, pu=0.00, pd=0.00
00000111-11   1111111---1   111111*0000   1111100--11    37 i= 8, side=3, j=27, pu=0.00, pd=0.00
00000111*11   1111111---1   1111111-000   1111100--11    38 i=10, side=1, j=33, pu=0.00, pd=0.00
00000111111   1111111*--1   11111110000   1111100--11    39 i= 9, side=2, j=30, pu=0.00, pd=0.00
00000111111   11111111-*1   11111110000   1111100--11    40 i=11, side=2, j=38, pu=0.00, pd=0.00
00000111111   11111111*11   11111110000   1111100--11    41 i=10, side=2, j=34, pu=0.00, pd=0.00
00000111111   11111111111   11111110000   1111100*-11    42 i= 9, side=4, j=32, pu=0.00, pd=0.00
00000111111   11111111111   11111110000   11111001*11    43 i=10, side=4, j=36, pu=0.00, pd=0.00
Found better integer solution
Backtracking  2 times
000000-1--1   111111----1   11111*--000   11111-----1    44 i= 7, side=3, j=23, pu=0.00, pd=INF
000000-1--1   111111----1   111111--000   11111*----1    45 i= 7, side=4, j=24, pu=0.00, pd=0.00
000000*1--1   111111----1   111111--000   111111----1    46 i= 8, side=1, j=25, pu=0.00, pd=0.00
00000011--1   111111*---1   111111-0000   111111----1    47 i= 8, side=2, j=26, pu=0.00, pd=0.00
00000011--1   1111111*--1   111111-0000   111111----1    48 i= 9, side=2, j=30, pu=0.00, pd=0.00
00000011--1   11111111--1   111111*0000   111111----1    49 i= 8, side=3, j=27, pu=0.00, pd=0.00
```

```
00000011--1   11111111--1   1111111--00   1111110--*1    50 i=11, side=4, j=40, pu=0.00, pd=7522.42
00000011--1   11111111-*1   1111111--00   1111110--11    51 i=11, side=2, j=38, pu=0.00, pd=42.33
00000011--1   11111111-11   1111111--00   1111110-*11    52 i=10, side=4, j=36, pu=0.00, pd=191.15
00000011--1   11111111*11   1111111--00   1111110-111    53 i=10, side=2, j=34, pu=0.00, pd=13.57
00000011-*1   11111111111   1111111--00   1111110-111    54 i=11, side=1, j=37, pu=0.00, pd=91.86
00000011-11   11111111111   1111111-000   1111110*111    55 i= 9, side=4, j=32, pu=0.00, pd=3.57
00000011*11   11111111111   11111110000   11111101111    56 i=10, side=1, j=33, pu=0.00, pd=2.07
Found better integer solution
Backtracking  4 times
00000001--1   111111----1   111111*--00   111111----1    57 i= 8, side=3, j=27, pu=0.00, pd=INF
00000001--1   111111*---1   1111111--00   111111----1    58 i= 8, side=2, j=26, pu=0.00, pd=0.00
00000001--1   1111111---1   1111111--00   111111*---1    59 i= 8, side=4, j=28, pu=0.00, pd=3.41
00000001--1   1111111*--1   1111111--00   1111111---1    60 i= 9, side=2, j=30, pu=0.00, pd=0.00
00000001--1   11111111--1   1111111*000   1111111---1    61 i= 9, side=3, j=31, pu=0.00, pd=0.00
Backtracking  3 times
00000001--1   1111111---1   1111111--00   1111110--*1    62 i=11, side=4, j=40, pu=0.00, pd=16007.93
00000001--1   1111111*--1   1111111--00   1111110--11    63 i= 9, side=2, j=30, pu=0.00, pd=0.00
00000001--1   11111111--1   1111111--00   1111110-*11    64 i=10, side=4, j=36, pu=0.00, pd=406.78
00000001--1   11111111-*1   1111111--00   1111110-111    65 i=11, side=2, j=38, pu=0.00, pd=249.58
00000001-*1   11111111-11   1111111--00   1111110-111    66 i=11, side=1, j=37, pu=0.00, pd=309.30
00000001-11   11111111-11   1111111-000   1111110*111    67 i= 9, side=4, j=32, pu=0.00, pd=0.00
Backtracking  12 times
00000000--1   111-------1   111----*-00   11--------1    68 i= 9, side=3, j=31, pu=0.00, pd=INF
00000000--1   111-------1   111----1-00   11*-------1    69 i= 4, side=4, j=12, pu=0.00, pd=0.00
00000000--1   111*------1   111----1-00   111-------1    70 i= 5, side=2, j=14, pu=0.00, pd=0.00
00000000--1   1111------1   111*---1-00   111-------1    71 i= 5, side=3, j=15, pu=0.00, pd=0.00
00000000--1   1111------1   1111---1-00   111*------1    72 i= 5, side=4, j=16, pu=0.00, pd=0.00
00000000--1   1111*-----1   1111---1-00   1111------1    73 i= 6, side=2, j=18, pu=0.00, pd=9.83
00000000--1   11111-----1   1111--*1-00   1111------1    74 i= 8, side=3, j=27, pu=0.00, pd=25.73
00000000--1   11111-----1   1111*-11-00   1111------1    75 i= 6, side=3, j=19, pu=0.00, pd=0.00
00000000--1   11111-----1   11111-11-00   1111*-----1    76 i= 6, side=4, j=20, pu=0.00, pd=0.00
00000000--1   11111*----1   11111-11-00   11111-----1    77 i= 7, side=2, j=22, pu=0.00, pd=0.00
00000000--1   111111----1   11111*11-00   11111-----1    78 i= 7, side=3, j=23, pu=0.00, pd=7.23
00000000--1   111111----1   11111111-00   11111*----1    79 i= 7, side=4, j=24, pu=0.00, pd=0.00
00000000--1   111111*---1   11111111-00   111111----1    80 i= 8, side=2, j=26, pu=0.00, pd=0.00
00000000--1   1111111---1   11111111-00   111111*---1    81 i= 8, side=4, j=28, pu=0.00, pd=0.00
00000000--1   1111111*--1   11111111-00   1111111---1    82 i= 9, side=2, j=30, pu=0.00, pd=0.00
00000000--1   11111111--1   11111111-00   1111111*--1    83 i= 9, side=4, j=32, pu=0.00, pd=0.00
00000000*-1   11111111--1   11111111-00   11111111--1    84 i=10, side=1, j=33, pu=0.00, pd=0.00
000000001-1   11111111*-1   11111111-00   11111111--1    85 i=10, side=2, j=34, pu=0.00, pd=0.00
000000001-1   111111111-1   11111111*00   11111111--1    86 i=10, side=3, j=35, pu=0.00, pd=0.00
Backtracking  28 times
```

Appendix C

Magnetic Compass

C.1 Introduction

C.1.1 Design Overview

We present here an electronic compass design based on the Honeywell HMC2003 magnetic sensors. The compass, shown in Figure C-1, is part of the helicopters sensor suit and provides low frequency heading information to the state estimator. The sensor has three analog outputs, it is sampled using the on board computers 12-bit A/Ds. The sampling rate is 20Hz. Using the compass, heading estimation error remains bellow 10 degrees irrespective of flight time. This is not the case when heading is estimated by open loop gyro integration.

The HMC2003 integrates three orthogonal sensing elements which can measure fields from 40 micro Gauss to ± 2 Gauss. The sensor has three analog outputs whose sensitivity is 1V/Gauss. The outputs have a 2.5V offset to accommodate the negative measurements. [13]

Integration of the HMC2003 with the rest of the helicopters avionics package requires some support circuitry and a reliable calibration procedure; such a circuit and procedure are described here. The support circuit performs two functions; it filters the sensor outputs and it periodically eliminates magnetic history effects. The calibration procedure is used to calculate scale and offset

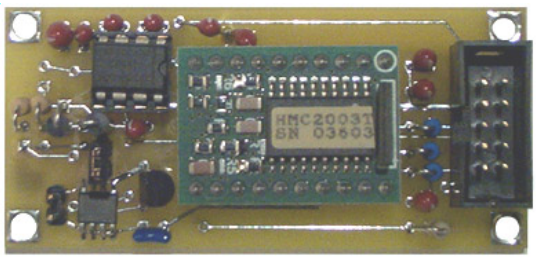

Figure C-1: Compass shown in actual size.

factors which are then applied as corrections to subsequent sensor measurements.

C.1.2 Background Information

The earths magnetic field at any given point on its surface can be represented by a vector. The angle between the vector and the earths surface is called the dip or inclination angle. In the Boston area the dip angle is approximately 70 degrees [14] . The magnetic fields strength is about 0.6 Gauss and thus the fields horizontal component has a magnitude of $0.6\cos(70) \approx 0.2$ Gauss. Given the magnetic fields horizontal component magnitude and the desired heading accuracy, the sensor requirements can be derived using the following formula:

$$\begin{matrix}\text{Horizontal}\\ \text{Component}\\ \text{Magnitude}\end{matrix} \times \tan\left(\begin{matrix}\text{Desired}\\ \text{Heading}\\ \text{Accuracy}\end{matrix}\right) = \begin{matrix}\text{Required Magnetic}\\ \text{Measurement}\\ \text{Accuracy}\end{matrix} \tag{C.1}$$

To achieve one degree heading accuracy in the Boston area magnetic measurements with accuracy better than $0.2\tan(1deg) \approx 0.0035$ Gauss are necessary.

C.2 Electronics

C.2.1 Compass Circuit

Figure C-2 shows the compass circuit diagram. The Honeywell sensing element (U3) has a current strap used to eliminate biases caused by magnetic history effects. Applying a 3-4 amp current pulse through the straps is required to eliminate such biases. The direction of the current determines if the sensor is the "Set" or "Reset" state. The sensor is equally sensitive in both these states, in the "Set" state the output sensitivity is 1V/Gauss and in the "Reset" state it is -1V/Gauss. Several circuits for generating this Set/Reset pulses are discussed in [12].

This design integrates the HMC2003 with a variation of the HEXFET based set/reset circuit in [12]. Timing for the set/reset pulses is provided by an external timing source, the helicopters computer. Since the computer also samples the sensor using its 12-bit A/D converter, having it trigger the set/reset pulses assures that no samples are obtained during a set/reset operation. This is essential since sampling the sensor at such a time will result in erroneous measurements.

The sensor is sampled by the computer at 20Hz. Anti-aliasing filters with 4.9Hz cutoff frequency are place on all three sensor outputs. Note that the sensors bandwidth is still sufficient to capture the helicopters rigid body dynamics whose bandwidth is bellow 2Hz in all axis.

As stated earlier the included set/reset circuit is a variation of the HEXFET based circuit found in [12]. Here the IRF7105 MOSFET is used instead of the IRF7106. The later has the least $R_{DS(on)}$

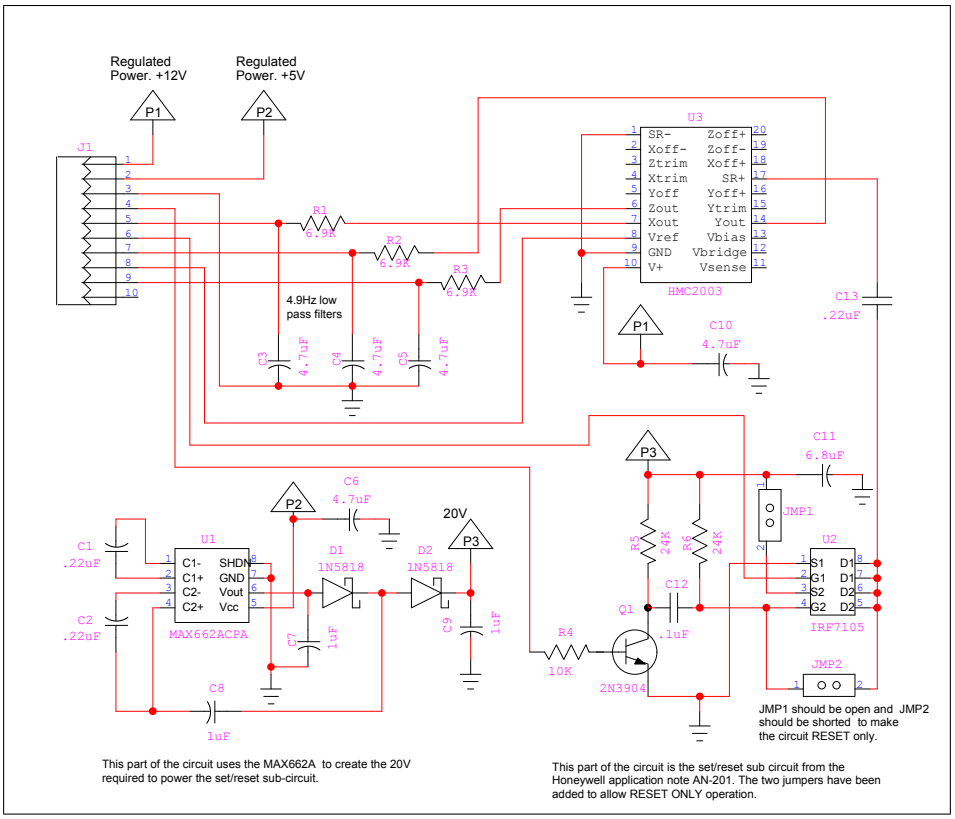

Figure C-2: Compass Circuit.

of the two, which is preferable. Due to lack of availability, the IRF7105 is used instead of the IRF7106. Despite this, circuit performance is still satisfactory. The jumpers J1,J2 were added to the original circuit to allow "Reset Only" operation when desired. When J1 is shorted and J2 is open, the circuit is in set/reset mode. When J1 is open and J2 is shorted only reset pulses can be generated.

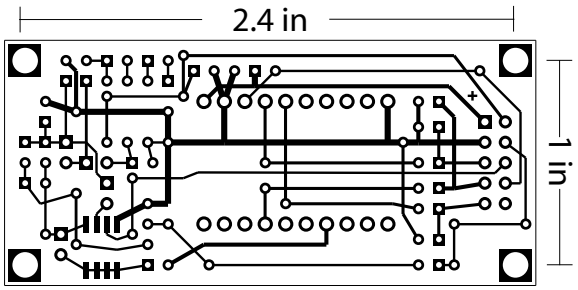

Figure C-3: Compass PCB shown in actual size.

DB9 Pin#	IDC10 Pin#	Pin Description
1	1	+12V in
2	3	Ground
3	5	Compass X Axis Output 0-5V
4	7	Compass Y Axis Output 0-5V
5	9	Compass Z Axis Output 0-5V
6	2	+5V in
7	4	Set triger input
8	6	Reset triger input
9	8	Compass 2.5V Referance Output
N/A	10	Not Connected

Table C.1: Compass Connector Pinout.

Drawing Referance	Description	Manufacturer	Manufacturer Part#	Qty.
U3	Three-Axis Magnetic Sensor	Honeywell	HMC2003	1
U1	Flash Programing Supply	Maxim	MAX662ACPA	1
U2	HEXFET	International Rectifier	IRF7105	1
C1,2	.22uF 50VDC Tantalum Cap.	Panasonic - ECG	ECS-F1HE224	2
C3,4,5,13	4.7uF 16V Tantalum cap.	Panasonic - ECG	ECS-F1CE475K	5
R1,2,3	6.98K 1/4W 1% Res	Yageo	MFR-25FBF-6K98	3
R5,R6	24K 1/4W 5% Res	Yageo	CFR-25JB-24K	2
R4	10K 1/4W 5% Carbon Film Res	Yageo	CFR-25JB-10K	1
R7,8	470 1/4W 5% Res	Yageo	CFR-25JB-470R	2
C6,7	1.0uF Tantalum Cap	Panasonic - ECG	ECS-F1VE105K	2
C11	Cap .22UF 50V	Panasonic - ECG	ECU-S1H224MEA	1
C9,12	CAP ELECT 47UF 50V	Panasonic - ECG	ECE-A1HN470U	2
Q1	NPN Transistor TO92	Fairchild Semiconductor	2N3904	1
D1,2	RECTIFIER SCHOTTKY 1A	General Semiconductor	1N5818	2
J1	SHROUDED HEADER	3M/ISD	2510-6002UB	1
N/A	IC SOCKET 8 PIN .300 GOLD	Mill-Max	110-13-308-41-001	1
JMP1,2	HEADER 2POS .100	Molex	22-10-2021	2
N/A	PIN RCPT	Mill-Max	0300-1-15-01-4727100	20
N/A	SHORTING JUMPERS	3M/ISD	929957-08	1

Table C.2: Compass Parts List.

C.2.2 Hardware Implementation

The custom printed circuit board shown in Figure C-3 has been designed to host the the compass circuit. Its mounting holes are spaced such that it can easily be mounted on the tail boom using two horizontal fin brackets.

The compass is connected to the avionics box using a flat rebon cable with a 10 pin IDC connector on one end and a male DB9 connector on the other end. Table C.1 shows the pin assignments for these two connectors. All the parts that comprise the compass are listed in Table C.2.

C.3 Calibration

Certain parts of the sensors host platform locally distort the earths magnetic field. To get accurate measurements calibration is required. The following procedure can be used to derive a set of calibration constants. These are applied to subsequent readings performed by the sensor in order to reverse

the errors caused by the magnetic field distortion. It is assumed that the parts on the platform that cause the distortion remain fixed with respect to the sensor. The sensor must be mounted in such a way that its x and y sensing elements are parallel to the ground when the platform is level.

We will first describe the x-axis and y-axis calibration, z-axis calibration will then be discussed and finally the calibration results will be discussed. It should be noted that the calibration procedure described in this section is designed such that it can easily be performed at the beginning of each helicopter flight operation.

C.3.1 X and Y Axis Calibration

Ideally, plotting the sensors x-axis and y-axis raw outputs while it is being rotated with its xy-plane level should produce a circle. In reality the resulting figure looks more like an ellipse. A collection of uncalibrated samples is shown in Figure C-4. The same data is plotted for a second time on the same plot after calibration. The uncalibrated data also has an ellipse fitted through it in a least squares sense. Fitting the ellipse to the data is part of the calibration process, the math used to do this is described in Section C.3.4.

Magnetic field data is collected while rotating the platform on a level surface. If the magnetic field is not distorted, plotting the x-axis verses the y-axes measurements produces a circle centered about 0,0 with a radius equal to the magnitude of the earths horizontal magnetic field component. Plotting data collected in the presence of magnetic field distortion results in an ellipse whose center has an offset with respect to the axes origin. By fitting the equation of an ellipse to the data using least squares method, scale and offset factors can be calculated that will project subsequent measurements to the circle described earlier.

By equating the ellipse equation to that of a circle with radius R equal to the horizontal component of the earths magnetic field the scale and offset factors can be found.

$$\left(\frac{\hat{x}}{R}\right)^2 + \left(\frac{\hat{y}}{R}\right)^2 = \left(\frac{x-\bar{x}}{a}\right)^2 + \left(\frac{y-\bar{y}}{b}\right)^2 \tag{C.2}$$

The sensor x and y axes calibrated outputs can be calculated using the following two equations.

$$\hat{x} = \frac{R}{a}(x-\bar{x}) \tag{C.3}$$

$$\hat{y} = \frac{R}{b}(y-\bar{y}) \tag{C.4}$$

$$\tag{C.5}$$

The scale and offset factors are:

$$X_{sf} = \frac{R}{a} \tag{C.6}$$

$$X_{off} = \bar{x} \tag{C.7}$$

$$Y_{sf} = \frac{R}{b} \tag{C.8}$$

$$Y_{off} = \bar{y} \tag{C.9}$$

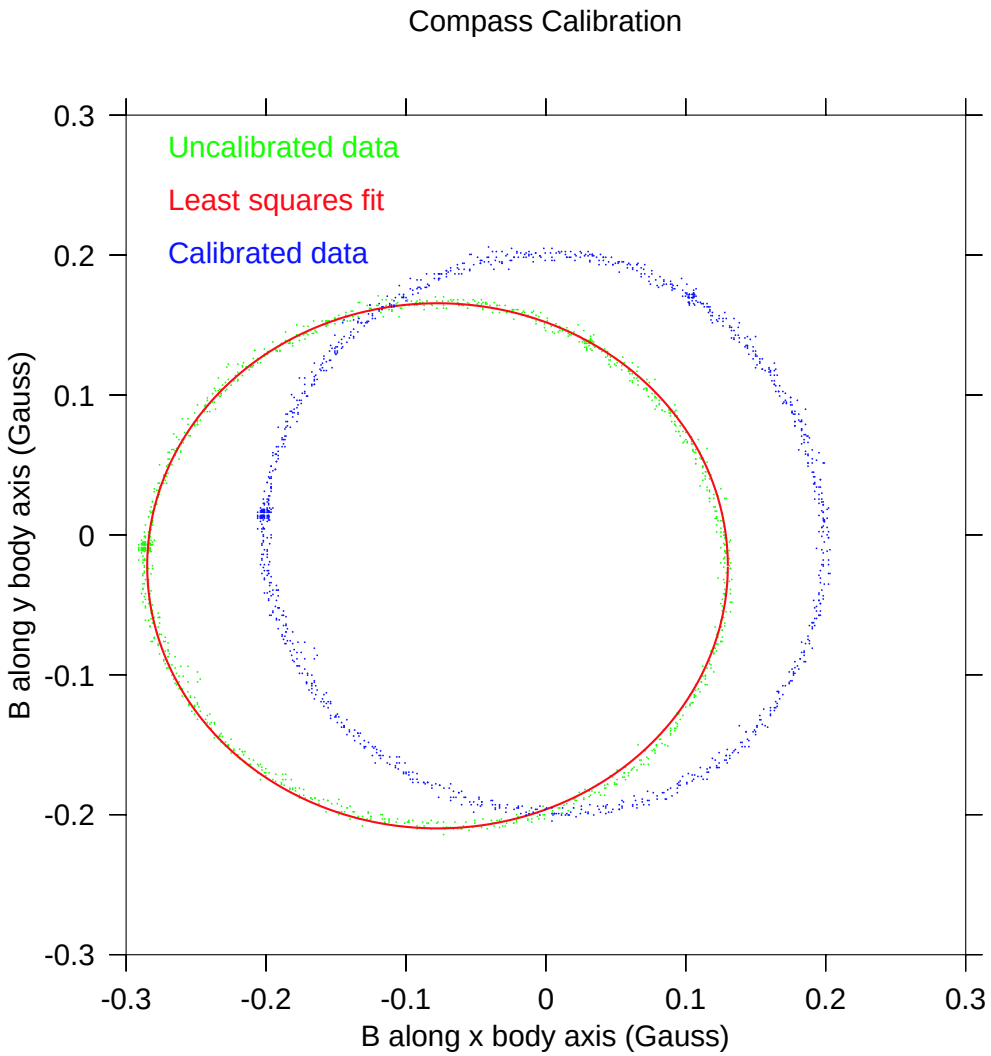

Figure C-4: Compass Calibration Plot.

C.3.2 Z Axis Calibration

The z-axis calibration could be similar to x-axis and y-axis calibration. This would be achieved by rotating the platform while maintaining the plane, formed by the sensors x and z or y and z axes, parallel to the ground. In the case of the helicopter this is more difficult and an alternative method is used.

On the helicopter the z-axis is parallel to the main rotor shaft. Data is collected while the helicopter is positioned on a level surface and while it is held in an inverted position. In both positions the main rotor shaft is perpendicular to the earths surface. The measurements are averaged for each position. The z-axis scale factor Z_{sf} and offset Z_{off} can be calculated using the following equations:

$$Z_{sf} = \frac{2Z_{local}}{Z_{av}^{Dn} - Z_{av}^{Up}} \tag{C.10}$$

$$Z_{off} = -Z_{sf}\frac{Z_{av}^{Dn} + Z_{av}^{Up}}{2} \tag{C.11}$$

Were Z_{local} is the z component of the earths local magnetic field with respect to the inertial frame. Z_{av}^{Up} and Z_{av}^{Dn} are the average values of the measurements made with the helicopter upright and inverted respectively .

C.3.3 Heading Calculation

Given one sensor sample consisting of measurements x,y and z we first need to apply the calibration constants.

$$\hat{x} = X_{sf}(x - X_{off}) \tag{C.12}$$

$$\hat{y} = Y_{sf}(y - Y_{off}) \tag{C.13}$$

$$\hat{z} = Z_{sf}(z - Z_{off}) \tag{C.14}$$

The magnetic sensor measure the magnetic field components with respect to the platforms body axis. In order to estimate heading the measurements have to be rotated into the inertial reference frame. This can be done with the following equations.

$$X_m^{NED} = \hat{x}\cos\theta + \hat{y}\sin\theta\sin\phi + \hat{z}\cos\phi\sin\theta \tag{C.15}$$

$$Y_m^{NED} = \hat{y}\cos\phi + \hat{z}\sin\phi \tag{C.16}$$

Magnetic heading can be calculated in the following way:

$$\psi_m = \tan\left(\frac{Y_m^{NED}}{X_m^{NED}}\right) \tag{C.17}$$

True heading can be calculated by subtracting the local variation:

$$\psi_{true} = \psi_m - variation \tag{C.18}$$

In the Boston area:
magnetic variation = 16 degrees
North = X = 0.189 Gauss
East = Y = -0.052 Gauss
Down = Z = 0.501 Gauss

C.3.4 Ellipse Least Squares Fit

We derive here the equations used to fit an ellipse to the uncalibrated data. We begin by writing the equation of an ellipse in the form;

$$x_n^2 + b_1 y_n^2 + b_2 x_n + b_3 y_n + b_4 = 0 \tag{C.19}$$

We compare it with the more familiar equation of an ellipse.

$$\left(\frac{x_n - \bar{x}}{a}\right)^2 + \left(\frac{y_n - \bar{y}}{b}\right)^2 = 1 \tag{C.20}$$

By expanding the terms in parenthesis and multiplying both sides of equation (C.20) by a^2 we can rewrite it in the following way.

$$x_n^2 + \left(\frac{a^2}{b^2}\right) y_n^2 + (-2\bar{x})\, x_n + \left(-2\frac{a^2}{b^2}\bar{y}\right) y_n + \left(\bar{x}^2 + \frac{a^2}{b^2}\bar{y}^2 - a^2\right) = 0 \tag{C.21}$$

This equation is now identical to (C.19) with:

$$b_1 = \left(\frac{a^2}{b^2}\right) \tag{C.22}$$

$$b_2 = (-2\bar{x}) \tag{C.23}$$

$$b_3 = \left(-2\frac{a^2}{b^2}\bar{y}\right) \tag{C.24}$$

$$b_4 = \left(\bar{x}^2 + \frac{a^2}{b^2}\bar{y}^2 - a^2\right) \tag{C.25}$$

From C.23 we have:

$$\bar{x} = -\frac{b_2}{2} \tag{C.26}$$

From C.22 and C.24 we have:

$$\bar{y} = -\frac{b_3}{2b_1} \tag{C.27}$$

From C.25 , C.22 and C.27 we have:

$$a = \sqrt{\frac{b_2^2}{4} + \frac{b_3^2}{4b_1} - b_4} \tag{C.28}$$

From C.22 we have:

$$b = \sqrt{\frac{a^2}{b_1}} \tag{C.29}$$

Now we can express equation C.19 in matrix form.

$$\begin{bmatrix} y_1^2 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ y_n^2 & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} -x_1^2 \\ \vdots \\ -x_n^2 \end{bmatrix} \tag{C.30}$$

We express equation C.30 in the following way:

$$\mathbf{Ab} = \mathbf{C} \tag{C.31}$$

Were $\mathbf{A} \in \mathbf{M}_{n,4}(\Re)$, $\mathbf{b} \in \mathbf{M}_4(\Re)$ and $\mathbf{C} \in \mathbf{M}_n(\Re)$.

We want to minimize:

$$\begin{aligned} f(\mathbf{b}) &= < \mathbf{Ab} - \mathbf{C}, \mathbf{Ab} - \mathbf{C} > \\ &= (\mathbf{Ab} - \mathbf{C})^T(\mathbf{Ab} - \mathbf{C}) \\ &= \mathbf{b}^T\mathbf{A}^T\mathbf{Ab} - \mathbf{bA}^T\mathbf{C} - \mathbf{C}^T\mathbf{A}^T\mathbf{b} + \mathbf{C}^T\mathbf{C} \end{aligned}$$

Its gradient is equal to:

$$\begin{aligned} \nabla_b f(\mathbf{b}) &= 2\mathbf{A}^T\mathbf{A}\mathbf{b} - \mathbf{A}^T\mathbf{C} - \mathbf{A}^T\mathbf{C} + 0 \\ &= 2\mathbf{A}^T\mathbf{A}\mathbf{b} - 2\mathbf{A}^T\mathbf{C} \end{aligned}$$

By setting the gradient equal to zero we get:

$$\begin{aligned} \nabla_b f(\mathbf{b}) &= 0 \Leftrightarrow \\ 2\mathbf{A}^T\mathbf{A}\mathbf{b} - 2\mathbf{A}^T\mathbf{C} &= 0 \Leftrightarrow \\ \mathbf{A}^T\mathbf{A}\mathbf{b} &= \mathbf{A}^T\mathbf{C} \Leftrightarrow \\ \mathbf{b} &= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{C} \end{aligned}$$

After $\mathbf{b}$ has been calculate Equations C.26, C.27, C.28 and C.29 can be used to find the ellipse parameters that fits the data in such a way that it minimizes the square of the error.

Bibliography

[1] A. Makhorin. GLPK (GNU Linear Programming Kit). http://www.gnu.org/software/glpk/glpk.html.

[2] J. S. Bellingham. Coordination and control of uav fleets using mixed-integer linear programming. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2000.

[3] D. Bertsimas and J.N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientic, Belmont, MA, 1997.

[4] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL A Modeling Language for Mathematical Programming.* Thomson Learning, Pacific Grove,CA, 2003.

[5] E. Frazzoli, M.A. Dahleh, and E. Feron. Real-time Motion Planning for Agile Autonomous Vehicles. In *AIAA Journal of Guidance, Control and Dynamics*, 2002.

[6] V. Gavrilets. *Autonomous Aerobatic Maneuvering of Miniature Helicopters.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2003.

[7] V. Gavrilets, I. Martinos, B. Mettler, and E. Feron. Control logic for automated aero-batic flight of miniature helicopter. In *AIAA Guidance, Navigation, and Control Conference*, Monterey CA, August 2002.

[8] V. Gavrilets, B. Mettler, and E. Feron. Nonlinear model for a small-size acrobatic helicopter. In *AIAA Guidance, Navigation, and Control Conference*, Montreal Canada, August 2001.

[9] V. Gavrilets, A. Shterenberg, I. Martinos, K. Sprague, M.A. Dahleh, and E. Feron. Avionics System for Aggressive Maneuvering. *IEEE Aerospace and Electronic Systems Magazine*, pages 38–43, September 2001.

[10] Making a DLL of GLPK 3.2 with VC++ 6.0 . http://mail.gnu.org/archive/html/help-glpk/2002-07/msg00003.html.

[11] L. Hafer. BonsaiG a Mixed-Integer Linear Programming Code. http://www.cs.sfu.ca/lou/BonsaiG/.

[12] Honeywell. *AN-201 Set/Reset Pulse Circuits for Magnetic Sensors*, 1996.

[13] Honeywell. *Three-Axis Magnetic Sensor Hybrid HMC2003*, 1997.

[14] Honeywell. *Applications of Magnetoresistive Sensors in Navigation Systems*, 1998.

[15] LP SOLVE: Linear Programming Code . `ftp://ftp.es.ele.tue.nl/pub/lp_solve/`.

[16] H. Mittelmann. Mixed integer linear programming benchmark (free codes), December 2002. `ftp://plato.asu.edu/pub/milpf.txt`.

[17] J. De Mot, V. Kulkami, and E. Feron. Spacial Distribution of Two-Agent Clusters for Efficient Navigation. In *IEEE Conference on Decision and Control*, December 2003.

[18] J. De Mot, V. Kulkami, S. Gentry, V. Gavrilets, and E. Feron. Coordinated Path Planning for a UAV Cluster. In *The First AINS Symposium*, Los Angeles, CA, May 2002.

[19] NEOS, Server for Optimization . `http://www-neos.mcs.anl.gov/neos/`.

[20] Wlodzimierz Ogryczak and Krystian Zorychta. MOMIP (Modular Optimizer for Mixed Integer Programming). `http://www.iiasa.ac.at/~marek/soft/descr.html#MOMIP`.

[21] A. Richards, Y. Kuwata, and J. How. Experimental Demonstrations of Real-time MILP Control. In *AIAA Guidance, Navigation and Control Conference*, Austin , TX, August 2003.

[22] Arthur Richards. Trajectory optimization using mixed-integer linear programming. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2000.

[23] T. Schouwenaars, E. Feron, and J. How. Safe Receding Horizon Path Planning for Autonomous Vehicles. In *40 th Allerton Conference on Communication,Control and Computing*, Allerton, IL, October 2002.

[24] T. Schouwenaars, B. Mettler, E. Feron, and J. How. Hybrid Architecture for Full-Envelope Autonomous Rotorcraft Guidance. In *American Helicopter Society 59 th Annual Forum*, Phoenix, AZ, May 2003.

[25] T. Schouwenaars, B. De Moor, E. Feron, and J. How. Mixed Integer Programming for Multi-Vehicle Path Planning. In *Proceedings of the 2001 European Control Conference*, Porto, Portugal, September 2001.

[26] K. Sprague, V. Gavrilets, I. Martinos, D. Dugail, B. Mettler, and E. Feron. Design and applications of an avionics system for a miniature acrobatic helicopter. In *Digital Avionics Systems Conference*, Daytona Beach FL, October 2001.

[27] J.A. Tomlin. *Integer and Nonlinear Programming*, chapter Branch-and-Bound Methods for Integer and Non-Convex Programming, pages 437–450. Amsterdam: North-Holland, 1970.