Neal Bhasin

15-424 Star-Lab
Project Report

Introduction

As part of my research with the Google Lunar X-Prize team at CMU, one of my goals is to develop a planner and controller for an autonomous lunar lander. I have previously developed a trajectory generator and simulator, and others on the team have developed Kalman filters for motion tracking. Of specific importance to our team is developing the capability for the lunar lander to use cameras and a laser to scan over a skylight, the entrance to a lunar cave, during landing. This entails flying and landing within close proximity to a lunar pit, something that has previously been considered impossible due to the safety risks.

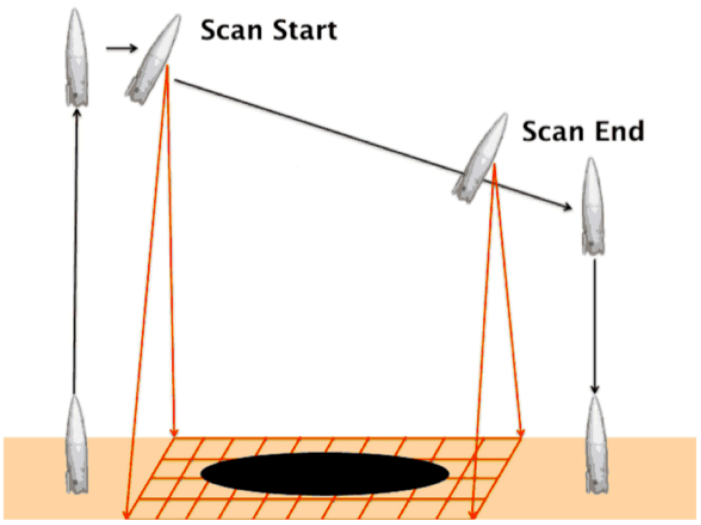


Figure 1. Example scan of a skylight during flyover

Using hybrid systems, I worked to prove the ability of a lunar lander's controller to achieve a safe landing after skylight flyover. Using simplified models of the system, I proved (with varying degrees of success) the safety of a set of control algorithms for controlling the motion of a lunar lander over a skylight or crater.

Hybrid Systems and Logical Analysis

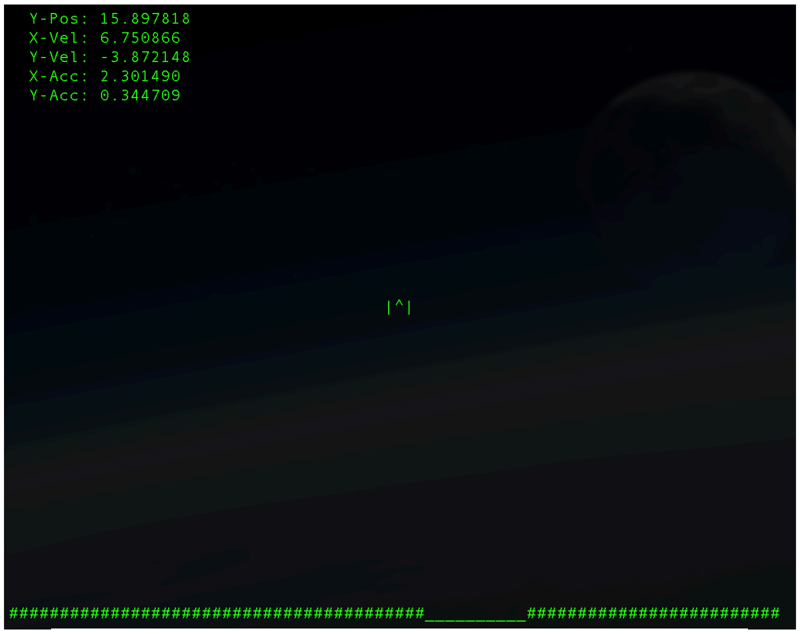
Hybrid systems are an especially appropriate and powerful tool for this kind of application. A realistic system's controller has both discrete and continuous elements that are often difficult or inappropriate to completely separate. First order logic alone enables a simple and very well understood method of determining and verifying properties of completely discrete systems. Simulation enables a simple method of improving confidence in certain claims about many systems, especially continuous ones. Unfortunately, even the most advanced and computationally expensive simulations must skip testing some cases when operating on multiple variables in the real numbers. This introduces uncertainty and edge cases into even rigorously tested systems, which is simply unacceptable for space systems.

Hybrid systems uniquely enable a difficult but straightforward method of formally verifying properties of the continuous and discrete elements of a system simultaneously. In the case of an application like lunar lander control, this is extremely desirable due to the extreme importance of verifying system safety properties to the highest degree possible.

Logical analysis performed upon hybrid systems can yield formal and easily reproducible claims of safety properties over an entire domain of real number variables. One surprising weakness of this analysis however, is the extreme computational complexity of numerical analysis on many variables. Due to the doubly exponential runtime of methods for quantifier elimination, hybrid systems with a non-trivial number of live variables quickly become infeasible targets for analysis.

Simulation

Simulation provides a relatively easy method for more casual verification of system properties. In the context of hybrid systems, I believe simulation represents an essential development tool. It can quickly enable visual feedback coupled with a concrete example of a system failure case. I often found that the only similar feedback provided by logical analysis of hybrid systems was from specifically knowing when and where to check for a counter-example to some property. This also relied upon the produced counterexample to be readable and required additional work to convert the counterexample state into an understandable program run.



```
Y-Pos: 15.897818
X-Vel: 6.750866
Y-Vel: -3.872148
X-Acc: 2.301490
Y-Acc: 0.344709

                                        |^|

##########################################__________#########################
```

Figure 2. Screenshot of text-based simulation utility

In order to make debugging my hybrid systems easier (and more enjoyable), I developed a text-based simulation utility in C (Appendix A). This utility allowed for the implementation and visual simulation of different control algorithms. Initial system conditions were either hard-coded or chosen randomly over a limited domain within the appropriate preconditions. For example, the location of the crater was randomly set at runtime to be within some 20-unit window, always located to the right of the initial lander position. Simple mistakes in the control algorithms that led to unsafe behavior were easily caught using this tool, which took less than one second to compile and run.

A more advanced simulation utility would be a definite component of future work on this project. Visual simulation, integration with logical analysis tools for generating counterexamples, and Monte Carlo methods for randomized trials would all be valuable next steps.

System Model

The lunar lander system is modeled in two dimensions with a set of world and lander constraints, initial preconditions to ensure safety, a hybrid program containing a time-driven controller and modeling world dynamics, and post-conditions that guarantee system safety if proven.

The lander's state begins in final planetary descent, flying at some positive altitude and some initial horizontal position, with some downward and some rightward velocity. There is some constant negative acceleration on the lander due to gravity. The planetary surface is located at an altitude of zero, and on the surface there is one crater with some positive radius, located to the right of the lander's initial position. The lander's time-driven control has some positive time interval between control adjustments, which consist of setting the horizontal and vertical acceleration resulting directly from rocket thrust. The vertical acceleration the lander can produce is limited to be both positive and under some constant system maximum. This maximum is constrained to be at least as much as necessary to reach some negative velocity upon reaching the planetary surface after constant acceleration. That negative surface velocity is the system's safe impact velocity. Any lander velocity at surface impact between this value and zero will ensure the lander momentum is small enough to not damage the lander upon landing. The lander safety is represented by post-conditions ensuring that upon impact with the surface, vertical velocity is within this safe threshold and horizontal position is to the right of the crater.

The system is designed to allow any number of runs of the time driven controller and uses an evolutionary domain constraint to ensure the system's differential evolution is limited between intervals of the controller. Additionally, the current system altitude is constrained to be non-negative in the evolutionary domain. This allows the system to evolve into its final state of zero altitude and ensures checking of the post-conditions at that point. Evolution of the system past this point is unimportant, as the moment of impact alone determines the system safety.

The controller for the system is designed to control vertical and horizontal acceleration independently. Vertical acceleration is set to zero if the controller is able to calculate the existence of a safe vertical descent option at the next time-step. It is otherwise set to a braking value that ensures the lander reaches the surface exactly at the maximum impact velocity after exerting constant acceleration in the presence of gravity during the remaining descent. Horizontal acceleration is set to a breaking value that reduces velocity to stop horizontal motion after flyover of the pit. If the lander has already surpassed the pit, acceleration is applied to correct any leftward motion.

Safety and Efficiency

My submission included four different models, each progressively more complex and representative of the full system. The first model (vertical_lander_a.key) represents a one-run controller that chooses positive acceleration without any upper limit. The system dynamics then undergo one continuous evolution and the safe landing velocity post-condition is checked. This safety property was successfully proven (vertical_lander_a.proof).

The second model (vertical_lander_b.key) represents a time-driven controller that follows one acceleration control strategy at all times, and has an upper bound on acceleration. The safety condition remains the same and is proven (vertical_lander_b.proof) with the assistance of advanced loop invariants ensuring the controller always has a possible and safe strategy.

The third model (vertical_lander_c.key) introduces the existence of the crater, but does not contain the crater's properties in the safety conditions. It also strengthens the controller by allowing zero acceleration when possible. This potentially improves the efficiency of the lander as it can make the same safe descent quicker by not resisting gravity for as much time. Efficiency of the lander could be properly represented by calculating the fuel used by the lander during descent. This would be a function of the acceleration and the mass of the lander, which is itself a function of fuel use. As this metric is very complicated and not the focus of this project, the controller was designed to allow efficiency but not formally guarantee any metrics about the lander's efficiency. This controller was partially proven (vertical_lander_c.proof). There remains an open goal but no KeyMaera-generated counterexample. I believe the controller satisfies its safety property, but without additional computational effort (and perhaps more precise loop invariants) the safety property could not be proven in a reasonable amount of computational time.

The final model (vertical_lander_full.key) reflects the full system model described previously. An incomplete proof of this model's safety is provided (vertical_lander_full.proof). Proving the safety of this model would guarantee the vertical landing momentum of the lander is within the safe limit and also that the lander is able to safely traverse the skylight or crater before landing. Proving the safety of the third model is a necessary precursor to proving the safety of the full model. The horizontal motion of the lander could be evaluated for efficiency in a manner similar to that of the vertical motion. This is not considered here for complexity reasons and is additionally likely to be far less important than vertical efficiency due to the large effect of planetary gravity. A system landing on a body with very weak gravity, like an asteroid, would need to treat horizontal and vertical efficiency more equally.

Future Work

With more time and resources, I believe this system could be enhanced to provide guarantees of both safety and efficiency for a more realistic and general model. First, the models I have developed could be proven (possibly with some adjustments). After this, the model could be augmented to track fuel usage in both dimensions. Efficiency post-conditions and additional system preconditions could be introduced to reason about the fuel usage of the lander. Using loop convergence or other similar techniques could help enable stronger liveness guarantees of the system.

The system could next be extended using stochastic techniques to account for the random noise introduced in both lander sensory readings and thrust. Additionally, the safety properties could then be enriched to assign negative values of different levels to different failure conditions. A realistic, noisy system cannot be proven completely safe, but using these techniques we can provide transparent and logical risk assessments.

Lastly, I believe the system could be extended to cover all six degrees of freedom in the lander's motion. This could be possibly enabled with the assistance of language or analysis tool support for vector and quaternion primitives. Another tool that I believe would be valuable is a mechanism for automatically modularizing and proving sub-problems of a complicated system. In my progressively more complicated models, this could have enabled a quick mechanism to isolate and only prove the novel part of each system. Although these systems may be true and are certainly decidable, their complexity severely limits the ability for currently available computers to reason about their safety.

Appendix A – Simulation Utility Source Code

```
// simulate.c
// Neal Bhasin
// 15-424 Fall 2013
// Simulation to assist controller development for Lab 6

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

const int HEIGHT = 30;
const int WIDTH  = 75;
const int GRAV   = -1; // pix / sec^2
const int FRAMES = 10; // frames / sec

void clearScreen() {
    for(int i=0; i<HEIGHT; i++)
        printf("\n");
    return;
}

void drawWorld(int cx, int cr,
               float px, float py,
               float vx, float vy,
               float ax, float ay) {
    for(int i=HEIGHT; i>=0; i--) {
        if(i == (int)py) {
            for(int j=1; j<(int)px; j++)
                printf(" ");
            printf("|^|\n");
        } else {
            if(i == 31) printf("  X-Pos: %f", px);
            if(i == 30) printf("  Y-Pos: %f", py);
            if(i == 29) printf("  X-Vel: %f", vx);
            if(i == 28) printf("  Y-Vel: %f", vy);
            if(i == 27) printf("  X-Acc: %f", ax);
            if(i == 26) printf("  Y-Acc: %f", ay);
            printf("\n");
        }
    }
    for(int i=0; i<=WIDTH; i++) {
      if(i >= cx - cr && i < cx + cr)
            printf("_");
      else
            printf("#");
     }
    printf("\n\n\n");
    return;
}

void enforceBounds(float *px, float *py) {
    if(*py < 0)      *py = 0;
    if(*py > HEIGHT) *py = HEIGHT;
    if(*px < 0)      *px = 0;
    if(*px > WIDTH)  *px = WIDTH;
}

float control_y(float vy, float py) {
     float vfy = -2;
     return (vy * vy - vfy * vfy) / (2 * py + 0.1) - GRAV;
}

float control_x(int cx, int cr, float px, float py,
                  float vx, float vy) {
     if(px < cx + cr) {
         return (vx * vx) / (2 * py + 0.1) - GRAV;
     } else {
         return -vx / FRAMES;
     }
}

int main() {
    srandom(time(NULL));
    float px = 25;
    float py = 45;
    float vx = 2;
    float vy = -5;
    float ax = 0;
    float ay = 0;
    float fy = 0;
    int cx = 30 + (random() % 20);
    int cr = 5;

    for(int i=0; i<1000; i++) {
        fy = control_y(vy, py);
        ax = control_x(cx, cr, px, py, vx, vy);
        ay = fy + GRAV;
        // Evolve system
        vx += ax / FRAMES;
        vy += ay / FRAMES;
        px += vx / FRAMES;
        py += vy / FRAMES;
        // Bound position and draw lander + world
        enforceBounds(&px, &py);
        drawWorld(cx, cr, px, py, vx, vy, ax, ay);
        usleep(1000000 / FRAMES);
      if(py < 0.25) i = 999;
    }
    return 0;
}
```

Appendix B – Full System Model

```
/* Lab 6 | vertical_lander_full.key
 * Neal Bhasin
 * nsbhasin
 *
 * Simplified model of autonomous lunar lander system.
 * Using a time-driven controller, the system aims to land
 *  on the surface within some acceptable velocity threshold
 *  within a mission time limit.
 */

\functions {
    R H;      /*  Initial system altitude               */
    R X;      /*  Initial system x-position             */
    R g;      /*  World gravity                         */
    R acc_l;  /*  Limit on rocket acceleration          */
    R vel_fl; /*  Limit on impact velocity              */
    R cx;     /*  Crater center                         */
    R cr;     /*  Crater radius                         */
    R TC;     /*  Time driven controller interval time  */
}

\programVariables {
    R x;      /*  Current x-position          */
    R h;      /*  Current altitude            */
    R vx;     /*  Current x-velocity          */
    R vy;     /*  Current y-velocity          */
    R ax;     /*  Current x-acceleration      */
    R ay;     /*  Current y-acceleration      */
    R tc;     /*  Current controller time     */
}

\problem {
    (
     /* System initialization and basic physical constraints */
     H > 0        &
     x = X        &
     h = H        &
     cx > 0       &
     cr > 0       &
     x < cx - cr  &
     vel_fl < 0   &
     vy < vel_fl  &
     vx > 0       &
     g < 0        &
     TC > 0       &

     /* Maximum thrust is enough to stop at surface */
     acc_l >= (vy^2 - vel_fl^2) / (2*H) - g  &
     /* Crater traversal is shorter than descent */
     cx - x + cr < h
     )
    ->
    \[
      (
       ?(h > 0);
       if( (h + vy * TC + 0.5 * g * TC^2) > 0 &
          ((vy + g * TC)^2 - vel_fl^2) /
          (2*(h + vy * TC + 0.5 * g * TC^2)) - g <= acc_l) then
          ay := 0
       else
          ay := (vy^2 - vel_fl^2) / (2*h) - g
       fi;
       ?(0 <= ay & ay <= acc_l);
       if( x < cx + cr) then
          ax := (vx^2) / (2*(cx - x + cr))
       else
          if(vx < 0) then ax := -vx / TC else ax := 0 fi
       fi;
       tc := 0;
       {
           h'  = vy,
           vy' = g + ay,
           x'  = vx,
           vx' = ax,
           tc' = 1
           &   h  >= 0
           &   tc <= TC
       }
      )*@invariant(h >= 0 &
                 ((h > 0) -> (vy < vel_fl &
                              acc_l >= (vy^2 - vel_fl^2) / (2*h) - g)) &
                 ((h = 0) -> (vy >= vel_fl)) )
      \]
     (
     /* At landing, velocity must be within threshold
      and lander must pass crater */
     (h = 0) -> (vy >= vel_fl & vx >= cx + cr)
     )
}
```