Planning for Autonomous Planetary Vehicles

Giuseppe Della Penna*, Benedetto Intrigila† ,
Daniele Magazzeni‡, Fabio Mercorio*

* Dept. of Computer Science, University of L'Aquila
† Dept. of Mathematics, University of Rome "Tor Vergata"
‡ Dept. of Science, University of Chieti

March 9, 2010

Outline

Outline

Motivation

Planetary rovers are particular autonomous systems that usually:

- receive an *activity sequence* from Earth operators
- move on an unknown, hazardous terrain up to a specific position;
- perform some *activities*, e.g., acquire data;
- all of this, while dealing with strict time and resource (especially energy) constraints.

Motivation

Planetary rovers are particular autonomous systems that usually:

- receive an *activity sequence* from Earth operators
- move on an unknown, hazardous terrain up to a specific position;
- perform some *activities*, e.g., acquire data;
- all of this, while dealing with strict time and resource (especially energy) constraints.

Motivation

Planetary rovers are particular autonomous systems that usually:

- receive an *activity sequence* from Earth operators
- move on an unknown, hazardous terrain up to a specific position;
- perform some *activities*, e.g., acquire data;
- all of this, while dealing with strict time and resource (especially energy) constraints.

Motivation

Planetary rovers are particular autonomous systems that usually:

- receive an *activity sequence* from Earth operators
- move on an unknown, hazardous terrain up to a specific position;
- perform some *activities*, e.g., acquire data;
- all of this, while dealing with strict time and resource (especially energy) constraints.

Motivation

Planetary rovers are particular autonomous systems that usually:

- receive an *activity sequence* from Earth operators
- move on an unknown, hazardous terrain up to a specific position;
- perform some *activities*, e.g., acquire data;
- all of this, while dealing with strict time and resource (especially energy) constraints.

Motivation

Main Challenges:

☹ the activity sequence (plan) must be very precise in order to optimise mission time and energy consumption

☹ the system dynamics is usual continuous and nonlinear (also in a simplified setting), and nonlinearity is still an open problem for many planners.

Motivation

Main Challenges:

☹ the activity sequence (plan) must be very precise in order to optimise mission time and energy consumption

☹ the system dynamics is usual continuous and nonlinear (also in a simplified setting), and nonlinearity is still an open problem for many planners.

State of the Art

Many planners and techniques have been proposed to deal with *"planning with time and resources consumption"*.

Non-optimal planners:

- MAPGEN [Bresina et al. 2005]: the user provides the planner with a qualitative evaluation of the generated plans;
- ASPEN [Chien el al. 2000]: the plan is iteratively refined to fulfill the constraints.

Optimal planners:

- TM-LPSAT [Shin&Davis 2005] and UPPAAL/TIGA [Berhmann 2007]: can handle only *linear* domains;
- MIPS [Edelkamp&Helmert 2001] manages hybrid systems but does not perform well with nonlinearity due to the use of *symbolic* model checking.

State of the Art

Many planners and techniques have been proposed to deal with *"planning with time and resources consumption"*.

Non-optimal planners:

- MAPGEN [Bresina et al. 2005]: the user provides the planner with a qualitative evaluation of the generated plans;
- ASPEN [Chien el al. 2000]: the plan is iteratively refined to fulfill the constraints.

Optimal planners:

- TM-LPSAT [Shin&Davis 2005] and UPPAAL/TIGA [Berhmann 2007]: can handle only *linear* domains;
- MIPS [Edelkamp&Helmert 2001] manages hybrid systems but does not perform well with nonlinearity due to the use of *symbolic* model checking.

State of the Art

Many planners and techniques have been proposed to deal with *"planning with time and resources consumption"*.

Non-optimal planners:

- MAPGEN [Bresina et al. 2005]: the user provides the planner with a qualitative evaluation of the generated plans;
- ASPEN [Chien el al. 2000]: the plan is iteratively refined to fulfill the constraints.

Optimal planners:

- TM-LPSAT [Shin&Davis 2005] and UPPAAL/TIGA [Berhmann 2007]: can handle only *linear* domains;
- MIPS [Edelkamp&Helmert 2001] manages hybrid systems but does not perform well with nonlinearity due to the use of *symbolic* model checking.

State of the Art

Many planners and techniques have been proposed to deal with *"planning with time and resources consumption"*.

Non-optimal planners:

- MAPGEN [Bresina et al. 2005]: the user provides the planner with a qualitative evaluation of the generated plans;
- ASPEN [Chien el al. 2000]: the plan is iteratively refined to fulfill the constraints.

Optimal planners:

- TM-LPSAT [Shin&Davis 2005] and UPPAAL/TIGA [Berhmann 2007]: can handle only *linear* domains;
- MIPS [Edelkamp&Helmert 2001] manages hybrid systems but does not perform well with nonlinearity due to the use of *symbolic* model checking.

State of the Art

Many planners and techniques have been proposed to deal with *"planning with time and resources consumption"*.

Non-optimal planners:

- MAPGEN [Bresina et al. 2005]: the user provides the planner with a qualitative evaluation of the generated plans;
- ASPEN [Chien el al. 2000]: the plan is iteratively refined to fulfill the constraints.

Optimal planners:

- TM-LPSAT [Shin&Davis 2005] and UPPAAL/TIGA [Berhmann 2007]: can handle only *linear* domains;
- MIPS [Edelkamp&Helmert 2001] manages hybrid systems but does not perform well with nonlinearity due to the use of *symbolic* model checking.

Outline

Planning through Explicit Model Checking

Model Checking refers to algorithms and tools which take in input the formal specification of a system S and of a property φ and return true if φ is satisfied by S, or return false and give a counterexample otherwise.

An Explicit Model Checker:

1. Obtains the transition graph of the system S
2. Computes the reachable states, starting from the initial states
3. Verifies φ on all reachable states.

Explicit Model Checking works well on nonlinear systems.

How to use a Model Checker as a Planner?

If we look at error states as goal states, we can use a model checker as a plan generator.

Planning through Explicit Model Checking

Model Checking refers to algorithms and tools which take in input the formal specification of a system S and of a property φ and return true if φ is satisfied by S, or return false and give a counterexample otherwise. An Explicit Model Checker:

1. Obtains the transition graph of the system S
2. Computes the reachable states, starting from the initial states
3. Verifies φ on all reachable states.

Explicit Model Checking works well on nonlinear systems.

How to use a Model Checker as a Planner?

If we look at error states as goal states, we can use a model checker as a plan generator.

Planning through Explicit Model Checking

Model Checking refers to algorithms and tools which take in input the formal specification of a system S and of a property φ and return true if φ is satisfied by S, or return false and give a counterexample otherwise.

An Explicit Model Checker:

1. Obtains the transition graph of the system S
2. Computes the reachable states, starting from the initial states
3. Verifies φ on all reachable states.

Explicit Model Checking works well on nonlinear systems.

How to use a Model Checker as a Planner?

If we look at error states as goal states, we can use a model checker as a plan generator.

Planning through Explicit Model Checking

Model Checking refers to algorithms and tools which take in input the formal specification of a system S and of a property φ and return true if φ is satisfied by S, or return false and give a counterexample otherwise.

An Explicit Model Checker:

1. Obtains the transition graph of the system S
2. Computes the reachable states, starting from the initial states
3. Verifies φ on all reachable states.

Explicit Model Checking works well on nonlinear systems.

How to use a Model Checker as a Planner?

If we look at error states as goal states, we can use a model checker as a plan generator.

Planning through Explicit Model Checking

Model Checking refers to algorithms and tools which take in input the formal specification of a system S and of a property φ and return true if φ is satisfied by S, or return false and give a counterexample otherwise.

An Explicit Model Checker:

1. Obtains the transition graph of the system S
2. Computes the reachable states, starting from the initial states
3. Verifies φ on all reachable states.

Explicit Model Checking works well on nonlinear systems.

How to use a Model Checker as a Planner?

If we look at error states as goal states, we can use a model checker as a plan generator.

Planning through Explicit Model Checking

Model Checking refers to algorithms and tools which take in input the formal specification of a system S and of a property φ and return true if φ is satisfied by S, or return false and give a counterexample otherwise.

An Explicit Model Checker:

1. Obtains the transition graph of the system S
2. Computes the reachable states, starting from the initial states
3. Verifies φ on all reachable states.

Explicit Model Checking works well on nonlinear systems.

How to use a Model Checker as a Planner?

If we look at error states as goal states, we can use a model checker as a plan generator.

UPMurphi [Della Penna et al. ICAPS2009]

UPMurphi

Universal Planner based on CMurphi model checker

UPMurphi is able to:

- exploit real numbers and external $C/C++$ functions to model complex systems;
- exploit several techniques (inherited from CMurphi) that help to mitigate well-known *state explosion* problem;
- reduce memory usage through *bit compression* and *hash compaction*

UPMurphi [Della Penna et al. ICAPS2009]

UPMurphi

Universal Planner based on CMurphi model checker

UPMurphi is able to:

- exploit real numbers and external $C/C++$ functions to model complex systems;
- exploit several techniques (inherited from CMurphi) that help to mitigate well-known *state explosion* problem;
- reduce memory usage through *bit compression* and *hash compaction*

UPMurphi [Della Penna et al. ICAPS2009]

UPMurphi

Universal Planner based on CMurphi model checker

UPMurphi is able to:

- exploit real numbers and external $C/C++$ functions to model complex systems;
- exploit several techniques (inherited from CMurphi) that help to mitigate well-known *state explosion* problem;
- reduce memory usage through *bit compression* and *hash compaction*

UPMurphi [Della Penna et al. ICAPS2009]

UPMurphi

Universal Planner based on CMurphi model checker

UPMurphi is able to:

- exploit real numbers and external $C/C++$ functions to model complex systems;
- exploit several techniques (inherited from CMurphi) that help to mitigate well-known *state explosion* problem;
- reduce memory usage through *bit compression* and *hash compaction*

UPMurphi [Della Penna et al. ICAPS2009]

UPMurphi

Universal Planner based on CMurphi model checker

UPMurphi is able to:

- exploit real numbers and external $C/C++$ functions to model complex systems;
- exploit several techniques (inherited from CMurphi) that help to mitigate well-known *state explosion* problem;
- reduce memory usage through *bit compression* and *hash compaction*

Contribution

We have used UPMurphi to *automatically* generate plans to:

- model Mars environmental conditions;
- model the rover dynamics (expressed by Ordinary Differential Equations);
- control rover's **engine** to move it for a specific distance, while satisfying system constraints and minimising both time and power consumption.

Contribution

We have used UPMurphi to *automatically* generate plans to:

- model Mars environmental conditions;
- model the rover dynamics (expressed by Ordinary Differential Equations);
- control rover's **engine** to move it for a specific distance, while satisfying system constraints and minimising both time and power consumption.

Contribution

We have used UPMurphi to *automatically* generate plans to:

- model Mars environmental conditions;
- model the rover dynamics (expressed by Ordinary Differential Equations);
- control rover's **engine** to move it for a specific distance, while satisfying system constraints and minimising both time and power consumption.

Outline

The Autonomous Planetary Lander [Lee 2003]

- The rover is equipped with batteries, solar panels and very limited communication and computational resources
- During each communication session, the Earth control sends to the rover a plan to drive it to the next place and perform an activity;
- The rover has no *error recovery* procedure: when something unexpected happens, it stops and waits for Earth instructions

GOAL

The rover has to move for d_{final} meters minimizing time and power consumption.

The Autonomous Planetary Lander [Lee 2003]

- The rover is equipped with batteries, solar panels and very limited communication and computational resources
- During each communication session, the Earth control sends to the rover a plan to drive it to the next place and perform an activity;
- The rover has no *error recovery* procedure: when something unexpected happens, it stops and waits for Earth instructions

GOAL

The rover has to move for d_{final} meters minimizing time and power consumption.

The Autonomous Planetary Lander [Lee 2003]

- The rover is equipped with batteries, solar panels and very limited communication and computational resources
- During each communication session, the Earth control sends to the rover a plan to drive it to the next place and perform an activity;
- The rover has no *error recovery* procedure: when something unexpected happens, it stops and waits for Earth instructions

GOAL

The rover has to move for d_{final} meters minimizing time and power consumption.

The Autonomous Planetary Lander [Lee 2003]

- The rover is equipped with batteries, solar panels and very limited communication and computational resources
- During each communication session, the Earth control sends to the rover a plan to drive it to the next place and perform an activity;
- The rover has no *error recovery* procedure: when something unexpected happens, it stops and waits for Earth instructions

GOAL

The rover has to move for d_{final} meters minimizing time and power consumption.

The Autonomous Planetary Lander [Lee 2003]

- The rover is equipped with batteries, solar panels and very limited communication and computational resources
- During each communication session, the Earth control sends to the rover a plan to drive it to the next place and perform an activity;
- The rover has no *error recovery* procedure: when something unexpected happens, it stops and waits for Earth instructions

GOAL

The rover has to move for d_{final} meters minimizing time and power consumption.

System Dynamics - real data from [Rocky7 1997],[Pathfinder 1996]

- The rover requires $energy_{standby}$ Joule/second energy to power the CPU;
- The rover dynamics is given by the following:

$$\begin{aligned} \frac{\partial v}{\partial t} &= a(t) - \mu \cdot g \\ \frac{\partial d}{\partial t} &= v(t) \end{aligned} \tag{1}$$

where:

$a(t)$ is the acceleration given by the rover engine at time t

System Dynamics - real data from [Rocky7 1997],[Pathfinder 1996]

- The rover requires $energy_{standby}$ Joule/second energy to power the CPU;
- The rover dynamics is given by the following:

$$\begin{aligned} \frac{\partial v}{\partial t} &= a(t) - \mu \cdot g \\ \frac{\partial d}{\partial t} &= v(t) \end{aligned} \quad (1)$$

where:

μ is the kinetic friction coefficient of the rover wheels.

System Dynamics - real data from [Rocky7 1997],[Pathfinder 1996]

- The rover requires $energy_{standby}$ Joule/second energy to power the CPU;
- The rover dynamics is given by the following:

$$\begin{aligned} \frac{\partial v}{\partial t} &= a(t) - \mu \cdot g \\ \frac{\partial d}{\partial t} &= v(t) \end{aligned} \tag{1}$$

- the energy required to move the wheels with speed v and acceleration $\dot{v}$ is given by [Tate&Boyd 2000]:

$$f(v, \dot{v}) = \left(\frac{1}{2} \cdot \rho \cdot v^2 \cdot Cd \cdot fa + m \cdot g \cdot \left(Crr + \frac{\dot{v}}{g} \right) \right) \cdot v \tag{2}$$

where:

ρ is the Mars' air density;

System Dynamics - real data from [Rocky7 1997],[Pathfinder 1996]

- The rover requires $energy_{standby}$ Joule/second energy to power the CPU;
- The rover dynamics is given by the following:

$$\begin{aligned} \frac{\partial v}{\partial t} &= a(t) - \mu \cdot g \\ \frac{\partial d}{\partial t} &= v(t) \end{aligned} \quad (1)$$

- the energy required to move the wheels with speed v and acceleration $\dot{v}$ is given by [Tate&Boyd 2000]:

$$f(v, \dot{v}) = \left(\frac{1}{2} \cdot \rho \cdot v^2 \cdot Cd \cdot fa + m \cdot g \cdot \left(Crr + \frac{\dot{v}}{g} \right) \right) \cdot v \quad (2)$$

where:

g is the Mars' gravitational constant;

System Dynamics - real data from [Rocky7 1997],[Pathfinder 1996]

- The rover requires $energy_{standby}$ Joule/second energy to power the CPU;
- The rover dynamics is given by the following:

$$\begin{aligned} \frac{\partial v}{\partial t} &= a(t) - \mu \cdot g \\ \frac{\partial d}{\partial t} &= v(t) \end{aligned} \quad (1)$$

- the energy required to move the wheels with speed v and acceleration $\dot{v}$ is given by [Tate&Boyd 2000]:

$$f(v, \dot{v}) = \left(\frac{1}{2} \cdot \rho \cdot v^2 \cdot Cd \cdot fa + m \cdot g \cdot \left(Crr + \frac{\dot{v}}{g} \right) \right) \cdot v \quad (2)$$

where:
fa is the frontal area of the rover;

System Dynamics - real data from [Rocky7 1997],[Pathfinder 1996]

- The rover requires $energy_{standby}$ Joule/second energy to power the CPU;
- The rover dynamics is given by the following:

$$\begin{aligned} \frac{\partial v}{\partial t} &= a(t) - \mu \cdot g \\ \frac{\partial d}{\partial t} &= v(t) \end{aligned} \quad (1)$$

- the energy required to move the wheels with speed v and acceleration $\dot{v}$ is given by [Tate&Boyd 2000]:

$$f(v, \dot{v}) = \left(\frac{1}{2} \cdot \rho \cdot v^2 \cdot Cd \cdot fa + m \cdot g \cdot \left(Crr + \frac{\dot{v}}{g} \right) \right) \cdot v \quad (2)$$

where:
Cd, Crr are drag and rolling coefficients of the rover;

System Constraints

The commands that can be used to control to the rover engine are:

- Accelerate: increase acceleration by $1.5 cm/s^2$;
- Decelerate: decrease acceleration by $1.5 cm/s^2$;
- Continue: continue with constant acceleration.

A correct plan (sequence of commands) for the rover engine must obey the following constraints:

- the rover speed must not exceed $v_{max} cm/s$;
- the rover must stop every d_{max} meters to perform a *cooling task* (needed to cool the rover's instruments);
- each cooling task lasts t_c seconds and requires $energy_{cooling}$ Joule/second;
- after d_{final} meters the battery charge must be higher than c_{min} Coulomb;
- the goal must be achieved in at most t_{max} seconds.

System Constraints

The commands that can be used to control to the rover engine are:

Accelerate: increase acceleration by $1.5 cm/s^2$;
Decelerate: decrease acceleration by $1.5 cm/s^2$;
Continue: continue with constant acceleration.

A correct plan (sequence of commands) for the rover engine must obey the following constraints:

- the rover speed must not exceed $v_{max} cm/s$;
- the rover must stop every d_{max} meters to perform a *cooling task* (needed to cool the rover's instruments);
- each cooling task lasts t_c seconds and requires $energy_{cooling}$ Joule/second;
- after d_{final} meters the battery charge must be higher than c_{min} Coulomb;
- the goal must be achieved in at most t_{max} seconds.

System Constraints

The commands that can be used to control to the rover engine are:

- Accelerate: increase acceleration by $1.5 cm/s^2$;
- Decelerate: decrease acceleration by $1.5 cm/s^2$;
- Continue: continue with constant acceleration.

A correct plan (sequence of commands) for the rover engine must obey the following constraints:

- the rover speed must not exceed $v_{max} cm/s$;
- the rover must stop every d_{max} meters to perform a *cooling task* (needed to cool the rover's instruments);
- each cooling task lasts t_c seconds and requires $energy_{cooling}$ Joule/second;
- after d_{final} meters the battery charge must be higher than c_{min} Coulomb;
- the goal must be achieved in at most t_{max} seconds.

System Constraints

The commands that can be used to control to the rover engine are:

- Accelerate: increase acceleration by $1.5 cm/s^2$;
- Decelerate: decrease acceleration by $1.5 cm/s^2$;
- Continue: continue with constant acceleration.

A correct plan (sequence of commands) for the rover engine must obey the following constraints:

- the rover speed must not exceed $v_{max} cm/s$;
- the rover must stop every d_{max} meters to perform a *cooling task* (needed to cool the rover's instruments);
- each cooling task lasts t_c seconds and requires $energy_{cooling}$ Joule/second;
- after d_{final} meters the battery charge must be higher than c_{min} Coulomb;
- the goal must be achieved in at most t_{max} seconds.

System Constraints

The commands that can be used to control to the rover engine are:

Accelerate: increase acceleration by $1.5cm/s^2$;
Decelerate: decrease acceleration by $1.5cm/s^2$;
Continue: continue with constant acceleration.

A correct plan (sequence of commands) for the rover engine must obey the following constraints:

- the rover speed must not exceed $v_{max} cm/s$;
- the rover must stop every d_{max} meters to perform a *cooling task* (needed to cool the rover's instruments);
- each cooling task lasts t_c seconds and requires $energy_{cooling}$ Joule/second;
- after d_{final} meters the battery charge must be higher than c_{min} Coulomb;
- the goal must be achieved in at most t_{max} seconds.

System Constraints

The commands that can be used to control to the rover engine are:

- **Accelerate:** increase acceleration by $1.5 cm/s^2$;
- **Decelerate:** decrease acceleration by $1.5 cm/s^2$;
- **Continue:** continue with constant acceleration.

A correct plan (sequence of commands) for the rover engine must obey the following constraints:

- the rover speed must not exceed $v_{max} cm/s$;
- the rover must stop every d_{max} meters to perform a *cooling task* (needed to cool the rover's instruments);
- each cooling task lasts t_c seconds and requires $energy_{cooling}$ Joule/second;
- after d_{final} meters the battery charge must be higher than c_{min} Coulomb;
- the goal must be achieved in at most t_{max} seconds.

System Constraints

The commands that can be used to control to the rover engine are:

- Accelerate: increase acceleration by $1.5 cm/s^2$;
- Decelerate: decrease acceleration by $1.5 cm/s^2$;
- Continue: continue with constant acceleration.

A correct plan (sequence of commands) for the rover engine must obey the following constraints:

- the rover speed must not exceed $v_{max} cm/s$;
- the rover must stop every d_{max} meters to perform a *cooling task* (needed to cool the rover's instruments);
- each cooling task lasts t_c seconds and requires $energy_{cooling}$ Joule/second;
- after d_{final} meters the battery charge must be higher than c_{min} Coulomb;
- the goal must be achieved in at most t_{max} seconds.

System Constraints

The commands that can be used to control to the rover engine are:

Accelerate: increase acceleration by $1.5 cm/s^2$;
Decelerate: decrease acceleration by $1.5 cm/s^2$;
Continue: continue with constant acceleration.

A correct plan (sequence of commands) for the rover engine must obey the following constraints:

- the rover speed must not exceed $v_{max} cm/s$;
- the rover must stop every d_{max} meters to perform a *cooling task* (needed to cool the rover's instruments);
- each cooling task lasts t_c seconds and requires $energy_{cooling}$ Joule/second;
- after d_{final} meters the battery charge must be higher than c_{min} Coulomb;
- the goal must be achieved in at most t_{max} seconds.

System Constraints

The commands that can be used to control to the rover engine are:

- Accelerate: increase acceleration by $1.5 cm/s^2$;
- Decelerate: decrease acceleration by $1.5 cm/s^2$;
- Continue: continue with constant acceleration.

A correct plan (sequence of commands) for the rover engine must obey the following constraints:

- the rover speed must not exceed $v_{max} cm/s$;
- the rover must stop every d_{max} meters to perform a *cooling task* (needed to cool the rover's instruments);
- each cooling task lasts t_c seconds and requires $energy_{cooling}$ Joule/second;
- after d_{final} meters the battery charge must be higher than c_{min} Coulomb;
- the goal must be achieved in at most t_{max} seconds.

System Constraints

The commands that can be used to control to the rover engine are:

Accelerate: increase acceleration by $1.5 cm/s^2$;
Decelerate: decrease acceleration by $1.5 cm/s^2$;
Continue: continue with constant acceleration.

A correct plan (sequence of commands) for the rover engine must obey the following constraints:

- the rover speed must not exceed $v_{max} cm/s$;
- the rover must stop every d_{max} meters to perform a *cooling task* (needed to cool the rover's instruments);
- each cooling task lasts t_c seconds and requires $energy_{cooling}$ Joule/second;
- after d_{final} meters the battery charge must be higher than c_{min} Coulomb;
- the goal must be achieved in at most t_{max} seconds.

System Constraints

There are essentially two *failure conditions*:

Engine blown: when the speed exceeds v_{max}, the entire mission fails;

No energy: if the energy becomes less than c_{min}, the rover stops and uses the residual energy to wait for Earth instructions;

Plan evaluation function

The function $C(s_i, a_i)$ evaluates the cost of a single plan step (considering time and energy).

For each state s_i and for each action a_i, $C(s_i, a_i)$:

System Constraints

There are essentially two *failure conditions*:

Engine blown: when the speed exceeds v_{max}, the entire mission fails;

No energy: if the energy becomes less than c_{min}, the rover stops and uses the residual energy to wait for Earth instructions;

Plan evaluation function

The function $C(s_i, a_i)$ evaluates the cost of a single plan step (considering time and energy).

For each state s_i and for each action a_i, $C(s_i, a_i)$:

System Constraints

There are essentially two *failure conditions*:

Engine blown: when the speed exceeds v_{max}, the entire mission fails;

No energy: if the energy becomes less than c_{min}, the rover stops and uses the residual energy to wait for Earth instructions;

Plan evaluation function

The function $C(s_i, a_i)$ evaluates the cost of a single plan step (considering time and energy).
For each state s_i and for each action a_i, $C(s_i, a_i)$:

System Constraints

There are essentially two *failure conditions*:

Engine blown: when the speed exceeds v_{max}, the entire mission fails;

No energy: if the energy becomes less than c_{min}, the rover stops and uses the residual energy to wait for Earth instructions;

Plan evaluation function

The function $C(s_i, a_i)$ evaluates the cost of a single plan step (considering time and energy).

For each state s_i and for each action a_i, $C(s_i, a_i)$:

System Constraints

There are essentially two *failure conditions*:

Engine blown: when the speed exceeds v_{max}, the entire mission fails;

No energy: if the energy becomes less than c_{min}, the rover stops and uses the residual energy to wait for Earth instructions;

Plan evaluation function

The function $C(s_i, a_i)$ evaluates the cost of a single plan step (considering time and energy).
For each state s_i and for each action a_i, $C(s_i, a_i)$:

System Constraints

There are essentially two *failure conditions*:

Engine blown: when the speed exceeds v_{max}, the entire mission fails;

No energy: if the energy becomes less than c_{min}, the rover stops and uses the residual energy to wait for Earth instructions;

Plan evaluation function

The function $C(s_i, a_i)$ evaluates the cost of a single plan step (considering time and energy).
For each state s_i and for each action a_i, $C(s_i, a_i)$:

$$\frac{{energy_{standby}}^2}{t_{max} - i} \text{ when the rover is stopped;}$$

System Constraints

There are essentially two *failure conditions*:

Engine blown: when the speed exceeds v_{max}, the entire mission fails;

No energy: if the energy becomes less than c_{min}, the rover stops and uses the residual energy to wait for Earth instructions;

Plan evaluation function

The function $C(s_i, a_i)$ evaluates the cost of a single plan step (considering time and energy).
For each state s_i and for each action a_i, $C(s_i, a_i)$:

$\frac{(energy_{standby} + energy_{cooling})^2}{t_{max} - i}$ when the rover is in a cooling phase;

System Constraints

There are essentially two *failure conditions*:

Engine blown: when the speed exceeds v_{max}, the entire mission fails;

No energy: if the energy becomes less than c_{min}, the rover stops and uses the residual energy to wait for Earth instructions;

Plan evaluation function

The function $C(s_i, a_i)$ evaluates the cost of a single plan step (considering time and energy).
For each state s_i and for each action a_i, $C(s_i, a_i)$:

$\frac{(energy_{standby}+f(v_i,\dot{v}_i))^2}{t_{max}-i}$ when the rover is moving.

System Constraints

There are essentially two *failure conditions*:

Engine blown: when the speed exceeds v_{max}, the entire mission fails;

No energy: if the energy becomes less than c_{min}, the rover stops and uses the residual energy to wait for Earth instructions;

Plan evaluation function

The function $C(s_i, a_i)$ evaluates the cost of a single plan step (considering time and energy).
For each state s_i and for each action a_i, $C(s_i, a_i)$:

0 if the rover triggers failure conditions;

Outline

Optimal Planning

- In our experiment, we set $d_{final} = 2m$, $t_{max} = 60s$ and $c_{min} = 1,000C$;
- All real variables were rounded up to first decimal digit;
- The number of different systems states, with the given variable approximation, is 2.2×10^{13};
- However, UPMurphi found that only $939,447$ states were actually reachable
- UPMurphi sythesised the optimal plan (w.r.t. the given cost function) in 2,257 seconds with a peak memory requirement of 500 MB;
- Optimisation allowed us to save $922,7C$ w.r.t. the required minimal battery charge c_{min}, and $17s$ w.r.t. the maximum allowed plan duration t_{max}.

Optimal Planning

- In our experiment, we set $d_{final} = 2m$, $t_{max} = 60s$ and $c_{min} = 1,000C$;
- All real variables were rounded up to first decimal digit;
- The number of different systems states, with the given variable approximation, is 2.2×10^{13};
- However, UPMurphi found that only 939,447 states were actually reachable
- UPMurphi sythesised the optimal plan (w.r.t. the given cost function) in 2,257 seconds with a peak memory requirement of 500 MB;
- Optimisation allowed us to save $922,7C$ w.r.t. the required minimal battery charge c_{min}, and $17s$ w.r.t. the maximum allowed plan duration t_{max}.

Optimal Planning

- In our experiment, we set $d_{final} = 2m$, $t_{max} = 60s$ and $c_{min} = 1,000C$;
- All real variables were rounded up to first decimal digit;
- The number of different systems states, with the given variable approximation, is 2.2×10^{13};
- However, UPMurphi found that only $939,447$ states were actually reachable
- UPMurphi sythesised the optimal plan (w.r.t. the given cost function) in 2,257 seconds with a peak memory requirement of 500 MB;
- Optimisation allowed us to save $922,7C$ w.r.t. the required minimal battery charge c_{min}, and $17s$ w.r.t. the maximum allowed plan duration t_{max}.

Optimal Planning

- In our experiment, we set $d_{final} = 2m$, $t_{max} = 60s$ and $c_{min} = 1,000C$;
- All real variables were rounded up to first decimal digit;
- The number of different systems states, with the given variable approximation, is 2.2×10^{13};
- However, UPMurphi found that only $939,447$ states were actually reachable
- UPMurphi sythesised the optimal plan (w.r.t. the given cost function) in 2,257 seconds with a peak memory requirement of 500 MB;
- Optimisation allowed us to save $922,7C$ w.r.t. the required minimal battery charge c_{min}, and $17s$ w.r.t. the maximum allowed plan duration t_{max}.

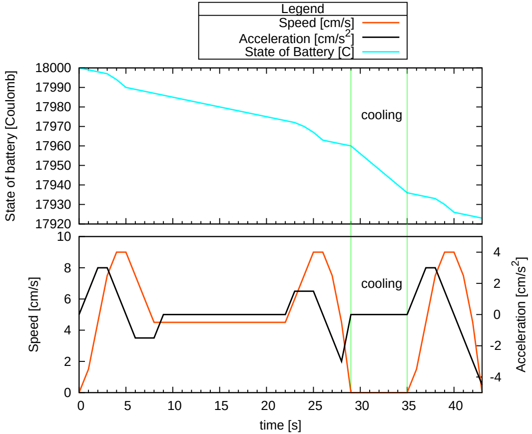
Optimal Planning

- In our experiment, we set $d_{final} = 2m$, $t_{max} = 60s$ and $c_{min} = 1,000C$;
- All real variables were rounded up to first decimal digit;
- The number of different systems states, with the given variable approximation, is 2.2×10^{13};
- However, UPMurphi found that only 939, 447 states were actually reachable
- UPMurphi sythesised the optimal plan (w.r.t. the given cost function) in 2,257 seconds with a peak memory requirement of 500 MB;
- Optimisation allowed us to save $922, 7C$ w.r.t. the required minimal battery charge c_{min}, and $17s$ w.r.t. the maximum allowed plan duration t_{max}.

Optimal Planning

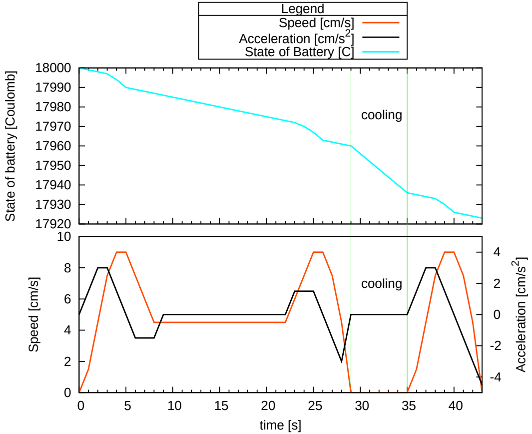
- In our experiment, we set $d_{final} = 2m$, $t_{max} = 60s$ and $c_{min} = 1,000C$;
- All real variables were rounded up to first decimal digit;
- The number of different systems states, with the given variable approximation, is 2.2×10^{13};
- However, UPMurphi found that only $939,447$ states were actually reachable
- UPMurphi sythesised the optimal plan (w.r.t. the given cost function) in 2,257 seconds with a peak memory requirement of 500 MB;
- Optimisation allowed us to save $922,7C$ w.r.t. the required minimal battery charge c_{min}, and $17s$ w.r.t. the maximum allowed plan duration t_{max}.

Optimal Plan Evolution



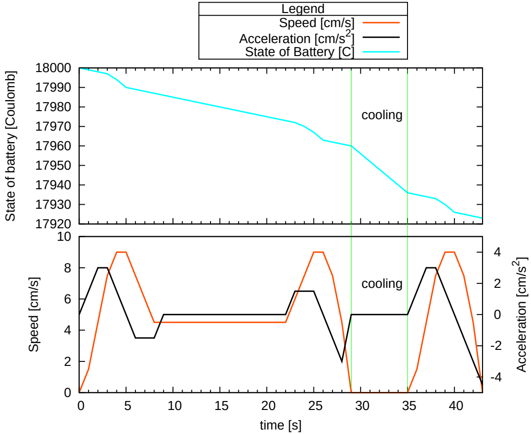
$t = 0$ the rover battery starts with $18,000C$ of charge and with $v = 0, \dot{v} = 0$

Optimal Plan Evolution



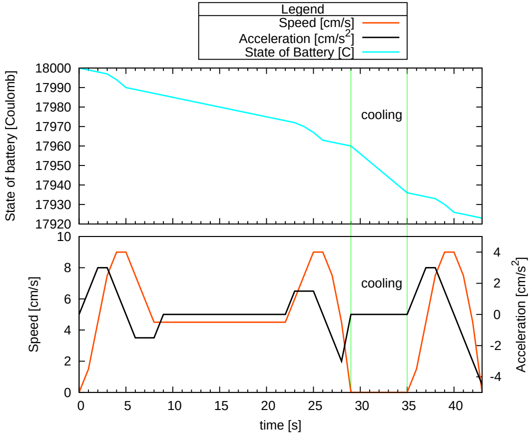
$t = 5$ during first 5 seconds the rover consumes a lot of energy to increase its speed

Optimal Plan Evolution



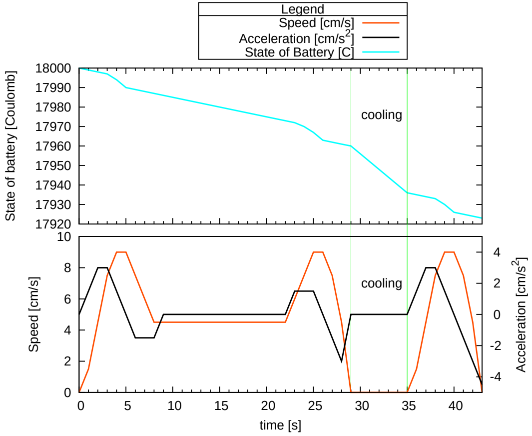
$5 < t \leq 22$ the rover reduces its speed to avoid an "engine blown" failure

Optimal Plan Evolution



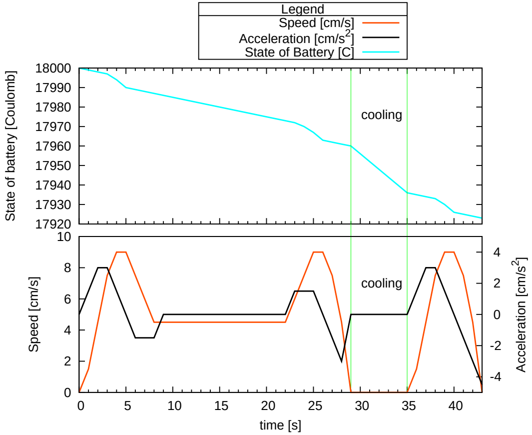
$22 < t \leq 25$ the rover increases its speed again

Optimal Plan Evolution



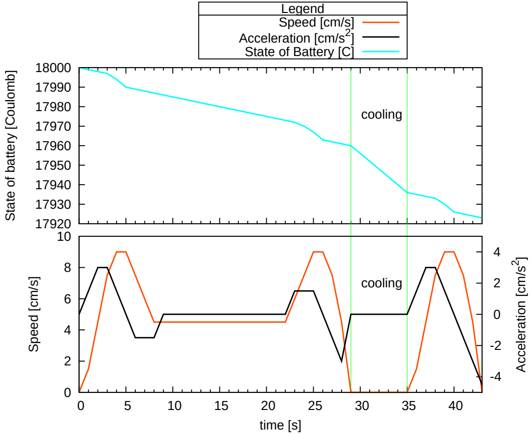
$25 < t \leq 29$ the rover brakes and stops to perform a cooling task

Optimal Plan Evolution



$29 < t \leq 35$ the rover performs a cooling task

Optimal Plan Evolution



$35 < t \leq 43$ the rover covers the remaining distance to the goal.

Outline

Conclusions

We showed how an explicit model checking based planner, namely UPMurphi, can be used to

- create a quite realistic model of a planetary rover (preserving its complex, nonlinear dynamics);
- generate time and resource-optimal plans to control rover engine;

Thus, UPMurphi could be an useful tool to plan activities for autonomous vehicles.

Conclusions

We showed how an explicit model checking based planner, namely UPMurphi, can be used to

- create a quite realistic model of a planetary rover (preserving its complex, nonlinear dynamics);
- generate time and resource-optimal plans to control rover engine;

Thus, UPMurphi could be an useful tool to plan activities for autonomous vehicles.

Conclusions

We showed how an explicit model checking based planner, namely UPMurphi, can be used to

- create a quite realistic model of a planetary rover (preserving its complex, nonlinear dynamics);
- generate time and resource-optimal plans to control rover engine;

Thus, UPMurphi could be an useful tool to plan activities for autonomous vehicles.